Live Score Following

Brandon Apodaca[1] and Marco Conati[2]

***Abstract*— This paper details the construction of an algorithm which utilizes Dynamic Time Warping to align a query audio clip to a reference music score in real time. The proposed method was consistently able to converge on the true score location. Over the course of eight iterations on queries between 240 and 480 seconds in length, the system converged to the true score location in 5.75 seconds on average.**

I. INTRODUCTION

Live score following is the process of aligning a predetermined musical representation with a live audio performance. This concept is not novel. Real-time score following has been explored since the 1980's and has recently been an active area of research for IRCAM, the Institute for Research and Coordination in Acoustics/Music[6].

Our goal was to create a live score following system for sonified monophonic MIDI audio and evaluate the error of our overall system. Additionally, our objective was to enhance the monophonic system to improve accuracy and increase the scope to polyphonic signals. This system could ideally be adapted to create an application to follow a performance and advance sheet music accordingly.

II. SYSTEM DESIGN

The problem can be broken down into three parts: data acquisition, algorithm development, and accuracy evaluation. A block diagram in Figure 1 shows how this is organized.

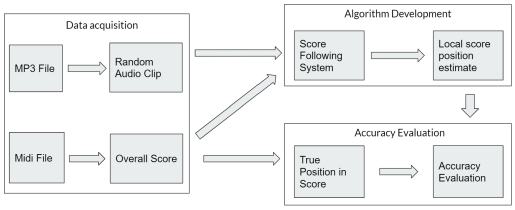

Fig. 1. Block diagram of design process

The dataset consisted of both manually recorded and MIDI generated audio. Annotations of the score information were needed for each audio file. For the manually recorded audio, a score was transcribed by hand and hard coded into the Jupyter notebook file with the score following algorithm. For the MIDI generated audio, the MIDI file was converted to a csv and parsed to extract score information.

For algorithm development, our primary objective was to create a system that can align the score and our query audio. To convert the score and query into a consistent form, both were converted to Chroma feature matrices. These matrices had shape $12 * n$ and $12 * m$, where each column i represents the 12 Chroma features frame i.

For aligning the two sequences, the primary tool used was Dynamic Time Warping (DTW). This algorithm that finds the lowest cost path through a cost matrix given allowed steps with corresponding weights. The cost matrix was based on cosine distance, so that a path through the sequence would be generated where cosine similarity was greatest. The full algorithm is detailed in section IV.

III. DATA PREPARATION

Audio data was both recorded manually and synthesized with MIDI files. 41 queries ranging from 30 seconds to a 90 seconds in length were played on a piano and recorded with an iPhone. An additional 33 audio samples were synthesized from MIDI files; the length of these performances ranged from 240 to 480 seconds. This section presents the techniques used for generating score information from the manually performed and MIDI synthesized audio samples.

For the manually performed samples, score information was transcribed by hand. This score information was stored as arrays containing sequences of notes played in each recording without any timing information (e.g [A,C,D...A]). For use in the main sequence alignment algorithm, these scores were converted to Chroma feature matrices. Chroma features are a collapsed form of frequency discretization. They are calculated from audio by collapsing the Short-time Fourier Transform into shape $12 * n$, where the columns represent the Chroma features ordered in time. The manually transcribed scores were converted into a Chroma matrix by substituting each note with a corresponding binary Chroma feature template as shown in Figure 2.

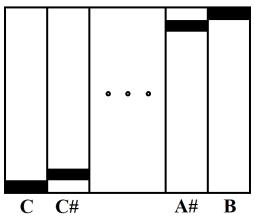

Fig. 2. Chroma Feature Matrix with corresponding notes

To validate this approach, a comparison of binary Chroma feature matrices and Chroma features of the corresponding audio recording are shown in Figure 3. While highly simplified, the binary matrix does capture the general shape of the Chroma feature matrix.

[1]Engineering Major at Harvey Mudd College
[2]Engineering Major at Harvey Mudd College

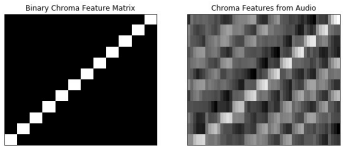

Fig. 3. Comparison of binary Chroma feature matrix with Chroma feature matrix generated from corresponding audio file

Through initial accuracy testing, the algorithm was very successful with the provided audio. However, the longest manually recorded audio clip was less than two minutes. To ensure the algorithm could handle a larger variety of environments, a longer data set was needed. To implement this, longer MIDI files of various classical songs were found online [4]. To parse these MIDI files, they were first converted to the csv format using an online tool [5]. In Jupyter notebook, python was used to parse these csv files for pertinent information. The timestamp, tempo, and note information was extracted from each file and stored as a vector of Chroma features.

For monophonic MIDI files, this process was fairly simple. Each note instance was represented by a binary template Chroma vector, emulating the process used for hand transcribed scores. When note ends weren't explicitly stated in the csv, the assumption was made that each note event lasted until the next note onset.

Polyphonic MIDI files required a more rigorous process to parse. Through many rounds of iteration, the following method emerged as the most effective. Within each file, individual tracks were treated the same as a monophonic MIDI file. This meant a timing vector and Chroma vector was produced for each track. These features were subsequently aligned according to the order of their timing vectors and summed. This is illustrated in Figure 4. The

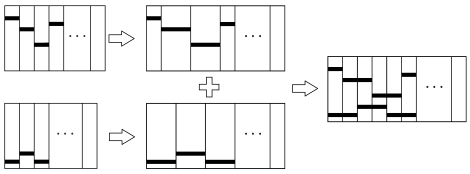

Fig. 4. Polyphonic Parsing

final matrix had to adjusted to prevent any complications with cost matrix calculation. First, any columns with zero magnitude removed to prevent undesired regions of zero cost. Additionally, all columns were normalized so that they summed to one. Both of these steps are necessary since the cost matrix is calculated using a dot product and all points in the cost matrix should be found from vectors with the same magnitude to ensure equal weighting.

IV. Algorithm Development

The basic algorithm consists of three steps: query Chroma feature extraction, Cost matrix calculation, and Dynamic Time Warping. This section will first explore the basic methodology of each component before discussing efficient real-time implementation techniques and modifications made to the system to improve robustness.

The query Chroma feature matrix represents the frequency content in condensed matrix form. Initially, a spectrogram of the query is calculated using the Short-time Fourier Transform. Next, this spectrogram is converted to a log frequency spectrogram, octaves are collapsed, and columns are normalized to create a Chroma feature matrix. This matrix has dimension $12 * n$ where rows represent the 12 note types and columns represent individual frames.

The cost matrix is computed based on the cosine similarity of the score Chroma matrix, explained in section III, and the query Chroma matrix. This is done by taking the dot product of the transposed query Chroma matrix and the score Chroma matrix. Each element of the resulting matrix represents a similarity metric, and since the Chroma matrices were normalized, all elements have magnitude between 0 and 1. Taking this into account, a cost matrix is calculated by subtracting the similarity matrix from a matrix of ones so that cost and similarity are inversely related. An example of a cost matrix is shown in Figure 5.

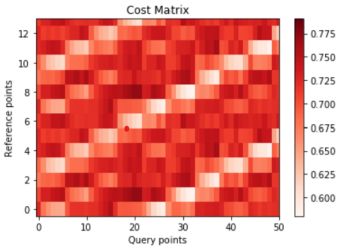

Fig. 5. Cost matrix for an A major scale

Dynamic Time Warping (DTW) uses the cost matrix to find the most closely matching sequences in the query and score. The objective is to align the query audio with the score, and find the path traversing the entire query audio that has the lowest cost. In doing so, DTW aligns the closest matching sequence of the score with the query. This process can be broken down into a few steps.

First, a set of steps and their corresponding weights are set. These steps set the allowable transitions within the matrix, and the weights represent multiplicative penalties for each transition. In this system, only two steps were used. The first step transitioned diagonally, in both the score and query direction, while the second step only progressed in the query direction. The second step was weighted with a slightly higher penalty so that the path would still progress in the score direction with repeated notes in the score. This is shown in Equation 1.

$$steps, weights = [[1,1],[1,0]],[1,1.1] \tag{1}$$

Despite not having a step to explicitly skip notes in the score, the system was still able to account for skipped notes since frames far exceeded reference points. As a result,

repeated diagonal steps could be taken to catch up to the correct score position when a note was skipped.

Second, the cumulative cost matrix (D) is calculated. The cumulative cost matrix is dynamically calculated so that each element in the matrix has value equal to the cost of the least expensive path that can be taken to reach that point. Concurrently with the calculation of the cumulative cost matrix, the last step taken to reach each element is stored in a back trace matrix (B). This makes it trivial to find the lowest cost path to any point in the cost matrix.

Third, the final point of the lowest cost path in the DTW process is taken as the output. This is done because the final point represents the score position at the end of the query.

The cumulative cost matrix and optimal path for the cost matrix from Figure 5 is shown in Figure 6.

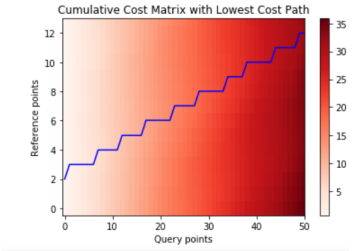

Fig. 6. Cumulative cost matrix and Optimal path for an A major scale

A naive approach to the score following problem is to perform DTW on the entire sequence at each time step. However, this process quickly becomes too inefficient to run in real time for long sequences. To adapt this process for use in real-time, two methods were explored: windowing the query and iteratively building the DTW matrices.

Windowing the search region caps the computational cost of the DTW process by limiting the size of the query. For each iteration of the real-time simulation, only the previous N frames of the query audio are used in the DTW process. By keeping the query size consistent, the run-time of the DTW process no longer grows for long queries. Examples of path generation with this method are shown in Figure 7.

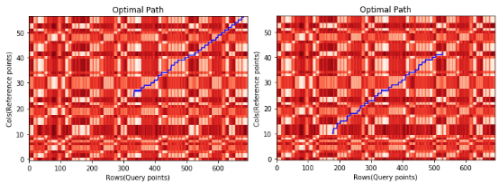

Fig. 7. Two paths generated with a set window of 350 frames

This method has one clear downside: it discards cost matrix data. This makes the windowing approach prone to repeating sections of the score as early query data is discarded.

To address this issue, another approach was used where the DTW matrices are iteratively constructed. This approach is rooted in the realization that only the previous column of the cumulative cost matrix and the current column of the cost matrix are needed to expand the cumulative cost matrix. Instead of recalculating the entire cumulative cost matrix at each time step, the matrices from the previous iteration are stored and expanded at each iteration. This is illustrated in Figure 8.

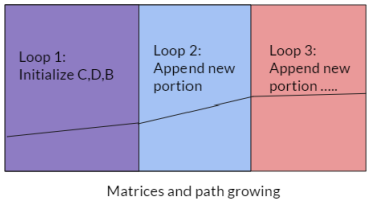

Fig. 8. Real time matrix construction

This has the benefits of fast runtime along with retaining all data during the real-time process. For these reasons, this approach was selected. In initial runs of the system, the system worked as intended, but often took dozens of iterations of the real time simulation to properly converge.

Two main modifications to the system were explored in order to improve convergence speed. First, harmonic summation was introduced to the Chroma feature extraction process. This was intended to help distinguish the fundamental pitch and prevent accidental detection of a harmonic.

Second, a version of the system was produced with a biased starting point. This was motivated by the possible usage scenario of an application that scrolls a music score during a performance. In this scenario, the starting point is known. To model this, an operating mode of the system was introduced where the cost matrix was initialized with smaller magnitudes around the predicted starting point. This was intended to encourage the score following algorithm converge more quickly.

V. Evaluation of Accuracy

To evaluate accuracy, a time location estimate given by the algorithm is compared to the actual time location in a real-time simulation. The error between the two is representative of the accuracy of the system.

Two accuracy benchmarks can be taken from the system. The accuracy of the DTW can be calculated as well as the accuracy of the endpoint.

The accuracy in the DTW gives insight into the quality of the data preparation. This accuracy benchmark shows how well the DTW can use the score information to find a match with the query information. This statistic is found by comparing the timing of each note transition in the DTW path with note onsets of the score. Here, the accuracy comes in the form of a time error.

The endpoint accuracy can be calculated progressively in the real time simulation. Within the real time simulation, the time estimates are tracked and recorded. Post simulation, the saved time estimates are compared with real time

progression, which is completely linear. This metric can be expected to have a worse accuracy when compared to that of the DTW accuracy. This is due to the algorithm's lack of sensitivity to note length. The algorithm isn't able to estimate the duration of single notes, so the location estimate only advances when the frequency content of the audio changes. Again, the accuracy metric comes in the form of a time error.

VI. Results

The results of the accuracy metrics described in section five are shown in tables I, II, and III. Table I shows the DTW accuracy metric for both the original and modified systems for a number of violin tracks. Table II shows the mean error after convergence and the number of iterations before convergence with the original system. Table III shows similar information for the modified system.

TABLE I

DTW ACCURACY METRIC

	Mean DTW Error (sec)	
FileNumber	Original System	Modified System
42	0.9980	
43	0.0013	
44	0.0030	
45	0.0084	0.0020
46	0.0037	
47	0.2771	
48	0.0145	
50		0.6348

TABLE II

ORIGINAL SYSTEM

File Number	Mean Error (sec)	Standard Deviation	Iterations
42	0.865	0.2944	38
43	0.570	0.3434	45
44	0.629	0.2322	54
45	0.00837	0.3735	1
46	1.273	0.1969	55
47	0.6547	0.2569	2
48	0.4311	0.4698	25

TABLE III

MODIFIED SYSTEM

File Number	Mean Error (sec)	Standard Deviation	Iterations
42	0.871	0.2874	4
43	0.577	0.2461	5
44	0.634	0.2433	6
45	0.0099	0.3921	6
46	1.281	0.4152	2
47	0.691	0.3564	7
48	0.486	0.2815	4

As expected, the accuracy of the endpoints is worse than the accuracy of the DTW path. We can see the lack of the the algorithm's ability to track progression within a note in the saw-tooth behavior of the location estimate error in Figure 9.

Run time statistics were measured for 8000 iterations of the score following system. No difference was found for the run time of the original and modified systems. The system had a mean run time of 0.0693 seconds with a standard deviation of 0.0345 seconds, as shown in Figure 11.

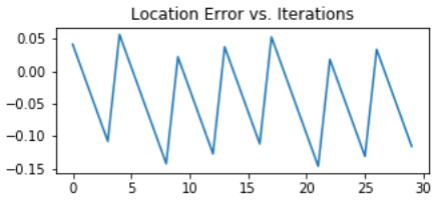

Fig. 9. Location Error vs. Iterations showing saw-tooth pattern

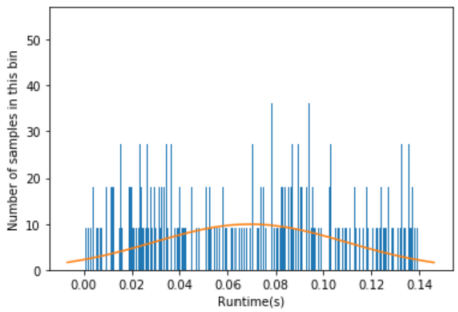

Fig. 10. Runtime over 8000 iterations

VII. Discussion

In running the original and modified systems, we found that the modifications had the intended effect. Incorporating harmonic summation and lowering the cost around the starting note made the algorithm converge to the true score location much more quickly, without any noticeable adverse effect in terms of run time or either of our error metrics. In the future, the weight of the starting note can be further tuned to further speed up convergence to the true score location.

Initially, run time was a large issue due to the use of dynamic time warping. However, run time became a non-issue due to shortcuts implemented in the code, the most significant being only running DTW on the new information. Because of this, more extensive analysis of the audio can be explored.

Since there is time headroom for the algorithm, it might be possible to implement instantaneous frequency estimation to address the low frequency resolution of the system. Since the frequency resolution of the STFT is linear and MIDI notes are spread on a log scale, lower notes are very difficult to distinguish. Instantaneous frequency estimation would allow this algorithm to more accurately distinguish lower frequency notes. This has been mostly implemented in code, but there are not available results for this modification at the time of this paper.

Additionally, adding a Hidden Markov Model to assist in location tracking would be useful due to the repetitive nature of music[1][2][3]. This type of probabilistic model could be readily incorporated by expanding the scope of the DTW to account for transition probabilities. Along similar lines, the system could be improved by accounting for known timing

information. Our system completely disregards relative note lengths. However, analyzing relative note lengths can provide information about windows where frequency transitions are likely. Implementing these probabilistic models represents an outlet for future work.

ACKNOWLEDGMENT

We would like to thank professor Tsai for his guidance in making this project possible.

REFERENCES

[1] Nicola Orio and F. Déchelle. Score Following Using Spectral Analysis and Hidden Markov Models. In Proceedings of the ICMC, Havana, Cuba, 2001.
[2] B. Baird, D. Blevins, and N. Zahler. Artificial Intelligence and Music: Implementing an Interactive Computer Performer. Computer Music Journal, 17(2):73–79, 1993.
[3] Christopher Raphael, "A Hybrid Graphical Model for Aligning Polyphonic Audio with Musical Scores," ISMIR, 2004.
[4] D. Grossman, "Sonatas and Partitas for Solo Violin," 1997.
[5] J. Walker, "MIDI CSV," 2004.
[6] Cuvillier, Philippe. "Score Following." Score-Following [Music Representations Team], 13 Oct. 2014, repmus.ircam.fr/score-following.