universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Automated queueing in opera.guru through live music alignment“

verfasst von / submitted by

Dennis Gubbels BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. Dr.Peter Reichl, Privatdoz.
Mitbetreut von / Co-Supervisor:	Dipl.-Ing. Dr.techn. Oliver Hödl

Abstract

opera.guru, the smart subtitle solution, provides users with subtitles during live performances, particularly Operas. At present, opera.guru requires human control for every performance to release subtitles with the correct timing. In this project, an automated system is created that replaces this role, provided that a single recorded performance is available. The system is built on real-time music sequence alignment through the Online Dynamic Time Warping method. It is successfully integrated into the existing opera.guru service. A number of novel extensions to the base algorithm and existing improvements are developed and evaluated. Of these, a new matrix pre-processing step to reduce linear patterns proves especially effective, producing a significant reduction in large alignment-errors. In simulated testing, the system proved capable of effectively tracking a performance of Verdi's Rigoletto, but was not able to guarantee high accuracy throughout the full performance. During a user experience trial, the algorithm effectively aligned an excerpt from Rigoletto with sufficient accuracy for the opera.guru use case. The survey responses from the audience of this trial indicate that there is no significant difference in user experience between a human operator and the automated system.

Zusammenfassung

opera.guru, "the smart subtitle solution", bietet eine Lösung Opern und andere Live-Aufführungen in Echtzeit zu untertiteln. Heutzutage ist ein Bediener erforderlich, der während der Aufführung Untertitelmeldungen zum richtigen Zeitpunkt sendet. Im Rahmen dieser Arbeit wurde ein automatisches System entwickelt, das diese Aufgabe übernehmen kann, sofern eine Tonaufnahme vorab vorhanden ist. Das System basiert auf dem Abgleich von Musiksequenzen, verwendet die Methode des Online Dynamic Time Warping und wurde erfolgreich in das bestehende System von opera.guru integriert.

Zusätzlich wurden einige Erweiterungen des Basisalgorithmus entwickelt und analysiert. Besonders effektiv ist eine Erweiterung, die Kostenmatrizen verarbeitet, womit lineare Muster reduziert werden. Dies führt zu einer signifikanten Reduktion von großen Abgleichs-Fehlern. Simulierte Tests erwiesen, dass das System eine Aufführung von der Oper Verdi's Rigoletto zwar effektiv verfolgen kann, mit der aktuellen Version aber nicht garantieren kann, dass der Abgleich während der gesamten Aufführung in hohe Maße akkurat bleibt. In einem Benutzertest, bei dem ein Teil von Rigoletto vorgeführt wurde, erwies sich das System bei der Automatisierung von opera.guru als effektiv. Die dazugehörige Umfrage zeigte, dass die Teilnehmer mit dem System in gleichem Maße zufrieden sind wie mit der manuellen Bedienung.

Contents

List of Tables

List of Figures

1 Introduction

opera.guru[1], the smart subtitle solution, is a mobile and web application providing smart subtitles live on personal devices in the users' language of choice. To make this work, opera.guru requires a human operator to follow the performance and release each subtitle message at the correct time. This task is referred to as queueing. The goal of this project is to automate the queueing process for opera.guru. The concept is to use a previous audio recording of the performance as well as the timing of the subtitle messages on this recording and adapt this to the live performance. Since any live performance will have differences in tempo as well as pauses, this requires an alignment that matches every moment in the live audio to the corresponding moment in the recording. It is the task of the music alignment algorithm to find this. This introductory chapter will first discuss opera.guru and the principles of music alignment. Then the research questions that this thesis attempts to answer are given. The chapter ends an outlook on the rest of the thesis.

1.1 opera.guru

1.1.1 What is opera.guru

opera.guru is developed by the COSY research group at the faculty of Computer Science at the University of Vienna[2].

opera.guru was started in 2015 with the Salome Experience and live open-air viewings at the Wiener Staatsoper [54]. The plot of an opera can be difficult to follow for newcomers who do not speak the original language. This is especially true for open-air performances, where live subtitle displays are generally not available. Here, opera.guru provides the solution by allowing users to view subtitles as well as to read back previous plot-points including summaries of previous scenes.

Although opera.guru is designed for opera performances, it is a flexible platform and can be used for a multitude applications that are enhanced by serving content to an audience live.

opera.guru has previously been used in the Vienna State Opera and Theater Regensburg among others and is currently used by Oper Wuppertal.

1.1.2 opera.guru today

Since the initial version in 2015, a number of major releases have been made of opera.guru for the front-end as well as the back-end. The latest version uses a Vue-based web-application[3], with both Android and IOS versions available which were built using the Cordova platform. The back-end runs a Spring-based Java application. A fully featured

[1] http://opera.guru
[2] https://cosy.cs.univie.ac.at/research/projects/project/275/
[3] http://opera.guru/web

management front-end, also Vue based, is available for the opera houses to create, manage and run performance projects.

An opera.guru project consists of multiple levels. The primary building block of a performance is a *chunk*. This is a single piece of content sent directly to the users. A chunk most commonly contains only text and thereby forms a single subtitle line. Besides plain text, chunks can also contain rich text or media such as images. Chunks in opera.guru are generally descriptive in style rather than direct transcription subtitles. The goal is to allow users to follow the plot more easily with minimal distraction. A collection of chunks is grouped in a *partition*. The partition itself also contains a piece of content. When a partition is reached, the chunks in it are collapsed and the partition content is shown as a summary of its contents. Users can always reopen a partition to read back the individual chunks. besides chunks and partitions, a special *instant* message type is also available. This type of message exists outside the pre-defined structure of the project and can be written, edited and sent during any moment of the performance.

opera.guru also provides multiple profiles for a performance. The user can select which profile they want to follow. All chunks and partitions have separate content for each profile. Profiles are mainly used to provide translations to users. Certain chunks can be left empty for a profile to provide options for the level of detail the user wants to receive, though these are still present in the project structure. Partitions are always sent and displayed, even if they have no content. A language code can also be attached to a profile, this allows the app to automatically select this profile if it matches the users' app and device settings.

1.1.3 Queueing

In order to run a live performance, the system needs to be told when to release a new chunk. This is the task of queueing. Queueing requires an operator who is familiar with both the piece performed and the project content and also has a good ear for timing. When a piece is performed multiple times this may be a repetitive task. The goal of this project is to create a system that can automate the task of queueing without sacrificing quality.

In order to achieve this, music alignment techniques will be used. Given a good reference recording of a previous performance (including both the audio and the queueing timestamps), music alignment should allow for a system to recreate the queueing of this reference on the live performance taking into account differences in timing and tempo.

1.2 Music Alignment

Music Alignment is the task of finding an alignment of two musical recordings such that the same notes in both recordings are matched up. Because music is never played at exactly the same tempo, and the difference in tempo can vary throughout the piece, this relation between the recordings is constantly changing. Generally, the goal is to find for every point in the *target* the corresponding position in the *reference*. This can be done live, or online, where the tracker needs to find the position in the reference that matches the current audio frame, or offline, where the full target audio is available. Given the design of opera.guru, this project is concerned with online alignment.

There are two common choices for target format. In audio-to-audio alignment, both target and reference are in audio signal form. In audio-to-score alignment, the reference is

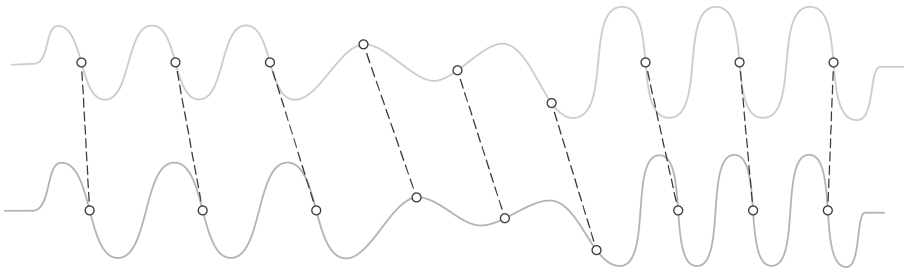


Figure 1.1: Example of an alignment between two simplified audio signals

some symbolic digital representation of the musical score. This can be a full representation of the score via e.g. *MusicXML*[4] or MIDI format, or a simplified representation such as a list of note pitches. More recently, there has also been interest in aligning audio directly to images of sheet music.

A number of factors influence the difficulty of performing the alignment. A first factor is whether the alignment in performed on-line or offline, as mentioned above. A second consideration is whether the music is monophonic or polyphonic. That is to say, how many simultaneous melody lines there are. The specific instrumentation is also relevant, sung voice especially is notoriously hard to track. Musical style similarly has great effect on the difficulty of the task.

A final factor to consider is the desired application. This influences many aspects of design and performance trade-offs such as how much latency is acceptable, given that higher latency may allow for a more robust alignment. The two most common applications used as context for research are score following and computer accompaniment. The goal of Computer accompaniment is to have a digital instrument match the tempo of a human performer and play alongside them.

1.2.1 Relation to offline music alignment

This project focuses on the problem of online music alignment, where the input is a live audio stream and the alignment has to be performed in real-time. This is in contrast to the significantly easier task of offline alignment, where the algorithms can use utilise the full length of both input and reference series. This use of 'future' information makes these algorithms more robust with less complex methods. Much of the early research in music alignment was done on this offline case and many of the techniques used in online alignment were originally applied to the offline case.

1.3 Research Questions

This project aims to achieve effective automation of the queueing process in opera.guru using music alignment to re-use queueing information associated with a previous recording of the same opera. To this end, the following research questions are raised:

Research Question 1: How can music alignment algorithms be used to automate queueing in opera.guru?

[4]See https://www.musicxml.com/

The first step to answering this question is to explore the design goals and to develop a set of requirements and constraints for the system. A suitable alignment algorithm is then implemented that will align a reference recording to live audio. This algorithm forms the core of a standalone program that is integrated into opera.guru to provide automated queuing. When provided with a suitable reference recording and the relevant queueing sequence, this program will re-play the queueing sequence adjusted for the timing difference between live audio and the reference recording.

Research Question 2: How can existing music alignment algorithms be improved to increase accuracy for opera tracking in the case of opera.guru?

The accuracy of the alignment algorithm is key to the usability of the automated system. Therefore, novel extensions to the algorithm are explored. Three ideas are developed and implemented; alternative heuristics for alignment advancement, similarity matrix pre-processing methods and the inclusion of future reference frames.

Research Question 3: How does an automated system for queueing in opera.guru affect the user experience?

To answer the final research question, a user experience trial is held. In this trial, automated queueing is tested in a practical setting. The user experience of using opera.guru controlled by automated queueing is then compared to control by the current process of manual queueing.

1.4 Structure of this thesis

This thesis is structured as follows: Chapter 2 provides an overview of the relevant literature. The existing literature on music alignment is covered by dividing the literature in two main groups based on the techniques used. These main techniques are derived from either Dynamic Time Warping or Hidden Markov Models. Literature on Audio Features is discussed separately due to the major role they play in all music alignment approaches. Chapter 3 concerns the implementation of the chosen algorithm as an application usable with opera.guru. Chapter 4 describes the main contributions of this thesis in improving the accuracy of the music alignment algorithm. This is done through presenting three novel improvements or extensions to the algorithm. Two of these extensions provide significant improvements to the accuracy of the algorithm. Chapter 5 discusses the evaluation of the implemented system. First, the algorithm, individual extensions and the full system are tested in simulated tests. Next, a user experience trial is presented, investigating the effects of using the new automated system on users' perception of and satisfaction with opera.guru. Chapter 6 presents a discussion of the project, analysing the successes as well as the shortcomings. Finally, chapter 7 concludes the thesis by providing a summary of the work and the main findings, as well as exploring possible future work.

2 Literature Review

This chapter discusses the literature regarding online music alignment. A music alignment algorithm generally requires two parts. One part is the audio feature. The continuous audio stream is first divided into short frames. An audio feature is then computed from the raw signal, which represents the musical content of the frame in some meaningfully comparable manner. The second part is to find the alignment which matches up the sequence of audio features with maximal similarity between matched features. This chapter starts with a short overview of the field and the various alignment techniques used over the years. Next, two major alignment techniques are covered in detail: Hidden Markov Models and Dynamic Time Warping. The audio features used with these techniques are discussed separately. Finally, a summary of the important and relevant findings from the literature is presented.

2.1 A brief history of techniques used

At the 1984 International Computer Music Conference (ICMC), two papers were presented that independently started the new field of online music alignment. Dannenberg [15] and Vercoe [63] both proposed algorithms for real-time score following in the context of computer accompaniment. The goal of computer accompaniment algorithms is to adjust the output of a digital instrument to match the live performance by a human musician. These early approaches tracked specific monophonic instruments in symbolic form based on string matching techniques. This was adapted to single-instrument polyphonic settings with relative ease. Puckette [50] describes an algorithm based on an ordered note list where the played notes are detected using pitch tracking. Grubb & Dannenberg [16] first tackle the problem of tracking ensembles. This system is based on the earlier work by Dannenberg and adapted to combine the output of multiple trackers. The played audio is symbolised using a microphone for each musician and a pitch-to-MIDI converter or directly exported from a digital keyboard. As ensembles grow larger, it becomes unfeasible to individually track each musician and the algorithm breaks down.

In 1998, Grubb published his PhD Thesis and accompanying short paper [25][26] introducing a probabilistic framework for audio tracking. Later stochastic models, starting with Raphael [52]) and Cano et al. [11] would use Hidden Markov Models. Variations on these probabilistic methods also exist; Duan & Pardo [19] used a specialised state space. Sako et al. [56] used Segmental Conditional Random Fields. Korzeniowski et al. [37] used Dynamic Bayesian Networks and particle filtering. Finally, Xia et al. [64] attempted to create a more human-like accompaniment system based on Linear Dynamic Systems. Although some of these system are very interesting and quite promising, they have not been very influential so far and will therefore not be discussed further here.

The use of Hidden Markov Models (HMM) proved to be critical, providing a powerful framework for further research. Many advancements have been made over the years. State-of-the-art models no longer use pure HMM but hybrid models using semi-Markov

states instead. These hybrid models allow for the explicit modelling of secondary factors such as tempo. HMM-based systems also form the heart of the most successful commercial systems; Metronaut (Antescofo)[1] and Tonara[2].

In 2005, Dixon introduced On-line Time Warping, a real-time variant of Dynamic Time Warping [17]. Dynamic Time Warping (DTW) has long been used for offline alignment of time series, including music. As with HMM, many improvements to DTW-derived approaches have been made over the years in order to increase robustness.

Recently, there has also been interest in the use of Deep Learning and Neural Networks for music alignment. Dorfer et al. [18] use a Neural network to map audio directly to a position in a score image. Henkel et al. [28] extend this system to achieve music alignment to full-page score images. These techniques are quite exciting but still in relatively early stages of research.

The rest of this chapter will focus on the two techniques of Hidden Markov Models and Dynamic Time Warping and more specifically the case of audio-to-score alignment for the former and audio-to-audio alignment for the latter. One final thing to note is that HMM and DTW are closely related techniques. At its core, the difference between DTW and HMM approaches is not about the functionality of the algorithms produced, but of the representation of the system. It has been shown by Fang [21] that DTW is a special case of an HMM.

2.2 Hidden Markov Models

2.2.1 What is a Hidden Markov Model

An HMM, see Rabiner & Juang [51], describes a system in a probabilistic fashion using a state-model. A Markov Process describes a process based on the probability of moving between states at each step. At any point, the probability to move to any state is determined only by the current state. In a *Hidden* Markov Model, as the name suggests, these states are hidden and can only be observed by the variables they emit. A simplified view of an HMM for audio-to-score alignment is that the hidden states constitute the notes written down in the score while the audio stream is represented by the emitted variables. Probabilistic modelling can then be used to estimate the notes that are most likely to have produced this sound.

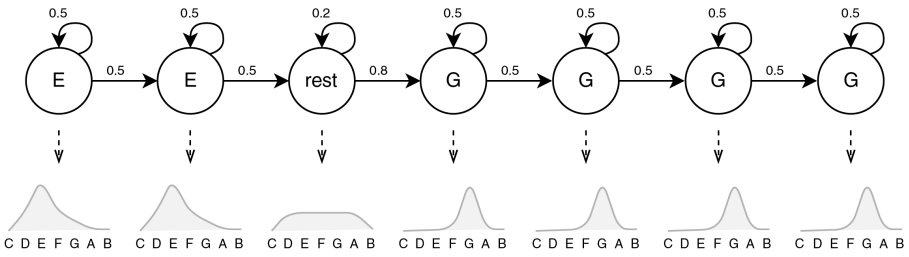


Figure 2.1: Simple HMM model of two notes with transitions and observation probability distributions shown

[1] https://www.metronautapp.com/
[2] https://www.tonara.com/

2.2.2 Details of the application to music alignment

Although Grubb & Dannenberg [25] did not use Hidden Markov Models, their use of probabilistic methods is still relevant. In this system, the score is represented by a sequence of events, each with a defined length and observation probability distribution. Note the similarity with the hidden states and their emission probabilities in the HMM model. As output, Grubb's model specifies at any time a conditional probability density of the output position based on three factors: the previous position, the most recent observation and the estimated traversed score distance. This can be calculated using the aforementioned probabilities and a number of simplifying assumptions.

The HMM music alignment system from Raphael [52] works as follows: The signal is first divided into frames. For each frame, a low-dimensional feature vector is calculated based on three statistics. This vector corresponds to the observable variable $y \in Y$. For each frame there is also a state variable x in the hidden layer X. Each state is associated with a note (or rest) in the score, though many states are usually associated with each note[3]. The following labels are used for the states in the hidden layer: $'rest'$, $'pitch(i)'$ and $'artic(i)'$ for every pitch i. The $'pitch(i)'$ label refers to the sustained part of the note with pitch i, whereas $'artic(i)'$ refers to the attack, i.e. the start of the note.

The output distribution, i.e. the probability distribution of states y emitted by x, is assumed to depend only on the label of x. The feature vectors of Y are discretised in order to achieve a limited observation space. The output distribution is then trained using a pruned variation of the Baum-Welch algorithm. As alignment, the system calculates a segmentation, defined as a vector listing the starting frame for each note. The best segmentation is found by minimising the segmentation errors, assuming an error occurs when the starting frame is estimated wrongly by more than one frame.

2.2.3 Advanced Hierarchical models and semi-Markov Models

Orio & Déchelle [47] introduced a two-level HMM with a high-level model for following the general performance and an embedded low-level model for every note, modelling the stages of the note. The high-level model has states representing every *event* in the score, these can be notes, rests, or chords among others. Every event is represented by two states: a *normal state* and a *ghost state*. The ghost state is used to model mismatches between the score events and the performance, for instance when a note is skipped. The low-level model represents the different signal sub-sections of a single note; i.e. the attack, sustain, and rest states. This same model is applied to tracking the polyphonic MIDI output of a keyboard in Schwarz et al. [58].

Cont [12] also used a two-level model. The first hierarchy models score events, similar to the previous system. This model switches to the second hierarchy when such an event is started and the second hierarchy then models the duration of the event. The major introduction of this paper is the use of Non-negative Matrix Factorisation (NMF) for multiple pitch detection. Used by Sha & Saul [59] for the determination of the different pitches of simultaneous voices, NMF is an unsupervised learning algorithm for finding the linear basis components of some complex output. The use of NMF allows this system to track raw polyphonic audio.

[3] Between 15 and 16 states for a quarter note at 120 bpm in 4/4 signature based on the frame size of 32ms. The same state can be visited multiple times through self-transitions.

Joder et al. [33] use a highly hierarchical model, introducing *bars* and *beats* as state model hierarchies above the low level *chords*. This hierarchy allows the algorithm to apply pruning of lower-level states, which greatly improves efficiency.

As noted by Raphael [53], conventional HMM models are unsuited to modelling the length of notes. Thus, new improvements focused on introducing tempo models that inform note length. Modelling tempo in Markov states proved ineffective due to the need for discretisation. Raphael therefore introduced a Hybrid graphical model with discrete HMM variables for position and a continuous variable for tempo. Cont [13] applied this idea as Hidden Hybrid Markov/semi-Markov chain following the model of Guédon [27]. This system is *anticipatory* in that it uses the tempo model to predictively adapt the tracking. The resulting model is flexible and also allows for the modelling of ornaments such as trills or glissandi.

2.3 Dynamic Time Warping

2.3.1 Offline DTW

Dynamic Time Warping has long been used in various fields for aligning time series. A comprehensive overview of its application in music analysis is given in Müller [44]. In the discrete case, comparing sequences reference $X = (x_1, x_2, ..., x_N)$ and target $Y = (y_1, y_2, ..., y_M)$, every x_i is matched to some y_j such that these correspond to the same, or an analogous, position. In the case of a piece of music, this can be understood as matching to the same position in the score. If the target is slower than the reference around a certain point x_i, then this x_i might have multiple points in Y matched to it. Conversely, if the target is faster then there may be multiple consecutive points in X matched to the same point y_j.

The first step for computing a DTW path is to create the cost matrix M_{xy} where every point $M(i, j)$ is the distance between x_i and y_j as defined by some selected cost measure. The cost measure should carefully be selected for the task at hand. Cost measures and the feature vectors used are discussed in section 2.4. The cost matrix can be filled in fully in this offline setting since the complete sequences can be accessed. Usually however, some bound or path constraint is used to limit the parts of the cost matrix that are to be calculated. This is done to reduce the runtime. These bounds are based around the diagonal since the sequences, while not perfectly aligned, which would equal this diagonal, are still supposed to be similar in time. Two popular DTW bounds are the Itakura-parallellogram [31] and the Sakoe-Chiba band [57]). When using such a bound, the optimality of the DTW path is no longer guaranteed. When the cost matrix is complete, dynamic programming is used to calculate a minimum-cost-path between M(0,0) and M(N, M). This calculation is based on the recursion of Eq. 2.1. The minimum cost path itself is obtained by traversing the accumulated cost matrix backwards.

$$D(i, j) = d(i, j) + \min \begin{cases} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{cases} \tag{2.1}$$

The backwards path construction means that in its typical form DTW cannot be used for real-time time warping. A second problem with the use of DTW is its computational

complexity; $O(NM)$ in both time and space. This is problematic for long sequences such as classical symphonies or operas. Recent improvements have been made here in a number of ways: Multiscale methods can be used to calculate the path at course detail level first so that the higher detail level has much less to compute. Prätzlich et al. [49] show an effective implementation of this idea, achieving constant memory requirement in practice. Tralie [61] introduce an algorithm with $O(M + N)$ memory bound and improved parallelisation for efficient runtime in practice. Gold & Sharir [23] Introduce a new method reducing the time complexity to $O(n^2 \log\log\log n / \log\log n)$.

2.3.2 On-line Time Warping

An on-line adaptation of DTW was introduced by Dixon [17]; On-line Time Warping (OLTW). Since then, others have independently proposed on-line dynamic time warping algorithms in other contexts, see Ko et al. [36], Li [38], Oregi et al. [46]. Macrae & Dixon [41] also introduce an alternative on-line algorithm with Windowed Time Warping. This has seen relatively little adaptation however. The original 2005 OLTW algorithm is the most used in state-of-the-art systems with various improvements made to it over the years.

The OLTW algorithm computes a greedy forward path in linear time. Where in offline DTW the accumulated cost matrix is constructed all at once beforehand, in OLTW this matrix is calculated incrementally up to a defined search width away from the current position. The cost calculation itself is the same as in traditional DTW. At each step of the algorithm, either a row or column is calculated. This corresponds to advancing either the target or the reference sequence. When advancing a sequence, the cells of the row or column are calculated up to search width back from the current position. In the initial phase, until the search width is reached, columns and rows are advanced alternately. After this, the cell within the current row and column with the minimal length-normalised path cost is found. If this is cell is found in the row then the row is advanced and if its found in the column then the column is advanced. If the current cell has minimum cost then both row and column are advanced. A final limitation is applied with the $MaxRunCount$ parameter which governs how many advances in the same direction the algorithm can make consecutively, before a step in the other direction needs to be made. This effectively limits how much faster/slower the target is allowed to be compared to the reference.

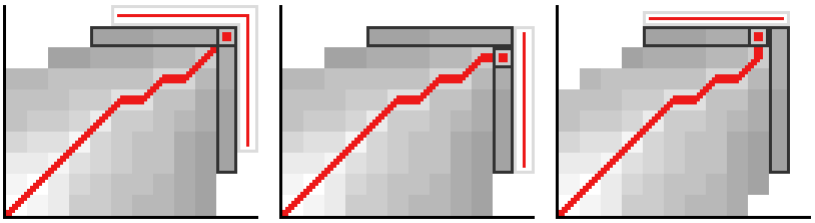


Figure 2.2: The three advancement options in OLTW with the path, current minimum cost cell and next expansion highlighted

2.3.3 General robustness improvements

Unlike HMM-based approaches, where the model itself can be redesigned in many ways, the DTW model itself is simple and robust, so approaches have focused mostly on adding new systems onto this base to intervene when the tracking is likely to go wrong.

Arzt et al. [4] introduced three strategies for improving robustness for score following: First is the backward-forward stategy: every n frames, a short smoothed backward alignment path is followed, thereby finding a more reliable base point for the path. From there a forward path is again computed up to the current target frame. This strategy utilises the robustness of offline backtracing DTW to improve accuracy.
The second strategy makes use of the symbolised score. It applies a heuristic to determine if the current audio frame matches the next expected note based on the score.
The third strategy is designed to handle structural changes in the performance. Multiple trackers are used at predefined sections. If one of the alternative position trackers match better than the standard alignment, that tracker becomes leading. This strategy requires manual defining of the alternative tracking positions, but this presents relatively little effort.

Arzt & Widmer [2] presented a two-level tracker with multiple hypotheses capable of correcting for structural mismatches without using manual input. The system first uses a 'rough' tracker that maintains a list of the most likely current positions at a hop size of 300ms and frame size of 600ms. The system also maintains a fixed number of low-level trackers with hop size of 20ms (frame size 46ms). A decision maker is used to decide which low-level tracker should be used for the current system output and whether to reject trackers that are clearly wrong. The system also uses a 'structure model' that it calculates from the score based on a self-similarity matrix. This model is needed to distinguish repeating motifs. It also allows a reduction of the number of concurrent hypotheses.

Arzt & Widmer [1] introduced tempo models to the OLTW-domain. This system uses tempo estimation for two purposes. The first is to predictively warp the reference recording in order to reduce the difference between target and reference. This is done using a tempo estimator based on the recent slope of the alignment path, modelled after the work of Müller et al. [45] on tempo estimation using offline DTW. In the second application, multiple reference recordings are first aligned offline and their tempo curves are extracted using the same method as before. This information is then used to inform the tracker of estimated future tempo changes. The results of these additions vary between different pieces, but show significant improvements in pieces with large and sudden tempo changes.

Arzt & Widmer [3] utilised multiple reference recordings of the same piece for greater robustness. These recordings themselves were aligned to score beforehand. During the performance, multiple trackers each try to align the target audio stream to a different reference recording. The output score position is then given by the median position from the different references.

Finally, Macrae et al. introduced some improvements driven by their practical application in the *MuViSync* system. In Macrae et al. [40]. they present *MuViSync* as a music video synchronisation system that works in real-time. This system starts with a buffering step and then determines the starting path of the alignment path, a problem left unsolved by previous systems, which required the system to be started at the same time as the music. After this, the system uses a forward-backward DTW strategy to perform alignment. A post-processing step is applied afterwards to smooth over rapid changes, thereby ensuring

smooth video playback.

2.3.4 Detailed look at a system for opera tracking

In 2020, Brazier and Widmer published two papers on their attempts to create a system capable of reliable real-time tracking of operas. Opera is a particularly difficult medium to track for a few reasons; The first is the great length of an opera performance which is usually not continuous in its music. Scene changes are often accompanied with applause or breaks resulting in relative silence. The second problem lies in the *recitativo* sections where the singers present dialogue in a style more similar to regular speech than singing. This leads to larger differences in both rhythm and voice, making them more difficult to track. Some sections, especially *secco recitativo*, feature little accompaniment, often only a harpsichord, and are more improvisational in nature. This can result in structural differences between performances as sections are omitted. Structural differences are not addressed in the studies discussed here.

Brazier & Widmer [7] focuses on robustness improvements in the face of three common problems through the use of specialised event detectors. The paper introduces three event detectors; An applause detector that activates when it hears applause lasting for longer than 400ms in certain predefined positions (based on score information) where applause that halts the performance is likely. When this is detected, the tracker is halted until the applause stops and then jumps to the next section. The second detector listens for pauses in the form of silence. This detector works in the same way as the applause detector. It is activated when neither music nor speech is detected. The third detector is designed to avoid getting lost during improvised interludes occurring before *recitativo* sections. This detector is activated when the reference sequence in the tracker reaches such a section as indicated by the voice activity and halts the live sequence until voice activity is detected here too. The detectors use Long Short Term Memory-Recurrent Neural Network (LSTM-RNN) classifiers based on specialised features from the relevant literature. LSTM-RNN are a type of neural network capable of processing data sequences with long-range dependencies, see [55] The detectors are trained using a custom set of annotated audio. The addition of the detectors proved very effective when tested on Mozart's *Don Giovanni*, solving most of the occurrences of the tracker 'getting lost'. It is uncertain however, how much of this improvement translates to other operas. Certainly the third detector is designed explicitly for the structure of the opera used for testing.

Brazier & Widmer [8] focuses on the difficulty of tracking *recitativo* sections. The premise of their method is to use a secondary tracker next to the regular music-tracker that is optimised for speech. This speech-tracker uses the same DTW-based algorithm as the music-tracker but uses audio features specialised for speech. Different features based on Mel Frequency Cepstral Coefficients (MFCC) were tested for use. An explanation of MFCC features is given in section 2.4. The most successful speech features used 25 MFCC features of a smoothed LPC spectral envelope from a down-sampled signal. LPC (Linear Predictive Coefficients) is a feature commonly used in speech-processing, see Atal & Hanauer [6]. The down-sampling is helpful in this case in order to limit the frequency range and avoid including the singer's *formant*, a peak in the frequency spectrum roughly around 3kHz, which differs among singers and is not well-informing of the sung pitch. In order to get the best tracking result from these two systems, a few combination strategies were tested. These were based on a weighted average with weights based on the output of

the speech and music detectors. The weights based on the ratio between the output of both detectors proved most effective. The final results of the additions of a speech-tracker showed limited improvement to the alignment accuracy. The system was tested on two different target recordings of *Don Giovanni*. The $\leq 1s$ error accuracy rate was increased by 8% and 10% on the *recicativo* sections specifically. The accuracy over the full opera was improved significantly on the first recording, with mean error halved from $811ms$ to $373ms$ and the $\leq 5s$ error rate increased from 97.3% to 99.0%. The results on the other recording were minimal however with mean error reduced by only $14ms$ and $\leq 5s$ error raised only 0.1% from 97.9% to 98.0%. Proving once more that the specifics of the recordings used are critical for the alignment accuracy.

2.4 Audio Features

All methods of audio tracking rely on the use of a cost measure. This cost measure allows the algorithm to determine how similar two frames of audio are. The goal of the tracking algorithm is to match up those audio frames (or symbolic notes) in the live and reference streams that correspond to the same bit of music. The cost measure is precisely there to determine how likely it is that two frames correspond to the same bit of audio. However, measuring cost on the raw audio signal is not very appropriate. The difference in audio signal does not inform those aspects of the sound that makes us humans recognise it as the same bit of music. The design of *Audio Feature* is tasked with transforming the signals in such a manner that the differences in those aspects that are musically relevant are exaggerated and the rest, whether it is noise or a legitimate difference between performances of the same piece, reduced as much as possible. The feature design might also be adapted to suit the specific use case.

The use of audio feature is most direct in audio-to-audio alignment, since the same feature can be used for both audio streams. In the case of audio-to-score alignment using symbolic methods such as HMM-based methods, the features are not compared directly. Instead, the symbolic score events are tied to the hidden states and the frames of the audio signal tied to the observables. The choice of available labels in for the hidden states and the design of the audio feature representing the observables are crucial for the design of the system. This feeds into the calculation of the probability distribution of observables emitted by the hidden states.

2.4.1 In HMM-based systems

In one of the first and most influential HMM-based alignment systems, Raphael [52] used three statistics per frame; (1) the average signal amplitude, allowing for the detection of rests. (2) an 'activity' statistic, detecting the start of notes. And (3) a frequency-based statistic informing the note pitch. As presented in section 2.2.2, the hidden states in this system are modelled with rest or attack or pitch labels. The audio feature has been carefully designed in tandem with the state model such that the statistics maximally inform the distinction of these state labels. Cano et al. [11] used a combination of as many as six (scalar) statistics. The two-level HMM of Orio & Déchelle [47] uses

Peak Structure Distance The core idea of Peak Structure Distance [48] is to explicitly model the note harmonics. For any note, a score spectrum filter is generated which filters out any activity in the performance spectrum that does not match the harmonic series of

said note. The ratio between energy within the filtered spectrum and without is then used to find a matching note pitch. A related method, *semitone energy*, filters the spectrum into semitones [32]. A template vector is then built for each chord for likelihood estimation. This template vector also includes a noise component.

Non-negative matrix factorisation Cont [12] employs a system based on multiple pitch detection through NMF. For each pitch, a spectral template column in matrix W is learned. Given observation V, the template matrix W and equation $V \approx WH$, a decomposition matrix H is learned during the performance. This shows which pitches are most likely to have produced V.

Generated spectrum A number of more recent systems focus on generating a spectrum from the symbolic note information. Raphael [53] recreates a spectrum from the note pitch with a peak for each harmonic and decreasing energy for higher harmonics. This is then compared to the frequency spectrum of the audio frame. A similar method is used by Cont [14]

Chroma features Chroma features are a compressed frequency feature that has seen use in many music-related tasks. In a Chroma feature the same notes from each octave, the pitch classes, are combined into one. This results in a feature measuring only tone height. Which is much more informative of the music than the octave of a note, see Müller [44]. As with semitone energy, these can be compared against generated templates for any chord. Joder et al. [33] use a combination of a Chroma feature and spectral flux. Spectral flux measures the energy increase in the frequency spectrum compared to the previous frame. A threshold filter is then used to obtain an onset feature from this.

Joder et al. [32] test a number of audio features for symbolic alignment. These include the methods listed above and a number of variations on the same ideas. They find the most success with semitone energy and Chroma features, especially in polyphonic settings.

2.4.2 In DTW-based systems

The original OLTW system (Dixon, 2005 [17]) used a relatively simple 84-bin frequency vector. This vector has linear spacing at lower frequency, logarithmic semitone-spacing at higher frequencies and finally all frequencies above a G9 combined into one. The cost function for this is the Euclidean distance of the energy increase in each frequency bin as compared to the last frame, similar to *spectral flux*. This results in only the start of notes being compared. This rather simple audio feature proved effective but left room for improvements. When applied to real-world music, the harmonics strongly pollute the frequency spectrum. A number of more advanced features have been used to combat this problem:

Chroma features As mentioned in the previous section, Chroma features are popular for many applications, including DTW-based audio alignment. A common extension used here is logarithmic compression, where the dynamic range in the chroma scale is compressed, thereby making secondary values more pronounced. This compression mimics the logarithmic nature of the Decibel scale. Evert et al. [20] compute a chroma feature from note onsets only. They also apply a local normalisation step which magnifies low energy onsets in low energy temporal neighbourhoods, to obtain the Locally Adaptive Normalised Chroma Onset feature (LNCO). In offline cases, temporal decay is added to this feature as a final step

Arzt et al. [5] propose a mixed audio feature, computed from a weighted sum of two

other features. The first is a locally normalised Semitone Onset feature, similar to the LNCO, but without the Chroma reduction step. the second sub-feature is a normalised Chroma feature. This mixed feature is an attempt to emphasise note onsets without fully removing information from the sustain phase of notes.

Constant Q transform The Constant Q transform (CQT) [9] is a replacement for the Fourier transform. Unlike the Fourier Transform which features a linear frequency scale, the CQT features a logarithmic frequency scale. This matches the logarithmic nature of the harmonic series, thereby giving them a constant pattern in the calculated frequency domain. This also fixes the issue of unequal bin resolution that characterises Fourier-Transform-based features. An efficient implementation is given by Brown & Puckette [10].

Mel Frequency Cepstral Coefficients MFCC's [39] are a multi-step audio feature imported from the speech processing field. As in other methods, the Fourier Transform is first used to obtain the frequency spectrum. From this, the logarithm is taken and the spectrum is divided into bins of varying width according to the Mel scale. The Mel scale is designed to emphasise and de-emphasise frequencies in such a way as to emulate human hearing. Finally a Discrete Cosine Transform is applied to produce the MFCC. When using MFCC's, it is common to use only the first n coefficients, possibly also skipping the very first ones.

A number of studies over the years have evaluated the effectiveness of different audio features. Kirchhof & Lerch [34] presented a large number of features, including even a few that have seen little use in practice. They found that Chroma features were especially effective. Grachten et al. [24] found that CQT and MFCC with $n = 50$ were the most promising. A recent study by Gadermaier & Widmer [22], tested on excerpts from classical symphonic works, found that MFCC's with a high coefficient count of 120[4] and skip-count between 20 and 40 were most effective. This last study also compared three different cost functions; Euclidean, city block, and cosine distance. The results of this were rather inconclusive however, favouring different functions depending on the piece tested.

2.5 Summary of the literature

For the task of music alignment, numerous techniques and approaches have been researched. Two broad categories of approaches have been established over the years. In all cases, audio features play an important role in alignment algorithms. An audio feature is the representation used for a frame of music. These are chosen to contain the most relevant information for determining musical similarity while including minimal extraneous information. HMM-derived systems have shown to be a powerful method for audio-to-score alignment. The explicit state models of the score provide a solid framework for tracking music. In order to improve the systems in the face of tracking errors, tempo models are used, expanding the model from pure HMM to hybrid models with semi-Markov states. Although early HMM-based techniques usually employed a number of combined low-dimensional statistics as audio feature, later developments turned to frequency-focused techniques and generating templates or even full spectra for likelihood estimation of matching a certain note or chord. Techniques based on Dynamic Time Warping provide a robust

[4] as opposed to the low numbers common in early uses in the field, 5 in the previously mentioned study by Kirchhoff & Lerch [34], or 13 in Hu et al. [29].

method for audio-to-audio alignment. They respond well to polyphonic music, though local differences can substantially derail tracking. In order to achieve high robustness, a number of techniques are used to correct common tracking problems. Mel Frequency Cepstral Coefficients's have proven to be the best performing audio-feature representation for DTW-based systems.

3 Algorithm and Implementation

To achieve the goal of automating queueing in opera.guru, a standalone program was created implementing music alignment. The program takes in audio and queueing data from a previous performance, compares this to live microphone input and connects to the opera.guru back-end service to control the queueing. The program was designed to satisfy the design goals laid out below. The algorithm itself follows the Dynamic Time Warping method and is covered next. The final part of this chapter discusses the overall design of the application and its integration with opera.guru.

3.1 General design

3.1.1 Design goals

Every live performance of a piece of music is different, with tempo changing according to the artistic interpretation of the musicians, in-the-moment. An automated queueing system for opera.guru should not force musicians to adjust, such as by mandating a certain tempo, but instead adjust itself to follow the artists.

A naive implementation of automated queueing would be to record timestamps for each chunk during the first performance and play this back as is during subsequent events. Due to the often quite large deviations in tempo during a multi-hour opera performance, this is not a feasible solution unless the performers are required to maintain a strict tempo-regime, which is undesired.

In order to perform automated queueing, the system will need some reference for the correct timing of releasing chunks. Voice recognition could provide a potential approach, though voice recognition for operatic singing is a challenging task. Moreover, a solution based on voice recognition would limit the potential content for chunks. Another possible approach would be to link timing to a musical score. Music alignment to musical score has been studied extensively, as discussed in chapter 2.

The approach chosen in this project is to link chunks to a reference audio recording. This requires less preparation for the opera house prior to the performance. This reference recording may be from a rehearsal performance or the premiere performance with live queueing also recorded. Alternatively, the chunks may be hand-aligned to an existing reference audio recording prior to live operation. Because there may be significant difference in audio profile between different performances of what is nominally the same piece of music, the scope of this project is limited to reference recordings from the same production. More specifically, any structural differences, as may also be caused by improvisations such as in *secco recitativo*, are considered out of scope.

3.1.2 Requirements

In order to be an acceptable alternative to manual queueing, the system should fulfil a number of requirements. These requirements are derived from the design of opera.guru and the envisioned main use case of the automated system.

1. The alignment accuracy must fall within a few seconds for nearly all chunks.
2. Asymptotic runtime complexity must be linear in respect to the length of the performance.
3. Asymptotic memory complexity must be linear in respect to the length of the performance.
4. The system must accept live microphone input.
5. The system must accept an audio recording file as reference.
6. The system must integrate with opera.guru seamlessly, showing no differences for the user compared to manual control.

3.1.3 Constraints

A few constraints will be assumed about the use of the system. These help make the task realisable. Like the requirements, the constraints are derived from the design of opera.guru. Though these constraints do limit the general applicability of the system, they should have no impact on the main use case of opera.guru.

1. The reference recording is assumed to be from the same production and cast or equally similar to the target performance.
2. opera.guru message content will be scene descriptions, not one-on-one transcription subtitling in line with opera.guru design principles and experience from opera houses.
3. Alignment accuracy is relevant only at the moment a chunk is to be released.

3.2 Algorithm implementation

The system implemented is based off the OLTW algorithm first introduced by Dixon [17]. The algorithm is centred around a cost matrix where indices on the horizontal and vertical axes represent frames in the target and reference stream respectively and every cell contains the difference between a pair of frames from these. This difference is calculated using the selected audio feature and represents the cost of alignment. The alignment path is then the diagonal path through the matrix with the lowest summed cost. The cost matrix is calculated progressively around the head of the forward-constructed alignment path.

Whenever a new target audio frame is accumulated from the live recording, the audio feature for said frame is first calculated which is then appended to the target buffer. The `step` method is then called, which uses the both the target buffer and the pre-calculated

reference buffer, as well as the current index (t, u) of the head of the alignment path, and extends the alignment path forward one step. The `step` method is called repeatedly until t corresponds to the end of the target buffer. This effectively means that `step` is called until a step on the target axis is taken.

3.2.1 Step

The `step` method extends the alignment path one step on either the *row*, *column* or *both*. `step` in turn calls the `getInc` method to determine this direction and then the `evalPathCost` method on the relevant row or column to further fill in the cost matrix. The *runCount* variable is also incremented if necessary to track the number of repeated steps in the same direction. Algorithm 1 shows the core logic of the step method.

Algorithm 1: Step

```
Data: t,u
if getInc(t,u) ≠ 'column' then
    t ← t + 1
    alignment[t] ← u
    evalPathCost(t, range(u-searchwidth))
end
if getInc(t,u) ≠ 'row' then
    u ← u + 1
    evalPathCost(range(t-searchwidth), u)
end
if getInc(t,u) then
    runcount ← runcount + 1
else
    runcount ← 0
end
```

3.2.2 evalPathCost

`evalPathCost` fills in one cell of the cost matrix by calculating the alignment cost of a pair of audio feature frames (t, u). When filling the cost matrix, first the base cost of a cell is calculated as the distance (euclidean distance by default) between the two feature vectors of the target and reference buffers at the respective indices. The path cost is then calculated as the base cost of this cell added to the path cost of the lowest cost possible preceding cell. More specifically, it is defined as

$$M[t, u] = min(M[t-1, u], M[t, u-1], c \cdot M[t-1, u-1]) \tag{3.1}$$

Where M is the path cost matrix, t and u are the sequence indices and c is the diagonal cost constant which is used to increase the cost for diagonal steps. This is often set to 2 so that a diagonal step is equivalent to one step in the row and one step in the column together. It may also be set lower than 2 to encourage the algorithm to follow diagonal paths.

Finally, a length-normalised cost is calculated. This is done by dividing the path cost by the distance from the current cell to the theoretical start of the path at index $(0, 0)$,

although this index may no longer exist in the actual matrix. The default method of calculating this cost is by Manhattan distance $t + u$, thereby equalling the steps taken to reach the current index.

3.2.3 getInc

After the cost matrix is filled, the `getInc` method determines the direction to advance in. On the start of the system, `getInc` returns 'both' until the searchwidth is passed and an informed calculation can be made. Then, the last filled row and last filled column in the cost matrix are compared to find the lowest cost cell. Based on the direction of this lowest cost cell either 'row' or 'column' is returned. If *runCount* exceeds *maxRunCount* the direction other than the previous direction is returned.

3.2.4 Data Structures

The following data structures are used by the algorithm. These help fulfil the requirements for asymptotically linear runtime and memory complexity. When the system is run in `analytics` mode, different data structures are used that avoid overwriting previous data.

1. The cost matrix is implemented as a two-dimensional ring buffer, generally of size $w \times w$, where w is the searchwidth parameter. The matrix is initialised with maximum cost dummy values such that any calculated cost will be lower in comparison.

2. Audio feature buffers T, U store the audio feature of every frame in the target and reference audio sequences respectively.

3. The found alignment is saved as an array a of length matching T, such that $a_i = j$ means that T_i matches U_j. This one-sided mapping provides the needed information for the system.

3.2.5 parameters

On initialisation, the algorithm instance is passed an audio feature object which it will use to calculate costs. In addition, a number of other parameters may be passed to control the algorithm.

`save_analytics` This enables `analytics` mode, which will retain almost all data from the algorithm execution. It helps to evaluate all parts of execution in detail. This feature is not intended to be used in production and it not optimised. It comes with a large memory overhead quadratic to the length of the input.

`searchwidth` Determines the maximum distance away from the path for which calculations are performed. Multiplied by the feature's hop-length, this determines the maximum difference between the audio streams compared to the current path. The runtime goes up quadratically by the searchwidth.

`maxRunCount` The maximum allowed number of consecutive steps in the same direction. The default value is 3.

`matrixsize` The size of the cost matrix. If the length is exceeded, calculations wrap around and overwrite the beginning. This may never be lower than the searchwidth and should usually equal it. In `save_analytics` mode, a matrixsize sufficient to fit the maximum expected audio stream should usually be chosen, although a lower amount is possible to save on memory cost.

`inc_measure` Controls which method is used for determining the direction to move in. The default is 'raw', which moves in the direction of the single point with lowest cost. Different methods were tested as discussed in section 4.1.

`diagonalCost` Specifies the cost multiplier of a diagonal step in the matrix. Previous literature often uses a value of 2 to ensure that a diagonal step is equal to alternating orthogonal steps. A lower value can also be used to encourage a diagonal path in cases where the algorithm has trouble finding the correct alignment.

`n_frames` In order to initialise the live audio stream buffer, the algorithm needs to know the length. This parameter specifies how many frames can at most be expected be needed. If the recorded lasts longer and runs past the buffer length, the alignment stops.

`start_deadzone` By default, the algorithm forces diagonal steps until the searchwidth has been reached. This is done because full data is not available for determining the direction to move in. The `start_deadzone` parameter can offset the point until which this procedure is followed. If volatile alignment is expected at the start of a piece of music, this can delay the point at which the algorithm takes control.

`d_measure` The distance measure is used to normalise the local cost matrix. Different approaches to this influence how costs depend on the angle of the diagonal. The default distance measure is manhattan distance.

Finally, flags can be passed to enable the use of a number of extensions/enhancements, possibly with additional parameters to control these subfeatures. These extensions are covered in chapter 4.

3.2.6 Audio features

A number of audio features were implemented. Based on previous literature, and preliminary results, MFFC features were used as default with 120 coefficients and a skip parameter of 20. Audio features were implemented according to the literature presented in section 2.4. Extensive evaluation of various audio features and their parameters was not part of project. The following features were implemented:

Semitone A simple audio feature based on dividing the frequency spectrum into bins with semitone spacing.

Semitone Onset An adaptation of the Semitone feature recording only the energy increase compared to the last audio frame.

Chroma A further derivation of the Semitone feature compressing bins into a single a octave.

Chroma Onset Onset variation of the chroma feature.

MFCC The efficient MFCC implementation from the Librosa library [42] was used.

3.3 Application

3.3.1 technology overview

The automated queueing system was developed as a standalone application. The following technologies were used to build the system:

`python` The automated queueing system is built on python[62]. Python was chosen primarily for the ease of prototyping and experimentation. The language design, in combination with the extensive ecosystem of tooling and packages allow not only for rapid development and testing of new ideas but also rapid evaluation and visualisation.

`Jupyter Notebooks` The Jupyter Notebook environment[35] was used to evaluate the system in parts.

`Librosa` The Librosa library[42] was used for the efficient implementation of MFCC feature calculation.

`matplotlib` Matplotlib[30] Was used for creating visualisations

`ffmpeg` ffmpeg[60] was used for pre-processing audio files

Furthermore, the python packages `numpy, sounddevice, soundfile` were used for audio processing. `Pathos` multiprocessing was used for parallel processing. `pyforms` was used to create the user interface.

3.3.2 System Architecture

The main logic of the system is run as a loop as shown in diagram of figure 3.1. A new iteration of the loop starts when a new target audio frame is accumulated. The length and timing of frames is determined by three parameters; Frequency, Frame length and Hop length. The frequency is the sampling rate of the recorder. Frame length is duration of a frame in milliseconds. Hop length specifies the time between frames in milliseconds. By specifying a hop length smaller than the frame length, overlapping frames are created. This is recommended practice. The default values are 48000Hz sampling frequency with a frame length of 250ms and a hop length of 100ms. This means that the algorithm will receive 10 frames per second with frames of $480000 \times 0.250 = 12000$ samples long. This resolution is more than sufficient for the envisioned use cases.

Accumulated frames are passed over to the OLTW class which represents the algorithm instance. From here, an audio feature is calculated for the frame which is then inserted into the target buffer. Next, the algorithm processes steps in rows or columns until the end of the target buffer is reached. When this is done, a callback from the queuing manager is called where it determines if the reference index has passed a queueing reference timestamp. From here control is released and the system waits for the next audio frame to accumulate.

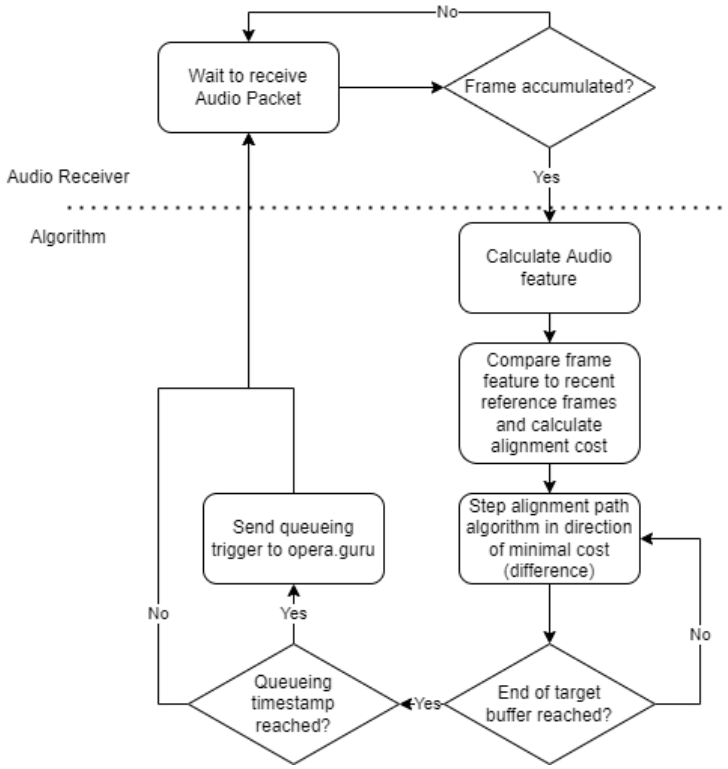


Figure 3.1: Diagram showing the core loop of the system

Classes

The system includes the following major classes:

`OLTW` represents the algorithm instance.

`AudioStream` presents an interface for the target audio stream. This can be a live recording through the SoundDevice package, or a local file, played back frame by frame. During a performance, the `AudioStream` class runs the main loop and is responsible for calling the algorithm every frame through a callback.

`Feature` represents an audio feature as described in sections 2.4 and 3.2.6.

`Queueing` The Queueing class loads the reference queueing data and maintains the active chunk progression. The queueing class is hooked into the frame loop by callback. Reference queueing files can be provided in two ways. The primary format is the CSV with an (id, timestamp) pair for every chunk for opera.guru projects. Additionally, SRT subtitle files are supported for non-opera.guru projects.

`Annotation` The annotation class is used for testing the algorithm. An annotation provides the desired alignment that the algorithm should recreate in an ideal case. Annotations can either be machine-constructed in offline settings or made by hand. Two files are loaded to provide the annotation, one representing the reference queueing file and another queueing file for the target stream. An annotation alignment is then created

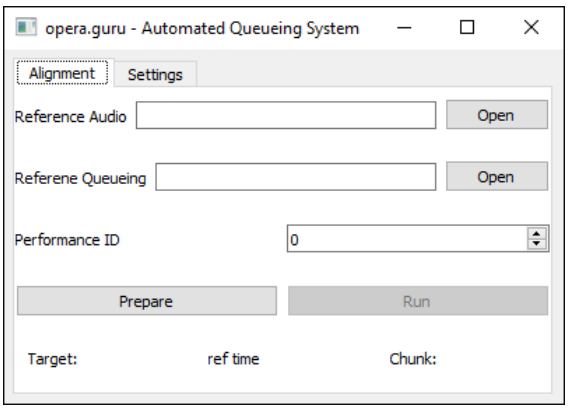


Figure 3.2: User interface for the opera.guru automated queueing system

and can be compared to which the algorithm-constructed alignment can then be compared.

`OGConnector` OGConnector provides the interface to the opera.guru back-end. API calls are made from this class. This class is called when a new chunk is reached.

Initialisation procedure

Before the performance can start, the application needs to be initialised. This first requires initialising the various modules with the desired parameters. Once all parameters have been set and links to data have been specified, a preparation step can be started. This routine creates the required data structures for the algorithm and loads the data. Audio features for the reference audio stream are computed at this point to reduce calculations during run-time.

3.3.3 Interface

A basic interface was developed for running a performance using the automated queueing system. The interface is implemented using Pyforms. The interface, see figure 3.2, consists of two tabs; alignment and settings. The alignment tab is used to import the required reference files, set the connection to the opera.guru project and run the system. Real-time feedback is given in the form of timers for both the live track and the aligned reference as well as the latest chunk id. The settings tab includes a few basic settings about the running and precision. More advanced settings, such as the selection of extensions and low-level configuration are not available in the interface.

3.3.4 Integration into opera.guru

The automated queueing system integrates into the existing back-end interface that is also used by the opera.guru management interface. The system sends `HTTP POST` requests to signal the back-end to release the next chunk. The processing of the alignment and queueing is performed offline. Future iterations could integrate timing data into the opera.guru back-end. For now, this, along with reference audio, is to be provided as local

files. The integration is implemented through a connector class object, initialised with the project information needed to access the project in the back-end.

Any time the algorithm's reference index passes a next chunk timestamp as defined in the queueing reference, a chunk forward request is sent to the back-end. Chunk-id's are provided in the queueing file for internal reference, but these do not control the queueing operation. Each forward request releases the next chunk in the back-end's performance queue.

The reference requires both an audio and a queueing reference file. The audio file should be given in signed 16bit little-endian PCM Waveform (WAV) format. This file is compared to the target audio stream and should therefore match the recording settings for the microphone setup. Testing for this project was done in single-channel audio mode with 48000 Hz frequency.

The format used for queueing files is a two-column CSV file. The first column represents the chunk id. The second column represents the reference timestamp relative to the start of the reference audio file.

4 Extensions and Improvements

Three novel techniques were developed that attempt to improve the accuracy of the alignment algorithm. The first technique introduces alternative heuristics for deciding the advancement direction the algorithm takes at every iteration. A number of heuristics are tested, based on different methods of averaging. The *weighted mean* method proves to be quite successful. The second technique introduces a pre-processing step to the similarity matrix used in the algorithm. Two techniques are tested. One fully novel technique, the *axes* method is specifically designed to break up axis-parallel lines, proves to be very effective. The second technique, the *diagonal* method, is aimed at highlighting the alignment path specifically. This technique is adapted from literature. It shows moderate success, but falls behind compared to the *axes* method. It also shows poor runtime performance. The final improvement technique attempts to improve accuracy by incorporating reference data from further on in the recording. This does not lead to accuracy improvements.

4.1 Advancement heuristics

In order to determine the direction to move in, the base algorithm always looks for the single lowest cost cell (within the last frame range). This presents a weakness in the algorithm in a number of ways and may lead to inaccurate alignment in some situations. These cases occur mostly when the target and reference streams do not match nicely. This might be a single outlier in the form of a low cost cell that does not lie on the actual minimum cost path, or it might be a larger area of lower cost. Under good operation with a clear correct alignment, a narrow 'valley' is formed in the cost-matrix. The correct alignment is found at the bottom, usually around the centre, of this valley. In low-accuracy areas it may occur that the 'valley' widens and there is no clear alignment path of lowest cost.

Figure 4.1 shows three matrices representing examples of the state of the algorithm execution at three different frames. The last row and column are highlighted, together with the lowest cost cell in these. The lowest cost cells from previous iterations are also shown. The dataset is an excerpt from Beethoven's ninth symphony, *Test Piece 1* as specified in table 5.1. The music is processed at 10 frames per second. In all three plots, the correct alignment is found roughly along the bottom-left to top-right diagonal. In frame 500, the lowest cost cell is found near the diagonal meaning that the alignment will continue as it has been. For frames 812 and 820, the lowest cost point is far away from the diagonal. These frames are located in an area in which the algorithm is not able to make out the correct alignment. This can also be seen by looking at the cost matrix directly. In the first plot, a narrow low cost 'valley' can be identified along the diagonal signifying the correct alignment. In the other plots, this valley 'opens up' into a wide area where the cost varies little. As a result, the single minimum cost cell does not always lie along the correct alignment, this is the case in frames 812 and 820.

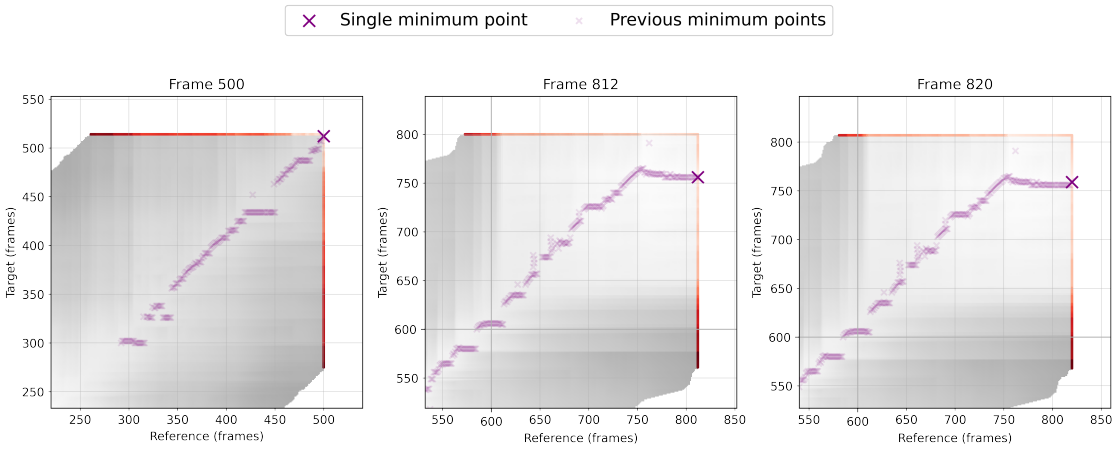


Figure 4.1: Three cost matrices showing how the single lowest cost point may lead to inaccurate alignment

In order to to reduce the negative impact of these low-accuracy areas on the alignment path, a number of alternative advancement measures were evaluated. Rather than advancing in the direction of the single lowest cost cell, these measures use some heuristic to determine the direction to advance in.

4.1.1 minimum mean

The minimum mean heuristic is designed to reduce the influence of outliers by taking the average position of the n lowest cost points rather than only the single lowest cost point. More specifically, the average position of the $n = 5$ points with lowest cost in the current frame range is taken. This position is then used to determine the direction to advance in. A side effect of this approach is that it is likely that the minimum mean point lies further back from the current position and not in either the final row or column. To compensate for this, the advancement direction is now chosen as the direction in which the minimum mean position lies closest to the last frame. The plot of frame 812 in figure 4.2 shows that the single lowest cost cell is far away from the alignment so far, indicating that the alignment should make a sharp turn away from the diagonal. However, some other cells with very low cost are roughly as far on the opposite side of the correct alignment. Taking the mean position of multiple low-cost points thus keeps the alignment roughly in the middle. The example of frame 820 shows that this effect varies from frame to frame, as here all 5 points are found at roughly the same spot, still far from the current alignment.

4.1.2 Rolling mean

The rolling mean is another heuristic based on averaging. Rather than averaging within one frame as with the minimum mean, the rolling mean averages the target position across multiple frames. Figure 4.3 shows the affect of applying a rolling mean after the minimum mean and run over the latest $n = 100$ frames. The clear disadvantage to using a rolling mean is that it uses old data. Therefore, it will be slow to react to changes in tempo. Additionally, small scale details in alignment are destroyed by smoothing over. This is

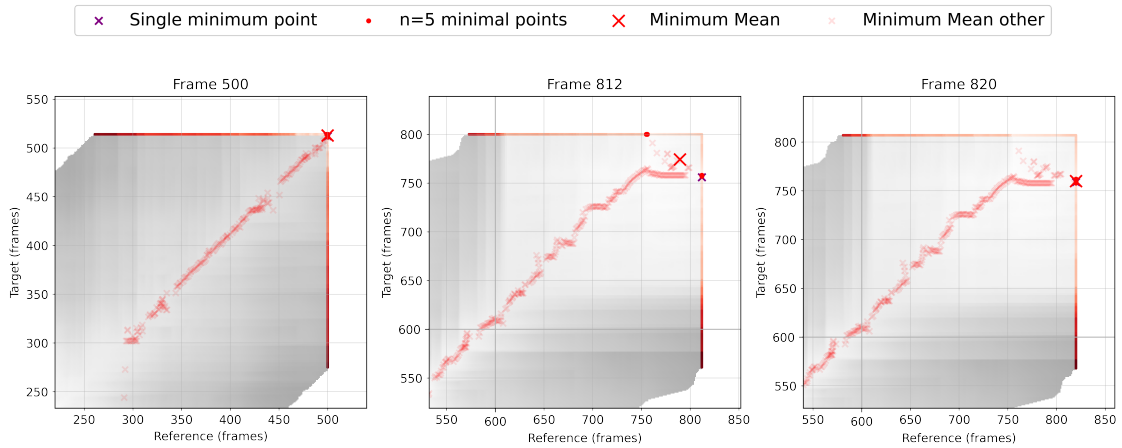


Figure 4.2: The same cost matrices as before, now with the 5 minimal points and the mean labelled and used for alignment

the desired effect for alignment errors, but of course also affects true detail. In practice the $n = 100$ used here is too high and would destroy more detail than is desired. A small value however, may be a good option to produce a smoother alignment.

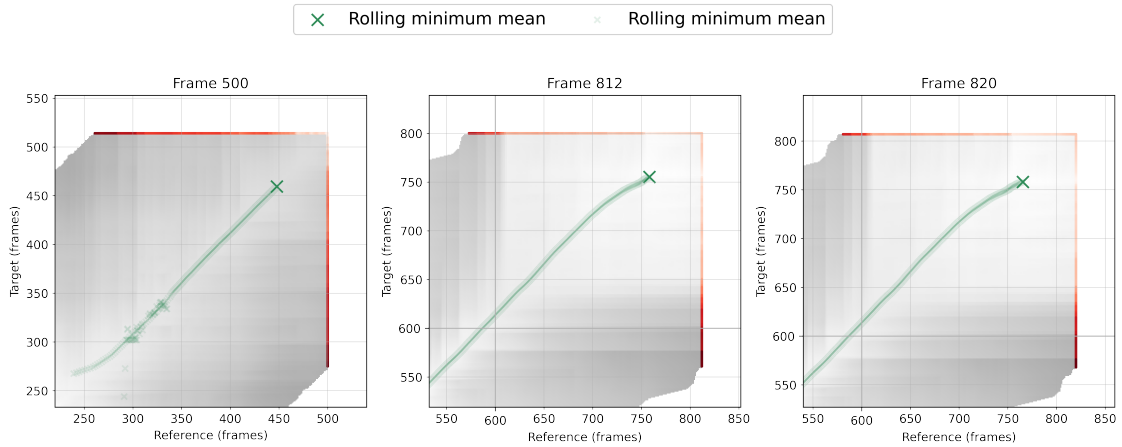


Figure 4.3: The same cost matrices as before, now with the rolling mean position labelled and used for alignment

4.1.3 weighted mean

The weighted mean heuristic attempts to use the value of all cells in the last frame. It does this by taking the cost-weighted average position of all cells in the last frame. The intuition for why the weighted mean should be an effective heuristic is that even when the alignment path is not clearly present, the cost matrix is largely symmetric around the correct alignment. Taking the weighted mean allows us to find this point of symmetry and find the alignment no matter how far the lowest cost points are removed from this alignment path.

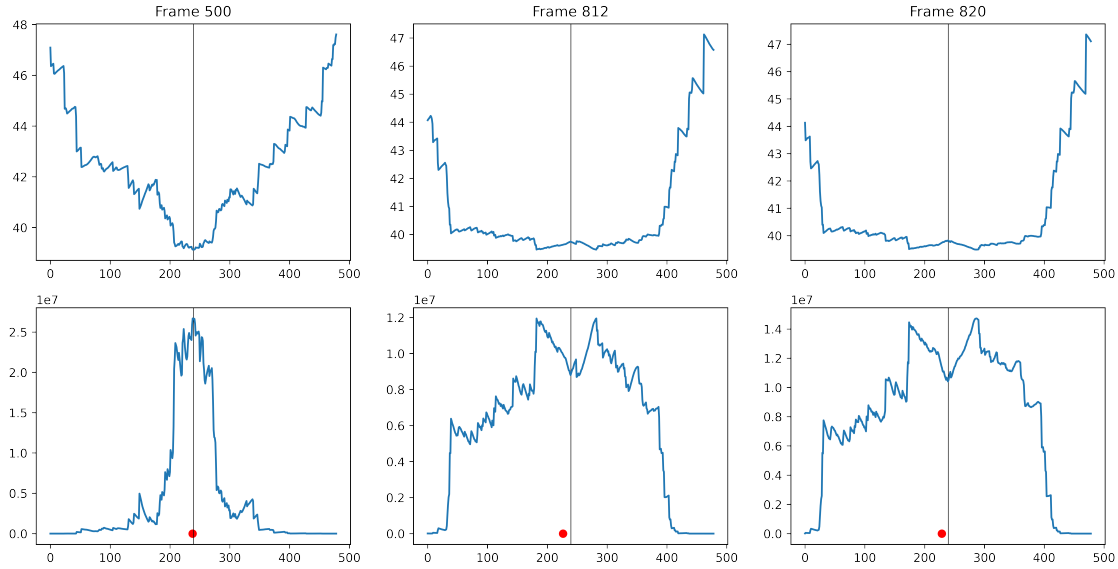


Figure 4.4: The two processing steps leading to the calculation of the weighted mean applied to the example frames

The first step is to take both the last column and last row representing the last frame out of the matrix and constructing an array of the cost values. The top row of figure 4.4 shows the result of this operation for the three example frames. The vertical centre-line denotes the current position, with the left half equalling the final row and the right half equalling the final column. The next step is to invert this array such that the indices with the lowest cost receive the highest weight. Then, these values are raised to the 8th power in order to emphasise the low-cost values much more strongly. Finally the weighted mean position of the resulting array is calculated to determine the final position. This is shown in the bottom row of figure 4.4 with the red dot indicating the weighted mean position. This position can be projected back onto the cost matrix to determine the advancement direction with the same method as before.

As can be seen in the resulting cost matrices for the three frames in figure 4.5, the weighted mean heuristic is quick to respond to changes in the alignment cost. However, because it considers the full range of the matrix it stays nearer the middle when the costs are balanced on both directions in the matrix. The downside of the weighted mean heuristic is that in cases where the alignment path is clear the alignment is less accurate. Given that sub-second accuracy is not important for the desired use case, it certainly seems worthwhile to trade this for more robustness in the face of alignment errors.

4.2 Pre-processing to remove axis-parallel lines

The main goal of the algorithm is to find the lowest cost path through the cost or similarity matrix. This path will roughly follow the diagonal of this matrix. The structure of any music similarity matrix is such that regular lines are formed parallel to the axes. These lines cross the actual alignment path and may therefore interfere in the alignment process. Figure 4.6 shows an example of a similarity matrix as used in the algorithm after applying

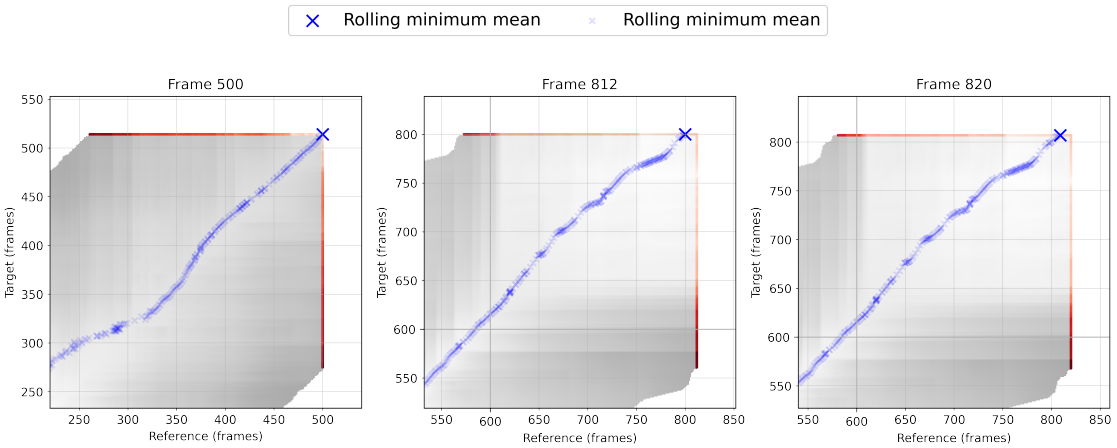


Figure 4.5: The same cost matrices as before, now with the weighted mean position labelled and used for alignment

an MFCC audio feature. Three types of low-cost lines can be distinguished in this matrix. First is the actual alignment path which goes fully across the matrix just below the diagonal from the bottom left to the top right. Secondly there are a few shorter lines running parallel to alignment path. These occur due to repeated structures in the music and occur often. The final type are the axis-parallel lines alluded to before. These lines form a major obstacle to the accuracy of the alignment algorithm. More specifically, if at any point, the alignment path is less prevalent than an axis-parallel line that crosses it, the algorithm may follow this rather than the correct alignment path.

Based on this background, a matrix pre-processing step is sought that will reduce the influence of these, or any, axis-parallel lines. The effectiveness of the method is measured by how strongly it reduces the influence of the linear patterns on the alignment path. As such, it is important that the intensity by which the alignment path stands out from the background is not reduced, or rather, is reduced less than any other pattern in the matrix. Additionally, since the pre-processing step must run in real-time for operations on the target axis. Any pre-processing step must be efficient as well as asymptotically linear in both memory and calculation usage to the length of the input sequences. Since the algorithm itself works on a moving matrix buffer, the matrix size will be linear by necessity. Care should still be taken however, that the pre-processing step is stable under moving values.

Two different pre-processing steps attempting to solve this issue were evaluated:

4.2.1 The 'axes' method

The first method was newly developed for this project. It aims to raise the calculated cost on the axis-parallel lines by subtracting previous values along the axes. First, a base constant is added to the existing cost, then the cost of preceding indices from first one axis, then the other axis, multiplied by some fraction, are subtracted from this cost. The parameters are chosen such that input and output are roughly balanced and the output will not be negative. This subtracting can work out in a number of different ways depending the existing values of both the current index as well as the preceding indices:

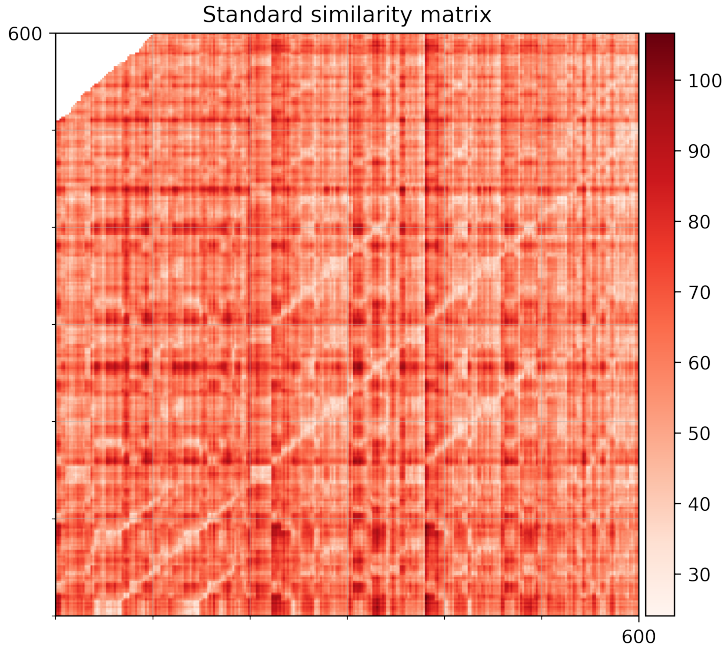


Figure 4.6: Similarity matrix from the Rigoletto test set (see chapter 5). Lower cost represents higher similarity

1. If the current value is small and the preceding values are large, then the new value will be small again.
2. if the current value is large and the preceding values are also large, then the new value will be average.
3. if the current value is small and the preceding values are also small, then the new value will be average.
4. if the current value is large and the preceding values are small, then the new value will be large

As a result of this, the lowest values occur when a low value is preceded by a high value. This occurs on mostly on diagonal lines such as the alignment path.

In order to best use parallel processing, the operations are first performed row-wise, then column-wise. The processing is performed on every frame, immediately after the cost matrix is filled. First, the base cost is calculated as normal and inserted into the cost matrix M which stores the unprocessed costs and is used to perform pre-processing on the next row and/or column. The new cost is then calculated in two steps as follows:

$$\begin{aligned} M'[t,u] = M[t,u] + offset &- a \cdot M[t-1,u] \\ &- b \cdot M[t-2,u] \\ &- c \cdot M[t-4,u] \\ &- d \cdot M[t-8,u] \\ &- e \cdot M[t-16,u] \end{aligned} \tag{4.1}$$

$$\begin{aligned} cost = M'[t,u] + offset &- a \cdot M'[t-1,u] \\ &- b \cdot M'[t-2,u] \\ &- c \cdot M'[t-4,u] \\ &- d \cdot M'[t-8,u] \\ &- e \cdot M'[t-16,u] \end{aligned} \tag{4.2}$$

Where $offset$ is a constant needed to ensure the output remains positive and a, b, c, d, e are weights that should add up to a value slightly below 1. The weights chosen for this implementation are $0.10.25, 0.2, 0.15, 0.1$. The first index has less weight since the alignment path may be more than one cell wide. From then on, the constants fade down. By taking indices spaced apart by in a power of 2 pattern, the number of required lookups is kept low, while the reach is higher.

As a final step to the pre-processing, the costs are squared and then divided by a constant to reduce the values. This helps to increase the weight of value differences. It also means that negative output costs are not possible in the unlikely event that a sum turns negative.

figure 4.7 shows an example of how the cost or similarity matrix changes by applying the pre-processing step. The alignment path is not quite as strong as in the raw matrix, but the effect on axis-parallel lines is much stronger. There are almost no low-cost vertical or horizontal lines present in the processed matrix. This should help prevent the algorithm from following the wrong path.

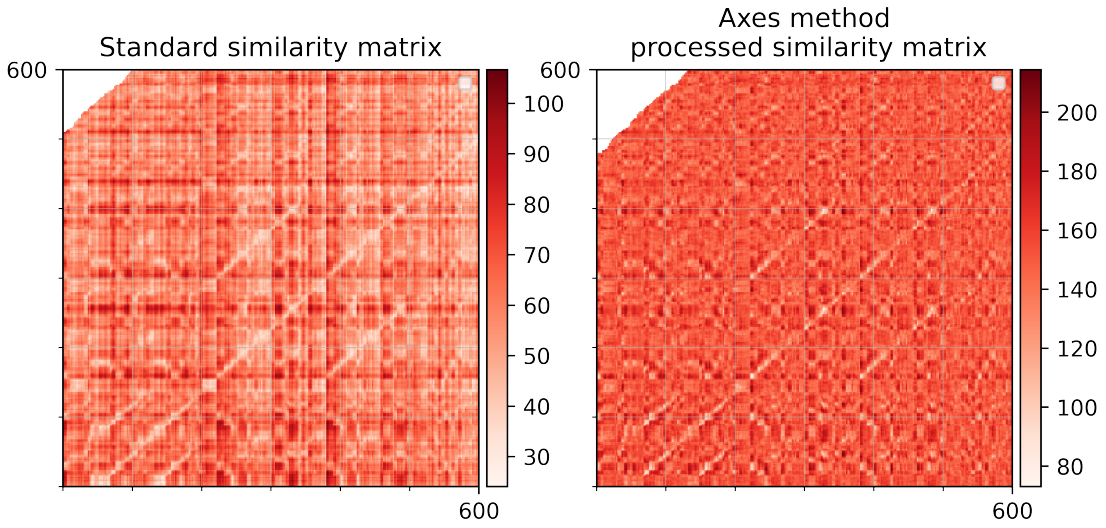


Figure 4.7: The cost matrix before and after applying axes-method pre-processing

Because the processing step requires preceding values, the starting L columns and rows can no longer be used for comparison. Although zero padding could be used to retain these values, this is not necessary since the start of the algorithm is not dependent on the calculated costs. The final L cells of the search width are also no longer used for the output.

In order to get a good start to the path all values before L on either axis are set to 1000, except for the diagonal, where they are set to 0. This does result in negative cost cells around the diagonal for a few row/columns after this, which after squaring incorrectly become positive high-cost cells, but these have no negative impact on the alignment.

4.2.2 The 'diagonal' method

Müller and Kurth [43] have developed a similar processing method for enhancing music similarity matrices, aiming to increase the strength of the alignment. Their method is aimed at offline applications of audio analysis. To replace the basic similarity matrix, they define a *contextual similarity measure* which enhances diagonal paths in the matrix by including contextual information. For better differentiation, this method will be referred to here as the *diagonal* method. The method is introduced in two steps. The first step enhances paths on a fixed diagonal angle (i.e. 45 degrees) by evaluating cost in a sequence diagonally forward from the current index. This is defined as

$$dL(n,m) := \frac{1}{L}\sum_{l=0}^{L-1} d(t(n+l), u(m+l)) \tag{4.3}$$

where $L \in$ is a length parameter, t, and u are the target and reference audio streams respectively and n, m are their indices. In a real time application, this contextual similarity should naturally by calculated forwards rather than backwards.

The second step to the diagonal method is to allow for tempo variability. This is done by explicitly calculating cost values for different simulated tempo ratios and choosing the lowest cost option. The tempo simulation is performed by changing the hop length, i.e. the index frequency. The final similarity measure is then defined as

$$dL^{min}(n,m) = min_r \frac{1}{L}\sum_{l=0}^{L-1} d(t(n+l), w_r(m/r+l)) \tag{4.4}$$

where L, t, n, m correspond to the same parameters as in the first step and w is a set of reference streams at different simulated ratios r.

The diagonal method was implemented on OLTW as follows: On loading the reference audio sequence, instead of generating a single feature array, a number of feature arrays are generated at different simulated tempos. Then, every iteration of the algorithm, the target feature is compared against all these tempo-specific feature arrays, rather than only one. The path cost is then the lowest-cost option of these tempos. The index is shifted on the feature arrays for changed tempos such that the index of comparison equates to the same timestamp.

In this implementation, 7 ratio choices in the range of [0.76, 1.24] in equal steps of 0.08 were used. The ratios need to be chosen carefully as high granularity is required to reach high accuracy. On the other hand the required processing increases with the number of number of ratios to be tested. The length parameter was set to 16.

Figure 4.8 shows how the diagonal method transforms a similarity matrix.

The alignment path is very strongly visible in the processed matrix. This also applies to the repeated structures to either side of the diagonal. Though everything is somewhat

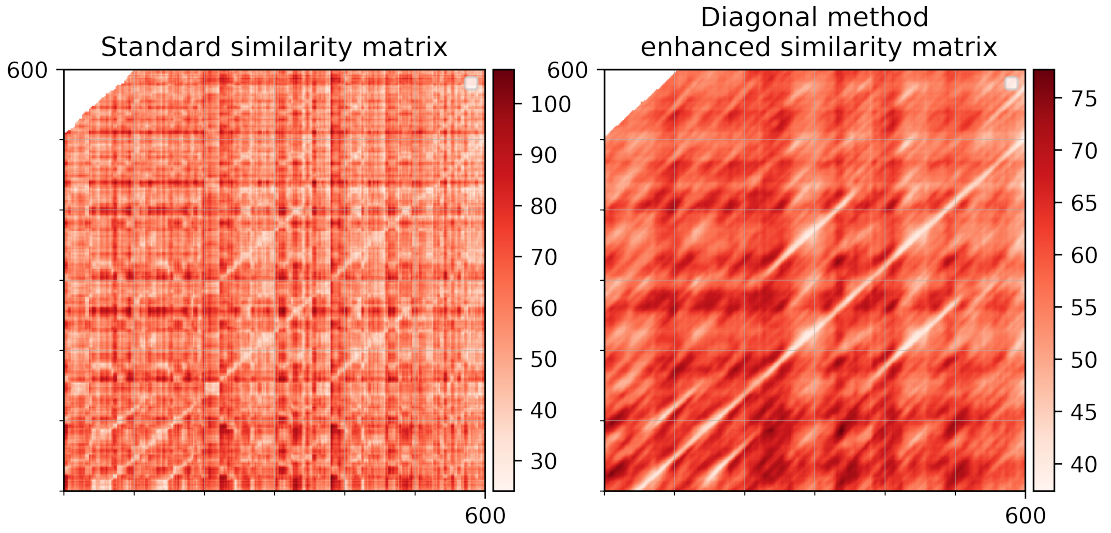


Figure 4.8: The same cost matrix before and after applying diagonal-method pre-processing

smeared out along the diagonal, there are still some axis-parallel lines visible. These do stand out much less compared to the alignment path.

A major downside of this method is the required processing. The explicit simulation of different tempo ratios multiplies the required calculations for comparisons by the number of ratios chosen. With the parameters chosen here, the algorithm could not be executed in real-time on any system tested. Due to the live execution, processing has to be done row-by-row and efficient parallel matrix algorithms cannot be used to their full extent. The current implementation has not been optimised. Despite this, it is likely that the diagonal method is significantly slower than the axes method.

4.3 Using future reference

The idea behind using future reference is simple; In the original implementation of OLTW, as well as those derived from it, the calculation space is restricted up to the current point in the audio stream. This in contrast to how offline DTW uses the full audio streams from start to finish. The online setting of course necessitates limiting the target audio stream index. For the reference stream however, all data is available at the point of initialisation. Thus, instead of only comparing the current target index to past reference indices or conversely comparing the reference index currently last on the alignment path to past target indices, we could instead use these 'future' reference indices. In this mode, we substitute the comparisons of current reference to past target indices with current target to future reference indices. Graphically, this corresponds to filling in the 'missing triangle' as shown in figure 4.9.

The implementation becomes somewhat less straightforward when we consider the process of determining the direction to advance in. In this original case, there are two ranges in which comparisons are made. These ranges go back from the current head of the path in one of two axes. The range in which the lowest cost is found then directly gives

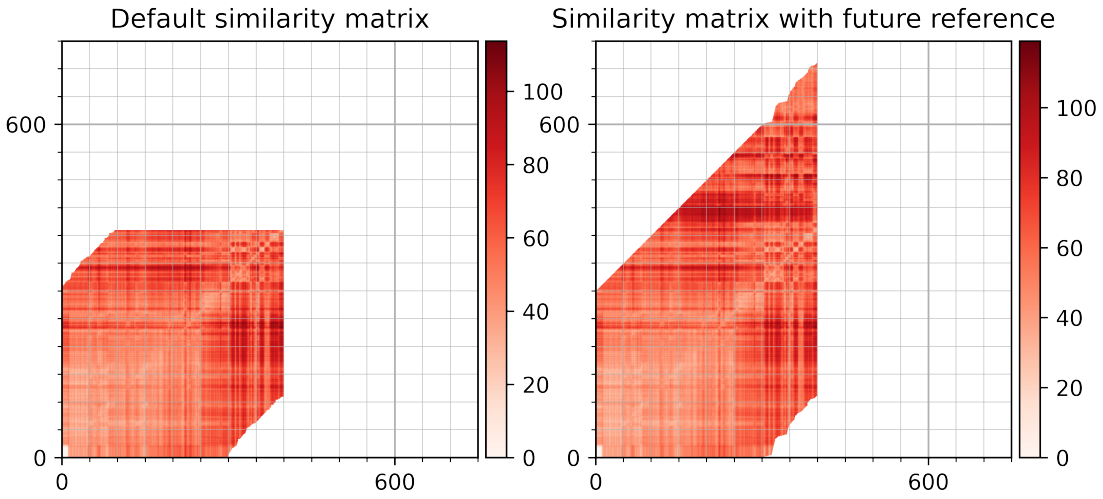


Figure 4.9: Cost matrices with and without future reference compared

the direction to advance in. Advancing one step then amounts to calculating a new range one index past this current one. For the future mode, there is only one range going in both directions from the path head on the reference axis only. In order to find the advancement direction, the range is still split in two, and the range moving in the future direction takes over the role of the range going back in the target axis. With this change, the symmetry between the two axes is lost. As a result of this, the number of calculations needed for advancing in either direction is different. More importantly, the loss of symmetry also has an effect on the alignment performance. Previous tests such as those in the first sections of this chapter show that symmetry around the correct alignment often allows the algorithm to follow the alignment path even if the lowest cost points in the cost matrix do not lie on this path.

Due to the changes made, using future reference is not compatible with the other two techniques presented in this chapter. These techniques could be adapted to work with future reference but the reasoning behind these does not apply very well after the loss of symmetry

5 Evaluation

Testing was carried out in two ways. The first is simulated testing, which was performed offline and without an audience. It was performed by running the algorithm on known recordings with one recording simulated as live input. The second is the user experience trial. In this, the system was employed in a realistic setting where a recorded opera performance was shown as audio and video to an audience instructed to use opera.guru.

5.1 Data

5.1.1 Symphonic test recordings

The most common test data used in literature on music alignment is western classical music. For this project, small scale testing using classical music was used as a less challenging alternative to testing with opera performances. The base algorithm for instance is not able to track opera with any meaningful accuracy, making tests with opera performances less useful.

Two excerpts of symphonic music were used for reference. Gadermaier and Widmer[22] provide annotations for a number of pieces. These annotations are used to compare the found alignment to. Two pieces were used for testing in this project as detailed in table 5.1.

Used as	Orchestra	Conductor	Year	duration
Test piece 1: Beethoven's ninth symphony, first movement				
Reference	Berlin Philharmonic Orchestra	H. von Karajan	1962	15:31
Target	Berlin Philharmonic Orchestra	H. von Karajan	1983	15:35
Test piece 2: Bruckner's ninth symphony, third movement from measure 150				
Reference	Berlin Philharmonic Orchestra	H. von Karajan	1975	09:30
Target	Vienna Philharmonic Orchestra	C. Abbado	1996	10:40

Table 5.1: Symphonic recordings used for testing the system

5.1.2 opera test recording

The opera used for testing is Giuseppe Verdi's Rigoletto. Rigoletto has been performed with opera.guru in the past. Table 5.2 shows the recordings used. For the trial, a section spanning scene 1 and the start of scene 2 was used. Along with the audio recording, the State Opera performance was shown as video on the lecture hall projector. During the transition from scene 1 to scene 2, there is a large discrepancy between the two recordings. In order to give the system the best chance of success during the trial, the reference recording was modified to cut out a 35 second section. This was done because it is easier for the algorithm to correct for a missing section than a mismatched section.

As previously mentioned, the intended use case of the system sees the use of a reference from the same production. Unfortunately such a set of recordings was not available. This left two options for the reference. Either the same recording would be used for both target and reference, which would make the test case easier than the use case. It should be noted using the same recording is still non-trivial due to the signal distortion from playback and recapture through microphone in the trial setup. This version would also suffer from the unfortunate effect of promoting methods that encourage diagonal paths over those more accepting of different alignment paths. The second option is to use a reference recording from a different production. This would result in a more difficult and thus stronger test. The Semperoper recording was used because it is a relatively close fit to the State Opera recording. Among other reasons, this is because Juan Diego Flórez played the part of Duca di Mantova in both performances.

Used as	Opera House	Director	Year	full dur.	start	end
Opera piece: Verdi's Rigoletto						
Reference	Semperoper Dresden	Nikolaus Lehnhoff	2008	2:16:28	3:32	21:33
Target	Vienna State opera	Pierre Audi	2016	2:17:56	6:30	24:10

Table 5.2: Opera recordings used for testing the system

5.1.3 Text

The opera.guru project used for the trials contains a total of 25 chunks in 3 partitions. The first is an empty chunk that should be released at performance start (t=0s). The second chunk describes the scene location and should be released a few seconds into the performance. The last chunk is an acknowledgement and link to the survey. The exact timing is unimportant for all three of these. This leaves 22 timing-critical chunks, since partitions are released automatically with the chunks. The timeline of these can be seen in figure 5.1.

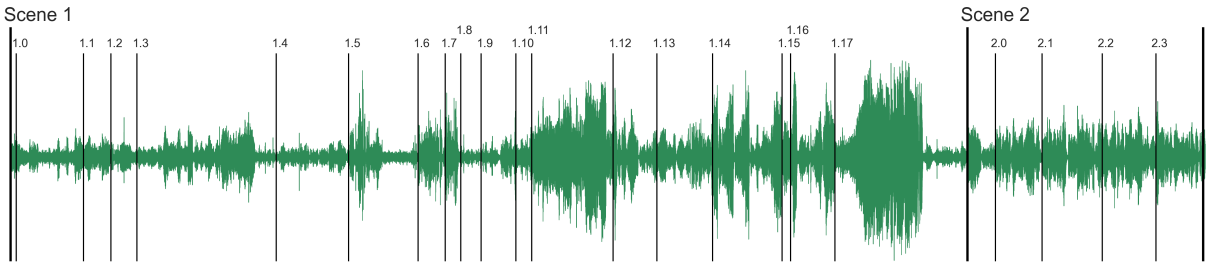


Figure 5.1: Timeline of chunks in the Rigoletto performance

A number of factors were taken into account on composing the chunks. The goal is to guide the audience through the plot without too much distraction. As such, a descriptive style was used over direct translation or transcription. This is in accordance with the general design principles of opera.guru. The content was based on previous performances of Rigoletto in which opera.guru was used but somewhat expanded to include more chunks.

5.2 Simulated Testing

5.2.1 Performance of advancement heuristics

The performance of these advancement heuristics was evaluated on *Test Piece 1* as specified in Table 5.1. Figure 5.2 shows the deviation of the algorithm output compared to this baseline over time. In this graph, the vertical axis represents the deviation, or alignment error, in seconds, whereas the horizontal axis shows the progression through the piece, again in seconds. A negative alignment error represents the found alignment falling behind compared to the baseline. At most points throughout the performance, the four variants perform very similar. Most of the time, the alignment error stays within one second of the baseline alignment. This is a very good result for the desired use case. The weighted mean variant stands out somewhat because it shows more fluctuation at the points where the other variants perform well. The most important parts of the graph are the three sections with larger error spikes. At the start, all variants show a bit more fluctuation with little difference between them. The errors of up to three seconds are still acceptable for the use case. The spike in the middle (around 300 seconds) shows a clear distinction between the variants. Both the base algorithm and the minimum mean heuristic show a large error spike around this point. The variants with both the rolling mean and weighted mean avoid this error spike. After the avoided upward spike, the rolling mean variant does show a prolonged downward error which is also avoided by the weighted mean variant. Just beyond 500 seconds into the test, all variants show a downward error spike. Here, the weighted mean variant shows the smallest error with the other variants all crossing the 5 second error threshold.

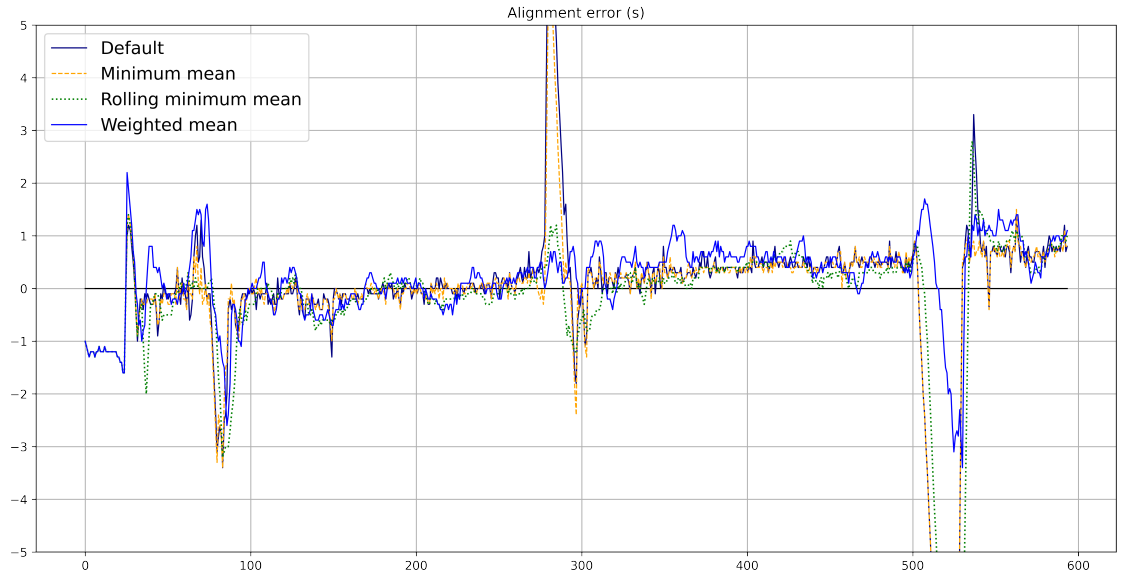


Figure 5.2: Deviation from a manual alignment of the four variants in advancement measures

From these results, the weighted mean heuristic performs the best and shows a significant improvement over the base algorithm.

5.2.2 Performance of Similarity Matrix pre-processing

Both methods were tested on a 10 minute excerpt from the Rigoletto recordings, see table 5.2. This dataset is more representative of the use case and significantly more challenging than the piece used in section 4.1. More in-depth discussion of the evaluation methods is found in chapter 5. The two methods, along with a control test using no pre-processing, were tested in two configurations: One configuration with default algorithm parameters and one configuration with the *weighted mean* measure described in section 4.1.3 used for the best performance.

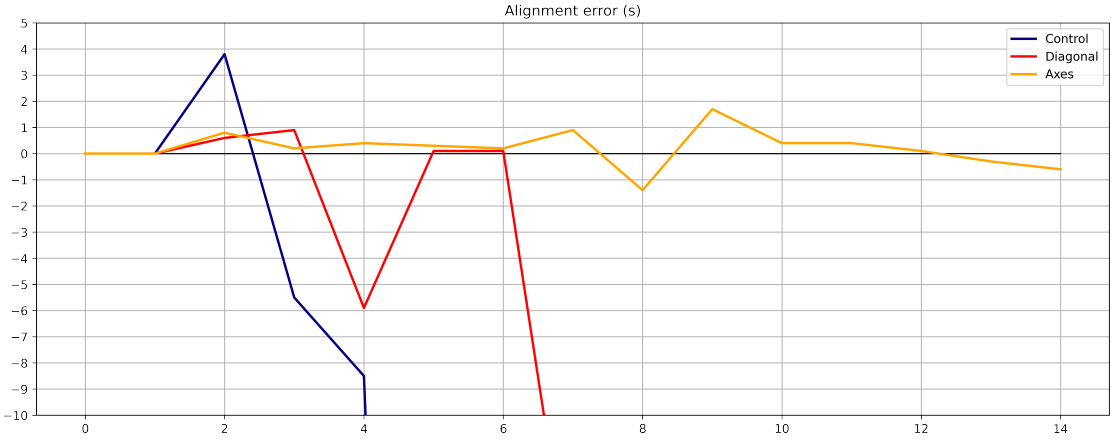


Figure 5.3: Deviation from manual alignment over the control points in the 10 min Rigoletto excerpt

Figure 5.3 shows the alignment error throughout the test performance. Both the control test and the diagonal method lost the alignment path completely and were unable to recover. That said, the inclusion of diagonal method pre-processing improved the performance, allowing it to maintain the alignment longer than the control test. The axes method showed great success, following the correct alignment to the end of the test with much smaller error. The maximum error was below 2 seconds, which is more than acceptable for the desired use case.

The axes method performed slightly worse over the test with weighted mean advancement heuristic, as shown in figure 5.4. The total accumulated error was 11.3 seconds, compared to 7.7 seconds for the previous test, though this is not a significant enough difference to make any definite conclusions over this small test set. The performance of the diagonal method however, improved markedly. The algorithm stayed within range of the alignment throughout the test. The error was still much greater than for the axis method with two spikes to 16 second errors. The total accumulated error for the diagonal method with weighted mean included was 57.5 seconds.

Comparing the cost matrices of figures 4.7 and 4.8, the diagonal method looks more effective to the human eye. The test results tell a different story however. To explain this difference, figure 5.5 shows the two processed matrices around a point where the diagonal temporarily loses the alignment.

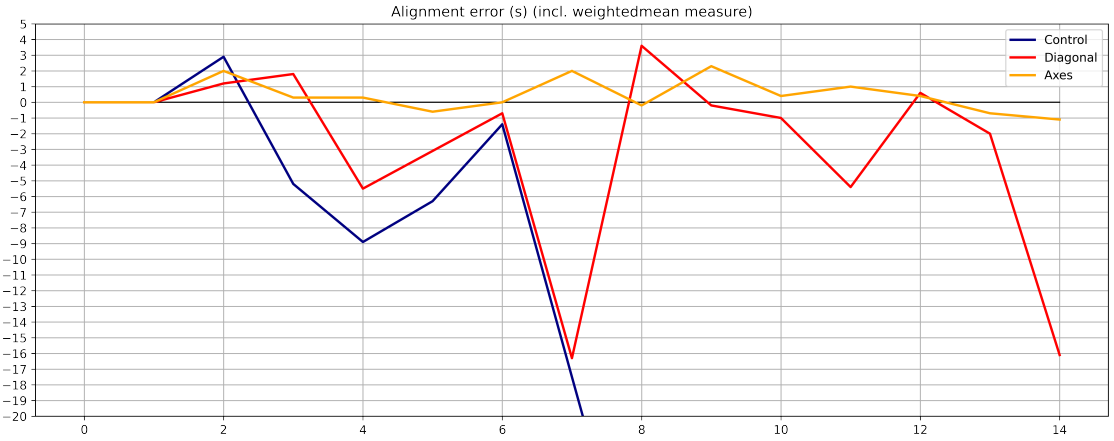


Figure 5.4: Deviation from manual alignment over the control points with the weightedmean measure enabled

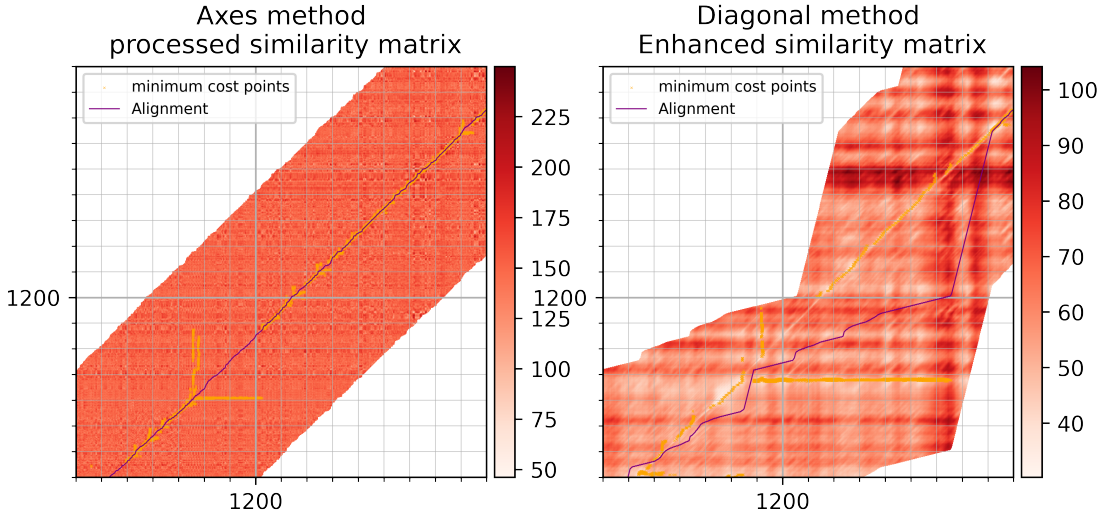


Figure 5.5: Cost matrices of axes and diagonal methods compared

This figure shows that while the alignment path is strengthened in the diagonal matrix, though obscured by the overlay, the vertical and especially horizontal bands of low-cost are still present. This leads to the exact problem the pre-processing methods are supposed to resolve, with the alignment pulled in the direction of the horizontal line around reference frame 1050 on the vertical axis. The *maxRunCount* rule, stipulating that the algorithm may never advance more than three times in the same direction, eventually pulls the alignment away from this line. From there the alignment finds its way back. The axes-method matrix shows no clear horizontal low-cost bands. There are still two short axis-parallel lines in of lowest cost points present, but here these are symmetrical around the alignment and thus do not impact the found alignment.

The results show that axes method pre-processing can provide a significant improvement

to alignment accuracy. It performs better than the diagonal method, which does still provide a small improvement over the baseline. The use of the axes method allows the system to accurately align an opera performance that it was previously incapable of aligning. The axes method is also resource efficient and does not significantly increase the runtime over the default algorithm.

5.2.3 Performance of using future reference

The performance of using future reference data was evaluated using the recordings of Beethoven's ninth symphony that were also used for the advancement heuristics in section 4.1. Figure 5.6 shows the deviation from the manual alignment over this test. The use of future reference worsened performance over nearly the entirety of the test.

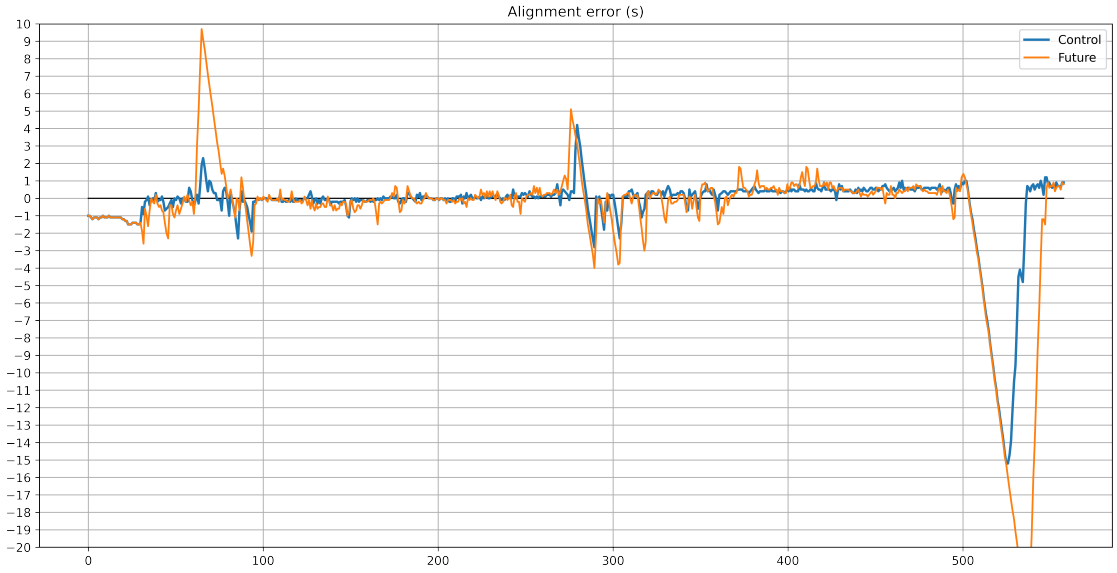


Figure 5.6: Deviation from manual alignment with and without future reference enabled

Using future reference did not prove to be a successful technique. The results clearly show decreased accuracy compared to the default algorithm. The poor compatibility with the other techniques that did prove to be effective is another clear downside.

5.2.4 Opera test

The final simulated test for the system was conducted with the Rigoletto recordings described in table 5.2. For this test, the full-length recordings were used starting from act one. Both the *weightedmean* advancement heuristic and the *axes*-method pre-processing extension were enabled for this test. The searchwidth was set to 600. Though no opera.guru performance exists for the full project, an annotated alignment was made with reference points distributed similarly to the chunks in the opera.guru project created for the trial. A total of 121 reference points were recorded. The alignment error over these reference points is shown in figure 5.7. This shows a somewhat mixed result. Although the error is very large in some places, the alignment does return to the correct path. This shows that the algorithm is capable of tracking the performance throughout the full recordings. The

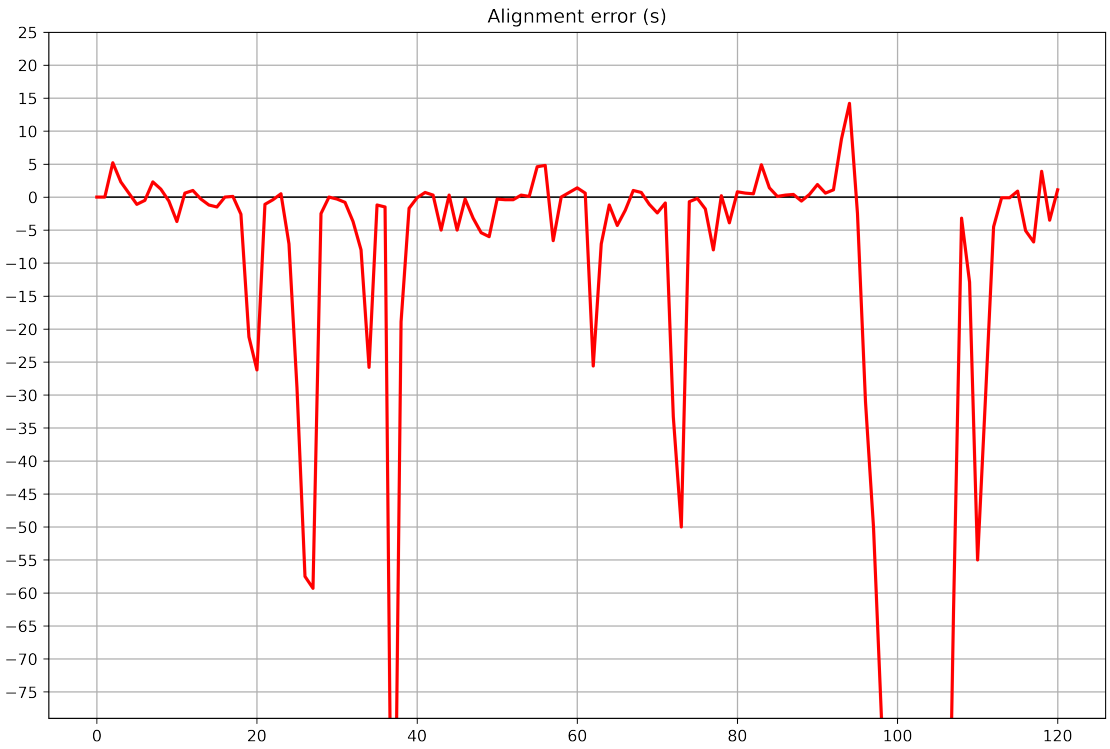


Figure 5.7: Deviation from manual alignment over the Rigoletto performance

found alignment is close to the correct alignment for large parts of the performance, with a number of spikes showing large error interspersed. Near the end of the performance, a very large error extends over 3 minutes behind the correct alignment and this error spike continues through multiple reference points before the alignment catches up. The error at reference point 38 extends to a maximum error of 127 seconds. With such large errors, the system is not usable with opera.guru. That said, it is important to remember that the recordings used in this test present a more difficult problem than is expected in the desired use case.

Closer inspection of the error spikes reveals that at least some of them have a clear cause. Many of these largest errors occur after applause breaks or periods without music during scene transitions. A second cause lies in structural differences between the recordings. Some of these can potentially be mitigated by cutting out sections from the reference recording as was also done with the first scene transition.

Applause breaks occur in nearly all opera performances. There is no meaningful measure of alignment between applause from two different recordings. Therefore, the alignment will start to fall behind. Brazier and Widmer [7] have developed subsystems aimed at solving this problem. They also introduced a subsystem dealing with longer stretches of silence, which cause similar issues.

Since the two recordings come from different productions, there are a number of differences in arrangement. In some places, sections are either missing or repeated in one performance compared to the other. This again will inevitably lead to alignment errors. This issue was addressed in design constraint 1. For practical use of the system, a reference

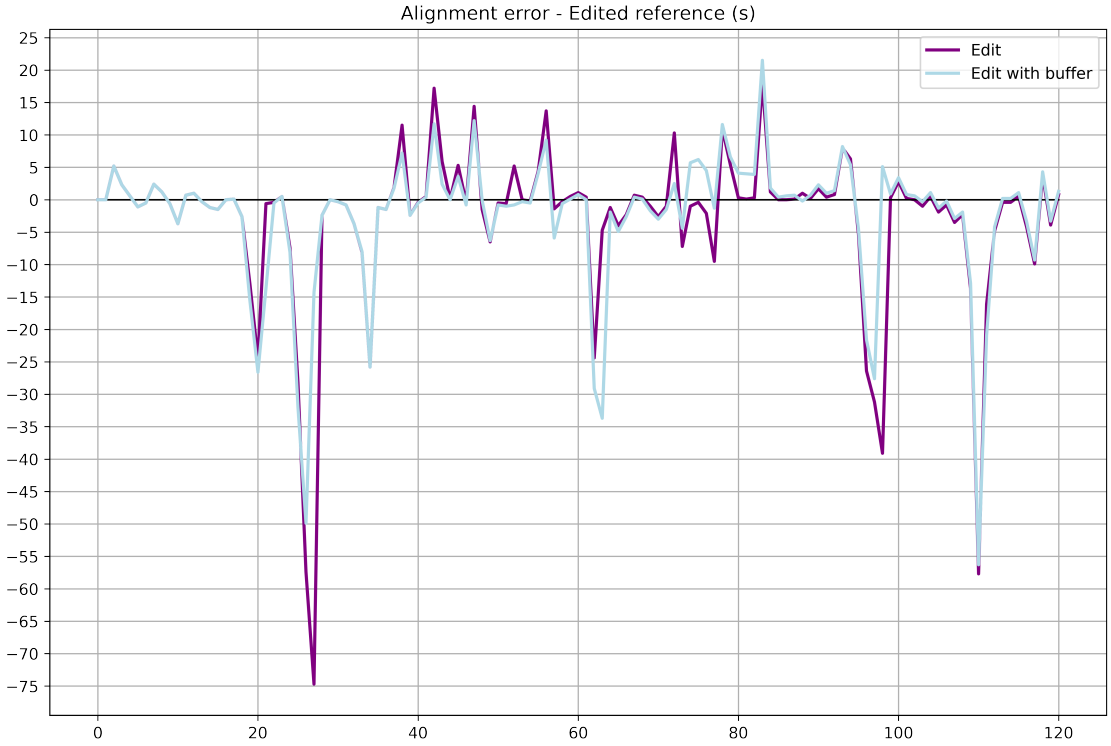


Figure 5.8: Deviation from manual alignment with edited reference for the Rigoletto performance

recording should not have any structural mismatches with the performance.

An edited reference recording was created in order to tackle these issues. Since only the reference recording is edited, this is a process that could feasibly be performed in practical use of the system. Editing was carried out as follows: All sections with applause, significant structural mismatch or scene transitions without musical accompaniment were removed from the reference recording. A total of 12 cuts were made, removing a combined 336 seconds of audio. Figure 5.8 shows the alignment error after rerunning the test with this edited reference recording. A second edit version was also created that includes buffers at the point of editing. The edited performance shows significantly improved performance, though a number of large errors still persist. Some errors do not fall into the categories targeted with the edit, and are thus still present. The edit also introduces new errors; a number of error spikes running 5-15s ahead of the correct alignment. These occur mostly when a cut was made at a point where there was no error spike in the original alignment. The buffered version was created to reduce this phenomenon. Instead of cutting out the relevant section entirely, it is replaced with a silent buffer of 25% the length of the removed section. This is done to give the alignment path some time to adjust its course. This buffered edit version slightly improves the accuracy compared to the initial edit, though the improvement to the newly introduced errors is small. Despite the newly introduced errors, either version of the edited reference recording leads to a significant overall improvement in accuracy.

5.3 User experience trial – Design

Two trials were held, a preliminary trial and a main trial, both held in lecture setting at the University of Vienna.

5.3.1 Procedure

The trials were set up according to the following procedure:

1. The reference recording including queueing points was loaded into to the system prior to the trial.
2. The automated queueing system was run on a computer separate from the one used to display the performance video. The computer was equipped with a high-quality microphone to record the audio in the room. The system sent queueing requests to the opera.guru server over the internet.
3. The audience was shown a section of a pre-recorded Rigoletto performance shown on the projector and lecture hall sound system.
4. The audience was asked to use the opera.guru web-app on their personal smartphones to receive smart subtitles of the performance.
5. The web-app automatically and silently selected one of two performance tracks, placing the user in the respective trial group without their knowledge. The groups were approximately equal in size.
    a) Track 1 received queueing updates released manually by an expert of opera present on location. This reflects the operation as opera.guru has been used so far.
    b) Track 2 received queueing updates placed by the automated queueing system based on comparing the live audio input to a previous reference recording of a different performance of the same piece, with known queueing points.
6. At the end of the performance, the audience were asked to fill in a short survey about the experience.

5.3.2 Audience

The preliminary trial was held during a lecture of the Cooperative Systems lecture at the University of Vienna. The audience was somewhat smaller than the expected 50 students. 31 survey responses were received. The main trial was held during a lecture of the TGI lecture at the University of Vienna. The audience consisted of roughly 100 students, mainly from the Computer Science Bachelor program. 93 survey responses were received.

5.3.3 Microphone placement

One complicating factor that affects the quality of operation during the trial is the placement of the microphone. Reverberation of audio through the lecture hall affects

the signal captured. This is a problem that affects real-world settings too. The correct placement of microphone for recording performances is a broad topic of research on its own. Because the audio was played through the lecture hall speakers rather than live on-location, the microphone setup requirements for the trials was somewhat different. Due to practical constraints, the microphone had to be placed in the front of the lecture hall. This meant that the microphone could not be positioned to adequately capture sound from all speakers. Instead, it was chosen to place the microphone aimed at one speaker in particular. This sound given it the best chance to capture the sound. Audio was recorded using a Blue Yeti microphone.

5.3.4 Queueing procedure

Manual queueing on location was performed by an expert with good familiarity with Rigoletto. Queueing for the automated track was manually aligned to the reference recording by the author. For this alignment procedure the video display was used as well as pausing and rewinding the recording to set timestamps accurately. Timing was registered in half-second increments. There was no coordination between the two queuers. Although this does introduce an additional degree of freedom in the trial outcome, this reflects the freedom granted by the use of a manual queuer who can make decisions in the moment.

5.3.5 Survey

The final chunk contained a link to an anonymous survey that students were asked to fill in. The survey was designed to gauge User Experience for opera.guru in general as well as more specific aspects pertaining to queueing. The trial consisted of a total of 13 questions across 5 pages. Pages 1 and 4 concerned the user's general opinion of opera.guru. Pages 2 and 3 concerned the queueing specifically. Finally, page 5 asked the respondents for miscellaneous feedback and to report any issues encountered. The majority of questions were asked on a 1-5 scale labelled *not at all...* to *very....* Multiple choice questions were mandatory to fill in, open questions were not.

5.4 User experience trial – Results

5.4.1 User Trial responses

In the COSY trial, a total of 31 responses were recorded. of these 6 were discarded. This left 18 responses from the automated track and only 7 responses from the manual track due to a track assignment balancing issue. In the TGI trial a total of 93 responses were recorded. 11 Responses were discarded, leaving 38 responses from the automated track and 44 responses from the manual track. The most common reason for discarding responses was the user experiencing connection issues. This makes the timing responses unreliable. Though these responses were discarded from consideration for gauging the success of the automated queueing system, the results are still considered for further development of the opera.guru system and app. Besides the connection issues, one response was removed for mentioning that they used the application on two devices, for which it is unknown if these two devices were on the same track. Finally, one response was discarded for not giving

a serious response (full response was "weird"). For the final evaluation, only the survey responses from the TGI trial were used.

5.4.2 General satisfaction

On the first page of the survey a number of questions regarding the general satisfaction with using the opera.guru app for subtitles were asked. The results are shown in table 5.3. Accompanying question 2 was an open question asking *"Please indicate why you find opera.guru helpful/unhelpful"*. The majority of responses to this open question were positive. The responses contained useful feedback for opera.guru, but nothing relevant to queueing automation. The questions on general opinion of opera.guru show very close results. Scores are high across the board. The scores for the manual track are slightly higher across the three questions. The differences are too small to say anything conclusively about a link between track and general opinion of opera.guru. What is clear is that if there is a link it is not a large effect. Thus, the automated queueing system was successful in that it did not reduce the general satisfaction with opera.guru by much if at all.

Manual track			Automated track		
avg.	sd.	distribution	distribution	sd.	avg.
1. How did you enjoy using opera.guru? (1-5)					
4.2	0.9			0.9	4.1
2. Did you find opera.guru helpful for following the plot of the opera scene? (1-5)					
4.8	0.7			0.6	4.7
3. Did you find it easy to keep track of both the projected opera scene and the opera.guru app? (1-5)					
4.2	1.1			0.9	4.1

Table 5.3: Results for the first page of the survey

5.4.3 Queueing timing

The second page of the survey asked questions directly relating to the quality of the timing of messages. This is influenced by more than just the performance of the alignment algorithm. As previously mentioned, the timing of messages was not coordinated between queueing directors for the two tracks, which may have had an influence. Additionally, the composition of messages may have had an effect on the perceived timing. The responses should give a reflection of the actual user experience of the message timing. The results, presented in table 5.4, show a small difference in favour of the automated track. Again, the difference is too small to show a clear link giving the sample size. For the perceived timing quality, the automated queueing system performed approximately equal or better compared to the manual track. This indicates a success for the system.

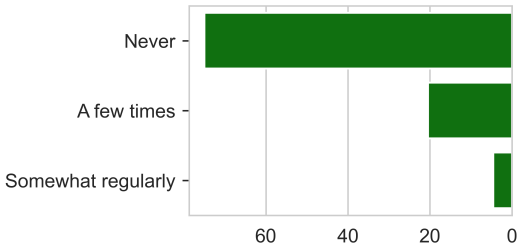
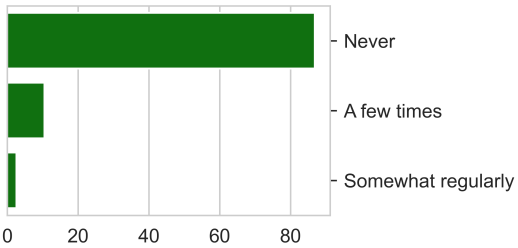
Manual track			Automated track		
avg.	sd.	distribution	distribution	sd.	avg.
4. Overall, how good was the timing of messages shown in the app? (1/5)					
4.2	0.8			0.8	4.3
5. Overall, how consistent was the timing of messages shown in the app? (1-5)					
4.2	0.9			0.7	4.4
6. Did you experience any instances of messages being delivered at completely the wrong time? (Never / A few times / Somewhat regularly / Often)					
					

Table 5.4: Results for the second page of the survey

5.4.4 Guessing the trial group

The third page of the survey discloses to the users that the audience was divided into two tracks and reveals the automated queueing system. Then, users were first asked question 7 from table 5.5 to gauge their pre-conceived ideas regarding the relative expected performance of the two tracks. Respondents slightly favoured automated control, though with a wide range of responses.

The second question asked users to guess which track they were in. Interestingly, the majority of respondents thought they were in the automatic track. This was virtually identical for both tracks respondents were actually in. Thus, with equal guesses from both tracks, respondents could not tells whether their performance was controlled manually or automatically. This makes a strong case for the effectiveness of the automated system.

A closer look at the relation between confidence in either control method and track guess, reveals an interesting correlation, see figure 5.9. Those who actually were in the manual control group and guessed this correctly, overwhelmingly had equal confidence in both control methods. Conversely those who incorrectly guessed they were in the automatic control group largely had a stronger opinion either way on which control method would perform better. This effect is largely absent in the larger subset of respondents that guessed they were in the automated control group.

5.4.5 General statements regarding opera.guru

Page four asked respondents for the degree to which they agreed with four statements regarding opera.guru. The statements are designed to measure the respondents' general

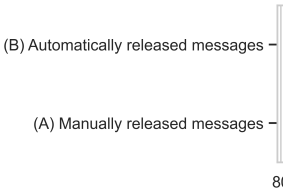
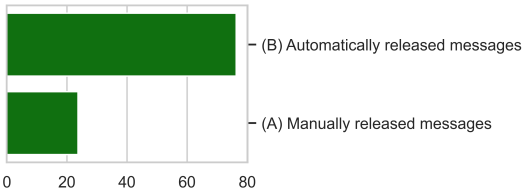
Manual track			Automated track		
avg.	sd.	distribution	distribution	sd.	avg.
7. In which method of control do you have more confidence to accurately release messages? (Much more confidence in manual control - Much more confidence in automated control (1-5))					
3.0	1.2			1.2	3.4
8. Which group do you think you were in?					
					

Table 5.5: Results for the third page of the survey

opinion about using opera.guru on different axes. The results are shown in table 5.6. The results are similar to the questions from page 1 with scores very close between the two tracks with a small edge to the manual track.

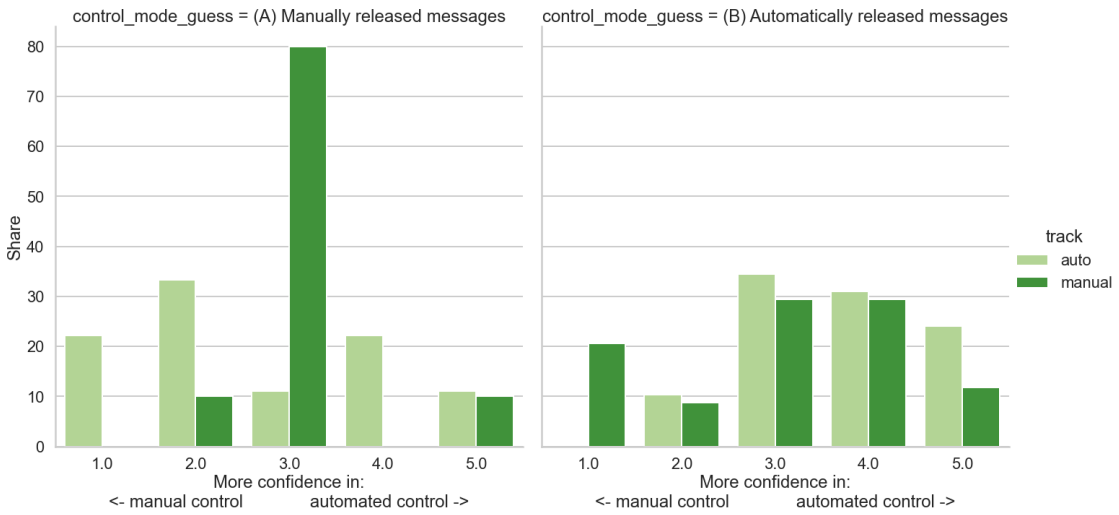


Figure 5.9: Comparison of control method confidence separated by control mode guess

Manual track			Automated track		
avg.	sd.	distribution	distribution	sd.	avg.
opera.guru is complicated / easy					
4.8	0.5			0.7	4.6
opera.guru is confusing / clear					
4.3	1.0			0.8	4.4
opera.guru is inefficient / efficient					
4.5	0.7			0.6	4.4
opera.guru is boring / exciting					
4.0	0.9			0.9	3.8

Table 5.6: Results for the fourth page of the survey

6 Discussion

This chapter discusses the project, examining the choices made and the results gathered as well as the interpretation of these results. This includes a look at potential shortcomings of the project and the degree to which the goal of automating queueing in opera.guru was achieved. A more practical view of how the system may be used and its limitations is also given.

6.1 Implementation

In accordance with the design goals, requirements and constraints, a program was developed that allows the automation of queueing in opera.guru based on a reference recording. The implemented algorithm is based on Dynamic Time Warping due to its suitability for audio-to-audio alignment. The alignment system was implemented as a standalone python program. It was designed such that asymptotic runtime was kept linear in regards to the performance length. Effective runtime efficiency was not optimised, however. In practice, the system ran in real time on the available hardware. With a more optimised and a fully parallelised implementation, lower hardware requirements could be achieved.

Most subsystems and algorithm extensions presented in the literature were not re-implemented for this project. This was done to keep the scope manageable while allowing for maximal new contributions from this project.

The algorithm, as initially implemented was not sufficiently accurate. The improvements presented in chapter 4 were aimed at changing this. Most of the different advancement heuristics presented were based on averaging between frames. These do not improve base alignment accuracy, but do mask poor accuracy at the cost of adding alignment latency. The weightedmean method is different. Evaluation showed a substantial increase in accuracy without introducing latency. The weightedmean method can reduce maximal accuracy, effectively trading a higher median error for a lower maximal error. In most cases this is preferable.

The matrix pre-processing extensions yielded good results as the axes method provides a significant improvement to the alignment accuracy. Of the two methods, the axes method has shown to be the most effective. There is likely still room for improvement with the specific implementation of the extension. A larger study of different approaches to reduces linear patterns is likely to produce an even bigger improvement. The diagonal method also showed some improvements over the baseline, but proved to be less effective than the axes method. It also suffered from poor computational efficiency. Due to the rapidly expanding number of calculations needed for more granular parameter settings, especially the ratio steps, the settings required for the best accuracy were not feasible in real-time use. The use of future reference did not lead to any accuracy benefits. Additionally, its incompatibility with the other extensions, which did prove to be effective, is a significant obstacle to its use.

6.2 Trial and evaluation

The trial, and the survey responses associated with it, showed that the system worked as desired. Observationally, the system followed the performance and mostly released messages when expected. The survey results match with this assessment as the automated system received near-identical scores to the manual system. It seems likely that the tolerance for deviation from the intended queueing point is high. There is no one obvious matching moment to which the type of descriptions used in opera.guru apply.

One major shortcoming of the trial is the audience, comprised of students in computer science courses. This is not an audience representative of the general public interested in opera performances. Audiences more familiar with opera, especially those already familiar with the specific work performed may be more observant to the timing differences. That said, opera.guru is targeted mostly at newcomers to opera.

For the trial, an excerpt of roughly 18 minutes of Rigoletto was used. This is a rather limited test case. For a conclusive argument for the efficacy of the automated system trials with multiple full opera performances are needed. Although the simulated test used the full Rigoletto recordings, slightly over two hours long, this is still rather limited. With only one opera used for evaluation, there is a risk of overfitting. It is possible that the improvements do not show the same results on other operas, especially those from different eras and styles.

From the survey results, a seemingly interesting link appeared that, while far from conclusive gives rise to speculation over the effect of queueing on user experience. The trend is that while the automated track resulted in higher user opinion over timing accuracy itself, the manual track resulted in higher general opinion of opera.guru. A possible interpretation is that while the automated queueing is accurate, the ability of a manual queuer to make choices in the moment allows for a more enjoyable experience. Of course, it is also likely that this effect is a result of the different styles of the queueing operators and not dependent on the method of control. Further trials are needed to show if this link actually exists.

The large alignment errors seen in the simulated opera test indicate that more improvements are still needed for the system to be usable in arbitrary circumstances. The test did show that system, designed to work on arbitrarily long audio sequences, is indeed not dependent on the length of the input. In this test, the disadvantage of using recordings from different productions becomes clear. The test does not provide a representative view of the algorithm performance in the opera.guru use case. Better performance is expected with recording from the same production, though it is unknown how much improvement this brings.

The project to improve accuracy in live opera tracking by Brazier and Widmer described in Ch. 2 the literature [7] [8] achieved limited success. The authors claimed that music alignment was not ready for the task. Has this project changed this? Direct comparisons between their work and this project are difficult. The first reason for this is caused by the use of different test pieces. The second reason is that the aims of the two projects differ. This project started from a specific use case. From this, the requirements and constraints laid out in sections 3.1.2 and 3.1.3 were chosen such that the system had the highest chance of success while achieving the important goals for the use case. The setup of opera.guru with its descriptive style messages requires accuracy within a few seconds. This assertion seems to match the observations from the trial. A different system providing

one-on-one transcription subtitles would require far higher accuracy compared to the more general descriptions favoured in opera.guru. Other projects in music alignment, such as the project of Brazier and Widmer, often have more strict accuracy requirements. Where in this case alignment within a few seconds of the correct alignment is sufficient, a different context may require sub-second accuracy.

A second difference is the reference material. In this project, it is assumed that a reference recording features the same arrangement and performing artists as the target performance. In the case of the Rigoletto performance used in the trial, the reference recording was specifically chosen because it was similar to the target performance. Nonetheless, there were significant differences between the two performances which will have had an impact on alignment accuracy. General purpose music alignment algorithms are expected to work with any reference of the same piece.

The constraints for the design do limit the usefulness of the system in a few areas. opera.guru is designed to be flexible. Multiple interpretations of profiles and partitions are possible. The limitations of the automated queueing system are such that it is designed to support only the core use case of opera.guru i.e. opera performances with relatively infrequent descriptive messages. The constraint of using only reference recordings from the same production also creates a higher bar of entry to using the system.

6.3 User experience of managing opera.guru performances

The goal of the automated queueing system is to remove the need for a repetitive manual task. It is worth looking into how exactly the workload needed to use opera.guru changes when using the automated system:

In current use, the following tasks are required to hold a performance with opera.guru: First, an opera.guru performance must be created using the management interface. This mainly entails the writing of chunk text and partition summaries for all desired profiles and languages. Then, for every performance of the work, the queueing operator must control the system requiring their attention for the full duration of the work.

When using the automated system, the step of creating the opera.guru performance remains the same. Additionally, a representative audio recording of the full work is needed. This can be obtained for instance by recording the final rehearsal, or if the system is not yet used, the premiere performance. A queueing timeline is also needed. This can be obtained either by manual alignment outside of a performance or by recording manual queueing during a performance. Creating the queueing timeline outside of a performance provides a few benefits such as allowing for pausing and rewinding as well as correcting mistakes. For the use of the system a recording setup is also needed. Once these prerequisites are met, the performance on opera.guru can be run without manual intervention during the performance.

The automated queueing system requires some work to be done before it can be used, although this not significantly more work than what is needed for a single performance without the automated system. As such it is not particularly useful for singular performances. The more times a performance is repeated however, the more work the system saves.

In case a suitable reference recording is not available, editing can be used to increase accuracy as seen in the evaluation. This thus increases the workload for using the system.

Even in cases where a suitable reference recording is available, the system has not shown to be dependable. Thus a human operator would most likely still have to be at hand.

In order to be usable, the system must be dependable, which it is not in its current state. There are also practical reasons why the automated system could fall short. If, for whatever reason the performance has to be different than expected, the system might not be able to keep alignment through these differences. In case something goes wrong with the automated system, a human operator might still have to be at hand, negating many of the advantages of the automated system. This also applies to the instant message chunks which are by their very nature written and sent during the performance by a human. On the other hand, automated queueing would give this operator more time to write such instant messages.

Despite these potential shortcomings, this project has been largely successful. It has shown that under the right circumstances and with a validated performance, the system works and works well.

7 Conclusion

7.1 Summary

With the aim of automating the queueing of an opera.guru performance, the main effort of this project was the implementation of a music alignment algorithm. The development of the axes method pre-processing extension additionally In chapter 2, the existing literature on music alignment was explored. Three subcategories of research topics were identified; the first two categories are distinguished by the technique used at the core of the alignment algorithm. The third category concerns the audio features used to represent the incoming audio stream, which are needed in all approaches. Of the two techniques, algorithms derived from Hidden Markov Models apply largely to audio-to-score alignment, whereas algorithms derived from Dynamic time warping apply largely to audio-to-audio-alignment. In chapter 3, the implementation of the algorithm and the system in general is described in detail. Since the opera.guru use case is best served with audio-to-audio alignment, a DTW-derived algorithm was chosen. This algorithm sequentially builds a cost matrix representing the space of possible alignments. Although multiple audio features were implemented, an MFCC feature is used as the most effective option, in accordance with the results from literature. The system is designed to provide live alignment to an existing reference recording. Integration with the existing opera.guru back-end allows the system to control a live opera.guru performance.

Chapter 4 concerns the exploration of potential improvements to the algorithm's alignment accuracy. The first extension presented, advancement heuristics, showed success, especially the *weightedmean* method. *Weightedmean* calculates a weighted mean position over the all cells in the final row and column of the cost matrix, which is then used to determine the direction to advance in. The second extension, *pre-processing* of the cost matrix was the most successful. Two different approaches to making the alignment path stand out from the background were evaluated. The newly developed *axes* method lead to significant accuracy improvements and proved to be a game-changer in that it made opera tracking a reachable goal with the system. This method reduces the formation of axis-parallel patterns by subtracting a combination of preceding cost values. The final potential improvement of including future reference did not show any useful improvements.

The capabilities of the system were evaluated in two ways, as described in chapter 5: Offline simulated tests and a live user experience trial. Through simulated testing, both the alignment accuracy for the improvements developed in chapter 4 and the overall accuracy of the system through a test on a full opera performance was evaluated. The full system test showed that the algorithm was generally capable of tracking the performance but occasionally showed large errors. In some of these errors the alignment fell behind by more than a minute. It should be noted that this test showcased somewhat of a worst case scenario since the two recordings used did not come from the same production as is expected for the desired use case of the system. The recordings used therefore show more differences than expected.

The user experience trial provided a showcase for the system in a more practical environment. The audience was shown a recording of Rigoletto in a lecture hall setting and were asked to use the opera.guru. Unknown to the participants, the opera.guru performance was controlled either by a manual queueing operator or the automated system. A post-trial survey was used to was the participants' user experience. The trial showed that it is possible to use the system to replace manual queueing and retain high user satisfaction and a good user experience. Users were incapable of distinguishing the methods of control, indicating that the automated system can seamlessly take over the role of a manual queueing operator.

7.2 Further research

Many extensions and improvements to DTW-based music alignment algorithms have been researched in the past. Implementing all of these was outside the scope of this project. Many of these features would improve the automated queueing system. Implementing these features would also provide better context for the effectiveness of the improvements presented in this thesis.

Another avenue for future research lies in the evaluation. Evaluating the system with more opera pieces from various periods and styles is needed to build a clear picture of the overall effectiveness and applicability of the automated system. More user experience trials would also improve the evaluation, especially trials held in an actual live opera setting. The many parameters and choices for audio features and extensions are also an opportunity for research to find out the most effective configurations.

The recording setup used is an important factor in the resulting audio. This naturally affects the alignment algorithm too. A future project could look into the recording setups used in various opera houses to find if there are major differences and how those differences affect alignment quality.

The current integration of the automated system with opera.guru is limited. More work could be done to improve the user experience of managing the automated system and deploying it with opera.guru. Integration in the existing management interface is a clear goal here. A particularly useful feature would be the automated creation of queueing reference files for performances. This would make the switch to automated queueing near-seamless after a single manual performance.

Although many ideas for improvements to the alignment algorithm have been explored, both in previous literature and the ones presented in this thesis, further improvements are most likely still possible. One idea that could be useful for the use case of opera.guru is a hierarchical multi-level tracker that takes advantage of the constraint that accuracy is only relevant at queueing points to distribute resources differently.

Finally, there are many ways in which the software implementation of the automated queueing system could be improved. A new version of the software could be designed to increase efficiency and optimise for runtime by making better use of parallelisation A more extensive user interface would improve user experience by reducing the need to edit files outside the system.

Bibliography

[1] A. Arzt and G. Widmer. Simple tempo models for real-time music tracking. In *Proceedings of the Sound and Music Computing Conference (SMC)*, page 9, 2010.

[2] A. Arzt and G. Widmer. Towards Effective 'Any-Time' Music Tracking. volume 222, pages 24–36, Jan. 2010.

[3] A. Arzt and G. Widmer. Real-Time Music Tracking Using Multiple Performances as a Reference. In *ISMIR*, pages 357–363. Citeseer, 2015.

[4] A. Arzt, G. Widmer, and S. Dixon. Automatic Page Turning for Musicians via Real-Time Machine Listening. In *ECAI*, pages 241–245, 2008.

[5] A. Arzt, G. Widmer, and S. Dixon. Adaptive distance normalization for real-time music tracking. In *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 2689–2693. IEEE, 2012.

[6] B. S. Atal and S. L. Hanauer. Speech analysis and synthesis by linear prediction of the speech wave. *The journal of the acoustical society of America*, 50(2B):637–655, 1971.

[7] C. Brazier and G. Widmer. Towards reliable real-time opera tracking: Combining alignment with audio event detectors to increase robustness. In *In Proceedings of the 17th Sound Music Computing Conference (SMC 2020)*, 2020.

[8] C. Brazier and G. Widmer. Addressing the recitative problem in real-time opera tracking. In *Advances in Speech and Music Technology*, pages 157–168. Springer, 2021.

[9] J. C. Brown. Calculation of a constant Q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[10] J. C. Brown and M. S. Puckette. An efficient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92(5):2698–2701, 1992.

[11] P. Cano, A. Loscos, and J. Bonada. Score-performance matching using HMMs. In *ICMC*. Citeseer, 1999.

[12] A. Cont. Realtime audio to score alignment for polyphonic music instruments, using sparse non-negative constraints and hierarchical HMMs. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE, 2006.

[13] A. Cont. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *International Computer Music Conference (ICMC)*, pages 33–40, 2008.

[14] A. Cont. A coupled duration-focused architecture for real-time music-to-score alignment. *IEEE transactions on pattern analysis and machine intelligence*, 32(6):974–987, 2009. Publisher: IEEE.

[15] R. B. Dannenberg. An on-line algorithm for real-time accompaniment. In *ICMC*, volume 84, pages 193–198, 1984.

[16] R. B. Dannenberg and L. Grubb. Automating ensemble performance. In *Procs of ICMC94*, pages 63–69, 1994.

[17] S. Dixon. Live tracking of musical performances using on-line time warping. In *Proceedings of the 8th International Conference on Digital Audio Effects*, volume 92, page 97. Citeseer, 2005.

[18] M. Dorfer, A. Arzt, and G. Widmer. Towards score following in sheet music images. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, 8 2016.

[19] Z. Duan and B. Pardo. A state space model for online polyphonic audio-score alignment. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 197–200. IEEE, 2011.

[20] S. Ewert, M. Muller, and P. Grosche. High resolution audio synchronization using chroma onset features. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1869–1872. IEEE, 2009.

[21] C. Fang. From dynamic time warping (DTW) to hidden markov model (HMM). *University of Cincinnati*, 3:19, 2009.

[22] T. Gadermaier and G. Widmer. A Study of Annotation and Alignment Accuracy for Performance Comparison in Complex Orchestral Music. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, pages 769–775, Delft, The Netherlands, Nov. 2019. ISMIR.

[23] O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms (TALG)*, 14(4):1–17, 2018. Publisher: ACM New York, NY, USA.

[24] M. Grachten, M. Gasser, A. Arzt, and G. Widmer. Automatic Alignment of Music Performances with Structural Differences. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, pages 607–612, Curitiba, Brazil, Nov. 2013. ISMIR.

[25] L. Grubb and R. B. Dannenberg. A stochastic method of tracking a vocal performer. In *ICMC*, 1997.

[26] L. V. Grubb. A probabilistic method for tracking a vocalist. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1998.

[27] Y. Guédon. Hidden hybrid Markov/semi-Markov chains. *Computational statistics & Data analysis*, 49(3):663–688, 2005. Publisher: Elsevier.

[28] F. Henkel, R. Kelz, and G. Widmer. Learning to read and follow music in complete score sheet images. In *Proceedings of the 21st International Society for Music Information Retrieval Conference*, pages 780–787, Montreal, Canada, Oct. 2020. ISMIR.

[29] N. Hu, R. B. Dannenberg, and G. Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*, pages 185–188. IEEE, 2003.

[30] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[31] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on acoustics, speech, and signal processing*, 23(1):67–72, 1975.

[32] C. Joder, S. Essid, and G. Richard. A comparative study of tonal acoustic features for a symbolic level music-to-score alignment. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 409–412. IEEE, 2010.

[33] C. Joder, S. Essid, and G. Richard. An Improved Hierarchical Approach for Music-to-symbolic Score Alignment. In *ISMIR*, pages 39–45, 2010.

[34] H. Kirchhoff and A. Lerch. Evaluation of features for audio-to-audio alignment. *Journal of New Music Research*, 40(1):27–41, 2011. Publisher: Taylor & Francis.

[35] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[36] M. H. Ko, G. West, S. Venkatesh, and M. Kumar. Using dynamic time warping for online temporal fusion in multisensor systems. *Information Fusion*, 9(3):370–388, 2008. Publisher: Elsevier.

[37] F. Korzeniowski, F. Krebs, A. Arzt, and G. Widmer. Tracking rests and tempo changes: Improved score following with particle filters. In *ICMC*, 2013.

[38] H. Li. On-line and dynamic time warping for time series data mining. *International Journal of Machine Learning and Cybernetics*, 6(1):145–153, 2015. Publisher: Springer.

[39] B. Logan. Mel Frequency Cepstral Coefficients for Music Modeling. In *Proceedings of the 1st International Symposium on Music Information Retrieval*, Plymouth, United States, Oct. 2000. ISMIR.

[40] R. Macrae, X. Anguera, and N. Oliver. Muvisync: Realtime music video alignment. In *2010 IEEE International Conference on Multimedia and Expo*, pages 534–539. IEEE, 2010.

[41] R. Macrae and S. Dixon. Accurate real-time windowed time warping. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, pages 423–428, Utrecht, Netherlands, Aug. 2010. ISMIR.

[42] B. McFee, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, D. Ellis, J. Mason, E. Battenberg, S. Seyfarth, R. Yamamoto, viktorandreevichmorozov, K. Choi, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, A. Weiss, D. Hereñú, F.-R. Stöter, P. Friesch, M. Vollrath, T. Kim, and Thassilo. librosa/librosa: 0.9.1, Feb. 2022.

[43] M. Muller and F. Kurth. Enhancing similarity matrices for music audio analysis. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE, 2006.

[44] M. Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Springer, 2015.

[45] M. Müller, V. Konz, A. Scharfstein, S. Ewert, and M. Clausen. Towards Automated Extraction of Tempo Parameters from Expressive Music Recordings. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, pages 69–74, Kobe, Japan, Oct. 2009. ISMIR.

[46] I. Oregi, A. Pérez, J. Del Ser, and J. A. Lozano. On-line dynamic time warping for streaming time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 591–605. Springer, 2017.

[47] N. Orio and F. Déchelle. Score following using spectral analysis and hidden Markov models. In *ICMC: International Computer Music Conference*, pages 1–1, 2001.

[48] N. Orio and D. Schwarz. Alignment of monophonic and polyphonic music to a score. In *International Computer Music Conference (ICMC)*, pages 1–1, 2001.

[49] T. Prätzlich, J. Driedger, and M. Müller. Memory-restricted multiscale dynamic time warping. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 569–573. IEEE, 2016.

[50] M. Puckette and C. Lippe. Score following in practice. In *Proceedings of the International Computer Music Conference*, pages 182–182. INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 1992.

[51] L. Rabiner and B. Juang. An introduction to hidden Markov models. *ieee assp magazine*, 3(1):4–16, 1986. Publisher: IEEE.

[52] C. Raphael. Automatic segmentation of acoustic musical signals using hidden Markov models. *IEEE transactions on pattern analysis and machine intelligence*, 21(4):360–370, 1999.

[53] C. Raphael. Aligning music audio with symbolic scores using a hybrid graphical model. *Machine learning*, 65(2-3):389–409, 2006. Publisher: Springer.

[54] P. Reichl, C. Löw, S. Schröder, T. Schmidt, B. Schatzl, V. Lushaj, O. Hödl, F. Güldenpfennig, and C. Widauer. The salome experience: Opera live streaming and beyond. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, page 728–737, New York, NY, USA, 2016. Association for Computing Machinery.

[55] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. Interspeech 2014*, pages 338–342, 2014.

[56] S. Sako, R. Yamamoto, and T. Kitamura. Ryry: A real-time score-following automatic accompaniment playback system capable of real performances with errors, repeats and jumps. In *International Conference on Active Media Technology*, pages 134–145. Springer, 2014.

[57] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[58] D. Schwarz, N. Orio, and N. Schnell. Robust Polyphonic Midi Score Following with Hidden Markov Models. In *International Computer Music Conference (ICMC)*, pages 1–1, Miami, United States, Nov. 2004.

[59] F. Sha and L. K. Saul. Real-time pitch determination of one or more voices by nonnegative matrix factorization. *Departmental Papers (CIS)*, page 168, 2004.

[60] S. Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.

[61] C. J. Tralie and E. Dempsey. Exact, parallelizable dynamic time warping alignment with linear memory. In *Proceedings of the 21st International Society for Music Information Retrieval Conference*, pages 462–469, Montreal, Canada, Oct. 2020. ISMIR.

[62] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

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

[64] G. Xia, Y. Wang, R. B. Dannenberg, and G. Gordon. Spectral Learning for Expressive Interactive Ensemble Music Performance. In *ISMIR*, pages 816–822, 2015.