Tempo Map Retrieval from the MIDI Clock Stream

Lutshayzar Gueorguieff[1] and Peter Antonov[2]

Abstract—**This paper contains algorithm flowchart, short description, and detailed test results and error statistics of a method for MIDI tempo map retrieval on MIDI-recording with external MIDI-clock synchronization. The test results prove that it is possible to overcome an annoying deficiency of the MIDI standard – the lack of a Universal real-time message system exclusive message providing information for the current tempo.**

Keywords—**MIDI, clock, tempo, synchronization, recording**

I. Introduction

There are three basic ways to convert a MIDI sequencer file from a proprietary format to a Standard MIDI File (SMF) – direct conversion to a SMF, a MIDI File Dump [1], and "synchronized overplay" [2]. Direct conversion (e.g. via the Roland MRM-500 OS or the converters by Gary Giebler) seems to be the most straightforward. But each disk and file format requires its own conversion program, and format descriptions are available for insiders only – others must rely on reverse engineering to obtain one. This means that there is no guarantee that all data are identified and processed correctly. The MIDI File Dump standard is not applicable for sequencers either, as units that do not support the SMF format do not support it either. So, synchronized overplay remains the only option. It is universal, as it does not depend on the sequencer brand and model. But unlike MIDI data, retrieval of Meta data (including tempo data) is impossible this way. Neither there is a Universal real-time system exclusive message that would provide information about the current tempo. This is a great problem for sequences with complex tempo maps that are completely lost on synchronized overplay. But the tempo map could still be retrieved on synchronized overplay. This paper describes how it can be done and what results can be expected, based on real world tests.

Actually, the method was already announced in a paper [3] by one of the authors, but only in Bulgarian, with some inaccuracies, a C-specific description, and few test results. This paper addresses these shortcomings with an algorithm flowchart and much more detailed test results, including detailed statistics.

II. Principle of Operation

Two programmable timers are required, both clocked by the system clock, divided with a fixed ratio. Timer #1 measures the MIDI clock byte period. Timer #2 can be regarded as the programmable divider of a virtual "frequency synthesizer". Its division ratio is proportional to the MIDI clock period (a fine frequency adjustment). Its output is compared with the MIDI clock stream by a virtual "phase detector" (implemented in software), which does a rough phase adjustment.

III. Algorithm

Fig. 1 presents the flowchart of the algorithm. Blocks 1–4 contain an excerpt from the second level received data interrupt handler. It gets executed only if the received byte is a MIDI clock byte. The *oldTick* counter registers the number of timer interrupts already processed on receiving the byte. *MAX_TICK* is the number of interrupts per crotchet (120), and *CLKS_PER_QUARTER* is the number of MIDI clock bytes per crotchet (always 24). So, 5 timer interrupts occur between each two MIDI clock bytes, and *oldTick* gets incremented by 5 too.

In block 5, the *TickAdjust* global variable registers the sign of the rough (phase) adjustment, which is done in blocks 5–9 on every timer interrupt, if *TickAdjust* is nonzero. The adjustment is disabled at the end of the system exclusive messages (block 2). The fine (frequency) adjustment is done in block 4. The *total* variable holds the number of timer clocks between each [sub]beat (time signature denominator), obtained through the *measureClock()* function. This function (not given here) uses timer #1 to measure the MIDI clock period and initializes timer #2 with a value five times less than the measured one. Thus, there are always 5 interrupts of timer #2 between two consecutive MIDI clocks.

Later, *total* is used to initialize the *Tempo* global variable.

Blocks 5–9 present the main part of the *newTick()* function for updating the timer interrupt counter *TimerTick*. This function is called on each interrupt of timer #2. It does the rough (phase) counter adjustment too. This is done with only one step forwards or backwards in order to avoid sharp changes, which would lead to generation of superfluous Tempo Meta events.

Blocks 10–12 show an excerpt from the *record()* function. If the tempo difference for each [sub]beat (denominator *d* of the time signature) is greater than the *dTempoMax* threshold (2%, the smallest instant tempo change the human ear perceives [4]), a Tempo event is generated. Its value (in µs per crotchet) is

$$t = \frac{d \cdot 1000000}{4 \cdot f_T} \cdot \left(oldTempo + \frac{dtempo}{tempos} \right) \qquad (1)$$

where f_T is the timer clock frequency, *oldTempo* is the previous *Tempo* value, *dtempo* is the accumulated error, and *tempos* is the Tempo event counter. The event is generated at the time of the last detected tempo change *tempoTickDelta*. After that, this time value, the tempo and the threshold are up-

[1]Lutshayzar Gueorguieff is with the Faculty of Computing and Automation at the Technical University of Varna, Studentska Str. 1, Varna 9010, Bulgaria. E-mail: lig@tu-varna.bg

[2]Peter Antonov is with the Faculty of Computing and Automation at the Technical University of Varna, Studentska Str. 1, Varna 9010, Bulgaria.

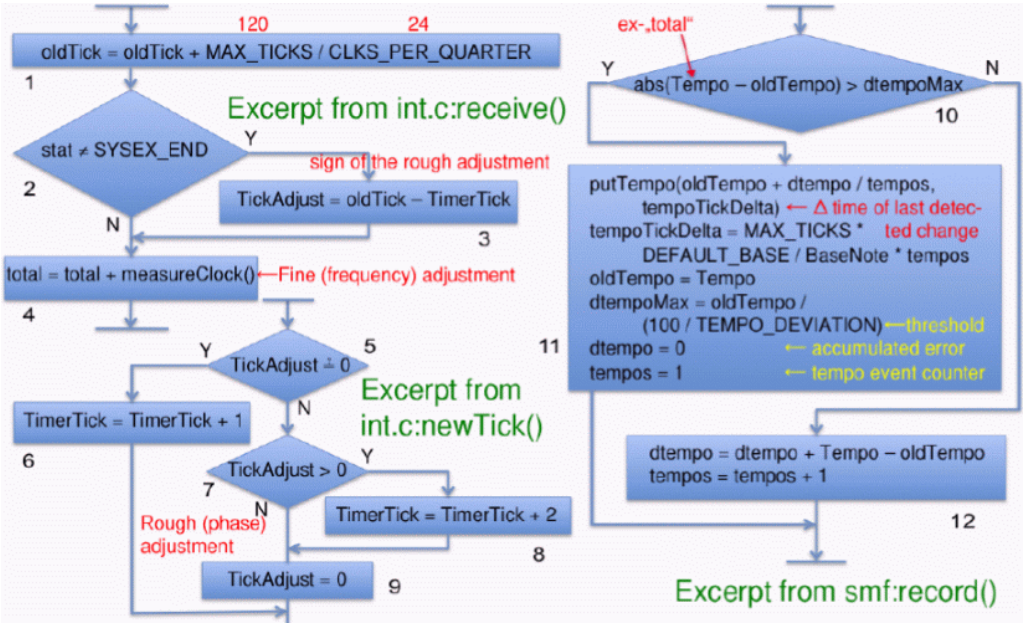

Fig. 1. Flowchart of the tempo map retrieval algorithm. Computational complexity: O(1), as there are no loops (also in the external functions).

dated, and the error and the tempo counter – initialized. Else (if the aforementioned tempo difference is less), the error just gets accumulated and the counter – updated. The last Tempo event is generated at the end of the recording (not shown here). So, only one Tempo event is thus generated for pieces having a constant tempo.

IV. Experimental results

The next figures show the results and statistics from the tempo map retrieval using the algorithm described above when doing synchronized overplay for several MIDI sequences.

1) «Arabesque»:

This is a classical piece for piano with tempo rubato, changing on each crotchet. Fig. 2 shows its original tempo map and the retrieval errors for this piece, aligned by crotchets. The errors result mostly from the imperfect playback timing, as the playback program in this and all other cases but one runs in Windows, where playback latency is considerable and variable.

2) «Clair de lune»:

This is very slow classical piece for piano, also with tempo rubato, changing on each crotchet. Fig. 3 shows its original tempo map and the retrieval errors for this piece, aligned by crotchets. The errors are more than an order of magnitude smaller because the sequence is played under "pure" MS-DOS.

Fig. 4 presents the error distribution for the first two pieces. It can be seen that the sharp tempo changes in both lead to very precise tempo retrieval as the 2% threshold is always exceeded.

3) "Baklava":

This is a fully orchestrated Russian folk song with a dynamic tempo. Unlike the first 2 pieces, the tempo changes here are so gradual that the 2% threshold is not always exceeded. This leads to fewer retrieved tempo events, and temporary errors (see measures 49–61 and especially 72 on fig. 5, top).

4) "Czardas":

This is an excerpt from the operetta "Die Csárdásfürstin" with a very dynamic tempo. The recovered tempo events are almost as many as the original ones, yet there are 2 false tempo events on measures 65 and 121 due to the aforementioned variable latency when playing MIDI in Windows (fig. 5 bottom, 6).

III. Conclusion

The tests confirm the accuracy and robustness of the tempo map retrieval algorithm, especially for sharp tempo changes. The induced error is way under the aural perception threshold. So, all the temporal information with musical meaning can thus be recovered on synchronized overplay, combining the advantages of MIDI Time Code and MIDI clock synchronization.

References

[1] MIDI Manufacturers' Association, The Complete MIDI 1.0 Detailed Specification, Version 96.1, 2nd Edition, Los Angeles, November 2001.

[2] Nathorst-Boos, E., "MIDI Xplained", Steinberg Hard- und Software GmbH, Hamburg 1993 ("Hyperbok om MIDI", Synkron musik & datorer, Stockholm 1991).

[3] Георгиев, Л., „Метод за автоматично възстановяване на картата на темпото при запис със синхронизация по MIDI-часовник", Национална конференция с международно участие „Телеком '97", т. I, Доклади на български език, 1998 г., стр. 574–580.

[4] Jones, M., Fay, R., Popper, A., "Music Perception", Springer Handbook of Auditory Research, Vol. 36, 2010, pp. 178–179.

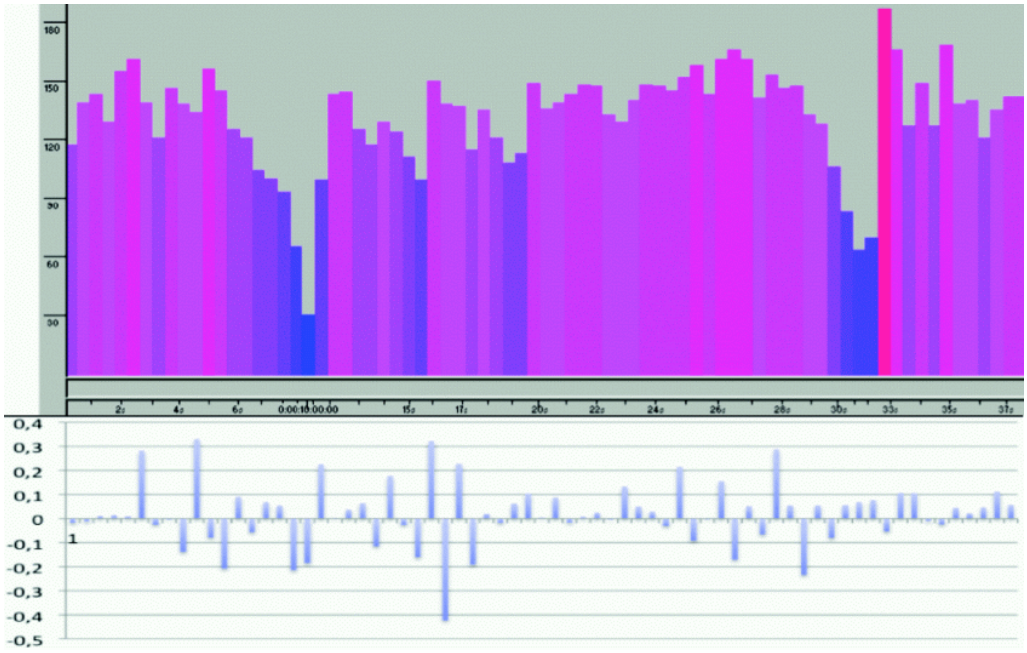

Fig. 2. Original tempo map (top) and retrieval errors [%] (bottom) for «Arabesque». Average error: 0.019%, standard deviation: 0.132%.

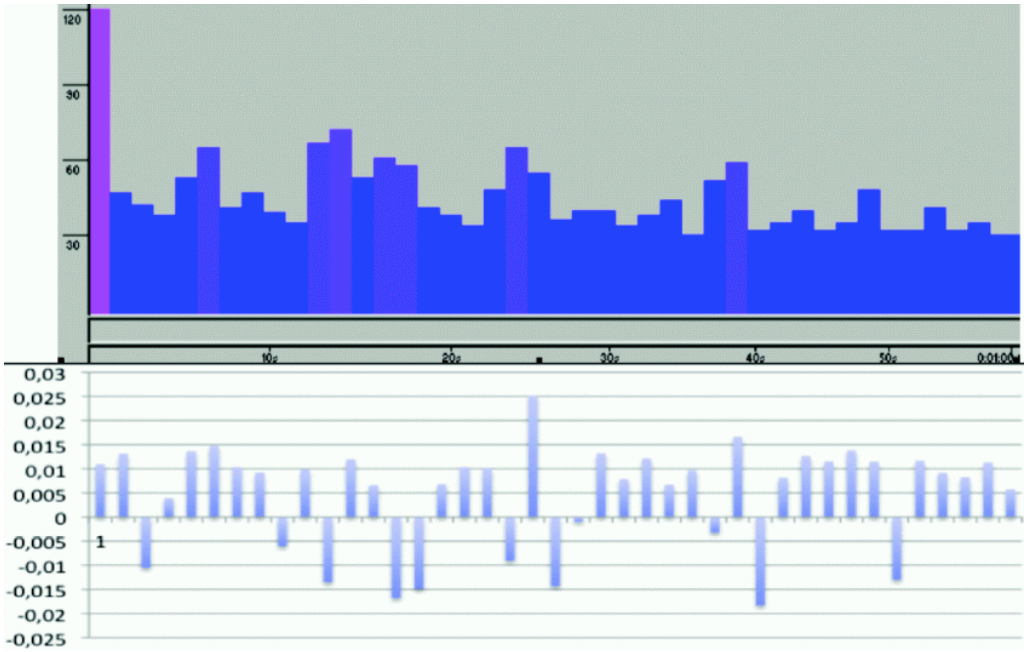

Fig. 3. Original tempo map (top) and retrieval errors [%] (bottom) for «Clair de lune». Average error: 0.005%, standard deviation: 0.011%.

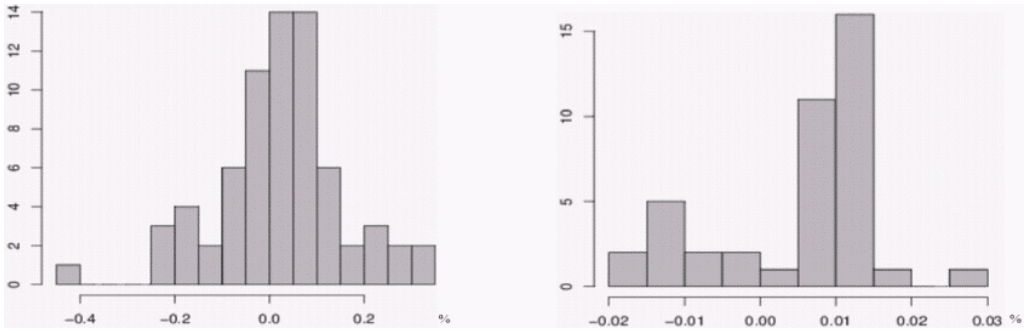

Fig. 4. Error distribution [frequency of occurring] for «Arabesque» (left) and «Clair de lune» (right).

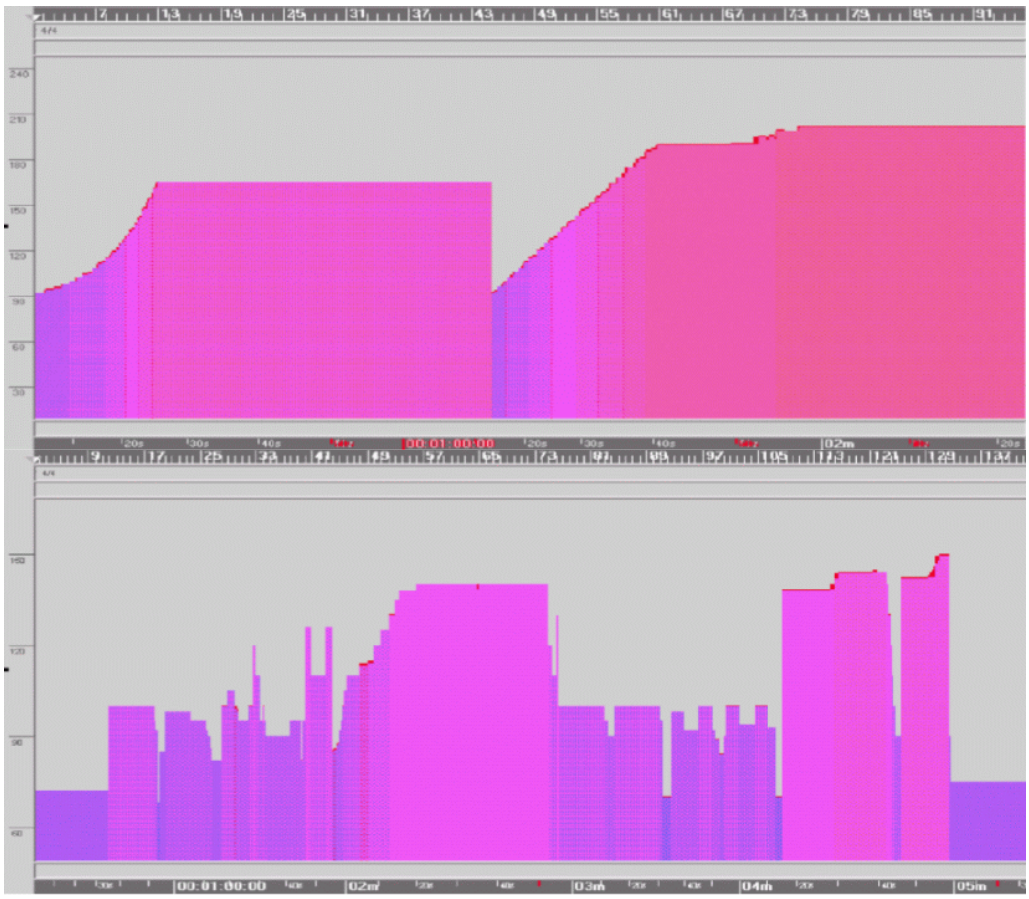

Fig. 5. Differences (in bright red) between the original and the recovered tempo map for “Baklava” (top) and “Czardas” (bottom).

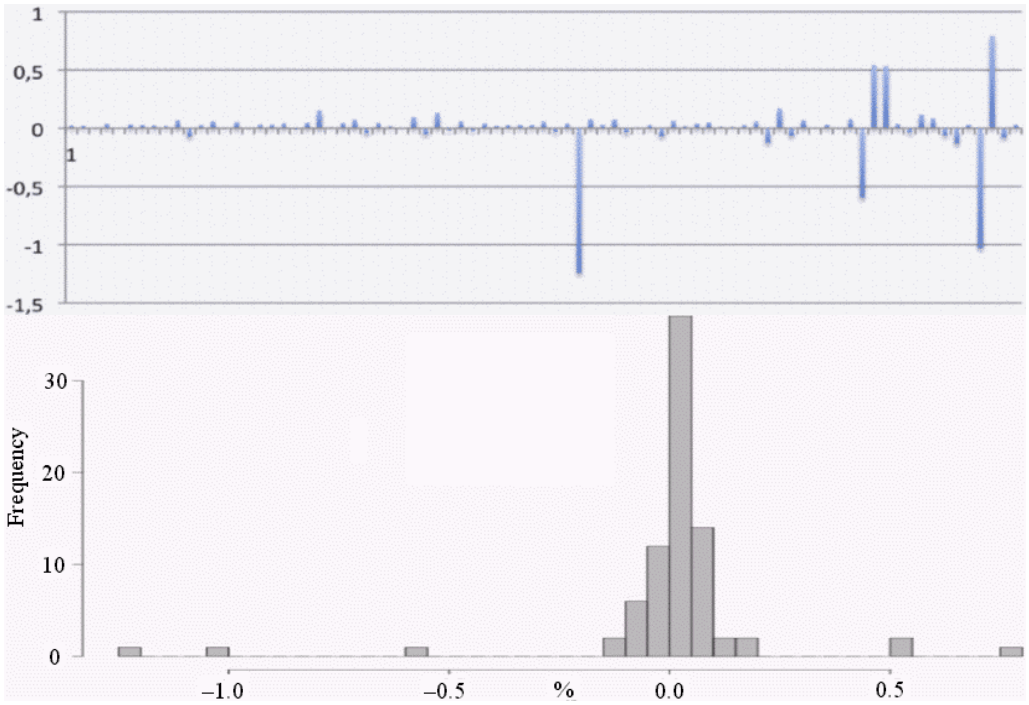

Fig. 6. Errors (top) and their distribution (bottom) for “Czardas”. Average error: 0.009%, standard deviation: 0.234%. Major errors coincide with fig. 5, bottom. (Note that the abscissa of the errors is ticked off in tempo events, unlike the one of the tempo map, ticked off in measures.)