Improvising with the Threnoscope: Integrating Code, Hardware, GUI, Network, and Graphic Scores

Thor Magnusson
Department of Music
University of Sussex
Brighton, BN1 9RH, UK
t.magnusson@sussex.ac.uk

ABSTRACT

Live coding emphasizes improvisation. It is an art practice merging the act of musical composition and performance into a public act of projected writing. This paper introduces the improvisational features of the Threnoscope system, which implements a live coding micro-language for drone-based microtonal composition. The paper discusses the aims and objectives of the system, elucidates design decisions, and describes its code score that can render a visual representation of past and future events in a real-time performance.

Keywords

Live coding, improvisation, code scores, notation, graphic scores.

1. INTRODUCTION

Live coding addresses a key problem of interaction in improvisation with computers, where the programming language is considered to have the potential for being an elegant instrument for musical expression [1]. As a method for computer-based performance, the typical setting is widely documented: it involves performers on stage programming computers in real-time where their actions are projected onto the wall for the audience to follow [4][6]. Live coding can be seen as a performance form where the instrument, the composition, and the performance are defined in front of audience members; an ad-lib public writing where the written text serves both as communication with the audience and as specific instructions that delegate actions to a musical system in constant state of revision.

The systems used in live coding have typically been designed for high-level musical expression through code, i.e., a few keystrokes result in expressive sound synthesis or musical patterns that can be changed in real-time. In this respect, live coders compose their systems as much as they code them. This is typically done prior to a performance, where musicians build their patches, libraries, or languages in concordance with their musical goals. Thus, behind every live coding system lies a well grounded set of compositional decisions, although they vary at the levels of abstraction. During performance the projected code can be seen as a musical score written in an improvisational context that emphasizes the dialogue between musician, audience, and the system [10].

2. THE THRENOSCOPE

This paper presents the Threnoscope, originally conceived of as a live coding piece for microtonal performance. It is a system built in SuperCollider [7] and makes use its powerful audio server to run complex synth instances that can be controlled through the interface modalities discussed in section 4. The design goal was to create a helpful graphical representation of the sonic texture in microtonal drone music. The system notates selected features of the drones, such as the spatial location, pitch, amplitude, filtering, and other parameters in a two dimensional score. Instead of the traditional linear score, the Threnoscope is circular, where drones (or the notes) circumnavigate a multichannel pitch space. The piece is composed for multichannel surround where the idea is to visualize the location of sound in space.

Figure 1 shows the score interface of the Threnoscope, with the harmonic series represented by the circles and the speakers by the lines crossing the score. In this instance the system is set up for eight channels. The colored annular wedges represent "drones", or long-duration note events. A drone intersecting a speaker line will sound out of that speaker. Thus, if the system is set for eight channels, we might say that we have eight static playheads, each of flexible rates, since the drones can travel at different speeds. This form of descriptive score represents the static and circular nature of the music, reducing the importance of linearity and temporality; the drones are not ephemeral musical events, but rather constants that can be altered in various ways, for example by silencing, pitch shifting, filtering, and moving them around in space.

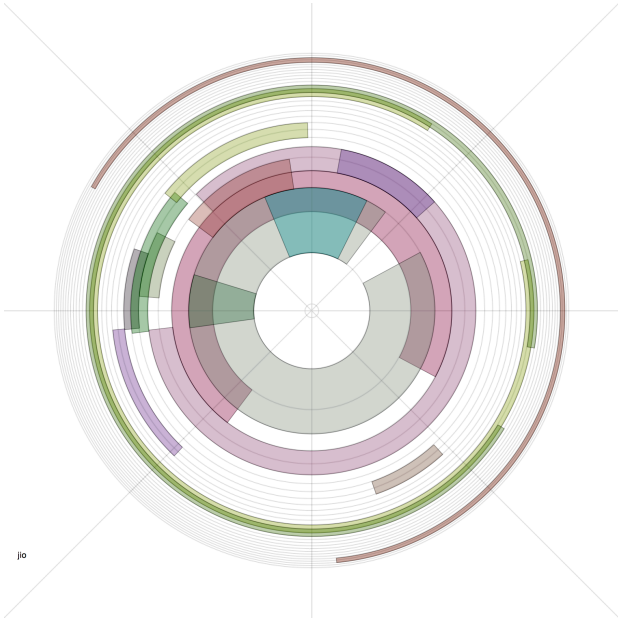

Figure 1. The Threnoscope's representational score in harmonics-mode. The crossing lines represent the speakers.

The center of the score is 0 Hz and the first circle is the tonic of the system, by default an A of 55 Hz. The second circle is the second harmonic, or 110 Hz, the third is 165 Hz, the fourth is 220 Hz, etc. The performer can create drones on any of these harmonics or anywhere in-between, using the following creation arguments: frequency, harmonic, ratio, degree, or combinations thereof. A drone could therefore be created like this:

```
// a saw wave on the first circle, with a cutoff on the third harmonic
~drones.createDrone(\saw, tonic: 1, harmonics: 3);
```

```
// a saw wave as a just intonation fifth above the tonic
~drones.createDrone(\saw, tonic: 3/2 );

//  triangle wave based on the second harmonic, on the fifth degree of the selected scale (depending on tuning, the fifth of the 2nd harmonic might, or might not, be equal to the third harmonic)
~drones.createDrone(\tri, tonic: 2, degree: 5);
```

A saw wave created on the tonic (55 Hz) with high cutoff frequency would naturally contain energy on all of the harmonics. A saw wave created on the second harmonic would contain energy on every other harmonic of the tonic, a saw wave created on the third harmonic would contain energy on every third harmonic, and so on. Drones can easily be transposed in pitch, supporting scale creation based on harmonic intervals. The system has two graphical pitch-notation modes: harmonics or scales. By default the circles represent harmonics, but the user can switch to pitch intervals of a selected scale. When displaying a scale, the octaves are indicated by a special color, and the degrees of the scales are drawn in another color between the octaves (unless it is a non-octave repeating scale). The Threnoscope supports the Scala file format (http://www.huygens-fokker.org/scala/scl_format.html), and custom made scales and tunings can be written in this format. Figure 2 shows the score in the scale display mode.

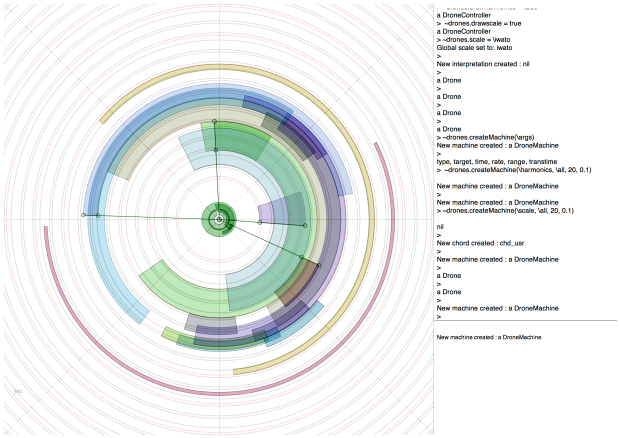

Figure 2. The Threnoscope in scale-mode. The live coding terminal is on the right. In the center a drone machine is working on the drones according to defined rules.

The drones themselves can be of various waveforms (e.g., a saw, triangle, square, noise, custom made waveform, or a sample). In addition to defining pitch, the initial arguments when a drone is created are the waveform type, harmonics, location, size, speed, amplitude, etc. The drones are color-coded according to the waveform type and have transparency according to their amplitude. Their width is the frequency bandwidth of the resonant lowpass filter. The drones represent complex synth instances, where any of the synthesis parameters can be controlled from the live coding interface, the visual interface, hardware, through the network, or from a score. Since all these parameters can be easily defined, the drone can range from a static drone that spreads over all the speakers, to a small fast-moving drone that triggers quick sounds when crossing speaker lines. Fast sequencing patterns can therefore be created using the system, for example drum beats or melodies, but the general system design discourages such compositional approach.

Various types of drone classes exist. *DroneChords* are a combination of two or more drones at specified intervals. *DroneSatellites* is a form of multi-drone creation that results in a cluster of short drones appearing randomly or within a scale at a certain range, often moving fast, and by default at a random speed. There are also *DroneGroups*, where information like pitch, amplitude, and location is defined and stored in a preset of relative features, so a group can be created at any pitch. The general state of the system can be stored and recalled at any time, superimposed upon other states, and set to appear or disappear over a longer time period.

```
// a minor 7th chord created on the fifth of the second octave
~drones.createChord(\saw, \minor7th, octave: 2, degree: 5 );

// a group of 40 satellites in in the iwato scale over four harmonics
~drones.createSatellites(\saw, \iwato, tonic:4, range:4, num:40 );
```

Finally, the system allows for the creation of a code score, where code snippets can be evaluated at specific points in time. The code score will be discussed below, as it presents unique opportunities for a live performance supported by either a linear or a generative score.

The performer can delegate control to the system itself through creating machines that operate on the drones. The drone machines work on diverse properties of the drones, such as pitch, amplitude, harmonics, etc. As seen in Figure 2, the machines appear in the middle of the circular score, and can be live coded during performance, allowing the user to specify their behavior in real-time. The machines function as co-players, or agents, that affect changes to the music, yielding more variety and excitement than the live coder might be able to create on their own with their single-task focus.

3. THE CODE SCORE

The system's initial design considerations focused on direct real-time performance through the method of live coding, where any parameter of a complex synth definition can be programmed. This includes creating temporal tasks that would change parameters at regular intervals in time independently of performer control. However, it slowly became clear that the system 'suggested' other performative modes, such as automated machines, hardware interfaces, GUI control, networked communication, and a temporal score. The linear temporal code score is designed for impromptu playback of pre-composed generative temporal patterns that run along with the performer's live coding, but it can also be used for offline or non-performative contexts. Such a setup mixes the improvisational nature of traditional live coding with a score-based approach to musical performance. It also poses questions as to what extent we compose our instruments, and to what degree musical theory is embedded in our new interfaces for musical expression. Can musical instruments be designed with inherent timelines? When does the system stop being an instrument and become a playback system? These concerns are not new, as arpeggios or chord structures in traditional synths are examples of such harmonic and temporal design. Early work with Max at IRCAM or controller design at STEIM are fine examples of this line of thinking in the areas of algorithmic composition and instrument design.

The code score in the Threnoscope is a two dimensional array listing the scheduled time and the associated code to be executed. The score is written in a simple, linear and human readable format that can be written in any text editor. It can be composed by hand, recorded from a live coding performance, or both (where the performer improvises on top of an already running score). Such textual format is not ideal for gaining an overview of what the piece is doing in time as array-based chronographics do not use the spatial dimension as an ordering principle [3]. There might be large quantities of code appearing within the same second, but then nothing might happen for a while. In order to gain a visual overview of the code score, it can be represented as a graphical timeline. This helps the composer to get an overview of the piece, but it also makes working with it easier as the score is interactive and code can be moved around using the graphical interface.

Figure 3 shows a visualisation of the code score as designed in the Threnoscope interface. The timeline runs from the top and down, like common tracker interfaces [9]. A general code-track exists where any system wide code can be inserted: both Threnoscope-specific code

and general SuperCollider code. The drones appear on the vertical tracks on the timeline. They have a beginning and an end, with code affecting the drones represented as square blocks on the drone track. The score can be manipulated in real-time, as we are accustomed to with MIDI sequencers or digital audio workstations; the drone can be dragged up and down, and the events within it can also be moved in time. By clicking on a drone, all the code related to that particular drone appears in a text field on top of the main circular score left to the code score. The user can change time values, rewrite instructions or create new events that will be automatically updated on the graphical representation of the drone's score.

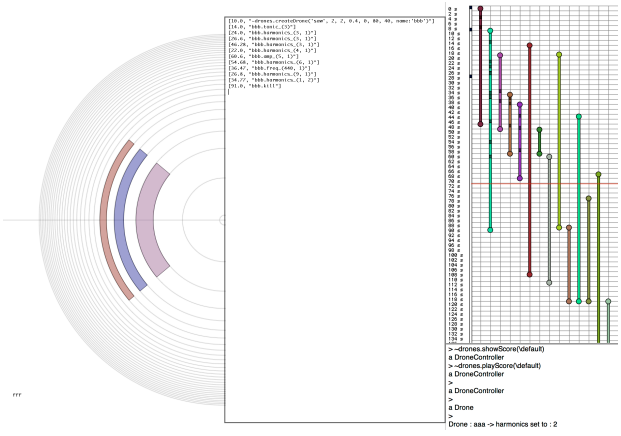

Figure 3. The visual representation of the code score. The drones are drawn on vertical tracks on a timeline. By clicking on a track, a text view shows the relevant code.

Most timelines in music software run horizontally from left to right. In the Threnoscope it was decided for various reasons to make the score vertical. Firstly, the screen space available, when the circular score has taken up the main space on the left, is rectangular. Secondly, when a user clicks on the visual representation of the drone, its score pops up in textual form where the order of events is the same in the textual drone score and the visual representation of the drone track itself.

Since any SuperCollider code can run within the system and its score, it is impossible to represent the code's output and its meaning as a graphical score: at this level of interpreted programming languages, the code itself is most often its best representation and its output (what it presents), such as sound or graphics, demonstrates best what it does. It should be noted here that we are not concerned with tracing the code and its internal functionality, but in direct representation of the code as a score and in its sonified output. There are impressive code tracing systems, e.g., for JavaScript (such as TraceGL and Google Trace), but those trace the internal logic and functionality with some graphical notation, and not representations or outcomes of the code.

4. PERFORMANCE INTERFACES

The Threnoscope is a work intended for live improvisation. The method of interfacing with the system is multi-modal as it aims to provide simple and natural interfaces for different tasks. For example, if the intention is to create a number of new drones every ten seconds over a few minutes, the best interface is probably code. This can be quickly written in a few lines of text resulting in an automated process, whereas a GUI interface would make the process complex and inflexible. If, however, the objective is to create a looping non-linear temporal trajectory of a drone's filter cutoff frequency, it might be better to draw this with the mouse, or some other hardware, as it would be too time consuming to write the algorithm for such a gesture. Similar examples can be provided for gestural interfaces and embodied musical performance. Below the five different control-modes of the Threnoscope are described.

4.1 The Terminal

On the right of the Threnoscope interface we find the live coding terminal where any SuperCollider code can be written. Below the terminal is a post window, displaying the state of the system and error messages.

This author has found that textual control over sound is often the ideal interface as graphical user interfaces or hardware are always at a higher abstraction level, limited to certain ranges and resolutions. Furthermore, graphical user interfaces "present" or even "suggest" possible actions through their visibility and prominency. Of course, some degree of algorithmic design and control can be achieved through graphical user interfaces, where the interface practically becomes a visual programming language, but at a certain level this flips over to the user interface becoming a much more complex and time consuming interface than the pure textual input of the programming language.

One limitation of the textual interface, as opposed to the graphical interface that "presents" its options, is remembering the right command names, the order of arguments, and what methods actually exist. The Threnoscope has methods that remind you of all these, printed in the post window (provided that the user remembers the names of those). For the well trained live coder, the lack of a GUI means freedom from compositional imperatives.

4.2 The Graphical User Interface

The representational score in the Threnoscope can be considered to be one large graphical user interface controller, although it serves mostly as a representation of the state of the music. However, the user can click on a drone to select it and alter its state in various ways, such as using the arrow keys to move it, number keys to jump to degrees in the scale, or use the delete key to remove the drone. As mentioned above, the user can also draw more complex patterns to be used as part of automating algorithms. This is shown in Figure 4. For example the following two commands:

```
~drones.setParameter(\freq, 300, 500);
~drones.recParameter(\harmonics, 2, 12);
```

create a line with crossing the interface with a number next to the box-shaped handle, enabling the performer to drag the handle to set the desired value of a selected parameter within the specified range. In this case, it is a frequency parameter in the range of 300 to 500 Hz. Since we don't always know what we want to do, this interface enables live coders to be less "cerebral" and get to a satisfying result through a more circular feedback process of constant action and evaluation. In the case of the "recParameter" method, the performer can drag the handle over a longer period of time, and on release, the recorded gestures are played back in a loop. These two examples illustrate cases where live coding is not an ideal interface due to either not knowing what you want, or the algorithm being to complex or time consuming to code in a live context.

4.3 Tangible User Interfaces

The system supports a range of tangible user interface methods, from basic mouse and keyboard to bespoke hardware communicating via OSC. MIDI controllers can be used although a majority of them are not particularly suitable for microtonal music.

4.4 Networked Music

Networked music is an interesting research area related to live coding. Projects vary in their design solutions, but the way the Thenoscope supports networked collaboration is through one computer being the server computer where other computers can send messages over the OSC protocol to the main server. This practically requires performers to be present in the same space although working on separate computers. However, future plans involve using the

OSCthulhu system as a server [8]. This is a system with a global server that receives commands from distributed clients and pushes the global state back.

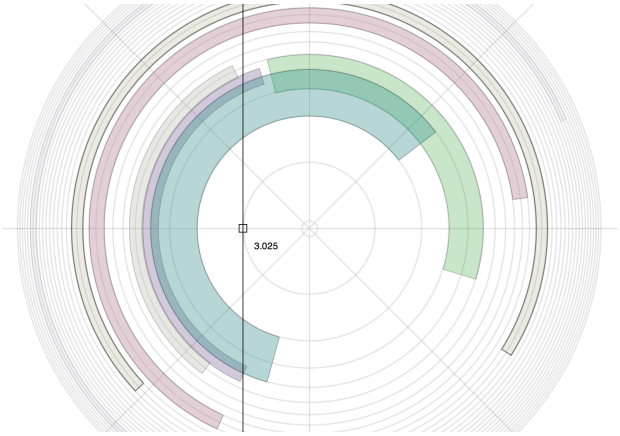

Figure 4. Drawing parameters with a handle on the graphical interface of the representational score.

4.5 The Code Score

The code score was initially implemented to enable small designed temporal patterns to be started at any point in a performance. However, it immediately became clear that the code score also served well as a notation system for offline composition, not necessarily deterministic as the code can be generative. The score can range from being a single event to hours of activity – it can be started and stopped at any point in a performance, and the performer can improvise on top of it. As an example, a performer in the middle of a performance might choose to run a three-second score that builds up a certain tonal structure. Another example would be a performer who might choose to run a long and slow score, improvising on top of it. The code below shows the code required to start a score.

```
~drones.playScore(\myScore, 1) // name of score and time scale
~drones.showScore(\def, 1) // visual display of the score
```

The first method simply plays the score. This is very flexible, as multiple scores can play simultaneously, or the same score started at different points in time. Scores can be stopped at any point. However, the "showScore" method creates the graphical representation of the score described above and shown in Figure 3.

For this author, the code score has been an interesting feature of this instrument. The code score could be characterized as an automation of the live performance, turning the instrument into a playback system, but we should note that most musical instruments contain parameters that happen outside the performer's control (such as timbre and amplitude envelopes, arpeggios, feedback, sympathetic strings, etc.). Here the digital instrument is simply given the feature of a designed process in the form of an extended score. This relates to what, in another paper, was called the parameter axes of *autonomy* and *music theory* in the epistemic dimension space used to analyze digital musical instruments [5].

5. THRENOSCOPE IMPROVISATION

The default mode of live coding performance is improvisation. Live coding is quite unique in that it is almost unimaginable that a performer would follow a score or a memorized performance. Since live coders often begin from the "blank slate," i.e., starting the performance with an empty text document, they need to know their system well. Using systems like SuperCollider or Extempore gives users much freedom in expression, but at the cost of speed and simplicity. Live coders therefore typically create their own systems on top of other languages. The Threnoscope is such a system, although it began as a constraint musical piece for live improvisation rather than a general live coding system. Of course, writing any live coding system involves defining possible sound objects and methods, their temporal and spatial structure, but when writing a musical piece the design decisions are stronger, options are closed, and a bespoke aesthetic is developed and emphasized.

The compositional limitations of the Threnoscope are primarily introduced through the graphical score and interface. The system encourages certain musical decisions and makes other more difficult. Through a constant visual feedback and limited syntax of the micro-language, the performer is habituating a space of musical possibilities, in a manner discussed by musical ecologists [2], where the surroundings and physical setup of equipment is seen as the design of performance affordances. Due to the spatio-visual nature of the Threnoscope, improvising with it is in many ways similar to the situation of an instrumentalist who has shaped or composed their instrument and surroundings with external technology and extended technique, and explored the musical potential of the particular setup.

Improvisation through code has proven to be very powerful. However, there are musical events that can better be defined through notation, drawing, and bodily gestures using hardware. This paper has introduced the ways in which the live coder can interact with the Threnoscope, and also explored the coexistence of concurrent score following and improvisation in a live performance.

6. CONCLUSION

The Threnoscope is a system that visualizes complex microtonal music, through graphical representation of independent drones. It is a descriptive score system that helps the performer to better understand the origins and behavior of the current musical activity and gain faster and more intuitive control over all the musical parameters.

This paper has presented a system still in development, and one that has changed considerably from being a musical piece for live coding performance to becoming a general musical instrument. Further possibilities have been suggested, such as the system serving as a music player for other composers' music. This author still considers the work a musical piece, but acknowledges the fact that categories such as instrument, composition, performer, and composer typically blur in digital musical systems.

7. REFERENCES

[1] A. Blackwell and N. Collins. The Programming Language as a Musical Instrument. In *Proceedings of PPIG05*, 2005.

[2] J. Bowers. Improvising Machines Ethnographically Informed Design For Improvised Electro-Acoustic Music. Stockholm: Kungl Tekniska Hogskolan. 2002. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.219.1699&rep=rep1&type=pdf [accessed April, 2014]

[3] S. Boyd Davis. History on the Line: time as dimension. In *Design Issues* 28, 4, 2012. 4-17.

[4] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. In *Organised Sound*, 8, 3, 2003. 321-330.

[5] T. Magnusson. An Epistemic Dimension Space for Musical Devices in *Proceedings of NIME*. 2010. 43-46.

[6] T. Magnusson. Herding Cats: Observing Live Coding in the Wild. In *Computer Music Journal.* 38, 1, 2014. 8-16.

[7] J. McCartney. Rethinking the Computer Music Language: SuperCollider. In *Computer Music Journal*. 26, 4, 2002. 61-68.

[8] C. McKinney and C. McKinney. OSCthulhu: Applying Video Game State-Based Synchronization to Network Computer Music. In *Proceedings of ICMC*, 2012.

[9] C. Nash and A. Blackwell. Tracking Virtuosity and Flow in Computer Music. In *Proceedings of ICMC*. 2011.

[10] J. Rohrhuber, A. de Campo, R. Wieser, J-K. van Kampen, E. Ho, and H. Hölzl. Purloined Letters and Distributed Persons. In *Music in the Global Village Conference*. 2007.