An Efficient Algorithm for Modelling Duration in Hidden Markov Models, with a Dramatic Application*

Søren Hauberg and Jakob Sloth

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF COPENHAGEN
COPENHAGEN, DENMARK
{HAUBERG,SLOTH}@DIKU.DK

September 24, 2010

To appear in Journal of Mathematical Imaging and Vision, Special Issue: Tribute to Peter Johansen. DOI:10.1007/s10851-007-0059-9

Abstract

For many years, the hidden Markov model (HMM) has been one of the most popular tools for analysing sequential data. One frequently used special case is the *left-right* model, in which the order of the hidden states is known. If knowledge of the duration of a state is available it is not possible to represent it explicitly with an HMM. Methods for modelling duration with HMM's do exist [7], but they come at the price of increased computational complexity. Here we present an efficient and robust algorithm for modelling duration in HMM's, and this algorithm is successfully used to control autonomous computer actors in a theatrical play.

Keywords Hidden Markov Models; modelling duration; filtering; optimal particle filters; theatrical play

1 Introduction

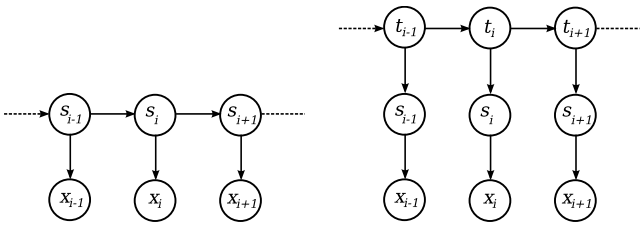
In this paper, we study how sequences of data can be modelled when knowledge about the order, and the duration of discrete states are available. The objective of the paper is to compute the probability that the data generating system is in a specific state using all available observations. If we know the order of the states (i.e. state 2 is directly preceded by state 1), the left-right hidden Markov model may be well-suited [7]. However, there is a limitation in this model in that it cannot directly represent knowledge about the duration of states. That is, if we know that the system will be in state 1 for approximately 3 minutes and thereafter in state 2 for about 1 minute, we cannot represent that knowledge properly with a hidden Markov model (HMM). Several solutions have been proposed to deal with this problem, see e.g. [5] for details. The most intuitive solution to this problem is essentially to partition each state into several sub-states, where states with long durations are partitioned into many sub-states, and states with shorter durations into few sub-states. It can be shown [7] that the expected duration of a state is proportional to the number of sub-states, which makes the strategy sensible. The problem is that it is hard to determine the best number of sub-states. Furthermore, the computational demands of the resulting algorithm increase with the total number of sub-states.

All duration modelling solutions seem to be based on models with a discrete state space. In this paper the approach taken is first of all to generalise the hidden Markov model, and second of all to implement the resulting model by using a particle filter. The basic idea is that the best way to model time, and hence duration, is by means of a continuous variable. For that reason, the discrete Markov model in an HMM is replaced by a state-space model, where a continuous hidden variable determines the discrete state of the system. The resulting model is then implemented using the optimal particle filter [3].

The algorithm is tested by producing a theatrical play with autonomous computer actors. In this setting, the manuscript of the play provides information about both state order and duration. The only previous attempt of producing such a play seems to be the work of Pinhanez and Bobick [6], where order and duration is modelled by means of temporal logic.

2 Deriving the Model

A basic model for describing order of events is the hidden Markov model [7]. In this model, we work with a hidden discrete state variable $s_i \in \{1,\ldots,H\}$ that only depends on the previous state s_{i-1}. In turn, the actual observation $\mathbf{x}_i \in \mathscr{R}^d$ only depends on the current state of the system. Thus, to specify



(a) (b)

Figure 1: (a) A graphical model of a Hidden Markov Model. (b) A graphical model of the suggested extension.

the system we need the state transition distribution $p(s_i|s_{i-1})$ and the observation distribution $p(\mathbf{x}_i|s_i)$. This model is illustrated graphically in figure 1a.

The HMM does not allow us to explicitly encode knowledge of the duration of states. We therefore seek a model that allows us to specify the system's expected amount of time in each state. In order to do this, the state transition distribution is changed from a Markov model to a state-space model. More specifically we let the current state s_i depend on the value of a continuous variable t_i. This variable in turn depends on its previous value. Now the state transition is represented by two distributions $p(s_i|t_i)$ and $p(t_i|t_{i-1})$. This model is represented graphically in figure 1b.

The intuition behind this model is that t_i represents time, and at every point in time we know the probabilities of each state. We assume that the entire sequence is finite. Then it makes sense to assume that t_i is confined to an interval $[a_0, a_J[$. To specify $p(s_i|t_i)$, we partition this interval into J non-intersecting intervals $[a_{j-1}, a_j[,\ j = 1 \dots J$. In each of these intervals, we assume that the state probabilities $p(s_i|t_i)$ are constant. More formally, we define

$$p(s_i = h|t_i) = \sum_{j=1}^{J} w_{hj} \mathbf{1}_{[a_{j-1},a_j[}(t_i), \tag{2.1}$$

where $\mathbf{1}_{[a_{j-1},a_j[}$ denotes the indicator function which is defined as

$$\mathbf{1}_{[a_{j-1},a_j[}(t_i) = \begin{cases} 1 & t_i \in [a_{j-1}, a_j[\\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

For (2.1) to be a valid distribution, we obviously have to assume that

$$\sum_{h=1}^{H} w_{hj} = 1 \qquad \text{and} \qquad 0 \leq w_{hj} \leq 1, \tag{2.3}$$

where H denotes the number of different states.

In an HMM, the state transition is a discrete distribution, which means it can be written as

$$p(s_i = h|s_{i-1}) = \sum_{j=1}^{J} w_{hj} \delta(s_i - h), \tag{2.4}$$

where the weights w_{hj} depend on the value of s_{i-1}. This distribution can be compared to (2.1), where we note that the delta functions have essentially been replaced with indicator functions.

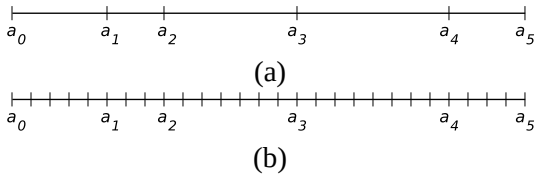


Figure 2: (a) An illustration of state duration in the suggested model. Each interval corresponds to a state, and the length of an interval corresponds to the expected duration of a state. (b) The traditional way of modelling duration with HMM's. Each interval is partitioned into several intervals. This corresponds to a quantisation of t_i.

For the model to be fully specified, we must also define $p(t_i|t_{i-1})$. Since t_i is confined to the interval $[a_0, a_J[$, we note that this distribution should be zero outside this interval. In this paper, we essentially assume that $p(t_i|t_{i-1})$ is a normal distribution, but other choices should be possible. Formally we define

$$p(t_i|t_{i-1}) = z^{-1}\mathbf{1}_{[a_0,a_J[}(t_i)\mathcal{N}(t_i|\mu,\sigma^2), \tag{2.5}$$

where both $\mu = \mu(t_{i-1})$ and $\sigma = \sigma(t_{i-1})$ can be any function of t_{i-1} as long as $\sigma > 0$. The normalisation constant z can easily be evaluated as the integral of the normal distribution from a_0 to a_J. This specifically applies to $z = \mathcal{G}(a_0, a_J)$, where

$$\mathcal{G}(a,b) = \int_a^b \mathcal{N}(t_i|\mu,\sigma^2)\mathrm{d}t_i = \frac{1}{2}\left(\mathrm{erf}\left[\frac{b-\mu}{\sqrt{2}\sigma}\right] - \mathrm{erf}\left[\frac{a-\mu}{\sqrt{2}\sigma}\right]\right). \tag{2.6}$$

We now have a fully specified model. To get a better understanding of the model, we assume that σ is constant, and $\mu = t_{i-1} + \Delta$, where Δ is constant. Since t_i is thought of as time, this choice simply tells us that time goes by. The state distribution $p(s_i|t_i)$ is defined as constant in the intervals $[a_{j-1}, a_j[$. We now assume that the system state is deterministically given by the interval, i.e. $w_{h\hat{j}} = 1$ for some value of $\hat{j}$. We then see that each interval $[a_{j-1}, a_j[$ corresponds to a state, and its length to the expected duration of the state. This is illustrated in figure 2a.

With HMM's we can compute the filtering distribution $p(s_i|\mathbf{x}_{1:i})$. This distribution can be computed directly from $p(t_i|\mathbf{x}_{1:i})$ as

$$p(s_i|\mathbf{x}_{1:i}) = \int p(s_i,t_i|\mathbf{x}_{1:i})\mathrm{d}t_i = \int p(s_i|t_i)p(t_i|\mathbf{x}_{1:i})\mathrm{d}t_i. \tag{2.7}$$

One algorithm for computing the filtering distribution $p(s_i|\mathbf{x}_{1:i})$ with this model would be to quantise t_i, and then use a standard HMM filter. This idea is illustrated in figure 2b. This is the standard way of modelling duration in HMM's [7]. The problem with this approach is that it is not clear how fine-grained t_i should be quantised. If t_i is quantised very fine-grainedly, the filtering algorithm becomes computationally expensive. More specifically, if we quantise t_i into Q states, the HMM filter runs in $\mathcal{O}(Q^2)$. If we use a less fine-grained quantisation instead, the results become less accurate. In this paper, we propose a filtering algorithm based on a particle filter rather than an HMM filter.

3 About Particle Filters

When the hidden state variable is continuous, the currently most popular approach to filtering is the particle filter. In this section, we review the basics of this algorithm. For a more in-depth description containing most of the proofs see [3].

The aim of the algorithm is to estimate moments of the filtering distribution $p(t_i|\mathbf{x}_{1:i})$ by means of samples. Here, $\mathbf{x}_{1:i}$ denotes all observations up to iteration i, i.e. $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i$. The idea is that samples are taken from $p(t_{1:i}|\mathbf{x}_{1:i})$ and then all values of the samples but t_i is ignored. Since the filtering distribution is unknown, we turn to importance sampling [1]. This means drawing samples $t_{1:i}^{(n)}$ from an *importance distribution* $q(t_{1:i}|\mathbf{x}_{1:i})$ and then estimating moments of the filtering distribution as

$$\bar{h} = \int h(t_i) p(t_i|\mathbf{x}_{1:i}) \mathrm{d}t_i \approx \sum_{n=1}^{N} \frac{w_i^{(n)}}{\sum_{m=1}^{N} w_i^{(m)}} h(t_i^{(n)}), \tag{3.1}$$

where we have defined the *importance weights* as $w_i^{(n)} = p(t_{1:i}^{(n)}|\mathbf{x}_{1:i}) / q(t_{1:i}^{(n)}|\mathbf{x}_{1:i})$. If $N \to \infty$ (3.1) is exact. The key to making this strategy work is to choose an importance distribution, which ensures that the resulting algorithm is recursive. With this in mind, it is chosen that the importance distribution should be factorised as

$$q(t_{1:i}|\mathbf{x}_{1:i}) = q(t_{1:i-1}|\mathbf{x}_{1:i-1}) q(t_i|t_{i-1}, \mathbf{x}_i). \tag{3.2}$$

With this choice, one can sample from $q(t_{1:i}|\mathbf{x}_{1:i})$ recursively by extending the previous sample $t_{1:i-1}^{(n)}$ with a sample $t_i^{(n)}$ from $q(t_i|t_{i-1}, \mathbf{x}_i)$. The weights $w_{i-1}^{(n)}$ can also be recursively updated by means of

$$w_i \propto w_{i-1} \times \frac{p(\mathbf{x}_i|t_i^{(n)}) p(t_i^{(n)}|t_{i-1}^{(n)})}{q(t_i^{(n)}|t_{i-1}^{(n)}, \mathbf{x}_i)}. \tag{3.3}$$

When extending the previous sample, we need to draw a sample from $q(t_i|t_{i-1}, \mathbf{x}_i)$. This, however, assumes that the true value of t_{i-1} is known, which is not the case. Several strategies can be used to approximate this value. *Sequential Importance Sampling (SIS)* assumes that the previous sample position was the true value, i.e. $t_{i-1} = t_{i-1}^{(n)}$. This is usually not stable, since errors accumulate in this estimate. The *particle filter* approximates the distribution of t_{i-1} with the weighted samples from the previous iteration, i.e.

$$p(t_{i-1}|\mathbf{x}_{1:i-1}) \approx \frac{w_{i-1}^{(n)}}{\sum_{m=1}^{N} w_{i-1}^{(m)}} \delta(t_{i-1} - t_{i-1}^{(n)}). \tag{3.4}$$

The value of t_{i-1} is then approximated by drawing a sample from this distribution. This simply corresponds to a resampling of the previous samples, where samples with large weights have a high probability of surviving. Since these samples are assumed to come from the true distribution $p(t_{i-1}|\mathbf{x}_{1:i-1})$, the associated weights have to be reset, i.e. $w_{i-1}^{(n)} = 1/N$ for all n.

We have still to choose the importance distribution $q(t_i|t_{i-1}, \mathbf{x}_i)$. The most simple choice is inspired by (3.3). Here, we note that the weight update is significantly simplified if we set $q(t_i|t_{i-1}, \mathbf{x}_i) = p(t_i|t_{i-1})$. With this choice, the resulting filter is called the *Bootstrap filter* [3]. This filter works quite well if the observations follow the predictions $p(t_i|t_{i-1})$ fairly well. Yet when this is not the case, the results are usually not very good. The reason for this is that new samples are not necessarily drawn from places in the state space where the likelihood $p(\mathbf{x}_i|t_i)$ is high. To avoid such problems, an alternative importance distribution is needed. The obvious choice is $q(t_i|t_{i-1}, \mathbf{x}_i) = p(t_i|t_{i-1}, \mathbf{x}_i)$. With this choice, and by means of the Markov property, it is easy to prove that

$$q(t_i|t_{i-1}, \mathbf{x}_i) = \frac{p(\mathbf{x}_i|t_i) p(t_i|t_{i-1})}{\int p(\mathbf{x}_i|t) p(t|t_{i-1}) \mathrm{d}t}. \tag{3.5}$$

With this importance distribution, the resulting method is called the *optimal particle filter*. We see that for this method, the weight update is

$$w_i^{(n)} \propto w_{i-1}^{(n)} \int p(\mathbf{x}_i|t)p(t|t_{i-1}^{(n)})\mathrm{d}t. \tag{3.6}$$

The problem with the optimal filter is that it usually cannot be implemented, since the integral in the weight update is often hard to evaluate. The only case in which this filter has actually been implemented seems to be the non-linear extensions of the Kalman filter [4]. We will, however, see that the optimal filter can be implemented in the model described in the previous section.

4 Implementing the Model

In this section, we show how the model can be implemented by means of an optimal particle filter. This algorithm requires a method for evaluating the new weight (3.6) of a particle and a method for simulating the importance distribution (3.5). These two methods will be described in the next sections.

First, we do note that since the entire state space is the interval $[a_0, a_J[$, the importance distribution can be expressed as

$$q(t_i|t_{i-1}, \mathbf{x}_i) = \mathbf{1}_{[a_0,a_J[}(t_i)\frac{p(t_i|t_{i-1})p(\mathbf{x}_i|t_i)}{\int_{a_0}^{a_J} p(t|t_{i-1})p(\mathbf{x}_i|t)\mathrm{d}t}. \tag{4.1}$$

4.1 Assigning Weights

To compute the weight of a particle, we need to evaluate (3.6). To simplify the notation, we omit the superscript $^{(n)}$ and subsequently compute the new weight as

$$\begin{aligned}
w_i &= \int p(\mathbf{x}_i|t)p(t|t_{i-1})\mathrm{d}t && (4.2)\\
&= z^{-1}\int_{a_0}^{a_J} \mathcal{N}(t|\mu,\sigma^2)\sum_{h=1}^{H} p(\mathbf{x}_i|s_i=h)p(s_i=h|t)\mathrm{d}t && (4.3)\\
&= z^{-1}\sum_{h=1}^{H} p(\mathbf{x}_i|s_i=h)\int_{a_0}^{a_J} \mathcal{N}(t|\mu,\sigma^2)\sum_{j=1}^{J} w_{hj}\mathbf{1}_{[a_{j-1},a_j[}(t)\mathrm{d}t && (4.4)\\
&= z^{-1}\sum_{h=1}^{H} p(\mathbf{x}_i|s_i=h)\sum_{j=1}^{J} w_{hj}\mathscr{G}(a_{j-1},a_j). && (4.5)
\end{aligned}$$

Since $\mathscr{G}$ can be evaluated using (2.6), we can evaluate the expression. The computational complexity of this operation is $\mathscr{O}(HJ)$.

4.2 Simulating the Importance Distribution

As the importance distribution $q(t_i|t_{i-1}, \mathbf{x}_i)$ is one-dimensional, we can simulate it by using *inverse transform sampling* [1]. This consists of computing the inverse of the cumulative distribution function and evaluating this function at a uniformly distributed point $r \in [0,1]$.

The first step of this approach is to compute the cumulative distribution function $F(y) = \int_{-\infty}^{y} q(t|t_{i-1}, \mathbf{x}_i)\mathrm{d}t$. Since the union of the intervals $[a_{j-1}, a_j[$ covers the entire state space, every value of y must fall within

one of these intervals. We now assume that this interval is $[a_{k-1}, a_k[$ and that k is known. We then get

$$F(y) = \int_{-\infty}^{y} q(t|t_{i-1}, \mathbf{x}_i)\mathrm{d}t \tag{4.6}$$

$$= w_i^{-1} \int_{-\infty}^{y} p(t|t_{i-1}) p(\mathbf{x}_i|t)\mathrm{d}t \tag{4.7}$$

$$= (zw_i)^{-1} \int_{a_0}^{y} \mathscr{N}(t|\mu, \sigma^2) \sum_{h=1}^{H} p(\mathbf{x}_i|s_i = h) \sum_{j=1}^{J} w_{hj} \mathbf{1}_{[a_{j-1}, a_j[}(t)\mathrm{d}t \tag{4.8}$$

$$= (zw_i)^{-1} \sum_{h=1}^{H} p(\mathbf{x}_i|s_i = h) \sum_{j=1}^{J} w_{hj} \int_{a_0}^{y} \mathscr{N}(t|\mu, \sigma^2) \mathbf{1}_{[a_{j-1}, a_j[}(t)\mathrm{d}t \tag{4.9}$$

$$= (zw_i)^{-1} \left[\sum_{h=1}^{H} \sum_{j=1}^{k-1} p(\mathbf{x}_i|s_i = h) w_{hj} \mathscr{G}(a_{j-1}, a_j) + \sum_{h=1}^{H} p(\mathbf{x}_i|s_i = h) w_{hk} \mathscr{G}(a_{k-1}, y) \right]. \tag{4.10}$$

We now turn to computing the inverse of $F(y)$. Setting $r = F(y)$ and rearranging (4.10) we get

$$\mathscr{G}(a_{k-1}, y) = \left[\sum_{h=1}^{H} p(\mathbf{x}_i|s_i = h) w_{hk} \right]^{-1} \left[(zw_i)r - \sum_{h=1}^{H} \sum_{j=1}^{k-1} p(\mathbf{x}_i|s_i = h) w_{hj} \mathscr{G}(a_{j-1}, a_j) \right] \tag{4.11}$$

Using the definition of $\mathscr{G}$ (2.6), we can solve this for y, which gives

$$y = \mu + \sqrt{2}\sigma \ \mathrm{erf}^{-1} \left[2 \Big[\sum_{h=1}^{H} p(\mathbf{x}_i|s_i = h) w_{hk} \Big]^{-1} \Big[(zw_i)r - \sum_{h=1}^{H} \sum_{j=1}^{k-1} p(\mathbf{x}_i|s_i = h) w_{hj} \mathscr{G}(a_{j-1}, a_j) \Big] + \mathrm{erf} \Big[\frac{a_{k-1} - \mu}{\sqrt{2}\sigma} \Big] \right]. \tag{4.12}$$

We can now simulate the importance distribution by generating a uniformly distributed number $r \in [0, 1]$ and inserting r in (4.12). The result will then follow the importance distribution. This, however, assumes that the interval $[a_{k-1}, a_k[$ containing y is known. To find this interval, we note that $p(t_i|t_{i-1}) > 0$, which makes $F(y)$ an increasing function. This tells us that

$$a_{k-1} \leq \quad y \quad \leq a_k \quad \Leftrightarrow \tag{4.13}$$

$$F(a_{k-1}) \leq \quad F(y) \quad \leq F(a_k). \tag{4.14}$$

Since $r = F(y)$, we can find k by searching the sequence $(F(a_k))_{k=1\ldots J}$ for a value of k such that (4.14) is fulfilled. It should be noted that $F(a_k)$ can be evaluated directly using (4.10). The computational complexity of the simulation is again $\mathscr{O}(HJ)$.

5 A Dramatic Application

The presented algorithm has been developed for the purpose of producing a theatrical play, where an autonomous computer acts together with a human actor. More specifically, the computer controls a robot, plays sounds, and displays animations on stage. In the theatrical setting, the manuscript of the play contains fairly detailed descriptions of both the order and the duration of the actor's actions. If an action is thought of as a state, the model is applicable. The idea is to use gesture recognition to determine what the human actor is doing and compare this with a manuscript of the human actor's

actions to get an estimate of the current position t_i in the manuscript. On this basis, it is easy to determine which actions the computer is to perform by comparing the estimate with manuscripts of the computer controlled actors.

Formally, we must define the distribution $p(s_i = h|t_i)$ in order to use the model. As state s_i describes the action performed by the human actor, the distribution corresponds to the manuscript of the play. Since the human actor is only able to perform one action at a time, the weights w_{hj} are set to 1 in the intervals where action h is performed and 0 elsewhere. We also have to define the temporal prediction $p(t_i|t_{i-1})$, which requires us to specify μ and σ. Here we let $\mu = t_{i-1} + \Delta$, where Δ is the number of seconds since the previous iteration. In the theatrical setting, the certainty of state duration information varies. When the information about duration, is very certain we set σ to a small value, and when the information is uncertain, we set σ to a large value. This has the practical consequence that particles spread across a larger area of the state space when the duration information is weak, which makes the algorithm more sensitive towards the input. When σ is small, the particles tend to form groups, which makes the algorithm less sensitive to the input.

If we assume that the likelihood $p(\mathbf{x}_{1:i}|s_i = h)$ can be evaluated, then we are able to estimate moments of the distribution of t_i. In section 5.1, we present the evaluation of this likelihood.

In order to determine which actions the computer actors are to perform, we compute the probability of an action a_c being performed. To compute this probability, we introduce a manuscript $p(a_c|t_i)$ which describes the computer actor's actions.

$$p(a_c|\mathbf{x}_{1:i}) = \int p(a_c|t_i)p(t_i|\mathbf{x}_{1:i})\mathrm{d}t_i. \tag{5.1}$$

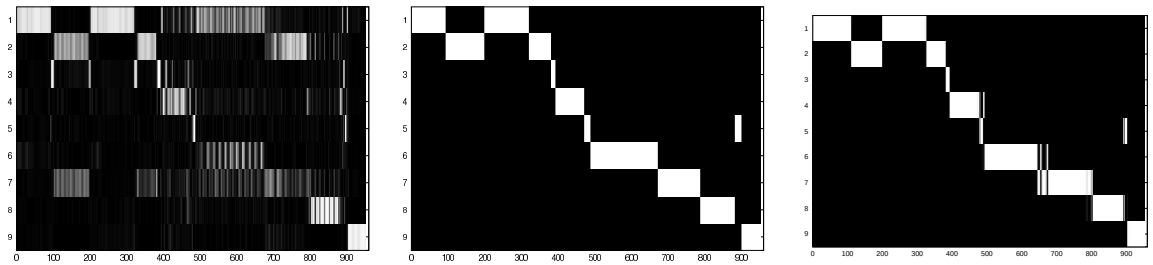
Since this is a moment of $p(t_i|\mathbf{x}_{1:i})$, we can evaluate it using (3.1). The manuscript $p(a_c|t_i)$ is specified in the same way as the manuscript of the human actor.

5.1 Gesture Recognition

To be able to use the algorithm, we need a measurement $p(\mathbf{x}_{1:i}|s_i = h)$. In the theatrical setting, we have chosen to use a gesture recognition system to provide this likelihood. More specifically, 13 different gestures are recognised. The recognition system is based on "Motion History Images" as described by Bobick and Davis [2]. Briefly put, these images reduce a sequence of black and white images to one gray-scale image that describes the current action. The gesture is then recognised by the matching of this image to a set of templates, which each describe an action. This matching is performed by computing global image features F_i of the current image and afterwards computing the likelihood of each action. Here, we assume that each action class is a Normal distribution, i.e. $p(\mathbf{x}_{1:i}|s_i = h) = \mathcal{N}(F_i|\mu_h, \Sigma_h)$. Bobick and Davis [2] use Hu moments as global features, but we have improved results greatly by using Zernike moments [8] instead. In fact, when using Hu moments the success-rate is 30%, whereas the success-rate is 60% when using 15 Zernike moments.

5.2 Results

To actually produce the play, it is essential to know when the computer has to perform an action. Using the methods described so far, we can compute the probability of an action. The actual decision algorithm is based on a simple threshold of this probability. If the probability of an action exceeds 60%, it is performed. By means of this simple rule, we have been able to produce a theatrical play successfully. The play lasts approximately 10 minutes, during which the computer performs 68 different actions. The play has been performed successfully several times, and could easily be extended to



(a) (b) (c)

Figure 3: Data from a short play designed for testing. The horizontal axis shows the image number, and the vertical axis shows the gesture number. (a) The likelihoods from the gesture recognition system. (b) The ground-truth data. (c) The resulting filtering distribution.

last longer. If the acting is to be convincing, timing is of the essence. Using the presented algorithm, the timing is actually surprisingly good, and, from a practical point of view, the actions are nearly performed at exactly the right time. Video material is available at the project website[1].

To get a better understanding of the quality of the algorithm, a shorter and more simple play has been produced. This consists of 9 different gestures performed by a human actor in a period lasting approximately 2 minutes. The actual likelihood $p(\mathbf{x}_{1:i}|s_i)$ is shown in figure 3a. This should be compared with the ground-truth data presented in figure 3b. As appears, the data contains a fair amount of noise, but the true pattern is visible. In figure 3c, the filtering distribution $p(s_i|\mathbf{x}_{1:i})$ is shown. As can be seen, the results are quite good, and even short actions with poor likelihoods are detected. It should be noted that the used manuscript $p(s_i|t_i)$ encodes the order of the actions correctly, but the durations are not perfectly aligned with the signal.

6 Conclusion and Future Work

This paper has presented an efficient algorithm for modelling duration in hidden Markov models. This algorithm is based on the optimal particle filter. It seems to be the first algorithm where duration is modelled explicitly with a continuous variable. It also seems to be one of the few cases where the optimal particle filter can be implemented. The algorithm was originally designed to produce a theatrical play, in which autonomous computer actors play together with a human actor. Such a play has been successfully produced, and the algorithm has proved to be very stable. It would, however, be quite interesting to see how well the algorithm would work with more traditional problems, e.g. speech recognition. Before such experiments can be performed, it would, however, be necessary to develop algorithms for learning manuscripts $p(s_i|t_i)$.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, August 2006.

[2] Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.

[1] `http://image.diku.dk/projects/robotics/theater/`

[3] O. Cappé, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95:899–924, May 2007.

[4] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[5] M.T. Johnson. Capacity and complexity of hmm duration modeling techniques. *Signal Processing Letters, IEEE*, 12:407–410, May 2005.

[6] Claudio S. Pinhanez and Aaron F. Bobick. ”it/i”: a theater play featuring an autonomous computer character. *Presence: Teleoper. Virtual Environ.*, 11(5):536–548, 2002.

[7] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[8] C.H. Teh and R.T. Chin. On image analysis by the methods of moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):496–513, 1988.