Technical Appendix for "A Systematic Evaluation of Real-Time Audio Score Following for Piano Performance"

This document contains technical definitions for the score follower models described in Section 3 of the paper.

1 Definitions

The following definitions will be used throughout this document.

- **Observations** $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_N\}$: The input features computed from an on-line audio stream. Each observation $\mathbf{o}_i \in \mathbb{R}^F$ is a F-dimensional vector, with F depending on the type of features (e.g., chroma features has $F = 12$, etc.).
- **Reference features** $\mathbf{R} = \{\mathbf{r}_1 \ldots, \mathbf{r}_{N_{ref}}\}$: Audio features for the synthesized score. Scores were synthesized using Fluidsynth. Each $\mathbf{r}_i \in \mathbb{R}^{N_{ref}}$ is an a F-dimensional vector. We slightly abuse notation and use $\mathbf{R}$ as a matrix in $\mathbb{R}^{N_{ref} \times F}$where the i-th row corresponds to $\mathbf{r}_i^\top$.
- **Hidden states** (score positions): $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, with $\mathbf{z}_i \in \{0, 1\}^{N_{\text{ref}}}$ and $\|\mathbf{z}_i\|_1 = 1$. These variables are one-hot encodings representing the 'indices' of a frame in the reference features.

Throughout this document, we use A_{ij} (i.e., not boldfaced) to refer to the (i, j)-th element of matrix $\mathbf{A}$. Additionally, in the algorithms, we use $\mathbf{A}[i, j]$ to refer to the $(i - 1, j - 1)$-th element of the matrix, following Python-style indexing (i.e., starting from 0).

2 Online Time Warping Dixon

2.1 Algorithm Description

The `OnlineTimeWarpingDixon` algorithm extends classical Dynamic Time Warping (DTW) for real-time applications by limiting computations to a sliding window of recent frames. Rather than computing the full cost matrix $\mathbf{D} \in \mathbb{R}^{N_{ref} \times N}$ for the entire sequences, the algorithm updates only a localized region, thereby reducing the overall computational complexity.

The basic DTW recurrence is given by

$$D[i, j] = d(\mathbf{r}_i, \mathbf{o}_j) + \min \begin{cases} D_{i-1,j} \\ D_{i,j-1} \\ D_{i-1,j-1} \end{cases}$$

where $d(\mathbf{r}_i, \mathbf{o}_j)$ denotes the local distance between the i-th reference feature and the j-th input feature. In the online setting, however, only a subset of the accumulated costs is updated as each new input feature is received, and an "adaptive diagonal" is computed through the cost matrix. This adaptive diagonal—representing the most likely alignment based on available data—allows the algorithm to avoid the full matrix computation typical of DTW.

At each iteration, the algorithm performs the following steps:

- It decides whether to retrieve a new input feature (advancing the input index) or to move forward in the reference sequence (updating the current position), based on a selection function that evaluates the normalized costs along the current matrix edges.
- It updates the accumulated cost matrix and a corresponding path-length matrix within a fixed window, using local distance computations.
- It calculates an adaptive diagonal through the cost matrix, which is then used to select a candidate alignment point. This point is appended to the warping path, ensuring a smooth, incremental update in accordance with the online constraint.
- Latency and run count statistics are updated concurrently to guarantee real-time processing capability.

Algorithm 1 outlines the entire procedure.

Algorithm 1: `OnlineTimeWarpingDixon`

Input:

1. $\mathbf{o}_n \in \mathbb{R}^F$: input features
2. $\mathbf{R} \in \mathbb{R}^{N_{\text{ref}} \times F}$: reference features
3. $\mathbf{D} \in \mathbb{R}^{(N_{\text{ref}}+1)\times 2}$: accumulated cost matrix
4. $n \in [0, N-1]$: input index
5. current_position $\in [0, N_{\text{ref}} - 1]$
6. $s \in \mathbb{Z}_{>0}$: step size
7. $d(\cdot,\cdot) : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$
8. $w \in \mathbb{Z}_{>0}$: window size

Output: Updated current_position and cost matrix $\mathbf{D}$

// Initialization
Set `previous` ← None and `runCount` ← 0;
EvaluatePathCost(n, current_position);
// Main Online Loop
while *input remains and current_position* $< N_{ref}$ **do**
  `// Update input branch: advance input pointer`
  $t \leftarrow t + 1$ (**retrieve** $o[t]$);
  for $k =$ *current_position* $- w + 1$ **to** *current_position* **do**
    if $k \geq 0$ **then**
      EvaluatePathCost(t, k);
  `// Update reference branch: if cost structure suggests`
  if *boundary costs indicate advancing the reference* **then**
    $j \leftarrow j + 1$;
    for $k = t - w + 1$ **to** t **do**
      if $k \geq 0$ **then**
        EvaluatePathCost(k, j);
  $i_{\text{cand}} \leftarrow \arg\min\{\, \text{normalizedCost}(i) \,\}$; `// pick the index` i `in the current boundary window that yields the lowest normalized cost`
  if *direction* = *previous* **then**
    `runCount` ← `runCount` + 1
  else
    `runCount` ← 1
  current_position $\leftarrow \min\Big(\max(\text{current\_position}, i_{\text{cand}}), \text{current\_position} + s\Big)$;
return (current_position, $\mathbf{D}$)

3 Online Time Warping Arzt

The `OnlineTimeWarpingArzt` algorithm is a different variant of OLTW that uses a normalized version of the accumulated cost matrix (normalized by the length of the path). Algorithm 2 shows this procedure.

Algorithm 2: `OnlineTimeWarpingArzt`

Input:

1. $\mathbf{o}_n \in \mathbb{R}^F$: input features
2. $\mathbf{R} \in \mathbb{R}^{N_{\text{ref}} \times F}$: reference features
3. $\mathbf{D} \in \mathbb{R}^{(N_{\text{ref}}+1) \times 2}$: accumulated cost matrix
4. $n \in [0, N-1]$: input index
5. current_position $\in [0, N_{\text{ref}} - 1]$
6. $s \in \mathbb{Z}_{\geq 0}$: step size
7. $d(\cdot, \cdot) : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$
8. $w \in \mathbb{Z}_{\geq 0}$: window size

Output: Updated current_position and $\mathbf{D}$

`// Initialization for first step only`

$\mathbf{D}[i, j] \leftarrow \infty$ for all i, j

`// Initialization for every step`

min_costs $\leftarrow \infty$

$i_m \leftarrow \max(\text{current\_position} - s,\, 0)$; `// Minimum index to consider`

$w_s \leftarrow \max(\text{current\_position} - w, 0)$; `// Start of the window`

$w_e \leftarrow \min(\text{current\_position} + w, N_{\text{ref}})$; `// End of the window`

for $i \leftarrow 0$ **to** $w_e - w_s - 1$ **do**

  $\mathbf{c}[i] \leftarrow d\left(\mathbf{R}[w_s + i],\, \mathbf{o}_n\right)$

`// Main OLTW loop`

$i_s \leftarrow w_s$

$i_c \leftarrow 0$

if $i_s = 0$ ***and*** $n = 0$ **then**

  $\mathbf{D}[1, 1] \leftarrow \sum_i \mathbf{c}[i]$

  min_costs $\leftarrow \mathbf{D}[1, 1]$

  $i_m \leftarrow 0$

while $i_s < w_e$ **do**

  if *not* *($i_s = 0$* ***and*** *$n = 0$)* **then**

    $$\mathbf{D}[i_s + 1, 1] = \mathbf{c}[i_c] + \min \begin{cases} \mathbf{D}[i_s, 1] \\ \mathbf{D}[i_s + 1, 0] \\ \mathbf{D}[i_s, 0] \end{cases}$$

    $$\text{norm\_cost} \leftarrow \frac{\mathbf{D}[i_s + 1, 1]}{(n + i_s + 1)}$$

    if *norm_cost* $<$ *min_costs* **then**

      min_costs $\leftarrow$ norm_cost

      $i_m \leftarrow i_s$

  $i_c \leftarrow i_c + 1$

  $i_s \leftarrow i_s + 1$

`// Update Accumulated cost matrix`

for $i \leftarrow 0$ **to** $N_{ref} - 1$ **do**

  $\mathbf{D}[i, 0] \leftarrow \mathbf{D}[i, 1]$

  $\mathbf{D}[i, 1] \leftarrow \infty$

`// Update current position without going backwards and limiting step`

if $n = 0$ **then**

  current_position $\leftarrow \min\Big(\max(\text{current\_position},\, i_m),\, \text{current\_position}\Big)$

else

  current_position $\leftarrow \min\Big(\max(\text{current\_position},\, i_m),\, \text{current\_position} + s\Big)$

return current_position, $\mathbf{D}$

4 Hidden Markov Model

4.1 Model definition

Hidden Markov Models (HMMs) are a probabilistic state-space model. The joint probability distribution of the full sequence of observations and hidden states is given as

$$p(\mathbf{o}_1, \ldots, \mathbf{o}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) = p(\mathbf{z}_1) \left[\prod_{n=2}^{N} p(\mathbf{z}_n \mid \mathbf{z}_{n-1}) \right] \prod_{n=1}^{N} p(\mathbf{o}_n \mid \mathbf{z}_n) \tag{1}$$

Where

1. $p(\mathbf{z}_n \mid \mathbf{z}_{n-1})$ are the transition probabilities, and are commonly represented by a matrix

   $$A_{ij} = p(z_{n,i} = 1 \mid z_{n-1,j} = 1) \tag{2}$$

   where $z_{m,k}$ represent the k-th component of the m-th hidden state, with $0 \leq A_{ij} \leq 1$ and $\sum_j A_{ij} = 1$.

2. $p(\mathbf{o}_n \mid \mathbf{z}_n)$ are the observation probabilities. We model these probabilities using a parametric distribution in the form

   $$p(\mathbf{o}_n \mid \mathbf{z}_n) = \prod_{m=1}^{N_{ref}} p(\mathbf{o}_n \mid \phi_m)^{z_{n,m}} \tag{3}$$

   where ϕ_m are the parameters of the distribution and $z_{n,m}$ represents the m-th component of the n-th hidden state vector. We define an observation model as a function $\mathcal{O}(\cdot)$ in $\mathbb{R}^F \mapsto \mathbb{R}^{N_{ref}}$

   $$\mathcal{O}(\mathbf{o}_n \mid \Phi) = \begin{bmatrix} p(\mathbf{o}_n \mid \phi_1) \\ \vdots \\ p(\mathbf{o}_n \mid \phi_{N_{ref}}) \end{bmatrix} \tag{4}$$

   where $\Phi = \{\phi_1, \ldots, \phi_{N_{ref}}\}$

3. $p(\mathbf{z}_1)$ are the initial probabilities. This is usually parametrize by a vector $\mathbf{a}$ whose k-th component is $a_k = p(z_{1,l} = 1)$, and $\sum_k a_k = 1$

4.2 Transition Probabilities

We use a simple transition matrix, with two parameters, p_{stay} and p_{trans}, each representing the probability of staying in a state or transitioning to the next state, respectively. In this case, we restrict all transitions to be strictly forward.

$$A_{ij} = \begin{cases} p_{stay} & \text{if } i < N_{ref} \text{ and } i = j \\ p_{trans} & \text{if } i < N_{ref} \text{ and } j = i + 1 \\ \frac{1 - p_{trans} - p_{stay}}{N_{ref} - i - 1} & i < N_{ref} \text{ and if } j > i + 1 \\ 1 & \text{if } i = j = N_{ref} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

4.3 Observation Probabilities

We use an exponential distribution and the cosine distance between observation

$$p_i(\mathbf{o}_n \mid \phi_i) = \lambda \exp\left(-\lambda \left[1 - \frac{\mathbf{o}_n \cdot \mathbf{r}_i}{\|\mathbf{o}_n\| \, \|\mathbf{r}_i\| + \epsilon}\right]\right) \tag{6}$$

where $1 - \frac{\mathbf{o}_n \cdot \mathbf{r}_n}{\|\mathbf{o}_n\| \|\mathbf{r}_n\|}$ is the cosine distance between $\mathbf{o}_n$ and $\mathbf{r}_n$, and $\lambda > 0$ is the rate of the distribution. To avoid numerical issues due to small values of $\mathbf{o}_n$ or $\mathbf{r}_n$, we add a small epsilon.

The parameters of this distribution are $\phi_i = \{\lambda, \mathbf{r}_i\}$. We can then write the observation model as

$$\mathcal{O}(\mathbf{o}_n \mid \mathbf{R}, \lambda) = \begin{bmatrix} \lambda \exp\left(-\lambda \left[1 - \frac{\mathbf{o}_n \cdot \mathbf{r}_1}{\|\mathbf{o}_n\| \|\mathbf{r}_1\| + \epsilon}\right]\right) \\ \vdots \\ \lambda \exp\left(-\lambda \left[1 - \frac{\mathbf{o}_n \cdot \mathbf{r}_{N_{ref}}}{\|\mathbf{o}_n\| \left\|\mathbf{r}_{N_{ref}}\right\| + \epsilon}\right]\right)) \end{bmatrix} \tag{7}$$

4.4 Inference

For inference use the forward algorithm, described in Algorithm 3.

Algorithm 3: Forward Algorithm

Input:

1. $\mathbf{o}_n \in \mathbb{R}^F$: input features
2. $\mathcal{O}(\cdot \mid \mathbf{R}, \lambda)$: Observation model
3. $\mathbf{A}$: Transition Matrix
4. $\mathbf{a}$: Initial probabilities
5. $\boldsymbol{\alpha}$: Forward variable (optional)

Output: Updated current_position, and forward variable $\boldsymbol{\alpha}$

if $\boldsymbol{\alpha}$ *is None* **then**
  `transition_prob` $\leftarrow \mathbf{a}$;
else
  `transition_prob` $\leftarrow \mathbf{A}^\top \cdot \alpha$;
end
`observation_prob` $\leftarrow \mathcal{O}(\mathbf{o}_n \mid \mathbf{R}, \lambda)$;
$\boldsymbol{\alpha} \leftarrow$ `observation_prob` $\odot$ `transition_prob` ; `// ⊙ is the element-wise product`
$\boldsymbol{\alpha} \leftarrow \frac{\boldsymbol{\alpha}}{\max\left(\sum_i \boldsymbol{\alpha}[i], 10^{-6}\right)}$; `// Normalize forward variable`
current_position $\leftarrow \arg\max \boldsymbol{\alpha}$;
return *current_position,* $\boldsymbol{\alpha}$