TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A S T E R A R B E I T

Score Following with Dynamic Time Warping

An Automatic Page-Turner

ausgeführt am
Institut für Computational Perception
der Johannes Kepler Universität Linz

unter der Anleitung von
Univ.-Prof. Dr. Gerhard Widmer

durch

Andreas Arzt
Haslingerweg 1, 4553 Schlierbach

Datum

Unterschrift

Abstract

Every pianist is familiar with the problem: Whether at practicing or during a concert it is often difficult to turn the pages of the score but at least it's disturbing. To solve this problem the Viennese company Quidenus invented a device which, controlled by a foot switch, turns pages of musical scores. The idea behind this master thesis is to go even further and to automatize this process completely by using a state-of-the-art score following algorithm.

Score following, the process of following a musical performance with respect to the score live and in realtime, is being researched intensely since the mid 1980s. Many methods were proposed, starting with string matching techniques, leading recently to probabilistic algorithms often based on hidden markov models.

In this thesis extensions to a known online algorithm based on dynamic time warping are presented which improve both precision and stability. As the structure of a musical piece is fixed in principle but sometimes changed spontaneously by the pianist, a further extension is presented which based on multiple matching algorithms running simultaneously tries to recognize these changes.

The result of this master thesis is an easy to use program which, connected to the page turner hardware using USB, follows the score and sends the signal to turn the pages on predefined positions.

Kurzfassung

Jeder Pianist kennt das Problem: Egal ob während des Übens oder während eines Konzertes, die Notwendigkeit des Umblätterns des Notentextes ist manchmal schwierig zu bewerkstelligen, zumindest aber störend. Um dieses Problem zu lösen hat die Wiener Firma Quidenus ein Gerät entwickelt, das, bedienbar per Fußschalter, Notenhefte umblättert. Die Idee hinter dieser Masterarbeit ist es nun noch einen Schritt weiter zu gehen und diesen Vorgang mithilfe eines State-of-the-Art Score Following Algorithmus vollkommen zu automatisieren.

Score Following, also das Verfolgen einer musikalische Darbietung anhand des Notentextes, live und in Echtzeit, wird seit Mitte der 1980er Jahre intensiv untersucht. Seither wurden viele Verfahren vorgeschlagen, anfangs hauptsächlich auf String-Matching Techniken basierend, später vor allem probabilistische Algorithmen, häufig basierend auf Hidden Markov Models.

In dieser Masterarbeit wird ein bekannter Online-Algorithmus, basierend auf Dynamic Time Warping, mit einigen Erweiterungen versehen, die sowohl die Präzision, als auch die Stabilität verbessern. Da die Struktur des Musikstückes zwar grundsätzlich festgelegt ist, manchmal aber vom Pianisten spontan geändert wird, wird weiters eine Möglichkeit, basierend auf mehreren parallel laufenden Matching-Algorithmen, beschrieben, die diese Änderungen erkennen kann.

Das Ergebnis dieser Masterarbeit ist schließlich ein einfach zu bedienendes Programm, das per USB-Schnittstelle mit der Umblätterhardware verbunden ist, dem Notentext folgt und an vorher vom Benutzer definierten Stellen Umblättersignale sendet.

Acknowledgments

I would like to thank my supervisor Gerhard Widmer and his colleagues for all their support, encouragement and helpful suggestions, especially Simon Dixon for providing the MATCH sources and Sebastian Flossmann for his helping me with the live tests, especially his piano playing.

David Zarfl and Lukas Achathaler for their help with proof-reading and their suggestions.

My parents for their support, both emotional and financial, and my whole family for being always there for me.

My girlfriend Lisa Achathaler for being there and her patience and understanding.

Contents

1 Introduction

The starting point for this master thesis was a page-turning hardware developed by the Viennese company Quidenus to support musicians during concerts. Controlled by a footswitch, the page-turner relieves the musician of the bothersome manual page-turning.

The goal of this thesis is to use state-of-the-art score following techniques – score following is the process of following the performance of a musician with respect to the score in realtime – to automatize the page-turning process.

Figure 1.1 shows a graphical summary of the topic of this master thesis. Before the performance the score of the piece is converted into a MIDI file. From the MIDI file tempo and bar borders are extracted. Using a software synthesizer an audio representation of the score is generated. The positions where to turn the pages have to be specified by the musician using bar numbers.

During the performance the live audio signal is aligned to the audio representation of the score. If a position in the score representation is reached which was marked by the musician, a signal to turn the page is sent to the page-turner via USB.

As at the start of this project Simon Dixon, who developed a score following and music alignment system based on dynamic time warping, was a member of ÖFAI's[1] intelligent music processing group I could use his expertise in this area and was given the opportunity to use his source codes as an initial point for my implementations. The goal was to evaluate the existing implementation and to search for improvements regarding stability and accuracy.

While many previous score following systems need MIDI data of the live performance the presented algorithm is based solely on the audio signal. This task is much more difficult but independence of MIDI data is of course a desirable property.

In chapter 2, after a short definition of the problem of score following, a history of audio alignment and score following can be found. The presented solutions range from the beginnings in the mid 80s – mostly string matching techniques – to the first probabilistic works in the 90s up to recent solutions based on hidden markov models, dynamic time warping and graphical models.

As my implementation uses dynamic time warping in chapter 3 this technique is discussed in more detail with special emphasis on the implementation of Simon Dixon.

In chapter 4 my implementation including the hardware connection, the software architecture and improvements to the algorithm is presented. The proposed improvements range from simple parameter optimization to a new technique incorporating the score information

[1] The Austrian Research Institute for Artificial Intelligence

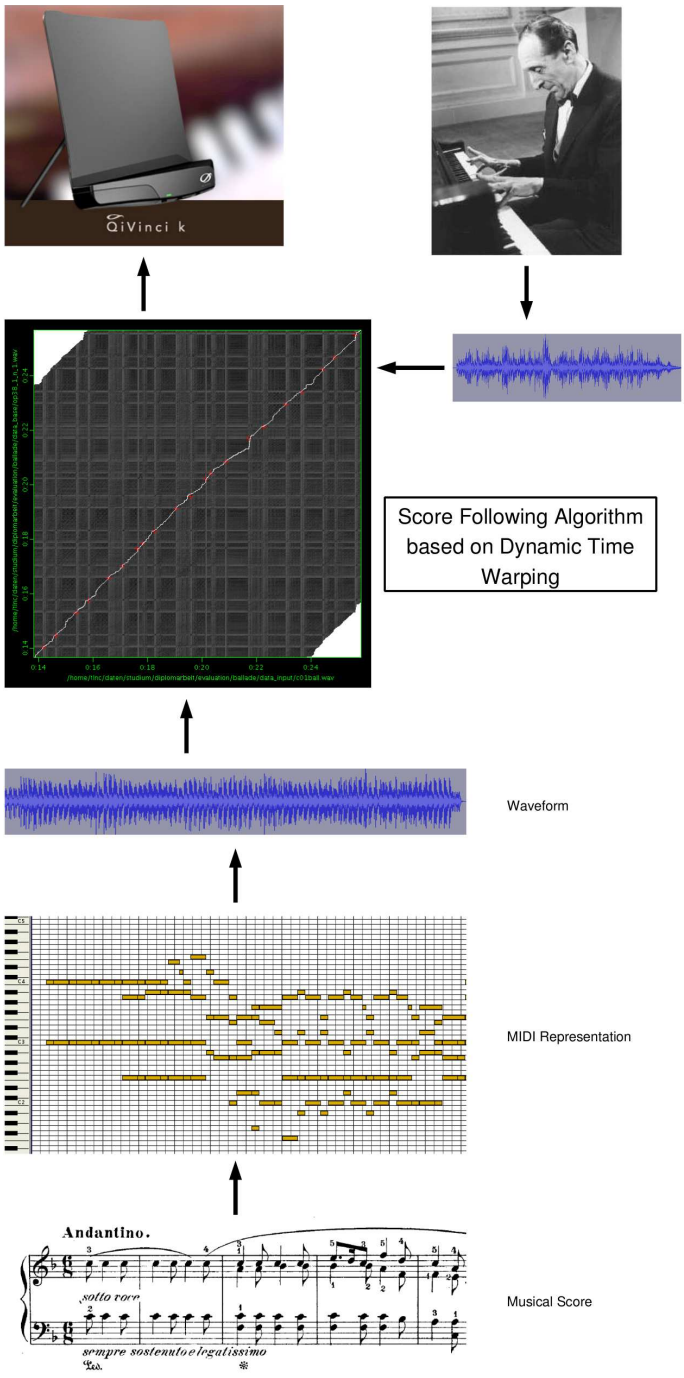

Figure 1.1: The Automatic Page-Turner

explicitly into the dynamic time warping algorithm and multi matcher strategies.

Extensive evaluations of my implementations and a comparison to the old algorithm can be found in chapter 5. They show that the presented algorithm performs better than the old implementation regarding both stability and accuracy. Two preliminary live tests showed that the performance is sufficient for the task of an automatic page turner.

Possible future improvements and a conclusion for this thesis are presented in chapter 6.

2 Score Following

2.1 A Definition of Score Following

Score following is the task of following the performance of a musician and deciding at which position he/she is with respect to a known score. As Arshia Cont puts it in his Master Thesis [Cont, 2004]:

"Score following serves as a real-time mapping interface from *Audio abstractions* towards *Music symbols* and from performer(s) live performance to the score in question."

The result of score following is a score-performance alignment. Thus offline score following is also often referred to as music alignment.

The challenge lies in the differences between the score and the musical performance. Especially in classical music tempo diversity, different styles of interpretation and generally errors of the musicians make it quite hard to find a proper alignment. Of course offline alignment is generally considered easier than online alignment.

Score followers can be used for a variety of applications including automatic accompaniment, live visualization and, as presented in this master thesis, automatic page turning.

In this chapter I will give a brief history of score following without focusing too much on technical details.

2.2 A History of Score Following

In general two epochs of score following can be identified. While from 1984 until about 1997 mainly string matching techniques were used, later the focus shifted to two methods known well in speech recognition: statistical approaches (e.g. HMMs) and dynamic time warping (DTW).

2.2.1 Early Approaches to Score Following

At the International Computer Music Conference (ICMC) in 1984 two papers from Roger Dannenberg and Barry Vercoe appeared independently and marked the start of research in the field of score following.

In "An On-Line Algorithm for Real-Time Accompaniment" [Dannenberg, 1984] Dannenberg described a score following system based on dynamic programming and a high-level symbolic representation of the performance. First the score and the audio input (MIDI events) are converted to strings and then the best match between these strings is computed. Since it is only useable for monophonic audio, it is not suitable for the problem of following a piano performance. Later on, this system was extended to handle polyphonic music [Bloch and Dannenberg, 1985]. Also handling for trills or glissando was introduced by using different matching techniques for each event. Furthermore he introduced the idea of multiple matchers running at different locations [Dannenberg and Mukaino, 1988] – an idea which I adopted for the automatic page-turner (see chapter 4).

As mentioned above Barry Vercoe also presented an article about score following [Vercoe, 1984]. His "Synthetic Performer"[1], an automatic accompaniment system, uses pitch as the main source but as pitch detection was not fast enough he used fingering information on the flute instead. The goal is to set the parameters of the automatic performance (tempo, loudness, phrasing) with respect to the actual human performance. This is done by pattern matching techniques. The "Synthetic Performer" also includes some learning strategies which are described in more detail in [Vercoe and Puckette, 1985].

In general Dannenberg's algorithms are more robust but less responsive than Vercoe's. While Vercoe assumes very skilled musicians, which change the tempo on purpose, Dannenberg does not trust his musicians to the same extent [Puckette, 1995].

Based on the above systems a new matching algorithm was presented in [Baird et al., 1990] and [Baird et al., 1993]. Instead of using single events like notes, matching is performed on segments of predefined length.

In 1990 Miller Puckette introduced "EXPLODE", a score follower also based on pitch detection [Puckette, 1990]. The matching is based on a pointer to the current note and a skip list containing previously unmatched notes. The algorithm tries to match each played live note first to a note in the skip list, then to the current note or a note in the near future. This is described in more detail in [Puckette and Lippe, 1992]. An interesting aspect is that several pieces were composed with having score following with EXPLODE in mind. Often the composers were forced to make compromises to ensure the score follower could follow the performance. One of these compositions – Philippe Manoury's *En Echo* for soprano and computer – showed the limits of an approach based on a finite alphabet of tempered-scale pitches and led to a more sophisticated method described in [Puckette, 1995].

So far the structure of a piece of music was only used implicitly in the matching process. In [Desain et al., 1997, Heijink et al., 2000] the temporal structure annotated in the score (chords, voices, etc) is used explicitly to predict note orders in the performance – for instance, notes in a melodic line are not likely to be played in a different order but parallel voices may be timed independently of each other – while dealing with the different voices independently

[1] At http://www.youtube.com/watch?v=vOYky8MmrEU a video showing the "Synthetic Performer" in action at the ICMC in 1984 can be found (last visited: September 18, 2007)

from each other. Multiple alternative matches are made and in the end dynamic programming is used to select the best path.

While all the previous papers mainly use pitch as the feature for score following, in [Vantomme, 1995] rhythm is used – only when the algorithm is completely lost pitch is used as a last chance. Of course, even if the musician plays totally false notes, as long as the timing is correct the algorithm is able to follow the performance. The algorithm does not seem very robust regarding typical errors like skipping or adding events.

Another approach based entirely on rhythm can be found in [Toiviainen, 1998]. Here an adaptive oscillator is used to track the beat of a MIDI input. There is no need for a specified score, even improvisations can be tracked. Around the beat tracker an accompaniment system is built which plays a predefined accompaniment in the perceived tempo. More examples on using a beat tracker for score following can be found in [Dannenberg and Mont-Reynaud, 1987] and [Allen and Dannenberg, 1990]. Of course every beat tracker could be used as a score follower – especially for improvisation this is a very reasonable technique. But for music where scores which can be used as an additional source of knowledge exist, an approach based entirely on rhythm is suboptimal.

While most approaches deal with a score which very closely specifies the pitch and the ordering of notes, Pardo and Birmingham try to match a performance to a partially specified score – a lead sheet as it exists in folk and popular music [Pardo and Birmingham, 2001]. This of course implies that a one-to-one mapping is not possible. The alignment is done by matching the chords extracted from the MIDI performance to the chords in the lead sheet by a technique to align two series while allowing gaps, drawn from gene-sequence analysis, based on dynamic programming.

2.2.2 The Statistical Approach

As even with perfect observations – correct information about pitch, tempo etc. in the live performance – a load of uncertainty due to errors and spontaneous decisions by the performers remains, a statistical approach is very natural.

The first to work on a solution to score following by using a statistical approach were Dannenberg and Grubb in 1997 [Grubb and Dannenberg, 1997]. In their approach at any point during the performance the position in the score is represented by a continuous probability density function, called the score position density. They define an observation distribution which specifies the probability of observing any possible value of a detected feature when the performer is performing this event. Using the current score position density and the observation distributions the new score position density is calculated.

A possibility to use probabilities to extend the string matching approach is shown in [Pardo and Birmingham, 2002]. They define a model of the transcriber error – useful for example when a quite unreliable pitch tracker instead of MIDI input is used – based on match probabilities. The search for a good alignment is done by dynamic programming.

Hidden Markov Models

Hidden Markov Models (HMMs) are very popular in speech recognition. As the focus of this master thesis is on DTW and not on HMMs I will not give an introduction on HMMs. A well known tutorial was written by Lawrence Rabiner [Rabiner, 1989].

One of the first approaches to use HMMs for score following is presented in [Cano et al., 1999]. The emissions of their model are a number of sound features like energy, zero crossing and fundamental frequency. Their note model architecture is based on 3 left-to-right HMMs: notes, modeled with 3 states (attack, steady state and release), silence and no-notes, which account for all unpitched sounds in the performance. Note lengths are modeled with self-transitions. The well known Viterbi algorithm is used for computing the alignment.

The big difference between the approach in [Cano et al., 1999] and Raphael's work [Raphael, 1999] is that the latter doesn't rely on pitch tracking routines but emits directly spectral features. Additionally to the Viterbi algorithm an alternative decoding technique is proposed. Using the information in the score, various graph topologies are used for different kinds of notes, such as long notes, short notes, rests and trills. Using his score following algorithm Raphael implemented a real-time accompaniment system [Raphael, 2003], [Raphael, 2004b] consisting of 3 components: listen (done by the HMM), synthesize (an audiofile is played with variable tempo) and anticipate (mediates between listen and synthesize using a bayesian network [Raphael, 2001]).

In [Orio and Dechelle, 2001] Raphael's score follower is extended by taking performer's errors into account by using "ghost states", which correspond to local mismatches. A two-level HMM is used to model the performance as a sequence of musical events and the signal as a sequence of features. Instead of the Viterbi algorithm, another algorithm, which was introduced in molecular genetics, is proposed, which showed lower delay and higher robustness to errors. A new algorithm for the training of the HMM is discussed too. A more detailed description of the implementation can be found in [Cont, 2004]. How this approach can be used with (polyphonic) MIDI data instead of the audio signal of the performance is shown in [Schwarz et al., 2004]. While the above implementations were only useable with monophonic or slightly polyphonic audio data, in [Cont, 2006] an extension to treat polyphonic music is described.

An interesting idea, which I did not find in other papers, is brought up in [Pardo and Birmingham, 2005]. As in a live performance musicians sometimes make spontaneous changes e.g. leaving out a repetition or repeating a part although not noted in the score, it is natural to take care of such situations in the score follower. So they model such possibilities explicitly in their score representation in the HMM.

2.2.3 Other Approaches

Neural Networks

One of the musicians who actually use score following algorithms to trigger events during their live performances even wrote their own software [Schreck-Ensemble, 2001]. Schreck

Ensemble's ComParser is based on a neural network. They call it a pseudo score follower as there is no actual score used. The musician just labels events in an existing recording of the piece of music. From the audio signal spectral features and amplitude measurements are extracted. The structure of the net is based on the avalanche structure. This approach is not an exact score follower which recognizes every note in the score. It is especially built to trigger MIDI events on certain cue points in electroacoustical pieces.

Graphical Models

Lately approaches based on graphical models were proposed for many music information retrieval tasks including music transcription [Kapanci and Pfeffer, 2005], beat tracking [Lung and de Freitas, 2004] and music alignment. A tutorial on graphical models can be found in [Murphy, 2001].

Christopher Raphael was the first to use graphical models for music alignment [Raphael, 2004a]. For him the problem with other approaches like HMMs and DTW lies in the too simple or not existing modeling of length for the individual notes: If treated at all, note lengths are either constrained to some range or modeled as random, with the distribution depending on a global tempo or learned from past examples. In this new approach a note-level model, explicitly representing tempo variations and note-by-note deviations, is combined with a simple frame-by-frame data model, based completely on the pitch content of the audio data. In this paper an offline version of the algorithm is presented, but it is stated that extensions to an online algorithm are possible.

2.2.4 Dynamic Time Warping

Dynamic time warping (DTW) is a technique for aligning time series. Typical applications are speech recognition [Rabiner and Juang, 1993], gesture recognition [Gavrila and Davis, 1995] and handwriting recognition [Rath and Manmatha, 2002], [Vinciarelli, 2002]. Unlike HMMs there is no training necessary. But DTW can be seen as a special case of HMMs. The cells of the matrix correspond to the states and the distances serve as output probabilities for a given state [Durbin et al., 1998].

In this section I will give an overview of the different approaches based on DTW. The technical background and the implementation for the automatic page-turner will be presented in chapters 3 and 4.

To my knowledge the first to apply DTW to music alignment were Nicola Orio and Diemo Schwarz in [Orio and Schwarz, 2001]. Their implementation followed exactly the standard definition of DTW in which they used the simplest local continuity constraint. As features they introduced a measure named "Peak Structure Distance" (PSD). The expected peaks in the spectrum are modeled from the pitches in the score and compared to the audio signal. To reduce complexity they didn't store all frames but only the first and the last score frame for each note. Another try to reduce the complexity is a kind of path pruning where in each

iteration only the best paths – determined by the minimum of this iteration plus a threshold – are kept [Schwarz, 2004].

Another possibility is to use discrete chromagrams of the audio signal as features for both sequences as shown in [Dannenberg and Hu, 2003]. A chroma vector is a 12 element vector, each representing one pitch class in the chromatic equal-tempered scale. The chroma representation proved to be a significantly better feature than MFCCs for this approach [Hu et al., 2003]. An interesting fact is that the chroma representation seems very insensitive regarding details like which instruments are used in the MIDI file. Even after substituting all instruments by a piano a good matching is still obtained. They took this one step further and tried to generate (simple) chroma vectors directly from the MIDI instead of using a synthesizer and analysing the audio signal. Even this worked quite well.

There exist some more approaches for reducing the complexity in time and space. Strictly speaking the algorithm used in [Mueller et al., 2004] is not DTW but another kind of dynamic programming. By using only (possible) note onsets – computed separately for every pitch – of the audio signal as features the amount of data is considerably reduced. The time axis is split evenly into segments and the onsets are assigned to bins according to their time position, discarding all empty bins. The score notes too are assigned to bins according to their score position. These two sequences are now used for the alignment.

Later on the same authors developed another approach which directly uses DTW [Mueller et al., 2006]. They use two strategies – reduction of the feature sampling rate and a global constraint region – iteratively in order to generate data dependent constraint regions. Their algorithm, called Multiscale DTW, showed the same aligning results as standard DTW while using more than 50 times less space and being 30 times faster.

Also aiming at complexity reduction is the approach presented in [Kaprykowsky and Rodet, 2006]. Based on the observation that one path in the matrix cannot cross another one, the amount of memory is significantly reduced. By backtracking "fusion points" which determine parts of the optimal path are found. After finding a fusion point it is sufficient to store the path found up to this fusion point and to clear all the previous data. Then the calculation continues. As this is an optimal approach the same result as with standard DTW is guaranteed.

By now all the presented approaches using DTW are purely offline algorithms. To my knowledge the only one to try to modify DTW to an online algorithm is Simon Dixon. As his implementations are the starting point of this diploma thesis they will be discussed in depth in chapter 3.3.

2.2.5 Comparison and Evaluation

I will concentrate here only on realtime audio-to-score alignment and leave all offline and midi-to-score approaches aside.

Still a comparison between the presented approaches is difficult. As stated in [Cont et al., 2007] there are hardly any systematic objective evaluations for the presented approaches. Usually the evaluation is limited to short live demonstrations and subjective ways to assess

the quality of the alignments. Luckily there are efforts now towards standardized evaluation methods.

In 2006 score following was one of the evaluation topics of the Music Information Retrieval Evaluation eXchange (MIREX)[2]. Sadly only 2 algorithms participated. IRCAM's score following algorithm based on HMM clearly outperformed an extended version of Dannenberg's 1984 algorithm.

In 2007 score following was postponed to possibly 2008.

If the 2008 evaluation takes place I will participate with my implementation. I'm very curious about a comparison between state-of-the-art algorithms based on ODTW, HMMs, graphical models and maybe other approaches.

[2] An annually evaluation of algorithms in a variety of music information retrieval tasks. More information can be found at http://www.music-ir.org/mirexwiki/index.php/Main_Page (last visited: February 7, 2008)

3 Score Following and Dynamic Time Warping

3.1 Dynamic Time Warping in Detail

DTW is a technique for aligning two time series or sequences. The series are represented by 2 vectors of feature vectors $U = u_1, ..., u_m$ and $V = v_1, ..., v_n$. To find the alignment with minimal costs the local distances $d_{U,V}(i, j)$ have to be computed. Usually these distances are represented as a $m \times n$ matrix. This matrix assigns a cost for aligning each pair (u_i, v_j). The costs are usually the Euclidean distance between the two vectors.

The goal of the DTW algorithm is to find a minimum cost path $W = W_1, ..., W_i$. 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. The alignment is done with respect to the local cost matrix. The cost of a path $D(W)$ is the sum of the local match costs of the path.

Several constraints are placed on the path W:

- W is bounded by the ends of both sequences
- W is monotonic
- W is continuous

Often additional global path constraints are used with the goal to reduce complexity such as the Sakoe-Chiba bound [Sakoe and Chiba, 1978] which constrains the path to lie within a fixed distance of the diagonal or the Itakura parallelogram [Itakura, 1975] which constrains the path to lie within a parallelogram around the diagonal of the matrix.

There are different local path constraints which can be used for the computation of the minimum cost path. In [Rabiner and Juang, 1993] these are presented. The simplest one is defined by the following recursion:

$$D(i, j) = d(i, j) + min \left\{ \begin{array}{c} D(i, j-1) \\ D(i-1, j) \\ 2 * D(i-1, j-1) \end{array} \right\}$$

This recursion can be computed in quadratic time by linear programming. $D(i, j)$ is the cost of the minimum cost path from $(1,1)$ to (i, j). $D(1,1) = d(1,1)$. In the end the path is extracted by following the recursion backwards from $D(m,n)$.

So alignment by DTW is essentially done in 3 steps:

- Extraction of comparable features from the 2 time series
- Calculation of local distances between the feature vectors of the 2 time series
- Computation of the optimal path with respect to the global distance

3.2 Features for Score Following by DTW

In the case of score following there are 2 different types of inputs. On the one hand there is the score and on the other hand there is the audio signal from the performance. Often the score is treated like the audio signal by creating a midi file and using a synthesizer to create an audio file [Dannenberg and Hu, 2003]. Then the same algorithms are applicable for the score and the performance.

Possible features are spectral representations of the audio data, generated from a windowed FFT of the signal. In [Dixon and Widmer, 2005] Simon Dixon maps the data into 84 frequency bins. The frequency axis is linear at low frequencies and logarithmic at high frequencies. This reduces the data and simulates the linear-log frequency sensitivity of the human auditory system. As the timing of the onsets is a very important factor for the alignment, a half-wave rectified first order difference is used, so that only increases in energy in each bin are taken into account.

Another way is to use a chroma representation [Dannenberg and Hu, 2003]. A chroma vector consists of 12 elements, where each element represents the spectral energy corresponding to one pitch class. This vector is easily computed by assigning each bin of the FFT to the nearest step in the chromatic scale. While according to [Hu et al., 2003] pitch histograms were outperformed by the chroma representation, in Dixon's implementation pitch histograms proved to be at least equally good.

It is also possible to use statistics over more than one frame. In [Mueller et al., 2005] these short time statistics, called Chroma Energy distribution Normalized Statistics (CENS), are computed over 41 chroma frames and used in addition.

A completely different approach is presented in [Schwarz, 2004]. Instead of an audio signal, a note model is generated from the score and compared to the audio signal.

Of course it is possible to use other well known features like MFCCs but they proved to be inferior compared to pitch-based representations [Hu et al., 2003].

3.3 Simon Dixon's Implementation of the DTW algorithm

Simon Dixon's implementation [Dixon and Widmer, 2005] was the starting point of this diploma thesis. I used some of his code and later on it was my reference implementation in the search for improvements.

3.3.1 Feature Extraction

As described above he uses a low-level spectral representation of the audio data which is generated from a windowed FFT of the signal. A Hamming window with the size of 46ms, and a hop size of 20ms is used. The data is mapped into 84 frequency bins. The lowest 34 bins up to 370Hz were mapped linearly and the bins from 370Hz to 12.5kHz were mapped logarithmically with semitone spacing. Finally the energy in higher bins was summed up into one last bin.

Between 2 audio frames a half-wave rectified first order difference is computed. So only the increases in energy $E'_x(f,t)$ in each bin f of the signal $x(t)$ at time frame t are taken into account:

$$E'_x(f,t) = max(E_x(f,t) - E_x(f,t-1), 0)$$

3.3.2 Distance Computation

For comparison of feature vectors the Euclidean distance is used:

$$d(i,j) = \sqrt{\sum_{b=1}^{84} (E'_u(b,i) - E'_v(b,j))^2}$$

As usual the distances are stored in a $m \times n$ matrix. The cost matrix is computed using the simplest DTW recursion.

$$D(i,j) = d(i,j) + min \left\{ \begin{array}{c} D(i,j-1) \\ D(i-1,j) \\ 2 * D(i-1,j-1) \end{array} \right\}$$

3.3.3 Path Computation

Even with constraints like the Sakoe-Chiba bound or the Itakura parallelogram the time and space complexity of the standard implementation of the DTW algorithm is quadratic in the length of the sequences. This is often cited as a limiting factor for the use of DTW with long sequences. Dixon proposed an easy way to get a linear algorithm by using a constant instead of a fraction of the total length for the width of the Sakoe-Chiba bound.

The danger is that the optimal solution could be too far away from the diagonal, so this solution could be excluded if the band is too narrow. To use this approach an (online) forward estimation of the center of this band – an *adaptive diagonal* – is necessary. After reaching the ends of both performances the optimal path is traced backwards as in standard DTW but using only the cells lying within the band computed by the forward estimation.

Dixon also implemented a smoothed path calculation using a least squares fit. This didn't improve accuracy at the note level as most unsmoothness occurs between notes and the algorithm recovers at the time of the next onset. But I could successfully use this path for my backtracking extension which is described in chapter 4.

3.3.4 Foward Estimation - Online DTW

This algorithm is both the heart of Dixon's DTW algorithm and a stand-alone Online DTW algorithm. It calculates an "adaptive diagonal" through the cost matrix which can be used to reduce the complexity of the DTW algorithm. Furthermore, this adaptive diagonal, which is of computed online, can be seen as a realtime mapping between the score and the performance.

As one of the two sequences is only known partially the algorithm differs from a standard DTW algorithm in some points:

- The length of the partially unknown sequence is unknown, so the global path constraints cannot be directly implemented.
- An incremental solution is required
- To run in realtime, the complete algorithm must be linear in the length of the sequence

In the following U is the partially unknown sequence.

At each time t, we seek the best alignment $u_1, ..., u_t$ to some initial subsequence of V. There is one parameter, c, which determines the width of the search band. The pointers t and j point to the current positions in U and V and are initialized to point to the start of each series.

The following pseudo-code of the online DTW algorithm is taken from [Dixon and Widmer, 2005].

Listing 3.1: ALGORITHM On-Line Time Warping

```
t := 1; j := 1
previous := None
INPUT u(t)
UpdatePathCost(t,j)
LOOP
    IF GetInc(t,j) != Column
        t := t + 1
        INPUT u(t)
        FOR k := j - c + 1 TO j
            IF k > 0
                UpdatePathCost(t,k)
    ELSE IF GetInc(t,j) != Row
        j := j + 1
```

```
        FOR k := t - c + 1 TO t
            IF k > 0
                UpdatePathCost(k,j)
    IF GetInc(t,j) == previous
        runCount := runCount + 1
    ELSE
        runCount := 1
    IF GetInc(t,j) != Both
        previous := GetInc(t,j)
END LOOP
```

Listing 3.2: FUNCTION GetInc(i j)

```
IF (t < c)
    return Both
IF runCount > MaxRunCount
    IF previous == Row
        return Column
    ELSE
        return Row
k := argmin(pathCost(t,*))
IF k < argmin(pathCost(*,j))
    return Column
ELSE
    k := argmin(pathCost(*,j))
    IF k == t
        return Both
    ELSE
        return Row
```

In the main loop of listing 3.1 partial rows and columns of the path cost matrix are calculated. To calculate a row (column) means to increment the pointer to the current position in the audio data which sits on the row (column). Then the path costs for the last c cells up to the current column (row) of this new row (column) in the path cost matrix are calculated.

The function *GetInc* (see listing 3.2) decides if a row or a column should be calculated as follows (see also figure 3.1). First the minimum path cost of the cells in the current row and column is found. There are 3 possible cases:

1. This occurs at the current position (t, j): Both the next row and the next column are calculated.

2. This occurs elsewhere in row j: The next row is calculated.

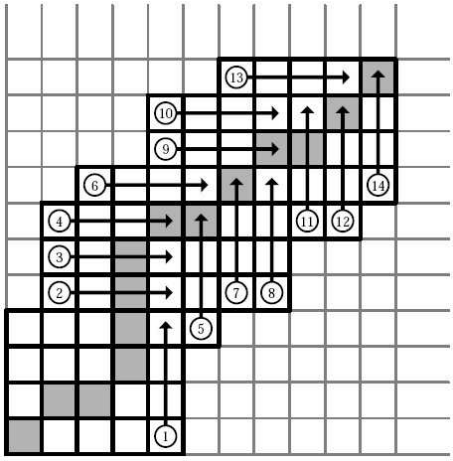

Figure 3.1: An example of the on-line time warping algorithm with search window c = 4, showing the order of evaluation for a particular sequence of row and column increments. All calculated cells are framed in bold, and the optimal path is colored grey. (Source: [Dixon and Widmer, 2005])

3. This occurs elsewhere in column t: The next column is calculated.

There are some special cases which override the default behavior:

- Less than c rows and columns have been computed: New rows and columns are calculated alternately.
- If a sequence is successively incremented more than *MaxRunCount* times, the other sequence is calculated next.

The calculation of the path cost up to a newly calculated cell is exactly like in the standard DTW algorithm, but it is restricted to use only the cells which have already been calculated. To make the comparison of paths of different lengths in *GetInc* possible, the path cost is normalized by the path length – in fact diagonal steps are simply multiplied by 2. The parameter c determines how many cells are calculated in each step. If a new row (column) is being calculated, the row (column) number is incremented and the cells in the last c columns (rows) are calculated.

In Dixon's implementation the parameter *MaxRunCount* is set to 3 which constrains the slope to range between $\frac{1}{3}$ and 3. As musical performances are not arbitrarily fast, the optimal alignment should always lie inside the reachable regions of the cost matrix.

The parameter c is set to 500 frames, which means that the width of the search band is 10 seconds. According to Dixon this is much larger than any error he had encountered in testing.

4 The Automatic Page-Turner

In the following I will present the architecture of the automatic page-turner. The software is based on Dixon's online DTW implemention. The hardware of the page-turner is produced by the Viennese company Quidenus[1].

Figure 4.1: The Quidenus Page-Turner for pianists

4.1 The Hardware

The hardware of the page-turner remained unchanged. The pedal controlling the page-turner by just closing a circuit was replaced by a relay. A Velleman K8055 USB interface board is used to switch the relay (see figure 4.3 for a picture of the page-turner controller).

4.2 The Input

In my implementation the score is represented as MIDI and converted into an audio file by a software synthesizer. As figure 4.2 shows there is a file called *Base File* (see listing 4.1).

[1] More information about the page-turner can be found at www.quidenus.com (last visited: February 7, 2008)

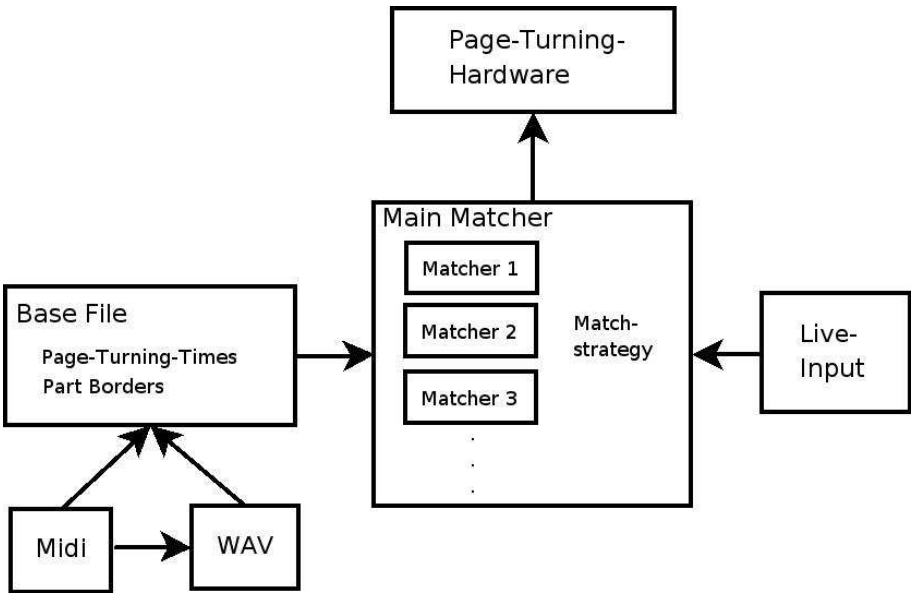

Figure 4.2: The Architecture of the Automatic Page-Turner

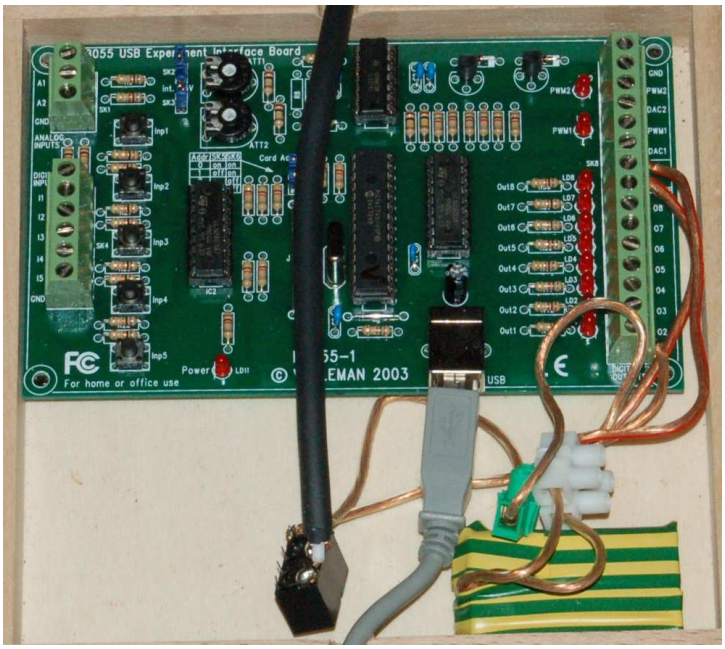

Figure 4.3: The Page-Turner Controller

This is a simple text file which contains the path to the MIDI and the audio. From the MIDI the measure, the length of the offbeat and the position of every note are extracted. With this information and the tempo of the piece, which is extracted from the MIDI too, it is possible to calculate the position of every note and every bar border in the audio file. Furthermore in this file the timing when to turn the pages (as bar numbers) can be specified. It is also possible to mark positions like repetitions or the start of a new movement in this file. These marks are important for the multi-matcher which will be described later.

Listing 4.1: A typical Base File

```
$description start
Sonate Nr. 2 F-Dur KV 280, 1. Satz
$description stop
$midi_file=[path_to_midi]/kv280_1.mid
$audio_file=[path_to_audio]/kv280_1.wav
$parts start
0.0
57.0
145.0
$parts stop
$marks start
27.0
57.0
89.0
114.0
$marks stop
```

To the audio data specified by the base file the input from the live performance is aligned. The feature extraction from the audio inputs is analogous to Dixon's implementation (see chapter 3.3). Of course the feature extraction from the basis audio file is done before the performance starts.

4.3 The Main Matcher

The main matcher controls the data flow to the matching algorithm. It receives one audio frame of the incoming audio stream, calculates the feature vector, hands it over to the matching algorithms, waits until they are done, then waits for the next frame and so on.

The main matcher can work in two modes: As a *single matcher*, when one matching algorithm is started which tries to align the performances, or as a *multi matcher*, when more than one algorithm runs simultaneously on different positions of the performance.

4.3.1 The Single Matcher

If the program runs in single matcher mode just one instance of the matching algorithm is used. The program waits for the live input to start and feeds the data frame by frame to the matching algorithm. If the algorithm reaches a frame index in the basis audio file which is marked as "end of page" (specified as bar numbers from the user and converted in frame indices using the midi information), the page-turner gets its signal to turn the page.

4.3.2 The Multi Matcher

A little bit more sophisticated is the use of more than one matcher. The idea behind this is that pianists sometimes do not play everything exactly as noted in the score. Sometimes they leave out or repeat parts, which are not marked as repetitions. This kind of "error" is impossible to catch with the online DTW implementation by Dixon because the window of about 10 seconds is just too small to recognize that a repetition of for example 15 bars is left out.

But if after the first repetition a second matching algorithm is started – one runs on the second repetition and the other one on the part after this repetition – this can be caught. You just have to pick the matching algorithm, which produced, after some time, the least error.

So my multi matching strategy is as follows: I am using 3 matchers – of course it would be possible to use more, but 3 are enough to prove the concept. In the input file the borders between possible repetitions or omissions are marked. I am starting with just one matcher until I come across the first mark. Then the two other matchers are added and are initialized as follows:

- Matcher 1 is just continuing matching from the reached mark
- Matcher 2 starts again from the previous mark – checking if this part is repeated
- Matcher 3 starts from the next mark – checking if now a part is skipped

After some time the matcher with the minimum error is selected and the others are stopped until the next mark is reached. During the evaluation it proved sufficient to decide which matcher is the correct one after 4 bars while ensuring these have a length of at least 10 seconds according to the base audio file.

The calculation of the error of the 3 matchers is very simple. The error accumulated during the last 1000 alignment steps is normalized by dividing by the number of frames processed during this time.

4.4 The Matching Algorithm

The following pseudo code shows what happens when a new vector describing a frame of the live performance is handed over to the matching algorithm. Of course this algorithm works on the path cost matrix.

```
while (input_not_advanced)
    select_advance_direction()
    update_matrix(direction)
    update_paths()
end while
```

So first the advance direction is computed which means that either the score or the performance coordinate (or both) are incremented. Then the matrix – by computing the path costs up to the new cells – and finally the paths are updated. This is done until the matrix was expanded into the direction of the live input. After that happened, the algorithm waits for the *main matcher* to hand over the next audio frame of the live performance.

I will describe now in detail what happens in the mentioned functions.

4.4.1 The Function *select_advance_direction()*

The two possible directions are *up* – meaning the next row of the matrix is computed – and *right* – meaning the next column is computed. It's also possible to advance in both directions, then the function *update_matrix* is called for *both*. In the following the base file sits on the y-axis and the live input on the x-axis. So to compute the next row – *up* – means to increment the score position coordinate, and to compute a new column – *right* – means to increment the position coordinate of the live performance.

The decision in which direction to advance next is made exactly as in Dixon's implementation. First the minimum path cost of the cells in the current row and column is found. There are 3 possible cases:

1. This occurs at the current position (t, j): Both the next row and the next column are calculated.
2. This occurs elsewhere in row j: The next row is calculated.
3. This occurs elsewhere in column t: The next column is calculated.

I used the same special cases which override the default behavior:

- Less then c rows and columns have been computed: New rows and columns are calculated alternately – while in Dixon's implementation c is set to 500, I set c in this context to 50. But the width of search band remains unchanged at 500, only the path calculation starts earlier.
- If a sequence is successively incremented more than *MaxRunCount* times, the other sequence is calculated next. As described in more detail later, I set this variable to 6 while in Dixon's implementation 3 was used.

4.4.2 The Function *update_matrix(direction)*

In this function the path cost matrix is updated. Again the computation is done similar to Dixon's implementation which follows the standard DTW algorithm except using only cells which have already been calculated.

The parameter c determines how many cells are calculated in each step – the width of the search band – which is 500 in both implementations. If a new row (column) is being calculated, the row (column) number is incremented and the cells in the last c columns (rows) are calculated.

In Dixon's implementation weights – 1 for normal steps and 2 for diagonal steps – were only used to normalize paths of different lengths to make them comparable. My idea is now to actually prefer diagonal steps while still leaving enough freedom to correctly align sequences which differ heavily in tempo. As an onset detection function is used there is few data for the decision making of the algorithm between onsets. As a result Dixon's implementation sometimes showed uncontrolled expansions in one single direction during notes. By preferring the diagonal such situations can be mostly avoided.

Changing the weights of normal steps to 1.3 while still using 2 for diagonal steps resulted in a more accurate alignment as can be seen in chapter 5.

4.4.3 The Function *update_paths()*

This function is the main difference between my implementation and the implementation by Simon Dixon. The following extension is motivated by a possible earlier discovery of suboptimal paths and a faster recovery after small errors by the pianists like some false note.

While normally the forward path calculation follows the one of Dixon described in 3.3.4, every i iterations a (smoothed) backward path is calculated. This path is followed s steps backwards, leading to a new point x. This point x lies with a high probability nearer to the globally optimal path than the corresponding point on the forward path because the backward path had more information for the computation of this point, especially some knowledge about the future.

From x a new forward path is calculated until a border – the last calculated cells – is reached. There are three possible situations (see also figure 4.4):

- The new forward path flows into the old path and ends in the same point. This can be seen as a confirmation for the current point.
- The new forward path ends below the old path. This means that the new point lies in the past. Of course in the domain of the page-turner it is not possible to go backwards, but it is possible to stop proceeding in time until the path reaches the old upper border again.
- The forward path ends above the old path. In this case more rows are calculated until the path reaches the right border.

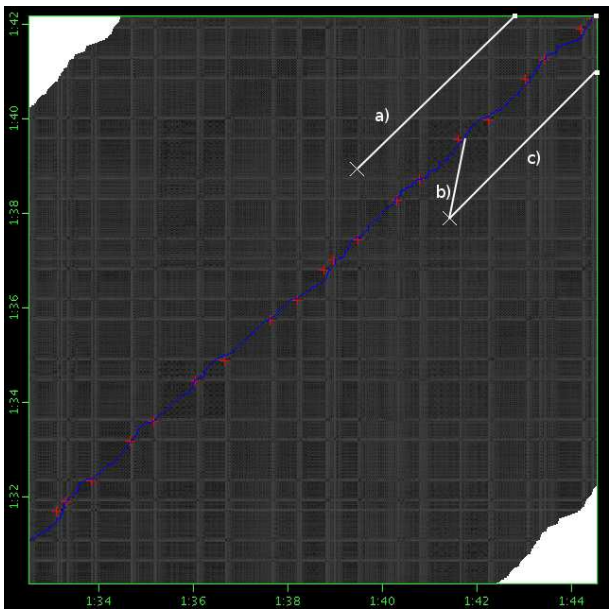

Figure 4.4: Examples for new forward paths: a) The path ends above the old path, b) The path flows into the old path, c) The path ends below the old path

Of course there are different parameters which control the extension to Dixon's online DTW algorithm:

- How often should this backtracking be applied?
- Should the parameter configuration for the computation of the forward path be the same as in the normal algorithm or can it be more aggressive?
- How far back should the starting point of the new computation be?

In fact it proved useful to use 2 different kinds of backtracking: One to correct small errors as fast as possible and one with the goal to recover faster from bigger errors. These 2 differ in only in one point: How far back the new forward path starts. Both share the same type of backtracking path, namely a path starting in the current point of the forward path, calculated as in the standard DTW algorithm. To avoid problems due to jumps between the notes the path is smoothed. There were some experiments using the point with the lowest error in the current row and column instead of the current point of the forward path, but this proved to be too unstable.

For the larger backtracking step the starting point lies 50 frames of the live input data in the past while for the smaller backtracking step the starting point only lies 10 frames in the past.

In my implementation backtracking is done every 2 frames, where after 4 smaller backtrackings one larger follows.

Using these backtracking methods it was possible to use a more aggressive path calculation. While in Dixon's implementation the algorithm is constrained to advance at most 3 times in a row in the same direction, I set this parameter to 6 for the normal path calculation and to 3 for the calculation of the forward path during the backtracking step.

4.5 Further Improvements

So far the alignment only used information from the audio signals. As in the domain of the score follower a score representation is used, which is MIDI in my case, it is natural to incorporate this information into the matching process. My idea is to use the calculated forward path only as an approximate solution and to search for onsets according to the MIDI information around this path. In contrast to the forward path calculation this search is based on a chroma representation, mainly because computations on chroma vectors are faster compared to the use of Dixon's representation. Nonetheless, it's also possible to use Dixon's representation instead.

The idea is simple: When the score representation coordinate of the forward path comes near to the next event (onset), the search for the onset is started. The goal is to find a frame which is likely to be a note onset, which is similar to the audio frame representing the note onset in the score and which is near to the onset coordinate according to the forward path calculation.

So for every new frame of the live input a function based on onset detection, the similarity between score representation and live input, and the distance of the forward path to the awaited onset is computed. If the value is higher than a defined border the algorithm marks the current event as detected and proceeds to the next event. In the following I will describe this process in more detail.

4.5.1 Preprocessing

The first step is to extract the note onset times from the MIDI and to compute the corresponding chroma vectors from the synthesized audio file. Then half-wave rectified first order differences between each chroma vector and the chroma vector of the forgoing frame are built. The vectors are normalized to sum up to 1. In the following I will refer to the vector corresponding to the current event as B_s.

4.5.2 The Matching Process

For each audio frame a chroma vector A is calculated. As for the audio representing the score, a second vector B_l is built using the half-wave rectified first order difference. Both vectors are normalized to sum up to 1.

The onset value OV should be a measure of how likely the current audio frame of the live input represents a note onset. The vectors A of the current and of the last audio frame are taken and the positive spectral differences are summed up. A maximum of 0.5 is used and the value is taken times 2 to ensure the value lies between 0 and 1.

The similarity value SV should represent how similar the current audio frame of the live input and the audio frame corresponding to the onset of the event in the audio file representing the score are. The Euclidean distance between B_l and B_s is calculated and normalized by a logarithm of the sum of both vectors. Values under 0.2 are very rare, so SV is lowered by 0.2 and 0 is set as the minimum and 0.5 is set as the maximum value. Then SV is taken times 2 to ensure the value lies between 0 and 1. As corresponding frames should have a high onset value and a high similarity value, SV is subtracted from 1.

The distance DS in frames between the happening of the event according to the MIDI file and the score-coordinate of the current point of the forward path is of course easily calculated by subtracting the path-coordinate from the MIDI-coordinate.

Based on OV, SV and DS a function can be described which accepts or rejects the current frame as the awaited event:

$$\frac{OV * OV_w + SV * SV_w}{\sqrt{\frac{30 - DS}{48 - DS * 4}}}$$

If the value of the formula is greater than 1 the current frame is accepted, otherwise it is rejected.

OV_w and SV_w are weights which should sum up to 1. I had the best results setting OV_w to 0.75 and SV_w to 0.25.

Of course in the denominator other formulas based on DS are possible. The important thing is that when we are – according to the forward path – still far away from the event the current frame should only be accepted if the algorithm is really certain. On the other hand when according to the forward path the event already happened, the algorithm should accept a frame even with low similarity.

After accepting a frame of the live input as the current event according to the MIDI the algorithm of course starts searching for the next event.

This search is only used if, according to the forward path, the current event is not more than 15 frames in the future and not more than 10 frames in the past. If it is too far in the future the search is simply skipped and if it is too far in the past the current frame is accepted as the event even with no similarity at all. By doing so it is impossible for the algorithm to get lost – as long as the forward path does not get lost.

As the algorithm got confused when many onsets occurred in a short time this extension is only used if there are more than 15 frames between the current and the previous onset.

4.6 Summary

This section presented both the hardware and the software of the automatic page-turner.

An interface board switching a relay and connected to the PC via USB is used to control the page-turner.

The software is based on Simon Dixon's online DTW algorithm. The following improvements were proposed:

Parameter Optimizations: The weights in the recursion calculating the path cost are changed towards cheaper diagonal steps. This enables the change of the local constraints towards more freedom for the path calculation and furthermore to start the path calculation earlier.

Backtracking Strategy: By calculating a new forward path starting in a point of a smoothed backward path – which is usually nearer to the optimal alignment – after every 2 processed frames, the algorithm is given the possibilities to wait for the musician and to jump forward in the score which was strongly constrained in the original algorithm.

Explicit Search for Onsets: As score information is available via the MIDI file it is natural to explicitly incorporate this information into the matching process. From now on the calculated path is only used as an approximate solution and a search for onsets is started based on an onset detection function, the similarity between score and performance and the distance to the forward path.

Multi-Matcher Strategy: Changes of the structure of the piece are impossible to catch for the original algorithm as the search window (the width of the "adaptive diagonal") is very limited. By using multiple instances of the matching algorithm on different parts of the piece and trusting the one which produces the least error such changes may be detected.

In the next chapter, these improvements will be evaluated thoroughly.

5 Evaluation of the Automatic Page-Turner

As for an evaluation a reference alignment is needed, results from offline-tests are presented. The results are the same as for the online alignment, only a small latency would occur.

Exact alignments for piano music are very rare, but I was allowed to use data from the ÖFAI[1] collected from the Boesendorfer SE290 grand piano. This piano measures precisely the time and the velocity of the hammers striking the strings.

The described algorithms were tested on 2 sets of piano recordings [Goebl, 1999]:

- Etude in E major, Op. 10, no. 3, bars 1–21 (40.5 beats in total) by Frederic Chopin played by 22 different pianists
- Ballade Op. 38, bars 1–45 (274 beats in total) by Frederic Chopin played by 22 different pianists

For the scores of the pieces see figures 5.1 and 5.2

The etude performances range from 70.1 to 94.4 seconds duration and the ballade performances range from 112.2 to 151.5 seconds.

For each piece of music the performances were aligned to an audio file generated from a MIDI file by the timidity++ software synthesizer. The MIDI files were generated directly from the score with constant tempo.

There were 5 different scenarios which were tested independently:

- The pianist performs as described by the score
- The pianist makes errors:
  - he/she leaves out bars
  - he/she plays additional bars
  - he/she plays false bars
  - "amateur pianist": errors like leaving out notes, adding notes and playing false notes are mixed.

Of course scenarios 2–4 are very unlikely to happen, no pianist e.g. plays false notes for a whole bar. These were just evaluated to find out how fast the algorithm recovers after really

[1] The Austrian Research Institute for Artificial Intelligence

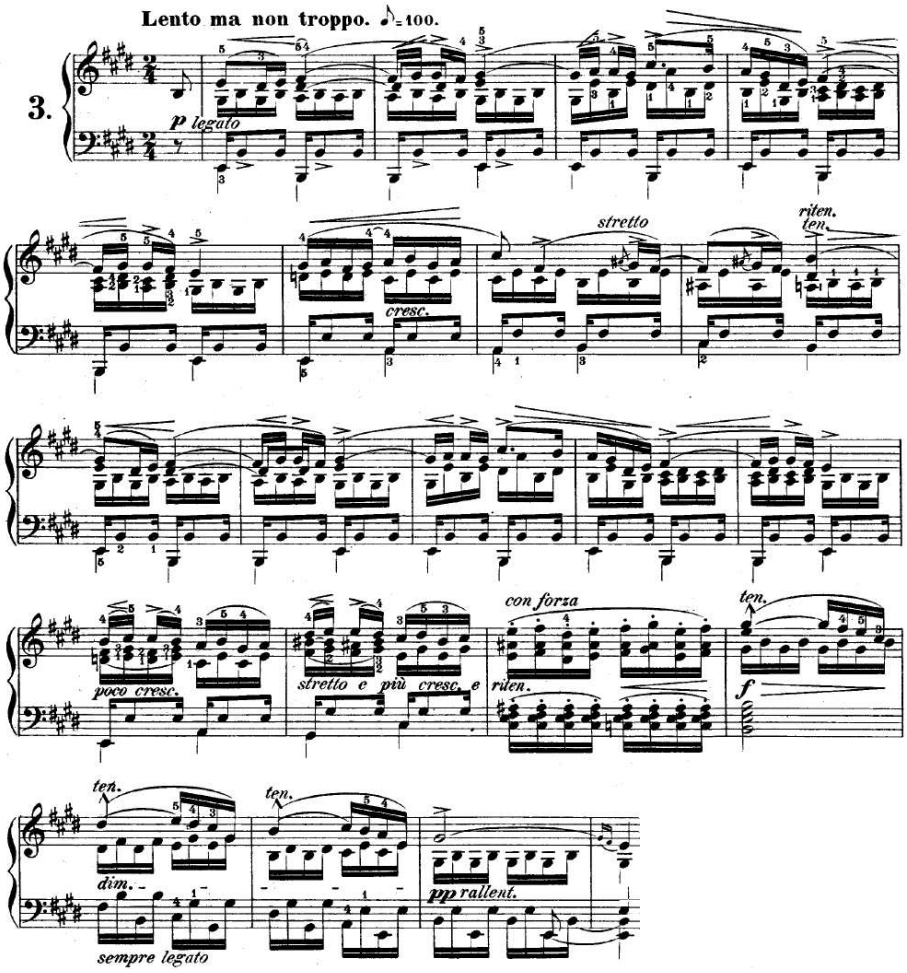

Figure 5.1: Etude in E major, Op. 10, no. 3, bars 1–21 (Source: http://www.free-scores.com/)

large errors. Much more common is a mix of added, left out and changed notes as in scenario 5.

As it is not possible to alter the performance for these tests the MIDI files are changed. For the case of leaving out notes, notes are repeated in the MIDI, for playing additional notes, notes are deleted from the MIDI and for playing false notes, notes are replaced by an augmented fourth. The augmented fourth is also known as tritone, often used as the main interval of dissonance, and was used to change the single notes as much as possible. A comparison between the results of my implementation and the implementation of Simon Dixon will be presented.

The described scenarios are tested with the *Single Matcher*. Additionally the *Multi Matcher*

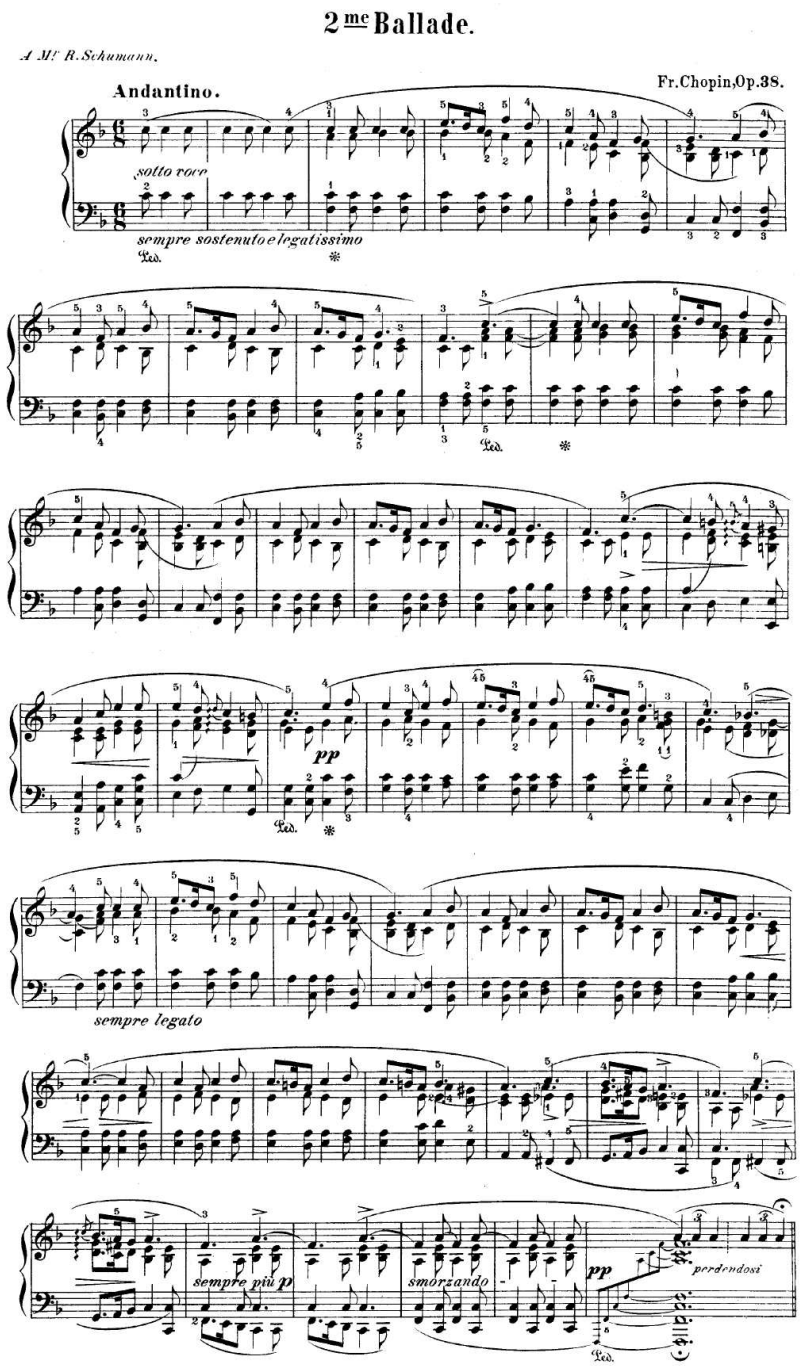

Figure 5.2: Ballade Op. 38, bars 1–45 (Source: http://www.free-scores.com/)

is tested regarding the recognition of additional repetitions and left out repetitions.

Furthermore the variability of the performance of the score following algorithm between different performances of the same piece of music will be evaluated.

For all scenarios all 22 performances of both pieces were used. For scenarios where the MIDI files were changed, the performance of the algorithm was evaluated multiple times for changes on several positions in the score to get more reliable results. More than 3500 test cases were computed for the evaluation using both my implementation and Dixon's implementation. As most of the test cases were aligned with 3 different algorithms, in total more than 10000 alignments were computed.

Of course the synthesizer and especially the soundfont influences the stability and the accuracy of the alignments. All MIDI files were synthesized using timidity++[2] and the grand piano soundfont from the freepats project[3]. One should not overrate the role of the soundfont. Tests showed that even with lower quality soundfonts the alignments are still good. Out of curiosity I tried other instruments than a piano and even with a font "Choir Aahs" the alignment was reasonable with a mean error of about 0.35 seconds on the etude.

The alignments themselves were evaluated as follows. First for each chord the average onset time was calculated. Then the corresponding chords in the two performances were aligned according to the symbolic data. The result was a set of points through which the path should pass. Now I used 2 different error functions to calculate the error for each point. The first one was the one used by Dixon in [Dixon, 2005] which calculated the Manhattan Distance between the note and the nearest point on the time warping path. This is not very meaningful in the application of an automatic page-turner. More meaningful is a function which computes the timespan between the supposed happening of an event and the real happening. So simply for an event e with coordinates (u_e, v_e) the distance between u_e and the point where the path hits the row v_e is calculated.

In the following, I will refer to Simon Dixon's online DTW algorithm as SODTW, my new implementation as AODTW and the implementation which incorporates the MIDI information into the matching process as AOODTW. As I will evaluate the implementations using both evaluation functions, I will use the abbreviation MDE for the evaluation function based on the Manhattan Distance and EDE for the function based on the timespan between the supposed and the real happening of an onset. If not mentioned otherwise, EDE is used.

The 2 pieces by Chopin are simply referred to as etude and ballade. The etude is in 2/4 time and the ballade in 6/8 time which is important to follow the tables as the recovery time is given in beats. For instance, in a piece in 6/8 time, a bar contains 6 beats with an inter-beat interval of 1/8 note.

[2] http://timidity.sourceforge.net/ (last visited: February 7, 2008)
[3] http://freepats.opensrc.org/ (last visited: February 7, 2008)

5.1 The Single Matcher

5.1.1 Scenario 1: "Normal" performance

The first notes

The new online DTW algorithm (AODTW) showed excellent results on the test data and outperformed Simon Dixon's algorithm (SODTW). One of the problems of SODTW is that at the beginning – when the first 500×500 frames of the matrix are calculated – the path follows simply the diagonal of this matrix. If the tempo of the live performance is significantly higher or lower than the tempo of the synthesized audio file, this of course results in a large error. Of course it was easy to change that. In AODTW the normal path calculation is started after the first 50×50 frames were calculated. Despite the small amount of data the path already proves to be quite stable and accurate (see tables 5.1 and 5.2).

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.52	0.08	0.58	0.15	0.11
Largest Error	2.18	0.86	2.46	1.22	1.14
1. Quartile	0.10	0.02	0.14	0.02	0.02
2. Quartile	0.36	0.04	0.42	0.08	0.02
3. Quartile	0.82	0.10	0.90	0.20	0.12

Table 5.1: Comparison of the performance on beats -0.5–4.5 of the etude (error in seconds)

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.26	0.04	0.30	0.09	0.06
Largest Error	1.30	0.54	1.30	0.96	0.70
1. Quartile	0.04	0.00	0.06	0.02	0.02
2. Quartile	0.12	0.02	0.18	0.04	0.02
3. Quartile	0.44	0.06	0.48	0.12	0.04

Table 5.2: Comparison of the performance on beats -4.0–23.0 of the ballade (error in seconds)

Furthermore, the explicit use of the MIDI data in AOODTW showed excellent results on the first notes and outclassed the performance of AODTW.

Normal alignment

As the last notes in this test set are in a way special too, to evaluate the general performance I will concentrate first on the middle parts of the etude and the ballade. In table 5.3 the performance on the beats 4.75–38.25 of the etude and in table 5.4 the performance on the beats 24.0–254.0 of the ballade can be seen.

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.09	0.04	0.16	0.08	0.06
Largest Error	1.60	0.78	2.76	1.76	2.12
1. Quartile	0.02	0.02	0.02	0.02	0.02
2. Quartile	0.04	0.02	0.06	0.04	0.02
3. Quartile	0.08	0.04	0.18	0.10	0.04

Table 5.3: Comparison of the performance on beats 4.75–38.25 of the etude (error in seconds)

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.10	0.04	0.17	0.08	0.05
Largest Error	1.86	1.26	2.54	1.74	1.78
1. Quartile	0.02	0.02	0.04	0.02	0.02
2. Quartile	0.04	0.02	0.08	0.04	0.02
3. Quartile	0.10	0.06	0.20	0.10	0.04

Table 5.4: Comparison of the performance on beats 24.0–254.0 of the ballade (error in seconds)

Once again there was a significant gain in both accuracy and stability. The main reason for this is the simple adjustment of the weights. As due to the reweighting in doubt the diagonal is preferred now, the path can not go wrong too far in a single direction. In SODTW this sometimes happened between notes when the path computation was “waiting” for new onset data.

Additionally the backtracking algorithm improved the stability, so I could give more freedom to the path algorithm. While in SODTW the path was constrained to expand at most 3 times into one direction in a row, in AODTW this variable is set to 6. This allows the algorithm to find back to the optimal path faster after errors. Of course during a “normal” performance without big errors the effect of this is very small.

The last notes

The last notes of a of this test set are problematic, especially of the ballade (see table 5.5 and table 5.6).

	SODTW (MDE)	AODTW (MDE)	SODTW (EDE)	AODTW (EDE)	AOODTW (EDE)
Mean Error	0.27	0.15	0.42	0.28	0.19
Largest Error	2.42	1.02	3.06	2.02	1.92
1. Quartile	0.04	0.04	0.08	0.04	0.04
2. Quartile	0.12	0.06	0.20	0.10	0.06
3. Quartile	0.34	0.16	0.60	0.42	0.22

Table 5.5: Comparison of the performance on beats 38.0–40.0 of the etude (error in seconds)

	SODTW (MDE)	AODTW (MDE)	SODTW (EDE)	AODTW (EDE)	AOODTW (EDE)
Mean Error	1.83	1.41	2.94	2.13	2.02
Largest Error	4.52	4.12	9.82	7.56	7.24
1. Quartile	0.36	0.04	0.62	0.12	0.04
2. Quartile	1.94	0.52	2.46	1.60	1.58
3. Quartile	3.24	3.20	5.30	4.18	4.14

Table 5.6: Comparison of the performance on beats 255.0-268.0 of the ballade (error in seconds)

The excerpt of the ballade and the etude end exactly at a phrase boundary. In music the term “phrase”, adopted from linguistc syntax, describes a section of music which is self contained and coherent over a medium time scale. In the case of the ballade at the end of the excerpt a phrase ends with a piano and pianissimo. The next phrase would start with a fortissimo an be totally different in tempo, figures and motifs.

Phrase boundaries are the most problematic parts in score following because there are very large variations in tempo and discontinuities in timing. Enormous errors may occur (see especially table 5.6) but if the performance continues after such a phrase boundary the algorithm recovers easily. Nonetheless if a page-turning mark happens to be in the area of a phrase boundary this could cause a delayed or a premature page-turning, which is as experiments showed more likely. Of course improvements on handling those boundaries should be one of the main subjects of future work.

Overall performance

The overall performance on the two pieces is shown in tables 5.7 and 5.8. As can be seen AODTW and AOODTW perform better than SODTW in every case.

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.15	0.05	0.23	0.10	0.07
Largest Error	2.42	1.02	3.06	2.02	2.12
1. Quartile	0.02	0.02	0.02	0.02	0.02
2. Quartile	0.04	0.02	0.08	0.04	0.02
3. Quartile	0.14	0.06	0.26	0.12	0.04

Table 5.7: Overall performance on the etude (error in seconds)

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Mean Error	0.20	0.11	0.32	0.19	0.15
Largest Error	4.52	4.12	9.82	7.56	7.24
1. Quartile	0.02	0.02	0.04	0.02	0.02
2. Quartile	0.06	0.02	0.08	0.04	0.02
3. Quartile	0.14	0.06	0.26	0.10	0.06

Table 5.8: Overall performance on the ballade (error in seconds)

Tables 5.9 and 5.10 show the performance on the 2 pieces as cumulative frequencies. In the first row the frequency of notes which are aligned with no error is shown, in the second row the frequency of notes which are aligned with an error smaller or equal 0.02 seconds is shown, and so on.

The incorporation of the MIDI data into the matching process proved to be very useful. Especially notes which were aligned with errors between 0.04 and 0.10 seconds by AODTW benefited from AOODTW (see tables 5.9 and 5.10. Only in very rare cases AOODTW showed larger errors than AODTW (e.g. the largest error shown in table 5.7).

Figures 5.3 and 5.4 show the error on every single aligned note by SODTW, AODTW and AOODTW. The direct comparison vividly shows the effect of the earlier start of the path computation and the improved alignment of AODTW and AOODTW. Once again witness the problems at the end of the excerpt of the ballade.

As AOODTW uses AODTW as an approximate solution it is not able too correct large

	SODTW (MDE)	AODTW (MDE)	SODTW (EDE)	AODTW (EDE)	AOODTW (EDE)
Error == 0.00 sec	0.15	0.21	0.10	0.16	0.16
Error <= 0.02 sec	0.37	0.54	0.27	0.40	0.53
Error <= 0.04 sec	0.53	0.73	0.39	0.56	0.79
Error <= 0.06 sec	0.62	0.83	0.46	0.65	0.84
Error <= 0.10 sec	0.72	0.81	0.56	0.75	0.87
Error <= 0.20 sec	0.81	0.97	0.70	0.87	0.92
Error <= 0.50 sec	0.91	0.99	0.87	0.97	0.98
Error <= 1.00 sec	0.97	1.00	0.96	0.99	1.00

Table 5.9: Overall performance on the etude given as cumulative frequencies

	SODTW (MDE)	AODTW (MDE)	SODTW (EDE)	AODTW (EDE)	AOODTW (EDE)
Error == 0.00 sec	0.13	0.21	0.08	0.13	0.18
Error <= 0.02 sec	0.34	0.52	0.23	0.36	0.51
Error <= 0.04 sec	0.49	0.72	0.35	0.52	0.74
Error <= 0.06 sec	0.58	0.83	0.44	0.62	0.84
Error <= 0.10 sec	0.69	0.92	0.55	0.75	0.89
Error <= 0.20 sec	0.82	0.96	0.70	0.88	0.93
Error <= 0.50 sec	0.91	0.97	0.86	0.95	0.96
Error <= 1.00 sec	0.96	0.97	0.95	0.97	0.97

Table 5.10: Overall performance on the ballade given as cumulative frequencies

errors of the path calculation. But, as already mentioned above, one can witness again that in correcting smaller errors AOODTW is very effective.

Furthermore, larger errors are very rare in the alignments of AODTW and AOODTW, and if they occur, they are corrected very fast.

According to these results the task of constructing an automatic page-turner could be seen as fulfilled – assuming correct play by the pianists as in the test data. Errors over 0.2 seconds are very rare and this should be sufficient for the domain of the page-turner. Of course one should not forget that there are still big problems at phrase boundaries.

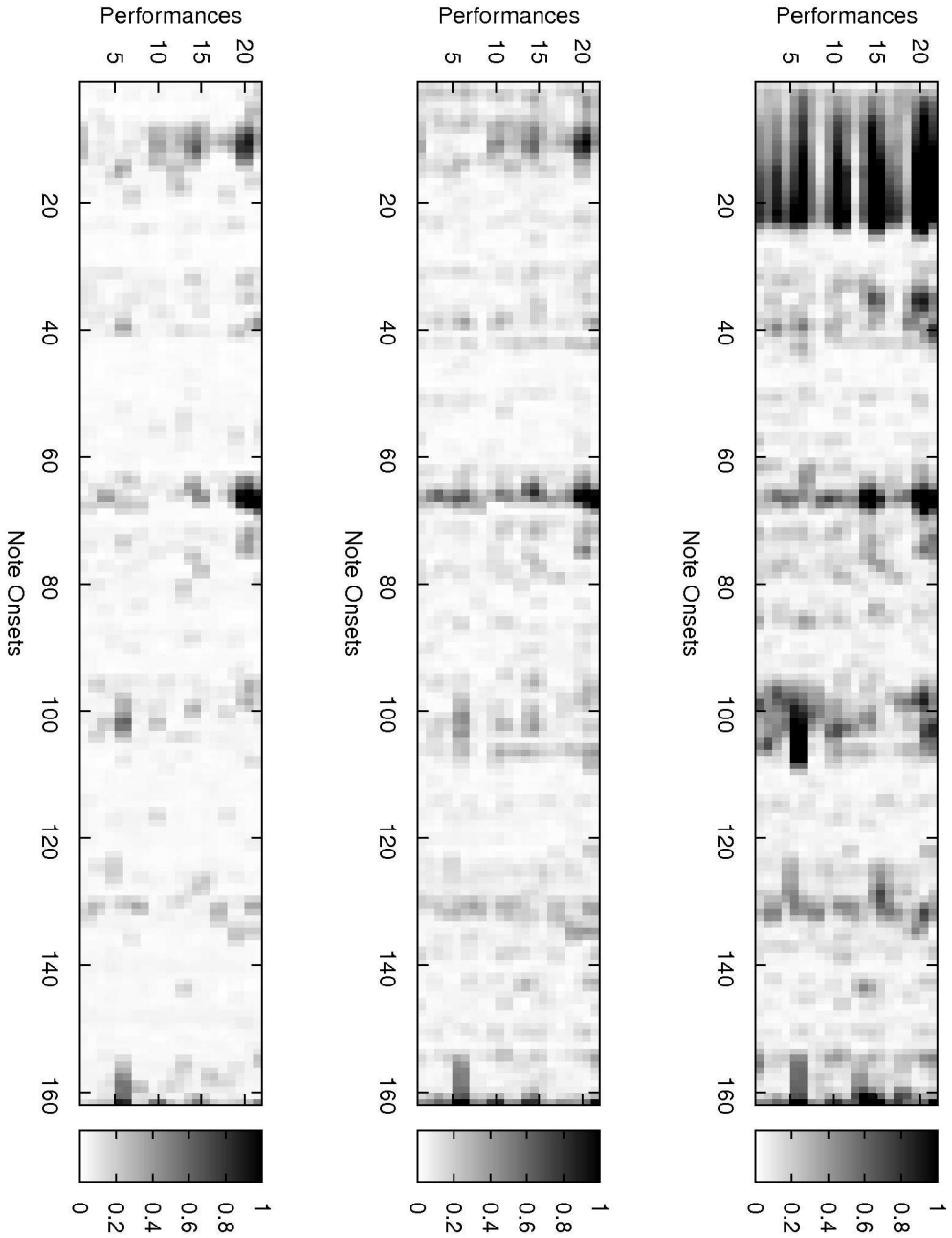

Figure 5.3: Comparison between AOODTW (left), AODTW (center) and SODTW (right), showing the error per note onset of the alignments of the 22 performances of the etude. The error is symbolized by the grey-level where white means no error and black an error higher or equal 1 second.

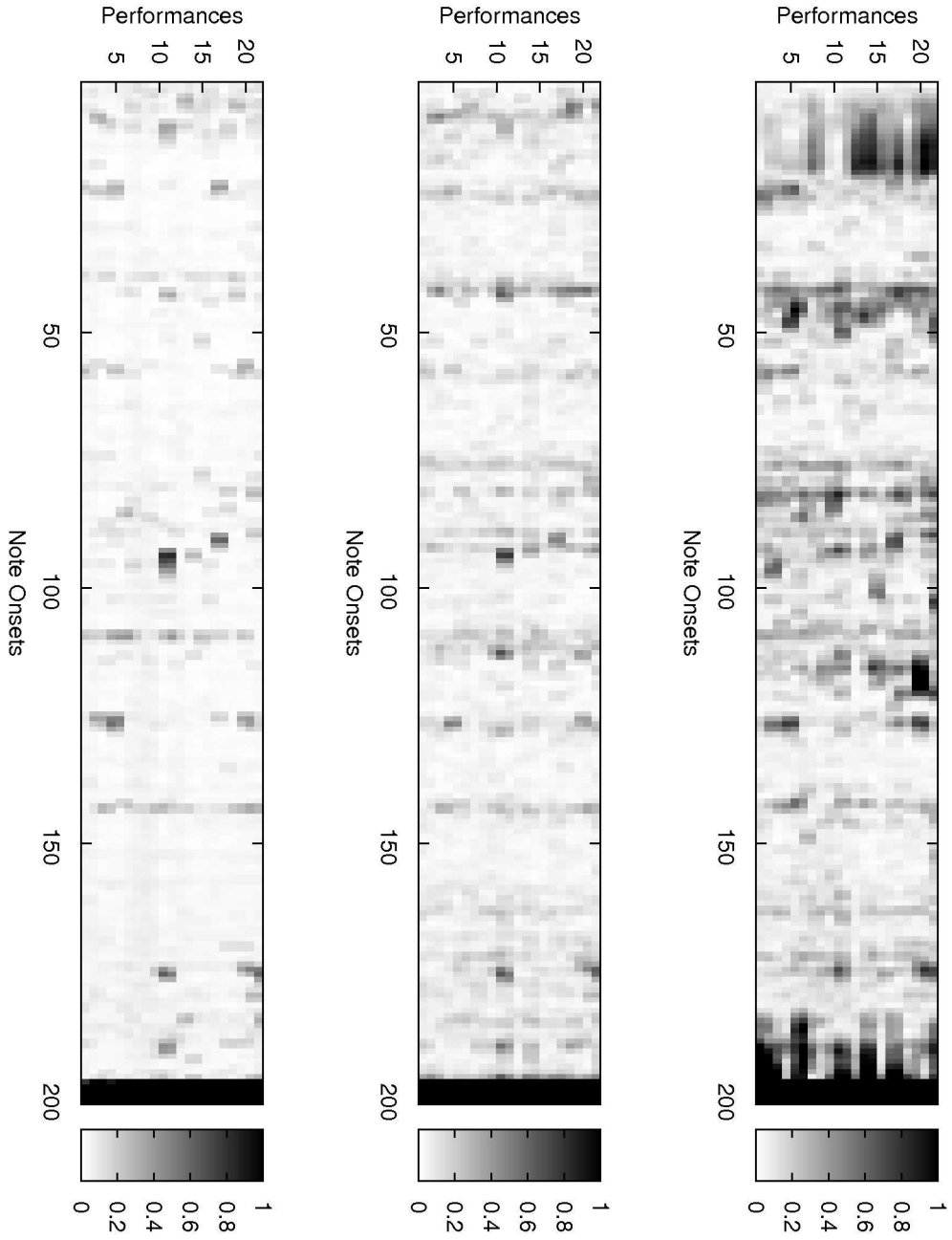

Figure 5.4: Comparison between AOODTW (left), AODTW (center) and SODTW (right), showing the error per note onset of the alignments of the 22 performances of the ballade. The error is symbolized by the grey-level where white means no error and black an error higher or equal 1 second.

Variability between different performances of the same piece

A score following algorithm should perform equally well on different performance styles. To evaluate this I once again had a look at the alignments for the 22 performances of the etude and the ballade.

As can be seen in figure 5.5 the presented extensions to Dixon's algorithm improved not only the accuracy but also decreased the variability between different performances. The low variability can be witnessed in figures 5.3 and 5.4 too.

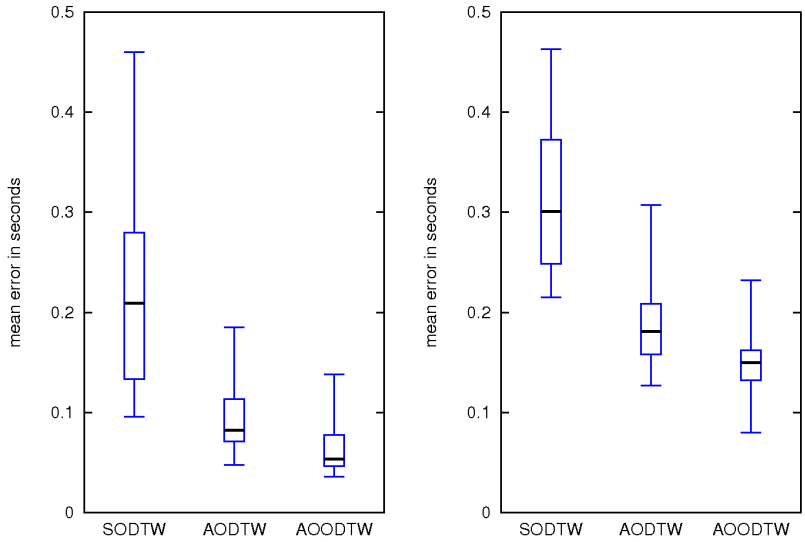

Figure 5.5: Variability between 22 different performances of the etude (left) and the ballade (right) as boxplot of the mean errors of the alignments. The lowest line represents the alignment with the minimum mean error, the lower border of the box the first quartile, the line in the box the median, the upper border of the box the third quartile and the upper line the maximum mean error.

The higher error on the ballade is mainly caused by the problems at the phrase boundary (see chapter 5.1.1).

What's striking is that even the worst performances of AOODTW are still nearly as good as the best of SODTW.

Furthermore, there was no performance which was better aligned by SODTW than by AODTW or which was better aligned by AODTW than by AOODTW.

Evaluation of the implemented improvements

Now I will have a closer look at the implemented improvements. Again the algorithms SODTW, AODTW and AOODTW are evaluated where SODTW serves as baseline. For both AODTW and AOODTW 3 different configurations are presented.

	SODTW	**AODTW**			**AOODTW**		
		A	**B**	**C**	**A**	**B**	**C**
Error == 0.00 sec	0.10	0.12	0.17	0.16	0.14	0.16	0.16
Error <= 0.02 sec	0.27	0.29	0.43	0.40	0.46	0.52	0.53
Error <= 0.04 sec	0.39	0.42	0.58	0.56	0.69	0.79	0.79
Error <= 0.06 sec	0.46	0.51	0.66	0.65	0.74	0.84	0.84
Error <= 0.10 sec	0.56	0.62	0.75	0.76	0.77	0.87	0.87
Error <= 0.20 sec	0.70	0.76	0.87	0.87	0.83	0.92	0.92
Error <= 0.50 sec	0.87	0.93	0.97	0.97	0.94	0.98	0.98
Error <= 1.00 sec	0.96	0.99	0.99	0.99	0.99	0.99	1.00

Table 5.11: Comparison between different implementations (etude)

	SODTW	**AODTW**			**AOODTW**		
		A	**B**	**C**	**A**	**B**	**C**
Error == 0.00 sec	0.08	0.08	0.13	0.13	0.14	0.17	0.18
Error <= 0.02 sec	0.23	0.24	0.37	0.36	0.40	0.50	0.51
Error <= 0.04 sec	0.35	0.36	0.53	0.52	0.59	0.73	0.74
Error <= 0.06 sec	0.44	0.45	0.63	0.62	0.67	0.83	0.84
Error <= 0.10 sec	0.55	0.58	0.75	0.75	0.72	0.88	0.89
Error <= 0.20 sec	0.70	0.72	0.87	0.88	0.78	0.92	0.93
Error <= 0.50 sec	0.86	0.87	0.95	0.95	0.88	0.96	0.96
Error <= 1.00 sec	0.95	0.94	0.97	0.97	0.94	0.97	0.97

Table 5.12: Comparison between different implementations (ballade)

- Configuration A: Only backtracking is used
- Configuration B: Only the reweight towards cheaper diagonal steps is used
- Configuration C: Both improvements are used (this is the standard configuration as used in all other experiments)

As can be seen in tables 5.11 and 5.12 the real source for the improved performance regarding accuracy is the change of the weights to make diagonal steps cheaper while there is little effect of the backtracking compared to SODTW. Sometimes using only reweighting is even better than using both improvements.

Still the backtracking is generally useful regarding stability as can be seen later.

5.1.2 Scenario 2: Bars are left out

For this scenario notes in the midi file were added by repeating bars. So it appears that in the performance the pianist removed some notes. This was done for 1, 2 and 3 consecutive bars, for every 3rd bar (bar sequence) of the etude (between bars 6 and 15) and every 5th bar (bar sequence) of the ballade (between bars 10 and 35), and evaluated for all 22 performances. In total there were 264 runs for the etude and 396 runs for the ballade. The recovery time in beats is calculated as the time in beats between the first correct note after this added sequence and the first note which is aligned with an error lower than 0.2 seconds.

An example of a typical path in this scenario is shown in figure 5.6

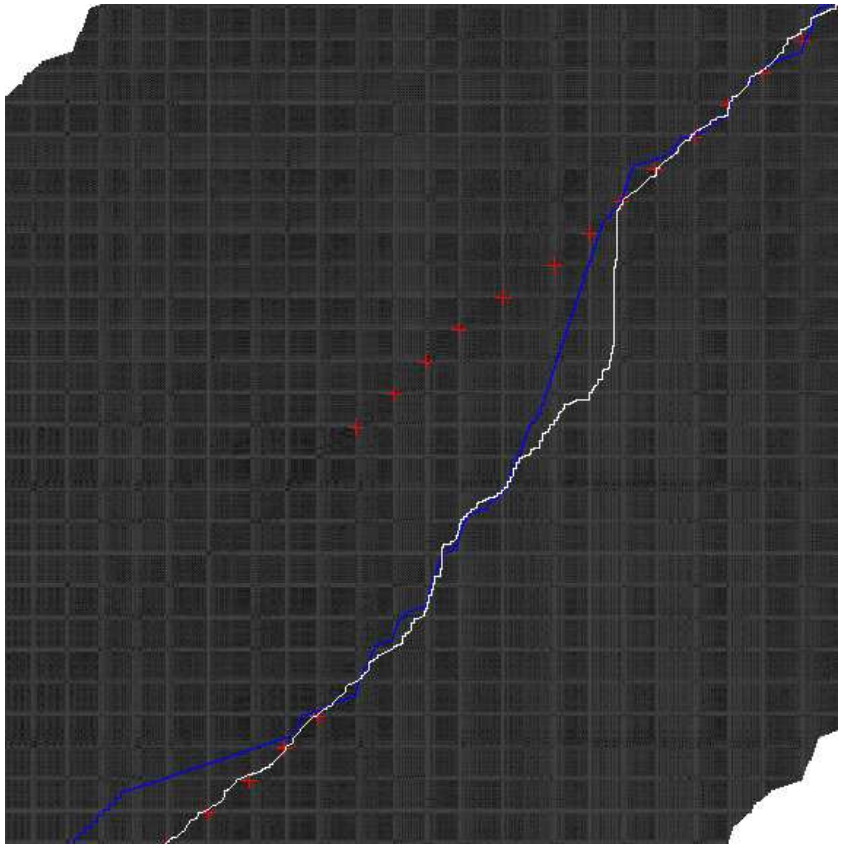

Figure 5.6: An example of a path where 1 bar is left out by the pianist. The red crosses show the correct onsets. Live performance on the x-axis, score representation on the y-axis. The white path is calculated by AODTW, the blue path by SODTW.

As can be seen in table 5.13 and table 5.14 the performance of my implementation with both “improvements” is slightly worse than the performance of Dixon’s algorithm.

	1 Bar		2 Bars		3 Bars	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0	0	0	0	0	0
Rec. <= 1 beats	0.03	0	0	0	0	0
Rec. <= 2 beats	0.41	0.35	0.01	0	0	0
Rec. <= 3 beats	0.86	0.94	0.15	0.03	0	0
Rec. <= 4 beats	1	0.98	0.26	0.19	0	0
Rec. <= 5 beats	1	1	0.45	0.47	0.05	0.05
Rec. <= 6 beats	1	1	0.59	0.61	0.25	0.15
Rec. <= 7 beats	1	1	0.70	0.77	0.35	0.19
Rec. <= 8 beats	1	1	0.75	0.84	0.40	0.29
Rec. <= 9 beats	1	1	0.83	0.94	0.53	0.36
Rec. <= 10 beats	1	1	0.83	0.96	0.59	0.54
Rec. <= 11 beats	1	1	0.83	0.96	0.75	0.58
Rec. <= 12 beats	1	1	0.83	0.97	0.76	0.6
Rec. <= 13 beats	1	1	0.83	0.97	0.77	0.63
Rec. <= 14 beats	1	1	0.83	0.97	0.77	0.64
Rec. <= 15 beats	1	1	0.83	0.97	0.77	0.66
Rec. <= 25 beats	1	1	0.83	0.97	0.80	0.70

Table 5.13: Recovery after n beats (etude)

The reason for this is that my implementation heavily relies on the backward path which of course shows a slightly slower reaction than the pure forward path. But as can be seen in the next two sections this is a desired property because it is much more expensive to correct a path which is ahead of the performance than a path which is behind regarding the score. When a path is ahead the error can only be corrected by "waiting" for the musician, while a path which is behind can be corrected by jumping forward in the score representation.

So the slightly bigger errors in this scenario caused by using backtracking are more than balanced by the better performance in scenario 3 and 4.

	1 Bar		2 Bars		3 Bars	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0.01	0	0	0	0	0
Rec. <= 1 beats	0.01	0	0	0	0	0
Rec. <= 2 beats	0.01	0	0	0	0	0
Rec. <= 3 beats	0.03	0	0	0	0	0
Rec. <= 4 beats	0.03	0	0	0	0	0
Rec. <= 5 beats	0.10	0.04	0	0	0	0
Rec. <= 6 beats	0.23	0.12	0.03	0	0	0
Rec. <= 7 beats	0.23	0.12	0.03	0	0	0
Rec. <= 8 beats	0.52	0.41	0.10	0.03	0	0
Rec. <= 9 beats	0.62	0.54	0.20	0.12	0	0
Rec. <= 10 beats	0.62	0.54	0.20	0.12	0	0
Rec. <= 11 beats	0.74	0.66	0.24	0.23	0	0
Rec. <= 12 beats	0.79	0.70	0.29	0.29	0.01	0
Rec. <= 13 beats	0.79	0.70	0.29	0.29	0.01	0
Rec. <= 14 beats	0.88	0.80	0.40	0.44	0.06	0
Rec. <= 15 beats	0.94	0.88	0.55	0.53	0.08	0
Rec. <= 16 beats	0.94	0.88	0.55	0.53	0.08	0
Rec. <= 17 beats	0.96	0.93	0.67	0.68	0.16	0.02
Rec. <= 18 beats	0.99	0.98	0.73	0.82	0.19	0.05
Rec. <= 19 beats	0.99	0.98	0.73	0.82	0.19	0.05
Rec. <= 20 beats	1	0.98	0.80	0.88	0.32	0.14
Rec. <= 21 beats	1	0.99	0.85	0.94	0.41	0.23
Rec. <= 22 beats	1	0.99	0.85	0.94	0.41	0.23
Rec. <= 23 beats	1	0.99	0.90	0.94	0.53	0.36
Rec. <= 24 beats	1	1	0.90	0.99	0.58	0.40
Rec. <= 25 beats	1	1	0.91	0.99	0.58	0.40
Rec. <= 26 beats	1	1	0.95	0.99	0.66	0.49
Rec. <= 27 beats	1	1	0.98	0.99	0.71	0.55
Rec. <= 28 beats	1	1	0.98	0.99	0.71	0.55
Rec. <= 29 beats	1	1	0.99	0.99	0.77	0.62
Rec. <= 30 beats	1	1	0.99	0.99	0.83	0.69
Rec. <= 40 beats	1	1	1	0.99	0.98	0.94
Rec. <= 50 beats	1	1	1	1	1	0.98

Table 5.14: Recovery after n beats (ballade)

5.1.3 Scenario 3: Additional bars are played

For this scenario notes in the midi file were removed. So it appears that in the performance the pianist added some additional notes. This was done for 1, 2 and 3 consecutive bars, for every 3rd bar (bar sequence) of the etude (between bars 6 and 15) and every 5th bar (bar sequence) of the ballade (between bars 10 and 35), and evaluated for all 22 performances. In total there were 264 runs for the etude and 396 runs for the ballade. The recovery time in beats is calculated as the time in beats between the first correct note and the first note which is aligned with an error lower than 0.2 seconds.

An example of a typical path in this scenario is shown in figure 5.7

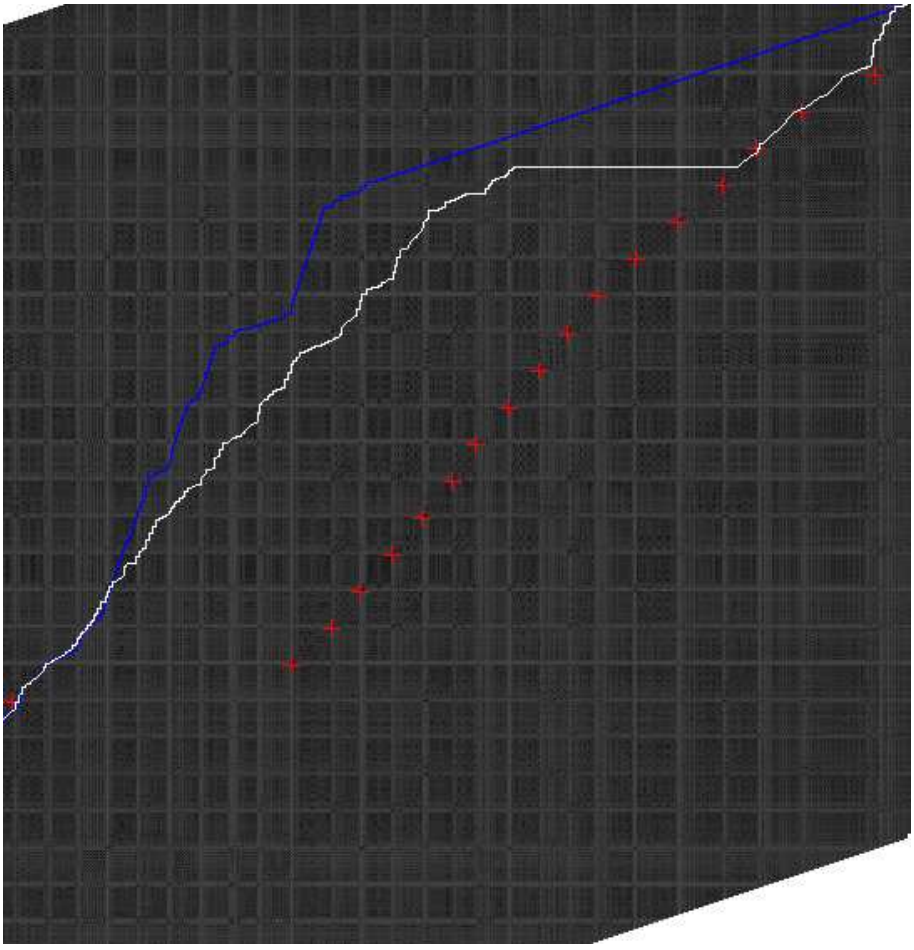

Figure 5.7: An example of a path where 1 bar is added by the pianist. The red crosses show the correct onsets. Live performance on the x-axis, score representation on the y-axis. The white path is calculated by AODTW, the blue path by SODTW.

As already mentioned above my implementation outclasses Dixon's in this scenario by far (see tables 5.15 and 5.16). There are 2 reasons for this:

- By using the backward path the calculation is generally more cautious. This means that the algorithm avoids big jumps of the path unless it is really certain.
- If the backward path ends below the current score coordinate, the recovery – the "waiting" – is a real staying at the current score coordinate (witness the straight horizontal line in 5.7), while in Dixon's implementation the score coordinate is still expanded according to the constraints.

	1 Bar		**2 Bars**		**3 Bars**	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0	0	0	0	0	0
Rec. <= 1 beats	0	0	0	0	0	0
Rec. <= 2 beats	0.01	0.04	0	0	0	0
Rec. <= 3 beats	0.04	0.20	0	0	0	0
Rec. <= 4 beats	0.25	0.53	0	0	0	0
Rec. <= 5 beats	0.55	0.98	0	0.05	0	0
Rec. <= 6 beats	0.74	1	0	0.33	0	0
Rec. <= 7 beats	0.90	1	0.08	0.56	0	0
Rec. <= 8 beats	0.95	1	0.19	0.61	0.01	0
Rec. <= 9 beats	0.99	1	0.34	0.61	0.01	0.02
Rec. <= 10 beats	1	1	0.39	0.63	0.04	0.03
Rec. <= 11 beats	1	1	0.45	0.65	0.05	0.06
Rec. <= 12 beats	1	1	0.46	0.68	0.05	0.09
Rec. <= 13 beats	1	1	0.49	0.68	0.06	0.15
Rec. <= 14 beats	1	1	0.50	0.69	0.06	0.15
Rec. <= 15 beats	1	1	0.52	0.69	0.08	0.15
Rec. <= 16 beats	1	1	0.53	0.69	0.08	0.16
Rec. <= 17 beats	1	1	0.53	0.69	0.09	0.18
Rec. <= 18 beats	1	1	0.53	0.69	0.10	0.21
Rec. <= 19 beats	1	1	0.53	0.69	0.11	0.21
Rec. <= 20 beats	1	1	0.53	0.69	0.13	0.21
Rec. <= 25 beats	1	1	0.53	0.69	0.15	0.26
Rec. <= 30 beats	1	1	0.54	0.69	0.17	0.26

Table 5.15: Recovery after n beats (etude)

	1 Bar		2 Bars		3 Bars	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0	0	0	0	0	0
Rec. <= 1 beats	0	0	0	0	0	0
Rec. <= 2 beats	0	0	0	0	0	0
Rec. <= 3 beats	0	0	0	0	0	0
Rec. <= 4 beats	0	0	0	0	0	0
Rec. <= 5 beats	0	0	0	0	0	0
Rec. <= 6 beats	0	0	0	0	0	0
Rec. <= 7 beats	0	0	0	0	0	0
Rec. <= 8 beats	0.05	0.09	0	0	0	0
Rec. <= 9 beats	0.11	0.18	0	0.01	0	0
Rec. <= 10 beats	0.11	0.18	0	0.01	0	0
Rec. <= 11 beats	0.26	0.42	0.01	0.03	0	0
Rec. <= 12 beats	0.33	0.50	0.02	0.10	0	0
Rec. <= 13 beats	0.33	0.50	0.02	0.10	0	0
Rec. <= 14 beats	0.46	0.62	0.05	0.13	0	0
Rec. <= 15 beats	0.53	0.72	0.06	0.23	0	0
Rec. <= 16 beats	0.53	0.72	0.06	0.23	0	0
Rec. <= 17 beats	0.62	0.80	0.18	0.42	0	0.01
Rec. <= 18 beats	0.74	0.91	0.27	0.56	0	0.03
Rec. <= 19 beats	0.74	0.91	0.27	0.56	0	0.03
Rec. <= 20 beats	0.81	0.94	0.42	0.79	0	0.09
Rec. <= 21 beats	0.86	0.97	0.51	0.86	0.01	0.10
Rec. <= 22 beats	0.86	0.97	0.51	0.86	0.01	0.10
Rec. <= 23 beats	0.88	0.98	0.64	0.94	0.03	0.18
Rec. <= 24 beats	0.91	1	0.71	0.94	0.05	0.24
Rec. <= 25 beats	0.91	1	0.71	0.94	0.05	0.24
Rec. <= 26 beats	0.94	1	0.79	0.99	0.11	0.32
Rec. <= 27 beats	0.97	1	0.82	0.99	0.16	0.40
Rec. <= 28 beats	0.97	1	0.82	0.99	0.16	0.40
Rec. <= 29 beats	0.98	1	0.86	0.99	0.26	0.55
Rec. <= 30 beats	0.98	1	0.90	0.99	0.30	0.62
Rec. <= 40 beats	0.99	1	0.98	1	0.62	0.99
Rec. <= 50 beats	1	1	0.98	1	0.75	0.99

Table 5.16: Recovery after n beats (ballade)

5.1.4 Scenario 4: False notes are played

This scenario was simulated by replacing all notes in a whole bar in the midi file by a augmented fourth. This was done for 1, 2 and 3 consecutive bars, for every 3rd bar (bar sequence) of the etude (between bars 6 and 15) and every 5th bar (bar sequence) of the ballade (between bars 10 and 35), and evaluated for all 22 performances. In total there were 264 runs for the etude and 396 runs for the ballade. The calculation of the recovery time in beats starts with the first correctly played note after these bars.

Of course this scenario is very unlikely to happen. Nonetheless, it was evaluated to find out how fast the algorithm recovers after large errors. Small errors like leaving out one single note did not pose a problem at all to the algorithms.

An example of a typical path in this scenario is shown in figure 5.8

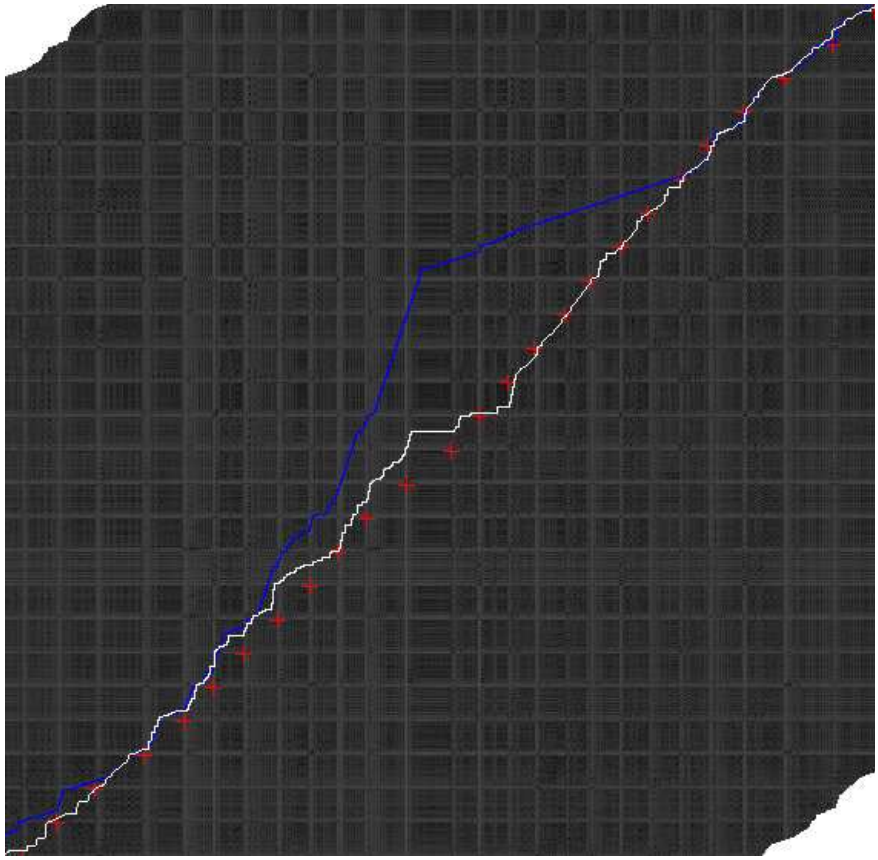

Figure 5.8: An example of a path where 1 bar is changed by the pianist. The red crosses show the correct onsets. Live performance on the x-axis, score representation on the y-axis. The white path is calculated by AODTW, the blue path by SODTW.

	1 Bar		2 Bars		3 Bars	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0.05	0.09	0	0.36	0.10	0.42
Rec. <= 1 beats	0.10	0.53	0.19	0.82	0.14	0.66
Rec. <= 2 beats	0.27	0.67	0.28	0.90	0.20	0.74
Rec. <= 3 beats	0.47	0.89	0.36	0.91	0.31	0.88
Rec. <= 4 beats	0.55	0.93	0.42	0.97	0.39	0.91
Rec. <= 5 beats	0.61	0.94	0.45	0.97	0.51	0.95
Rec. <= 6 beats	0.69	0.99	0.51	0.97	0.57	0.97
Rec. <= 7 beats	0.76	0.99	0.63	0.99	0.57	0.97
Rec. <= 8 beats	0.81	0.99	0.65	0.99	0.57	0.97
Rec. <= 9 beats	0.89	1	0.65	0.99	0.58	0.97
Rec. <= 10 beats	0.89	1	0.65	0.99	0.61	0.97
Rec. <= 11 beats	0.89	1	0.65	0.99	0.66	0.97
Rec. <= 12 beats	0.89	1	0.68	0.99	0.66	0.97
Rec. <= 13 beats	0.90	1	0.70	0.99	0.66	0.97
Rec. <= 14 beats	0.91	1	0.70	0.99	0.66	0.97
Rec. <= 15 beats	0.92	1	0.70	0.99	0.67	0.97
Rec. <= 20 beats	0.92	1	0.70	0.99	0.67	0.97

Table 5.17: Recovery after n beats (etude)

I originally planned to evaluate this scenario as scenarios 2 and 3 – counting the beats until the algorithm recovered from the error. But due to the distorted matrix huge errors often occur some onsets after the incorrect notes. For this reason I accepted the path calculation as "recovered" if the current onset is aligned with an error lower or equal 0.2 seconds *and* the next 10 onsets are aligned with an error not higher than 0.4 seconds each.

This error occurs if the algorithm for a short time finds a "better" alignment by matching the incorrectly played notes to different notes in the score and recovers some onsets later. Interestingly this effect more often happend with the ballade than with the etude and more likely happens for the change of 1 bar – which explains the values in the first rows of table 5.17, where the algorithm sometimes recovers faster after the change of 2 and 3 bars. But in general larger errors by the musician of course imply a longer recovery time for the algorithms.

Again AODTW shows the better performance (see tables 5.17 and 5.18). This is caused by both the use of backtracking and changing of the weights. Both make the path calculation more cautious and the path lies nearer to the diagonal while SODTW produces a very unstable path which sometimes leads to hard to correct errors.

At smaller errors the speed of the recovery is the main difference between SODTW and

	1 Bar		2 Bars		3 Bars	
	SODTW	**AODTW**	**SODTW**	**AODTW**	**SODTW**	**AODTW**
Rec. == 0 beats	0.10	0.55	0.05	0.43	0.02	0.18
Rec. <= 1 beats	0.10	0.55	0.05	0.43	0.02	0.18
Rec. <= 2 beats	0.19	0.69	0.11	0.58	0.03	0.31
Rec. <= 3 beats	0.23	0.71	0.18	0.66	0.07	0.43
Rec. <= 4 beats	0.23	0.71	0.18	0.66	0.07	0.43
Rec. <= 5 beats	0.33	0.78	0.25	0.75	0.17	0.55
Rec. <= 6 beats	0.34	0.83	0.30	0.79	0.24	0.61
Rec. <= 7 beats	0.34	0.83	0.30	0.79	0.24	0.61
Rec. <= 8 beats	0.36	0.84	0.39	0.81	0.27	0.72
Rec. <= 9 beats	0.42	0.85	0.43	0.83	0.27	0.73
Rec. <= 10 beats	0.42	0.85	0.43	0.83	0.27	0.73
Rec. <= 11 beats	0.44	0.86	0.38	0.85	0.30	0.73
Rec. <= 12 beats	0.47	0.89	0.54	0.85	0.31	0.73
Rec. <= 13 beats	0.47	0.89	0.56	0.85	0.31	0.73
Rec. <= 14 beats	0.52	0.93	0.56	0.89	0.33	0.80
Rec. <= 15 beats	0.58	0.93	0.59	0.89	0.33	0.80
Rec. <= 16 beats	0.58	0.93	0.69	0.89	0.33	0.80
Rec. <= 17 beats	0.62	0.94	0.62	0.89	0.37	0.85
Rec. <= 18 beats	0.67	0.94	0.63	0.89	0.41	0.88
Rec. <= 19 beats	0.67	0.94	0.63	0.89	0.41	0.88
Rec. <= 20 beats	0.67	0.95	0.66	0.90	0.44	0.89
Rec. <= 25 beats	0.70	0.96	0.70	0.95	0.54	0.91
Rec. <= 30 beats	0.75	0.98	0.80	0.95	0.56	0.92
Rec. <= 35 beats	0.83	0.98	0.80	0.96	0.64	0.93

Table 5.18: Recovery after n beats (ballade)

AODTW and AOODTW. But while at larger errors AODTW and AOODTW still recover most of the time without a problem, SODTW quite often got completely lost.

5.1.5 Scenario 5: Additions, Deletions and Changes of Notes

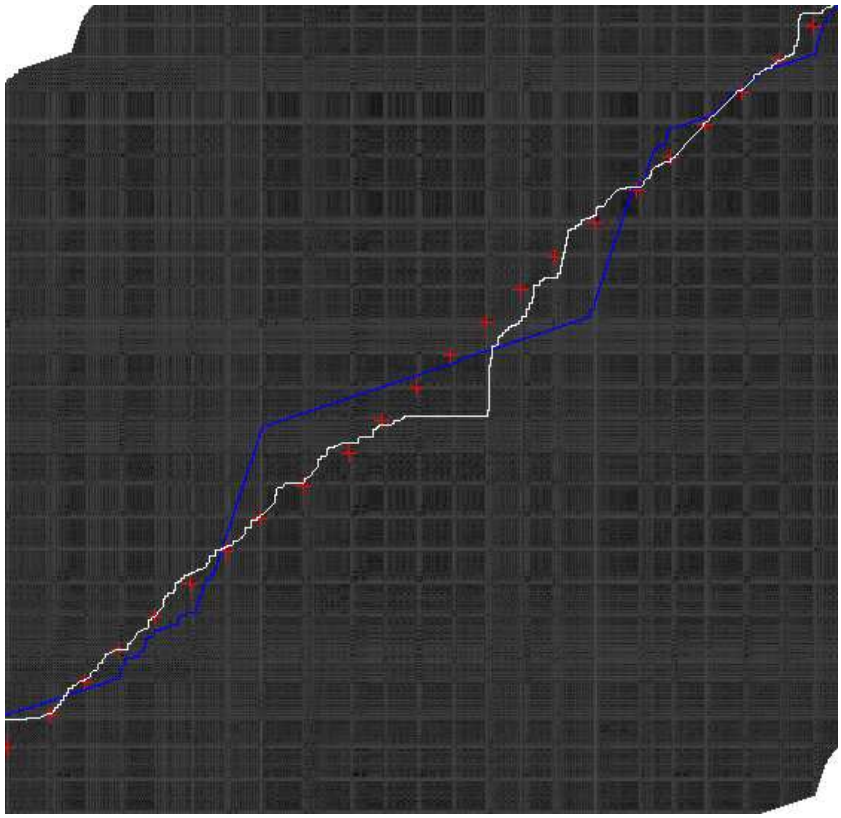

Figure 5.9: An example of a path where some notes are added, changed and removed by the pianist. The red crosses show the correct onsets. Live performance on the x-axis, score representation on the y-axis. The white path is calculated by AODTW, the blue path by SODTW.

Now I'm going to look at a more realistic scenario: a pianist who adds, removes or changes single notes by error. In fact I simulated a very incompetent pianist. The test data was prepared as follows:

1. Notes were removed with a probability of 10%.
2. The accidental and the note name of the remaining notes were changed randomly with a probability of 10%.
3. Existing notes were copied somewhere else between the first and the last midi tick with a probability of 10%. The octave remained unchanged but the accidental and the note name were changed randomly.

This was done 20 times to produce faulty representations of the etude and of the ballade. They were aligned to the 22 performances. In total for each piece 440 alignments were computed.

A typical path for such a faulty performance is shown in figure 5.9.

Of course these alignments (see tables 5.19 and 5.20) are much worse compared to alignments of correct performances.

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Error == 0.00 sec	0.06	0.14	0.04	0.09	0.09
Error <= 0.02 sec	0.16	0.35	0.11	0.25	0.34
Error <= 0.04 sec	0.25	0.50	0.17	0.35	0.52
Error <= 0.06 sec	0.31	0.59	0.22	0.42	0.58
Error <= 0.10 sec	0.40	0.70	0.29	0.53	0.62
Error <= 0.20 sec	0.54	0.81	0.43	0.68	0.71
Error <= 0.50 sec	0.76	0.93	0.66	0.86	0.87
Error <= 1.00 sec	0.92	0.99	0.85	0.95	0.96

Table 5.19: Overall performance on the heavily changed etude given as cumulative frequencies (see text)

	SODTW (MDE)	**AODTW (MDE)**	**SODTW (EDE)**	**AODTW (EDE)**	**AOODTW (EDE)**
Error == 0.00 sec	0.03	0.08	0.02	0.05	0.07
Error <= 0.02 sec	0.09	0.22	0.06	0.15	0.22
Error <= 0.04 sec	0.14	0.33	0.09	0.23	0.34
Error <= 0.06 sec	0.18	0.42	0.13	0.30	0.41
Error <= 0.10 sec	0.25	0.53	0.18	0.41	0.47
Error <= 0.20 sec	0.37	0.66	0.30	0.57	0.55
Error <= 0.50 sec	0.58	0.84	0.49	0.75	0.72
Error <= 1.00 sec	0.78	0.94	0.68	0.88	0.87

Table 5.20: Overall performance on the heavily changed ballade given as cumulative frequencies (see text)

Nevertheless, they are still reasonable. A closer look at the alignments shows that after errors the algorithm recovers very fast if the following 2 or more notes are played correctly. Only if there are many mistakes in a short time the algorithm needs some more correct notes

to recover. Despite this huge amount of errors the algorithm got never lost.

5.1.6 Summary

As the evaluation of the *single matcher* shows all proposed extensions improved the performance of the algorithm. While the simple reweighting led to a huge gain in accuracy, the use of a combination of backward and forward paths increased the ability to correct large errors. Furthermore, the direct use of the MIDI data in a search process for note onsets (AOODTW) again led to higher accuracy.

Larger alignment errors still occur at phrase boundaries due to the huge variations in tempo and discontinuities in timing. Still, after phrase boundaries the algorithm recovers easily. Nonetheless, a solution to this problem should be a topic for future work.

To summarize, the *single matcher* is a robust score following algorithm which is capable of controlling the page-turning hardware with the needed accuracy.

5.2 The Multi Matcher

The multi matcher was tested on edited versions of the ballade and the etude where repetitions were inserted. To the ballade a repetition starting at bar 10 and lasting 10 bars and to the etude a repetition starting at bar 5 and lasting 8 bars were added. Again to this data the 22 interpretations were aligned.

The following marksfiles were used for the alignment:

Listing 5.1: Etude

```
$description start
$Auto-Generated: op10_3_1_a8_5
$description stop
$midi_file=[path_to_midi]/op10_3_1_a8_5.mid
$audio_file=[path_to_audio]/op10_3_1_a8_5.wav
$parts start
0
5
13
21
$parts stop
```

Listing 5.2: Ballade

```
$description start
$Auto-Generated: op38_1_a10_10
$description stop
$midi_file=[path_to_midi]/op38_1_a10_10.mid
$audio_file=[path_to_audio]/op38_1_a10_10.wav
$parts start
0
10
20
30
$parts stop
```

So for both performances the same 4 parts were defined.

1. From the start of the piece until the start of the repetition
2. From the start of the repetition until the end of the first repetition
3. From the start of the second repetition to the end of the second repetition

4. From the end of the second repetition to the end of the piece

As in the performances no repetition is played there are 2 possible correct paths through the piece: 1-2-4 and 1-3-4. As described in chapter 4.3.2 the decision which matcher to take is made after 4 bars while ensuring these have a length of at least 10 seconds according to the base audio file. This means that a full 500×500 cost matrix has been calculated for the newly started matching instances – as already mentioned the path calculation has already started after 50 frames – which proved enough for stable results. Of course it would have been possible to delay the decision until one of the matchers reaches the next border of a part or a page-turning mark.

Using the described strategy, for all 22 alignments of the etude and all 22 alignments of the ballade the path calculation followed one of the correct paths which gives a success rate of 100%.

5.3 First live tests

During the work on this theses 2 live tests were done (for some impressions see figure 5.10). One was done with a simple electronic piano, one with a grand piano. The audio signal was recorded with a single microphone. No preprocessing was done. Both times an advanced hobby pianist played two pieces of Chopin: The ballade op. 52 in A-flat major and the etude op. 25, No. 11 in A-minor.

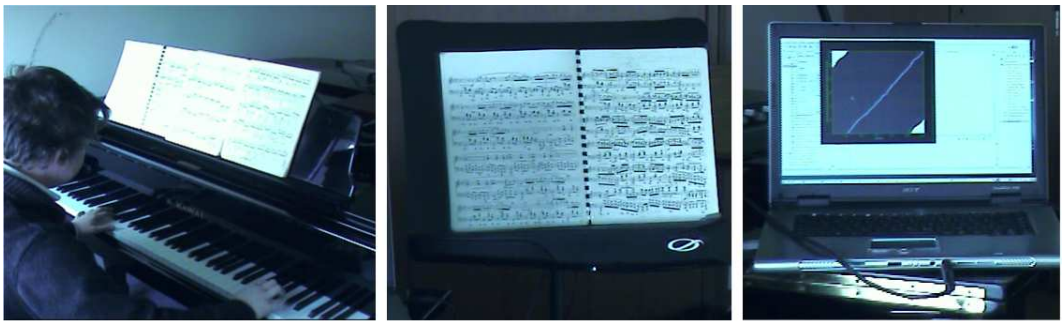

Figure 5.10: Some impressions of the second live test

In both tests everything worked fine and the algorithm showed sufficient results for the domain of the page-turner. It turned out that the more onsets are played (the faster the piece is) the better the alignment becomes. So even the very fast etude was aligned perfectly despite some errors by the pianist.

We encountered only one real problem during our tests. During very quiet parts the algorithm is not able to follow the performance correctly. As in the tested pieces pianissimo parts

are very short the algorithm recovered without a problem but there will be problems with longer pianissimos. One possible solution to this could be to use an audio compressor.

6 Future Work and Conclusion

In this master thesis an automatic page-turner based on an online dynamic time warping algorithm was presented. As the extensive evaluation showed the proposed extensions enhanced both the stability and the accuracy of the algorithm.

This thesis is mainly a demonstration of what is already possible in score following. Of course the algorithm can prove to be useful in many score following tasks, e.g. live visualization and automatic accompaniment. The automatic page-turner may actually be used during concerts. But I think much more interesting is the idea of building a practicing environment for musicians around this algorithm.

On the technical side there are some directions for future work. First thing to do would be to improve the algorithms behaviour during pianissimo passages – maybe preprocessing using an audio compressor could already do the trick – and at phrase boundaries. As the concept of multiple matchers is limited to fixed parts of the piece one could also think about better ways to analyse the structure and determine which part of the piece the pianist is playing.

While this thesis focused on an online algorithm one of the proposed improvements could especially be of use for offline DTW, namely the incorporation of the MIDI data into the matching process by taking the path only as an approximate solution and searching around the path for onsets.

As the algorithm was designed with piano music in mind there were no tests by now on other types of classical or popular music. I do not see many reasons why it should not perform well on other kinds of music but as the algorithm uses an onset detection function non-percussive instruments could be more problematic. Of course further evaluation has to be done.

To summarize, the presented algorithm was tested twice in live environments and showed high accuracy and – what is even more important for an automatic page-turner – high stability. It recovered fast from mistakes by the pianist and fulfilled the page turning task reliably.

Bibliography

[Allen and Dannenberg, 1990] Allen, P. and Dannenberg, R. (1990). Tracking musical beats in real time. In *Proceedings of the International Computer Music Conference*, volume 140-143. International Computer Music Association.

[Baird et al., 1990] Baird, B., Blevins, D., and Zahler, N. (1990). The artificially intelligent computer performer: The second generation. in interface. *Journal of New Music Research*, 19:197–204.

[Baird et al., 1993] Baird, B., Blevins, D., and Zahler, N. (1993). Artificial intelligence and music: Implementing an interactive computer performer. *Computer Music Journal*, 17(2):73–79.

[Bloch and Dannenberg, 1985] Bloch, J. and Dannenberg, R. (1985). Real-time computer accompaniment of keyboard performances. In Truax, B., editor, *Proceedings of the International Computer Music Conference*, pages 279–289, San Francisco. International Computer Music Association.

[Cano et al., 1999] Cano, P., Loscos, A., and Bonada, J. (1999). Score-performance matching using hmms. In *Proceedings of the ICMC*.

[Cont, 2004] Cont, A. (2004). Improvement of observation modeling for score following. Master's thesis, University of Paris 6, IRCAM, Paris.

[Cont, 2006] Cont, A. (2006). Realtime audio to score alignment for polyphonic music instruments using sparse non-negative constraints and hierarchical hmms. In *Proceedings of IEEE ICASSP*, Toulouse. IEEE.

[Cont et al., 2007] Cont, A., Schwarz, D., Schnell, N., and Raphael, C. (2007). Evaluation of real-time audio-to-score alignment. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*, Vienna, Austria.

[Dannenberg, 1984] Dannenberg, R. (1984). An on-line algorithm for real-time accompaniment. In *Proceedings of the International Computer Music Conference*, pages 193–198, San Francisco. International Computer Music Association.

[Dannenberg and Hu, 2003] Dannenberg, R. and Hu, N. (2003). Polyphonic audio matching for score following and intelligent audio editors. In *Proceedings of the International*

Computer Music Conference, pages 27–34, San Francisco. International Computer Music Association.

[Dannenberg and Mont-Reynaud, 1987] Dannenberg, R. and Mont-Reynaud, B. (1987). Following an improvisation in real time. In *Proceedings of the International Computer Music Conference*, pages 241–248. Computer Music Association.

[Dannenberg and Mukaino, 1988] Dannenberg, R. and Mukaino, H. (1988). New techniques for enhanced quality of computer accompaniment. In Lischka, C. and Fritsch, J., editors, *Proceedings of the 14th International Computer Music Conference*, San Francisco. International Computer Music Association.

[Desain et al., 1997] Desain, P., Honing, H., and Heijink, H. (1997). Robust score-performance matching: Taking advantage of structural information. In *Proceedings of the International Computer Music Conference*, pages 337–340, San Francisco. International Computer Music Association.

[Dixon, 2005] Dixon, S. (2005). An on-line time warping algorithm for tracking musical performances. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1727–1728, Edinburgh.

[Dixon and Widmer, 2005] Dixon, S. and Widmer, G. (2005). Match: A music alignment tool chest. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*.

[Durbin et al., 1998] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis*. Cambridge University Press.

[Gavrila and Davis, 1995] Gavrila, D. M. and Davis, L. S. (1995). Towards 3-d model-based tracking and recognition of human movement. In Bichsel, M., editor, *International Workshop on Face and Gesture Recognition*, pages 272–277.

[Goebl, 1999] Goebl, W. (1999). Numerisch-klassifikatorische interpretationsanalyse mit dem boesendorfer computerfluegel. Master's thesis, University of Vienna.

[Grubb and Dannenberg, 1997] Grubb, L. and Dannenberg, R. (1997). A stochastic method of tracking a vocal performer. In *Proceedings of the International Computer Music Conference*, pages 301–308, San Francisco. International Computer Music Association.

[Heijink et al., 2000] Heijink, H., Desain, P., Honing, H., and Windsor, W. L. (2000). Make me a match: An evaluation of different approaches to score-performance matching. *Computer Music Journal*, 24(1):43–56.

[Hu et al., 2003] Hu, N., Dannenberg, R., and Tzanetakis, G. (2003). Polyphonic audio matching and alignment for music retrieval. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 185–188, New York. IEEE.

[Itakura, 1975] Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Speech and Audio Processing*, 23:52–72.

[Kapanci and Pfeffer, 2005] Kapanci, E. and Pfeffer, A. (2005). Signal-to-score music transcription using graphical models. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, Edinburgh.

[Kaprykowsky and Rodet, 2006] Kaprykowsky, H. and Rodet, X. (2006). Globally optimal short-time dynamic time warping applications to score to audio alignment. In *IEEE ICASSP*, Toulouse.

[Lung and de Freitas, 2004] Lung, D. and de Freitas, N. (2004). Beat tracking the graphical model way. In *Advances in Neural Information Processing Systems*.

[Mueller et al., 2005] Mueller, M., Kurth, F., and Clausen, M. (2005). Audio matching via chroma-based statistical features. In *Proceedings of the 5th International Conference on Music Information Retrieval*, London, GB.

[Mueller et al., 2004] Mueller, M., Kurth, F., and Roeder, T. (2004). Towards an efficient algorithm for automatic score-to-audio synchronization. In *Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain.

[Mueller et al., 2006] Mueller, M., Mattes, H., and Kurth, F. (2006). An efficient multiscale approach to audio synchronization. In *Proceedings of the 7th International Conference on Music Information Retrieval*, Victoria, Canada.

[Murphy, 2001] Murphy, K. (2001). An introduction to graphical models. Technical report, University of California, Berkeley.

[Orio and Dechelle, 2001] Orio, N. and Dechelle, F. (2001). Score following using spectral analysis and hidden markov models. In *Proceedings of the ICMC*, Havana, Cuba.

[Orio and Schwarz, 2001] Orio, N. and Schwarz, D. (2001). Alignment of monophonic and polyphonic music to a score. In *Proceedings of the ICMC*, Havana, Cuba.

[Pardo and Birmingham, 2001] Pardo, B. and Birmingham, W. (2001). Following a musical performance from a partially specified score. In *Proceedings of the 2001 Multimedia Technology and Applications Conference*, Irvine, California.

[Pardo and Birmingham, 2002] Pardo, B. and Birmingham, W. (2002). Improved score following for acoustic performances. In *Proceedings of the International Computer Music Conference*, Gothenburg, Sweden.

[Pardo and Birmingham, 2005] Pardo, B. and Birmingham, W. (2005). Modeling form for on-line following of musical performances. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania.

[Puckette, 1990] Puckette, M. (1990). Explode: A user interface for sequencing and score following. In *Proceedings of the International Computer Music Conference*, pages 259–261, San Francisco. International Computer Music Association.

[Puckette, 1995] Puckette, M. (1995). Score following using the sung voice. In *Proceedings of the International Computer Music Conference*, pages 175–178, San Francisco. International Computer Music Association.

[Puckette and Lippe, 1992] Puckette, M. and Lippe, C. (1992). Score following in practice. In *In Proceedings of the ICMC*, pages 182–185.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285.

[Rabiner and Juang, 1993] Rabiner, L. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ.

[Raphael, 1999] Raphael, C. (1999). Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370.

[Raphael, 2001] Raphael, C. (2001). A bayesian network for real-time musical accompaniment. *Neural Information Processing Systems (NIPS)*, 14:2001.

[Raphael, 2003] Raphael, C. (2003). Orchestra in a box: A system for real time musical accompaniment. *IJCAI2003 workshop program, APP-5*, 5:5–10.

[Raphael, 2004a] Raphael, C. (2004a). A hybrid graphical model for aligning polyphonic audio with musical scores. In *Proceedings of the 5th International Conference on Music Information Retrieval*.

[Raphael, 2004b] Raphael, C. (2004b). Musical accompaniment systems. *Chance Magazine*, 17(4):17–22.

[Rath and Manmatha, 2002] Rath, T. and Manmatha, R. (2002). Word image matching using dynamic time warping. Technical Report MM-38, Center for Intelligent Information Retrieval, University of Massachusetts Amherst.

[Sakoe and Chiba, 1978] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Speech and Audio Processing*, 26:43–49.

[Schreck-Ensemble, 2001] Schreck-Ensemble (2001). Comparser. http://kmt.hku.nl/ pieter/-SOFT/CMP/doc/cmp.html.

[Schwarz, 2004] Schwarz, D. (2004). *Data-Driven Concatenative Sound Synthesis*. PhD thesis, Ircam-Centre Pompidou.

[Schwarz et al., 2004] Schwarz, D., Orio, N., and Schnell, N. (2004). Robust polyphonic midi score following with hidden markov models. In *Proceedings of the International Computer Music Conference*, Miami, Florida.

[Toiviainen, 1998] Toiviainen, P. (1998). An interactive midi accompanist. *Computer Music Journal*, 22(4):63–75.

[Vantomme, 1995] Vantomme, J. D. (1995). Score following by temporal patterns. *Computer Music Journal*, 19(3):50–59.

[Vercoe, 1984] Vercoe, B. (1984). The synthetic performer in the context of live performance. In *Proceedings of the International Computer Music Conference*, pages 199–200, San Francisco. International Computer Music Association.

[Vercoe and Puckette, 1985] Vercoe, B. and Puckette, M. (1985). Synthetic rehearsal: Training the synthetic performer. In *Proceedings of the International Computer Music Conference*, pages 275–278, San Francisco. International Computer Music Association.

[Vinciarelli, 2002] Vinciarelli, A. (2002). A survey on off-line cursive word recognition. *Pattern Recognition*, 35(07):1433–1446.