Recognition and Segmentation of 3-D Human Action Using HMM and Multi-class AdaBoost*

Fengjun Lv and Ramakant Nevatia

University of Southern California,
Institute for Robotics and Intelligent Systems,
Los Angeles, CA 90089-0273
{flv, nevatia}@usc.edu

Abstract. Our goal is to automatically segment and recognize basic human actions, such as stand, walk and wave hands, from a sequence of joint positions or pose angles. Such recognition is difficult due to high dimensionality of the data and large spatial and temporal variations in the same action. We decompose the high dimensional 3-D joint space into a set of feature spaces where each feature corresponds to the motion of a single joint or combination of related multiple joints. For each feature, the dynamics of each action class is learned with one **HMM**. Given a sequence, the observation probability is computed in each HMM and a weak classifier for that feature is formed based on those probabilities. The weak classifiers with strong discriminative power are then combined by the Multi-Class AdaBoost (**AdaBoost.M2**) algorithm. A dynamic programming algorithm is applied to segment and recognize actions simultaneously. Results of recognizing 22 actions on a large number of motion capture sequences as well as several annotated and automatically tracked sequences show the effectiveness of the proposed algorithms.

1 Introduction and Related Work

Human action recognition and analysis has been of interest to researchers in domains of computer vision [1] [2] [9] [3] [10] [12] for many years. The problem can be defined as: given an input motion sequence, the computer should identify the sequence of actions performed by the humans present in the video. While some approaches process the video images directly as spatio-temporal volumes [10] , it is common to first detect and track humans to infer their actions [3] [12]. For finer action distinction, such as picking up an object, it may also be necessary to track the joint positions. The methods may be further distinguished by use of 2-D or 3-D joint positions.

In this paper, we describe a method for action recognition, *given* 3-D joint positions. Of course, estimating such joint positions from an image sequence is a difficult task in itself; we do not address this issue in this paper. We use

* This research was supported, in part, by the Advanced Research and Development Activity of the U.S. Government under contract No. MDA904-03-C1786.

A. Leonardis, H. Bischof, and A. Pinz (Eds.): ECCV 2006, Part IV, LNCS 3954, pp. 359–372, 2006.

data from a Motion Capture (*MoCap*) system[1]or from a video pose tracking system. It may seem that the task of action recognition given 3-D joint positions is trivial, but this is not the case, largely due to the high dimensionality (e.g. 67-D) of the pose space. The high dimensionality not only creates a computational complexity challenge but, more importantly, key features of the actions are not apparent, the observed measurements may have significant spatial and temporal variations for the same action when performed by different humans or even by the same person. Furthermore, to achieve continuous action recognition, the sequence needs to be segmented into contiguous action segments; such segmentation is as important as recognition itself and is often neglected in action recognition research. Our method attempts to solve both the problem of action segmentation and of recognition in presence of variations inherent in performance of such actions. Even though we assume that 3-D positions are given, we present results illustrating the effects of noise in the given data (for positions derived from videos).

Previously reported action recognition methods can be divided into two categories with respect to the data types that they use: those based on 2-D image sequences, e.g. [2] [9] [3] [10] [12] and those based on 3-D motion streams, e.g. [1] [7]. The above 2-D approaches can be further divided by the image features they use: those based on object contours [2] [12], those base on motion descriptor such as optical flow [3] or gradient matrix [10] and those base on object trajectories [9]. A 3-D approach has many advantages over a 2-D approach as the dependence on viewpoint and illumination has been removed. Nonetheless, many algorithms use a 2-D approach because of the easy availability of video inputs and difficulty of recovering 3-D information.

In the above 3-D approaches, [1] decomposes the original 3-D joint trajectories into a collection of 2-D projected curves. Action recognition is based on the distances to each of these curves. However, the predictors they use are static constructs as the correlation between 2-D curves are lost after projection. In [7], actions are represented by spatio-temporal motion templates, which correspond to the evolution of 3-D joint coordinates. As matching results can be affected by temporal scale change, each template is resampled and multiple-scale template matching is performed.

The approaches cited above (except for [7]) assume that the given sequence contains only one action performed throughout the length of the sequence; it is not clear how and whether they could be extended to segment a video containing sequence of actions and recognize the constituent actions. In [7] action segmentation is obtained to some extent by classifying action at each frame by considering a fixed length sequence preceding it; such segmentation is inaccurate at the boundaries and over-segmentation may occur. Explicit action segmentation is considered in the context of complex actions consisting of a sequence of simpler actions but only when a model of the sequence is available, such as in [5]. We consider a sequence of actions where such models are not

[1] Part of data comes from mocap.cs.cmu.edu, which was created with funding from NSF EIA-0196217.

known and actions may occur in any sequence (though some sequences may be eliminated due to kinematic constraints; we do not consider such constraints in this work).

Our approach is based on 3-D joint position trajectories. The actions we consider are *primitive* components that may be composed to form more complex actions. The actions are grouped according to the set of body parts that are related to the action. In our approach, the high dimensional space of 3-D joint positions is decomposed into a set of lower dimensional feature spaces where each feature corresponds to the motion of a single joint or combination of related multiple joints. For each feature, the dynamics of one action class is learned with one continuous Hidden Markov Model (HMM) with outputs modeled by a mixture of Gaussian. A weak classifier for that feature is formed based on the corresponding HMM observation probabilities. Weak classifiers of the features with strong discriminative power are then selected and combined by the Multi-Class AdaBoost (**AdaBoost.M2**) algorithm to improve the overall accuracy. A dynamic programming-based algorithm is applied to segment and recognize actions simultaneously in a continuous sequence.

To our knowledge, there has been little work done in computer vision that integrates HMM with AdaBoost. In [13], an integration called "boosted HMM" is proposed for lip reading. Their approach is different from ours in that they use AdaBoost first to select frame level features and then use HMM to exploit long term dynamics. This does not suit the action recognition problem well because the full body motion is much more complex than the lip motion. Without the dynamic information, the static features tend to cluster in the feature space and thus can not discriminate different actions well; their combination (using AdaBoost) is unlikely to alleviate the problem much. Another difference is that [13] works with pre-segmented sequences only.

We show the results of our system on a large collection of motion capture sequences with a large variety of actions and on some hand annotated as well as automatically tracked video sequences. The recognition results are very good and the method demonstrates tolerance to considerable amount of noise. We also compare performance with our earlier work [7] which was based on a simpler algorithm. Most importantly, the new approach can take a data stream containing a sequence of actions and then segment and recognize the component actions which is not possible with the earlier approach.

2 The Dataset and Feature Space Decomposition

We collected 1979 MoCap sequences consisting of 22 Actions from Internet. We also generated some sequences of 3-D joint position from a 3-D annotation software [6] and from an automatic 3-D tracking software [6]. The generated data are much less accurate compared with MoCap. They are used in testing only to show that our algorithm can work on real data and that training on MoCap data transfers to video sequences; we do not claim to have solved the tracking problem as well.

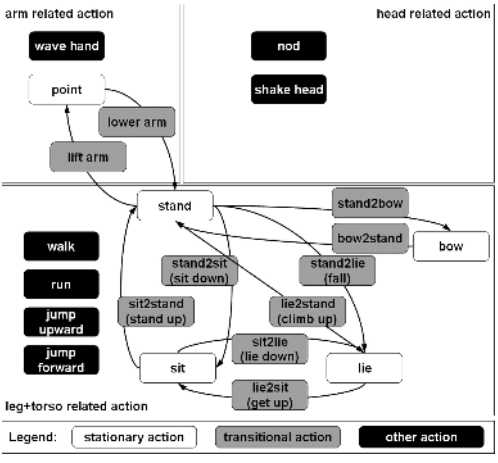

Fig. 1. Categorization of actions that need to be recognized

Actions in these videos can be grouped into 3 categories according to the involved primary body parts: {*leg+torso, arm, head*}. The categorization is illustrated in Fig.1. Actions in the same group are mutually exclusive to each other, but actions from different groups can be recognized simultaneously. This allows us to execute logical queries such as *find the sequence in which the subject is walking while his head is nodding.*

Actions can also be classified based on whether the primary body parts move or not. We view a *stationary* pose, *e.g. stand or sit* (*white blocks* in Fig.1) as a special type of action, with the constraint that duration of the action should be long enough (longer than some threshold). The *transitional* actions (*gray blocks*) transit from one stationary pose to another. The remainder (*black blocks*) consist of *periodic* actions (*e.g. walk, run*) and other actions.

Different joint configurations (i.e. number of joints/bones, joint names and joint hierarchy) are unified to one that consists of 23 joints. The joint positions are normalized so that the motion is invariant to the absolute body position, the initial body orientation and the body size.

Since there are 23 joints and each has 3 coordinates (only y coordinate is used for hip, the root joint), the whole body pose at each frame can be represented by a 67-D vector (called a *pose vector*). We performed some experiments to evaluate the effectiveness of using the 67-D pose vector on a simpler subset of action classes Walk, Run, Stand, Fall, Jump, Sit down. We first used a Bayesian network to classify static pose and found the classification accuracy to be less than 50%. Then, we trained and tested a 3-state continuous HMM (the same as one described in section 3.1) and found the accuracy to be still low (below 60%). These experiments do not conclusively prove that using the full pose vector is undesirable but it is reasonable to think that relevant information can get lost in the large pose vector.

Rather than enumerate all combinations of different components in the pose vector (as in [1]), we design feature vectors, called just *features* from now on, such that each feature corresponds to the pose of a single joint or combination of multiple joints. Following is the list of different types of features that are included.

Type 1: one coordinate of a joint, e.g. the vertical position of the hip
Type 2: coordinates of each non-root joint
Type 3: coordinates of 2 connected joints, e.g. neck and head
Type 4: coordinates of 3 connected joints, e.g. chest, neck and head
Type 5: coordinates of one pair of symmetric nodes, e.g. left hip and right hip
Type 6: coordinates of all leg and torso related joints
Type 7: coordinates of all arm related joints

Features are designed in this way based on our analysis of the actions and the features that can distinguish them. Different types characterize different levels of dynamics of an action. For example, Type 2,3 and 4 corresponds to joint position, bone position and joint angle, respectively. Type 5 features are useful for detecting periodic motions and type 6 and 7 features provide an overall guidance for recognition. In total we have 141 features.

3 Integrated Multi-class AdaBoost HMM Classifiers

We now describe our action recognition methodology. It consists of a combination of HMM classifiers, which are treated as *weak* classifiers in the AdaBoost terminology, and whose outputs are combined by a multi-class AdaBoost algorithm (**AdaBoost.M2**). We first describe the HMM classifiers and then their combination. The issue of segmenting the pose sequence into separate actions is addressed separately in section 4 later.

3.1 Learning Weak Classifiers Using HMMs

We choose a hidden Markov model (HMM) to capture the dynamic information in the feature vectors as experience shows them to be more powerful than models such as Dynamic Time Warping or Motion Templates. An HMM is defined by states, transition probabilities between them and probabilities of outputs given a state. Well known algorithms [8] are available to answer the following questions:

1. How to compute $P(O|\lambda)$, the probability of occurrence of the observation sequence $O{=}O_1O_2...O_T$ given model parameters λ? This problem can be solved by the **Forward-Backward** procedure.
2. How to select the best state sequence $I{=}i_1i_2...i_T$ such that $P(O, I|\lambda)$ is maximized? This problem can be solved by the **Viterbi** algorithm.
3. How to learn model parameters λ given O such that $P(O|\lambda)$ is maximized? This problem can be solved by the **Baum-Welch** algorithm.

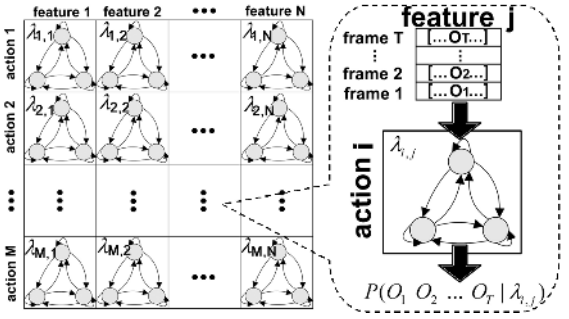

Fig. 2. The matrix of HMMs. Each HMM has 3 hidden states. Each state contains a 3-component mixture of Gaussian. Once $\lambda_{i,j}$, the parameters of $\text{HMM}_{i,j}$ is learned, the probability of the observation sequence $O_1O_2...O_T$ is computed using the **Forward** procedure. The action with the maximum probability in the same column is selected.

For the action classification problem, suppose there are M action classes and N features (feature classes). For the j-th feature (j = 1,...,N), we learn one HMM for each action class and the corresponding parameters λ_i, i = 1,...,M. Given one observation sequence O, we compute $P(O|\lambda)$ for each HMM using the **Forward-Backward** procedure. Action classification based on feature j can be solved by finding action class i that has the maximum value of $P(O|\lambda_i)$, as shown in Eq.1.

$$action(O) = \underset{i:i=1,\ldots,M}{\arg\max} \left(P(O|\lambda_i)\right) \tag{1}$$

We call the set of these HMMs and the decision rule in Eq.1 as the weak classifier for feature j (a term used in boosting algorithm literature). These M action classes and N features form an $M \times N$ matrix of HMMs, as shown in Fig.2. We denote by $\text{HMM}_{i,j}$ the HMM of action i (i-th row) and feature j (j-th column) and its corresponding parameters is $\lambda_{i,j}$. The set of HMMs in column j correspond to the weak classifier for feature j.

Recall that there are three action groups based on involved body parts. Therefore, to recognize actions in a specific group, only related features need to be considered. In other words, a feature can only classify related actions. For example, feature neck_{xyz}+head_{xyz} can only classify action "nod" and "shake head". Therefore, there are three matrices of HMMs, corresponding to three action groups. As HMMs in three groups are used in the same way; unless otherwise stated, we do not specify which group that they belong to.

In our system, each HMM has 3 hidden states and each state is modeled by a 3-component mixture of Gaussian. The following parameters of each HMM are learned by the **Baum-Welch** algorithm: (1) Prior probabilities of each state, (2) Transition probabilities between states and (3) Parameters of each state s (s=1,2,3): Mean vector $\mu_{s,m}$, covariance matrix $\Sigma_{s,m}$ and weight $w_{s,m}$ of each mixture component (Gaussian) (m=1,2,3).

The training and classification algorithm of weak classifiers are listed as follows:

Algorithm 1. Training of weak classifiers

1. Given Q training samples $\langle (x_1, y_1), ..., (x_Q, y_Q) \rangle$ where x_n is a sequence with action label y_n, $y_n \in \{1, ..., M\}$, n=1,...,Q, M is the number of action classes
2. Divide the Q samples into M groups such that each group contains samples with the same label.
3. for j=1 to N (N is the number of features)
   for i=1 to M
   3.1 Crop the training samples in group i, such that they contain only coordinates that belong to feature j
   3.2 Train $\text{HMM}_{i,j}$ using **Baum-Welch** algorithm

Algorithm 2. Classification algorithm based on weak classifier j

1. Given an observation sequence O=$O_1O_2...O_T$
2. for i=1 to M, Compute $P(O|\lambda_{i,j})$
3. Return $\underset{i:i=1,...,M}{\arg\max} (P(O|\lambda_{i,j}))$

Both **Forward** and **Baum-Welch** algorithm need to compute $P(O_t|s_t = s)$, the probability of observing O_t given that state s at time t. Unlike a discrete HMM, a continuous HMM uses a Probability Density Function (PDF) to estimate $P(O_t|s_t = s)$. This is because no point has a probability in a continuous distribution, only regions do. For the Gaussian mixture model used here, $P(O_t|s_t = s)$ is computed as follows:

$$\sum_{m=1}^{3} \left(w_{s,m} \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_{s,m}|^{\frac{1}{2}}} e^{-\frac{1}{2}(O_t - \mu_{s,m})\Sigma_{s,m}^{-1}(O_t - \mu_{s,m})^T} \right) \tag{2}$$

In practice, Log-likelihood $log(P(O_t|s_t = s))$ is used to avoid numerical underflow. Another consideration is that the probability of occurrence of the observation sequence $O_1O_2...O_T$ tends to decrease exponentially as T increases. But this causes no problem here because for feature j, the probability computed in each of $\text{HMM}_{i,j}$ $(i = 1, ..., M)$ decreases comparably.

The complexity of **Algorithm 2** is $\text{O}(MN_{st}{}^2T)$, where N_{st} is the number of states in HMM. Deciding automatically the appropriate value of N_{st} is difficult and therefore in practice, it is usually specified in advance. Our experiments show that an HMM of 3 states with a 3-component mixture Gaussian can capture rich dynamic information in the actions and can achieve desired high classification rate. So the complexity of **Algorithm 2** is approximately $O(MT)$.

The complexity of **Algorithm 1** is $\text{O}(NN_{it}N_{st}{}^2L_{all})$, where N_{it} is the number of iterations in **Baum-Welch** algorithm and L_{all} is the total length of training samples of all action classes. In our experiment, **Baum-Welch** algorithm usually converges in less than 5 iterations. So the complexity of **Algorithm 1** is approximately $O(NL_{all})$.

3.2 Boosting Classifiers Using AdaBoost.M2

Experiments show that individual learned HMM classifiers have reasonably good performance; for example, feature (all leg and torso related joints) alone can correctly classify 62.1% of 16 leg and torso related action classes. However, we expect much better performance from the final classifier. This can be done by combining HMM classifiers, considered to be weak classifiers. This is made possible due to the fact that each weak classifier has different discriminative power for different actions.

Inspired by success of boosting methods for many problems, particularly face detection of Viola and Jones [11], we use the AdaBoost [4] algorithm to combine results of weak classifiers and to discard less effective classifiers to reduce the computation cost. This algorithm works in an iterative way such that in each iterationthe newly selected classifiers focus more and more on the difficult training samples. In this paper, we use **AdaBoost.M2** [4], the multi-class version of AdaBoost. Some limitations of **AdaBoost.M2** for feature selection were stated in [13]; we believe that those limitations hold only when the weak classifiers have very limited discriminative power.

We rephrase the **AdaBoost.M2** algorithm here to accommodate our specific problem.

Algorithm 3. AdaBoost.M2 for action classification

1. Given Q training samples $\langle (x_1, y_1), ..., (x_Q, y_Q) \rangle$ where x_n is a sequence with action label y_n, $y_n \in \{1, ..., M\}$, n=1,...,Q
2. Train weak classifiers using **Algorithm 1** in section 3.1
3. Test these Q samples on each $\text{HMM}_{i,j}$, record the value $P(x_n|\lambda_{i,j})$, i=1,...,M, j=1,...,N, n=1,...,Q
4. Let $B = \{(n, y) : n \in \{1, ..., Q\}, y \neq y_n\}$ be the set of all mislabels; Let $D^{(1)}(n, y) = 1/|B|$ for $(n, y) \in B$ be the initial distribution of mislabels
5. for k=1 to K (K is the number of iterations)
   5.1 Select a weak classifier h_k that has minimum pseudo-loss $\varepsilon_k = \frac{1}{2} \sum_{(n,y)\in B} D^{(k)}(n, y)(1 - P(x_n|\lambda_{y_n,h_k}) + P(x_n|\lambda_{y,h_k}))$
   5.2 Set $\beta_k = \varepsilon_k/(1 - \varepsilon_k)$
   5.3 Update $D^{(k)}$: $D^{(k+1)}(n, y) = \frac{D^{(k)}(n,y)}{Z_k} \beta_k^{\frac{1}{2}(1+P(x_n|\lambda_{y_n,h_k})-P(x_n|\lambda_{y,h_k}))}$ where Z_k is normalization constant so that $D^{(k+1)}$ will be a distribution
6. Let $f = \sum_{k=1}^{K} (\log \frac{1}{\beta_k}) P(x|\lambda_{y,h_k})$. Return the final classifier $h(x) = \arg\max_{y\in\{1,...,M\}} (f)$ and likelihood $H(x) = \max_{y\in\{1,...,M\}} (f)$

The idea of this algorithm can be interpreted intuitively as follows:(1) $\log \frac{1}{\beta_k}$ is the weight of the selected classifier h_k. Intuitively, as $P(x_n|\lambda_{y_n,h_k})$ increases (which means h_k labels x_n more accurately), the pseudo-loss ε_k decreases and consequently $\log \frac{1}{\beta_k}$ increases. So in each iteration the new selected classifier has the strongest discriminative power given current $D^{(k)}$. (2)$D^{(k)}$, the distribution

of mislabels, represents the importance of distinguishing incorrect label y on sample x_n. As $P(x_n|\lambda_{y,h_k})$ increases and $P(x_n|\lambda_{y_n,h_k})$ decreases (which means that h_k labels x_n less accurately), $D^{(k)}$ increases because $\varepsilon_k \leq \frac{1}{2}$ and thus $\beta_k \leq 1$. By maintaining this distribution, the algorithm can focus not only on the hard-to-classify samples but also on the hard-to-discriminate labels. This is the major improvement over **AdaBoost.M1**, the first version of multi-class AdaBoost [4].

Care needs to be taken when applying **AdaBoost.M2** on continuous HMMs because it is critical in **AdaBoost.M2** that the value of hypotheses generated by weak classifiers not exceed 1 so that the pseudo-loss ε_k is in the range of [0,0.5]. However, for a continuous HMM, the observation probability $P(x_n|\lambda_{i,j})$ computed by the **Forward** procedure is based on a Gaussian function, as shown in Eq.2. Keep in mind this is the probability *density* function. Therefore, $P(x_n|\lambda_{i,j})$ can be greater than 1 in practice. If this occurs, all $P(x_n|\lambda_{i,j})$ computed in step 3 of **Algorithm 3** will be normalized by a scale factor such that the maximum of all $P(x_n|\lambda_{i,j})$ does not exceed 1. (Theoretically, $P(x_n|\lambda_{i,j})$ can be an infinite number, which indicates the sample should be definitely labeled as action i, but this does not occur in our experiments.)

Results show that after combining 15 weak classifiers by the boosting procedure, the final classifier achieves a classification rate of 92.3% on the leg and torso related actions, showing the effectiveness of the algorithm.

4 The Segmentation Algorithm

Action classification method described in section 3 assumes that each input sequence belongs to one of the action classes. To achieve continuous action recognition, a (long) sequence needs to be segmented into contiguous (shorter) action segments. Such segmentation is as important as recognition itself.

Here is the definition of segmentation: Given an observation sequence $O = O_1O_2...O_T$, a segmentation of O can be represented by a 3-tuple $S(1,T)$= (N_S, s_p, a_p).

- N_S: the number of segments, $N_S \in \{1, ..., T\}$
- s_p: the set of start time of each segment, $s_p \in \{1, ..., T\}$, $p = 1, ..., N_S$, s_1 is always 1 and we add an additional point s_{N_S+1}=T+1 to avoid exceeding the array boundary
- a_p: the corresponding action labels of each segment, $a_p \in \{1, ..., M\}$

For a sub-sequence $O_{t1}O_{t1+1}...O_{t2}$, we compute the following functions:

$$h(t1,t2) = \arg\max_{y\in\{1,...,M\}} \sum_{k=1}^{K} (\log \frac{1}{\beta_k}) P(O_{t1}...O_{t2}|\lambda_{y,h_k}) \tag{3}$$

$$H(t1,t2) = \max_{y\in\{1,...,M\}} \sum_{k=1}^{K} (\log \frac{1}{\beta_k}) P(O_{t1}...O_{t2}|\lambda_{y,h_k}) \tag{4}$$

$h(t1,t2)$ and $H(t1,t2)$ is the action label and likelihood computed by **Algorithm 3**.

Given a segmentation S of $O_1O_2...O_T$, a likelihood function L is defined as:

$$L(1,T,S(1,T)) = \prod_{p=1}^{N_S} H(s_p, s_{p+1}-1) \tag{5}$$

A maximal likelihood function L^* is defined as:

$$L^*(1,T) = \max_{S(1,T)} L(1,T,S(1,T)) \tag{6}$$

The goal of the segmentation problem is to find the maximal likelihood function $L^*(1,T)$ and the corresponding segmentation $S^*(1,T)$. Enumeration of all possible values of (N_S, s_p, a_p) is infeasible because of the combinatorial complexity. However, the problem can be solved in $O(T^3)$ time by a dynamic programming-based approach: Suppose a sub-sequence $O_{t1}...O_{t2}$ is initially labeled as $h(t1,t2)$ and t is the optimal segmentation point in between (if $O_{t1}...O_{t2}$ should be segmented). If $L^*(t1,t-1)L^*(t,t2)$ is larger than $H(t1,t2)$, then $O_{t1}...O_{t2}$ should be segmented at t. The idea is shown in Eq.7.

$$L^*(t1,t2) = \max_t(H(t1,t2), L^*(t1,t-1)L^*(t,t2)) \tag{7}$$

This recursive definition of L^* is the basis of the following dynamic programming-based algorithm.

Algorithm 4. Segmentation algorithm
/*$L^*(t1,t2)$ abbr. as $L^*_{t1,t2}$, same for other variables*/

1. Given an observation sequence O=$O_1O_2...O_T$. l_{min} is the limit of minimum length of a segment
2. Compute $h_{t1,t2}$ and $H_{t1,t2}$, $t1,t2 \in \{1,...,T\}$ and $t2 \geq t1 + l_{min} - 1$
3. /*too short to be segmented*/
   for l=l_{min} to $2l_{min}-1$
   for $t1$=1 to $T-l+1$
   $L^*_{t1,t1+l-1}$=$H_{t1,t1+l-1}$
   Record the corresponding action labels
4. /*dynamic programming starts here*/
   for l=$2l_{min}$ to T
   for $t1$=1 to $T-l+1$
   $L^*_{t1,t1+l-1}$=max$(H_{t1,t1+l-1}, \max_t(L^*_{t1,t-1}L^*_{t,t1+l-1}))$
   where $t1 + l_{\min} \leq t \leq t1 + l - l_{\min}$
   Record the corresponding segmentation point and action labels
5. return $L^*_{1,T}$ and the corresponding $S^*_{1,T}$

We use l_{min} here to avoid over segmentation as well as impose a constraint on the stationary actions (or precisely, poses, e.g. stand) such that duration of a stationary action should be long enough.

The complexity of the above algorithm is $T^3/6$ in terms of computation of $L^*_{t1,t-1}L^*_{t,t1+l-1}$. The cost is still high if T is very large. Significant speedup can be achieved if there are pauses or stationary actions (which usually occur in a long sequence), which can be easily detected beforehand by sliding a temporal window and analyzing the mean and variance of motion within the sliding window. These pauses or stationary actions are then used to pre-segment the long sequence into several shorter sequences.

The above *step 2* requires computation of $P(O_{t1}...O_{t2}|\lambda)$, the probability of observing each valid sub-sequence $O_{t1}...O_{t2}$ by an HMM. To avoid repeated computation of such $P(O_{t1}...O_{t2}|\lambda)$ in the **Forward** procedure, we augment the **Forward** procedure so that instead of returning the probability of observing $O_{t1}...O_{t2}$ only, we return the probabilities of observing each sub-sequence of $O_{t1}...O_{t2}$ starting from O_{t1}.

5 Experimental Results

To validate the proposed action classification and segmentation algorithm, we tested it on a large MoCap database as well as several annotated and automatically tracked sequences to investigate the potential of its use with video data. The results on these two types of data are shown in section 5.1 and 5.2. In section 5.3, we show a comparison with our earlier template matching based approach algorithm [7] on the same dataset.

5.1 Results on MoCap Data

The 1979 MoCap sequences in our dataset contain 243,407 frames in total. We manually segmented these sequences such that each segment contains a whole course of one action. In total we have 3745 action segments. The distribution of these segments in each action class is not uniform. *Walk* has 311 segments while *lie2stand* has only 45 segments. The average number is 170. The length of these segments are also different, ranging from 43 to 95 frames. The average is 65 frames.

In **Experiment 1**, we randomly selected half of segments of each action class for training and the remainder for classification. In **Experiment 2**, we reduced the amount of training data to 1/3. We repeated these experiments five times and the average classification rate of each class is shown in Table 1.

The overall classification rate of each action group {*leg+torso, arm, head*} are $\{92.3\%, 94.7\%, 97.2\%\}_{Exp.1}$ and $\{88.1\%, 91.9\%, 94.9\%\}_{Exp.2}$, respectively.

As expected, the performance of Experiment 2 is lower, but not by much, indicating that the algorithm is robust in terms of the amount of available training data. Compared with Experiment 1, most of the first 3 best features (not shown here due to limited space) for each action did not change (although the order may be different). This shows consistency of **AdaBoost.M2** in selecting good classifiers.

Results show that individual learned HMM classifiers have reasonably good performance; for example, in Experiment 1, one feature (all leg and torso related joints) alone can correctly classify 62.1% of leg and torso related action

Table 1. Classification rate of each action class

action	walk	run	j upward	j forward	stand	sit	bow	lie
Exp.1	94.1%	95.5%	92.2%	91.2%	91.8%	92.4%	89.8%	88.7%
Exp.2	89.0%	91.3%	87.3%	86.6%	87.9%	90.5%	86.0%	84.8%
action	stand2sit	sit2stand	stand2bow	bow2stand	stand2lie	lie2stand	sit2lie	lie2sit
Exp.1	89.7%	89.8%	89.0%	88.3%	92.4%	88.2%	91.2%	91.8%
Exp.2	84.7%	86.6%	84.8%	86.5%	88.7%	84.5%	86.6%	86.1%
action	wave hand	point	lower arm	lift arm	nod	shake head		
Exp.1	95.8%	94.2%	92.7%	92.3%	97.9%	96.7%		
Exp.2	91.3%	92.8%	89.2%	89.4%	95.1%	94.8%		

classes. The effectiveness of **AdaBoost.M2** in combining good features can be clearly seen by a gain of about 30% (from 62.1% to 92.3%) in the classification rate.

In **Experiment 3**, we tested our segmentation/recognition algorithm on 122 unsegmented long sequences (from testing set) with average length of 949 frames (please see the supplementary material for some result videos). We use the classifier learned in Experiment 1 and we set l_{min}=20 frames to avoid over segmentation. The algorithm achieves a recognition rate of 89.7% (in terms of frames).

In terms of speed, on a P4 2.4GHz PC, Experiment 1 took 153 minutes for training and 42 minutes for classification (∼47 fps). Experiment 2 took 76 minutes and 66 minutes (∼41 fps). Experiment 3 took 68 minutes to segment 122 sequences (∼28 fps). The results show that the classification as well as the segmentation/recognition algorithm works in real time.

5.2 Results on Annotated and Tracked Data

In **Experiment 4**, we used a 3-D annotation software and a 3-D tracking software developed in our group [6] to generate two annotated (994 frames in total) and one automatically tracked (159 frames) sequence.

Fig.3 shows some key frames of one annotated sequence. The rendered annotation results using a human character animation software called POSER (by Curious Labs) are displayed on the right. The ground truth action and the recognized action are shown at the top and the bottom, respectively.

Results show that most of actions have been correctly recognized although the segmentation is not perfect. Errors occur when the subject turns around because we don't model such actions in our action set. *Carry* was not recognized for the same reason. *Reach* and *crouch*, however, were recognized as *point* and *sit*, which are reasonable substitutions for *reach* and *crouch*.

The recognition rate on the annotated and tracked data is 88.5% and 84.3%, respectively. This is satisfactory considering a substantial amount of jittery noise contained in the data (root mean square position error rates of about 10 pixels (∼10 cm) and joint angle errors of about 20°). The proposed algorithm, in

Fig. 3. Key frames of one annotated video. Ground truth and recognition result is shown in top-right and bottom-right, respectively.

general, is robust to these types of errors with short duration because of the longer-term dynamics captured by the HMMs.

5.3 Comparison with Template Matching Based Algorithm

We tested our earlier template matching based algorithm in [7] using exactly the same experimental setup. In [7], each action is represented by a template consisting of a set of channels with weights. Each channel corresponds to the evolution of one 3D joint coordinate and its weight is learned according to the Neyman-Pearson criterion. χ^2 function is used as the distance measurement between the template and the testing sequences. The results of [7] are listed as follows: **Exp.1**:{leg+torso:83.1%, arm:84.8%, head:88.4%}, **Exp.2**:{leg+torso:79.4%, arm: 80.5%, head: 82.3%}, **Exp.3**:80.1%, **Exp.4**:{annotated: 82.3%, tracked:80.6%}.

The new algorithm has significantly better results. We note that the detection results for the template matching based methods are inferior to those originally reported in [7] because the sequences in this test are much more demanding and include walking styles with large variations such as staggering, dribbling and catwalk. The method described in this paper outperforms template matching not only because Boosted HMMs provide a more powerful way to model such variations but also because it is less sensitive to temporal scale changes.

Also note that the recognition algorithm in [7] does not segment long sequences in Experiment 3. It simply searches for the best matched template within the preceding window. As action label at next frame may change, that leaves many small misclassified fragments. In contrast, the segmentation based method provide some global guidance for the process to make sure that the long sequence is not over segmented.

6 Summary and Future Work

We have presented a learning-based algorithm for automatic recognition and segmentation of 3d human actions. We first decompose 3D joint space into feature spaces. For each feature, we learn the dynamics of each action class using one HMM. Given a sequence, the observation probability is computed in each HMM and a weak classifier for that feature is formed and then combined by the **AdaBoost.M2** algorithm. A dynamic programming algorithm is applied to segment and recognize actions simultaneously in a continuous sequence.

Our major contributions are a framework that boosts HMM-based classifiers using multi-class AdaBoost and a dynamic programming-based action recognition and segmentation algorithm. Our future work plan includes adding more complex actions.

References

1. L. Campbell and A. Bobick. Recognition of human body motion using phase space constraints. In *Proc. of ICCV*, pp. 624-630, 1995.
2. J. Davis and A. Bobick. The Representation and Recognition of Action Using Temporal Templates. In *Proc. of CVPR*, pp. 928-934, 1997.
3. Alexei A. Efros, Alexander C. Berg, Greg Mori and Jitendra Malik. Recognizing Action at a Distance. In *Proc. of ICCV*, pp. 726-733, 2003.
4. Y. Freund and R.E. Schapire. A decision theoretic generalization of on-line learning and application to boosting. *Journal of Computer and System Science* 55(1), 1995, pp. 119-139.
5. S. Hongeng and R. Nevatia. Large-Scale Event Detection Using Semi-Hidden Markov Models In *Proc. of ICCV*, pp. 1455-1462, 2003.
6. M.W. Lee and R. Nevatia. Dynamic Human Pose Estimation using Markov chain Monte Carlo Approach. In *Proc. of the IEEE Workshop on Motion and Video Computing* (WACV/MOTION05), 2005.
7. F. Lv and R. Nevatia. 3D Human Action Recognition Using Spatio-Temporal Motion Templates. In *Proc. of the IEEE Workshop on Human-Computer Interaction* (HCI05), 2005.
8. L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proc. of the IEEE*, 77(2):257-286, 1989.
9. C. Rao, A. Yilmaz and M. Shah. View-Invariant Representation and Recognition of Actions. In *Int'l Journal of Computer Vision* 50(2), Nov. 2002, pp. 203-226.
10. E. Shechtman and M. Irani. Space-Time Behavior Based Correlation. In *Proc. of CVPR*, I pp. 405-412, 2005.
11. P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Proc. of CVPR*, pp. 511-518, 2001.
12. A. Yilmaz and M.Shah. Actions Sketch: A Novel Action Representation. In *Proc. of CVPR*, I pp. 984-989, 2005.
13. P. Yin, I. Essa and J. M. Rehg. Asymmetrically Boosted HMM for Speech Reading In *Proc. of CVPR*, II pp. 755-761, 2004.