Portfolio of compositions: A dialogue between the human body and computer music

by

TSUN WINSTON YEUNG

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

Department of Music
College of Arts and Law
The University of Birmingham
September 2018

Abstract

This PhD thesis consists of a portfolio of five works of electroacoustic music, an installation, software, and a written commentary created at the University of Birmingham. The primary research focus is on creating electroacoustic music with external sensors and data related to the human body. In both instrumental (acoustic or electronic) music or fixed media, there is considerable space to improve or modify the way humans interact with interfaces or systems for musical realisation. Through the creation of the works in this portfolio, various interfaces were examined in order to investigate if there is any room for improvement when applying them in music creation. These interfaces include non-tactile gesture controller, EEG sensors, live coding, and sonification of DNA data. Possible ways to overcome or work around the limitations when using interfaces were examined, proposed, and used throughout the composition process. This commentary will discuss both the creative and technical processes behind the creation of these works. It also contains software created as an aid to computer music generation.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Scott Wilson for his encouragement and guidance throughout this PhD.

Special thanks to all my supportive friends: Nikki Sheth, Luca Danieli, Konstantinos Vasilakos, and Mon Leung.

Most importantly, thanks to my parents and brother for all their unconditional support and encouragement over many time-zones, words cannot describe how grateful I am for their patience and sacrifices.

Table of Contents

List of Figures

The external media

The attached USB drive consists of the musical works and software for this portfolio. There are two folders inside:

Media: All the media files including video and audio;

Codes: The code for the creation of the works as well as the software discussed in Chapter 3 of this commentary.

Chapter 1

Introduction

This written commentary is a supporting document for the music and other works that I have completed during my time pursuing a PhD at The University of Birmingham. The composition portfolio submitted with this commentary is comprised of live-electroacoustic music, fixed media acousmatic music, and a sound installation. This commentary also discusses some music-related software, which aids computer music creation and performance, that I created during my research.

At the early stage of my research, my main focus was on the integration of music with stochastic processes and live electroacoustic performance. I chose to begin my work in this way because I was intrigued by the utilisation of new technology in computer music creation, which provides many possibilities for how music can interact with the real world. At that time, I was exposed to the world of sensor-based musical performance, including music that uses ready-made devices (e.g., Leap Motion Controller and Electroencephalography headsets) as well as some DIY devices that I made with micro-controllers (e.g., Arduino). My exploration in using sensors and interactive devices has since drawn my attention from stochastic music to how sound can echo the performer's body and vice versa. In the first two works in this portfolio, there are some composition and sound synthesis techniques that involve stochastic processes and interactive devices.

The other works concentrate more on the relationship between the body, body movements and computer music.

A commonality between all of the pieces in the portfolio is that they all involve data or signals that are derived in some manner from the human body. Most of them aim to modify the relationship between the involvement of the human body and musical performance. I am intrigued by this topic due to the fact that in both instrumental (acoustic or electronic) music or fixed media, there is considerable space to improve or modify the way humans interact with interfaces or systems for musical realisation. For example, in the first piece (*(un)touched*), which is a half-improvised piece of music that uses a non-tactile gestural controller, I explore how the instrument (synthesiser) design can change the overall performance. Both instruments or computer interfaces that make music can technically be mastered with some practice, but my question was what will happen if the mechanism of the synthesiser is designed to be very sensitive to input data resulting in it being almost impossible to control even if the performer is experienced with the hardware. In the third piece (*nootherthanthevoid*), which is a live-coding piece with sensors, I explore the possibility of enhancing flexibility and controllability of the sound in a live-coding performance as "the performing of 'thinking-in- action'" (Cocker, 2016, 1). In order to achieve this, I added the use of sensors and programming utilities that can be utilised during live coding when the performer is typing.

In addition to creating the pieces in this portfolio, I have also been participating in a laptop ensemble. Since 2013, I have been a member of BEER (Birmingham Ensemble for Electroacoustic Research). This opportunity offered me experience and knowledge of both live coding techniques and sonification, which I have applied to my own music. Although the work I have done with the ensemble is not included in this portfolio, it is worth mentioning, in particular a project named *Dark Matter* that I was involved during my time in the ensemble, since I gained a lot of experience from it. Appendix B is a commentary about the work I have done with that project.

Tools

All the pieces in this musical portfolio involve the aid of the computer, and some of the software and programming platforms developed for this were used in more than one piece. The main functions of these programming platforms were to gather and process data, transform the data into sonic parameters, and generate sounds and visuals. For data processing and sound generating, I used the music programming language SuperCollider[1]; and for visuals, I used the graphic programming language Processing[2]. Below, I am going to discuss how and why used them.

SuperCollider

All the music in this portfolio is made of sounds that are generated with the computer, either with sonification of scientific data or with software synthesizers. There are

[1] SuperCollider is a free and open-source programming language and Integrated Development Environment for digital audio synthesis and algorithmic composition. It was initially developed by James McCartney in 1996. See https://supercollider.github.io

[2] Processing is a free and open-source programming language and integrated development environment for visual design and new media art. It was first released in 2001 and it was designed by Casey Reas and Ben Fry. See https://processing.org

numerous platforms that are capable of doing these jobs, most notably Max/MSP[3], PureData[4], and Supercollider. Among these software/languages, I chose to use SuperCollider as my main tool because of its versatility, its popularity, and its efficiency in communicating with other platforms. SuperCollider can be used in algorithmic composition as well as live coding. Its functionality can be easily extended by encapsulating custom code into user classes. Also, its server-client architecture makes it a robust system for musical performance.

Processing

Processing was used in two pieces in this portfolio (*Variations* and *nootherthanthevoid*) that involve visuals elements. It is a graphical library and Integrated Development Environment (IDE) based on the programming language Java. Compared to other alternative platforms such as openFrameworks[5] it is a very well documented programming environment with many software examples. In the second piece in this portfolio, *Variations*, Processing was used to display a visualisation of some brainwave signals. It gathered the data received from SuperCollider and then drew shapes with different sizes and colours to indicate which

[3] Max/MSP is proprietary visual programming language for computer music and multimedia initially devised by Miller Puckette in the 1980's. It is currently developed and maintained by Cycling '74 (as of 2019). See https://cycling74.com

[4] PureData is an open-source visual programming language for computer music and multimedia which offers a similar infrastructure like Max/MSP. It was also designed by Miller Puckette. First released in 1996. See https://puredata.info (accessed 2019)

[5] openFrameworks is an open-source programming framework written in C++. It was designed for creative coding and visual design. It was initially designed by Zachary Lieberman in 2005. Currently, it is developed by the author, Theo Watson, and Arturo Castro. See https://openframeworks.cc

part of the head has more active brainwave activity. The communication between the two programming environments was done with Open Sound Countrol[6] (OSC). In both pieces, an external library called oscP5 was used, which is an implementation of OSC in Processing.

Below, I will firstly discuss my musical works in this portfolio and explain the techniques that I applied to them. I will also discuss some of the software that I created in detail, its creation process, and how I used the software in the works.

[6] Open Sound Control (OSC) is a protocol that allows electronic musical instruments and computer to communicate via the network. It is supported by most of the common music software including SuperCollider, Max/MSP, PureData etc. See http://opensoundcontrol.org

Aim of research

The research undertaken in this practice-based portfolio of creative works mainly aims to explore ways to generate sound-based artistic content by experimenting with interactions between the human body and computer, as well as to extend some of the existing musical performance practices involving the use of the body. To fulfill these goals, a variety of tools and approaches such as sonification, hardware and software instrument design were considered. Since each piece in this portfolio has a different approach to try to achieve this goal, I will discuss and evaluate each work individually in the following chapter.

Research enquiry

Although the works in this portfolio are fairly diverse in terms of how aspects of the human body were involved, this practice-led research explores a number of related questions, such as:

- What can be done to modify the way musicians (both composers and performers) and the computer interact?

- What are the limitations of a particular interface (loosely defined), and how can it be extended or augmented to (perhaps partially) overcome those limitations?

As the nature of creative and practice-based research involves aesthetic as well as other forms of enquiry, the results will necessarily be provisional and at times incomplete, but the list above gives some sense of the range of ideas explored herein.

Chapter 2

Works

This chapter presents the commentary for each piece in this portfolio in chronological order. It discusses the background, aims, and the process of creation for each piece in this portfolio to offer some overall insights of this practice-based research.

#1 (un)touched

Year of composition: 2014 - 2015
Duration: 9'43"
Format: stereo
Live electroacoustic for Leap Motion and SuperCollider

Synopsis

(un)touched is the first piece completed in this portfolio. It is a ten-minute live electroacoustic piece for gesture controller, Leap Motion[7] and the audio programming language, SuperCollider. In this piece, I experimented with two different kinds of aleatoric elements, one involving the performance part – which is partly improvisational – and one involving the sonic part which is stochastically generated.

Aim

(un)touched aims to explore the use of a non-tactile gestural controller in live-electroacoustic music and investigate how software instrument design affects difficulty of playing. It tries to answer the questions below:

- What are some of the limitations of using non-tactile gestural controller like Leap Motion as a musical instrument?

[7] Leap Motion Controller is a computer sensor that tracks hand position using infrared cameras with requires no physical contact. See https://www.leapmotion.com

- What can be done to overcome these limitations?

Aesthetic

The title implies the main features of the piece, which involves two meanings of the word "touched": one is being physically touched (the performing part), the other is being modified (the sound generation part). Traditionally, playing with musical instruments or computer devices requires physical contact, which is more intimate between the player and the instrument. The fundamental difference between using conventional instruments and a non-tactile gestural controller like the Leap Motion controller is that the performer is not physically touching any objects. Instead, the controller collects data from the position of the performer's hands. In other words, there is no physical interaction between the performer and instrument, but instead a relationship with space. The action of "controlling" for physical controllers/instruments, depends on the interaction between the performer's body and the instrument such as fingers and piano keys, lips and mouthpiece, and foot and pedals. In the case of using a "touchless" gestural controller, the action of "controlling" involves the interaction between hands and space. This action is similar to sculpting, but instead of on a surface, in the air.

An early example of "touchless" instrument or controller is the theremin, which was invented by Leon Theremin in the 1920's. There are some similarities between the theremin and the Leap Motion controller. The former usually consists of two antennas, one is installed vertically and one horizontally. The antennas sense the distance of the player's hands and control the frequency and amplitude of an oscillator. The Leap Motion controller also sense the distance of the user's hands, but with a totally different technology. It uses two monochromatic infrared cameras and three infrared LEDs to capture data of the hands at almost 200 frames per second and send the data to the host computer. The computer will then analysis and compute the position of the joints of the hands with an average accuracy of 0.7 millimeters (Weichert et al., 2013). While the theremin offers one-dimensional position of the hands, Leap Motion controller provides the position of the hands in three-dimensional.

Musically, this piece is aleatoric in a sense that parts of it happens by chance. Although many performing actions are practiced, the results are different each time it is played. As some of the materials are very sensitive and difficult to accurately control, my approach is instead of actively controlling the sound, I accept the sonic events as they initially happened and modified them to desired results by experience.

Materials

Stochastic processes were brought into sound synthesis by Iannis Xenakis (1922-2001) in the 1970's. The synthesis technique he developed is called Dynamic Stochastic Synthesis which later used in his implementation Gendyn. Since most of the materials in *(un)touched* were produced with this synthesis method, it worth discussing some of its background and concept.

The principle of this technique is to construct sound with repeating sonic fragments on the microsound scale. The concept is similar to wavetable synthesis, but instead of sonifying a static waveform, the waveform changes slightly every time it repeats with random walks. The basic waveform is formed of a user-defined number of interpolated control points which determines the complexity of the produced sound. When it iterates, it distorts both horizontally (duration) and vertically (amplitude) from the previous one.

The horizontal and vertical changes are determined by probability distributions. The sound tends to be noisy if the waveform is less symmetric while it tends to be a pitched tone if the waveform is more periodic. There have been several implementations of this synthesis technique developed on various platforms. For example, iGendyn[8] on iOS written by Nick Collins, and the five Gendy[9] Unit Generators in SuperCollider.

To classify the sonic elements in this piece, I grouped them into two categories according to their controllability and sonic complexity. The first kind is less sensitive to control signals, only few sonic parameters like pitch and amplitude are changeable, which makes controlling them easier and straightforward. Source 1, 3 and 5 are examples of these. Source 2, 4 and 6 are the second kinds that are more chaotic and sensitive to control data. Not only the pitch or dynamic changes but also the timbre transforms from single tone to noise quickly with slight movements, making them more difficult to be accurately controlled and predicted. I will discuss each of them in detail below, regarding their sonic characteristics and how they react to the performance actions.

The reason I chose to use stochastic synthesis in this piece was mainly because of the variety of timbres it can produce. This is ideal for creating the software instruments

[8] iGendyn is an iOS application that implements the Dynamic Stochastic Synthesis developed by Nick Collins. It offers an interface that responds to multitouch and accelerometer inputs from iOS devices. See https://composerprogrammer.com/iphone.html

[9] Gendy1 to Gendy5 are 5 implementations of the dynamic stochastic synthesis generator in SuperCollider. It was written by Nick Collins. See http://doc.sccode.org/Classes/Gendy1.html

mentioned above. With some mappings, a sound can change vastly or slightly according to the parameters.

Source 1

This sound changes between low rumble to a high screaming and screeching sound, sometimes dampened and sometimes enhanced in timbre. With the presence of the left hand, the choice of probability distribution[10] for the amplitude and duration variation changes, which broaden the frequency spectrum. This was created with the `Gendy1` Unit Generator in SuperCollider. The minimum frequency and maximum frequency are modulated by individual `Gendy1` Unit Generators. It is mainly controlled by the right-hand. The roll motion of the hand determines a parameter for the shape of the probability distribution for the durations between control points making it changes between lower and higher register, the Y position of the hand determines the number of control points, which affect the brightness, and the Z position determines the de-correlation in the stereo field.

Source 2

This material has a wide variety of sonorities and is very sensitive. It varies from sharp pitched tone to harsh noise. It was created with the `Gendy2` Unit Generator in SuperCollider, which is a variant of the same synthesis technique. In contrast, it is mainly controlled by the left hand. The Y position determines the number of control points; the

[10] Linear, Cauchy, Logistic, Hyperbolic Cos, Arcsine, and Exponential distribution

Z position determines the value of multiplication for the amplitude distribution's delta value; the roll motion determines the maximum frequency parameter.

Source 3

The timbre of Source 3 shares some similarities with Source 1 but instead of time-varying waveform generation, I used a static wavetable synthesis. The wavetable was generated once the code is executed and does not change throughout the piece. It uses a single randomly generated wavetable with Cauchy distribution. To match this source sound to other materials, I added a frequency fluctuation with random walks controlled by the right-hand X and Y positions.

Source 4

This is a variant of Source 2. The number of control points for the waveform construction has only two values, either two or eight, making it either very bright or very damped. Instead of having a fixed value for the minimum and maximum frequency parameter, they are controlled by the degree of yaw motion of the hand.

Source 5

Its sonority resembles a combination of higher string and brass instruments with tremolos. With the gesture similar to turning pages on the right-hand, it reacts with an update of the pitch combination. It was made with six `Gendy2` generators with different frequency constraints. Since the pitch parameter is determined by a random value, the combination of pitches is different every time the sound is updated. While the Z position on the left-hand determines the speed of the tremolo-like effect, the Z position on the right-hand controls the thickness of the texture of the sound.

Source 6

Source 6 resemble a sound of a bee or fly and fluctuates in the middle register. It was made of a combination of two synths panned left and right separately. When the hands move toward each other, the sounds go to the centre, and when the hands move outward, the sounds go to the side. The speed of the fluctuations is varied by the degree of yaw motion of the hands while the Y position controls the pitch.

Execution

Compared to using conventional controllers, using a non-tactile controller feels vastly different. The main distinction is that there is no tactile or visible cue of where the hands should be placed, making it hard to accurately control. At first, I found this one of the limitations of using this kind of controller to make music, as it is hard to reproduce the same sound. After some practice and tests with different code and settings in SuperCollider, I found this limitation may be especially useful for this piece given the random nature of the sound materials, as this adds some level of unpredictability to this piece.

Data mapping and manipulation

In order to utilize the data from Leap Motion, a piece of third-party software called Geco[11] was used. It transforms the positions of the joints[12] of the hands to 40 different easy-to-access values and forward them via MIDI[13] or OSC. The values used in this piece include the open and closed states, the positions on the abscissa, ordinate and applicate axes (X, Y, and Z), the degree of pitch, roll, and yaw. It sends the data to SuperCollider via OSC.

In SuperCollider, for the sake of convenience of data mapping, I allocated 42 control buses, which all the received data streams were mapped to. Figure 1 displays an example of SuperCollider code that shows the buses and how the OSC data were mapped. There are open and closed versions for each value (e.g. open left-hand Y position and closed left-hand Y position). To maintain a continuous data stream, I sum the values for both open and close hand gesture and mapped them into a single control bus. For example, ~lOY returns the vertical position of open left-hand and ~lCY returns the vertical position of closed left-hand, ~lY returns the sum of the two and ignores the state of the hand. With this arrangement, the data can be easily mapped with a simple ".kr" method in any sound

```
["~lY", "~lOY ", "~lCY"].do{|key| (key ++ " = Bus.control").interpret};
OSCdef(‘osclOY’,{|msg| ~lOY.set(msg[1]);~lY.set(msg[1])},"/left/open/posy");
OSCdef(‘osclCY’,{|msg| ~lCY.set(msg[1]);~lY.set(msg[1])},"/left/closed/posy");
```

Figure 1 SuperCollider code example for control bus creation and OSC data mapping

generation function.

[11] Geco, developed by Uwyn , a software that transforms Leap Motion data and forwards via MIDI, OSC, and CopperLan.

[12] The three joints among the bones on each finger, Distal phalanges, Intermediate phalanges, Proximal phalanges, and Metacarpals, only the first three bones were count for the thumb, and the Carpus (wrist).

[13] Musical Instrument Digital Interface is a protocol, and interface for transmitting digital musical command.

```
{SinOsc.ar(freq: ~lY.kr * 400 + 400)}.play
```

Figure 2 Mapping vertical position of left-hand to frequency

For example, the above code (figure 2) shows a sine wave oscillator with its frequency controlled by the left-hand Y position (open or closed are ignored). Since most of the received data are in the range of 0 to 1, it was simple to map them to any output ranges by multiplication and addition. In this example, the value is scaled to a range of 440 to 880.

Signal chain

`NodeProxy`[14] was used to build the main audio chain between sound materials and processing units. Figure 3 illustrates the signal chain for this piece. Each of the six source sounds has its own node proxy definition (`Ndef` in SuperCollider). They were all mixed into a single node that was placed before a chain of three processing units.

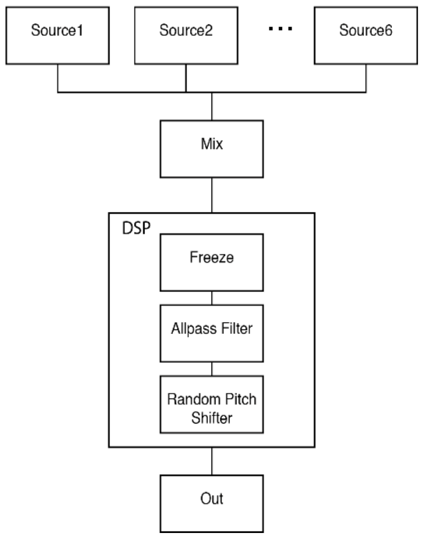


Figure 3 Signal chain of (un)touched

The mix values of the source sounds can be modified individually, as well as the dry/wet values of the processing units. In the process of composing, I tried to use another MIDI controller with sliders and knobs to handle the volume balance of the sources and the mix of the effects. However, after a number of experiments, I found that using another physical controller is somewhat distracting visually. Therefore, I decided to execute the cues (e.g. change of materials, the balance of layers) with either automation or specific gestures. For

[14] NodeProxy is part of the SuperCollider JITLib library (Just In Time programming library). It is a placeholder of an audio node, the contents of which can be re-written/replaced while it is running. Similar classes includes ProxySpace, Tdef, Ndef, and Pdef. See http://doc.sccode.org/Overviews/JITLib.html

example, at 2'13", when the left-hand was moved away, Source 1 and 3 were stopped, and Source 4 entered.

Structure and manipulation of materials

There are roughly five sections in this piece. Figure 4 displays the times of them, and the main elements used therein. For section I and II, the sound was mainly controlled with a single hand, with the aid of the other hand to manage the processing part. Section III serves as an interval between the first two and the last two sections. Section IV is a recapitulation of Section I but with a different development. Lastly, Section V is the ending section in which all the materials except Source 5 are involved.

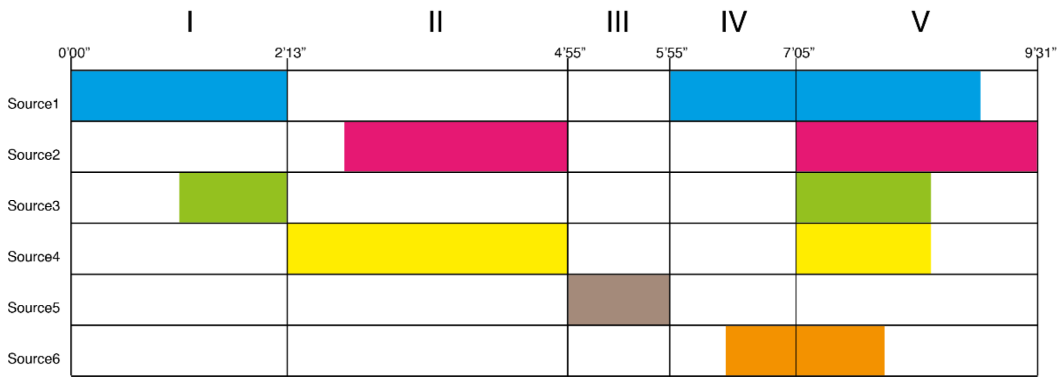


Figure 4 Structure of (un)touched

Evaluation

After creating this piece, I find one limitation of working with this type of non-tactile gestural controller is the difficulty of controlling things accurately because there is no reference point to where the hands should be placed. Although, to some extent, this can be worked around by constraining the controller's values in the parameter mapping process, it is (more interestingly) possible to utilise this limitation as a tool to add unpredictability to the piece.

#2 Variations

Year of composition: 2015
Duration: 9 minutes
Format: 14 channels (The documentation is in stereo)
Live electroacoustic for an electroencephalographic sensor (EEG) and stage performance

Synopsis

Variations is a live electroacoustic piece. It involves sonification of real-time electroencephalographic (EEG) data and computer-controlled stage performance. During the performance, the performer is asked to do a series of actions including meditation, reading, calculating and memorizing numbers, while wearing an EEG headset which read his/her brainwave activity. While the first four actions are ordered randomly, starting from the fifth, the order of the actions is determined by the computer according to the performer's level of concentration and frustration. The EEG data are sonified with various techniques such as audification and parameter mapping. The headset used for this piece provides 14 channels of EEG data stream which corresponds to 14 areas of brainwave activity over the skull. The resulting sonification of each channel is mapped to speakers that correspond to the position of the head.

Aim

The aim of *Variations* is to sonify brainwave activities in real-time while the performer is asked not to be actively controlling the brain activity such as boosting some frequency bands by meditation deliberately. This piece tries to explore the following questions:

- How to reconsider the relationship between the performer and a performance system of this type?
- What are some of the limitations of using EEG sensor?

Background

There have been several musical pieces that involve the use of an EEG signal. One of the most notable examples is *Music for Solo Performer* (1965) by the American composer Alvin Lucier. It involves the uses of amplified EEG signals, which tracks the brain's alpha waves[15] of the performer, to excite percussion instruments while the performer wearing an EEG headset and sits motionlessly on stage (Lucier et al. 1980, p.69). David Rosenboom's *On Being Invisible* is also an example of this kind of music in the 1970's. "On Being Invisible is a self-organizing, dynamical system, not a fixed musical composition." (Rosenboom, 1990). In this piece, the composer devised a purpose-built system that reads the biofeedback of the performer and creates a musical structure by analyzing the data. A more recent example is *Eunoia*[16] (2013), by the New York-based artist, Lisa Park. In this piece, the artist sits among an array of dishes that contain water and connected to

[15] The EEG signal with a frequency range of 8Hz to 12 Hz.

[16] See http://www.thelisapark.com/eunoia

loudspeakers. Each of the dishes responds sonically and visually (via vibration of the water inside) the different emotional states of the artist.

Unlike the pieces mentioned above, in which the performers are asked to be motionless and actively control the output of the EEG data, *Variations* instructs the performer to undertake actions that are determined by another source (the computer).

Materials

All sonic materials in *Variations* are computer-generated using various methods including audification and parameter mapping. Audification is a method to sonify a sequence of data by translating its waveform to an audible version. Parameter mapping is another sonification approach which scales the numerical values in the data to a desired range and maps the result to sound parameters.

Audification

The main frequencies of the EEG waves of a human are divided into five bands, which are Delta (4Hz or below), Theta (4Hz to 7Hz), Alpha (8Hz-15Hz), Beta (16Hz-31Hz), and Gamma (32Hz and above). Most of the signals are relatively low compared to human-audible frequencies, which are commonly give as the range between 20Hz to 20,000Hz. In order to transform the EEG signal into a waveform with frequencies in the audible range, it is first captured and written into sound buffers in SuperCollider, and then played back with the following methods: (1) Using the `BufRd` UGen to playback the recorded EEG signal at a higher frequency via a faster than normal playback rate, and (2) using a granulation technique, which involves playing back fragments of the sound buffers rapidly at a higher rate, with the intensity of each EEG channel mapped to the playback rate. With the granulation technique, a higher pitch on an audio channel represents a higher intensity of the EEG channel. This approach can be understood as mixing straightforward audification of the signals with a parameterised sonification approach.

Parameter mapping

Apart from audification of the EEG signal, I also used sounds produced via stochastic synthesis. For instance, the beginning sound is generated with the `Gendy1` Unit Generator in SuperCollider, with the amplitude of each EEG channel mapped to its maximum frequency parameter. The resulting sound is a rumbling tone. When a part of the brain becomes more active, the corresponding audio channel rises in pitch and intensity.

Execution

Technical setup

The technical setup for this piece involves various pieces of software and hardware. There are several brain-computer interfaces available in the consumer market, such as the OpenBCI[17] project and the MindWave[18] headsets. The one I chose for this project is called EPOC+[19].

[17] Open Source Brain-Computer Interface. See https://openbci.com

[18] MindWave is a family of EEG headset manufactured by NeuroSky. See http://neurosky.com

[19]. EPOC+ is an EEG headset manufactured by EMOTIV. See https://www.emotiv.com/epoc

This is a 14-channel EEG headset manufactured by the company EMOTIV. The reason I chose EPOC+ was that it provides multichannel EEG sensors that output raw data, and it has a rich software library. Also, it runs wirelessly with the Bluetooth technology which makes it convenient for live performance. Figure 5 shows the sensor locations with the international 10-20 system[20]

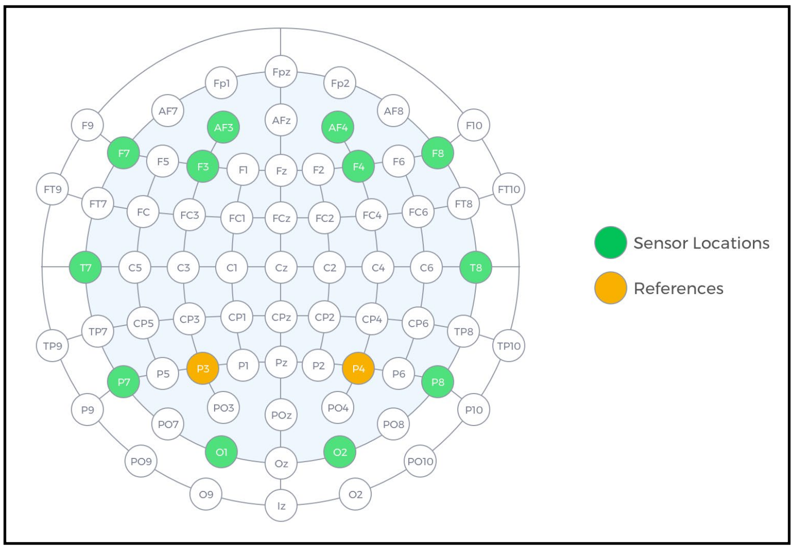


Figure 5 Location of each EEG sensor of the EPOC+ headset

The EPOC+ headset sends EEG data to the computer via a USB Bluetooth dongle. The data is then processed and transformed into OSC messages with two software, which are HoMy_EmoRaw[21] and MindYourOSC[22]. While HoMy_EmoRaw handles raw data, MindYourOSC handles data that is processed by the EMOTIV software, which includes

[20] 10-20 system is an internationally recognized system for electrode distribution on the scalp in EEG exams.

[21] A software that sends EPOC+ raw data in OSC written by Horácio Tomé-Marques in 2013

[22] MindYourOSC is a software that sends the analysed data from to EPOC+ including engagement, frustration, meditation, and excitement level, to other software via OSC.

the level of engagement, frustration, meditation etc. The OSC messages with those data are then sent to SuperCollider to be manipulated.

```
e = EPOCData();
{SinOsc.ar(freq:e.excitement.kr*440+440)}.play
```

Figure 6 Creating an instance of EPOCDta and map excitement value to frequency

In SuperCollider, I created two classes called EPOCData and EPOCRaw. The purpose of both of the two classes is to read the input OSC messages and map them into control buses, which can be used in different functions that make sound by simply calling the .kr method. In the example code above (figure 6), a variable e is assigned to a new instance of EPOCData, and its excitement level is mapped to the frequency argument of a sine oscillator.

Audio

The 14-channel sonification of EEG signals are handled separately and spread across 14 audio channels. The speaker distribution is made up of two layers of speaker rings. The lower layer is set at eye-level with four speakers at the front and four speakers at the rear of the listening point. The upper layer is set at a higher level (depends on the venue) with four speakers at the front and two speakers at the rear of the listening point. This distribution loosely resembling a combination of two French 8[23] speaker array without the

[23] French 8 is a speaker arrangement which divides eight speakers into four stereo pairs and locates them from front to back across the listening area.

rear-side pair on the upper layer. Each speaker channel represents an EEG signal channel as the speaker positions are analogous to the location of the EEG electrodes on the scalp as shown in figure 7.

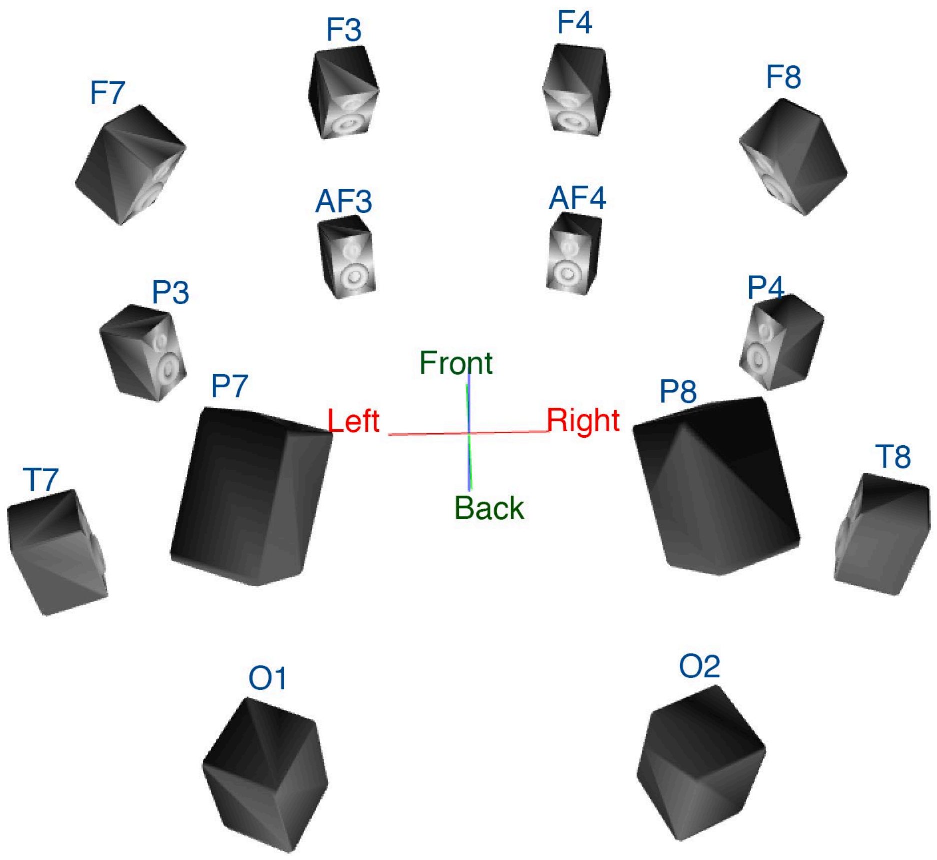


Figure 7 Speaker arrangement for Variations

Stage performance

The stage performance of *Variations* is indeterministic. The performer is asked to undertake a series of actions, which are meditating, reading a book, memorizing, and calculating, in an order that is determined by the computer. In the beginning, the performer does each of the actions once in a random order. Between each consecutive action, the computer will determine the next one by analyzing the level of frustration of the performer and then provides cues to the performer on the computer screen on stage. Each action has a duration of 40 seconds. Meditating herein indicates that the performer should sit motionlessly and try to clear their mind. The reading action here instructs the performer to read a book. In the realisation of this piece, I chose the Oxford Dictionary of Mathematics[24] because I found reading this book requires comprehending knowledge that I do not know. Memorising herein involves reading a page consisting of randomly generated numbers. The performer is asked to memorise fragments of these and then write them down on another piece of paper. Calculating herein requires the performer to continuously doing addition of random numbers on a blank paper. The idea of this approach is that the performer does not attempt to actively control the resulting brainwave signal, unlike in the earlier works cited above.

Visuals

The visual elements in this piece were produced using Processing[25]. The visualisation has a minimal design that visualizes the brain activity during the performance. It contains some

[24] Clapham, Christopher, and James Nicholson. *The Concise Oxford Dictionary of Mathematics 5/e*. 5 edition. Oxford: OUP Oxford, 2014.

[25] See footnote 2

square, rectangular, and L-shape shapes with similar colours. Each of them represents one channel of the 14-channel EEG data received from the EEG headset. The colour and size of each shape fluctuate during the performance, which indicates the intensity of the corresponding areas of the brain activity. There is also a verbal indication of what action is being done in the centre of the screen. The example below (figure 8) is a snapshot of the visualisation with indications of which part of the brain each shape is representing. In this example, the shapes in the bottom-right corner (P4, T8, and O2) are brighter than the rest of the shapes, showing that the corresponding area (rear-right of the brain) had more activity at the time the snapshot was taken.

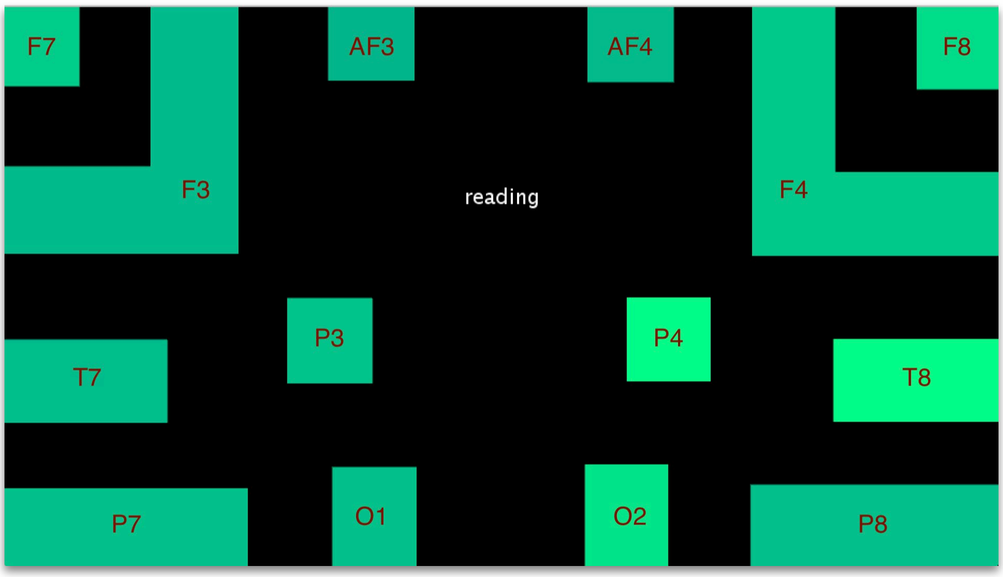


Figure 8 Example of the brain activity visualisation of Variations

Evaluation

Variations is the most sophisticated piece in this portfolio in terms of technical setup. In terms of hardware, the EPOC+ headset is time-consuming to setup and to adjust position of the sensors. To overcome this, although not mentioned above, I devised a SuperCollider class called OSCFile which records and playbacks OSC messages (a detailed description of OSCFile can be found in Chapter 3). This was very useful during the creation of this piece as it allowed me to record the data from the headset and play it back to test with sounds in SuperCollider without setting up the headset. For software, because there are multiple pieces of software involved (SuperCollider, HoMy_EmoRaw, MindYourOSC, as well as the visualisation on Processing), this increases complications when software crashes. When performing this piece, I tried to test separating myself from the performance system and not actively controlling the EEG signal by not thinking about the fact that I was performing on stage, and only concentrating on the actions that the performance system required me to do. I found this difficult to comply with because my consciousness of performing was always present, and the difference in resulting sound is hardly noticeable. It is also worth mentioning that because this piece mainly tries to sonify raw EEG signals instead of analysed data such as alpha and beta wave, the resulting sound does not react to the signal as expected. There are only slight differences in sound when the performer does different actions.

#3 nootherthanthevoid

Year of composition: 2016
Duration: 14 minutes
Format: stereo
Live electroacoustic piece involving Live Coding in SuperCollider and sensors

Synopsis

nootherthanthevoid is a live coding audiovisual piece incorporating the use of sensors. The main limitation of live coding as a performance practice is the lack of responsiveness compared to other forms of musical expressions. During a performance of live coding, whether the musician is writing code from scratch or modifying pre-written code, it can take a rather long time for a new block of code to be executed, with a possibly significant gap between an idea arising, and it being manifested in sound. During the performance of *nootherthanthevoid*, with the aid of Electromyographic (EMG) sensors as well as pressure sensors attached to the laptop, the live coder is able to make immediate responses to the sound and control the sonic parameters while the visuals are being generated. The data from the devices is directed to some control buses in SuperCollider which can be used on-the-fly and flexibly mapped to any sound and visual parameters in performance. There are two programming utilities called Snippet and Autoargs in SuperCollider created during the making of this piece which are discussed in Chapter 3.

Aim

The aim of this work was to seek ways to enhance the controllability and expressiveness of live coding performance in ways which do not hinder a live coder from typing the code. This work investigates the questions below:

- What are some ideas of what can be done to modify the way live coders and interact with computers?

- What are some of the limitations of live coding as a performance interface?

- Can sensors be used to overcome these limitations?

Background

The piece was my first attempt at solo live coding. Before creating this work, I had been participating in a live-coding laptop ensemble, BEER (Birmingham Ensemble for Electroacoustic Research). The main approach of the ensemble involves collaborative improvisation where all the performers plays independently within specific structures, algorithms, and materials belonging to a given piece. To enhance the efficiency of playing, the ensemble devised some practices and tools such as a chat system and code sharing system among the players. Specifically, with the code sharing system, which broadcasts code from each player whenever it is executed, the players can reuse other's code to generate coherent sonic material by copying and modifying it. When I started to work on this piece, I tried to rethink the nature and limitations of live coding in order explore how a solo performance could be more efficient in terms of transforming the player's thoughts or physical actions into sound. In order to achieve this, as a pianist, I tried to practise the action of typing as if practising on a piano by repeatedly typing some of the most common or frequently used "phrases" or "keywords" when I was live-coding. After some experiments, I found that physical virtuosity does not contribute much in this form of musical performance because regardless of how quickly a live-coder types, it always takes at least some time to transform a musical thought into meaningful code to be executed. I did find some inspirations from this process of comparing playing a piano and live coding, which was the use of pedals as a modifier of the sound. After some research on using sensors with microcontrollers, I decided to utilise pressure sensors and Electromyographic (EMG) sensors since they do not require much body movement that could hinder a performer while typing.

Aesthetic

Sonically, the piece used a feedback mechanism which is also used in some other pieces in this portfolio. It processes some simple audio signals including impulses and pure sine tones and transforms them into sounds with a richer timbre. This feedback mechanism, which is later named Looper, is mainly made up of a pair of record and playback UGens with which the samples recorded in the buffer are scrambled or shifted. The initial idea of this mechanism was to create a simple rhythmic repeater which loops a sample with a specific start and end time when triggered. After some experiments, I found that it can produce sounds with a rich sound colour because of the artefacts of shifting samples indices during recording to and playing from the sound buffer. A detailed explanation of this mechanism can be found in Chapter 3.

The title was inspired from a classical mantra of Buddhism, the Heart Sutra. "Form is no other than the void, the void is no other than form, form is the void, the void is form".

From these four lines, the words “no”, “other”, “than”, “the”, “void” appear in various places, shifting their position may mean the same or mean drastically different depending on how much to shift, this is analogous to the sample-shifting mechanism devised for this piece.

Hardware

There are two pieces of external hardware used in this piece, a purpose-built circuit that reads data from the pressure sensors, and an electromyographic (EMG) sensor which reads the skeletal muscle activity of the performer’s forearm. The advantages of using sensors in a live-coding performance are that they allow the performer to manipulate the sound even when typing code.

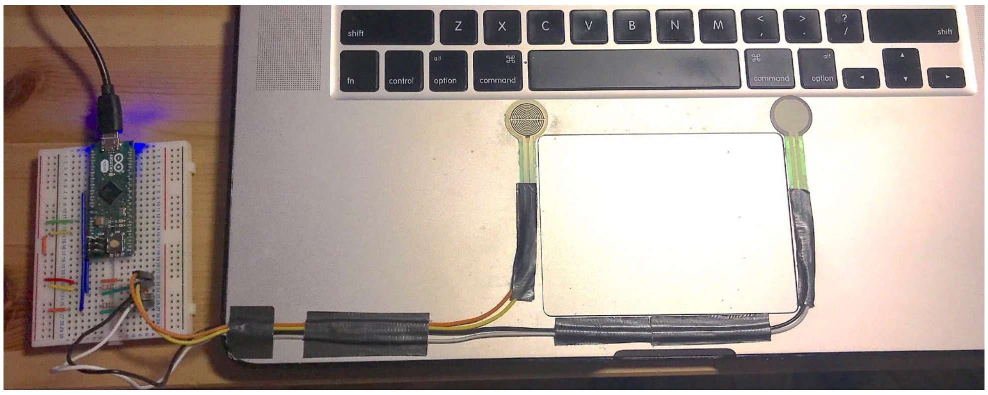


Figure 9 Arduino with pressure sensors attached to the surface of the laptop

Pressure sensors

A simple Arduino[26] circuit was used in order to utilise the force sensors attached near the keyboard on the laptop. The values from the sensors are read from two analog pins on the

[26] An open-source platform for electronic prototyping

Arduino microcontroller with a 260 Ohms resistor for each. Each of the sensors provides a resolution of 10 kilograms of force. The data are then read with the Serial class in SuperCollider. I undertook some experiments involving the placement of the sensors, for example, placing them on a separate surface and placing them vertical to the circuit. I found the placement shown in figure 8 the most suitable because the performer can use it with thumbs while typing with the other fingers.

Electromyographic (EMG) sensor

Electromyography is a technique to examine the electrical activity of muscles. In this piece, I used an EMG armband called Myo[27]. In the early stage of development of this piece, I had tried to use some other EMG sensors, but the result was rather unstable, and recalibration was required every time it operates. The Myo armband offers a relatively steady signal.

SOFTWARE

Apart from utilising the hardware sensor above. I also implemented two programming utilities, Snippet and AutoArgs, which are discussed in Chapter 3 in detail. Similar features are available in other programming environments such as Emacs but I found it missing in the SuperCollider IDE. The Snippet utility acts as a programming code shortcut: A user can convert short keywords into longer code by hitting a shortcut key. The AutoArgs fills a UGens creation statement with its default values, making them easily visible and

[27] Myo is an 8 channel EMG armband manufactured by Thalmic Labs Inc.

modifiable. These features enhance the efficiency of live coding as well as programming in general for SuperCollider.

Visuals

The visual accompaniment for this piece has two main components, one is the code that is being executed during the performance, another one is the background which consists of many tiny grids of colour that changes over time. Every line of code on the performer's screen is projected on the screen. Although the basic structure is written in the code, the control of the frequency of the colour changes and how they interact with the sound are controlled via OSC communication. During the performance, OSC commands are sent from SuperCollider to the programme that runs the visuals to change the parameters (e.g. the alternating rate for a specific colour, the size of the code, and the speed the code scrolls).

Audio

All the source sounds are generated in real-time with code in the SuperCollider environment. To make use of the sensors mentioned in the previous section, I used a software called MyoMapper[28], which sends OSC messages with the parameter of the Myo armband. Because of the nature of this piece (mainly live-coding), there is only one parameter being used from the armband, which is the average intensity of the muscle in

[28] MyoMapper is a software that reads the data from the Myo armband and converts them into OSC messages, written by Balandino Di Donato.

the forearm. Both the signal from the EMG armband and the pressure are mapped into control buses which can be used in functions that makes sound by calling the .kr method.

Execution

This piece is a rehearsed improvisation. While the main elements of the structure in this piece such as the choice of source sound and the overall shape is fixed, the parameters in the synthesisers are improvised. Therefore, it sounds slightly different every time it is performed.

Evaluation

This piece is the most performed piece in the portfolio. The main limitation of a live coding performance is it is difficult to give instant responses and interact with the sound. Through the creation of this piece, I found two solutions to overcome this limitation.

1. Software-based solution - to accelerate the process of coding by using code snippets and automatic code completion (Snippets and AutoArgs in Chapter 3)
2. Hardware-based solution – to let the performer control some sound parameters while coding

By using hardware, I find it changed the way the computer and the live coder interacts. During a live coding performance without using sensors, live coders transform their musical ideas into computer codes, and then execute the code to create sound. In this model, there is not much physical interaction between the live coder and the computer apart from typing. With the use of sensors, the live coder can treat the computer more similarly to a musical instrument than in the case of live coding alone.

#4 "Please Save Me"

Year of composition: 2017
Duration: varies (20 minutes in realisation)
Format: Installation/Performance

Synopsis

"Please save me" is a combination of installation and performance. This work involves an immobile performer as an installation that interacts with the sound in the venue. The performer is asked to sit still while wearing two transcutaneous electrical nerve stimulation (TENS) machines which is connected to a computer wirelessly. The machines send electrical impulses to the performer's body to trigger movements with various rhythms and speeds according to the sonic environment.

Aim

The aim of this work is to create an installation with a human body as a physical object that is placed in an environment and interact with it in an involuntary way, and to thus investigate an alternative mode of Human-Computer interaction. It explores the following questions:

- Can sound be transformed into body movement (which is in reverse of the mechanism of the previous pieces)?
- What are some of the challenges of implementing this idea?

Background

The original idea of this piece was to create a performance artwork involving a human body controlled by a computer via some algorithms. After some investigation into finding possible ways of realising this idea. I learnt that there is a long history of manipulating the body with electricity in performance art.

The roots of this art form can be traced to the 18-century when animals were used in experiments (Elsenaar et al, 2002). Until recently, there have been many artists who practice this art form, including Stelarc[29], Arthur Elsenaar[30], Choy Ka Fai[31], as well as Daito Manabe[32]. Arthur Elsenaar created performances of externally controlled facial expression in his long-term research project *Artifacial*[33]. Singaporean artist Choy Ka Fai's project, *Prospectus for a Future Body*[34], involves capturing the body movements of a dancer and reproducing the movements by imposing them onto another person. *"Please save me"* is at some level inspired by the works of these two artists.

---

[29] Stelarc is a performance artist based in Australia. His works mainly concentrate on extending the human body, notable projects include *Exoskeleton* and *Ear On Arm Suspension*. See http://stelarc.org/projects.php

[30] Arthur Elsenaar is a Dutch artist who focus on externally controlled human face. See https://artifacial.org/about/arthur-elsenaar

[31] Choy ka Fai is a Singaporean artist based in Berlin. His works intersects the domain of dance, media art, and performance. See http://www.ka5.info/about/biography.html

[32] Daito Manabe is a Tokyo-based artist. See http://www.daito.ws/en/biography

[33] See https://artifacial.org/about/artifacial

[34] See http://www.ka5.info/prospectus.html

The title was taken from a line spoken by an android in the Japanese animation Ghost in the Shell. In one scene, there was an hi-jacked android being terminated, and it is later revealed that there was a human conscious being encoded and trapped inside the machine.

Technical execution

While the mechanism is rather simple, there were several considerations regarding the setup of this piece. The first consideration was where to place the electrode pads and hence which body parts is triggered. I considered putting all the pads on the arms, but it seemed not very effective since placing the pads on and under the arm can result in a kind of cancelling of the movement. Thus, I decided to spread the 4 channels to all the limbs. The second consideration was the posture of the performer. On one hand it is difficult to sit immobile in some posture for a long period of time, on the other hand the floor would hinder some body movements for example while laying down. I have tried sitting upright on a chair and lying on the floor. After some experiments, I decided the posture to be sitting on the floor with the legs stretch loosely towards the front and resting both arms on the knees.

The hardware devices used in this work include a computer with a microphone that runs SuperCollider, two TENS machines, a 4-channel relay, a microcontroller with WiFi module, and a WiFi router. A router creates a local wireless network that connects the computer and the microcontroller, which is an ESP8266[35] in this realization. The computer

[35] ESP8266 is a compact microcontroller and Wi-Fi module manufactured by espressif

runs the main SuperCollider programme and sends Open Sound Control (OSC) messages to the microcontroller within the local network.

The TENS machines used in the first performance of this piece have two channels. For each channel, there are two electrode pads to be connected; positive and negative. The electrical current passes through these two pads. The device sends small electrical impulses to the attached skin. The original purpose of a TENS machine was to serve as a solution to reduce pain by blocking pain signals from a specific area to the spinal cord and the brain. With a higher intensity of voltage, the machine can trigger contraction of muscles and thereby move the body part it is attached to. A four-channel relay is placed between the connection of the machines and the electrode pads attached to the body. On the other end of the relay, it is connected to the ESP8266. I chose to use ESP8266 because of its a versatility and as it supports Arduino[36] code, which I also used in other projects such as *nootherthanthevoid* (another piece in this portfolio).

Evaluation

During the first performance at BEAST FEaST 2017, there were a few attempts from the audience trying to interact by clapping hands, which did trigger some changes in the rhythm, and there some positive feedback from the audience. After the performance, when reflecting on this work, I found a few neglected problems with this implementation of the idea. Considering the first question explored by this project, sound can be transformed

[36] Arduino is a family of microcontrollers for building hardware devices that can process and interactive with different sensors and components.

into body movements, but one problem is, the coherence between the externally controlled body movements and the sound in the environment. Since the mechanism devised for this piece is only capable of triggering jerking movements to the limbs, there is thus a lack of variety of motion in terms of how the resulting movements are representing the sound. It is dissimilar to the mapping positions of hands to sound parameters which is done in *(un)touched*, mapping sound intensity to body movement lacks correlation between the two. There is room for improvement in terms of spreading different aspects of the sound into different limbs. Another problem was related to technical aspect of this work. As the mechanism relies on wireless network communication, the wireless devices such as laptop computers and mobile phones in the venue can interfere with the work. On the positive side this represented some interaction between the piece and the space, but it was not anticipated.

#5 DDS.SRY

Year of composition: 2017-2018
Duration: 5 minutes
Format: stereo fixed media

Synopsis

DDS.SRY is a piece of fixed media music that uses sonic materials generated with sonification of genetic data. While it serves as a proof of concept, it utilizes the DNA (deoxyribonucleic acid) sequence of the human SRY gene as source data. The SRY gene is responsible for giving instructions to produce the sex-determining region Y protein, which is the key to male humans' sexual development. This piece tries to create a sonic representation of the gene by converting each nucleotide (the building-block of the DNA sequence) into a single rhythmic pattern. The rhythmic patterns are then presented with various sonic materials. This piece is distinct from the previous four works in this portfolio not only in that it is fixed media, but also in the involvement of sonifying static data instead of dynamic data. This piece is in the same series of music with the following piece (*DDS.CFTR*), which share a lot of its ideas in the composition process.

Aim

The aim of this work is to create a fixed media piece having sonic materials and structure that echo a DNA sequence via the sonification of the genetic data in a rather straightforward manner. Straightforward here means that the resulting sounds are directly converted from the DNA data in a way in which the converting processes could be reversed. It contrasts with the next piece (*DDS.CFTR*), which intended to achieve the same goal in a more indirect manner. The work also tries to find the answers to the following questions:

- What are some of the limitations of sonifying non-numerical data like DNA sequences in terms of music-making?
- What are some things that can be done to overcome these limitations?

Background

The original idea of this work was to create music using the sonification of data derived from some unique features of a person. Examples of these include fingerprints, palmprints, and iris patterns, which are usually used for biometric identification. I found it interesting at first because this resembles the use of musical cryptograms[37] by some composers in the past, such as Bach or Shostakovich, who embedded their name in musical notes as a signature. After some research into biometric identification, I considered using DNA, which is widely used for forensic purposes to identify people. Compared to the features mentioned above, DNA is perhaps more directly applicable to making sounds because it consists of sequences. Initially, I would have liked to have had my own DNA sequenced to be used to generate musical materials, but it turned out to be too costly and time consuming. Also, because this was my first attempt at working with biotechnological data, after some considerations and research, I decided to treat this piece as a proof of concept

[37] Musical cryptogram is a cryptogrammatic sequence which some composers used to convert extra-musical texts such as names into musical notes, most notably Johann Sebastian Bach's BACH motif which transforms in to notes B Flat, A, C, B Natural.

and use some available data on the internet. There were a few experiments in realizing the aim of this piece. One of them was to create a compositional model that transforms the whole human genome into music. This was found impractical because the human genome consists of more than 3 billion base-pairs (the length unit of the genetic sequences) which is hard to be compressed in a way that it can be used. In lieu of using the whole genome, I found using a single gene more applicable since in a human, the length of a gene could vary from few hundreds to more than 2 million, which is more manageable compared to the genome as a whole.

Sonification

Before discussing about the sonification process, I shall briefly explain the structure of the DNA data used herein[38]. Firstly, DNA sequences are ordered sequences composed of four kinds of nucleotides, which are Adenine(A), Guanine (G), Cytosine (C) and Thymine (T). The length of a gene is usually counted in base-pairs, which is a pair of nucleotides bound to each other that form a two-stranded spiral structure called a double helix in a DNA sequence. The length of a gene could range from a few hundred base-pairs to 2 million base-pairs.

Out of several genes I investigated, I selected the SRY gene to be used in this piece. The reason I chose this gene is purely because of its length being suitable. I have tried to apply

38 See https://www.ncbi.nlm.nih.gov/gene/6736/

some other genes to the sonification method used in this piece, the differences among the results sounds are almost unidentifiable, due to the lack of sufficient variation in DNA sequence. The data file I used was obtained from the GenBank[39] website. It contains 887 nucleotides which are part of the SRY gene.

All source sounds in this work are made from simple oscillators including sine waves, impulses, square waves, and brown noises. The reason that I chose these simple oscillators was because I want to keep the timbre of the result sound as simple/clean as possible.

The difference between sonifying genetic data and other data is that it is not numerical, and it cannot be directly mapped to sound parameters through scaling. This opens an opportunity for having various approaches and methods of sonification. To generate sound materials in this piece, I assigned each nucleotide a rhythmic pattern. Although there were a few attempts to compose a rhythmic motif specifically for each nucleotide, I decided to convert them into Morse code at the end. Morse code was chosen because of a few considerations. Writing dedicated rhythmic patterns would involve subjectivity in the character-to-rhythm conversion process. I would like to leave room for subjectivity in the composition of the piece but not the creation of the sonic materials. Also, Morse code

[39] GenBank is the genetic database of National Institutes of Health (NIH). It is a collection of DNA sequences with annotations which is available to public.

was widely used in communication, and it was created specifically for this kind of conversion.

In order to transform the DNA codes into musical rhythms, I used the class `Morse`[40] in SuperCollider which converts a string of characters into Morse code. For example, a DNA sequence of "`TGTT`" would be transformed into "`- --. - -`" where dashes represent long tones and dots represent short tones. The Morse class will return an array of `[ 0.3, 0.3, 0.3, 0.1, 0.3, 0.1, 0.1, 0.3, 0.3, 0.3, 0.3, 0.7 ]` (Odd-index values refer to note duration and even-index values refer to silence) as the rhythmic values.

Composition process

In the beginning of creating this piece, I tried to generate the whole piece by only processing the source data using the sonification strategy mentioned above. I tried different ways to add aesthetic variations such as changing timbre, layering, and dynamics. I found it difficult with this approach to achieve a balance between aesthetics choices and how 'authenticly' the resulting sounds corresponded to the original data. This problem was mainly due to the lack of variations in the source data, and as there are only four possible outcomes of rhythmic patterns. At last, I decided to apply processing to the resulting sounds, including filtering, reverberations, as well as using the `Looper` processor (which is discussed in chapter 3). This adds more variation in terms of tone colour and texture to the music.

[40] Morse is a SuperCollider class implemented by Alberto de Campo. See https://github.com/supercollider-quarks/Morse

Execution

The piece starts with a sonification of genetic data using a high-pitched sine tone which gradually fades in. It has a rhythmic pattern of the SRY gene from the start. At 0'38", the sonification becomes split across the two channels, where the left channel represents the DNA sequence from start to end direction; the right channel represents the DNA sequence in the reverse direction. There is also a low-frequency sine tone that sonifies the data with the same method but at a slower speed.

After converting the genetic data into rhythmic values, it can be applied to sound generation by setting it as the duration key in patterns. I created a pseudo-score that managed the overall structure of the piece in SuperCollider. The resulting sound was recorded and placed in Ableton Live[41] for further processing and arrangement.

Evaluation

Like many pieces in this portfolio, my compositional strategies and approaches changed during the creation process. The result was successful in my opinion in terms of serving as a proof of concept. Considering the questions raised above, one limitation of sonifying non-numerical data like DNA sequences I found during the process of composing this piece was that it is hard to avoid subjectivity in this kind of sonification. Since there is no scalable content in the data, it is not possible to apply any parametric sonification techniques, nor applicable to use audification as used in *Variations*. Also, it is challenging

[41] Ableton Live is a digital audio workstation and music sequencer primarily used in live electronic music See See https://www.ableton.com/en/live

to obtain a balance between having the resulting sounds accurately represent salient aspects of the original data as in scientific sonification and having sufficient artistic freedom to make an aesthetically acceptable result. To overcome these limitations, I chose to map the sequence to rhythms using an existing and widely known system (Morse code) instead of composing dedicated rhythmic motifs. Also, instead of intervening in the process of sonification, I added audio processing to the resulting sounds to add aesthetically chosen variation to the more straightforward sonification result that the approach afforded.

#6 DDS.CFTR

Year of composition: 2017-2018
Duration: 11 minutes
Format: stereo fixed media

Synopsis

DDS.CFTR is another piece in this portfolio that uses genetic data. It shares many similarities in the process of composition with the previous piece. The source data used in this piece is called the CFTR gene, which is related to the genetic disorder, Cystic fibrosis. The DNA data was mainly transformed into rhythmic patterns in this piece, but the approach is different from the previous work. Instead of the nucleotides being sonified individually, this work tries to sonify their frequency of occurrence in groups.

Aim

The aim of this work is similar to the one of DDS.SRY, which is to create a fixed media piece having sonic materials and structure that echo a DNA sequence via sonification of the genetic data. While DDS.SRY tries to achieve this in a more straightforward manner, this piece attempts to implement a more indirect approach by creating an audio representation of the occurrence or frequency of the nucleotides. This work also tries to explore the following questions:

- What are some of the limitations of sonifying processed DNA data?
- What are some things that can be done to overcome these limitations?

- What might be a viable approach allowing genetic sequences to be transformed into numerical data for use in parameter-based sonification?

Background

The piece was created after composing DDS.SRY. At that time, I was wondering about other ways to sonify genetic data and started to investigate the structure and content of the DNA sequence. After some investigation, I learned that nucleotides are treated as groups to form larger units. There have been studies about frequency of mononucleotide, dinucleotide, as well as trinucleotide (codon) groups in different genomes. I decided to experiment with different ways to sonify these groups of nucleotides. In the beginning, there was an attempt to devise a sonification system that assigns codons (trinucleotides) to notes and form harmonies. I tried to extract all start and stop codons[42] in a given DNA sequence to create block chords out of the it, which could potentially display the sizes and the constituents of some mRNAs. This idea was later set aside later because I was not confident that it is scientifically legitimate. At the same time, I was also experimenting with other ways to utilise dinucleotides as sonification source data. This was the approach chosen for this piece.

Sonification

The sonification method for this piece is relatively indirect compared to the approach used in DDS.SRY. The idea was to filter and generate numerical information from the DNA

[42] Start and Stop codons are nucleotide triplets in DNA and RNA that signal a start and stop of the translation into proteins.

sequence. The nucleotides are grouped into pairs (dinucleotides) during the sonification process. The positions of the pairs in the whole sequence are extracted and converted into 16 rhythmic patterns. While there are 4 nucleotides (A, G, C, and T), grouping them into pairs will give 16 different combinations (AT, GC, CA, TC etc.). For example, in figure 10, the gene begins with a sequence of "AATTGGAAGCAA…". By looking up the positions of the dinucleotide AA, a rhythmic value of 3:2 can be generated. The result rhythmic patterns of AA are [3, 2, 12, 1, 58, 10…] from the data used. This processed was done to all the pairs. After the generation of rhythmic patterns, I mapped them into 16 notes with different pitches. With this method, the repeated note in the resulting sound with the same pitch will show the frequency of a specific dinucleotide in the section of the sequence.

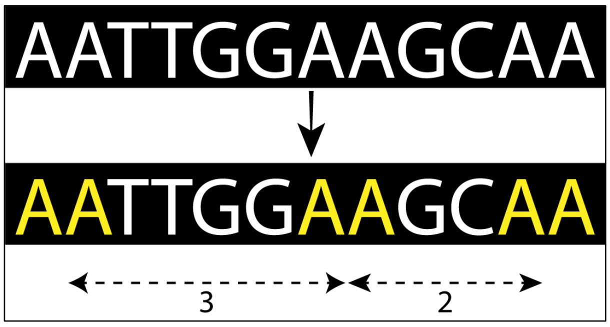


Figure 10 Converting positions of dinucleotide into rhythmic patterns

The advantage of using this method of sonification is that it can involve more data in a short period of time. The sequence used in this piece has 188706 base-pairs, which is more than 200 times longer than the genetic sequence used in the previous piece.

Like DDS.SRY*,* most of the source sound is generated with some simple oscillators in SuperCollider. I also used other synthesis techniques such as stochastic synthesis to generate sounds to serve as an accompaniment and add timbral variations.

Structure

The piece is roughly divided into 3 sections. In the first one the main materials are some unprocessed beeping sine waves sound generated with sonification. The beeping sounds are spread slowly and pointillistically in the beginning and gradually increase in density. The second section starts at 3'00". It starts with an abrupt short clap-like sound and enters a quieter sonic atmosphere with a prolonged high-pitched tone. This section becomes more and more violent from 4'30" onward with lower-frequency contents. The third section starts at 6'20", it also starts quietly, but in a different kind of sonic atmosphere. It begins with a drone sound with some long metallic sounds fading in and out sporadically. It develops to a built-up in texture at 8'08", where more sounds which appeared in the previous section reappear, and it reaches its climax at 9'10". The piece ends with a low-frequency drone.

Evaluation

Although this piece is also about sonifying DNA data, the process and the resulting sound are distinct from the previous piece. The main idea of the sonification approach of this piece is to transform non-numerical genetic data into numerical data for parameter mapping. In DDS.SRY, the sonification process can be reversed and transformed back to data; it is impossible to do the same with DDS.CFTR because it was not the nucleotides being captured but their occurrence and frequencies. The resulting sound is different as well, while the sound material in DDS.SRY is more rhythmic and energetic, the sound materials in DDS.CFTR has a more sporadic and slower character.

Chapter 3

Software

In this portfolio, I have designed some software with SuperCollider which creates sound and aids the process of composition and performance. For sound creation, I designed a signal processor that manipulates sound at the sample level. I also created three coding utilities for SuperCollider and the SuperCollider Integrated Development Environment (ScIDE). The first one is a class called OSCFile which can record a large number of OSC messages using a specific file format. The other two are tools for coding in SuperCollider. One is a snippet tool for quick programming code regeneration called Snippet, and the other one is a tool for auto-completion of arguments called AutoArgs.

Digital Signal Processor

All of the sound materials in this portfolio are computer-generated without any real-world sound recorded. While many of them are the result of sonification of data, including direct audification (e.g., Electroencephalogy data in *Variations*), and parameter mapping with data (e.g., mapping DNA data in DDS.SRY), most of the pieces utilized a feedback-based digital signal processor called Looper written in SuperCollider.

Looper

Looper is a feedback-based Unit Generator[43](UGen) which shifts and scrambles samples from the input signal. It is a SuperCollider Pseudo-UGen[44] with a simple record and playback mechanism. It is composed of a LocalBuf (a buffer inside a SynthDef), a BufWr, and a BufRd with changeable read/write parameters. BufWr and BufRd is a set of recording and playback UGens in the SuperCollider language, while the other two are RecordBuf and PlayBuf. BufWr and BufRd provide control of the recording and playback position in the buffer with the phase argument.

There are 9 arguments in Looper. The first 3, which are in, numChannels, and maxDuration, represent the input signal, the number of channels in the signal, and the maximum duration of the loop respectively. They cannot be changed after the synthesiser is created. The later six arguments are recRate, recScale, recDur, playRate, playScale, and playDur, which represent the rate, the scale of the position according to the buffer, and the duration of the buffer during recording and playback. The rate arguments (recRate and recScale) affect the rate of change of the recording and writing position within the buffer. The default rate is an incrementation of 1 sample per sample synthesised. When it is not equal to 1 and greater than 0, this change recording/playing position is multiplied by the value thus creating a slower (when the values are greater than 1) or faster (when the values

[43] Unit Generators are SuperCollider programming classes that process or generate sounds in various ways.
[44] Pseudo-Unit-Generators are written by combining existing Unit-Generators into a sub-graph rather than directly in C++. From a user point of view they do no significantly differ from normal UGens.

are below 1) version of the original sound with glitches. The scale arguments work in a similar way as the rate arguments, but it constrains the recording/playing position in the buffer and fill the samples in between with zeros. When the recording rate or scale arguments are set to 0, the recording will stop, and the resulting sound will be a loop of the content in the buffer.

For example, in Figure 11, when a signal of a ramp from zero to the sampling rate (0, 1, 2, 3…44100) comes in and the recScale is set to 1, the output will be identical to the input; when the recScale is set to 2, the output signal will shift by 2 (0, 0, 1, 0, 2, 0, 3…); when recScale is set to 0.025, the resulting sound will be a high-pitch glitch.

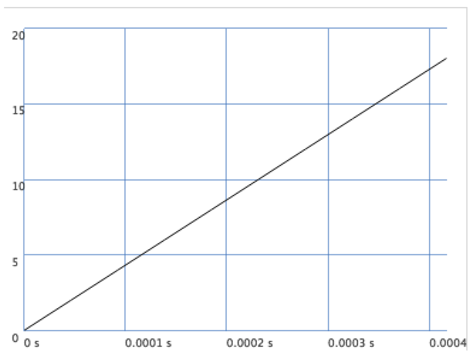


Figure 11a Resulting waveform when recScale is set to 1 in Looper

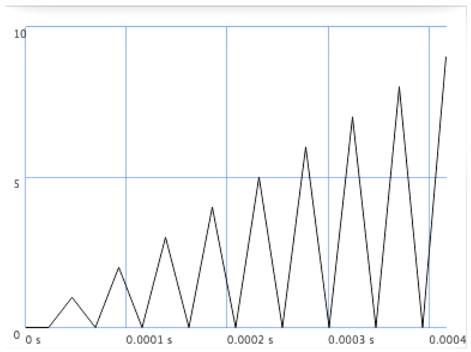


Figure 11b Resulting waveform when recScale is set to 2 in Looper

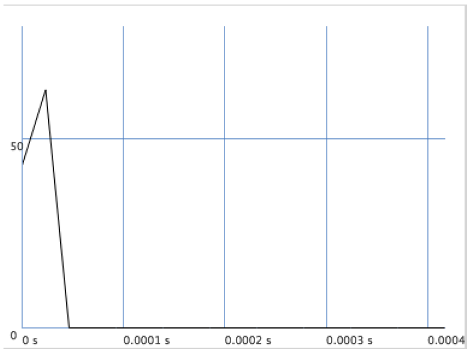


Figure 11c Resulting waveform when recScale is set to 0.5 in Looper

```
Looper: UGen {
	*ar{
	arg in, numChannels = 2, maxDuration = 1, recRate = 1, recscale = 1, recDur = 1,
playrate = 1, playScale = 1, playDur = 1;
	var lbuf, duration;
	duration = server.default.sampleRate * maxDuration;
	lbuf = LocalBuf(duration, numChannels).clear;
	BufWr.ar(in, lbuf, Phasor.ar(0, recRate, 0, duration * recDur) * recScale % (duration *
recDur));
	^BufRd.ar(numChannels, lbuf, Phasor.ar(0, playRate, 0, duration * playDur) * playScale
% (duration * playDur));
	}
}
```

Figure 12 The source code of Looper

Utilities

In this section, I will discuss some programming utilities that I wrote to enhance the functionality for SuperCollider. Because the computer platform that I used most is macOS, all of the utilities are optimized for the macOS version of SuperCollider.

OSCFile

Background

OSCFile is a class and file format in SuperCollider that records and plays back Open Sound Control (OSC) messages. The idea was to create something similar to the common MIDI file format, which is also used to record musical instruments or interface data. The idea of making this tool arose because some of the pieces in this portfolio require the use of external devices that send a large number of OSC messages at a high frequency (e.g. hand gesture data from the Leap Motion Controller used in *(un)touched*, and EEG data used in *Variations*) and it would be convenient if the data could be recorded and played back at a later time without the need of connecting the device to the computer. It can also be used to record a performance. For example, pieces like *(un)touched* and *Variations* vary slightly each time when they are played, and with this tool, a given performance can be recreated later, at least in terms of the input data. It also saves hardware setup time when testing. The EPOC+ EEG headset used in *Variations* needs to be moisturized every time it is used, which takes on average 15-20 minutes. With this tool, one can just record some sample data from the machine and play back the data without having to set up the device.

There are several similar tools I found on the internet such as OSCSeq[45] as well as the SuperCollider Quarks[46] OSCFileLog[47]. The different between OSCFile and them is that all of them save the OSC messages in either a text-based or XML file format, while OSCFile saves it in a binary format. With some experiments, I found them very useful when the devices in use send OSC messages sporadically, but when using high data rate devices, they become slightly impractical due to the file size and the playback rate. With an XML or text-based file, the file size accumulates quickly as all of the data values as well as metadata such as timestamp and OSC path are stored as chars. For instance, in a test using OSCSseq, a file for a 5-second OSC data at a rate of 1792Hz (14 channels * 128Hz as from the EPOC+ sensor) has a size of 1.1 megabytes, while a file created with OSCFile with the same data is only 200 kilobytes, which is approximately 5 times smaller. Also, the latency when using XML is very high.

Usage

The operation of OSCFile is quite straight-forward. Below are some of the messages that it accepts.

- new - create a new instance of OSCFile; there are four arguments:
    1) precision - Precision of numerical values, it can be either 'double' or 'float'. 'double' writes floating point number as 64 bits and 'float' writes as 32 bits. Default setup uses 'double';

[45]an OSC message logging software.

[46] Quarks is a package manager that handles external classes for SuperCollider.

[47] a SuperCollider external class written by Marije Baalman.

2) includeAddress - A boolean to indicate if the network address of the OSC message source is recorded in the file;

3) mutePaths - An array of OSC paths to be ignored;

4) muteAddrs - An array of net addresses to be ignored.

- start - Start recording OSC messages.
- read - Opens an existing file.
- play - Starts playback.
- stop - Stops recording or playback.
- setPos - Set the current playback position of the file.
- write - Save the recorded data into hard disk.
- writeToTextFile - Export the recorded data into human-readable text format.
- gui - Create a new instance of OSCFile and open a Graphical User Interface

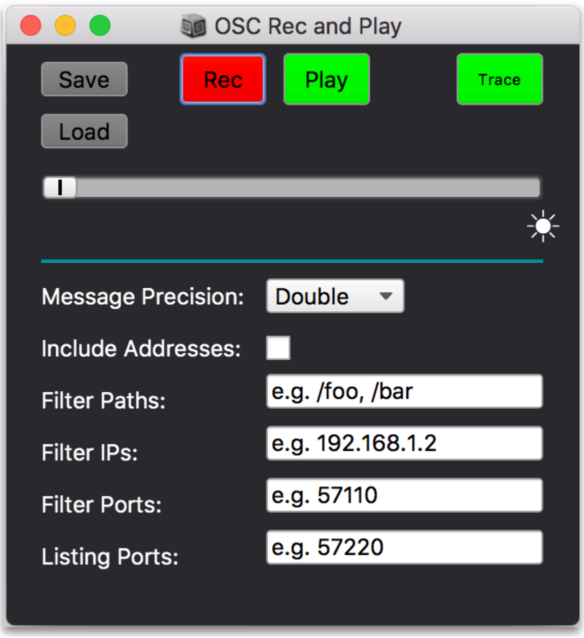
```
x = OSCFile('double', false);
x.start;
// recording OSC messages
x.stop;
x.write("/PATH_TO_SAVE_FILE");
// open an existing file
x.read("/PATH_TO_SAVE_FILE");
x.play
```

Figure 13 SuperCollider code example for record and playback OSC Message

Above is example code that records some OSC messages in double floating-point number format, saves them to hard disk, and plays them back.

GUI

OSCFile has a graphical user interface (GUI) that supports the same functionality as the text version. The parameters are similar to the arguments mentioned above but written in a more generic way. There is a playback bar in the GUI to control the playback position of the file and also a trace button to execute the code `OSCFunc.trace` to print all the incoming OSC messages.



Message Precision: Double
Include Addresses: ☐
Filter Paths: e.g. /foo, /bar
Filter IPs: e.g. 192.168.1.2
Filter Ports: e.g. 57110
Listing Ports: e.g. 57220

Figure 14 OSCFile Graphical User Interface

File Format

OSCFile uses a binary file format specifically designed for it. The format is similar to that of WAVE files. To reduce the file size, all the OSC paths and source network address are saved in the metadata section (no.9 and no.10 in table 1) and can be retrieved with indices from the actual data section.

Table 1: OSC File format

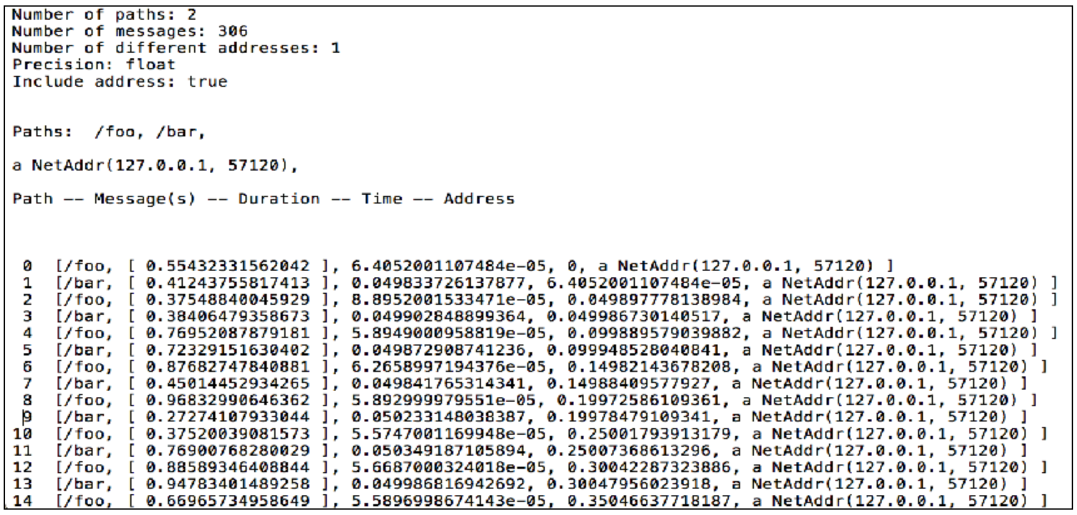
No.	Offset	Size (in byte)	Name	Description
1	0	4	ChunkID	Contains the letters "OSCL" in ASCII form
2	4	4	ChunkSize	Size of the chunks (i.e. the starting position of the actual data)
3	8	4	Subchunk1ID	Contains the letters "info" in ASCII form
4	12	4	NumPaths	Total number of OSC paths
5	16	4	NumMessages	Total number of OSC messages
6	20	4	NumAddr	Total number of OSC source network addresses which consists of IP address and port
7	24	4	DataType	Precision of numerical data, either float or double
8	25	1	IncludeAddr	Indicate if the file includes any OSC source network address
9	26	(n)	Paths	An array of all the different OSC paths. n is the number of OSC paths times the number of characters.
10	26+n	(m)	Addrs	An array of all the OSC source network addresses. m is the number of OSC source address times the number of characters.
11	26+n+m	4	Subchunk2ID	Contains the letters "data" in ASCII form
12	30+n+m	…	Data	The actual OSC data in the following order: message size -> path -> message(s) -> time -> source address(optional)

Table 2: OSC File data format

No	Data	Size (in byte)	Description
1	Number of values	2	The number of values in the message
2.	Index of the OSC path	2	The index corresponds to the stored OSC paths
3.	Message values	(varies)	The actual value(s) contained in the message
4	Time	8	The time that the message was sent relative to the first message

Export

OSCFile can export the existing OSC messages into a human-readable text file which lists all the data including the OSC paths and the actual messages.



```
Number of paths: 2
Number of messages: 306
Number of different addresses: 1
Precision: float
Include address: true


Paths:  /foo, /bar,

a NetAddr(127.0.0.1, 57120),

Path -- Message(s) -- Duration -- Time -- Address



 0   [/foo, [ 0.55432331562042 ], 6.4052001107484e-05, 0, a NetAddr(127.0.0.1, 57120) ]
 1   [/bar, [ 0.41243755817413 ], 0.049833726137877, 6.4052001107484e-05, a NetAddr(127.0.0.1, 57120) ]
 2   [/foo, [ 0.37548840045929 ], 8.8952001533471e-05, 0.049897778138984, a NetAddr(127.0.0.1, 57120) ]
 3   [/bar, [ 0.38406479358673 ], 0.049902848899364, 0.049986730140517, a NetAddr(127.0.0.1, 57120) ]
 4   [/foo, [ 0.76952087879181 ], 5.8949000958819e-05, 0.099889579039882, a NetAddr(127.0.0.1, 57120) ]
 5   [/bar, [ 0.72329151630402 ], 0.049872908741236, 0.099948528040841, a NetAddr(127.0.0.1, 57120) ]
 6   [/foo, [ 0.87682747840881 ], 6.2658997194376e-05, 0.14982143678208, a NetAddr(127.0.0.1, 57120) ]
 7   [/bar, [ 0.45014452934265 ], 0.049841765314341, 0.14988409577927, a NetAddr(127.0.0.1, 57120) ]
 8   [/foo, [ 0.96832990646362 ], 5.892999979551e-05, 0.19972586109361, a NetAddr(127.0.0.1, 57120) ]
 9   [/bar, [ 0.27274107933044 ], 0.050233148038387, 0.19978479109341, a NetAddr(127.0.0.1, 57120) ]
10   [/foo, [ 0.37520039081573 ], 5.5747001169948e-05, 0.25001793913179, a NetAddr(127.0.0.1, 57120) ]
11   [/bar, [ 0.76900768280029 ], 0.050349187105894, 0.25007368613296, a NetAddr(127.0.0.1, 57120) ]
12   [/foo, [ 0.88589346408844 ], 5.6687000324018e-05, 0.30042287323886, a NetAddr(127.0.0.1, 57120) ]
13   [/bar, [ 0.94783401489258 ], 0.049986816942692, 0.30047956023918, a NetAddr(127.0.0.1, 57120) ]
14   [/foo, [ 0.66965734958649 ], 5.5896998674143e-05, 0.35046637718187, a NetAddr(127.0.0.1, 57120) ]
```

Figure 15 Exported OSCFile into plaintext

Snippet

Snippet is a programming tool for the SuperCollider Integrated Development Environment (ScIDE). The main purpose of this tool is to store code snippets (programming code templates) in SuperCollider and recall them with or without computer shortcuts. It stores code snippets in the form of key-value pairs, where the keys are an arbitrary shortcut of the snippet in the form of a SuperCollider Symbol and the values are the actual code stored as a String. Users can also apply code snippets with some pre-defined shortcuts.

```
Snippet.addSnippet(‘egen’ -> "EnvGen.kr($$Env.adsr$$, gate, doneAction:$$2$$);")
```

Figure 16 Adding a new code snippet

Usage

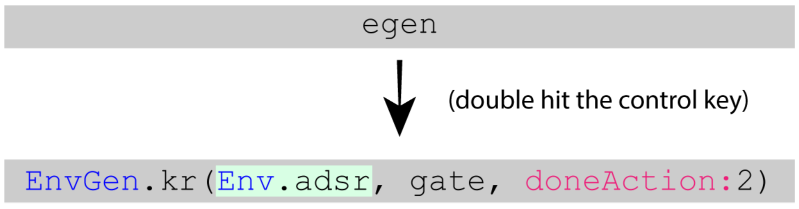
Storing code snippets

There are two recommended ways to store code snippets. The first one is to call the class method `Snippet.addSnippet` and use an Association[48] of key (short form of code) and value (the actual code) as its arguments. The snippet will automatically be stored in a global dictionary of code snippets and can be recalled. Below is an example for storing one snippet that generates an EnvGen[49] UGen with some default paremeters:

[48] An object that stores a key/value pair.

[49] An UGen that generates an audio envelope.

In this example, `'egen'` is the key of this snippet and `"EnvGen.kr($$Env.adsr$$, gate, doneAction:$$2$$);"` is the actual code to be stored. Users can set some default parameters in some parts in the code with a pair of "`$$`" before and after as indication (e.g. `$$Env.adsr$$` and `$$2$$` in this example). Additionally, `Snippet.addSnippets` can also be used to store multiple snippets.



```
egen
```

```
EnvGen.kr(Env.adsr, gate, doneAction:2)
```

Figure 17 Example code for recalling code snippets

The second way to store snippets is to type the code to be stored, then highlight it and double hit the control button. A small window as shown below will appear and users can enter the short form of the code.

Furthermore, users can also manually edit the file that stores all the snippets by calling the class method `Snippet.edit`.

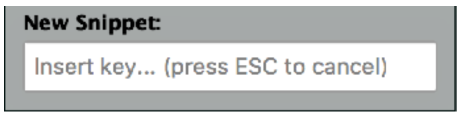


Figure 18 A window for creating new snippet

Recalling code snippets

To recall the stored code snippets, users can enter the key for a specific snippet, and then double hit the control key on the keyboard, the key will then transform to its corresponding code. The below example shows how it is being transformed.

When the code is recalled, the defaults in the code will be highlighted and can be edited instantly. These are indicated with a pair of `$$` in the definition when it is stored.

Other methods

Below is a list of other methods of the Snippet tool:

- enable - Enable the snippet mode.
- disable - Disable the snippet mode. When snippet mode is disabled, users cannot recall code snippets with the shortcut (double hitting the control key).
- saveSettings - To save the settings into file that enable or disable automatic starting and automatic save.
- save - Save the added snippets into a file.
- removeSnippet - Remove a specific snippet by indicating with the key.
- settings - Open the setting file.
- keys - Return an array of keys of all the snippets.
- search - search for code snippets with keyword

AutoArgs

AutoArgs is a tool for auto argument completion of Unit Generator creation messages in SuperCollider. Similar to Snippet, it is also a shortcut tool. The reason that I created this tool is that I hoped to generate code more efficiently during live-coding performances. This tool also serves as a mode in SuperCollider similar to Snippet, which can be enabled/disabled when coding. It fills the Unit Generators' arguments with their default values stored in the SuperCollider class file.

Usage

To use this tool, the class method `AutoArgs.enable` needs to be executed first. There are three options available regarding the content to be automatically filled for the arguments. To apply this auto completion, users only need to place the cursor within the bracket following the Unit Generator and the method that was called (e.g., SinOsc.ar( )), and then press one of the following shortcuts:

- Control + j — Fills the bracket only with the default values (e.g., `SinOsc.ar(440, 0, 1, 0)`);
- Control + k —— Fills the bracket with the default values and their keyword argument names (e.g., `SinOsc.ar(freq: 440, phase: 0, mul: 1, add: 0)`), filling with argument names can be useful when the user is not familiar with the order of arguments;
- Shift + Control + k —— Similar to the above but filling in the following format: `argument_name.kr(value)`. The advantage of using this format is that it creates control rate inputs in the SynthDef that the Unit Generator is enclosed within which can be changed and mapped to other sources immediately. `(e.g.,SinOsc.ar(\freq.kr(440),\phase.kr(0),\mul.kr(1), \add.kr(0)))`.

Chapter 4

Conclusion

This practice-based research explored various topics and approaches of using different aspects of the human body for music-making. It investigated the use of different interfaces, sonification techniques, as well as hardware and software design to achieve its goal.

The first half of this portfolio mainly explores the use of various hardware interfaces in a live-electroacoustic music setup. Through the creation of these pieces, some limitations of using these hardware interfaces were examined regarding their technical nature. Possible ways to overcome or work around these limitations were examined, proposed, and used throughout the composition process. For instance, in nootherthanthevoid, a software utilities (OSCFile) to record and playback OSC messages in binary format, two software utilities (Snippet and AutoArgs) to increase the efficiency of live coding. In contrast, the installation/performance piece *"Please Save Me"* explores the topic by investigating the possibility of manipulating the human body electronically with sound, instead of controlling sound with the human body through the use of various interfaces. The last two pieces in this portfolio, *DDS.SRY* and *DDS.CFTR*, attempt to probe the use of genetic data in fixed media electroacoustic music via sonification.

I consider the main contribution of my work to be providing examples to demonstrate the use of different interfaces for human-computer interaction (HCI) in music making and installation . This portfolio examines five different approaches, which includes the use of optical-based non-tactile gesture controller (the Leap Motion Controller), 14-channel EEG sensors (the EPOC+ headset), EMG sensors (the Myo armband), TENS machines combining with microcontroller, and DNA data sonification.

Apart from providing examples, this practice-based research also examined ways that may expand or modify some of the existing models of live-electroacoustic music creation, which I also consider as a contribution. For example, using sensors and some programming utilities to enhance the efficiency of live coding which was examined in the piece *nootherthanthevoid.*

This portfolio also tries to combine various existing techniques with technology to form new musical pieces. For instance, *(un)touched* tries to combine the use of non-tactile controller and stochastic synthesis and *Variations* tries to combine EEG sonification with performance art.

Furthermore, this research also provides some practical solutions that may aid the process of computer music creation. One example is the OSCFile, which records and playback OSC messages stored in binary format. It can be used not only to record network-based performances, but also to record data from sensors which can be reuse during the composition process. Snippet and AutoArgs help to improve the efficiency in live coding performances as well as programming in SuperCollider in general.

New technologies are evolving quickly and also becoming increasingly accessible over the last century, especially the last decades. Some of the new technologies involved in this portfolio are not "new" anymore (at the time of completing my PhD). Throughout the journey of pursuing this PhD, I had the opportunity to attend various academic events such as NIME (New Interfaces for Musical Expression) and ICLC (International Conference for Live Coding) to learn new approaches in computer music and witness what other contemporaries are exploring with new technologies. In the future, I plan to research on creating computer music with other technologies for human-computer interaction such as applying the concepts of biohacking and wearable electronics in music. While these kinds of techniques increase the intimacy between the human body and computer, it also opens up a lot of possibilities for music making.

Appendix A: Performance Information

Below is a list of performances of the pieces in this portfolio.

- (un)touched (2014)
  - ‣CrossCurrents Festival, Birmingham, UK 2014
- Variations (2015)
  - ‣BEAST FEaST 2016: Real/Unreal, Birmingham, UK, 2016
- nootherthanthevoid (2016)
  - ‣BEASTdome: Envision, Birmingham, UK, 2016
  - ‣The International Conference on New Interfaces for Musical Expression (NIME), Blacksburg, VA, USA, 2018
  - ‣International Computer Music Conference (ICMC), Daegu, Korea, 2018
- "Please Save Me" (2017)
  - ‣BEAST FEaST 2017: Figure | Landscape | Seascape | Sky, Birmingham, UK, 2017

Appendix B: Dark Matter

Dark Matter is a research project carried out by the Birmingham Ensemble for Electroacoustic Research (BEER) in collaboration with art@CMS at CERN (The European Organization for Nuclear Research). Since the beginning of my PhD study (2014), I have been an active member of the ensemble. This project started in early 2016 and is still ongoing (as of 2019). The main goal of this project was to create electronic music and visuals in improvised live-coding performances by transforming particle physics data generated from the experiments in the Large Hadron Collider situated at CERN. In order to achieve this, we designed a performance interface for accessing the data as well as some SuperCollider patterns. The project has been presented in numerous places around the globe including the UK, Greece, Canada etc. on different occasions such as conferences and music festivals. In this appendix, I will discuss the aim, the work undertaken, as well as the outcome of the project.

When we started to develop this project, there were 4 active members in the ensemble and each of us has different responsibilities. Since I was not involved in all the processes in the project, I shall focus on my contributions including gathering the source data, devising the base system for SuperCollider to read the data file, and developing some of the visuals part of the project. I will also briefly illustrate some other work done by other members in the ensemble.

Aim

The aim of this project was to create music and visuals performances by transforming the data generated by the LHC into musical materials.

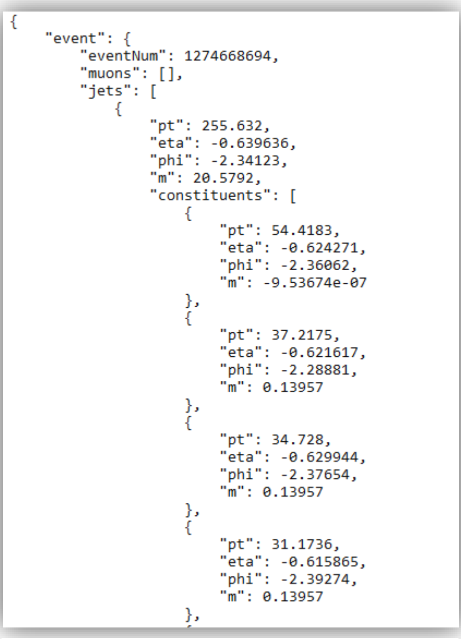
Background

Since the establishment of the ensemble (2011), it has developed a group improvisation practice involving the sharing of musical materials in networked music setups, particularly via the Utopia[50] library. For example, *Not So Mass Radio Coding* is a piece that involves players adding layers of sound modification while playing back the same synchronous sound file; in *Pea Stew*, players start by seeding a fully meshed feedback network, and then then modifying the resulting sound streams by mixing adding layers of processing. The Dark Matter project was initiated in the late 2015 by Konstantinos Vasilakos, who is one of the members of the ensemble.

[50] Utopia is a project created by BEER, with input from other authors working in a similar way. It is a platform written in SuperCollider for aiding group live coding performances, which includes a mechanism for code sharing, a chat system, as well as sharing clocks and programming object sharing. See https://github.com/muellmusik/Utopia

Methods and Processes

The development of this project involves creating a purpose-built performance system which contains three main components: a graphical performance interface, a series of SuperCollider Pattern classes, and a visualisation programme that interacts with the sound performance. The main purpose of the graphical interface is to allow the players to browse and access the data file, while the pattern classes transform the data into musical parameters. Before discussing the creation of these components, I shall briefly explain the structure of the data that we were handling and manipulating. The data file that we obtained from art@CMS was a series of particle collision events derived from what is called the scouting stream. The file contains information of more than 660,000 collision events and it is formatted in JSON[51].



```
{
    "event": {
        "eventNum": 1274668694,
        "muons": [],
        "jets": [
            {
                "pt": 255.632,
                "eta": -0.639636,
                "phi": -2.34123,
                "m": 20.5792,
                "constituents": [
                    {
                        "pt": 54.4183,
                        "eta": -0.624271,
                        "phi": -2.36062,
                        "m": -9.53674e-07
                    },
                    {
                        "pt": 37.2175,
                        "eta": -0.621617,
                        "phi": -2.28881,
                        "m": 0.13957
                    },
                    {
                        "pt": 34.728,
                        "eta": -0.629944,
                        "phi": -2.37654,
                        "m": 0.13957
                    },
                    {
                        "pt": 31.1736,
                        "eta": -0.615865,
                        "phi": -2.39274,
                        "m": 0.13957
                    },
```

Figure 19 Data file for Dark Matter

Each event is identifiable by an event number. Each of the 660000 events contains a number of jets, which by definition are "*the experimental signatures of quarks and gluons produced in high-energy processes such as head-on proton-proton collisions*"[52]. Each jet contains numerous constituents with their spatial coordinates specified in a kind of cyclindrical format

[51] JSON is a file format for Javascript.

[52] See http://cms.web.cern.ch/tags/particle-jet

commonly used in experimental particle physics, including the transverse momentum (pt), pseudorapidity (eta), azimuth angle (phi), and one non-coordinate value, mass (m). Figure 20 illustrates the structure of an event in the data file.

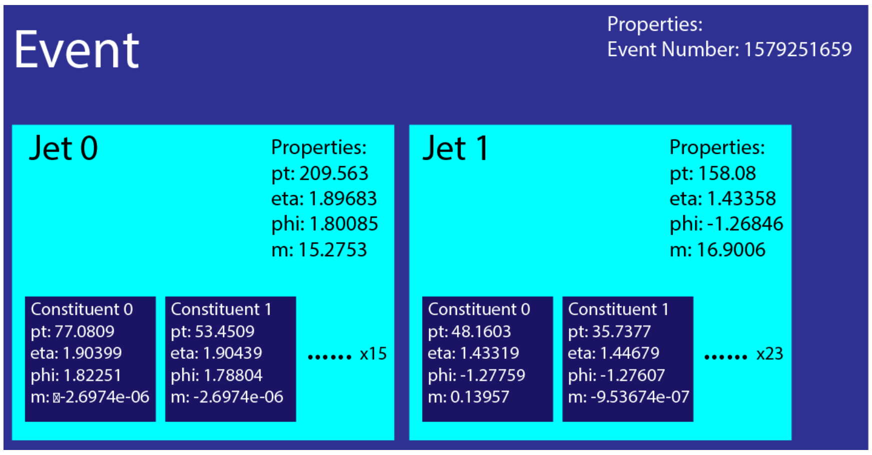
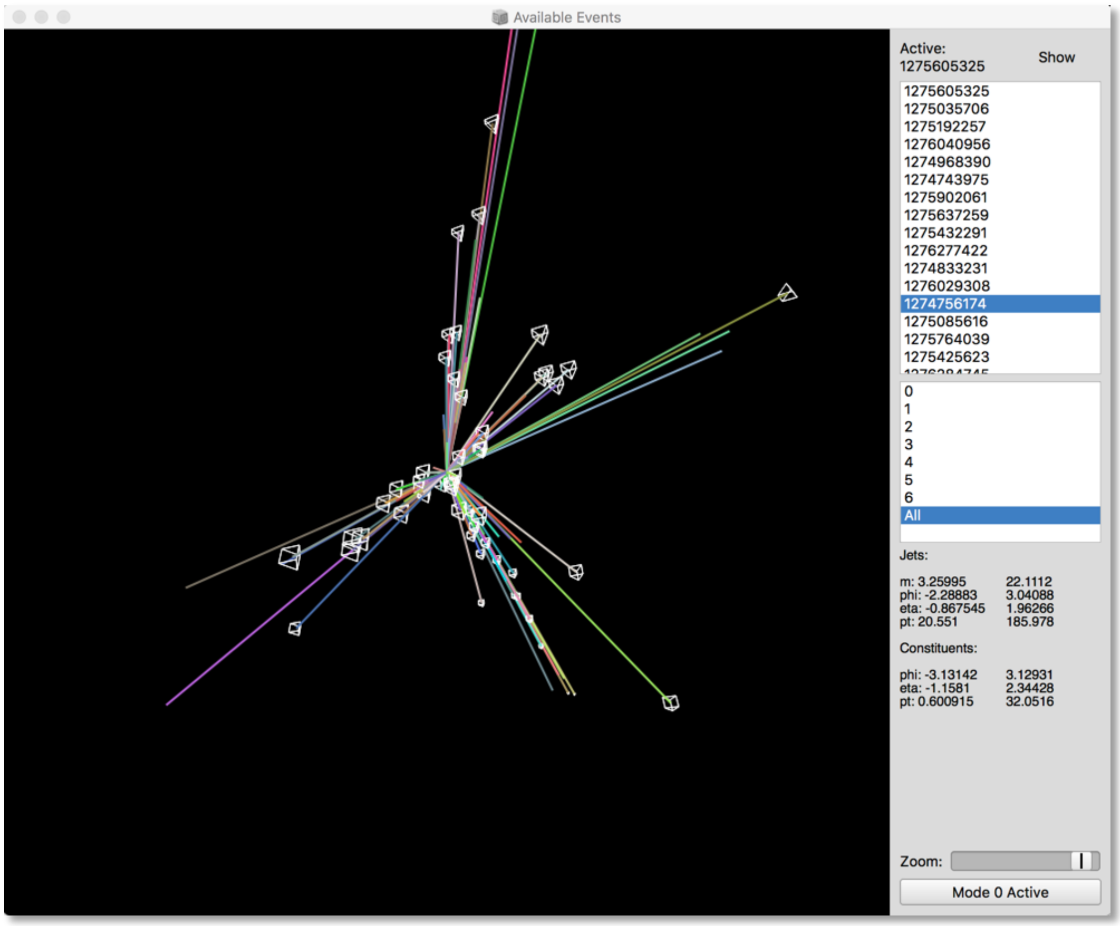


Figure 20 Structure of a pariticle collision event

In the example above (figure 20), the particle collision event with number 1579251659 has two jets and each jet has its own constituents. The left box in light-blue marked "Jet 0"represents the first jet while the right one marked with "Jet 1" represents the second jet. They contain 15 and 23 constituents respectively. Each small box in dark blue represents a constituent with its coordinate stated; for instance, "constituent 0" of "Jet 0" has its pt, eta, phi of 77.08, 1.90 and 1.822 respectively.

In order to access the data during the performance, a graphical interface was created by Scott Wilson, the leading member of the ensemble. The interface allows players to view in detail each collision with all the parameters as well as to activate them. It also offers a three-dimensional graphical representation of the chosen event.



```
Active:
1275605325

1275605325
1275035706
1275192257
1276040956
1274968390
1274743975
1275902061
1275637259
1275432291
1276277422
1274833231
1276029308
1274756174
1275085616
1275764039
1275425623

0
1
2
3
4
5
6
All

Jets:

m: 3.25995        22.1112
phi: -2.28883     3.04088
eta: -0.867545    1.96266
pt: 20.551        185.978

Constituents:

phi: -3.13142     3.12931
eta: -1.1581      2.34428
pt: 0.600915      32.0516
```

Figure 21 The graphical interface for performance of Dark Matter

Figure 21 displays the graphical interface for Dark Matter. On the right-hand side, the top section is a list of collision events that can be selected, below is another list showing the minima and maxima of the parameters of the jets of the selected event. Below the jet list are the minima and maxima of the values of the constituents contained in the selected jet. On the left-hand side is a three-dimensional graphical representation of the selected event, indicating also the selected jet if any… On the bottom right corner, there is a button labeled “Mode 0 active”, this is a button to switch between different visualisation modes on the screen for a particular player (visualisation will be discussed later in this appendix).

SuperCollider Patterns

For easy access of the collision data, there are 9 purpose-built SuperCollider patterns[53] created for this project. The table below briefly illustrates their function:

Table 1 Dark matter SuperCollider Patterns

Pattern	Description
Pjet	Access a parameter of an indicated jet
PjetS	As Pjet, but scaling values between 0 and 1 based on minima and maxima in the active event
Pconstituent	Access a parameter of an indicated constituent of an indicated jet
PconstituentS	As Pconstituent, but scaling values between 0 and 1 based on minima and maxima in the active event
PallConstituents	As Pconstituent, but rather than specifying first a jet, groups all constituents as one ordered collection
PallConstituentsS	As PallConstituents, but scaling values between 0 and 1 based on minima and maxima in the active event
PnumJets	Output the number of jets in the currently selected event
PnumConstituents	Output the number of constituents of a specific jet in the currently selected
PtotalConstituents	Output the number of constituents in the currently selected event

[53] These SuperCollider patterns can be found at: https://github.com/KonVas/DarkMatter (Accessed July 2019)

Visualisation

There are two versions of the visualisation programme created for this project, which mainly serve as an interactive accompaniment for the music. They are both written in Processing with the use of the oscP5 library. I was responsible for creating the first version while the second one was created using the first version as a framework. Both versions are made of two layers of graphical elements. One is the code-relay, which is shows the code of each player as it is executed. The second is a dynamic visualisation of the selected particle event for each player. The code-relay is based on the usage of the CodeRelay class included in Utopia, which was intended for code sharing during networked music setup. With Utopia set up, players' SuperCollider automatically broadcasts the code they execute to the local network via OSC with the address pattern "/codeRelay". The visualisation programmes can be run on one of the computers of the performers or on an independent machine. It joins the Utopia network automatically. Upon receiving the "/codeRelay" OSC message, it displays the code labelled with the name of the player. The second layer, in the version I devised, displays all constituents of the selected event of each player in two modes. In one mode, constituents are displayed as spheres with different colours determined by their cartesian coordinates converted from the pseudorapidity coordinates. When a constituent is accessed (e.g. a musical note which has parameters mapped to the value is played), the corresponding sphere flashes. In the other mode, jets are displayed as polygons with the vertices correspond to the coordinates of the constituents in the jet.

Outcome

The project has been performed around the globe since 2016 in various occasions such as conferences, music festivals and some other dedicated concerts. Apart from performance, there were also workshops held in various places. Below is a list of selected performances:

- Durham, UK (University of Durham)– 29 Oct 2018
- Manchester, UK (MANTIS Festival, University of Manchester) – 29 Oct 2018
- Aberdeen, UK (University of Aberdeen) – 20 Sep 2018
- Brighton, UK (University of Sussex) – 21 Feb 2018
- Bangor, UK (Bangor Music Festival) - 03 Feb 2018
- Hamilton, ON, Canada (International Conference for Live Coding) – 13 Oct 2016
- Birmingham, UK (premiere) (Birmingham Open Media) – 17 Mar 2016

Bibliography

Ben-Tal, Oded, and Jonathan Berger. "Creative Aspects of Sonification." *Leonardo* 37, no. 3 (2004): 229–32.

Cocker, Emma. "Performing Thinking in Action: The Meletē of Live Coding." *International Journal of Performance Arts and Digital Media* 12, no. 2 (July 2, 2016): 102–16.

Collins, Nick. "Live Coding of Consequence." *Leonardo* 44, no. 3 (2011): 207–30.

Cornett-Murtada, Vanessa. "From Beta to Theta: Human Consciousness, Hypnosis and Music Performance." *College Music Symposium* 49/50 (2009): 265–70.

Elsenaar, Arthur, and Remko Scha. "Electric Body Manipulation as Performance Art: A Historical Perspective." *Leonardo Music Journal* 12 (2002): 17–28.

Freeman, Jason, and Akito Van Troyer. "Collaborative Textual Improvisation in a Laptop Ensemble." *Computer Music Journal* 35, no. 2 (2011): 8–21.

Hermann, Thomas, Andy Hunt, and John G. Neuhoff. *The Sonification Handbook*. Isd, 2011.

Linz, Rainer. "Towards the Design of a Real-Time Interactive Performance Sound System." *Leonardo Music Journal* 6 (1996): 99–107.

Lucier, Alvin, and Douglas Simon. *Chambers: Scores by Alvin Lucier*. Middletown, UNITED STATES: Wesleyan University Press, 1980.

Lucier, Alvin, and Arthur Margolin. "Conversation with Alvin Lucier." *Perspectives of New Music* 20, no. 1/2 (1981): 50–58.

Magnusson, Thor. "Algorithms as Scores: Coding Live Music." *Leonardo Music Journal* 21 (2011): 19–23.

Miranda, Eduardo Reck, and Andrew Brouse. "Interfacing the Brain Directly with Musical Systems: On Developing Systems for Making Music with Brain Signals." *Leonardo* 38, no. 4 (2005): 331–36.

Rosenboom, David. "The Performing Brain." *Computer Music Journal* 14, no. 1 (1990): 48–66.

Rosenboom, David, and David Paul. "Biomusic and the Brain." *Performing Arts Journal* 10, no. 2 (1986): 12–16.

Straebel, Volker, and Wilm Thoben. "Alvin Lucier's Music for Solo Performer: Experimental Music beyond Sonification." *Organised Sound* 19, no. 1 (April 2014): 17–29.

Visi, Federico. "Methods and Technologies for the Analysis and Interactive Use of Body Movements in Instrumental Music Performance." Thesis, University of Plymouth, 2017.

Weichert, Frank, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. 'Analysis of the Accuracy and Robustness of the Leap Motion Controller'. *Sensors* 13, no. 5 (May 2013): 6380–93. https://doi.org/10.3390/s130506380. Accessed July 2019

Weinberg, Gil, and Travis Thatcher. "Interactive Sonification: Aesthetics, Functionality and Performance." *Leonardo Music Journal* 16 (2006): 9–12.