Score-Performance Matching using HMMs

Pedro Cano, Alex Loscos, Jordi Bonada
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
{ pcano, aloscos, jboni }@iua.upf.es http://www.iua.upf.es

[Published in the Proceedings of the ICMC99]

Abstract

In this paper we will describe an implementation of a score-performance matching, capable of score following, based on a stochastic approach using Hidden Markov Models.

1. Introduction

Linking notes in a musical performance to the corresponding notes in a score is called score-performance matching. Proper matching algorithms are crucial for automatic accompaniment systems or in systems where the computer has to find out where the musician is with respect to a known score, in order to make an appropriate musical response. For example audio effects can be applied to the performer's sound depending on his/her location in the score. In the context of musical performance research, matching algorithms are valuable to measure differences between the performance and the score, thus being able to extract expressive timing patterns, calculate tempo, pitch deviation patterns, or any other performance characteristics.

One category of algorithms focuses on real-time matching and they are often called score-following. Early work in score matching was performed by Dannenberg [1] and Vercoe [2], both primarily interested in making real-time matchers. Dannenberg describes an algorithm solely based on pitch information, intended to robustly follow a monophonic instrument. Later these algorithms were extended to deal with polyphonic music and multiple instruments [3]. Most algorithms primarily match pitch, possibly in combination with time information. The method presented here focuses on monophonic music and can be seen as a continuation of Puckette's work [4] moving from knowledge-based to stochastic models.

2. Stochastic modeling

The input features needed to make a decision when performing a match, namely fundamental frequency, notes duration, etc, are inherently uncertain. The uncertainty rises from the fact that instrument players or singers do not perform an ideal realization of what it is written in the score and the fundamental frequency algorithms are not absolutely reliable. Others algorithms can perform reasonably well with specific instruments like flute but they fail when the problem above stated is especially troublesome. This occurs very clearly with the singing voice in which the output does not resemble at all a sequence of discrete tempered pitches attained at well-defined times.

Stochastic modeling is a flexible general method for situations like the above described. It consists on employing a specific probabilistic model for the uncertainty or incompleteness of the information. A music performance is a nonstationary process since its statistical parameters vary over time. Hidden Markov Models (HMM) are well studied and used for statistical modeling of nonstationary stochastic processes such as speech and our work has been to apply them to our application.

3. Hidden Markov Models

A HMM is most easily understood as a generator of vector sequences. It is a finite state machine which changes state once every time unit and each time t that a state j is entered, a n acoustic speech vector y_t is generated with probability density $b_j(y_t)$. Furthermore, the transition from state i to state j is also probabilistic and governed by the discrete probability a_{ij}. In the figure below we show an example of this process where the model moves through the state sequence X=1,1,2,2,2,2,2,3,3,3 in order to generate the 10 observation vectors of *k-index model*.

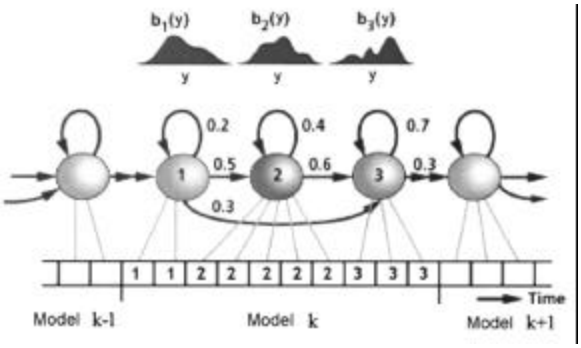

Figure 1: Markov Process example

The joint probability of a vector sequence *Y* and state sequence *X* given some model M is calculated simply as the product of the transition probabilities and the output probabilities. The joint probability of an acoustic vector sequence *Y* and some state sequence *X = x(1), x(2), x(3),…,x(T)* is:

$$P(Y, X|M) = a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(y_t) a_{x(t)x(t+1)}$$

In practice, of course, only the observation sequence *Y* is known and the underlying state sequence *X* is hidden. This is why it is called *Hidden Markov Model*.

In our problem, the model M is the sequence of notes specified in the score script, Y the feature sequence extracted from the audio signal and X, the state sequence, which is actually what we aim for in this paper.

4. Note models based in HMM

In this section we present the features and the characteristics of the HMMs used to model the notes.

4.1 Front-End Parameterization

The function of front-end parameterization stage is to divide the input signal into blocks and to extract from each block relevant features. The six features that will be used for the observation sequence are:

Energy,

$$Energy = 20 \log_{10}\left(\sum_{k=0}^{N-1} |X(k)|\right)$$

Delta Energy,

$$\Delta Energy(n) = Energy(n+1) - Energy(n-1)$$

Zero Crossing,

$$Z_s(n) = \frac{1}{N} \sum_{m=n-M+1}^{n} \frac{|\mathrm{sgn}\{x(m)\} - \mathrm{sgn}\{x(m-1)\}|}{2} w(n-m)$$

Delta Fundamental Frequency,

$$\Delta F_0(n) = \begin{cases} \log\left(\dfrac{F_0(n+1)}{F_0(n-1)}\right) & \text{if } F_0(n-1)\cdot F_0(n+1) \neq 0 \\ g & \text{elsewhere} \end{cases}$$

Fundamental Frequency, and *Fundamental Frequency Error*, which is a measure of "goodness" of the F_0 estimated.

Obtaining fundamental frequencies is a popular subject of study. The particular algorithm we use is an adaptation of the Two-Way Mismatch [5]. In the procedure, the estimated F_0 is chosen as to minimize discrepancies between measured partial frequencies and harmonic frequencies generated by trial values of F_0. For each trial F_0 mismatches between the harmonics generated and the measured partial frequencies are averaged over a fixed subset of the available partials. The predicted to measured error is defined as:

$$Err_{p\to m} = \sum_{n=1}^{N} E_w(\Delta f_n, f_n, a_n, A_{max})$$

$$= \sum_{n=1}^{N} \Delta f_n \cdot (f_n)^{-p} + \left(\frac{a_n}{A_{max}}\right) \times \left[q\Delta f_n \cdot (f_n)^{-p} - r\right]$$

where Δf_n is the difference between a predicted and its closest measured peak, f_n and are the frequency and magnitude of the predicted peaks, and A_{max} is maximum peak magnitude. The measured to predicted error is defined as:

$$Err_{m\to p} = \sum_{k=1}^{K} E_w(\Delta f_k, f_k, a_k, A_{max})$$

$$= \sum_{k=1}^{K} \Delta f_k \cdot (f_k)^{-p} + \left(\frac{a_k}{A_{max}}\right) \times \left[q\Delta f_k \cdot (f_k)^{-p} - r\right]$$

where Δf_k is the difference between a measured and its closest predicted peak, f_k and a_k are the frequency and magnitude of the measured peaks, and A_{max} is maximum peak magnitude. The total error is:

$$Err_{total} = Err_{p\to m} / N + r\, Err_{m\to p} / K$$

This pitch estimation method gives a temporal evolution of the F_0. This envelope is then used to calculate some of the system's input features.

4.2 Note Model Architecture

We have three left-to-right HMM models: a note, a no-note and a silence model. We model notes with 3 states so as to mimic the note behavior of attack, steady state and release. The silence is modeled with only 1 state, since there is no temporal structure to exploit. The no-note, modeled with 3 states, aims to account for all the unpitched sounds that appear in the performance. This can include noisy spurious

sounds, or in the case of singing voice aspirations, fricatives, plosives…

The choice of output probability function is crucial since it must model the intrinsic variability of the score realizations. Some HMM systems use discrete output probably functions in conjunction with a vector quantizer. Each incoming feature vector is replaced by the index of the closest vector in a precomputed codebook and the output probability functions are just look-up tables containing the probabilities of each possible VQ index. This approach is computationally very efficient but the quantization introduces noises, which limit the precision that can be obtained. Hence, it seems a better choice to use parametric continuous density output distribution, which model the feature vectors directly, for instance with multivariate mixture Gaussian.

$$b_j(y_t) = \sum_{m=1}^{M} c_{jm} N(y_t; m_{jm}, \Sigma_{jm})$$

where c_{jm} is the weight of mixture component m in state j and $N(j;m,S)$ denotes a multivariate Gaussian of mean m and covariance S.

This method results in a system with too many parameters to train. Thus, in this work, we use the same large Gaussian codebook for all the states and only estimate different mixture weights for each state.

We build a codebook with vectors that are composed of all the above features but the F_0, which will be treated differently. To construct a codebook l, the N_l corresponding observations, $y_t \sim l$ are clustered into M_l subsets, being M_l the number of mixture components of the codebook. To do this clustering we use LBG algorithm. Then

$$m_{lm} = \frac{1}{N_l} \sum_{\substack{m=1 \\ y_t \sim l}}^{M} y_t$$

$$\Sigma_{lm} = \frac{1}{N_l} \sum_{\substack{m=1 \\ y_t \sim l}}^{M} [y_t - m_{lm}][y_t - m_{lm}]^T$$

The mixture coefficient c_{jm}, for the case N_{jm} vectors are assigned to the mth mixture of the codebook, results in

$$c_{lm} = \frac{N_{lm}}{N_l}$$

The output probability of the observation F_0 is defined with two discrete symbols, one symbol when F_0 is not defined and another one when F_0 has been found. The discrete symbol in the states of note models is decomposed in a continuous density function whenever F_0 has been found as shown in figure 2.

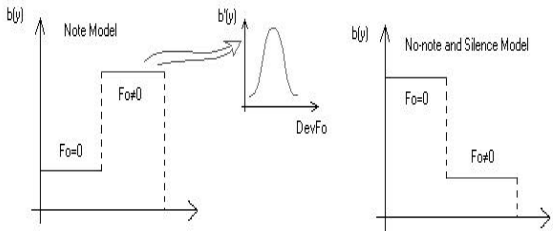

Figure 2: F_0 observation probability function

The continuous density function is modeled as a mixture of gaussians whose input is:

$$F_0 Dev = \left| \log(F_{Perfect\ 0}) - \log(F_0) \right|$$

The total output probability is the product of the five-feature vector's and the fundamental frequency's.

5. Model Training

For the training of models we need labeled training set of musical phrases, where each sentence consists of the music waveform and its transcription into notes. According to the transcription we concatenate the HMMs of the musical units to build an extended finite-state network (FSN). It is important to realize that, even though, apart from silences and unvoiced sounds, most models are notes, each HMM note model differs from each other because they keep information of the fundamental frequency associated and also its duration. This information is needed to calculate some input features.

Once a composite sentence FSN is created for each sentence in the training set, the training problem becomes one of estimating the unit model parameters that maximize the likelihood of the models for all the given training data. The maximum likelihood parameters can be solved for using either the forward-backward procedure or the segmental k-means training algorithm. For our system we have chosen the Segmental K-Means [6] and we have implemented it as follows:

1. Initialization: Linearly segment each training utterance into units and HMM states.

2. Estimation: The transition probabilities are estimated by merely counting the number of times the transition is used and dividing it by the number of times the source state for the transition is used. This requires maintaining counters to track each transition and each output symbol during training**.**

The mixture weights for the five-feature vector probability function are estimated for each state *i* as:

$$c_{im} = \frac{N_{im}}{N_i}$$

For the F_0, the output probability function is estimated:

$$\hat{b}_j = \frac{no.\text{ of times in } s_j \text{ and observed symbol } v_k}{no.\text{ of times in } s_j}$$

For the discrete symbol that expands to a continuous density function, we estimate this function the same way as the five-feature one but now it is only the states belonging to note models that share the codebook. This codebook has to be re-estimated every iteration.

3. Segmentation: The updated set of unit models (based on the estimation of step 2) is used to re-segment each training utterance into units and states (via Viterbi decoding).

4. Iteration: Steps 2 and 3 are iterated until convergence.

Since we wanted the system to be accurate in the definition of borders, we manually supervised the resulting segmentation and adjusted some note borders. By doing so, we got a growing parallel database where the notes are labeled and segmented with fixed borders. These segmented sentences are used for the training then in a different way than the not segmented ones. Instead of linearly segmenting each training utterance into units and HMM states, for the note-segmented utterances, we only linearly segment the HMM states inside the unit borders. When we iterate, only these inside HMM states borders are allowed to reallocate, the note borders are left fixed. Doing this resulted in a more accurate alignment.

6. Viterbi decoding

For alignment the trained models are concatenated and run against the feature vectors extracted from the sound using Viterbi decoding. As a result, the most probable path through the models is found, giving the points in time for every transition from one model to the following.

The Viterbi algorithm is an efficient algorithm to find the state sequence that most likely produced the observations. Let $f_j(t)$ represent the maximum likelihood of observing speech vectors y_1 to y_t and being in state j at time t. This partial likelihood can be computed using the following recursion

$$f_j(t) = \max_i \{ f_j(t-1) \cdot a_{ij} \} b_j(y_t)$$

where

$$f_1(1) = 1$$
$$f_j(1) = a_{1j} b_j(y_1)$$

for $1<j<N$. The maximum likelihood P'(Y|M) is then given by

$$f_N(T) = \max_i \{ f_j(T) a_{iN} \}$$

By keeping track of the state j giving the maximum value in the above recursion formula, it is possible, at the end of the input sequence, to retrieve the states visited by the best path, thus obtaining the time-alignment of input frames with models states.

It is possible to modify the algorithm to work on-line. To do so, the backtracking is adapted to determine the best path at each frame iteration instead of waiting until the end of the utterance. This adjustment implies a lost of robustness, some hints to overcome it can be found in [7].

The implementation for explicit note duration modeling by modifying the Viterbi algorithm [6] allows us to include the note duration information. The proposed modified Viterbi algorithm keeps track of the duration $D(t)$ of each note n at time t and introduces a duration penalty P of making a transition from state i at time t to state j at time $t+1$ given by,

$$P = \begin{cases} 0 & \text{if } \Delta D(t) < 0 \text{ and } i = j \\ l(\Delta D(t+1)) - l(\Delta D(t)) & \text{if } \Delta D(t) \geq 0 \text{ and } i = j \\ l(\Delta D(t)) & \text{if } \Delta D(t) < 0 \text{ and } i \neq j \\ l(0) & \text{if } \Delta D(t) \geq 0 \text{ and } i \neq j \end{cases}$$

where $\Delta D(t) = D(t) - D_n$ is the difference between the duration of the model and the note duration, D_n, specified in the score, and $l(u)=\log p(u)$, $p(.)$ is the probability density of ΔD and it is modeled by mixture gaussian densities.

7. Conclusions

The instrument we have mainly worked with is the singing voice. This choice is due not only to the fact that, from all instruments, the larger training database available for us is the singing voice one, but also because we believe this instrument is the most critic, and once solved the score-matching problem for the singing voice case, we will have solved it for any other harmonic instrument. There is still experimentation to be done to tune the different parameters but results are promising.

Improvements can be achieved by considering context. We can train different note models

depending on the notes that precede and follow them. In the case of the singing voice, using the lyrics [7] can add robustness to the alignment. This kind of information can also be used to improve accuracy.

References

[1] R. Dannenberg, "An On-line Algorithm for Real-Time Accompaniment". *Proceedings of the ICMC 1984*.

[2] B. Vercoe. "The Synthetic Performer in the Context of Live Musical Performance", *Proceedings of the ICMC 1984*.

[3] P. Desain, H. Honing and H. Heijink. "Robust Score-Performance Matching: Taking Advantage of Structural Information". *Proceedings of the ICMC 1997*.

[4] M. Puckette. "Score following using the sung voice *Proceedings of the ICMC 1995*.

[5] P. Cano. "Fundamental Frequency Estimation in the SMS Analysis". *DAFX Proceedings 1998*.

[6] L. Rabiner and B.H. Juang *Fundamentals of Speech Recognition*. Prentice Hall, 1993

[7] A. Loscos, P. Cano, J. Bonada. "Low-Delay Singing Voice Alignment to text". *Proceedings of the ICMC 1999*.