SynthAssist: Querying an Audio Synthesizer by Vocal Imitation

Mark Cartwright
Northwestern University
EECS Department
mcartwright@u.northwestern.edu

Bryan Pardo
Northwestern University
EECS Department
pardo@northwestern.edu

ABSTRACT

Programming an audio synthesizer can be a difficult task for many. However, if a user has a general idea of the sound they are trying to program, they may be able to imitate it with their voice. This paper presents SynthAssist, a system for interactively searching the synthesis space of an audio synthesizer. In this work, we present how to use the system for querying a database of audio synthesizer patches (i.e. settings/parameters) by vocal imitation and user feedback. To account for the limitations of the human voice, it uses both absolute and relative time series representations of features and relevance feedback on both the feature weights and time series to refine the query. The method presented in this paper can be used to search through large databases of previously existing "factory presets" or program a synthesizer using the data-driven approach to automatic synthesizer programming.

Keywords

synthesizer interface, query-by-example, audio retrieval, voice controlled interfaces

1. INTRODUCTION

As software-based synthesizers have become more advanced, their interfaces have become more complex and therefore harder to use. For example, Apple Inc.'s ES2 synthesizer has 125 controls, mostly consisting of knobs, buttons, and sliders. If those controls were simply binary switches, the control space would consist of 2^{125} (i.e. 10^{38}) possible combinations. Controls with more settings (e.g. knobs and sliders) allow even more combinations. Fully exploring such a large space of options is difficult. Compounding this problem is the fact that controls often refer to parameters whose meanings are unknown to most (e.g. the 'LFO1Asym' parameter on Apple's ES2 synth).

For many musicians, the opacity of the controls, combined with the large number of possible combinations translates into an inability to actualize one's ideas. Even for experienced users, the tedium of these interfaces takes them out of their creative flow state, hampering productivity. Although simpler interfaces do exist (e.g. Apple Inc.'s GarageBand), their simplicity is limiting. They lack the flexibility of the complex interfaces, resulting in a small timbre palette constructed of a small number of factory presets, templates, and parameters with few creative options. Some manufacturers address this problem by having many, many presets (e.g. Native Instruments Kore Browser). However, searching through a vast number of presets can be a task as daunting as using a complex synthesizer.

In this work, we present SynthAssist, a system that interactively helps the user find their desired sound (synthesizer patch) in the space of sounds generated by an audio synthesizer. The goal is to make synthesizers more accessible, letting users focus on high-level goals (e.g. 'sound brassy') instead of low-level controls (e.g. 'What does the LFO1Asym knob do?'). The inspiration for SynthAssist was how one might interact with a professional music producer or sound designer: imitate the desired outcome vocally (e.g. "make a sound that goes like <sound effect made vocally>", have the producer design a few options based on the example, give them evaluative feedback (e.g "that doesn't sound as good as the previous example.")

With SynthAssist, a user can quickly and easily search through thousands of synthesizer sounds to find the desired option. The user first provides one or more *soft examples* (queries) that have some, but not all, of the characteristics of the desired sound. The soft examples can either be vocal imitations of the desired sound or existing recordings that are similar to the desired sound. Given these examples, SynthAssist guides the user in an interactive refinement process, where the system presents sounds for the user to rate. Based on these ratings, the system refines its estimate of the user's desired concept and learns which audio features are important to the user so that it can present sounds that more closely matches the desired concept. While this method could potentially be used for other audio Query-By-Example (QBE) applications (e.g. searching through a sample or sound effects database), it has been particularly designed for searching synthesizer and musical instrument sound databases.

2. RELATED WORK

2.1 Synthesizer Interfaces

Research related to the development of intuitive interfaces to audio synthesizers has been ongoing for several decades, with numerous approaches having been proposed [1, 3, 5, 9–11, 14, 19–22]. Researchers have sought to reduce the dimensionality of the synthesis parameter space by re-mapping controls to perceptual dimensions [20,21], high-level descriptive dimensions [5, 10, 11], exploratory maps [1, 14], other timbral spaces (e.g. the voice [6,19]) and more. Researchers have also developed methods to allow users to explore the synthesis space using interactive genetic algorithms that tune the synthesis parameters [3]. While good for exploration, the number of evaluations required to program a specific desired sound using genetic algorithms is far too

NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.

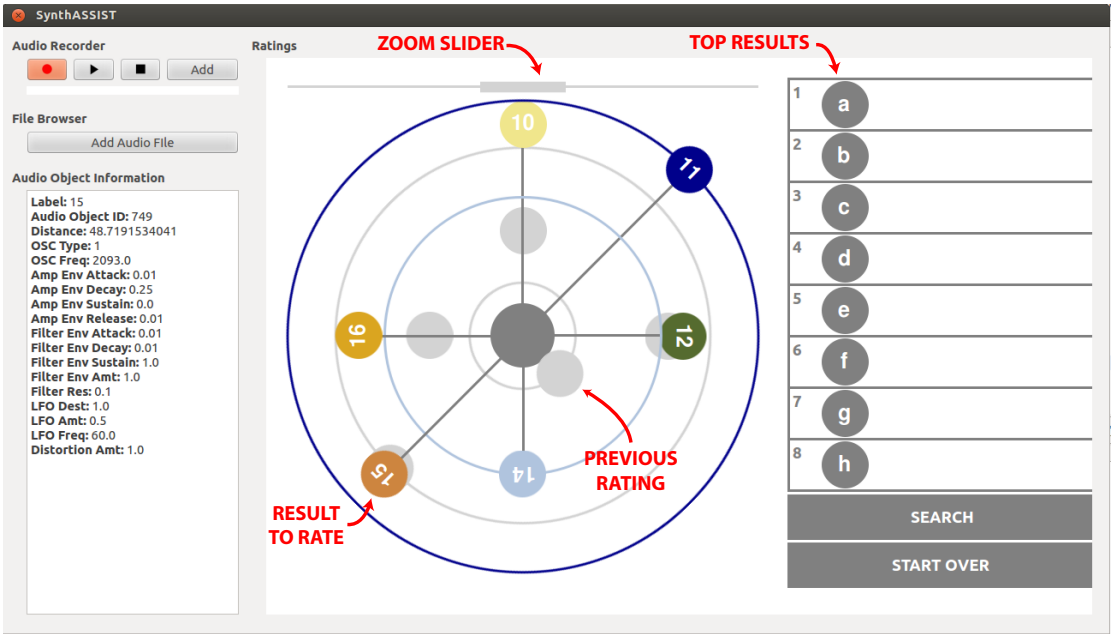

Figure 1: Screenshot of the SynthAssist interface.

great to be completed by a human. Another approach is to have the computer compute the fitness function in an optimization algorithm [7–9, 22]. However, in these "tone matching" approaches the user must provide an existing audio file of the *exact* desired sound, a requirement that may be difficult for the user to satisfy.

Recent work suggests that vocal imitation is a promising way of communicating audio concepts to software [12] and that comparing time series representations would be a good approach to doing so [4]. However, to date, only one very limited study has used vocal imitation for audio retrieval [2]. There has also been some research into using the voice for continuous real-time control of a synthesizer [6, 19]. However, in many scenarios, this is not feasible and/or desirable. In our system, the user's voice is needed when initially programming the synth but is not required when performing with the synth.

3. THE SYNTHASSIST SYSTEM

In the SynthAssist system, we want to support vocal imitation queries. Our approach is similar to the approach suggested in [4]. The motivation is that while the human voice has a limited timbral range [19], it is very expressive with respect to how it changes in time. For example, your voice may not be able to exactly imitate your favorite Moog bass sound, but you may be able to imitate how the sound changes over the course of a note (e.g. pitch, loudness, brightness, noisiness, etc.). When comparing sounds in SynthAssist, we focus on these changes over the course of a note.

The interaction for a session in SynthAssist is as follows:

1. To communicate their desired audio concept, the user gives one or more initial input examples by either recording a new sound (e.g. vocalize an example) or choosing a prerecorded sound.

2. The user rates how close each example is to the target sound.

3. Based on the ratings, SynthAssist estimates what the target sound must be like.

4. SynthAssist generates suggestions from the synthesizer that are similar to the estimated target sound.

5. The user rates how close the suggestions are to the desired target sound.

6. If a suggestion is good enough, return the synthesizer parameters. Else, repeat Steps 3 through 6.

Figure 1 shows a screenshot of SynthAssist. Each suggestion is represented by one of the colored circles. When a user clicks on a suggestion, the sound plays. Users rate how similar suggestions are to their target by moving them closer to or farther from the center, "hub", circle. If the suggestion is irrelevant, the user can inform SynthAssist and remove it from the screen by double clicking on it. The user plays a synth patch by either single clicking or moving (rating) a colored circle. Dragging a suggestion to the center of the circle indicates indicates this is the desired sound and terminates the interaction. The overall system architecture is shown in Figure 2.

3.1 Query and Search Key Representation

A *query* is an example provided to the system to guide the search for the desired *item* (a synthesizer sound) from a database. Each item in the database consists of a *sample* (an audio recording), a *patch* (the set of synthesizer parameters required to create the sample), and a *search key* (an abstract representation of the sample's audio features).

All audio (queries and samples in the database) is summed to mono and RMS-normalized. Features for the search keys are extracted using a frame size of 1024 and a hop size of 512 at a sample rate of 32 kHz. Focusing on these changes through time for both queries and search keys, we extract the time series of a small number of high-level features from the audio: *pitch*, *loudness*, *inharmonicity*, *clarity*, *spectral centroid*, *spectral spread*, and *spectral kurtosis*. Definitions for all of the features but the *clarity* measure can be found in [17]. Similar to autocorrelation height, the clarity measure is a measure of how "coherent a note sound is" [13]. We then augment this representation by also standardizing each of these features with themselves to capture the relative changes through time (e.g. $\mathbf{x}_{std} = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$ where $\mathbf{x}$ is

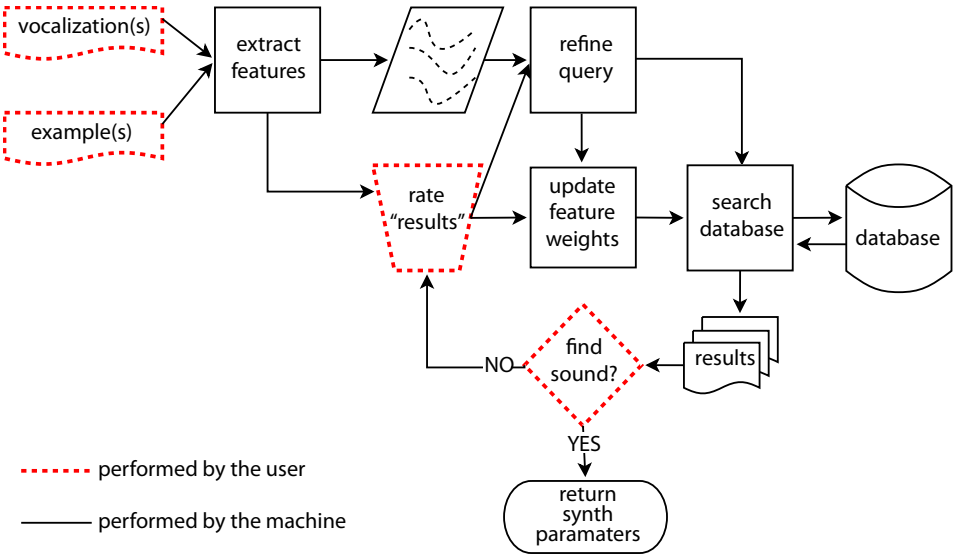

Figure 2: System flow of SynthAssist.

the time series and $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ are the mean and variance of $\mathbf{x}$).

Therefore we represent each query and search key as 14 time series, one per feature: 7 "absolute features" and 7 "relative features". While many QBE systems characterize these time series as either distributions (e.g. modeling them with a Gaussian Mixture Model) or extract statistics and features (e.g. mean, variance, slope, modulation), we retain the time series representation to capture the temporal evolution of each sound. We represent each query or search key as a matrix where each feature time series is a row of length N, where N is the number of feature frames, e.g. $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{14}]^T \in \mathbb{R}^{14 \times N}$.

3.2 Rank Calculation

To search the database we calculate the distance from the query to each search key. However, the query and the keys may not be of the same length. Therefore, we use a distance measure based on dynamic time warping (DTW) [15] and treat each feature time series independently. We calculate distance using the following equation:

$$D_{\mathbf{X},\mathbf{Y}} = \sum_{i=1}^{14} w_i DTW(\mathbf{x}_i, \mathbf{y}_i) \quad (1)$$

where i is the index of the 14 features, $\mathbf{X}$ and $\mathbf{Y}$ are the query and search key matrices, $\mathbf{x}_i$ and $\mathbf{y}_i$ are the time series of the i^{th} feature for the query and key, w_i is the weighting coefficient of the i^{th} feature, and $DTW(\mathbf{x}_i, \mathbf{y}_i)$ is the dynamic time warping distance between $\mathbf{x}_i$ and $\mathbf{y}_i$.

After calculating distance for each key, the system returns two sets of results: *top results* and *rating results*. The first set, *top results*, consists of the 8 nearest neighbors in increasing order of the distance function in Equation 1. Initially, the weighting coefficients, $\mathbf{w}$, are all equal, but after the first search round, they are refined as specified in Section 3.3. Therefore, the top results may change each round (where a round is steps 3 to 6 in Section 3). This set of results appears on the right hand side of the interface as the array of grey circles as shown in Figure 1.

Since the top results may consist of many similar items, rating all of them may not give us much useful information. Therefore, the second set of results, *rating results*, consists of the examples we want the user to rate. This includes the nearest key that has not been rated yet and also the keys which are nearest in each of the 14 feature dimensions (i.e. the keys that were closest while just considering each feature distance, $DTW(\mathbf{x}_i, \mathbf{y}_i)$, independently). This is a computationally efficient way of increasing the diversity of the results while also maintaining relevance. However, to avoid crowding the screen with too many results, we randomly select 7 of these 14 keys for a total of 8 keys in *rating results* each round. The *rating results* appear as the small colored circles radiating out of the "target hub" in Figure 1.

3.3 User Feedback and Query Refinement

Once presented with the two sets of results (see Section 3.2), the user can listen to both sets and can potentially select their desired synth example, returning the parameters of the synth. If their desired synth example is not in the results however, the user can give feedback to the system to improve the search. To give feedback, the user marks which of the *rating results* are irrelevant (by double clicking to remove the results) and rates each remaining relevant result by moving it closer to the center (more relevant) or farther from the center (less relevant). All relevant results are added to the relevant set, Z. The relevant set includes all the relevant examples for the current search session and the initial examples provided by the user.

Our initial estimate of the target concept is given by the query example. We then refine our target concept by creating a weighted average of the feature time series of relevant examples, weighting examples based on the user-provided ratings. In combining these, we are dealing with time series that may be of different lengths, since different synthesizer parameter settings can result in sounds of different lengths.

Dynamic time warping [15] can help us out in this scenario. We adopt *Prioritized Shape Averaging* (PSA) presented in [16] to create a weighted average of the time series. To average many time series, this method first performs agglomerative clustering (a type of hierarchical clustering that results in a binary tree) on the time series, using DTW as the distance function. It then aligns (using DTW) and averages pairs of time series from the bottom of the tree on up, weighted according to the user-provided rating. For further details on PSA, we refer the reader to [16].

Again, we treat each feature independently, and we use the PSA method to average the relevant time series for each feature. The resulting weighted average is our new estimate

of the target concept, called the *refined query* $\bar{\mathbf{X}}$.

In addition to refining our query, we also refine our distance measure in response to the user's relevance feedback. Recall that each of the examples presented to the user for rating is the closest one to the query along one of the 14 feature dimensions. To refine the weight w_i applied to each of the 14 features we use a simple inverse standard deviation relevance feedback mechanism, similar to those in the MARS and MindReader (when constrained to weight each feature dimension independently) systems [18]. However, since we are dealing with time series, we calculate distance in the weighted variance function using DTW rather than the difference function. The calculation is as follows:

$$w_i = \left(\frac{1}{\sum_{k=1}^{|Z|} s_k} \sum_{k=1}^{|Z|} s_k DTW(\mathbf{y}_i^k, \bar{\mathbf{x}}_i)^2 \right)^{-\frac{1}{2}} \quad (2)$$

where w_i is the weight of the i^{th} feature, s_k is the user's similarity rating, $\mathbf{y}_i^k$ is the time series of the i^{th} feature of the k^{th} relevant example, and $\bar{\mathbf{x}}_i$ is the time series of the i^{th} feature of the refined query.

4. FUTURE WORK

Currently, the system can only search synthesizer parameter combinations that have been added to the database. In our initial tests, we used 10,000 samples from a 15 parameter synthesizer. For more complex synthesizers, an adequate sampling may not be possible due to space requirements. We plan to extend this work to also search the full parameter space, while maintaining the current interaction paradigm. We also plan on running a user study that evaluates this software both as an audio retrieval tool and as a creativity support tool.

5. CONCLUSION

In this work, we presented system for searching the space of a synthesizer by querying a database of audio synthesizer patches using "soft-examples" (i.e. examples that have some but not all of the characteristics of the desired sound) such as vocal imitations as input. To allow for such "soft-examples" and account for the limitations of the human voice, the system leverages the information in how perceptual audio features of the sounds change over time, using both absolute and relative time series representations of features and a weighted dynamic time warping as the distance function. The query and the feature weights are interactively refined through user-provided relevance feedback on the search results. Using this system, a user simply needs to know what they are looking for, give it an initial example (e.g. using their voice to imitate the target), and be able to rate how similar example sounds are to their target sound.

6. ACKNOWLEDGEMENTS

This work was supported by NSF Grant Nos. IIS-1116384 and DGE-0824162.

7. REFERENCES

[1] R. Bencina. The metasurface: applying natural neighbour interpolation to two-to-many mapping. In *Proc. of NIME*, 2005.

[2] D. S. Blancas and J. Janer. Sound retrieval from voice imitation queries in collaborative databases. In *Proc. of AES 53rd Int'l Conference*, London, UK, 2014.

[3] P. Dahlstedt. Evolution in creative sound design. *Evolutionary Computer Music*, pages 79–99, 2007.

[4] P. Esling and C. Agon. Multiobjective time series matching for audio classification and retrieval. *IEEE Transactions on Speech Audio and Language Processing*, 21(10):2057–2072, 2013.

[5] R. Ethington and B. Punch. Seawave: A system for musical timbre description. *Computer Music Journal*, 18(1):30–39, 1994.

[6] S. Fasciani and L. Wyse. A voice interface for sound generators: adaptive and automatic mapping of gestures to sound. In *Proc. of NIME*, 2012.

[7] R. Garcia. Growing sound synthesizers using evolutionary methods. In *Proc. of Workshop on Artificial Life Models for Musical Applications*, 2001.

[8] S. Heise, M. Hlatky, and J. Loviscach. Aurally and visually enhanced audio search with soundtorch. In *Proc. of International Conference Extended Abstracts on Human factors in Computing Systems*, 2009.

[9] A. Horner, J. Beauchamp, and L. Haken. Machine tongues xvi: Genetic algorithms and their application to fm matching synthesis. *Computer Music Journal*, 17(4):17–29, 1993.

[10] C.-Z. A. Huang, D. Duvenaud, K. C. Arnold, B. Partridge, J. W. Oberholtzer, and K. Z. Gajos. Active learning of intuitive control knobs for synthesizers using gaussian processes. In *Proc. of Int'l Conference on Intelligent User Interfaces*, Haifa, Israel, 2014.

[11] C. G. Johnson and A. Gounaropoulos. Timbre interfaces using adjectives and adverbs. In *Proc. of NIME*, 2006.

[12] G. Lemaitre and D. Rocchesso. On the effectiveness of vocal imitations and verbal descriptions of sounds. *The Journal of the Acoustical Society of America*, 135(2):862–873, 2014.

[13] P. McLeod and G. Wyvill. A smarter way to find pitch. In *Proc. of Int'l Computer Music Conference*, 2005.

[14] A. Momeni and D. Wessel. Characterizing and controlling musical material intuitively with geometric models. In *Proc. of NIME*, 2003.

[15] M. Müller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin Heidelberg, 2007.

[16] V. Niennattrakul and C. A. Ratanamahatana. Shape averaging under time warping. In *Proc. of Int'l Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2009.

[17] G. Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. Technical report, IRCAM, 2003.

[18] Y. Rui and T. Huang. Optimizing learning in image retrieval. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 236–243 vol.1, 2000.

[19] D. Stowell. *Making music through real-time voice timbre analysis: machine learning and timbral control.* PhD thesis, Queen Mary University of London, 2010.

[20] R. Vertegaal and E. Bonis. Isee: An intuitive sound editing environment. *Computer Music Journal*, 18(2):21–29, 1994.

[21] D. L. Wessel. Timbre space as a musical control structure. *Computer Music Journal*, 3(2):45–52, 1979.

[22] M. J. Yee-King. *Automatic sound synthesizer programming: techniques and applications.* PhD thesis, University of Sussex, 2011.