TIME-VARIANT DELAY EFFECTS BASED ON RECURRENCE PLOTS

Taemin Cho [†], *Jon Forsyth* [†], *Laewoo Kang* [‡] *and Juan P. Bello* [†]

[†] Music and Audio Research Lab (MARL)
Music Technology, New York University
New York, USA
{tmc323,jpf211,jpbello}@nyu.edu

[‡] Department of Information Science
Cornell University
Ithaca, NY 14850 USA
lk423@cornell.edu

ABSTRACT

Recurrence plots (RPs) are two-dimensional binary matrices used to represent patterns of recurrence in time series data, and are typically used to analyze the behavior of non-linear dynamical systems. In this paper, we propose a method for the generation of time-variant delay effects in which the recurrences in an RP are used to restructure an audio buffer. We describe offline and realtime systems based on this method, and a realtime implementation for the Max/MSP environment in which the user creates an RP graphically. In addition, we discuss the use of gestural data to generate an RP, suggesting a potential extension to the system. The graphical and gestural interfaces can provide an intuitive and convenient way to control a time varying delay.

1. INTRODUCTION

Recurrence plots (RP) are binary matrices representing patterns of repetition in sequential data. They are used for analyzing and visualizing the behavior of non-linear dynamical systems, and have been applied in fields as diverse as physics, genomics, chemistry, and economics [1, 2]. More relevant, RPs have also been applied to the analysis of music, e.g. for understanding rhythmic structure [3], cover-song identification [4] and measuring structural similarity [5].

In this paper, we propose a system that inverts this process: instead of using RPs to analyze an audio sequence, our system restructures music audio using the patterns of recurrence represented by a given RP. The approach works by combining blocks of the input signal such that the repetitions characterized by the RP are enforced on the output signal. The system thus acts as a time-variant delay line, able to produce complex patterns of repetition. Furthermore, we can use a graphical or gestural interface to modify the topology of the RP, hence providing a novel mechanism for system control that is more intuitive than existing hardware and software implementations of similar effects. Finally, note that our approach operates on an audio buffer, either offline or in real time, as a digital audio effect, and is thus unlike previous approaches that synthesize audio directly from the output of non-linear systems [6].

The remainder of this paper is structured as follows. Section 2 discusses the basics of delay lines and related commercial and non-commercial work. In section 3 we briefly define recurrence plots and propose a method for adapting them to the task of restructuring audio. Section 4 describes a Max/MSP implementation of our system, while section 5 discusses a preliminary gestural control mechanism. Finally, Section 6 discusses our conclusions and some directions for future work.

2. RELATED WORK

Delay lines are widely used audio effects [7]. In their simplest form, the output signal $y[n]$ consists solely of the input signal $x[n]$ delayed by an integer number of samples M, i.e.

$$y[n] = x[n - M] \tag{1}$$

In addition, the delay can feed the output signal $y[n]$, scaled by a gain factor g, back into the input:

$$y[n] = x[n] + g \cdot y[n - M] \tag{2}$$

where $0 \leq g \leq 1$ in order to ensure stability. While in equations 1 and 2, M is restricted to integer values, fractional delay techniques allow for arbitrary delay times. This model serves as the basis for most audio delays, both in hardware and software.

Using a fixed delay time (i.e., a constant value of M) produces a regular pattern of repetition, resulting in a time-invariant system. Alternatively, some delay line implementations allow M to vary over time, either manually or by a low-frequency oscillator, thus creating irregular patterns of repetition. Techniques such as time shuffling (brassage) and granular synthesis [7], in which an output buffer consists of random combinations of segments of an input buffer, can be regarded as time varying delay effects.

Such variations are common practice in commercial hardware implementations, e.g. as pedals or rack-mounted units. A number of artists, for example guitarists David Torn and Bill Frisell, use such devices in performance to achieve various glitching, stuttering, and similar effects in real time. Frequently, these artists produce these effects by changing the delay time (and other parameters) via direct and active manipulation of various controls on the delay device. In an effort to improve these interactions, novel user interfaces have been proposed. For example, the MATRIX interface [8] is a physical controller consisting of a 12x12 array of vertical rods, each of which able to move up and down independently. By varying the pressure applied to the rods and the orientation of the hands, the user generates a continuous signal that can be used to control effects and synthesis parameters. In one example, the control data is used to continuously change the delay time parameters of a multi-tap delay.

Other time-variant delays have been implemented in environments such as Pure Data, Max/MSP, and Supercollider. For example, Jonny Greenwood, guitarist for the rock band Radiohead, uses a Max/MSP patch that repeats segments of live audio in unpredictable patterns [9]. Another example is the BBCut2 library for SuperCollider [10] that allows users to splice and rearrange audio in real time or offline. BBCut2 makes use of "cut procedures," which define how an audio buffer is subdivided, including tools

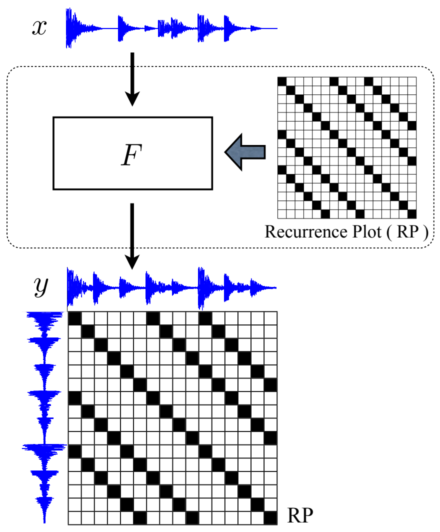

Figure 1: System overview : Input signal x is filtered by F yielding y. A given recurrence plot (RP) is a parameter set of F. As seen in the bottom half of the figure, the resulting sound y has a structural form that can be described by the given recurrence plot (RP).

to analyze the rhythmic structure of the audio. Users can create their own cut procedures programmatically, or use predefined procedures included with the library. While BBCut2 is undoubtedly a powerful tool capable of creating a wide variety of interesting effects, it is a library and not a stand-alone application with a fully developed user interface. Thus, use of BBCut2 entails a certain amount of programming knowledge, not necessarily available to most composers and performers. In addition, there are a number of freeware and commercially available plugins, such as iZotope's Stutter Edit plugin[1], that allow users to create various types of glitch and stutter edits.

3. APPROACH

While the systems and implementations discussed above can create compelling musical results in the right hands, none offer an intuitive interface that allows a user to quickly create and experiment with different time varying delay patterns. In this paper we argue that recurrence plots (RP) can be used to exercise (and visualize) control of audio delays, and propose a method for doing so, which can be seen in Figure 1. An input signal x is rearranged and remixed by a function F in order to produce the output signal y. F is fully defined by a given RP, such that the recurrences in the plot should also appear in y. In the following sections we will discuss how RPs are obtained, and describe a method for using the plots to specify F, first offline, and then in real time.

3.1. Recurrence Plots

Let us assume the time series y to describe the output of a dynamical system, such that if the state of the system at time i recurs at time j, then y_i is the same as y_j within a margin of error, ϵ. In this case, cell (i, j) of the RP matrix is set to 1 (black in the graphical representation); otherwise, the cell is set to 0 (white). More formally, the RP matrix R can be defined as follows:

[1] http://www.izotope.com/products/audio/stutteredit

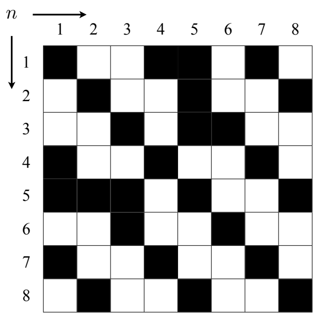

Figure 2: An example of recurrence plot with $N = 8$.

$$R_{i,j} = \begin{cases} 1, & ||y_i - y_j|| < \epsilon \\ 0, & ||y_i - y_j|| \geq \epsilon \end{cases}, \quad i, j = 1, \ldots, N \tag{3}$$

where N represents the length of the time series, and thus the number of rows and columns in the plot, and $||\cdot||$ denotes a norm (e.g., Euclidean norm). The resulting plot is a symmetric, binary matrix such as the one shown in Figure 2.

3.2. Audio Restructuring using a Recurrence Plot

Next, we must develop a method for restructuring audio using an existing RP. To simplify the reconstruction problem, assume that the input signal $x[n]$ is finite and divided into N segments, $n \in [1, N]$. The output sound $y[n]$ can be obtained by a linear combination of the input segments as:

$$y[n] = \sum_{i=1}^{N} x[i] \cdot c_{i,n} \tag{4}$$

where $c_{i,n}$ is a reconstruction coefficient satisfying the condition that $y[n]$ should have the temporal structure described in the RP.

A simple approach to find an appropriate coefficient set c is by direct application of the information about the temporal recurrences described in the RP. That is, if a state at time n_1 recurs at time n_2, the states at time n_1 and n_2 are assumed to be the same, i.e. $y[n_1] = y[n_2]$. As an example, consider the simple RP shown in Figure 2. In the first column of this RP, the rows at time $n = \{1, 4, 5, 7\}$ are activated, indicating that the output sounds $y[1]$, $y[4]$, $y[5]$, and $y[7]$ should be identical. In the same manner, from column 2, we can infer that $y[2] = y[5] = y[8]$, from column 3 that $y[3] = y[5] = y[6]$, etc. However, the fact that $y[5]$ appears in each of the first three columns implies that all the segments are equal (i.e. that $y[1] = y[2] = \cdots = y[N]$). This result is in conflict with the given RP: if all the segments were identical, all the cells of the RP matrix would be black.

The problem with this approach is the assumption that recurring segments are identical. In fact, the margin of error ϵ from equation (3) implies that the RP represents only approximate relationships between segments. Therefore, in the previous example, while the first column indicates that $y[1] \approx y[5]$, and the second indicates that $y[2] \approx y[5]$, we cannot assume that $y[1] \approx y[2]$.

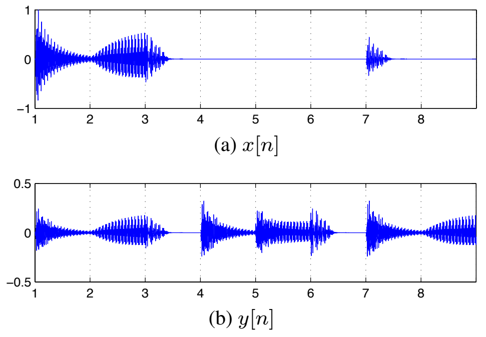

Figure 3: Reconstruction using the RP in Figure 2 : (a) the input signal $x[n]$ (b) the resulting sound $y[n]$.

Because information is lost when computing an RP (i.e., the exact distance between $y[i]$ and $y[j]$), precise reconstruction of y is impossible. However, we can approximate $y[n]$ by considering only column-wise recurrences, and ignoring any relations that appear within a given column. By doing this, each row becomes independent, indicating the recurrences of an individual component, $x[n]$. With this in mind, the coefficients $c_{i,n}$ can be derived from the nth column vector of RP as follows:

$$c_{i,n} = \frac{R_{i,n}}{\sum_{i=1}^{N} R_{i,n}} \tag{5}$$

where the denominator is a normalization factor accounting for the number of activated components in the column. Therefore, we can derive the following reconstruction rule from equations (4) and (5):

$$y[n] = \frac{\sum_{i=1}^{N} x[i] \cdot R_{i,n}}{\sum_{i=1}^{N} R_{i,n}}, \tag{6}$$

Figure 3 shows the input sound $x[n]$ and the resulting sound $y[n]$ using the RP in Figure 2. Accordingly, $y[1] \approx y[4] \approx y[7]$, with each output segment a combination of the input components $x[1, 4, 7]$. Likewise, $y[1] \approx y[5]$, with each output segment a combination of $x[1, 5]$.

3.3. Real-time Approach

There are two main restrictions to adapting the reconstruction approach discussed in Section 3.2 to real-time processing. First, unlike the non-real-time situation, the length of the incoming signal is unknown, and thus assumed to be infinite. Second, future audio segments are not available. The following solutions and compromises are necessary to cope with these restrictions.

First, for a given $N \times N$ plot, we use an N-length circular buffer $B[\hat{n}]$, $\hat{n} \in [1, N]$. The buffer stores the last N L-long segments of the incoming signal x, such that the oldest segment is always the one to be overwritten. For this to happen, we define the buffer index $\hat{n}$ as follows:

$$\hat{n} = \Big((n-1) \bmod N\Big) + 1, \quad n \in \mathbb{Z}^{+} \tag{7}$$

such that $\hat{n}$ is reset to 1 after each group of N signal blocks has been stored. The index $\hat{n}$ is also used to index the rows and columns of the RP, allowing us to rewrite equation (6) as:

$$y[n] = \frac{\sum_{i=1}^{N} B[i] \cdot R_{i,\hat{n}}}{\sum_{i=1}^{N} R_{i,\hat{n}}}, \tag{8}$$

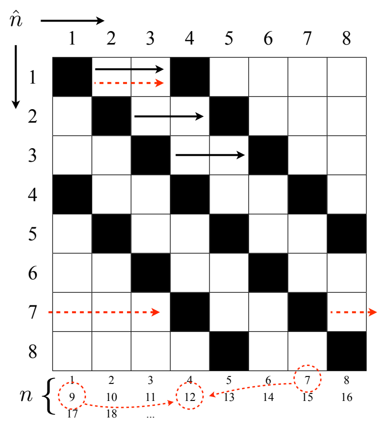

Figure 4: A given RP ($N = 8$) for the real-time process: $\hat{n}$ indicates the corresponding row and column indexes at time n (bottom of the figure).

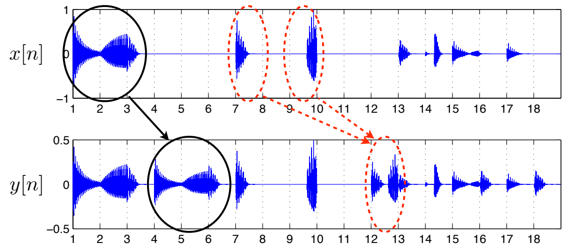

Figure 5: Real-time reconstruction example: the resulting sound $y[n]$ from the input $x[n]$. $x[1, 2, 3]$ recur at $y[4, 5, 6]$ (solid circles and arrow), and both $x[7]$ and $x[9]$ recur at $y[12]$ (dotted circles and arrows). The magnitude differences are due to normalization.

Second, the causality of the system means that the current output is necessarily a function of previous inputs. However, the use of the circular buffer introduces boundary effects that are not immediately apparent. Take for example the RP in Figure 4: $x[1]$ recurs at $n = 4$, $x[2]$ recurs at $n = 5$, and $x[3]$ at $n = 6$ (indicated by solid arrows in the figure), thus seemingly defining a standard delay with delay time 3. However, due to the modulo operation, $x[7]$ recurs at $n = 12$, a delay of 5 instead of 3. In fact, the fourth column of the RP indicates that $y[12]$ is a linear combination of $x[7, 9, 12]$ (dotted arrows and circles in the figure).

Figure 5 shows an example of real-time reconstruction using the RP in Figure 4. As described above, $x[1, 2, 3]$ recur at $y[4, 5, 6]$ (indicated by the solid arrow and circles in the figure), and both $x[7]$ and $x[9]$ are mixed into $y[12]$ (indicated by the dotted arrows and circles in the figure).

In practice, due to the fact that the buffer B is initially empty, the direct implementation of equation (8) yields an unwanted fade-in effect at the beginning of $y[n]$, for the length of one cycle of buffering (i.e., for $n \in [1, N]$). We thus modify the normalization factor in equation (8) as follows:

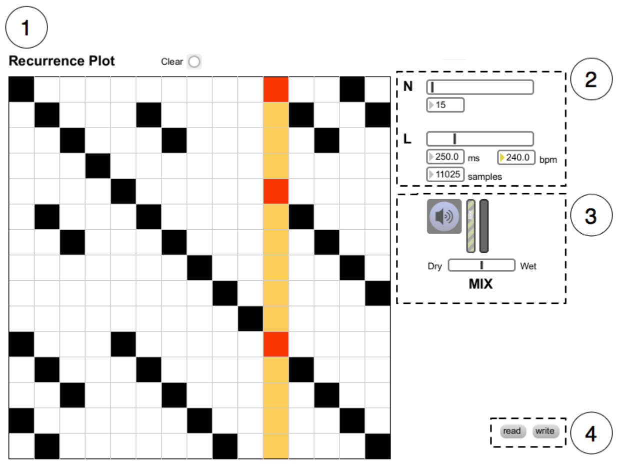

Figure 6: Max/MSP user interface.

$$y[n] = \frac{\sum_{i=1}^{N} B[i] \cdot R_{i,\hat{n}}}{K}$$

$$\text{where } K = \begin{cases} \sum_{i=1}^{n} R_{i,n} & n \le N \\ \sum_{i=1}^{N} R_{i,\hat{n}} & n > N \end{cases} \tag{9}$$

4. IMPLEMENTATION

4.1. Technical details

The RP-based delay approach described above was implemented using Max/MSP, a graphical programming language designed specifically for use in realtime systems. It also provides a number of useful user interface elements, making it an excellent prototyping environment. The external object, rpprocessor~, was developed in C using the Max 5 API, in order to compensate for the shortcomings of the native audio buffers in Max/MSP. This object maintains the audio buffer and an internal representation of the RP, and reconstructs the output signal according to equation (9). In addition, rpprocessor~ manages the creation, loading, saving and display of the RP data.

4.2. User interface

The user interface for the Max patch is shown in Figure 6. The user draws the desired recurrence pattern in the large grid area marked ①, by clicking/toggling cells between black and white. Once again, note that black squares indicate a recurrence. To maintain the standard topology of RPs, a non-editable main diagonal is added by default. Additionally, symmetry is enforced by automatically toggling equivalent cells the other side of the diagonal. The "clear" button above the RP deactivates all the cells in the block except those in the main diagonal.

The sliders marked ② allow the user to set N, the size of the RP, and L, the duration of each cell in the RP. Both of these values can be set using the sliders or the associated number boxes. Adjusting N changes the resolution of the RP grid, with higher values corresponding to increased resolution. The segment duration is displayed both in milliseconds and samples.

The value of L is also displayed in beats per minute (bpm) in the "bpm" number box. According to equation (7), the value of $\hat{n}$ varies between 1 and N, and represents a particular column of the RP. Because the duration of each cell in the RP is L, $\hat{n}$ advances at a rate determined by L. We can therefore assume $\hat{n}$ to represent a beat, and compute the value of L in bpm as follows: $bpm = 60/L$ (where L is measured in seconds).

The sliders in ③ are used to adjust the mix between the dry and wet signals and the overall gain. To enable the system, the user presses the large button in this region of the UI, thus turning on the analog-to-digital and digital-to-analog converters. Once the system has been enabled and is in play mode, the columns of the RP are highlighted in sequence, with the cells corresponding to the current $\hat{n}$ in a given column colored red and the unactivated cells colored yellow. The highlighting proceeds from left to right, wrapping back to the first column once $\hat{n}$ reaches N. As discussed above, $\hat{n}$ is incremented every L seconds. The highlighting thus makes the value of L explicit, allowing the user to more easily synchronize his or her playing with the system.

The user can save the current RP into a text file by using the "write" button in ④. Pressing this button brings up a file chooser window, allowing the user to specify the desired name and location of the file. Pressing the "read" button also brings up a file chooser, allowing the user to load an RP from an existing text file. Once loaded, the RP will be displayed in the editing region (① in figure 6).

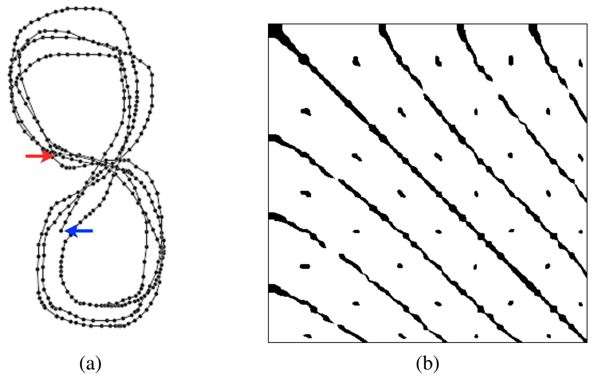

Figure 7: Constructing an RP from gestural data: (a) captured hand movements (the upper arrow and the lower arrow indicate the start point and the end point respectively) (b) RP generated from the gesture data shown in (a).

5. GENERATING AN RP FROM GESTURAL DATA

In the previous section, we have discussed how the user can draw a recurrence plot to control a time-variant delay line. However, because each RP is inherently associated with a time series, we have the ability to use any time series to control the audio effect. In this section, we provide a simple extension to our system in which we generate RPs from hand gestures.

To capture hand movements, we use an infrared LED and an Apple iSight web camera. The camera is fitted with a filter that blocks all visible light letting only infrared light pass. To capture

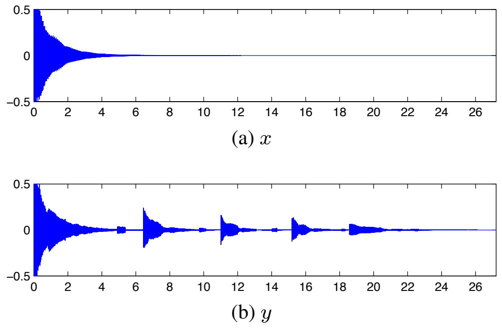

Figure 8: Real-time reconstruction example: (a) an input signal (b) a resulting sound using the RP in Figure 7b.

a gesture, the user holds the infrared LED in his or her hand, and moves it in view of the camera. The resulting X-Y positions of the LED, which we will define as $P(n) = (X_n, Y_n)$, are recorded at fixed time intervals.

Figure 7a shows a 400-point example time series of captured hand gestures, showing four repetitions of a figure-8 pattern. Positions are represented with black dots, with the starting point marked with a red arrow, and the end point marked with a blue one. The example trajectory illustrates an important issue with this approach. At the center of the figure-8 pattern the four upward and four downward trajectories pass through a small region of the 2-D space. According to equation (3), the proximity between these points makes them recurrences of each other. However, it can be argued that the upward and downward sub-trajectories are different enough that they should be regarded as different, and that the closeness between their constituent points is circumstantial. This implies measuring recurrences between sub-trajectories instead of between individual points, in order to avoid such spurious detections (and the resulting noisy plots). This can be done using a technique known as time-delay embedding [1, 2], where the n^{th} point of the embedded time-series is defined as:

$$\hat{P}(n) = \big(X_n, X_{n-1}, \ldots, X_{n-\omega+1}, \\ Y_n, Y_{n-1}, \ldots, Y_{n-\omega+1}\big) \quad (10)$$

where $\omega \in \mathbb{N}$ is known as the embedding dimension (note that we assume the embedding delay to be 1). The RP can be obtained as:

$$R_{i,j} = \begin{cases} 1, & ||\hat{P}(i) - \hat{P}(j)|| < \epsilon \\ 0, & ||\hat{P}(i) - \hat{P}(j)|| \geq \epsilon \end{cases} \quad (11)$$

where, as before, ϵ is the margin of error and $||\cdot||$ is the Euclidean norm.

Figure 7b shows the RP computed from the gesture data using $\epsilon = 50$ and $\omega = 3$. It has three diagonal lines in the upper and lower triangular parts of the plot, indicating that the main diagonal recurs three times. The shorter diagonals in the RP correspond to the crossing of the sub-trajectories discussed above. They can be filtered out entirely with a larger ω value, however at the cost of missing recurrences in the larger diagonals. Finding an optimal parameterization for embedding is an active area of research [2].

Figure 8 shows the result of applying the RP from Figure 7b to an input signal x, with $L = 46.5$ms (i.e., 2048 samples in 44.1 kHz sample rate). Sound examples, including stutter and time shuffling effects, as well as the sounds described above, are available at `http://marl.smusic.nyu.edu/rpprocess`.

6. CONCLUSIONS AND FUTURE WORK

We have presented a method for transforming an input signal based on the patterns of recurrence represented in an RP, and a realtime implementation of this method in which the user creates the RP using a graphical interface. The output signal from this system exhibits the recurring structures described by the RP. Unlike existing systems, however, the graphical interface in our system allows a user to easily experiment with different delay patterns. The use of gestural data to control the system suggests another intuitive means of controlling the delay parameters.

While the realtime system achieved the goal of producing complex delay effects, its time varying nature made it somewhat difficult to use in practice. In contrast to the case of a standard delay line, in which a user can easily synchronize his or her playing with the repeats, the time dependent behavior of our system made such synchronization difficult. The highlighting of the current column of the RP, while intended to ameliorate this problem, proved to be of limited use. Although further practice with the system would likely make synchronization easier, other remedies could include the use of an onset detector to trigger the start of an RP cycle.

We are also interested in more fully exploring the generation of the RP through gestural data. We feel that with these improvements, our system could prove to be a useful tool for performers and composers interested in producing complex delay effects.

7. REFERENCES

[1] J.P. Eckmann, S.O. Kamphorst, and D. Ruelle, “Recurrence plots of dynamical systems,” *EPL (Europhysics Letters)*, vol. 4, pp. 973, 1987.

[2] N. Marwan, N. Wessel, U. Meyerfeldt, A. Schirdewan, and J. Kurths, “Recurrence plot based measures of complexity and its application to heart rate variability data,” *Physics Reports*, vol. 438, no. 5 - 6, pp. 237 – 329, 2002.

[3] J. D. Reiss and M. B. Sandler, “Nonlinear time series analysis of musical signals,” in *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX-03)*, 2003.

[4] J. Serrá, X. Serra, and R. G. Andrzejak, “Cross recurrence quantification for cover song identification,” *New Journal of Physics*, vol. 11, art. 093017, September 2009.

[5] J. P. Bello, “Measuring structural similarity in music,” *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

[6] M. Witten, “The sounds of science: II. Listening to dynamical systems–Towards a musical exploration of complexity,” *Computers and Mathematics with Applications*, vol. 32, no. 1, pp. 145 – 173, 1996.

[7] U. Zölzer (Ed.), *DAFX: digital audio effects*, John Wiley & Sons Inc, 1st edition, 2002.

[8] D. Overholt, “Control of Signal Processing Algorithms using the MATRIX Interface,” in *Audio Engineering Society 114th Convention*, 2003.

[9] N. Collins, “Kid A, Amnesiac, and Hail to the Thief (review),” *Computer Music Journal*, vol. 4, no. 1, 2004.

[10] N. Collins, “BBCut2: Integrating beat tracking and on-the-fly event analysis,” *Journal of New Music Research*, vol. 35, no. 1, pp. 63–70, 2006.