PnP Maxtools: Autonomous Parameter Control in MaxMSP Utilizing MIR Algorithms

Austin Alexander Franklin
Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://repository.lsu.edu/gradschool_dissertations

Recommended Citation
https://repository.lsu.edu/gradschool_dissertations/6025

This Dissertation is brought to you for free and open access by the Graduate School at LSU Scholarly Repository. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Scholarly Repository. For more information, please contact gradetd@lsu.edu.
PNP.MAXTOOLS: AUTONOMOUS PARAMETER CONTROL IN MAXMSP UTILIZING MIR ALGORITHMS

A Dissertation

Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College in partial fulfillment of the requirements for the degree of Doctor of Philosophy in The School of Music

by

Austin Alexander Franklin
B.M., Lamar University, 2016
M.M., Louisiana State University, 2019
May 2023
ACKNOWLEDGEMENTS

I would like to thank the faculty and students of the Experimental Music and Digital Media and Composition programs at LSU for being such an enriching and warming community. A special thanks is needed for Jesse Allison, Mara Gibson, and Stephen Beck who have all been supportive and helpful throughout the research process. I would like to thank my family and friends for their constant encouragement and excitement to learn about my work. I would also like thank my fiancé Mathilde for her unrelenting love, support, and editing skills. Finally, to those who have been eager to perform my music or compose using my software, you are the reason this project exists. Thank you.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................. iii

LIST OF FIGURES ................................................................. vii

ABSTRACT ................................................................................. ix

INTRODUCTION ........................................................................... xi

CHAPTER 1. THE FOUNDATIONS OF REFLEXIVE AUTOMATION .... 5
  1.1. Automation ................................................................. 5
  1.2. Mapping ................................................................. 9
  1.3. Music Information Retrieval ....................................... 13
  1.4. Introduction of Reflexive Automation ......................... 23

CHAPTER 2. PRECEDENTS FOR REFLEXIVE AUTOMATION ....... 31
  2.1. Automation in Music ............................................... 31
  2.2. Mapping in Music .................................................. 36
  2.3. Music Information Retrieval in Music ......................... 42

CHAPTER 3. PNP.MAXTOOLS OVERVIEW ............................... 46
  3.1. Package Design .................................................... 46
  3.2. Filters ................................................................. 49
  3.3. Sound Content Descriptors ....................................... 54
  3.4. Additional Controls ............................................... 71
  3.5. Effects ............................................................... 72
  3.6. Event-Based Applications ....................................... 73

CHAPTER 4. EVALUATION AND TESTING .............................. 82
  4.1. Evaluation Using the Cranfield Model ......................... 82
  4.2. Research Considerations ......................................... 90
  4.3. Presentation of Data ............................................... 93
  4.4. Conclusion .......................................................... 111

CHAPTER 5. AN OVERVIEW OF COMPOSED WORKS ............ 113
  5.1. Bloom ............................................................... 113
  5.2. Concentric Circles ............................................... 115
  5.3. I/O ............................................................... 117
  5.4. Sing! ............................................................... 121

CONCLUSION .............................................................................. 126

APPENDIX A. PNP.MAXTOOLS DOCUMENTATION ................ 129
  A.1. Filters ............................................................... 129
  A.2. Sound Content Descriptors .................................... 130
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Four types of automation in DAWs: fades/curves, binary, steps, and spikes</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Score excerpt of <em>Music of Changes</em> by John Cage</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>A simple example of mapping in Max</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>An example of mapping types in Max</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Schaeffer’s three maps</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>Timbrespace 3D visualization of classification</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>An Mpeg-7 extract for describing information about a music title</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>Visualization of a signal in the frequency domain</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>The patellar reflex test (left) and a reflexive musical system (right)</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>A 1-dimensional gesture (left) and a 2-dimensional gesture (right)</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>The TibetBot, lrBot, and GuitarBot</td>
<td>32</td>
</tr>
<tr>
<td>12</td>
<td>Haile’s anthropomorphic design</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>The Robo-Cajon diagram</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>Live stream of the Telematic Piano performance</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>Architecture of ml.lib workflow</td>
<td>37</td>
</tr>
<tr>
<td>16</td>
<td>A participant playing upward scales in Scriabin excerpt</td>
<td>40</td>
</tr>
<tr>
<td>17</td>
<td>GUI for the Freesound Player</td>
<td>43</td>
</tr>
<tr>
<td>18</td>
<td>The top-level phrase behavior word of Voyager</td>
<td>45</td>
</tr>
<tr>
<td>19</td>
<td>Proposed signal chain</td>
<td>47</td>
</tr>
<tr>
<td>20</td>
<td>PnP.Maxtools package homepatcher</td>
<td>48</td>
</tr>
<tr>
<td>21</td>
<td>Analysis tab for demo</td>
<td>50</td>
</tr>
<tr>
<td>22</td>
<td>FFT implemented in pnp.binpass˜</td>
<td>51</td>
</tr>
<tr>
<td>23</td>
<td>FFT implemented in pnp.notch˜</td>
<td>52</td>
</tr>
<tr>
<td>24</td>
<td>FFT implemented in pnp.overtone˜</td>
<td>53</td>
</tr>
<tr>
<td>25</td>
<td>FFT implemented in pnp.pitchclass˜</td>
<td>53</td>
</tr>
</tbody>
</table>
ABSTRACT

This research presents a new approach to computer automation through the implementation of novel real-time music information retrieval algorithms developed for this project. It documents the development of the PnP.Maxtools package, a set of open source objects designed within the popular programming environment MaxMSP. The package is a set of pre/post processing filters, objective and subjective timbral descriptors, audio effects, and other objects that are designed to be used together to compose music or improvise without the use of external controllers or hardware. The PnP.Maxtools package objects are designed to be used quickly and easily using a ‘plug and play’ style with as few initial arguments needed as possible. The PnP.Maxtools package is designed to take incoming audio from a microphone, analyze it, and use the analysis to control an audio effect on the incoming signal in real-time. In this way, the audio content has a real musical and analogous relationship with the resulting musical transformations while the control parameters become more multifaceted and better able to serve the needs of artists. The term Reflexive Automation is presented that describes this unsupervised relationship between the content of the sound being analyzed and the analogous and automatic control over a specific musical parameter. A set of compositions are also presented that demonstrate ideal usage of the object categories for creating reflexive systems and achieving fully autonomous control over musical parameters.
INTRODUCTION

The work presented in this dissertation is interdisciplinary in both thought and practice. It exists primarily in three domains: concept, application, and demonstration, the first of which requires a subtle re-framing of well understood concepts that currently exist within the computer music paradigm. It presents a new approach to computer automation for live electronic performance and audio effects processing, termed Reflexive Automation, that relies on the simultaneous use of automation, mapping, and music information retrieval in software systems. These concepts represent the foundation and historical relevance for Reflexive Automation and are presented in detail in Chapters 1 and 2 along with an in depth discussion of Reflexive Automation, its numerous applications, and potential future ramifications:

- Automation - in the case of music production, automation is a method for automatically performing tasks over time
- Mapping - the process of connecting data with a sound producing or sound altering source within a range that fits the need of the interaction
- Music information retrieval - the interdisciplinary science of retrieving information from music

Traditionally, automation in computer music is understood as an offline approach to controlling musical parameters over time, while Reflexive Automation views computer automation in a manner similar to classification and regression algorithms found in machine learning systems, which automate outputs based on training data and eventually inputs. However, reflexive systems do not require training. Instead, they respond to specific timbral features of live inputs through real-time analysis determined by the music information retrieval algorithm(s) used in the system. By connecting, or mapping, the output of a real-time music information retrieval algorithm to control an audio effect that is processing the same signal used for analysis, Reflexive Automation affords new and meaningful interactions be-
tween human performers and computers. This shift in framing automation as an automatic response to external stimuli rather than a user-supervised method for producing change over time is at the core of Reflexive Automation.

Reflexive Automation can also be understood as a tool capable of abstraction that provides greater ease of use than most current live electronic systems. Reflexive systems are designed to require no training data and no additional hardware or software other than a microphone, audio interface, and laptop, so they can be designed to alleviate the technological burden that is often placed on performers by composers with regards to set up, rehearsal, cable management, transportation, and more. Furthermore, reflexes can be achieved by preserving traditional music notation symbols which can correspond with gestures that can be detected through analysis.

The \textit{PnP.Maxtools} software package for MaxMSP developed for this dissertation includes filters, timbral descriptors, controls, and effects that are designed for the real-time implementation of reflexive systems for music composition and improvisation. While most software packages in MaxMSP contain a set of similar objects based around a single tool or technique and provide many functions with that tool, the \textit{PnP.Maxtools} package is built using modular categories, or categories which function together and allow for numerous configurations using different objects from each category. Chapter 3 introduces the software package and describes the novel music information retrieval algorithms created specifically for this project and the psycho-acoustical and mathematical research upon which these new additions to the Max community are based.

The majority of the timbral descriptors in the \textit{PnP.Maxtools} package are designed as subjective descriptors, meaning that they subjectively describe the sensation associated with the spectral content of the sound rather than objectively quantify it. Given that the way humans perceive sound varies significantly by age and culture, the study of classifying and formalizing subjective descriptions of music, while extremely difficult, is at the forefront of music information retrieval research. The \textit{PnP.Maxtools} package is the first set of objects
in MaxMSP to provide subjective timbral descriptors and is the largest collection of timbral descriptors within Max generally, the efficacy of which was evaluated against other well-known feature extraction toolboxes and by conducting subjective listening tests to determine the degree to which the output from these models align with perceived characteristics of that sound. The findings from these studies is presented in detail in chapter 4.

Finally, several compositions are presented that exhibit one of many possible solutions for creating reflexive systems using the PnP.Maxtools objects. Some of these solutions rely on event detection functions, which are methods for detecting singular events and controlling audio effects discretely rather than using continuous controls. Others involve detecting more complex musical gestures through the development of a cooperative descriptor, which is the implementation of several simultaneous descriptor outputs that are scaled, summed, and used to control the behavior of an audio effect. With each of these systems, the added complexity in terms of ease of usage, consistency across performances, and other dependant variables such as microphone type, room size, etc. that results when building reflexive systems is described. In particular, the compositions were created expressly for demonstrating:

- How Reflexive Automation is applicable to the creative process
- An overview of the PnP.Maxtools category framework
- Novel implementations of the PnP.Maxtools objects which achieve a wide variety of creative goals
- The mapping methods used to control musical parameters

The compositions also exhibit wide variation with regards to style, instrumentation, and notation to demonstrate the flexible and ubiquitous application of Reflexive Automation and the PnP.Maxtools objects. Some of the compositions are strictly notated and some are aleatoric and require the performers to make their own decisions in terms gesture and input to reflexive systems. In these instances of freedom, reflexive systems are best understood as a kind of sonic augmentation, or an acousmatic extension of the performers own sound. While most compositions are for solo instruments, the chamber work presents a hierarchy of
reflexes that are designed to control both local audio effects on individual instruments as well as global effects that change when the entire ensemble makes musical gestures simultaneously. This mediates interaction between players in new ways and creates interesting combinations of reflexes.
CHAPTER 1. THE FOUNDATIONS OF REFLEXIVE AUTOMATION

1.1. Automation

Computer musicians are often motivated to write computer programs to achieve artistic goals or generate novel musical structures. Most of time, these programs operate independently, with human interaction needed only to set up and begin the operation. The ability of computers to automatically perform tasks over time is a revolutionary concept that within the computer music paradigm and can be found in many disciplines, such as computer science, robotics, and machine learning. Alban Berg wrote that the fundamental contribution of the computer in music is one that empowers the composer to “hear that which could not be heard without the computer, to think that which could not be thought without the computer, and to learn that which could not be learned without the computer. The computer can allow a composer to write music that goes beyond that which she is already capable of.”

Additionally, computer-based automation allows composers achieve a level a speed and precision that humans simply cannot. In its simplest form, automation is a horizontal-only approach: a fixed stream of time-tagged values that progress over a finite amount of time to control the state of a parameter, often with interpolation from one value to the next. The use of automation is fundamental to mediums that are linear and time-based, such as digital audio workstation and video editing environments. When performing live music with electronics, automation can become more of an art-form than a science, requiring the composer to carefully consider the interactions between human and computer. In some cases the computer becomes an extension of the performers movement or sound. In other cases it can behave like a musical partner that is capable of responding in real-time to the movement and sound of the human performer.

---

2Timothy Place et al., “Flexible control of composite parameters in Max/MSP,” 2008,
1.1.1. Automation in Digital Audio Workstations (DAWs)

Automation was introduced in early DAWs as an alternate method for recording fader movements on continuous MIDI controllers. This approach enabled users to use MIDI data to control certain parameters within a mix. Today, most DAWs have multiple methods and data types for recording and editing fader movements more precisely. There are two main ways to add automation to a mix in a DAW: offline and real-time. The offline method involves using the mouse to manually click and add automation points at specific places throughout the track. When playback is enabled, the parameter value will follow the created automation envelope until a point is reached, where the parameter value will then begin following the new envelope towards the next point. The real-time method involves setting the parameters into write mode. This enables playback and allows user to physically move faders and knobs on a control surface to manipulate on-screen controls. Once all the movements are made and write-mode is disabled, the on-screen controls will move according to the automation pass. If a mistake is made during the recording process, the user can simply re-record the automation envelope or use an offline method to correct the preexisting automation on the track.
There are four types of automation in DAWs: fades and curves, binary, steps, and spikes. Fades and curves are most commonly used to automate the volume of tracks, the dry/wet on effects (i.e. delay), ADSR values on synthesizers, and EQ and filter parameters such as high and low pass filters and cutoff frequency in a linear, logarithmic, or exponential manner. Binary automation values can only be 0 and 1 and are used when something needs to happen instantly. This type of automation is most frequently used to toggle similar synth parameters and effects on and off and mute and unmute tracks. Step automation controls parameters discretely in a manner similar to binary automation. However, step automation is capable of specifying automation values between 0 and 1. Although this type of automation is less common, it can be used to change synth parameters such as LFO frequency and oscillator pitch. Other uses include the transposition of MIDI pitches, A/B crossover on plugins that allow for A and B settings, and syncing rhythmic effects such as gating and stuttering plugins. Spike automation is commonly used when something needs to happen quickly. Fades and curves are often too slow, while binary and step automation can be too limiting and abrupt. Spike automation can be thought of as micro-fades or curves, since the only distinguishing factor is the time scale on which they operate. This type of automation is most commonly used for quickly changing effects, such as filter cutoff, distortion, or reverb, for transitional purposes.

1.1.2. Automation in Live Electronics

To start a discussion about this particular subject, it is necessary to first define the term live-electronics and its implications. The “ElectroAcoustic Resource Site project” (EARS) defines it as: “A term dating from the analogue age of electroacoustic music that describes performance involving electronic instruments which can be performed in real-time. The term is more commonly expressed today as music involving interactive instruments.”

---


generally implies the interaction, which transforms and/or processes live or recorded sound in real-time.

If automation is used to achieve an effect like these described above, the composer/programmer normally has two basic decisions to make: firstly, the extent to which the live electronics should use automation and how the performer can control or interact with it; secondly, the composer must weigh the musical and technical advantages and disadvantages of automation for the piece and its performance. The degree of automation that exists within a system or music composition or performance can range from absolutely passive, in which the performer/composer is in complete control, to fully autonomous, in which the system or music composition is human independent. The degree of control a performer has over a performance is known as agency.

1.1.3. Automated Composition

Automated composition is the technique or process by which music is composed independently from the composer. Perhaps the simplest form of automating music is by using randomness to guide the formal structure of the composition. In acoustic music, examples can be found in the musical dice games of Mozart and the chance music of John Cage, both of which piece together new musical fragments chosen by rolling dice or consulting I Ching tablets.

In computer music, Lejaren Hiller was one of the first to compose a piece of music using a computer. In 1957, he used the Illiac computer to compose the *Iliac Suite* for string quartet, which employed a different algorithm in each movement. There were several methods and rules defined throughout the four movements work to generate musical material, such as the

---


probabilistic and stochastic methods such as a Markov chain, which is a process based on
the principle that within a sequence of events, the choice of a new event is closely related to
an immediately preceding event, without any consideration of earlier events.

1.2. Mapping
MaxMSP is a popular programming language for musical applications because it allows for
quick prototyping and composition of interactive elements. Using the metaphor of an analog
synthesizer, this popularity is related to the inherently modular approach to programming
offered by these programs. In Max, modules are connected in an environment, or patch,
where the data flow is unidirectional. When connecting objects in Max, it is often necessary
to scale or convert values from one type to another, so that the output values of one object
can be used as the input for another object.

When considering an acoustic instrument, the interface for interaction and the source of
the sound production are usually the same mechanism. For a drum, the head both produces

---

the sound and is the interface by which to produce that sound. Hunt et al. suggest that the same is not true for virtual instruments\(^{10}\). For example, an oscillator producing tones which correspond to MIDI representation of pitch functions according to two different scales. The performer plays the keys of the keyboard, each of which corresponds to a number between 0 and 127. Using this representation, an octave anywhere on the keyboard is equal to +12 above the bottom note. However, an octave on the keyboard represents a doubling of the frequency of the bottom note. For the production of a sound to be possible, it is necessary to convert, or map, MIDI pitch representation to frequency in order to synthesize the correct pitch. Mapping is the process of connecting the sound-producing source with the interface for the performer to interact within a range that fits the need of the interaction. An example of mapping in MaxMSP is shown above where a linear change in pitch corresponds to an exponential change in frequency using the *mtof* object.

1.2.1. Mapping Methods

According to Hunt et al., the process of determining mapping with respect to virtual instrument parameters filters down into the two following choices: procedural and explicit. The key difference between these two methods is that the procedural method such as neural networks or genetic algorithms uses a procedure to map parameters in an artificially intelligent manner as opposed to having the performer retain direct control of parameter mapping.

1.2.2. Procedural Mapping

The procedure in the case of the neural network is powered by mathematical algorithms that analyze data and use it to “learn” how to make decisions without a human directly controlling the decision-making process or the procedure past the initialization phase. The process is similar when using genetic algorithms. These mathematical algorithms also analyze data, but a genetic algorithm models evolutionary biology and assigns fitness scores to the data, uses the best fitness data to produce a new population of data, and then repeats this process until a 100% fit data point/data group is produced.

Lee et al. suggest a multi-dimensional neural network to control audio synthesis by analyzing data from a MIDI keyboard controller and using that data to teach the neural network to dynamically control timbre parameters in a synthesis engine. This is an example of a procedural mapping process.

Fels et al. proposed an adaptive neural network interface in the shape of a glove worn on the user’s hand that creates and adapts a mapping process based on a training phase of data collected from the performer. The glove, named Glove-TalkII, continuously maps hand

---


gestures to a speech synthesis engine. This type of adaptive mapping in neural networks is based on the following three features:

- user demonstration of which inputs should lead to which outputs to train the neural network
- new user-provided data to retrain the neural network
- once trained, the neural network runs rapidly and efficiently

The glove allows the performer to control a granular synthesis model that plays speech files containing 203 words from the English language that correspond to the gestures made by the performer’s hand movement. Five neural networks are trained; one for each set of sub-tasks required by the performer’s interpretation of controlling the speech synthesis model and produces 8036 levels of weight data used to label each entry of training data.

The Glove-TalkII showed success in synthesizing speech. Only 1% of the synthesized words were incorrect and only 5% of attempted words failed to be synthesized due to improperly recognized gestures.

1.2.3. Explicit Mapping

Explicit mapping is defined as a mapping method where a user retains direct control of the parameters that control synthesis. Hunt el al. suggest that many performer-controlled virtual instruments are explicitly mapped in one-to-many relationships. An example of this type of relationship would be a series of delay lines, where the delay times are set by one control parameter that the user controls. By uniquely scaling each input using a different scaling function, each delay line is mapped to a specific parameter space.

---


14 Fels and Hinton.

The type of relationship between groups of parameters can be characterized as one of the following: one-to-one, many-to-one, one-to-many.\(^\text{16}\)

![Figure 4. An example of mapping types in Max](image)

The one-to-one relationship is defined as a mapping method where one performer parameter controls one parameter at any given time. The many-to-one relationship is defined as a mapping method where many performer parameters control one parameter at any given time. Lastly, the one-to-many relationship is defined as a mapping method where one performer parameter controls many parameters at any given time. One-to-many relationships are the most common method found in the literature on parameter mapping.

\[1.3. \text{Music Information Retrieval}\]

Music Information Retrieval (MIR) is an interdisciplinary science with important applications to musicology, music theory, and music scholarship along with many commercial consumer applications. Searching for music features or analyzing a large body of music for similar compositional techniques are examples of how MIR can assist music research. MIR implies the use of analysis for music in a variety of representations, and much of the research in the MIR and computer music communities revolve around developing methods for quan-

\[^{16}\text{Guy E Garnett and Camille Goudeseune, “Performance Factors in Control of High-Dimensional Space.” in ICMC (1999).}\]
What are good computer representations for music? What characterizes a style of music? What distinguishes one composer from another? Can we synthesize examples of style, genre, compositional techniques, rhythmic patterns, instruments and orchestration to render queries into sounds and to better understand our representations? These are fundamental questions for music information retrieval and computer music research.

Most MIR research is concerned with describing and distinguishing timbral features of audio signals. Timbre is defined by the American National Standards Institute as the conglomerate of features which distinguish sounds that are identical in pitch and intensity. This research often concludes with the development of models or sound descriptors that can be used to classify a large number of audio samples. However, sound descriptors are rarely used for music composition and audio processing, particularly in a real-time context. This is likely due to the following factors:

- The lack of knowledge of the relationship between the descriptor and the perceptual characteristic of the sound
- The lack of standardization amongst sound descriptors in terms of output
- The lack of an existing framework by which sound descriptors can be used and understood by composers as a fundamental component of the music composition or process

Some real-time packages have been implemented in visual programming languages like MaxMSP and Pd, such as Zsa.Descriptors and the FTM/Gabor Object Library, but these are used infrequently for creative ends and are more commonly used for real-time analysis.

1.3.1. Musique Concrète and Information Theory

Music Information Retrieval began with the magnetic tape recorder and the Office de Radiodiffusion-Télévision Française (ORTF), the public radio and television center in France in the second half of the 20th Century. Pierre Schaeffer worked in the ORTF and began experimenting with the musical applications of the magnetic tape recorder in the 1940’s, which resulted in a series of pieces entitled *Etudes de Bruits* (Studies of Noise). These were the first pieces of Musique Concrète utilizing recording and tape manipulation techniques. The success of these pieces led to the development of the Groupe de Recherches Musicales (GRM), an electro-acoustic music studio devoted to the study and research of l’objet sonores (sound objects) and spectral analysis following the philosophies of Pierre Schaeffer, prompting the development of three plans for the purposes of music composition and sound analysis.

![Plan Dynamique, Plan Harmonique, Plan Mélodique](image)

**Figure 5. Schaeffer’s three maps**

Visualized using 3-dimensional space, Plan Dynamique (the dynamic map) shows the development of a sounds amplitude with respect to time. The Plan Harmonique and Mélodique

---

(the harmonic and melodic maps) show the development of timbre as a function of the entire range of audible frequencies over time and the development of pitch and tone sequences over time, respectively. The term Musique Concrète, given by Pierre Schaeffer, was intended to describe the ‘reversal’ of the typical way in which music was composed. Traditionally, composers wrote down their musical ideas using Western music notation symbols that were then executed by acoustic instruments. The magnetic tape recorder made it possible to start with a sound containing preexisting elements and abstract musical symbols from it or compose with the sound directly through montage.

The terms concrete and abstract, which are used recurrently in Schaeffer’s writings, refer to this distinction between recorded sounds (concrete sounds) and Western music notation symbols (abstract). The term l’objet sonore is used to identify the recorded sound used as the basis for a composition as well as the conceptual apparatus that describes this new approach to composition. This terminology was developed in collaboration with Abraham Moles, who was a pioneer in the field of Information Theory and an early member of the GRM. Moles was particularly interested in the perceptual characteristics of the sound object and saw the potential for a new system for encoding musical information through analysis. In 1956, he writes:

The revolution of musical doctrine, under the impact of electroacoustical techniques, Information Theory brings some leading concepts for building a new theoretical system of music. Music is one of the messages of our environment, and the human channel, as a consequence of psycho-physical uncertainty principles, determines basically the repertories of sound symbols. According to the “point of view” of the receiver, various repertories appear for both “semantic”, i.e. intelligible sequences, and “esthetic”, i.e. sensory or emotional, messages, both intricately involved in the sound sequences. It is possible to separate experimentally the information content of the various repertories, which can be distinguished according to the time scale of perception.

---


The terms semantic and esthetic used here refer to what would eventually be known as objective and subjective perceptual characteristics, or whether the information derived from a sound can be objectively quantified or subjectively described. Much of the current MIR research revolves around these characteristics, in part because of philosophical foundations of Pierre Schaeffer, Abraham Moles, and the GRM school of thought.

1.3.2. Perceptual Space and Attributes

Today, the musical characteristics derived from the analysis of audio files can be used to create a set of attributes from which a perceptual space is formed. Each audio file can be assigned a unique position within this space, allowing the distinguishing features of each recording to be visualized on a map. MIR research has led to the development of MARSYAS, a software program written in C++, that allows for rapid prototyping of computer audition research. An example of a perceptual map is shown below using Timbrespace, developed with MARSYAS, which is used for 3-Dimensional feature visualization. Each sound (vector) is represented by a single point in the 3D space. Timbrespace reveals the similarity of sounds based on their proximity in the space.

Perceptual attributes can be placed into two groups: objective and subjective. Objective attributes include elements such as tempo and rhythmic style, orchestration, and musical style. Subjective attributes focus on elements that are more descriptive in nature, such as the weight of the music, its effect on mood and energy, etc. Subjective attributes can be described in terms that non-experts might use to describe music. Musical style is by far the most important attribute to MIR researchers.

1.3.3. Parameterization

Before a perceptual map can be made, parameters need to be extracted from sound files and fed into a mapping system so that the mapping system can enable estimations of perceptual

---


27Dannenberg et al., “Panel: new directions in music information retrieval.”
distance. Once the useful parameters are identified one can attempt the construction of a suitable mapping system. The purpose of the parameterization phase is to remove as much information from the raw audio data as possible without removing the data that allows a mapping from parameters to perceptual distance. If not for the parameterization phase, the machine learning algorithms would be swamped by the sheer amount of data represented by the raw PCM data of audio files\footnote{Dannenberg et al., “Panel: new directions in music information retrieval.”} The mapping of the parameter space to the perceptual space is carried out by the mapping system using traditional classification machine learning techniques. Parameterization also takes place, albeit in a more sophisticated fashion, in the human hearing system.
1.3.4. Metadata

Modern society continuously produces an increasing amount of information about music. This information, called metadata, typically includes the artist name, producer, writer, song title, release date and has taken a growing importance in the music industry.²⁹ There are millions of music titles produced by the major music labels in the Western and non-Western world every year. To make all this music easily accessible to listeners, it is important to describe music in ways that machines can understand. Music knowledge management focuses on this issue: building meaningful textual descriptions of music and exploiting these descriptions to build efficient music access systems.³⁰

The issue of building a music description system is the subject matter of the Mpeg-7 standard. Mpeg-7 focuses only on metadata and proposes schemes to represent arbitrary symbolic and numeric information about multimedia objects such as music or movies. However, Mpeg-7 deals only with the syntax of these descriptions and not on the way these descriptions are to be produced.³¹ Figure 7 shows an extract of an Mpeg-7 description of “Blowin’ in the Wind” by Bob Dylan. This extract declares the name of the artist, the song, and its genre:

Pachet proposes an expansion of metadata classification into three distinct categories: editorial, cultural, and acoustic.³² Editorial metadata refers to metadata obtained, literally, by the editor. Practically this means that the information is provided manually, by authoritative experts. Examples of editorial metadata in music range from album information (e.g. the song “Yellow Submarine” by the Beatles appears on the Album “Revolver” issued in the UK) to administrative information such as the dates of recording, the composers or


³¹Nack and Lindsay.

Figure 7. An Mpeg-7 extract for describing information about a music title

performers. Because editorial metadata covers a wide range of information it is difficult to define precisely its scope other than by stating how it was produced.

Cultural information or knowledge is produced by the environment or culture. Contrary to editorial information, cultural information is the aggregate of subjective perspectives about music and is not prescribed or even explicitly entered in some information system. Cultural information results from an analysis of emerging patterns, categories, or associations from a source of documents.

The last category of music information is acoustic metadata. Acoustic here refers to the fact that this information is obtained by an analysis of the audio file, without any reference to a textual or prescribed information. It is intended to be purely objective information, pertaining to the “content” of the music. A typical example of acoustic metadata is the tempo, i.e. the number of beats per second. Beat and tempo extraction have long been addressed in
the community of audio signal processing and current systems perform excellently. Other, more complex rhythmic information can also be extracted, such as the metric structure (is it a ternary rhythm, like a waltz, or binary rhythm?), or the rhythmic structure itself.

### 1.3.5. Fast Fourier Transform

Traditional retrieval strategies typically rely on metadata as described above. In the case that such textual descriptions are not available or are insufficient for working on a large-scale corpus of material, content-based retrieval strategies are required. Other types of information can be extracted from a digital audio file besides the beat and tempo, such as the spectral content. This type of analysis is often utilized by performing a Fast Fourier Transform, or FFT, on the audio signal. It functions by reducing a complex waveform into its individual sinusoidal components by probing the target waveform with sine and cosine waves for a match. If the frequency and phase inside the target is the same as either of the probe signals, that signal exists within the complex waveform. If the frequency is the same but the phase lies in between the sine and cosine signals, both get a partial match. The basis for the FFT is the following formula:

\[
\sin^2 \theta + \cos^2 \theta = 1
\]

The target signal could be probed at every integer multiple of the fundamental frequency, but this would result in redundant calculations, particularly at places where either the sine or cosine wave overlap with their own multiples. This operation is optimized by removing overlapping probes that occur at zero-crossings. In short, the FFT only re-

---


quires integer multiples of power of two lengths above the fundamental, greatly reducing the number of probe signals needed. For example, a 1024 point FFT needs: $1024 \log_2 1024 = 1024 \cdot 10 = 10,240$ basic operations, while a standard DFT needs $1024^2 = 1,048,576$ basic operations.

Figure 8. Visualization of a signal in the frequency domain

The FFT is used to describe the component frequencies that belong to a given sound, and is particularly useful for timbral analysis. Many of the sound descriptors built using an FFT are used for browsing and indexing the content of sound databases, such as the models employed by Audio Commons for the Freesound.org online repository.\textsuperscript{36} \textsuperscript{37} \textsuperscript{38} \textsuperscript{39} \textsuperscript{40} 

\textsuperscript{36}Tim Brookes Andy Pearce Saeid Safavi, “Release of timbral characterisation tools for semantically annotating non-musical content,” 2019,


sic information retrieval. Besides analysis, the FFT has a wide variety of applications in computer music, such as performing resynthesis using an Inverse Fourier Transform\(^{39}\) and convolution for creating filters.\(^{40}\)

Audio applications utilizing an FFT take place over a short number of samples called a frame. The more samples used for the calculation, the more frequency information can be derived from the result. For instance, an FFT with a frame size of 2048 samples divides the entire frequency range from 0Hz to 44,100Hz (in the case of a sampling rate of 44.1kHz) into 2048 equally-spaced sections, or bins. The frequency range of each bin is determined by the sampling rate divided by the frame size, so an FFT with a sampling rate of 44.1kHz and a frame size of 2048 results in a bin size of 21.53Hz while a frame size of 1024 results in a bin size of 43.07Hz. However, using more samples results in longer calculation times and increased latency. In real-time contexts, the amount of latency corresponds with the number of samples used for the analysis, so an FFT with a frame size of 2048 and a sampling rate of 44.1kHz will result in a delay of 46.44 milliseconds between the analyzed sound and the product from the FFT calculation while a 1024 frame size results in a delay of 23.22 milliseconds. The trade-off that exists between resolution and latency is what makes real-time applications with an FFT problematic. Generally, increased frequency resolution is preferred over decreased latency since precision on time scales smaller than 46.44 milliseconds is not typically necessary.

### 1.4. Introduction of Reflexive Automation

In this section I will describe the concept I have termed Reflexive Automation and its relationship to the concepts presented above. Reflexive Automation is crucial to the development of the PnP.Maxtools objects in concept and in practice, and it demonstrates new possibilities in human-computer interaction and MIR applications. It is perhaps easiest to understand Reflexive Automation as the simultaneous use of automation, mapping, and music informa-

---


\(^{40}\)Andrew Reilly and David McGrath, “Convolution processing for realistic reverberation,” in *Audio Engineering Society Convention 98* (Audio Engineering Society, 1995).
tion retrieval in a system, the output of which results in a ‘reflex’ that is the direct result of
the input. Reflexive Automation uses information derived using music information retrieval
 techniques from an audio signal and uses that information to automate effects or parameters
 on the same signal in real-time, to change or manipulate it in some way. The manner in
 which the effects or parameters change as a result of the input is determined by the mapping
 techniques utilized in the system.

Reflexive systems can already be found in electronics and audio systems. For example,
a microphone that clips does so in response to an input amplitude level that is too high to
properly record. An over-driven amplifier results in distortion due to high gain levels that
exceed voltage capacity. Systems such as these respond to changes to the input and result
in an analogous change to the output, regardless of whether the transfer functions of these
systems are linear or nonlinear. Another less obvious example is feedback in audio systems.
The output frequency caused by a feedback loop changes according to the distance between
the microphone and the loudspeaker as well as the frequency response of the microphone.
The output frequency decreases as the distance between the microphone and loudspeaker in-
creases and increases when the distance between the microphone and loudspeaker decreases.
All of these systems utilize music information retrieval, albeit in a very non-purposeful man-
ner, and a system of mapping that determines how the system automates or responds to the
input.

The term Reflexive Automation is, in part, inspired by a naturally occurring reflexive
system known as the patellar reflex test. In this analogy, the plexor is the incoming signal,
or the action from which information is derived. Once the information is retrieved through
contact with sensory neurons in the patellar, it is transmitted through the neuromuscular
system and triggers an impulse in the motor neurons in the quadriceps muscles. The reflex
arc occurs at the level of the spinal cord, meaning that the associated movement occurs
without involvement from the brain. The result is a short jerk in the knee of the patient
that happens automatically and involuntarily. It is an autonomous reaction that occurs as
the direct result of a system of mapping of neurons in the body for the purpose of responding to external stimuli.

In the context of computer music with live instruments, this conceives of audio processing as a gestural analog to the gestures of performers, as extensions of the sounds of their acoustic instruments. Like the rapid movement of the knee, the effect on the resulting sound changes as a response to the input, to the specific and unique sounds of the performance. The system must be designed to require no additional involvement or supervision from a controlling force to operate internally on the system, such as the composer or additional performers. In other words, it requires the absence of the involvement of the ‘brain.’ For a system to exhibit Reflexive Automation, it must possess the following characteristics:

- Information must be retrieved externally, and preferably, from live instruments
- Mapping can be linear or non-linear, but must reflect an analogous relationship between the input and output
- Automation must be dependent upon external information only

The reflexivity of a system can be determined by how closely the characteristics of the system resemble the characteristics described above. If a system possesses two of the characteristics described above, it is not fully reflexive. For example, if the amplitude of a flute is 25
being used to control the delay time of the flute in milliseconds, but the composer maintains
direct control over how the amplitude values from 0-1 are scaled and changes them through-
out a performance, this system is not fully reflexive or autonomous because the mapping
method is supervised and is not fully dependant upon external information. A fully reflexive
system would require the mapping method to be controlled by the system itself or another
musical parameter controlled by the flautist.

1.4.1. Affordances of Reflexive Automation

Much live electronic music today requires performers to utilize several controllers, such as
footpedals, while playing their instruments. The often overly elaborate technical require-
ments and physical demands of live electronic music can often alienate and overwhelm po-
tentially interested musicians. It also requires performers to wear the hats of technologist,
audio engineer, and performer simultaneously. By relying on specific qualities of the sound
of an instrument as the only external controller, the composer will inherently become more
sensitive to the technical demands of the performance and the performer will have a better
working knowledge, through their instrument, of the system for which they are expected to
interact. In this way, Reflexive Automation has the potential to be much more accessible to
performers.

Reflexive Automation also offers a more intimate relationship between the performer and
the computer. Computer programs used for live performance often require the performer
to synchronize with a variety of cues, perform with a stopwatch and adjust the speed of
their performance to arrive at a certain place in the score at a specific time, or perform
alongside a track for which sometimes vague or unspecific graphic notation exists. These
aspects of live electronic music can place the role of the performer as secondary, or as
accompaniment, to the electronics. Failure with regards to live electronic performance in
this context often means not performing the part “correctly” with regards to the score —
meeting all of the synchronization points, alignment, etc. — and does not often refer to
the subtleties of performance interpretation. Failure with regards to performing Chopin,
for example, is related to the specifics of the interpretation, whether by tempo fluctuations (rubato), dynamic fluctuations, or other subtle distinctions not expressly notated in the score.

In the case of many instruments, traditional music notation already affords much of what can be retrieved through MIR models, such as dynamics, rhythm, and pitch. The language of notation already familiar to performers can be preserved and better utilized to determine aspects of the electronics. For instance, in a reflexive system that analyzes the amplitude of a signal and uses that to control the amount of reverb that is applied to the signal, the traditional dynamic markings $p$ and $f$ already exist and work well for achieving this end. However, using $fff$ will result in much more reverberation. Aleatory in notation can still be utilized when and where indeterminacy is desired, but much of how the instrumentalists conceive of the capabilities of their instrument can be preserved by utilizing familiar notation when and where it can be used to achieve specific interactions.

With regards to timbre, some notation exists such as *sul ponticello* and *sul tasto* on string instruments. Or, wind instruments can utilize a wide range of effects such as harmonics, multiphonics, noisier sounds such as flutter tonguing and singing and playing simultaneously. Each of these has their own unique spectra. Outside of this, performers have a wide range of terms that they use to classify the types of timbre that can be produced on their instruments, such as brightness and warmth. While these are perceived characteristics of timbre as opposed to a purely objective timbral analysis, they provide a useful common language for notation. In most cases, it is not necessary to include a separate notation for electronic elements in the score alongside the acoustic instruments, because the notation for the performer and the resulting electronic manipulation can be written using the same symbols or text. Text in the front matter of the score will suffice, in most cases, for conveying the behavior of the electronics to the performer.

Reflexive Automation acknowledges that technology now allows for the simplification and abstraction of electronic music and computer music software for the betterment and ease of
The intimate and analogous relationship between the source of the sound and the proceeding effect on that sound is its primary point of concern, while accessibility, elegance, and abstraction is its secondary point of concern. Building systems that are accessible, elegant, and abstract complexity for the benefit of the performer is the responsibility of the composer, while defining the needs and limitations of the instrument and the notation is, in some small part, the responsibility of the performer. For now it seems that the building of a reflexive system is and should be a dialectical and collaborative effort.

1.4.2. Strategies for Building Reflexive Systems

The following strategies for building reflexive systems are not intended to supplant other creative interactions and processes not explicitly mentioned, but are instead intended be considered a starting point. These strategies will hopefully lead to more elegant, interesting, and commonplace interactions over time through the combined efforts of many brilliant composers and performers, just as any genuinely profound artistic achievement of thought or practice is created through the intellectual and physical labor of entire communities.

Any attempt to construct a reflexive system should begin with audio analysis. It is through audio analysis that the range of features present within a particular sound can be understood and quantified. Features which vary dramatically are better equipped to control a wider range of possible outcomes or control a narrow range of possible outcomes with greater resolution. For example, the range of the grand piano is 8 octaves from A0 at 27.5Hz to C8 at 4186Hz. This range is significantly larger than the range of the flute, which spans from C4 at 256Hz to C7 at 2096Hz. Since the range of the piano more than doubles that of the flute, it is possible to construct value ranges with more significant digits using the number of distinct pitches possible on the piano rather than on the flute. If the ranges of both instruments were scaled between 0 and 1, it would be possible to divide it into equal increments of 0.011364 using the number of available notes on the piano as opposed to equal increments of 0.027027 using the number of available notes on the flute. In short, features which yield a wider range of possible values can be used to create higher resolution
counting systems and ultimately finer control musical parameters with greater specificity and precision.

In terms of mapping, it is perhaps easiest to consider using the gestures of performers to control audio processors rather than individual features that can be derived from the incoming audio signal. Often times, musical gestures are comprised and understood as changes to several features of the sound simultaneously. By using a mapping method of many-to-one, accompanied by scaling and other functions, multiple features can control a single audio effect as well as more specific interactions between the performer and the reflexive system. In short, it is possible to define any gesture as the sum of the changes between component features. For example, digital instrument designers are keenly aware of the idea of acoustic viability, or the ability for digital instruments to behave like acoustic instruments in terms of amplitude and spectral energy. Acoustic viability is often achieved by relating the increase of the key velocity value of a MIDI controller with an increase of spectral energy. This is an example of a type of gesture built from the composite features of amplitude and spectral energy.

When thinking about gesture it can be helpful to define a series of x-dimensional regions, where x is determined by the number of features used to control an audio process. A 1-dimensional gesture would simply map one feature onto an audio process and could be visualized as a slider that corresponds to the output value of that feature. A 2-dimensional gesture would combine 2 features and control an audio process based on the values of both features whether they are summed, averaged, scaled, or inverted. This could be visualized as a plane where the X and Y positions correspond to each feature and the distance from the top-right corner of the plane would determine the amount processing on the signal.

Reflexive systems have the capacity to sum, average, or perform multiple operations on inputs from more than one instrument simultaneously. If they are processed with more than one audio effect, it is possible to map a different feature to each control parameter, one of which could be an average of composite features from more than one instrument. This is an
example of a cooperative reflex rather than an individual reflex. Using the flute and piano in the previous example, the amplitude of both instruments could be halved, summed, and used to control the amount of reverb on both simultaneously. When one instrument is played loudly and the other softly, the amount of reverb would be minimal. However, when both instruments are played at their loudest the amount of reverb would be at its maximum.
CHAPTER 2. PRECEDENTS FOR REFLEXIVE AUTOMATION

2.1. Automation in Music

Modern digital artists employ a wide variety of automation techniques such as Markov Chains and machine learning algorithms to create deterministic processes by which musical elements such as pitch, rhythm, and timbre change over time. Some of these models are capable of responding to real-time inputs that are analyzed from an external source, while the complexity of these systems is determined by a number of factors, such as whether they are expected to perform in real-time, whether they are mimicking the expressiveness of human performers, or whether they are non-linear. This section presents artworks that employ complex automation techniques along these lines.

2.1.1. LEMUR

Robotic instruments make music with the use of motors, solenoids, and gears and come in many forms including robotic pianos, percussion instruments, turntables, plucked and string instruments, and wind instruments. They are often designed to perform or create music automatically using specially designed algorithms that govern the nature of the performance and can often perform with precision and speed that is impossible for human performers to replicate. Robotic instruments have been created that are even capable of listening and improvising in real-time alongside a human performer. While robotic instruments are typically capable of performing complex music with ease, the ability of the robots to behave expressively and collaboratively with humans is more difficult to accomplish. This concept, known as Mechatronic Expression, is the primary goal of robotic instrument designers.

The League of Electronic Musical Urban Robots, or LEMUR, is an artist collective dedicated to creating robotic musical instruments. LEMUR has created several percussion

---

1 Ajay Kapur, “A history of robotic musical instruments,” in ICMC (Citeseer, 2005).
and stringed musical robots, such as the TibetBot which plays Tibetan singing bowls, the !rBot which plays goat-hoof rattles, and the GuitarBot which is an electric guitar instrument.

Figure 11. The TibetBot, !rBot, and GuitarBot

These robots have been used in automated and interactive art installations, live performances, and generated musical compositions. They are all controlled by various MIDI hardware and software controllers and can be performed live, giving human performers inhuman capabilities in terms of speed, pitch, and expression. They can also be played using generative or sequencing algorithms for improvisation.

2.1.2. Haile
In recent years, the software for musical robots utilizes machine learning algorithms which are used to make informed decisions based on the initial training data used to train the models. In the case of live performance with musical robots, these are used to classify components of a performance from the human performer or an external source and respond according to the input.

In 2007, Weinberg et al. developed a musical percussion robot named Haile that listens to live human performers, analyzes aspects of their playing in real-time, and then uses this
to improvise in a collaborative manner. Haile’s appearance is anthropomorphic to better visually communicate the idea that it interacts with humans through listening and analysis.

Figure 12. Haile’s anthropomorphic design

Using directional microphones embedded inside the drums of human performers, Haile detects several perceptual aspects such as hit onset, onset amplitude, pitch, tempo, and note density. The manner of interaction is categorized into several modes: imitation, stochastic

---

transformation, simple accompaniment, beat detection, and perceptual accompaniment and transformation. Each of these modes governs how Haile responds to the human performers, either by playing in unison over a looped passage, accompanying the human performer if they are playing many notes, or playing specific rhythms that imitate the human performer to create a call-and-response.

2.1.3. The Robo-Cajon

The Robo-Cajon is a musical robot created by the author that is capable of live performance with a human performer. It is a custom-built wooden cajon mounted with two push/pull solenoids that receive input from a separate cajon is mounted with piezo contact microphones. Rhythmic data from the human performer is classified using a Multi-Layer Perceptron (MLP) and a Hidden Markov Model (HMM) in Max and is used for prediction and generating rhythmic patterns based on this prior classification. The Robo-Cajon’s improvisation is based on the human performer and controlled by the human performer with a footpedal.

The novelty of the Robo-Cajon performance is that it both improvises alongside a human performer and behaves like an instrument which can be controlled by the human performer simultaneously. Timing data is calculated and classified by the MLP as levels of subdivisions as small as 16th notes using the tempo provided in the global transport in Max. This data is stored as a list that continuously grows as new classifications are made until triggered by the human performer. Once triggered, all of the stored timing data is fed into an Hidden Markov Model which begins outputting new timing values based on these inputs to the Robo-Cajon through Arduino. The performer can also make the Robo-Cajon loop a particular rhythmic figure, pause for a brief period of time, or take a solo and play a rapid succession of notes. These different components allow multiple methods of control to shape the overall phrasing and musical structure of the improvisation.

---

5Austin Franklin, “The Robo-Cajon: An Example of Live Performance with Musical Robotics,” 2021,
2.1.4. Telematic Piano

The Player Piano is one of the earliest examples of an automatic musical instrument. Originally, compositions were punched into paper that was then read and performed by the piano. Now, automatic pianos exist that are capable of parsing through the instructions contained in MIDI files that govern pitch, volume, and other parameters.

In 2020, the Laptop Orchestra of Louisiana at Louisiana State University was inspired to overcome the deficit of live events during the COVID-19 Pandemic. This prompted a ‘Concert of Telepresence’. Shown below, the Telematic Piano was a work which incorporated

---

6Alyssa Michaud, “After the music box: A history of automation in real-time musical performance,” 2020,
the automation capabilities of a Yamaha Disklavier Mark IV Player Piano and the networking capabilities of the internet.

There were four performers for the work, each in separate locations, which were networked as clients to the same MaxMSP patch host which was in the same room as the player piano. The patch was designed to send MIDI note values from the performers to the player piano, where they would be performed in real-time. Rather than playing MIDI keyboards, performers sent MIDI renderings of excerpts of famous classical pieces such as Chopin’s Fantaisie Impromptu and Beethoven’s Symphony No. 5.

2.2. Mapping in Music
Mapping is a fundamental principle in computer music because many of the technologies artists use are modular. This often results in mapping systems that are incredibly complex and can often incorporate many software and hardware systems simultaneously. Several technologies, such as machine learning, utilize mapping principles to create a connection.

---

7Jesse Allison, “EMDM Online Concerts in the Pandemic,” 2021,
between the input and output according to the initial training data. This section presents work that emphasizes mapping systems that are complex and dynamic, gestural, and operate in real-time.

### 2.2.1. ML.Lib

Machine learning is a field of research that has grown in popularity over the past several decades, particularly within the computer music community. The interest in these technologies is due to their ability to learn how to respond to training data sets, or their procedural mapping capabilities. The cross-platform open source software package in MaxMSP and Pure Data known as ml.lib is useful for employing a variety of machine learning algorithms in real-time. This makes it ideal for computer music and live performance needs which depend on the automatic mapping and automation of musical gestures.

![Figure 15. Architecture of ml.lib workflow](image)

The objects in this package for Max and Pure Data fall into the following categories:

- **Pre-processing**: processes data prior to model training
- **Post-processing**: processes data after being output from models

---

• Feature extraction: extracts features from data that can be used to train models
• Classification: takes vectors as inputs and outputs values which represent the class of the input
• Regression: performs mapping between input and output vectors
• Clustering: partitions vectors into clusters

Several experiments were conducted with these objects to test their functionality. These included a test to classify the orientation of a mobile phone using gyroscopic data as inputs, a test to classify gestures from accelerometer data from a mobile phone using a Dynamic Time Warping algorithm (DTW), and a test to control synthesis parameters such as the fundamental frequency, amplitude, filter bandwidth, and vibrato frequency using the outputs from the classification and DTW algorithms.

2.2.2. MnM: A MaxMSP Mapping Toolbox

The MnM Mapping Toolbox is the first set of objects with help patches in Max for the purpose of implementing complex mappings using singular value decomposition. The goal is to give programmers and composers the ability to build and implement many-to-many mapping systems through the combination of several uses of one-to-one, one-to-many, and many-to-one mappings. Many of the objects included in this toolbox operate using matrix modulations such as inversion, linear regression, and principal component analysis. These modulations are achieved using Single Value Decomposition (SVD) methods specified with training examples that perform linear transformations on the incoming data. This means that incoming data can be stretched, compressed, scaled, or inverted to map onto multiple high and low output values simultaneously.

2.2.3. Wekinator

Wekinator is an intermediary software solution that provides a system the ability to apply machine learning techniques to musical performances in real-time. The program combines real-time interaction with fundamental machine learning concepts such as classification, linear regression, and the evaluation of results. Wekinator is the first real-time machine learning system for musical applications. It includes a GUI where the user creates and modifies training data sets and configures learning algorithms, and the results of the analysis can be sent using the OSC protocol to other programs to control audio synthesis, visuals, or any number of other processes.

2.2.4. Air Instruments

Jensenius et al. conducted an experiment using gestural analysis from videos images of participants air playing instruments in order to replicate the sound and gesture of the instrument being mimicked. Imitation through physical action is considered fundamental to learning and socialization, and in the context of music there is a significant link between auditory and physical gesture. Air instrument playing, known as motormimetic sketching, refers to the imitation of sound producing gestures as well as the approximate nature of the imitation. In order to gauge motormimetic sketching, participants with different musical and movement-related training were recruited and asked to air play piano music covering various performance techniques and styles, such as Chopin’s Scherzo no. 2 in Bb minor, Scriabin’s Sonata no. 5, Beethoven’s 3rd Piano Concerto, and others.

---


11Rebecca Fiebrink, Daniel Trueman, Perry R Cook, et al., “A meta-instrument for interactive, on-the-fly machine learning,” 2009,


Several features of the participants' gestures were measured, including the overall movement, the pitch-space of the hands as it relates to the keyboard, the synchronization with musical events, and the size and speed of gesture related to the loudness of the music.

2.2.5. Data Sonification and Musification

Sonification and musification refer to the audible or musical representation of data. Sonification is generally described as “the use of synthetic non-verbal audio to support information processing activities,” and is used to transform inaudible information into audible information for the purpose of reflecting properties or trends in data. On the other hand, musification implies the musical representation of data through elements of tonality, such as using scales and chords, in an attempt to engage listeners through musical interactions that may occur as a result of changes in the data. Unlike sonification, musification often results in higher-level musical features such as polyphony or functional harmony as a result.

---


of systems of mapping that determine characteristics of the sound through the interpretation of data.\textsuperscript{17}

There are many similarities between data sonification, musification, and Reflexive Automation. Firstly, the resulting sound is the direct result of a system of mapping utilizing mapping methods that control audio synthesis and/or effects. As the input data is being sonified, the perceptual changes are the consequence of reflexes that occur through automation. While MIR is not a part of this process, the input data itself often reflects information retrieved from the physical world through various research.

In 2016, Ian Walker was commissioned by the ClimateMusic Project to compose a piece of instrumental music using the trends in climate data from the year 1897 to 2125.\textsuperscript{18} The data used for the work was supplied by the Intergovernmental Panel on Climate Change (IPCC) from known data as well as predicted future numbers.\textsuperscript{19} The almost 30-minute work was constructed into sections approximately one minute in length, with each representing 10 years on earth. The carbon dioxide levels in the air determined the ensemble tempo, the surface air temperature determined pitch and harmony, the Earth energy balance determined the ensemble’s volume and modulation and distortion amount on a synthesizer, and the ocean pH levels determined the shape of musical form.

The first 10 minutes of the work were relatively calm and static as the environment wasn’t yet affected by quick Co2 rise and other metrics. As the C02 levels began to rise more quickly, the music tempo increased. The harmony gradually became more dissonant and pitches detune as the temperature increased. Then, when the Earth’s energy balance skewed, distortion and modulation was introduced. Finally, as the ocean pH levels plummeted the music lost all sense of structure.


\textsuperscript{18} https://climatemusic.org/

\textsuperscript{19} https://www.ipcc.ch/data/
The exponential rate of change in the data was reflected in the music, similar to the reflex that occurs through timbral changes in the qualities of analyzed sounds. While there are many other examples and methods of sonification and musification, one overarching difference is the time scale at which the reflex occurs. As reflexive systems require real-time inputs, sonification and musification allow for the offline mapping and automation of input data in order to create reflexes. This typically results in the sonification of large amounts of data, decades worth, that would simply not be practical in a real-time context.

2.3. Music Information Retrieval in Music

Music Information Retrieval primarily exists in the form of non-real-time algorithms designed for the classification of large online repositories of music. Many real-time algorithms and descriptors are currently studied for the purposes of music creation, but there are several barriers to creating and implementing these practically. Some of the challenges include: a lack of knowledge of the descriptors and the perceptual characteristics of the sound, the fact that a sound is often characterized as changes to multiple features simultaneously (one descriptor is generally not sufficient for achieving any meaningful musical interaction), and the lack of a large choice of descriptors available to musicians. This section presents Music Information Retrieval techniques and concepts centered on the utilization of these techniques for music composition and real-time applications.

2.3.1. Zsa. Descriptors

There are several real-time and offline examples of music information retrieval strategies. For real-time applications, popular software environments such as MaxMSP, Pure Data, and SuperCollider offer a selection of tools. Malt et al. developed a collection of real-time sound descriptors for analysis in Max that includes models that can detect objective features such as the spectral centroid, spectral spread, and spectral roll-off using an FFT.

---

21 Malt and Jourdan.
2.3.2. The Freesound Player

The Freesound Player is a digital instrument developed by the author in MaxMSP that uses the Freesound API to make requests for as many as 16 sound samples which are filtered based on sound content. Once the samples are returned they are loaded into buffers and can be performed using a MIDI controller and processed in a variety of ways. The goal of The Freesound Player is to create a robust compositional tool that both utilizes this repository of collective commons audio samples while streamlining the compositional process by allowing composers to more quickly narrow down search results using the API and allowing the samples to be quickly recalled, processed, and performed within the MaxMSP environment.

![Figure 17. GUI for the Freesound Player](image)

The Graphical User Interface (GUI) for The Freesound Player is shown above. Querying a search using a specific filter along with a value range between 0-100 (100 being the most intense and 0 the least) will return sound samples tagged with the search term as well as the timbral content identified by the API. The Freesound API documentation includes a variety of timbral models, or filters, described by Audio Commons such as depth, brightness, boominess, sharpness, and more. The PnP.Maxtools package implements all of the Audio Commons timbral models in a real-time context.

---


23 Andy Pearce, “Release of timbral characterisation tools for semantically annotating non-musical content.”
2.3.3. Voyager

Voyager is an interactive software program designed by George Lewis that dialogues with a human performer that is improvising in real-time. The computer analyzes aspects of the human performer’s performance and uses that analysis to guide an automatic composition capable of complex responses and independent musical choices that arise from internal processes\(^\text{24}\). On the analysis side, Voyager uses pitch following algorithms that detect and parse the sounds of the acoustic instruments into MIDI data streams. It then identifies several aspects, such as pitch class sets, velocity, probability of note activity, and the time interval between notes, and uses this to generate novel musical structures which are performed by Voyager in real-time.

Lewis was inspired to create Voyager, in part, through his experiences with the Association for the Advancement of Creative Musicians (AACM), which was founded on Chicago’s South Side in 1965 by four African American composers. The social philosophies of the AACM of innovation and collaboration are at the core of Voyager. His sensibilities as an improviser and collaborator led him towards conceiving Voyager as a program capable of dialogic communication. Lewis states:

> What people play into the computer should come out of the computer with some aspect of the emotional and other messages that are part of the sound intact. What people are playing are carriers for another signal; the sounds we hear aren’t the main thing... You have to approach it on the level of emotion, on the level of creating dialogue.\(^\text{25}\)

The top-level phrase behavior of Voyager is shown in Figure 18. This is one of the key features of Voyager that is designed to give it its own unique sound, similar to how musicians are often identified through recurring themes, interpretations, fluctuations, and other stylistic features in their performances. This phrase behavior is achieved, in part, based

---


\(^{25}\)Paul Steinbeck, “George Lewis’s voyager,” in *The Routledge companion to Jazz studies* (Routledge, 2018), 261–270.
Figure 18. The top-level phrase behavior word of Voyager

on how Voyager responds to inputs in order to generate phrases. It uses 64 asynchronously operating single-voice MIDI-controlled streams that generate music simultaneously. From here, a global sub-routine is designed to specify the overall behavior of all asynchronous voices, such as which ones are grouped and how they transition from one phrase to the next.
CHAPTER 3. PNP.MAXTOOLS OVERVIEW

3.1. Package Design

The first release of the PnP.Maxtools package contains 43 objects utilizing a wide variety of signal processing techniques and algorithms. These can be combined in numerous ways to achieve fully autonomous control over musical parameters. The objects are grouped by function into the following categories: filters, descriptors, additional controls, effects. Each object is designed to be used interchangeably with any other from the same category, allowing users to quickly edit and implement the package objects in a ‘plug and play’ style, hence the prefix ‘PnP’ in the package title.

An incomplete description of all package objects is presented here, since some of the objects utilize modified versions of techniques that are common throughout the computer music paradigm, such as convolution, plate reverb, granular synthesis, and more. Instead, the objects which are brand new to the MaxMSP environment are described in detail here with an emphasis on the timbral descriptors. The objects in this category are novel and represent the majority of research, development, and testing for the package. These objects make the PnP.Maxtools package the first set of objects in MaxMSP to utilize real-time subjective MIR algorithms for the purposes of music composition and live performance. A complete list of all PnP.Maxtools objects is detailed in Appendix A.

3.1.1. Framework

Figure 19 demonstrates the proposed ideal signal chain for the package objects, where the dashed lines indicate MSP patch cables and the solid lines indicate Max patch cables. MSP patch cables route audio data from one object to the next at the signal vector rate, while the Max patch cables route event-based or control data. While this is not the only possibility for automating control parameters, it is provided here as the starting point to understand and use the objects collectively.
All of the signal analysis and sound descriptors are designed to output a normalized floating point number between 0-1 based on the content of the incoming signal, while the effects are designed to receive numbers between 0-1 as control data. When the same signal is used for both a descriptor and an effect, a change in the content of the sound is represented analogously with a change in the audio effect. The standardization of inputs and outputs allows for a large selection of objects to be used interchangeably, empowering composers with numerous musical possibilities.

3.1.2. Package Homepatcher

The main page of the *PnP.Maxtools* package contains the full list of all available objects, grouped by category, along with a brief description of what the object does when the mouse is hovering over the name. Clicking on an object opens the help file for that object where
additional information, such as how it functions, its inputs and outputs, and arguments is
detailed. The package homepatcher is shown in Figure 20.

Figure 20. PnP.Maxtools package homepatcher

3.1.3. Documentation

In Max specifically, many of the existing packages (such as Zsa.Descriptors\(^1\), MnM Toolbox\(^2\), FTM/Gabor Object Library\(^3\), Ml.Lib\(^4\) and fiddle\(^-\) and bonk\(^-\) by Miller Puckette\(^5\)) require

\(^1\)Malt and Jourdan, “Zsa. Descriptors: a library for real-time descriptors analysis”
\(^2\)Bevilacqua, Müller, and Schnell, “MnM: a Max/MSP mapping toolbox”
\(^3\)Schnell and Schwarz, “Gabor, multi-representation real-time analysis/synthesis”
\(^4\)Bullock and Momeni, “Ml.lib: robust, cross-platform, open-source machine learning for max and pure data.”
a certain level of pre-existing knowledge of the models which can make them difficult to use. Most of the documentation is also not readily equipped with examples that demonstrate creative implementation, and to the extent that they are, there is usually a different creative approach associated with each object because the inputs and outputs are not compatible with other objects from the same package. In short, they are often not able to be used interchangeably and require additional objects, such as scaling or transform functions, to be mapped properly. Furthermore, none of these packages offer specific musical effects that are designed to be used in conjunction with the descriptors.

In order to further demonstrate the automation of control parameters, the documentation for \textit{PnP.Maxtools} objects provides a demo patch with several tabs including a demonstration patch where objects from each category can be randomly implemented, a patch that describes the framework and each object category in more detail, a patch demonstrating cooperative descriptors, a more musical example where effects are chained together, and an analysis patch utilizing all of the descriptors. The examples provided in these tabs can be used as starting points for new creative projects, or they can be copied, pasted, and directly implemented into existing patchers.

The analysis tab contains several ways to visualize the content of the incoming audio or file. There is a set of sliders that corresponds to each timbral descriptor, a pair of 2-dimensional planes that corresponds to amplitude and frequency distribution, and frequency range and spectral roughness. It also includes onset and BPM detection visualizations. The final visualization is a spectroscope which tracks the spectral centroid and spectral spread of the signal. If data is noisy, the smoothness can be set which filters out rapid value changes. The analysis tab is useful for experimentation to see which descriptor would be best to use for controlling effects or analyzing a variety a similar sounds for comparison.

3.2. Filters

The filters provide pre-processing functionality and are intended to be used to restrict the range of frequencies or remove unwanted sounds from an audio signal before analysis. Many
of the filters are implemented using the pfft object in MaxMSP using a frame size of 2048 samples. This object is a method for implementing a Fast Fourier Transform that manages windowing and overlapping and is an efficient solution for real-time implementation with minimal latency. These objects function by gating frequency bins, or allowing certain ones to pass through the pfft object unaffected while multiplying the real and imaginary values of other bins by zero. In the pfft object subpatcher, the furthest right outlet of the fftin object indexes the signals bin number and corresponds to the real and imaginary signals for that bin. Using a few boolean operators such as $<$, $<=$, $==$, etc. ("\~" denotes signal operators), it is possible to isolate any number of bins or bin regions for calculation.

---

6 "pfft Reference," https://docs.cycling74.com/max7/refpages/pfft~
3.2.1. pnp.binpass

pnp.binpass is a filter modeled after a classic ‘bandpass’ filter that utilizes an FFT to remove frequencies from an input signal outside of a designated low and high frequency range. Since it can only remove entire bins in the frequency domain, it calculates which bins contain the entire low and high frequency range. This means that the output signal will sometimes contain additional frequencies outside of this range, but only the frequencies that lie in the same bin as either the low or high frequency arguments.

Figure 22. FFT implemented in pnp.binpass
3.2.2. pnp.notch

pnp.notch is a filter modeled after a classic ‘notch’ filter that utilizes an FFT to remove frequencies from an input signal inside of a designated low and high frequency range. Since it can only remove entire bins in the frequency domain, it calculates which bins contain the entire low and high frequency range. This means that the output signal will sometimes contain additional frequencies outside of this range, but only the frequencies that lie in the same bin as either the low or high frequency arguments.

Figure 23. FFT implemented in pnp.notch
3.2.3. pnp.overtone

pnp.overtone is an FFT filter that takes a fundamental frequency as an argument and filters an incoming signal using integer multiples up to 8*fundamental. Each frequency is given a separate outlet that can be combined with others or used independently.

![Figure 24. FFT implemented in pnp.overtone](image)

3.2.4. pnp.pitchclass

pnp.pitchclass is an FFT filter that takes a fundamental frequency as an argument and filters an incoming signal up to 7 octaves above the fundamental (128*fundamental). Each octave is given a separate outlet that can be combined with others or used independently.

![Figure 25. FFT implemented in pnp.pitchclass](image)
3.3. Sound Content Descriptors

Many sound descriptors are implemented based on models described in AudioCommons “Release of timbral characterization tools for semantically annotating non-musical content.” This release describes models that analyze perceptual characteristics of sounds and were developed and fitted using linear regression based on subjective listening experiments to gauge their effectiveness. Each model described by AudioCommons was either developed based on these experiments or on existing models and literature pertaining to the acoustic correlates of timbral attributes. They have been adapted for real-time use and are implemented in MaxMSP using a variety of methods, including pfft and gen. Gen is an environment in Max that compiles the contents of the patch in C++ code. It also has the benefit of being able to run at the audio sampling rate rather than the audio vector rate, making it ideal for very low level audio processing. For models that implement an FFT a frame size of 2048 samples is used.

3.3.1. pnp.boominess

A boomy sound is one that conveys a sense of loudness, depth and resonance. Several boominess calculations have been proposed, such as the Booming Index as described by Shigeko Hatano and Takeo Hashimoto in “Booming Index as a Measure for Evaluating Booming Sensation.” Boominess is a useful metric in the automobile industry where it is used to partially quantify the sound quality of engine noise. In this case the booming sound of an engine is thought to relate specifically to the engine harmonics. The method of calculation

---

7 Andy Pearce, “Release of timbral characterisation tools for semantically annotating non-musical content.”
9 David Gerhard, Audio signal classification: History and current techniques (Citeseer, 2003).
10 https://docs.cycling74.com/max8/vignettes/gen,opic
Hatano et al. proposes makes use of the ‘order’ analysis of a sound, relating the sound from the source to the revolution speed or r.p.m. of operation. From this the fundamental frequency and harmonics can be determined and the loudness of these calculated. The AudioCommons boominess model is a direct implementation of the Hatano booming index algorithm.

In MaxMSP, pnp.boominess calculates the apparent boominess of an incoming signal based on the sharpness model described by Fastl and Zwicker in “Psychoacoustics: Facts and Models.” However, Fastl et al. proposes that boominess is a measure of the low frequency content of a sound rather than high frequencies; the greater the proportion of low frequencies the greater the ‘booming’ sound. So boominess can be considered to be the opposite of the sensation of sharpness. Using Fastl and Zwicker’s approach boominess can be calculated as:

\[
boominess = 0.11 \frac{\sum_{n(0Hz)}^{n(13,500Hz)} x(n) * g(z(n)) * z * 0.1}{\sum_{n(0Hz)}^{n(13,500Hz)} x(n)0.1}
\]

where \(N\) is the total spectral loudness, \(g(z)\) is the weighting factor for boominess as a function of the critical-band rate, and \(dz\) is a scaling factor. Only for critical-band rates less than 22 bark does the weighting factor increase from unity to a value of 4.5 at the end of the critical-band rate near 0 bark.

**3.3.2. pnp.brightness**

A bright sound is one that is clear/vibrant and/or contains significant high-pitched elements. pnp.brightness calculates the apparent brightness of an incoming audio signal. Brightness is a timbral attribute that has been studied in some detail. It has been shown by Schubert

---


14Russell Mason Andy Pearce Tim Brookes, “First prototype of timbral characterisation tools for semantically annotating non-musical content,” 2017,
and Wolfe[15] Poirson et al.[16] and Grey and Gordon[17] that the spectral centroid is a measure that correlates with perceived brightness. However, some research by Omori[18] Lartillot and Toiviainen[19] Juslin[20] and Laukka et al. [21] also suggests that the ratio of high frequencies to the sum of all energy is a better predictor. In recent work, Pearce[22] surveyed existing models and developed a new model of brightness incorporating both a spectral centroid variant and a spectral energy ratio. This model calculates the upper spectral centroid as the spectral centroid of the frequencies between 3KHz and the Nyquist frequency:

Figure 26. Weighting, g(z), as a function of critical band rate for boominess

---


[22] Andy Pearce, *“First prototype of timbral characterisation tools for semantically annotating non-musical content”*. 56
Upper spectral centroid = \[
\frac{\sum_{n(3kHz)}^{n(Nyquist)} f(n)x(n)}{\sum_{n(3kHz)}^{n(Nyquist)} x(n)}
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. The model also calculates the ratio of energy between 3 kHz and the Nyquist frequency compared to all energy up to the Nyquist frequency:

\[
\text{Ratio} = \frac{\sum_{n(Nyquist)}^{n(Nyquist)} x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}
\]

where \(n(Nyquist)\) is the frequency relating to the Nyquist frequency. The Max implementation is a direct implementation of the model described by Pearce which calculates both the spectral centroid as the spectral centroid of the frequencies between 3KHz and the Nyquist frequency and the ratio of energy between 3kHz and the Nyquist frequency compared to all energy up to the Nyquist frequency.

3.3.3. \texttt{pnp.centroid}\textsuperscript{−}

\texttt{pnp.centroid} calculates the spectral centroid in hertz, or the barycentre of spectra, of an incoming audio signal\textsuperscript{23}. The spectral centroid is the weighted average frequency for a given subband, where the weights are the normalized energy of each frequency component in that subband\textsuperscript{24}. The spectral centroid is a well known timbral feature calculated as:

\textsuperscript{23}Malt and Jourdan, "Zsa. Descriptors: a library for real-time descriptors analysis."

\textsuperscript{24}Jia Min, Karen Kua et al., "Investigation of spectral centroid magnitude and frequency for speaker recognition.,” in Odyssey (2010), 7.
\[
\text{Spectral centroid} = \frac{\sum_{n(0Hz)}^{n(Nyquist)} f(n)x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin.

### 3.3.4. pnp.depth^−

A deep sound is one that conveys the sense of having been made far down below the surface of its source. pnp.depth^− calculates the apparent depth of an incoming audio signal.\(^{25}\)

Whilst the attribute of depth is mentioned in several academic papers, only Audio Commons has proposed a model and suggested acoustic correlates. However, an online experiment by Cartwright et al. called Social-EQ asked subjects to submit a timbral descriptor together with an appropriate setting on a 40-band graphic equalizer that demonstrates that descriptor.\(^{26}\) Six subjects chose to submit the term deep. The 40-band equalisation treatment submitted by each subject is shown in the figure below. The mean equalization of all subjects, and 95% confidence intervals, are shown in the thicker black line.

There is a clear trend in this Figure that shows that all subjects’ EQ treatments emphasized the low frequency content of the signal. Since there is a large degree of commonality in these EQ treatments, it is likely that timbral depth is related to having emphasized low frequency content. Pearce suggests that a suitable model for depth would be to analyse: 1) the spectral centroid of the lower frequencies (energy pulling towards the low-end); 2) the

---

\(^{25}\) Andy Pearce, “First prototype of timbral characterisation tools for semantically annotating non-musical content.”

Figure 27. Social-EQ graphic equaliser representing the timbral descriptor deep proportion of low frequency energy; and/or 3) the low-frequency limit of the audio extract (the low frequency roll-on)\textsuperscript{27}

The Max implementation is a direct implementation of the model described by Pearce. It includes calculation of the lower spectral centroid and the ratio of energy between 30Hz and 200Hz compared to all energy up to the Nyquist frequency. The lower spectral centroid is calculated using:

\[
\text{Lower spectral centroid} = \frac{\sum_{n(200Hz)} f(n)x(n)}{\sum_{n(30Hz)} x(n)}
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. The model also calculates the ratio of energy between 30Hz and 200Hz compared to all energy up to the Nyquist frequency:

\textsuperscript{27}Andy Pearce, “First prototype of timbral characterisation tools for semantically annotating non-musical content.”
where \( n(\text{Nyquist}) \) is the frequency relating to the Nyquist frequency.

### 3.3.5. \texttt{pnp.descriptor} \textsuperscript{28}

\texttt{pnp.descriptor} \textsuperscript{28} is a ‘blank’ descriptor that analyzes an incoming audio signal using an adjustable range. Similar to the brightness and depth models, this model calculates a frequency-limited spectral centroid as the spectral centroid of the frequencies between low and high frequency arguments given for the object:

\[
\text{Frequency-limited spectral centroid} = \frac{\sum_{n=\text{low}}^{n=\text{high}} f(n)x(n)}{\sum_{n=\text{low}}^{n=\text{high}} x(n)}
\]

where \( n(\omega) \) is the bin number relating to variable frequency \( \omega \), \( f(n) \) is the frequency of the \( n^{th} \) bin, and \( x(n) \) is the magnitude of the \( n^{th} \) bin. The model also calculates the ratio of energy between the low and high frequency arguments compared to all energy up to the Nyquist frequency:

\[
\text{Ratio} = \frac{\sum_{n=\text{low}}^{n=\text{high}} x(n)}{\sum_{n=0}^{n=\text{Nyquist}} x(n)}
\]

where \( n(\text{Nyquist}) \) is the frequency relating to the Nyquist frequency.

\[\textsuperscript{28}\text{Andy Pearce, “First prototype of timbral characterisation tools for semantically annotating non-musical content.”}\]
3.3.6. pnp.energy^-

Total spectral energy calculation already exists in Max and is a well known feature.\(^{29}\) pnp.energy^- calculates the total energy of each FFT frame:

\[
\text{Energy} = \sum_{n(0Hz)}^{n(Nyquist)} (r^2 + i^2)
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(r\) is the real part of the FFT calculus, and \(i\) is the imaginary part of the FFT calculus.

3.3.7. pnp.flatness^-

pnp.flatness^- calculates the spectral flatness of each FFT frame.\(^{30}\) The spectral flatness is used to quantify the tonal quality, i.e. how much tone-like the sound is as opposed to being noise-like.\(^{31}\) Spectral flatness is defined by the ratio of the geometric mean to the arithmetic mean of the power spectral density components in each critical band. It is calculated as:

\[
\text{Spectral flatness} = \sqrt[\prod_{n(0Hz)}^{n(Nyquist)} x(n)} \bigg/ \frac{1}{n} \sum_{n(0Hz)}^{n(Nyquist)} x(n)
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(n(Nyquist)\) is the frequency relating to the Nyquist frequency, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. The Max implementation is a modified implementation of the model by Izmirli that performs calculations on each bin within a frame rather than on critical bands.

3.3.8. pnp.hardness^-

A hard sound is one that conveys the sense of having been made (i) by something solid, firm or rigid; or (ii) with a great deal of force. pnp.hardness^- calculates the apparent hardness of

\(^{29}\)Malt and Jourdan, “Zsa. Descriptors: a library for real-time descriptors analysis.”

\(^{30}\)Malt and Jourdan.

\(^{31}\)Ozgur Izmirli, “Using a spectral flatness based feature for audio segmentation and retrieval,” 2000,
an incoming audio signal. Although no explicit model of hardness exists in the literature, there is an indication that the attack and the spectral content of the attack determine the apparent hardness. Research by Williams suggests that the onset portion of a sound determines the perception of hardness. Additionally, Freed presents a model of mallet hardness perception for single percussive sounds with respect to four acoustic correlates: 1) spectral mean level (a form of long term average spectrum, LTAS); 2) spectral level slope; 3) spectral centroid mean (mean spectral centroid over time, measured on the bark scale); and 4) spectral centroid TWA (time weighted mean of the spectral centroid).

A model of hardness was developed by Pearce et al. which employs three metrics: (i) attack time; (ii) attack gradient; and (iii) spectral centroid of attack. The model calculates the attack gradient (difference in amplitudes of the attack start and end levels divide by the linear attack time) of the sound using a fixed attack time of 125ms:

\[
\text{Attack gradient} = \frac{a_{\text{end}} - a_{\text{start}}}{125}
\]

where \( a \) is the amplitude relating to the attack of the signal. The attack spectral centroid is then calculated over the first 200ms before the attack and 125ms after the attack start, or until the next onset time if it happens before 125ms:

\[
\text{Spectral centroid} = \frac{n(Nyquist) \sum_{n(0Hz)} f(n)x(n)}{n(Nyquist) \sum_{n(0Hz)} x(n)}
\]

where \( n(\omega) \) is the bin number relating to frequency \( \omega \), \( f(n) \) is the frequency of the \( n^{th} \) bin, and \( x(n) \) is the magnitude of the \( n^{th} \) bin.

---

32 Andy Pearce, “First prototype of timbral characterisation tools for semantically annotating non-musical content.”

33 Duncan Williams, *Towards a timbre morpher* (University of Surrey (United Kingdom), 2010).

The implementation of hardness in Max is a modified version of the model proposed by Pearce et al. that calculates the attack gradient using a fixed attack time of 125 and the brightness and depth of the attack using the PnP.Maxtools descriptors previously described over the first 10ms before the attack and 125ms after the attack start. This is done to capture as much of the period before the onset as possible without adding noticeable latency. The attack gradient, depth, and brightness are scaled so the maximum value that can be returned from the model is 1.

\[(depth + brightness) \times 0.15 + attackgradient \times 0.85\]

### 3.3.9. pnp.metallic\(^*\)

pnp.metallic\(^*\) calculates the probability that an incoming sound is produced by a metallic source.\(^{35}\) Aramaki et al. identifies four timbre descriptors that are relevant signal features for the discrimination between sound categories: attack time, spectral bandwidth, roughness, and normalized sound decay. These are used to determine whether characteristics of a sound resemble that of sounds made by metallic objects. In general, metallic sounds contain rich and complex spectra relative to other sounds, such as those made by wooden or glass objects.\(^{36}\)

First, the spectral standard deviation is calculated with the equation:

\[
\text{Spectral standard deviation} = \sqrt{\frac{\sum_{n(0Hz)}^{n(Nyquist)} (f(n) - \mu)^2 x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}}
\]

where \(\mu\) is the spectral centroid in hertz, \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. The normalized

---

\(^{35}\)Andy Pearce, “First prototype of timbral characterisation tools for semantically annotating non-musical content.”

decay time is calculated by taking the absolute of the Hilbert Transform of the signal, followed by a low pass second-order Butterworth filter with a cut-off frequency of 50Hz. The logarithm of this is taken after adding 1 to the result, which ensures that the logarithm of 0 is never calculated. This is expressed with the equation:

\[ Envelope = \log_{10}(F(|H(x)|) + 1) \]

where \( x \) is the audio signal, \( H(x) \) is the Hilbert Transform of \( x \), and \( F(x) \) represents filtering of the signal. The roughness is then calculated with the equation:

\[ r = 0.5X^{0.1}Y^{3.11}Z \]

with:

\[ X = A_{\text{min}} \cdot A_{\text{max}} \]
\[ Y = \frac{2A_{\text{min}}}{A_{\text{min}} + A_{\text{max}}} \]
\[ Z = e^{-3.5s(f_{\text{max}} - f_{\text{min}})} - e^{-5.75s(f_{\text{max}} - f_{\text{min}})} \]
\[ s = \frac{0.24}{0.0207f_{\text{min}} + 18.96} \]

where \( r \) is the roughness, \( A_{\text{max}} \) and \( A_{\text{min}} \) are the maximum and minimum magnitudes of the pair of peaks, and \( f_{\text{max}} \) and \( f_{\text{min}} \) are the maximum and minimum frequencies of the two peaks respectively. Finally, a logistic regression model was calibrated based on the normalized decay time, spectral spread, and roughness parameters. This is calculated with the equation:

\[ p = \frac{e^y}{1 + e^y} \]

where \( p \) is probability of the signal being metallic, and \( y \) is the regression equation calculated as:

\[ y = (normalized \ decay \times 4) + (spectral \ spread \times 0.0005) + (roughness \times 0.5) \]

The implementation in Max is a direct implementation of the model proposed by Aramaki et al., where the metallic probability of each FFT is calculated. Only the attack time was omitted from the calculation because the attack time is dependant upon the detection and analysis of onsets in the signal, making resonance and the gradual decay metallic sound more difficult to detect.

3.3.10. pnp.roughness

A rough sound is one that has an uneven or irregular sonic texture. pnp.roughness~ calculates the apparent roughness of an incoming audio signal using an FFT and Gen~ according to Vassilakis.\(^{40}\) Gen is an extension in Max that converts the patch into compiled C++ code.\(^{41}\) It also makes calculations at the audio sampling rate rather than the vector rate. The term auditory roughness was first introduced in the literature by Helmholtz to describe the buzzing, harsh, raspy sound quality of narrow harmonic intervals.\(^{42}\) The dimension of

---

\(^{40}\) Vassilakis and Fitz, “SRA: A web-based research tool for spectral and roughness analysis of sound signals.”

\(^{41}\) https://docs.cycling74.com/max7/vignettes/gen~

dissonance correlating best with auditory roughness has been termed sensory or tonal dissonance or auditory dissonance.

The Vassilakis Roughness model detects all peaks in the frequency spectrum for each frame where: (i) the magnitude of the frequency bin is greater than 0.01; (ii) the magnitude of the previous and next bins are less than the current bin; and (iii) in the frequency range between successive peaks the magnitude drops at least 0.01 below the magnitude of the lower peak. For each pair of peaks within a frame, the roughness is calculated with the equation:

\[ r = 0.5X^{0.1}Y^{3.11}Z \]

with:

\[ X = A_{\text{min}} \times A_{\text{max}} \]

\[ Y = \frac{2A_{\text{min}}}{A_{\text{min}} + A_{\text{max}}} \]

\[ Z = e^{-3.5s(f_{\text{max}}-f_{\text{min}})} - e^{-5.75s(f_{\text{max}}-f_{\text{min}})} \]

\[ s = \frac{0.24}{0.0207f_{\text{min}} + 18.96} \]

where \( r \) is the roughness, \( A_{\text{max}} \) and \( A_{\text{min}} \) are the maximum and minimum magnitudes of the pair of peaks, and \( f_{\text{max}} \) and \( f_{\text{min}} \) are the maximum and minimum frequencies of the two peaks respectively. The product of this calculation is normalized by dividing by the spectral frame size used for the FFT and scaled so that the range of the output is between 0-1.

---


45 Vassilakis and Fitz, “SRA: A web-based research tool for spectral and roughness analysis of sound signals.”
3.3.11. pnp.sharpness

A sharp sound is one that suggests it might cut if it were to take on physical form. pnp.sharpness calculates the apparent sharpness of an incoming signal based on the model described by Fastl and Zwicker. Closely related to sharpness, however inversely, is a sensation called sensory pleasantness. Fastl et al. defines a sound of sharpness 1 acum as “a narrow band noise one critical band wide at a centre frequency of 1kHz having a level of 60dB.” However, sharpness is a metric which has not yet been standardised. Consequently there are several methods to calculate the metric including: Von Bismarck’s method introduces the idea of a weighted first moment calculation, Aures’s method is a modified version of Von Bismarck’s equation, and Fastl and Zwicker’s method which is a version of Von Bismarck’s equation with a modified weighting curve. Like boominess, sharpness has been used to partially quantify sound quality in examples such as measuring engine noise, and some domestic appliances such as vacuum cleaners and hair dryers. It has also been used in the calculation of a sensory pleasantness metric and an unbiased annoyance metric. Using Zwicker and Fastl’s approach sharpness can be calculated as:

\[
\text{sharpness} = 0.11 \frac{ \int_{0}^{24\text{Bark}} N' g(z) z \, dz \text{accum}}{\int_{0}^{24\text{Bark}} N' \, dz}
\]

where \(N\) is the total spectral loudness, \(g(z)\) is the weighting factor for sharpness as a function of the critical-band rate, and \(dz\) is a scaling factor. Only for critical-band rates greater than 16 bark does the weighting factor increase from unity to a value of 4 at the end of the critical-band rate near 24 bark.

46Zwicker and Fastl, *Psychoacoustics: Facts and models*


49Zwicker and Fastl, *Psychoacoustics: Facts and models*
The AudioCommons implementation differs slightly from the above model. It windows the sound into frames of 4096 samples and then calculates the loudness of all 1/3 octave bands within the window up to the Nyquist frequency. The implementation in MaxMSP is similar, except that it uses an FFT to calculate the sharpness of each frame up to 13,500Hz:

\[
sharpness = 0.11 \frac{\sum_{n(0Hz)}^{n(13,500Hz)} x(n) * gz(n) * z * 0.1}{\sum_{n(0Hz)}^{n(13,500Hz)} x(n)0.1}
\]

with:

\[
gz = \begin{cases} 
1 & \text{if } n \leq 2899 \text{ Hz} \\
0.00012 * (z/10.0)^4 - 0.0056 * (z/10.0)^3 \\
+0.1 * (z/10.0)^2 - 0.81 * (z/10.0) + 3.5 & \text{if } n \geq 2900 \text{ Hz}
\end{cases}
\]

\[
z = [v_1, v_2, ..., v_n]
\]

where \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(x(n)\) is the magnitude of the \(n^{th}\) bin, \(v_n\) is the size of the FFT frame divided by 10, and \(gz(n)\) is the weighting factor.
for sharpness as a function of the critical-band rate. Only for critical-band rates greater than 2899Hz does the weighting factor increase from unity to a value of 4 at the end of the critical-band rate near 13,500Hz.

3.3.12. \texttt{pnp.spread}^\textsuperscript{\textdagger}

\texttt{pnp.spread}^\textsuperscript{\textdagger} is another well known timbral descriptor that calculates the variance of the spectral centroid of an incoming audio signal.\textsuperscript{50} The spectral centroid is considered the first moment of spectra, considered as a frequency distribution, which is related with the weighted frequency mean value. The spectral spread is the second moment, i.e., the variance of the mean calculated as:

\[
\text{Spectral spread (variance)} = \frac{\sum_{n(0Hz)}^{n(Nyquist)} (f(n) - \mu)^2 x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}
\]

where \(\mu\) is the spectral centroid in hertz, \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. \texttt{pnp.spread}^\textsuperscript{\textdagger} includes an additional outlet that gives the spectral standard deviation of the audio signal. This is achieved by taking the square root of the spectral spread:

\[
\text{Spectral standard deviation} = \sqrt{\frac{\sum_{n(0Hz)}^{n(Nyquist)} (f(n) - \mu)^2 x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}}
\]

where \(\mu\) is the spectral centroid in hertz, \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin.

\textsuperscript{50}Malt and Jourdan, “Zsa. Descriptors: a library for real-time descriptors analysis.”
3.3.13. pnp.warmth

A warm sound is one that promotes a sensation analogous to that caused by a physical increase in temperature. Several methods for calculating warmth have been proposed, all of which indicate that concentrated low spectral energy correlates with the perception of warmth. Pratt et al. proposes that a low spectral centroid and high energy in the first three harmonics above the fundamental frequency indicates that a sound is warm. pnp.warmth calculates the apparent warmth of an incoming audio signal using a direct implementation of the model described by AudioCommons. This model calculates the spectral centroid of the mean warmth region (the area between the fundamental frequency and the fundamental frequency $3.5$):

$$\text{Mean warmth region spectral centroid} = \frac{\sum_{n(fund \times 3.5)} f(n) x(n)}{\sum_{n(fund \times 3.5)} x(n)}$$

where $fund$ is the fundamental frequency relating to the signal, $n(\omega)$ is the bin number relating to frequency $\omega$, $f(n)$ is the frequency of the $n^{th}$ bin, and $x(n)$ is the magnitude of the $n^{th}$ bin. The model also calculates the ratio of energy between the mean warmth region compared to all energy up to the Nyquist frequency:

$$\text{Ratio} = \frac{\sum_{n(fund \times 3.5)} x(n)}{\sum_{n(0Hz)} x(n)}$$

where $n(Nyquist)$is the frequency relating to the Nyquist frequency.

---


52Andy Pearce, “Release of timbral characterisation tools for semantically annotating non-musical content.”
3.4. Additional Controls

The additional controls provide a wide range of functionality. Most objects in this category modify output values from descriptors to provide users with more control in terms of parameter automation or mapping. Several of these objects remove either low or high values to prevent extreme values and large leaps, such as pnp.nozero and pnp.noone, while others smooth output values to prevent rapid value changes.

3.4.1. pnp.autoscale~

Many of the sound descriptors are amplitude dependant, so they are more likely to output higher values if the amplitude of the incoming audio signal is higher. To prevent this,
pnp.autoscale is designed to track the amplitude of the incoming signal and scale it upwards or downwards towards a target amplitude value specified with an argument. The figure above is a modified version of the Adaptive Signal Level Scaling object proposed by Mikahil Malt and Emmanuel Jordan. The main difference is that pnp.autoscale uses amplitude values from 0-1 as inputs, which allows it to be easily controlled by other objects from the package. The second inlet (the patch cable connected to the box labelled 2) sets the amplitude level to maintain, while the third inlet specifies the trigger threshold. When the amplitude value threshold from the third inlet is met, the object will scale the signal to the amplitude level specified by the second inlet.

3.5. Effects
The audio effects are an assortment of various signal processing techniques and algorithms. Most objects in this category are designed to be controlled with a number between 0-1. These objects implement well-known algorithms for achieving effects.

3.5.1. pnp.pluck
pnp.pluck is a modified version of the famous Karplus-Strong plucked string physical model. While the original model uses a short impulse of noise to excite a digital filter and feedback delay line, the pnp.pluck adaptation uses live input from a microphone to create impulses to the system which are multiplied by noise.

Using longer sustained sounds generally makes the output differ from the sound of a string since the decay of longer sounds causes the natural decay on the Karplus-Strong model to distort. However, it is intended to work well with sounds with shorter attacks, such as the sounds from percussion instruments.


3.5.2. pnp.split

pnp.split splits an audio signal using an FFT at a specified frequency bin. The split position, which can be set using a floating point number between 0-1, determines how much of each frame is output from the left and right channels. A split position of 0 means that the entire signal will be output from the right outlet or channel, while a split position of 1 will be output from the left outlet or channel. The object functions by allowing users to set a floating point number between 0-1 that corresponds with the bin number as a threshold.

3.5.3. pnp.wonky

pnp.wonky incorporates a variable delay line and feedback, where the delay line is controlled by a random signal number generator that interpolates linearly between values. This causes the pitch of the audio coming from the delay line to change as the random number generator ramps to each new destination. Feedback causes these pitch shifts to compound depending on the scaling factor used for the feedback signal. When both the randomness and feedback amount arguments are close to 1, this effect is exaggerated and causes interesting and sometimes chaotic results.

3.6. Event-Based Applications

Reflexive Automation has been previously defined as the analogous relationship between a change in an audio effect and the change in the quality of the sound that is used to control that effect through the utilization of mapping and MIR techniques. However, in the context
of the flow of a musical passage, the need to detect more complex singular events may overcome the need for mapping features in a continuous fashion. In these instances, event detection techniques are required.

Event detection, rather than onset detection, is defined by Jourdan et al. as “any kind of musical event that could be perceived as a discontinuity within a musically static flow.” These singular events may arise through abrupt changes in timbre, amplitude, or register. Sometimes detection may simply be a matter of setting a threshold that triggers an event when the threshold has been met, although there are situations where event detection may be more complex and require specific techniques.

---

55 Malt and Jourdan, “Real-time uses of low level sound descriptors as event detection functions using the max/msp zsa. descriptors library.”
Contemporary composers often employ extended instrument techniques such as multiphonics or keys clicks on a woodwind instrument. However, current real-time event detection methods are primarily based on amplitude and pitch. While this section will present current event detection methods by measuring variations in timbre, it will also speculate about future potential areas of research along with strategies for building event detection functions.

### 3.6.1. Event Detection by Spectral Slope

The spectral slope is a measurement of how quickly the spectrum of an audio signal decreases across the frequency range.\[56\] It is typically calculated on the magnitude spectra using linear regression. The ZsaDescriptors library implements a modified version of an algorithm

---

\[56\] Malt and Jourdan, “Real-time uses of low level sound descriptors as event detection functions using the max/msp zsa.descriptors library”
proposed by Peeters that is based on the covariance and variance of the frequency bins and energy of an FFT frame. It is calculated using:

\[
slope[t] = \frac{\text{cov}(f, a^2)}{\text{var}(f)}
\]

where \( a \) is the linear amplitude vector frame and \( f \) is the frequency vector frame. Computing the slope will yield an envelope that differs significantly from other types of spectral analysis, such as energy. Figure 33 compares the envelopes from an energy and spectral slope analysis on the same audio sample.

![Figure 33. Event detection function comparison: energy vs. slope](image)

It can be seen that the slope is affected by amplitude variation far less than energy is. Additionally, the slope logarithm results in more precise, well-defined peaks. The result from a spectral slope calculation can be used to detect peaks in the envelope. This can be done using a low pass filter to smooth the signal, a multiplicative stage to scale the range of the output between -1, +1, and threshold detection using MaxMSP objects such as thresh, edge, or other Boolean objects that operate on signals.

---

3.6.2. Event Detection by Spectral Standard Deviation

Spectral standard deviation has been shown to be a useful event detection function in noisy and complex musical situations. A noisy event (e.g. breathing through an instrument) can be differentiated from harmonic and pitched material. A flute has a standard deviation around 400-500Hz while a key click has a standard deviation in the range of 1500-2500Hz. Malt et al. proposes a spectral standard deviation detection function by taking the square root of the spectral spread (variance):

\[
d = \sqrt{\sum_{n=0}^{n(Nyquist)} \frac{(f(n) - \mu)^2 x(n)}{\sum_{n=0}^{n(Nyquist)} x(n)}}
\]

where \(\mu\) is the spectral centroid in hertz, \(n(\omega)\) is the bin number relating to frequency \(\omega\), \(f(n)\) is the frequency of the \(n^{th}\) bin, and \(x(n)\) is the magnitude of the \(n^{th}\) bin. The standard deviation is then gated by a K factor:

\[
event\ function = d \star K
\]

where the K factor is defined as a conditional:

\[
K(A_{rms-dB}) = \begin{cases} 
1; & A_{rms-dB} \geq A_{min-dB} \\
0; & A_{rms-dB} < A_{min-dB} 
\end{cases}
\]

where \(A_{rms-dB}\) is the rms signal level in dB and \(A_{min-dB}\) is the threshold in dB. The K value is important in soft or silent musical passages where noisy or flat spectra will cause quick increases in the standard deviation value. As with the spectral slope event detection

---

58 Malt and Jourdan, “Real-time uses of low level sound descriptors as event detection functions using the max/msp zsa. descriptors library”
function, a low pass filter and other MaxMSP threshold objects may be warranted in order to smooth the values and constrain the output to a range appropriate for detecting peaks.

Standard deviation event detection was used successfully by Italian composer Daniell Ghisi in his piece “Comment pouvez vous lire à présent ? Il fait nuit” for Alto Sax and real-time electronics that was created at IRCAM in 2009. The function was used to detect key clicks in the final moments of the piece and trigger changes in the electronic sounds in real-time. An excerpt of the score is shown in Figure 34.

![Figure 34. Comment pouvez vous lire à présent ? Il fait nuit, measures 90 to 94](image)

The spectral standard deviation, the derivative, and the onset detection of the first gesture (four key clicks followed by an E4-G4 in crescendo/decrescendo) of measure 90 are shown in the next image. The difference between the standard deviation from key clicks (around 1500-2000Hz) and for the pitches in the range of E4-G4 (around 150Hz) is extremely noticeable and highlights the advantage of using the spectral standard deviation to detect noisy events.

### 3.6.3. Strategies for Building Event Detection Functions

Extraneous noise strongly affects all methods of event detection and are dependant upon microphone type, input gain amount, distance from sound source, and physical features of the room. The values output from descriptors or other algorithms will inevitably vary from performance to performance because of these factors. However, there are several strategies that can be implemented to ensure more reliable and consistent outcomes.
Firstly, autoscaling values before analysis using the \texttt{pnp.autoscale} object can prevent unwanted changes in amplitude from performance to performance. Additionally, it may be necessary to clip values within a narrow range. For example, clipping a value range of 0-1 between 0.25 and 0.75 can be used to limit the low and high extremes for a particular effect that may be more musically effective if used without drastic changes.

When detecting events using a threshold, the analyzed value may occasionally hover around the trigger threshold and cause multiple consecutive triggers when only one trigger is needed. Secondly, the \texttt{onebang} object in Max may be used to limit additional triggers. An example is shown in figure 36.

Figure 35. Spectral standard deviation, discrete derivative, and onset detection
The onebang object only passes a bang received in the left inlet once it has received a bang in the right inlet. The object can be initialized with a bang in the right inlet using the loadbang object. From here, a delay of 250ms is used on the bangs from the left outlet to reset the onebang object. This effectively prevents more than one trigger within the span of 250ms, and this window may be adjusted to any desired length of time.

Finally, more complex event detection involves the use of a cooperative descriptor, or a descriptor created through the implementation of more than one descriptor simultaneously. For example, a particular sound may be best described as having both significant spectral roughness and depth. In this case, a threshold may be set on the scaled sum of these descriptors. This will distinguish the event from others which may correlate with significant spectral roughness or depth, but not both. This may be calculated using:

\[
D(n) = (\text{Roughness} \times 0.6) + (\text{Depth} \times 0.4)
\]

\[
T(n) = \begin{cases} 
T(n) = 1; D(n) \geq 0.9 \\
T(n) = 0; D(n) < 0.9 
\end{cases}
\]
where T is the trigger value and \((n)\) is the value at the \(nth\) index or frame. When the scaled sum of the roughness and depth descriptors is greater than or equal to 0.9, a trigger will occur. The relative strength of each descriptor may be controlled by scaling by a different factor as long as the scaled sum of all descriptors never exceeds 1. More than two descriptors may be used at a time, and additional techniques described above may be used simultaneously with a cooperative descriptor.
CHAPTER 4. EVALUATION AND TESTING

4.1. Evaluation Using the Cranfield Model

The *PnP.Maxtools* package was reviewed and evaluated using the Cranfield Model, which is designed for the evaluation of information retrieval systems. The Cranfield Model is a six point scale developed by Cleverdon et. al. that measures several properties, such as coverage, effort, presentation, and time lag, that are suitable for evaluating the efficacy of audio feature extraction toolboxes and MIR systems. The presentation of an MIR toolbox describes the output file format options for various feature extraction algorithms. There are several formats, such as CSV, TSV, JSON, and XML that are typically used for MIR systems. However, output formats associated with real-time environments, such as *PnP.Maxtools* in MaxMSP, where the data can be written in many formats is not the best metric to use for evaluation and has been omitted. Two additional properties, precision and recall, are included in the Cranfield Model but are not considered relevant to audio feature extraction toolbox evaluation, and therefore have also been omitted. Prior work by J. Stephen Downie discusses challenges associated with using precision and recall as metrics for MIR evaluation. The criteria used to evaluate audio feature extraction toolboxes and their descriptions are as follows:

- **Coverage** - the range of audio descriptor features presented by a toolkit, along with additional preprocessing or post processing functionality
- **Effort** - how easily one can create a new specific query or modify queries, and appropriate documentation


• Time Lag - computational efficiency of each tool

Ten additional audio feature extraction toolboxes are evaluated based on the Cranfield Model as proposed by J.D. Reiss et. al. and compared to the functionality of \textit{PnP.Maxtools} package with respect to three of the six criteria of the Cranfield Model. The data collected on the range of audio features that can be extracted, additional processing tools implemented, interface options, and computational time required for each tool is presented.

4.1.1. Existing Audio Feature Extraction Toolboxes

There are numerous audio feature extraction toolboxes available that are delivered in different formats. They are usually delivered as stand alone applications, plug-ins for host applications, or as software function libraries. Some toolboxes use specialized Application Programming Interfaces (APIs) that allow feature extraction plugins to be developed, such as the Vamp Plugin API or the Feature Extraction API (FEAPI) written in C++. Others use web based audio feature extraction APIs that result in large music feature data-sets.

The feature extraction toolboxes that will be evaluated and compared to the \textit{PaP.Maxtools} package are:

• Aubio - a high level feature extraction library that extracts features such as onset detection, beat tracking, tempo, melody

• Essentia - full function workflow environment for high and low level features, facilitating audio input, and statistical analysis of output

\footnotesize
\begin{itemize}
\item \cite{downie2003} Downie, “The scientific evaluation of music information retrieval systems: Foundations and future.”
\item \cite{cannam2009} Chirs Cannam, “The vamp audio analysis plugin api: A programmer’s guide,” \textit{Available online: http://vamp-plugins. org/guide. pdf}, 2009,
\item \cite{lerch2005} Alexander Lerch, Gunnar Eisenberg, and Koen Tanghe, “FEAPI: A low level feature extraction plugin API,” in \textit{Proceedings of 8th International Conference on Digital Audio Effects} (2005), 73--76.
\item \cite{bogdanov2006} Bogdanov et al., “Essentia: An audio analysis library for music information retrieval.”
\end{itemize}
- **jAudio** - Java based stand alone application with Graphic User Interface (GUI) and Command-Line Interface (CLI); designed for batch processing to output in XML format or ARFF for loading into Weka\(^\text{11}\)
- **Librosa** - API for feature extraction, for processing data in Python\(^\text{12}\)
- **LibXtract** - low level feature extraction tool written with the aim of efficient real-time feature extraction, originally in C++ but now ported to Max-MSP, Pure Data, Super Collider and Vamp formats\(^\text{13}\)
- **Marsyas** - full real-time audio processing standalone framework for dataflow audio processing with GUI and CLI; this program includes a low level feature extraction tool built in C++, with ability to perform machine learning and synthesis within the framework; the feature extraction aspects have also been translated to Vamp plugin format\(^\text{14}\)
- **Meyda** - Web Audio API based low level feature extraction tool, written in Javascript; designed for web browser based efficient real time processing\(^\text{15}\)
- **MIR Toolbox** - audio processing API for offline extraction of high and low level audio features in Matlab. Includes preprocessing, classification and clustering functionality along with audio similarity and distance metrics as part of the toolbox functionality; algorithms are fragmented allowing detailed control with simple syntax, but often suffers from standard Matlab memory management limitations\(^\text{16}\)

---


\(^{12}\) McFee et al., “Librosa: Audio and music signal analysis in python.”

\(^{13}\) Jamie Bullock and UCEB Conservatoire, “Libxtract: A lightweight library for audio feature extraction,” in *ICMC* (Citeseer, 2007).


\(^{16}\) Olivier Lartillot and Petri Toiviainen, “A Matlab toolbox for musical feature extraction from audio,” in *International conference on digital audio effects*, vol. 237 (Bordeaux, 2007), 244.
• Timbre Toolbox - a Matlab toolbox for offline high and low level feature extraction; a toolbox that provides different set of features to the MIR Toolbox, specifically made efficient for identifying timbre and to fulfil the Cuidado standards.\textsuperscript{17}

• YAAFE - low level feature extraction library designed for computational efficiency and batch processing by utilising data flow graphs, written in C++ with a CLI and bindings for Python and Matlab.\textsuperscript{18}

This is not a complete list of audio feature extraction toolboxes. However, these were chosen because a Cranfield Model evaluation already exists based on their popularity, programming environment range, and how frequently they are updated.

4.1.2. Coverage

The coverage of an information retrieval system is defined as the extent to which all matters relevant to the system are covered. With audio feature extraction toolboxes specifically, coverage describes the range of features a tool can extract. This section presents the number of features provided by each toolbox with respect to spectral features as well as the total number of features offered by these toolboxes. While many of these toolboxes offer a wide range of features that operate on real-time and offline audio signals, such as those that utilize the MPEG-7 low level audio descriptors, many of these are not relevant to determining the coverage of these packages as it relates to the \textit{PnP.Maxtools} package. Since timbral feature extraction is a broad category of MIR research and development and represents the entire scope of research with regards to the \textit{PnP.Maxtools} package, it was decided to limit the scope of evaluation to features within this category only. The results of this can be seen in Figure 37.


\textsuperscript{18}Benoit Mathieu et al., “YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software.,” in \textit{ISMIR} (Citeseer, 2010), 441–446.
As is shown on the graph above, LibXtract and Essentia contain the most spectral descriptors, containing a total of 31 and 32 available to users, respectively. The third largest toolbox is a tie between MIR Toolbox and the PnP.Maxtools package, each of which contain 19 descriptors. Both the jAudio and Marsyas toolboxes contain 16 descriptors, while the majority contain between 11 and 14. Aubio contains the fewest number of timbral descriptors at only 7. Only two features, spectral centroid, and signal energy, are present in all the toolboxes, and most contain similar objective features such as spectral spread, rolloff, and flatness. Meyda and YAAFE contain approximately the same number of total and spectral features as the majority of extraction algorithms have been adapted from the YAAFE library.

Many of the toolboxes, particularly the ones which contain fewer numbers of spectral descriptors, contain many other MIR models and pre/post processing functions. For example, Aubio is designed specifically for high level feature extraction with a particular focus on segmentation. It is similar to PnP.Maxtools in that it includes several digital filters and utilities for music applications. On the other hand, LibXtract, jAudio, and Meyda are designed
to extract low level features.\(^{19}\) Several of the toolboxes also include various bark computing algorithms, such as Essentia, LibXtract, Meyda, MIR Toolbox, and YAAFE which perform calculations over the bark scale to more closely align with the non-linear perception of frequency.

All of the toolboxes include a resampling function that allows for the standardization of sampling rates when extracting features from audio signals. For example, it would be possible to have a spectral centroid of 30kHz if a sampling rate of 96kHz is used. However, this frequency is impossible to represent with a sampling rate of 44.1kHz. Meyda, MIR Toolbox, \textit{PnP.Maxtools}, and Timbre Toolbox inherit resampling functions from their parent environments, such as Matlab, MaxMSP, and the Web Audio API, while the others include resampling functions as part of the toolbox.

While most toolboxes contain more total features than the \textit{PnP.Maxtools} package, most actually contain a fewer number of unique spectral features. Only Essentia, LibXtract, and MIR Toolbox contain the same or greater number of unique spectral features than the \textit{PnP.Maxtools} package. These same toolboxes are also the toolboxes which contain the greatest number of total features.

\subsection{Effort}

The Effort of an information retrieval system is used to define how challenging a user finds the system to use, and whether any user experience considerations have been made while developing the system. This section evaluates effort relative to the user interface that is provided by a toolbox, whether that is a Graphical User Interface (GUI), Command Line Interface (CLI), or an Application Program Interface (API). Additionally, the existence or quality of documentation and suitable examples is evaluated with the purposes of evaluating how intuitively a tool’s interface is presented to a user. The following table shows the different user interfaces presented by each toolbox.

\footnotesize{\textsuperscript{19}}Moffat, Ronan, and Reiss, “An evaluation of audio feature extraction toolboxes”
Figure 38. Overview of feature extraction toolboxes

Several of the toolboxes include both low and high level feature extraction algorithms. Examples of high level features include chord or key detection, BPM estimation, and other musical elements that hold higher semantic meaning. Low level features are those that can be computed on a frame-by-frame basis directly from the audio signal. Examples of low level features include spectral centroid, spectral spread, and energy. These generally have less perceptual relevance than high level features. Most toolboxes include both low and high level feature extraction algorithms in varying degrees. Aubio toolbox is specifically designed for high level feature extraction, while LibXtract, Meyda, and jAudio are designed to extract low level features. While PnP.Maxtools is mostly designed for real-time low level feature extraction, it does include real-time onset detection and BPM estimation.

It can be seen in Figure 38 that most toolboxes include at least one user interface, with Aubio, LibXtract, Marsyas, and YAAFE including a Vamp Plugin, GUI, and CLI. A Vamp Plugin is a C++ API which is capable of functioning within standalone applications. Several of the toolboxes, such as Librosa, Meyda, MIR Toolbox, and Timbre Toolbox are only accessible through software API’s and require additional software before feature extraction is possible. PnP.Maxtools does not utilize the MaxMSP API, but instead is a package built

\[\text{Reference: Moffat, Ronan, and Reiss, “An evaluation of audio feature extraction toolboxes.”}\]
using abstractions in Max rather than being written in C++. Currently, it is only accessible through the MaxMSP environment, and its GUI is included as part of the feature extraction package.

Most toolboxes provide clear documentation that includes examples, but LibXtract, Meyda, and Timbre Toolbox provide much less information in terms of basic access and practical application. Most toolboxes are also equipped with implementation examples. Out of these, Aubio, Essentia, MIR Toolbox, PnP.Maxtools, and YAAFE provide a large range of examples that can be used immediately. Marsyas also contains a large collection of examples but requires user to learn a proprietary language before use.

4.1.4. Time Lag

The time lag of an information retrieval system is a measure of how long a given task will take to complete. Comparing the relative speeds of different systems will give users an informed choice as to what system to use. This section discusses the computational complexity of the toolboxes and identifies whether they are implemented in real-time or offline methods.

The offline toolbox evaluation was done by Moffat et al.\textsuperscript{21} which used a subset of the Cambridge Multitrack Data Set. In total, 561 tracks were used from this data set with an average duration of 106 seconds. Each toolbox was used to calculate the mel frequency cepstral coefficients (MFCCs) of the data set with a window of 512 samples and a hop size of 256 samples. A MFCC is used to describe the overall shape of the spectral envelope and is a useful measurement for timbral analysis. The MFCC is also a computational method that exists in all toolboxes except Meyda and PnP.Maxtools, which only operate in real-time, and the Timbre Toolbox. The computational time needed to complete feature extraction for offline toolboxes is shown below.

Figure 39 shows that YAAFE is the fastest toolbox, followed by Essentia and LibXtract in close second and third place. The slowest toolboxes are the MIR toolbox and Librosa, which took close to 30 minutes and two hours to complete, respectively. LibXtract, Meyda, and

\textsuperscript{21}Moffat, Ronan, and Reiss, “An evaluation of audio feature extraction toolboxes.”
Figure 39. Computation time for offline feature extraction

*PnP.Maxtools* run in MaxMSP, Pure Data, and Supercollider in real-time. These real-time environments allow for visualization and interactivity with output data.

These toolboxes, specifically for real-time spectral feature extraction, are built using an FFT. The time lag in this case is equal to the frame size, or the number of samples used for a single FFT frame calculation, which can be adjusted by the user in any of these environments to meet any computational needs. This makes determining computational time for these toolboxes difficult. In general, most latency in these environments ranges from 20-90 milliseconds.

**4.2. Research Considerations**

Many Music Information Retrieval researchers agree that understanding the relationship between timbral descriptors and users’ needs and perception is critical for developing MIR systems. When discussing the challenges concerned with evaluating MIR systems, it is
important to distinguish between systems-based and user-centric MIR. Systems-based MIR research is concerned solely with the performance of a computer, primarily through the evaluation of algorithms on digital databases. On the other hand, user-centric MIR research involves human subjects that interact with MIR systems.

Systems-based MIR traditionally focused its efforts on describing universal aspects of human perception, such as similarity measurements. This type of research assumes an objective truth against which MIR algorithms are evaluated. For example, MIR systems have been evaluated using genre classification experiments for years while it was shown in 2003 that genre classification is not the best measure of an MIR system. However, it still serves as a proxy by which the similarity and retrieval approaches of systems-based MIR systems are assessed.

With regards to user-centric MIR systems, Flexer et al. identifies four key requirements for elaborating user-centric music retrieval systems: Personalization, User Models, Multifaceted Similarity, and Evaluation. Personalization refers to the cognitive component in understanding music and the subjectivity of personal appeal. User Models are sensitive to different social scopes. This may be through the use of individual or group models as well as cultural or even global models. Multifaceted Similarity combines features and feature categories, and the evaluation design must include all independent variables that are capable of influencing dependent variables. Where possible, the research parameters were designed with respect to the requirements proposed by Flexer et al. and user-centric MIR systems.


4.2.1. Research Parameters

The exact method of analysis using the *PnP.Maxtools* objects is difficult to determine since they operate in real-time. The silences within each presented audio file cause the descriptors to output a stream of 0 values that result in a mean calculation that skews low. The peak value identifies the single most intense moment of the audio file with respect to a descriptor, so it too is an inaccurate measurement that skews high because the peak value does not reflect changes to the output of that descriptor over time. Machines are incapable of directing their focus to key features of a sound, or rather, they do not naturally make a distinction between signal, noise, or silence with regards to their analysis. Humans are able to naturally filter sounds in order to focus on a few prominent features. The issue of how to direct the focus of machines, known as scope, presents challenges when determining the efficacy of real time music information retrieval algorithms.

The subjective listening study was used to determine the efficacy of the *PnP.Maxtools* timbral descriptors. The study was conducted with 23 anonymous participants. Each participant was asked to listen to 8 separate audio files and rank each sound according to how strongly they felt a timbral descriptor most represented that sound. The 8 sounds chosen for the study were gathered from the free online repository Freesound.org\(^\text{26}\) and exhibit a wide range of timbres and ranges, including instruments such as drums, flutes, and other non-instrumental sounds such as a heartbeat and knocking on wood. The name of each sound file describes the object or instrument used to make the sound. The scale with which participants ranked each sound ranges from 0 to 100, where 100 is the maximum and 0 is the minimum or none. For example, a sound with a ranking of 90 for brightness is very bright, and a sound with a ranking of 25 for hardness is not very hard. Participants were then asked to select the names of the descriptors they felt best described the content of the sound.

To present the findings of the subjective listening study, the waveform for each normalized audio file is shown alongside a radar chart depicting the mean ranking from all participants,

\(^{26}\)https://freesound.org/
the mean value given from each descriptor, and the peak value given from each descriptor. The name of the descriptor that received the most votes by participants is also shown. Since many of the descriptors are amplitude dependant, each audio file was normalized to -12 dB before being analyzed. To account for the quiet portions of the audio files, anything quieter than -48 dB was detected and left out of the analysis. This was achieved by using a -48 dB level to trigger a sample and hold on the output of the descriptors. The repeating values while this signal remained under this threshold were discarded until the signal rose back above -48 dB. In order to visualize the data alongside data submitted by the participants the normalized floating point numbers output from each descriptor were multiplied by 100 after analysis. This means that a brightness level of 100 on the radar chart reflects a brightness level of 1 from the pnp.brightness descriptor.

4.3. Presentation of Data

4.3.1. Birds

The audio file containing bird sounds has significant energy at high and mid-range frequencies. 15 participants chose brightness as the descriptor which best represents the sound while 11 chose sharpness. The descriptor with the next highest number of votes was warmth with a count of 2. Despite the descriptor peak for warmth being equal to the peak for brightness, the participant mean for warmth is only 48 as opposed to 84 for brightness. It should also be noted that the standard deviation and variance for the participants rating of the warmth descriptor is significantly higher than that of the rating for brightness, with a standard deviation of 25.51 and a variance of 650.95 for warmth, and 15.61 and 243.73 for brightness, respectively. Sharpness has a standard deviation of 9.57 and a variance of 91.56 which is also significantly smaller than warmth. The descriptor peaks shows that sharpness and brightness correlate most with perceived characteristics of the sound as rated by participants.

One interesting note is the high rated value from the boominess descriptor mean and peak, both of which are significantly higher than the participant mean. One potential cause for this is the presence of wind and other rustling noises in the audio file which contains
significant energy between 0 and 200Hz and caused the boominess descriptor to output high values for most of duration of the analysis. A spectroscope is shown in Figure 41 that views low frequencies in this range next to a spectrogram which views all frequencies up to the Nyquist Frequency for comparison.

However, the most likely cause of this outlier is that the bird calls themselves contain some significant energy in the range of 0-200Hz. It can be seen that changes in the magnitude of frequencies in the range of 0-200Hz correspond with magnitude changes in frequencies in the range of 1500-5000Hz, where the majority of spectral energy in the sounds of birds is concentrated in this audio sample.
4.3.2. Drumbeat

The audio file containing a drumbeat has many unique sounds with different frequency components and amplitude envelopes. 14 participants chose metallic as the descriptor which best represents the sound. Boominess received six votes while sharpness, brightness, and hardness all received 5 votes. The descriptor mean for roughness, sharpness, and brightness are almost identical to the mean ratings chosen by participants for those descriptors. The peak mean for those descriptors, although significantly higher, follows the same contour as both the descriptor and participant mean. The descriptor peak, mean, and participants mean for metallic are closer together than all other descriptor and participant ratings. The standard deviation and variance for the participant ratings is also lower than all other de-
Figure 43. Radar chart of drumbeat timbral analysis

scriptor ratings at 14.62 and 213.64, respectively. The descriptor mean and peak show that metallic correlates most with perceived characteristics of the sound as rated by participants.

The descriptor mean and peaks values generally skew high for all of the descriptors. This is most likely due to the wide range of frequency components because of the various drum instruments such as the bass drum, snare drum, and cymbals. This is shown in Figure 44 where there is significant energy above 3kHz. The descriptor mean for warmth, depth, and boominess are the lowest of all the descriptors because of the wide distribution of frequencies.

In the case of warmth, it is likely that one of the instruments contains concentrated frequencies withing the mean warmth region above the fundamental that caused the descriptor peak to output high values. This instrument is not present for the majority of the audio
file. The result is a significantly lower descriptor mean. For depth, the wide distribution of frequencies makes the descriptor mean for depth quite low but the peak high for a reason similar to the warmth peak. The only sound capable of changing the output value of the boomininess descriptor is the bass drum in this audio file, so it can be reasoned that the frequencies of this instrument was simply not low or loud enough to cause this descriptor to output lower values. The participant ratings for these three descriptors are also the most varied, with a minimum and maximum rating of 15 and 100 for boomininess, a minimum and maximum rating of 10 and 100 for depth, and a minimum and maximum rating of 5 and 99 for warmth. The standard deviation and variance of the participants mean for these descriptors are also the highest of any descriptors.

4.3.3. Flute

The audio file containing a simple flute passage has a similar timbre throughout, regardless of changes to pitch and register. 12 participants chose warmth as the descriptor which best represents the sound, with brightness receiving 10 votes. Sharpness and depth received 6 and 4 votes, respectively. Figure 46 shows that the descriptor mean, peak, and participant mean for warmth are the highest out of all descriptors. This shows significant correlation with perceived characteristics of the sound as rated by participants.
Participants also rated descriptors which describe high frequency content of the sound, such as sharpness and brightness, much higher than the ratings from the descriptors mean and peak. However, the highest possible note on the flute is C7 which has a frequency of 2093Hz. This is not high enough to surpass the 3kHz threshold needed to excite the brightness timbral descriptor. Even though there are flute overtones that exist higher than the fundamental above this threshold, the ratio is still close to 0 because most of the spectral energy exists below this threshold.

One interesting outlier is the incredibly high values from the descriptor mean and peak for boominess. A likely cause for this is the hard compression on the audio file, which can be viewed on the waveform in Figure 47. A compressed audio file often looks full and rounded compared to uncompressed waveforms, and when makeup gain is added the distance between peaks and silences decreases. This compression would cause the noise floor and other faint sounds in the file, such as breathing and air traveling over the flute mouthpiece to become much louder. The boominess model does not measure a ratio between low and high frequencies, but rather the presence and strength of low frequencies independent of high frequencies, so the increase in loudness for the actual flute in the recording as a function of compression would have no effect on the measure of boominess.
4.3.4. Heartbeat

The audio file containing heartbeats has very low frequencies with a relatively short attack and decay of the sound. 13 participants chose boominess as the descriptor which best represents this sound, while warmth received 7 votes and depth received 6 votes. All participants emphasized descriptors which relate to low frequency content in their ratings. The participant mean and descriptor mean for boominess are almost identical with values of 77 and 78, respectively. They are also very close to the descriptor peak for boominess which has a value of 90. This shows significant correlation with perceived characteristics of the sound as rated by participants.
The roughness descriptor mean and peak is an outlier with this audio sample. Upon further analysis of the sample, it was determined that a quiet and consistent hum within the range of 0-100Hz exists throughout the entire duration of the sample. This is likely attributable to the manner in which the heartbeat was recorded. This volume of this hum hovers around -44dB, meaning that it lies above the threshold used to remove quiet portions of the sound and was therefore analyzed. Given the inconsistent spectral nature of this hum, it is reasonable to assume that this resulted in a high descriptor mean and peak for roughness. A spectrogram of the heartbeat audio file is shown in Figure 49.

In order to determine the degree to which the noise floor contributed to the high roughness descriptor peak and mean values, the audio file was analyzed again using roughness timbral descriptor with a threshold of -36dB, which effectively removes all of the noise. The data from this analysis shows the descriptor peak maintaining its value of 100, but the descriptor mean value decreased from 55 to 12. Overall, more of the noise floor was removed and the descriptor mean for roughness was lowered as a result.
Figure 48. Radar chart of heartbeat timbral analysis

Figure 50. Waveform of heartbeat audio sample
4.3.5. Metal

The audio file containing the sound of a metal object being struck contains a complex spectra with a very short attack and gradual decay. 18 participants chose metallic as the descriptor which best represents the sound. 8 participants chose sharpness and 7 chose brightness. It is shown that high frequencies correlate most with perceived characteristics of this sound as rated by participants.

Other than an emphasis on high frequencies, the data show very little correlation elsewhere. One exception to this is hardness, which shows higher ratings from the descriptor mean and peak and participant mean values than other descriptors which refer to high frequency content. The reason for poor correlation with regards to other descriptors could be attributable to the length of relevant information contained in the audio file, which only occurs for about 1500ms until the decay of the sound falls below -48dB.

Another analysis of the audio file was taken using only the moment of attack. This was achieved using the pnp.beat` and the pnp.freeze` objects to detect the onset and then sustain and loop the spectral frame which contains the onset. This frame was then analyzed over a period of approximately 5000ms. The new descriptor mean and peak values are shown in Figure 52 against the original participant mean values. There is significantly more
correlation with perceived characteristics of the sound as rated by participants using the moment of attack.

4.3.6. Noise

The audio file containing noise has significant energy at low frequencies and has a complex amplitude envelope. It is more similar to the sound of air blowing into a microphone at close range. This differs from white noise which contains equal energy at all frequencies as a result of random and uniform sample amplitude distribution. The spectra for the noise audio file is shown in Figure 54 next to the spectra white noise for comparison.
Figure 52. Radar chart of metal timbral analysis using onset of attack

18 participants chose roughness as the descriptor which best represents the sound. 3 participants chose sharpness and 2 chose hardness. The descriptor mean and peak values for sharpness and boominess emphasize both low and high frequency content, while the descriptor mean and peak values for warmth show that the majority of the energy is concentrated around low frequencies. There is some correlation between the abrupt amplitude changes in the sound and the perception of hardness as rated by participants. However, the descriptor peak shows that roughness and sharpness correlate most with perceived characteristics of the sound as rated by participants.

The participant mean for roughness has the smallest standard deviation and variance at 17.45 and 304.40, respectively. The next smallest standard deviations and variances are
found in the participant mean ratings for metallic, sharpness, and hardness. It is clear from the results of this analysis that the spectral content of the noise audio file is varied and complex and is best described as not being particularly associated with one descriptor, perhaps with the exception of roughness.
Figure 55. Radar chart of noise timbral analysis

Figure 56. Waveform of noise audio sample
4.3.7. Rainstick

The audio file containing a rainstick has significant energy at high frequencies. 7 participants chose to submit roughness as the descriptor which best represents the sound. After this, 6 participants submitted hardness and 5 submitted brightness and sharpness. It is clear that participants mean reflects the presence of high frequency energy as shown the roughness, sharpness, and brightness descriptors. The descriptor mean and peak values for these descriptors follows a similar contour, with the mean value being significantly larger than the participant mean in these cases.
The descriptor mean and peak also shows high values for the metallic descriptor, while the participant mean is significantly lower. The minimum and maximum ratings submitted by participants is 3 and 94, while the standard deviation and variance is 30.94 and 957.01. This large discrepancy in rating is likely due to participants rating this sound low on account of it not being produced by a metallic source rather than containing qualities resembling those that metallic objects produce. While the metallic descriptor is, in part, a measurement of high frequencies, the ratings by participants with regards to this descriptor is the most varied of all descriptors for this audio file. The second largest standard deviation and variance is the hardness descriptor with values of 25.15 and 632.57.

![Waveform of rainstick audio sample](image)

**Figure 58. Waveform of rainstick audio sample**

### 4.3.8. Wood

The audio containing knocking on wood contains a complex spectra and very short attacks and decays in the envelope. The ratings by participants were significantly more varied with this audio file more than others, with 14 participants choosing hardness as the descriptor which best represents the sound. The next highest choices were sharpness and warmth with 6 and 5 votes, respectively. Overall, there is significant correlation between the participants mean and both the descriptor mean and peak as is evident by the general contour of each metric.
As with the metal sound file, the sparse amount of information because of the extremely short envelopes makes accurate analysis difficult. Another analysis was performed using the onset to trigger a freeze on the frame of attack that was analyzed for a period of approximately 5000ms. The results of this new analysis are shown in Figure 60.

In this analysis, the participant mean is overall much closer to both the descriptor mean and peak for all descriptors. The outliers that exist in the onset analysis are with the depth and brightness descriptors, where the participant mean is significantly higher than the descriptor mean and peak values for those descriptors. It is likely that participants identified low and high frequency content in the sound that actually exists just outside of
Figure 60. Radar chart of wood timbral analysis using onset of attack in the range in which the pnp.depth^ and pnp.brightness^ descriptors detect. The spectrogram in Figure 61 shows the prominence of frequencies in the range of 200-500Hz, which lies just above the range of 30-200Hz that the depth descriptor is capable of recognizing.

The strength of frequencies in the range of 20-200Hz is less than the area just outside of this range, so it is possible that these frequencies, along with an extremely short attack and decay from the sound of the wood caused some participants to rate this sound as having a deep quality. Incidentally, the standard deviation and variance for the descriptor mean and peak for depth is higher than all other models for this sound file at 25.99 and 675.32. Similarly, the strength of frequencies just below the 3kHz threshold for the pnp.brightness^ descriptor likely caused some participants to rate this sound as having a bright quality.
4.4. Conclusion

Evaluation using the Cranfield Model shows that the *PnP.Maxtools* package is one of the more robust toolboxes in terms of coverage with respect to spectral feature extractors. Only two packages, Essentia and LibXtract, contain more unique spectral features. The *PnP.Maxtools* is accessible via a graphical user interface built in MaxMSP as part of the toolbox. Additionally, all resampling and pre/post processing functionality can be accessed as part of the Max environment, although the toolbox does provide some filters. Finally, the latency for the models is a function of the frame size used for the FFT in a real-time context. The models are built using a frame size of 2048 samples, which results in a minimum lag.
of approximately 40ms. Overall, the PnP.Maxtools is a robust solution for real-time MIR applications within the MaxMSP environment.

In terms of the PnP.Maxtools descriptors efficacy, there is significant correlation between the participants chosen verbal descriptor and either the peak, mean, or both peak and mean values for all audio samples. For most audio samples, the overall contour of the participants descriptor ratings closely matched both the model peak and mean values with few exceptions. In these exceptional cases, it is likely there was at least one identifiable contributing factor, such as an overly compressed audio file or a file which did not contain enough audio information with which to make an accurate analysis. Where possible, additional analysis was conducted to determine whether the suspected cause could be identified and to what degree it had an effect on the outcome of the original analysis.

A few of the PnP.Maxtools timbral descriptors either over-perform or are extremely sensitive to ranges of frequencies that are sometimes not prominent enough to correlate with perceived characteristics of that sound. This is particularly the case with the pnp.boominess descriptor, which over-performed on the Birds, Flute, Metal (before onset analysis), and Noise audio files. Each of these audio files contains significant low frequency energy that was simply not identified by participants because of the perceptual strength of high frequency energy.
CHAPTER 5. AN OVERVIEW OF COMPOSED WORKS

5.1. *Bloom*

*Bloom* is a composition for solo cello and reflexive electronics that primarily uses the pnp.amplitude˜ and pnp.register˜ objects for analysis of the cello and control of musical parameters. As stated in the program notes, the piece explores tension, using the metaphor of a blooming flower as the basis from which the musical material and form are derived. The work opens with a very simple melodic idea using natural harmonics on the open strings of the cello. These harmonics are developed throughout the piece, eventually blurring the lines between pitch and noise, meter and aleatory, and acoustic and electronic elements. The technical requirements and set up for the piece are shown below.

![Diagram of technical requirements and set up for Bloom](image)

Figure 63. Technical requirements and set up for *Bloom*

When the signal is initially fed into Max, the frequency of the signal is analyzed using a Ztx-based pitch detection algorithm named retune˜. This returns a floating-point number that represents the approximate frequency of the incoming signal in hertz. These values are then clipped at a minimum and maximum frequency of 98Hz and 440Hz, meaning that any frequency that is higher than 440Hz will output 440, and any frequency that is lower
than 98Hz will output 98. This minimum and maximum output range is used to control a variable delay time on three copies of the original signal. Clipping the maximum value range to 440 means that the delay time will not increase unless the analyzed frequency is below 440Hz. An output of 440 has no effect on the delay time, so this is a way of “evolving” the electronics sounds over time as the pitch content of Bloom trends downwards as the piece progresses.

Before setting the delay time this value is multiplied by three separate values which result in different delay times for each copy. These three copies are the ones that are processed and ultimately heard during the performance. On the longest delayed copy of the signal, `pnp.pitchshift` object is implemented that performs pitch shifting on all incoming audio down by a perfect 5th. A onepole lowpass filter with a cutoff of 880 Hz is immediately applied to remove frequencies above the cutoff, resulting in a lower pitch being heard only when the pitch in the score drops below A5. At the same time, the peak amplitude of the audio signal is analyzed on a scale from 0.0 to 1.0 where 1.0 is equal 44.1kHz. This number is used for several different purposes. First, it is used to control the effective sampling rate of the signal. As the amplitude reaches 1.0, the sampling rate for one of the copied signals reaches 0.35. The lower the sampling rate ratio, the lower the quality of audio and the more noise that is introduced into the signal. This is heard as high frequency “glitter” throughout the piece.

Next, the amplitude is used to control the panning speed of the three copied signals in the stereo field. When no sound is present the signals will not pan from left to right, but as the amplitude reaches 1.0 the panning speed reaches its maximum. This is controlled by a sine wave, the frequency of which is the amplitude value not scaled. So, an amplitude value of 1.0 will result in a panning speed of 1Hz. Finally, the amplitude value is used to control the reverb decay time of all three audio signals. The decay time is at its maximum when the amplitude value is at 1.0, which also effects the clarity and brightness of the down sampling, panning, and pitch shifting effects that have been previously applied.
It is worth mentioning that these amplitude and frequency tracking algorithms are extremely noisy when implemented alone. They often return values that result in large leaps or change more rapidly than aurally is apparent. Because of this, these values were smoothed by taking an average of values over time rather than jumping between them using the `pnp.smoother`. This object was applied at many points along the signal chain to increase the smoothness of values and to assure that all audio effects gradually grew and diminished. Finally, these three processed copies of the signal were routed to gain sliders along with the original dry signal. These are used to control the volume balance between the processed audio and the unprocessed audio and are ultimately what is output for an audience during a live or recorded performance.

5.2. Concentric Circles

*Concentric Circles* is a piece for cello, percussion, piano, and reflexive electronics that utilizes several filters, descriptors, and effects from the *PnP.Maxtools* package simultaneously. As stated in the program notes, *Concentric Circles* is a piece that explores overlapping patterns and cycles. It is based upon an initial harmonic progression that descends by step and is varied upon each repetition. Throughout the piece new patterns are introduced or layered on top of existing patterns. The electronics operate by analyzing and averaging data from all players simultaneously, among other processes. This creates new patterns and interactions between performers which evolve throughout the piece until they transform back into the opening statement.
The electronics operate differently depending on the instrument and the range in which the instrument plays. The piece calls for four microphones, one on each the percussion and cello and two on the piano. In general, each the range of each instrument is segmented into three sections (low, middle, and high) using the pnp.binpass object. From here, each range section for each instrument is processed in a different way.

5.2.1. Percussion Processing

The entire range of the crotales is processed using a variable delay and feedback. The entire range of the vibraphone is processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave. The register of the vibraphone determines what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent).

5.2.2. Cello Processing

Pitches above C5 (notated as harmonics in the score) are processed using a delay line that is pitch shifted up a 5th. Brightness, or “glitteryness” is also increased via down sampling on the signal. Loudness determines the length of the delay and the brightness of the signal. Pitches above G2 and below C5 are processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave respectively. The register of the cello determines
what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent). Pitches below G2 are processed using a delay line and amplified. There are no controlling parameters for this register.

5.2.3. Piano Processing

Pitches above C5 are processed using a delay line that is pitch shifted up a 5th. Brightness, or “glitteryness” is also increased via down sampling on the signal. Pitches above G2 and below C5 are processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave respectively. The register of the piano determines what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent). Pitches below G2 are processed using a delay line and amplified. There are no controlling parameters for this register.

5.2.4. Collective Ensemble Processing

Unless explicitly mentioned above, the amount of processing is determined by the loudness of the individual performers within a particular register (described above) as well as the average overall loudness of the ensemble. When all performers are playing at their loudest dynamic, the loudness of the electronics is at its maximum. When a single performer plays loudly while the rest of the ensemble is quiet, the amount of processing is minimal. Reverb decay time is also affected solely by the average loudness of the ensemble. Performers are instructed to pay special attention to their dynamics throughout the piece, specifically during moments where accents or “swells” are present and bring these out amongst the surrounding texture.

5.3. I/O

I/O is a piece for solo snare drum and reflexive electronics that detects whether the snares on the drum are on or off and processes the sound differently for each. As stated in the program notes, I/O is a piece that explores duality. The title comes from the primary method of analysis, determining when the snares are turned on and off, which controls how the electronics behave throughout the work as well the main thematic idea using the on/off
switch on the drum. Other features, such as loudness and the presence of rim clicks, control many of the electronic sounds. The result is a performance that is both mechanical and organic, incredibly varied, and even unpredictable and chaotic. The technical requirements are shown below.

Figure 65. Technical requirements and set up for I/O

The primary method for audio processing uses a gate with 30 outputs that routes audio to various places within the rest of the patch. When an onset is detected, a random number is generated that opens the gate and sends the signal out of the outlet that corresponds to that random number. When the patch determines that the snares are on, the sequencer is allowed to randomly choose between the first 15 outputs by constricting the random number range generated by each onset to 1-15. When the patch determines that the snares are off, the random number range changes from 1-30, allowing the signal to come out of any output from the gate. These additional outlets create new musical interactions and sounds as the score instructs performers how and when to turn on and off the snares throughout the piece. Some of the outlets are not connected to anything and therefore do not process the sound.
in any way. This is done to ensure that space exists in the texture of the electronics and to add a degree a chance to when and how the audio processing occurs.

![Diagram of the sequencer for I/O](image)

**Figure 66. The sequencer for I/O**

The spectral standard deviation using the pnp.spread object is used to determine when the snares are on or off. The standard deviation when the snares are on is between 1500-2500Hz, while off is significantly less at 100-500Hz. The detection function has a tendency to switch on and off rapidly because the silences are interpreted as the snares being engaged. In order to prevent these sudden changes and to prevent maintaining an incorrect detection for 250-500ms using the onebang object, an if/then statement is used on the output of the detection function and the same output delayed by 500ms using the pipe object. This means that the detection function has to determine that the snares are off for a period of 500ms before changing audio effects associated with the snares being off. If at any time the output
switches on and off suddenly as the result of an incorrect detection, this will not result in any change to the output of the electronics. This is shown below.

![Diagram of snare off detection and if/then statement](image)

**Figure 67. Snare off detection and if/then statement**

There are a large variety of sounds and effects in this piece, such as glitches made using noise generators and short envelopes and FM synthesis to create sounds that are similar to Morse Code. The live audio from the snare drum is also processed using a variety of effects, such as pnp.wonky~, pnp.pitchshift~, and pnp.flange~. In general, the electronics are responsive to amplitude, meaning that they become louder as the performer plays louder. They are also determined by tempo, which is detected using the pnp.bpm~ object. There are two tempi throughout the piece: quarter note equals 155 and 95. A threshold of 135BPM is set in Max to detect when the tempo changes. This threshold is primarily used to determine the range and register of effects.
5.4. *Sing!*

*Sing!* is an improvisational piece for solo singing bowl and reflexive electronics where the electronics are built into three different ‘modes’ that can be activated by the performer using a specific event as a trigger for each mode. Each mode changes several parameters within the patch to create new and developing sounds and textures. These modes are similar to sections within a traditional piece of music, except that the score provides instructions for how the performer activates and deactivates each mode and the performer is allowed to improvise based around these instructions. The technical requirements and set up for the piece are shown below.

![Diagram of technical requirements and set up for Sing!](image)

Figure 68. Technical requirements and set up for *Sing!*

When no mode is activated, the patch uses three instances of the pnp.pitchshift\^ object to create seventh chords which harmonize with the fundamental frequency of the singing bowl. When the singing bowl is sustained, the chords sustain as well. However, when an onset is detected, such as hitting the bowl with the striker, the chords change. The manner in which they change is determined by the amplitude of the strike. For instance, an onset amplitude less than 0.9 results is an instantaneous chord change while an onset amplitude
greater than 0.9 causes the pitches to glissando from the previous to the next chord over an amount of time also determined by the onset amplitude.

Mode 1 is activated using the spectral standard deviation detection function that distinguishes between strikes using the felt portion of the striker and the wooden portion of the striker. When hit with the wooden handle of the striker, the resulting sound is much brighter and yields a standard deviation in the range of 3000-4000Hz. This is significantly higher than the standard deviation of a traditional hit with the felt portion of the striker, which yields a standard deviation in the range of 1000-2000Hz. This mode disables chord changes and glissandi from future onsets within this mode and freezes the current chord. Additionally, mode 1 processes a copy of the signal using the pnp.wonky~, pnp.reverb~, and pnp.panner~ objects, which creates sounds that are much more active and move around in the stereo field. Each of these parameters are controlled by several other descriptors, such as the pnp.amplitude~ object which determines the amount of feedback and randomness of the
pnp.wonky\textsuperscript{−} object and the panning speed of the pnp.panner\textsuperscript{−} object. The degrade\textsuperscript{−} object in Max is also used and controlled by the amplitude to change the effective sampling rate and create distortion on these newly added sounds.

In order to activate mode 2, two onsets must be detected within the span of 500ms, meaning the performer must strike the singing bowl two times quickly. This mode disables chord changes and glissandi from future onsets within this mode and freezes the current chord, which can be deactivated by striking the singing bowl two additional times within the span of 500ms. Mode 2 adds three delay lines of different lengths and pitches using the pnp.delay\textsuperscript{−} and pnp.pitchshift\textsuperscript{−} objects. This mode also uses the pnp.grain\textsuperscript{−} object to create high pitched sounds that accompany the harmony presented by the delayed tones.
Mode 3 also uses onset detection, requiring 15 seconds without an onset in order to trigger activation. This is done by sustaining a sound on the singing bowl without striking it. Once this mode is activated, several new pitches and reverb using the pnp.reverb^ object is gradually introduced. To deactivate this mode, sustaining the sound for 15 seconds without an onset is required. While in this mode, the other two modes may be activated. This gives the performer some amount of agency in terms of the performance, as they go between modes or activate several simultaneously.
A diagram detailing how each mode is activated, whether by onset or spectral content detection, is shown above. This diagram is given to the performer as a visual reference for what mode is triggered by what action. Additionally, the presentation view of the patch displays the current mode for the performer as a reference throughout the performance.
CONCLUSION

Music information retrieval research is an interdisciplinary science that is still in its infancy, particularly within the computer music paradigm. This research is the first to define a performance practice around music information retrieval algorithms for music composition and improvisation rather than strictly for analysis. The approach outlined in this dissertation is advantageous for creating novel and meaningful interactions between human and computer in numerous ways. These interactions are similar to interactions found in machine learning and robotics systems, which respond according to inputs to the system, although they do not require initial training data.

The potential benefits for composers and performers using these systems, called reflexive systems, are numerous. Systems built using \textit{PnP.Maxtools} package were demonstrated as tools for abstraction, making tasks such as technological requirements and set up a much simpler and more efficient process. They also preserve traditional performance practices with regards to notation and composition while encouraging composers to design systems that are thoughtful with respect to identifiable features within a performer's and instruments sound.

The term Reflexive Automation describes this novel approach to computer automation that the \textit{PnP.Maxtools} objects afford. All of the timbral descriptors are implemented in a real-time context, making this the first real-time timbral descriptor package for subjective descriptors in Max. It is also the first MIR package to include additional pre and post processing functionality, including filters, additional controls, and audio effects that are all designed to use normalized floating point numbers between 0-1. When compared to other toolboxes which utilize MIR algorithms, \textit{PnP.Maxtools} offers more control over the creative process through a ‘plug and play’ style of programming that enables the user to achieve fully autonomous control over musical parameters using only live input from a microphone.

Several compositions were written that demonstrate the categorical framework and various reflexive systems. While only a handful of systems using the \textit{PnP.Maxtools} objects were
built and described, the potential degree of variety and interaction that can be created with this software is incredibly vast because it can be adapted to any live performance setting and instrumentation. The pieces presented in this research include traditionally notated scores and improvisational elements as well as continuous and event detection functions utilizing many of the package objects on solo and chamber compositions.

Finally, future work and research with regards to building event detection functions and sonification and musification is warranted. For event detection, several strategies, including automatic signal scaling, avoiding repetitive triggers, and compound descriptors were shown as beneficial outside of previously described event detection functions using timbral variations such as changes in the spectral slope and spectral standard deviation. This is presented as a starting point for detecting events and is an area ripe with exploration and experimentation.

The similarities between sonification and musification were highlighted in terms of mapping and automation techniques, but the difference in scope and offline use are important factors that keep sonification and musification methods from becoming fully reflexive. However, implementing offline data and/or inaudible data such as that from a live video feed present interesting research opportunities that can expand the possibilities and conceptual scope detailed in this dissertation. More work is also needed to reduce technological demands further, and could warrant developing a single hardware device capable of running reflexive systems that only needs to be powered on to use. This could also be developed as an educational tool for introducing young musicians to electronic music performance.

The challenge facing artists utilizing the \textit{PnP.Maxtools} software is twofold with high reward, requiring an artist to wear the hat of programmer and scientist. An understanding of music information retrieval and Reflexive Automation is necessary in order to use this technology for creative endeavors. However, building systems with this technology that are elegant, thoughtful, and creative with respect to identifiable spectral features and resulting reflexes is an art form without any singular ideal approach. The affordances of this technology rewards us with the ability to create art that would not be possible without its aid, and many
new interactions will ultimately be achieved as a result of this technology and the labor of many talented musicians, programmers, and researcher to come.
APPENDIX A. PNP.MAXTOOLS DOCUMENTATION

The following is the complete documentation for the *PnP.Maxtools* objects. The link to download the repository is [https://github.com/austinfranklin/pnp.maxtools](https://github.com/austinfranklin/pnp.maxtools).

A.1. Filters

A.1.1. **pnp.binpass˜**

*pnp.binpass˜* is a filter that utilizes an FFT to remove frequencies from an input signal outside of a designated low and high frequency range. Since it can only remove entire bins in the frequency domain, it calculates which bins contain the entire low and high frequency range. This means that the output signal will sometimes contain additional frequencies outside of this range, but only the frequencies that lie in the same bin as either the low or high frequency arguments.

A.1.2. **pnp.notch˜**

*pnp.notch˜* is a filter that utilizes an FFT to remove frequencies from an input signal inside of a designated low and high frequency range. Since it can only remove entire bins in the frequency domain, it calculates which bins contain the entire low and high frequency range. This means that the output signal will sometimes contain additional frequencies outside of this range, but only the frequencies that lie in the same bin as either the low or high frequency arguments.

A.1.3. **pnp.overtone˜**

*pnp.overtone˜* is an FFT filter that takes a fundamental frequency as an argument and filters an incoming signal using integer multiples up to 8*fundamental. Each multiple is given a separate outlet that can be combined with others or used independently.

A.1.4. **pnp.pitchclass˜**

*pnp.pitchclass˜* is an FFT filter that takes a fundamental frequency as an argument and filters an incoming signal up to 7 octaves above the fundamental (128*fundamental). Each octave is given a separate outlet that can be combined with others or used independently.
A.1.5. `pnp.reduce`

`pnp.reduce` performs an FFT analysis on an incoming signal and filters out bins that are quieter than a specified amplitude threshold. This object can be used to reduce ambient and room noise from signals.

A.2. Sound Content Descriptors

A.2.1. `pnp.amplitude`

`pnp.amplitude` outputs the amplitude of an incoming signal in the range of 0-1. An additional smoothness argument may be specified that smooths the output values. This can sometimes be used to get a more accurate or cleaner representation of the amplitude signal.

A.2.2. `pnp.autoamp`

`pnp.autoamp` outputs the amplitude of an incoming signal in the range of 0-1 like `pnp.amplitude`, except the output is first scaled using the loudest measured part of the signal as the upper threshold. An additional smoothness argument may be specified that smooths the output values as well as manually sets the peak amplitude. This can sometimes be used to get a more accurate or cleaner representation of the amplitude signal.

A.2.3. `pnp.autoregi`

`pnp.autoregi` returns a normalized floating point number that indicates where the detected pitch is located between detected low and high frequency arguments. The object sets the low and high frequency arguments automatically using fundamental frequency detection. The smoothness argument can be used to output a more gradual change in output.

A.2.4. `pnp.beat`

`pnp.beat` reports the onset of an incoming signal as a bang, as well as the velocity of each reported onset as a normalized floating point number between 0-1.

A.2.5. `pnp.boominess`

A boomy sound is one that conveys a sense of loudness, depth and resonance. `pnp.boominess` calculates the apparent boominess of an incoming audio signal by measuring the low fre-
quency content of a sound. The greater the proportion of low frequencies the greater the measure of boominess will be.

**A.2.6. pnp.bpm~**

pnp.bpm~ estimates the BPM of the onsets of an incoming audio signal.

**A.2.7. pnp.brightness~**

A bright sound is one that is clear/vibrant and/or contains significant high-pitched elements. pnp.brightness~ calculates the apparent brightness of an incoming audio signal by calculating the upper spectral centroid between 3000-20,050Hz and the ratio of energy between this range and 0-20,050Hz.

**A.2.8. pnp.centroid~**

pnp.centroid~ calculates the spectral centroid of an incoming signal as well as the amplitude of the bin that contains the centroid.

**A.2.9. pnp.depth~**

A deep sound is one that conveys the sense of having been made far down below the surface of its source. pnp.depth~ calculates the apparent deepness of an incoming audio signal by calculating the lower spectral centroid between 30-200Hz and the ratio of energy between this range and 0 and the Nyquist Frequency.

**A.2.10. pnp.descriptor~**

pnp.descriptor~ is a variable descriptor that calculates a frequency limited spectral centroid between low and high frequency arguments and the ratio of energy between this range and 0-20,050Hz.

**A.2.11. pnp.energy~**

pnp.energy~ calculates the spectral energy of each FFT frame.

**A.2.12. pnp.flatness~**

pnp.flatness~ calculates the degree to which the frequencies in the frequency spectrum of the signal are evenly distributed (noise-like). Smaller values indicate a noisier signal and larger values indicate purer tones.
A.2.13. pnp.hardness

A hard sound is one that conveys the sense of having been made (i) by something solid, firm or rigid; or (ii) with a great deal of force. pnp.hardness calculates the apparent hardness of an incoming audio signal by measuring the attack gradient and spectral content of each attack.

A.2.14. pnp.metallic

pnp.metallic calculates the probability that the source of the incoming sound is produced by a metallic object by measuring the roughness, the spectral spread, and the normalized decay time of the signal.

A.2.15. pnp.multi

pnp.multi calculates the boominess, brightness, depth, hardness, metallic probability, roughness, sharpness, and warmth of an incoming signal simultaneously. This object can be used in place of many individual descriptors on the same signal for increased computational efficiency, since only one FFT is used for all descriptors.

A.2.16. pnp.register

pnp.register returns a normalized floating point number that indicates where the detected pitch is located between low and high frequency arguments. The smoothness argument can be used to output a more gradual change in output.

A.2.17. pnp.roughness

A rough sound is one that has an uneven or irregular sonic texture. pnp.roughness calculates the apparent roughness of an incoming signal by measuring how “rough” the spectral content of the sound is.

A.2.18. pnp.sharpness

A sharp sound is one that suggests it might cut if it were to take on physical form. pnp.sharpness calculates the apparent sharpness of an incoming audio signal by measuring the high frequency content of a sound. The greater the proportion of high frequencies the greater the measure of sharpness will be.
A.2.19. pnp.spread~

pnp.spread~ calculates the spread, or the spectral centroid variance of an incoming audio signal. Additionally, the spectral standard deviation may be calculated using the right outlet from the object.

A.2.20. pnp.warmth~

A warm sound is one that promotes a sensation analogous to that caused by a physical increase in temperature. pnp.warmth calculates the apparent warmth of an incoming audio signal by using a fundamental frequency estimator to calculate the mean warmth region. The spectral centroid in this range is calculated along with the ratio of energy between this range and 0-20,050Hz.

A.3. Additional Controls

A.3.1. pnp.autoscale~

pnp.autoscale~ automatically scales an incoming audio signal to a specified target amplitude level in the range of 0-1. The object takes two arguments: the amplitude level to maintain, and the minimum amplitude that triggers scaling. This object can be used before descriptors in a signal chain to prevent changes in amplitude from effecting the output.

A.3.2. pnp.bangs

pnp.bangs outputs an integer number of bangs separated by a specified delay time in milliseconds. The delay time can be fixed or it can change during a series of bangs. This object can be used as a substitute for a metro or counter object to trigger events. By setting the delay time to 0ms it can also function as a uzi object.

A.3.3. pnp.noone

pnp.noone filters out every 1 value it receives, holding over the previous value until a new non 1 value is received. A bang is also sent out of the right outlet that can be used as a trigger for events. Additionally, a ramp time can be specified in the right inlet that will ramp between non 1 values.
A.3.4. **pnp.nozero**

pnp.nozero filters out every 0 value it receives, holding over the previous value until a new nonzero value is received. A bang is also sent out of the right outlet that can be used as a trigger for events. Additionally, a ramp time can be specified in the right inlet that will ramp between nonzero values.

A.3.5. **pnp.smoother**

pnp.smoother smoothes and can apply exponential or logarithmic curves to incoming data in the range of 0-1. Out of the right outlet, the output values are inverted (0=1, 1=0).

A.4. **Effects**

A.4.1. **pnp.convolve**

This effect utilizes and FFT to convolve two signals together. Two input signals are needed as inputs as well as an additional optional argument which determines the relative strength, or mix, of each signal in the output. This can be set with a floating point number between 0 and 1.

A.4.2. **pnp.delay**

pnp.delay is a variable delay line. The delay time can be specified in milliseconds and changed in real-time to create delay times of different lengths. An optional argument, window size, can be specified but defaults to 256 if left unspecified.

A.4.3. **pnp.distort**

pnp.distort applies distortion to an incoming signal using a drive and distortion parameter. Both of these can be set using a floating point number between 0 and 1, but default to 0 (no distortion) if unspecified.

A.4.4. **pnp.flange**

pnp.flange is a flanger built in gen that uses speed (Hz), width, and feedback parameters to create the effect. Width and feedback can be set using a floating point number between 0 and 1, but default to 0 (no flange effect) if unspecified.
A.4.5. pnp.freeze˜

pnp.freeze˜ is an object that captures and loops and FFT frame using a jitter matrix. This object has no arguments that can be specified. To freeze audio, send a bang message into the left inlet.

A.4.6. pnp.grain˜

pnp.grain˜ takes an incoming audio signal and performs real-time granular synthesis, using parameters such as grain rate, duration, and pitch to vary the grains. These parameters can be set using a floating point number between 0 and 1, but default to 0 if unspecified.

A.4.7. pnp.panner˜

pnp.panner˜ pans an incoming signal between the left and right channels depending on the position argument set. The fade speed can also be set that determines the rate of transition between changes to the position.

A.4.8. pnp.pitchshift˜

pnp.pitchshift˜ is a pitchshifter that uses an FFT to transpose an incoming signal using semitones, where an increase by 1 is a transposition factor of 1 semitone. Setting negative values transposes the resulting pitch down. The object semitone argument defaults to 0 if unspecified.

A.4.9. pnp.pluck˜

pnp.pluck˜ is an implementation of the famous Karplus-Strong plucked string algorithm where live input is substituted for a burst of noise. The delay time and feedback can be adjusted with a number between 0-1. The filter cutoff can be specified as an additional argument, but it defaults to 8kHz.

A.4.10. pnp.reverb˜

pnp.reverb˜ is a stereo plate reverb effect where the room size and decay time can be specified using a floating point number between 0 and 1. A wet/dry mix parameter can also be set using a floating point number between 0 and 1.
A.4.11. pnp.shuffle

pnp.shuffle delays multiple copies of itself at different lengths and randomly sends them out at different times, creating a shuffling effect. The overall shuffle speed can be set as a floating point number between 0 and 1, but defaults to 0.5 if unspecified.

A.4.12. pnp.split

pnp.split splits an audio signal using an FFT at a specified frequency bin. The split position determines how much of each frame will come out of either the left or right outlets. It can be set with a floating point number between 0 and 1, but defaults to 0.125 if left unspecified.

A.4.13. pnp.wonky

pnp.wonky is an effect that changes delay time using a randomly changing delay time and feedback amount. The frequency of the changing delay time and the feedback amount can be adjusted using a floating point number between 0 and 1.
APPENDIX B. *BLOOM SCORE*
Austin Franklin

Bloom
for violoncello and reflexive electronics

(2021)
“A flower does not use words to announce its arrival to the world; it just blooms.”
— Matshona Dhliwayo
**Program Notes**

Bloom is a piece for violoncello and reflexive electronics that explores tension, using the metaphor of a blooming flower as the basis from which the musical material and form are derived. The work begins with a very simple melodic idea using natural harmonics. These harmonics are developed throughout the piece, eventually blurring the line between pitch and noise, meter and aleatory, and acoustic and electronic elements. The electronic element of the piece is realized using live input from the cello only. This relies on specific musical parameters (namely amplitude and frequency) to control how the input is processed. The piece concludes with a quasi-recapitulation of the opening, this time incorporating non-harmonic tones. This is the most mature statement of the original melodic idea in the piece, which signifies completion of the flowering process.

**Technical Notes**

Bloom uses a Max MSP program (or patch) by which to process and transform the audio. The patch contains instructions on how to set up, operate, rehearse, and finally perform the piece. Please email me for the patch after purchasing, or if you have any questions directly at austinalexanderfranklin12@gmail.com. As for the technical requirements and equipment set up, please use the following configuration:
**Performance notes**

Ord.  
ordinario (normal manner of playing)

OP  
apply bow pressure to distort pitch, but not remove it completely

highest possible pitch on the given string(s)

The electronic sounds used in the piece come solely from the live input to the microphone. There are no triggers for the individual sections, and no MIDI keyboard or instrument is required. Instead, the processing is controlled via specific musical parameters. The resulting sound is not notated in the score since it will sound different for repeat performances. However, the musical parameters and their general effects on the input are described as follows:

- As the loudness increases, the reverb decay time is increased. Brightness, or “glitteryness” is also increased via down sampling on the signal.

- As lower pitches are performed, a variable delay time is increased. There are three separate delay lines in total, each with independent delay times that are determined by pitch. Frequencies below A 440Hz are also pitch shifted down a perfect 5th.

- As the rate of discrete pitch change increases, the location of a sound in the stereo field is varied more rapidly.

With the exception of the improvised section at letter C, the pitches and dynamics should be taken as literally as possible because they exhibit the most control over audio processing.

The notation provided at the improvised section should be used as a guide, considering the way the electronics operate. This section may be performed as written or may be changed to incorporate more percussive elements (knocking on body of instrument, snap pizz., etc.). However, the overall length of the section should be mostly left unchanged. The electronics during this section should also be at their most active.

**Bloom** was commissioned by Eduard Teregulov

**Duration:** ca. 8’00”
Cello: Natural Harmonics

The fingerings and harmonics in the piece were composed using the following natural harmonics chart by Andrew Hugill. Sounding pitches are not provided in the score but can be referenced here if required. The roman numeral notation in the score corresponds to the string number on which the harmonic should be performed.
Bloom
for violoncello and Reflexive electronics

\[\text{\textit{Tranquil}}\]  

\(\text{\textit{Emerging}}\)
APPENDIX C. *CONCENTRIC CIRCLES* SCORE
Austin Franklin

Concentric Circles
for percussion, cello, piano, and reflexive electronics

(2021)
Program Notes

Concentric Circles is a piece that explores overlapping patterns and cycles. It revolves around an initial harmonic progression that descends by step and is varied upon each repetition. Throughout the piece new patterns are introduced or layered on top of existing patterns. The electronics operate by analyzing and averaging data from all players simultaneously, among other processes. This creates new patterns and interactions between performers throughout the piece. These patterns evolve throughout the piece until they transform back into the opening statement.

Technical notes

Concentric Circles uses a Max MSP program (or patch) by which to process and transform the audio. The patch contains instructions on how to set up, operate, rehearse, and finally perform the piece. Please email me for the patch after purchasing, or if you have any questions directly at austinalexanderfranklin12@gmail.com. As for the technical requirements and equipment set up, please use the following configuration:

- A four-channel audio interface is required to perform the piece, and 4 large diaphragm microphones should be used for inputs if possible.
- Microphones should correspond to the input on the interface provided in the graphic above (1-4). Inputs 3 and 4 should be used on the low and high register of the piano, respectively.
- Microphone placement will generally work best closest to the instrument (or amp) but any position that captures the full range of the instrument is sufficient.
- An attempt should be made to place microphones in a position that does not capture other instruments or reduces the amount of bleed from other instruments as much as possible.
- Loudspeakers should be positioned in front of the ensemble to prevent feedback.
Performance notes

The electronic sounds used in the piece come solely from the live input to the microphone. There are no triggers for the individual sections, and no MIDI keyboard or instrument is required. Instead, the range of each instrument is divided into different sections that are processed independently from the others via specific musical parameters. The resulting sound is not notated in the score since it will sound different for repeat performances. The musical parameters and their general effects on the input are described as follows:

Percussion:
- The entire range of the crotales are processed using a variable delay and feedback.
- The entire range of the vibraphone is processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave respectively. The register of the vibraphone determines what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent).

Bassoon:
- Pitches above C4 are processed using a delay line that is pitch shifted up a 5th. Brightness, or “glitteryness” is also increased via down sampling on the signal. Loudness determines the length of the delay and the brightness of the signal.
- Pitches above G2 and below C4 are processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave respectively. The register of the cello determines what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent).
- Pitches below G2 are processed using a delay line and amplified. There are no controlling parameters for this register.

Synthesizer:
- Pitches above C5 (notated as harmonics in the score) are processed using a delay line that is pitch shifted up a 5th. Brightness, or “glitteryness” is also increased via down sampling on the signal.
- Pitches above G2 and below C5 are processed with two variable delay lines that are each pitch shifted up a perfect 5th and octave respectively. The register of the piano determines what transposition is loudest (in the lower register the perfect 5th transposition is prominent, in the higher register the octave transposition is prominent).
- Pitches below G2 are processed using a delay line and amplified. There are no controlling parameters for this register.

Unless explicitly mentioned above, the amount of processing is determined by the loudness of the individual performers within a particular register (described above) AND the average overall loudness of the ensemble. When all performers are playing at their loudest dynamic, the loudness of the electronics is at its maximum. When a single performer plays loudly while the rest of the ensemble is quiet, the amount of processing is minimal. Reverb decay time is also affected solely by the average loudness of the ensemble. Performers should pay special attention to their dynamics throughout the piece, specifically during moments where accents or “swells” are present and bring these out amongst the surrounding texture.

Concentric Circles was commissioned by Jacob Ottmer

Duration: ca. 10'00”
Concentric Circles
for percussion, cello, piano, and reflexive electronics

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \frac{\text{\textcopyright}}{\text{Austin Franklin}} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)

\( \text{\textcopyright} \text{Austin Franklin} \)
Concentric Circles - Full Score

Vib.

Vc.

Pno.

Play any note from the given chord on any beat without a notehead, using the stems register as a guide. Accented notes must stand out from the texture.

A

Ebm Bbm Db+

Crotales

mf

Vib.

mp

sim.

Vc.

Pno.

mf mp

sim.
Concentric Circles - Full Score

F#m  A+  C#m  Fm  Eb

Crot.

Vib.

Vc.

Pno.

B

Crot.

Vib.

Vc.

Pno.

mf  mp

mf  mp

154
Play independently. Dashed lines between instruments indicate approximate synchronization. All note changes should occur within 3-6 seconds of each other. Accidentals apply to only the note they are on.

Crot.

Vib.

Vc.

Pno.

pedal every change

pedal every change
Concentric Circles - Full Score

rit.

Vib.

Vc.

Pno.

\[ \text{G} \quad \downarrow = 155 \quad \text{repeat 1-3x} \]

mf pedal freely

pizz.

mp

mf pedal freely
All instruments accel. independently until I. All metric modulations until N relate to the same tempo ($J = 155$).
Play each figure repeatedly and independently. Each instrument should cue when they change pattern, and there should be 5-8" between changes.
Vib.
slowly begin to ritard.

Vc.
slowly begin to ritard.

Pno.
slowly press pedal

Vib.
imf

Vc.

Pno.
APPENDIX D. I/O SCORE
Austin Franklin

I/O
for snare drum and reflexive electronics

(2022)

© Copyright 2022, Austin Franklin Music (BMI)
All Rights Reserved
Program Notes

I/O for snare drum and reflexive electronics begins with a single two note idea that is developed and transformed over the duration of the piece. This idea is developed differently depending on whether the snares are turned on or off on the drum. When they are on, the character more closely resembles a drum-core or traditional playing style, and when they are off it becomes improvisational and explosive. The electronics also behave differently depending on whether the snares are on or off. It is this concept of duality that is expressed throughout the work, held together by a single short melodic idea.

Technical Notes

I/O uses a Max MSP program (or patch) by which to process and transform the audio. The patch contains instructions on how to set up, operate, rehearse, and finally perform the piece. Please email me for the patch after purchasing, or if you have any questions directly at austinalexanderfranklin12@gmail.com. As for the technical requirements and equipment set up, please use the following configuration:

![Diagram of audio setup](image.png)

Performance notes

There are no controllers or triggers for the electronics. Instead, the sound is processed using live input from the microphone only, so the performance will vary depending on the snare drum used and other musical factors. The resulting sound is not notated in the score since it will sound different for repeat performances. However, the musical parameters and their general effects on the input are determined primarily by whether the snares on the drum are on or off. When off, the electronic sounds should generally be much more chaotic. The patch contains a display that shows whether the patch has detected whether the snares are on or off.

The following notation is used throughout the piece:
I/O was commissioned by Micheal Barnes

Duration: ca. 6’00”
I/O
for solo snare drum and reflexive electronics

Austin Franklin

\( \text{\textdollar} = 155 \)

- turn snares on/off
- r w/ stick

on rim

mp

f

mp
170 gradually turn on snares on

172

175 f mp ff
APPENDIX E. SING! SCORE
Austin Franklin

**Sing!**
for singing bowl and reflexive electronics

(2022)

© Copyright 2022, Austin Franklin Music (BMI)
All Rights Reserved
Program Notes

Sing! is an improvisational piece that explores an instrument with a single sound: a singing bowl. Using several MIR algorithms and reflexive electronics, the sound of a single sustained pitch is transformed throughout the piece. The piece is not notated, but rather instructs the performer to improvise using specific ways of triggering changes in the electronics.

Technical Notes

Sing! uses a Max MSP program (or patch) by which to process and transform the audio. The patch contains instructions on how to set up, operate, rehearse, and finally perform the piece. Please email me for the patch after purchasing, or if you have any questions directly at austinalexanderfranklin12@gmail.com. As for the technical requirements and equipment set up, please use the following configuration:

Performance notes

There are no controllers or external triggers for the electronics. Instead, the sound is processed using live input from the microphone only, so the performance will vary depending on the fundamental frequency and resonance of the singing bowl. The sound of the electronics is built into 3 different ‘modes.’ Activating one mode will cause the electronics to vary the sound in a particular way that distinguishes it from other modes. When no mode is activated, the electronics harmonize with the singing bowl to create seventh chords. These chords can be changed by hitting the singing bowl with the striker. A soft hit will result in an instantaneous chord change while a hard hit will result in a gradual chord change using glissandi between notes.

The modes and method of activation is shown below on the following diagram. Mode 1 is activate using the brightest sound on the signing bowl, a loud strike with the wooden portion of the striker. Mode 2 and Mode 3 are activated using onset, with Mode 2 being activated using two
quick hits less than 500ms apart. Finally, Mode 3 can be activated by sustaining the sound on the singing bowl for at least 15 seconds without striking it.

The sound of Mode 1 will result in distortion and additional high frequencies added to the sound. Mode 2 creates several delays and granulates the sound, while Mode 3 adds reverb and additional harmonies to create a wide texture. Each Mode may be activated while any other Mode is activated. This allows for two or all Modes to be activated simultaneously, allowing for any combination of effects.

Duration: ca. 5-7’
APPENDIX F. INTERNATIONAL REVIEW BOARD DOCUMENTS
TO: Jesse Taggart Allison
LSUAM | Col of MDA | Music | CC00229
FROM: Alex Cohen
Chairman, Institutional Review Board
DATE: 08-Dec-2022
RE: IRBAM-22-0395
TITLE: Dissertation: Autonomous Parameter Control in MaxMSP
Utilizing Music Information Retrieval Algorithms
SUBMISSION TYPE: Initial Application
Review Type: Exempt
Risk Factor: Minimal
Review Date: 08-Dec-2022
Status: Approved
Approval Date: 08-Dec-2022
Approval Expiration Date: 07-Dec-2025
Exempt Category: 2a
Requesting Waiver of Informed Consent: Yes
Re-review frequency: Three Years
Number of subjects approved: 10
LSU Proposal Number:

By: Alex Cohen, Chairman

Continuing approval is CONDITIONAL on:

1. Adherence to the approved protocol, familiarity with, and adherence to the ethical standards of the Belmont Report, and LSU's Assurance of Compliance with DHHS regulations for the protection of human subjects*
2. Prior approval of any change in protocol, including revision of the consent documents or an increase in the number of subjects over that approved.
3. Obtaining renewed approval (or submittal of a termination report), prior to the approval expiration date, upon request by the IRB office (irrespective of when the project actually begins); notification of project termination.
4. Retention of documentation of informed consent and study records for at least 3 years after the study ends.
5. Continuing attention to the physical and psychological well-being and informed consent of the individual participants, including notification of new information that might affect consent.
6. A prompt report to the IRB of any adverse event affecting a participant potentially arising from the study.
8. SPECIAL NOTE: When emailing more than one recipient, make sure you use bcc. Approvals will automatically be closed by the IRB on the expiration date unless the PI requests a continuation.

* All investigators and support staff have access to copies of the Belmont Report, LSU's Assurance with DHHS, DHHS (45 CFR 46) and FDA regulations governing use of human subjects, and other relevant documents in print in this office or on our World Wide Web site at http://www.lsu.edu/research

Louisiana State University
131 David Boyd Hall
Baton Rouge, LA 70803
O 225-578-5833
F 225-578-5983
http://www.lsu.edu/research
BIBLIOGRAPHY


Bogdanov, Dmitry, Nicolas Wack, Emilia Gómez Gutiérrez, Sankalp Gulati, Herrera Boyer, Oscar Mayor, Gerard Roma Trepat, Justin Salamon, José Ricardo Zapata González, Xavier Serra, et al. “Essentia: An audio analysis library for music information retrieval.” In Britto A, Gouyon F, Dixon S, editors. 14th Conference of the International Society for Music Information Retrieval (ISMIR); 2013 Nov 4-8; Curitiba, Brazil.[place unknown]:

194


Franklin, Austin, Dylan Burchett, and William Thompson IV. “Composing and Improvising Using Sound Content-Based Descriptive Filtering,” 2021.


Omori, H. “Determining the nature of the difference between perceived brightness and brightness predicted by spectral centroid.” Undergraduate technical project, Institute of Sound Recording, University of Surrey, Guildford, Surrey, UK, 2011.


“pfft Reference.” https://docs.cycling74.com/max7/refpages/pfft~


Rawlinson, Hugh, Nevo Segal, and Jakub Fiala. “Meyda: an audio feature extraction library for the web audio api.” In The 1st web audio conference (WAC). Paris, Fr. 2015.


Williams, Duncan. *Towards a timbre morpher*. University of Surrey (United Kingdom), 2010.


VITA

Austin Franklin is an internationally recognized composer and sound artist currently based in Baton Rouge, LA. His interests include music involving process, such as algorithmic composition and music incorporating music information retrieval and machine learning technologies. His music has been described as having “striking effects of togetherness” and “a sense of an ending” (New York Concert Review). His album in 2020 entitled Four Idols has been described as “an elegant, artistic statement that demonstrates the flexible possibilities of electronic music” (The Sybaritic Singer). Austin has several pieces for percussion published through C-Alan Publications and his music has been performed throughout North America, South America, Europe, Africa, and Asia by ensembles such as Hypercube, The Estrella Consort, and the Four Corners Ensemble. In 2022, he made his Carnegie Hall debut with a performance of his String Quartet No. 1 “Lanterns”.

Austin is the recipient of numerous awards and commissions, including 2nd Place in the American Prize for Composition, the RMN Call for Electroacoustic Works, PARMA Winter Call for Scores, and the ABLAZE Electronic Masters Series Call for Recordings. His music has also been selected for festivals and conferences such as the Fairbanks Summer Arts Festival, International Music Information Retrieval Society (ISMIR), New Music Mosaic Festival (NMM), Society of Electro-Acoustic Music in the United States (SEAMUS), Napoleon Electronic Media Festival (NEMF), Festival Ecos Urbanos, the New Music on the Bayou Festival (NMOB), Splice Institute, New York City Electronic Music Festival (NYCEMF), Workshop on Computer Music and Audio Technology (WOCMAT), Alba Music Festival, Society of Composers Incorporated (SCI), and Electric LaTex. As a technologist, he has presented research at the Web Audio Conference (WAC) that explores using Web API’s as the basis for designing digital instruments along with net art installations that explore collaboration and interactivity on the web, and at the New Interfaces for Musical Expression
Conference (NIME) that involves simultaneous auditory and vibrotactile stimuli. For more visit austinfranklinmusic.com.