The Taxidermy of Negative Space

For T-Stick, live electronics, fixed media, dancer, video projection

Erich Barganier

2020

Instrumentation

Tenor T-Stick

Laptop

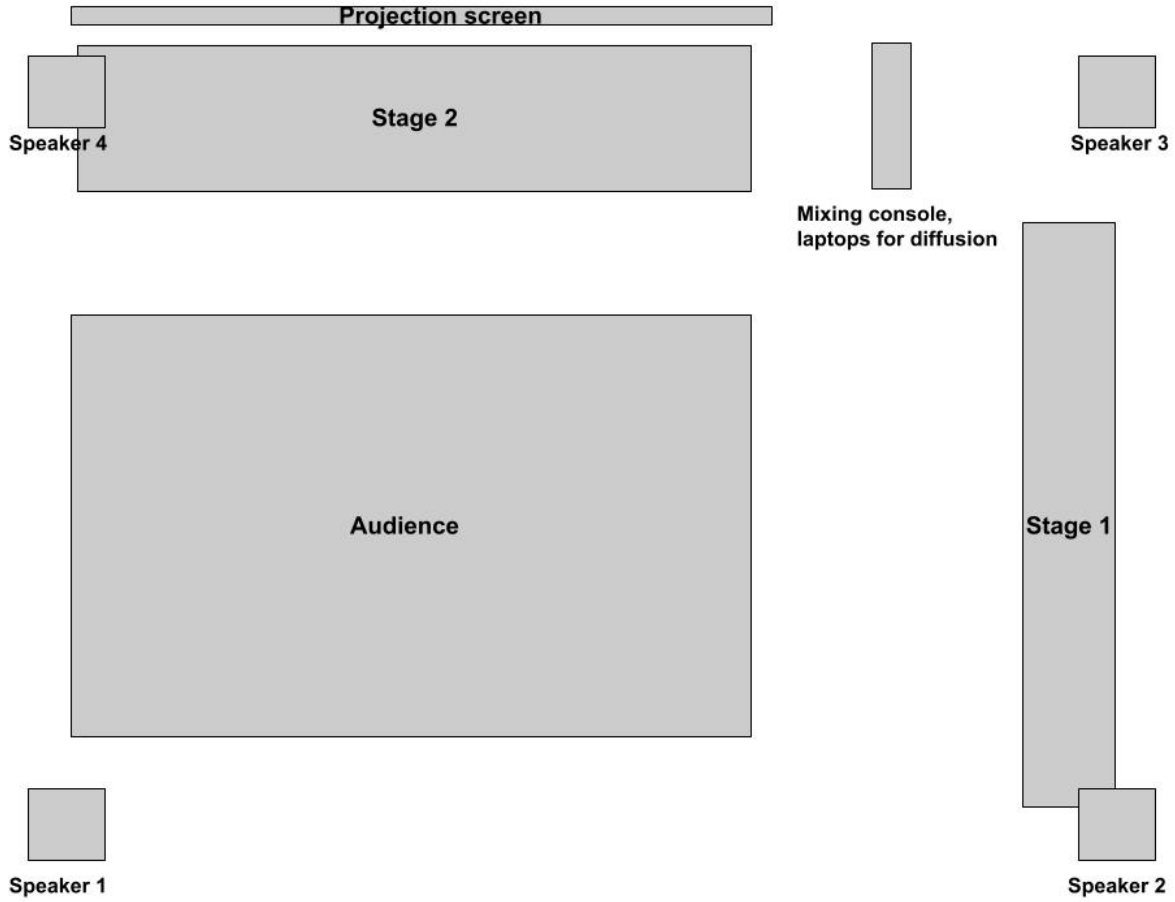
Projector

Duration: 9 minutes, 13 seconds

Performance Instructions

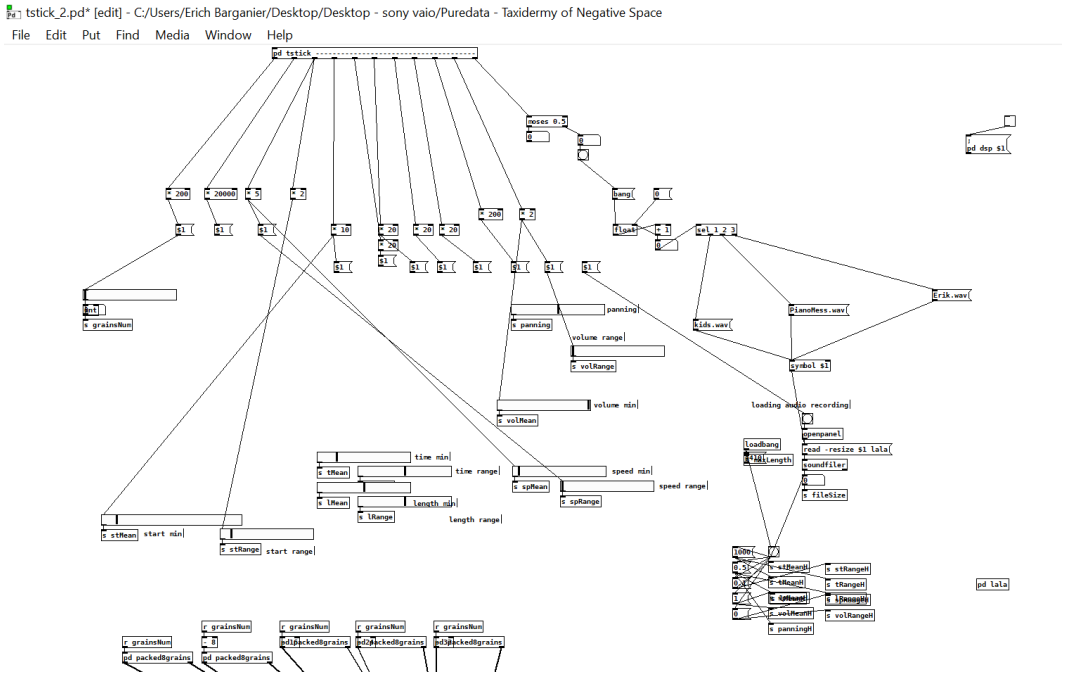
This piece should be performed by a technician controlling all electronic elements and a dancer holding a Tenor T-Stick, a DMI conceived by Joseph Malloch and D. Andrew Stewart at the Input Devices and Music Interaction Laboratory (IDMIL) at McGill University. It wirelessly outputs data from tactile sensors and a gyroscope to a connected computer as a means to create sound.

This composition should be performed in a space with a quadraphonic sound system and two stages, as diagramed below:

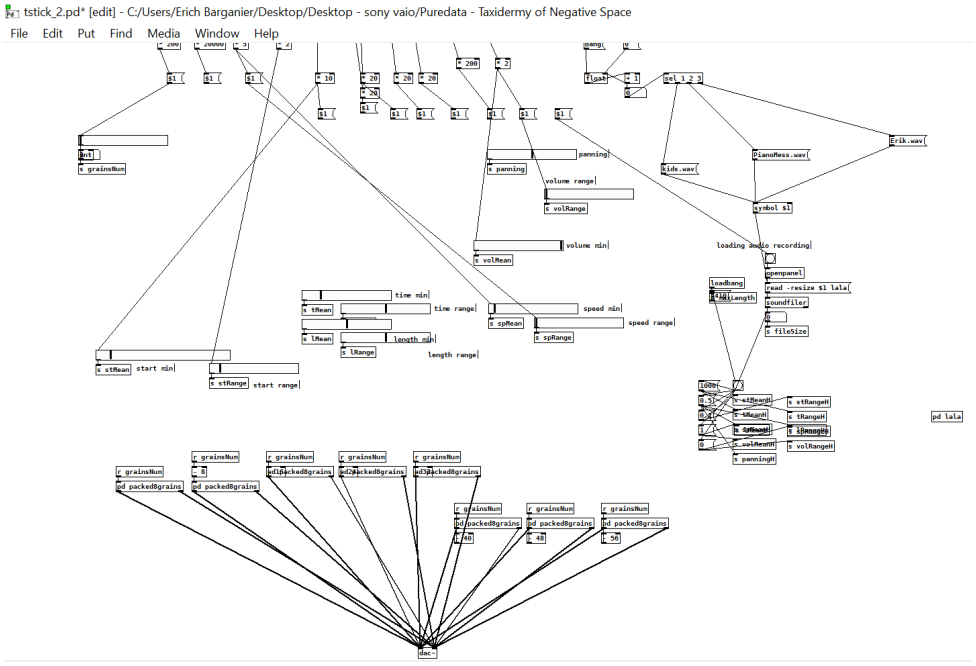


To start the piece, the technician must preemptively activate and connect the T-Stick to their laptop. A detailed set of instructions and a troubleshooting FAQ can be found here: https://github.com/IDMIL/T-Stick-Archive

Once the T-Stick is wirelessly connected to the computer, the following Pure Data patch should be activated:



Top half of Pure Data Patch



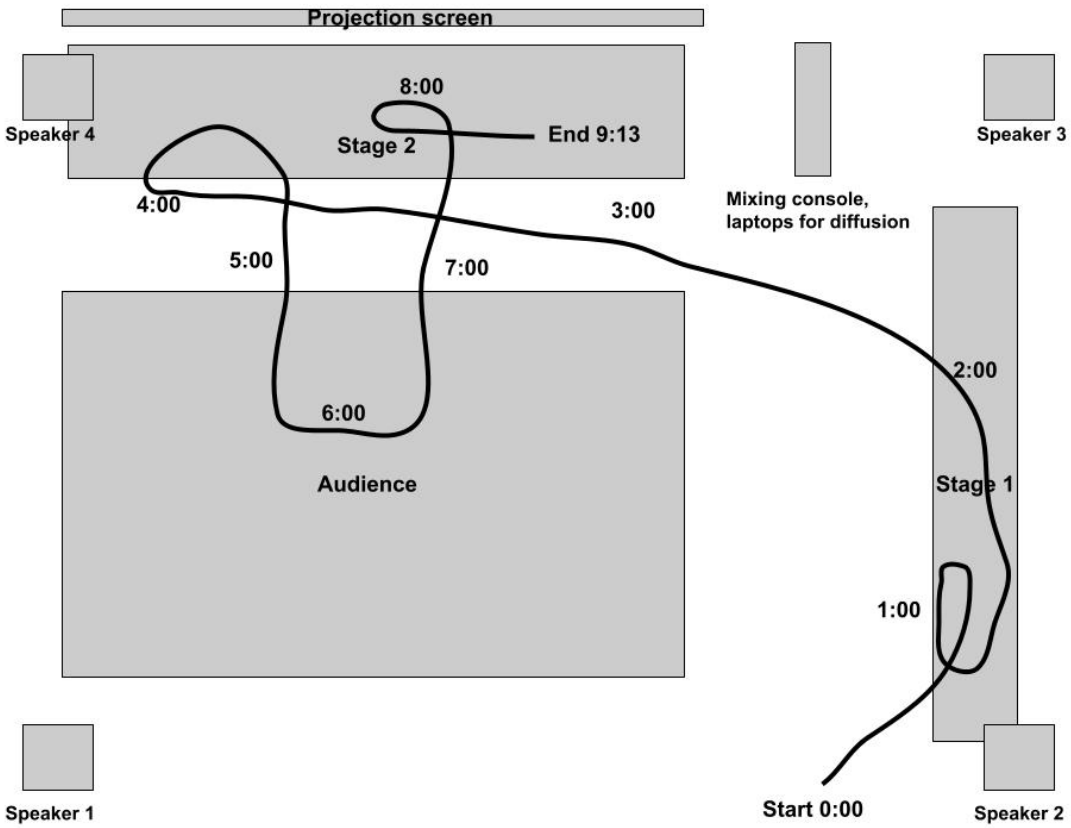
Lower half of Pure Data Patch

Three separate sound files must then be synced to the Pure Data patch. In the premiere performance, two piano samples and a voice sample of children were connected to the patch, but the technician may experiment with different sound files.

After ensuring that the T-Stick is sending data to the laptop and that the laptop is outputting audio, the technician may initiate a fixed media file that combines a premade video with a stereo audio track. This audio track should be diffused in quadraphonic sound by pairing the left channel to speakers 1 and 3, and pairing the right channel to speakers 2 and 4.

The code to create the fixed media audio is found at the end of the documentation. A fixed media file may be requested from Erich Barganier for any performance.

Once the fixed media begins, the dancer should pick up the T-Stick and follow general pathway as diagramed below:



The timestamps on the diagram indicate where the dancer should approximately be at any given time throughout the piece.

While moving along the designated pathway, the dancer may freely move with the T-Stick and use it as a musical prop as they respond to the fixed media. They may activate any sensors on the T-Stick that they wish, and may position it however they wish as they move through the space. They may additionally design their own unique choreography for each iteration of the piece, as long as they follow the designated pathway in the proper amount of time.

The piece is over when the fixed media comes to an end. At the end of the piece, the technician should slowly turn down the laptop volume to niente over the course of 7 seconds. The dancer should have reached the end of the designated pathway and slowly freeze in place over the course of 7 seconds. The technician and dancer should hold for applause for a beat and then relax. They may bow at the end of the performance.

The fixed media includes a video projection that must be used during the piece. The video should be projected behind the dancer on the wall or screen behind Stage 2. It is integral that the video be projected, not displayed on a monitor. During the course of the performance, the dancer should allow the projection to hit their body. As such, they should wear light or white clothing that will help amplify this effect.

The fixed media may be found here: https://youtu.be/-NatRCS5Dng

For reference purposes, the audio included in the fixed media was created using SuperCollider. The code for the fixed media may be found on the next page.

```
s.makeGui;
////////
(
~mySynthesisDefinition={{Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05,
0.051], 1), Mix.fill(2, {Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160,
320].choose, [0.05, 0.051], ChaosGen.allSubclasses.choose.ar(rrand(1,
30)))})*0.3,
SinOsc.ar([22.5, 22.51], mul: 0.3), SinOsc.ar([440, 441], mul: 0.003),
Pulse.ar([440, 441], width: SinOsc.kr([0.01, 0.012]), mul: 0.003)])*0.06};
{Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05, 0.051], 1), Mix.fill(2,
{Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160, 320].choose, [0.05,
0.051],
ChaosGen.allSubclasses.choose.ar(rrand(1, 30)))})*0.3, SinOsc.ar([22.5,
22.51],
mul: 0.3), SinOsc.ar([440, 441], mul: 0.003), Pulse.ar([440, 441], width:
SinOsc.kr([0.01, 0.012]), mul: 0.003)]) *
ChaosGen.allSubclasses.choose.ar(rrand(1,
30))};
{FreeVerb.ar({Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05, 0.051],
1),
Mix.fill(2, {Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160, 320].choose,
[0.05, 0.051], ChaosGen.allSubclasses.choose.ar(rrand(1, 30)))})*0.3,
SinOsc.ar([22.5, 22.51], mul: 0.3), SinOsc.ar([440, 441], mul: 0.003),
Pulse.ar([440, 441], width: SinOsc.kr([0.01, 0.012]), mul: 0.003)]) *
ChaosGen.allSubclasses.choose.ar(rrand(1, 30))})};}
)
(~mySynthesisDefinitionM = {CmdPeriod.run; ~mySynthesisDefinition.play;})
(~mySynthesisDefinitionM.value)
////////////
(
~synth1={
~a={SinOsc.ar([22.5, 23], mul: ChaosGen.allSubclasses.choose.ar(rrand(1,
30))*
0.3)};
~b={SinOsc.ar([22.5, 23], mul: 0.3)};
~c={SinOsc.ar([22.5, 23]*400, mul: ChaosGen.allSubclasses.choose.ar(rrand(1,
30))*
0.03)};
~d={SinOsc.ar([22.5, 23]*400, mul: 0.03)};
Out.ar(0, ~a+~b+~c+~d);
}.scope;
)
//////////////
(
~synth2={Mix.fill(4, {Mix.new([SinOsc.ar([22.5, 22.5137] * rrand(0.8, 1.2),
mul:0.3), SinOsc.ar([22.5, 22.5137] * rrand(0.8, 1.2) * 40, mul:0.03),
SinOsc.ar([22.5, 22.5137] * rrand(0.8, 1.2) * 80, mul:0.003),
SinOsc.ar([22.5,
22.5137] * rrand(0.8, 1.2) * 150, mul:0.003)])}) * 0.3};
).play;
```

```
/////////////////
(
~synth3={{Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05, 0.051], 1),
Mix.fill(2, {Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160, 320].choose,
[0.05, 0.051], ChaosGen.allSubclasses.choose.ar(rrand(1, 30)))})*0.3,
SinOsc.ar([22.5, 22.51], mul: 0.3), SinOsc.ar([440, 441], mul: 0.003),
Pulse.ar([440, 441], width: SinOsc.kr([0.01, 0.012]), mul: 0.003)])*0.06};
{Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05, 0.051], 1), Mix.fill(2,
{Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160, 320].choose, [0.05,
0.051],
ChaosGen.allSubclasses.choose.ar(rrand(1, 30)))})*0.3, SinOsc.ar([22.5,
22.51],
mul: 0.3), SinOsc.ar([440, 441], mul: 0.003), Pulse.ar([440, 441], width:
SinOsc.kr([0.01, 0.012]), mul: 0.003)]) *
ChaosGen.allSubclasses.choose.ar(rrand(1,
30))};
{FreeVerb.ar({Mix.new([Resonz.ar(WhiteNoise.ar, [55, 55.1], [0.05, 0.051],
1),
Mix.fill(2, {Resonz.ar(WhiteNoise.ar, [55, 55.1] * [40, 80, 160, 320].choose,
[0.05, 0.051], ChaosGen.allSubclasses.choose.ar(rrand(1, 30)))})*0.3,
SinOsc.ar([22.5, 22.51], mul: 0.3), SinOsc.ar([440, 441], mul: 0.003),
Pulse.ar([440, 441], width: SinOsc.kr([0.01, 0.012]), mul: 0.003)]) *
ChaosGen.allSubclasses.choose.ar(rrand(1, 30))})};};
).play;
//////////////////////
(
~synth4 = {[Resonz.ar([[Dust,
Dust2].choose.ar(ChaosGen.allSubclasses.choose.ar(0.1).abs * 10), [Dust,
Dust2].choose.ar(ChaosGen.allSubclasses.choose.ar(0.1).abs * 10)], rrand(30,
1200),
0.05)*60, FreeVerb.ar(Resonz.ar([[Dust,
Dust2].choose.ar(ChaosGen.allSubclasses.choose.ar(0.1).abs * 10), [Dust,
Dust2].choose.ar(ChaosGen.allSubclasses.choose.ar(0.1).abs * 10)], rrand(30,
1200),
0.05)*60)]};
).play;
//////////////////
(
~synth5={var a, b, c, d;
a={[SinOsc.ar(rrand(30, 75), mul: ChaosGen.allSubclasses.choose.ar(rrand(1,
30))),
SinOsc.ar(rrand(30, 75), mul: ChaosGen.allSubclasses.choose.ar(rrand(1,
30)))] *
0.3};
b={[SinOsc.ar(rrand(220, 440), mul: ChaosGen.allSubclasses.choose.ar(rrand(1,
30))), SinOsc.ar(rrand(220, 440), mul:
ChaosGen.allSubclasses.choose.ar(rrand(1,
30)))] * 0.1};
c={[SinOsc.ar(rrand(880, 2200), mul:
ChaosGen.allSubclasses.choose.ar(rrand(1,
```

```
30))), SinOsc.ar(rrand(880, 2200), mul:
ChaosGen.allSubclasses.choose.ar(rrand(1,
30)))] * 0.064};
d={[SinOsc.ar(rrand(8800, 22000), mul:
ChaosGen.allSubclasses.choose.ar(rrand(1,
30))), SinOsc.ar(rrand(8800, 22000), mul:
ChaosGen.allSubclasses.choose.ar(rrand(1,
30)))] * 0.064};
Out.ar(0, (a+b+c+d)*0.4);
};
).play;
////////////////////Improvise with next grouping of synths
(
SynthDef(\kick1, {
var snd;
snd = DC.ar(0);
snd = snd + (SinOsc.ar(XLine.ar(800, 400, 0.01)) * Env.perc(0.0005,
0.01).ar);
snd = snd + (BPF.ar(Hasher.ar(Sweep.ar), XLine.ar(800, 100, 0.01), 0.6) *
Env.perc(0.001, 0.02).delay(0.001).ar);
snd = snd + (SinOsc.ar(XLine.ar(172, 50, 0.01)) * Env.perc(0.0001, 0.3, 1,
\lin).delay(0.005).ar(2));
snd = snd.tanh;
Out.ar(\out.kr(0), Pan2.ar(snd, \pan.kr(0), \amp.kr(0.1)));
}).add;
)
Synth(\kick1, [amp: 0.4]);
(
SynthDef(\kick2, {
var snd;
snd = DC.ar(0);
snd = snd + (HPF.ar(Hasher.ar(Sweep.ar), 1320) * Env.perc(0.003, 0.03).ar *
0.5);
snd = snd + (SinOsc.ar(XLine.ar(750, 161, 0.02)) * Env.perc(0.0005,
0.02).ar);
snd = snd + (SinOsc.ar(XLine.ar(167, 52, 0.04)) * Env.perc(0.0005,
0.3).ar(2));
snd = snd.tanh;
Out.ar(\out.kr(0), Pan2.ar(snd, \pan.kr(0), \amp.kr(0.1)));
}).add;
)
Synth(\kick2, [amp: 0.4]);
////////////////////////////////////
({
Limiter.ar(
GVerb.ar(
(
BPF.ar(
WhiteNoise.ar([0.07,0.07]) +
Blip.ar([13,19], 200.rand, mul:0.5),
```

```
SinOsc.kr(
SinOsc.kr([1/108,1/109]).range(1/108, 1/13)
).exprange(10, 2300),
PMOsc.kr(1/54,1/216, 3).range(0.1, 2)
)
*
SinOsc.ar(Array.rand(20, 1/216, 1), mul:
Array.rand(20, 0.2, 1)).reshape(10,2)
).sum,
roomsize:10,
damping: PMOsc.kr(1/27, 1/108, 3).range(0.5, 1),
drylevel: SinOsc.kr(1/9).range(0.1, 1)
)
+
GVerb.ar(
Pan2.ar(
LPF.ar(
DynKlank.ar(
`[
Array.rand(6, 600,
4000).collect({|freq|
SinOsc.kr(1/108).range(freq/2,freq)
}),
nil,
Array.rand(6, 1/108,
1/27).collect({|freq|
SinOsc.kr(freq).range(1/108,1/3)
})
],
Limiter.ar(
Dust.ar(
SinOsc.kr(1/256).exprange(1/27, 3), TRand.kr(0.15, 0.25, Dust.kr(1/9)))
+
Impulse.ar(
SinOsc.kr(1/108).exprange(1/54, 3), 0, TRand.kr(0.6, 0.8, Dust.kr(1/3)))
)
),
1700,
LFPar.kr(1/27).exprange(0.05, 0.2)
),
SinOsc.kr(1/9).range(-0.2, 0.2)
),
roomsize: 30,
drylevel: 0.5
)
)
}.play;
)
//////////////////////////
(
```

```
SynthDef(\noise, { arg freq, amp, q;
var signal = BPF.ar(
WhiteNoise.ar,
freq: In.kr(freq),
mul: In.kr(amp),
rq: In.kr(q)
);
Out.ar(0, signal ! 2)
}).add;
SynthDef(\line, { arg start, end, dur, bus;
var env = Env([start, end], [dur]);
var signal = EnvGen.kr(env, doneAction: 2);
Out.kr(bus, signal ! bus.numChannels);
}).add;
)
(
~noiseRamper = { arg initialFreq, initialAmp, initialQ, group=nil;
var lineGroup = Group.new(group, \addToTail);
var soundGroup = Group.after(lineGroup);
var freqBus = Bus.control(s, 2).set(initialFreq);
var ampBus = Bus.control(s, 2).set(initialAmp);
var qBus = Bus.control(s, 2).set(initialQ);
var noise = Synth.head(soundGroup, \noise, [
\freq, freqBus,
\amp, ampBus,
\q, qBus
]);
var ramp = { arg bus, end, dur;
lineGroup.freeAll;
bus.get { arg value;
Synth.head(lineGroup, \line, [
\bus, bus,
\start, value,
\end, end,
\dur, dur
])
};
};
(
free: { arg self;
noise.free;
lineGroup.freeAll;
soundGroup.freeAll;
},
rampFreq: { arg self, end, dur;
ramp.value(freqBus, end, dur);
self
},
rampAmp: { arg self, end, dur;
ramp.value(ampBus, end, dur);
```

```
self
},
rampQ: { arg self, end, dur;
ramp.value(qBus, end, dur);
self
},
)
};
~noisesFromNetwork = { arg network, fundamental, masterGroup;
network.collect { arg layer, lIndex;
var baseIndex = lIndex + 1;
layer.asArray.collect { arg row, rIndex;
var rowIndex = rIndex + 1;
row.collect { arg value, vIndex;
var valueIndex = vIndex + 1;
var freqOffset = (2 + value) ** (rowIndex +
valueIndex + baseIndex).nextPrime;
var freq = (fundamental * baseIndex) + freqOffset;
var amp = value + 0.9;
var q = value * 0.001;
~noiseRamper.value(freq, amp, q, masterGroup)
}
}
}
};
)
(
~neuralNetwork = { arg layerDimensions;
layerDimensions.collect { arg dimensions;
Matrix.with(Array.fill(dimensions.first, {
Array.rand(dimensions.last, 0, 1.0) }))
}
};
~forward = { arg network, input;
var weightedInputs = [];
var activations = [input];
var currentWeightedInput, currentActivation = input;
network.do { arg layer;
currentWeightedInput = currentActivation * layer;
currentActivation = currentWeightedInput.tanh;
weightedInputs = weightedInputs add: currentWeightedInput;
activations = activations add: currentActivation;
};
[weightedInputs, activations]
};
~guess = { arg network, input; ~forward.value(network, input).last.last };
~outputError = { arg guess, target; target - guess };
~sigmoidPrime = { arg x; 1 - (x.tanh ** 2) };
~backProp = { arg network, input, target; // calculate weight updates to
improve
```

```
network performance
var result = ~forward.value(network, input);
var weightedInputs = result.first;
var activations = result.last;
var guess = activations.last;
var error = target - guess;
var delta, updates;
delta = Matrix.with(error.asArray *
weightedInputs.last.collect(~sigmoidPrime).asArray);
updates = [activations.drop(-1).last.flop * delta];
(network.size - 2).to(0, -1) do: { arg index;
var weightedInput = weightedInputs at: index;
var activation = activations at: index;
var layer = network at: (index + 1);
var derivative = weightedInput.collect(~sigmoidPrime);
delta = Matrix with: ((delta * layer.flop).asArray *
derivative.asArray);
updates = [(activation.flop * delta)] ++ updates
};
updates
};
~applyUpdates = { arg network, updates, learningRate=0.1;
network collect: { arg layer, index; layer + (updates.at(index) *
learningRate) }
};
~trainLoop = { arg network, observations, steps=10, learningRate=0.1,
action={};
Routine.new {
steps do: { arg idx;
var observation = observations.choose;
var input = observation.first;
var target = observation.last;
var updates = ~backProp.value(network, input, target);
network = ~applyUpdates.value(network, updates,
learningRate);
action.value(idx, network, input, target);
};
'done'.postln;
}
};
)
(
var network = ~neuralNetwork value: [[2, 3], [3, 1]];
var observations = [
[[[0, 1]], [[1]]],
[[[1, 0]], [[1]]],
[[[1, 1]], [[0]]],
[[[0, 0]], [[0]]],
].collect { arg obs; obs collect: { arg x; Matrix.with(x) } };
~master = Group.new;
```

```
~fundamental = 300;
~stepDur = 10;
~noises = ~noisesFromNetwork.value(network, ~fundamental, ~master);
~netLoop = ~trainLoop.value(network, observations,
steps: 100000,
learningRate: 0.01,
action: { arg count, net, in, t;
~stepDur.wait;
"-------------".postln;
net.asArray do: { arg layer, lIndex;
var layerIndex = lIndex + 1;
layer do: { arg row, rIndex;
var rowIndex = rIndex + 1;
row do: { arg value, vIndex;
var noise =
~noises.at(lIndex).at(rIndex).at(vIndex);
var valueIndex = vIndex + 1;
var freqOffset = (2 + value) ** (rowIndex +
valueIndex + layerIndex).nextPrime;
var freq = (~fundamental * layerIndex) +
freqOffset + (value * 25);
var amp = value + 0.9;
var q = value * (0.001 + 0.009.rand);
[freq, amp, q].postln;
if(noise.notNil && [true, false].choose, {
noise.rampAmp(amp,
~stepDur).rampFreq(freq, ~stepDur).rampQ(q, ~stepDur);
})
}
}
};
}
).play
)
////////////////////////////starts slowly, inaudibly
(
SynthDef(\bank, {
arg startFreqs = #[300, 400, 500, 600], endFreqs = #[300, 400, 500, 600],
vol = 40, bw = 0.0025, dur;
var signal = WhiteNoise.ar;
var shape = EnvGen.kr(
Env([0, 1, 1, 0], [dur * 0.05, dur, dur * 0.05], [\sin, \hold,
\lin]),
doneAction: 2
);
signal = BBandPass.ar(
signal,
freq: (startFreqs +++ endFreqs).collect { arg freqPair;
var freq = Line.kr(freqPair[0], freqPair[1], dur: dur);
var wobble = SinOsc.kr(0.01 + 0.05.rand, mul: 10, phase: 1
```

```
- 2.rand);
freq + wobble
},
mul: shape * vol,
bw: bw
);
Out.ar(0, Mix.ar(signal) ! 2)
}).store;
)
(
var freqPairs = Pseq([
[[300, 400, 500, 600], 450 ! 4],
[450 ! 4, [300, 400, 500, 600]]
], 1
);
Pbind(
\instrument, \bank,
#[startFreqs, endFreqs], freqPairs,
\dur, 3600,
\vol, 1.0,
\amp, 0.9
).play;
)
//////////
(
// 1 - The eigenvalues of the feedback coefficient matrix
d = [
-1,
Polar(1, -3pi / 4),
Polar(1, -pi / 2),
Polar(1, -pi / 6),
1,
Polar(1, pi / 6),
Polar(1, pi / 2),
Polar(1, 3pi / 4)
];
// 2 - Compute the feedback matrix from the given eigenvalues
n = d.size;
a = (Matrix.newIDFT(n) * Matrix.withFlatArray(n, 1, d)).real.flat / sqrt(n);
)
(
s.waitForBoot({
var primePowerDelays = { arg delays;
(delays collect: { |delay, i|
var prime = i.nthPrime;
prime ** ((log(delay) / log(prime)) + 0.5).floor;
}).asInteger / s.sampleRate;
};
var delayLengths = { arg n, dmin, dmax;
var nm1 = n - 1;
```

```
var d = dmin * ((dmax / dmin) ** ((0..nm1) / nm1));
(d * s.sampleRate).round(1.0).asInteger;
};
SynthDef(\sine, { arg out, freq = 440, amp = 0.5, trigFreq = 1;
Out.ar(out, Decay.ar(Impulse.ar(trigFreq), 0.2, SinOsc.ar(freq, 0,
amp)))
}).add;
SynthDef(\fdn, { arg in, out, scale = 1.0, coef = 0.5;
var a, x, w, fb, delT;
fb = LocalIn.ar(n);
a = \a.kr(0 ! n);
delT = \delT.kr(primePowerDelays.(delayLengths.(n, 0.03, 0.06)));
x = In.ar(in);
w = a.size collect: { arg i;
DelayN.ar(a.rotate(i).inject(x, { |input, coef|
coef * fb[i] + input
}), 1, (delT[i] * scale - ControlDur.ir).fold(0.0, 1.0))
};
LocalOut.ar(LeakDC.ar(OnePole.ar(w, coef)));
Out.ar(out, w.sum.tanh ! 2)
}).add;
{ s.scope(2) }.defer
});
)
(
s.makeBundle(nil, {
x = Synth(\sine, [\out, 10, \trigFreq, 0.25, \amp, 0.1], 1, \addToHead);
y = Synth(\fdn, [\in, 10, \out, 0, \a, a, \scale, 1, \coef, 0.5], 1,
\addToTail);
});
)
//////////
(
SynthDef(
\isochronic,
{
arg freq1=97.94, freq2=136.1, freq3=174, freq4=285, freq5=396,
freq6=417,
freq7=432, freq8=528, freq9=639, freq10=741, freq11=852,
freq12=963, beat1=2.6, beat2=3.9,
amp1=0.0, amp2=0.0, amp3=0.0, amp4=0.0, amp5=0.0, amp6=0.0,
amp7=0.0, amp8=0.0,
amp9=0.0, amp10=0.0, amp11=0.0, amp12=0.0, amp=0.0, out=0,
isofreq=1, width=0.5;
var sin1, sin2, sin3, sin4, sin5, sin6, sin7, sin8, sin9, sin10,
sin11, sin12, brainmix, isochronic;
sin1=SinOsc.ar([freq1-(beat1/2),freq1-beat1,freq1+(beat1/2),freq1+beat1],
0,amp1,0);
sin2=SinOsc.ar([freq2-(beat2/2),freq2-beat2,freq2+(beat2/2),freq2+beat2],
0,amp2,0);
```

```
sin3=SinOsc.ar([freq3-(beat1/2),freq3-beat1,freq3+(beat1/2),freq3+beat1],
0,amp3,0);
sin4=SinOsc.ar([freq4-(beat2/2),freq4-beat2,freq4+(beat2/2),freq4+beat2],
0,amp4,0);
sin5=SinOsc.ar([freq5-(beat1/2),freq5-beat1,freq5+(beat1/2),freq5+beat1],
0,amp5,0);
sin6=SinOsc.ar([freq6-(beat2/2),freq6-beat2,freq6+(beat2/2),freq6+beat2],
0,amp6,0);
sin7=SinOsc.ar([freq7-(beat1/2),freq7-beat1,freq7+(beat1/2),freq7+beat1],
0,amp7,0);
sin8=SinOsc.ar([freq8-(beat2/2),freq8-beat2,freq8+(beat2/2),freq8+beat2],
0,amp8,0);
sin9=SinOsc.ar([freq9-(beat1/2),freq9-beat1,freq9+(beat1/2),freq9+beat1],
0,amp9,0);
sin10=SinOsc.ar([freq10-(beat2/2),freq10-
beat2,freq10+(beat2/2),freq10+beat2],
0,amp10,0);
sin11=SinOsc.ar([freq11-(beat1/2),freq11-
beat1,freq11+(beat1/2),freq11+beat1],
0,amp11,0);
sin12=SinOsc.ar([freq12-(beat2/2),freq12-
beat2,freq12+(beat2/2),freq12+beat2],
0,amp12,0);
isochronic=LFPulse.ar(MouseX.kr(1,60,'linear'),0,MouseY.kr(0.01,1,'linear'),a
mp,0).
lag(0.01);
//isochronic=LFPulse.ar(isofreq,0,width,amp,0).lag(0.001);
brainmix=Mix.new([sin1, sin2, sin3, sin4, sin5, sin6, sin7, sin8,
sin9, sin10, sin11, sin12]*isochronic);
Out.ar(out,brainmix, 0, 1);
}
).add
)
//Run Synth
~isobrainmachine = Synth(\isochronic,[\out,0,\amp,0.5, \amp1,0.24,
\amp2,0.22,
\amp3,0.20, \amp4,0.18, \amp5,0.17, \amp6,0.16, \amp7,0.14, \amp8,0.13,
\amp9,0.12,
\amp10,0.11, \amp11,0.1, \amp12,0.1,\beat1,2.6,\beat2,3.9]);
//Modify parameter
~isobrainmachine.set(\beat1,0.4);
//Stop Synth
~isobrainmachine.free;
//////////////
(
TempoClock.default.tempo = 2.5;
b = b ?? ();
f = f ?? ();
b.reverb = Bus.audio(s,2);
b.delay = Bus.audio(s,2);
```

```
)
(
SynthDef(\sine,
{
|freq=200,amp=0.1,harmonics=1,pan=0,cutoff=1000,out=0|
var audio, env;
freq = freq + (freq * LFNoise1.kr(0.2,0.01));
env = EnvGen.kr(Env.perc,1,amp,doneAction:2);
audio = Blip.ar(freq,harmonics,env);
audio = MoogFF.ar(audio,cutoff);
audio = Pan2.ar(audio,pan);
Out.ar(out,audio);
}
).add;
SynthDef(\reverb,
{
|in,out,roomsize=10, revtime=3, damping=0.5, inputbw=0.5,
spread=15, drylevel=1|
var audio;
audio = In.ar(in, 2);
audio = GVerb.ar(audio, roomsize, revtime, damping, inputbw,
spread, drylevel);
Out.ar(out,audio);
}
).add;
SynthDef(\hiss,
{
|out=0,pan=0,amp=0.01,hpffreq=8000|
var audio = PinkNoise.ar(amp);
audio = HPF.ar(audio,hpffreq) * LFNoise0.kr(24,0.2,1.8);
audio = audio * LFNoise0.kr(0.2).range(0.5,1.2);
audio = Pan2.ar(audio,pan);
Out.ar(out,audio);
}
).add;
SynthDef(\delay,
{
|in,out=0|
var audio = DelayC.ar(In.ar(in,2), 0.3,
LFNoise2.kr(0.2).range(0.0,0.2));
Out.ar(out,audio);
}
).add;
)
(
f.delay.free;
f.delay = Synth(\delay,[\in,b.delay]);
)
(
Pdef(\sine,
```

```
Ppar([
Pbind(
\instrument, \sine,
\scale, Scale.harmonicMinor,
\degree, Ppatlace([
Pseq([0,2,4,6],inf),
Pseq([0,2,4,6],inf)+7
],inf),
\harmonics, Pbrown(-20,6,3,inf).clip(1,6) + 10,
\pan, Pbrown(-1.0,1.0,0.03,inf),
\dur, 0.25,
\cutoff, Pbrown(0,100,5,inf).linexp(0,100,200,2000),
\highamp, Pbrown(-1.0,1.0,0.3,inf),
\amp, Ppatlace([
Pseq([0.13,0.05,0.07,0.12],inf),
Pseq([0.13,0.05,0.07,0.12],inf) * Pkey(\highamp)
],inf) * Pgauss(1, 0.3,inf),
\type, Pif(Pkey(\amp) > 0, \note, \rest),
\mtranspose, Pstutter(Pwhite(1,5)*8,Pwhite(0,7,inf)),
\shifttempo, Pstutter(8,Pbrown(1.8,3.1,1.0,inf)),
\shifttempo, Pfunc({|e|TempoClock.default.tempo =
e.shifttempo}),
\out, b.reverb
),
Pmono(
\reverb,
\in, b.reverb,
\roomsize,10,
\revtime,3,
\damping,0.5,
\inputbw,0.5,
\spread,15,
\drylevel,1,
\out, b.delay
),
Pmono(
\hiss,
\amp, [0.02,0.03],
\hppffreq, [20,40],
\pan, [-1,1],
\out,b.delay
)
])
).play
)
Pdef(\sine).stop;
//////////
({
SinOsc.ar(10 + (50 * SinOsc.kr([50, 51], 0, SinOsc.kr(101, Saw.kr(0.12345,
678, 9), 0.2, 0.8), Pulse.kr([25, 25.5], 0.25, 0.125, -0.25))), 0, 0.5, 0);
```

```
}.play();
)
////////////
(
SynthDef("plucking", {arg amp = 0.1, freq = 440, decay = 5, coef = 0.1;
var env, snd;
env = EnvGen.kr(Env.linen(0, decay, 0), doneAction: 2);
snd = Pluck.ar(
in: WhiteNoise.ar(amp),
trig: Impulse.kr(0),
maxdelaytime: 0.1,
delaytime: freq.reciprocal,
decaytime: decay,
coef: coef);
Out.ar(0, [snd, snd]);
}).add;
)
(
Pbind(
\instrument, "plucking",
\scale, Scale.harmonicMinor,
\degree, Prand([1, 3, 5, 7, 9, 11, 14], inf),
\amp, Pwhite(0.1, 0.5),
\decay, Pwhite(1, 2),
\coef, Pseq([0.7, 0.8, 0.4], inf),
\dur, Prand([0.1, 0.2, 0.4, 0.7, 1, 1.5, 2], inf)
).play;
)
////////////
(
~name = \ghosts;
SynthDef(~name, {
arg freq=440, detune=3.0, atk=6, sus=4, rel=6,
curve1=1, curve2=(-1),
minCf=30, maxCf=6000, minRq=0.005, maxRq=0.04,
minBpfHz=0.02, maxBpfHz=0.25,
lowShelf=220, rs=0.85, db=6,
gate=1, amp=1, spread=1.0, out=0;
var sig, env;
env = EnvGen.kr(Env([0,1,1,0],[atk,sus,rel],[curve1,0,curve2]),
gate, levelScale:amp, doneAction:2);
sig = Saw.ar(
freq +
LFNoise1.kr({LFNoise1.kr(0.5).range(0.15,0.4)}!8).range(detune.neg,detune));
sig = BPF.ar(
sig,
LFNoise1.kr({LFNoise1.kr(0.13).exprange(minBpfHz,maxBpfHz)}!8).exprange(minCf
,
maxCf),
LFNoise1.kr({LFNoise1.kr(0.08).exprange(0.08,0.35)}!8).range(minRq, maxRq)
```

```
);
sig = BLowShelf.ar(sig, lowShelf, rs, db);
sig = SplayAz.ar(4, sig, spread);
sig = sig * env * 2;
Out.ar(out, sig);
}).add;
);
(
Pdef(~name, Pbind(
\instrument, ~name,
\freq, Pseq([8,4,2,4,1],inf) * Pwhite(0.2,3.3).stutter(2),
\detune, 0,
\minBpfHz, 0.1,
\maxBpfHz, Pseq([1,2,3,2],inf).expexp(1.0,16.0,0.1,076.0),
\minRq, 0.003,
\maxRq, Pseq([
Pexprand(0.008,0.028,4).stutter(2),
Pexprand(0.008,0.228,1).stutter(1),
],inf),
//\minCf, Pseq([65,75,50,55,58,60,78],inf).midicps * Pwhite(0.9,3.3),
\minCf, Pseq([90,94,80,150],inf) * Pexprand(1.0,4.3).stutter(3),
\maxCf, Pkey(\minCf) * Pwhite(1.3,1.5),
\amp, Pexprand(0.20,0.21) *2,
\atk, Pexprand(0.007,8),
\rel, Pwhite(0.5,3.5),
\sus, Pwhite(0.2,4.0),
\spread, Pexprand(1.5,8.0),
\dur, Pseq([2,1],inf)*Pwhite(0.8,2.3),
)).play
);
////////////
(
Ndef(\ro_v19, {
var freq = 40.rrand(90).debug('base freq');
var seed = thisThread.randSeed = 1000000000.rand.debug('seed');
var gen = [LFNoise2,LFNoise0,SinOsc,LFPulse,LFSaw];
var sig = Splay.ar({
var i = 4.exprand(40).asInteger;
Median.ar(
3.exprand(18),
EnvGen.ar(
Env( // wave form
[0] ++ Array.fill(i-1, {
gen.choose.kr(
0.25.coin.if({
gen.choose.kr(0.01.exprand(0.1)).exprange(0.1,1) }, { 0.1.exprand(1) }),
mul: 0.25.coin.if({
gen.choose.kr(0.05.exprand(10)).range(0.1,1) }, { 0.1.rrand(1) })
)
}) ++ [0],
```

```
Array.rand(i-1, 0.1,1.0).normalizeSum,
[[-5.0.rrand(5),\sin].choose] ++
Array.fill(i-2, {
[\sin ! 1.rrand(6), -5.0.rrand(5) !
1.rrand(4)].flat.choose
}) ++ [[-5.0.rrand(5),\sin].choose]
).circle,
timeScale: Duty.kr( // freq
Drand([4,8,16,24], inf), 0,
Dwrand([1,2,4,8,16,32,64]*freq, ((14..7) **
1.rrand(3)).normalizeSum, inf) // octave
* 0.75.coin.if({ Duty.kr( // note
[Dxrand,
Drand].choose.new([0.125,0.25,0.5,1,2,4,6,8,12], inf), 0,
Dshuf({ (0..11).choose.midiratio }
! 1.exprand(8), inf)
) }, { (0..11).choose.midiratio })
).reciprocal
)
) * EnvGen.kr( // rhythm
Env.perc(
0.01.exprand(0.4) * 0.25.coin.if({ Duty.kr( // atk
Drand([0.5,1,2,4,8,12,16], inf), 0,
Dshuf({ [1,2,3,4].choose } ! 1.exprand(8),
inf)
) }, { 1 }),
0.1.exprand(4) * 0.75.coin.if({ Duty.kr( // release
Drand([0.5,1,2,4,8,12,16], inf), 0,
Dshuf({ [1,2,4,8,16].choose.reciprocal } !
1.exprand(8), inf)
) }, { 1 }),
LFNoise2.kr(0.1.exprand(10)).range(0.05,1), //
appearance
LFNoise2.kr(0.1.exprand(10)).range(-4,4),
).circle
);
} ! 3.rrand(9).debug('source'));
BHiShelf.ar(
GVerb.ar(
Rotate2.ar(sig[0], sig[1], Duty.kr(Drand([2,4,8],inf), 0,
Drand([-1,1,0],inf)) * LFSaw.kr(0.1.exprand(1))),
40.rrand(150).debug('room')
),
LFNoise2.kr(0.1.exprand(0.5)).range(4000,6500),
LFNoise2.kr(0.1.exprand(1)).range(1,3),
LFNoise2.kr(0.1.exprand(1)).range(-24,-16)
);
}).play;
)
```