Kinect Gesture Recognition for Interactive System

Hao Zhang, WenXiao Du, and Haoran Li

***Abstract*— Gaming systems like Kinect and XBox always have to tackle the problem of extracting features from video data sets and classifying the body movement. In this study, reasonable features like human joints positions, joints velocities, joint angles and joint angular velocities are extracted. We used several machine learning methods including Naive Bayes, Support Vector Machine and Random Forest to learn and classify the human gestures. Simulation results and the final confusion matrix show that by combining delicately preprocessed data sets and Random Forest methods, the F-scores of the correct predictions can be maximized. Such methods can also be applied in real-time scenarios.**

***Index Terms*— Kinect gesture recognition, SVM, random forest, naive bayes**

I. Introduction

The last few years witnessed a great increase of the prevalence of body-movement based interface. Among all the modes, touchless body movement interface has obviously caught more attentions since it can offer more friendly user experience. In the application area, traditional video cameras can be used to caption the body movements to enable interactive system. However, due to the limitation of the usage on applications, such technology did not own a large user set. On the other hand, some gaming systems like Microsoft Kinect, Nitendo Wii and Sony Playstation, have made such touchless body-movement based interface more popular and convenient to use by introducing a depth sensor to capture video data in 3D.

In the paper, we focus on gesture recognition of such interactive system. In this sense, by analyzing and training the video data from Microsoft Kinect, we would design a machine learning method to classify the actual movement captured from the Kinect video with high accuracy. When a movement is finished, it would automatically classify the movement into one of the 6 gestures $asshowninFig.1$ to further implement the built-in functions in Kinect.

The remainder of the paper is organized as follows. Section II describes the collection of data sets. The method of extracting features from the data sets is detailed in Section III. Section IV describes the learning process by SVM. And in Section V, the learning process of random forest is explained. Section VI provides simulation performance to compare different learning methods and analysis the learning results. Finally, we conclude the paper in Section VII.

The authors are with the Department of Electronic Engineering at Stanford University, CA, 94305, United States. Email: {hzhang22, wxdu, aimeeli}@stanford.edu.

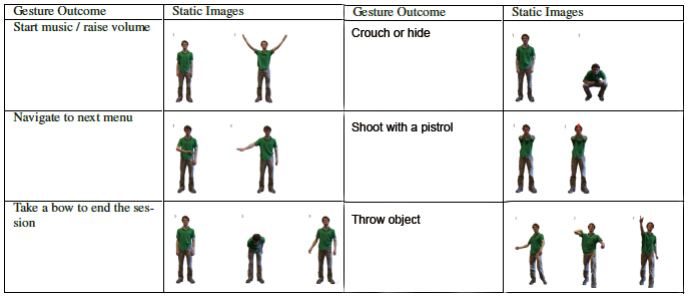

Fig. 1. Input Gestures.

II. The Kinect 3D data sets

The data sets we are using in this paper are provided by Microsoft Research Cambridge-12 (MSRC-12). Microsoft has also provided a programming toolkit of *Kinect for Windows Software Development Kit Beta*. The SDK offers the capability to track the skeleton 3D model and obtain the data of joints positions [5]. The data sets are collected from 30 people performing 6 different gestures $(Fig.\ 1)$ with approximately 3 hour 34 minutes. More specifically, with a sample rate of 30Hz, it is composed of 30 sequences, 385732 frames, with a total of 3054 gesture instances. Mathematically speaking, there are 6 gestures with about 50 sequences each. And each sequence is composed of about 800 frames constituting approximately 10 gesture instances. Labeling of the data is automatically done by the related 'tagstream' files.

There are two kinds of gesture types in the data sets: Iconic gestures - those imbue a correspondence between the gesture and the reference, and Metaphor gestures - those represent an abstract concept. A table of the gesture is given on the top of the next page.

TABLE I
GESTURE CLASSIFICATION

Iconic Gestures	Number of Instance	Metaphoric Gestures	Number of Instance
Crouch or Hide	500	Start Music/Raise Volume	508
Throw an Object	515	Navigate to Next Menu	522
Kick	502	Take a Bow to End Music	507

III. FEATURE EXTRACTION

In the original data set, each frame of gestures is recorded as the absolute position of 20 joints of human body in xyz-coordinates, 60 data total per frame. Meanwhile, in each sequence, a single gesture has been repeated for several times. Therefore, some preprocess should be applied to the raw data sets in order to form the proper training examples and informative feature vectors. For time t, we derive a feature vector of $\phi_t = \phi(x_{t:t-l})$from the last l observations x_t to x_{t-l}. Indicated by the paper [2] that when $l = 35$, the real-time performance can be achieved. Therefore we define every 35 frames as a training example. Even though we cannot precisely subsume a gesture instance into every 35 frames, the relative displacement of adjacent frames within a training example can also provide enough information to make the classification. For each pair of adjacent frames, 4 kinds of factors can be considered as the possible components of a feature vector, which are:

- 3 xyz-coordinates per joint, 60 total
- 3 xyz-velocities per joint, 60 total
- 35 joint angles
- 35 joint angular velocities

The skeletal structure of human body is shown in Fig. 2.

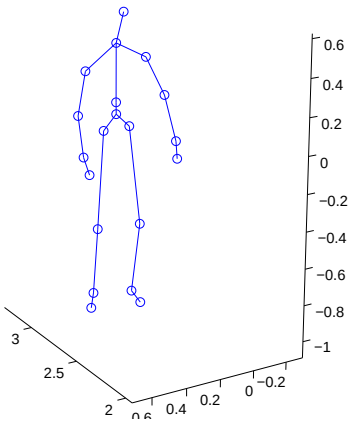

Fig. 2. Skeletal Structure of Human Body.

Here are some details about each components. The xyz-velocities are straightforwardly defined as the difference between xyz coordinates of corresponding joints between each pair of adjacent frames. The joint angle is simply the angle between the two segments on either side of the joint. For the joint like shoulder-center, which has multiple segments around it, we compute the angles pairwise. Besides only extracting angles between adjacent segments, we put an imaginary joint in (0, 0, 0) in world coordinates, which is the location of camera. This is helpful because all other angles are rotation invariant, but this one allows you to distinguish changes with respect to the camera, for example, when we want to recognize whether a person throws an object to the left or to the right of the camera. The joint angular velocity is the rate of change of joint angle, which is computed by the difference of the corresponding joint angle in each pair of adjacent frames.

IV. SVM CLASSIFICATION

After preprocessing the feature vectors, we first use a SVM tool [7] to train our data. The link to the SVM package is from [3]. We randomly divided the given data into 70% for training and 30% for testing. The following five steps are gone through in the process of training and testing:

A. Define Feature Vector

In this step, we would determine the composition of feature vector. As described in the previous section, four classes of features are considered. We run the forward search on these feature classes and obtain the following results. The feature selection is run on the data sets with 60 sequences for training and 12 sequences for test.

The result of feature selection is shown in the table on the next page:

We choose xyz-velocity, joint angle and joint angular velocity as the feature vector for each frame, and the dimension is $60 + 35 + 35 = 130$. As described in the previous section, 35 frames are included in each training example. Thus the dimension of a feature vector of a training example is 4420.

B. Data Scaling

Scaling before applying SVM is of great significance. By scaling we can avoid the scenario that attributes in

TABLE II
FEATURE CLASS SELECTION

Feature in Use	Feature Fixed	Accuracy	Real Proportion
1	ϕ	40.3794%	149/369
2	ϕ	23.5772%	87/369
3	ϕ	65.3117%	241/369
4	ϕ	23.5772%	87/369
3+1	3	56.6396%	209/369
3+2	3	73.9837%	273/369
3+4	3	57.4526%	212/369
3+2+4	3+2	79.4038%	293/369
3+2+1	3+2	63.4146%	234/369
3+2+4+1	3+2+4	63.1436%	233/369

* *1 is xyz position, 2 is xyz velocity, 3 is joint angle,4 is joint anglular velocity*

larger numeric range dominate the ones in small numeric range. And it can alleviate the mathematical calculation workload.

In this paper, the features are linearly scaled into a range of $[-1, +1]$. By libsvm, the improvement on the accuracy can be seen:

TABLE III
DATA SCALING

Mode	Accuracy	Real Proportion
Before Scaling	62.496%	1938/3101
After Scaling	63.1732%	1959/3101

From the table, it can be observed that by preprocessing of scaling on the data, the accuracy can improved by about 3%. Therefore, in the afterwards experiment, we will use the scaled data in order to achieve a better prediction.

C. Kernel Selection

We tried three kernels in our process of kernel selection: linear kernel, polynomial kernel and radial basis function (RBF) kernel. The accuracy of prediction using the three kernels after scaling is in the table below:

TABLE IV
KERNEL SELECTION

Kernel	Accuracy	Real Proportion
Linear	61.0448%	1893/3101
Polynomial	33.0539%	1025/3101
RBF	63.1723%	1959/3101

Note that RBF kernel achieves the highest accuracy. This kernel nonlinearly maps samples into a higher dimensional space and it has fewer hyperparameters than polynomial kernel which influences the complexity of model selection. From the result, we can see that the polynomial kernel is overfitting. Linear kernel also provides us with a comparable accuracy due to the large number of features. But since RBF kernel gives us a higher accuracy, we determined to use RBF kernel.

D. Parameter Selection

There are two parameters for RBF kernel: C and γ, which is not known beforehand, thus some kinds of parameter search must be done. The goal is to identify good (C, γ) so that the classifier can accurately classify unknown data. A common strategy is to use n-fold cross-validation, which divides the training set into n subsets of equal size and sequentially one subset is tested using the classifier trained on the remaining n-1 subsets. We use a grid-search on C and γ by cross-validation. Various pairs of (C,γ) values are tried and the one with best cross-validation accuracy is picked. After running the parameter selection script, we got the parameter $C = 32$ and $\gamma = 0.0078125$ with an accuracy of 90.2439%. We used this parameter in later training.

E. Final Result

After defining the feature vectors, scaling the data, choosing the most accurate kernel and got the parameters, we used svm-train and svm-predict again with the chosen kernel and parameters, we finally got an accuracy of prediction of 67.3331% (2088/3101), which is acceptable.

V. RANDOM FOREST LEARNING METHOD

As in [6], Random Forest works as described below. After given a set of training examples, a random forest is created with H random decision trees. And for the $k - th$ tree in the random forest, a random vector ϕ_k is generated independently of the past random vectors $\phi_1, ..., \phi_{k-1}$. This vector ϕ_k is then used to grow the trees resulting in a classifier $h_k(x, \phi_k)$ where x is the feature vector. For each tree, a decision function splits the training data that reach a node at a given level in the tree [4].Then each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

The resulting forest classifier H is used to classify a given feature vector by taking the mode of all the classifications made by the tree classification $h \in H$ for all the forest.

A. Growing Trees

The following approach is similar to that of [1]. At test time t, we derive a vector $\phi_t = \phi(x_{t:(t-l+1)}) \in \mathbb{R}^d$ from the last l observations x_t to x_{t-l+1}. According to what we have described in the SVM method, the training examples are set to $l = 35$ frames, which obtains $d = 4420$ features. The feature vector ϕ_t is evaluated by a set of M decision trees in the random forest, where simple test

$$f_\omega : \mathbb{R}^d \rightarrow \{\text{left,right}\} \tag{1}$$

are performed recursively at each node until a leaf node is reached. In our experiment, the number of random decision trees is set to be $M = 300$. The parameters $\omega \in \Omega$ of each tests are determined separately during the training phase, and the determination process is described below.

For each tree $m = 1, ..., M$, it produces one class decision y_t^m and the posterior class distribution

$$p(y_t = a|x_{t-l+1:t}) := \frac{1}{M}\sum_{m=1}^{M} I(y_t^m = a) \tag{2}$$

over gesture class $\mathcal{A}$. At the same time, we have to add an extra class "None" to indicates whether a gesture has been recognized. If for a gesture class $a \in \mathcal{A}$ we have $p(y_t = a|x_{t-l+1:t}) \geq \delta$, we can then determined the gesture being detected at current time t. We used a fixed value $\delta = 0.16$ [2] for all the random forest experiments.

B. Random Forest Training and Predicting

For the training, we use approximately 70% of all the observations together with the action point annotations for a set of N sequences, where the $n-th$ sequence is an ordered list $(x_t^n, y_t^n)_{t=1,...T_n}$. Our goas is to learn a set of M decision trees that classify the action points in these sequences correctly by means of (2). Then for the decision parts, we use simple "decision stump" tests [6] with $\omega = (i, h), 1 \leq i \leq d, h \in \mathbb{R}$,

$$f_{(i,h)}(\phi_t) = \begin{cases} \text{left} & \text{if}[\phi_t]_i \leq h \\ \text{right} & \text{otherwise} \end{cases} \tag{3}$$

Standard information gain criterion and training procedure are used in the method. We greedily select a split function $f_{(i,j)}$ for each node in the decision tree from a set of randomly generated proposal split functions. The tree is then grown until the node is pure. In a sense, all training examples assigned to that node have the same label.

After all the decision trees are finally formed, the random forest is well set. And we can use the random forest model to make classifications by simply putting the test examples into the random forest.

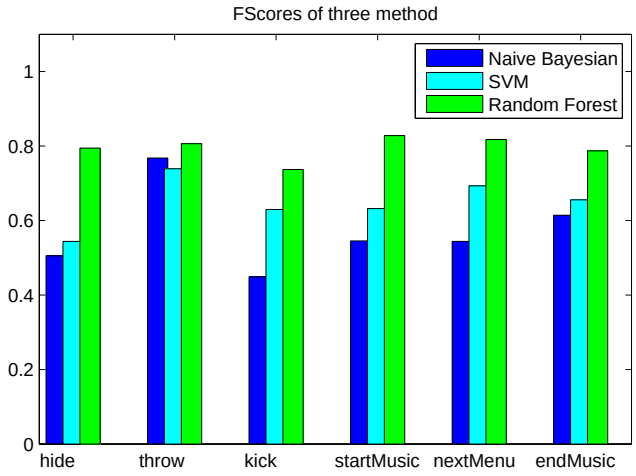

Fig. 3. Fscore of three method.

VI. Simulation Results and Performance Assessment

The accuracy of the three algorithms are summarized in table below:

TABLE V
Accuracy Comparison

Algorithm	Accuracy
Naive Bayes	56.33%
SVM	67.33%
Random Forest	80.69%

In the final performance assessment, the whole data set is randomly split into two parts, 210 sequences for training and 70 sequences for testing. We use F-score and confusion matrix to evaluate the performance of each algorithm. The F-score of the three methods is Fig. 3 and the confusion matrix of the three methods is Fig. 4.

The accuracy of the prediction by SVM is 67.33%. From the confusion matrix, we can see that the performance on recognizing gesture 1, 2, 5 is relatively better than on other gestures. However, since other gestures can also easily be misclassified into gesture 1, the recall of gesture 1 is low, which makes its Fscore great lower than its accuracy.

We can see that SVM preforms worse than Random Forest Algorithm, probably because there are too many features in each training example. Although the feature class selection has been conducted on the data set, the over-fitting still slightly exists. Such guess can also be confirmed by the fact that the final accuracy on the whole data set is poorer than the accuracy when the algorithm is conducted on the small data set in the feature selection step.

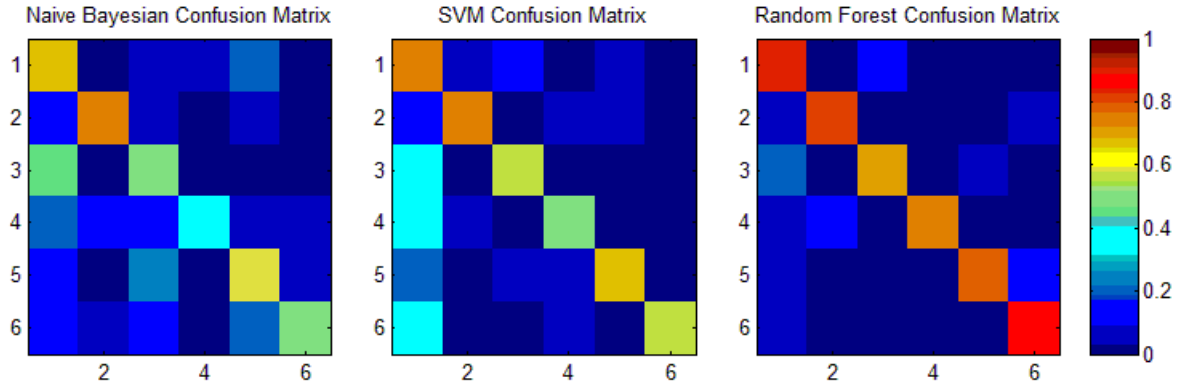

Fig. 4. Confusion Matrix.

We also implemented Naive Bayes as a benchmark to compare with the results got by SVM and Random Forest. At first we used normal distribution to model the data and created class variable for training taking 1000 distinct levels. Then we have a train category vector that defines which class the corresponding row of training belongs to. We used Naive Bayes classifier with the multinomial event model and Laplace transform to classify each gesture. Then we compared with the actual category and got the confusion matrix and F-score. The accuracy of the prediction by Naive Bayes is 56.33%. From the confusion matrix, we can see that the performance of predicting gesture 1, 2, 5 is better compared to the other gestures. But overall, the performance is worse than that of SVM and Random Forest. Since Naive Bayes discretizes the feature values and instead uses a class variable, it loses some accuracy in the process of discretization, which is reasonable.

The performance of the random forest is the best among three algorithms. The accuracy by random forest can reaches as high as 80.69%. Meanwhile the F-scores of all six gestures are higher than other methods.

VII. Conclusion

In this report, we have studied the methods to pre-processing the given data sets to find the best features. And then, in SVM process, after feature class is selected, scaling, Kernel selection, RBF kernel parameter selection, we have decided the final SVM model. And the F-scores of every classs in the SVM model can be seen on Fig. 3. Then, we have tried random forest method, after growing a forest with 300 decision trees. The F-score of every class in the model has increased a lot. As a benchmark, a naive bayesian model was also simulated. By comparing all three models, it can be found that by combining delicately preprocessed data sets and Random Forest methods, the F-scores of the correct predictions can be maximized. In a sense, the Kinect system can thus differentiate one human gesture from the other trained gestures with high accuracy.

In the future work, a more accurate feature selection on the data sets can be conducted. If we are given more powerful computation resources, we would like to experiment with a larger data sets, and are capable of conducting the large computation required delicate feature selection. Meanwhile, another improvement in the future can be focused on the vision part, which is the method to extract joints data sets from the kinect video. It is indeed a challenging task.

References

[1] Gabriele Fanelli Angela Yao, Juergen Gall and Luc Van Gool. Does human action recognition benefit from pose estimation?, 2011. http://dx.doi.org/10.5244/C.25.67.

[2] Simon Fothergill, Helena Mentis, Pushmeet Kohli, and Sebastian Nowozin. Instructing people for training gestural interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1737–1746, New York, NY, USA, 2012. ACM.

[3] Simon Fothergill, Helena M. Mentis, Pushmeet Kohli, and Sebastian Nowozin. Instructing people for training gestural interactive systems. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *CHI*, pages 1737–1746. ACM, 2012.

[4] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P.H.S. Torr. Randomized trees for human pose detection. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8, june 2008.

[5] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *In In CVPR, 2011. 3*.

[6] Leo Breiman Statistics and Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[7] www.csie.ntu.edu.tw/∼cjlin/libsvm/.