

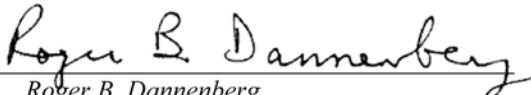
Computer Accompaniment System for Polyphonic Keyboard Performance

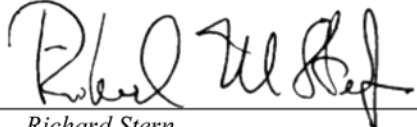
by Hao Huang

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Music and Technology
at the School of Music
Carnegie Mellon University

May 2017

Thesis Committee: Roger B. Dannenberg, Chair
Richard Stern

Approved by:  May 18, 2017
Roger B. Dannenberg *Date*
Department of Computer Science

Approved by:  May 18, 2017
Richard Stern *Date*
Department of Electrical and
Computer Engineering

Computer Accompaniment System for Polyphonic Keyboard Performance

Hao Huang

School of music
College of Fine Arts
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
Roger B. Dannenberg
Richard M. Stern

Abstract

Computer accompaniment systems expand the possibilities of musical expression in live performance and enable live performance by human performers on traditional instruments along with computer performers playing electronics. Given a score containing the solo and the accompaniment parts, real-time computer accompaniment systems match a live performance against the solo part and synchronize the accompaniment to the human performer. Computer accompaniment systems handle wrong notes, extra notes, skipped notes and unsteady tempos.

In general, a computer accompaniment system consists of an input preprocessor, a matcher, an accompanist, and a synthesizer. The input preprocessor can be a pitch detector or a keyboard scanner. A MIDI keyboard can extract totally correct information including the pitch, onset time, offset time, loudness, etc. Therefore, a MIDI-based accompaniment system can be more robust than systems that accept audio as input. Many algorithms have been presented for score following and automatic accompaniment, and many systems based on these algorithms have been constructed. However, most of these systems only accept audio input which is less reliable than MIDI, or only provide a function for score following without synchronizing the accompaniment, or only handle monophonic performance. Unlike monophonic matchers which only deal with one event at a time, polyphonic matchers need to deal with multiple events at the same time. Since there does not exist a reliable, simple and easy-to-use real-time computer accompaniment system for polyphonic keyboard performance, my thesis focuses on implementing a MIDI-based real-time computer accompaniment system for polyphonic keyboard performance.

This polyphonic computer accompaniment system is implemented in *Serpent* and is evaluated using two types of dataset. One uses synthetic performances; another uses recorded piano duet performances. The synthetic performance is used to evaluate how the system handles each type of note mistake or tempo change. The recorded duet performances contain reasonable mistakes and tempo changes that can happen in the real world, which is a good way to evaluate how the system accompanies a real human performance. The results show that this polyphonic accompaniment system can handle unsteady tempo and note mistakes for polyphonic keyboard performance. It successfully synchronizes with the solo performance and generates corresponding accompaniment.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Professor Roger Dannenberg, for his guidance, patience, and encouragement over the past two years. Without his persistent help this thesis would not have been possible.

I owe a very important debt to Professor Richard Stern for serving on my committee and giving insightful comments and suggestions. I am very grateful to Teaching Professor Riccardo Shultz for his valuable comments and warm encouragement.

I have greatly benefited from Gus Xia and Mutian Fu. Discussions with them have been illuminating. Special thanks to Dawen Liang for his enormous contribution to HCMP that forms the basis of my thesis project.

I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my life.

Finally, I would like to thank Cong, Jessica, Tuna, and Lucky, for being by my side for all these years.

Contents

1	Introduction	1
1.1	Different types of accompaniment	1
1.2	Usage of computer accompaniment	1
1.3	Components of a computer accompaniment system	2
1.4	Monophonic versus polyphonic.....	3
2	Related work.....	5
3	The algorithms	7
3.1	Monophonic matching	7
3.1.1	Introduction of monophonic matcher.....	7
3.1.2	A dynamic programming technique	9
3.1.3	Monophonic score following algorithm	11
3.1.4	Space and time saving	13
3.1.5	Jump ahead problem and the solution.....	14
3.2	Polyphonic matching	16
3.2.1	Introduction of polyphonic matcher	16
3.2.2	Computation of association rating	17
3.2.3	Reporting matches	17
3.2.4	The grouping algorithm	17
3.3	Time adjusting.....	20
3.3.1	Virtual time definition	20
3.3.2	Virtual time adjusting	20
3.3.3	Virtual clock time speed adjusting	21
4	Implementation	23
5	Evaluation	26
5.1	Dataset	26
5.1.1	Synthetic performance	26
5.1.2	Recorded piano duet performance	26
5.2	Metrics.....	27

5.3	Results	28
5.3.1	Synthetic performance	28
5.3.2	Recorded piano duet performance	38
5.4	Analysis.....	40
6	Conclusion.....	45
	Reference.....	47

1 Introduction

1.1 Different types of accompaniment

In the past, when sound could not be recorded, a human soloist could only be accompanied by human accompanists. In this type of accompaniment, musicians coordinate with each other to keep synchronized.

After Thomas Edison invented the first machine to record and play back sound and with the development of sound recording, a soloist could be accompanied by a pre-recorded accompaniment. Since the accompaniment is fixed, the human soloist has to pay attention to the tempo of the accompaniment and adjust the human performance if necessary, which can result in synchronization problems. Moreover, fixed accompaniments limit the possibilities of musical expression in live performance.

With the development of score following and automatic accompaniment, in real-time computer accompaniment, the computer has the ability to follow a human soloist and synchronize an appropriate accompaniment. This solves most of the problems of synchronization in pre-recorded accompaniment and expands the possibilities of musical expression in live performance.

1.2 Usage of computer accompaniment

In modern musical thinking, the traditional roles of “soloist” and “accompanist” are now considered to form a “collaborative performance,” acknowledging the significant contributions of each performer. The terms “solo” and “accompaniment” imply foreground and background, leader and follower, which are roles that may not strictly apply. Nevertheless, from a technical or engineering standpoint, it helps to have terms for the human-generated part that forms the input to a score-following system, and the computer-generated output that constitutes the computer performance. We use the terms “solo” and “accompaniment” but wish to emphasize that collaborative performance is the goal.

Given the score containing the solo part and the corresponding accompaniment part, real-time computer accompaniment systems match a live performance against the solo score and synchronize the accompaniment to the human performer. A human soloist may make mistakes and notes may be incorrectly detected, so it is important that computer accompaniment systems handle wrong notes, extra notes, skipped notes and unsteady tempos.

Real-time computer accompaniment systems provide more possibilities for computer music composers and performers. Many composers write for live performance and for live performers. Real-time computer accompaniment enables live performance of human performers on traditional instruments with computer performers playing electronics. The composer can use computer accompaniment to enable the expression of a lot of new musical ideas such as impossibly complex rhythms.

1.3 Components of a computer accompaniment system

In general, a computer accompaniment system consists of four important parts [2]: the input preprocessor, the matcher, the accompanist and the synthesizer. See Figure 1-1. The input preprocessor detects the input from the live performer and extracts the information about the human performance, such as time and pitch. The derived information is sent to the matcher, which matches the preprocessed input against a score which contains every expected event and the expected time of each event. The score is a machine-readable description of the correct performance. The matcher reports the match status, usually including timing and location information, to the accompanist. The accompanist estimates tempo, adjusts score position, decides how to smoothly synchronize with the human performance and send commands to the synthesizer to generate the accompaniment.

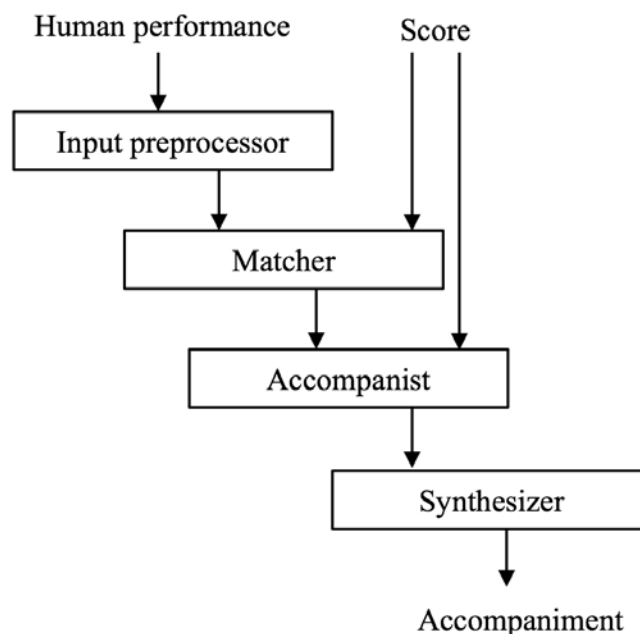


Figure 1-1: Components of a computer accompaniment system

1.4 Monophonic versus polyphonic

A piano note onset and a pitch played by the viola, for example, are both events that a computer accompaniment system can detect. The real-time performance can be regarded as a sequence of events. The score can be regarded as a sequence of ordered expected events. The task of a matcher is to compare the events between the human performance and the score.

Monophonic matchers deal with a totally ordered sequence of events. In this case, only one event occurs at a time. Single-voice instruments are monophonic, such as flute, trumpet, etc. If multiple events can occur simultaneously, for example, a chord played on the keyboard, a monophonic matcher will not be able to track the performance and match the performance events with the score because events are partially ordered. A partially ordered set is a set with a partial order. A relation " \leq " is a partial order if for all a, b, c in the set, (1) $a \leq a$, (2) $a \leq b$ and $b \leq a$ implies $a = b$, (3) $a \leq b$ and $b \leq c$ implies $a \leq c$. In a partial order, it is not required that ($a \leq b$ or $b \leq a$).

In practice, input is usually serialized (e.g., MIDI note-on events arrive sequentially). Figure 1-2 illustrates an example of three chords and all the possible ways that events of each chord can be serialized and still obey the partial orders implied by the score. The first chord is C E G. But when a human performer plays this chord on the keyboard, the key-down times of the three notes may be different. As long as the notes are played at nearly the same time, any order is correct. The performance events of this chord can be C E G, C G E, E C G, E G C, G C E, G E C. But no matter what the order of notes within a chord is, the order of chords is fixed. The number of all the possible serial orderings of the performance containing these three chords is $6 * 6 * 2 = 72$.



CEG	DFB	EG
CGE	DBF	GE
ECG	FDB	
EGC	FBD	
GCE	BDF	
GEC	BFD	

Figure 1-2: An example of chords and all the possible ways that events of each chord can be serialized

Where there are chords, the order in which chord notes are produced and/or detected can vary from one performance to the next, thus the assumptions of a strictly ordered sequence of events as in monophonic matchers is not applicable to polyphonic input.

2 Related work

In 1984, Dannenberg [1] presented an on-line algorithm to match monophonic performances against a stored score. This efficient dynamic programming algorithm finds the (heuristically) best match between monophonic human performance and the score, allowing for mistakes in the performance. The notion of virtual time was proposed at the same time which could be applied to synchronize the accompaniment according to the real-time human performance. A computer accompaniment system was constructed based on the monophonic matching algorithm. This system allowed the user to perform on trumpet and listen to a synthesized accompaniment. The algorithm could reliably follow and accompany the human performance even with wrong notes, extra notes, skipped notes and unsteady tempo. However, this computer accompaniment system could only deal with totally ordered events. The monophonic matching algorithm would not work for events that occur simultaneously.

In 1985, Bloch and Dannenberg [2] designed a set of algorithms to match a polyphonic performance against a stored score. Some accompaniment techniques were presented to improve the musicality of the accompaniment. The polyphonic computer accompaniment systems were successfully constructed using a piano-like keyboard as the input device. Like the monophonic accompaniment system, polyphonic accompaniment systems could also follow and accompany the real-time human performance with a stored score, and handle changes in tempo, wrong notes, extra notes, and skipped notes.

Baird, B., Blevins, D., and Zahler, N. [3] implemented an artificially intelligent computer performer, a MIDI-based polyphonic accompanist, to track a live performance using pattern-matching and generate an appropriate response. But it is not available to run on contemporary computer systems. Grubb, L. [4] presented a stochastic model for score following and implemented a robust system for tracking vocal performances and automatically accompanying a singer given a musical score. Müller [5] presented an algorithm for non-real-time automatic score-to-audio synchronization and solved the synchronization problem accurately and efficiently for complex but non-real-time polyphonic piano music. Raphael [6] used a graphical model containing discrete variables corresponding to score positions to establish the alignment between a sampled audio performance and a polyphonic score. Dixon [7] presented an on-line time warping algorithm and applied the algorithm to the alignment of audio signals to follow a performance of arbitrary length. Antescofo [8], a score following and synchronous programming system for real-time computer music, is frequently used in computer accompaniment applications. Antescofo accepts audio input that is either monophonic or polyphonic. But it is complex and optimized for audio input which is less reliable than MIDI.

There is a score-following object in MAX/MSP that accepts MIDI input, but composers complain that it is unreliable, and it is designed for score-following only, not synchronizing an accompaniment, which requires tempo estimation and some musical decision making.

The monophonic [1] and polyphonic [2] matching algorithms are both fast methods to follow the human performance. The matching result is available at the current performance event without the need for knowing future performance events. Unfortunately, the original implementations of these polyphonic matching algorithms cannot run on contemporary computer systems. Gus Xia implemented the monophonic matching algorithm in HCMP [9]. HCMP is a framework for conducting and tempo control. Based on Gus' work, polyphonic matching algorithms can be implemented in HCMP.

There does not exist a reliable, simple and easy-to-use real-time computer accompaniment system for polyphonic keyboard performance. When MIDI is available from a keyboard or other controller, a MIDI-based accompaniment system can be extremely robust and should be of great interest to composers of interactive computer music. My thesis focuses on implementing a MIDI-based real-time computer accompaniment system for polyphonic keyboard performance to fill this gap.

3 The algorithms

This section describes the algorithms that can be applied to implement the computer accompaniment system for polyphonic keyboard performance. The monophonic and polyphonic matching algorithms are used for the monophonic and polyphonic matcher to follow the score according to real-time human performance. The time adjusting algorithm is used for the accompanist to synchronize the accompaniment

3.1 Monophonic matching

Dannenberg [1] presented a score following algorithm to track the score position of a monophonic real-time performance. The monophonic matching algorithm is the basis for the polyphonic matching algorithm.

3.1.1 Introduction of monophonic matcher

The key point of the monophonic matching algorithm is to find the best match between the human performance and the score efficiently in real-time. In order to translate this key point to a computer-friendly statement, two questions need to be answered. The first question is how a computer stores the human performance and the score. The second question is what makes a match better than another match, that is, what kind of match is the best match. To answer the first question, the human performance and the score can both be regarded as a sequence of events which can be stored as an array with notes as elements in a computer. The second question has more than one answer. Dannenberg defined the best match as “the longest common subsequence” of two sequences. A subsequence Z of sequence X is defined by $z_k = x_{n_k}$, where $n_1 < n_2 < \dots < n_k$. That is to say, if we delete some elements from a sequence and at the same time keep the order of the remaining elements, we get a subsequence of that sequence. The longest common subsequence gives the most matched totally ordered events between the human performance and the score. In the context of real-time computer accompaniment, the human performer keeps playing over time, so that more and more events keep appending to the end of the sequence of human performance events. Therefore, this algorithm solves the problem of finding the longest common subsequence of two sequences stored in dynamic arrays. Figure 3-1 illustrates the discussion about the key point of the monophonic matching algorithm. Figure 3-2 demonstrates how to get the longest common subsequence of sequences of the human performance and the score. In this example, the F in the score does not have a corresponding F in the human performance. This F is a skipped note and cannot be counted as a match.

The second C in the human performance does not have a corresponding C in the score. This C is an extra note and cannot be counted as a match either.

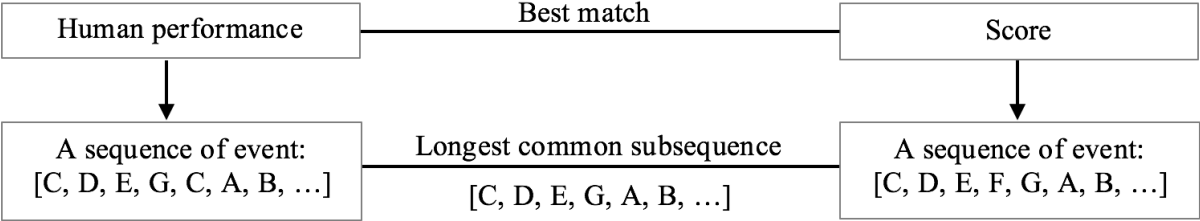


Figure 3-1: Key point of the monophonic matching algorithm

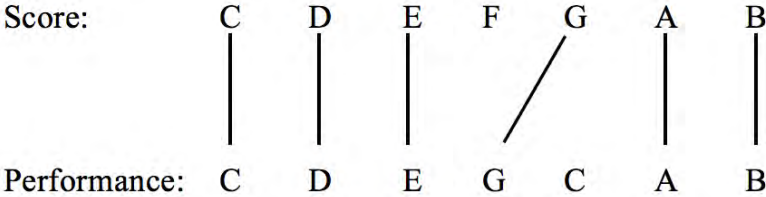


Figure 3-2: How to get the longest common subsequence of sequences of the human performance and the score

The input of the monophonic matcher is a new event in the real-time human performance. This new event will be appended to the dynamic array of the performance sequence. The monophonic matcher runs the powerful score following algorithm which determines if the new performance event is a new match to some event in the stored score. Figure 3-3 illustrate the input and output of a monophonic matcher. Each time a new performance event is detected as input, the monophonic matcher outputs the location information if there is a new match.

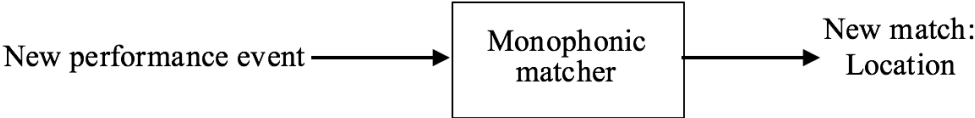


Figure 3-3: The input and output of the monophonic matcher

3.1.2 A dynamic programming technique

The monophonic matcher stores a matrix which indicates the length of the best match. Each time a new performance event is detected, this matrix will be updated. Figure 3-4 illustrate an example of the best-match matrix. Each row represents a score event, while each column represents a performance event. Each cell represents the length of the best match. Specifically, the cell at the r th row and the c th column represents the length of the best match at the r th score event and the c th performance event. At the time when the matrix is initialized, we already have complete knowledge of the score events, however, we do not know any event of the human performance before the performance starts and any performance event is detected. Therefore, the best-match matrix can be initialized as an N by M matrix where N denotes the total number of score events and M denotes the total number of performance events. Each time a new performance event is detected, a new column will be computed and updated.

In Figure 3-4, assume the row index and the column index both start from 1 instead of 0. Row 0 and column 0 are ignored for now but they will be used in the actual implementation of the algorithm, which will be discussed later. When the first performance event C is detected, the first column will be updated from the first row. The cell at [1, 1] should be updated by computing the longest common subsequence of the human performance sequence [C] and the score sequence [C]. This cell is set to be 1 because the new performance event C matches the first score event C which means the current longest common subsequence is [C] whose length is 1. The cell at [2, 1] should be updated by computing the longest common subsequence of the human performance sequence [C] and the score sequence [C, D]. This cell is also set to be 1 because the longest common subsequence is still [C]. Similarly, all the remaining cells in the first column are set to 1. When the second performance event D is detected, the second column will be updated. The cell at [1, 2] should be updated to 1 because the longest common subsequence of the human performance sequence [C, D] and the score sequence [C] is [C] whose length is 1. The cell at [2, 2] should be updated to 2 because the longest common subsequence of the human performance sequence [C, D] and the score sequence [C, D] is [C, D] whose length is 2. All the remaining cells in the second column are set to be 2 because their longest common subsequences are [C, D]. The third column can be updated with a similar analysis when the third performance event is detected. Notice that the fourth performance event G makes a little different when the new column is updated. As with the previous columns, the first three cells in the fourth column can be updated to 1, 2, 3. For the cell at [4, 4], the longest common subsequence is the same as the one for the cell at [3, 4], which is [C, D, E]. So, the cell at [4, 4] is set to 3. For the cell at [5, 4], the longest common subsequence of the human performance sequence [C, D, E, G] and the score sequence [C, D, E, F, G] is [C, D, E, G]. The fourth score event F is

skipped by the human performer. So, the cell at [5, 4] is set to be 4, as are remaining cells in the fourth column. The fifth performance event C is an extra event. Each cell in the fifth column has the same length of the longest common subsequence as the cell in the same row of the previous column has. Therefore, the fourth column and fifth column share the exactly same best match status. Filling the columns for the next two performance events A and B is similar to filling in the first three columns.

		Performance:						
		C	D	E	G	C	A	B
Score:	C	1	1	1	1	1	1	1
	D	1	2	2	2	2	2	2
	E	1	2	3	3	3	3	3
	F	1	2	3	3	3	3	3
	G	1	2	3	4	4	4	4
	A	1	2	3	4	4	5	5
	B	1	2	3	4	4	5	6

Figure 3-4: The structure of the best-match matrix

Actually, it is not necessary to update the cells by computing the longest common subsequence of the two whole sequences. (And in fact, we have not even shown how to do this yet.) This is the definition of the best match, but any algorithm that looks at all previous events at each matrix location will be expensive to compute. By observing the values in Figure 3-4 and analyzing the process to compute each cell, the computation of the matrix update can be simplified into a dynamic programming problem. Each cell can be computed by considering: (1) the comparison of the current performance event and the current score event, (2) the best match rating up to the previous performance event and the current score event (the cell in the same row of the previous column), (3) the best match rating up to the current performance event and the previous score event (the cell in the same column of the previous row), (4) the best match rating up to the previous performance event and the previous score event (the cell in the previous row of the previous column). Figure 3-5 illustrates the relationship of the cells involved in solving this dynamic programming problem. c and $c - 1$ represent the current and previous human performance event. r and $r - 1$ represent the current and previous score event. Let the best match rating up to the performance event c and the score event r be $rating[r, c]$. Given $rating[r - 1, c - 1]$, $rating[r - 1, c]$ and $rating[r, c - 1]$, the current best match $rating[r, c]$ can be computed by considering: (1) $rating[r, c] \geq rating[r - 1, c]$ because the longest common subsequence will not be shorter with one more score event, similarly (2)

$rating[r, c] \geq rating[r, c - 1]$ because the longest common subsequence will not be shorter with one more performance event, (3) if the human performance event c matches the score event r , $rating[r, c] = rating[r - 1, c - 1] + 1$ because the new match adds one to the longest common subsequence of the previous human performance sequence and the previous score sequence.

	Performance:		
		$c - 1$	c
Score:	$r - 1$		
	r		

Figure 3-5: The cells used to compute the best match rating

3.1.3 Monophonic score following algorithm

Figure 3-6 illustrate the computation of the best match matrix in monophonic score following algorithm. The best match matrix, $rating$, is initialized first. We set the index of the first performance event and score event to 1, because row 0 and column 0 are expected to be used here for the convenience of setting initial conditions to make the initialization process cleaner. After the initialization, the update rule is realized by two nested for loops. Each time a new performance event is detected, loop through each score event to compute the best match rating.

```

for all  $i$ 
   $rating[0, i] \leftarrow 0$ 
   $rating[i, 0] \leftarrow 0$ 
for performance event  $perf\_arr[c]$  in  $perf\_arr$ 
  for score event  $score\_arr[r]$  in  $score\_arr$ 
     $rating[r, c] \leftarrow \max(rating[r, c - 1], rating[r - 1, c])$ 
    if performance  $perf\_arr[c]$  matches  $score\_arr[r]$ 
       $rating[r, c] \leftarrow \max(rating[r, c], 1 + rating[r - 1, c - 1])$ 

```

Figure 3-6: Computation of the best match rating matrix
in the monophonic matching algorithm

For now, the monophonic matcher and its input are described. The next question is about the output of the matcher, which is also the input of the accompanist. How should we determine if the new performance event should be counted as a match and how should we decide the matched position in the score corresponding to the detected performance event? The largest value in the matrix is the largest best match

rating. Each time a performance event is detected, the matrix is updated. If the updated matrix contains a larger largest value than the largest value in the previous matrix, we assume this performance event matches some event in the score. The corresponding match position in the score is the row that this new largest value is in. The match positions are underlined in Figure 3-7. In this example, at the fourth performance event G, a larger value is achieved, but the row of the larger value is five instead of four, which means this performance event matches the fifth score event skipping the fourth score event. At the fifth performance event C, the largest value does not change, which means no match happens because it is a wrong performance event. At the sixth performance event A, a larger value is achieved, and the row of the larger value is six, which means this performance event matches the sixth score event.

		Performance:						
		C	D	E	G	C	A	B
Score:	C	<u>1</u>	1	1	1	1	1	1
	D	1	<u>2</u>	2	2	2	2	2
	E	1	2	<u>3</u>	3	3	3	3
	F	1	2	3	3	3	3	3
	G	1	2	3	<u>4</u>	4	4	4
	A	1	2	3	4	4	<u>5</u>	5
	B	1	2	3	4	4	5	<u>6</u>

Figure 3-7: Match positions based on the best match rating matrix

It should be noted that this part of the algorithm is a good heuristic, but it may report matches that later turn out not to be in the longest common subsequence. Figure 3-8 illustrates such a situation. In this example, the fifth score event G is matched when the fourth performance event G is played. At the fifth performance event F, the largest value does not change and it is still 4, however, the first 4 occurs on the row of the fourth score event F. At the sixth performance event G, a larger value is achieved and this performance event matches the fifth score event G. Although the fourth performance event G is reported to match the fifth score event G at first, this match turns out not to be in the longest common subsequence when all performance events are considered.

		Performance:						
		C	D	E	G	F	G	B
Score:	C	<u>1</u>	1	1	1	1	1	1
	D	1	<u>2</u>	2	2	2	2	2
	E	1	2	<u>3</u>	3	3	3	3
	F	1	2	3	3	<u>4</u>	4	4
	G	1	2	3	<u>4</u>	4	<u>5</u>	5
	A	1	2	3	4	4	5	5
	B	1	2	3	4	4	5	<u>6</u>

Figure 3-8: An example that reports matches that later turn out not to be in the longest common subsequence

3.1.4 Space and time saving

This algorithm may use much space because the best match matrix with N by M size can be very large for a long piece. The purpose of the matcher is to provide the location information of each match to the accompanist. According to the algorithm, we get the location information from the row index of the new largest value. In order to get the new largest value, we just need to update the matrix each time a new performance event is detected. With the dynamic programming technique, the update process only involves two columns: the current one and the previous one. Therefore, only two columns need to be stored at a time instead of storing the entire matrix all the time.

To make the algorithm faster, one method is to only consider part of the column instead of the complete column, so that less computation is needed. This method is workable if the human performer keeps within a reasonable distance from the expected match position. A window can be applied to each column to denote the range that the algorithm is going to search. If the most tolerable consecutive error is l_e , the size of the window should be no less than $2l_e + 1$. The center of the window should be around the expected match position to reduce the possibility that a matcher loses the tracking. Figure 3-9 illustrate the computation of the best match matrix using windows with size 3. The rule of moving the center of the window: (1) if the previous column has a new match, move the center to the row below the match, (2) if the previous column does not have a new match, move the center to the row below the current center. Using windows saves time and space. In practice, window sizes of 20 or so are very fast to compute and allow quite a few mistakes without getting lost.

		Performance:						
		C	D	E	G	C	A	B
Score:	C	<u>1</u>	1					
	D	1	<u>2</u>	2				
	E	1	2	<u>3</u>	3			
	F			3	3			
	G				<u>4</u>	4		
	A					4	<u>5</u>	5
	B					4	5	<u>6</u>
	C						5	<u>6</u>

Figure 3-9: Computation of the best match matrix using a window

3.1.5 Jump ahead problem and the solution

There is a problem hidden in the monophonic score following algorithm describe above. This problem may happen when an extra or wrong performance event is detected and does not match the next score event. Under such a circumstance, if there is a same score event in the future, this extra or wrong event will jump ahead and be matched to that score event in the future. This is the “jump ahead” problem. Figure 3-10 illustrates an example of such a jump ahead problem. The first performance event is wrong and the second performance event is correct. According to the rating matrix, the performance event F matches the fourth score event. The performance event D ignores the seemingly correct D at the second score event, and instead, it matches the sixth score event D.

		Performance:	
		F	D
Score:	C	0	0
	D	0	1
	E	0	1
	F	<u>1</u>	1
	C	1	1
	D	1	<u>2</u>

Figure 3-10: An example of jump ahead problem

This algorithm is very robust and is able to recover from the incorrect tracking as long as there are enough correct performance events following. Figure 3-11 illustrate an example of recovery from the jump ahead problem. The matcher finds the correct score match position at the fourth performance event F. However, this automatic recovery is not a good solution for the jump ahead problem for two reasons. First, unlike the example in Figure 3-11, in the real world, the score can be very long, and it is highly possible that there are many matching score events in the future which will further delay the recovery process. Second, the computer accompaniment cannot be musically pleasing if it follows the matcher and makes large jumps in the accompaniment.

		Performance:			
		F	D	E	F
Score:	C	0	0	0	0
	D	0	1	1	1
	E	0	1	2	2
	F	<u>1</u>	1	2	<u>3</u>
	C	1	1	2	3
	D	1	<u>2</u>	2	3

Figure 3-11: An example of recovery from the jump ahead

Three remedies are presented to solve this problem. First, set a maximum jump length for the center of the window, such as 2. Second, deduct one penalty from the rating of every performance-score event pair that does not match. Third, if the same largest number occurs more than once in a column, select the top one (lowest row number) as the match position.

3.2 Polyphonic matching

The monophonic matcher is not applicable to polyphonic input. In 1985, Bloch and Dannenberg [2] presented polyphonic score following algorithms to track the score position of a polyphonic real-time performance.

3.2.1 Introduction of polyphonic matcher

Like a monophonic matcher, a polyphonic matcher receives real-time performance events as input and reports the new match to the accompanist. Unlike a monophonic matcher which only deals with a totally ordered sequence of events, a polyphonic matcher needs to deal with a partially ordered sequence of events. In a monophonic matcher, only one note event is played at a time. In a polyphonic matcher, multiple note events can occur simultaneously.

Figure 3-12 illustrates an association between a polyphonic performance and the score. The best association is represented by lines. The polyphonic score can be described as a sequence of sets of symbols. Symbols in the same set are represented in the same pair of parentheses. The order within a set is not important, but the polyphonic matcher must consider every possible ordering of each set. In a monophonic performance and score, the best match lines cannot be crossed, while in polyphonic performance and score, symbols within the same set can have crossed association lines.

The symbols in the performance and the solo score can be classified into four categories: extra, missing, wrong, and right. A symbol in the performance is an extra symbol if it does not have corresponding symbol in the score. A symbol in the score is a missing symbol if it does not have corresponding symbol in the performance. A symbol in the performance is wrong if it does not match the corresponding symbol in the score. A symbol in the performance is right if it matches the corresponding symbol in the score. In Figure 3-12, the score event F is a missing symbol, and the performance event C is an extra symbol.

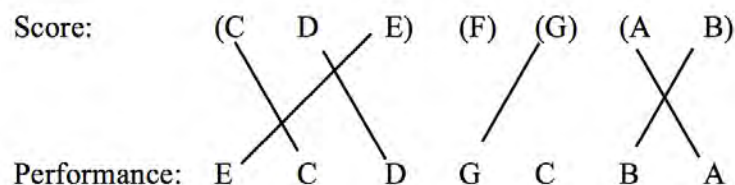


Figure 3-12: An association between a polyphonic performance and the score

3.2.2 Computation of association rating

The best association between the performance and the score is defined to be the association with the maximum value of this rating function:

$$\text{len}(\text{score prefix}) - (c_w * \# \text{ of wrong notes} + c_m * \# \text{ of missing notes} + c_e \# \text{ of extra notes})$$

where c_* means cost coefficient.

Given a performance prefix, the rating function determines the best score prefix. Let p be the sequence of performance events, and s be the sequence of score events. Let $p[i]$ be the i th event in the performance, and $s[j]$ be the j th event in the score. $p[1:i]$ represents the first i events in the performance. $s[1:j]$ represents the first j events in the score. Given i , if some value of j makes $\text{rating}(p[1:i], s[1:j])$ maximum, then $s[1:j]$ is the best match for $p[1:i]$. This maximum value is the best match rating. If more than one values of j achieve the maximum rating, select the smallest j to break the tie.

3.2.3 Reporting matches

The matcher should report a match to the accompanist when there is enough confidence in the correction of the match because reporting wrong match locations can mislead the accompanist and have bad effects on the synchronization. The approach to measure the confidence is similar with how the monophonic matcher decides to report the match to the accompanist. The assumption is also related with one of the solutions for the jump head problem described in the monophonic matching algorithm. It assumes that the best match rating will generally increase with more and more performance events are detected, but it will decrease if errors occur. The location is reported when the current best match rating is higher than any previously best match rating.

3.2.4 The grouping algorithm

The monophonic score is totally order. The polyphonic score is partially ordered. However, on the level of chords, chords are totally ordered. (This is also referred to as a level graph.) If the polyphonic matching problem can be reduced to monophonic matching, the monophonic matching methods can be applied.

Each polyphonic score can be broken into compound events whose note-on events happen simultaneously. A compound event can contain one or more notes. However, a performance cannot be split into compound events by using the same onset time because of two reasons. First, it cannot be guaranteed that

all the notes in a chord will be pressed simultaneously when a chord is played on a keyboard. Second, most of the keyboard input systems serialize the notes in a chord, making one note slightly after the other. Even in a slow system, the time between notes in a chord should be no more than 90ms. Consider a reasonable upper inter-onset time for playing sequential notes, 16th notes at 120 bpm, which means 125ms between notes. Therefore, to determine if a newly detected event is in a compound event with the previous event, compute the difference of their note-on times. If the time difference is more than 125ms, it is highly possible that they are not in the same compound events. If the time difference is less than 90ms, it is highly possible that they are in the same compound events. However, this criterion does not work well for rolled chords. The time between notes in a rolled chord could be more than 125ms. Therefore, assigning a certain value as the threshold to split the score into compound events can introduce more errors if there are many rolled chords in the performance. Notice that a note in a rolled chord is much closer to the previous note in the same chord than to the first note of the next chord. If the time between two note-on events is less than $\frac{1}{4}$ of the predicted time from the first event to the next compound events, these two note-on events should be in one group. For each note in the score, we group the newly detected performance event into the compound event that the previous note is in, or treat it as a new compound event using the above methods.

In the monophonic matching algorithm, the best-match function is computed each time a new solo performance event is detected. In the polyphonic matching algorithm, when a new solo event is detected and if this event is not in the same group with the previous detected event, treat this event as a compound event and compute a tentative best match. Then, the tentative best match of the previous compound event can be regarded as a correct best match. When a new solo event is detected and if this event is in the same group with the previous detected event, update the tentative best match of this compound events. In order to let the accompanist have the most updated information, report the tentative best match to the accompanist each time a new event is detected.

It is clear and direct for a monophonic matcher to decide if the performance events match with the score events. However, the polyphonic matcher needs to compare a perhaps incomplete compound events of the performance to the score compound events to decide if they match. The intuition is that if more performance events are in the score, then the rating for the match will increase. If more performance events are not in the score, then the rating for the match will decrease. A function is designed to measure if two compound events match. If the rating is greater than or equal to 0.5, then these two set of compound events match, otherwise, they do not match.

$(\# \text{ of performed events in score compound events} -$
 $\# \text{ of performed events not in score compound events}) / \# \text{ of performed events}$

For example, for performance compound events (C, A, E, F) and score compound events (C, D, E, F):

$\# \text{ of performed events in score compound events} = 3$

$\# \text{ of performed events not in score compound events} = 1$

$\# \text{ of performed events} = 4$

$rating = (3 - 1) / 4 = 0.5$ which means these two compound events match

3.3 Time adjusting

After the match is reported by the matcher to the accompanist, the next problem is when to perform the following accompaniment events. The goal is to mimic what a human musician would do under similar circumstances.

3.3.1 Virtual time definition

Virtual time was presented [1] as a solution to how to adjust the tempo according to the solo performer. The definition of the virtual time is:

$$(R - R_{\text{ref}}) * S + V_{\text{ref}}$$

R is the real time that cannot be changed or reset. S is the speed of the virtual clock. R_{ref} and V_{ref} are the real time and the virtual time when the virtual clock was last set. S can be set to change the speed of virtual time. Setting V_{ref} and assigning the current real time to R_{ref} can set the virtual clock. Each solo score event has a virtual time. Therefore, each time the matcher reports a new match to the accompanist, the virtual clock should be reset to allow the accompanist to speed up or slow down to synchronize with the solo performance.

When the matcher reports the new match to the accompanist, the accompanist receives three pieces of information: (1) the virtual time of the matched performance event, (2) if the matched event is the next event in the sequence after the last matched solo event, (3) the virtual time of the next solo event in the score.

Virtual time adjusting and virtual clock time speed adjusting respectively change the position and speed of the virtual clock. For each new match, the accompanist calls virtual time adjusting first to change the position of the current note that is playing, then calls virtual clock time speed adjusting to change the accompaniment tempo to synchronize with the solo performance.

3.3.2 Virtual time adjusting

When virtual time adjusting is invoked, it checks the time difference between the matched solo event and current virtual time. If the time difference is less than a certain small value (e.g., 100ms), it can be regarded as the subtleties introduced by the performer and there is no need to adjust the virtual time. If the

time difference between the matched solo event and current virtual time is more than the certain small value, virtual time should be adjusted.

If the matched event is the next event in the sequence after the last matched solo event, we can know that only tempo changes and there is no note mistake. We have two choices here: (1) jump to the new score position, (2) catch up or wait. If the solo performer skipped a line by accident, the first choice is better because catching up will take a long time to catch up and may sound unpleasant. If the matched event is behind or ahead for a reasonable time, the second choice is more suitable because catching up or waiting reduce the possibility of skipping events or playing twice. We need a threshold to decide the choice. It can be a few seconds, or less if there are too many notes in the accompaniment score. If the matched event is not the next event in the sequence after the last matched solo event, it should be synchronized with immediately because all the previous knowledge about position can be incorrect.

3.3.3 Virtual clock time speed adjusting

The task of the virtual clock time speed adjusting is to adjust the tempo. The time of the matched solo event is the current virtual time. Speed S should be set to the slope of the virtual-time vs. real-time graph. In real performances, local speed variations occur frequently global tempo tends to be fairly steady. The solution for smooth synchronization is to keep the last few match points and do some linear regression.

If the matched event is the next event in the sequence after the last matched solo event, we assume the performance is going well and matching the score, and therefore this new note is a good source of tempo information. The newly matched point is appended to an array. If the array is full, remove the oldest point and keep the new one. If the array is full or the real time difference between the first and last point in the array is greater than some certain value, for example, 1s, compute the linear regression of these points and assign the slope to speed S . If the real time of the newly matched point is too close to the previous point in the array, for example, 200ms, this matched point should not be appended into the array. This is because when events are too close, musicians tend to perform them as a whole instead of considering about the tempo.

If the matched event is not the next event in the sequence after the last matched solo event, we assume something is wrong with the performance and timing is not reliable. Clear the table and append this newly matched point.

One potential problem in the virtual clock time speed adjusting is that if the tempo at the beginning of the performance is very different from the initial tempo of the accompaniment, the first few events can be very unsynchronized because the accompanist needs time to figure out the tempo of the solo performer. The solution is to assign an initial speed to S in advance to reduce the difference between the speed of the solo performance and the accompaniment.

When no match is reported, including when the performer stops playing, the accompaniment will keep going at the latest computed tempo.

4 Implementation

I use Serpent as the programming language to implement a system for real-time computer accompaniment of polyphonic keyboard performance. Gus Xia implemented the monophonic computer accompaniment system in HCMP, a framework for conducting and tempo control. I implemented the polyphonic computer accompaniment system based on the monophonic one.

Figure 4-1 illustrates the structure of the polyphonic computer accompaniment system. The Music folder and the Source Code folder are in the same directory. The Music folder contains folders of pieces. In each folder of the piece, there are three folders respectively containing the solo score, the accompaniment score, and the test performance in standard MIDI files. See the solid arrows in this figure. The command to start the system is `wxserpent64` which calls `init.srp`. The only task for `init.srp` is to load `interface.srp`. `interface.srp` is responsible for initiating the scheduler, preparing the interface, opening midi devices, instantiating the conductor, the `midi_player`, and the `score_follower`. Conductor is responsible for tempo control. `Midi_player` is responsible for scheduling the accompaniment. `Score_follower` keeps the best match matrix and reports the match status to the accompanist. The accompanist decides how to change the tempo and asks the conductor to finish this task. After that, `midi_player` schedules the next accompaniment event. The synthesizer plays the accompaniment.

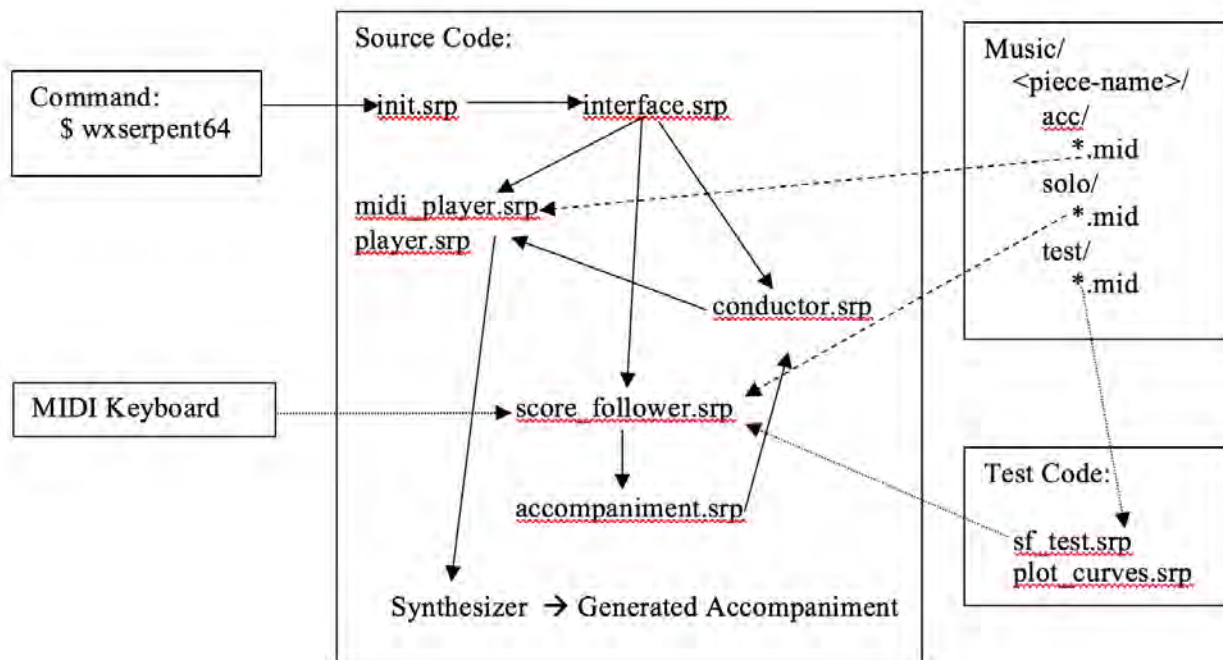


Figure 4-1: The structure of the polyphonic computer accompaniment system

The dashed arrows in Figure 4-1 illustrate that the score for the accompaniment is read by `midi_player` and the score for the solo performance is read by `score_follower`. The dotted arrows illustrate how the real-time performance is sent to `score_follower`. The system has two modes: user mode and test mode. The user mode is for users to perform using a MIDI keyboard and listen to the synchronized accompaniment. The test mode is for the developer to test and evaluate. When in user mode, the input is from a MIDI keyboard and there is a `midi_handler` which deals with different midi messages. When in test mode, the test performance is read by `sf_test.srp` and sent to `score_follower`.

Figure 4-2 shows the user interface of this system under the user mode. A user can select the piece, decide the initial accompaniment tempo, choose whether ignore octave, then start to play on the MIDI keyboard. Ignore Octave can be a useful function especially when using a MIDI keyboard as the input. Many people may choose to have a 25-key, 37-key or 49-key MIDI keyboard instead of an 88-key MIDI keyboard out of consideration for budget or storage space. As a result, the pitch range may be limited. If Ignore Octave is checked, all the pitches of events in the performance and score will be processed using $(pitch \% 12)$, where *pitch* ranges from 0 to 127. The processed pitch ranges from 0 to 11. If a user stops playing in the middle of a piece, the accompaniment will keep going at the latest computed tempo. Therefore, if the user wants to stop the automatic accompaniment, the user should click the stop button.

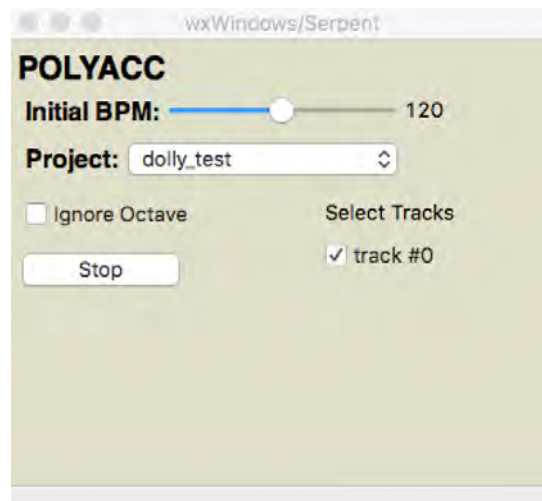


Figure 4-2: User interface under the user mode

Figure 4-3 shows the user interface of this system under the test mode. Like the score for the solo and the accompaniment, the performance is also stored in a standard MIDI file. There are two more buttons on this test mode user interface compared to the user mode user interface: Acc Test and Log Plot. The

developer can use the system exactly the same as a normal user if these two extra buttons are not clicked. Clicking Acc Test button will command the system to read the test performance MIDI file and simulate the human performance. The Log Plot button is used for the experimental data visualization. It is helpful for the test and the evaluation.

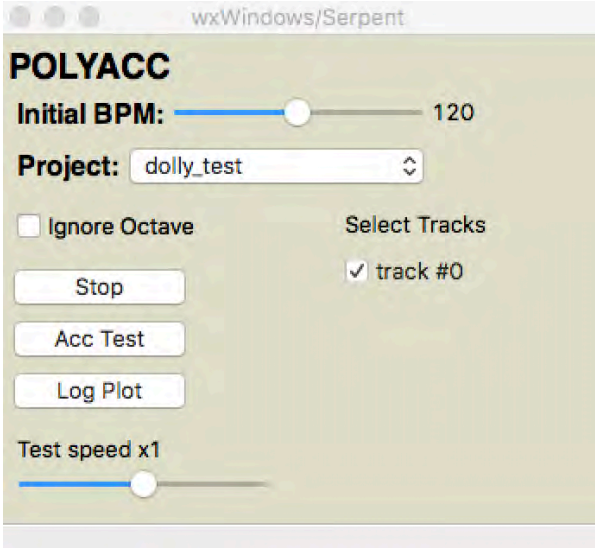


Figure 4-3: User interface under test mode

5 Evaluation

5.1 Dataset

Two types of test data are used for the evaluation of the polyphonic computer accompaniment system: synthetic performance and recorded piano duet performance. All the data is stored in standard MIDI files. In the experiment, the computer accompaniment system first reads the solo score and accompaniment score, then conducts the preprocessing. The performance is simulated by using the pitch, loudness, onset time, and offset time information in the standard MIDI file. The purpose of using synthetic performances for evaluation is to observe how the accompaniment is synchronized under the circumstances of unsteady tempo, wrong notes, extra notes and skipped notes. Pieces for recorded piano duet performance are longer and more complex. The purpose of using recorded performance is to evaluate the performance of the computer accompaniment system under the circumstances of real human performance.

5.1.1 Synthetic performance

Synthetic performance and the corresponding solo score and accompaniment score are created by using MIDIUtil, a python library for creating MIDI files. As Figure 5-1 shows, the solo score is based on the C-major scale with additional parallel major fifths. As Figure 5-2 shows, the accompaniment score is based on the C-major scale. Different performances are created. Unlike the solo scores which have steady tempo and correct notes, tempo changes and note mistakes are introduced to the test performance.



Figure 5-1: The solo score for synthetic performance



Figure 5-2: The accompaniment score for synthetic performance

5.1.2 Recorded piano duet performance

The recorded piano performance data [10] was collected from 10 music master students from the School of Music in Carnegie Mellon University. 10 students were split into 5 pairs to perform duet pieces by sitting face to face. Pieces are recorded by electronic pianos with MIDI output. All the parameters of

every note are recorded in real time so that the performance can be simulated using these parameters. Each piece contains a monophonic part as the accompaniment and a polyphonic part as the solo. Table 5-1 shows the pieces of the piano duet performance used to evaluate the accompaniment system.

Table 5-1: Piano duet performance dataset

Piece name	#total performances
Danny Boy	42
Serenade	35
La dernière Rose	12

5.2 Metrics

Lorin Grubb [4] presented a system for automatically accompanying a singer given a musical score. The accompaniment system was evaluated by comparing the notes in the computer-generated accompaniment against the soloists' performance to assess the synchronization of the performance.

In this polyphonic accompaniment system, the virtual time and real time are recorded each time a performance event is detected and an accompaniment event is played. The virtual time vt is in beats. The real time rt is in seconds. The $rt-vt$ lines of the performance and the computer-generated accompaniment can be plotted to show the tempo change. If the two lines are plotted in the same figure, one for the performance and one for the accompaniment, we can observe how they are synchronized. However, a clearer way to show how the accompaniment follows the performance is to compute Δrt , the difference between the real time of the performance and the accompaniment given the virtual time. $\Delta rt = rt_{perf} - rt_{acc}$. We compute Δrt at virtual time vt if and only if there is a computer-generated accompaniment event and a matched performance compound events at this virtual time. At the same virtual time, if the accompaniment is earlier than the performance, $\Delta rt > 0$; if the performance is earlier than the accompaniment, $\Delta rt < 0$. A $vt-\Delta rt$ line can be plotted to show more details about the synchronization of the performance and the accompaniment.

$|\Delta rt|_{ave}$ represents the average $|\Delta rt|$ computed from a test solo performance and the computer-generated accompaniment. It can be used to measure the synchronization performance of the system accompanying the polyphonic part of a piano duet performance. The mean value of $|\Delta rt|_{ave}$ among all the test performances of the same piece can be computed for the evaluation.

5.3 Results

5.3.1 Synthetic performance

Before assessing how this system handles tempo change and note mistakes, we use the solo score to simulate a perfectly correct performance. Figure 5-3 shows the plot of rt and vt of the performance and accompaniment for the test performance that is the same as the solo score. Every performance event is marked as a “+” on the line of the performance. Every accompaniment event is marked as an “x” on the line of the accompaniment. The accompaniment line almost completely covers the performance line which means that the accompaniment follows the performance almost perfectly.

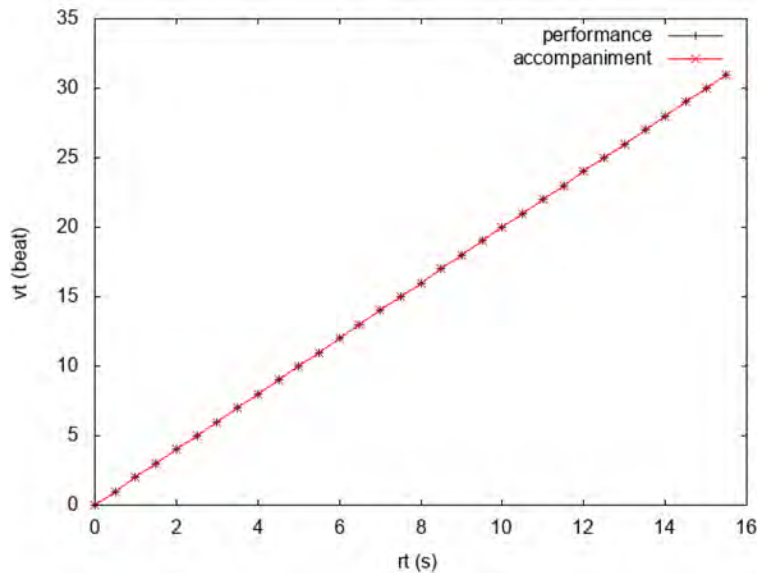


Figure 5-3: Plot of rt and vt of performance and accompaniment for the test performance that is the same as the solo score

Figure 5-4 shows the plot of vt and Δrt between the performance and accompaniment. Since there is no tempo change or note mistake, the synchronization error can be introduced by two facts. The first fact is the frequency that the scheduler dispatches pending events which is set by a parameter called polling time. The performance onset times and output scheduling are quantized to the real-time scheduler’s polling interval. The second fact is the computation time which may add additional delays in milliseconds. The polling time is set to 2ms by default and the maximum value of Δrt in Figure 5-4 happens to be close to 2ms. To show the relationship of synchronization error to polling time, we measured Δrt with polling periods of 2ms, 5ms, 10ms, and 20ms. We used 1.001s as the interval between chords in the test performance to reduce the possibilities that events are scheduled at the time of polling. As Figure 5-5

illustrates, the synchronization errors increase linearly and are mostly of the same magnitude as the polling period, which indicates that most of the error is caused by the quantization to the polling period. Sometimes the maximum error can be several milliseconds more than the polling period which may be caused by other computation and is negligible for music.

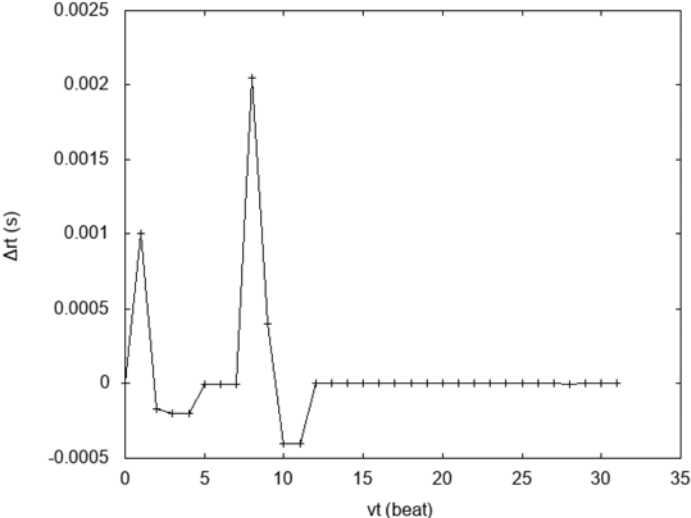


Figure 5-4: Plot of vt and Δrt between performance and accompaniment for the test performance that is the same as the solo score

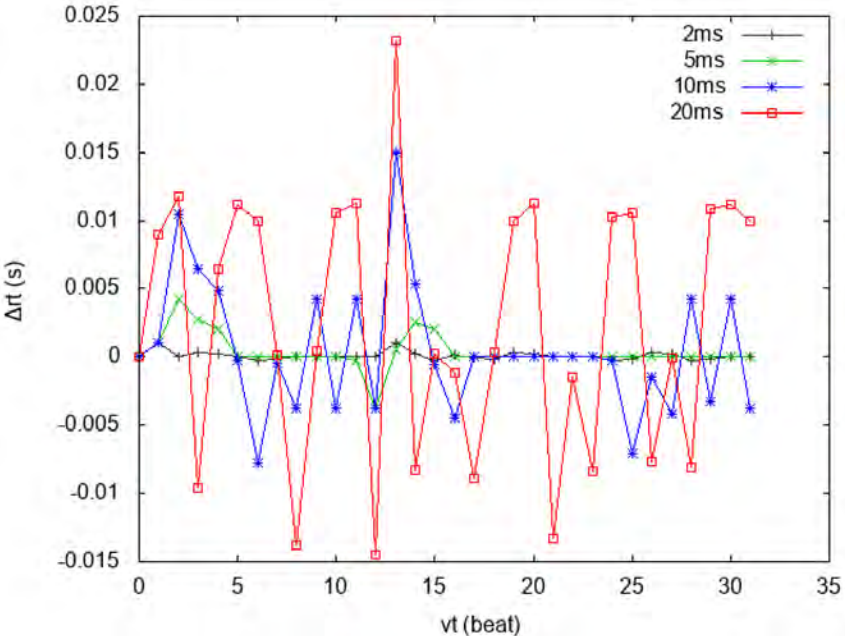


Figure 5-5: Plot of vt and Δrt between performance and accompaniment with polling periods 2, 5, 10, and 20ms

Figure 5-6 shows the test performance that slows down. The tempo becomes twice as slow after the fifth measure. The interval between consecutive chords is 0.5s in the first 4 measures and 1s in the last 8 measures. We label the six chords starting from measure 5 as C, D, E, F, G, and A.



Figure 5-6: Test performance that slows down

Figure 5-7 shows the plot of rt and vt of the performance and accompaniment for the test performance that slows down. Figure 5-8 shows the corresponding plot of vt and Δrt . The performance and accompaniment are synchronized in the first 4 measures. When chord C in measure 5 is played, the corresponding accompaniment for chord C also occurs at the same time. 0.5s later, the corresponding accompaniment for chord D is played because the system does not know the change of the tempo and keeps the previous tempo. 0.5s later, the accompanist continues to play the corresponding accompaniment for chord E because there is no new match reported since chord C. At the same time, as the dotted line in Figure 5-7 shows, the chord D in the performance is detected. But this chord D is 0.5s later than its corresponding accompaniment. This chord D matches the score, so the system knows that the accompaniment is earlier than the performance and the tempo needs to be adjusted. Then the chord E in the performance is played 1s later than its corresponding accompaniment while the accompanist is waiting for the performance to catch up. The next three corresponding accompaniment events for chord F, G, and A are played at different adjusted tempos to smoothly synchronize with the performance again.

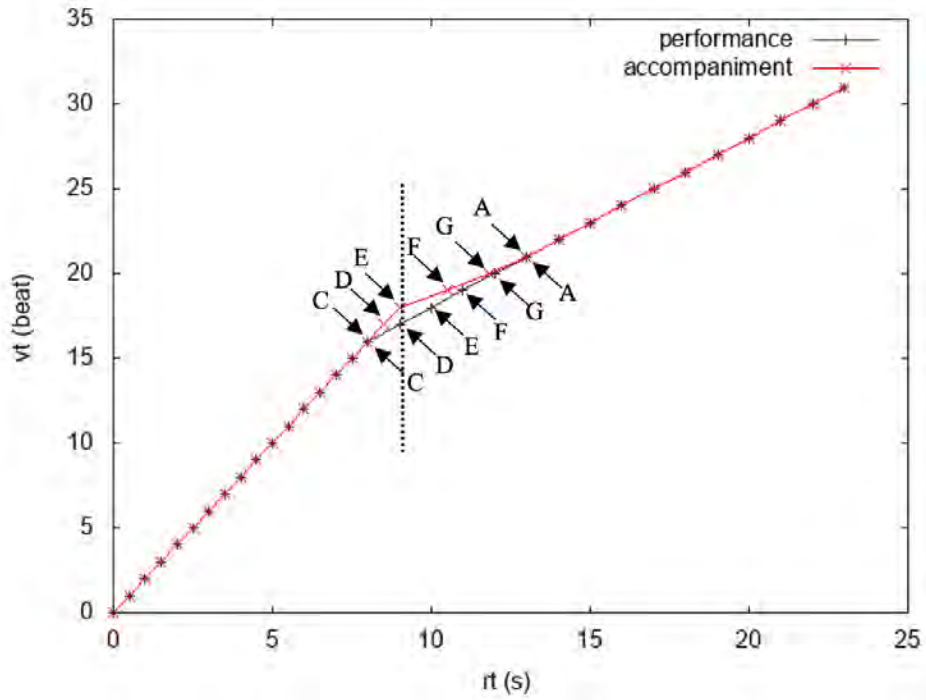


Figure 5-7: Plot of rt and vt of performance and accompaniment for the test performance that slows down

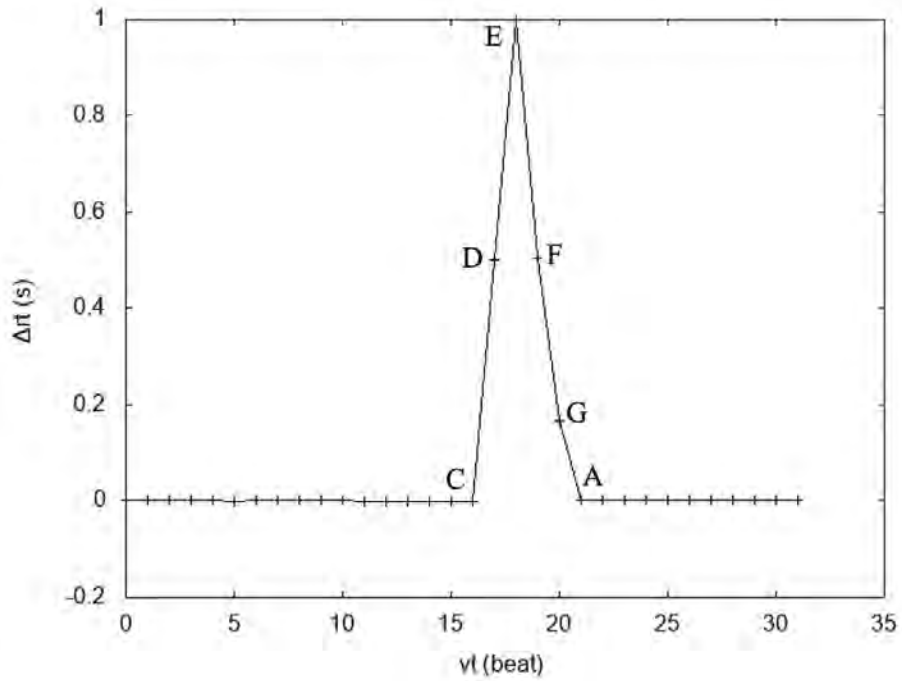


Figure 5-8: Plot of vt and Δrt between performance and accompaniment for the test performance that slows down

Figure 5-9 shows the test performance that speeds up. The tempo becomes twice as fast after the fifth measure. The interval between consecutive chords is 1s in the first 4 measure and 0.5s in the last 2 measures. We label the six chords starting from measure 5 as C, D, E, F, G, and A.



Figure 5-9: Test performance that speeds up

According to the plot in Figure 5-10 and the plot in Figure 5-11, the accompaniment smoothly synchronizes with the performance with the new tempo when the performance speeds up. The performance and accompaniment are synchronized in the first 4 measures. When chord C in measure 5 is played, its corresponding accompaniment also occurs at the same time. The system expects the chord D to be played 1s later, however, chord D is played only 0.5s later. Since chord D is a new match, the accompanist knows that the performance is earlier than the accompaniment and the accompaniment needs to catch up. Instead of continuing with the original tempo, the accompaniment speeds up smoothly. The corresponding accompaniment events for chord D, E, F, and G all occur later than chord D, E, F, and G in the performance are played. The performance and accompaniment become synchronized again at chord A in measure 5. Comparing the above two examples, we can observe the computer accompaniment system reacts faster when the tempo speeds up compared to the situation when the tempo slows down. For the test performance that speeds up, the system realizes the tempo change before the corresponding accompaniment for chord D is played, while for the test performance that slows down, the system realizes the tempo change at the corresponding accompaniment for chord E. This is one reason why Δrt between the performance and accompaniment in Figure 5-11 is much smaller than the difference in Figure 5-8. Another reason is that the time interval between beats is wider when slowing down which increases Δrt .

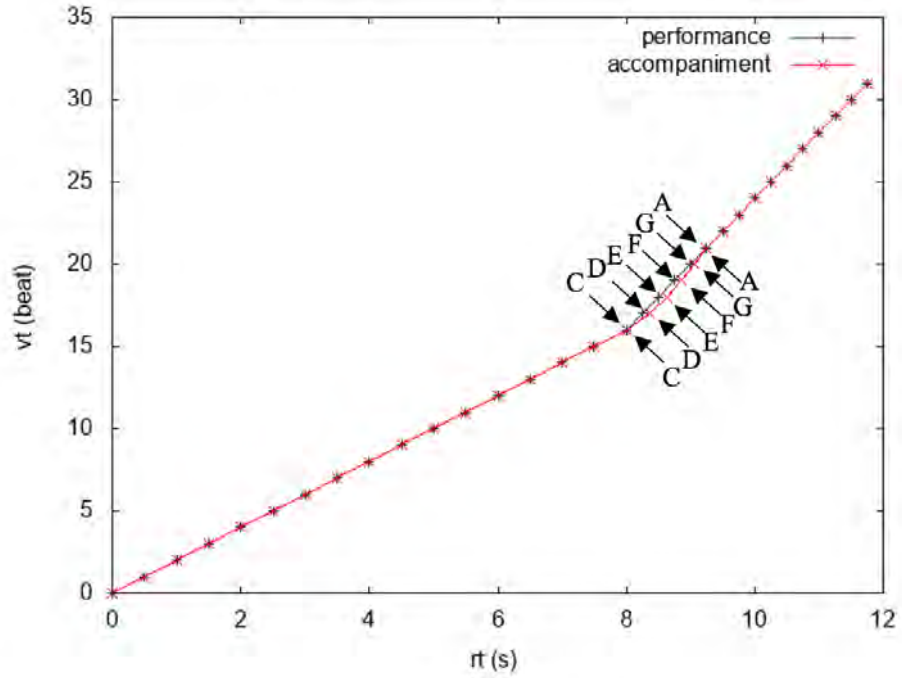


Figure 5-10: Plot of rt and vt of performance and accompaniment for the test performance that speeds up

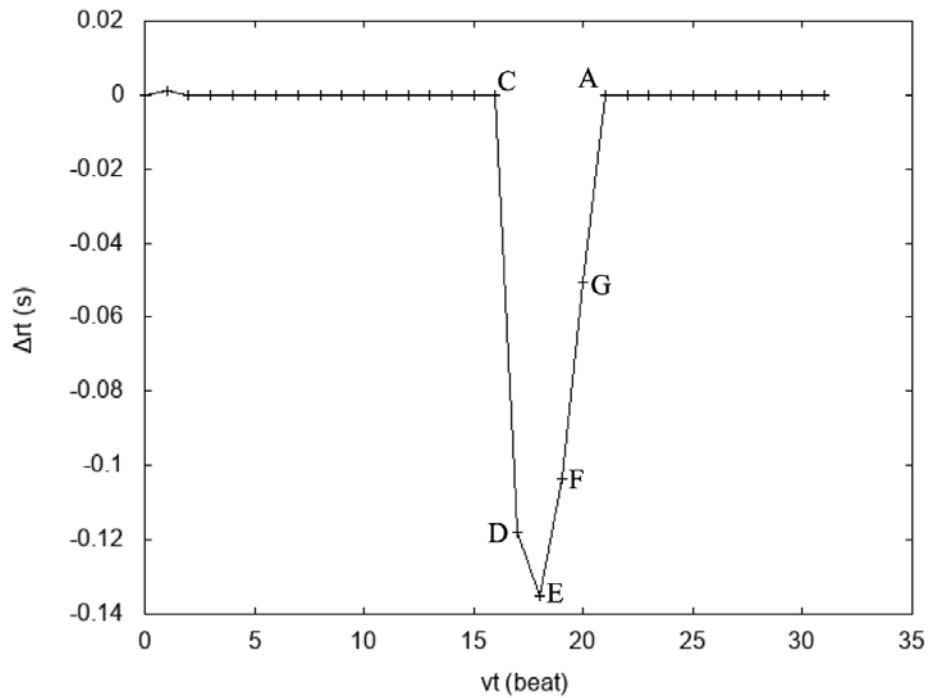


Figure 5-11: Plot of vt and Δrt between performance and accompaniment for the test performance that speeds up

Figure 5-12 shows the test performance with wrong notes which are circled. We label the wrong chord in measure 3 as W_1 , the wrong chord in measure 7 as W_2 .



Figure 5-12: Test performance with wrong notes

The plot in Figure 5-13 and the plot in Figure 5-14 are very similar to the results of the test performance that is the same as the solo score: accompaniment almost perfectly synchronizes with the performance and the maximum Δrt is close to 2 ms. W_1 and W_2 are wrong notes and cannot match the score, but the accompaniment line still covers the performance time. It is natural to expect such a result. According to the matching algorithm, if no new match is found, the accompanist just keeps playing with the latest updated tempo so that nothing changes.

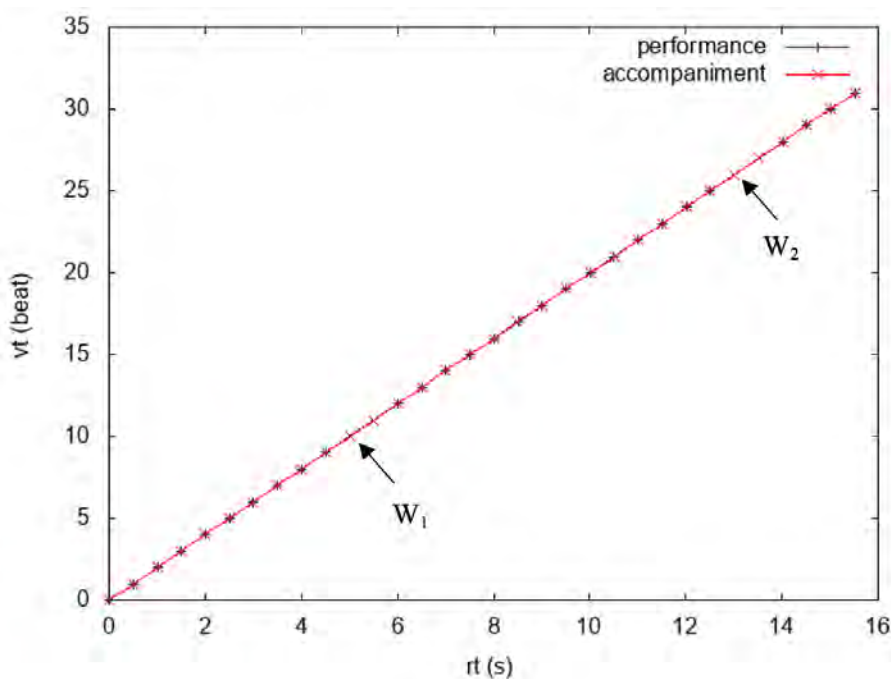


Figure 5-13: Plot of rt and vt of performance and accompaniment for the test performance with wrong notes

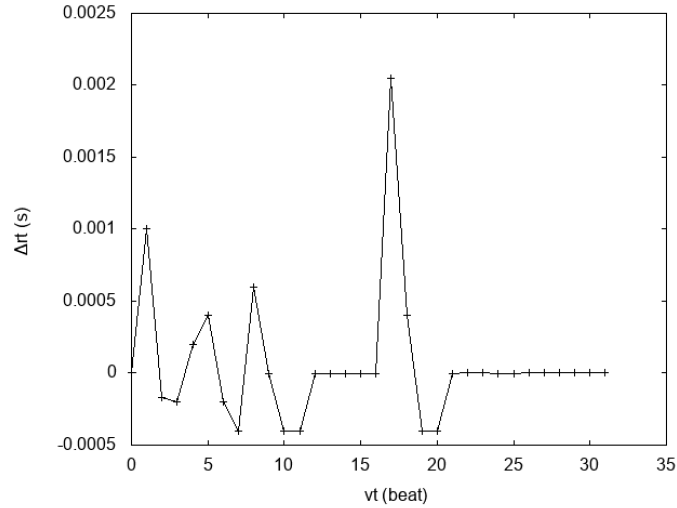


Figure 5-14: Plot of vt and Δrt between performance and accompaniment for the test performance with wrong notes

Figure 5-15 shows the test performance with extra notes. The extra notes are circled. We label the chord before the extra note as B, the chords following it as A, G, F, E, D, C.



Figure 5-15: Test performance with extra notes

Figure 5-16 shows the plot of rt and vt of the performance and accompaniment for the test performance that has extra notes and Figure 5-17 shows the corresponding Δrt . The performance and accompaniment are synchronized before the extra chord occurs. The extra chord occurs at the same time with the corresponding accompaniment for chord A. Therefore, the accompaniment becomes earlier than the performance. The corresponding accompaniment for G occurs at the same time when chord A in the performance is played, as the dotted line in Figure 5-16 shows. Chord A matches the score and it is the first new match after chord B is played. Then the system realizes that the accompaniment is earlier than the performance. Therefore, it seems like this problem becomes a tempo slow down problem. After that, chord G and F are played with the original tempo before the next accompaniment, the corresponding accompaniment event for chord F, occurs. Now the accompaniment has to speed up to catch up with the performance. The performance and accompaniment become synchronized again at chord C. The accompaniment system reacts to extra notes in a pretty similar way as how it reacts to the situation when

tempo slows down. Extra notes can be regarded as a temporary slower tempo, which means after two chords following the extra chord (chord A and chord G) are played, the tempo of the performance comes back to normal

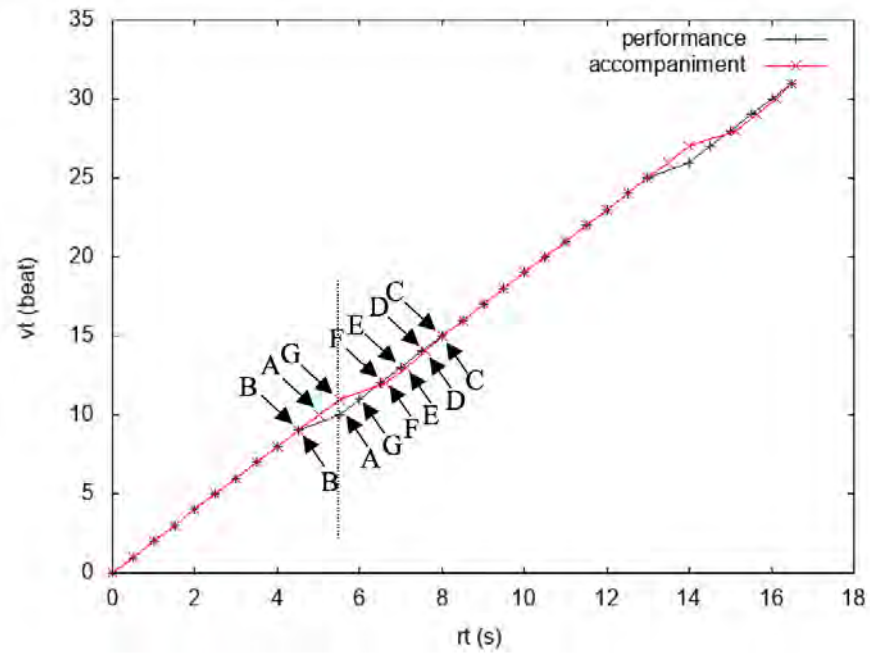


Figure 5-16: Plot of rt and vt of performance and accompaniment for the test performance with extra notes

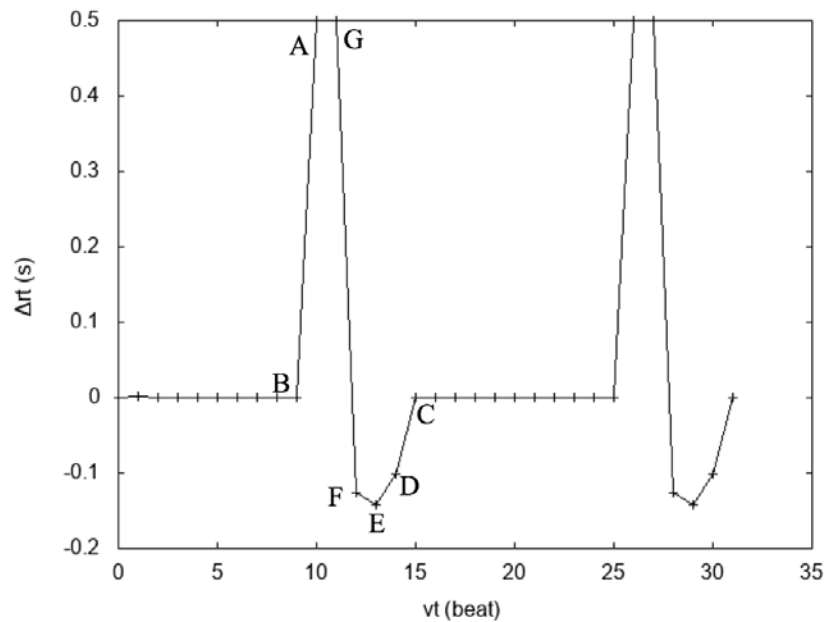


Figure 5-17: Plot of vt and Δrt between performance and accompaniment for the test performance with extra notes

Figure 5-18 shows the test performance with a skipped note which is represented by circles. We label the missing chord in the 3rd measure as M, the chord before it as B, the chords following it as G, F, E, and D.



Figure 5-18: Test performance with skipped notes

According to the plot in Figure 5-19 and Figure 5-20, the accompaniment system reacts to skipped notes in a pretty similar way as how it reacts to the situation when tempo speeds up. Skipped notes can be regarded as a temporary speed up, which means after two events following the skipped note (chord G and chord F) are played, the tempo of the performance comes back to normal. The performance and accompaniment are synchronized at chord B. Then the corresponding accompaniment for the skipped note and the chord G in the performance occur at the same time, as the dotted line in Figure 5-19 shows. It seems like the chord G matches the score and the system should react to catch up, however, a penalty mentioned in the algorithm section is applied here. Therefore, at the performance chord F, a new match is reported, the accompanist starts to adjust the tempo. The performance and accompaniment become synchronized again at chord D.

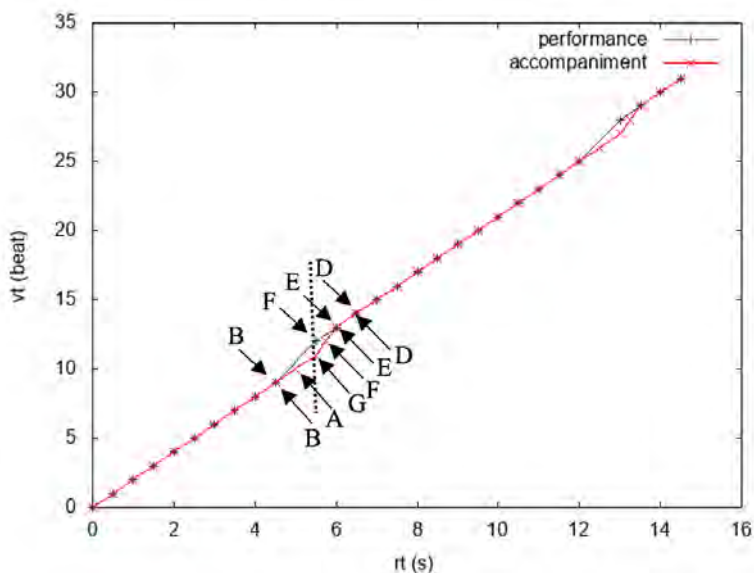


Figure 5-19: Plot of rt and vt of performance and accompaniment for the test performance with skipped notes

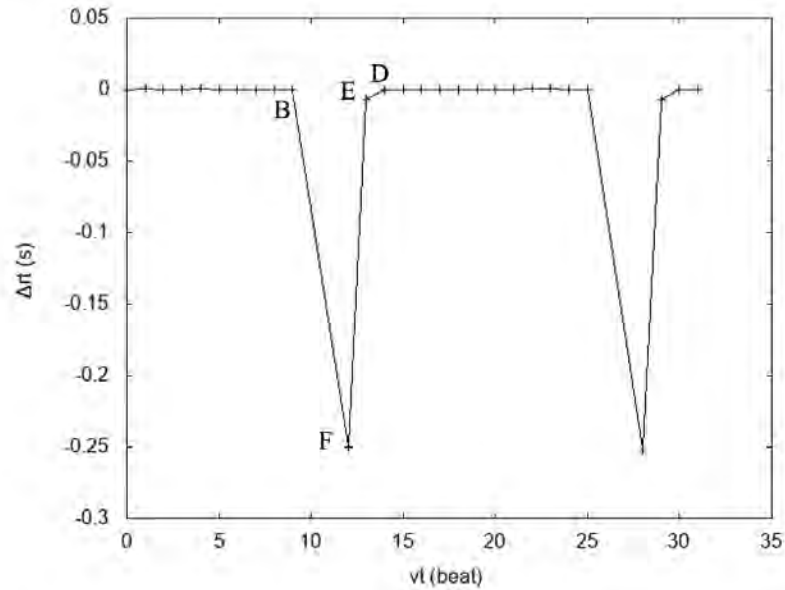


Figure 5-20: Plot of vt and Δrt between performance and accompaniment for the test performance with skipped notes

5.3.2 Recorded piano duet performance

For the same piano duet piece, different recorded performances have different beats per minute. Therefore, the polyphonic computer accompaniment system needs to figure out the tempo for each test performance when the simulated performance starts to play. All the performances have a reasonable number of note mistakes and unsteady tempo. For the evaluation using synthetic performances, the plot of vt and Δrt between the performance and accompaniment is plotted to show how they are synchronized. For each recorded piano duet performance, the average $|\Delta rt|$ among accompaniment events can be computed to measure the synchronization performance.

Table 5-2 illustrates $|\Delta rt|_{\text{ave}}$ of all the test performances for the piece of Danny Boy. The mean value of $|\Delta rt|_{\text{ave}}$ among all the performances is 0.0865803s. Table 5-3 illustrates $|\Delta rt|_{\text{ave}}$ of all the test performances for the piece of Serenade. The mean value of $|\Delta rt|_{\text{ave}}$ among all the performances is 0.0505401s. Table 5-4 illustrates $|\Delta rt|_{\text{ave}}$ of all the test performances for the piece of La dernière Rose. The mean value of $|\Delta rt|_{\text{ave}}$ among all the performances is 0.0560346s.

Table 5-2: Results of $|\Delta rt|_{ave}$ for test performances of Danny Boy

	$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)
1	0.1477160	8	0.0370765	15	0.1323370	22	0.0504487	29	0.1322920	36	0.1351150
2	0.0335107	9	0.0872842	16	0.1962520	23	0.0355220	30	0.0591177	37	0.0426883
3	0.1459950	10	0.1100590	17	0.0756882	24	0.0338080	31	0.1523400	38	0.0804868
4	0.0466160	11	0.0420698	18	0.1271630	25	0.0804962	32	0.1258860	39	0.0833166
5	0.1901080	12	0.0787874	19	0.0229604	26	0.0283103	33	0.0480882	40	0.0726486
6	0.0817291	13	0.0728359	20	0.0376662	27	0.0656379	34	0.1138740	41	0.0677503
7	0.0762181	14	0.0516108	21	0.0949125	28	0.0340777	35	0.1117580	42	0.1941130

Table 5-3: Results of $|\Delta rt|_{ave}$ for test performances of Serenade

	$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)
1	0.0224031	8	0.0633195	15	0.0374114	22	0.0489429	29	0.0423139
2	0.0242313	9	0.0551428	16	0.0458098	23	0.0372623	30	0.0430136
3	0.0269433	10	0.0701211	17	0.0474460	24	0.0580590	31	0.0421933
4	0.0354680	11	0.0591699	18	0.0635487	25	0.0534466	32	0.0313594
5	0.0396549	12	0.0608684	19	0.0448988	26	0.0653556	33	0.0863104
6	0.0724113	13	0.0709298	20	0.0443525	27	0.0284227	34	0.0784586
7	0.0581509	14	0.0624161	21	0.0299323	28	0.0600659	35	0.0590691

Table 5-4: Results of $|\Delta rt|_{ave}$ for test performances of La dernière Rose

	$ \Delta rt _{ave}$ (s)		$ \Delta rt _{ave}$ (s)
1	0.0505065	7	0.0364900
2	0.0255504	8	0.0407825
3	0.0357890	9	0.0217626
4	0.0260972	10	0.2492690
5	0.0228528	11	0.0255733
6	0.0378595	12	0.0998824

5.4 Analysis

Xia, G. [10] applied machine learning techniques to expressive collaborative music performance. In his work, the ground truths of human performances and the average absolute asynchrony of the linear regression baseline were computed using the same piano duet performance dataset. He evaluated the system using monophonic parts as the solo performance and polyphonic parts as the accompaniment, while our polyphonic accompaniment system was evaluated using polyphonic parts as the solo performance and monophonic parts as the accompaniment. Figure 5-21 shows the comparison of average absolute asynchrony. Each piece has three bars to show its average $|\Delta r|_{ave}$ in three different accompaniment types: GT, Mono, and Poly. The left green bars represent GT, which means the ground truths of human performances. The middle red bars represent Poly, which means the results we computed above. The right blue bars represent Mono, which means the baseline using linear regression from the work of Xia, G. In Figure 5-21, the average absolute time difference of GT can be less than or more than the average absolute time difference of this polyphonic accompaniment system. This is because the computer accompanist can react to performance that speeds up faster than the human accompanist. For Danny Boy and Serenade, the polyphonic accompaniment system has better synchronization performance than Xia, G.'s system. For La dernière Rose, Xia, G.'s system synchronized better than this polyphonic accompaniment system. We analyze the reasons for the performance difference between Poly and Mono by replicating the accompaniment results of the monophonic accompaniment system.

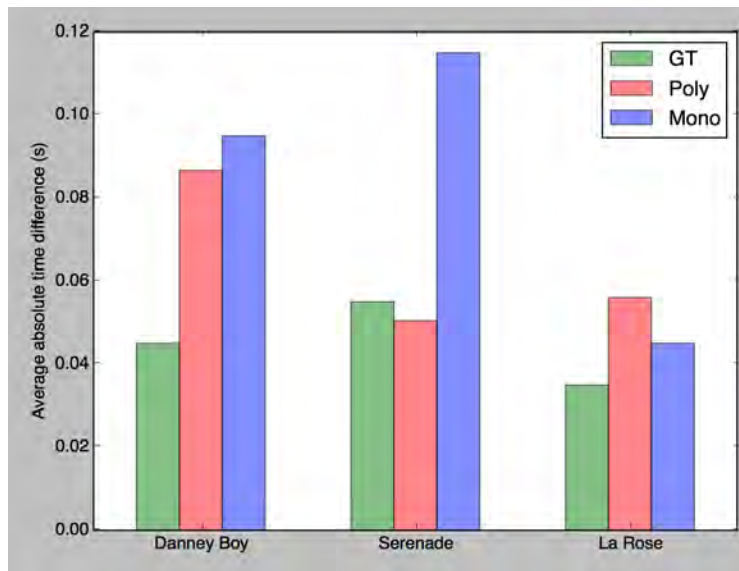


Figure 5-21: Comparison of average absolute asynchrony

Figure 5-22 illustrates part of the plot of rt and vt of the monophonic performance and polyphonic accompaniment for a test performance of Serenade. Within the range of the two arrows, the distance between the performance line and the accompaniment line become greater and greater as more and more accompaniment chords are played. The absolute time difference at the next matched melody note and the corresponding accompaniment chords is large. Figure 5-23 illustrates part of the plot of rt and vt of the polyphonic performance and monophonic accompaniment. The accompaniment line covers the performance line much better than the monophonic accompaniment system. The corresponding solo score is shown in Figure 5-24. With many rest signs, the number of monophonic events is not enough for accurately predicting the time of following accompaniment chords using linear regression.

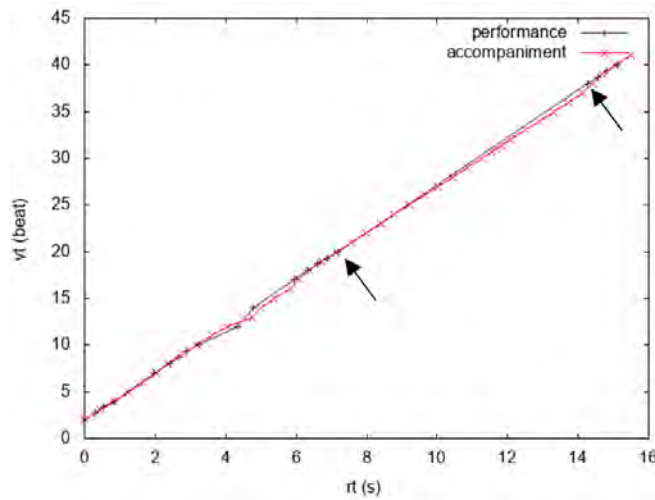


Figure 5-22: Part of the plot of rt and vt of mono performance and poly accompaniment for a test performance of Serenade

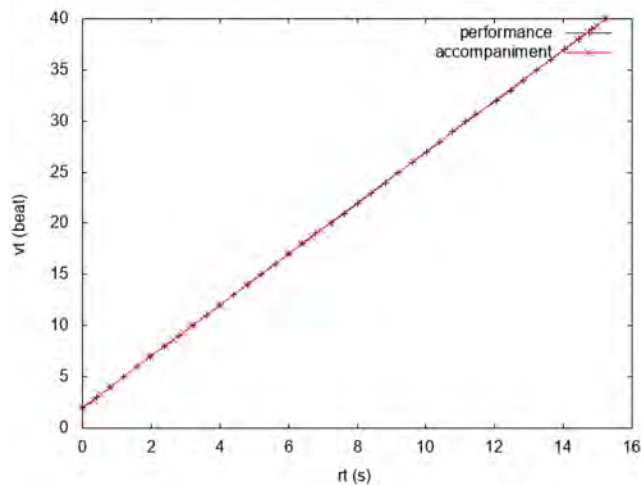


Figure 5-23: Part of the plot of rt and vt of poly performance and mono accompaniment for a test performance of Serenade



Figure 5-24: Part of the score of the monophonic part of Serenade

Figure 5-25 illustrates part of the plot of rt and vt of the monophonic performance and polyphonic accompaniment for a test performance of Danny Boy. Figure 5-26 illustrates part of the plot of rt and vt of the polyphonic performance and monophonic accompaniment. The distribution of melody events is close to the distribution of accompaniment chords, so using the monophonic part or the polyphonic part as the solo performance have similar synchronization results.

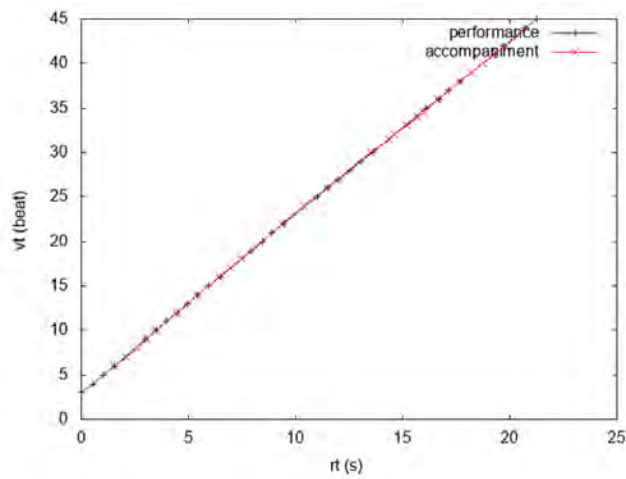


Figure 5-25: Part of the plot of rt and vt of mono performance and poly accompaniment for a test performance of Danny Boy

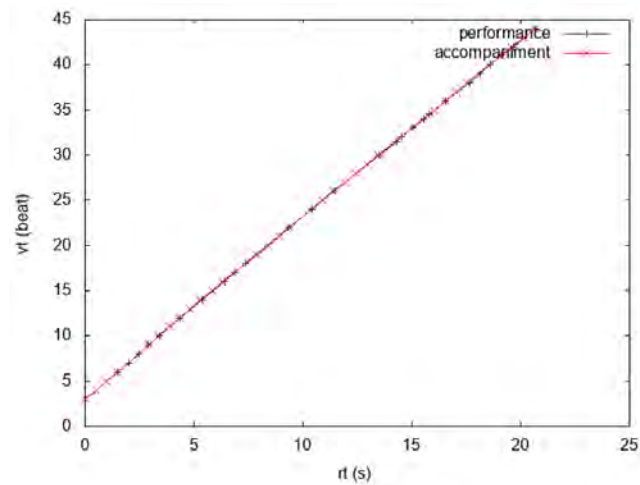


Figure 5-26: Part of the plot of rt and vt of poly performance and mono accompaniment for a test performance of Danny Boy

Figure 5-27 illustrates part of the plot of rt and vt of the monophonic performance and polyphonic accompaniment for a test performance of La dernière Rose. Figure 5-28 illustrates part of the plot of rt and vt of the polyphonic performance and monophonic accompaniment. The monophonic events distribute more densely than the accompaniment chords. Using the dense monophonic events can achieve better prediction. That is why using monophonic events as the input has better synchronization performance than using polyphonic chords as the input.

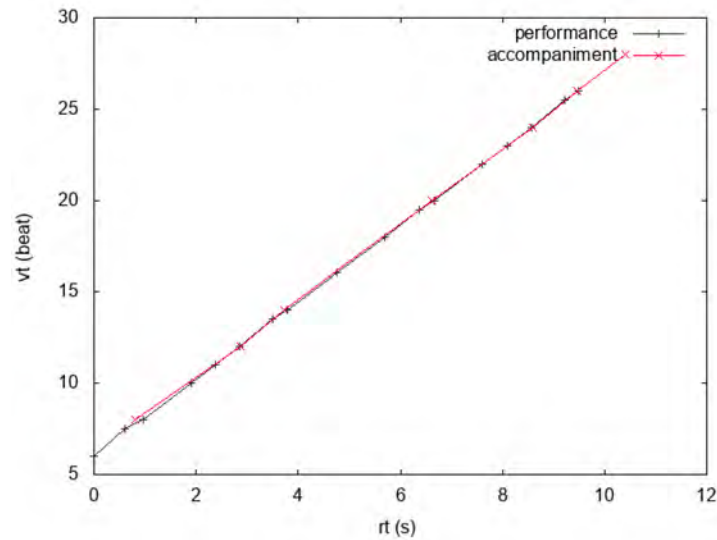


Figure 5-27: Part of the plot of rt and vt of mono performance and poly accompaniment for a test performance of La dernière Rose

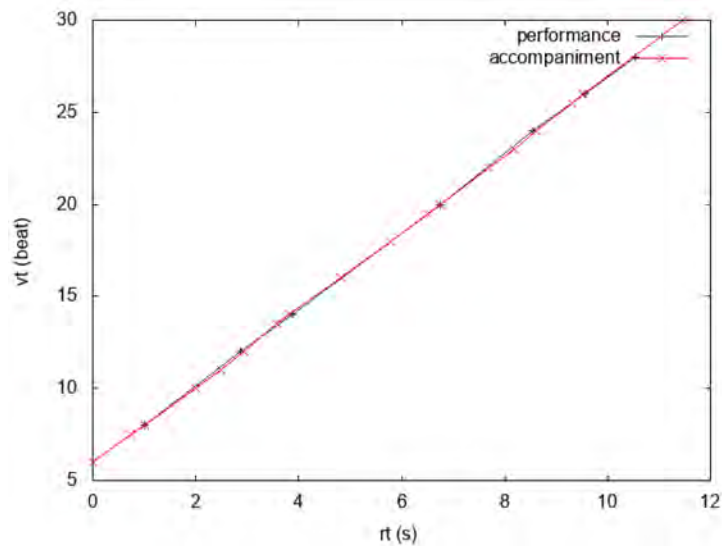


Figure 5-28: Part of the plot of rt and vt of poly performance and mono accompaniment for a test performance of La dernière Rose

In general, whether using the monophonic or polyphonic part as the input depends on the distribution of monophonic melody events and polyphonic accompaniment chords of a piece. If melody events distribute more densely, then using the monophonic part as the input results to better prediction and better synchronization. If accompaniment chords distribute more densely, then using the polyphonic part as the input results in smaller absolute time difference and less synchronization errors.

6 Conclusion

My thesis project implements a computer accompaniment system for polyphonic keyboard performance. This system uses a MIDI keyboard as the solo performance input. Given the solo and accompaniment score, users can listen to the corresponding accompaniment when they perform on the MIDI keyboard. The solo performance can also be simulated for testing if the necessary note event information is available from a standard MIDI file. Evaluations are conducted on two types of datasets. By using synthetic performance data, we can explore how the accompaniment adjustment works to synchronize with tempo change or incorrect notes. By using recorded piano duet performance data, we observe how this accompaniment system follows a realistic solo performance. The results show that this accompaniment system successfully tracks the real-time solo performance and synchronizes the corresponding accompaniment. The accompaniment system handles unsteady tempos, extra notes, skipped notes and wrong notes.

To make it easier for users to download and install the accompaniment system, we can bundle Serpent into an application on OS X. Also, good documentation helps new users to explore and enjoy the system as soon as possible.

The current implementation accepts input from only one MIDI keyboard. However, it has the potential to accept the input from multiple MIDI keyboards at a time. Each performer can be tracked separately with his or her own matcher and the matching results from these matchers can be merged to decide the score position which is sent to the accompanist to synchronize the corresponding accompaniment [11].

The current system only allows accompaniment parameters to be changed from the source code before the system starts to run. In different applications, it is better to adjust these parameters to customize the accompaniment, in which case the features for setting accompaniment parameters should be added. Also, sometimes the soloist wants to follow the accompaniment. This function can be realized by fixing the accompaniment tempo in some sections. Therefore, mechanisms to change parameters during the performance can be useful.

One application of accompaniment is to synchronize signal processing effects with live input, so it is important to make it possible for the accompaniment system to interact with other computer music systems or audio processing systems, such as MAX/MSP. One way for the interaction is to use Open Sound Control.

Another application of accompaniment is to help musicians rehearse for a concert. Consider the case of a harpsichordist preparing to play a concerto with strings. It would be great to have an interactive accompaniment system to simulate the concerto performance experience, but an "accompanist" that obediently follows every nuance of the harpsichord is not a realistic model of a string orchestra. This raises the question of how to best simulate the behavior of an orchestra (or any collaborative performance, for that matter), and what would provide the most helpful rehearsal and practice system for the harpsichordist? More generally, for any musician in any musical performance, how can score following and computer accompaniment best be used to practice and prepare for real rehearsals and performances?

Reference

- [1] Dannenberg, R. (1984). "An On-Line Algorithm for Real-Time Accompaniment." Proceedings of the International Computer Music Conference, 193-198.
- [2] Bloch, J. & Dannenberg, R. (1985). "Real-Time Accompaniment of Keyboard Performance." Proceedings of the International Computer Music Conference, 279-289.
- [3] Baird, B., Blevins, D., & Zahler, N. (1993). "Artificial Intelligence and Music: Implementing an Interactive Computer Performer." Computer Music Journal, 17(2), 73-79.
- [4] Grubb, L. (1998). "A Probabilistic Method for Tracking a Vocalist." Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- [5] Müller, M., Kurth F., & Röder, T. (2004). "Towards an Efficient Algorithm for Automatic Score-to-Audio Synchronization." Proceedings of the 5th ISMIR Conference.
- [6] Raphael, C. (2004). "A Hybrid Graphical Model for Aligning Polyphonic Audio with Musical Scores." Proceedings of the 5th ISMIR Conference.
- [7] Dixon, S. (2005). "Live Tracking of Musical Performances Using On-Line Time Warping." Proceedings of the 8th International Conference on Digital Audio Effects, 92-97.
- [8] Cont, A. (2008). "ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music." Proceedings of the International Computer Music Conference, 33-40.
- [9] Dannenberg, R., Gold, N., Liang, D., & Xia, G. (2014). "Methods and Prospects for Human-Computer Performance of Popular Music." Computer Music Journal, 38(2), 36-50.
- [10] Xia, G. (2016). "Expressive Collaborative Music Performance via Machine Learning." Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- [11] Grubb, L. & Dannenberg, R. (1994). "Automating Ensemble Performance." Proceedings of the International Computer Music Conference, 63-69.