A TOOLKIT FOR EXPERIMENTATION WITH SIGNAL INTERACTION

Øyvind Brandtsegg

Department of Music,
Norwegian University of Science and Technology
(NTNU)
Trondheim, Norway
`oyvind.brandtsegg@ntnu.no`

ABSTRACT

This paper will describe a toolkit for experimentation with signal interaction techniques, also commonly referred to as *cross adaptive processing*. The technique allows analyzed features of one audio signal to inform the processing of another. Earlier used mainly for mixing and post production purposes, we now want to use it creatively as an intervention in the musical communication between two performers. The idea stems from Stockhausen's use of *intermodulation* in the 1960's, and as such we might also call the updated technique *interprocessing*. Our interest in the technique comes as a natural extension to previous research on *live processing* as an instrumental and performative activity. The automatic control of signal processing routines is related to previous work on adaptive audio effects and automatic mixing. The focus for our investigation and experimentation with the current toolkit will be how this affects the musical communication between performers, and how it changes what they *can and will* play. The program code for the toolkit is available as a github repository[1] under an open source license.

1. INTRODUCTION

The Music Technology section at NTNU Department of Music has researched *live processing* as an instrumental activity for music performance, for example in the ensemble T-EMP (Trondheim Ensemble for Electroacoustic Music Performance). In this ensemble, new modes of improvisation and music making have been explored, utilizing the possibilities inherent in contemporary electroacoustic instrumentation. One particularly interesting aspect of this research is the manner in which live processing affects the communication and interplay between performers. To enhance the focus on possible interventions in this communication, we look into a more direct signal interaction, where analyzed features of one signal are used to control the parameters of processing for another. The term *interprocessing* is derived from Stockhausen's use of *intermodulation*[1] in the 1960's, and is here used to describe any direct signal interaction where features of one signal is allowed to affect the processing of another signal. Where Stockhausen's intermodulation was mainly applied as amplitude modulation, we would like to expand the signal interaction to allow any kind of processing technique, and also include a wide selection of analyzed features from the controlling signals as parametric inputs to the process. With respect to the sonic interaction of two audio signals, this also ties into our earlier work on dynamic convolution [2] and cross convolution techniques [3]. The objective is to find ways of close interaction and sonic merging by enabling musical performative actions of one performer to directly influence the sound of other instruments in the ensemble. As a practical example, say we would let the spectral characteristics of a banjo affect control the immediate filtering of a saxophone, or the noise content of the drums to affect the reverberation of the vocals. Combining analyzed features from several signal allows for a tightly interwoven timbral ensemble interaction (see figure 1).
The toolkit utilize automated control of effect processing parameters, where the automation is based on analyzed features of an audio signal. In this respect it ties closely with the field of adaptive audio effects [4], [5], adaptive modulation, [6], [7] and automatic mixing via cross adaptive techniques [8], [9], [10], [11], [12]. In terms of the instrumental control of the processing it also relates to [13], [14] and [15].
In addition to the signal interaction potential, the toolkit also naturally allows features of a signal to affect its own processing. This can be useful a starting point of experimentation with the analysis signal mappings, and can also be envisioned to yield useful adaptive effects control mappings for studio and post production type effects. The analysis methods and the effect processing methods used in the toolkit are well known from existing DSP literature, it is the configuration as a tool for live and performative experimentation with cross-adaptive effects control that constitute the new or added value of the work. This is intended to allow the mindset of the programmer to be set aside, focusing more on an intuitive and empirical approach for musical experimentation with the techniques. Such experimentation can also lead to a deeper understanding of the analysis techniques involved, and as such may be useful practice for researchers within the field of audio analysis.

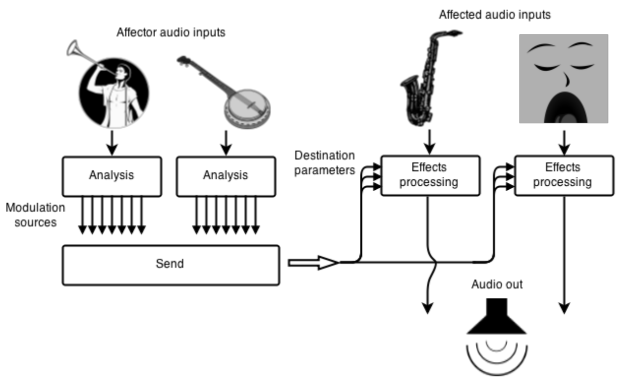

Figure 1: Distribution of analysis signals to several effects

[1]https://github.com/Oeyvind/interprocessing

2. TOOLKIT REQUIREMENTS

The need for a software environment to host and facilitate this kind of audio processing has been suggested by [10] and [16], the latter also providing an example solution within the framework of Max/MSP. For the design of the current toolkit, we have looked specifically at the integration of the cross adaptive techniques into a standardized DAW so that it can easily be combined with other techniques and workflows. The use of open source components has also been preferred.
Since the use of extended signal interaction for live processing touches on musically unfamiliar territory it seems reasonable to make a toolbox for experimentation, and to enable it to be used in such a way that allows the focus of experimentation to remain on the aesthetics and musicality of live processed sound, both in terms of the compositional and the performative. The toolkit is not designed for the general music software end user at this point. This is something that can be built later as a result of experimentation on usability of the different signal interactions. To allow for experimentation the toolkit should be very flexible with regards to configuration and routing of the signals. It should also allow for an intuitive workflow, despite the relatively high number of possible parameter connections/mappings. The toolbox should be easily integrated with other tools, so a method of interfacing with a selection of regular DAWs is strongly preferred. Using a regular DAW as a host program also provides "bread and butter" functionality like a GUI, audio i/o, audio and cpu metering, and audio routing. Methods for analysis of the input signal features has been implemented, as well as routing and mapping of the analysis signals. Integration with an existing set of effects processor implementations is to be preferred in our context, and we have looked at methods to adapt existing effects to allow parametric control by the analysis signals. A methods for integration with standard VST effects has also been implemented. The toolkit is open source and available on several platforms (Linux/Windows/OSX). The audio programming language Csound[2] was selected as the implementation language due to its extensive library of audio processing routines, and also since it fits the requirements of integration with a DAW (via Cabbage[3]) and it is open source and cross platform. For easy integration with a wide selection of DAWs, the toolkit has been implemented with VST wrappers to allow the processors to be used as regular VST plugins.

3. ROUTING

This section will look at possible options for signal routing as a background for the currently chosen model. This relates to the routing of both audio and analysis signals. In the signal interaction, we have an *affected* signal and an *affector* signal. The affected signal is the one where effects processing is being applied, according to analyzed features of the affector signal. Following the conventions of an audio mixer, we would route the signal to be affected into a channel strip and apply an insert effect on this strip. One thing to note here is that the affected signal can be seen as a single source, but the affector signals may come from several audio sources (i.e. using analyzed features of many sources to selectively affect the processing). We can use the term *modulation sources* for the signals resulting from the analysis of the affector signal. Similarly, we can use the term *destination parameters* to refer to the parameters of the effect processing done on the affected signal.

[2]http://www.csounds.com

[3]https://github.com/rorywalsh/cabbage/releases

3.1. Sidechaining

Signal interaction can be found in a conventional audio production signal chain in the form of *sidechaining*. Most commonly, it is used for dynamic processing, for example the genre-typical sidechaining compressor of electronic dance music. Here, the kick drum is used to "duck" the synth pads (or a whole submix), creating a rhythmic pumping effect. Another well known application is ducking of background instruments to give precedence to vocals. A form of sidechaining is also used in de-essers, but in that case the sidechain signal is not an independent audio input but a filtered version of the signal being processed. As a general method for our toolbox, sidechaining has the advantage that it is commonly used and most audio mixers have functionality to provide the appropriate signal routing. The disadvantage is that in general, the sidechain input is a single source. The analysis of the sidechain input will also have to be done in the effects processor for the affected signal, possibly duplicated in other effects processors using the same sidechain source. Due to these limitations, another routing model was needed.

3.2. Dual mono

A variant of the sidechain method would be to create special effects processors with dual mono inputs. One input would be the affected signal and the other the affector signal. This method has similar advantages and drawbacks as the sidechaining method. It could be a viable solution where regular sidechaining is problematic for some reason, but it would require a type of signal routing that many would find counter-intuitive or downright complicated (putting the effect on an aux track, using aux sends and panoramic controls to route the two source signals accordingly). Both the sidechaining and the dual mono configurations may be relevant formats for plugins tailored to a specific set of signal interactions, presumably something that might result from this initial phase of experimentation. Such variants could have the analysis methods inlined in the same plugin, potentially reducing the latency between a feature change in the affector signal and a processing change in the affected signal. The analysis would however, in general be limited to features from one single affector signal. A practical solution would be to implement multichannel VST plugins on multichannel tracks as for example available in Reaper[4], that would however greatly reduce the choice of DAW to use as a host. The issue of duplicated efforts regarding the analysis stage also apply to this model, so it is not the most effective and flexible method for routing and mixing of the control signals. For some traditional spectral interaction techniques, like cross synthesis or cross convolution [3], this routing scheme may still be as good solution, due to the necessity of synchronized frame-by-frame processing.

3.3. Control signal layer

If features of the same affector signal is to be used to control several different affected signals, it might seem reasonable to keep all signal analysis in one place to relieve the system of duplicated efforts. This requires a method of sending analysis signals from one

[4]http://www.reaper.fm/

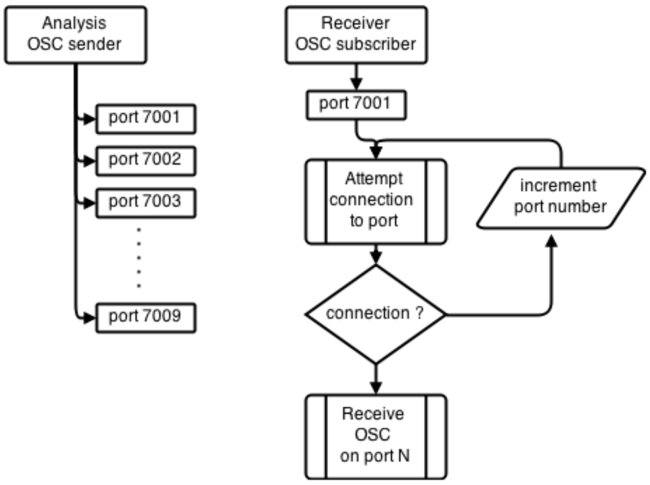

Figure 2: Sending Open Sound Control messages to several network ports on the same computer

process (e.g. audio track in the DAW) to another. Even though this is not implemented as standard in any DAW (with the exeption of using a simple envelope follower as a parametric automation, available in Reaper and other hosts), it can be implemented with relatively widespread tools of interprocess communication (Open Sound Control, ZMQ, named pipes etc). The implementation of a custom routing system as we have done in the toolkit allows for combination of features from several affector signals, in scalable proportions and mappings. With respect to control parameter mapping this is analoguous to the *many to many* mapping described by Hunt et al in [17]. Combining features from several signals in this manner opens up for fine tuned complex interactions between musical signals, but will also require particular care in designing appropriate and musically effective mappings. The provision of a control signal layer to easily experiment with different mappings is assumed to facilitate the design of appropriate settings. To minimize the need for installing third party libraries, OSC was selected as the communication protocol. As we want to distribute the analysis signals (from the affector) to several effects processors (affected signals, also called destinations in the following) we needed to devise a simple form of multicast for the OSC messages. Since the destinations may well reside on the same computer, and the OSC signals are transported via network sockets, we need to open a separate network port for each destination, as only one process can read from a network port at any given time. A somewhat crude solution for this is to send all analysis signals (from all affectors) to a pre-selected series of network ports. Each receiver (destination) process will then scan for an available network port on startup, opening the first available port (figure 2). Additional receivers will scan similarly and open the next available port on startup. In this manner, all analysis signals are available to all receiver plugins, and we can use address filtering to select the signals to be actively used in the effect processing of the affected signal. The method also allows the analysis signals to be available as OSC messages freely routable in the host DAW, and in this manner the toolkit enables cross adaptive control of any VST plugin. The toolkit includes routines for scaling, shaping and mixing of the analysis signals (outlined in section 5), to use these routines in the mapping to any standard VST effect, we provide a special translator plugin (see section 8).

4. SIGNAL ANALYZER PLUGIN

The signal analyzer plugin provides a selection of analysis routines. In addition to the amplitude (rms) analysis, we have used spectrally based analysis methods from the timbre toolbox [18] as well as selection of pitch tracking methods. The timbral analysis methods include *centroid, spread, skewness, kurtosis, flatness, crest and flux*. The centroid represents the spectral center of gravity, while the spread represents the spread of the spectrum around its mean value. The spectral skewness gives a measure of the asymmetry of the spectrum around its mean value, while kurtosis gives a measure of the flatness of the spectrum around its mean value. These parameters (centroid, spread, skewness and kurtosis) are commonly referred to as the first four statistical moments of the spectrum. Further, the spectral flatness measure is obtained by comparing the geometrical mean and the arithmetical mean of the spectrum, it gives an indication of the balance between tonal and noisy components in the sound. Similarly, the spectral crest is obtained by comparing the maximum value and arithmetical mean of the spectrum. Finally, the spectral flux represents the amount of variation of the spectrum over time. The timbre toolbox paper (ibid.) also describes several other analysis measures. Selection of the parameters to be used in our study here has been made in part on the basis of which measures can be calculated in a realtime streaming manner (on a frame by frame basis, without knowing the whole duration or evolution of the sound). An educated guess about the expected usefulness and redundancy of the different measures in our context has also affected the selection.

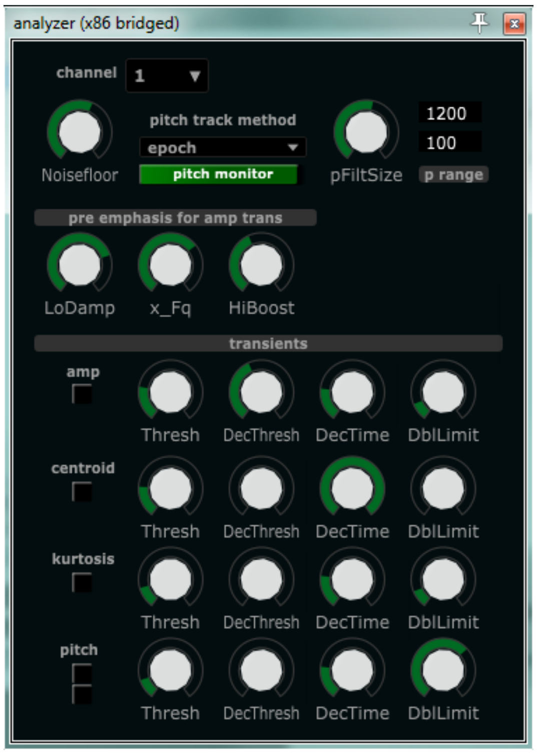

Figure 3: The analyzer plugin GUI

The pitch tracking methods are partly borrowed from the repertoire of Csound, and an additional method has been implemented based on *epoch analysis* [19]. The pitch tracking methods from the Csound repertoire is *ptrack, plltrack, and pitchamdf*. The ptrack method use a STFT method and extracts an estimated pitch for its fundamental frequency, based on an original algorithm by M.Puckette et. al [20]. The plltrack method use a phase-locked loop algorithm based on [21]. The pitchamdf method uses an average magnitude difference function [22]. The active pitch tracking method can be chosen from the GUI of the analyzer plugin (figure 3), and the effective pitch tracking result can be monitored as a sine wave audio output from the analyzer. The pitch monitor signal can be turned on or off via a GUI control. To enable a stable pitch tracking output, median filtering has been applied to the pitch tracker output. The size of this median filter can be adjusted in the GUI (*pitch filter size*)

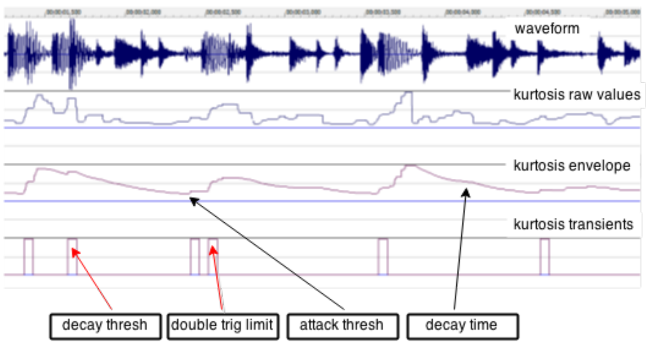

Figure 4: Transient detection, here used on kurtosis.
Arrows in red indicating transients that could be filtered out by raising the parameter values

Transient indicators for amplitude, centroid and spectral kurtosis are extracted using rate of change analysis. The signals are conditioned with an envelope follower filter and mapped to a perceptual scale (e.g. dB for amplitude) before transient detection. The sensitivity of the transient detection can be adjusted with an *attack threshold* parameter. To limit the amount of false trigging, some filtering methods have been implemented. When a transient is detected, the current signal level is recorded, and a *decay threshold* sets the relative negative change needed in the input signal before a new transient is allowed to be registered. The envelope filtering on the signal to be analyzed has an adjustable *decay time* to smooth out fluctuations after a peak in the signal, and this works in tandem with the decay threshold to reduce the amount of false positives. Finally, a timer is used as a secondary means to limit the rate of transients, ensuring that a certain (*double trig limit)* amount of time must pass after a transient has been recorded before another transient is allowed (figure 4). The transient detection algorithm has been developed empirically by the author, inspired by numerous sources over the last few years. It was originally devised for detecting amplitude transients, but here adapted to also work on other kinds of signals. The pitch transients has separate triggers for upwards and downwards pitch change. The aforementioned envelope filtering is then duplicated to create the two different envelopes needed. The detection parameters are the same for upwards and downwards pitch transients, and the detection threshold is in semitones (pitch change needed before issuing a trigger). However, due to inherent weaknesses of the pitch tracking algorithms, the transient detection parameters must be regarded as candidates for empirical adjustment.

5. SCALING, SHAPING, TRANSLATION

5.1. Normalization

Due to the fact that the different analysis track can produce signals within widely varying ranges, normalization of the analysis signal is done before it being sent from the analyzer plugin (figure 5). This allows signals to be interchanged and routed more freely without too many surprises due to out of range values. The analysis response will of course vary significantly on different input sounds with widely differing characteristics, so the said normalization is a trade off aquired by empirical testing. The purpose of the normalization is as far as possible to keep the ranges of the different analyzed features within the same range. Features like skewness, kurtosis and flatness may still vary to an extent that no all-purpose solution has been found. Special treatment is done on the pitch tracking, where two versions of the signal are created. One version is simply normalized by dividing by the max pitch value, another is divided by the effective pitch range and offset with the minimum pitch (so as to create a more full range 0.0-1.0 normalized signal). These signals are called "cps" and "pitch" respectively. The normalized signals will be scaled to the appropriate range for the destination parameters on the receiving side. As an additional convenience, the raw pitchtracking values are available ("cps_raw" parameter) for straightforward routing of pitch to e.g. filter cutoff frequencies. Use of the raw valued parameter selectively bypasses normalization in the analyzer and also bypasses autoscaling (see section 5.4) to the destination parameter range in the receiver plugin.

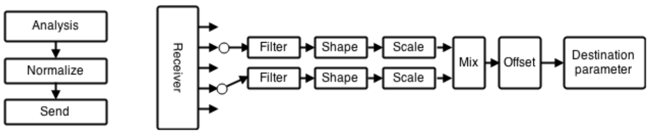

Figure 5: Normalization, signal conditioning, mixing of different modulation sources, offset and route to destination.

5.2. Filters

Some of the analysis tracks contains transient triggers (for amplitude, pitch tracking, centroid and spectral kurtosis). Since these signals are very brief pulses, an envelope generator is triggered in the receiver plugin upon reception of these signals. The rise (attack) and fall (decay) times can be set in the GUI (see figure 6 for an excerpt showing a single destination parameter). For continuous signals, a similar kind of filtering is implemented, making a smoothing filter with separate rise and fall times. The algorithm for the smoothing filter is lifted from Csound's *follow2* opcode, which in turn attributes the algorithm to Jean-Marc Jot. The filter is an exponential moving average, with different coefficients used for rising and falling slopes. It can be described as:

$$Y_N = X_N + (C * (Y_{N-1} - X_N))$$

where

$$C = \begin{cases} 0.001^{(1/(\tau_a * f_s))} & \text{if } (X_N > Y_{N-1}) \\ 0.001^{(1/(\tau_b * f_s))} & \text{otherwise} \end{cases}$$

where τ_a is the rise time and τ_b is the fall time

Envelope generator or filter is selected automatically according to signal type. The filtering is done prior to scaling, as the scaling may invert the sign of the modulation signal and it was assumed it would be more intuitive to control the rise and fall time with respect to the input signal before the (possible) inversion.

5.3. Shaping

After filtering, the signal is shaped (also called *warping* in [23]) by a curve parameter. This is to allow a dynamic and gradual change between log/linear/exponential mappings. The shaping can be described by this algorithm:

$$Y = (1 - \exp(X * curve))/(1 - \exp(curve))$$

Where we use a range of -5.0 to 5.0 for the curve parameter. A curve value of close to zero yields a linear mapping (no shaping), but note that an actual curve value of zero will have to be handled by an exception. A curve value of 1.0 approximates an exponential mapping, while larger values provide increasingly steeper curves. Similarly a curve value of -1.0 approximates a logarithmic mapping, with an increasingly steeper curve for higher negative values.

5.4. Scaling and offset

Each modulator signal can be scaled to set the degree of modulation to the destination parameter. The scaling is done on the receiving side, so that individual scaling can be set for each modulation source to each destination parameter. The modulator range is automatically scaled to the range of the destination parameter via global variables for min and max, so that a normalized modulator signal should be able to use the full range of the destination parameter. In special cases the modulator signal may have a smaller range, depending on the characteristics of the analysis input signal. For these cases, the scaling can be boosted by an additional switch (x10). The offset for each parameter is normally in the same range as the min and max values for the parameter. However, with some routing/mappings, we might want to extend the offset range (e.g. if the mapping of the affector signal constantly makes it go out of range). For this purpose, additional switches has been added to the offset setting, allowing it to extend its range to +/- 1x the original range.

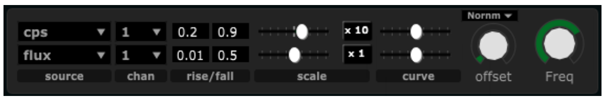

Figure 6: Example of destination parameter GUI

6. EXAMPLE PROCESSORS

Some example processors have been implemented to start working with the toolkit. Little is known as to what type of effects may be musically useful for live cross-adaptive processing, which is also part of the incentive for making a toolkit for experimentation. Some effects has been chosen due to a clearly identifiable or obvious relationship between parameter variations and sonic results. For example *stereo panner, tremolo/AM, and a lowpass filter with distortion*. Other effects has been chosen due to expectations of musical expressiveness, like *time modification* (by means of phase-locked vocoder processing[5]), *stereo delays and reverb*. Yet another type is effects that has the potential of strongly imprinting sonic characteristics (from the modulator) onto the processed sound, for example convolvers and physical models. In this category, a simple waveguide was implemented, with the audio input to the effect being used to excite the physical model. If the fundamental frequency of the waveguide is modulated for example by the pitch of the modulating signal, we get a tight sonic interaction between the two audio inputs. Each of the above effects in and of itself may provide less than exciting musical results, but the combination of several effects modulated by several different characteristics of the modulator signal seems to have the potential for a rich and multidimensional sonic interaction.
The program code for these effects can be found in the github repository[6]. In addition it may be useful to implement a selection of granular delays and transformation effects, flanger/chorus, pitch modulation, spectral panners and other spectral modulations, and dynamic convolver effects, to name a few.

7. ADAPTING EXISTING EFFECTS AND UPDATING THE TOOLKIT

Making a script to automatically modify existing effects implemented in Csound would be handy. However, if such a script should be able to take any implemented effect and modify it to become a signal-interaction-enabled effect we would have to make assumptions about how the parameter control was implemented in the effect to be modified. Rather than making such assumptions, we have made a Python script (*codeUtility.py*) that automatically generates essential include files and also generates the relevant parts of the GUI widget code. The code repository provides a template Csound file for this purpose. To modify an existing effect, one will have to make a list of the control parameters and their associated range. This can be entered as a list into the python script codeUtility.py, and this script will generate the relevant code (when run with python codeUtility.py *effectName*). The GUI code will have to be copied and pasted into the new effect, and the header and score section of the file needs to be modified according to the template effects file. The necessary modifications has been marked with comments in capital letters in the template.csd file. Python writes the GUI code to *effectName*_gui_scratchpad.inc, from where it can be copied into the csd. The GUI caption and plugin id (line 19 in the template.csd) should also be edited to reflect the newly created effect. As the toolkit is still in it's early stages of development, it is highly probable the parameter set of the analysis needs to be updated and expanded. Additional analysis methods should be put in the analyze_audio.inc file. The *readme* file in the repository provides

[5]http://www.csounds.com/manual/html/mincer.html
[6]https://github.com/Oeyvind/interprocessing

additional details as to what components need to be updated to allow the new parameter set to be picked up by the system.

8. CONNECTING TO STANDARD VST EFFECTS

The OSC messages from the analyzer can be used to control standard VST plugins or a host program parameter. Flexible modern DAWs provide mapping options for OSC messages to any control parameter in the host. To aid in mapping and scaling in relation to the control of standard VST plugins, a special OSC translator plugin was devised. This plugin provides the same signal conditioning and mixing as the example effects in the toolkit. The difference is that it will output its destination parameter value via OSC, using the OSC address *parmN* with N being an integer in range 1-8. Selection of network port is available in the plugin GUI. This OSC message can easily be routed to any destination parameter in the host. For more info on setting up OSC control in Reaper there are details online [7], other hosts will have similar methodologies. Some bandwidth issues were encountered when using these OSC message for host automation (the parameter values in the receiving effect would choke and stop moving after a short time). Apparently, the rate of transmission can overflow the host OSC input buffer. A quick attempt was made to reduce the rate of transmission without adding significant latency or jitter; By quantizing values to be sent to the host (to 0.001 steps) and sending only when the value changes, the host seemed capable of handling the automation signal stably for a long time (> 1 hour).

9. CONCLUSIONS AND FURTHER WORK

We have shown a toolkit for experimentation with signal interaction as a technique for adaptive parameter control of audio effect processors. The system includes methods for audio analysis, as well as routing, mapping and scaling of modulation sources. A number of example effects processors has been implemented as proof of concept and as a starting point for further investigation, and an interface to enable control of generic VST plugins (or any host program automatable parameter) has been shown. Some demonstration sounds of possible sonic interactions have been published at the author's Soundcloud page[8]. Initial experimentation with the toolkit has shown it to be a useful and potentially musically valid technique. First impressions also include the potential to use the toolkit to familiarize oneself with the different analysis concepts, as the mapping of analysis tracks directly to changes in the processing of audio gives a very immediate feedback on the features tracked by the different forms of analysis. Further work needs to be done on practical and musical exploration of the technique, and the mapping between sound features and effects controls can be developed further. The *playability* of *the expressional capabilities* [14] of the system is of special interest in this context, also a subject related to instrumental training for this specific music performance system. For extensions to the mapping we may look at higher level sound descriptors and feature combinations discussed in [23] as well as the relationship between musical gestures and actions with regards to playability, possibly touching on machine learning issues as discussed in [24] and [15]. A series of specially designed effect processing methods can also be envisioned, where the experiences from work with the current toolkit can inform the design of effects that has modulation parameters designed for cross-adaptive control, with an emphasis on the expected relation between musical gestures and the processed sound (of another instrument). The incentive for the toolkit has been to provide some means of experimentation, since little is known about the actual musical usefulness of this kind of interprocessing. The results of experimentation with the toolkit may lead to implementation of more targeted effects processors. In this case, the analysis may be implemented as an integral part of the processor, relieving the need for inter-plugin communication and enabling lower latencies and tighter signal interaction. Higher level sound descriptors could be implemented, and a better interface for live performance may be devised as a means of making the tool accessible to a larger base of users of music software for production and performance.

[7]http://www.reaper.fm/sdk/osc/osc.php

[8]https://soundcloud.com/brandtsegg/sets/interprocessing-demo-sounds

10. ACKNOWLEDGMENTS

I would like to acknowledge the suggestions, corrections and valuable discussions from the research environment around music technology at NTNU. In addition, significant input has come from the Csound community and developers.

11. REFERENCES

[1] A. Moritz, "Introduction to hymnen," `http://home.earthlink.net/~almoritz/hymnenintro.htm`, 2003 (accessed May 19, 2015).

[2] Ø. Brandtsegg and S. Saue, "Experiments with dynamic convolution techniques in live performance," *Linux Audio Conference*, 2013.

[3] L.E. Myhre, A.H. Bardoz, S. Saue, Ø. Brandtsegg, and J. Tro, "Cross convolution of live audio signals for musical applications," in *Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research*, 2013, pp. 878–885.

[4] V. Verfaille, U. Zolzer, and D. Arfib, "Adaptive digital audio effects (a-DAFx): a new class of sound transformations," *Audio, Speech and Language Processing, IEEE Transactions on [see also Speech and Audio Processing, IEEE Transactions on]*, vol. 14, no. 5, pp. 1817–1831, 2006.

[5] Vincent Verfaille and Daniel Arfib, "ADAFx: Adaptive digital audio effects," in *Proceedings of the COST-G6 Workshop on Digital Audio Effects (DAFx-01)*, Limerick, Ireland, 2001, pp. 10–4.

[6] Victor Lazzarini, Joseph Timoney, and Thomas Lysaght, "The generation of natural-synthetic spectra by means of adaptive frequency modulation.," *Computer Music Journal*, vol. 32, no. 2, pp. 9–22, 2008.

[7] Victor Lazzarini, Joseph Timoney, Jussi Pekonen, and Vesa Välimäki, "Adaptive phase distortion synthesis," *DAFx 09 proceedings of the 12th International Conference on Digital Audio Effects, Politecnico di Milano, Como Campus, Sept. 1-4, Como, Italy*, pp. 1–8, 2009.

[8] E. Perez-Gonzalez and J. D. Reiss, "Automatic mixing," in *Digital Audio Effects, Second Edition*, U. Zoelzer, Ed., book section 13, pp. 523–550. John Wiley & Sons, Ltd, Chichester, UK, 2011.

[9] J. D. Reiss, "Intelligent systems for mixing multichannel audio," in *17th International Conference on Digital Signal Processing (DSP2011)*, 2011, pp. 1–6.

[10] Enrique Perez-Gonzalez and Joshua Reiss, "Improved control for selective minimization of masking using interchannel dependancy effects," in *Proc. of the 11th Int. Conference on Digital Audio Effects (DAFx-08)*, Sept. 2008.

[11] Brecht De Man and Joshua D. Reiss, "Adaptive control of amplitude distortion effects," in *53rd Conference of the Audio Engineering Society*, January 2014.

[12] Stuart Mansbridge, Saorise Finn, and Joshua D. Reiss, "An Autonomous System for Multitrack Stereo Pan Positioning," in *AES 133rd Convention*, Oct. 2012.

[13] Cornelius Poepel and Roger B. Dannenberg, "Audio signal driven sound synthesis," in *ICMC 2005 International Computer Music Conference*, Barcelona, Spain, September 2005, ICMC, pp. 391–394.

[14] T. Todoroff, "Control of digital audio effects," in *Dafx: Digital Audio Effects*, U. Zoelzer, Ed. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[15] S. Fasciani, *Voice-controlled Interface for Digital Musical Instruments*, Ph.D. thesis, National University of Singapore, 2014.

[16] M. Stabile, "Adapt: A networkable plug-in host for dynamic creation of real-time adaptive digital audio effects," M.S. thesis, University of California, Santa Barbara, 2010.

[17] Andy Hunt, Marcelo Wanderley, and Ross Kirk, "Towards a model for instrumental mapping in expert musical interaction," *Proceedings of the 2000 International Computer Music Conference*, pp. 209–212, 2000.

[18] G. Peeters, B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams, "The timbre toolbox: Extracting acoustic descriptors from musical signals," *Journal of The Acoustical Society Of America*, vol. 130, pp. 2902–2916, 2011.

[19] B Yegnanarayana and S Gangashetty, "Epoch-based analysis of speech signals," *Sadhana*, vol. 36, no. 5, pp. 651–697, 2011.

[20] M. Puckette, T. Apel, and D. Zicarelli, "Real-time audio analysis tools for pd and msp," in *Proceedings of the International Computer Music Conference*, 1998, pp. 109–112.

[21] U. Zolzer, S.V. Sankarababu, and S. Moller, "Pll-based pitch detection and tracking for audio signals," in *Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2012 Eighth International Conference on*, July 2012, pp. 428–431.

[22] M. Ross, H. Shaffer, A. Cohen, R. Freudberg, and H. Manley, "Average magnitude difference function pitch extractor," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 22, no. 5, pp. 353–362, Oct 1974.

[23] Vincent Verfaille, Marcelo M. Wanderley, and Philippe Depalle, "Mapping strategies for gestural and adaptive control of digital audio effects," *Journal of New Music Research*, vol. 35, no. 1, pp. 71–93, 2006.

[24] E. Metois, *Musical Sound Information: MusicalGestures and Embedding Synthesis*, Ph.D. thesis, Massachusetts Institute of Technology, 1997.