Towards Effective ‘Any-Time’ Music Tracking

Andreas Arzt$^{(1)}$ and **Gerhard Widmer**$^{(1,2)}$ [1]

Abstract. The paper describes a new method that permits a computer to listen to, and follow, live music in real-time, by analysing the incoming audio stream and aligning it to a symbolic representation (e.g, score) of the piece(s) being played. In particular, we present a multi-level music matching and tracking algorithm that, by continually updating and evaluating multiple high-level hypotheses, effectively deals with almost arbitrary deviations of the live performer from the score – omissions, forward and backward jumps, unexpected repetitions, or (re-)starts in the middle of the piece. Also, we show that additional knowledge about the structure of the piece (which can be automatically computed by the system) can be used to further improve the robustness of the tracking process. The resulting system is discussed in the context of an automatic page-turning device for musicians, but it will be of use in a much wider class of scenarios that require reactive and adaptive musical companions.

1 Introduction

Computers that can listen to music and follow it in real time promise to be useful in a wide range of applications, from synchronisation tasks to automatic monitoring of radio stations or web streams, from live music visualisation in artistic contexts to real-time accompaniment of soloists. A specific application example was given in [2], where an *automatic sheet music page turner* (a real mechanical device) for musicians is controlled by a computer that listens to the musicians (e.g., a pianist), aligns the incoming audio stream to an internal (audio) representation of the printed score of the music, follows the musician through the performance and autonomously decides when to trigger the page turner. Such a system would be especially useful during practicing, where instrumentalists generally don’t have human page turners at their disposal[2] and also need to focus on the sheet music, as they have not yet memorised the piece.

For the purposes of this paper, then, we define *‘music tracking’* as follows: for a machine to listen to live music through a microphone, to identify and track the corresponding positions in the printed score, and, in this way, to always ‘know’ where the musicians are in the piece, even if the live performance varies in tempo and sound and perhaps even deviates from the score in certain places. In this paper, we present a new, extremely robust algorithm for score-based music tracking, where the live music takes the form of an audio stream, and the system has an audio representation of the printed score of the piece being played (produced using synthesiser software). Thus, the tracking problem is one of real-time audio-to-audio alignment.

The specific *goal* we wish to achieve with our algorithm is robustness in the face of almost arbitrary disruptions or re-starts within a performance, where current music tracking algorithms would simply ‘get lost’. For instance, in a practicing scenario this will be the norm: the musician will want to repeat difficult passages an arbitrary number of times, stop after a mistake and re-start somewhere, skip some parts, or take a break and then re-start somewhere in the middle of the piece – all of these without having to tell the system the precise starting position each time.

The two *key aspects of the solution* we propose are: (1) a two-level hypothesis tracking process, with a high-level tracker constantly evaluating *all possible* (!) positions in the score as potential ‘current points’, and more detailed hypothesis evaluators checking, selecting from, and refining these hypotheses – all in real time; and (2) the additional introduction of knowledge about the structure of the piece being played (which, as we will show, can be computed by the system itself), which facilitates heuristic guesses as to the most likely points of continuation.

As the experiments will show, the resulting system is indeed capable of quickly reacting to jumps, omissions, or performers starting at arbitrary points in the middle of a piece. For lack of a better term, we chose to call this *‘Any-time Music Tracking’*: continuously being ready to receive input and to revise one’s hypothesis as to what the performers are doing.

2 Related Work

While there has been quite some work on real-time music tracking, starting as early as 1984 [4, 10] and including some recent publications on very advanced systems (e.g. [3, 9]), the problem of how to deal with changes to the structure of a piece on-line has so far been largely neglected (in contrast to the off-line case, see [6]), with two notable exceptions.

In [8] HMMs are used to model the structure of the piece. This is a very static approach as only deviations which are modeled beforehand can be detected. The same applies for the method devised in [2], where multiple instances of matching algorithms are started at predefined positions (e.g. notated repeats) to detect deviations via the alignment costs. The finding that it is possible to recognise which instance of the matching algorithm is working at the correct position by comparing their alignment costs sets the ground for our new and more dynamic approach.

3 Preliminaries: Data Representation and Basic Matching Algorithm

3.1 Data Representation

Rather than trying to transcribe the incoming audio stream into discrete notes and align the transcription to the score, we first convert

[1] $^{(1)}$ Department of Computational Perception, Johannes Kepler University, Linz, Austria; $^{(2)}$ Austrian Research Institute for Artificial Intelligence, Vienna, Austria

[2] Note, by the way, that music page turners are highly trained and specialised professionals!

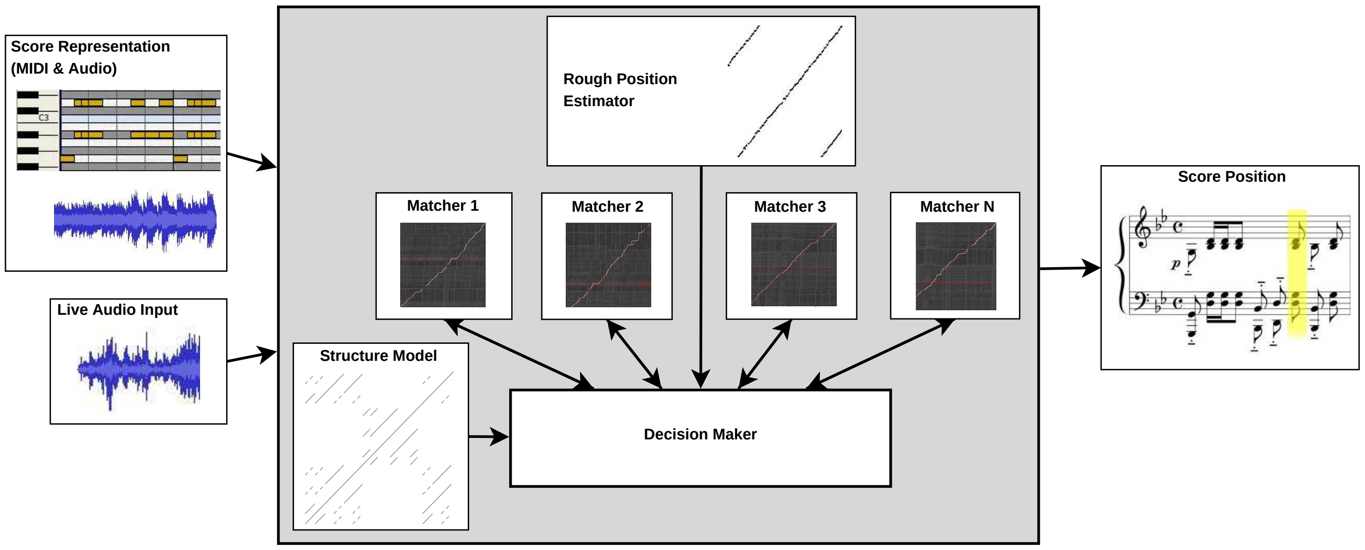

Figure 1. Overview of our 'Structure-aware Any-Time Music Tracking System' showing the different modules and the data flow.

a MIDI version of the given score into a sound file by using a software synthesizer. Due to the information stored in the MIDI file, we know the time of every event (e.g. note onsets) in this 'machine-like', low-quality rendition of the piece.

The audio streams to be aligned are represented as sequences of analysis frames, using a low-level spectral representation, at two different resolution levels. This approach and the way the lower-resolution features are computed are very much inspired by [7].

The high-resolution features are computed via a windowed FFT of the signal with a hamming window of size 46ms and a hop size of 20ms. The data is mapped into 84 frequency bins, spread linearly up to 370Hz and logarithmically above, with semitone spacing. In order to emphasize note onsets, which are the most important indicators of musical timing, only the increase in energy in each bin relative to the previous frame is stored.

The low-resolution features are computed by convolving the sequence of high-resolution features component-wise with a Hann window with length 30 (or 600ms). This new sequence is down-sampled by a factor of 15, resulting in a hop-size of 300ms, and each vector is normalized to sum up to 1.

For both feature resolutions the cost of aligning two feature vectors is computed as the Euclidean distance of two vectors. While the feature extraction from the live audio signal can of course only be done on-line, the feature extraction from the score representation is done beforehand, so that information about the whole piece of music is available during the matching process.

3.2 On-line Dynamic Time Warping (ODTW)

This algorithm is the core of our real-time audio tracking system; slight variants of it are used at both levels of our system. ODTW takes two time series describing the audio signals – one known completely beforehand (the score) and one coming in in real time (the live performance) –, computes an on-line alignment, and at any time returns the current position in the score.

ODTW is based on the original DTW algorithm, which works as follows: Given 2 time series $U = u_1, ..., u_m$ and $V = v_1, ..., v_n$, an alignment between U and V is a path $W = W_1, ..., W_i$ (through a cost matrix) where each W_k is an ordered pair (i_k, j_k) such that $(i, j) \in W$ means that the points u_i and v_i are aligned. W is constrained to be monotonic and continuous. A local cost function $d(i, j)$ assigns costs to the alignment of each pair (u_i, v_i). The cost of a path W is the sum of the local alignment costs along the path. The $m \times n$ path cost matrix D is computed using the recursion:

$$D(i,j) = d(i,j) + min \left\{ \begin{array}{c} w_a * D(i, j-1) \\ w_a * D(i-1, j) \\ w_b * D(i-1, j-1) \end{array} \right\} \quad (1)$$

$D(i, j)$ is the cost of the minimum cost path from $(1, 1)$ to (i, j), $D(1, 1) = d(1, 1)$, $w_a = 1$ and $w_b = 2$. The weights w_a and w_b are used to normalize paths of different lengths to make them comparable. After the computation of the matrix the path itself is obtained by tracing the recursion backwards from $D(m, n)$.

Originally proposed by Dixon in [5], the ODTW algorithm is based on this algorithm, but has two important properties making it useable in real-time systems: it has linear time and space complexity and the alignment (a *'forward path'*) is computed incrementally.

For every iteration the number of cells calculated is given by a search width parameter $c = 500$, e.g. for a new column i the local distances $d(i, j-c), d(i, j-(c-1)), ..., d(i, j)$ are calculated, where j is the index of the current row. The calculation of the minimum cost paths using formula 1 is restricted to using only calculated cells, thus reducing time and space complexity from quadratic to linear.

During every iteration the minimum path cost matrix is expanded by either calculating a new row or a new column. Calculating a row (column) means incrementing the pointer to the next element of the respective time series, calculating the new local distances within the defined search width c, and updating the cost matrix D by using formula 1.

To decide if a row or a column should be computed (i.e., which of the two time series to advance), the minimum path cost for each cell in the current row j and column i is found. If this occurs in the current position (i,j) both a new row and column are calculated. If this occurs elsewhere in row j a new row is calculated and if this occurs elsewhere in column i a new column is calculated. If one time series has been incremented more than $MaxRunCount = 3$ times, the other series is incremented. In our musical setting, this embodies the assumption that a given performance will not be more than 3 times faster or slower than the reference score, and prevents the alignment algorithm from 'running away' too far.

At any time during the alignment it is possible to compute a *'backward path'*, i.e. a path starting at the current position and computed backwards in time, leading to an off-line alignment of the two time series which generally is much more accurate.

Improvements to this algorithm, focusing both on adaptivity and robustness, were presented in [2] and are incorporated in our system. That includes the 'backward-forward strategy', which reconsiders past decisions and tries to improve the precision of the current score position hypothesis, and the utilization of onset information derived from the MIDI score.

Both of these strategies aim at an improvement in alignment precision. In the present paper, we provide a more dynamic and more general solution to the third problem discussed in [2], namely the problem of how to deal with structural changes effectively on-line.

4 'Any-time' Music Following via Two-level Tracking

While previous systems – if they did deal at all with serious deviations from the score – had to rely on explicitly provided information about the structure of a piece of music and points of possible deviation (e.g., notated repeats, which a performer might or might not obey), our system does without any such information and continuously checks all (!) time points in the score as alternatives to the currently assumed score position, thus theoretically being able to react to arbitrary deviations (jumps etc.) by the performer. The overall structure is shown in Figure 1. A *'Rough Position Finder'* continually evaluates every possible score position by computing a very rough alignment of the last few seconds of the live audio input to the score at that point; that is done at the level of low-resolution features and $600ms$ frames. A *'Decision Maker'* takes these candidate score positions and tries to verify or falsify them via *several parallel ODTW matchers* at the detailed, high-resolution level. At any time one instance of the matching algorithm is marked as 'trusted', representing the system's current belief about the correct position. A model of the structure of the piece, which is computed automatically, proves to be very useful during this process, as will become clear later.

4.1 Rough Position Estimator (RPE)

The task of the Rough Position Estimator (RPE) is to continually provide our system with rough but reliable hypotheses about possible current positions in the score. It takes two low-resolution time series (with frame size $600ms$ and hop time $300ms$; thus a piece of, say, 10 minutes will consists of 2,000 such frames) as input. At any time the RPE has a set of hypotheses about possible score positions. The number of hypotheses is not limited, but they are ordered by a value representing the 'probability' that the score position in question is corresponding to the current position in the live input.

For every new incoming low-resolution audio frame the RPE is updated in the following way. First the local 'low resolution' cost matrix, which contains the Euclidean distances of the frames of the live input series to every frame of the score representation, is updated. As the live input stream sits on the x-axis this means adding a new column to the matrix.

Next, starting in every new cell rough backward alignments on the last $n = 9$ seconds of the live performance are computed using a greedy local search algorithm constrained to step at most 3 times in a row into a single direction. The alignment costs of the resulting paths are normalized by the number of steps and then scaled to lie between 0 and 1. The result is a vector giving the similarity values of rough alignments of the last n seconds of the live audio to score passages ending at any possible score position (with a resolution of 600ms).

To reduce the computational complexity and to add some robustness to the algorithm, only the cells ('score positions') with a similarity value of at least 0.95 are selected as possible hypotheses to be passed on to the lower-level matchers for evaluation (see 4.2 below).

As only the last n seconds on the live input axis are needed and due to the low resolution of the two streams, the necessary computations are easily done in real time, even for pieces as long as 30 minutes.

4.2 Detailed Matching Algorithm (DMA)

Our system maintains a fixed number of instances of the ODTW matching algorithm, working on the high-resolution features (frame size $46ms$, hop time $20ms$ – thus a piece of 10 minutes consists of $30,000$ frames), which are managed by the 'Decision Maker' (see below). The number of matcher instances is limited by the computing power of the computer in use, but usually 3 or 4 instances are sufficient. The matching algorithm is identical to the one described in 3.2 above, including all extensions proposed in [2] with one notable difference: In [1] we introduced a simple reactive *tempo model* based on the performed tempo relative to the score representation, computed over the last couple of seconds of the live performance. This model is used to stretch or compress the score representation accordingly and therefore reduce differences in absolute tempo between the score representation and the live performance. This increases robustness in the face of global and local tempo deviations.

4.3 Decision Maker (DM)

The Decision Making Component (DM) decides on which hypotheses to investigate in detail, and which to accept or reject. To achieve this it manages instances of the DMA of which at any time at least one is active. One of the active instances is marked as 'trusted', representing the system's current belief about the score position.

If there is an idle DMA instance available, the first step in every iteration is to analyse the output of the RPE and to find the best hypothesis for which further investigation is not in progress. The idle DMA instance is initialised accordingly, but not with the current time points in score and live audio, but with points a few seconds in the past – the end positions of the rough backward alignment leading to this hypothesis, as computed by the RPE. The DMA then starts aligning there. That gives the DMA more context for evaluating the current hypothesis and accelerates the DM's decision making process (because it will not have to wait for several seconds of new live audio until the hypothesis can be evaluated).

Next, the DM iterates through the instances of the DMA, trying to decide, for each matcher, whether to reject it, to accept it as the new 'trusted' one, or to postpone the decision. Only matching algorithms that have tracked at least 5 seconds of audio input are considered here, as of course no decision should be made without sufficient grounding. A matcher may need some time to converge on the correct alignment path as the RPE's hypotheses are only rough estimates.

The decision itself is made as follows: for each matcher i, the difference d_i between the current alignment cost of matcher i and the alignment cost c_t of the currently 'trusted' matcher is computed; these differences are summed up over the whole lifetime of the matcher. After every iteration the sums are compared to two heuristic thresholds. If the sum is lower than $t_r = -rc_t$ the hypothesis is rejected and the matcher is set to idle. If the sum exceeds $t_a = sc_t$, the hypothesis is accepted and the matcher is marked as the new 'trusted' one. In our system we use $r = 2$ and $s = 10$, which gives a good trade-off between fast response times and overall robustness.

5 Adding Musical Knowledge

Music is a highly structured artifact, and most music exhibits a certain degree of repetition – a significant motif may reappear many times, a refrain is a repeated part by definition, whole sections are sometimes repeated. Thus, we generally have repeating passages at many levels – from short segments of just a few beats, to segments lasting several minutes.

Repetition in music presents a problem to a music following system that always considers all possible positions in the score: which of the instance of a repeating pattern corresponds to the passage currently played by the performer? In order to cope with this problem, we introduce knowledge about the structure of the current piece, by way of a *Structure Model (SM)* that identifies repeated parts at all levels and builds an equivalence relation that specifies which parts are instances of the same class. As shown in 6.2 below, without such knowledge it is not possible to distinguish these parts and make musically sensible decisions about the current position in the score. Furthermore the SM also adds to the robustness of our system.

The SM is extracted automatically from the score by computing a *self-similarity matrix* based on the rough feature representation of the score. In this matrix the diagonal represents a performance of the piece as annotated in the score. Simple image processing techniques are used to identify connected lines with maximum similarity which are parallel to the diagonal. These represent repetitions of the same part at another time in the score (see Figure 1, which shows the structure model computed for the Impromptu in A-flat major D.935-2 by Franz Schubert).

This additional knowledge proves useful for several tasks in our system: It is used to (1) prune the list of hypotheses by identifying equivalent hypotheses and discarding all hypotheses equivalent to positions that are already checked in detail. This greatly reduces computation time and gives the system the ability to track more different hypotheses at once. Furthermore, the DM is used to (2) control the pruning process by introducing a configurable matching strategy. Therewith it is possible to affect the behaviour of the system, e.g. by using the strategy 'always prefer the shortest jump in the score'. And (3) we use information from the model to double the acceptance threshold t_a if an active hypothesis supports (i.e., belongs to the same class as the segment processed by) the 'trusted' matcher, which add to the robustness of the system.

6 Evaluation

Our system was evaluated on 3 classical piano pieces: the Valse brillante Op. 34 No. 1 by Frédéric Chopin, the Impromptu in A♭ major D 935-2 by Franz Schubert and the Prelude in G minor Op. 23 No. 5 by Sergei Rachmaninoff. For each piece we selected as least 5 different performances by well-known classical pianists, including e.g. Vladimir Horowitz, Evgeny Kissin, and Vladimir Ashkenazy. The pieces are of considerable complexity, with a rich structure of repetitions at various levels and, belonging to the romantic piano genre, they are generally played with a lot of expressive freedom (in terms of tempo changes etc.), which is a challenge to the ODTW matching algorithms. The experiments were performed off-line but, except for a very small latency, the results are the same for on-line alignments.

For a quantitative evaluation a correct reference alignment is needed. As the goal of our experiments was not to evaluate the overall accuracy of the alignments – this was already done in [2] – but the performance of the multi-level approach in reacting to structural changes to the piece, data acquired by off-line alignments was sufficiently accurate as ground truth for this task. Manual examination guaranteed that no large errors were included in the reference alignments. As shown in [5] these alignments are generally very accurate.

6.1 Evaluation of the 'Rough Position Estimator'

The overall performance of our system depends heavily on the accuracy of the RPE. If its hypotheses about possible score positions are poor, the whole system will rarely detect jumps during a performance or, even worse, may get confused even in the course of a 'normal' performance. The RPE, which at any time gives an ordered list of possible current score positions, was first evaluated in isolation, by analysing the output at every note onset in our test set. For every output the minimum distance to the actual score position or an equivalent position (e.g. due to a repetition) was computed.

As Table 1 shows, the performance of the RPE is sufficient for the task we have in mind. In general, due to the robustness of the ODTW matching algorithm, even hypotheses with an error of up to 5 bars are useable as a starting point for our detailed matchers.

Error ≤	**Selecting 'best' of top n hypotheses**				
	1	**2**	**3**	**4**	**5**
0 bars	41%	56%	58%	60%	60%
1 bar	70%	78%	83%	85%	86%
2 bars	78%	83%	87%	89%	90%
3 bars	80%	85%	89%	91%	92%
4 bars	82%	87%	90%	92%	93%
5 bars	83%	87%	91%	93%	94%

Table 1. Cumulative percentages of the best of the top n proposed hypotheses of the RPE with an error up to the given value – e.g. cell (3,2) should be read as 'at 83% of all note events in the performances, the best of the top 3 hypotheses proposed by the RPE is at most 1 bar away from the correct position (or an equivalent position according to the structure model)'.

6.2 Evaluation of the Any-Time Music Tracker

In the following the basic system without the SM will be referred to as *'Any-Time Music Tracker' (AMT)*, and the system including the SM will be referred to as *'Structure-aware Any-Time Music Tracker' (SAMT)*. The evaluation of the systems is based on versions of the performances in our data set which were edited by removing parts of various length, ranging from 10 to 250 bars.

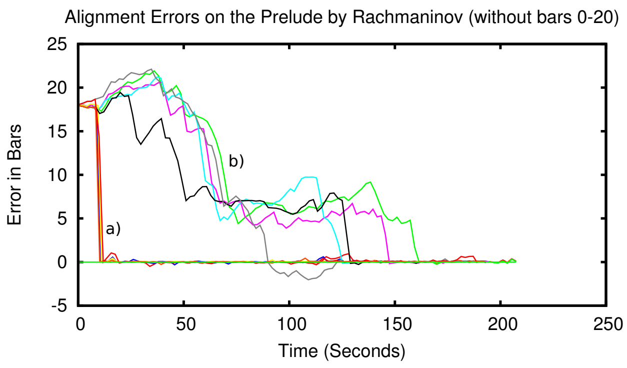

Figure 2. 'Starting in the middle': A visual comparison of the capabilities of the old and the new AMT real-time tracking system. 5 performances of the Prelude by Rachmaninov, with bars 0-20 missing, are aligned to the score by the old and the new system. For all performances, the new one (a) almost instantly identifies the correct position, while the old one (b)finds the correct position by mere chance.

6.2.1 *'Starting in the Middle'*

For the first experiment we deleted the first $x = 10, 15, 20, ...$ bars from the performances, resulting in performances where the pianist skips some bars at the beginning, thus simulating 'random' starts somewhere in the score. In total we computed alignments on 600 performances, with x ranging from 10 up to 250 (the larger value of course depending on the total number of bars of the piece), amounting to about 38 hours of live performance. We provide 2 different measures for the performance of the algorithm:

- **Measure 1 (M1):** The time spent matching wrong positions until the algorithm actively jumps to a bar in the score corresponding to the actual performance. It is important to note that this might not be the exact same position as in the performance, as parts often occur more than once in a piece completely identical and without some context it is impossible, even for humans, to decide which instance of a part is currently being played.
- **Measure 2 (M2):** The time spent matching wrong positions until the algorithm reaches (possibly by jumping more than once) the exact bar corresponding to the live performance.

In both cases time spent matching positions which according to the structure model are equivalent is not added to this measure.

The results of this experiment can be found in Tables 2 and 3. Just for comparison, Table 3 also shows the results of the system described in [2], on the same data. The latter can only detect predefined structural changes and thus has no means to cope with the arbitrary changes we made to the performances. Loosely speaking, it is mere chance if the correct position is ever detected, which becomes more improbable as the changes grow bigger (see also Figure 2).

The performance of the AMT (the system without Structure Model) depends on the characteristics of the piece in question. While the results on the Chopin and Rachmaninov pieces are quite similar – especially considering M2 as shown in Table 3 – the results for the Schubert piece are considerably worse. Considering the structure of this piece, which consists of a lot of repetitions, this does not come as a surprise as the AMT has no means to distinguish these parts and does not use any context that would allow for musically sensible decisions. This leads to a very annoying behaviour, which is not reflected in these tables: The algorithm sometimes jumps between different instances of the repetion in the score – not only while trying to find the correct position, but also after such a position is found (see Table 4).

	AMT			SAMT		
	C	**R**	**S**	**C**	**R**	**S**
Best	3.52	5.18	0.12	3.51	4.98	4.60
Quartile 1	6.78	8.10	7.52	6.62	8.88	7.36
Median	7.42	10.32	10.44	7.16	10.52	8.74
Quartile 3	10.18	14.56	30.70	8.04	15.76	11.92
Worst	60.26	51.96	128.86	56.18	56.04	68.7

Table 2. Statistics about the erroneous matching time (in seconds) until the system actively jumps to the next correct score position (error measure M1). The data is computed on performances of the pieces by Chopin (C), Rachmaninov (R) and Schubert (S) starting somewhere in the middle of the score instead of at the beginning (see text).

This behaviour, which for musicians is very tiresome, finally led us to introduce the Structure Model (see Section 5). For our experiments we used the structure model to make SAMT prefer the linearly next segment if several alternative positions pertaining to a class of repeated segments are available. As discussed in Section 7, other strategies are entirely possible and in fact depend on the usage scenario of the system.

As can be seen in Tables 2 and 3 the introduction of a SM leads to a similar performance on the pieces by Chopin and Rachmaninov while reducing the time needed to adapt to deviations in the Schubert piece, which is rich in repetitions. But even more important is the increase in robustness, as shown in Table 4. There still occurred some erroneous jumps in the Chopin piece, but over all aligned performances a wrong jump or an unnecessary jump roughly occurs only once every 2 hours of a live performance. Moreover, due to the ability of the algorithm to identify equivalent hypothesis, SAMT needs only half of the computation time of AMT.

6.2.2 *'Leaving out Parts'*

In the course of a second experiment we simulated performances where an arbitrary part in the middle of the piece is skipped. In total we used 394 performances in which parts with lengths from 20 to 200 bars were removed at different positions. The results (see Table 5) show that it generally takes a little bit longer to find the correct position than in the first experiment. Closer investigation showed that this is caused by the extra (and misleading) context given by the last couple of seconds of the live performance which increases the time the RPE needs to supply the correct position as hypothesis. This also accounts for the noticeably bigger difference in the results between the two experiments for the piece by Rachmaninov. It seems that the similarity of many short segments of this pieces makes the task of the algorithm more difficult. A possible improvement may be to make the thresholds t_a and t_r adaptive, thus speeding up the detection. Of course there is a trade-off between detection speed and robustness.

6.3 Conclusions from the Experiments

Although during the evaluation we focused on cases where parts from the score are left out during the performance, insertions or backward jumps should make no difference. This estimation is also supported by preliminary experiments.

In general the performance of our approach depends very much on how a piece of music is performed and where jumps occur. The

	Old System (ECAI 08)			AMT			SAMT		
	C	R	S	C	R	S	C	R	S
Best	15.98	24.74	25.08	4.74	5.18	0.86	4.64	4.98	4.60
Quartile 1	130.8	106.88	243.14	7.14	8.10	7.84	7.2	8.88	8.16
Median	not found	not found	not found	8.36	10.32	14.32	8.34	10.52	13.04
Quartile 3	not found	not found	not found	15.42	14.56	43.66	13.09	15.76	19.04
Worst	not found	not found	not found	60.26	51.96	128.86	56.18	56.04	82.36

Table 3. 'Starting in the Middle': Statistics about the erroneous matching time (in seconds) until the system reaches the exact same position in the score as in the live performance (error measure M2). The data is computed on performances of the pieces by Chopin (C), Rachmaninov (R) and Schubert (S) starting somewhere in the middle of the score (with omissions up to 250 bars) instead of at the beginning (see text).

	AMT			SAMT		
	C	R	S	C	R	S
$\mathbf{J}_{wrong}$	2%	0%	28%	2%	0%	0%
$\mathbf{J}_{equiv}$	4%	2%	30%	3%	0%	0%

Table 4. 'Starting in the Middle': Percentage of performances in which AMT/SAMT jumps to a completely wrong position (J_{wrong}), or 'unnecessarily' jumps to equivalent parts after reaching the correct position (J_{equiv}), which is also musically wrong.

	M1			M2		
	C	R	S	C	R	S
Best	3.98	3.08	3.26	3.98	3.08	3.26
Quartile 1	6.38	8.86	6.12	7.30	10.42	6.12
Median	9.62	14.68	9.28	12.82	15.20	9.28
Quartile 3	15.38	20.60	13.46	19.6	23.72	15.84
Worst	55.02	41.48	86.76	55.02	55.62	86.76

Table 5. 'Leaving out Parts': Statistics about the erroneous matching times (in seconds) until SAMT actively jumps to the (next) correct score position (M1 and M2). The data is computed on performances of the pieces by Chopin (C), Rachmaninov (R) and Schubert (S) which skip parts of different length somewhere during the performance (see text).

performance decreases with extreme tempo changes within a performance, and with large differences in absolute tempo between the (audio representation of) the score and the live performance. Problems are especially apparent at phrase boundaries, where huge differences in timing occur. Jumps to these areas account for many of the bigger errors found in Tables 2, 3 and 5.

Depending on the scenario one could think of a different alignment strategy than we used for our experiments, e.g. if it is known beforehand that the musician is practicing and therefore may repeat parts multiple times the strategy 'always jump to the nearest hypothesis' instead of 'always jump to the linearly next hypothesis' would be more appropriate.

7 Conclusion and Future Work

The paper has presented an robust on-line audio alignment algorithm which is capable of coping with arbitrary structural changes during a musical performance. The algorithm may also prove useful for other domains that need robust on-line alignment of time series which possibly include structural deviations.

A possible future scenario would be to extend this algorithm to operate on a whole database of musical pieces, automatically recognising both the piece being played, and the current position. An off-line matching/retrieval scenario related to this has been described in [7]. Practically this will require a clever indexing scheme based on musically relevant high-level features to quickly find those pieces and time points most likely to match the ongoing sound stream.

More directions for future work are to find ways to cope with phrase boundaries and more elaborate methods to determine the actually played part from a number of equivalent hypotheses (e.g. by analysis of the musical context). To achieve the latter the additional use of high-level musical features for the RPE should be considered.

ACKNOWLEDGEMENTS

This research is supported by the City of Linz, the Federal State of Upper Austria, the Austrian Federal Ministry for Transport, Innovation and Technology, and the Austrian Science Fund (FWF) under project number TRP 109-N23.

REFERENCES

[1] Andreas Arzt and Gerhard Widmer, 'Simple tempo models for real-time music tracking', in *Proc. of the Sound and Music Computing Conference (SMC)*, Barcelona, Spain, (2010).

[2] Andreas Arzt, Gerhard Widmer, and Simon Dixon, 'Automatic page turning for musicians via real-time machine listening', in *Proc. of the 18th European Conference on Artificial Intelligence (ECAI)*, Patras, Greece, (2008).

[3] Arshia Cont, 'A coupled duration-focused architecture for realtime music to score alignment', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **99**, (2009).

[4] Roger Dannenberg, 'An on-line algorithm for real-time accompaniment', in *Proc. of the International Computer Music Conference (ICMC)*, San Francisco, (1984).

[5] Simon Dixon, 'An on-line time warping algorithm for tracking musical performances', in *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, (2005).

[6] Meinard Mueller and Daniel Appelt, 'Path-constrained partial music synchronization', in *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Las Vegas, (2008).

[7] Meinard Mueller, Frank Kurth, and Michael Clausen, 'Audio matching via chroma-based statistical features', in *Proc. of the 5th International Conference on Music Information Retrieval (ISMIR)*, London, (2005).

[8] Bryan Pardo and William Birmingham, 'Modeling form for on-line following of musical performances', in *Proc. of the 20th National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, (2005).

[9] Christopher Raphael, 'Current directions with music plus one', in *Proc. of the Sound and Music Computing Conference (SMC)*, Porto, (2009).

[10] Barry Vercoe, 'The synthetic performer in the context of live performance', in *Proc. of the International Computer Music Conference (ICMC)*, San Francisco, (1984).