A realtime synthesizer controlled by singing and beatboxing

Bart BROUNS
studio magnetophon
Biesenwal 3
Maastricht, Netherlands, 6211AD
bart@magnetophon.nl

Abstract

A realtime instrument was implemented in pd-extended [1] that translates singing, beatboxing, and both simultaneous, to a wide range of melodic and percussive synthetic sounds. Analysis, synthesis and processing algorithms where selected and integrated with a focus on expressive and reliable voice-control, with an intuitive correlation between the input and output.

For the parameters that are not voice controlled, an XY-pad interface was created allowing the user to interpolate between four settings.

Keywords

Live-performance, Analysis, Synthesis, Processing, Interface design.

1 Introduction

1.1 Related work

There is quite some literature on the combination of analysis and synthesis, even in the context of realtime singing and beatboxing, but often it focuses on either

- mapping any input sound to any synthesizer[5]
- closely emulating the input sound and then applying transformations [17]
- strictly categorizing input sounds and triggering corresponding output sounds

1.2 Aim

In this work, realtime analysis and synthesis algorithms where selected and integrated with a focus on expressive, intuitive and reliable control by simultaneous singing and beatboxing. Within those bounds, an effort was made to make the widest possible range of timbres available.

1.3 Overview

This paper is organized into the categories analysis, user interface and synthesis, roughly following the structure of the patch. The Beats section is discussed separately as it combines additional analysis and synthesis in the same sub-patch.

1.3.1 Analysis

Five kinds of parameter are analyzed for use by the melodic synthesizers:

- pitch: the fundamental frequency
- timbre: the levels of 16 audio bands[1]
- formant: the strongest spectral peak excluding the fundamental
- onsets: start-point of a new sound
- voiced/unvoiced

1.3.2 User interface

Where it made sense, the analysis was linked to the synthesis parameters in a user-adjustable way. For example, in the basic subtractive synthesizer discussed in section 5.8, the cutoff frequency of the low-pass filter is linked to the detected formant frequency; the user can control how much the formant modulates the cutoff.

Both these 'mapping modifiers' and the synthesis parameters that aren't mapped to the analysis where made available in a graphical user interface. Four instances of this interface where linked to an XY-pad, each corresponding to one corner, allowing the user to interpolate between four settings. (see Figure 1).

[1]Actually there are two of these sets, and the level of other parts of the spectrum is also used.

There are separate preset systems for the four individual interfaces, defining one sound, and one for the overarching user interface, combining four sounds. The time it takes to move between points on the XY-pad is user-adjustable.

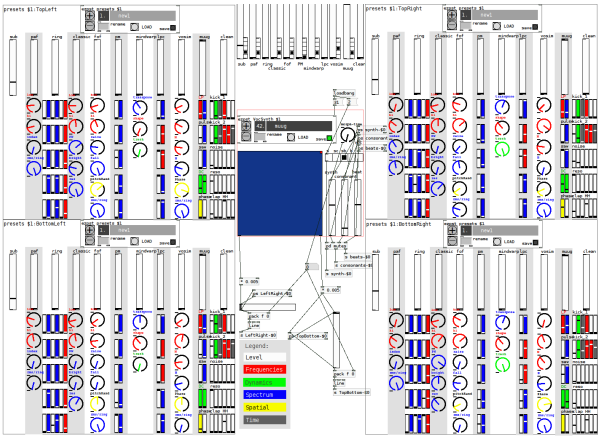

Figure 1: Graphical user interface

1.3.3 Synthesis

There are twelve parallel synthesizers, processors and combinations. (See Table 1)

Synthesizer	Description	Sect.
Clean	dynamics processor	
Sub	sine-wave	5.1
CZ Ring-modulator	Pitch-shifter with timbre-modification implemented as a ring-modulator	5.2
Classic vocoder	variation on an analog vocoder	5.3
PAF vocoder	vocoder with formant oscillators in place of output-filters	5.4
Vosim vocoder	variation on PAF	5.5
FOF vocoder	variation on PAF	5.6
PM vocoder	variation on PAF	5.7
muug	subtractive synthesizer	5.8
lpc	linear predictive coding	5.9
mindwarp	an extended pitch-shifter	5.10
beats	a kick, snare and hi-hat	6

Table 1: Synthesizers

All of these, except the muug and the beats, can produce intelligible speech. They are also capable of totally warping the input sound, while retaining an intuitive correlation between the input and output. They also all remain an octave relation to the input pitch at all times, again twith the exception of the beats.

In the next sections I will explain the global workings of each block.

I will skip over the nitty-gritty like how parameters are scaled, sometimes in quite elaborate ways. Still, getting such details right is crucial to the intuitive functioning of this instrument.

2 Analysis

This section discusses the analysis used by the melodic synthesizers. The additonal analysis for the beats is discussed in section 6.

2.1 Pitch-tracker

The Pitch-tracker is based on helmholtz~ [2]. Even though this is by far the best pitch-tracker I could find, I think I was able to improve it somewhat. The author of helmholtz~ is currently investigating if it makes sense to incorporate my algorithm into the extension.

2.1.1 Issues

helmholtz~ has a pitch output and an output that indicates the extent to which the signal is periodic; called fidelity. This gets used by a built in gate: you set a threshold value for fidelity, below which the pitch stops outputting new values. This prevents wild pitch fluctuations when silent or making un-pitched noises, but it has issues: With a high threshold it perfectly eliminates the warble caused by un-pitched noises or silence, but it misses semi-pitched sounds like singing below your comfortable range. With a low threshold it tracks below-range singing though with occasional warbling, but warbles a lot with un-pitched noises.

2.1.2 Solutions

To remedy this I made multiple fidelity ranges and kept a moving average of good pitch values. Pitches with medium fidelity are compared with this average and only outputted if close to it.

This has the wanted effect of ignoring false pitch measurements, but the side effect of limiting the speed at which the detected pitch can change. The number of samples to average now gets a similar influence as the fidelity threshold had before: too low and most proposed pitches will be let trough, resulting in a return of the warbling. Too high and it sometimes misses sudden pitch changes.

By varying the number of pitches to average, we can give the tracker a different sensitivity in different situations. Specifically, these are the actions taken for each of the four ranges of pitch fidelity, and the effect they have:

- very low fidelity: the pitch is ignored and the number of samples to average is set to medium low, so that the tracker is quite flexible right after a silence or un-pitched noise.
- low fidelity: the pitch is compared to the average and if close to it, it gets treated as a OK fidelity pitch. Otherwise it is ignored.
- OK fidelity: the pitch is outputted, the number of values to average is increased by one, and a new average is calculated, including the new pitch. After about 0.2 seconds the number of samples to average has risen enough for the tracker to outputs steady notes, even with difficult inputs
- high fidelity: This is the most common condition; it happens as soon as you sing a clear pitch. It outputs the pitch and sets the number of values to average to low, so that the tracker is sensitive to quick changes in pitch.

This sub-patch also outputs pitch fidelity, as it is a very telling audio feature in itself [5], that gets used in the onset detector, the formant detector and the voiced/unvoiced detector amongst others.

2.2 Onset Detector

The onset detector is based on bark~/.[4]

2.2.1 Issues

It has a threshold value to determine the sensitivity, but again, it is impossible to find a setting that detects all wanted transients, but none of the unwanted ones.

2.2.2 Solution

The solution is to vary the threshold with pitch fidelity; now the onset detector is medium sensitive when singing clear pitches, and very sensitive when making un or semi pitched sounds.

This is a joy to use: no false positives, but in between notes even the tiniest lip or tongue smack gets detected. When you move between different vocal sounds, often you automatically make one or two of these tiny smacks, and as they are a side-product of your mouth moving about rhythmically, they fit perfectly with the explicit notes of the rhythm. These grace-notes are translated to soft high-hats (section 6.6), so they end up sounding as ghost-notes like a drummer would play them.

This hyper sensitive setting works great in a silent location, even with the monitors turned up, but needs to be adjusted for live use, as it picks up every little background noise when the performer is silent.

3 Timbre analyzer

The timbre analyzer is a variation on the analysis part of an analog vocoder: a bank of bandpass filters followed by envelope followers, in this case 16 pairs. It differs in that the frequency of the filters gets modulated by the pitch, so that the first filter-envelope pair is always tracking the fundamental and each next pair one of the partials.

There are two versions of this running together:

- one that tracks up to the 32th partial, which translates to an upper boundary of between 2 and 10 kHz with my voice.
- The other goes from slightly below the fundamental to the 128th partial; covering the whole spectrum of the input signal

and at the upper boundary often even more.[2]

The wide-band analyzer is at the moment only used by the PM synthesizer.

To make the filter frequencies between the lowest and the highest logarithmically spaced, I made an abstraction called VocoderFreqs. I will explain it when discussing the first synthesizer that uses it too, as it makes more sense in that context.

The signal to be analyzed is compressed, expanded and limited by the wonderful qompander~ [3]

4 Formant Tracker

The formant tracker is based on specCentroid~ [4]; an external that reports the frequency associated with
spectral center of mass.

I'm not sure what that means but perceptually it is close to what you would expect a formant tracker to do, except it tends to get confused by low frequencies and it tends to warble.

The first was solved by high-passing the signal to be analyzed at the frequency of the fundamental, and the warbling with a similar approach as in the pitch tracker discussed earlier.

Specifically, rogue values for the formant are weeded out with the following steps:

- Only get the specCentroid when pitch fidelity is high. Simply because I noticed specCentroid~ outputs mostly garbage at moments when pitch fidelity is low.
- Ignore the specCentroid if it is lower than the current pitch. Shouldn't happen AFAIK, but does...
- Count the number of pitch estimations since the last onset. (as detected by the OnsetDetector discussed earlier)
- If this is low, so at the very start of a note, directly feed the specCentroid to the formant output.
- After the very start of the note, compare each proposed value with the moving average, and only let it trough if it is close.

This leaves us with a stream of values that reacts fast at the very beginning of notes, and a bit more steady after that. It has all of the clearly wrong values taken out but is still a bit warbly. Therefore it is smoothed with another moving average, making it react slower when the formant is low and faster when the formant is high, and also slower when there is a lot of variation in the formant estimates and vice versa.

5 Synthesizers

Where possible, all synthesizers are driven by a single phaser~, by using lookup tables or lookup functions as oscillators.

This has three advantages:

- Lower CPU overhead.
- Since all 'oscillators' are in phase they mesh better. In other words: they sound more like one oscillator, because they are!
- When I want them to sound like multiple oscillators, I can vary the phase of each relative to the master oscillator.

With the right lookup table or function you can emulate any oscillator in any pitch, as long as it is an integer number of octaves above the master oscillator's pitch. Therefore the master oscillator always runs at the lowest pitch that can be used anywhere in the synthesizer: two octaves down from the detected pitch.

5.1 Sub synthesizer

This sub-patch contains the master oscillator discussed above and a sine wave derived from it. This sine is normally used in bass synthesizers , set to an octave below the input frequency. Therefore it follows the volume of the bass frequencies in the input-signal, ducked by the kick drum to preserve headroom and bass clarity.

5.2 CZ Ring-modulator

This is pd example patch E03 with CZ-oscillators [6] as the modulator.

The example patch demonstrates using ring modulation to alias a sound up or down by one or more octaves. Since the pitches of both signals are integer multiples of each other, no dissonant partials are created, so it sounds quite different

[2]Contrary to what one would expect, the filters in the wide-band analyzer have a much higher q than the ones in the other analyzer. I am not sure why, but the synthesizer that's driven by it sounds better that way.

from what most people associate with ring-modulation. This trick is also used in some of the other synthesizers.

When you do this with a sine as the modulator, as in the example patch, the result is a reasonably clean pitch-shifted version of the input signal. When you do it with other oscillators, the result has almost the same intelligibility as the input, but with the character of the oscillator.

For this reason the CZ-oscillators where chosen, as most of them can steplessly change from sine-wave to something else, in an interesting way. They are an emulations of the Casio CZ oscillators.

One of them features a resonant peak with controllable frequency. This was linked to the detected formant frequency, scaled by a user controllable amount.

There are five sets of each oscillator; one for each octave from -2 to +2. When morphing between settings, these oscillators are first mixed, and only then ring-modulated, so when morphing between octaves it sounds much more interesting than a normal audio cross-fade would.

These five sets of oscillators where also made for the other synthesizers, where it was not too expensive in terms of CPU-usage, and sounded interesting when morphed. For most synthesizers this was too expensive and/or the morph could only be implemented as a cross-fade between the oscillators; thus sounding uninteresting.

5.3 Classic vocoder

This is a slight variation on a normal vocoder.

These are the differences:

- It uses the analyzer discussed in section 3
- Obviously the oscillator of the carrier follows the pitch of the modulator.
- The top and bottom frequencies of the bandpass-filters in the carrier are determined by multiplying the pitch by a user control for each. The frequencies in between are automatically set to logarithmic spacing.

The result is that you can move the formants of your speech up and down, and make them span a wider or narrower range. All with just two intuitive knobs, one for the top of the range and one for the bottom.

The algorithm to calculate the filter frequencies is the one also used in the analyzer:

n = [1-16] (band number)
B= bottom frequency knob
T=top frequency knob
x = (B/T) pow (1/15)
frequency(x)=B*x pow (n-1)

The sound-source is a band-limited pulse oscillator, slaved to the master oscillator.

5.4 PAF-vocoder

This is like a normal vocoder, but instead of a single sound source trough multiple bandpass-filters it uses an oscillator that has a bandpass-like quality of itself, in place of each bandpass. The oscillators are made with the PAF (phase-aligned formant) synthesis algorithm (patented 1993 by IRCAM). Example patch F13 from pd-extended was adapted to be slaved to the master-oscillator.

As with the classic vocoder discussed in section 5.3, you can set the top and bottom frequencies of the formants.

In a similar vain as the CZ-ringmod synthesizer, each oscillator can be ring-modulated, but this time with the audio from the corresponding bandpass-filter in the timbre-analyzer. To my ears this sounds much more pleasing than ring-modulating them with the full bandwidth input signal.

A knob was made to cross-fade between:

- The volume of each oscillator being controlled by the envelope of the corresponding analyzer band.
- Each oscillator oscillator being ring modulated as discussed above. This also has the side effect of controlling the volume.

5.5 Vosim-vocoder

This is the same idea as in PAF-vocoder, but with vosim oscillators [7] instead of PAF, again slaved to the master oscillator.

The oscillators are routed to the output alternating between left and right for each one, giving a nice stereo spread. This spread can be widened by the phase knob, which alters the phase of the oscillators, with the ones on the left

channel getting a positive phase offset, and the ones on the right a negative one.

Finally the oscillators can be ring-modulated with the input.

5.6 FOF -vocoder

This is another variation on the above theme, with fofsynth~ from Ggee[8] as oscillators. Since this external is an actual oscillator and not a lookup function or table, it is possible to detune each individually. Each oscillator has it's own random generator, detuning it by an amount controlled by a global fof detune parameter. To maximize the dramatic effect of detuning, there are two complete synthesizers, one each for left and right.

As with the PAF vocoder, each oscillator can be ring-modulated with the audio from the corresponding bandpass-filter in the timbre-analyzer.

5.7 PM vocoder

This is a cross between a vocoder and an additive synthesizer, with Phase modulation oscillators as the partials.

Each partial consists of one carrier and five modulators, all slaved to the master oscillator. The carriers have pitches that are a consonant multiple of the input fundamental, spaced over the audio band, so with the phase modulation turned down this is a 16 partial voice controlled additive synthesizer. This in contrast with the other variations on a vocoder where all the oscillators have the same pitch but a resonant frequency that is different for each band.

The five modulators have fixed pitches, relative to the analyzed pitch: each octave from -2 to +2. As with the other synthesizers that have such arrays, this was done to make interesting morphs between perceived octaves possible.

Each modulator also has a variable waveshaper, drastically widening the available timbre space.

5.8 Muug

This is a subtractive synthesizer, that follows the input-signal's pitch, volume, and cutoff (trough the formant tracker). These are scalable by the user, the pitch in octave steps.

It has a PWM oscillator [1], a triangle-to-saw oscillator [9] and a muug~ [10] filter. The oscillators are band-limited and the filter emulates both the non-linearities and self-oscillation of a Moog ladder filter.

To further exercise these non-linearities, I've added adjustable DC offset, both static and proportional to the level of input-signal.

The combination of DC and lots of gain on the input-signal has great sonic potential, but can sometimes choke the filter. To avoid this:

- The static and dynamic offsets are of opposite polarity. So that they partially cancel out when the input-signal has a high level.
- The gain is automatically turned down when the level of the input-signal is higher than the output-level, as that indicates a choked filter. [3]

This synthesizer can go from mono to stereo, under control of the input-level. To this end there are two of these synthesizers: left and right.

Since they share the same parameter values and the oscillators are slaved to the master, they output identical sound; thus mono.

Stereo width is created in two ways:

- By making the DC offsets of opposite polarity for left and right. This is still mono when you are within the linear range of the filter, but gets wider the more distorted it is
- By making a phase offset in the oscillators, the amount varied with input signal level.

This is the only synthesizer in the set that can not produce ineligible speech, though one could easily make it do so by adding in ring modulation as described in section 5.2.

This sub-patch is upsampled to twice the samplerate. Oddly, upsampling more increased digital artifacts again.

---

[3] This works OK but has a tendency to oscillate the gain when it's pushed hard. At the moment I'm using a very long release time to make the oscillating less obvious but I'm looking into alternatives. (fuzzy logic maybe?)

5.9 LPC

Adapted from the lpc~ [11] cross synthesis help file. It has the same "5 oscillators in 5 octaves" array as some of the other synthesizers.

In this case using a lookup formula. Unfortunately I can't trace back the author of it. If anyone can tell me, I will update this document.

5.10 Mindwarp

A copy of the mindwarp~ [12] help file. So not even a synthesizer! Sounds great though...

6 Beats

Beats consist of kick, snare and hi-hat, implemented as:

- pitched [13] and un-pitched kick [14]
- snare noise [14], snare resonance [15] and clap[16]
- hi-hat [14]

They are triggered when a corresponding sound is made at the input. In most other implementations, this is done by classifying input sounds as either one drum or the other, so that samples can be triggered. Here, a more fluid approach is chosen, where the drums vary in level, pitch, length, timbre and stereo-width in parallel to what the performer is doing.

In broad lines, the effect is that hi-hats are played on every onset, kicks and snares have a cross-fade area, and similarly, snares plus hi-hats fade into hi-hats only. It was quite convoluted to make this work reliably when the beat-boxing is done simultaneously with singing a melody, and at the same time have it react expressively and be sensitive to details.

Arguably the most reliable way to categorize audio in real time is using BFCCs. [4][5]

timbreID [4] already has a help file demonstrating this.[4] While it works quite well, it has two downsides:

- It requires training the detector with example sounds
- It only outputs two usable parameters: the index of the nearest match and the distance to it.
- It takes separate training to 'teach' it that, for example, a snare drum on the input should give the same output as a snare drum 'sung' together with a bass note.

Therefore I made a system where all drums get triggered by each onset, but their level, decay time and other parameters are modulated by various audio features. As there are quite a lot of feature to parameter mappings, the analysis and synthesis was not (yet) separated into sub-patches.

I will explain the mappings in the following subsections, starting with the snare noise, as some of the others depend on it in various ways.

6.1 Snare noise

A snare is played when there is:

- no kick
- no hi-hat
- no voice (except during the start)

These five features are used to decide this[5]:

- overall level
- level of the mid frequencies
- level of the high frequencies
- Spectral centroid [4]
- Spectral rolloff [4]

The first three are measured continuously, and the last two only at each onset.

Snares are usually vocalized ending with just high frequencies. The level of the high frequencies determines the level of the snare, but not that much level is needed to pin it at maximum. This allows some control over the level of the snare, but mostly controls when it stops. In other words: you can control the length of the snare by extending the high frequency noise and then abruptly stopping it.[6]

The overall input level turns down the snare again, but with a slow attack, so when you sing a

[4] timbreID-examples/drum-kit.pd

[5] These where chosen after personal testing, later confirmed here.

[6] To enhance the effect, this parameter also controls the level of an actual reverb on all the snare components. The reverb comes from [15].

note and make a snare sound simultaneously, the snare is still synthesized, but cuts out fairly quickly.

The level of the mid frequencies is mapped to the decay time, so that soft to medium sounds get an almost zero to medium decay time. This is predominantly what gives the impression that the snare noise level follows the input level.

Anything louder than medium quickly gets very long decays, in effect leaving control of the length to the first two parameters.

The above three parameter mappings may sound a bit counter-intuitive when written down, but they where actually chosen for the intuitive control they offer.

Spectral rolloff is the frequency below which a certain amount of total spectral energy is concentrated. It is used to turn down the snare when there is more low frequency energy then high, in other words, when the onset is a kick.[7]

Spectral centroid, discussed in section 4, is used to determine when there is mostly high frequency energy in the onset, turning down the snare.

The noise is generated by waveshaping a feedback loop. [14] This algorithm was chosen because you can simultaneously change the volume, spectrum and density in a way not too different from how a real snare changes in spectrum and density when changing volume. This is done by changing the gain inside the feedback loop. It is this gain and it's envelope that I referred to in the above passage.

The snare noise starts out mono and gradually gets wider. To this end, there are two of the above generators, left and right, plus a third, normal noise source. A tiny but increasing amount of normal noise is added into the feedback loop, in opposite polarity for left and right.

[7]This on the other hand, sounds perfectly intuitive on paper but isn't in practice: you need to take care not to make your snares too bassy.

6.2 Snare resonance

This is made with two identical sets of resonating bandpass-filters; left and right. They are excited by a short burst of the noise discussed above.

Therefore it's level is dependent on the level of the first few milliseconds of noise. To better match the perceived input and output-level a few more mappings where needed:

First an additional level control for the excitation noise. Since this lasts only very short time, the measurement also needed to be done ultra quickly: After experimenting with envelope followers with different analysis window sizes, I found I got the best results by combining two different ones. This measurement was also mapped to the decay time of the excitation pulse and to the q of the filters, so that soft inputs create soft, short, noisy outputs, and the output gets louder, longer and more tone-like as the input-level rises, just like a real snare.

The pitch of the resonances drop a bit as the snare progresses. The louder the input the higher they start and the longer the drop takes, again emulating a real snare.[8]

6.3 Clap

This is an emulation of an 808 clap, again made with a bandpass-filter and using the noise from section 6.1 as the exciter. The frequency of the filter is set by the level measurement from the previous section.

6.4 Un-pitched kick

This follows the level of the low frequencies, but is turned down as the snare gets louder. To get out of the way of the bass, it gets shorter as the input gets more melodic, as defined by the pitch fidelity with a long release time.

6.5 Pitched kick

This is a falling sine wave, like a 909 kick. It starts at a user set pitch and falls to the analyzer pitch. This allows you to sing melodies with bass

[8] Or at least emulating a 909 emulating a real snare...

drums. The level is determined similarly to the section above. It features a distortion followed by a muug~ low pass filter, which can also be driven into distortion itself.

6.6 Hi-hat

Based on a combination of PM and ring-modulation. It follows the level of the very high frequencies, and it's decay is determined by a combination of timbreID [4] analyzers.

7 Mixer

Apart from mixing, this sub-patch turns down synthesizers on unvoiced signals such as fricatives in speech and beatboxed elements. What happens on those moments is determined by a user set parameter. Either the beats take over, or noise with the spectrum of the input-signal is faded in, emulating fricatives.

8 Morpher

This sub-patch contains the user interface and the interpolation between settings.

To maximize ease of use, each knob and fader is assigned to one of four categories, and color coded accordingly. (See Table 2)

category	description	color
level	volume knobs	white
frequencies	filter frequencies etc.	red
dynamics	parameters that change with input-level	green
spectrum	anything else that changes the spectrum	blue
spatial	parameters that change the stereo image	yellow
time	envelope times etc.	gray

Table 2: Color coding

9 Conclusion

An instrument was discussed that allows the user to quickly create electronic music, sounding like a finished product, using just a voice. It's useful in both live and studio settings.

9.1 Future work

Apart from the obvious enhancements like more synthesizers, more beats and more/better mappings, there are a couple of directions in which I would like to develop this instrument.

- Making it possible to save intermediate settings on the XY-pad to a new preset.
- Allowing additional interpolation points on the XY-pad, appart from the current four, possibly using the morphOSC library[9].
- Changing out the XY-pad for a 3D and/or gesture based controller[10].
- Using fuzzy logic to better map analysis to synthesis[11].
- Automatically setting the parameters of the muted synthesizers to blend with the current sound, for smoother morphing [5].

The main direction I would like to take this in is integrating it with a looper. A looper is a device that records audio and loops it on the fly. By having loops with separate tracks for source audio, processed audio, and synthesizer parameters, one can do things like:

- manipulate synth parameters after you've made a loop[12]
- these manipulations can also be recorded
    - either overwriting the old values
    - or extending the length of the loop
- change one loop while recording another

The next step after that is an algorithmic looper: By feeding the individual pitches of the loops to chord recognition software[13], and linking that to

[9] https://github.com/LiamOSullivan/MorphOSC

[10] Onix Ashanti , Imogen Heap

[11] Www.rodrigocadiz.com/publications/icmc2007.pdf and Controlling a sound synthesizer using timbral attributes

[12] Beardyman Talks the BeardyTron 5000

[13] I already made a patch that takes melodies in the form of midi notes, turns them into block chords, records those and sees if, and where the pattern loops.

algorithmic music software[14], one can create an advanced voice controlled auto-accompaniment.

This would also open up possibilities such as intelligent pitch-shifting. Ultimately, advanced workflows such as the following could be possible:

1) start recording a one bar loop

2) at the end of the loop, press "transpose whole loop" mode

3) simultaneously you play the note you want the transposed version to start with

4) you hear the pitch you are playing, but with the sound of the loop you just recorded

5) switch to another mode, for example to "bypass" so you can solo over what you just played without it being recorded or anything

6) the loop keeps playing transposed by the same amount

7) each time you want the loop to modulate, you press "transpose whole loop" and play a note

8) at the end of the chord progression you press a button to indicate this

The software waits until the end of the loop, and then starts playing back the whole chord progression.

10 Acknowledgements

While making this patch I've often been amazed at the amount and quality of the work others have done and shared. I will spare you the overly dramatic, but heart felt monologue I had written before, and just say:
Thank you.

References

[1]pd-extended graphical programing language

[2]helmholtz~ pitch tracker pd-external

[3]qompander~ compressor expander limiter pd-external

[4]TimbreID library of pd-externals for feature extraction

[5]Stowell 2010: Making music through real-time voice timbre analysis: machine learning and timbral control

[6]CZ-oscillators abstraction

[7]vosim~ oscillator abstraction

[8]Ggee library containing fofsynth~

[9]dotmmb library containing blosc~

[10]muug~ moog ladder filter emulation

[11]ekext library containing lpc~

[12] FFTease library containing mindwarp~

[13] DIY2 library containing the basis for the pitched kick

[14] fairly-efficient-analog-drums kick, snare and hi-hat abstractions

[15] Tonal and Atonal patch containing the snare resonance and the reverb

[16] pdmtl library containing the 808 clap

[17] Singing voice analysis/synthesis

[14]Musical Midi Acompaniment