6.UAP Project

FunPlayer:

A Real-Time Speed-Adjusting Music Accompaniment System

Daryl Neubieser

May 12, 2016

Abstract: This paper describes my implementation of a variable-speed accompaniment system that can follow along with a real-time MIDI piano performance based on a chord-matching algorithm. I first provide a general background on previously developed accompaniment systems. I then give the implementation details of my project. Lastly, I analyze the performance of the system and offer suggestions for future work that could continue to improve it.

1 Introduction

When trying to play covers of popular songs on the piano, components such as the vocals, drums, or guitar, are lost. Because of this, playing a transcribed version of a song often sounds lacking compared to the original. The system presented here is a variable-speed accompaniment player that fills out those other components with the timing of the cover. As a user plays an instrument in real time, the system plays a digital audio file as an accompaniment that stays with the performer by listening and adjusting.

The software is used as follows. The user inputs a youtube link or audio file of the song they want to learn, as well as its associated accompaniment, for instance a vocal track of the song. The system is then attached to a MIDI input. As the performer plays the song, MIDI events are transmitted to the system, and the system plays a tempo-adjusted audio track to match the estimated tempo of the MIDI notes it receives. The end result is the performer and the accompaniment playing in synchrony. Figure 1 outlines the connections between the inputs to the system with the tempo-adjusted output.

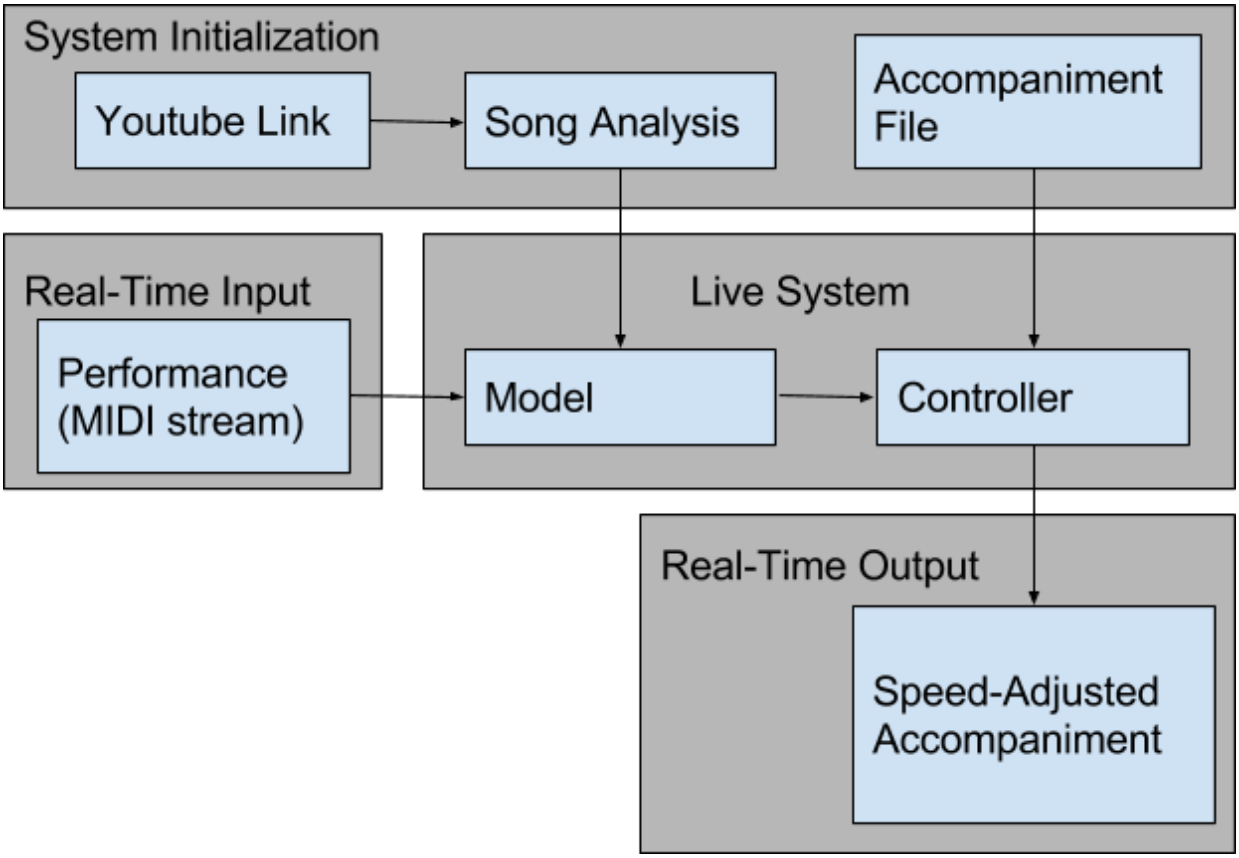


Figure 1: Flow chart of inputs and outputs to FunPlayer system.

1.1 Related Work

Computer controlled accompaniment is not a new problem. In developing FunPlayer, three accompaniment players were researched: Cadenza,[1] Orchestral Accompaniment for Piano,[2] and Antescofo.[3]

Cadenza is an iPad app produced by Sonation that provides orchestral

[1] "Cadenza | FAQ," *Cadenza by Sonation,* 2016. http://www.sonacadenza.com/support/.
[2] Raphael, Christopher, and Yupeng Gu. "ORCHESTRAL ACCOMPANIMENT FOR A REPRODUCING PIANO." (n.d.): n. pag. Web. 7 May 2016.
[3] Arshia Cont. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music.. *International Computer Music Conference (ICMC)*, Aug 2008, Belfast, Ireland. pp.33-40, 2008. <hal-00694803>

accompaniment to violin and singing. They are limited to providing accompaniments for certain songs because they require marking up each of those songs to know what parts are soloist-driven, in order to know how to time the accompaniment. Essentially, every song needs to be specifically tuned to be compatible with their system.

Orchestral Accompaniment for Piano supports a more complex input - piano, which means the system must, unlike Cadenza, handle polyphonic inputs. However, this system also needs to know the exact score in advance.

Antescofo is another real-time score-following system. It can be used to synchronize a live performance with computer-generated music elements, and is continuing to be developed to improve tools for writing and timing computer music interaction.

In contrast to the aforementioned systems, FunPlayer is more flexible in that it only requires an estimate of what kinds of notes will be present in a song, allowing it to operate on a wider variety of songs with less human involvement to initialize the system. However, as a tradeoff for increased accessibility, FunPlayer has a disadvantage in terms of accuracy of score position estimation.

Additionally, both Cadenza and Orchestral Accompaniment for Piano also improve accuracy with multiple iterations of performances by their models learning a performer's habits through each iteration. This was not a priority for the first version of FunPlayer, but it may be possible to learn from these systems and implement a similar

feature in the future.

1.2 Goals

The main design goal for this system is to create a seamless process to create songs and play right away.

The system allows a musician to play a song by ear and make mistakes, but still be able to be accompanied moderately reliably. The ultimate goal is for the system to aid the performer in the entire process of learning a song by ear: adjusting for increases in tempo as the performer becomes more familiar with different sections of a piece and providing real-time audio feedback on incorrect notes.

2 Design & Implementation

The FunPlayer system consists of a model and controller, as well as a set of libraries to help analyze the music and process audio samples to provide a pitch-shiftless tempo-altered playback. This section describes the implementation of the Model and controller and how certain outside libraries are used to complement the system.

2.1 Model

The Model analyzes a real-time stream of note-playtime and pitch information to help the system predict how the rest of the song will be played. It converts note-playtime and pitch information from the MIDI input to inform the Controller a timestamp of the

performer's estimated position in the original song.

The model is based on an analysis of the original song's audio file. The analysis used in the first iteration of FunPlayer breaks the song into sections of different chords, as shown in the Figure 2 below.

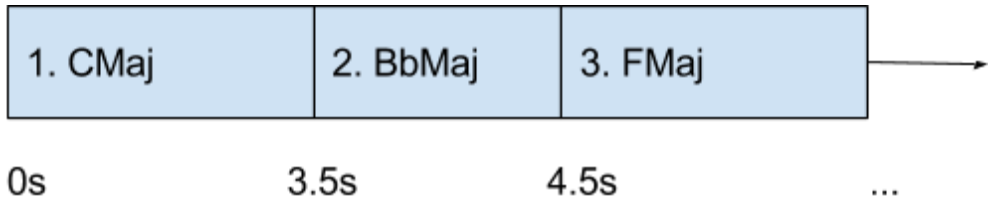


Figure 2. Result of chord analysis used by model

A model has to implement two functions.

- getScore() : Return a numerical value indicating the likelihood of the model being accurate.

  The highest scoring model is the one the system thinks is most likely
- addNote(<Pitch,Time>): Update the model based on new received information. If the note is in line with the model's expectations, it will increase the score.

  Otherwise, it will decrease score.

The system uses the Model as follows. Many possible Models are created and are scored according to how well they conform to observed note pitches and timings. At the time of the latest received note, the highest scoring Model is used to inform the Controller of its timestamp estimation. Two types of models were tested: the Note Assignment Model and

the Tempo Offset Model. The Note Assignment Model was developed first, but due to performance issues, FunPlayer currently uses the Tempo Offset Model.

2.1.1 Note Assignment Model

The Note Assignment Model works by assigning input note to a number of possible *chords* it could belong to. Every combination of assignments constituted a potential model. Based on evenness of tempo and individual likelihoods of each given note belonging to a given chord, all potential models are given a score, and the highest scoring model is used to give direction to the Controller.

The Note Assignment Model works well for simple songs and short songs, but it has two main drawbacks. First, the number of potential models to evaluate increased exponentially with every new note received. And second, even given a "correct" model, it is difficult to translate a set of note assignments into a tempo change input for the Controller. For these reasons, an alternate model design was sought and the Tempo Offset Model was developed.

2.1.2 Tempo Offset Model

In the Tempo Offset Model, each possible model is parameterized with a tempo and time offset from which the played input differs from the original. This gives the model two key advantages over the Note Assignment Model. The first is that the model

description is simpler. Two constants - the tempo and the offset - describe each possible model. In contrast, the number of assignments in the Note Assignment Model scaled linearly with the number of notes.

The second advantage is that Tempo + Offset translates easily to instructions for the Controller, where it seeks to reach the modeled tempo, while minimizing the difference in offset.

Though the optimal choice of what tempo and offset combinations to test may depend on certain expectations of the performer, the system was tested to perform well testing up to 20,000 combinations at a time. The default range of models tests tempo differences between factors of 0.9 and 1.1, and offsets of +/- 2.5 seconds.

The Tempo Offset Model adjusts score in the following way. Each new note's time is adjusted according to the following equation.

adjustedNoteTime = tempo * receivedNotetTime + offset

Then, it searches the chord list to find the chord associated with that time. Based on the likelihood of the note's pitch appearing in that chord, the score is adjusted. A note that is the same as the root of the chord increases score by 2, and a note that is a third or fifth in the chord increases score by 1. Variations in scoring values were tested, but did not make any substantial difference in system performance.

Finally, the score is more heavily weighted towards the most recently received notes. This allows the model to handle tempo changes mid-song by exponentially

reducing the impact of earlier played notes on the score.

As a final enhancement to the system, the offset of the highest scoring model is further adjusted to match the beat markings obtained from BeatRoot[4], a beat-marking command-line tool. BeatRoot further divides the each of the chords obtained in the analysis into individual beats. This is necessary since the other calculations don't take into account where in the chord each note is played. The offset chosen is the one that minimizes the sum of differences between beats and the nearest played notes.

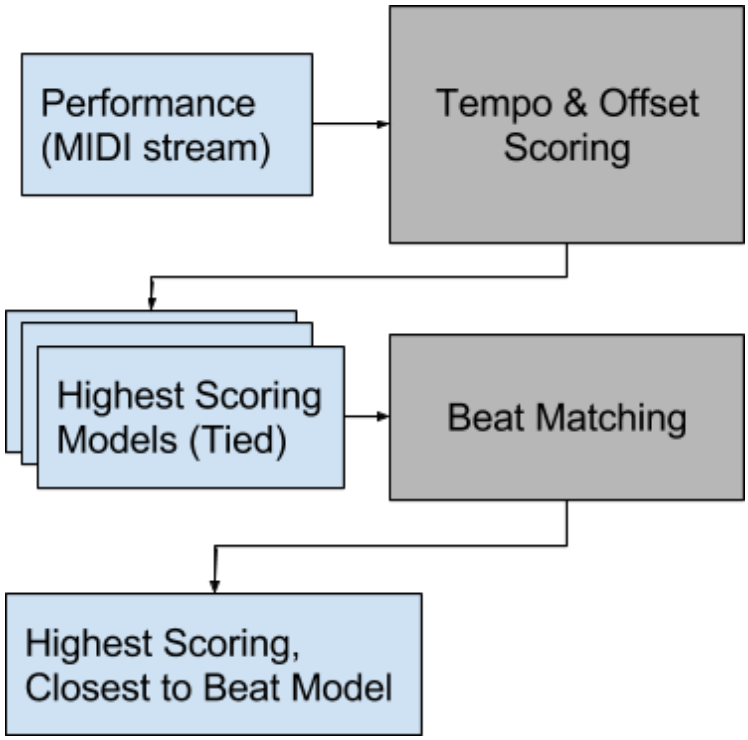


Figure 3: The order of processing the input MIDI stream. Because chords span many seconds, models that differ in offsets by tenths will score the same. Then, select the offset that minimizes distance from notes to expected beats.

[4] Dixon, Simon. "Evaluation of the Audio Beat Tracking System BeatRoot." *Journal of New Music Research* 36.1 (2007): 39-50. *BeatRoot*. Web.

2.2 Controller

The controller's purpose is to adjust the speed of the accompaniment file so that it synchronizes with the input.

The controller first obtains the desired tempo and offset from the model, and calculates the current offset based on how many samples of the accompaniment file have been processed. Then, it sets the accompaniment file tempo according to the following expression:

Set tempo = **modelTempo** + (currentOffset-**modelOffset**)*(alpha).

Thus, the controller will first play the accompaniment file at a tempo that will correct the error in offset. As that error approaches 0, the tempo of the accompaniment playback will approach the modelled tempo of the input. Alpha was set to .5 for all tests, allowing for both fast convergence and infrequent overshoots.

2.3 Libraries Used

FunPlayer uses libraries and services to aid both the model and the controller. The Model uses Riffstation[5], an online chord analysis to tool, to obtain a mapping of times to chords to define likelihoods of certain note pitches at certain times. It also uses beat markings created by BeatRoot to determine the offset that lines up played notes best with the calculated beat of the song.

---

[5] "Play Riffstation." *Riffstation*. N.p., n.d. Web. 10 May 2016.

The Controller uses outside libraries to handle the actual modification of the audio stream. TarsosDSP is used as an interface for music processing. The specific process applied in the system is through the RubberBand JNI interface, which allows the controller to apply a time-stretch on audio samples without causing any change in pitch.

3 Benchmarks

This section evaluates the ability of FunPlayer to adjust to deviations between played notes and the expected chord progressions and timings generated by analysis of the original song.

The benchmarks focus solely on objective, repeatable metrics that measure how quickly the model FunPlayer uses is able to adjust to changes in played notes. Though the goal of the project is accompanying real-time music, the system was tested on a series of MIDI sequences that simulate a live performance.

The tested MIDI sequences were generated to exactly correspond to a known sequence of chords and durations. Then, different transformations were applied to that MIDI sequence to simulate performance idiosyncrasies, such as pauses, tempo changes, or note errors. An error was calculated as the difference between the model's estimation of song location and the transformed-back location of the MIDI sequence. All of the plots below were constructed by evaluating system response to this test MIDI sequence.

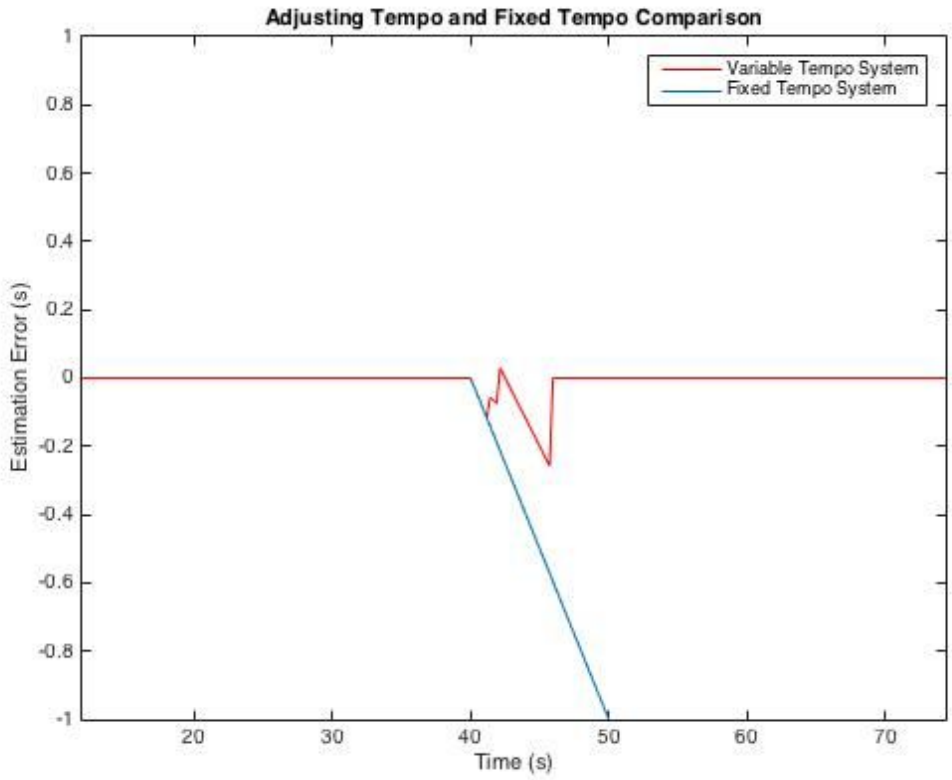
This error metric measures the difference in tempo or offset for two reasons. First,

it is easier to visualize the single time error value than the two dimensional <tempo,offset> vector. Second, the controller adjusts the playback of the original audio at a rate proportional to the error, so it is also a significant value with respect to the function of the system.

3.1 Comparison to Fixed Tempo Audio

As seen in plot 1, the system is able to quickly adjust to a change in tempo that would be a problem for a constant-speed accompaniment. Despite the input adjusting by an instantaneous tempo increase of 10% at 40 seconds in, the system was able to remain within 0.25s of the playback, and averaged an absolute error of 0.13s. Though it initially follows the trajectory of the fixed tempo system, at 41.1 seconds, a note from the next expected chord is played earlier than expected, changing the highest scoring tempo+offset model.

By 47.3 seconds, the weight of the initial 40 seconds of recorded notes has fallen off exponentially by enough that the notes following the faster tempo outweigh, and the system returns to an error of zero.

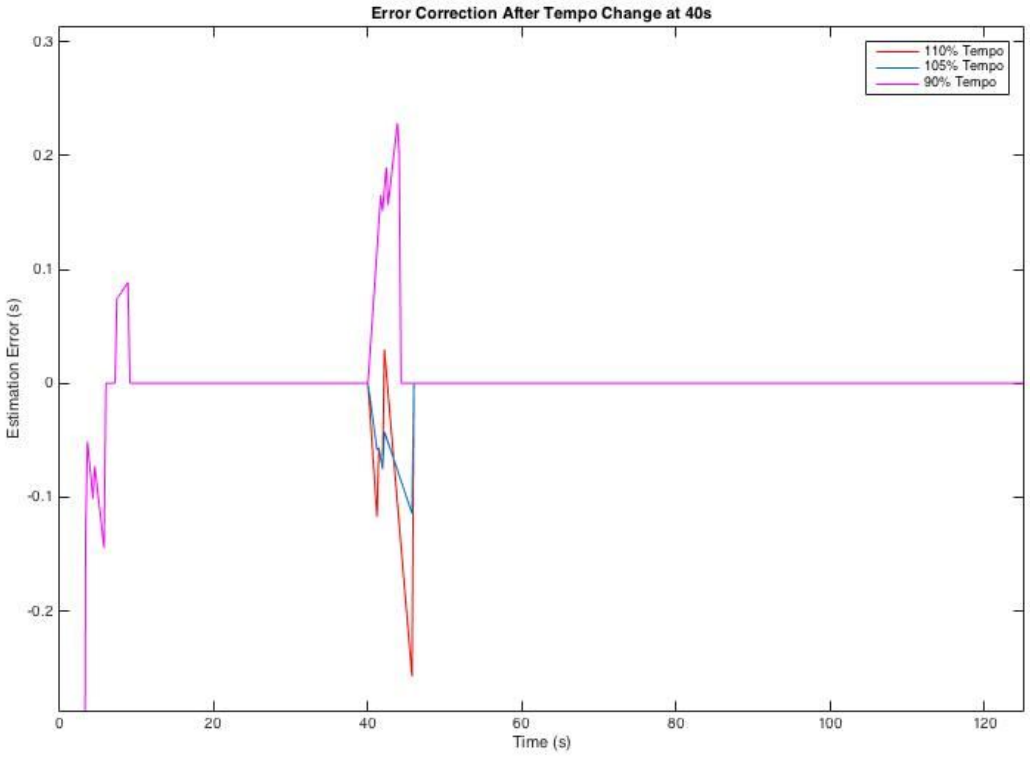


Plot 1: Though FunPlayer (red line) takes time to completely adjust to a change in tempo of 10%, it remains within a much lower error than an accompaniment that uses a constant tempo (blue line).

3.2 Latency of Tempo Adjustment

When changing tempo abruptly in the middle of a song, FunPlayer will take time to correct its modeled tempo to the new tempo. As shown in Plot 2, at instantaneous tempo shifts of up to 10%, the system was able to maintain an average error of about 0.1s, peaking at less 0.25 seconds.

Also worth noting is that, as one might expect, the smaller changes in tempo such as the 5% increase shown in the plot result in a lower error effect than the 10% increase.



Plot 2. Though FunPlayer takes up to 10 seconds to adjust to a change in tempo, average error during the adjustment time is relatively low at under 0.2s and not too noticeable audibly.

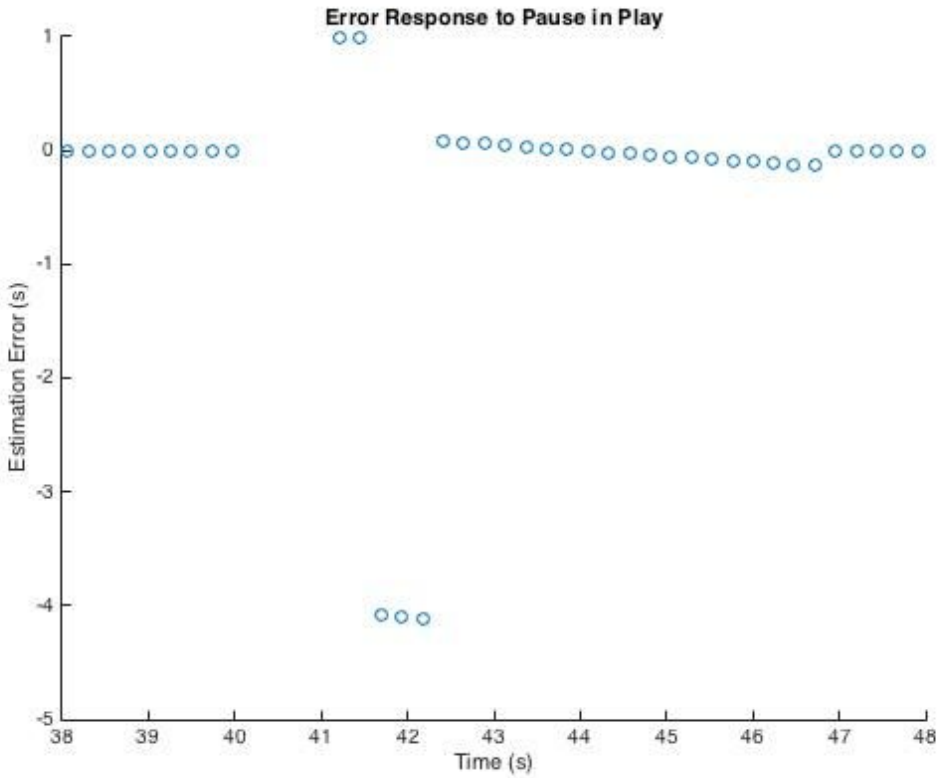
3.3 Detecting Pauses

Plot 3 below depicts the system's response to a pause in performance by the user. An example scenario where a pause like this may take place would be the user taking a second time turn the page on his sheet music, and then continuing with the piece at the same tempo.

The reason the maximum error is higher in this scenario than the previous is there is much more uncertainty during a pause. The system has no current way of determining whether a pause is due to a mistake on the performer's part, or if it's intentional waiting

through a solo section of another instrument.

The temporary error in pausing is so large compared to the speed-changing error for another reason. Sometimes performers will miss notes, but keep playing the rest of the song as normal. In this case, the tempo and offset should not change during the pause. Minimizing the error for these two different types of pauses is impossible; having a large temporary error for one of them is inevitable.



Plot 3: After a pause at 40 seconds, estimation error is relatively large for the next 5 notes played. Then, error decreases to within .1s until finally reaching 0.

4 Analysis and Future Work

As one might expect, actually performing a song does not correspond exactly with any of the simulated scenarios listed above. However, these scenarios give some insight to how well the system performs on an actual song and performance.

Overall, the system performed inconsistently depending on what song was chosen. Especially since the "best" accompaniment timing is subjective, it is difficult to know exactly what makes some songs work better than others; however, this section lists a few observations of areas that could be improved.

RiffStation gives the wrong chords for certain songs, or is not specific enough. Though the system is robust enough to handle a slightly incorrect analysis, occasionally RiffStation's chord timings would lack in several places throughout a song. An example of a common error was listing the same chord for 10 seconds, when actually 2 different chords were being alternated. Since the system relies on the differences in expected notes between chords, having chord segments so long limits the ability of the system to accurately track the performer's location in the song.

Access to better chord analysis tools would help solve this problem, but future work could also include using melody extraction to provide a second set of data points to compare against. This would also allow the system to handle songs that have infrequent chord changes or a lack of them entirely. Since the current version relies on chord changes to build its estimation model, it would need an additional feature like melody

comparison to function.

Another limitation of the system is it takes time at the start of the song to converge upon the correct model. Other similar systems rely on training the models on the same performer in order to improve accuracy, which would be especially noticeable at the beginning. A future iteration of this system could apply the same principles, or simply include an option to specify an estimated starting tempo to facilitate reaching a small error faster.

5 Conclusion

Overall, FunPlayer works quite well with songs that the chord analysis is accurate on. Most accompaniment software requires knowledge of every expected note a performer will play. However, FunPlayer has shown that, especially as music analysis technology improves, accompaniments are able to be programmed according to less definite models of note likelihood. Hopefully, FunPlayer can open the door for a greater variety of music to be made into an automatic accompaniment, and aid in the process of learning and enjoying music.

6 References

1. "Cadenza | FAQ," *Cadenza by Sonation,* 2016. http://www.sonacadenza.com/support/.
2. Raphael, Christopher, and Yupeng Gu. "ORCHESTRAL ACCOMPANIMENT FOR A REPRODUCING PIANO." (n.d.): n. pag. Web. 7 May 2016.
3. Arshia Cont. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music.. *International Computer Music Conference (ICMC)*, Aug 2008, Belfast, Ireland. pp.33-40, 2008. <hal-00694803>

4. Dixon, Simon. "Evaluation of the Audio Beat Tracking System BeatRoot." *Journal of New Music Research* 36.1 (2007): 39-50. *BeatRoot*. Web.
5. "Play Riffstation." *Riffstation*. N.p., n.d. Web. 10 May 2016.