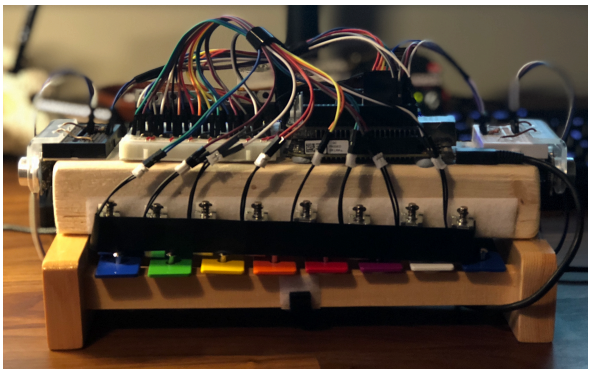
Candidate: Samuel Bilbow (148508)
Module: Generative Art and Musical Machines
Project: Therephone

Report: Therephone

an interactive xylophone robot



Introduction

For this project, I have created and engineered a xylophone that responds to movement when placed in any room environment. Using two HC-SR04 ultrasound sensors, 8 actuating 5V solenoids and a PureData patch containing the control logic, the distance that someone is from either side of the xylophone affects both the pitch and tempo of the played notes on the xylophone. Inspiration for this project stemmed primarily from my long-standing interest of the Theremin, but also from watching Wintergatan's demonstration of their 'Marble Machine' (Wintergatan, 2016). Both of these machines combined in my head to make a robotic mechanical theremin which I have called the Therephone

The Therephone succeeds to fulfil the criteria of a musical robot by being active (it contains eight actuating solenoids, that move freely without direct human input), responsive (the two ultrasound sensors allow the machine to sense directional distance data and respond accordingly), musically intelligent (it contains internal pattern generation processes within PureData and C++), and social (it can interact with humans on an instrument level, but also on an installation level with any moving objects within the throw of the ultrasound sensors).

Background and Context

I started my research for this machine by reading Ajay Kapur's "A History of Robotic Musical Instruments" (Kapur, 2005). In the paper, Kapur details the history and approaches to creating different types of musical robots, including percussion robots, and within that section, idiophones. Mentioned, is the approach of using electromagnetic plungers and solenoids to strike the keys of instruments such as xylophones. Looking deeper into the use of solenoids with idiophones lead me to discover the works of Gerhard Trimpin, particularly his Conloninpurple installation. In this installation, Trimpin creates a "7 octave instrument with wooden bars and metal resonators", the bars are actuated by electromagnetic plungers which are in turn controlled via MIDI (Willey, 2012).

Wintergatan's Marble Machine (both the machine, and song named after the machine), provided me with a lot of inspiration, and showed me the potential for how big autonomous musical machines could be. The use of marbles intrigued me, however I had already decided on using solenoids, as I believed they would lead to less hassle in the development stage, there would also have to be quite a large marble replacement system.

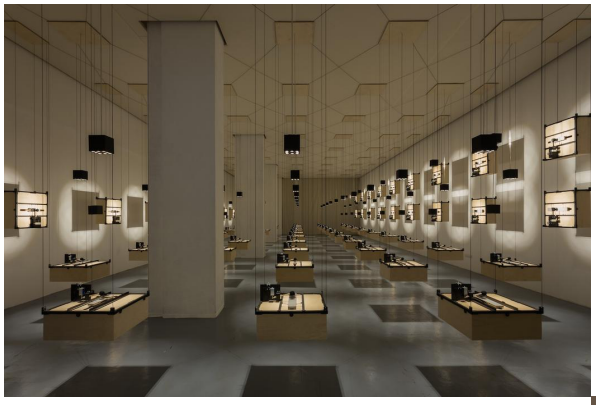


Figure 1a: Andrius' Installation

Finding Andrius Sarapovas' kinetic generative music installation at Technarium solidified my plan to use solenoid based actuation. In this installation, 77 iterations of a solenoid and damper system on different length xylophone keys hang from the ceiling and walls (TWMW, 2017). Andrius uses upturned push solenoids to drop beaters on each key, and a similar mechanism to damp the key. The input data to the system is all of the 4G connections across the different regions of Lithuania. A pattern generator / algorithm turns turns the data into different length notes, with different parts of the installation correlating to the 4G usage in the four corners of Lithuania. Andrius said that "statistics *draw* the map of reality, making it two-dimensional, and the effect of music is directly contrary", he says that the installation makes it easier to spot unseen correlations in the data due to it being three dimensional

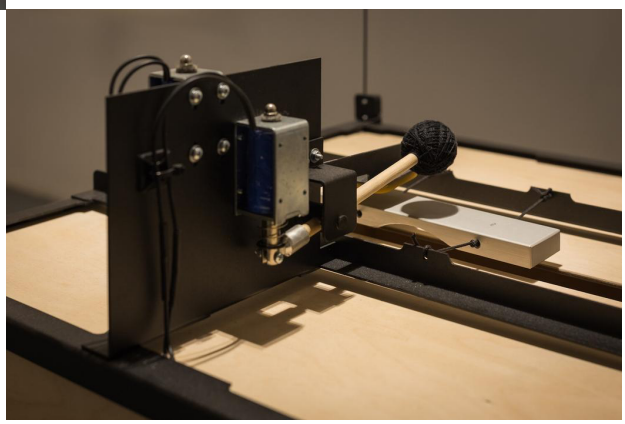


Figure 1b: Solenoid/beater mechanism

Methodology

After learning about various solenoid actuation techniques, I decided to use the push approach, and drafted a Google Sketch Up concept of what I thought the Therephone would look like. This helped me throughout the development process and even though I could make changes, the simplicity of my design has carried on to the final machine, which looks very much like the beginning concept.

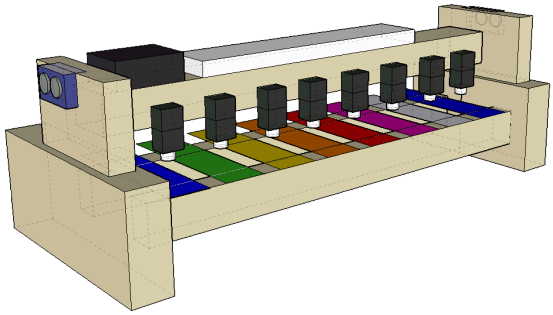


Figure 2: Therephone concept

Electronics and Construction

After learning how to construct a basic solenoid circuit in week 1, I developed a plan on Fritzing for how to fit 8 iterations of the circuit onto a half length breadboard that would in turn fit on the top of the central beam along the top of the xylophone. Buying, sawing and connecting the central beam onto the xylophone was the next step, and this is where the design considerations in the concept came to use, as I already knew the size and position I wanted the beam to be in. After the xylophone and beam were connected, it was just a matter of fixing the breadboards and Beaglebone/Bela to the top of the beam, and letting the solenoids rest on the same surface until I had sorted out the basic PureData patch. After the patch was finished, I decided to use a Velcro strip to attach the solenoids to the beam. I found this to be the best solution, as it allows the user to change the height of the solenoids at will. Hot glue meant that the solenoids couldn't be adjusted, and tape was not strong enough.

Pure Data and C++

I decided to buy two HC-SR04 ultrasound sensors for the machine inputs, as they are the most supported ultrasound sensors with the Bela system with there actually being an example C++ render file that you can use and tweak, and there also being a fair amount of online support for the sensors. These I decided to mount straight away, onto the sides of the xylophone for testing.

The next stage of development required me to learn skills outside of the module's scope. After posting to the Bela forums asking what the most effective way of extracting sensor data from the sensors was, I managed over a couple of weeks, with a lot of help from Giulio Moro, PhD researcher at Queen Mary University of London, to adapt the example render C++ render file for the sensors to include a 'send' to the libpd part of the Bela project (Bela Forums, 2017). Figure 3a shows the line of code that creates a libpd send. This line of code is

```
void render(BelaContext *context, void *userData)
{

    for(unsigned int n = 0; n<context->digitalFrames; n++){
        gTriggerCount++;
        if(gTriggerCount == gTriggerInterval){
            gTriggerCount = 0;
            digitalWriteOnce(context, n, gTriggerDigitalOutChannel, HIGH); //write the status to the trig pin
        } else {
            digitalWriteOnce(context, n, gTriggerDigitalOutChannel, LOW); //write the status to the trig pin
        }
        int pulseLength = pulseIn.hasPulsed(context, n); // will return the pulse duration(in samples) if a pulse just ended
        float duration = 1e6 * pulseLength / context->digitalSampleRate; // pulse duration in microseconds
        static float distance = 0;
        if(pulseLength >= gMinPulseLength){
            static int count = 0;
            // rescaling according to the datasheet
            distance = duration / gRescale;
            libpd_float("UltrasoundDistance", distance);
            //rt_printf("Sending distance to pd\n");
            if(count > 5000){ // we do not want to print the value every time we read it
                rt_printf("pulseLength: %d, distance: %fcm\n", pulseLength, distance);
                count -= 5000;
            }
            ++count;
        }
```

Figure 3a: sending sensor data to libpd

duplicated for the second sensor. The code allowed me to create a [r UltrasoundDistance] object in PureData that streamed the sensor data into the patch.

After the *input* part of the patch had been implemented, I decided to work on the *output* before the *pattern recognition*, so that I could monitor from the start what the effect on the solenoids was, instead of working on pattern recognition blindly. Figure 3b shows my original test patch that I used to actuate the solenoids. The complexity of the patch increased exponentially as I thought of new ways of processing the data. This was great, as I had so many ideas of how to use PureData to reach the goal that I wanted. However, as I quickly realised, implementing changes without testing them, and then implementing new ideas, was a very easy way to get confused by my own code. This happened by the 5th patch iteration. Luckily, I 'Save As' at each change (Figure 3c), so I did manage to find the part of the code that was misbehaving. Another issue was that I decided to create PureData abstractions way too early in my understanding of my own patch. This lead to further confusion, and is something I rectified by the sixth iteration.

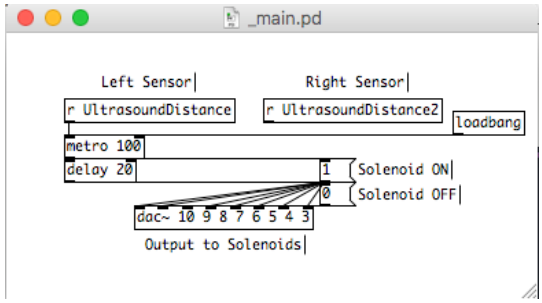


Figure 3b: Original test patch

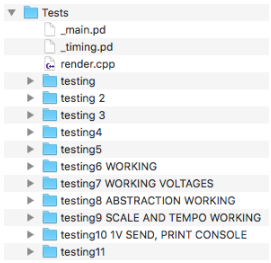


Figure 3c

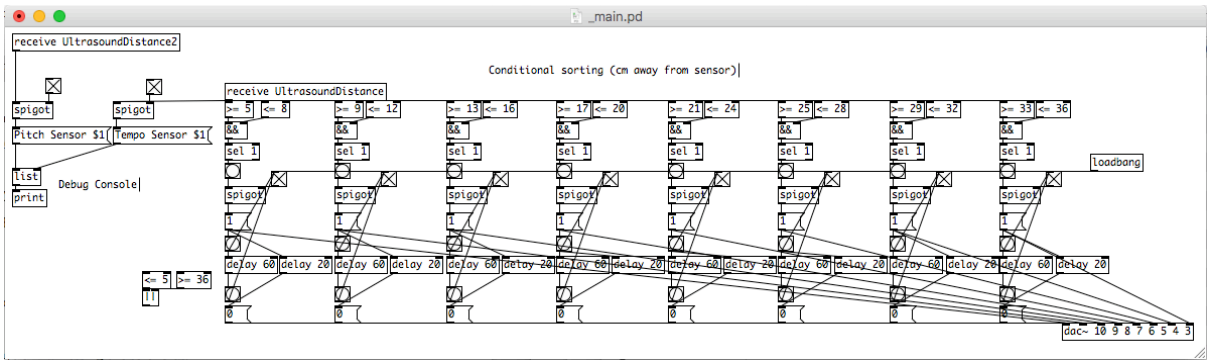


Figure 3d: Fifth iteration of PureData patch after unpacking from abstractions

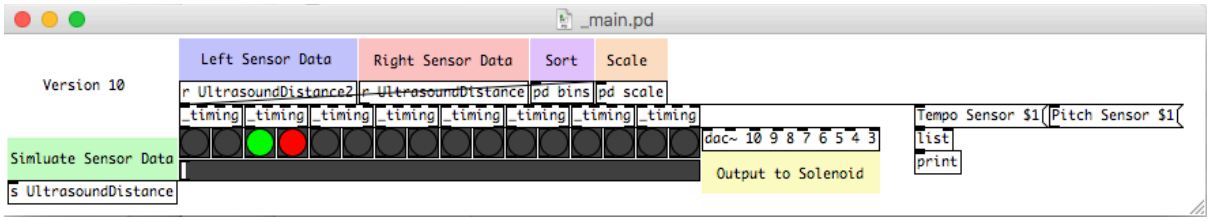


Figure 3e: Tenth iteration of PureData patch containing abstractions, subpatches and simulation

Shown above is the tenth iteration of the patch. It contains a console printing section, that gives me the ability to see any sensor inaccuracies in the Bela IDE console. By this stage, I had figured out that the 1 and 0 messages that I was sending the solenoids was actually the voltage being given to them via the analog outputs (and by extension the 12v power supply via a transistor). Commenting my code allowed me to clearly see what effect the process was having on the data as it passed through the patch. It actually took writing the patch out on paper, and starting the PureData patch from scratch for me to realise the that the 1 and 0 message was the voltage. The tenth iteration also contains a simulator. The horizontal slider acts as the a hand moving toward and away from the sensor. This allowed me to test offline if the [_timing.pd] abstraction was working, as it gave visual feedback for when each solenoid would be triggered.

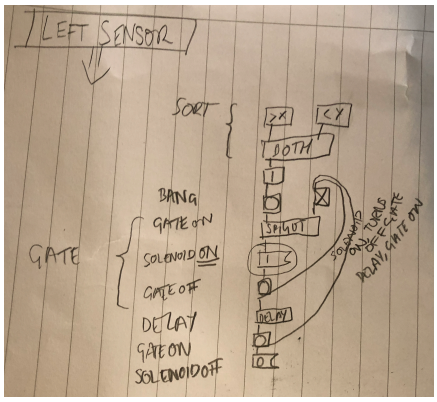


Figure 3f: Paper patch

The pattern recognition in the PureData patch is shown in Figure 3g. The data undergoes conditional sorting, the centimetre increments are found in the [pd bins] subpatch. At initiation, the [>=] and [<=] objects for each iteration of [_timing] are filled with these numbers. When the right sensor reports back a distance within one of these 'bins' or ranges, the respective solenoid receives a voltage dependent on the message box under the [spigot] object, and then sends 0 to the solenoid after 20ms. As soon as as this happens, the gate activates, shutting off all subsequent sensor data coming into the abstraction. After a delay of 1000ms (by default), the gate deactivates, and the abstraction will take data from the sensor again. The gate delay is changed by the distance perceived by the left sensor, thus essentially imposing a tempo or rate of actuation on each key dependent on that distance.

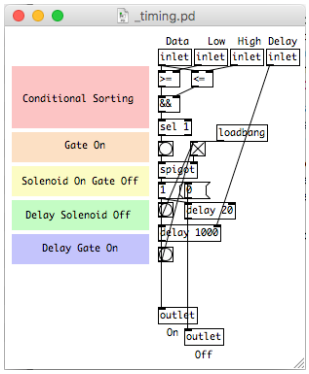


Figure 3g: [_timing]

The scaling of each sensor can be changed easily. For the right sensor (pitch), the [pd bins] subpatch contains the centimetre ranges. For the left sensor (tempo), the [pd scale] subpatch contains a linear scaling object [* 23], which can be changed to allow for a larger range of tempos. This ability to change the scaling allows the Therephone to act as an autonomous robot in an installation, but also an instrument if the scaling amount is reduced to a matter of

tens of centimetres. A smaller scale means that the scope received by the sensors will be within a persons arm span, essentially giving the user the ability to play the Therephone like a mechanical Theremin. As the sensors have a maximum throw of 4 metres at 30°, the patch can be tweaked to suit most installation sizes, and the voltage can be changed as well, meaning that the volume of the machine is variable.[1]

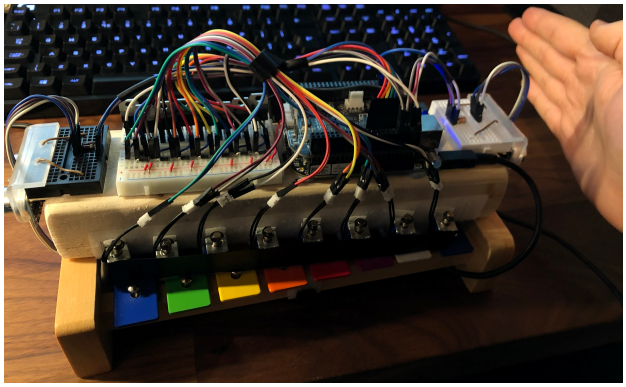
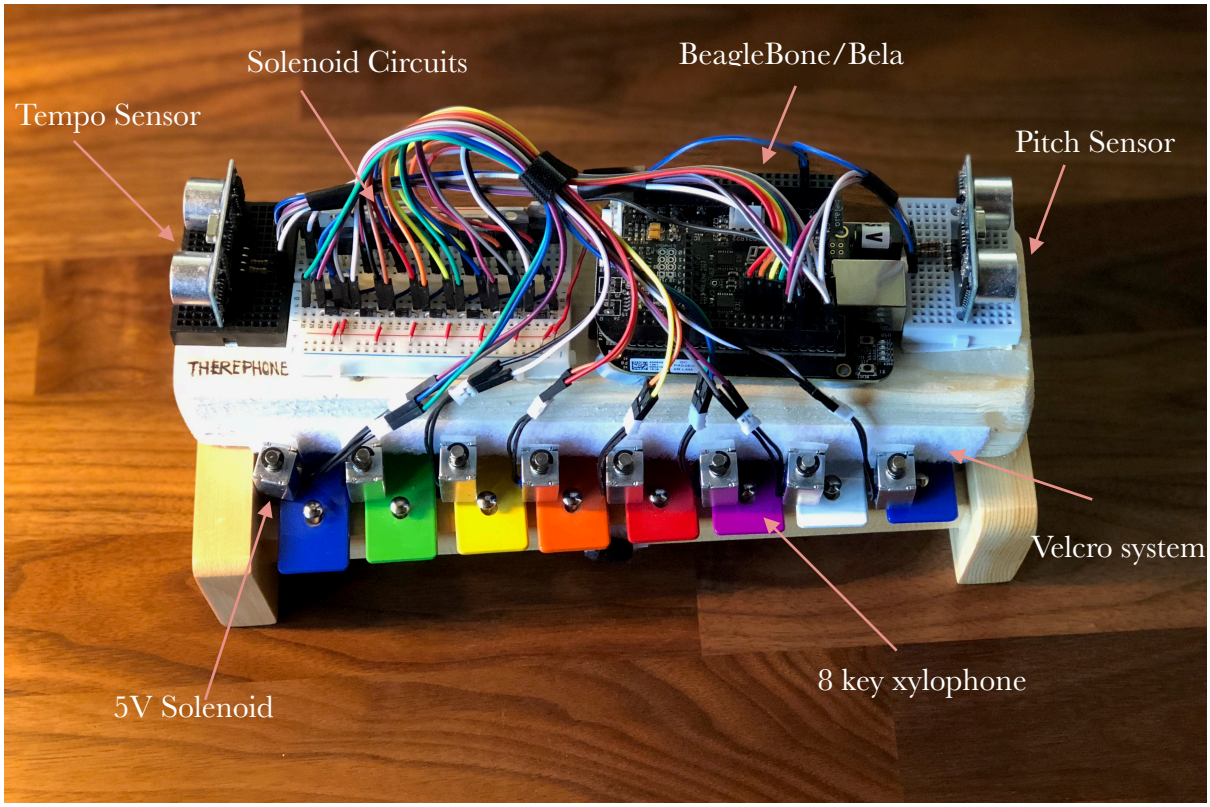


Figure 3h: Instrument use of Therephone

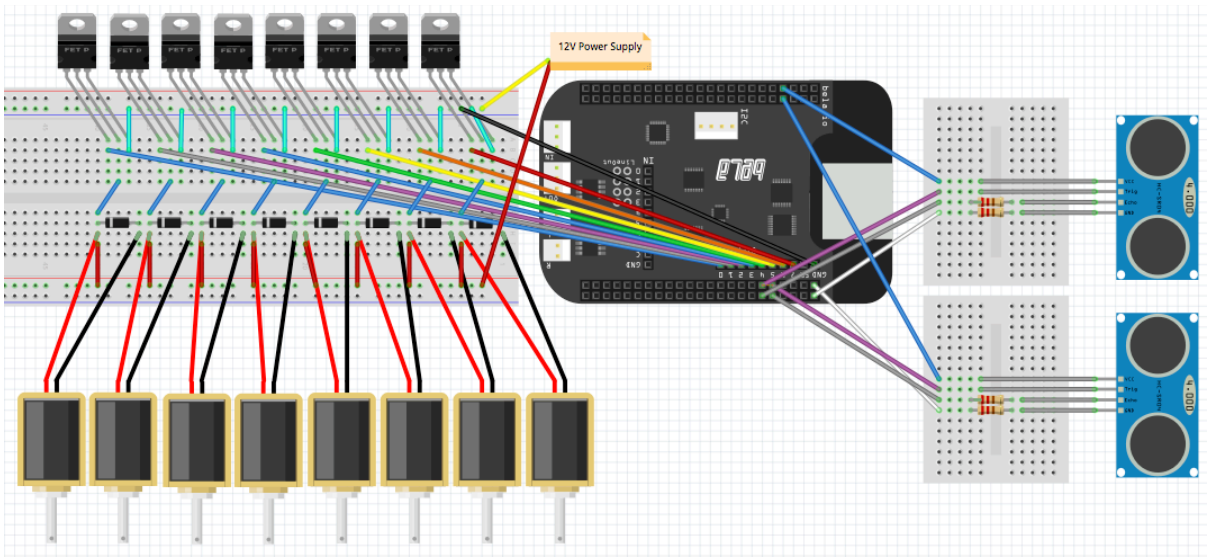
Demonstration

With my submission I have attached two videos showcasing the Robot and Instrument modes of the Therephone. Below is a full annotation of the machine.



[1] N.B. When changing the voltage, the user will have to raise or lower the solenoids using the Velcro system, otherwise the pins might not reach the xylophone keys, or rest on them, both cases resulting in a different sound that I have intended.

Below is the Fritzing depiction of the wiring and circuits if not visible in the photos that I have provided.



Evaluation

The Therephone fulfils the musical robot classification of being active, responsive, musically intelligent, and social. It particularly succeeds in being responsive and active, with precise sensors and an array of actuating solenoids. It is versatile as well, it is able to be played as an instrument as well as being able to flourish in an installation scenario.

The Therephone is not without fault or room for improvement however. Although the sensors are accurate up to 4 metres, due to them being ultrasonic, they need to "hear" the echo from the sent wave, and this will only happen if the surface is directly facing it. This means that in some circumstances, the Therephone will occasionally actuate a group of keys if the sensor does not receive a signal back. I am still trying to implement a solution to this problem. Part of the problem is due to the Bela system overloading when the sensor does not hear the echo back. Also related to the issue of ultrasound sensors, is that the object obstructing the field of the sensor needs to be as wide as the 30° throw. This can be calculated with the below equation where x = the distance from the sensor:

$$Width\ of\ object\ must\ be = 2x\left(\frac{\sin(15)}{\cos(15)}\right)$$

At 1 metre, this means that to receive the correct response from the xylophone, the object needs to be ~50cm wide. Upon testing the Therephone, this hinders the performance quite drastically, as random actuations start to occur at distances further than 30cm. Improvements in the near future for the Therephone include implementing a generative "player" mode, where through machine learning, it listens to the movements around itself, and plays a piece

of music derived from those movements. Also upgrading parts, such as a beater system like Andrius' in his 4G installation, and a full key xylophone would improve the machine.

From this project, I have learnt some basic C++ coding, bolstered my own PureData knowledge through the use of new objects and methods of implementation such as libpd, and improved my understanding of electronic circuits and micro-controllers. Most importantly, I have created a musically intelligent robot, and in the process, engaged with practitioners and increased my problem solving abilities.

Bibliography

Bela Forums. (2017). *Help needed involving HC-SR04 Ultrasound Sensors - Bela*. [online] Bela. Available at: https://forum.bela.io/d/418-help-needed-involving-hc-sr04-ultrasound-sensors.

Kapur, A. (2005). A History of Robotic Musical Instruments. pp.3-4.

TWMW. (2017). *A 4G network turned into music*. [online] Available at: http://www.thosewhomakewaves.com/home/2017/8/14/a-4g-network-turned-into-music [Accessed 10 Nov. 2017].

Willey, R. (2012). *Five Pieces for conloninpurple and 12-channel sound system*. [online] Conlon Nancarrow Online Symposium. Available at: http://conlonnancarrow.org/symposium/conloninpurple.html [Accessed 15 Dec. 2017].

Wintergatan (2016). *Wintergatan - Marble Machine*. [video] Available at: https://www.youtube.com/watch?v=IvUU8joBb1Q [Accessed 24 Oct. 2017].