M.I.T. Media Laboratory Perceptual Computing Section Technical Report No. 505
Appears in Proceedings of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, Corfu, Greece, September 26-27, 1999

Realtime Online Adaptive Gesture Recognition

Andrew D. Wilson Aaron F. Bobick
Vision and Modeling Group
MIT Media Laboratory
20 Ames St., Cambridge, MA 02139
(drew, bobick@media.mit.edu)

Abstract

We introduce an online adaptive algorithm for learning gesture models. By learning gesture models in an online fashion, the gesture recognition process is made more robust, and the need to train on a large training ensemble is obviated. Hidden Markov models are used to represent the spatial and temporal structure of the gesture. The usual output probability distributions — typically representing appearance — are trained at runtime exploiting the temporal structure (Markov model) that is either trained off-line or is explicitly hand-coded. In the early stages of runtime adaptation, contextual information derived from the application is used to bias the expectation as to which Markov state the system is in at any given time. We describe the *Watch and Learn* system, a computer vision system which is able to learn simple gestures online for interactive control.

1 Introduction

One of the challenges in implementing gesture recognition systems is to design gesture models that work across a wide variety of users and environments. The problem of generalization is particularly acute when computer vision techniques are used to derive features. Lighting conditions, camera placement, assumptions about skin color, even the clothing worn by the user can disrupt gesture recognition processes when they are changed in ways not seen during training.

We argue that rather than attempt to construct training ensembles that cover all possible scenarios, it is preferable to adapt existing models to the situation at hand. This paper presents preliminary work in developing a system that learns gestures in an online manner. The only knowledge explicitly encoded into the model *a priori* is a Markov model representing the temporal structure of the gesture.

We demonstrate the technique in a simple gesture recognition system based on computer vision as input. Gesture is used in an interactive context for controlling interaction events. We show that with the online adaptive approach, it is possible to use simple features that are not necessarily invariant to the usual set of transformations that disrupt recognition processes.

2 Motivation: Online Adaptive Learning of Gesture

Typically gesture recognition systems are trained by gathering a number of sequences that serve as examples of a class of gestures, and a model of the class of gestures is constructed automatically from these examples. Hidden Markov models (HMMs) are a popular choice to model gestures because they are easily trained and are efficient in the testing phase [3].

One of the drawbacks of this traditional approach is that the trained models only work well in testing if the situation under which the testing data are collected is typical of the situations in which the training sequences were collected. A systematic bias present in the testing conditions with respect to the training data conditions may confuse the classification. For example, if the input features are not translation-invariant and the user has moved a bit, the trained models may no longer be appropriate.

There are two common approaches to this problem: collecting data over many different sessions, and choosing a feature space that generalizes well. By collecting data over many sessions and incorporating them all into the example set, the hope is that the resulting model will encapsulate all the kinds of variation in the gesture that the system is likely to see during runtime. The drawbacks of this approach are two-fold: first, the number of examples that are required may in fact be too great to be practical, and second, as the number of distinguishable situations increase, the model will require more and more degrees of freedom to adequately represent the set of gestures.

The second approach, that of choosing the right feature space, has the chief drawback that it is in general difficult to craft a feature set that at once collapses the variation of the signal so that a manageable number of examples are sufficient, and still allows sufficient detail that the gesture may be recognized among a set of gestures.

We argue that these difficulties may be somewhat eased if we let part of the feature selection process happen during runtime. In the next section we show how an *a priori* model of the temporal structure of the gesture, when combined with constraints from context, makes runtime feature selection possible. We call this the *online adaptive learning* of gesture to differentiate it from the usual gesture recognition methodology in which the gesture models are trained off-line.

3 Temporal Structure, Context and Control

The idea of the online adaptive gesture learning algorithm presented in this paper is that if the system has a representation of the temporal structure of the gesture in question and this can be combined with real-time information derived from the application context, then the situation is sufficiently constrained that a system may conduct feature selection on the fly. Then later, when context information is

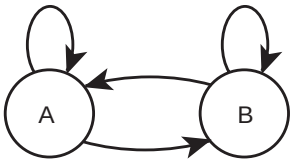

Figure 1: The simplest Markov model appropriate for a periodic signal. Given equal transition probabilities from state A to B, the Markov model is symmetric in A and B. Without contextual information, alignment of a signal to this Markov model would yield one of two possible equivalent assignments of semantics to A and B.

unavailable, the system will be able to recognize the gesture via the learned representation.

The need for context arises because typically there is insufficient structure in the temporal model to unambiguously align a given input sequence with a potential traversal of *a priori* defined states. For example, consider a gesture composed of "up" and "down" phases. The temporal structure of such a gesture would be represented by the periodic Markov model in Figure 1. If we try to align an observation sequence to the Markov model in the figure, we find there are two ways to do this. One possible outcome assigns state A a mean feature vector that we call "down" and B a mean vector of "up". The other outcome swaps the assignment of "down" and "up".

If our only concern is recognition then such a transposition is unimportant; the likelihood of the learned HMM producing the observed sequence is the same in either case. However, our goal is to use gesture for *control* of dynamic human-computer interactions. As described in section 7 we exploit the temporal-context sensitivity of HMMs by allowing a high likelihood of being in particular states to trigger application events. In this approach, an inversion of, say, "up" and "down" states is unacceptable. Note that the ambiguity may happen not at just the level of single states, but at the level of groups of states, such as whole gesture models.

A way to resolve this ambiguity is to resort to some external information, such as that provided by *application context*. If we can get a hint from the application to differentiate "down" from "up", the ambiguity is removed. Now that the features that correspond to the states has been unambiguously determined, the context information is no longer required to perform an alignment which avoids the original ambiguity.

The learning algorithm presented in this paper incorporates the real-time learning of a hidden Markov model given application context information.

4 Related Work

In [5] we use an approach similar to that presented in this paper to extract two broad classes of the natural, spontaneous gestures that people make when they are telling a story: *biphasic* gestures, which involve moving the hands out of a rest position, into the gesture space, and back to the rest position, and *triphasic* gestures, which consist of an additional stroke phase while the hands are in the gesture space. This classification scheme highlights the temporal differences of an ontology of gestures developed in [1]. A Markov model was hand-designed for each of the two classes of gesture, which differed in their temporal structure. These Markov models were then combined with image-based features derived from a long (5-minute) video sequence to derive the appearance of various rest-states used by the speaker. The present work similarly fits a hand-coded Markov model with a block of video input.

In [4] we introduce the parametric hidden Markov model (PHMM) formalism for representing a family gestures with a hidden Markov model. A PHMM encodes the manner in which the spatial form of the gesture changes as a semantically meaningful parameter changes. For example, the form of the gesture indicating the size of object depends on the size of the object. The PHMM testing phase involves computing the value of the parameter that maximizes the likelihood of the gesture. PHMM output probability distributions depend on a global vector-valued parameter. The present work uses a similar style of online EM optimization to determine the value of the parameters used in HMM output state distributions.

Oliver, Pentland and Berard [2] use online EM methods to adapt color-class models in real-time for their face and lips tracking system.

5 Learning Algorithm

5.1 Expectation-Maximization Algorithm for Hidden Markov Models

A hidden Markov model uses the topology of a Markov model and its associated transition probabilities to express the temporal structure of the gesture. For example, a periodic motion may be represented by a simple Markov model with two states and transitions back and forth between them, as in Figure 1. During testing, the Viterbi or forward/backward algorithms are used to compute the likelihood that an input sequence has the same temporal structure of the HMM, as well as match the state output probability distributions. In the process of calculating this likelihood, the forward/backward algorithm computes the posterior probability $\gamma_{tj} = P(q_t = j \mid O, \lambda)$, the probability that the HMM was in state j at time t, given the observation sequence O and HMM λ. The quantities γ_{tj} represent the parse of the HMM.

If all the values γ_{tj} are known, it is easy to see how to update the output probability distributions. For example, if the output probability distributions are Gaussian with mean μ_j and covariance Σ_j, the update equations are:

$$\mu_j = \frac{\sum_t \gamma_{tj}\, \mathbf{x}_t}{\sum_t \gamma_{tj}} \tag{1}$$

$$\Sigma_j = \frac{\sum_t \gamma_{tj} (\mathbf{x}_t - \mu_j)(\mathbf{x}_t - \mu_j)^T}{\sum_t \gamma_{tj}} \tag{2}$$

This is the Baum-Welch update used in training an HMM. The Baum-Welch algorithm is an expectation-maximization (EM) algorithm, where the expectation step involves calculating the γ_{tj} and the maximization step involves updating the parameters of the output probability distributions and transition probabilities.

5.2 Controlling the Online Adaptation

In the online adaptive learning of gesture, we use HMMs to represent the gesture we wish to recognize, but instead of running the Baum-Welch algorithm off-line, we run it during runtime to update the output probability distributions. In the present work, we start with a known Markov model and transition probability matrix that represents the temporal structure of the gesture of interest, while the parameters of the output probability distributions are randomized at the start. As discussed in Section 3, without some hints from application context, the states of the learned hidden Markov model may not obey the proper semantics required by the application (for example, "down" and "up" may be swapped, or the "up" gesture may be swapped with the "down" gesture).

The modification required to the Baum-Welch algorithm for it to exploit context is basically to bias the γ_{tj} after the initial computation of the expectation. Since the γ_{tj} are used as weights to update the state output distribution parameters, the biased γ's may be thought of as an attention focusing mechanism. We may bias γ_{tj} as a way to incorporate exterior knowledge to influence this focus of attention and thus guide the learning. In the exposition that follows we give one method to create a lattice of biased γ_{tj}, by defining a new quantity that is a linear combination of γ_{tj} and probabilities derived from application context.[1]

The information from application context is assumed to take the form of posterior probabilities for each state:

$$\omega_{tj} = P(q_t = j \mid \Omega) \tag{3}$$

where Ω represents application context. These posterior probabilities are then combined with the usual HMM posterior probabilities $\gamma_{tj} = P(q_t = j \mid \lambda)$ to obtain a new posterior probability which incorporates the HMM and the application state context:

$$\Gamma_{tj} = \rho_j \gamma_{tj} + (1 - \rho_j)\omega_{tj} \tag{4}$$

which is subsequently normalized so that $\sum_j \Gamma_{tj} = 1$. ρ_j is a scalar quantity that is proportional to how much the HMM state j has been tuned during online adaptation.

In the current system, we set ρ_j to be proportional the number of frames for which γ_{tj} is greater than some fixed value (say, 0.7). When beginning the adaptation, ρ_j is at its minimum value, then increases to some maximum value during adaptation. The intuition is that this quantity controls the degree to which the system follows the application context versus the HMM. It also overcomes the fact that when b_{jt} takes the form of Gaussian distributions, starting with large covariances to represent uncertainty brings b_{jt} to zero and so the state is never exploited by the HMM. The effect of ρ_j during runtime is to artificially bias the algorithm to use neglected states.

We also incorporate a global learning rate α to control the adaptation process. The idea is that at the start of the algorithm, when the state output distributions parameters have random values the algorithm should learn aggressively, and that later when the parameters have approached good "final" values the algorithm should change the values less aggressively. This prevents the algorithm from changing the gesture model to match some spurious input.

In the present system α is derived from the confidence value ρ_j described above. We currently set the relationship between ρ_j and α in an *ad hoc* manner. For a state which has seen no probability mass γ_{tj}, we would like quantity to be 1.0. It is important that the learning rate always have some value greater than zero, so that the algorithm can continually adapt to slow changes in the gesture signal. Optionally, we normalize α by the frame rate of the online EM process.

The learning rate α is incorporated in the EM update by simply mixing the old value of the parameter with the new value:

$$\mu_j' = (1 - \alpha)\mu_j + \alpha \sum_t \Gamma_{tj}\, \mathbf{x}_t \tag{5}$$

The quantities $P(\mathbf{x}_t \mid q_t = j, \lambda)$ are computed over the sequence $\langle \mathbf{x}_{t-T} \ldots \mathbf{x}_t \rangle$, and the EM algorithm is run once over the sequence. At some time $t + \Delta t$, this process is repeated (caching values where possible) over the next window $\langle \mathbf{x}_{t+\Delta t-T} \ldots \mathbf{x}_{t+\Delta t} \rangle$, and so on, throughout the lifetime of the application – there are no distinct training and testing phases.

6 Images as Input

6.1 Tracking

The *Watch and Learn* system uses the online adaptive algorithm described above with whole images as input. Color images are acquired at a rate of 30Hz from a camera pointed at the user. A body-centric image of the user is derived from the input image by subtracting the pixel values of the image of the scene without the user (the background image) from the current image. This difference image is then binarized to obtain a silhouette image. A simple EM-based tracking algorithm updates the center of body-centric image at the center of the silhouette. The body-centric silhouette image is then multiplied by the original image to obtain a color image that is body-centric and does not include the background.

The EM-based tracking algorithm models the spatial distribution of the silhouette pixels over the image as a Gaussian with fixed covariance.

$$h_{x,y} = \frac{1}{\sqrt{2\pi \mid \Sigma \mid}} e^{-\frac{1}{2}\left([\mathrm{x\ y}]^T - \mathbf{c}\right)^T \Sigma^{-1} \left([\mathrm{x\ y}]^T - \mathbf{c}\right)} \tag{6}$$

$$\mathbf{c}' = \sum_{I_{x,y} > b} h_{x,y} [\mathrm{x\ y}]^T \tag{7}$$

where $I_{x,y}$ is the value of the pixel at (x, y), $\mathbf{c}$ is the vector of tracked coordinates from the previous time step, b is threshold for binarizing the image and Σ is the constant covariance matrix that is chosen to approximate the size of the user in the image.

$h_{x,y}$ is calculated over a window centered about $\mathbf{c}$. If the likelihood of this model falls below a threshold, the algorithm enters a seek mode in which the mean of the Gaussian is assigned a random value at each successive frame until the likelihood is above the threshold. Otherwise, the mean

[1] In the present system, we implement this bias by altering the form of the output probability distribution rather than by directly manipulating γ_{tj}.

of the Gaussian is updated to reflect the translation of the silhouette.

6.2 Output Probability Distribution

The pixel values of the cropped color foreground image centered about $\mathbf{c}$ at time t are concatenated to form the feature vector $\mathbf{x}_t$. The last two seconds of the color foreground images are buffered in memory. These form the observation sequence over which the online adaptive EM algorithm updates the output probability distribution parameters.

The output probability distributions $b_{jt} = P(\mathbf{x}_t \mid q_t = j)$ take the form:

$$b_{jt} = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{1}{2\sigma_j^2 XY}(\mathbf{x}_t - \mu_j)^T(\mathbf{x}_t - \mu_j)} \tag{8}$$

where σ_j is a scalar, and X and Y are the dimensions of the image corresponding to $\mathbf{x}_t$.

The output probabilities $b_{jt}(\mathbf{x})$ are computed over all states j for the current time step only; the values are saved in a buffer of the last T time steps. Updating the output probability distribution parameter μ_j proceeds as equation 1, here involving the weighted sum of the images $\mathbf{x}_t$ in the image buffer. The update of σ_j is similarly a weighted sum:

$$\sigma_j = \frac{\sum_t \Gamma_{tj}(\mathbf{x}_t - \mu_j)^T(\mathbf{x}_t - \mu_j)}{\sum_t \Gamma_{tj}} \tag{9}$$

After each maximization step, it would be correct to recompute the value of the output probability distributions for each state and each image in the image buffer, since the parameters of the distribution have changed by the update equations. In the present system, however, we do not recompute these likelihoods for the reason that much computation may be avoided by computing the likelihoods for only the newest image. If the buffer is small and the changes in the parameters are continuous, then the outdated values of the likelihood associated with the oldest frames in the buffer will not upset the learning algorithm. Empirically we have found this to be the case.

In the *Watch and Learn* system, computing the weighted sum of images for the maximization step is the most computationally intensive step of the algorithm and need not be executed at every new time step. Thus with the current system, the input image buffer is updated at 30Hz, while the learning algorithm executes at no more than 4Hz. Both the expectation and maximization steps of *Watch and Learn* have been implemented to use MMX single instruction/multiple data (SIMD) instructions available on the Intel Pentium II processor.

7 Application: Conducting

One activity in which there is strong contextual information is musical conducting, where both the musicians and the conductor follow a score. The *Watch and Learn* system has been applied to a simplified conducting scenario to demonstrate that a simple beat gesture may be learned by adaptive online learning.

The interaction between the user who plays the role of the conductor and the system is as follows. The user steps in front of the camera and waits for the system to play a series of beats (wood block sounds) that establish a tempo. After a few beats, the user begins to follow the beat with his hand. After a few bars of following the system's beat, the system begins to play a piece of music. Having taught the system his own beat gesture, the user is now free to change the tempo of the piece currently playing.

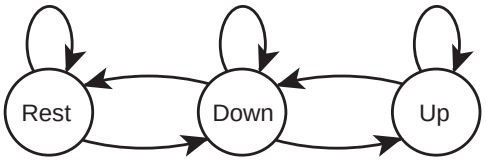

Figure 2: The Markov model used to represent the temporal pattern of a beat in the *Watch and Learn* system applied to the simple conducting scenario.

For the simple beat pattern described, a simple three state Markov model is used to model the temporal structure of the gesture (see Figure 2). The Markov model begins in a rest state, which is learned at first when the user is standing in front of the camera waiting for the system to establish a beat. During the fourth beat generated by the system, the user is supposed to have begun following the beat with his gesture. At this point, contextual priors are changed to match the expectation that at the instant of the system-generated beat, the user should be in the "downbeat" state. Given that at this stage in the learning the rest state has already been learned by the system, the "upbeat" state will be learned correctly because temporal structure provided will lead the system to ascribe the moments before the downbeat to the "upbeat" state, and furthermore, presumably the images during the actual upbeat motion will look different than the "rest" state.

As the user counts out the beats, the appearance models (the means of the output probability distribution) associated with each state gradually appear as reasonable approximations to what an observer might call the "upbeat", "downbeat" and "rest" phases of the gesture. Figures 3 and 4 show a typical set of appearance models for the beat Markov model during the adaptation process. Figure 3 shows the appearance models in the middle of the adaptation process, Figure 4 after the adaptation is complete. The system continually adjusts these states to reflect the subtle changes in the way the user executes the gesture from instance to instance.

Figures 5 and 6 show γ_{tj} and ω_{tj} over a window of 64 frames (about 2 seconds) of video. Figure 5 is taken during adaptation, Figure 6 after adaptation. Note that during adaptation, the probabilities from the application ω_{tj} (square wave) guide the state membership γ_{tj}.

QuickTime video of the camera input and the learning algorithm are located at `http://www.media.mit.edu/~drew/watchandlearn`.

Figure 7 shows appearance models learned in a second session with the system. Note that even while many of the viewing circumstances such as pose, distance to camera, and clothing have changed, the adapted appearance models have the correct semantics.

We wish to remind the reader that *Watch and Learn* in no way attempts to track the user's hands; it is purely through

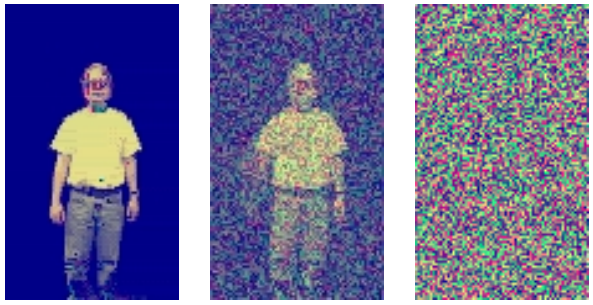

Figure 3: The appearance models (images) associated with each state of the beat HMM during online adaptation. When the algorithm is started, the pixels values of the images are randomized. At this point, the appearance model of the rest state has been trained, and the appearance model of the "up" state is being adapted.

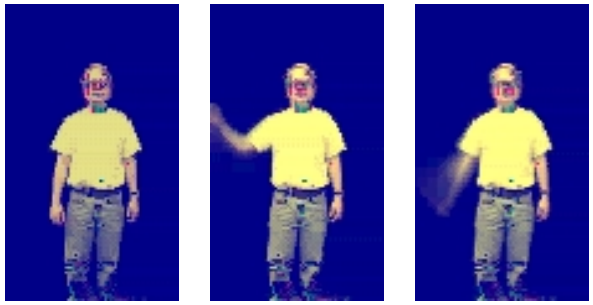

Figure 4: The appearance models (images) associated with each state of the beat HMM after online adaptation.

the combination of the temporal structure model and the contextual information that gives rise to the semantically correct appearance models. Ultimately, the most important requirement is that the user be *cooperative*, especially during the periods of high learning rate, and *consistent*. It is quite possible to train *Watch and Learn* to recognize foot tapping instead of hand beats, as long as the user consistently does so.

Changes in tempo are made in a very simplistic manner according to the rise and fall of $\gamma_{t,up}$: when $\gamma_{t,up}$ falls below a certain threshold, a "beat" event is generated. The time between the current beat and the last beat is calculated, converted to MIDI clock units and passed on to the MIDI time-keeper running on the host computer. Presently, no attempt is made to synchronize where the particular downbeat falls with the downbeat in the score. If at some point the user returns to the rest state, tempo changes are not made and the piece continues playing at the last tempo.

8 Discussion and Future Work

An online adaptive learning algorithm for learning gestures has been presented. The approach differs from the usual train/test paradigm in that much of the training process may occur online. The algorithm requires a Markov model that represents the temporal structure of the gesture to be learned. This is combined with contextual information to train the output probability distributions during

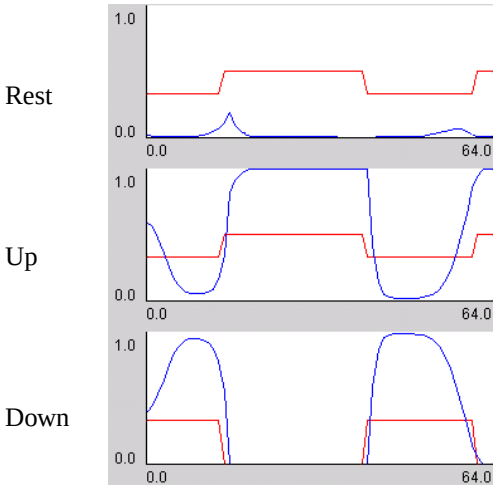

Figure 5: Plots of γ_{tj} and ω_{tj} (square wave) for the beat gesture, during online adaptation.

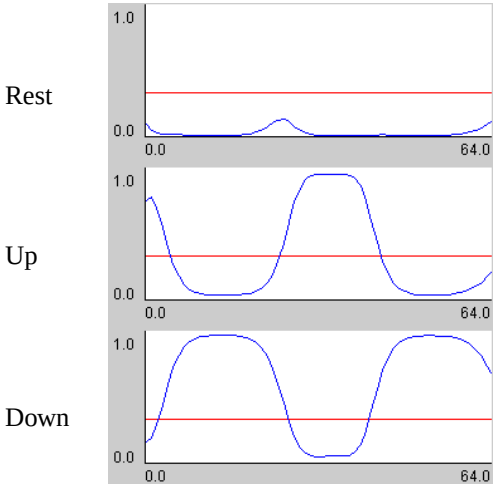

Figure 6: Plots of γ_{tj} for the beat gesture, after online adaptation.

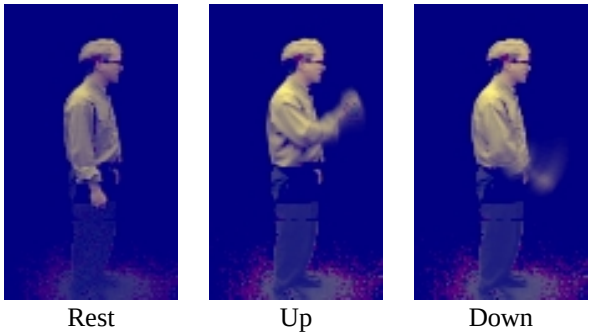

Figure 7: The appearance models (images) associated with each state of the beat HMM after online adaptation during a second session.

runtime. *Watch and Learn* succeeds in learning a simple beat pattern, and in another configuration has been applied to learning a mapping to various drum sound patches with a slightly more complex temporal model (see Figure 8).

We argue that the problem of generalization by feature selection is eased with the online adaptive learning algorithm presented above. By delaying the estimation of the output probability density parameters to runtime, the online algorithm is free to choose only those values which fit the current set of data. Thus any particular bias in the features present in runtime that would have upset an off-line approach is absorbed in the online learning process.

The net result is that with the online algorithm, feature selection is not as crucially important as with the off-line algorithm in obtaining generalization performance. As long as the features are consistent over the set of learned states, the online algorithm will set the output probability distribution parameters appropriately. For example, image space itself may make an extremely poor feature space for many gesture applications because many of the usual desired invariants are absent.

Although in general computer vision has been dismissed as a technique useful to computer music on the grounds that the techniques are too computationally complex to run quickly on today's hardware, we note without hard justification that *Watch and Learn* is quite responsive. One reason for this is the fact if events are triggered from γ_{tj}, the temporal model enables the system to anticipate events: for example, a MIDI note-on event may be generated at the moment that $\gamma_{t,up}$ begins to fall below a threshold, which is in a moment in advance of the peak of $\gamma_{t,down}$ (see Figure 6). Also recall that $\gamma_t j$ is being updated at 30Hz.

There are two drawbacks to the *Watch and Learn* system as it is currently implemented. First, the system assumes that the user is being cooperative at all times. This drives the learning initially, but can be a problem once gesture models have been learned. For example, once the beat gesture is learned reliably, if the user does a completely different gesture, this new movement should not be incorporated into the model. However, if the gesture appears to have the same temporal structure as the original beat, and occurs at the moment in time during which the system expects a beat, the system should incorporate the new information.

The second drawback to the *Watch and Learn* system is the *ad hoc* manner in which the confidence values ρ_j and the learning rate α is determined. We expect to incorporate more principled ways of controlling the adaptation.

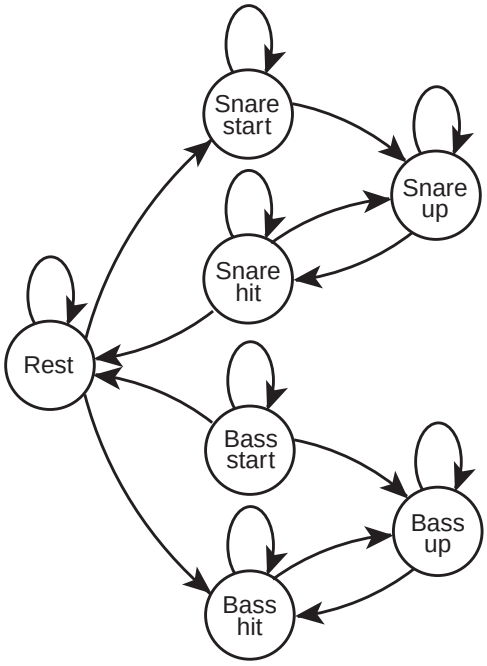

Figure 8: The Markov model used for a drum-set configuration of *Watch and Learn*. MIDI events are generated when the Markov model enters the "Snare hit" and "Bass hit" states. "Snare start" and "Bass start" states capture the preparation of the gesture and do not generate MIDI events. The appearance models of the start states may adapt to resemble those of the hit states, thus they are necessary to prevent spurious MIDI events during the gesture's preparation phase.

References

[1] D. McNeill. *Hand and Mind: What Gestures Reveal About Thought*. Univ. of Chicago Press, Chicago, 1992.

[2] S. Pentland N. Oliver and F. Berard. A real-time lips and face tracker with facial expression recognition. *Proc. Comp. Vis. and Pattern Rec.*, 1997.

[3] L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. Prentice Hall, Englewood Cliffs, 1993.

[4] A. D. Wilson and A. F. Bobick. Recognition and interpretation of parametric gesture. *Proc. Int. Conf. Comp. Vis.*, pages 329–336, 1998.

[5] A. D. Wilson, A. F. Bobick, and J. Cassell. Temporal classification of natural gesture and application to video coding. *Proc. Comp. Vis. and Pattern Rec.*, pages 948–954, 1997.