International Conference on New Interfaces for Musical Expression

Aaf.Maxtools: Autonomous Parameter Control in Max Utilizing MIR Algorithms

Austin Franklin[1], Jesse Allison[1]

[1]Louisiana State University

Published on: Aug 26, 2021

ABSTRACT

This paper documents the development of *Aaf.Maxtools*: a set of open source objects designed for the real-time automation of control parameters within the popular programming environment Max/MSP. The package is a set of filters, signal analysis and sound description tools, audio effects, and other objects that utilize several music information retrieval algorithms and are designed to be used together to compose music or improvise without the use of external controllers or hardware. A framework is also presented that demonstrates ideal usage of the objects for achieving fully autonomous control over musical parameters.

Author Keywords

Autonomous Parameter Control, Real-time, Music Information Retrieval

CCS Concepts

- **Software and its engineering → Software notations and tools → General programming languages → Language features** → *Modules / packages*;
- **Information systems → Information retrieval → Document Representation →** *Content analysis and feature selection;*
- **Human-centered computing → Interaction design** → *Interaction design process and methods;*

Introduction

Overview

Our interest concerns the use of real-time MIR and signal analysis algorithms for achieving autonomous control [1] of musical parameters. The *Aaf.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. This is done by specifying presets and default states for each object. Arguments for each object can be specified for extended control but are not needed for immediate use.

The *Aaf.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. The filters and additional controls can be used to aid with analysis or control audio effects. In this way, the audio content has a real musical relationship with the

resulting musical transformations while the control parameters become more multifaceted and better able to serve the needs of artists [2].

Framework

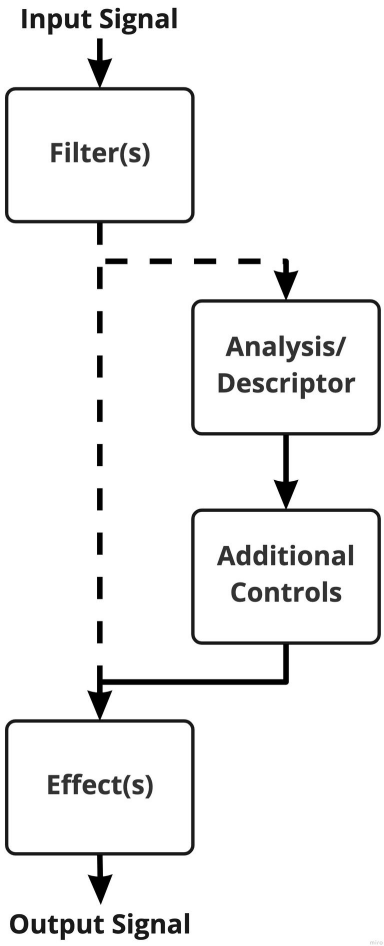

Figure 1. Proposed Signal Chain

Figure 1 demonstrates the proposed signal chain for the objects of the package, where the dashed lines indicate MSP patch cords and the solid lines indicate Max patch cords. While this is not the only possibility for automating control parameters, it is provided here as the starting point by which 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.

Implementation

Considerations

In Max specifically, many of the existing packages (such as Zsa.Descriptors [3], MnM Toolbox [4], FTM/Gabor Object Library [5], Ml.Lib [6], and fiddle~ and bonk~ by Miller Puckette [7]) require 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. Furthermore, none of these packages offer specific musical effects that are designed to be used in conjunction with the descriptors.

Documentation

In order to further demonstrate the automation of control parameters, the documentation for *Aaf.Maxtools* objects provides musical examples that utilize other objects in the package (Figure 2). An example of a help patch is shown below:

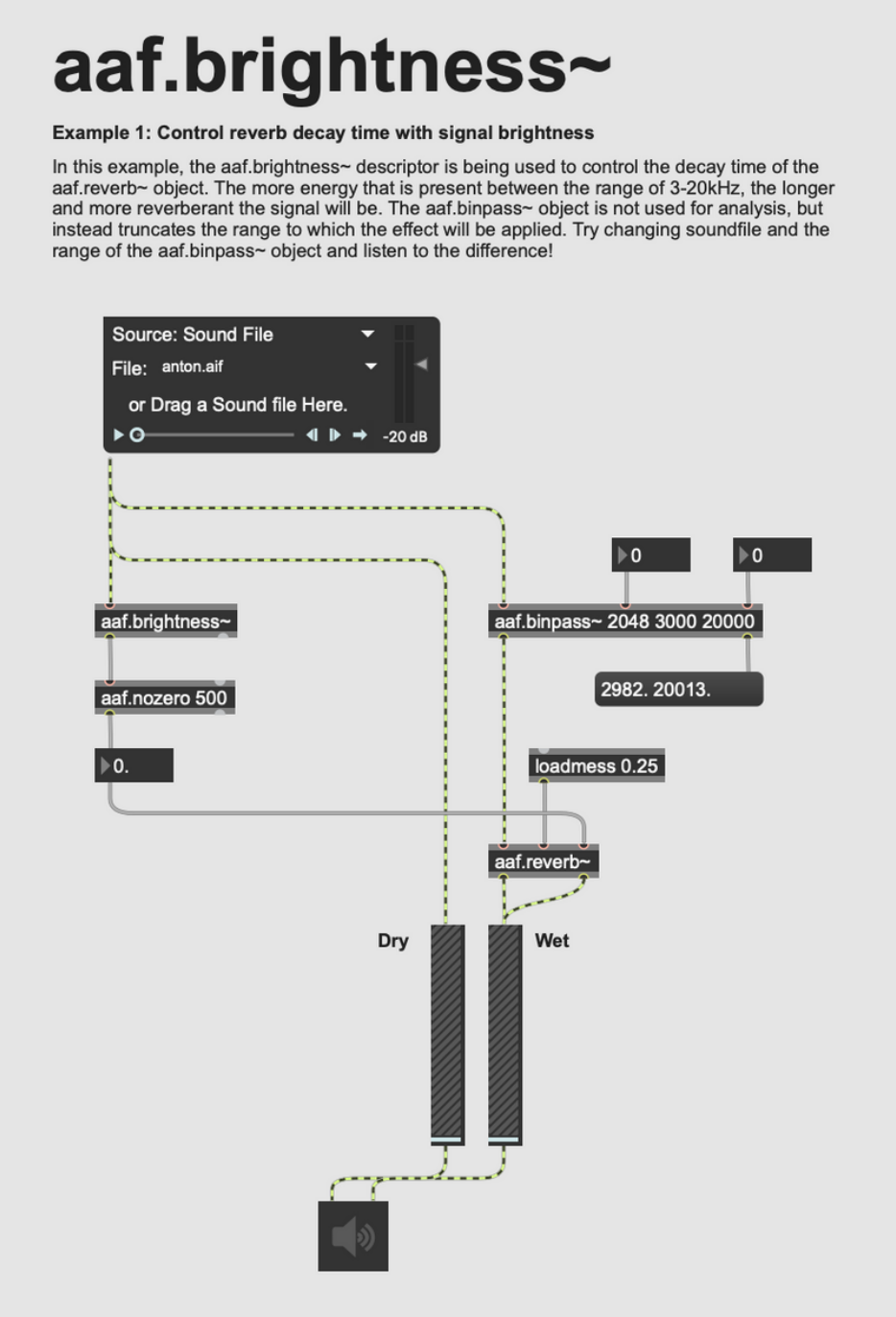

Figure 2. aaf.brightness~ Help Patch Example

The primary help patch describes the object: what it does, its inputs/outputs, and arguments (if required). Each object is also given at least two secondary help patches which demonstrate different musical applications for each object, and a final help patch is offered that gives a more in-depth and technical description of what algorithm is being implemented.

Package Design

The first release of the package contains a total of 36 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:

- 3 Filter Objects
- 8 Signal Analysis Objects
- 9 Sound Descriptors
- 4 Additional Controls
- 12 Audio Effects

Filters

Many of the filters are implemented using the pfft~ object in Max/MSP. 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 [8]. The objects and their descriptions are as follows:

- **aaf.binpass~** - An FFT filter modeled after a classic 'bandpass' filter that passes frequencies within a select frequency bin range. Low and high frequency arguments are required for this object.

- **aaf.notch~** - An FFT filter modeled after a classic 'notch' filter that removes frequencies within a select frequency bin range. Low and high frequency arguments are required for this object.

- **aaf.reduce~** - An FFT filter that reduces the noise content of an incoming audio signal. An amplitude argument between 0-1 is required for this object.

Signal Analysis

The signal analysis tools provide a wide range of data from an incoming audio signal. Most objects in this category output a number between 0-1. The objects and their

descriptions are as follows:

- **aaf.amplitude~** - Tracks the amplitude of an incoming audio signal. It outputs the amplitude between 0-1. No arguments are required for use.
- **aaf.register~** - A pitch detector that tracks the position of a frequency between a low and high frequency. It outputs a number between 0-1 that indicates where the detected pitch is located between a low and high frequency. Low and high frequency arguments are required for this object.
- **aaf.beat~** - An onset detector that outputs a bang when an onset is detected as well as a number between 0-1 that corresponds to the amplitude of the onset. No arguments are required for use.
- **aaf.interval~** - An interval detector that outputs a bang when an interval is detected. This object requires an argument between 1-12 that corresponds to the interval (in half steps) which is to be detected.
- **aaf.bpm~** - A BPM estimator that outputs a tempo in BPM [9]. No arguments are required for use.
- **aaf.fundamental~** - A fundamental frequency estimator that outputs a frequency that corresponds to the fundamental frequency of the signal being analyzed [10]. No arguments are required for use.
- **aaf.autoamp~** - An amplitude tracker that tracks the amplitude within an amplitude range, where 1 is set by the maximum amplitude of the signal. The range automatically adjusts if the maximum amplitude is exceeded and the amplitude position value is immediately scaled to the new maximum. No arguments are required for use.
- **aaf.autoregi~** - A pitch detector that tracks the position of a frequency between a frequency range where 0.0 is the lowest and 1.0 is the highest frequency detected. The frequency range automatically adjusts if the low or high frequency is exceeded and the frequency position value is immediately scaled to the new minimum and maximum. No arguments are required for use.

Sound Content Descriptors

The sound descriptors were implemented based on models from the AudioCommons “Release of timbral characterization tools for semantically annotating non-musical

content" [11]. This release describes models that analyze perceptual characteristics of sounds that were developed based on subjective listening experiments to gauge their effectiveness [12]. 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 [13]. They have been adapted for real-time use and are implemented using the pfft~ object in Max/MSP. The objects and their descriptions are as follows:

- **aaf.centroid~** - Calculates the spectral centroid (in hertz), or the barycentre of spectra, of an incoming audio signal [14]:

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

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

- **aaf.spread~** - Calculates the variance of the spectral centroid of an incoming audio signal [15]:

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

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

- **aaf.energy~** - Calculates the energy of each frequency bin [16]:

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

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

- **aaf.flatness~** - Calculates the spectral flatness of each frequency bin, or the ratio of the geometric mean to the arithmetic mean of the power spectrum [17]:

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

where *n(ω)* is the bin number relating to frequency *ω*, *n(Nyquist)* is the frequency relating to the Nyquist frequency, and *x(n)* is the magnitude of the n^{th} bin.

- **aaf.depth~** - Calculates the apparent 'depth' of an incoming audio signal [18]. The model calculates the lower spectral centroid as the spectral centroid of the frequencies between 30 Hz and 200 Hz:

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

where *n(ω)* is the bin number relating to frequency *ω*, *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:

$$\text{Ratio} = \frac{\sum_{n(30Hz)}^{n(200Hz)} x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}$$

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

- **aaf.brightness~** - Calculates the apparent 'brightness' of an incoming audio signal [19]. The model calculates the upper spectral centroid as the spectral centroid of the frequencies between 3KHz and the Nyquist frequency:

$$\text{Upper spectral centroid} = \frac{\sum_{n(3kHz)}^{n(Nyquist)} f(n)x(n)}{\sum_{n(3kHz)}^{n(Nyquist)} x(n)}$$

where *n(ω)* is the bin number relating to frequency *ω*, *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:

$$\text{Ratio} = \frac{\sum_{n(3kHz)}^{n(Nyquist)} x(n)}{\sum_{n(0Hz)}^{n(Nyquist)} x(n)}$$

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

- **aaf.hardness~** - Calculates the apparent 'hardness' of an incoming audio signal [20]. 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 125 ms:

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

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

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

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

- **aaf.warmth~** - calculates the apparent 'warmth' of an incoming audio signal. The model calculates the spectral centroid of the mean warmth region (fundamental * 3.5) [21]:

$$\text{Mean warmth region} = \frac{\sum_{n(fund)}^{n(fund*3.5)} f(n)x(n)}{\sum_{n(fund)}^{n(fund*3.5)} x(n)}$$

where *fund* is the fundamental frequency relating to the signal, *n(ω)* is the bin number relating to frequency *ω*, *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:

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

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

- **aaf.descriptor~** - A 'blank' descriptor that analyzes an incoming audio signal using an adjustable range [22]. 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:

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

where *n(ω)* is the bin number relating to frequency *ω, 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:

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

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

Additional Controls

The additional controls provide a wide range of functionality. Most objects in this category output a number between 0-1 that provides users with more control in terms of parameter automation or mapping. The objects and their descriptions are as follows:

- **aaf.bangs** - An object that extends the functionality of a bang. This object 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 a specific number of events. By setting the delay time to 0 ms it can also function as a uzi object.

- **aaf.smoother** - An object that 'smooths' values and can apply exponential curves to incoming values between 0-1. This object requires an argument that corresponds to the smoothness of the output values.

- **aaf.nozero** - An object that removes all ‘0’ values from incoming data. If a 0 is received by the object, it will ignore it and continue displaying the last received non-zero value. This object is design to be used in conjunction with sound analysis and descriptor objects that have a tendency to jump to 0 instantaneously when gaps in the sound are present. The ramp time (ms) between values can be set for more gradual changes. No arguments are required for use.

- **aaf.mmmr** - An object that calculates the mean, median, mode, and range of all normalized floating point values it receives. The object outputs all metrics when it receives a bang. This object can be used to calculate the mean, median, mode, or range from several descriptors simultaneously. No arguments are required for use.

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. The objects and their descriptions are as follows:

- **aaf.wonky~** - 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 number between 0-1.

- **aaf.panner~** - A stereo panner. The position between stereo left and right can be adjusted, where 0 is left and 1 is right. No arguments are required for use.

- **aaf.pluck~** - An implementation of the famous Karplus-Strong plucked string algorithm (Figure 3) [23]:

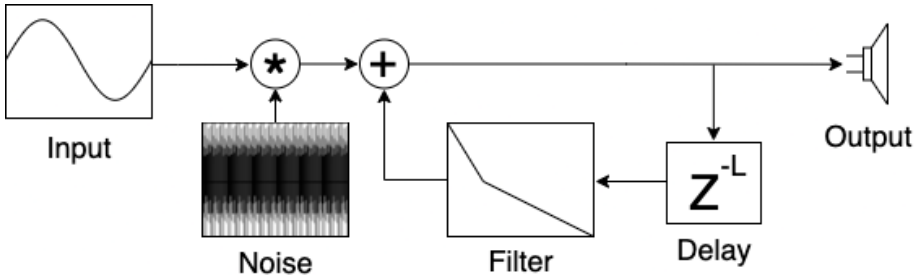

Figure 3. Modified Karplus-Strong Algorithm

  where the delay time can be adjusted with a number between 0-1. The filter cutoff can be specified as an additional argument, but it defaults to 8 kHz.

- **aaf.pitchshift~** - A pitch shifter implemented in pfft~. The object requires an argument that corresponds to the interval (in half steps) in which the signal is to be

transposed. Positive integers correspond to a higher transposition and negative integers correspond to a lower transposition.

- **aaf.grain~** - A real-time audio granulator. The size of the grain and the grain density can be specified using a number between 0-1. No arguments are required for use.
- **aaf.ang~** - An adaptive noise generative. This model analyzes peaks of an incoming audio signal and creates noise using frequencies present in the signal. The overall activity of the noise being generated is controlled with a number between 0-1. No arguments are required for use.
- **aaf.reverb~** - A plate reverb object. The room size and decay length can be adjusted with a number between 0-1. No arguments are required for use.
- **aaf.delay~** - A variable delay line. The delay time can be adjusted with a number between 0-1, or a time specified in milliseconds. No arguments are required for use.
- **aaf.tremolo~** - An object that creates a tremolo effect. The tremolo depth and frequency can be controlled with a number between 0-1. No arguments are required for use.
- **aaf.ringmod~** - An object that creates a ring modulated effect on an incoming signal. This object can either take two signals as inputs or control a sine wave with a number between 0-1 to create the effect. No arguments are required for use.
- **aaf.freqmod~** An object that creates a frequency modulated effect on an incoming signal. This object can either multiple signals as inputs or control the modulator and harmonicity with a number between 0-1 to create the effect. No arguments are required for use.
- **aaf.stutter~** - An object that is built using several delay lines of different lengths and a gate that randomly allows one to pass through at a time. The speed of the gate can be controlled with a number between 0-1. No arguments are required for use.

Future Work

Future work includes designing the next set of objects, including additional descriptors, controls, and effects. Some of the objects are being tested and discussed, but the following objects are currently in development:

- **aaf.roughness~** - Analyzes the apparent ‘roughness’ of an incoming audio signal [24] [25].
- **aaf.boominess~** - Analyzes the apparent ‘boominess’ of an incoming audio signal [26].
- **aaf.sharpness~** - Analyzes the apparent ‘sharpness’ of an incoming audio signal [27].
- **aaf.metallic~** - Analyzes the apparent ‘metallic-nature’ of an incoming audio signal [28] [29].
- **aaf.decay~** - Analyzes the apparent ‘reverberation’ of an incoming audio signal [30] [31].
- **aaf.chaos~** - An implementation of the famous Hindmarsh-Rose chaotic oscillator [32].

Additional future work includes considering methods to detect complex events from the analysis and descriptor objects [33], building GUI elements from collections of package objects that do specific musical things, and possibly developing a series of M4L plugins for real-time performance.

Demonstration

This demonstration is a prior work composed by the authors entitled Bloom for cello and live electronics which makes use of several objects, including aaf.amplitude~, aaf.range~, aaf.pitchshift~, aaf.reverb~, and aaf.panner~. The effects object are being controlled by the amplitude and frequency range of the instrument only, and requires only a clip on microphone and a computer running the Max patch for performance.

Visit the web version of this article to view interactive content.

Bloom for cello and live electronics (2021)

Conclusions

When compared to other packages from visual programming languages which utilize MIR algorithms, *Aaf.Maxtools* offers more control over the creative process through a ‘plug and play’ style of programming that includes both analysis and descriptor objects as well as a set of effects objects that are designed specifically to use the output from the analysis and descriptor objects in the form of a normalized range from 0-1. This enables the user to achieve fully autonomous control over musical parameters using only input from a microphone (or multiple microphones) and empowers performers

and creators to be able to engage with the music without interference from additional software or hardware controllers. Moreover, our package differs from prior packages in that the filters and additional controls are included and designed alongside the descriptors with the purpose of real-time creative musical expression. This project is ripe with the potential of additional effects and controls that go beyond the structural framework and objects currently presented.

Ethics Statement

Our project seeks inclusivity in the mode of accessibility and engagement through high abstraction and an emphasis on creative implementation. Our goal is to provide access to the technological means to be able to participate in the artistic process.

Citations

1. Michaud, Alyssa. "After the music box: A history of automation in real-time musical performance." (2020). ↩
2. Place, Timothy, Trond Lossius, Alexander Refsum Jensenius, and Nils Peters. "Flexible control of composite parameters in Max/MSP." (2008): 233-236. ↩
3. Mikhail Malt, Emmanuel Jourdan. Zsa.Descriptors: a library for real-time descriptors analysis. 5th Sound and Music Computing Conference, Berlin, Germany, Jul 2008, Berlin, Germany. pp.134-137. hal-01580326 ↩
4. Bevilacqua, Frédéric, Rémy Müller, and Norbert Schnell. "MnM: a Max/MSP mapping toolbox." In *New Interfaces for Musical Expression*, pp. 85-88. 2005. ↩
5. Schnell, Norbert, and Diemo Schwarz. "Gabor, multi-representation real-time analysis/synthesis." In *COST-G6 Conference on Digital Audio Effects (DAFx)*, pp. 122-126. 2005. ↩
6. Bullock, Jamie, and Ali Momeni. "Ml. lib: robust, cross-platform, open-source machine learning for max and pure data." In *NIME*, pp. 265-270. 2015. ↩
7. Puckette, Miller S., Miller S. Puckette Ucsd, and Theodore Apel. "Real-time audio analysis tools for Pd and MSP." (1998). ↩
8. https://docs.cycling74.com/max7/refpages/pfft~ ↩
9. Stark, Adam M., Matthew EP Davies, and Mark D. Plumbley. "Real-time beat-synchronous analysis of musical audio." In *Proceedings of the 12th Int. Conference*

on Digital Audio Effects, Como, Italy, pp. 299-304. 2009. ↩
10. Zbyszynski, Michael, David Zicarelli, and Regina Collecchia. "fzero~: fundamental estimation for Max 6." (2013): 342-347. ↩
11. Pearce, Andy, Saeid Safavi, Tim Brooks, Russell Mason, Wenwu Wang, and Mark Plumbley. “Release of timbral characterisation tools for semantically annotating non-musical content” *Audio Commons project deliverable D*5.8 (2019) ↩
12. Pearce, Andy, Tim Brookes, and Russell Mason. "Hierarchical ontology of timbral semantic descriptors." *Audio Commons project deliverable D*5.1 (2016). ↩
13. Gerhard, David. *Audio signal classification: History and current techniques*. Department of Computer Science, University of Regina, 2003. ↩
14. Mikhail Malt, Emmanuel Jourdan. Zsa.Descriptors: a library for real-time descriptors analysis. 5th Sound and Music Computing Conference, Berlin, Germany, Jul 2008, Berlin, Germany. pp.134-137. hal-01580326 ↩
15. Mikhail Malt, Emmanuel Jourdan. Zsa.Descriptors: a library for real-time descriptors analysis. 5th Sound and Music Computing Conference, Berlin, Germany, Jul 2008, Berlin, Germany. pp.134-137. hal-01580326 ↩
16. Mikhail Malt, Emmanuel Jourdan. Zsa.Descriptors: a library for real-time descriptors analysis. 5th Sound and Music Computing Conference, Berlin, Germany, Jul 2008, Berlin, Germany. pp.134-137. hal-01580326 ↩
17. Mikhail Malt, Emmanuel Jourdan. Zsa.Descriptors: a library for real-time descriptors analysis. 5th Sound and Music Computing Conference, Berlin, Germany, Jul 2008, Berlin, Germany. pp.134-137. hal-01580326 ↩
18. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
19. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
20. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio*

*Commons project deliverable D*5.2 (2 ↩
21. Pearce, Andy, Saeid Safavi, Tim Brooks, Russell Mason, Wenwu Wang, and Mark Plumbley. “Second Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.6 (2017) ↩
22. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
23. CELLA, CARMINE EMANUELE. "ON PHYSICAL-INSPIRED SYNTHESIS OF SOUNDS." ↩
24. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
25. MacCallum, John, and Aaron Einbond. "Real-time analysis of sensory dissonance." In *International Symposium on Computer Music Modeling and Retrieval*, pp. 203-211. Springer, Berlin, Heidelberg, 2007. ↩
26. Hatano, Shigeko, and Takeo Hashimoto. "Booming index as a measure for evaluating booming sensation." In *Proc. Inter-Noise*, no. 233, pp. 1-6. 2000. ↩
27. Pearce, Andy, Saeid Safavi, Tim Brooks, Russell Mason, Wenwu Wang, and Mark Plumbley. “Second Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.6 (2017) ↩
28. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
29. Brent, William. *Physical and perceptual aspects of percussive timbre*. University of California, San Diego, 2010. ↩
30. Pearce, Andy, Tim Brooks, and Russell Mason. “First Prototype of Timbral Characterisation Tools for Semantically Annotating Non-Musical Content.” *Audio Commons project deliverable D*5.2 (2017). ↩
31. Löllmann, Heinrich, Andreas Brendel, and Walter Kellermann. *Comparative study of single-channel algorithms for blind reverberation time estimation*.

Universitätsbibliothek der RWTH Aachen, 2019. ↩

32. Viator, Landon P., "Finding Music in Chaos: Designing and Composing with Virtual Instruments Inspired by Chaotic Equations" (2020). *LSU Doctoral Dissertations*. 5177. https://digitalcommons.lsu.edu/gradschool_dissertations/5177 ↩

33. Malt, Mikhail, and Emmanuel Jourdan. "Real-time uses of low level sound descriptors as event detection functions using the max/msp zsa. descriptors library." *Proceedings of the 12th Brazilian Smposium on Computer Music* (2009). ↩