Extracting Commands From Gestures: Gesture Spotting and Recognition for Real-time Music Performance

Jiuqiang Tang

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Music and Technology

Carnegie Mellon University

May 2013

Thesis Committee:

Roger B. Dannenberg, Chair

Richard M. Stern

Richard Randall

Abstract

During a music performance a violinist, trumpeter, conductor and many other musicians must actively use both hands. This makes it impossible to also interact with a computer by pressing buttons or moving faders. Musicians must often rely on offstage sound engineers and other staff acting on a predetermined schedule. The goal of this thesis is to create an interactive music system able to spot and recognize "command" gestures from musicians in real time. These gestures will trigger new sound events and control the output sounds. The system will allow the musician greater control over the sound heard by the audience and the flexibility to make changes during the performance itself. In this thesis, I will try to combine a gesture threshold model with a Dynamic Time Warping (DTW) algorithm for gesture spotting and classification. The following problems will be discussed:

- First of all, when we receive a multi-dimensional feature vector from a sensor, the first step we should do is to preprocess the data. Since we have multiple methods to smooth the data, I discuss the advantages and disadvantages of two smoothing methods.
- The Dynamic Time Warping algorithm is a powerful method for pattern matching. I also discuss how we can apply the DTW algorithm for both isolated gesture recognition and continuous gesture recognition.
- Additionally, we need to select the features and parameters we utilize in DTW-based recognition algorithms. Since the combinations of features and parameters have so many probabilities, I introduce a threshold and feature selection method based on F-measure evaluation to find a good combination according to training data the user has performed.
- Furthermore, how to build a good gesture vocabulary is also important. A good gesture vocabulary benefits both the users and the system performance. I will introduce several gesture vocabulary design guidelines from the HCI perspective.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Roger B. Dannenberg, for all the guidance and assistance to me during the past two years. Without his help, it would be impossible for me to finish this thesis and the music gesture recognition system. I would also like to thank Prof. Richard M. Stern and Prof. Bhiksha Raj for teaching me Digital Signal Processing, Machine Learning for Signal Processing and Automatic Speech Recognition. Those courses help me solve several important problems in this thesis and will still benefit my future study and career. Furthermore, I must thank my friend, Zeyu Jin, who has helped me a lot since the first day I came to Carnegie Mellon University. When I came to CMU two years ago, I almost knew nothing about programming, algorithms, computer systems and signal processing. At that time, I could never imagine that I would finish such gesture recognition system for music performance as my master thesis.

Finally, I would like to extend my appreciation to my parents, Dejiong Tang and Zhuli Zhen for raising me and support me in the past twenty-five years. I would like to dedicate this work to you. Thank you.

Contents

I. Introduction

In the field of computer music, real-time musical interaction has been a novel and attractive focus of research. It is related to all aspects of interactive processes including the capture and multimodal analysis of gestures and sounds created by artists, management of interaction, and techniques for real-time synthesis and sound processing. Moreover, with the development of gesture sensing technology, the measurement of gestures is becoming more and more accurate and can be utilized by music interactive systems.

In general, gestures of musicians can be interpreted as either discrete commands or as continuous control of a set of parameters. For example, a command gesture could represent a command to start a sound, enter a new state, or make a selection among several discrete choices. Alternatively, continuous control gestures may communicate sound parameters such as gain, pitch, velocity, and panning. Continuous parameter sensing is relatively easy since most physiological parameters captured by sensors are represented as a continuous stream. Converting this stream to a continuous parameter is often a simple matter of applying a mapping function to each sensor value to compute a control value. On the other hand, it can be very difficult to spot and recognize discrete commands within a continuous signal stream. A common solution is to use multiple sensors such as keys in a keyboard or to use multiple spatial positions as triggers to initiate commands. In this approach, the space of all sensor values is partitioned into different regions or "states" which are mapped to commands. Alternatively, statistical classifiers can recognize states. However, locating physical keys or learning to reproduce absolute positions in live performance can be difficult. Some "natural" gestures, such as nodding the head or pointing, offer an alternative way to communicate commands, but detection of these types of gestures is more difficult than recognizing static poses or positions because a sequence of sensor values must be considered.

Reliable physiological parameters can be acquired from sensors. How to process data and achieve a high recognition rate is much more difficult. In this thesis, I examine one approach to the recognition of discrete command gestures within a stream of continuous sensor data. The approach is based on the well-known sequence matching

algorithm dynamic time warping. While dynamic time warping provides a powerful mechanism for comparing two temporal sequences, it does not tell us what features will work best, how to pre-process the data to avoid outliers, how to spot a finite gesture in an infinite stream of input, and how to set thresholds to optimize recognition and classification performance. The goal of my study is to design and evaluate a gesture recognition strategy especially for music performance, based on the dynamic time warping algorithm and using an F-measure evaluation process to obtain the best feature and threshold combination. The proposed strategy will select features by searching over all feature combinations, obtain the optimal threshold for each gesture pattern of each feature combination in terms of the F-measure, and automatically generate a gesture recognizer.

The organization of the rest of the thesis is as follows: In Section II, related work will be briefly summarized. Section III will provide an overview of this real-time music gesture recognition system. Sections IV and V will describe technical details, including dynamic time warping and a threshold model of the music gesture recognition system. Section VI will address the definition of a music gesture vocabulary from an HCI perspective. Experiments and result analysis will be included in Section VII. Several possible applications of this music recognition system will be introduced in Section VIII. Finally, Section IX will conclude this thesis and discuss possible future work based on the results.

II. Related Work

Recently, a lot of research focuses on applying machine learning approaches to gesture-to-sound mapping for interactive music performance [1, 2, 3, 4]. Rebecca Fiebrink completed the Wekinator as her PhD thesis in 2011 [2]. Fiebrink's work explores how users can incorporate machine learning into real-time interactive systems, but the API/ Toolbox she provides uses a feature vector as input and creates a class as output. There is no provision for reducing a temporal sequence of sensor data to a feature vector or of reliably determining when a command has been recognized. In Francoise's work, he built two segmental models, which are based on Hierarchical Hidden Markov Models (HHMMs), to segment complex music gestures [1] in real time. Francoise applied two strategies, a forward algorithm and Fixed-Lag smoothing, to recognize violin-bowing techniques during a real performance. His research mainly focused on how different strategies affect segmentation accuracy and recognition latency but did not clearly point toward a good combination over parameters that can increase detection accuracy.

In the field of gesture recognition, major approaches for analyzing spatial and temporal gesture patterns include Dynamic Time Warping [5, 6, 7, 8], Neural Networks [9, 10, 11], Hidden Markov Models [12, 13] and Conditional Random Fields [14]. Many existing gesture spotting and recognition systems apply a threshold model for discriminating valid gesture patterns from non-gesture patterns [12, 15]. Hyeon-Kyu Lee and Jin H. Kim developed an HMM-based adaptive threshold model approach [12] in 1999. They took advantage of the internal segmentation property of the gesture HMMs to build an artificial HMM that acts as the threshold model. The threshold model calculates the likelihood of a threshold of an input pattern and provides a confirmation mechanism for the provisionally matched gesture patterns. However, this method runs off-line over a non-complex background and is not a suitable threshold model for real-time musical gesture recognition and identification. It still demonstrates to us that the calculated likelihood can be used as an adaptive threshold for selecting the proper gesture model. AdaBoost is another approach to obtain thresholds for classifying gesture and non-gesture patterns. Narayanan Krishnan et al. proposed an adaptive threshold model based on the individual AdaBoost classifiers by training classifiers on test

samples [15]. In their model, several weakest classifiers, which can satisfy a majority of samples of all the training classes, are used to test whether the input is too general. If the likelihood of this invalid gesture threshold model is higher than any value of the valid gesture models, the system considers the gesture to be too general and invalid. They built a discriminative gesture spotting network consisting of the individual gesture HMMs and a threshold model for recognizing activity gestures from a continuous data stream. In my work, since the training process is based on a small limited-sample gesture vocabulary, Adaboost may not have enough features to train and get good performance. Additionally, in order to deal with multiple deformations in training gesture data, Miguel A Bautista et al. proposed a probability-based DTW for gestures on RGB-D data [8], in which the pattern model is discerned from several samples of the same gesture pattern. They used different sequences to build a Gaussian-based probabilistic model of the gesture pattern. Although my recognition system is not based on a probabilistic model, it still suggests that for a gesture recognition system, the detection threshold it generates for each gesture should both be able to tolerate deformation in training/testing gesture data (high recall score) and classify different gesture patterns clearly (high precision score).

III. System Overview

An overview of the music recognition system will be introduced in this section. The system contains two main phases: training and testing. The training phase finds the best parameter and threshold combination based on the templates and training samples. The testing phase utilizes this combination to spot and recognize the gesture patterns in a parameter stream.

1. System Workflow

This thesis provides an implementation of a music gesture recognition system that uses training data to learn gestures. In the training process, by applying dynamic time warping and F-measure evaluation, the system finds the best combination of thresholds and feature parameters for gesture recognition. Results, generated by the training process, are utilized in a real time recognition process to find the start and end positions of a valid gesture. Figure 1 illustrates the general architecture of the methodology.

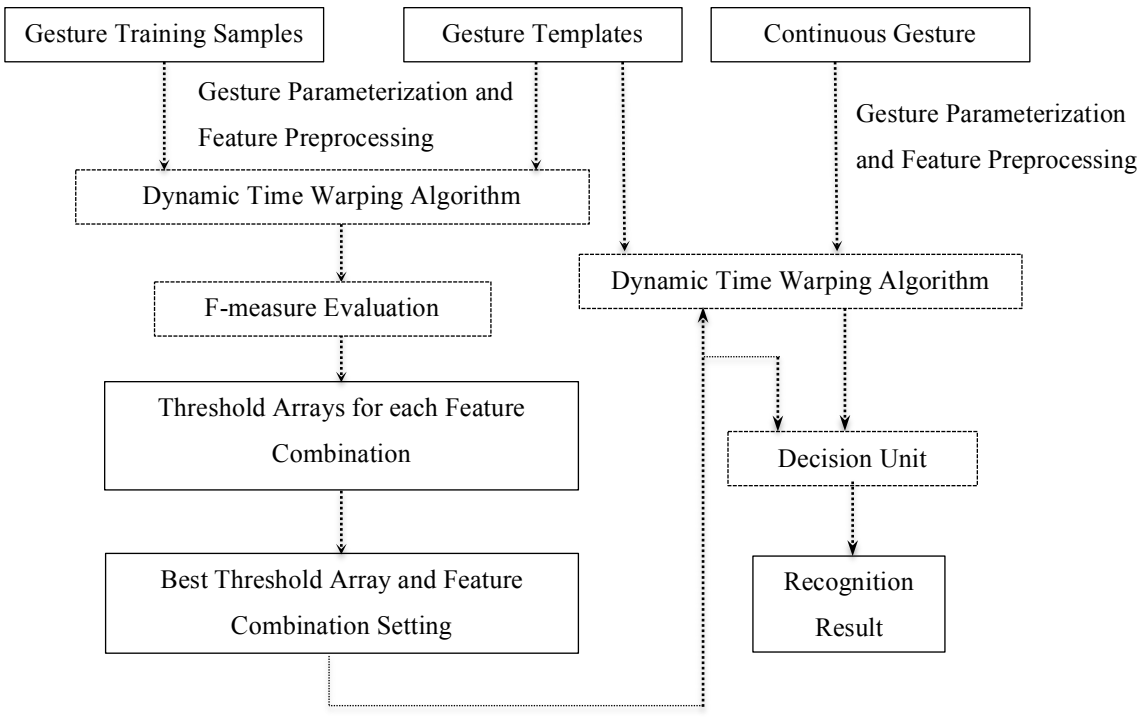

Figure 1: General architecture of training and recognition process.

2. Gesture Recognition

Gestures are recognized from multi-dimensional sensor data. For example, an accelerometer sensor might provide x, y, and z-axis acceleration data at a 100 Hz sample rate. The goal is to spot gestures within this stream of data and output a gesture identifier whenever a gesture is reliably detected. There should be no output if no gesture is spotted. Gestures are specified to the system using templates.

Before the training process, musicians define their own gesture vocabulary, such as nodding their heads, waving their hands and moving hands from left to right, and construct the mapping function between gesture patterns and target "commands." After defining a valid gesture vocabulary, the user records both gesture templates and test samples by using a motion capture sensor, such as Microsoft Kinect or an on-body acceleration sensor. Gesture templates and test samples are represented as multidimensional feature vectors and are stored in the system.

Typically, there will be multiple templates (examples) for each category of gesture. The system uses dynamic time warping (DTW) to compare input data to each template and compute a distance. The input stream is searched in overlapping windows because DTW compares finite sequences of data. When one template is deemed to be sufficiently close to the input data, the corresponding gesture is considered to be recognized. In practice, multiple gestures might be recognized simultaneously, so some further processing is needed to select only one gesture. The decision of whether a template is "sufficiently close" is based on thresholds, and the determination of thresholds is one of the main goals of the training process.

3. Features

Raw data from sensors is one possible set of features that could be used in the DTW recognition process. However, better recognition can often be obtained by deriving more abstract features from the raw data. For example, if the raw data includes absolute coordinates (such as x, y, z position information from a Kinect sensor), then a gesture that differs only in starting position from the template may be computed by DTW to be quite distant. Assuming that starting position is irrelevant, it is common to take

derivatives or to subtract the starting location to produce a sequence of "features" that are position independent.

Sometimes, adding additional features such as derivatives, integrals, smoothed values, normalized values, etc., can improve gesture recognition. Alternatively, adding irrelevant features can hurt performance. The current system does not generate features automatically, but given a set of candidate features, the training process can help to select the best features to use.

4. Training Process

The goal of the training process is to determine what features and what thresholds to use in order to get the best recognition performance. The details are covered in following sections. For each possible feature combination, the training system calculates and records the minimum distance between each pair of gesture templates and test data by running a dynamic time warping algorithm. Next, an F-measure is calculated to measure how well a threshold works for a particular gesture pattern and feature combination. The threshold with the highest F-measure score value is chosen as the threshold and stored into a threshold array.

To select features, the system evaluates different feature combinations and selects the combination that gives the best results on the training data. For each feature combination, the system sums up the F-measure for each best threshold in its threshold array. The feature combination with the highest cumulative F-measure score is used as a feature preprocessing setting in recognition process.

5. Recognition Process

In the recognition process, the threshold array and feature combinations generated by the training process are applied. When a new feature vector is produced by the capture system, the features are preprocessed and normalized according to the feature preprocessing setting. In real-time testing, if the minimum distance between a gesture pattern and a test sequence is less than or equal to the threshold value, the gesture is said to be recognized. The timestamp where the minimum distance occurred will be treated as the end point of this gesture. Accordingly, by applying backtracking method, the start point of gesture can be estimated. Using this information, an output message is

generated to report the type of gesture, the start time, and the end time. This message may then trigger some music events according to the detected gesture pattern.

The components in those two phases, such as the dynamic time warping algorithm and threshold selection mechanism will be introduced in detail in the following sections, both in terms of the theoretical foundation and the system implementation.

IV. Gesture Comparison with Dynamic Time Warping

The comparison between two gesture sequences is the foundation of a gestural command recognition system. In my implementation, the dynamic time warping algorithm is applied to compare the similarity of two time series. In this section, we will discuss several critical problems for applying the dynamic time warping algorithm to the gesture recognition problem. A description of DTW-based isolated and continuous gesture recognition processes will be presented in the second half of this section.

1. Feature Detection

Since gesture recognition is based on parameters, the measurement of gestures affects recognition accuracy directly. The feature detection stage is concerned with the detection of features, which are used for the estimation of gesture parameters. With the development of sensing technology over the last few years, both on-body and computer vision based gesture recognition systems have been employed for measuring body posture and motion. Both types of sensing have their own advantages and disadvantages. In my implementation, I provide an interface for receiving data from any input system via Open Sound Control [16] (OSC). The system can use any input device that reports one or more continuous (sampled) parameters.

1.1. Sensor System

1.1.1 Microsoft Kinect

Microsoft Kinect [17] is a motion sensing input device launched by Microsoft for the Xbox 360 game console in 2010. Kinect features an RGB camera and an IR depth sensor which provides full-body 3D motion capture and facial recognition capabilities. Figure 2 specifies sensing technology details of the Microsoft Kinect.

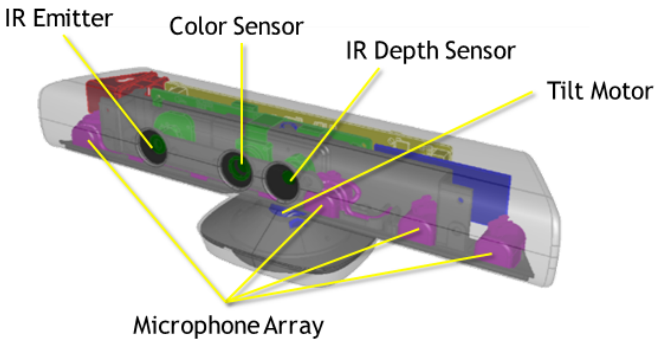

Figure 2: The Microsoft Kinect motion sensing input device.

Many developers are researching possible applications of Kinect that go beyond the system's intended purpose of playing games. For example, one program called OSCeleton [18] takes Kinect skeleton data from the OpenNI [19] framework and splits out the coordinates of skeleton's joints via OSC messages. By running OSCeleton, the gesture recognition system receives OSC messages from a defined port of the local host. Thus, skeleton data can be easily incorporated into system. Since the Kinect detects musicians' motion without putting any extra contact device on their hands or arms, it can capture perfomers' motion without causing any discomfort. However, Kinect still has four main problems. First, when analyzing gesture sequences captured by Kinect, the frame loss rate is very high among several particular gestures. In real time performance, high frame loss rates will cause unpredictable results. Secondly, since Kinect outputs video at a frame rate of 33 Hz, fast motions will probably cause frame loss or inaccurate measurement. Thirdly, because the tracking range of the Kinect is limited, the moving range of the performer is correspondingly limited. Finally, since the RGB camera requires good lighting conditions, a low light condition may result in a capturing failure. Figure 3 shows how different lighting conditions affect the measurement accuracy. Although the same Kinect device captured the same gestures performed by the same tester, the trajectory in the good lighting condition is much smoother than that in the bad lighting condition.

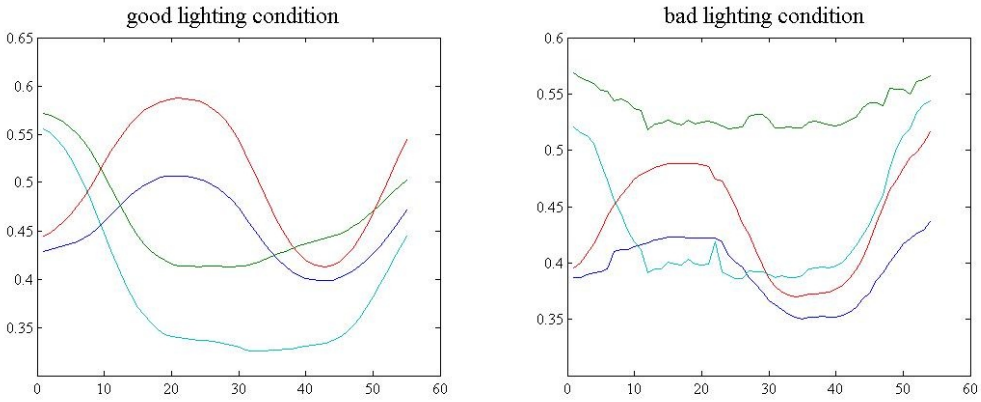

Figure 3: Trajectories of the same gesture measured in good and bad lighting conditions.

1.2. Feature Preprocessing

Since the data sequences captured by sensors are often loosely controlled, in order to eliminate out-of-range / missing data (e.g. the position of hand is Not a Number (NaN)), reduce unnecessary / redundant features (e.g. the position of knees is not useful for hand gesture recognition) and smooth each feature dimension, feature preprocessing is an essential step before applying gesture recognition.

1.2.1 Sample Inspection

In my implementation, when the system receives a multidimensional feature vector, it will first check the length of the incoming feature vector. If the sequence length is significantly shorter than the minimum sample length baseline, the system will treat the sequence as an invalid sample and throw it away. The minimum sample length baseline is dynamically set according to the sample recording duration and the default sampling rate of particular sensor. For example, for Kinect, since the default sampling rate is 33 frames per second, if the preset recording duration is 3 seconds, the baseline will be set to 90, which is $33 \times 3 \times 90\%$. Thus, any sample with the length less than 90 will be dropped directly. After checking sequence length, system will inspect each feature dimension separately, since out-of-range / missing data should be eliminated, for each feature dimension, if the ratio of out-of-range / missing data is higher than 10%, the feature dimension cannot be used in any training / recognition process. However, since other feature dimensions are still usable, the sample will be stored into system memory

but be labeled as a "partially usable" sample. In practice, this situation always occurs. For example, when tester stands too close to the Kinect sensor, it captures upper body skeleton motion but misses the lower body. But for hand gesture recognition, since only hand, elbow, shoulder and head position data will be utilized in training and recognition process, this "partial useable" sample is still good enough to use.

1.2.2 Missing Data Prediction

Although those feature vectors with high out-of-range / missing data ratio are banned from use for training and recognition, those feature vectors with low out-of-range / missing data ratio can still be utilized. The manner I applied to deal with out-of-range/missing data is to fill in the mean value among the nearest four valid data values.

For instance, assume a one-dimensional data sequence the system received is $S = \{0.1, 0.15, NaN, NaN, 0.4\ NaN, 0.26, NaN, 0.1, -0.1, \dots\}$. The third element will be the mean of $\{0.1, 0.15, 0.4, 0.26\}$, which is 0.2275. The fourth element will be the mean of $\{0.15, 0.2275, 0.4, 0.26\}$, which is 0.26. Accordingly, the sixth element is 0.255 and the eighth value is 0.13. This method biases the data and might create incorrect data. However, since not more than 10% of the data is missing, this is an effective and reasonable manner to use in practice.

1.2.3 Data Smoothing

For both the training and recognition process, data smoothing is applied before applying the matching algorithm. Since the data values in each feature dimension have redundancy (we assume that gestures are slow-moving compared to the sensor sample rate, or in signal processing terms, gestures are mainly low in frequency and thus oversampled), we can smooth the data points by the general information of each feature dimension to eliminate possibly inaccuracy measurement and reduce the impact of severe outliers.

For real-time feature processing, the data we can utilize are only the past data points. The system implements the common smoothing method known as the exponential filter, which forms a smoothed output sequence as a linear combination of the current sample s_i and the previous output value s'_{i-1}. The derived smooth sequence is given by

$$s_i' = \alpha\, s_i + (1-\alpha) s_{i-1}' \quad (1)$$

1.2.4 Normalization

To ignore the absolute position and size of gestures for the purposes of command recognition, it is common to normalize the data. In order to eliminate the impact of the starting position in a given dimension, we subtract the initial value in this dimension from each other corresponding data point. Thus, the first element of each feature dimension will be 0.0 and the rest of the sequence will be displacements relative to the initial value.

Features can also be normalized to obtain a mean of zero and a standard deviation of one. First, we compute the mean and standard deviation of unnormalized data:

$$mean: \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \quad (2)$$

$$standard\ deviation: \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad (3)$$

For each data value s_i in S, the system produces a new s_i' with zero mean and unit variance according to

$$s_i' = \frac{s_i - s_1 - \mu}{\sigma} \quad (4)$$

Although zero mean and unit variance are nice mathematical properties, a potential problem of this normalization is that it scales the gesture trajectory and the unpredictable scaling of tiny spontaneous motions by users might cause false triggering. In my implementation, mean and variance normalization are optional. They generally help with isolated gesture recognition but lead to false triggering in continuous gesture recognition. For the test results reported later, I used position independent features by subtracting the starting position, but the mean and variance normalization are disabled.

2. Dynamic Time Warping

2.1. Original Dynamic Time Warping

Since the lengths of the template feature vector and real input feature vector are varied, Dynamic Time Warping (DTW) is one of the best algorithms for defining and computing the distance between the pair of sequences. The original DTW algorithm was defined to measure the similarity between two time series $T = \{T_1, T_2, T_3 \dots T_m\}$ and $I = \{I_1, I_2, I_3 \dots I_n\}$. By computing distance $dist(i,j)$ between each vector T_i and I_j, a cost matrix which has a size of $m \times n$, will be filled to store each minimum distance from the beginning of two sequences to the current position of two sequences (T_i, I_j) . The value of any cell $C(i,j)$ in the cost matrix can be calculated by following the simple rule:

$$C(i,j) = dist(i,j) + \min \begin{cases} C(i-1,j) + insert\ cost \\ C(i-1,j-1) + subsititution\ cost \\ c(i,j-1) + deletion\ cost \end{cases} \quad (5)$$

After computing a full cost matrix, the DTW algorithm can do backtracking from the last cell of the cost matrix, $C[m][n]$, to find a least distance path from the end to the beginning. Due to the streaming nature of real-time gesture input, the gesture input data stream can have any arbitrary length and might contain several "command" gesture patterns. In order to find the end point of a candidate gesture, we will check each value in the last row of the cost matrix and compare it with the threshold we learned from training samples. Details will be presented in Section V.

2.2. Distance Function

Since parameters captured by sensors contain different degree features, such as the absolute x-, y- and z-axis position and the velocity among those axes, each node T_i or I_j in a time series is a multi dimensional feature vector. If we apply DTW algorithm on each dimension, the synchronized points in each dimension will not be in the same position or correlated. Thus, each feature vector should be considered as an "atom," and

entire atoms must align when we apply DTW. In my implementation, my default distance function is simply based on Euclidean distance, which is

$$dist(i,j) = \left(\sum_{d \in feature\ dimension} \left(T_{id} - I_{jd}\right)^2 \right)^{\frac{1}{2}} \quad (6)$$

In principle, different distance functions could be defined and learned, or different weights could be applied in the case of Euclidean distance. This would greatly impact the cost of training and perhaps introduce so many parameters that overfit to limited training data could be a problem. Therefore, we assume features are scaled to a reasonable range and rely on good feature selection alone to optimize the distance function.

3. Dynamic Time Warping for Gesture Comparison

In order to use dynamic time warping for measuring the similarity between gesture sequences, at least two valid processed data sequences should be prepared and passed to the DTW method. Those data sequences should use corresponding features. In my implementation, each gesture recorded by Kinect or on-body sensor is saved as an entry in a MAT file and the system stores each gesture as a two-dimensional array of numbers. Moreover, DTW should also be informed of the costs of insertion and deletion for initializing the edge costs.

Dynamic Time Warping is an algorithm with $\Theta(mn)$ complexity, where *m* and *n* are the lengths of two gesture sequences and the constant value depends on how many neighbors can have the access to a new node. For the gesture matching problem, I allow three possible edges (one horizontal, one vertical and one diagonal) to enter a new node. Thus, the constant value is 3. Moreover, the memory cost for one DTW comparison is mn. If we are not attempting to backtrack to the start pointer of a gesture, we can just maintain one column for tracking the previous values and update it in-place. Since gesture recognition should track the starting point of a gesture, a cost matrix C with the size of $m \times n$ should be kept to store minimum score for each pair (i, j) where $i \in [1, m], j \in [1, n]$. By applying the node distance function and following dynamic time

warping algorithm, each cell of the cost matrix will be filled with the minimum score from $C[1][1]$ to $C[m][n]$. A DTW class will create an object, PathInfo, for storing the cost matrix after finishing computation. PathInfo calls methods to find the minimum distance between the two gestures and backtrack the best path from the end to the beginning. PathInfo keeps the minimum distance, the best "warped" path and the average distance of the best path. Finally, DTW destroys the cost matrix to save memory.

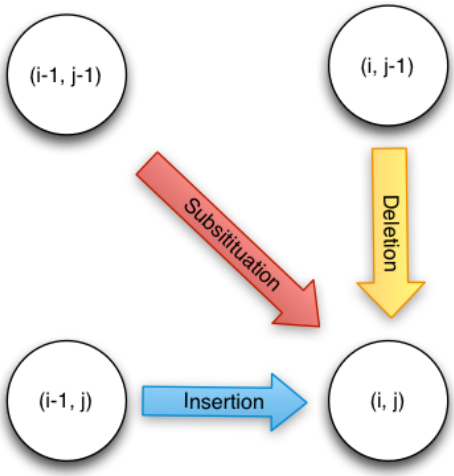

Figure 4: Three possible edges to enter a new node in a cost matrix.

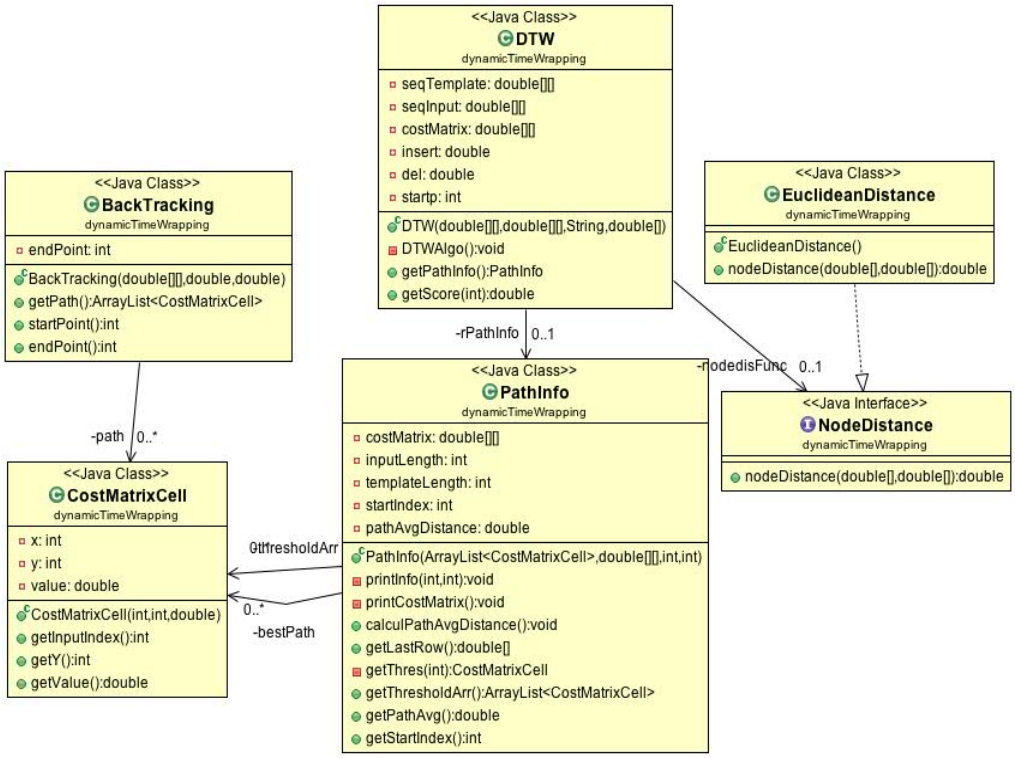

Figure 5: .The UML for java package dtw.

4. Isolated Gesture Recognition

Isolated segment gestures are finite-length sequences of sensor data that contain one gesture from the beginning to the end. For example, a conductor holding his left hand for two seconds and then rapidly moving the hand from bottom left to upper right could be treated as a gesture to trigger the next cue in a preset event list in a live performance. A segment gesture lets the musicians or dancers define the gesture "vocabulary" and thus work on a symbolic level to send a clear signal to an interactive music system.

In order to recognize isolated gestures, a training process should be applied first to find reasonable gesture thresholds. One user will be asked to perform several gestures in the gesture vocabulary in order to create a gesture template set and a training sample set. Gesture template set T contains m gestures and each gesture has n ($n \geq 5$) templates. Training set Tr holds both gestures in the vocabulary and non-gestures. Each gesture in the training set is labeled by its corresponding gesture index (1 to m) and non-gestures are labeled as 0. After preparing the template and the training set, for each template-training pair of (i, j) where $i \in T, j \in T$, we run Dynamic Time Warping and get a minimum distance array with the length of $m \times n \times \#\ training\ samples$. By applying a threshold selection manner that will be introduced in Section V, the system will select the best feature and threshold combination and store it in the memory.

In the testing process, the system will prepare $m \times n$ DTW instances for matching each gesture template against the incoming real-time gesture sequence. After initialization, the system will send an alert sound to indicate the start point of capturing. A user should start performing a gesture right after hearing the alert. Each input feature vector captured by the sensor at time t will be preprocessed and then passed to each DTW instance to compute the costs.

Since the user defines a recording duration before performing each gesture, the system automatically stops capturing after the fixed duration has elapsed. The recorded gesture is then compared using DTW to every template. If the distance (DTW cost value) between the real-time input and any gesture template is lower than the threshold of the corresponding gesture, the gesture will be selected as a "candidate." Since several gestures might be the valid "candidates," gesture classification will rely on a K-Nearest-Neighbor approach, which will be covered in Section V.

5. Continuous Gesture Recognition

For isolated gesture recognition, it would be nice to have a triggering mechanism to activate the recognition process. In this case, the recognizer would know where to start looking for a gesture. However, for real performances, a triggering strategy seems to be naïve. After all, the benefits of a free-hand gesture are minimized if the performer is constrained to simultaneously operate a foot pedal or some other start signal. If one requires the use of foot pedals or some other triggering device, perhaps the trigger itself could better serve to give commands in the first place. If the user can operate a mouse for signaling, then the mouse could be used more easily to simply click on a button to issue a command.

Therefore, interesting gestural control requires a method to spot a meaningful gesture within a continuous signal stream. Initially, I tried the same method used for isolated gesture recognition, which creates $m \times n$ DTW instances before accepting data from sensors and then compares the minimum cost between each template and the real-time input when input data comes. However, the system gets poor recognition performance. The biggest problem is that the features used are positions relative to the starting position, which is the position when tracking begins. In practice, the position where tracking begins and the position where a gesture begins may not be very close, so the best alignment score will be much higher than the gesture threshold obtained in the training process. In other words, in order to minimize the error due to start position mismatch, a user should start performing a gesture close to the initial tracking position.

This is not very satisfying, and it defeats the purpose of computing position-independent features in the first place. An alternative would be to use basic features that are independent of absolute position. For example, I tried subtracting an exponentially smoothed version of the position from the current position to obtain a feature that depends upon recent history but not absolute position. This did not work well for the sensor systems, features, and gestures under study, but this might be a good direction for future research. Similarly, derivatives and other position-independent features did not perform as well as the displacement-from-starting-position features.

The solution I finally chose is to have the system create new DTW instances fairly often, and for each DTW instance use a new start position for the real input. Thus, a

dynamic time warping with sliding window method was adopted for continuous gesture recognition.

5.1. Dynamic Time Warping with Sliding Window

For each gesture template, before running real time testing, the system allocates a queue Q with a capacity of *n* in memory. When input data is received by the system, it pushes a new DTW instance into Q every *t* samples. Moreover, if Q reaches the capacity, it pops the first DTW instance out of Q. DTW instance keeps the current input start point and does DTW comparison according to this start point. Since after each new data vector is passed to a DTW instance, DTW does preprocessing according to the "calibrated" start position. The cumulative error between the "calibrated" start position and "true" start position is acceptable.

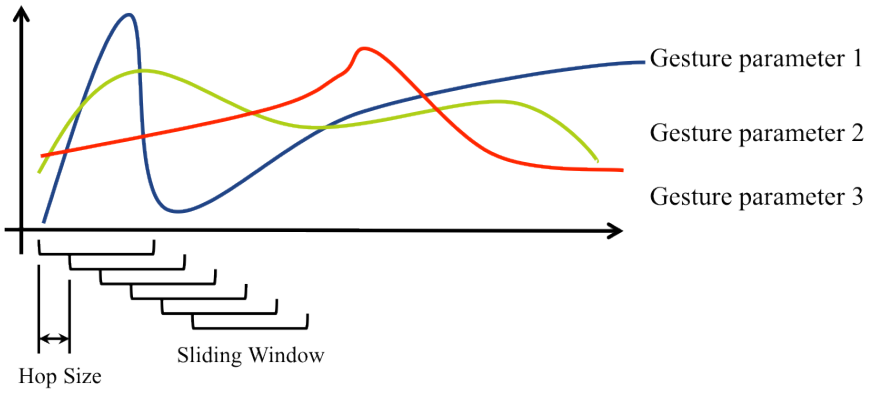

Figure 6: For a continuous gesture, system runs a new DTW instance in each window.

The system compares the latest real time observations captured by sensors with each gesture template in search of a significant match. The system applies the following decision method for each sample point until a command is recognized.

Step 1: Read a new input feature vector.
Step 2: If it is time to begin a new window, create a new DTW instance, which keeps the current input vector as its starting point. Destroy the first DTW instance in the queue.
Step 3: For each DTW instance, subtract its own starting point from the input vector and do data preprocessing.

Step 4: Compute the best alignment cost between template and preprocessed data vector.
Step 5: If the lowest cost is lower than that of the threshold model, set the gesture as a "candidate." If no threshold is exceeded, then no command is recognized. Jump back to step 1.
Step 6: If only one candidate exists, output it as the recognition result.
Step 7: If the system provides more than one candidates, run the K-Nearest-Neighbor algorithm to find the best result.

In this section, we discussed how to preprocess data and compare two processed gesture sequences by using the dynamic time warping algorithm. For isolated gesture recognition, since the starting and end points of a gesture are given, the system can run the DTW algorithm between input and each template directly. The template with the lowest best alignment score is more likely to be the result to output. However, for continuous gesture recognition, since gesture patterns are not restricted to any particular spatial locations, any data point in the input sequence can be a potential starting point of a gesture. A sliding window method is applied to allow for constantly changing starting points.

V. Threshold Model

After the process in Section VI, the system gets the best alignment scores between the training data or real time input and the templates. However, the system should still figure out whether the best alignment between an input and a template is good enough to output or which gesture pattern matches the input best. In order to solve these problems, a threshold model is used to find the candidate threshold for each gesture pattern. Since the best threshold is based on a measure of best performance (the F-measure), we can also search for the best feature combination that optimizes the system recognition performance.

1. Feature Combination Space and Feature Selection

To train a meaningful gesture recognition model, a critical step is to select few but useful features. However, for different people and for different gestures, the gesture patterns can be totally different. Thus, setting a fixed feature combination rule might not work optimally for to every person and every set of gestures. My solution is to use the users' personal gesture patterns and train the threshold model by trying each reasonable feature combination to find the best combination. In my implementation, the variables in a feature combination are the following:

1.1. Input Feature

Sensors capture the motion of the users and translate it into a high-dimensional feature vector. When the system receives the data sequence, it should conduct feature selection at first. For example, the feature vector produced by a Kinect contains 45 or 60 (depending on different software) dimensions. These represent the x-, y- and z-axis position of 15 or 20 skeleton joints. If the gesture is performed by the user's left hand, the system can either take the 3-D feature vector of the left hand into consideration or use the 3-D positions of left hand, left elbow, and even left shoulder as the input features. Thus, even for a simple hand gesture recognizer, the system can many alternatives for input feature selection.

In principle, if there are *N* features, one could search all combinations of features to find the combination that gives the best performance. However, with up to 60

dimensions, the search space could be 2^{60}, which is prohibitive. Rather than search every combination, the system searches only a set of hand-selected feature combinations. For the gestures studied here, there are three combinations: hand *x* and *y*, hand *x* and *y* plus elbow *x* and *y*, and hand *x* and *y* plus elbow *x* and *y* plus shoulder *x* and *y*.

In terms of matching two sequences, it might also be useful to add the velocity of each dimension as extra features. So including the velocity information of each dimension will be another option.

1.2. Preprocessing Function and Parameters

Secondly, the system should also define which preprocessing function should be used to smooth data. Moreover, each preprocessing function has parameters to be determined. In my implementation, for real-time continuous gesture recognition, the system provides two smoothing methods to do preprocessing. For the exponential smoothing method, the system also should assign an alpha value, which is in the range between 0 and 1. Typically, the system will try four or five different values for alpha . Thus, there are 1 approach $\times$ 4 possible parameter values, or 4 in total to be taken into consideration.

1.3. Costs of Dynamic Time Warping

For the Dynamic Time Warping Algorithm, the costs of insertion and deletion bias the best "warping" path. Additionally, different insertion and deletion cost combinations certainly change the best alignment cost for each sample pair. Thus, setting both insertion and deletion costs as variables should help obtain the best threshold for the best match. To limit the search, insertion and deletion costs are chosen from only 4 possible values: 0.0, 0.1, 0.3 and 0.5. Therefore, the number of possible combinations for different insertion and deletion costs is 16.

If we go though all possible input feature combinations, different preprocessing functions with parameters and possible DTW cost combinations, there are in total $3 \times 2 \times 4 \times 16 = 384$ different combinations of parameters to evaluate over all the training data.

2. F-measure Evaluation and Threshold Selection

2.1. F-measure Evaluation

In statistics, the F-measure is a measure of a test's accuracy [20]. The F-measure considers both precision *p* and recall *r* of the test to compute its F-measure score *f*. Moreover, the score *f* can be interpreted as a weighted average of the precision *p* and recall *r*. The best score the F-measure can achieve is 1 and the worst score is 0. Specifically,

$$p = \frac{correct\ result\ (true\ positive)}{correct\ result + unexpected\ result\ (false\ positive)} \quad (7)$$

$$r = \frac{correct\ result\ (true\ positive)}{correct\ result + missing\ result\ (false\ negitive)} \quad (8)$$

$$f = \frac{2 \times precision \times recall}{precision\ +\ recall} \quad (9)$$

To evaluate the performance of the gesture threshold, the F-measure is a reasonable and effective measurement method. For each gesture, the threshold with the highest F-measure score will be considered as the most reliable threshold to be applied in real time.

2.2. Threshold Selection

In the training process, by computing the best alignment cost between each gesture sample pair (i, j), where sample *i* comes from template set of size *q* and sample *j* is in the training set of size *r*, the system can obtain a two-dimensional minimum distance matrix *Distance*, which has the size of $q \times r$. $Distance[j][i]$ keeps the best alignment cost of sample pair(i, j). Assuming there are *m* gesture patterns or categories and each gesture pattern has *n* templates in the template set (so $q = m \times n$), for each training example *j*, each gesture pattern has *n* minimum distance values, one for each template. We can find one best (the lowest score) and one worst (the highest score) thresholds among those *n* scores. In other words, for each row of matrix *Distance,* we can find *m*

best and *m* worst thresholds for *m* gesture patterns. For each gesture pattern, the system maintains two arrays with the length of r for storing the best and worst thresholds of each training sample.

In order to set a reasonable threshold for the binary classification of one gesture pattern, we can compute the F-measure on the overlap of the best thresholds of those incorrect gesture patterns and the worst thresholds of the correct gesture pattern. For each gesture pattern *k*, the system will merge the best threshold array of *k* and the worst threshold arrays of the other templates except *k* by the following:

for each gesture *k*:
    initialize a new array *T;*
    for each training sample *j*:
        if the label of *j* is equal to gesture *k*
            T [*j*] = the *j*th element
                of the worst threshold array of gesture *k*;
        else
            T [*j*] = the *j*th element
                of the best threshold array of gesture
                whose index is the label of *j*;
    end
    do F-measure on *T* to find the best threshold with the highest F-score;
end

Then, for each array *T*, the system sorts the array and tries every element as the threshold for classifying gesture *m* and computes the F-measure. The system can get true positive, false positive and false negative values for each element. The rule is the following:

for each element *t* in the sorted array T

threshold = *t. score value*;

gesture index = *t . gesture index;*

for each element *s* in the sorted array T

if s. score value is not greater than *threshold*

if s. gesture index is equal to *gesture index*

true positive + 1;

else

false positive + 1;

else if s. gesture index is equal to *gesture index*

false negative + 1;

end

end

precision = true positive / (true positive + false negative);

recall = true positive / (true positive + false negative);

$f = (2 \times precision \times recall) / (precision + recall)$;

end

After running the F-measure evaluation on every gesture pattern, the system can obtain a gesture classification threshold array. The following figure shows the best gesture classification threshold of each gesture pattern (gesture 1 to 5) and its true positive (tp), false positive (fp), false negative (fn), precision and recall value.

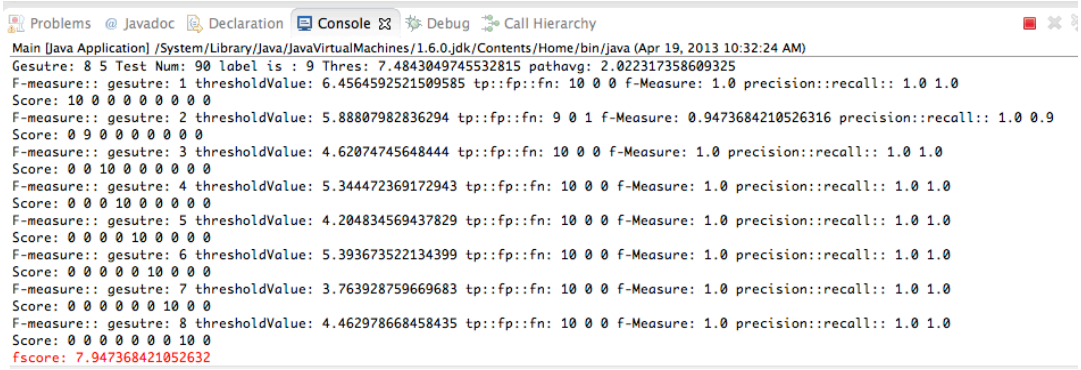

Figure 7: F-measure evaluation result. Details including classification threshold, true positive (tp), false positive (fp), false negative (fn), precision, recall value and F- measure score for each gesture.

3. Solving Conflicts by K-Nearest-Neighbor

In real-time testing, although the system has classification thresholds for each gesture pattern, it is still possible that the best (lowest) alignment scores fall below the predetermined thresholds for at least two different gestures. In those cases, the system should choose one over several "candidate" gestures as the final result. The selection mechanism I implement uses the K-Nearest-Neighbor algorithm. The K-Nearest-Neighbor algorithm is a type of instance-based learning algorithm, which classifies objects based on the K closest training examples in the feature space. Since for each input vector, the system runs q DTW comparisons to align the input vector against q templates, it can get a set of best alignment scores S between each template and the input vector, which $S = \{s_1, s_2, s_3, s_4, \dots, s_q\}$. Considering the impact of the different threshold values, the algorithm normalizes the distance between each template and the real input vector by its threshold.

$$d_i = \frac{s_i}{threshold_i} \qquad (10)$$

After ranking d_i from low to high, the system can make the decision by K-Nearest-Neighbor "majority voting." However, one of the drawbacks of "majority voting" is that an instance might be misclassified if it is too close to the classification boundary between two classes. In order to solve this problem, in my implementation, each neighbor votes with weighted values. The weighted value of the jth nearest neighbor is calculated by

$$v_j = \frac{K + 1 - j}{\sum_{j=1}^{K} j} \qquad (11)$$

For example, for 5-Nearest-Nneighbor, the weighted values from the first neighbor to the fifth neighbor are $\{0.33, 0.26, 0.2, 0.133, 0.0667\}$. The class with the highest sum of weights will be selected as the final result. However, if two classes get the same highest value, the system will suspend the decision until next input feature vector comes.

In this section, an F-measure evaluation based threshold model is introduced to find the best combination of gesture pattern threshold and feature parameters according to the gesture templates and training data. Moreover, in order to get the recognition result over multiple candidates, a K-Nearest-Neighbor algorithm is applied to classify the input sequence in the real-time decision process.

VI. Gesture Vocabulary

Defining a feasible gesture vocabulary is always a difficult aspect of creating a gesture recognition system. A well-designed gesture vocabulary will not only make the system become user-friendlier but also improve the recognition accuracy. In this section, I will discuss how to build a gesture vocabulary from an HCI perspective.

1. Gestures

When we are talking about the word "gesture," it always means a form of non-verbal communication in which visible bodily actions communicate particular messages. According to how people perform the gesture, it can be labeled as a static gesture, a dynamic gesture or a spatio-temporal gesture [21]. Specifically, static gestures are postures, which do not take movements into account. Dynamic gestures can be defined as the sequences of static gestures. Spatio-temporal gestures are the subgroup of dynamic gestures that move through the workspace over time. In this thesis, I am focusing on those spatio-temporal command gestures that can signal some specific event.

2. Design Guidelines

2.1. Avoiding Simple Gestures

Simple gestures, such as moving hand horizontally or vertically, can easily performed by the testers. In isolated gesture recognition testing, those simple gestures get high recognition rates. However, for continuous gesture recognition, if we include those simple gestures in our gesture vocabulary, we increase the hazard of false triggering. Gesture recognition must be very tolerant of spontaneous motion by the users that might match a gesture. Otherwise, users would suffer rigid constraints to their behavior. Thus, the best way to minimize both rigid constraints and false triggering is to avoid defining those simple gestures in the vocabulary. When users try to add a new gesture into vocabulary, the system should suggest designing a more complicated gesture rather than a simple one. If the user can record "background" activity, that is, typical movement not containing gestures, then this background activity can be searched for any candidate

gesture. If one or more fairly good matches are found, the user can be warned that the candidate is not clearly distinguishable from typical non-command movements.

2.2. Minimizing the Similarity between Gestures

Moreover, the similarity between two gestures is another big issue to cause false recognition. Since the Dynamic Time Warping algorithm tries to find the best alignment between a real-time input sequence and gesture templates, if two gestures share a great similarity, both of them will be selected as gesture candidates. Although the recognition system provides a mechanism to solve such conflicts, there is no guarantee that the decision will be always right. Moreover, any deformation of the real input sequence or inaccurate measurement can result in a false recognition. Therefore, if we can minimize the similarity among different gestures, it will make the recognition more accurate.

2.3. Considering the Overlap between Gestures

Complex gestures are often composed of simpler gestures. When the user defines a gesture, it may overlap with existing gestures. Consider the following example: Gesture 1 and Gesture 2 (see Figure 8) are gesture patterns stored in the vocabulary. When the recognition system receives an input sequence, which is drawn in the third column of Figure 8, it is hard to classify the input since the middle of the input gesture matches the end of Gesture 1 and the beginning of Gesture 2. Although the user intended to perform Gesture 2, rather than Gesture 1, the system recognizes the sequence as Gesture 1 since it matches Gesture 1 first. Overlaps between gesture patterns bring uncertainty to the recognition system, and it is hard to disambiguate overlapping gestures just within the recognition system itself.

Figure 8: Input gesture sequence is mixed by gesture 1 and gesture 2.

2.4. Limiting the Vocabulary

Limiting the vocabulary is important, and will both benefit the users and the recognition system. Since the users need to utilize those gestures in real time, they should be comfortable with all the gestures in the vocabulary. After the training process, users should practice gestures before performing with them. Only when the gestures are performed without hesitation, the users will keep their performance in focus rather than be distracted by gesture production and recognition. Additionally, limited gestures will not only help the users reduce their learning curves but also cut the computation and make recognition easier. The current system has demonstrated good performance with five to eight gestures.

2.5. Minimizing Gesture Deformation

Since all thresholds are based on alignments between gesture templates and training samples, a deformed gesture sample may raise the threshold value and lead to the recognition of a larger set of gestures. Thus, the users should be very careful when they perform any gesture. Especially, for Kinect, since it records the x-, y- and z-axis position of skeleton joints, bad trajectories in templates and training will be dangerous.

On the other hand, training data should be representative of actual gestures. If gestures must have substantial variation in a performance, the training data should reflect this. Otherwise, some gestures may not be recognized. If a large degree of gesture deformation is allowed, then the challenge will be making gestures distinctive enough that they are not confused.

VII. Experiments and Results

In this section, a gesture vocabulary, which contains eight gestures, is defined and applied for testing gesture recognition accuracy. The experimental results are presented, including both isolated and continuous gesture recognition.

1. Gesture Vocabulary

To evaluate both isolated and continuous gesture recognition, a human tester is asked to perform eight simple hand gestures (see Figure 9) using his left hand. Each "command" gesture is repeated five times. Thus, the template set stores ten similar gesture feature vector sequences for each gesture pattern. The tester is also asked to perform ten training gestures for each gesture pattern. Another ten non-gesture training samples are also recorded and labeled with 0 (non-gesture). Each sample in the template set and training set has the same duration, which is 2 seconds. Figure 9 shows the hand 2-D trajectories of those gestures. The start point of each gesture is labeled as a dot and the end of the gesture is labeled as an arrow.

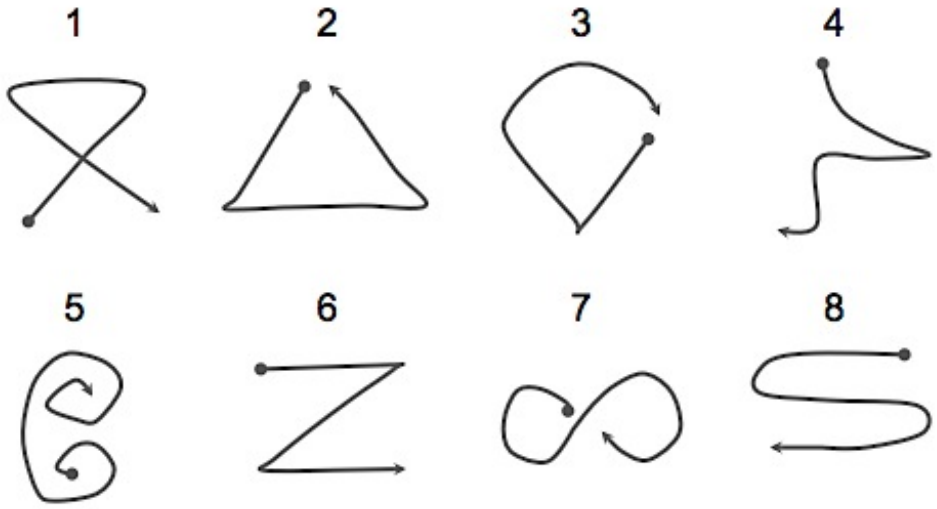

Figure 9: Gesture vocabulary for testing. The start point of each gesture is labeled as a dot and the end of the gesture is labeled as an arrow.

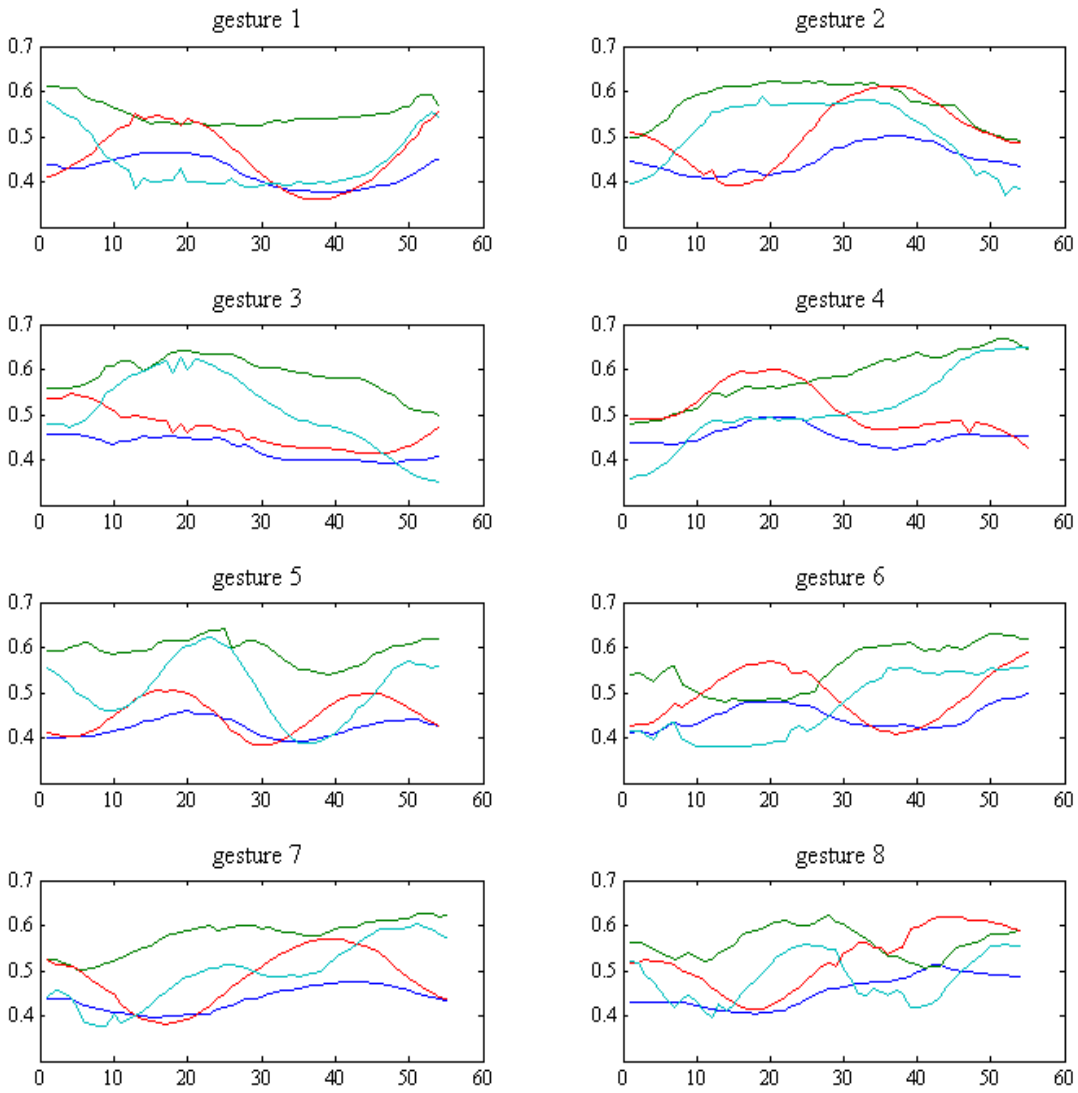

Figure 10: The 4-dimensional hand gesture trajectory of each gesture pattern**.**

All gestures are stored as an entry in a MATLAB mat file. After collecting data, we can visualize gestures in MATLAB. Figure 10 shows the 4-dimensional hand gesture trajectory of each gesture (gesture 1 - 8). Specifically, the blue curve represents the x-axis trajectory of the tester's left elbow. The dark green curve stands for the y-axis trajectory of tester's left elbow. The red curve characterizes the x-axis trajectory of tester's left hand and the light green curve depicts the y-axis trajectory of his left hand.

2. Isolated Gesture Recognition Testing

To evaluate isolated gesture recognition, the tester is asked to perform 100 test samples, which contains 10 samples for each gesture and 20 non-gesture samples. Each test sample is 2.5 seconds long. The tester performs the testing gesture in random order. The recognition accuracy depends on the recognition precision, which shows how many results the system provides match the true label of the samples. Since my implementation can assign different feature combinations, the recognition accuracy varies according to different feature settings. Table 1 shows the details of isolated recognition accuracy giving different feature and parameter combinations.

Feature Vector	Smoothing Method	Relative Position	Alpha	Insertion Cost	Deletion Cost	Recognition Accuracy
x-/y- hand	Exponential	False	0.5	0.0	0.0	89%
x-/y- hand	Exponential	False	0.8	0.0	0.1	94%
x-/y- hand	Exponential	False	0.8	0.0	0.0	87%
x-/y- hand	Exponential	False	0.8	0.1	0.3	87%
x-/y- hand	Exponential	True	0.8	0.1	0.3	85%
x-/y- elbow and hand	Exponential	False	0.6	0.1	0.3	87%
x-/y- elbow and hand	Exponential	True	0.5	0.1	0.4	81%
x-/y- elbow and hand	Exponential	False	0.5	0.1	0.4	84%

Table 1: Isolated Gesture Recognition Accuracy.

Since most of previous research evaluated the accuracy of recognition based on fixed parameter settings, we cannot compare their results to find how different parameter settings influence the accuracy. However, we can compare the overall recognition rate with their results. In [7], A.Corradini tested his DTW-based off-line recognition approach on a small gesture vocabulary, which is composed of five preliminary gestures: stop, waving right, waving left, go right and go left. For each gesture, he reported the recognition accuracy rates of each gesture are 91.3%, 88.5%, 89.2%, 91.6% and 91.3%. Thus, the overall recognition rate of his DTW-based approach is 90.4%. According to

the different parameter settings, the recognition rates my implementation can achieve are mainly floating between 84% and 88% and the highest accuracy rate is 94%. Those two results are comparable, and a good parameter tuning can achieve a high accuracy for a specific testing set.

3. Continuous Gesture Recognition Testing

To evaluate continuous gesture recognition, the tester is asked to perform 25 testing samples, which have varying durations between 10 seconds to 20 seconds. Each sample contains 1 to 3 gestures in the vocabulary and those gestures are labeled after recording. Figure 11 shows the two-dimensional trajectory of one of the test samples, which contains gesture 8 and gesture 1 in the vocabulary. The evaluation measurements are the accuracy of the recognition. We consider that a gesture is correctly detected if the start / end point the recognition system finds the correct gesture within 6 samples (0.2 seconds) of the "true" label.

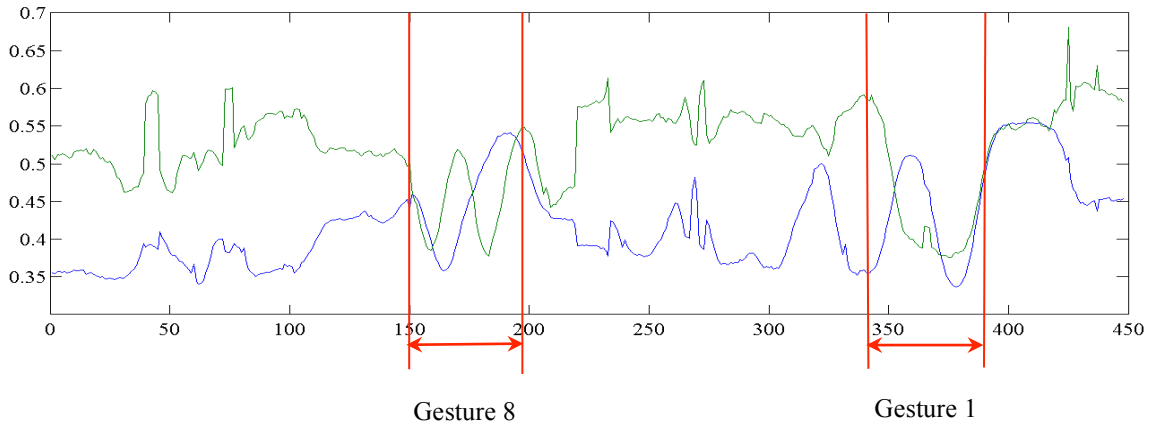

Figure 11: Two-dimensional trajectory of one test sample, which contains gesture 8 and gesture 1.

The accuracy varies according to different feature combinations and threshold settings. If the system has a high F-measure score on the training data, then gestures in a continuous stream can be detected by the system with high confidence. For instance, for the test sample shown in Figure 11, since the system selects the thresholds with high F-measure score (7.83 / 8.0), two gestures in the sequence are detected clearly. The best alignment cost between gesture templates and the real input fall below the threshold of gesture 8 around the 193rd sample, which is considered as the end point of a gesture.

Similarly, the best alignment between gesture 1 and real input falls below the threshold of gesture 1 around the 384th sample, which is considered as the end point of gesture 1. Figures 12 and 13 are snapshots of this real-time testing. Figure 14 shows the best alignment between each templates and each input vector.

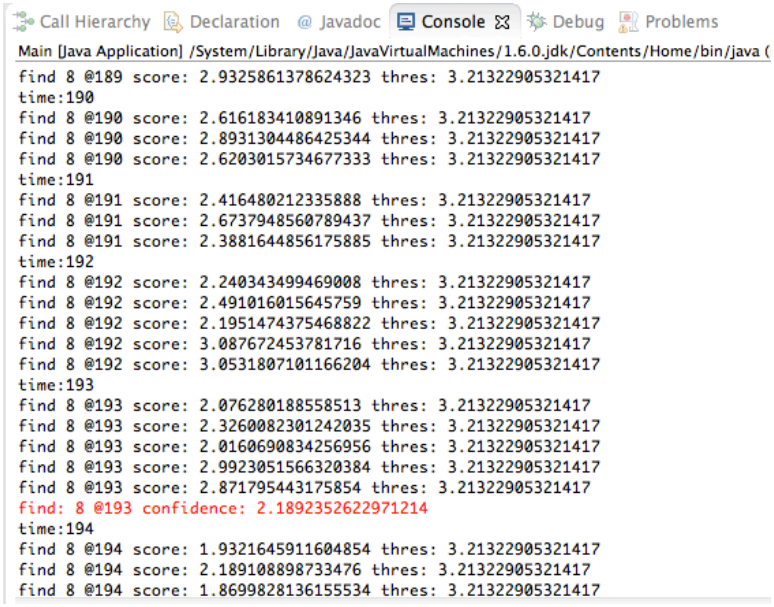

Figure 12: The system detects gesture pattern 8 around the 193rd sample.

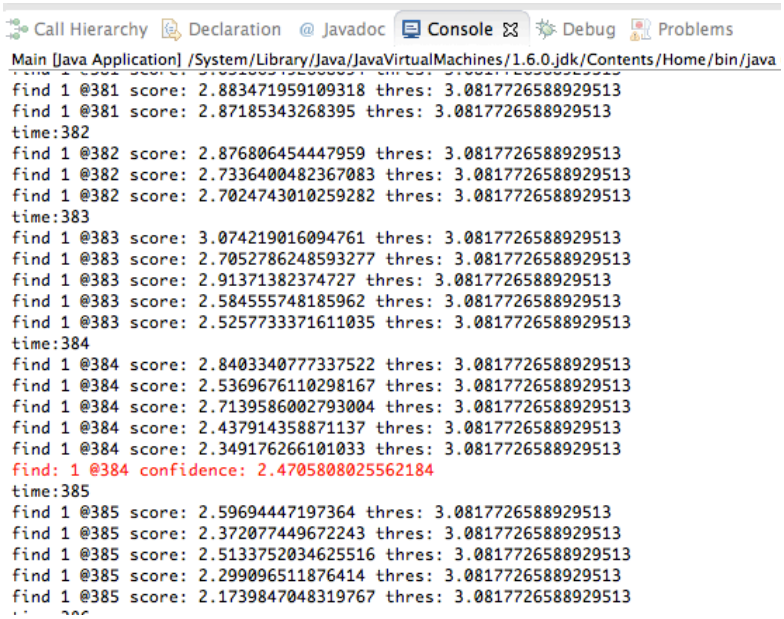

Figure 13: The system detects gesture pattern 1 around the 384th sample.

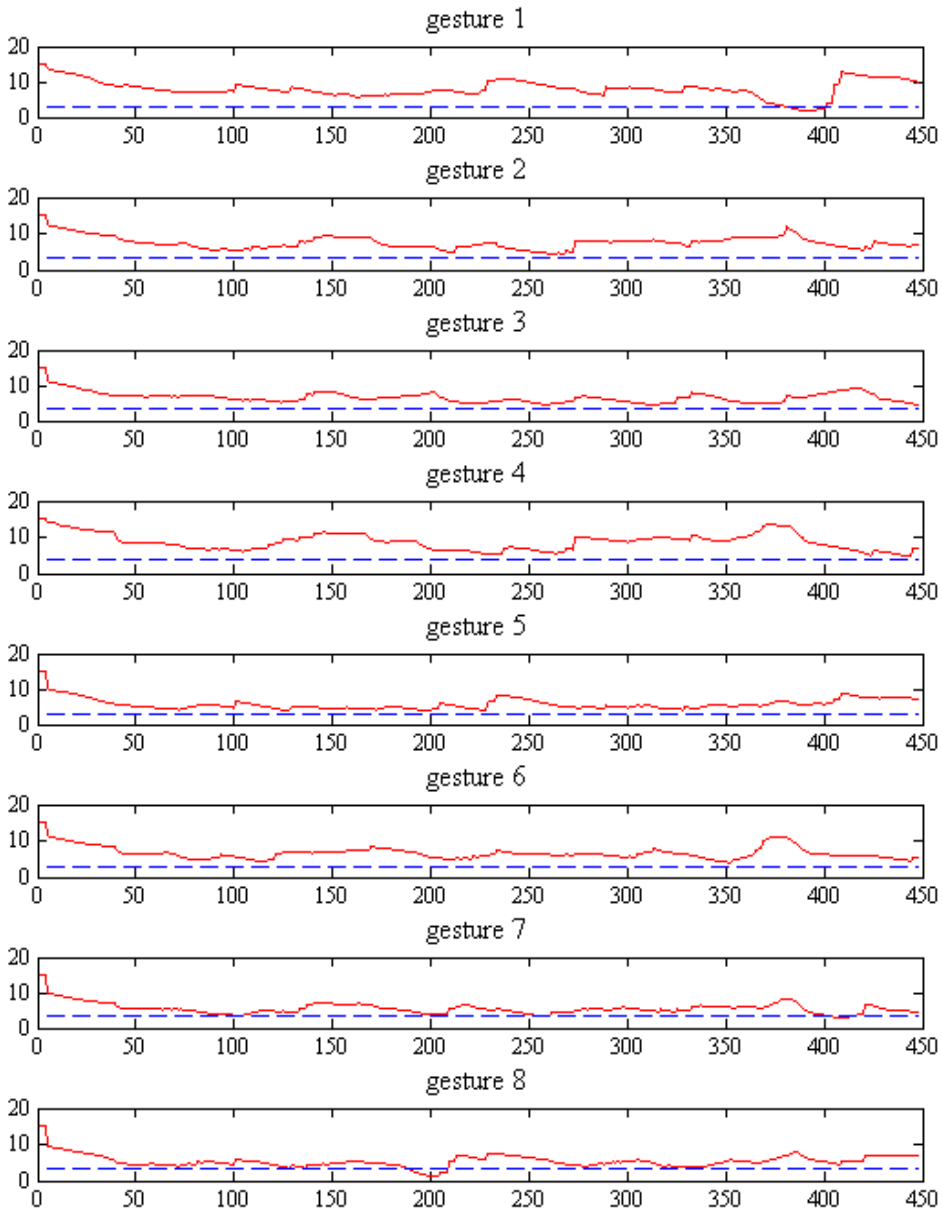

Figure 14: Red solid curves are the best alignment score between each template and each input vector. The dotted **blue** lines illustrate the thresholds of each gesture template.

When testing 60 gestures in 25 continuous sequences, we find there is a big difference in the recognition accuracy with different feature combination. Generally, the feature combination with the higher F-measure score achieves higher recognition

accuracy. The highest recognition accuracy is 89%, where the F-measure score is 7.83/8.0.

Feature Vector	Smoothing Method	Relative Position	Alpha	Insertion Cost	Deletion Cost	F-Score Value	Recognition Accuracy
x-/y- hand	Exponential	False	0.5	0.0	0.0	6.37	72%
x-/y- hand	Exponential	False	0.5	0.1	0.3	6.26	70%
x-/y- hand	Exponential	False	0.8	0.0	0.0	6.44	75%
x-/y- hand	Exponential	False	0.8	0.1	0.3	7.63	83%
x-/y- hand	Exponential	True	0.8	0.0	0.1	7.83	89%
x-/y- elbow and hand	Exponential	False	0.6	0.1	0.3	6.10	67%
x-/y- elbow and hand	Exponential	True	0.5	0.1	0.4	7.21	79%
x-/y- elbow and hand	Exponential	False	0.5	0.1	0.4	6.67	74%

Table 2: Continuous Gesture Recognition Accuracy.

In previous research, Miguel A Bautista et al. achieved 67.81% recognition accuracy by using a probability-based DTW [8]. Moreover, Narayanan C. Krishnan et al. got a recognition result where precision is 0.78 and recall is 0.93 by using Adaboost-based adaptive threshold model [15]. Although different gestures and sensors are involved, making comparisons difficult, our DTW-based gesture spotting approach, with about 90% recognition accuracy, has the same order-of-magnitude error rate.

VIII. Applications

In this section, we will discuss several possible applications by using the proposed music gesture recognition system. Generally speaking, this gesture recognition system can either assist the musicians in real time performance or act as a new musical interface for interactive music performance.

1. Real-Time Music Performance Gesture Recognizer

The original purpose of building a gesture recognizer is to use it in real-time performance. By training a personal small gesture vocabulary before the performance, musicians can use the gesture recognizer to send their "commands" to the computer. Thus, musicians can control sound parameters or output sounds without the help of off-stage sound engineers.

2. Gesture-based Interactive Music Performance

Moreover, as multimedia performance becomes popular, this system can also be used for gesture-based interactive music performance. A performer can act as a "conductor" of a virtual band in the real performance. According to the predetermined mapping strategy, each virtual instrument reacts to the performer's gestures by performing specific sound events.

3. Multimedia Installation

Since the gesture recognizer communicates to other programs via OpenSoundControl, it can also cooperate with a multimedia installation to act as a bridge between users and the core system. According to users' gestures or facial expressions, the recognition sends a determined command to the core system and the system reacts to the input via audio and video.

IX. Conclusion and Future Work

In this thesis, a Dynamic Time Warping based gesture spotting and recognition system is built to spot and recognize musicians' "command" gestures in real performance. In order to find the best feature combinations, the system evaluates different combinations by applying F-measure evaluation on the training data. The system sends recognized commands via OSC messages. It currently supports Microsoft Kinect and an on-body wireless sensor as the input. Currently, the system recognizes isolated gestures with almost 94% accuracy and achieves 89% accuracy on continuous gesture recognition. A working implementation has been written in Java. However, there is still work to be done in the future to improve the performance of the system.

1. Improving Recognition Accuracy

The system still does not get a perfect performance on continuous gesture spotting and recognition. In my testing, some particular non-gesture patterns can easily match one of the gesture patterns and make a false triggering. Moreover, if two gestures share a great similarity in one of their feature dimensions, they also can be confused. I believe we can try to improve the recognition accuracy by paying attention the following things:

1.1. The Distance Function

In my implementation, the Euclidean distance function is used to compute the distance between two feature vectors. Some people have used Mahalanobis distance, which is preferred over Euclidean distance for computing the similarity between the feature vectors [22] [23]. I think we can improve the recognition accuracy with an improved distance function.

1.2. The Matching Algorithm

In my thesis, the original DTW is selected as the algorithm to measure the similarity between two time series. I wonder if we can modify the original DTW or apply newer technology to get a good classification result. R. Oka purposed a modified DTW algorithm, Continuous Dynamic Programming (CDP) in his paper for spotting words in continuous speech [24]. This idea might help us think about how to modify the basic DTW algorithm and get a better performance. Moreover, trying to combine other latest

technology, such as Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs), with the threshold model might be a good approach.

2. Enhancing the Efficiency

Since the system computes the best alignment costs between every template in the vocabulary and the real input, it uses a lot of computation. Assuming the system has *m* gestures, which have *n* templates with the length of *l*, we create a DTW comparison for every *t* samples, and the computation for one DTW matrix column is *c.* the total computation is $m \times n \times \frac{l}{t} \times c$. In my implementation, there are 8 gestures (m = 8), each gesture has 5 templates (n = 5), the length of each template is 2 seconds or 60 samples (l = 60), the time interval between two DTW sliding windows is 4 samples (t = 4) and for one DTW column, the system takes 0.0000026 seconds (2.6 μs) and the total cost for comparing one input vector with all gesture templates is 0.00156 seconds (1.56 ms) on average. Since Kinect pushes data into system every 30.3 ms, the computation is still affordable. In fact, the total computation takes less than 5% of a core. However, if we have more gestures, the system will finally reach its limit. A possible solution might be breaking a gesture into several short sub-gesture units. Since the comparison is over a smaller space of sub-gesture units and the input vector, many invalid gestures can be eliminated early if the prefix sub-gesture units match the templates poorly. Alternatively, there is no point to continuing a DTW computation, where distance increases monotonically with the length of the match, after the distance computed so far exceeds the match threshold.

3. A Comprehensive Gesture Study

In my thesis, I briefly discuss how to build a gesture vocabulary. However, in order to use the program in a real performance, more comprehensive HCI research on Gestural User Interfaces should be conducted. In fact, an HCI evaluation process should be used to evaluate each gesture pattern from an HCI perspective and the gesture design should be guided by the results of the evaluation process. After several gesture design iterations, a gesture library will be built for recommending to users gesture patterns from which they can build future gesture vocabularies.

References

[1] J. Francoise, “Real time Segmentation and Recognition of Gestures using Hierarchical Markov Models,” Master’s Thesis, Universite Pierre et Marie Curie, Ircam, 2011.

[2] R.Fiebrink, “Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance,” Ph.D. dissertation, Faculty of Princeton University, 2011.

[3] F. Bevilacqua and N. Schnell, “Wireless sensor interface and gesture-follower for music pedagogy,” *Proceedings of the 7th international conference on New interfaces for musical expression*, pp. 124–129, 2007.

[4] N. Gillian and R. Knapp, “A Machine Learning Tool-box For Musician Computer Interaction,” in *Proceedings of the 2011 Conference on New Interfaces for Musical Expression*, 2011.

[5] K. Takahashi, S. Seki, and R. Oka, “ Spotting Recognition of Human Gestures from Motion Images,” Technical Report IE92 134, The Inst. of Electronics, Information, and Comm. Engineers, Japan, pp. 9-16, 1992 (in Japanese).

[6] A. Akl and S. Valaee, “Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, and compressive sensing,” In ICASSP, pp. 2270–2273, 2010.

[7] A. Corradini, “Dynamic time warping for off-line recognition of a small gesture vocabulary,” in RATFG-RTS ’01: *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems (RATFG-RTS’01).* Washington, DC, USA: IEEE Computer Society, 2001.

[8] Bautista, Miguel A, and Hernández, Antoni and Ponce, Victor and Perez Sala, Xavier and Baró, Xavier and Pujol, Oriol andAngulo, Cecilio and Escalera, "Probability-based Dynamic Time Warping for Gesture Recognition on RGB-D data," *Proceedings of the 21st International Conference on Pattern Recognition*. International Conference on Pattern Recogntion Workshops, WDIA, 2012.

[9] R. Kjeldsen and J. Kender, " Visual Hand Gesture Recognition for Window System Control," *Proc. Int'l Workshop Automatic Face- and Gesture-Recognition*, pp. 184-188, Zurich, Switzerland, 1995.

[10] S. Fels and G. Hinton, "Glove-talk: A neural network interface between a data-glove and a speech synthesizer," *Neural Networks, IEEE Transactions* on, vol. 4, no. 1, pp. 2–8, 1993.

[11] X. Deyou, "A Neural Approach for Hand Gesture Recognition in Virtual Reality Driving Training System of SPG," *International Conference on Pattern Recognition*, Vol. 3, pp. 519-522, 2006.

[12] H. Lee and J. Kim, "An HMM-Based Threshold Model Approach for Gesture Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21, No. 10, pp. 961-973, 1999.

[13] K. P. Murphy and M. A. Paskin, "Linear Time Inference in Hierarchical HMMs," *Advances in Neural Information Processing Systems*, Vol.2, 2001.

[14] M. Elmezain and A. Al-Hamadi and B. Michaelis, " Robust methods for hand gesture spotting and recognition using Hidden Markov Models and Conditional Random Fields," *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium*, 2010.

[15] Krishnan, Narayanan C., Lade, Prasanth and Panchanathan, Sethuraman, "Activity gesture spotting using a threshold model based on adaptive boosting," *Multimedia and Expo (ICME), 2010 IEEE International Conference*, Vol.1, pp. 155-160, 2010.

[16] http://en.wikipedia.org/wiki/Open_Sound_Control

[17] http://en.wikipedia.org/wiki/Kinect

[18] https://github.com/Sensebloom/OSCeleton

[19] http://www.openni.org/

[20] http://en.wikipedia.org/wiki/F1_score

[21] P. Kortum, *HCI Beyond the GUI*, Chapter 3, Morgan Kaufmann, pp. 75-102, 2008.

[22] Jonathan Alon, Vassilis Athitsos, Quan Yuan, and Stan Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, pp. 1685–1699, 2008.

[23] Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan, "uwave: Accelerometer-based personalized gesture recognition and its applications," *IEEE International Conference on Pervasive Computing and Communications*, pp. 1–9, 2009.

[24] R. Oka, "Spotting method for classification of real world data," *The Computer Journal*, Vol. 41, no. 8, pp. 559–565, July 1998.