Automatic Synchronization between Audio and Partial Music Score Representation

Antonello D'Aguanno and Giancarlo Vercellesi

Laboratorio di Informatica Musicale (LIM)
Dipartimento di Informatica e Comunicazione (DICo)
Universitá degli Studi di Milano,
Via Comelico 39,
I-20135 Milano, Italy
{daguanno,vercellesi}@dico.unimi.it
http://www.lim.dico.unimi.it

Abstract. In this paper we propose an algorithm namely ParSi (Partial Synchronization) which aligns different PCM executions with a partial music score representation. Generally, most synchronization algorithms require a complete score representation; despite of them, our algorithm is able to synchronize different PCM audio executions with a partial music score codified in MIDI. Therefore, only one MIDI instrument -chosen by user- is used during audio-score alignment. The audio analysis is performed using a notch filter. A set of conveniently parameters are used during the decisional phase. The analysis of synchronization results is provided. It shows we can have a good automatic synchronization even if just a partial score is available.

Keywords: MIR, MIDI, synchronization, alignment, score following.

1 Introduction

Contemporary digital music archives consist of huge collections of heterogeneous documents. For a music piece, an archive may contain corresponding scores in different versions, for example, voice and piano or orchestral, as well as several interpretations, e.g. played by different performers and recorded in diverse formats (CD recordings, MP3, FLAC and so on). The heterogeneity of music information makes retrieval hard to accomplish [1] [2] [3] and as a consequence many problems remain unsolved [4]. One important problem that needs a solution is synchronization, which requires the implementation of algorithms that automatically link different audio streams of the same piece to symbolic data formats representing the different scores.

The automatic score-audio alignment can be useful for musicologists who want to investigate agogics and tempo [5]. In addition, temporal linking of score and audio data can be useful for automatic tracking of score positions during a performance. Such algorithms may be also useful for score-following applications [6]. These possibilities represent a useful tool for music students, who can listen to music audio and, at the same time, see the corresponding notes.

The state of the art proposes algorithm which generally requires a complete score representation. Unfortunately, we may have many cases where a partial score is available. For this reason in this paper we describe an algorithm which would like to propose a possible solution to that situations. The ParSi (Partial Synchronization) algorithm is able to synchronize different PCM audio executions using a partial music score at symbolic level. The score has to be codified in MIDI. therefore, only one MIDI instrument -chosen by user- is used during audio-score alignment.

This article is organized as follows: section 2 presents an overview of score following and synchronization algorithms proposed in literature. Section 3 describes the ParSi algorithm. The analysis of results is discussed in section 4. Finally, conclusions will be summarized in 5.

2 Related Works

Many algorithms have been proposed in literature that deal with synchronization. The majority of them can be subdivided into two groups: in the first one, audio and score have to be analysed, while the second one requires the realisation of correct links between these two layers ([7][8][9][10][11][12][13] [14]). The algorithms proposed in the literature use several different systems to implement audio analysis, with well-known tools from audio signal processing. For example, [12] uses a Short Time Fourier Transform, in [8] proposes an onset detection followed by pitch detection. In [11] the feature extraction procedure performs these operations: decomposition of the audio signal into spectral bands corresponding to the fundamental pitches and harmonics, followed by the computation of the positions of significant energy increases for each band - such positions are candidates for note onsets. Other popular solutions proposed in the literature to select the correct links between audio and score are based on the *template matching technique* ([12][7][13]). Such algorithms build a MIDI score to obtain a template of the real execution, which is compared to the real audio using a DTW[1] programming technique [15]. The correct synchronization is then obtained from the difference between the agogics of the real execution and that of MIDI.

The algorithm described here uses an approach based on recursive research with notch filter for audio analysis and a partial score event extraction from MIDI coding.

3 The ParSI Algorithm

The algorithm proposed here is able to synchronize a MIDI score with one or more PCM audio executions using only one instrument codified in the MIDI file; this instrument has to be selected by the user. Therefore, we do not try to

[1] Dynamic Time Warping is a technique for aligning time series that has been well known in the speech recognition community since the 1970s.

synchronize the whole score with the audio but we extract score informations from one instrument in order to automatically align it with the audio execution.

ParSi algorithm consists of three different phases (figure 1):

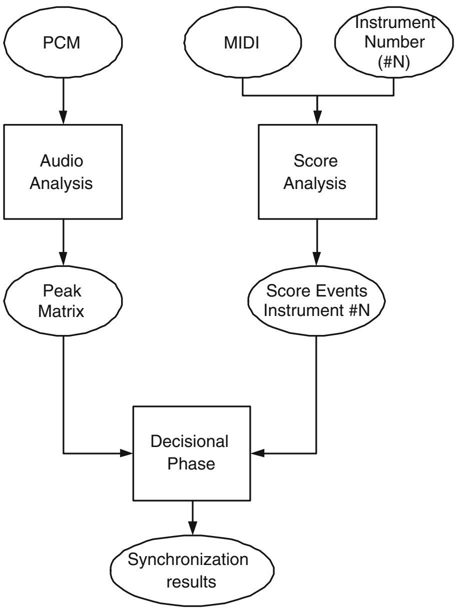

Fig. 1. The ParSi architecture

- in the first phase, the selected instrument codified in the MIDI score is read in order to extract all relevant score events.
- in the second phase, the PCM audio signal is analysed driven by the extracted score events, in order to identify all possible time-attacks notes in terms of energy peaks.
- in the third and last phase, a decisional matching is performed to relate the event at the score level of the selected instrument with the same event at the audio level. The decisional method is based on the attack time of notes extracted at audio level.

Each phase is described in the following sections.

3.1 The MIDI Score Analysis

In this phase, the selected instrument codified in the MIDI file is read to extract the score events as illustrated in figure 2.

For each measure, only notes with strong accent are selected because it is easier to recognize them at audio level. Thus the score is represented in a suitable manner for synchronization. For each strong accent each note is codified with

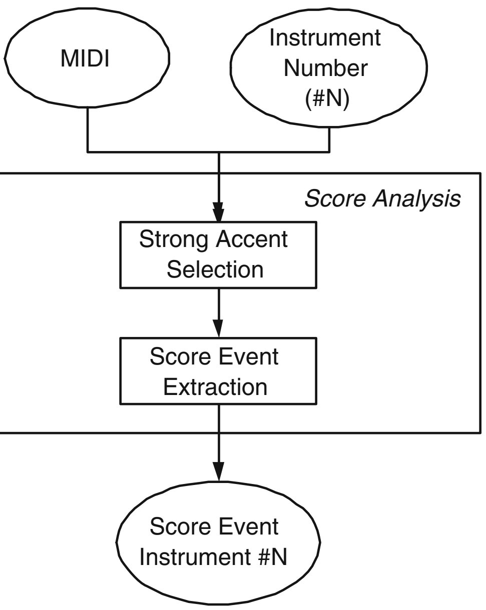

Fig. 2. Score Analysis Flowchart

its fundamental frequency, the duration in MIDI ticks, the beat and measure number. Notes grouped between two asterisks (∗) correspond to a chord. A vertical rest is represented with −1. An example is shown in the table below (1); the corresponding score is shown in figure 3:

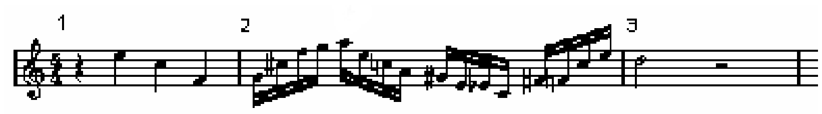

Fig. 3. The score represented in table 1

3.2 The PCM Audio Analysis

In the audio analysis, a recursive, second-order notch filter bank [16] is used as shown in figure 4.

The aim of this phase is the detection of each attack-time for each note codified in the selected instrument at score level. For each verticalized score event (see table (1)), the audio signal is filtered with a notch filter centered both on the fundamental frequency and on the first four harmonics of the examined note. The portion of signal analyzed is calculated defining a conveniently cluster position and size.

The cluster position is obtained as follow. Let T_{track} and T_{score} be the audio track duration in seconds and the MIDI track duration in MIDI ticks, respectively. We calculate the tick duration $T_{tick} = T_{track}/T_{score}$. The cluster position

Table 1. Example of a single instrument coding

Fundamental Frequencies	Durations	Beat	Measure
*			
-1	480	1	1
*			
659	480	1	2
*			
523	480	1	3
*			
349	480	1	4
*			
392	480	2	1
*			
880	480	2	2
*			
415	480	2	3
*			
370	480	2	4
*			
698	960	3	1
*			
-1	960	3	3
*			

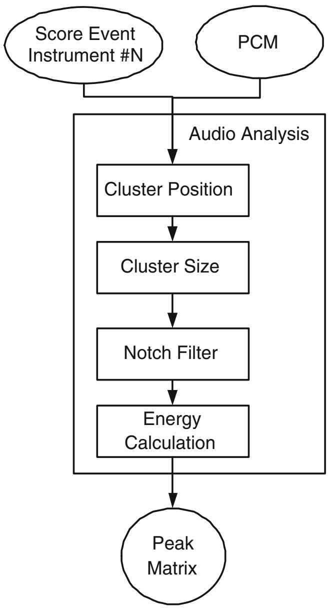

Fig. 4. Audio Analysis Flowchart

is determinated as $T_{cluster} = N_{tick}/T_{tick}$ where N_{tick} is the sum of ticks associated to the previous notes.

The cluster size is calculated as $Cluster\_size = ((T_{track}/N_{events}) * variation$ where N_{events} is the total event number extracted at score level; $Variation$ is an input parameter -related to the interpretation- which represents the amount of tempo variation of audio execution with respect to the MIDI performance. $Variation$ lies on a range from 1 to 4 where 2 means we have not meaningful variation. In other words, this parameter allows to increase or decrease the $cluster\_size$ with respect to the amount of expected tempo changing.

After notch filter, we calculate the energy envelope in order to generate the so called *peak matrix*. After energy computation we calculate its peaks with respect to the score notes. Then, all the possible retrieved peaks are stored into peak matrix as shown in table 2. The first column contains the analyzed fundamental frequency; the matrix row number is equal to the 'number of events extracted at the score level' N_{events}. The second column contains a list of all possible peak times (in seconds). If no peak is retrieved, the matrix is filled with 0.

Table 2. An example of *peak matrix*

Fundamental Frequencies	Retrieved Peak Time
659	$T_1^1, T_2^1, ...T_n^1$
523	$T_1^2, T_2^2, ...T_n^2$
349	$T_1^3, T_2^3, ...T_n^3$
....	....

An example of Notch filtering applied on fundamental frequency is shown in figure 5. Consider note A at 440Hz. While (a) shows its waveform signal, this signal is filtered with the notch filter with center frequency 440Hz (b), which yields a filtered signal with high energy. Note that if the signal is filtered at different center frequencies, the signal will contain less energy: see the cases for center frequency at 220Hz (c) and 880Hz (d).

To be able to retrieve every note, avoiding superimposition between adjacent semitones, the bandwidth of the notch filter is dependent to the analyzed frequency with respect to this formula: $bandwidth = 3 * 2^{\log_{10}(f)}$ where f is the frequency in Hertz.

3.3 The Decisional Phase

The last step of ParSi algorithm is the decisional phase where the time-alignment between audio and score events is created (figure 6). As we said before, only score events belonged to only one instrument (conveniently chosen by user) is considered during automatic synchronization.

We start invoking the **init** function which initializes some variables used during the algorithm: SS_{size}, $N_{virtual\_peaks}$ and $tolerance$:

– the maximum size of subsequence SS_{size} is provided as input by user. This value is used to define the subsequence limits where our algorithm will extract the candidate synchronization sequences;

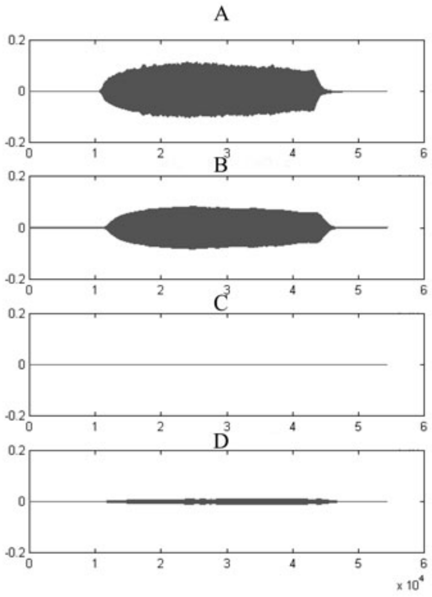

Fig. 5. Spectrum examples of a 440Hz signal (a) filtered with a notch filter having its center frequency at 440Hz (b), 220Hz (c) and 880Hz (d)

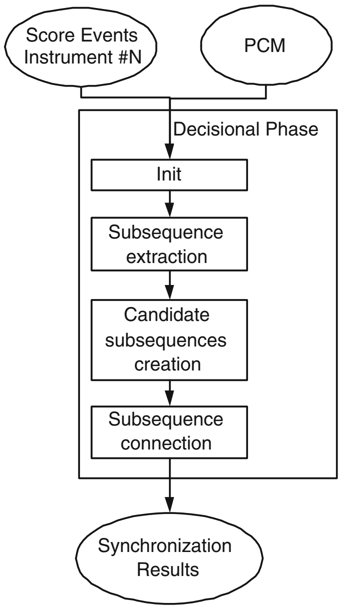

Fig. 6. Decisional Phase Flowchart

- $N_{virtual\_peaks}$ is the number of virtual peaks. They are inserted into the matrix peak when a 0 is found. To avoid bad alignment results, this number must not be over the 35% of subsequence length SS_{size};
- the *tolerance* range is used to evaluate the reliability of retrieved event time at audio level with respect to score event time in MIDI ticks. The default range lies between 0.4 and 0.8.

Then we define a subsequence region by means SS_{size} value and we invoke the **subsequence extraction** function. Aim of this function is to find the more conveniently values for *tolerance* and $N_{virtual\_peaks}$ in order to retrieve all possible valid synchronization events which will be inserted into the candidate subsequence. We may have the following situations:

- we do not find an acceptable subsequence. In this case we increase the tolerance range of one third: $\Delta\_tolerance = \Delta\_tolerance + 1/3$. If this operation is not sufficient, we increase the virtual peak number.
- we find too much acceptable subsequences. In this case we decrease the tolerance range of one third: $\Delta\_tolerance = \Delta\_tolerance - 1/3$.

In case we found a sequence of chords, it is decomposed in all the possible monody sequences.

The output of this function provides a two or three dimension matrix which contains, for each note/chord belonging to the considered subsequence, the possible synchronization events which respect the tolerance. If we get the limit of *tolerance* and $N_{virtual\_peaks}$, and no acceptable subsequences are found, we fill the matrix with 0.

At the end of this phase, the **candidate subsequences creation** function is invoked. Its objective is to select the most reliable synchronization events given in output by **subsequence extraction** function. The candidate subsequence is filled selecting, for each note, the peak which has the lowest distance between the peak time position in seconds retrieved at audio level and the time position in ticks calculated at score level.

Table 3. An example of subsequence connection

CS #1	CS #2	CS #3	CS #4	FS
0.00				0.00
0.50				0.50
1.15	0			1.15
1.91	0			1.91
	0			2.09
	0			2.66
	0	3.25		3.25
		3.54		3.54
		4.25	3.90	4.075
		4.60	4.32	4.46
			4.88	4.88

The final step is the **subsequence connection** where we create the final sequence. Peak for peak, we build the final sequence linking together the candidate subsequences as follow:

- if we have a single candidate event greater than 0, and its value is greater than the previous, this value is inserted into the final sequence;
- if we have two or more candidate events greater than 0, the arithmetic mean of these values is calculated;
- if we have all the candidate events equal to 0, this value is fitted by linear interpolation.

We end the decisional phase generating the synchronization results.

4 Experiments and Results

A prototype of our synchronization algorithm has been implemented in C++. In this section experiments and results are provided. Tests were run on an Intel Pentium IV, 2.6 GHz with 512 MByte RAM under Windows XP SP2.

To perform our objective tests, 31 pieces of music have been considered. They have a length from 30 seconds to 416 seconds.

They can be grouped in:

- polyphonic and polytimbric MIDI pieces;
- polyphonic piano pieces;
- opera pieces (with human voice);
- symphonic pieces (without human voice);
- pop pieces.

The complete list can be found at the website www.lim.dico.unimi.it/parsilist.html.

The results given by ParSi algorithm have been compared with manual synchronization of the same pieces. We provide the objective evaluation in term of percentage of *right measures* (figure 7).

We have a right measure if all synchronization events of the measure are right. A synchronization event retrieved by ParSi algorithm ($se_i^{automatic}$) is right if it does not come before or does not come after the previous and the next synchronization events obtained by manual synchronization (respectively se_{i-1}^{manual} and se_{i+1}^{manual}).

$$se_{i-1}^{manual} < se_i^{automatic} < se_{i+1}^{manual}$$

We can consider this criteria enough for applications where an accurate and precise synchronization is not required.

In order to obtain the best synchronization results, the selected instrument has been selected in order to get available the most score events number. Furthermore, when possible, we have selected the piano instrument because the state of the art shows (see section 2) it is generally easier to synchronize.

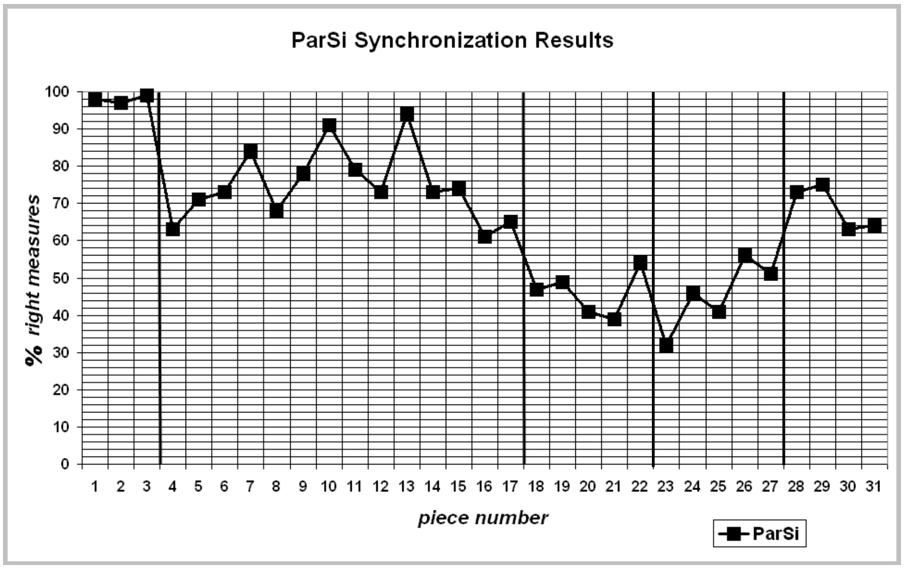

Fig. 7. ParSi Synchronization Results

Experiments show that ParSi algorithm synchronizes correctly (96% - 100%) polyphonic and polytimbric MIDI pieces (piece numbers from 1 to 3).

ParSi synchronization of polyphonic piano pieces (numbered from 4 to 17) gives interesting results, with a percentage of success between 60% and 95%.

Opera pieces (numbered from 18 to 22) have a worse synchronization precision (35% - 60%) than piano pieces because of the more complexity of audio signal. We have a similar situation with symphonic pieces (pieces numbered from 23 to 27). Here, we have the worst case for the piece number 23 ('Arabic Dance' from *The Nutcracker* by Tchaikovsky) because of several repeated notes which generate similar time peaks.

Finally, pop music (pieces numbered from 28 to 31) provides a interesting percentage of success which lies between the and 55% and 70%.

If we compare these results with other works which requires a complete score ([8] [9] [10] [11] [12] [14]), we can state ParSi provides good automatic synchronization results with a partial score representation.

5 Conclusions

In this work we have proposed an algorithm to automatic synchronize different PCM audio performance to a partial music score codified in MIDI. We use a notch filter during audio analysis and a set of conveniently parameters to perform the decisional phase. Of course, the instrument selection is the most critical decision and it could deeply affect the quality of synchronization results. An instrument with the most number of events is recommended.

Analyzing the experimental results we can state that the ParSi approach provides interesting results, even if a partial score is used. The pop music test is the more interesting experiment because, generally, incomplete scores are available. The synchronization results of opera and symphonic pieces should be improved; in fact future works are mainly addressed to algorithm improvement for a better time-alignment of that music genres.

References

1. Stephen Downie, J.: Music information retrieval. Annual Review of Information Science and technology, ch. 7, vol. 37, pp. 295–340. Blaise Cronin, Medford (2003)
2. Wiering, F., Veltkamp, R.C., Typke, R.: A survey of music information retrieval systems. In: 6th International Conference on Music Information Retrieval, ISMIR 2005, London, UK, September 11-15, pp. 153–160 (2005)
3. Foote, J.: An overview of audio information retrieval. Multimedia Systems 7(1), 2–10 (1999)
4. Clausen, M., Kurth, F., Müller, M., Ribbrock, A.: Content-based retrieval in digital music libraries. In: Heery, R., Lyon, L. (eds.) ECDL 2004. LNCS, vol. 3232, pp. 292–303. Springer, Heidelberg (2004)
5. Bainbridge, D., Dewsnip, M., Witten, I.H.: Searching digital music libraries. In: Lim, E.-p., Foo, S.S.-B., Khoo, C., Chen, H., Fox, E., Urs, S.R., Costantino, T. (eds.) ICADL 2002. LNCS, vol. 2555, pp. 129–140. Springer, Heidelberg (2002)
6. Schwarz, S., Orio, D., Lemouton, N.: Score following: State of the art and new developments. In: Proceedings of the Conference on New Interfaces for Musical Expression, pp. 36–41 (2003)
7. Dannenberg, R.B., Hu, N.: Polyphonic audio matching for score following and intelligent audio editors. In: Proceedings of the 2003 International Computer Music Conference (2003)
8. Arifi, V., Clausen, M., Kurth, F., Muller, M.: Automatic synchronization of music data in score-, midi- and pcm- format. In: 4th International Conference on Music Information Retrieval, ISMIR 2003 (2003)
9. Hewlett, W., Selfridge-Fields, E. (eds.): Automatic Synchronization of Musical Data: A Mathematical Approach. Computing in Musicology. MIT Press, Cambridge (2004)
10. Dixon, S., Widmer, G.: Match: A music alignement tool chest. In: 6th International Conference on Music Information Retrieval, ISMIR 2005 (2005)
11. Muller, M., Kurth, F., Roder, T.: Towards an efficient algorithm for automatic score-to-audio synchronization. In: 5th International Conference on Music Information Retrieval, ISMIR 2004 (2004)
12. Soulez, F., Rodet, X., Schwarz, D.: Improving polyphonic and poly-instrumental music to score alignment. In: 4th International Conference on Music Information Retrieval, ISMIR 2003, pp. 143–148 (2003)
13. Turetsky, R.J., Ellis, D.: Ground-truth transcriptions of real music from force-aligned midi syntheses. In: 4th International Conference on Music Information Retrieval, ISMIR 2003 (2003)

14. D'Aguanno, A., Vercellesi, G.: Automatic synchronisation between audio and score musical description layers. In: Falcidieno, B., Spagnuolo, M., Avrithis, Y., Kompatsiaris, I., Buitelaar, P. (eds.) SAMT 2007. LNCS, vol. 4816, pp. 200–210. Springer, Heidelberg (2007)
15. Rabiner, L.R., Juang, B.H.: Fundamentals of Speech Recognition. Prentice-Hall, Englewood Cliffs (1993)
16. Oppenheim, A.V., Schafer, R.W.: Discrete-time signal processing. Prentice-Hall, Englewood Cliffs (1989)