Creating an Algorithmic System to Participate in Freely Improvised Performance

Mark Sandford

Music Technology Project Final Year Dissertation

BMus in Music Technology

Reid School of Music
University *of* Edinburgh
August 2021

Supervisor 1: Prof. Stefan Bilbao
Supervisor 2: Prof. Raymond MacDonald

Abstract

This dissertation explores how technology can be used alongside performance to enhance a freely improvised piece of music. It explores a method of musical communication designed to deliberately step away from the traditional idea of notation and conduction, combining this with various techniques of musical analysis through digital signal processing in order to create a 'feedback loop' of information, presented in an abstract but inherently musical fashion.

Declaration

I do hereby declare that this dissertation was composed by myself and that the work described within is my own, except where explicitly stated otherwise.

Mark Sandford
August 2021

Acknowledgements

I wish to express my gratitude to my supervisors for their advice which helped direct and transform this project into its final form; I would like to extend that thanks to all the other members of staff from the Reid School of Music who helped shape my university journey over the past four years, and to the friends and family who supported me along the way.

Most importantly, I need to thank my flatmates for their care, pasta, and patience; my brother for his technical advice, late-night companionship, and listening ear; and my girlfriend, as without her motivation and support this dissertation would still be a collection of blank pages.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Graphic scores are a common technique of composition for freely improvised music. In the context of this dissertation, the phrase 'free improvisation' is used to refer to a performance where the musical outcome is not known before the first note is sounded. Performers within a free improvisation are free to make their own decisions about key, tonality, or rhythm but may be given some direction to follow, often abstract in nature.

A graphic score may be as simple as a few spots of colour on a page or a dense manual with different instructions for different performers. At either end of the spectrum, emphasis is put on the performer's own interpretation of the medium, which can vary wildly from person to person. This project looks at adapting this concept into a digital display, so that the score changes as the music progresses. In this way the display is not really a score in the traditional sense but more akin to a virtual conductor.

1.2 Development

The initial aim of the project was to create an interface between a performer or ensemble and conductor which embraced the freedom presented through an abstract medium such as a graphic score. Three objectives were identified:

1. To investigate, code, and prototype a suitable interface for controlling a graphical display as part of a 'free improvisation' performance.

2. To explore how the interface could heighten a performance by additionally being used as a way of interacting with real-time DSP-based effects.

3. To examine how such an interface could help people be part of a performance even with little or no knowledge of music.

However, the project aims were revised early on to narrow the scope of the project and focus on digital signal processing (DSP) techniques. The objectives were changed and expanded to focus on algorithmic composition in place of a traditional conductor, and the third objective was dropped to bring the project in line with current coronavirus government guidelines.

This second revision of the project attempted to develop the concept within the wider sphere of 'music technology' by creating a graphical display from which a free improvisation could be performed, and which would change and adapt based on real-time analysis of the improvised material. Through this, the project attempts to blend the fields of algorithmic composition and human computer interaction with the more traditional performance aspects of improvisation, resulting in a performative work that cannot be described as properly composed nor fully improvised.

The revised objectives were as follows:

1. To code a system that can make decisions to control a graphical display autonomously, based upon variables relating to previous decisions and outcomes.

2. To find out what analysis is required to create variables that can influence those decisions.

3. To explore DSP-based effects and how these can be implemented in a performance led by an algorithmic system.

1.3 Related Work

William Hsu describes improvisation systems which accompany performers by "playing" virtual instruments and making 'simple high-level decisions' over their musical gestures [1, 2]. In a similar vein, there are other notable interactive systems such as George Lewis' *Voyager* or Adam Linson's *Odessa*, both designed to accompany a live performance with the generation of audio in real-time [3, 4].

However, there are few documented examples of using these techniques in a free improvisation outside of generating audio. Roth et. all describe a method of directing an improvisation with 'animated puppets', but these were controlled by people instead of algorithmically[5]. This system blends these two contrasting approaches to examine previously unexplored areas of improvised performance, by allowing the system to participate in decision making without limiting the performers' ability to express themselves musically.

From these, and other similar systems key areas of analysis were identified which ended up being implemented [6, 7]:

From these, and other similar systems, key areas of analysis were identified which ended up being implemented:

- Volume or loudness.
- Space between notes, or length of silences.
- Speed at which notes changed.
- Playing in an extreme or normal range (pitch following).
- Sporadical notes, or large jumps in volume or pitch.

An extra criterion, timbral analysis, was identified but was not implemented due to the technical difficulty of calculating the analysis.

Chapter 2

Software

As was suggested in the project proposal, the system was coded using Max MSP (discussed in section 2.1). Max provided a good fit for the project as it already includes several objects design for audio analysis, interacts well with MIDI for ease of comparing detected notes [1], and handles audio input easily through an external audio interface. Additionally, the Jitter environment was able to host the visual portion of this project(section 2.2).

2.1 Audio analysis

The areas identified in chapter 1 were used as focus points for analysing the audio data. Eventually, four variables were decided upon which could represent a variety of musical textures that would occur in a free improvisation. These identify the performer's 'contributions to the ever-evolving texture', rather than analysing specific notes and melodies [8].

The analysis for these variables takes place within two identical **pitchTracking** subpatches. Each variable is represented as a scale from 0 – 1000, with each end representing one extreme. For example, the loudness variable has a **hslider** object associated with it where 0 corresponds to 'quiet' and 1000 to 'loud'. The default value is 500 and increases or decreases depending on the measured amplitude of the incoming signal.

To get the best reading from the incoming audio data, some initial clean-up is performed. This happens at the same time as a Fast Fourier Transform (FFT), as Max includes the **pfft~** object to perform a real-time Short Term Fourier Transform (STFT). The FFT size used is 2048, which allows for narrow windows for more accurate pitch detection across a variety of instruments. For efficiency, the **pfft~** object only calculates a real FFT, producing a spectrum from 0Hz to the Nyquist frequency. The FFT uses a Hanning window function and a hop size of 4. The minimum frequency which can be detected is calculated from the sample rate:

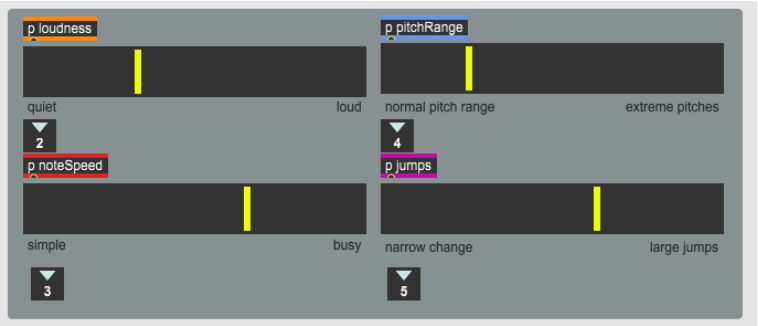

Figure 2.1: Screenshot of the four main analysis variables, displayed as sliders.

$$fundamentalFrequency = \frac{sampleRate}{numberofSamples} \tag{2.1}$$

The audio is filtered inside the FFT loop by trimming any frequency bins that lay outside the range 100 - 20000 Hz. Any bins with an amplitude below a threshold amount are zeroed before the frame data is exported. The pitch is estimated by looking for the frequency bin with the highest amplitude within the current frame and is stored for later use. To get an accurate pitch estimate, each input has a gain slider within the top-level patch. It was found that the input level had to be increased much higher than one usually would while recording audio, so having the gain controllable inside Max allows for the user to record the audio at the same time in another application if they wish.

2.1.1 Loudness

The overall loudness of the audio is calculated by measuring the amplitude of the incoming signal. A larger amplitude means that the *loudness* variable is increased and vice versa. When the audio is 'silent', *loudness* is rapidly decreased. This is determined in several stages. Firstly, the absolute value is taken and then the incoming audio data smoothed using the **slide**~ object. A gate is applied that only allows signals above a certain threshold through, as some inherent noise in the signal was creating false measurements. Instead of being discarded, signals below the gate threshold are measured to see if they are below an even smaller threshold, representing true silence.

2.1.2 Speed of changing notes

Signals that make it through the first gate of the *loudness* stage are rapidly compared against each other to see if incoming signals are louder than the previous - indicating either the attack of a new note or a crescendo. The length of time before the next

louder signal is measured to find the average length of each note. To filter out held notes, any note length that is exceptionally short is discarded - these were usually the case of a sustained note rapidly increasing in volume (resulting in false triggers) rather than a new note attack.

The note length is used to change *noteSpeed*. A short average note length leads to an increase towards the 'busy' end of the scale, while a long average note length decreases the variable towards the 'simple' end.

2.1.3 Range of notes

When setting up the performance, each instrument's comfortable range or tessitura must be input. A pitch value is estimated using a fast Fourier Transform (FFT), as described above, and from then it is simple to compare whether the performers are playing notes at the extremes of their range or not. As with loudness and note speed, this analysis shifts *pitchRange* from between 'normal' to 'extreme'.

If the user wanted, they could choose to narrow the 'comfortable range' even further, for example just to include a single octave. In this instance, the extremes of *pitchRange* are better thought of as 'narrow' and 'wide'.

A timer is used to clear the list of values used to calculate the average every 2500ms, and update *pitchRange* every 500ms. By running this timer from the main patch, the computational load is decreased, since most aspects of the patch are using the same internal clock to update together.

2.1.4 Size of intervals

The final analysis area focuses on the size of the interval between consecutive notes, and the difference between the dynamic levels of consecutive notes. Both methods of analysis employ the same method of calculation: taking the mean average between the absolute difference of the last two note values, for pitch and *loudness* respectively. Like the previous section, the frequency is estimated from the initial FFT in the patch. It is then converted into a MIDI note, allowing easier comparison along a linear scale.

The two averages, pitch and *loudness*, are compared and the more extreme value was scaled and used to control the *jumps* variable. Large differences push *jumps* toward the 'large' end of the scale, while smaller differences bring it towards 'narrow'. If the average pitch change is zero, this would register as a decrease. The *jumps* variable is updated at the same rate as *pitchRange*, using this main control timer.

2.1.5 Combining two inputs

For a performance, the Max patch was designed to take two audio inputs. All analysis runs identically between channels 1 and 2, but the data needs to be combined and

scaled into a useable range. This is achieved through two methods within the **analysis** subpatch, each trying to get the most interesting data for a specific variable.

An improvisation that works well relies on transferring 'intentionality through sounds', and often the easiest way to understand an intention is through loud or obvious cues [8]. It was decided that the maximum value would be taken for both *loudness* and *jumps*, as this represents a dominating aspect of the performance at that point. This means that if one instrument were to play loudly it could take the lead over the other, much as we would expect within a traditional ensemble. For *noteSpeed* and *pitchRange*, it was decided that taking a mean average would produce the most usable results.

Although this piece was designed for two players, it is possible to have just one instrumentalist by routing the same audio input to each channel in Max. However, in this case the performance load is still as heavy as if two instrumentalists were performing since the analysis is being run twice.

2.2 Visual display

In addition to audio processing, Max is host to video and graphics tools with Jitter. Jitter has a built-in physics engine which, when the environment is limited to two dimensions, makes it easy to create fluid motion for a graphical display. To create a display that conveys information to a performer, a ball object is created within the **jit.phys.world** translating to two axes, representing sparseness and business along the horizontal and a gradient of voiced to unvoiced along the vertical. In free improvisation, to play 'unvoiced' is to make sound in an unusual way (for example, blowing air through a saxophone without sounding a note).

2.2.1 Jitter

The display plays a crucial part in the performance as it is visible to both the audience and the performers, so some limited visual setup options for customising the display have been implemented. The ball size, ball colour and background colour are changeable. Additionally, the window can be resized, and the aspect ratio can be changed. This is to make sure that the boundaries of the Jitter world are correct, so the ball can use all the available space while not disappearing off screen.

In the jitterRender subpatch, management of the objects within the **jit.phys.world** takes place. The jitter environment is redrawn every 5ms, or rendered at a rate of 12 frames per second. Objects like the **jit.phys.body**, the ball within the physics simulation, has sends and receives issued to them to all for movement to be directed from the separate **decisionMaking** subpatch.

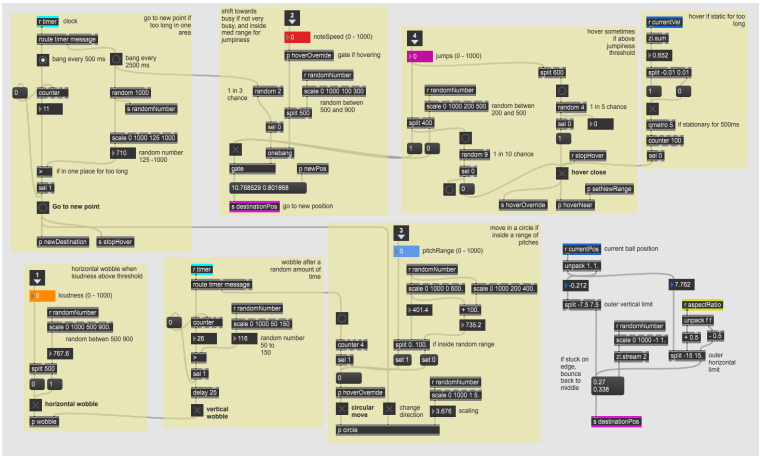

Figure 2.2: Screenshot of a Max patch. Each area in yellow represents a seperate area of decision making.

2.2.2 Movement routines

After the analysis stage, there are four main variables used to determine the outcome of the performance, as described previously: *loudness*, *noteSpeed*, *jumps*, and *pitchRange*. Each variable has a value between 0 and 1000. Additionally, the position and velocity of the ball can be requested with a **recieve currentPos** or **recieve currentVal** command. This information is used to move the ball across the plane by triggering one of several movement routines (see section 2.2.3). These movements include:

- *Wobble* (vertical or horizontal) - the object would shake left to right or up and down.
- *Go to new point* - the object would move to a new space within the display environment.
- *Circular move* - the object would move in a circular motion around a point.
- *Hover close* - the object would pick a few points close together and quickly move between them.

2.2.3 Decision making

Most of the decision making involves checking a value inside a range or against a threshold. These thresholds are constantly updated to allow for a less predictable outcome.

Some decisions are made independently, while in other cases several variables are combined to arrive at an outcome. To reduce computational load, every 2500ms the patch updates a single random number between 0 and 1000, which is then scaled for

each application. Additionally, a 'bang' message is sent every 500ms to iterate various counters and comparisons.

- The two *wobble* routines are triggered independently. The horizontal *wobble* triggers when the *loudness* is greater than a random number scaled between 500 and 900.
- The *vertical wobbe* alternates at random intervals between 25000 and 75000ms. This is one of the few routines not controlled by audio analysis, instead constantly introducing variety to inspire new improvisatory ideas. Since the performer does not know what aspects of their playing controls each movement, it works well to create contrasting textures.
- The *go to new point* routine is triggered at random intervals between 62500 to 500000ms. Since this can move the ball from one side of the display to the other, it can have a drastic change on the improvisation so was chosen to happen quite rarely.
- The *circular move* routine again triggers rarely. If the *pitchRange* variable is between a randomly determined range of values, there is then a one in four chance of starting the routine. Additionally, the size of the circle can change based on a random probability and when the *go to new point* routine is triggered the direction of travel changes.
- The *jumps* variable is used to trigger the hover close routine if it is either exceptionally low or high and a 1 in 5 check is met. When this routine is triggered, it will stop all other routines currently running. Additionally, this routine stops if another is triggered after it.
- If both *noteSpeed* and *jumps* are low the ball is pushed towards the right side of the screen, suggesting a busier style of playing.

2.3 Alternative methods

Additional research was done into different methods of real-time analysis. Two objects were found that appeared especially suitable: **fiddle~** is a 'monophonic or polyphonic maximum-likelihood pitch detector', while **bonk~** is especially suited to percussive sounds [9, 10]. Due to the varying nature of timbre and sounds present in free improvisation, having both methods of analysis would be beneficial for this application.

Unfortunately, these objects are outdated and no longer supported on current versions of Max. The open-source alternative Pd contains both objects which work well but lacks the graphical applications of Jitter, and Max's performance optimisation. Therefore, the decision was made to contain the project solely within Max. As the

source code for these objects are available online, future iterations of the project could examine these to implement more advanced analysis techniques.

There are other potential methods of detecting attacks [9]; the method used in this project (section 2.1.2) was chosen for simplicity, but additionally could be expanded to detect crescendos or decrescendos.

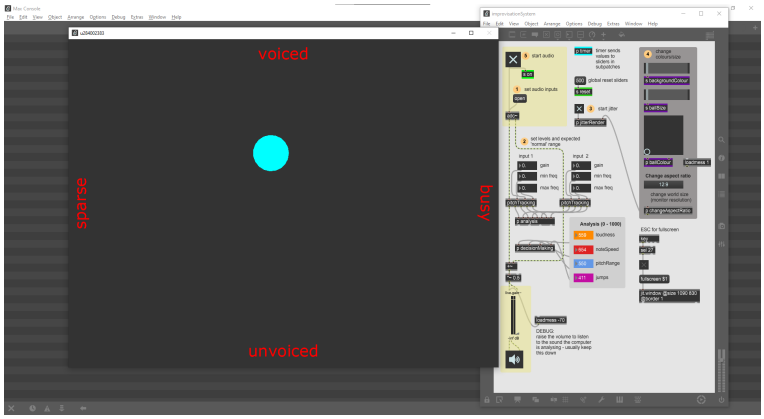

Figure 3.1: Screenshot of the patch used in a performance setting.

Chapter 3

Performance

Several performances using the system were held with two musicians each time. One performance involved an electric bass guitar and electric piano, and the other a guitar and trombone. The Max patch was run on a Microsoft Surface Pro laptop with its output mirrored on a television screen. Audio was captured from the two musicians through a two-channel M-Audio M-Track 2X2M audio interface. For the bass, piano, and guitar, a direct output could be taken from the instrument since both instruments were electric. The trombonist performed into an AT2035 microphone.

There were some limitations of the instrumentation and setup that should be considered. Firstly, the system is unable to identify polyphonic notes. All players were suggested to stick to monophonic lines for most of their playing but encouraged to let the music take precedent and to fully explore their improvisation. Additionally, while taking a direct line input from each instrument allows for the best chance at pitch recognition with a clear signal, it comes with the caveat that any different noises that the performer creates with their instrument in an unorthodox way may not be picked up.

In the first performance, the bass went through channel 1 of the **adc~** object and the keyboard went through channel 2. Some initial testing was run to set levels for the most accurate recognition. The bass had its lower octave set as the expected range, so that playing very high up or with harmonics would be classified as 'extreme pitches'. Although the musical output was good, a couple of problems arose in the audio analysis: the lower frequencies of the bass were not able to be detected very reliably, and the lack of polyphonic recognition restricted the pianist's ability to perform fully.

The second performance was more successful as the instrumentation allowed for more accurate analysis. Additionally, the use of a microphone instead of a direct output allowed the trombonist to experiment with more abstract ways of performing the instrument when the display moved towards 'unvoiced'.

Chapter 4

Conclusions

While this project works as a proof of the concept, it is still a way off from being fully realised. Unfortunately, the analytical sections of code do not work reliably across the whole range of an instrument. The pitch detection will often have trouble determining the correct octave, which can transfer inaccurately to *pitchRange*. Additionally, the patch has difficulty identify the pitches of notes played very quickly, which may confuse the *noteSpeed* variable. As noted in chapter 2, there exist much more efficient and accurate ways of analysing pitch and note attacks such as the **bonk~** or **fiddle~** objects. If this project were to be developed further, these pieces of code should be examined to see if it is possible to translate them to the more recent versions of Max.

Despite this, several musical results were obtained and it is clear that there is much more work to be done within this exciting new field of research. While this project was not designed specifically for a certain instrument, narrowing the choice for the next stage of development would allow the analysis methods to be specifically tailored for a performance rather than the broader methods described in this paper.

During the project development, the third objective was dropped due to time constraints. If this work continues, this is an area that should be researched as it presents a method for the system to interact directly with the audience and thus will be able to show a clearer demonstration of the system's intentions.

Another notable area of development that could be implemented in the future would be to allow the system to take control of the look of the graphical display. This would be relatively simple to implement, as within the top-level Max patch it is already possible to change the size and colour of the ball and change the background from light to dark. However, within the current set of rules defined for the piece these have no meaning for a performer. In the same way that the outcome of the musical analysis is used to trigger a set of predetermined movement routines, variables like the ball colour could be controlled very easily by the system.

Appendix A

Resources

A.1 Max

Max 8 is software from Cycling '74, available to download from `https://cycling74.com/downloads/`.

Miller Puckette's Max externals - [fiddle~], [plonk~], and [bonk~]- are available to download from `http://vud.org/max/` (Macintosh 32 bit only).

A.2 Performance resources

The audio interface used was the M-Audio M-Track 2X2M (`https://m-audio.com/m-tracks/2x2m`). Trombone was recorded with anaudio-technica AT2035 (`https://www.audio-technica.com/en-gb/at2035`). For the recorded performance, audio was recorded using Audacity (`https://www.audacityteam.org/` and the screen was recorded using OBS Studio (`https://obsproject.com/`).

Appendix B

Music Technology Project Proposal

B.1 Goals

This project aims to explore how technology can enhance a 'free improvisation' performance through the combination of visuals stimuli, digital signal processing (DSP), and live improvisation, and how technology would enable performers to join in an improvisation without necessarily knowing any musical theory or terms.

B.2 Objectives

1. Investigate, code, and prototype a suitable interface for controlling a graphical display as part of a 'free improvisation' performance.

2. Explore how the interface could heighten a performance by additionally be used as a way of interacting with real-time DSP-based effects.

3. Examine how such an interface could help people be part of a performance even with little or no knowledge of music.

B.3 Background

This project is based off a previous piece of coursework for the *Improvisation as Social Process* class, led by Raymond MacDonald. In this course we explored various relationships between improvised music, graphic scores, and conduction. This led to my project for that course revolving around the idea of a 'conducted graphic score' – using a graphics tablet to move shapes around on a screen to develop a free improvisation and react in real time.

For more detail, the analysis of the previous performance is in the appendix so here I will simply summarise how the performance worked and then describe how I am looking to develop this for a larger scale project. One performer, the 'conductor', uses a graphics tablet to move a ball around inside a jitter environment within Max. The other performers, the 'improvisers', are seated facing the conductor and a screen displaying the ball within the jitter environment. Both parties work together to craft an improvisation either based on the ball's location, size, and colour. These could represent any characteristics the performers choose, however in taking this further I could investigate some of the features described by Hsu, 'intensity... timbre, and rhythm' [1].

To refine the system above, I need to investigate what is needed from an interface, and how this can be used to best translate the user's gestures. A touch screen lends itself well to gesture control, while a game controller is a shape with known gestures already well established. Alternatively, an Arduino opens many possibilities for gestural control [11], but complicates the process of taking an input when compared to devices with a 'low degree of freedom' such as '2D touchpads' [12].

To take this further, I am interested in both refining the system above, and developing a part of the system that could translate the conductor's gestures into audio effects to enhance the performance. For example, while leading the performance, the conductor might be able to select a couple of seconds of sound just passed to sample and loop underneath the rest of the performance. The conductor could use the interface to add effects like delay, reverb, or distortion to players within the ensemble. Ultimately this project looks to turn the conductor into a digital improviser and performer themselves.

Finally, a system like this could potentially be used to support musical communication outside of traditional language, conduction, or notation. In the research stage of this project I have been researching how human-computer interaction (HCI) has been used within performance and improvisation, and how this project could benefit the field by creating a means of communication that doesn't rely on any knowledge of musical terminology. In my research leading to the development ideas for this project, I encountered an app (Mood Conductor), which follows similar design principles to what I'm trying to achieve, albeit within a slightly different musical context [13].

B.4 Risks and mitigating the spread of COVID-19

Obviously, there is a lot of uncertainty surrounding the year due to the coronavirus pandemic. While this project ultimately is aimed for performances within larger ensembles, it can easily be adapted to work specifically within smaller groups or duets. Most of the design, prototyping, and coding can be completed without any performers present, so there need not be any in-person interaction until the *Performance and*

Further Development stage. As the DSP-based part of the performance requires the instrumentalists to be amplified with microphones, it would be easy to spread them apart - while still ensuring that everyone can be heard - in line with the current Government guidance at that time. There are no additional risks outside of normal concert/performance considerations.

B.5 Supervisors and assistance

As this project will be using Max, I feel that Martin Parker or Tom Mudd would be a suitable choice for Supervisor. We have also asked Raymond MacDonald for his input on the improvisation and performance part of the project.

B.6 Project needs

A suitable performance space

The space needs to be large enough to facilitate any extra measurers recommended within the Government's guidelines for coronavirus in Scotland. It should include either a projector or a large TV for displaying the graphical portion of the performance. Alison House Atrium or the Reid Concert Hall would be suitable for larger scale performances while Music Studio 1 would be suitable for a duet.

Interface devices

Several devices will be examined: Wacom Intuos Graphics Tablets are available from the Music Studios Store, I own several Xbox 360 game controllers, Arduino kits are available to borrow from ECA (if needed).

Recording and sound equipment

The instrumentalists would need to be recorded and amplified to allow for DSP-based effects. Depending on the scale of the performance, headphones or additional monitors may need to be set up but this level of complexity can be adjusted to suit the project's needs. This equipment would be available to borrow from the Music Studios Store.

Computer and software

Max contains Jitter, for graphical processing; handles audio, DSP, and MIDI; and communicates well with various interface devices. This could be run on a reasonable powered laptop, although a university desktop would ideally be better.

Performers

Any performance can be adapted to suit the number of improvisers. Performers will supply their own instrument. Instruments that are portable, easy to record, and capable of producing a variety of sounds would be best. There is no requirement for the same improvisers to be present throughout, although this could be beneficial.

B.7 Timeline

Planning and research stage (September to October)

- Gather resources on relevant projects within HCI and music and use these to develop the concept and specify how this device would be used within a performance, and what functions it has. Explore and find any similar freely improvised performances and evaluate if these can help refine my ideas.

Prototype and design stage (November to January)

- Prototype and code a suitable interface (Objective 1). This should be able to interact with Max, translating a variety of gestures into variables to be used for graphical and audio processing. The graphical design concept will develop alongside as gestures are developed.
- Experimentation with how DSP-based effects or sampling could be used within the performance and how these might link to the gestures designed previously (Objective 2).
- Finalisation of software. At this stage I should be able to start using this in small performances for feedback, evaluation, and refinement.

Performance and further development stage (Feburary to May)

- Further refinement and investigation into real world applications (Objective 3).
- Collection of findings, planning and drafting of dissertation. Trails within small group performances.
- Write-up, consolidation of findings, evalution.
- Presentation - depending on COVID restrictions, this could include a performance.

Bibliography

[1] W. Hsu, "Design issues in interaction modeling for free improvisation," in *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, ser. NIME '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 367–370. [Online]. Available: https://doi.org/10.1145/1279740.1279821

[2] ——, "Strategies for Managing Timbre and Interaction in Automatic Improvisation Systems," *Leonardo Music Journal*, vol. 20, pp. 33–39, 12 2010. [Online]. Available: https://doi.org/10.1162/LMJ_a_00010

[3] G. E. Lewis, "Too many notes: Computers, complexity and culture in "voyager"," *Leonardo music journal*, vol. 10, pp. 33–39, 2000.

[4] A. Linson, C. Dobbyn, G. E. Lewis, and R. Laney, "A Subsumption Agent for Collaborative Free Improvisation," *Computer Music Journal*, vol. 39, no. 4, pp. 96–115, 12 2015. [Online]. Available: https://doi.org/10.1162/COMJ_a_00323

[5] B. Hayes-Roth, E. Sincoff, L. Brownston, R. Huard, and B. Lent, "Directed improvisation with animated puppets," in *Conference Companion on Human Factors in Computing Systems*, ser. CHI '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 79–80. [Online]. Available: https://doi.org/10.1145/223355.223438

[6] J. Bryson, A. Smaill, and G. Wiggins, *The reactive accompanist: applying subsumption architecture to software design*. Citeseer, 1992.

[7] W. F. Walker, "A computer participant in musical improvisation," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 123–130. [Online]. Available: https://doi.org/10.1145/258549.258629

[8] P. Steinbeck, "George lewis's voyager," in *The Routledge Companion to Jazz Studies*, 1st ed. Routledge, 2019, pp. 261–270.

[9] M. S. Puckette, T. Apel, and D. D. Zicarelli, "Real-time audio analysis tools for pd and msp," 1998.

[10] A. Robertson and M. Plumbley, "Post-processing fiddle : A real-time multi-pitch tracking technique using harmonic partial subtraction for use within live performance systems," 08 2009.

[11] R. F. Cadiz, "icons," in *Music Proceedings of the International Conference on New Interfaces for Musical Expression*. Porto Alegre: UFRGS, 2019, pp. 29–31. [Online]. Available: http://www.nime.org/proceedings/2019/nime2019_music007.pdf.

[12] J. Bowers and S. O. Hellström, "Simple interfaces to complex sound in improvised music," in *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 125–126. [Online]. Available: https://doi.org/10.1145/633292.633364

[13] M. Barthet, F. Thalmann, G. Fazekas, M. Sandler, and G. Wiggins, "Crossroads: Interactive music systems transforming performance, production and listening." San Jose: CHI, 2016. [Online]. Available: http://qmro.qmul.ac.uk/xmlui/handle/123456789/12502