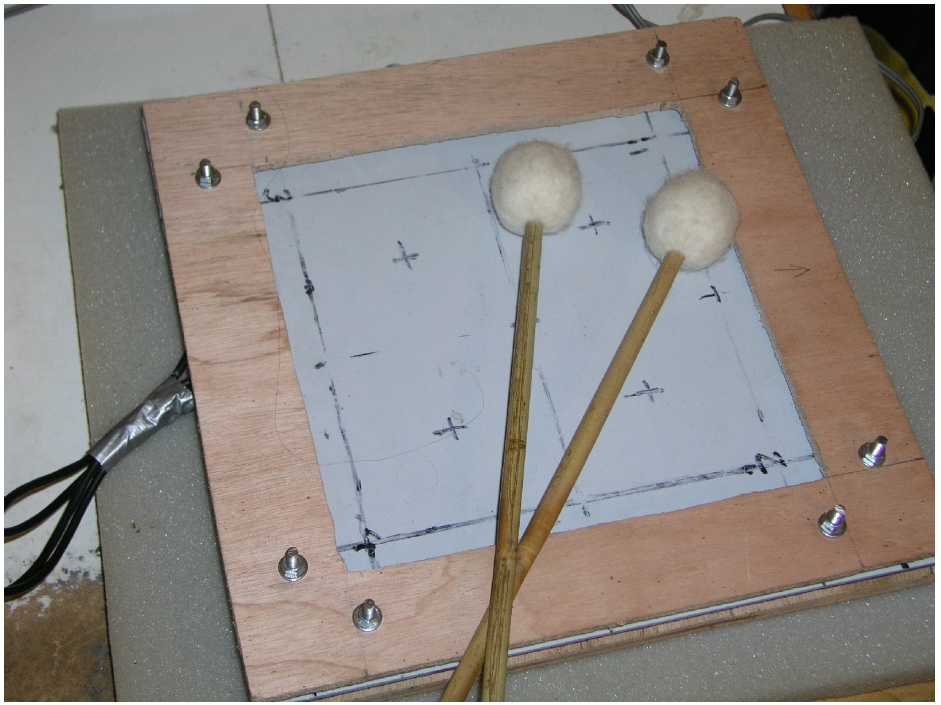
The synpad – a position sensing midi drum interface



I will be talking today about my attempts to build a cheap, playable, midi drum interface with position sensing capability.

Overview

- Motivations – why build this?
- Research – previous designs.
- Physical design.
- Electronics.
- Driver software – firmware and position mapper.
- Synth software – using supercollider.
- Results.
- Similar work.
- Future directions.
- Conclusion.

Motivations

Why build something like this?

- I find drum machines limiting and awkward.
- I wanted something more immediate and responsive.
- Drum triggers are good but you can't change the tone.
- Wanted something more like a real drum.

Research

- Tried a couple of other designs.
- Voltage gradients in a conductive rubber sheet.
- Time of flight of pressure waves.
- Just sensing transferred pressure worked ok.

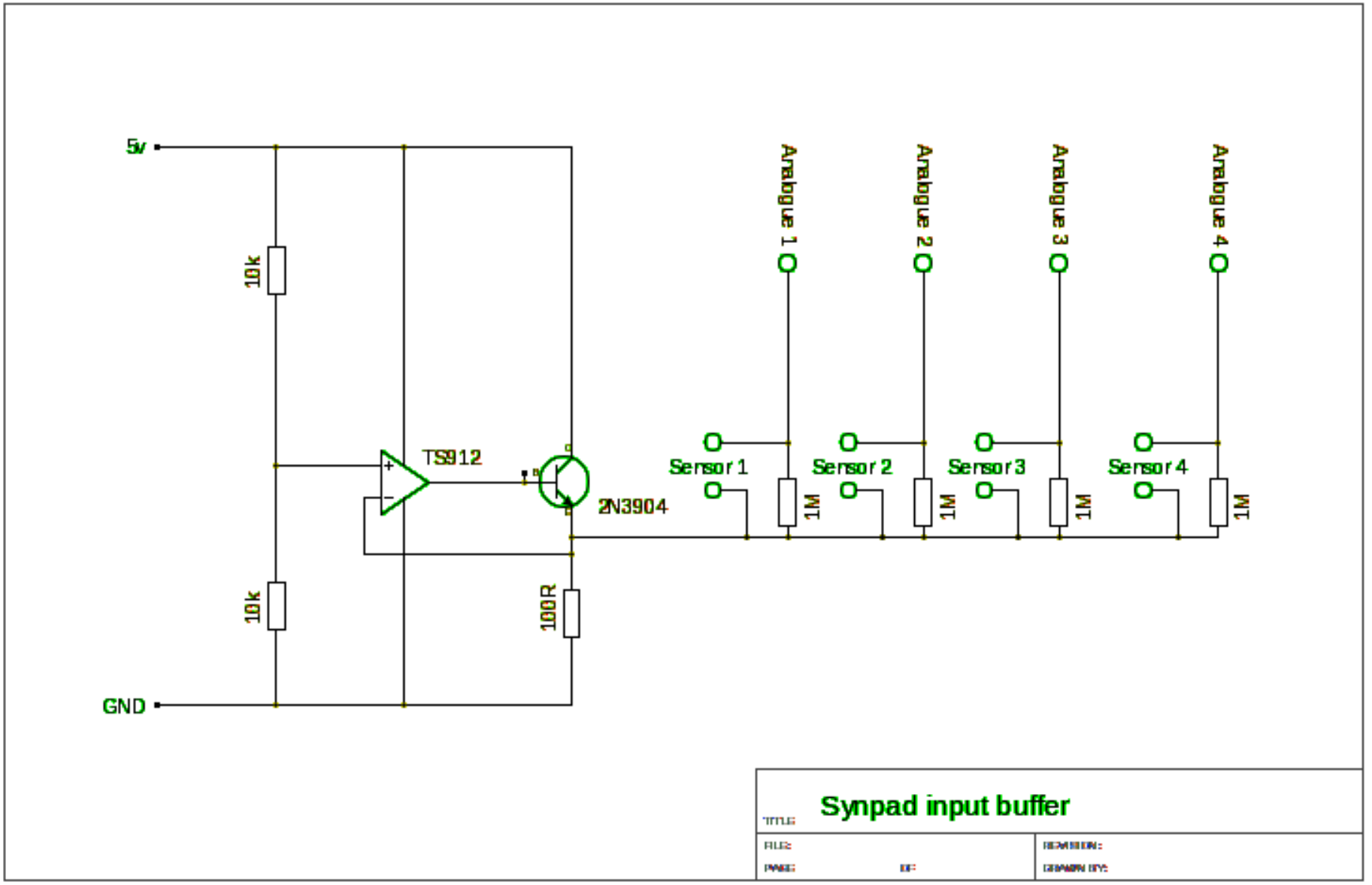
Physical design

- Playing surface is an aluminium sheet.
- The sheet moves freely in a wooden frame.
- Piezo sensors under each corner detect differences in transferred strike pressure.
- The signals are brought out to an arduino board.

Electronics

- Quite simple electronics.
- Provides a false ground level.
- Prevents voltage drift on the piezos.

Circuit diagram



Driver software - firmware

- Firmware turns analogue signals into 4 velocities.
- ADC sample rate: 76 kHz across 4 lines.
- Trigger level for detection.
- Absolute values summed for 16 samples = 0.84 ms
- Summed values are written to USB at 230400 baud (0.34 ms)
- 1300 sample / 68ms release period.

Position mapping software

- Python program maps sensor readings into x, y and velocity coordinates.
- Pad calibrated by striking at known positions.
- Least squares curve fit for position and velocity.
- X,Y and V are converted to midi note and controller values.

The position mapping equation.

```
# s[n] = reading for sensor n
# x = x or y return value.
# k[n] | f[n] = adjustable coefficients.
f1=s1
f2=s2*k2
f3=s3*k3
f4=s4*k4
x=(l1*f1+l2*f2+l3*f3+l4*f4)/
(f1+f2+f3+f4)
return x
```

Synth software

- The pad is just an interface.
- Existing synths not suitable.
- I wrote my own in SuperCollider.
- Could have used CSound or PureData.
- Could also use a graphical modular synth like Ingen.
- Learning to write synthdefs.
- 'Synth Secrets' series from 'Sound on sound'.

```
SynthDef.new("MidiDrum", { |vel=100, x=64, y=64,out=0|
    // resonant snare sound
    var sndbuf= Buffer.readChannel(s,
"/home/andy/Desktop/music/supercollider/samples/84001__s
andyrb__KBSD_C42_VELOCITY9.wav", channels:0);
       var rq=10**((16-y) / 41);
    var env,amp;
    var noteMin=30;//54;
    var noteMax=128;//66;
    var note=(x*(noteMax-noteMin)/127)+noteMin;
  //note=(note/12).floor*12;
    vel=vel+((127-note)/40)+((127-y)/50);
       amp=((vel-96)/3).dbamp;
    env=EnvGen.kr(Env.triangle(1,4),1,doneAction:2);

Out.ar(out,amp*env*Pan2.ar(RLPF.ar(PlayBuf.ar(1,sndbuf),
note.floor.midicps, rq  ), 0) );
    } ).store;
```

```
SynthDef.new("MidiDrum", { |vel=100, x=64, y=64,out=0|
    // synth drum with pink noise, comb delay line and low
pass filter.
  var rq=10**((y-40) / 41);
    var env,amp;
    var noteMin=55; // 200Hz
    var noteMax=128;//66;
    var note=(x*(noteMax-noteMin)/127)+noteMin;
  var baseFreq=100;
  amp=16*((vel-96)/3).dbamp;
    env=EnvGen.kr(Env.perc(0.01,0.5,1),1,doneAction:2);

Out.ar(out,amp*env*Pan2.ar(LPF.ar(CombC.ar(PinkNoise.ar(
0.1),1,1/baseFreq,rq),note.midicps), 0) );
    } ).store;
```

```
SynthDef.new("MidiDrum", { |vel=100, x=64, y=64, out=0|
   // bass drum patch with variable square wave / saw wave ratio.
   var baseFreq=50, baseDelayMin=0.1, baseDelayMax=3, baseAmp=1,
attack=0.01;
   var baseFreqMod=1, harmLPFreqMin=baseFreq,
harmLPFreqMax=baseFreq*10;
   var fmBaseFreq=500, fmModSig=250, fmAmp=0.5, fmDelay;
   var amp,ampEnv, baseFreqEnv, harmSig, harmLPFreq, baseDelay, sawRatio,
oscSig;
   baseDelay=0.5; //((y/128)*(baseDelayMax-baseDelayMin))+baseDelayMin;
   fmDelay=baseDelay/5;
   amp=((vel-32)/3).dbamp;
   harmLPFreq=((x/128)*(harmLPFreqMax-harmLPFreqMin))+harmLPFreqMin;

ampEnv=amp*EnvGen.kr(Env.perc(attack,baseDelay,baseAmp),1,doneAction:2)
;
   baseFreqEnv=EnvGen.kr(Env.perc(attack,baseDelay,baseFreqMod,'sine'));
   sawRatio=(y/128);
   oscSig=sawRatio*LFTri.ar(baseFreq+baseFreqEnv)+(1-
sawRatio)*Saw.ar(baseFreq+baseFreqEnv);
   harmSig=LPF.ar(oscSig,harmLPFreq);
   Out.ar(out,Pan2.ar(ampEnv*harmSig,0));
   } ).store;
```

```
SynthDef.new("MidiDrum", { |vel=100, x=64, y=64, out=0|
   // snare drum from synth secrets (based on roland 909).
   // different version with fixed noise delay and low pass filter.
   var part1Freq=180, part1Amp=0.1, part2Freq=330, part2Amp=0.05,
minDistortPow=0, maxDistortPow=3, partDelay=0.7;
   var attack=0.01, noiseLPFreq=10000, noiseHPFreq=2000, noiseAmp1=0.005,
noiseAmp2Ratio=2;
   var noiseDelay=0.4;
   var partSig, partEnv, amp, noiseEnv, noiseSig1, noiseSig2, noiseSig,
outSig,lpFreq;
   var distort;
   distort=10**(((x/128)*(maxDistortPow-minDistortPow))+minDistortPow);
   amp=((vel/4)-28).dbamp;
   partEnv=amp*EnvGen.kr(Env.perc(attack,partDelay,1),1,doneAction:2);
   partSig=part1Amp*atan(SinOsc.ar(part1Freq, 0, distort))
+part2Amp*atan(SinOsc.ar(part2Freq,0,distort));
   noiseSig1=noiseAmp1*LPF.ar(WhiteNoise.ar(1),noiseLPFreq);
   noiseSig2=(amp**0)*HPF.ar(noiseSig1*noiseAmp2Ratio,noiseHPFreq);
   noiseSig=(noiseSig1+noiseSig2);
   noiseEnv=amp*EnvGen.kr(Env.perc(attack,noiseDelay,1),1,doneAction:0);
   lpFreq=((y*3/5)+51).midicps;
   outSig=RLPF.ar(partSig*partEnv+noiseSig*noiseEnv,lpFreq,0.5);
   Out.ar(out,Pan2.ar(outSig,0));
   } ).store;
```

```
SynthDef.new("MidiDrum", { |vel=100, x=64, y=64,out=0|
    // interfering oscillators.
    var env,amp;
    var noteMin=54;
    var noteMax=66;
    var note=(x*(noteMax-noteMin)/128)+noteMin;
    var noteMiny=66;
    var noteMaxy=78;
    var notey=(y*(noteMaxy-noteMiny)/128)+noteMiny;
    amp=((vel-96)/3).dbamp;
    env=EnvGen.kr(Env.perc(0.1,0.5,1),1,doneAction:2);

Out.ar(out,amp*env*Pan2.ar(SinOsc.ar(note.midicps)*SinOsc
.ar(notey.midicps), 0) );
    } ).store;
```

Results

- Physically easy to construct.
- Low cost (approx 50-60 pounds)
- Playability is not bad.
- The accuracy of position mapping is about 15-20%.
- Velocity mapping is ok in practice. Lower cutoff.
- Latency of the firmware is about 1.1ms.
- Have written some playable synths.

Similar Work

- Various people have produced similar instruments.
- Korg Kaoss pad and Kaoscillator.
- Mandala drum from Synaesthesia Corp.
- Randall Jones's MSc thesis.

Future Directions

- Physical design could be improved.
- Might try a different design.
- A graphical interface would be good.
- Morphing presets.
- Synth design.

Conclusions

- The basic concept is sound.
- However this particular design has some weaknesses.
- Not much interest from people building their own.
- I enjoyed making it.
- I intend to develop the idea further.
- Learning to play it.

Acknowledgements

Thanks to Alaric Best and Dave Leack of Veraz Ltd. for their ideas and support with this project.

Thanks also to the Supercollider developers.

For more information and updates see:

http://highfellow.org/synpad