CS229 Final Project

Gesture Recognition and Classification using the Microsoft Kinect

Justin Huang
Dept. of Electrical Engineering
Stanford University
jjhuang0@stanford.edu

Chun-wei Lee
Dept. of Electrical Engineering
Stanford University
chunweil@stanford.edu

Junji Ma
Dept. of Electrical Engineering
Stanford University
junjima@stanford.edu

***Abstract*—In this report we detail a few machine learning methods in which a system can learn to recognize and classify human gestures. Our focus is primarily on Hidden Markov Models, and we discuss the advantages of using such a method.**

I. Introduction

The field of human-computer interface (HCI) has in the past few decades made strides in making human control of a computer system more convenient and more intuitive. From starting with a keyboard, the field progressed onto graphical user interfaces (GUI), to the mouse, to allowing the computer to understand speech. The idea of allowing a computer system to understand human gestures, however, has always been difficult to reach, due to the sophistication in technology required—such as fast processing power and good camera systems. In 2010, Microsoft Corporation launched the Kinect system for the Xbox 360, which combines both a stereoscopic camera, infrared sensor and more into a compact device, but more importantly was available at an easily affordable price.

Figure 1: Microsoft Kinect

The Kinect served as an interpreter that translated human gestures into game commands. Although Microsoft later released the Kinect SDK in early 2011, granting developers access to many of the Kinect's sensors, its gesture recognition algorithms remained proprietary and hidden from the public. This project is thus an effort to implement a real time gesture classification system using the Kinect for data sensing and collection.

II. Project Goals

Our project aims to explore various learning techniques for modeling human gestures, with the ultimate goal of being able to train a real time action classifier. The classifier should be able to distinguish between a gesture and nothing at all, as well as the distinguishing between different types of gestures.

A. Baseline Gestures

Our baseline gestures are the following: punch, swing, wave and slap. These are all actions performable by a single hand (in our case, we focused on the right hand).

B. Coding Environment

Our algorithms are primarily coded in the MATLAB environment, although code to trigger the Kinect and interface it with MATLAB is written in C#.

III. Dataset

A. Kinect Sensor Capability

The Kinect's image processing power allows it to track 'skeleton' for up to 3 users in front of it, but we will focus on the first person being tracked. The SDK allows us to obtain the double precision XYZ coordinates of the twenty key joint points that make up the skeleton.

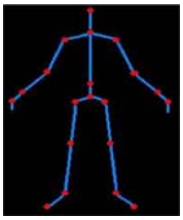

Figure 2: Kinect Skeleton

B. Data Collection

Using these coordinates, we collected four sets of data for each of our baseline gestures, with each of our team members performing 100 training examples each, resulting in a total of at least 300 examples per gesture. Each training window lasts a second, where the Kinect takes 30 frames worth of data and in which an individual must begin and finish the action in its entirety. In addition, we recorded an additional set of data to represent random noise, involving lack of action or a random flurry of action, intended for use in evaluating our models' accuracies later on in validation.

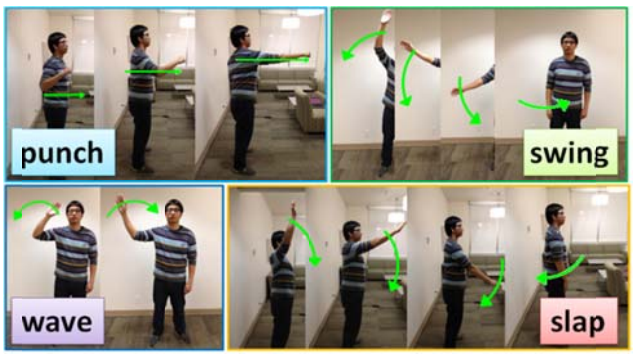

Figure 3: Examples of Baseline Gestures

Since our gestures mainly concern the movement of the right arm, we took only the XYZ coordinates of the head, spine, right hip, right elbow, right wrist and right hand, for a total of 18 values. Thus, an individual training example vector is 1x540.

C. Feature Extraction

When deliberating our features, we determined that much of the body does not move when performing our designated actions and rather only the right hand does. Thus, we decided to make our features merely dependent on the relative position of our hand to our spine coordinates. To compensate for bodies of different statures, we normalize (divide) these values by the distance between the head and the spine points, using the assumption that most of our users have a fairly consistent head-body-arm ratio. Thus our baseline features are calculated as such:

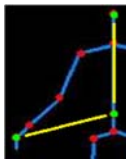

Figure 4: Selected Feature Points

$$X'_{righthand} = \frac{X_{righthand} - X_{spine}}{|X_{head} - X_{spine}|} \quad (1)$$

$$Y'_{righthand} = \frac{Y_{righthand} - Y_{spine}}{|Y_{head} - Y_{spine}|} \quad (2)$$

$$Z'_{righthand} = \frac{Z_{righthand} - Z_{spine}}{|Z_{head} - Z_{spine}|} \quad (3)$$

We also considered several other features, such as the angle between hand and elbow coordinates, as well as the velocity of the hand coordinate, though the latter is theoretically accounted for in our time frames.

IV. Exploration of Model Options

Before implementing our system, we gave consideration to a few different types of learning techniques for distinguishing gestures from one another based on our training examples.

A. Support Vector Machine (SVM)

SVMs were one of the first tools we turned to, given the availability of 'off-the-shelf' libraries such as libsvm. Our procedure was to first calculate the above features for all of our sequences of data. These vectors are then shuffled, before we divide it up such that we have 300 training examples from each set, reserving the remaining as the validation set. We then train a model from our training matrix and fit it over the test matrix to determine the recognition rate for each of our actions.

TABLE I. SVM Gesture Recognition Rates

True Labels	Detected Actions			
	Punch	***Slap***	***Swing***	***Wave***
Punch	100%	0%	0%	0%
Slap	0%	100%	0%	0%
Swing	7.1%	0%	85.7%	7.1%
Wave	0%	8.7%	0%	91.3%

It would turn out that the overall SVM accuracy was about 92.3%, which is pretty good for a first attempt; although it is clear some actions are harder for the system to detect. However, this changes when we introduce the possibility of false positives.

TABLE II. SVM Gesture Recognition Rates (w/ False Positives)

True Labels	Detected Actions				
	Punch	***Slap***	***Swing***	***Wave***	***False***
Punch	12.5%	0%	0%	87.5%	0%
Slap	0%	100%	0%	0%	0%
Swing	7.1%	33.3%	35.7%	7.1%	16.7%
Wave	82.6%	0%	13%	0%	4.35%
False	2.56%	0%	7.69%	25.6%	64.1%

The overall accuracy in this latter case dropped to a 40.97%. Ordinarily, one might improve the feature selection to boost our score. However, we determined that SVMs were not worth pursuing, due to their inability to efficiently model the time slice nature of our problem. The SVM models our training vectors as is from start to end, but realistically the same action may start in a later frame and end in an earlier frame. The SVM's lack of flexibility in accounting for this makes it a poor candidate for modeling gestures. In addition, the SVM is good for multi-class classification. However, 'no action' is very broad and cannot possibly encapsulate all possibilities and still train a good and consistent model. What we require is the model's ability to default to 'no action' if a certain likelihood threshold for the action was not reached, rather than being forced to choose one of *k* classes.

B. K-Means Clustering

One other option we explored was investigating whether a pattern existed in the sequences of the gestures such that a clustering algorithm such as K-Means could mark divisions for each class. Due to the high dimensions of our training examples (1x90, after feature selection), the classification was poor and resulted in every single example being classified as a 'punch.' Intuitively, it is understandable that K-Means would perform horribly, due to it looking at every single time slice altogether at once, making it rather inflexible.

V. Hidden Markov Model (HMM)

A. Introduction to HMMs

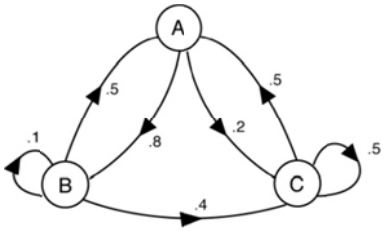

Figure 5: Example of a Hidden Markov Model

We ultimately settled on utilizing Hidden Markov Models, a graph-based model which considers a sequence of symbols emitted by true hidden states in time. Critical to the HMM are transitive probabilities, which determine how likely a state transitions to another state (or to itself), as well as emission

probabilities that dictate how likely an observation occurs at a certain state. While running the model, the true state is hidden to us, and only observations as a result of the state are shown. Based on these observations, our system calculates the log-likelihood of the most likely true state and the following sequence. In our case, our observations are the skeleton key points available to us. The hidden states are the poses or postures of the human user, and the sequence of states constitutes an action.

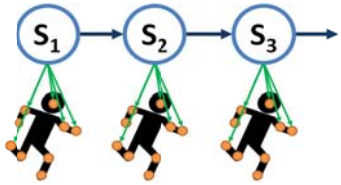

Figure 6: Hidden Markov Model of a Gesture

As we have four baseline actions, we correspondingly need to train four HMMs in which we can match our test gesture sequences against and determine the likelihood that our test data occurs in the given models.

B. The Baum-Welch Algorithm

While we possess observations/emission in the form of training data, it is rather difficult as humans to predict what a corresponding transition and/or emission probability matrix for a certain type of action is. Fortunately, the Baum-Welch algorithm allows us to estimate these parameters given an observation sequence and random initialized probability matrices. Unfortunately, due to the presence of local maxima, the methodology of random initialization does influence the end result, but was not thoroughly explored during this project.

Following the calculation of the probability matrices, determining the likelihood of a sequence would be a simple forward algorithm tracing the steps of the sequence of observations and determining the most likely states [1] [2] [3].

C. Codebook of Symbols

To continue with HMMs, our features need to be further processed. Our simple HMMs emit discrete observations out of a set list of possibilities. However, our current features originate from a theoretically infinite set of 3D points, making it rather infeasible to capture in an HMM emission probability matrix. To cope with this issue, we introduce the idea of a vocabulary, or HMM codebook. Essentially, we segment our 3D space into discrete blocks (symbols), instead grouping points together. And the 3D space is calculated relatively and normalized by the height of the person in the scene. This allows the HMM to estimate the probability that a symbol from a set vocabulary was observed.

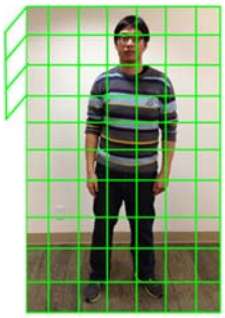

Figure 7: Segmentation of 3D Space

The segmentation was done in an arbitrary matter but a preference to give greater resolution to the XY plane, rather than to depth, since most of the baseline actions span a wide area at approximately the same depth level. If setting the number of symbols to be 98, we would make 7 divisions each along the X and Y axis, but only 2 divisions along the Z axis.

D. Model Classification Threshold

Having prepared our data, we trained our five datasets (including the noisy action set) using Baum-Welch to produce five HMMs. As mentioned before, we prepare our testing data in a similar manner, and then match the data to each of the five models to find the one that reflects the highest log-likelihood. However, it is not adequate to simply pick the 'best' model as a result of that calculation, as this would force the system to make a choice even if none of the actions were good candidates. To deal with this, we set a threshold value that the likelihood scores must surpass before the program even considers it as a best match candidate [4].

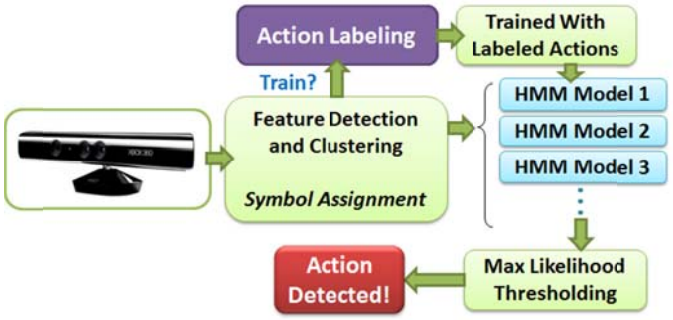

Figure 8: System Flow Chart

VI. Simulation Results and Analysis

We ran our verification set on the trained models, while exploring the effects of adjusting the number of hidden states as well as the number of symbols, in search of an optimal value.

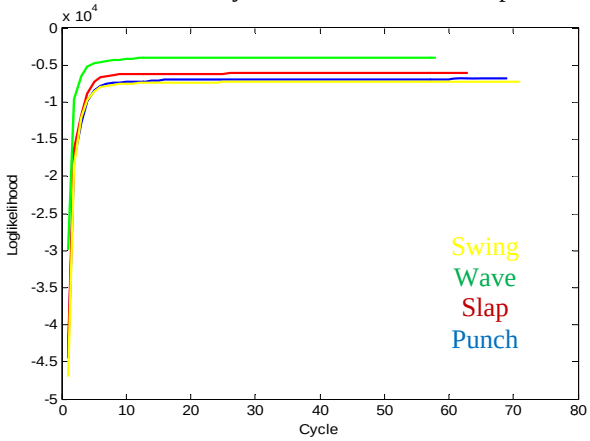

Figure 9: HMM Training Likelihood Curves

We observed that under reasonable parameters, the learning curves generally appeared to have flattened out by around the 20th cycle of training, but would not truly converge until approximately 70 cycles. In the interest of time, we placed caps to stop training iterations upon 100 cycles. Such situations imply a failure for the algorithm to converge at local optima, but surprisingly, often the resulting model still captures the designated action very well.

Simulation often yielded very good results. For num states = 30, num symbols = 72 (the same parameter values for the curves in Figure 8), we achieved the following:

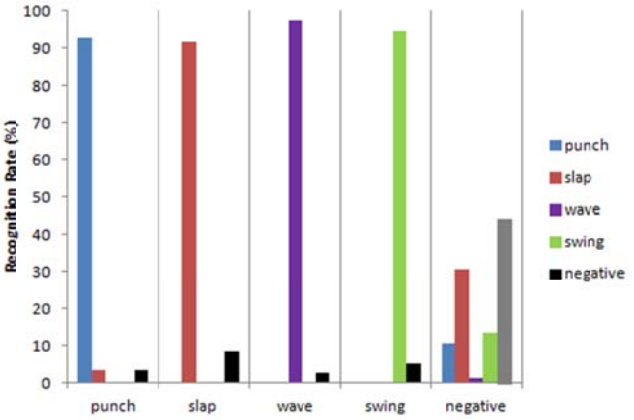

Figure 10: Gesture Recognition by Each Classifier

TABLE III. GESTURE RECOGNITION RATES (STATES=30, SYMBOLS=72)

True Labels	Detected Actions				
	Punch	***Slap***	***Swing***	***Wave***	***False***
Punch	92.86%	3.57%	0%	0%	3.57%
Slap	0%	91.67%	0%	0%	8.33%
Swing	0%	0%	97.46%	0%	2.54%
Wave	0%	0%	0%	94.63%	5.37%
False	10.46%	30.54%	1.26%	13.39%	44.35%

Based on these two charts, one can see that the designated action is being recognized very well by the corresponding HMM, with at least 90% recognition in every single case. For the 'negative' example, we see that the recognition is much more distributed between all the actions, with none having more than 50% recognition, which is good.

A. Effect of Number of States

It would be interesting to observe the effect of number of states on our model results. One might theorize that the more hidden states there are, the model should train to be better, up to a point where having too many states might become redundant if the action set is too basic. On the other hand, having too many states may slow down the training significantly, due to the extra computation needed for the matrices. Conversely, having too few states should make it difficult for the system to distinguish between all of our different actions, leading to mixed up accuracies. We first explore the effect this number has on the training runtime.

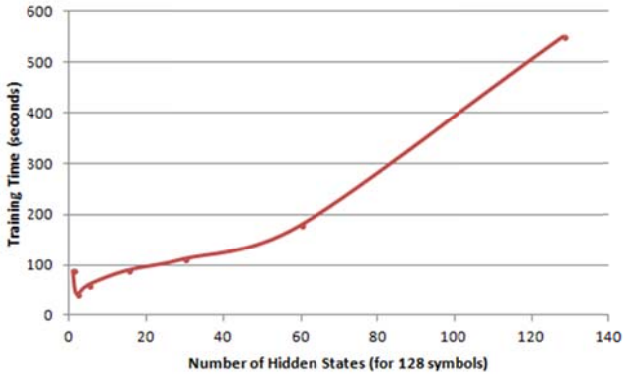

Figure 11: Train Time vs Num. of States (Symbols=128)

As expected, the training time does appear to increase for more states within the system. For very small number of states, such as in the range of 1 to 5, however, the difference is negligible enough that the computation is complete in roughly the same amounts of time.

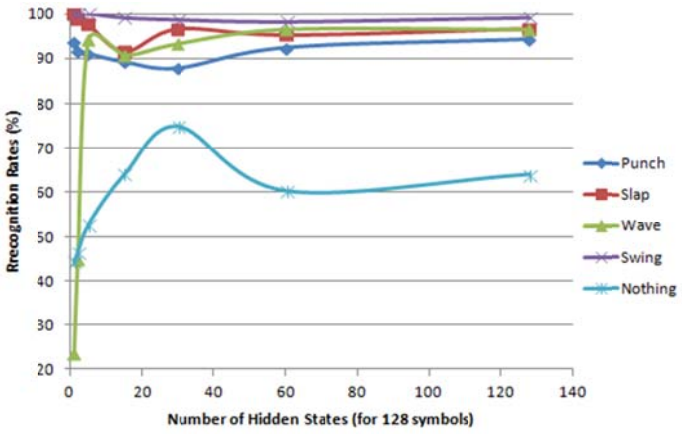

Figure 12: Recognition Rate vs Num. of States (Symbols=128)

The effect on accuracy by the number of states is perhaps more surprising. As expected, the improvement in accuracy as number of states increases stagnates. However, it appears that when the number is reduced to very small, our recognition rate remains very high, all except for the 'wave' action which drops significantly in lower state numbers, eventually becoming confused for another action (not shown in figures).

We believe that this counter intuitive result can be partially explained by the fact that although we have, say, only a single state, we have a lot of symbols. We believe that our four actions are different enough that they do not have many overlapping or shared symbols. Thus, during the likelihood calculation, although the transition probability is always 1 (one state transitioning to itself), the emission probabilities for each symbol can still be different. This allows for the trained models to classify our validation data based on emission probabilities alone. The 'wave' accuracy then dropped, likely because it shared a few common symbols with another action, enough for it to be recognized more frequently as the other. In practice, when this one-state model was applied to our real-time program, only the 'punch' action was consistently detected.

Based on our data, we determined that the optimum number of states for our particular set of actions is around 30. This is the point where all of our actions are recognized fairly well, and our models are able to distinguish the noise better as well, preventing more false positives.

B. Effect of Number of Symbols

In addition, we also performed a sweep on number of symbols (from 18 to 126), keeping Z-segmentation = 2, and X and Y-segmentations the same, such that X*Y*Z = number of symbols. We performed a runtime analysis as well, but surprisingly, the number of symbols seemed to have negligible impact on the runtime, and thus the chart is not included.

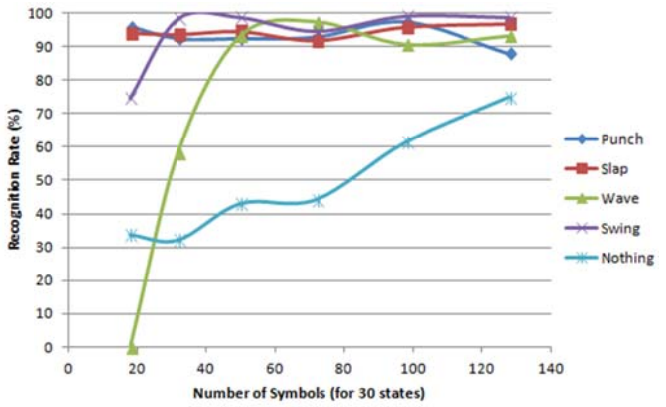

Figure 13: Recognition Rate vs Num. of Symbols (States=30)

Since a lower number of symbols equates to lower resolution in terms of our segmented, this would cause feature points that may not originally be very similar to get clustered together, confusing the model. Indeed, Figure 12 does show that as the number of symbols decrease, the recognition rate for wave and swing do significantly fall as well. Curiously, punch and slap remain very recognizable, possibly due to the fact that these are the only two actions that change in Z-coordinates, which we did not alter in terms of segmentation. Conversely, it seems that the more symbols there are, the better the overall recognition rate can become. Perhaps if we had more features and more clustering required, the runtime would increase dramatically, but in our particular case it is not evident. Thus, for our case, the optimal number of symbols seems to be around 98 and more, based on keeping all the recognition rates above 90%.

VII. Real Time Implementation

A. Implementation Details

Following our MATLAB simulations, we interfaced the Kinect API with MATLAB via C#. The models were pre-trained, with the code only involved in capturing the single second worth of data, sending it to the MATLAB module for calculation and obtaining the returned label.

Our system begins picks a valid starting state only when the hand has been stable for a while (with a difference^2 < 0.01) between frames. If the frame remains stable after this point, we do not begin recording until the difference is beyond the threshold. If the movement lasts beyond a second, we will only take the most recent 30 frames for feature processing, assuming the user wants to finalize their gesture and that the clipped portion is simply noise. The reason for such a mechanism is that our actions usually begin with zero velocity or with slight jitter that could potentially set off recording falsely. These criteria allow us to have more accurate and controlled recordings [5].

B. Observations

Our system trained on 30 states and 98 symbols is fairly accurate in recognizing the correct gesture and seldom confuse one for another. The few errors that do occur are due to the system sometimes being unable to recognize a movement as one it has been trained to learn. This problem is a lot of times due to the user not performing the action within a second, particularly for 'swing.' Other times, the user needs to be rather precise in where relative to the body they perform the action, due to the way in which we defined our features.

C. Additional Gestures

To include more variety in our gesture library, we included two more actions, a 'fist pump' and a 'circle' drawn in midair. The 'fist pump' is an action that mainly only spans Y-coordinates, whereas the circle spans both X and Y at length.

TABLE IV. Gesture Recognition Rates (States=30, Symbols=256)

True Labels	Detected Actions						
	Punch	***Slap***	***Swing***	***Wave***	***Fist***	***Circle***	***False***
Punch	93.55	0	0	0	0.81	0	5.65
Slap	0	99.14	0	0	0	0	0.86
Swing	0	0	99.30	0	0	0	0.70
Wave	0	0	0	94.44	0	0	5.56
Fist	1.0	0	0	0	71.0	0	28.0
Circle	0	0	12.75	0	0	58.82	28.43
False	3.41	17.61	0	21.02	4.55	1.14	52.27

It can be seen from this new data that recognition remains very good for the baseline gestures, but less so for our new additions. This may be due to there being less variance in the training/validation data for the fist pump and circle, as they were both trained by the same individual (whereas the baseline involved the efforts of the entire team, at least). Secondly, it is possible that the 'negative' data, which did involve some random movements, inadvertently contained fist pump and circle actions, as this data tried avoiding the baseline gestures but was collected before we made the two new inclusions. Otherwise, it makes sense that 'swing' siphoned some of 'circle's recognition, as swing shares roughly half of circle's gesture.

VIII. Conclusion

From this project, we may conclude that HMMs are an effective and efficient method of both recognizing and classifying human gestures. To further improve this concept, we may try to build a more sophisticated feature set besides the simple normalized XYZ coordinates of the right hand that we currently use. This may allow greater differentiation between each action. In addition, it would be interesting to add many more Kinect data points and allow our classifier to recognize more complicated gestures involving the entire body. We may also extend our recording time beyond a second. This, again, allows for more complex recognitions, but also complicates the process by testing the system's ability to recognize actions of different duration.

We may possibly improve our results a lot by recruiting many more people to perform actions for us, merely adding more variance into our dataset, preventing trained models that are too constrained.

Finally, it may be desirable to further optimize the HMM training and expectation-maximization algorithms to allow the system to give near instant feedback, instead of having to wait a delay of approximately one second. This would make it truly viable for an HCI application.

Acknowledgment

We would like to thank Professor Andrew Ng for teaching such a large CS229 class, as well as all the TAs for helping us out throughout the quarter.

REFERENCES

[1] L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” Proceedings of the IEEE, vol. 77, no. 2, February 1989.

[2] Z. Ghahramani. “EM for Hidden Markov Models for Discrete-Valued Observations” [Online]. Available: http://mlg.eng.cam.ac.uk/zoubin/software.html

[3] J. C. Hall. (2011, Dec 22). “How to Do Gesture Recognition With Kinect Using Hidden Markov Models (HMMs)” [Online]. Available: http://www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-hmms/

[4] 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, October 1999.

[5] Y. Wang, C. Yang, X. Wu, S. Xu and H. Li, “Kinect Based Dynamic Hand Gesture Recognition Algorithm Research,” 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, 2012.