Jesse Austin-Stewart

Creating Interfaces for Live Octophonic Spatialisation of Sound

This publication may be cited as: Jesse Austin-Stewart. (2017). Creating Interfaces for Live Octophonic Spatialisation of Sound. *Pūrātoke: Journal of Undergraduate Research in the Creative Arts and Industries*, 1(1), 035-048.

Founded at Unitec Institute of Technology in 2017

ISSN 2538-0133

An ePress publication
epress@unitec.ac.nz
www.unitec.ac.nz/epress/

Unitec Institute of Technology
Private Bag 92025
Victoria Street West
Auckland 1010
Aotearoa New Zealand

Jesse Austin-Stewart

CREATING INTERFACES FOR LIVE OCTOPHONIC SPATIALISATION OF SOUND

Abstract

This essay will detail the process and creation of *elle – an octophonic drone interface* for the CMPO 381 – Interface Design for Live Electronics paper at Victoria University of Wellington.

elle – an octophonic drone interface is a 2-channel hardware interface that enables users to control an 8-channel MaxMSP patch, of the same name, with which one can create continuous drones by dropping any audio sample into the patch. These drones can then be spatialised within an octophonic array. The hardware and MaxMSP interface interact with one another to enable aural and visual feedback of the physical changes input by the user. It is designed as a simple interface to use, so that the knowledge barrier to entry is small.

Background

Intentional spatialisation of audio has existed since Alan Blumlein invented modern stereophonic sound in the 1930s to solve the issue of sound in film appearing to come from a different direction from where the sound was occurring on screen (BBC News, 2008). From then on, spatialisation of sound continued to develop, with quadraphonic and octophonic works first appearing in the 1950s, exemplified in works by Karlheinz Stockhausen and John Cage.

The New Zealand School of Music at Victoria University of Wellington has a strong culture of spatialisation of sound with multi-speaker arrangements. Recently, work has appeared from lecturers and postgraduate students in this realm, along with a strong push for realising ideas around multi-speaker spatialisation within undergraduate courses. Lecturers Dugal McKinnon, Jim Murphy and Mo Zareei collaborated on *Lost Oscillations* in 2015, "a sound installation that requires the human touch – literally – of its audience in reactivating and feeling through the layered sonic archaeology of Christchurch, the city's contemporary and historic soundscapes and ever-shifting spatial character" (McKinnon, 2015, para. 1). In her thesis, recent PhD graduate Bridget Johnson (2017) identifies a need for development of interfaces designed to spatialise audio in order to engage with spatial aspects of music. Her iPad interface *tactile.motion* enables the user to spatialise sound by moving objects around the iPad touch screen, which in turn moves sound around an octophonic speaker array in an intuitive, simple-to-use interface. One hope of Johnson's is that through the development of new spatial interfaces, new spatial aesthetics may emerge.

Numerous sonic arts courses at the New Zealand School of Music encourage live spatialisation of audio (mostly spatialising stereo works for an 8-speaker arrangement) and writing works for octophonic arrangements.

It is important to understand this context to understand from where the work originates and its influences derive.

MaxMSP patch

The MaxMSP patch will be detailed before describing the physical interface and its code, so that there is context to support and understand the coding and design decisions.

Figure 1. *ELLE – A DRONE INTERFACE* by Jesse Austin, MaxMSP patch GUI.

The octophonic MaxMSP patch derives from a patch of similar nature. MaxMSP is a visual programming language used for music and multimedia. Figure 1 is a mono version of *elle – an octophonic drone interface*. It was created as a part of the CMPO 211 – Projects in Interactive Music and Sound paper in 2016, and was designed so that users could drop in audio, and also effect live audio, to create continuous drones and manipulate textures.

The signal chain to control the samples began with manipulating the sample speed, where users could change the speed between -1 to 1 times the original speed. This then ran into a bandpass filter with which users can change the frequency (Freq.) and bandwidth of that filter (Q). This then ran into a reverb with all values predetermined, excluding the Dry/Wet, which you can change within the patch. It is then adjusted by a volume dial before being sent to the master channel. If selected, it can be sent to a bus channel before reaching the master. It can also be muted and soloed. The bus and master channels also run through a filter, with which you can change the Freq. and Q before both run through a volume dial. This signal flow proved effective, partly due to being similar to mixer views in traditional DAWs and also being relatively intuitive. This signal flow was copied and added to the newer octophonic patch.

Figure 2. *elle – an octophonic drone interface* by Jesse Austin, MaxMSP patch GUI.

In Figure 2, much of the adjusted octophonic patch is the same. There are additional 'lights' by channels 'Sample 1' and 'Sample 2' as well as 'Sample Speed', however these lights will be detailed in the explanantion of the hardware.

After running through the same signal flow as the mono patch, the interface runs into an 'octophonic dial'. By turning this, users can position the sound around the octophonic array. A 'Spread' control is placed below this. This enables you to make the sound come from all speakers at equal volume (value 0 on the spread dial) or be quite directional, mainly coming from one speaker (value 1) and also in between those points (values 0-1).

Mathematics for the 'octophonic dial'

The next sections describe the software, signal processing and hardware implementation of the octophonic interface.

For this patch, the octophonic dial has a value of 1024. This value is essentially arbitrary, however, it was originally chosen as it has enough steps to ensure that there are no aural jumps between speakers. The value between 0-1024 will be called 'x'. 'i' will equal the result of the previous equation. The equations below explain each step of the process. It is important to note that this math is applied to all eight outputs of each channel and adjusts the volume of those individual outputs (speakers) to move

the sound around the circle:

> $x - 64$ (to offset the ring by half the space between two speakers so that it is set up as a flat octophonic system)
>
> $i - \pi/1024$ (to take the value between -64 - 960 and convert to a value of pi to work in radians. Is not 2π, as cos^2 has two periods in 2π and only one in π)
>
> $cos(i - (speaker\ number\ *\ 1/8\pi)$ (cosine is used because $cos(x) = cos(-x)$. This makes our output between values -1 to 1. The 'speaker number' variable is converted to terms of π, so that it can be adjusted based on the specific speaker it is adjusting)
>
> i^2 (converts our value to a value between 0-1)

At this point, we obtain a float value between 0-1 by adjusting our 'octophonic dial'. This value then runs into an equation in which we also take into account our 'Spread' value (a float value between 0-1). The math for this 'Spread' value will be explained, and then it will be explained how they work together. will be used to represent the 'Spread' value and its changes, in the same way has been used:

> $1 - \sqrt[3]{1 - f}$ (this equation sharpens the curve of the value between 0-1, rather than being a linear change. The cube root is arbitrary, however it is what sounded the best)

These i and f values are then used together to adjust the volume of the eight different speakers:

> $(i - 0.87) * f + 0.87$ (the 0.87 value is used to equalise the volume across the speakers when the 'Spread' value changes. This value was found by trial and error and what 'sounded best'. It is included in the first part of the equation to balance out the addition of 0.87 in the last part of the equation)
>
> $i * 127$ (multiplies the result of the previous equation by 127, as the volume/gain sliders in MaxMSP work with values between 0-127)

Assistance from Christopher Milson, an undergraduate mathematics student at Victoria, was used to obtain the above equations, and the end result was a collaborative effort.

Hardware interface

The next section details the physical construction and layout of the hardware interface.

While the MaxMSP patch is designed with eight channels, the physical interface

only has two. This was done for multiple reasons. Firstly, the Teensy 3.5 didn't have enough pins to connect to for all the parts needed to complete a full eight channels. There were limited parts available to work with and limited money to spend on more parts. Time was also another constraint, as to complete and test a full 8-channel version of the board would have taken too long, if allowing time for mistakes and fixing errors.

The lack of pins on the Teensy 3.5 is also the reason why there are not as many potentiometers on the physical interface as there are dials on the software interface. Both the reduced number of channels and reduced number of potentiometers has meant that solutions had to be created to be able to physically manipulate all parameters.

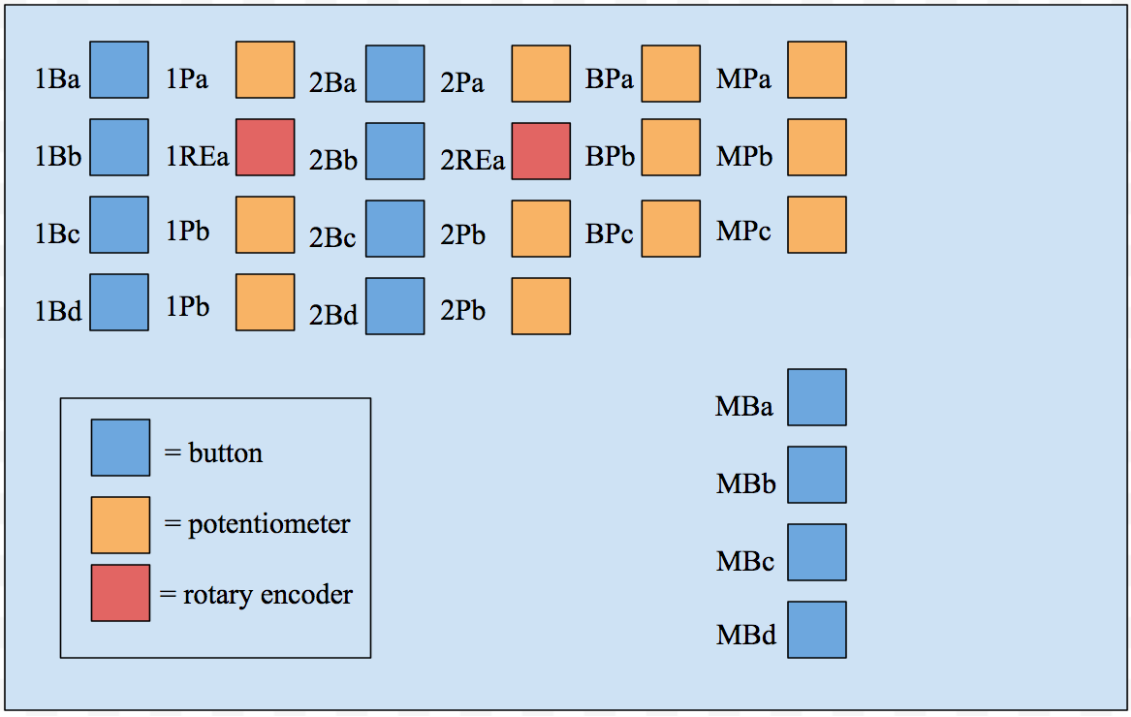

Figure 3. A model of the physical interface.

If button MBa is pressed, it sends a message to MaxMSP (the way it sends a message, and the other details surrounding this, will be explained later), which switches between channel groups. This goes between channels 1-2 (which are active on the image of the interface in figure 2, as shown by the 'light'), 3-4, 5-6 and 7-8. Also, if 1Ba or 2Ba are pressed it will cycle the potentiometers 1Pa and 2Pa respectively between effecting 'Sample Speed', 'Freq.', 'Q' and 'Dry/Wet' on the software interface. Like the channel switching, the light will change depending on which dial is active.

This solution saves time and money and solves the issue of not having enough pins on the Teensy. The main problem that comes with this is that when you move a dial after switching channels or dials, the value will jump to wherever that potentiometer's last value was. This could potentially be solved within either the MaxMSP patch or the Arduino code, by ramping the value to slowly move to where their potentiometer is. But this could be a complicated solution that would still not 100% solve the problem.

Figure 4 shows the physical interface. The case was collaboratively designed and 3D printed with Victoria University computer science undergraduate student, Joshua Hylton. The open aesthetic of the case fits the bare design of the board and lack of caps on top of the buttons, potentiometers and rotary encoders. The board on which everything is connected has been hot-glued to the case and when in use feels sturdy, providing a robust platform for performance.

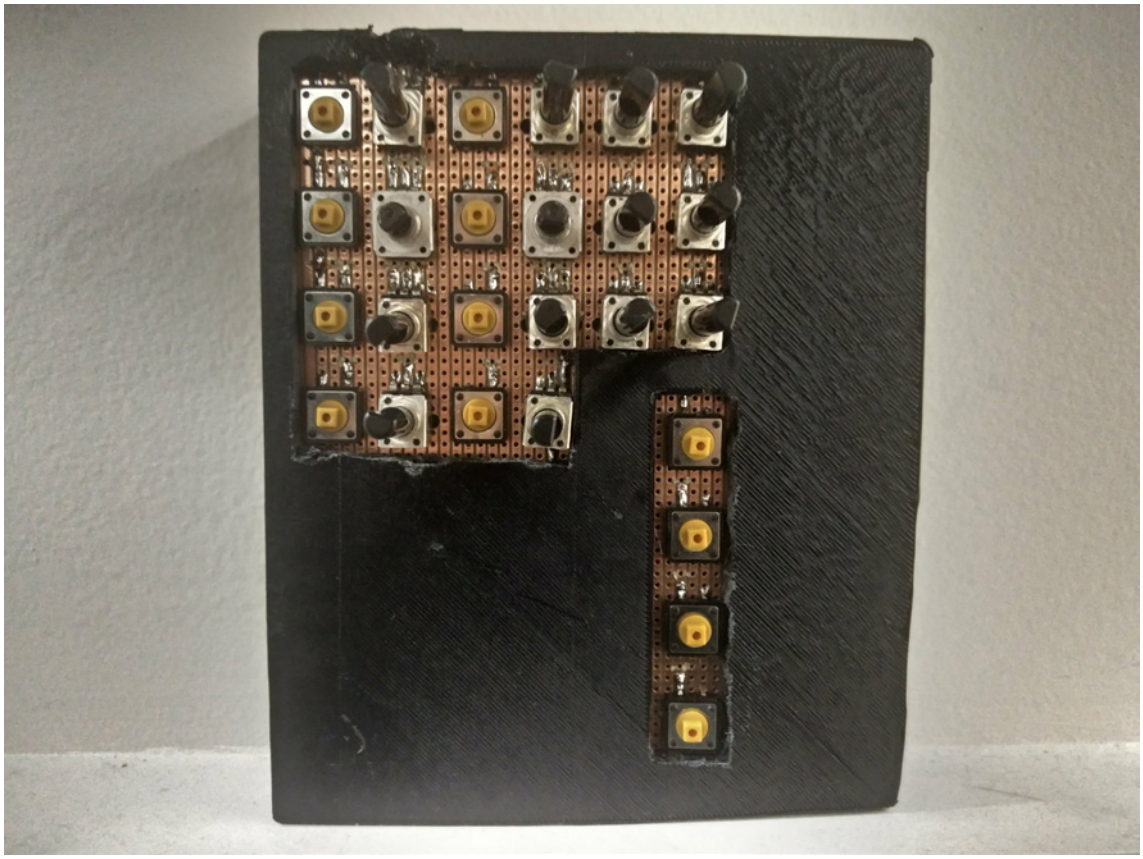

Figure 4. *elle - an octophonic drone interface* by Jesse Austin, hardware interface.

The parts used in the physical interface are as follows:

- Teensy 3.5
- 12 potentiometers
- 12 buttons
- 2 rotary encoders
- Veroboard

The following schematic (Figure 5) details the layout of the board. It is important to note that while all potentiometers have been connected to analog pins, there was not a model of the Teensy 3.5 on Fritzing, so the buttons, potentiometers and rotary encoders are not connected to the same pins as the they are on the Teensy, however, the general layout explains what is necessary to understand.

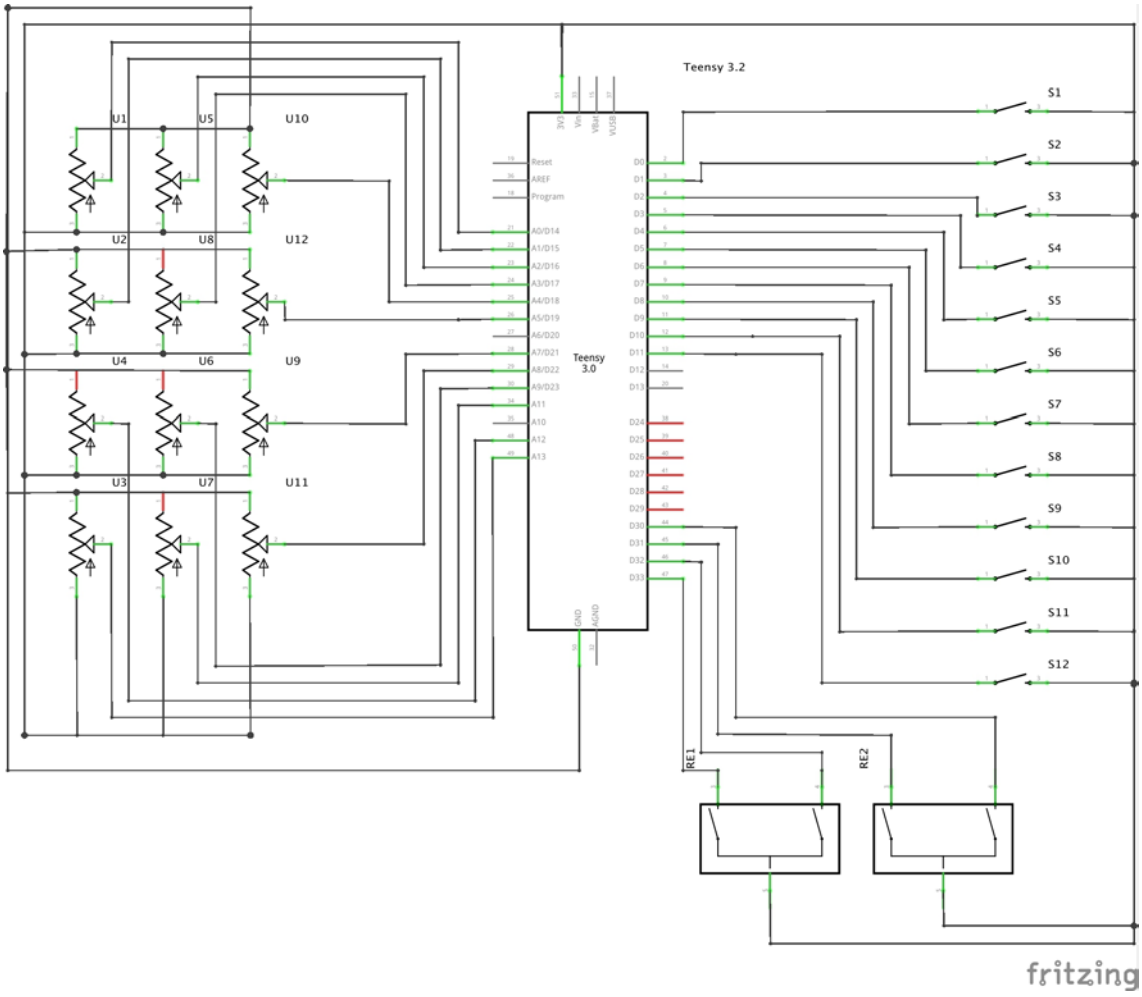

Figure 5. Schematic of *elle*'s hardware interface.

Function/purpose of sensors

Buttons are used to send bangs within MaxMSP. The potentiometers are used to send values between 0-127 to the 'Sample Speed', 'Freq.', 'Q', 'Dry/Wet', 'Spread' and 'Volume' controls. The rotary encoders are used to send values to control the 'octophonic dial', as they can turn 360° and are continuous.

Arduino code

In the following passage the arduino code, and how that code takes the information received from the sensors and communicates that with MaxMSP, will be detailed.

Buttons

The buttons use state change detection methods to change a value when pressed.

```
if (button1Channel0neState != lastButton1Channel0neState) {
  if (button1Channel0neState == LOW) {
    button1Channel0neCounter++;
    if (button1Channel0neCounter == 2) {
      button1Channel0neCounter = 0;
    }
  }
} else {
  button1Channel0neCounter = x;
}
```

Figure 6

This code compares the current state of the button (whether it is pushed or not) with the previous state. If it is different, it will then see if the state is 'LOW' or 'off' (in most cases, this would usually be 'HIGH'), and the Teensy's inbuilt pullup resistors are then used on each pin, instead of a hardware resistor, to work the buttons. In 'void setup' the pin number was set to INPUT_PULLUP instead of INPUT. If it is 'LOW', then it adds to a counter. If the counter then becomes '2', it sets the counter to 0. This means that a push of the button alternates the value between 0 and 1. If the state hasn't changed, it sets the counter value to 'x', which will be explained further on when talking about communicating with MaxMSP.

Potentiometers

State change detection methods are used to get values from the potentiometers.

```
if (effectChannelOnePotOne != previousEffectChannelOnePotOne) {
  previousEffectChannelOnePotOne = effectChannelOnePotOne;
} else {
  effectChannelOnePotOne = x;
```

Figure 7

If the potentiometer value has changed from the previous value, the previous value is then made equivalent to the current value to then notice another change. If the values are the same, then it sets the potentiometer value to 'x'. The potentiometers read a value between 0-1024, that has been mapped down to 0-127, as this is MIDI resolution.

Rotary encoders

```
if (newRotaryChannelOne != positionRotaryChannelOne) {
  positionRotaryChannelOne = newRotaryChannelOne;
} else {
  newRotaryChannelOne = x;
}
```

Figure 8

The rotary encoder code is set up the same way as the potentiometer code. The Teensy rotary encoder library (PJRC) is used, which means fewer arguments have been included in the code. A full rotary encoder rotation gives 96 values, and the rotary encoder provides infinite positive and negative values.

Serial print and 'x'

All the individual pieces of data of all the potentiometers, buttons and rotary encoders are printed in a long line and are printing continuously. The firmware sends this serial information to MaxMSP as ASCII, where it is unpacked, converted back to

serial and distributed to the correct parameters to control.

To print in a line and have individual printing channels (as opposed to just one) every sensor must print at the same time. The issue is that if MaxMSP is always receiving information, it will be constantly changing and adjusting parameters, and when using 'bangs' this becomes an issue. To combat this, an 'x' has been set up inside arduino as a character to equal 'a', because this character has a serial value of 170. Inside MaxMSP, every channel that is unpacked has been set to not pass on value 170 (only the rotary encoders send 170, but it is relatively inconsequential to remove that value).

Related works

Accepting the lack of ability of the MaxMSP patch to spatialise in a new way means this work somewhat ignores Johnson's (2017) interest in creating new spatial aesthetics through new interfaces, and could be considered a critique of my work. Johnson's research "stems from artistic practice and a desire to deeply explore spatial aesthetics in sound art" (p. V), and this is where our main intentions differ. While exploring spatial aesthetics in sound art is still an aim in my work, I want to do so within the confines of the interface I have designed. Exploring how to spatialise sound differently was not my intention, rather, it was being able to implement and create a working interface for live spatialisation. Further, this work promises much progress on subsequent multi-input hardware interfaces, a topic left unexplored by Johnson.

Darren Copeland's *NAISA Spatialization System* explores sound beyond the two-dimensional design of Johnson's interface, allowing the user to use physical gestural control to manipulate sound by using all three dimensions, as well as tilt, roll and directional control. The physical controller, like mine, controls a MaxMSP patch. This interface creates and offers a new way to explore physicality within spatialisation, as demonstrated in his 2015 performance at MANTIS festival (NOAVARS, 2015).

In contrast to Copeland's work, one could mention the diffusion of stereo works for multiple stereo pairs. The technique, as described by Smalley in an interview with Larry Austin (2000), was explored in the CMPO 210 – Projects in Electronic Music course. The process involved using four groups of faders to control four stereo pairs of speakers in an octophonic array and performing a live spatialisation of a stereo work.

These three interfaces and methods, as well as *elle*, demonstrate the wide variety of options there are in performing and spatialising sound, but also show that there is a lot of room to further innovate in the way we approach creating interfaces for spatialisation of sound.

Collaboration

Collaboration was a large factor in completing this project. The nature of academia makes you realise how little you know, and with the short timeframe that undergraduate courses have, there was little opportunity to expand upon mathematics and 3D modelling skills.

To work on the equations, Christopher Milson and I sat with an octophonic array, trying different equations to reach the end result. The collaboration involved me describing to him, in abstract musical and sound terms, what I was trying to achieve, and him interpreting that mathematically and implementing it into the visual code until we found something that aesthetically suited the aim.

Creating the 3D-modelled case involved measuring the vero board with the sensors and collaboratively deciding how we would position the case around the unit. With his experience of 3D modelling and printing and my direction, we were able to conceive a structure together that fits and houses the unit in a robust and sturdy fashion.

Future work

I would like to make improvements and adjustments to the current interface that would include completing a full 8-channel hardware interface that mimics the design of the GUI, fixing the solo channel function inside of the MaxMSP patch (it only allows one channel to be soloed at once), including some switches to shift between different speaker configurations (whether that be speaker routing or number of speakers) and replacing the buttons used with quieter ones.

Beyond this, it would be interesting to take inspiration from Johnson, and use the mathematics that I have worked out for this to try and create new interfaces for spatialisation that enable me to explore different spatial aesthetics, or use the mathematics in the development of my own creative practice.

Bibliography

Austin, L. (2000). Sound diffusion in composition and performance: An interview with Denis Smalley. *Computer Music Journal*, *24*(2), 10-21.

BBC News. (2008, August 1). Early stereo recordings restored. Retrieved from http://news.bbc.co.uk/2/hi/technology/7537782.stm/

Copeland, D. (2014). NAISA Spatialization System. Retrieved from http://www.darrencopeland.net/web2/?page_id=400/

McKinnon, D. (2015). Lost Oscillations. Retrieved from https://dugalmckinnon.com/2015/11/10/lost-oscillations/

Johnson, B. (2017). *Designing dynamic systems for advancing aesthetic spatial engagement in sound art*. (PhD thesis). Wellington, NZ: Victoria University of Wellington.

NOVARS Research Centre. (2015). MANTIS Spring 2015: Darren Copeland and the NAISA Spatialisation System. Retrieved from https://vimeo.com/120937533

PJRC. (n.d). Encoder library. Retrieved from https://www.pjrc.com/teensy/td_libs_Encoder.html

Author

Jesse Austin-Stewart has just completed his Bachelor of Music, majoring in composition, specialising in sonic arts, at Victoria University of Wellington. Heading into postgraduate study, Jesse intends to explore the relationship sound has with space in various capacities.

Course information

CMPO 381 is a course run by Dr Jim Murphy at Victoria University of Wellington, in which the objective is to design a custom digital interface for musical expression. The course focuses on the use of microcontrollers to control computer-based music software and different sensors to vary input data.