Live Coding and Machine Listening

Nick Collins
Durham University, Department of Music
nick.collins@durham.ac.uk

ABSTRACT

Live coding control of machine listening processes, or more radically, machine listening control of live coding, provides an exciting area of crossover between two research frontiers in computer music. This article surveys the state of the art, and reports a number of experimental projects that point to potentially productive further directions.

1. Introduction

Live coding has established itself as a viable and productive method of computer music performance and prototyping, embracing the immedicacy of modern computer programming languages (Collins et al. 2003; Ward et al. 2004; Blackwell and Collins 2005; Brown and Sorensen 2009; Collins 2011; McLean 2011; Magnusson 2014). New avenues in interfacing for programming are being embraced, including tangible computing of unit generator graphs and musical sequences (see for example Mónica Rikić's *buildacode*), natural language processing (e.g., Craig Latta's Quoth) and computer vision (e.g., Nik Hanselmann's *bodyfuck* (2009) brainfuck interface). Performance ventures range from the current vogue for algoraves, showcasing generative and interactive creation of electronic dance music (Collins and McLean 2014) to contemporary dance and live arts.

Complementing existing work, this paper explores the interaction of live coding with machine listening, from listening functionality within existing performance systems towards the use of live audio analysis as programming interface itself. Although there are some existing projects which have utilised machine listening components, this area of computer music has received less attention in live coding, perhaps because of its technical challenges. Machine listening is not a solved problem, and whilst such elements as realtime monophonic pitch tracking, percussive onset detection and timbral feature detection are by now a staple part of computer music environments, more involved polyphonic pitch tracking, auditory scene analysis and live music information retrieval inspired work remains at the cutting edge (Casey et al. 2008; Klapuri and Davy 2006). It is implausible to claim any parity of machine algorithms to human listening at the present time, though many interesting artificial listening resources have been developed.

The paper proceeds after some further review of the confluence of live coding and machine listening within live code performance, to experiments of the author pointing to further possibilities for interaction.

2. Machine listening as resource within a live coding performance system

Musical live coding performance systems have predominantly explored deterministic and probabilistic sequencing of samples and synthesized sound events, driven from an interpreted or jit-compiled textual or graphical computer music language. Audio input, and particularly live analysis of audio input, is much rarer than direct synthesis.

Nonetheless, the work of Dan Stowell on beat boxing analysis (Stowell 2010) and associated MCLD live coding and beat boxing performances is a good example of the co-occurence of machine listening and live coding (Stowell and McLean 2013). Matthew Yee-King has also embraced dynamic live code control of listening systems, notably in improvised collaboration with Finn Peters (Yee-King 2011) with Markovian and evolutionary systems reconfigured on-the-fly to track a saxophonist-flutist. The feedback system between live coder and dancer set up by McLean and Sicchio (2014) utilises both computer vision (tracking the dancer) and machine listening (onset detection on the audio output), perturbing graphical placement of code elements within McLean's wonderfully idiosyncatic Texture language to affect choreographic instruction and audio synthesis code respectively. Such feedback loops of audio and video analysis were anticipated by the now defunct audiovisual collaboration klipp av, who live coded and live remapped audiovisual algorithms simultaneously in SuperCollider and Max/MSP (Collins and Olofsson 2006).

2.1. Machine listening as controller alongside live coding

Machine listening unit generators are available to the live coder, and highly beneficial in creating more involved synthesis patches, as well as feature adaptive processing of live audio input. Examples are provided here within the SuperCollider audio programming language, much used for live coding and well suited to live analysis and synthesis. For instance, an abstract feedback loop can easily be constructed where an audio construct's output is analyzed, the analysis result used to feedback into control of the audio, and parameters of this graph (or the complete graph) made available for live coding manipulation:

```
(
a = {arg feedbackamount = 0.5;

    var feedback, sound, sound2, freq;

    feedback = LocalIn.kr(2);

    //source sound, feedback into first input (frequency)
    sound = Pulse.ar(
                ((1.0-feedbackamount)* (LFNoise0.ar(2).range(10,1000)))
                    +
                (feedbackamount*feedback[0]),
                feedback[1]
            );

    freq = Pitch.kr(sound)[0]; //pitch detection analysis of output sound frequency

    LocalOut.kr([freq,(freq.cpsmidi%12)/12.0]); //send feedback around the loop

    sound

}.play
)

a.set(\feedbackamount, 0.98) //change later
```

Triggering sound processes from analysis of a driving sound process, through the use of onset detection, provides a typical use case (derived rhythms can be productive, for instance via onset detection interpretation of a source sound). The very typing of a live coder can spawn new events, or combinations of voice and typing drive the musical flow:

```
(
a = {arg threshold = 0.5, inputstrength = 0.1;

    var audioinput, onsets, sound;

    audioinput = SoundIn.ar;

    //run onset detector on audioinput via FFT analysis
    onsets = Onsets.kr(FFT(LocalBuf(512),audioinput),threshold);

    //nonlinear oscillator is retriggered by percussive onsets
    sound = WeaklyNonlinear2.ar(inputstrength*audioinput,reset:onsets, freq: Pitch.kr(audioinput)[0] );

    sound

}.play
)

a.set(\threshold, 0.1, \inputstrength,0.01)
```

```
a.set(\threshold, 0.8, \inputstrength,0.7)
```

It is possible to experiment with a much larger number of interesting processes than onset and pitch detection, whether timbral descriptors, or more elaborate sound analysis and resynthesis such as utilising a source separation algorithm to extract the tonal and percussive parts of a signal.

2.2. The Algoravethmic remix system

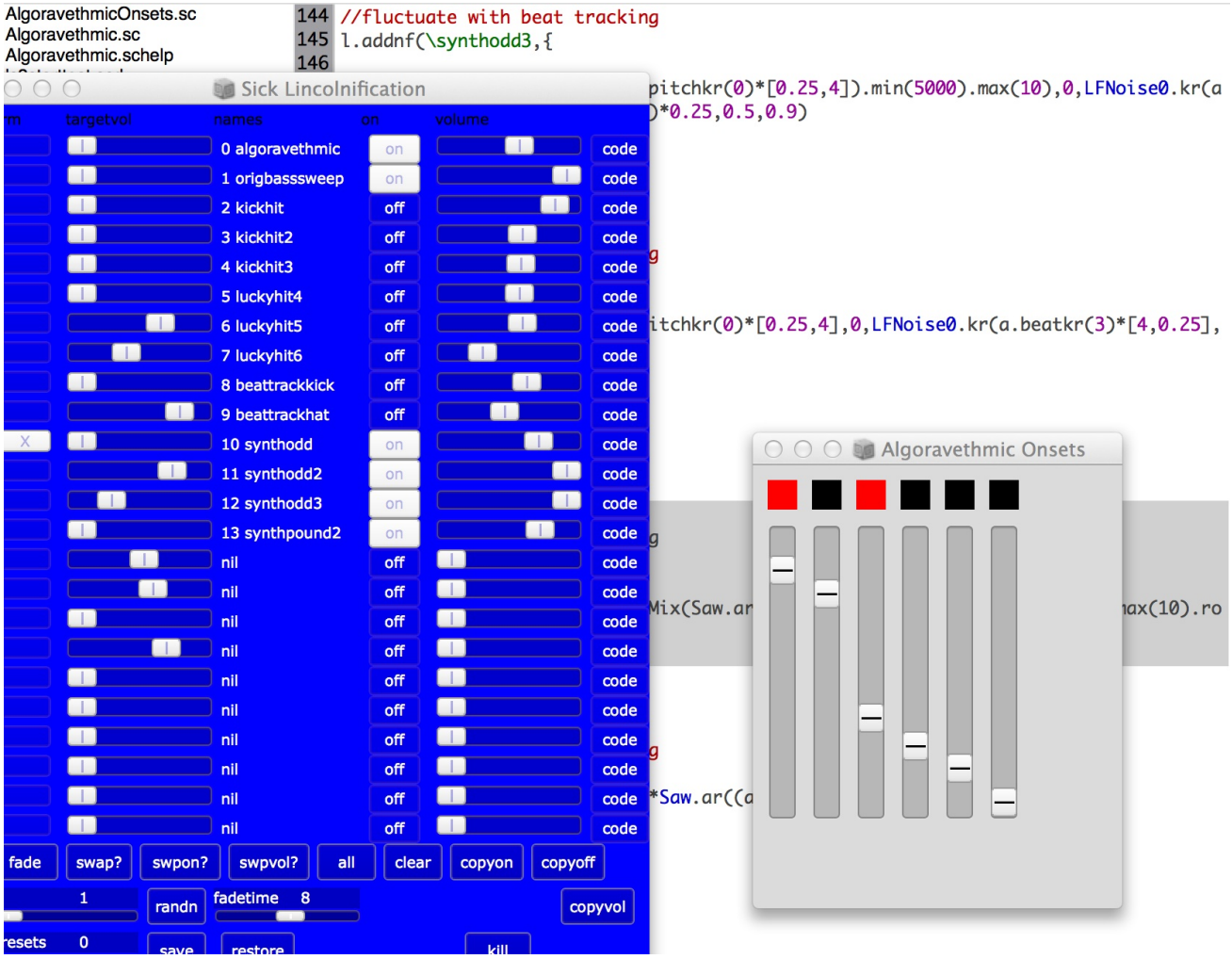

Figure 1: *Algoravethmic session in progress with live coding mixer and onset detection GUI*

An example of a more involved relationship between machine listening algorithms and live coding control is the *Algoravethmic* system (for SuperCollider), which empowers live remixing of an existing source work. The audio from the piece to be remixed is tracked by a variety of feature detectors, including beat tracking, onset detection, timbral and pitch content, and the results fed to control busses for re-use in feature-led synthesis and sound processing . The live coder works on patches exploiting the source-driven data to produce their remix. The live remix process is carried out within a live coding mixer, so that many different coded layers can be interwoven, including processing of the original source audio according to feature-adaptive effects and re-synthesis driven by feature data (Verfaille and Arfib 2001; Park, Li, and Biguenet 2008).

Line by line commands from a typical session might look like:

```
a = Algoravethmic();

(
//fluctuate with beat tracking
```

```
l.addnf(\origbasssweep,{
    BLowPass.ar(
        DelayN.ar(InFeedback.ar(a.busnum,1),0.02,0.02)*5,
        SinOsc.ar(a.beatkr(3)/4).range(200,3000)
        )
    })
)

//l contains the live coding mixer system, to be sent new sounds on the fly

//drum triggering
(
l.addnf(\kickhit,{
    PlayBuf.ar(1,~dubstepkicks[1],trigger:a.onsetkr(4))*3
})
)

l.addnf(\wobblyacid,{Blip.ar(a.pitchkr(0),a.timbrekr(3).range(1,100));})

//original track, delayed to help sync after machine listening delay
l.addnf(\original,{DelayN.ar(InFeedback.ar(a.busnum,1),0.02,0.02)*5})
```

where the beatkr, pitchkr, onsetkr and timbrekr are accessing different feature detector outputs on particular control busses.

Figure 1 illustrates the live coding mixer and a GUI for the bank of onset detectors (with threshold sensitivity controls) used for triggering new percussive events from the track to be remixed. Two of these detectors are adaptations of code from (Collins 2005; Goto 2001) specifically for kick and snare detection, implemented in custom SuperCollider UGens.

3. Machine listening as programming interface

The aforementioned graphical language perturbations of McLean and Sicchio (2014) provide a rare precedent for machine listening actually influencing the code interface itself. In this section two examples consider machine listening control of the instruction state list and memory contents of a running sound synthesis program. This should be understood as the radical adoption of machine listening as the basis for writing computer programs. In the two examples here, the programs so generated act within novel small scale programming languages specialised to musical output, but it is possible to envisage a machine listening frontend as the entry point to a more general purpose programming language.

3.1. TOPLAPapp variations in the web browser

TOPLAPapp is a promotional app for the TOPLAP organisation by this author, originally released as a free and open source iPhone app, and now available within the web browser in a javascript implementation thanks to Web Audio API. Javascript has enabled some immediate experiments extending the core TOPLAPapp instruction synthesis engine, for example, including a convolution reverb.

A version of *TOPLAPapp* with machine listening control has been devised (Figure 2). Machine listening with Web Audio API is a little more involved due to permissions requirements, but still feasible give or take a user having to allow a served web page access to their microphone. Indeed, the core code calls come down to:

```
navigator.getUserMedia({audio:true}, initAudio, function(e) {
            alert('Error getting audio');
            console.log(e);
        });

function initAudio(inputstream) {

audiocontext = new AudioContext();
```

```
audioinput = audiocontext.createMediaStreamSource(inputstream);

node = audiocontext.createScriptProcessor(1024, 1, 1); //mono in, mono out

audioinput.connect(node);

node.onaudioprocess = processAudio; //sample by sample processing function

}
```

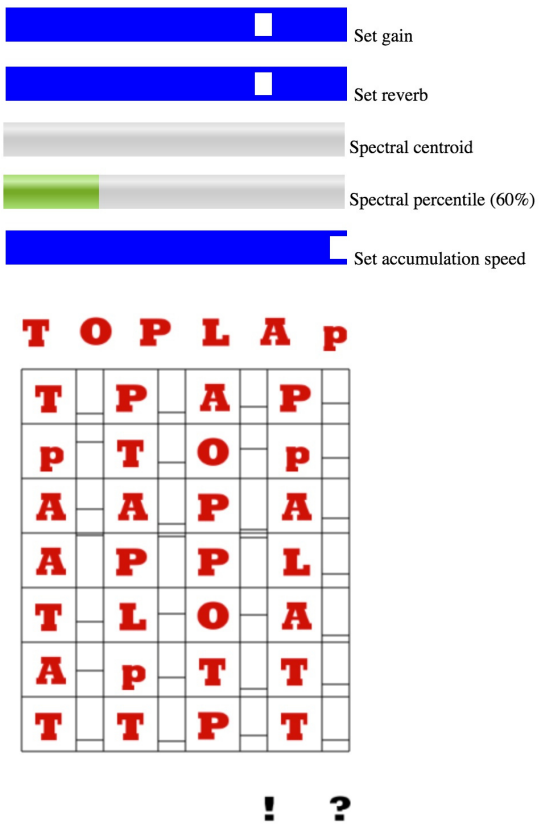

Figure 2: *TOPLAPapp with machine listening front end*

As a prelude to live spectral feature extraction, Nick Jones' fft.js code was co-opted; spectral centroid and 60% spectral energy percentile were then straight forward to calculate. The features were accumulated over multiple (non-overlapping) FFT frames to form feature means, where the duration between means is available as a user set parameter. For each new mean, one state in *TOPLAPapp*'s limited instruction set is updated, and one associated state value, according to:

```
stategrid.states[statepos] = Math.floor(5.9999*spectralcentroidmean);

stategrid.setSlider2(statepos,spectralpercentilemean);

statepos = (statepos +1 )%(stategrid.numunits);
```

Aside from vocal control of the state engine mediated via human ears, the system can also be sent into direct feedback, for example, by the simple expedient of having the built-in mic in a laptop allow feedback from built-in speakers, or via a virtual or physical patch cable. The reader can try out the process here with a recent compatible browser such as Chrome, and is encouraged to make a variety of brightnesses/frequency ranges of noise to reach different states and levels. The app can find itself restricted to only one state if left unstimulated by input, but also has allowed access to novel sonic outputs that the previous drag and drop state construction interface was too slow and deliberative to quickly explore, and the random function didn't tend to find.

3.2. Timbral instructions

A further experiment in machine listening control of a program was conducted in SuperCollider, with instruction states again determined from extracted features (in this case, perceptual loudness and spectral centroid). As the program stepped through the instruction list, it in turn influenced a memory array of floating point numbers, which are themselves used as source data for a monophonic pitched sequence (Figure 3).

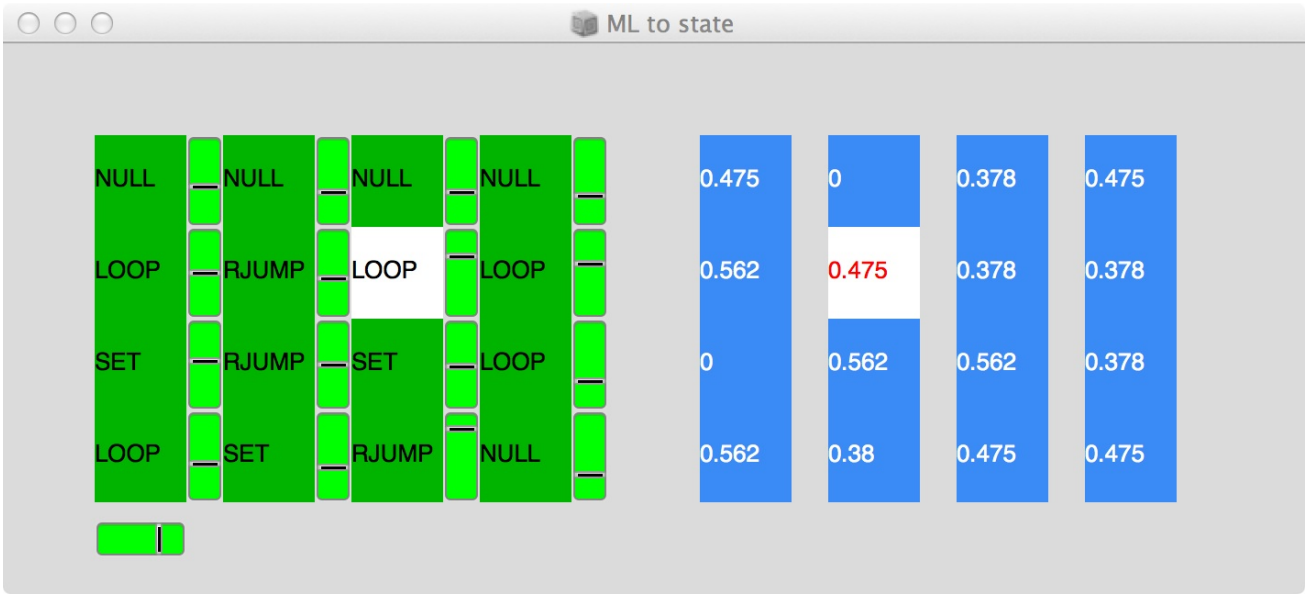

Figure 3: *Machine listening to instruction state engine to memory sequencer demonstration screenshot*

Each instruction slot has a corresponding state value (from 0.0 to 1.0 for easy remapping) to influence the result of the instruction applied to the memory. The (cyclic) instruction list is stepped through (clocked) at the same rate as the memory block (a slider lets a user set the clock rate), though their relative positions can diverge. The list of instructions in pseudocode are:

0) NULL : do nothing
1) JUMP : move in memory block to a position given proportionally by the state value
2) RJUMP : move in memory to a new random position
3) LOOP : loop on current memory location 2 to 8 times
4) SET : set memory contents at current location to state value
5) COPY : store state value for later use
6) PASTE : set memory contents at current location to last stored value
7) RAND : randomise memory contents at current location

The abstract relationship of the musical result to the machine listening is that from the contents of the memory block to the limited instruction set state engine which modulates it. Although there is little sense of reproducible controllability, there is a certain virtue to the unpredictable outputs easily obtained through audio input.

3.3. Further possibilities

The examples elucidated above are by no means exhaustive and are only meant to promote some of the potential interactions of machine listening and live coding.

A possibility not yet seriously exploited in live coding practice is the use of speech recognition to drive a live coding language. Hacking something together is perfectly possible using current generation tools; indeed, the dictation and

speech system preference within recent versions of OS X allows access to speech recognition within any text environment, including SuperCollider's IDE.

Live coding the fundamentals of a machine listening algorithm itself (aside from basic control parameters like a threshold or window size) remains a further tantilizing possibility. Aside from work to construct machine listening algorithms out of existing UGens (for instance, a SuperCollider SynthDef for a custom onset detector), just-in-time compilation facilities such as those in Extempore, Max/MSP's Gen, chuck's Chugins, or the ClangUGen for SuperCollider, provide routes to the live coding of sample by sample analysis DSP.

Are there worthwhile musical purposes, or would effort in coding of such live analysis DSP just be time wasted that could have been spent on details of the audio output? Well, changing the very nature of the computational listening experience underlying a current musical texture would seem an interesting avenue. Live coding offers unconventional takes on computer music performance, and for performance action to engage with an audio input tracking process as a dependent component within a signal chain has potential in this light.

A further option, feasible within the research domain, is an automated machine listening critic. Machine listening and learning over larger audio databases empowers this capability, and a specific corpus of live coding performances seems a feasible target. Deployment in live coding performance remains speculative at present; performance systems with live critics trained from MIR processes over larger audio databases have already been built by this author, though the on stage re-coding of such a critic is a challenge to come.

More unfettered speculation might ponder the future aesthetics of live coded machine listening work. Value may well be placed on individualised languages, perhaps through speech recognition or other sound taxonomic classification enabling coding to head to the territory of a unique constructed language between a given human and machine symbiont. A partnership of human and computer may build such a language as they grow up together; the virtuosity acheivable through such an interface, given substantial investment of learning, would be considerable, though it may only ever be intended for private use. For the form of a performance, the blank slate starting point frequently used in live coding extends straight forwardly from the blank page awaiting text to silence awaiting sound. Continuing the analogy, the full slate performance beginning from pre-composed code can be paralleled by an existing sound, analyzed via machine listening, the sound's features initialising the live code system which must be performed within; this may provide an alternative method of handover from performance to performance or performer to performer. Existing feedback systems incorporating audio, visual and textual material can themselves be extended through additional pathways, allowing generalised functions of multiple audiovisual and textual sources called via iteration or more complicated recursion. This article has said less concerning the multi-performer situation of multiple channels of machine listening information passed around a network, but interesting complexity lies therein. Whilst human listening remains the essential research challenge and touch point, there is also potential for aesthetics of novel machine listening systems divergent from the human ear; ultrasound and infrasound live coding, sensor data/live coding conglomerates, and the future artificially intelligent live coders with alternative auditory capabilities may provide some inspiration to researchers and artists, even if such systems will confound human audiences in further directions!

4. Conclusion

This paper has covered the conflation of live coding and machine listening, exploring different levels of re-configuration for such live analysis processes. There are a number of precedents, and this paper has also explored a number of projects including example code meant to illustrate the potential.

5. References

Blackwell, Alan, and Nick Collins. 2005. "The Programming Language as a Musical Instrument." *Proceedings of PPIG05 (Psychology of Programming Interest Group).*

Brown, Andrew R, and Andrew Sorensen. 2009. "Interacting with Generative Music Through Live Coding." *Contemporary Music Review* 28 (1): 17–29.

Casey, Michael, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. 2008. "Content-Based Music Information Retrieval: Current Directions and Future Challenges." *Proceedings of the IEEE* 96 (4) (April): 668–696.

Collins, Nick. 2005. "DrumTrack: Beat Induction from an Acoustic Drum Kit with Synchronised Scheduling." *Icmc*. Barcelona.

———. 2011. “Live Coding of Consequence.” *Leonardo* 44 (3): 207–211.

Collins, Nick, and Alex McLean. 2014. “Algorave: Live Performance of Algorithmic Electronic Dance Music.” *Proceedings of New Interfaces for Musical Expression*. London.

Collins, Nick, and Fredrik Olofsson. 2006. “Klipp Av: Live Algorithmic Splicing and Audiovisual Event Capture.” *Computer Music Journal* 30 (2): 8–18.

Collins, Nick, Alex McLean, Julian Rohrhuber, and Adrian Ward. 2003. “Live Coding Techniques for Laptop Performance.” *Organised Sound* 8 (3): 321–29.

Goto, Masataka. 2001. “An Audio-Based Real-Time Beat Tracking System for Music with or Without Drum-Sounds.” *Journal of New Music Research* 30 (2): 159–71.

Klapuri, Anssi, and Manuel Davy, ed. 2006. *Signal Processing Methods for Music Transcription*. New York, NY: Springer.

Magnusson, Thor. 2014. “Herding Cats: Observing Live Coding in the Wild.” *Computer Music Journal* 38 (1): 8–16.

McLean, Alex. 2011. “Artist-Programmers and Programming Languages for the Arts.” PhD thesis, Department of Computing, Goldsmiths, University of London.

McLean, Alex, and Kate Sicchio. 2014. “Sound Choreography<> Body Code.” In *Proceedings of the 2nd Conference on Computation, Communcation, Aesthetics and X (XCoAx)*, 355–362.

Park, Tae Hong, Zhiye Li, and Jonathan Biguenet. 2008. “Not Just More FMS: Taking It to the Next Level.” In *Proceedings of the International Computer Music Conference*. Belfast.

Stowell, Dan. 2010. “Making Music Through Real-Time Voice Timbre Analysis: Machine Learning and Timbral Control.” PhD thesis, School of Electronic Engineering; Computer Science, Queen Mary University of London. http://www.mcld.co.uk/thesis/.

Stowell, Dan, and Alex McLean. 2013. “Live Music-Making: a Rich Open Task Requires a Rich Open Interface.” In *Music and Human-Computer Interaction*, 139–152. Springer.

Verfaille, Vincent, and Daniel Arfib. 2001. “A-DAFx: Adaptive Digital Audio Effects.” In *International Conference on Digital Audio Effects (DAFx)*. Limerick.

Ward, Adrian, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Nick Collins, and Amy Alexander. 2004. “Live Algorithm Programming and a Temporary Organisation for Its Promotion.” In *Proceedings of the README Software Art Conference*, 243–261.

Yee-King, Matthew John. 2011. “Automatic Sound Synthesizer Programming: Techniques and Applications.” PhD thesis, University of Sussex.