A MOBILE CONDUCTING APP WITH A SWITCHING BEAT-FOLLOWING ALGORITHM

Zengguang Wu, William Raffe and Andrew Johnston
University of Technology Sydney, Australia

ABSTRACT

Conducting an orchestra had been a privilege reserved for professional conductors, until there are music conducting systems that can follow the user's lead. The music playing and beat tracking components of conducting systems have been well developed, but the beat-following algorithm has room for improvement. In this paper we present a mobile conducting app with a novel beat-following algorithm, which can switch between different behaviours to accommodate different musical situations and user preferences, and allow the user to make their own interpretations of classic orchestral works.

KEYWORDS

Mobile, Conducting App, Switching, Beat-Following Algorithm

1. INTRODUCTION

Orchestral music has a significant impact on human culture as a kind of music that can bring emotions and tell stories with a group of instruments. Conducting an orchestra had traditionally been a privilege reserved for highly trained professional conductors. However, with modern interactive conducting systems, a wider audience of enthusiasts can now experience the creative freedom of conducting a large digital orchestra.

As a form of play, a conducting system is an example of *paidia* as opposed to *ludus* (Caillois, 1958/2001). As a game, a conducting system is a *non-game*, "a form of entertainment that really doesn't have a winner, or even a real conclusion" (Iwata, 2005), similar to simulation games like SimCity and The Sims. As a music game, a conducting system is an *instrument game* (Pichlmair & Kayali, 2007), which has a reverse mechanism to rhythm games. In a rhythm game, the user follows the music, and performs actions when they need to; in a conducting system, the user gives beats when they want to, and the music follows their lead.

2. BACKGROUND: MUSIC CONDUCTING SYSTEMS

Music conducting systems have appeared in various forms: multimedia installations, such as Personal Orchestra (Borchers et al., 2004), Mendelssohn Effektorium (WHITEvoid Studio, 2014); computer programs, such as Semi-conductor (Google Creative Lab, 2018), Live Tempo of Avid Sibelius (Avid Technology, 2023); and mobile apps, such as Bravo Gustavo (Hello Design, 2010), vMaestro (Lim & Yeo, 2014), Airconductor (Micro App Brewery, 2019). All these systems have three major components: a music player, a beat tracker, and a beat-following algorithm to let the music follow the beats. The music playing and beat tracking components have been well developed, but the beat-following algorithm has room for improvement.

2.1 The Music Player

A conducting system can work with either a MIDI sequencer (vMaestro, Airconductor) or some recorded sound (Personal Orchestra, Bravo Gustavo). MIDI sequencers are more flexible with tempi, and can chase or wait for the user without the sound quality being compromised. Recorded sound in conducting systems is often performed by prestigious artists, with the artistic value being an objective of the system.

2.2 The Beat Tracker

Beat tracking has previously been achieved with various types of input. Common input devices such as keyboards (Avid Sibelius) and touch screens (Bravo Gustavo) are simple but efficient ways to get beats from the user. Physical devices with sensors such as accelerometer (Bravo Gustavo) and gyroscope (vMaestro) can gather beats from the user's motions, and therefore enable the use of real-world conducting gestures. Cameras (Semi-conductor) provide the most freedom for the user, allowing them to conduct with bare hands, but they also pose the biggest challenge for the algorithm.

2.3 The Beat-Following Algorithm

Looking at past and present conducting systems, we see two core beat-following algorithms, one responsive and flexible, the other smooth and stable; and two complex algorithms that combine the core ones, one by mixing them, and one by weighting between them.

To show how these algorithms work, we use position-time graphs as below. The horizontal t-axis is time in the real world; the vertical x-axis is time in the music file, which is also referred to as position. On the graph, each orange is a beat from the user. The solid line is the system's position in the music file, with a steep slope indicating a fast tempo. A vertical section of the solid line shows the system fast-forwarding to chase the user; a horizontal section of the solid line means the system pausing to wait for the user.

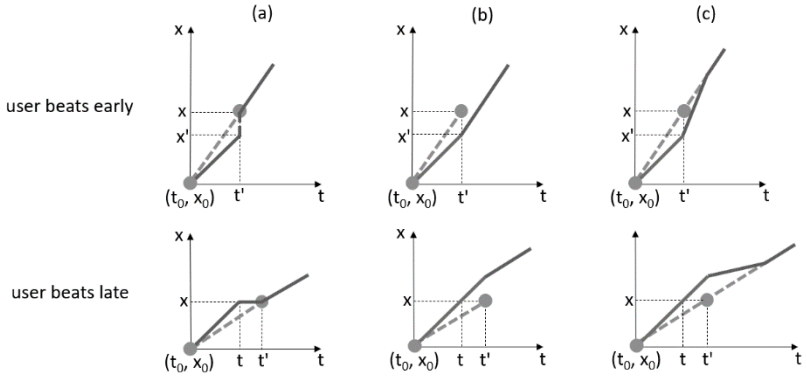

Figure 1. Position-time graphs of existing algorithms

Figure 1(a) shows the algorithm of Bravo Gustavo. In this responsive algorithm, every beat from the user corresponds to a beat in the music. If the user beats early, the music fast forwards to the corresponding beat; if the user beats late, the music waits before the corresponding beat. At each beat from the user, a new tempo is calculated from the time difference between the last two beats, referred to as "the simplest option" (Dannenberg & Bookstein, 1991), and no averaging is involved. This algorithm gives the user more control over the music, but can cause disruptions in playback.

Figure 1(b) illustrates the algorithm of several MIDI-based systems (vMaestro, Airconductor, Mendelssohn Effektorium). In this smooth algorithm, there is no chasing or waiting. At each beat from the user, the tempo is set to the average rate of the last several beats, referred to as "a history buffer" (Dannenberg & Bookstein, 1991). This algorithm can prevent the tempo from shaking up and down, and provide smooth playback. But there is no correspondence between beats in the music and beats from the user, and the music cannot follow the user's individual beats.

Figure 1(c) shows a solution by Personal Orchestra, a mix of the two simple algorithms. In this algorithm, chasing is not fast forwarding, but playing at a tempo faster than the user's desired tempo, to catch up with the user after a certain time; waiting is done in a similar but opposite way. This algorithm allows chasing and waiting without causing disruptions in the playback, although going further than the user's intended tempo change may not sound natural.

In comparison, the algorithm of the Live Tempo feature of Avid Sibelius is a weighting between the two simple algorithms. In the options, there is a "sensitivity" slider with "smoother" at one end and "more

responsive" at the other, which can be adjusted according to the character of the piece and the preference of the user. But during a performance, the slider is not accessible, and the weighting remains the same.

However, passages and beats in music are not all the same. For example, Dannenberg and Bookstein (1991) "found a conflict between the necessity for fine control of tempo in the slow, expressive passages, and the necessity in very rhythmically stable passages for the computer to be less sensitive". Such differences in musical situation need to be accommodated by the algorithm of a conducting system. Baba et al. (2010) used performance templates to help with differences in musical situation, but when the user's intention is significantly different from the template, it may not work well. Such differences in user preference also need to be accommodated by the algorithm of a conducting system.

3. IMPLEMENTATION: A MOBILE CONDUCTING APP

Having identified the beat-following algorithm as a niche area of music conducting systems, now we present our implementation, a mobile conducting app with a switching beat-following algorithm. This algorithm is our main focus, so for the music playing and beat tracking components, we took simple but effective approaches previously used in other systems.

3.1 The Music Player

The app uses Swift's AVMIDIPlayer, and works with specially made MIDI files, where the tempo is set to 60 beats per minute (bpm) despite the actual tempo. 60 bpm is one beat per second, so the beat positions are exactly the integer positions of the MIDI file, which makes it easy for the app to find the beat positions.

3.2 The Beat Tracker

The app uses iPhone's gyroscope to catch the user's beats. The user is expected to hold the mobile phone in their right hand, with the top of the phone pointing to the front, and the screen facing to the left. In real-world conducting, each beat of the conductor is a downward motion followed by an upward bounce (McElheran, 2004, Ch. VI), and the orchestra responds when the conductor's hand hits the lowest point and starts the bounce. In our mobile conducting app, such gestures bring a positive rotation around the z-axis followed by a negative rotation, and the app detects a beat when the rotation rate goes first above 3 and then below 0.

3.3 The Beat-Following Algorithm

The app has a novel beat-following algorithm, which can switch between different behaviours to accommodate different musical situations and user preferences. In the app, each MIDI file for the MIDI player comes with a text file for the algorithm. In the text file, each row corresponds to a beat in the music, and contains settings of the app's behaviour at this beat. The settings can be adjusted via a user interface.

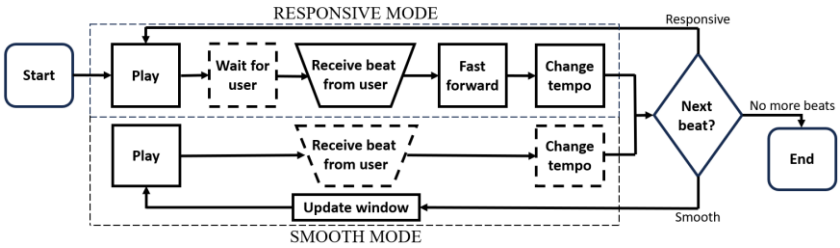

Figure 2. The switching beat-following algorithm

Figure 2 is an illustration of the algorithm. The switching is our main innovation; the responsive mode is the same as the responsive algorithm mentioned above; the smooth mode is similar to the smooth algorithm, but here each beat from the user corresponds to a beat in the music. In the responsive mode, the app follows

the user closely, and can wait for/chase the user; the user's beat is compulsory, and the music will only play on when it receives a beat from the user. In the smooth mode, there is no waiting or chasing; the user's beat is optional, and the music plays on whether the user beats or not; for each beat in the music, there is a time window for the user to give the beat; beats received between windows can be ambiguous, and will be ignored.

$$c = \frac{t' - t_0}{t - t_0}$$

$$c' = \frac{c + m}{1 + m}$$

$$v' = \frac{v}{c'}$$

Figure 3. The calculation of the new tempo

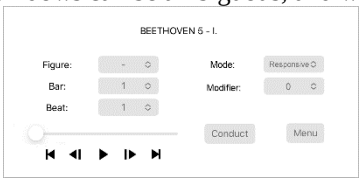

Figure 4. The User interface

In both modes, the new tempo is calculated as shown in Figure 3. Suppose the previous beat of the music was at time t_0, the current beat is expected at time t, and the user's beat is received at time t'. First, the indicated duration change c is calculated as the ratio between the new beat duration t'-t_0 suggested by the user's beat and the old beat duration t-t_0 at the current tempo. Then, the modified duration change c' is calculated from $(c+m)/(1+m)$, where m is a modifier that determines the app's resistance to tempo change. A positive m would shrink the change and stabilise the tempo; the larger m, the stronger the impact; an m between 0 and -1 would magnify the change. Finally, we multiply the current tempo v by the reciprocal of the duration change, $1/c'$, which is the tempo change, and get the new tempo v'.

Figure 4 shows the app's user interface. The user can navigate inside the piece with the slider or the buttons underneath (previous figure, previous beat, play, next beat and next figure). When the user presses the play button, the app plays a pre-recorded performance of the piece. The pickers on the left are another way to navigate, and the pickers on the right allow the user to specify the app's behaviour at this beat, namely the mode and the modifier m. When the user is ready to conduct, they hit the "Conduct" button, and the gyroscope will start catching the user's beats.

4. THE ALGORITHM IN ACTION

Similar to the corresponding separate algorithms, the responsive mode is more flexible, and gives the user more control; the smooth mode is more stable, and keeps the music flowing. For the modifier m, setting it to zero would let the app follow the user's tempo change as it is, while setting it to a very large number would prevent the tempo from changing. With these settings, the app can accommodate beats of different natures, passages of different characters and users of different preferences. To see this, here are some examples.

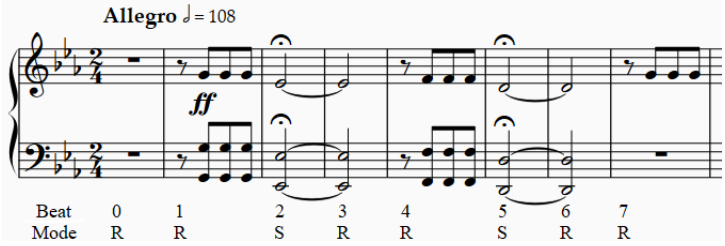

Figure 5. Beethoven: Symphony No. 5 (1808, excerpt)

Figure 5 is the opening of Beethoven's Fifth Symphony. At the start of a piece, the orchestra needs two instructions from the conductor: when to start, and at what tempo. [6, Ch. XIV] Before the first beat of the music (beat 1), the conductor often gives an extra beat (beat 0); the orchestra start playing at the conductor's beat 1, at the tempo indicated by beats 0 and 1. These beats are supposed to be followed closely, so here we use the responsive mode.

For a fermata, the app clearly needs to allow waiting for the beat after the fermata, but this is not enough. After a fermata it is a new start, and we need an extra beat for the user to set the tempo. In this example, for the first fermata, beat 3 has been added to the score to help with beat 4; for the second, the fermata has been moved from beat 6 to beat 5, this has no impact on the long note, but allows beat 6 to help with beat 7. Beats 3, 4, 6 & 7 all need to be followed closely, and call for the responsive mode.

At the above beats, the app gives the user full control by allowing waiting and chasing, but at beats 2 and 5, such control is unwanted. Sitting on these beats are the fourth notes of the fate motif, and playing them early or late would upset the rhythm, so an early or late beat from the user is more likely an error than an intention. Therefore, here we use the smooth mode to keep the fate motif in good shape. As shown here, the app can accommodate beats of different natures, and handle them in the appropriate ways.

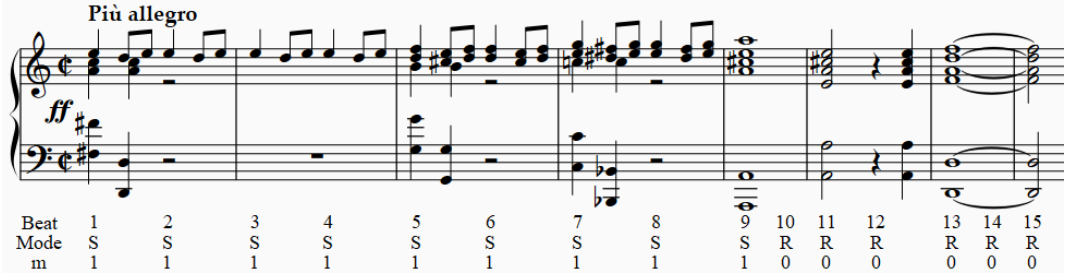

Figure 6. Brahms: Symphony No. 1 (1876, excerpt)

This excerpt from the finale of Brahms's First Symphony (Fig. 6) features good examples of "rhythmically stable passages" and "slow, expressive passages" in the words of Dannenberg and Bookstein. The first four bars are densely populated with a three-note phrase derived from the main theme of the movement, which is very rhythmic, and tends to keep flowing without being disrupted. For such a passage, we cannot allow waiting or chasing, so we use the smooth mode; tempo changes are possible, but a sudden tempo change would break the momentum, so we set the modifier to 1, and hence reduce the user's tempo changes by a half. The last four bars recall a chorale theme from the movement's introduction, which is a series of long chords, and invites the conductor to do anything they want with the tempo. For such a passage, we need to allow waiting and chasing, and use the responsive mode; the modifier is set to 0, and the user's tempo changes are followed faithfully. In this example, passages of different characters are treated in the way they are supposed to be treated.

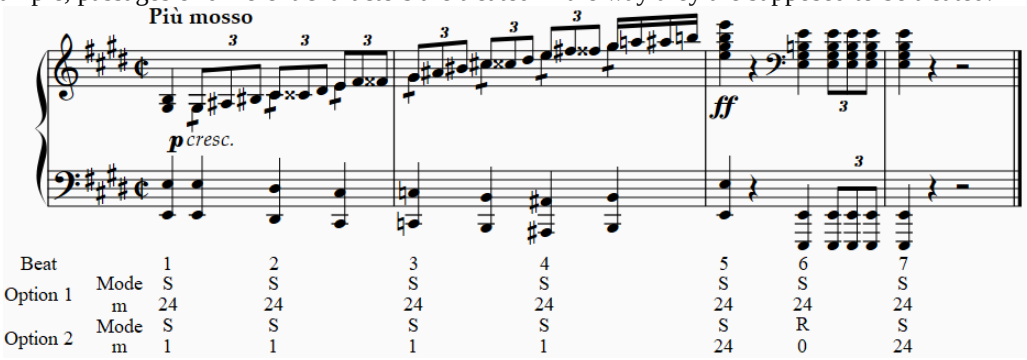

Figure 7. Rachmaninov: Symphony No. 2 (1908, excerpt)

Some passages, such as the final bars of Rachmaninov's Second Symphony (Fig. 7), can be played in very different ways. As a typical loud and fast ending, it can be kept to a fast tempo, bringing the symphony to a straightforward close. To do this, we simply set all beats to the smooth mode, and m to a large number to prevent the tempo from changing. But if we want, we can also add some drama here. The first five beats have to be in the smooth mode, but m can be small to allow tempo changes. At beat 5, there is only one chord on the beat, forming a break between beats 5 & 6, making it possible for us to change to a slower tempo at beat 6 to add some weight to the final chords. To do this, we set beat 6 to the responsive mode with an m of zero. Then, the final chords will be played at the tempo suggested by beats 5 and 6 from the user. With the proper setting, the app can play any passage in the user's preferred manner.

5. CONCLUSION

In the current implementation, the new tempo at each beat is a modification of the current tempo according to the beat from the user. Some musical situations, such as "rit. … a tempo" (gradually slow down, then suddenly

return to the original tempo), cannot be handled by this mechanism, and call for more options. We are going to polish the beat-following algorithm, combine it with more sophisticated music playing and beat tracking, and aim for a better conducting experience.

Orchestral music is a true pearl of human culture. We hope that, with our mobile conducting app, more orchestral music enthusiasts will be able to make their own interpretations of classic orchestral works, and more human hearts will be blessed with the profound sound of orchestral music.

REFERENCES

Avid Technology (2023). Avid Sibelius [Computer software].

Baba, T., Hashida, M., & Katayose, H. (2010). "VirtualPhilharmony": A Conducting System with Heuristics of Conducting an Orchestra. Conference on New Interfaces for Music Expression, 2010.

Beethoven, L. v. (1808). Symphony No. 5 in C minor, op. 67.

Borchers, J., Lee, E., Samminger, W., & Mühlhäuser, M. (2004). Personal orchestra: A real-time audio/video system for interactive conducting. Multimedia Systems, 9(5), 458-465.

Brahms, J. (1876). Symphony No. 1 in C minor, op. 68.

Caillois, R. (1958/2001). Man, play, and games. University of Illinois press.

Dannenberg, R. B., & Bookstein, K. (1991). Practical aspects of a midi conducting program. Proceedings of the 1991 International Computer Music Conference.

Google Creative Lab (2018). Semi-Conductor [Computer software].

Hello Design (2010). Bravo Gustavo [Mobile app].

Iwata, S., (2005). Keynote speech at the Game Developers Conference. https://www.ign.com/articles/2005/03/11/gdc-2005-iwata-keynote-transcript

Lim, Y. K., & Yeo, W. S. (2014). Smartphone-based Music Conducting. Conference on New Interfaces for Music Expression, 2014.

McElheran, B. (2004). Conducting technique: For beginners and professionals. Oxford University Press.

Micro App Brewery. (2019). Airconductor [Mobile app].

Pichlmair, M., & Kayali, F. (2007). Levels of Sound: On the Principles of Interactivity in Music Video Games. DiGRA Conference.

Rachmaninov, S. (1908). Symphony No. 2 in E minor, op. 27.

WHITEvoid Studio (2014). Mendelssohn Effektorium [Multimedia Installation].