A VIRTUAL ORCHESTRA FOR HUMAN-COMPUTER MUSIC PERFORMANCE

Roger B. Dannenberg
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

A virtual orchestra was implemented to augment a human jazz big band with the addition of violins, violas, and cellos. In an effort to maximize musical quality, the orchestra uses a high-quality studio recording of each individual part, yet adapts to the tempo of the band in real time using a custom multi-channel PSOLA algorithm. The tempo is tracked through a foot-tapping interface, with additional cueing that allows for deviation from the nominal score and recovery from counting errors. Output is through an array of 8 high-quality studio monitors arranged to provide the illusion of many point sources configured as an acoustic ensemble.

1. INTRODUCTION

Human-computer music performance (HCMP) [8] refers to a developing practice of integrating human and computer performers. Unlike interactive systems for experimental art music [11] that one might typically find at ICMC performances, or even computer accompaniment systems [6] that rely on score following and models of tempo adjustment to play music with expressive timing, HCMP specifically aims to address the performance of popular music forms such as rock, jazz, and folk, where tempo is mostly steady, but improvisation is prominent.

While popular music is often dismissed as simple and uninteresting, there are many technical challenges posed by HCMP that have not been solved (even though interactive music systems have been around for decades). One problem is that popular music (as we will call it for lack of a better term) is largely improvised in the sense that chord voicings, strumming styles, and drum patterns, and even vocal melodies are not spelled out completely by the score and are likely to vary from one performance to the next. Furthermore, even the structure in terms of introductions, repetitions, order of sections, and endings can be altered in the middle of a performance.

In spite of this ambiguity, musicians are expected to conform quite closely to chordal, melodic, and stylistic structures, requiring them to be precisely synchronized at all times. The present work explores the problem of synchronization both at the level of beats and that of high-level musical structure in HCMP.

2. THE VIRTUAL ORCHESTRA CONCEPT

2.1. Requirements

We set out to create a string orchestra that could play along with a live jazz band. Our musical goal was that the strings plus jazz band should sound as good as possible. We decided to emphasize practical considerations and reliability over exotic or cutting-edge research.

One exception to this set of priorities is that we feel that HCMP must extend the capabilities of humans rather than replace them. We could have easily had a human play string parts on a keyboard connected to a string synthesizer. This would be simple and robust, and with work might even sound good, but our objection is that it takes the entire attention of an expert musician who might otherwise play piano or guitar or some other instrument. A similar problem would exist with a conducting interface [2, 5], which would require the conductor to simplify gestures to be reliably interpreted by a computer. We much prefer systems that need no extra personnel to operate, yet bring new capabilities to the human ensemble.

2.2. Components

Our approach consists of several components. First, we have music representation issues: How will string parts be created, represented, and translated all the way from score to sound? Second, synchronization is critical: How will we keep the string parts synchronized to the band? Third is sound generation: How will we make convincing acoustic string sounds electronically? Finally, there is the diffusion problem: How will we organize and project string sounds into the hall?

The representation was solved through a design process involving the arranger, John Wilson, who was commissioned to write for jazz band and strings. We decided to structure the string parts as a sequence of sections. This allowed for interplay between the band and the strings. It also had functional purposes: Because the strings were silent at many times, each entrance could be cued separately. If anything went wrong, there would soon be an opportunity to make another entrance. In addition, the sectional nature allowed for efficient recording and editing.

To the computer, the string parts are just sounds that need to be cued to begin on a particular beat and

synchronized to following beats. When each sound ends, the system prepares to play the next.

Synchronization is handled in the simplest way imaginable, yet it was still somewhat difficult. A foot pedal is used to tap beats (in cut time, about 85 taps per minute) to establish the tempo and throughout the performance. A small keyboard is used to cue entrances.

Sound generation is based on the pitch-synchronous, overlap-add (PSOLA) approach to time stretching. The requirement for real-time performance, continuous (every tap) update, latency compensation, and synchronization across multiple (20) channels led to some innovative implementation details.

Finally, sound diffusion is based on multiple (8) speaker systems arranged across the stage. Each of the 20 input channels represents one close-miked string (violin, viola, or cello). Each instrument channel is directed to only one speaker. Rather than a homogenized orchestra sound spread across many speakers, we have individual instrument sounds radiating from multiple locations and mixing in the room as with an acoustic ensemble.

The following sections will describe these components in greater detail along with related work, followed by conclusions.

3. MUSIC ORGANIZATION AND REPRESENTATION

The jazz standard "Alone Together" by Arthur Schwartz was chosen for performance, in part for its title's implicit commentary on human-computer performance. The string parts show off the system's capabilities, including lush counter melodies, alternations with the live horns, chordal backups behind live soloists, and a pizzicato interlude with a live bass soloist.

From a computational perspective, the string parts are organized as a set of sound files. Each file has a list of time offsets corresponding to beat times, and the task of the computer system is to start playing the file at the proper time (on the proper beat) as well as to vary the playback speed so that the designated file time offsets synchronize with beats in the live performance.

The sound files were recorded two or three tracks at a time in a studio using close microphones to capture a dry sound. To create a realistic performance situation for the players, we first recorded a click track, then recorded the actual live rhythm section (using headphones to stay with the click track). Finally, the string players played along while listening to the rhythm section over headphones. We feel that this approach gives the strings a useful rhythmic and pitch reference, avoiding any tendency to play the parts "straight" or mechanically as might happen playing along with a simple click track. On the other hand, the original click track reference makes it easy to identify beat times in the recordings, which is necessary for their ultimate synchronization to the live band.

The recordings were mixed to 20-track sound files representing 12 violins, 4 violas, and 4 cellos. Each file represents a set of contiguous measures beginning at an entrance of the string ensemble and ending at a point where the entire ensemble has a rest of significant duration (at least 16 measures).

4. SYNCHRONIZATION

Synchronization requires us to begin the playback of each sound file at the proper moment and at the proper tempo, and to track the tempo and beat times of the band until the end of each file.

In principle, beat tracking [3] and score following [6] could be used for automatic synchronization. However, beat tracking – especially in real time – is not highly reliable, and there are practical consideration such as how to start the strings and the band at the same time on cue from a conductor. B-Keeper is a related system based on beat-tracking. [10] Score following might work sometimes, but it would require following individual non-improvised parts played by acoustic instruments in a dense and loud ensemble. Furthermore, there are entrances for the strings in sections that are entirely (except for the chord structure) improvised by the rhythm section. We decided to use a simple foot-tapping interface [4] to communicate beat times and use an additional keyboard to cue some of the entrances.

The beat and tempo detection software interprets input according to different states. In the "initial" state, input is ignored until there are 3 successive taps at approximately equal time intervals. This sets an initial tempo and causes a transition to the "run" state. In the run state, the software uses linear regression over up to 6 previous taps to predict the next tap time. A tap that arrives within 1/3 beat period of the expected time is added to the list of beats, and a new regression is performed to update the estimated tempo and predict the next beat time. If no tap arrives during the expected time window, the system waits for a tap near the following beat. If there is no tap near this second estimated beat time, the system goes back to the "initial" state.

The main output of the tapping system is the linear regression of recent beats. This provides a mapping from time to beat number that can be used to schedule events in the future. (See Figure 1). Because of the latencies in tap detection, sound synthesis, and the operating system, it would be very complicated to create event-driven software, e.g. have the tap detection software forward taps directly to the sound synthesis component. The taps would inevitably arrive a little late, the software synthesis would inevitably need the information a bit early, and synchronization would suffer. Instead, we send an entire mapping from time to beat (which is linear and described by just the slope, a beat offset, and a time offset). When it comes time to compute audio, the future output time of the audio (estimated by the PortAudio library) can be mapped easily to an estimated beat time and tempo. This approach simplifies reasoning about timing and synchronization.

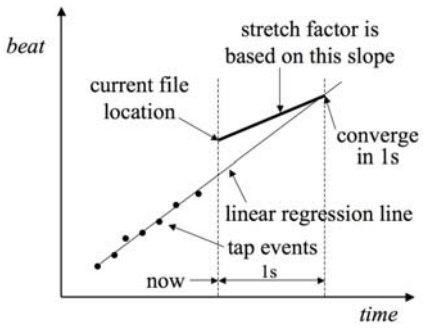

Figure 1. Timing diagram. (For clarity, the 1s time interval is not drawn to scale with tap timing.)

We have discussed beat-level synchronization, but not higher-level cues. The idea here is that taps provide a mapping from time to beats where beats are integers with an arbitrary offset. Thus, we can determine the tempo and the times that correspond to beats, but this does not tell us global information such as measure number. When a cue occurs, the system finds the nearest beat number b_{tap} that is nearest in time. Using a table containing a sequence of cues, it finds the absolute beat number b_{cue} for this cue. The difference, $b_{cue} - b_{tap}$, can be stored and added to the time-to-beat map to obtain an absolute score position as a function of time.

5. SOUND GENERATION

Sound generation is coupled to the tapping component through the time-to-beat mapping which is typically updated in response to each new tap. The goal of sound generation is to have the audio output correspond to the currently estimated beat position (we treat beats as continuous, so it makes sense to say that the current beat position is 23.17, or 17% of the way from beat 23 to beat 24). To accomplish this, we cannot simply jump to the corresponding location in the audio file, which would create obvious and unnatural audio artefacts. Instead, we must continue a smooth playback of the audio but modulate the stretch factor to speed up or slow down in order to synchronize. We will next describe the time stretching process and then return to the problem of synchronization.

Time stretching uses the PSOLA approach. [12] Our time stretching is mostly provided by the Elastique library [7] from Zplane, which provides for the time stretching of a single channel of audio by a given stretch factor. The system works as follows (see Figure 2): First, audio to be processed is analyzed (off-line, in our case) to detect and label pitch periods in the original audio. The labels provide not only locations and periods, but some spectral properties used by the stretch algorithm. At runtime, the complete analysis data are provided to the time stretch module, but audio is processed incrementally. To process audio, the audio stream is segmented into pitch periods, and each period is isolated by multiplication by a smoothing window, centered on the pitch period, but overlapping with adjacent periods. The windowing is organized so that if the windows are summed at their original spacing, the original waveform is recovered. To stretch the sound, windowed periods are occasionally output twice (using the pitch period to determine spacing, as shown in Figure 2), thus extending the sound. To contract the sound, windowed periods are occasionally dropped. Of course, the rate of duplicating or dropping periods determines the overall stretch factor, but the algorithm has some leeway in deciding which periods to duplicate or drop, and presumably duplicating or dropping highly periodic portions of the signal will minimize the artefacts. In practice, the author was unable to hear any artefacts using small stretch factors, and the most obvious give-away is that as the stretch factor increases, vibrato begins to sound unnatural. (There is no attempt to remove and restore vibrato, but in a dense collection of 20 strings, even these artefacts will be masked.)

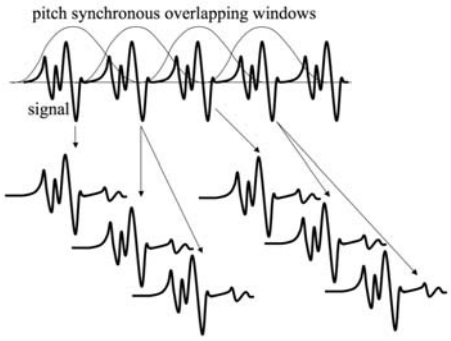

Figure 2. Pitch Synchronous Overlap Add (PSOLA)

Our system must modulate the stretch factor continuously to track a continuously varying tempo. Imagine a contrived situation where a momentary change in stretch factor causes the duplication of a period in track A but not track B. The stretch factor is then returned to 1.0, so now track A is slightly behind B. This could be repeated arbitrarily many times, causing A to drift further and further behind B. Considering that only 10 periods of a cello's low C last a total of 153ms, the potential for loss of synchronization over time is worrisome.

To avoid any long-term drift between tracks, we use an active servo mechanism to measure the drift and drive it to zero. This is combined with additional servo control to synchronize the audio playback to the beat-to-time map. To begin with, we need to sense the sound file position of each of the 20 instances of the time stretch algorithm. The Zplane library operates incrementally, issuing a request for input data before generating each output buffer. By counting how many samples have been read as input, we can estimate the file position (this is only an estimate because there is an unknown but small amount of read-ahead).

Our goal now is to update the stretch factor for each time stretcher instance. We know that the current tempo is, say, r_c, and the recorded tempo is r_r, so the stretch factor should be r_r / r_c, but this ignores the fact that the estimated tempo can jump around as new taps are entered. This not only leads to drift as tempo errors accumulate, but it could cause sudden and unnatural

speed changes. What we actually do is a bit more complicated. We look ahead in time on the time-to-beat mapping and solve the following problem (see Figure 1): Given the current estimated file position, what tempo should we adopt now in order to synchronize with the time-to-beat mapping in exactly 1s? Since each time stretcher has a slightly different file position, this calculation yields a slightly different slope and stretch factor for each one, but in aggregate, all tracks will converge to the same file location and that file location will converge to the current real-time performance beat position. Similar rate adjustment has been described [5, 9] , but this may be the first attempt at synchronizing many independent PSOLA time-stretching processes.

Loosely coupled to all this activity is a process that reads 20-channel sound files, de-interleaves the samples, and inserts them into FIFO queues, one for each time stretcher. This allows each time stretcher to manage a slightly different file read position. We read data from disk in large blocks for efficiency, but use a low-priority task that can be pre-empted for low-latency audio computation. By keeping ahead of the audio processing, we avoid blocking the audio computation to wait for a disk read to complete.

6. CONCLUSIONS

None of the techniques described here (tapping, time stretching, multi-channel audio) are entirely new, but even after decades of interactive computer music it is not common to have high-quality multi-channel synchronized audio used in live performance. We are unaware of any precedent. There is even a demonstrated need for this as seen for example in Quadraphenia performed by the Who with extensive but troublesome backing tapes in the 70's and the common use of click tracks on backing "tapes" in venues such as theme parks and cruise ships. Some programs such as Ableton Live [1] incorporate some support for time stretching and live synchronization, but Live does not provide a direct or complete solution to human-computer music performance (HCMP).

We gave one performance with our experimental system. By chance, there was an "extra" percussionist with nothing else to do in this piece, so she provided the taps. One interesting problem occurred in rehearsal where the tapper was naturally listening to the strings but started to tap along with them rather than the band, causing the system to drift out of synchronization. As soon as this became apparent, she began tapping with the band to correct the problem, but now the taps were falling outside of the 1/3 beat window, causing them to be ignored. This failure illustrates the subtleties of even a problem as simple as tapping beats in live performance.

After learning a few lessons, the public performance went very well, and we have begun to explore other aspects of HCMP. In particular, we feel we "cheated" a bit by dedicating a performer to the tapping task. We have since been working with a smaller group where the tapping, cueing, and control tasks are performed by someone who is also playing an acoustic instrument. This creates a host of human-computer interaction problems, and we plan to address them in a series of future systems.

7. REFERENCES

[1] Ableton AG. *Ableton Reference Manual,* 2010.

[2] Baba, T., Hashida, M., and Katayose, H. "'VirtualPhilharmony': A Conducting System with Heuristics of Conducting an Orchestra" in *Proceedings of the 2010 Conference on New Interfaces for Musical Expression* (NIME 2010), ACM Press, 2010, 263-270.

[3] Brossier, P. *Automatic Annotation of Musical Audio for Interactive Applications*. Ph. D. thesis, Department of Electronic Engineering, Queen Mary, University of London, 2006.

[4] Dannenberg, R. "New Interfaces for Popular Music Performance," in *Seventh International Conference on New Interfaces for Musical Expression: NIME 2007 New York,* New York, NY: New York Univ., June 2007, pp. 130-135.

[5] Dannenberg R. and Bookstein, K., "Practical Aspects of a Midi Conducting Program," in *Proceedings of the 1991 International Computer Music Conference*, ICMA, (October 1991), pp. 537-540.

[6] Dannenberg, R. and Raphael, C. "Music score alignment and computer accompaniment." Commun. ACM 49, 8 (Aug. 2006), pp. 38-43.

[7] Flohrer, T. *Elastique 2.0 SDK Documentation,* zplane.development, 2007.

[8] Gold, N. and Dannenberg, R. "A Reference Architecture and Score Representation for Popular Music Human-Computer Music Performance Systems," in *New Interfaces for Musical Expression,* 2011.

[9] Lee, E., Karrer, T. and Borchers, J. "Toward a framework for interactive systems to conduct digital audio and video streams." *Computer Music Journal*, 30(1) (Spring 2006), pp. 21-36.

[10] Robertson, A. and Plumbley, M., "B-Keeper: A beat-tracker for live performance," in *New Interfaces for Musical Expression,* 2007, pp. 234-237.

[11] Rowe, R. *Interactive Music Systems*. MIT Press, Cambridge, MA (1993).

[12] Schnell, N., Peeters, G., Lemouton, S., Manoury, P., and Rodet, X., "Synthesizing a Choir in Real-Time Using Pitch Synchronous Overlap Add (PSOLA)," in Proceedings of the 2000 International Computer Music Conference, 2000, pp. 102-108.