FAST IDENTIFICATION OF PIECE AND SCORE POSITION VIA SYMBOLIC FINGERPRINTING

Andreas Arzt[1], **Sebastian Böck**[1], **Gerhard Widmer**[1,2]
[1]Department of Computational Perception, Johannes Kepler University, Linz, Austria
[2]Austrian Research Institute for Artificial Intelligence (OFAI), Vienna, Austria
andreas.arzt@jku.at, sebastian.boeck@jku.at, gerhard.widmer@jku.at

ABSTRACT

In this paper we present a novel algorithm that, given a short snippet of an audio performance (piano music, for the time being), identifies the piece and the score position. Instead of using audio matching methods we propose a combination of a state-of-the-art music transcription algorithm and a new symbolic fingerprinting method. The resulting system is usable in both on-line and off-line scenarios and thus may be of use in many application areas. As the evaluation shows the system operates with only minimal lag and achieves high precision even with very short queries.

1. INTRODUCTION

Over the last few years efficient systems for content-based audio retrieval have been a major topic in music information retrieval research. These systems allow the user to browse and explore large music collections without the need for meta-data and other external information sources. In this context methods to automatically retrieve all pieces (and/or all the excerpts of the pieces) matching a given example query (in the form of a short audio clip) play an important role (and actually are in everyday commercial use). This task, most commonly called *audio identification* or *audio fingerprinting*, can be considered as solved (see e.g., [5, 10]).

Audio identification by definition only finds exact replicas of the query in the database, possibly distorted in some ways (e.g., compression artefacts, noise). Especially for classical music, this is not sufficient, because there are generally large numbers of different performances of the same piece, all different in terms of tempo, expressive timing, and other performance aspects. The relationship between these performances (that they derive from a common musical score) in general goes unnoticed by an audio identification algorithm. In this paper we propose a method for *score identification*: instead of identifying a particular performance it returns the musical score on which the query snippet is based. For example, if we present an audio excerpt of Vladimir Horowitz playing Chopin's Nocturne Op. 55 No. 1 to the system, it will return the name and data of the piece (Nocturne Op. 55 No. 1 by Chopin) rather than the data of the specific performance. Moreover, the system we propose returns not only the corresponding score, but also the exact position within the score. Accordingly, the database for this task does not contain audio recordings, but symbolic representations of musical scores (i.e., to identify the piece being played, the system only uses the symbolic score and has no information about the specific performance by Horowitz in the database).

This task is related to *cover song identification* (see [9] for an overview), where the goal is to identify different versions of one and the same song, in order to detect cover songs in popular music for commercial applications. To perform this task algorithms have to cope with large variations in parameters like timbre, tempo, timing and structure between different performances. Score identification can be seen as a special case of cover song identification. A MIDI version of the score of a classical piece of music can be synthesized and then be treated as a "normal" performance in this task, i.e., performances become "cover songs" of the synthesized version of the score. Still, the difference to our approach to score identification is that for our system very short queries (e.g., 5 seconds) are sufficient, while cover song identification algorithms generally assign similarity values to whole pieces of music.

As an alternative to our symbolic approach *audio matching* can be considered (see e.g., [7]). In this case the score is again first transformed into an audio file (or a suitable in-between representation). Then an audio matching algorithm, most commonly based on dynamic programming techniques, retrieves all excerpts from a database which musically correspond to a short query clip. In contrast to audio fingerprinting methods audio matching can also cope with (non-linear) timing deviations. The downside of audio matching is that in general these methods are very slow compared to fingerprinting methods. To cope with the computational costs, [6] presented clever indexing strategies that greatly reduce the computation time. Still, due to the coarse feature resolution, relatively large query sizes are needed.

Another related task, especially regarding the on-line capabilities of the proposed algorithm, is *score following* (see e.g., [3,8] for state-of-the-art score following systems). In contrast to the algorithm presented in this paper, a score

follower needs to know a-priori which piece the performers are playing, and then tracks the on-going performance and continuously returns the current score position. To do so it relies both on access to the complete performance (up to the current point in time) and on the performers closely following the score (i.e., without any additional repeats or any jumps). It contrast to this, the algorithm presented in this paper is able to identify the piece being played, and to identify the actual (or at least an identical) score position from only a small, arbitrary snippet of the performance.

In the following we will describe a new symbolic approach to score identification. Instead of creating an audio representation of the score we create the database directly from the symbolic score information. Then we transform the audio query into a symbolic representation – a list of note onset times with their respective pitch – and use a symbolic fingerprinting algorithm, inspired by the algorithm described in [10], to find matching positions in the score database. This process is very fast, can be used in real-time, on-line applications, and yields very high precision (as can be seen in Section 4). Note that our algorithm involves music (audio) transcription – which is still basically an unsolved problem – as a query preprocessing step. For our system we use a state-of-the-art music transcription system for piano music which, despite many errors, provides us with transcriptions of sufficient quality for the robust symbolic fingerprinting algorithm. Still, this also means that our system currently only works for piano music (because of specific properties of the transcription system).

2. BUILDING THE SCORE DATABASE

Before actually processing queries the score database has to be built. In our system we use deadpan MIDI files as the basis for the score database. The duration of these MIDI files is similar to the duration of a 'typical' performance of the respective piece, but without encoded timing variations. From these files a simple ordered list of note events is extracted where for each note event the exact time in seconds and the pitch as MIDI note number is stored.

Next, for each piece fingerprint tokens are generated. In contrast to [10] we create them from 3 successive events according to some constraints (also see Figure 1) to make them tempo independent. Given a fixed event e we pair it with the first n_1 events with a distance of at least d seconds "in the future" of e. This results in n_1 event pairs. For each of these pairs we then repeat this step and again pair them with the n_2 future events with a distance of at least d seconds. This finally results in $n_1 * n_2$ event triplets. In our experiments we used the values $d = 0.05$ seconds and $n_1 = n_2 = 5$. Also inspired by [10] we further constrain the pair creation steps to notes which are at most 2 octaves apart.

Given such a triplet consisting of the events e_1, e_2 and e_3 the time difference $td_{1,2}$ between e_1 and e_2 and the time difference $td_{2,3}$ between e_2 and e_3 are computed. To get a tempo independent fingerprint token we compute the time difference ratio of the time differences: $tdr = \frac{td_{2,3}}{td_{1,2}}$.

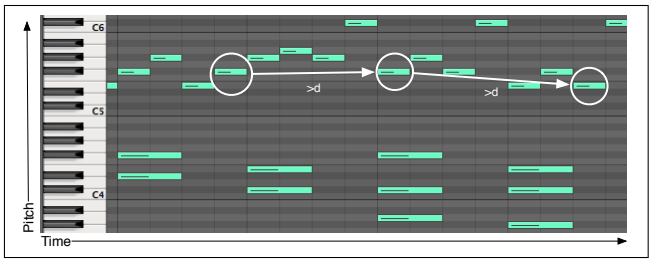

Figure 1. Fingerprint Token Generation: Example of 1 generated Token

This finally leads to a fingerprint token $[pitch_1 : pitch_2 : pitch_3 : tdr] : pieceID : time : td_{1,2}$, where the hash key $[pitch_1 : pitch_2 : pitch_3 : tdr]$ can be stored in a 32 bit integer. The purpose of storing $td_{1,2}$ in the fingerprint token will be explained in the description of the search process itself (see Section 3.2 below).

The result of the score preprocessing is our score database; a container of fingerprint tokens which provides quick access to the tokens via hash keys.

3. QUERYING THE DATABASE

3.1 Preprocessing: Transcribing the Query

Before querying the database the query (an audio snippet of a performance) has to be transformed into a symbolic representation. The algorithm we use to transcribe musical note onsets from an audio signal is based on the system described in [2], which exhibits state-of-the-art performance for this task. It uses a recurrent neural network to simultaneously detect the pitches and the onsets of the notes (see Figure 2 for an illustration of the algorithm).

For its input, a discretely sampled audio signal is split into overlapping blocks before it is transferred to the frequency domain with two parallel Short-Time Fourier Transforms (STFT). Two different window lengths have been chosen to achieve both a good temporal precision and a sufficient frequency resolution for the transcription of the notes. Phase information of the resulting complex spectrogram is discarded and only the logarithm of the magnitude values is used for further processing. To reduce the dimensionality of the input vector for the neural network, the spectrogram representation is filtered with a bank of filters whose frequencies are equally spaced on a logarithmic frequency scale and are aligned according to the MIDI pitches. The attack phase of a note onset is characterized by a rise of energy, thus the first order differences of the two spectrograms are used as additional inputs to the neural network.

The neural network consists of a linear input layer with 324 units, three bidirectional fully connected recurrent hidden layers, and a regression output layer with 88 units, which directly represent the MIDI pitches. Each of the hidden layers uses 88 neurons with hyperbolic tangent activation function. The use of bidirectional hidden layers enables the system to better model the context of the notes, which show a very characteristic envelope during their de-

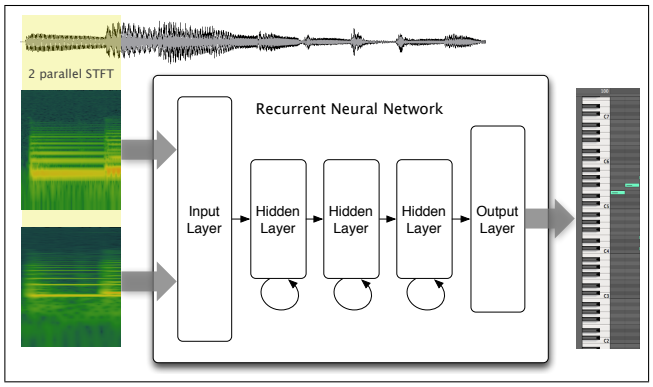

Figure 2. The Transcription System

Detection Window	Precision	Recall	F-measure
20 ms	0.586	0.489	0.533
40 ms	0.812	0.678	0.739
60 ms	0.851	0.710	0.774
80 ms	0.864	0.720	0.786
100 ms	0.869	0.725	0.790

Table 1. Results of the On-line Transcription Algorithm, for different detection window sizes.

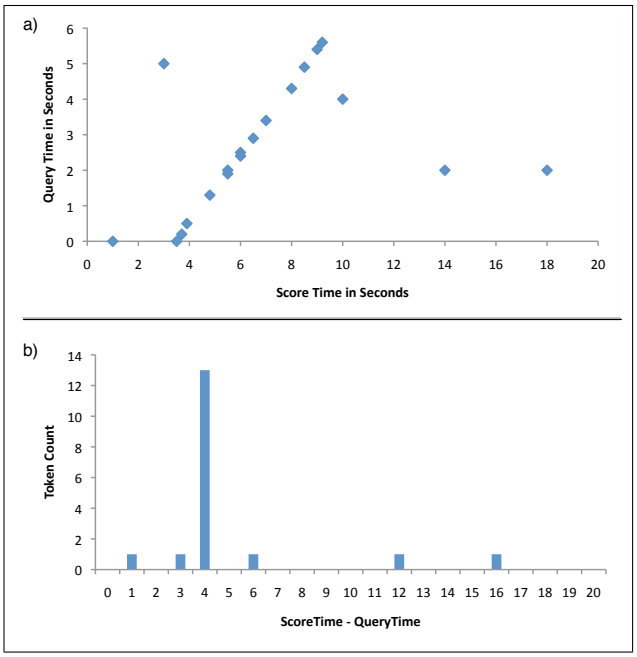

Figure 3. a) scatter plot of matching tokens and b) computed histogram for diagonal identification

cay phase.

The network is trained with supervised learning and early stopping. The network weights are initialized with random values following a Gaussian distribution with mean 0 and standard deviation 0.1. Standard gradient descent with backpropagation of the errors is used for training. The network was trained on a collection of 281 piano pieces recorded on various pianos, virtual and real (seven different synthesizers, an upright Yamaha Disklavier, and a Bösendorfer SE grand piano).

To make the transcriber applicable also in on-line scenarios, instead of preprocessing the whole piece of audio at a time, the signal is split into blocks of 11 frames centered around the actual frame. The use of 11 frames is a trade-off between keeping the system's ability to model the context of the notes and to keep the introduced delay at a minimum. In the current system the constant lag caused by the query preprocessing amounts to about 210 ms.

Table 1 shows the on-line transcription results for the complete test set described later on in Section 4.1. A note is considered to have been discovered correctly if its position is detected within a 'detection window' of given size around the annotated ground truth position. As can be seen in the table, the results are far from perfect (though they are very good, considering the state of the art). If the proposed fingerprinting system is used in an off-line scenario, the use of an off-line transcription algorithm is an option to slightly improve the results.

3.2 Querying the Database

The transcription of the query results in a list of note pitches with timestamps. This list is then processed in the same way as described in Section 2 above to produce query tokens. Of course in this case no piece ID is known and furthermore each query starts at time 0. These query fingerprint tokens are now used to query the database. The method described below is again very much inspired by the audio fingerprinting method proposed in [10].

The general idea is to find regions in the score database which share a continuous sequence of tokens with the query. To do so first all the score tokens which match the query tokens are extracted from the database. When plotted as a scatter plot against their respective time stamps (see Figure 3a) matches will be indicated by (rough) diagonals (i.e., these indicate that the query tokens match the score tokens over a period of time). As identifying these diagonals directly would be computationally expensive we instead use a simpler method described in [10]. This is based on histograms (one for each piece in the score database, with a time resolution of 1 second) into which the matched tokens are sorted in a way such that peaks appear at the start points of these diagonals (i.e., the start point of a query, see Figure 3b). This is achieved by computing the bin to sort the token into as the difference between the time of the score token and time of the query token. The complete process will be explained in more detail below.

For each of the query tokens qt with $[qpitch_1 : qpitch_2 : qpitch_3 : qtdr] : qtime : qtd_{1,2}$ the following process is repeated. First, matching tokens are extracted from the score database via the hash key. To allow for local tempo differences we permit the normalized time difference to be within $\frac{1}{4}$ of $qtdr$. This normally results in a large number of score tokens $[spitch_1 : spitch_2 : spitch_3 : stdr] : spieceID : stime : std_{1,2}$. Unfortunately directly sorting these tokens into bin $round(stime - qtime)$ of the histogram $spieceID$ does not necessarily make sense because of the query possibly having a different tempo than

Data Description	Number of Pieces	Notes in Score	Notes in Performance	Performance Duration
Chopin Corpus	154	325,263	326,501	9:38:36
Mozart Corpus	13	42,049	42,095	1:23:56
Additional Pieces	16	68,358	–	–
Total	183	435,670		

Table 2. Pieces in Database

	Query Length in Notes						
	5	10	20	30	40	50	60
Corr. Piece as Top Match	22.55%	78.33%	94.07%	96.70%	97.50%	98.01%	98.42%
Corr. Piece in Top 2	29.23%	83.22%	96.07%	97.67%	98.28%	98.64%	98.87%
Corr. Piece in Top 3	33.00%	85.50%	96.74%	98.12%	98.57%	98.91%	99.09%
Corr. Piece in Top 4	35.33%	86.88%	97.15%	98.32%	98.76%	99.09%	99.22%
Corr. Piece in Top 5	37.24%	87.86%	97.44%	98.49%	98.87%	99.17%	99.32%
Corr. Position as Top Match	14.41%	60.47%	80.35%	84.63%	84.86%	83.91%	83.70%
Corr. Position in Top 2	21.94%	75.09%	91.11%	93.39%	93.77%	93.39%	93.17%
Corr. Position in Top 3	25.77%	79.70%	93.69%	95.36%	95.73%	95.85%	95.84%
Corr. Position in Top 4	28.20%	81.94%	94.69%	96.14%	96.61%	96.84%	96.93%
Corr. Position in Top 5	30.02%	83.29%	95.22%	96.55%	97.05%	97.34%	97.47%
Mean Query Duration	0.60 sec	1.33 sec	2.78 sec	4.21 sec	5.63 sec	7.04 sec	8.48 sec
Mean Query Exec. Time	1.71 ms	5.13 ms	11.76 ms	16.86 ms	20.76 ms	26.36 ms	31.89 ms

Table 3. Results of the proposed piece and score position identification algorithm on the test database. Each estimate is based on 50,000 random audio queries.

expected by the score.

As an illustration let us assume a slower tempo for the query than for the respective score. Then the diagonal in Figure 3a would be steeper and when computing the bins via $round(stime - qtime)$ the first few tokens may fall into the correct bins. But soon the tokens, despite belonging to the same score position, would get sorted into lower bins instead.

Thus we first try to adapt the timing by estimating the tempo difference between the score token and the query token. First we compute the tempo ratio of both tokens $r = \frac{std_{1,2}}{qtd_{1,2}}$ and then adapt the time of the query event when computing the bin to sort the token into: $bin = round(stime - qtime * r)$.

We now have a number of histograms, one for each score in the database, and need a way of deciding on the most probable score position(s) (and, by implication, the most probable piece), for the query. We did experiments with different methods of computing the matching score but in the end simply taking the number of tokens in each bin as the score produced the best results.

4. EVALUATION

4.1 Dataset Description

For the evaluation of our algorithm a ground truth is needed. We need exact alignments of performances of classical music to their respective scores such that we know exactly when each note given in the score is actually played in the performance. This data can either be generated by a computer program or by extensive manual annotation but both ways are prone to annotation errors.

Luckily, we have access to two unique datasets where professional pianists played their performances on a computer-controlled piano [1] and thus every action (e.g., key presses, pedal movements) was recorded in a symbolic way. The first dataset (described in [11]) consists of performances of the first movements of 13 Mozart piano sonatas by Roland Batik. The second, much larger, dataset consists of nearly the complete solo piano works by Chopin performed by Nikita Magaloff [4]. For the latter set we do not have the original audio files and thus replayed the symbolic performance data on a Yamaha N2 hybrid piano and recorded the resulting performances.

As we have both symbolic and audio information about the performances, we know the exact timing of each played note in the audio files. The performances were manually aligned to electronic (symbolic) versions of the original sheet music. To build the score database we converted the sheet music to MIDI files with a constant tempo such that the overall duration of the file is similar to a 'normal' performance of the piece.

In addition to these two datasets we added some more scores to the database, solely to provide for more diversity and to make the task even harder for our algorithm (these include, amongst others, the Beethoven Symphony No. 5, the Mozart Oboe Quartet KV370, the First Mephisto Waltz by Liszt and Schoenberg Op. 23 No. 3). To the latter, we have no ground truth, but this is irrelevant since we

[1] Bösendorfer SE 290

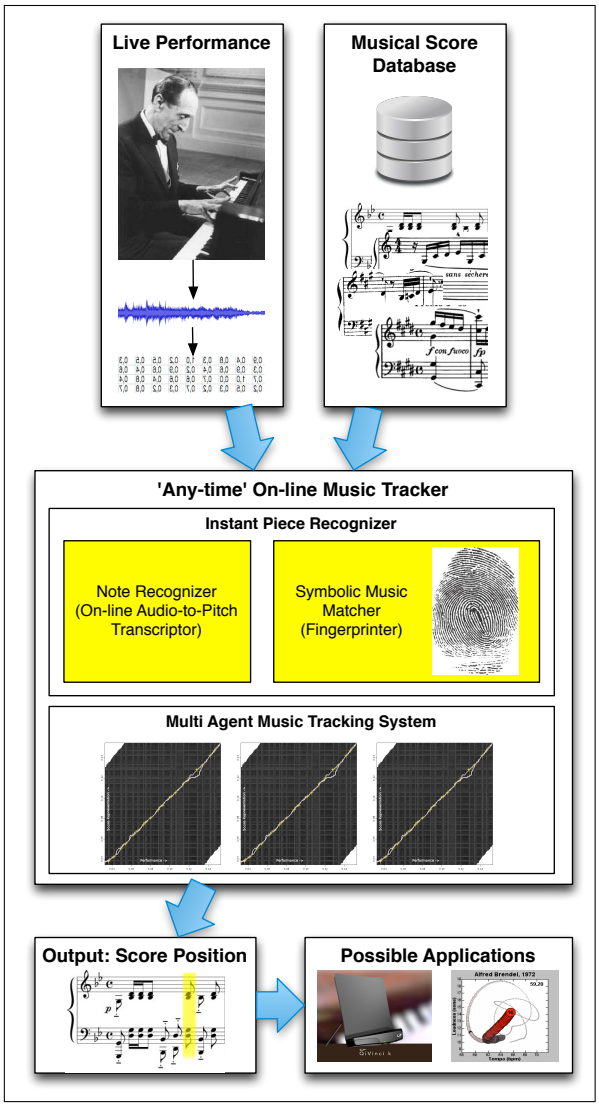

Figure 4. The Ultimate Music Companion

do not actively query for them with performance data in our evaluation runs [2]. See Table 2 for an overview of the complete dataset.

4.2 Results

An evaluation of the transcription stage (query preprocessing) was already presented in Section 3.1 above. As Table 1 shows the results of this stage are rather noisy. Still, the quality of the transcription is sufficient to be used with our robust fingerprinting technique.

We tested the algorithm with different query lengths: 5, 10, 20, 30, 40, 50 and 60 notes (in number of transcribed notes during the preprocessing step). For each of the query lengths, we generated 50,000 queries by picking random points in the performances of our test database, and used them as input for the proposed algorithm.

[2] Additionally we performed some non-systematic experiments with data from different sources (e.g., Youtube videos, both by amateurs and by professional pianists, with differing recording qualities (including noisy 'old' recordings and noisy amateur recordings)), for which we have no ground truth data. The general impression is that the system works well too in these scenarios, but of course the performance worsens in the presence of noise.

	Score Tempo		
	Normal	Double	Half
Corr. Pos. as Top M.	84.63%	83.30%	85.15%
Corr. Pos. in Top 2	93.39%	92.61%	92.70%
Corr. Pos. in Top 3	95.36%	95.06%	94.42%
Corr. Pos. in Top 4	96.14%	96.11%	95.00%
Corr. Pos. in Top 5	96.55%	96.55%	95.26%

Table 4. Results of the algorithm with score representations with altered tempi. The results are based on query lengths of 30 notes.

The results of this experiment are shown in Table 3. In this table we present two measures: the percentage of correctly identified pieces, and the percentage of cases in which both the piece and the exact position in the score were correctly identified.

For the evaluation a score position X is considered correct if it marks the beginning (+/- 1.5 seconds) of a score section that is identical in note content, over a time span the length of the query (but at least 30 notes), to the note content of the 'real' score situation corresponding to the audio segment that the system was just listening to (we can establish this as we have the correct alignment between performance time and score positions — our *ground truth*). This complex definition is necessary because musical pieces may contain repeated sections or phrases, and it is impossible for the system (or anyone else, for that matter) to guess the 'true' one out of a set of identical passages matching the current performance snippet, given just that performance snippet as input. We acknowledge that a measurement of musical time in a score in terms of seconds is rather unusual. But as the MIDI tempos in our database generally are set in a meaningful way, this seemed the best decision to make errors comparable over different pieces, with different time signatures – it would not be very meaningful to, e.g., compare errors in bars or beats over different pieces.

As can be seen, even queries of only a length of 5 notes lead to a surprising number of correct position identifications, and already for a query length of 20 notes (which corresponds to a mean query duration of 2.78 seconds) the correct position in the score is contained in the top 5 matches for more than 95% of the cases.

To show the tempo independence of our method we also ran experiments with big tempo differences between the score and the performance. We simulated this by manipulating the scores to have double and half the original tempo. The results for these experiments are shown in Table 4. As can be seen the performance only decreases slightly and the proposed algorithm still recognizes the correct position in the vast majority of the cases.

A flaw of the current approach is that it cannot cope with non-linear tempo deviations (i.e., with tempo variations *within* a query). As we are using very short queries and a rather coarse resolution in the histograms this is only

a minor problem. But for longer queries (e.g., with durations of over 10 seconds) explicitly dealing with non-linear tempo deviations becomes more of an issue. To make our approach useable with longer queries we propose to split the long query into smaller, overlapping queries (e.g., of size 30 notes with 15 notes overlap) and then use some simple tracking and scoring algorithm to combine the individual results into a single score. Preliminary experiments with this approach suggest that this leads to a very robust and accurate algorithm.

5. CONCLUSION

5.1 Applications

The proposed system is useful in a wide range of applications. For the off-line case it may either be used standalone or as a preprocessing step to audio alignment algorithms. This enables fast and robust (inter- and intra-document) searching and browsing in large collections of musical scores and corresponding performances.

Originally this work was motivated by an on-line scenario (see [1]). In connection with an on-line score following algorithm we are currently building a system that we somewhat immodestly call 'the ultimate classical music companion' (see Figure 4 for a sketch of the system). This system will be able to recognize arbitrary pieces of classical music, immediately identify the position in the score, provide meta-information, track the piece via an on-line score following algorithm, display the score, and visualize musical aspects of the performance like the structure of the piece, tempo and expressive timing deviations – all done on-line with minimal delay. This ultimate music companion may be used to enhance the listening experience but may also be of use for performers, especially during rehearsal. Besides this very specific application the proposed algorithm may also be of use in any application which requires monitoring of an audio stream of classical music on-line with minimal delay.

5.2 Future Work

Regarding the improvement of the algorithm, we see some future work in making it better useable with longer queries including tempo variations. As already mentioned above a combination of small overlapping subqueries with simple tracking of the results seems to be the way to go. Also a larger scale evaluation of the algorithm (with thousands of classical piano scores) has to be done.

The algorithm in the current state is able to recognize the correct piece and the score position even for very short queries of piano music. Based on this algorithm we have already implemented a real-time on-line piece and score position identification system that shows a level of performance that even human experts in classical music will find hard to match. This will be demonstrated live at the conference. We are now working on the integration of the proposed algorithm in our score following system as a next step towards our ultimate goal: 'the ultimate classical music companion'.

6. ACKNOWLEDGMENTS

This research is supported by the Austrian Science Fund (FWF) under project numbers Z159 and P22856-N23.

7. REFERENCES

[1] A. Arzt, G. Widmer, S. Böck, R. Sonnleitner, and H. Frostel. Towards a complete classical music companion. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2012.

[2] S. Böck and M. Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012.

[3] A. Cont. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):974–987, 2010.

[4] S. Flossmann, W. Goebl, M. Grachten, B. Niedermayer, and G. Widmer. The Magaloff project: An interim report. *Journal of New Music Research*, 39(4):363–377, 2010.

[5] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002.

[6] F. Kurth and M. Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, 2008.

[7] M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2005.

[8] C. Raphael. Current directions with music plus one. In *Proceedings of the Sound and Music Computing Conference (SMC)*, 2009.

[9] J. Serra, E. Gómez, and P. Herrera. Audio cover song identification and similarity: background, approaches, evaluation, and beyond. In Z. W. Ras and A. A. Wieczorkowska, editors, *Advances in Music Information Retrieval*, pages 307–332. Springer, 2010.

[10] A. Wang. An industrial strength audio search algorithm. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, 2003.

[11] G. Widmer. Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries. *Artificial Intelligence*, 146(2):129–148, 2003.