Hand Gesture Recognition, Prediction, and Coding Using Hidden Markov Models

by

Katerina H. Nguyen

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
MASTER OF ENGINEERING
in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
May 1996

Author ______________________________
Department of Electrical Engineering and Computer Science
May 28, 1996

Certified by ______________________________
Aaron F. Bobick
mputational Vision
Thesis Supervisor

Accepted by ______________________________
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Hand Gesture Recognition, Prediction, and Coding Using Hidden Markov Models

by

Katerina H. Nguyen

Submitted to the
Department of Electrical Engineering and Computer Science

May 28, 1996

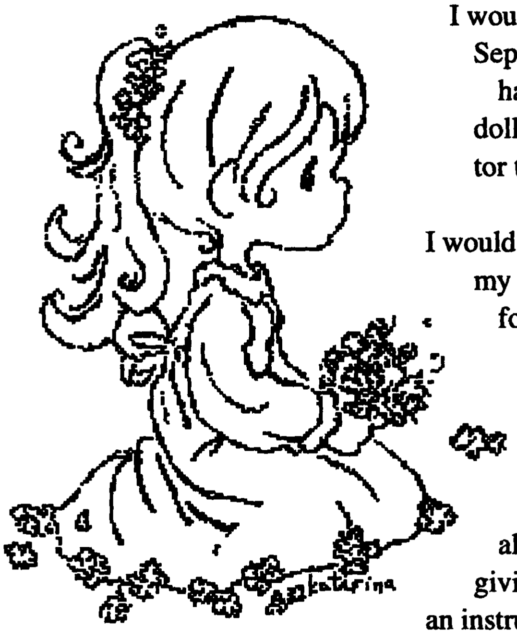
In Partial Fulfillment
of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis is a feasibility study to determine whether knowledge of gestures can be used to improve coding for the transmission of hand gestures. We implemented a coding system which uses hidden Markov modeling (HMM) to predict the identity of the ongoing gesture. If a prediction can be made with reasonable certainty, only a gesture ID code is sent. At the receiver, an appropriate hand image is selected from the identified prototype gesture sequence to be the "reconstructed" image. If a prediction cannot be made, the system uses principal component analysis (PCA) to compress and reconstruct the actual hand image. In this case, 50 projection coefficients must be sent. We evaluated the system using two different gestures. We found that when tested on unseen sequences of the two gestures, the system can reconstruct the hand using only the gesture ID code 97.5% of the time. Image sequences reconstructed from the prototype gestures also appear cleaner and more natural than sequences reconstructed using PCA.

Thesis Supervisor: Aaron F. Bobick
Title: Assistant Professor of Computational Vision

Acknowledgments



I would like to thank Aaron for taking a giant leap of faith back in September and for always willing to part with his "two-cents". I have learned a lot this past year, and I think I have more than a dollar's worth of wisdom now. Some day I hope I can be a mentor to someone as Aaron has been to me.

I would like to thank Ann for her juicy gossip to keep me sane during my final thesis hours, for inducting me into the witches' coven, for dragging me to dinner in the middle of thunderstorms, and for always believing in me. Ann has been better for my self-esteem than a whole roll of Smarties.

My sincere "gracias" goes to Nuria for always checking my happiness meter and for being only a zephyr away. I would also like to thank Donald for proof-reading this thesis and for giving me an unlabeled Pandora's box. I wish it had come with an instruction manual though.

I am thankful to Sue, who has been my cultural pill and whose friendship and understanding have been a great comfort to me through all my years at MIT. My gratitude also goes to Funmi for putting things in perspective and for always nagging me about the dreaded "T" word. I would like to thank my wonderful suitemates, Rosanna, Cathy, Mary, and Winnie, for being so forgiving when I neglected my kitchen duties and for being willing listeners. Finally, I dedicate this thesis to my parents and brood of siblings, who think it is absolutely okay for me to take a couple more terms to finish my thesis!

Contents

1 **Introduction** **7**
- 1.1 Scope of Thesis .8
- 1.2 Overview .9

2 **Background** **10**
- 2.1 Kalman Filtering .10
- 2.2 Principal Component Analysis .12
  - 2.2.1 Eigenspace Construction .12
  - 2.2.2 Reversible representation .14
- 2.3 Hidden Markov Modeling .15
  - 2.3.1 HMM Algorithms .16
  - 2.3.2 Types of HMMs .19
- 2.4 Previous Work .21

3 **System Description** **23**
- 3.1 Image Preprocessing .24
  - 3.1.1 Hand Tracking and Segmentation .25
  - 3.1.2 Feature Extraction .27
- 3.2 Gesture Coding .30
  - 3.2.1 Knowledge-free PCA Coding .31
  - 3.2.2 Knowledge-based HMM Coding .32
- 3.3 Gesture Reconstruction .34
  - 3.3.1 Knowledge-free PCA Reconstruction .35
  - 3.3.2 Knowledge-based HMM Reconstruction .36

4 **System Evaluation** **39**
- 4.1 Gesture Data .39
- 4.2 PCA Approach .40
  - 4.2.1 Eigenspace Selection .40
  - 4.2.2 Final PCA Results .43
- 4.3 HMM Approach .45
  - 4.3.1 Model Selection .45
  - 4.3.2 Prototype Gesture Mapping .47
  - 4.3.3 Final HMM Results .48
- 4.4 Combined System .50

5 **Conclusion** **52**

5.1 Summary 52
5.2 Future Work 54

Bibliography **56**

A Implementation **58**

List of Figures

2-1 Timing diagram for the discrete Kalman filter 12
2-2 First-order Markov process 15
2-3 Forward algorithm for HMM evaluation. 16
2-4 Backward algorithm for HMM evaluation 17
2-5 Viterbi algorithm for HMM evaluation and decoding. 18
2-6 Baum-Welch algorithm for reestimation of model parameters 19
2-7 Types of hidden Markov models (HMMs):. 20
3-1 Schematic of the coding and transmission system 24
3-2 Schematic of the image preprocessor 25
3-3 Typical segmentation results 26
3-4 Examples of where the segmentation algorithm fails 27
3-5 Schematic of the dual-system gesture coder 31
3-6 Schematic of the twofold gesture reconstructor 35
4-1 Low-pass effect of PCA reconstruction. 42
4-2 Comparison of innovation space and hand space 42
4-3 Modified schematic of the PCA coding and reconstruction system 43
4-4 Comparison of the "self", "side", and combination spaces. 44
4-5 Transition diagrams for the "self" and "side" HMMs 47
4-6 Mapping trajectories 48
4-7 A "self" sequence reconstructed using prototype gestures 49
4-8 Example of the higher quality of a HMM-reconstructed sequence. 50
4-9 Sequence reconstructed using the combined PCA-HMM system. 50

Chapter 1

Introduction

Gestures are repeated actions by humans that follow a general pattern. Most current research efforts in gesture recognition concentrate on identifying a gesture *after* seeing the complete execution of the gesture's pattern. The goal of this thesis is to determine whether a system can identify a gesture "on-the-fly" without having to see an action in its entirety. A human observer, for instance, recognizes the act of sitting down within the first few tenths of a second in which a person bends his knees and leans back. If a computer can likewise predict the most likely gesture by observing only the onset, the computer would become much more responsive. This ability to automatically anticipate or predict the user's action has high entertainment as well as practical value.

One practical yet interesting application of automatic gesture prediction involves efficient model-based image coding for transmission purposes, such as video teleconferencing. In this scenario, once the system has observed enough data to predict the most likely gesture, it would have hints on the appearance of the rest of the gesture, or at least of the next few frames. If this knowledge is available at both the transmitter and receiver, an interesting coding scheme can be devised to take advantage of this information and significantly reduce the amount of transmitted data.

The work described here is a feasibility study to determine whether knowledge of gestures can be used to improve coding for the transmission of hand gestures.

1.1 Scope of Thesis

Most current commercial video teleconferencing systems use block-by-block motion-compensated interpolation or other MPEG-standard techniques [12]. We design a dual-system framework, which includes a knowledge-free system and a knowledge-based system. By knowledge-free, we mean that the system has no concept of what it means to be a hand gesture. The same system design, for instance, can be used to code buildings or the motion of sailboats. In contrast, the knowledge-based system uses specific gesture knowledge and hence is not easily extensible to non-gesture applications.

The two systems described run in parallel, and their purpose is to convert input video into a more condensed form for transmission. The knowledge-free system uses principal component analysis (PCA) to handle compression and reconstruction of hand gestures. The knowledge-based system uses hidden Markov modeling (HMM) to recognize the ongoing gesture. Once the HMM system can predict the current action with reasonable certainty, it needs to transmit only the gesture identification code and a few update parameters. Reconstruction of the rest of the gesture is accomplished at the receiver using dynamic time warping of a stored prototype sequence for the gesture. The intention is for the knowledge-free system to handle data transmission while the knowledge-based system has not seen enough data to reliably identify the current action or while no gesture is being performed. When a reliable prediction can be made, transmission is turned over to the knowledge-based system to achieve lower bit rates.

Our primary focus is to determine whether automatic gesture prediction is possible and whether this knowledge can be used intelligently to accomplish more efficient coding for transmission of hand gestures. Thus, we are not concerned about the whole image; rather, we concentrate our attention on the hand segmented from the scene. Coding and reconstruction apply only to the segmented hand on a black background. Furthermore, to avoid discussing the intricacies of the actual data transmission, we assume standard encoders and decoders are available and that variable-length codewords are used to achieve bit rates equal to the entropy of the “message”, as in Huffman coding.

Our implementation includes: a preprocessor which uses Kalman filtering to automatically segment the hand from the input video, a knowledge-free coder and reconstructor, and a knowledge-based coder and reconstructor. The design of these system components are described in detail in a later chapter.

1.2 Overview

The thesis is organized as followed. First, Chapter 2 provides background information on the mathematical algorithms and models used. Specifically, Kalman filtering, principal component analysis, and hidden Markov modeling are discussed. A brief summary of previous related works on gesture recognition is also given in Chapter 2. Next, Chapter 3 describes the design as well as the designing issues of our video coding system for the transmission of hand gestures. Chapter 4 describes the data used to train and evaluate the system. Results and analysis of the system evaluation are included. Finally, summary of the work and discussion of future extensions are presented in Chapter 5.

Chapter 2

Background

Before presenting the system design, it is necessary to establish an understanding of the algorithms and models used. First, we provide a brief description of Kalman filtering, which is used in this thesis to track the hand from frame to frame and to "smooth out" the measurements of the hand features. Principal component analysis and its application to data compression are then discussed. Next, we explain the mechanics of hidden Markov modeling, which is used for gesture recognition. Finally, we conclude this chapter by presenting a survey of previous works which guide the direction of this thesis.

2.1 Kalman Filtering

Kalman filtering is the standard technique for obtaining optimal linear estimates of the state vector of a dynamic model from measurements corrupted by noise. For Gaussian noises, Kalman estimates are the maximum-likelihood estimates. For non-Gaussian noises, they are the minimum mean-square-error estimates.

In the discrete case, the state of the dynamic system is governed by the following equations:

$$\begin{aligned} z_k &= H_k x_k + v_k \\ x_{k+1} &= \Phi_k x_k + w_k \end{aligned}$$

The first equation gives the measurement model, which is the relationship between the state vector

x and the measurements z at time k. If the measurements were exact, the equation reduces to $z_k = H_k x_k$. However, since the measurements are corrupted by noise, the error term v_k is needed. The second equation yields the system dynamic model. The change in state as time goes forward is governed by a known law $x_{k+1} = \Phi_k x_k$. But again, there may be error in the model, which is accounted for by the term w_k. In general, v_k and w_k are modeled as Gaussian noises with zero mean and covariance matrices R_k and Q_k respectively. It is also generally assumed that v_k and w_k are uncorrelated at all time ($\mathrm{E}\left[v_i w_j^{\mathrm{T}}\right] = 0$ for all i, j).

Kalman filtering provides a recursive way for estimating the state vector of a dynamic system defined by the above equations. The recursion involves two steps: prediction and update. At time k-1, a predicted estimate of the state vector for time k is made. Then, at time k, the estimate is updated using incoming measurement z_k, and a new prediction is made for time k+1.

Kalman's recursive algorithm is presented below. For a detailed derivation of the algorithm, [5] is recommended.

Prediction:

$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+)$$

$$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}{}^{\mathrm{T}} + Q_{k-1}$$

Update:

$$K_k = P_k(-)H_k{}^{\mathrm{T}}[H_k P_k(-)H_k{}^{\mathrm{T}} + R_k]^{-1}$$

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k[z_k - H_k\hat{x}_k(-)]$$

$$P_k(+) = [I - K_k H_k]P_k(-)$$

In the above algorithm, $\hat{x}_k(-)$ denotes the *predicted* state vector for time k, and $\hat{x}_k(+)$ denotes the *updated* estimate after taking into account the measurement z_k. P is the error covariance, I is the identity matrix, and K is the Kalman gain. The error covariance ($P = \mathrm{E}[\,(\hat{x} - x)\ (\hat{x} - x)^{\mathrm{T}}]$) provides an indication of the accuracy of the estimate. Large P indicates large measurement noise and hence not so reliable estimates. Conversely, a small P means that the estimates are close to

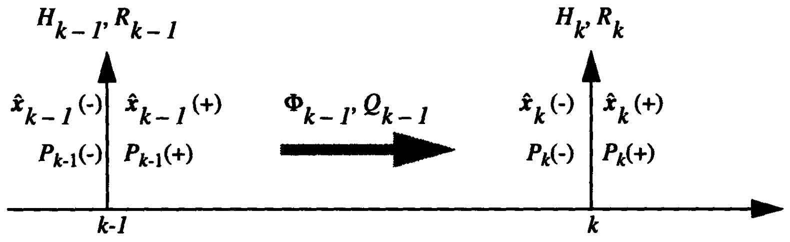


Figure 2.1 Timing diagram for the discrete Kalman filter.

their true values. The Kalman gain matrix K is essentially the ratio of the uncertainty in the state estimate to the uncertainty in the measurement. From the first update equation, K determines how much of the innovation $[z_k - H_k\hat{x}_k(-)]$ is used to correct the predicted state estimate $\hat{x}_k(-)$. The timing diagram for the predict-update recursion is shown in Figure 2.1.

2.2 Principal Component Analysis

In this thesis, principal component analysis (also known as Karhunen-Loeve expansion) is used to transform high-dimension hand images into a small number of coefficients. To accomplish this, PCA is first applied to a set of training images to construct an image space. The axes of this space are statistically uncorrelated eigenimages. Usually the M' most significant subset of these eigenimages (those corresponding to the largest eigenvalues) are chosen to span the space. By projecting a novel image onto the M' axes of this space, we obtain M' unique coefficients that characterize the image. The process is reversed by reconstructing the original image from the projection coefficients and their associated space. The reconstructed image is in fact the minimum-mean-square-error estimate of the original. Below we discuss the mathematics dealing with eigenspace construction and reversible PCA representation of new images.

2.2.1 Eigenspace Construction

Principal component analysis basically seeks a set of orthonormal vectors which best accounts for the distribution of the data in a given training set. If Γ_k is an NxN image, we can think of Γ_k as a vector of dimension N^2. Thus, a set of M training images $[\Gamma_1, \Gamma_2, \ldots, \Gamma_M]$ makes up a matrix of size N^2xM. If we define the average image as $\boldsymbol{m} = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$ and the difference between each training image and the average image as $\boldsymbol{d}_k = \Gamma_k - \boldsymbol{m}$, then the desired orthonormal vectors $\boldsymbol{e}_k$ are chosen such that

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}\left(\boldsymbol{e}_k^{\mathrm{T}}\boldsymbol{d}_n\right)^2$$

is maximum subject to

$$\boldsymbol{e}_j^{\mathrm{T}}\boldsymbol{e}_k = \begin{cases} 1 & j = k \\ 0 & \text{otherwise} \end{cases}$$

Solving the above set of equations is equivalent to finding the eigenvectors $\boldsymbol{e}_k$ and the eigenvalues λ_k of the covariance matrix

$$C = \frac{1}{M}\sum_{n=1}^{M}\boldsymbol{d}_n\boldsymbol{d}_n^{\mathrm{T}} = \frac{1}{M}(DD^T)$$

where the matrix $D = [\boldsymbol{d}_1\ \boldsymbol{d}_2 \ldots \boldsymbol{d}_M]$.

Since C is N^2xN^2, determining the N^2 eigenvectors and eigenvalues is difficult if not intractable. The computation is on the order of O(N^6). However, if the number of training images M is less than the dimension of the image space ($M < N^2$), there can be only M-1 meaningful eigenvectors $\boldsymbol{e}_k$[15]. The remaining eigenvectors have associated eigenvalues of zero. We can solve for these meaningful, N^2-dimensional eigenvectors by first solving for the eigenvectors $\boldsymbol{u}_k$ of an MxM matrix and then taking appropriate linear combinations of the difference images $\boldsymbol{d}_k$.

To be more mathematically precise, consider the MxM matrix $D^{\mathrm{T}}D$. Its M eigenvectors $\boldsymbol{u}_k$ are defined as $D^{\mathrm{T}}D\boldsymbol{u}_k = \mu_k\boldsymbol{u}_k$. Left-multiplying both sides by D, we obtain $DD^{\mathrm{T}}D\boldsymbol{u}_k = \mu_k D\boldsymbol{u}_k$, which states that Du_k are the eigenvectors of $C = \frac{1}{M}DD^{\mathrm{T}}$. Hence, the meaningful eigenvectors e_k and eigenvalues λ_k of the covariance matrix are

$$e_k = Du_k = \sum_{n=1}^{M} u_{kn}d_n \qquad k = 1, \ldots, M$$

$$\lambda_k = \begin{cases} \mu_k & k = 1, \ldots, M \\ 0 & \text{otherwise} \end{cases}$$

The computation is reduced from $O(N^6)$ to $O(M^3)$. In practice, $M << N^2$, and calculating the eigenvectors and eigenvalues of the $N^2 x N^2$ covariance matrix is computationally feasible. In this particular application, the eigenvectors are linear combinations of the original hand images. Hence, they are hand-like in appearance and may be referred to as "eigenhands". The eigenhands can be ranked according to their associated eigenvalues. Those that have the largest eigenvalues account for the most variance within the set of training images and are therefore most useful.

In practice, only the best M' out of M eigenvectors are used to approximate the original space. This reduces the number of coefficients and the amount of computations needed to represent a novel image. Issues relating to using PCA as a compact representation of images are discussed next.

2.2.2 Reversible representation

Given a set of M' eigenvectors, a new image Γ is transformed into its PCA components by the simple projection operation

$$\omega_k = e_k^{\mathrm{T}}(\Gamma - m) \qquad k = 1, \ldots, M'$$

The above equation involves pixel-by-pixel (N^2) image multiplications and additions. The projection coefficients form a vector $\Omega = [\omega_1, \omega_2, \ldots, \omega_{M'}]$, which describes the contribution of each eigenvector in representing the new image. Put another way, Ω is a measure of how similar the new image is to each of the eigenvectors.

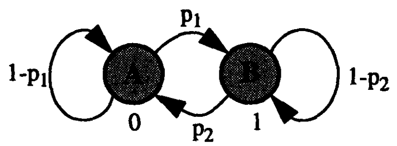


Figure 2.2 First-order Markov process. State A always outputs a "0", and state B always outputs a "1".

To reconstruct the image from the projection coefficients, we reverse the above operation:

$$\Gamma' = m + \sum_{n=1}^{M'} \omega_n e_n$$

If M' is equal to M (the number of meaningful eigenvectors) and there is no quantization, then the process is truly reversible. Otherwise, the reconstructed image Γ' is only an approximation of the original image Γ. Using the eigenvalues, we can determine the minimal set of eigenvectors which approximates the image space and maintains a certain percentage of the variance seen in the training examples.

2.3 Hidden Markov Modeling

The hidden Markov model is a stochastic process built on top of another stochastic process, the Markov process. A time domain process exhibits first-order Markov property if the conditional probability density of the current event, given all past and present events, depends only on the most recent event. It is conventional to represent a Markov process using a state transition diagram such as the one shown in Figure 2.2. Notice that the output at each state is deterministic, and the state sequence can be directly extracted from the observation sequence.

As defined, the Markov model is too restrictive to be widely applicable. The HMM, on the other hand, is a powerful extension of the Markov model and can be applied to many problems of interest. In a hidden Markov model, the output for each state corresponds to an output probability distribution instead of a deterministic event. Thus, the state sequence is no longer directly observable. It is *hidden* behind a layer of observable stochastic processes.

```
1. $\pi_i = 1/N_I$ for all states $i \in S_I$, and 0 otherwise.
2. Find $\alpha$ for all states $i$ at time $t$=1:
       $\alpha_1(i) = \pi_i b_i(O_1)$
3. Calculate $\alpha$ for all states $j$ at time $t = 2, \ldots, T$:
       $\alpha_t(j) = \left[\sum_i \alpha_{t-1}(i) a_{ij}\right] b_j\left(O_t\right)$
4. Final probability:
       $\Pr(O|\lambda) = \sum_{i \in S_F} \alpha_T(i)$
```

Figure 2.3 Forward algorithm for HMM evaluation.

Below we discuss the algorithms generally used for lifting the veil between the observer and the state sequence as well as the different types of HMMs. For more detail on these topics, references [7], [10], and [11] are recommended.

2.3.1 HMM Algorithms

An HMM can be represented compactly as $\lambda = [A, B, \pi]$, where A is the state transition probability distribution, B is the set of output probability distributions corresponding to the different states, and π is the initial state distribution. Given the definition of HMMs, there are three main problems to be solved: evaluation, estimation, and decoding.

Evaluation determines, out of several competing HMMs, which model best matches a given sequence of observations. The evaluation problem can be expressed as $max_i\,[\Pr(O|\lambda_i)]$, where O is the observation sequence. The most straightforward way to compute $\Pr(O|\lambda)$ is to sum the probability over all possible state sequences in the model for the observation sequence. Hence, $\Pr(O|\lambda) = \sum_i \Pr(O|S_i, \lambda) \Pr(S_i|\lambda)$. The computational load of this direct approach, however, is on the order $O(2TN^T)$ where N is the number of states of the HMM and T is the number of time steps in the observation sequence.

A more efficient algorithm, with $O(N^2T)$, is the forward-backward algorithm. First, we define the forward variable $\alpha_t(i)$ as the probability of the partial observation sequence up to time t and

```
1. $\beta_T(i) = 1/N_F$ for all states $i \in S_F$, and 0 otherwise.
2. Find $\beta$ for all states $j$ at time $t = T-1, T-2, \ldots, 1$:
       $\beta_t(j) = \sum_i a_{ji} b_i(O_{t+1}) \beta_{t+1}(i)$
3. Final probability:
       $\Pr(O|\lambda) = \sum_{i \in S_I} \pi_i b_i(O_1) \beta_1(i)$
```

Figure 2.4 Backward algorithm for HMM evaluation.

state *i*, given the model λ. Similarly, the backward variable $\beta_t(i)$ is defined as the partial observation sequence from time *t*+1 to *T*, given state *i* at time *t* and the model λ. Expressed mathematically, $\alpha_t(i) = \Pr(O_1, O_2, \ldots, O_t, s_t = i|\lambda)$ and $\beta_t(i) = \Pr(O_{t+1}, O_{t+2}, \ldots, O_T | s_t = i, \lambda)$.

Figure 2.3 and Figure 2.4 illustrate how the forward and backward variables are used to find $\Pr(O|\lambda)$ respectively. Alternatively, the forward and backward variable can also be used together to find $\Pr(O|\lambda) = \sum_i \alpha_t(i) \beta_t(i)$.

In practice, the Viterbi algorithm is generally used for evaluation at recognition time. Viterbi can be viewed as a special form of the backward-forward algorithm, where only the optimal path at each time step is taken instead of all paths. Thus, Viterbi finds $max_i[\Pr(O, S_i|\lambda)]$ rather than $\sum_i \Pr(O, S_i|\lambda)$. Fortunately, the probabilities obtained from the forward-backward and Viterbi algorithms have been shown experimentally to be very close.

The Viterbi method is extremely efficient since it does not take into account all possible paths. Also, the Viterbi algorithm yields the optimal state sequence as a by-product. Hence, evaluation and decoding can be done in a single step. The complete Viterbi algorithm is shown in Figure 2.4.In this thesis, evaluation at recognition time will be done using the Viterbi procedure.

Estimation is by far the most difficult and time-consuming problem of hidden Markov modeling. Given an observation sequence, estimation deals with adjusting the model parameters λ, such that $\Pr(O|\lambda)$ is maximized. Unfortunately, there is no known way to solve this analytically. Instead, an iterative algorithm called the Baum-Welch algorithm is generally used for estimation.

```
1. Initialization: for all states $i$,
       $\delta_1(i) = \pi_i b_i(O_1)$    $\Psi_1(i) = 0$
2. Recursion: for all states $j$, from time $t$=2 to $T$,
       $\delta_t(j) = \mathrm{Max}_i[\delta_{t-1}(i) a_{ij}] b_j(O_t)$    $\Psi_t(j) = \mathrm{argmax}_i[\delta_{t-1}(i) a_{ij}]$
3. Termination:
       $P^* = \mathrm{Max}[\delta_T(s)]$    $s_T^* = \mathrm{argmax}_{s \in S_F}[\delta_T(s)]$
4. Recovery of state sequence: from $T$-1 to 1,
       $s_t^* = \Psi_{t+1}(s_{t+1}^*)$
```

Figure 2.5 Viterbi algorithm for HMM evaluation and decoding.

The Baum-Welch method is proven to converge to locally optimal HMM parameters for a given set of training data.

The Baum-Welch algorithm starts by first assuming a set of model parameters. Then, the forward-backward procedure is used to evaluate the probability that the given observations were generated by this initial set of model parameters. Using the probability obtained, we can update our estimates of the model parameters and repeat the whole process. The reestimation or training process ends when the model parameters converge.

Before expressing the Baum-Welch algorithm mathematically, it is convenient to introduce the variables $\xi_t(i,j)$ and $\gamma_t(i)$. The first variable $\xi_t(i,j)$ is the a posteriori probability of going from state i to state j at time t, given the observation sequence and a set of model parameters, and the second variable $\gamma_t(i)$ is the a posteriori probability of being in state i at time t, given the observation sequence and model.

Given the definitions of $\xi_t(i,j)$ and $\gamma_t(i)$, the Baum-Welch algorithm can be expressed as shown in Figure 2.6. Note that in the reestimation procedure we used the forward-backward algorithm instead of the Viterbi algorithm to find $\Pr(O|\lambda)$. This is because the advantage of Viterbi over the backward-forward algorithm is efficiency at the cost of accuracy. But since training can be done beforehand, efficiency is much less of an issue compared to accuracy.

```
1. Find $\gamma_t(i,j)$ and $\gamma_t(i)$ :
   $\gamma_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \beta_{t+1}(j)}{\sum_{k \in S_F} \alpha_T(k)}$    $\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{k \in S_F} \alpha_T(k)}$
2. Calculate $\bar{a}_{ij}$ and $\bar{b}_j(k)$ :
   $\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$    $\bar{b}_j(k) = \frac{\sum_{t \in O_t = v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$
3. Determine $\pi_i = \gamma_1(i)$ .
4. Repeat until model $\bar{\lambda}$ converges.
```

Figure 2.6 Baum-Welch algorithm for reestimation of model parameters.

The version of the Baum-Welch algorithm shown in Figure 2.6 is for a single observation sequence. In order to train the HMMs adequately, a set of independent observation sequences from the same source is needed. The Baum-Welch algorithm shown can be easily extended to include multiple observation sequences.

Decoding deals with recovering the *hidden* part of the model. It is the problem of finding the most likely state sequence corresponding to the observation sequence. As mentioned earlier, the Viterbi algorithm yields the state sequence as a by-product of the evaluation process (step 4 in Figure 2.4).

2.3.2 Types of HMMs

So far we have been ambiguous about the nature of the output probability distributions B. These distributions can be either discrete, continuous, or a combination of both. This leads to three different types of HMMs, namely discrete, continuous, and semi-continuous.

In the discrete case, the observations are discrete symbols of a finite set. Examples of discrete observations are the results of flipping a coin, course grades, and shoe sizes. Each state of the

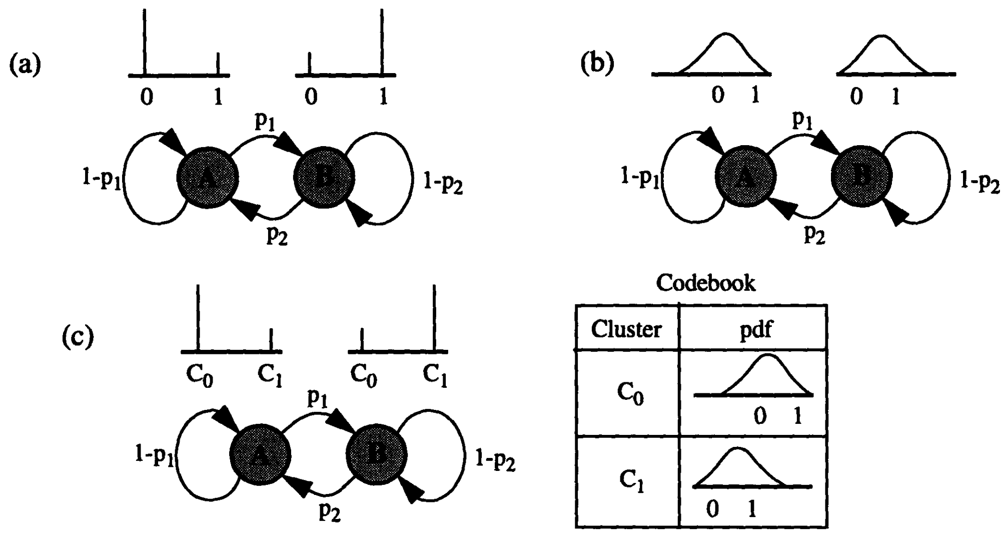


Figure 2.7 Types of hidden Markov models (HMMs): (a) discrete; (b) continuous; and (c) semi-continuous. (After [14])

HMM has a discrete probability mass distribution to describe the likelihood that the state would produce a particular symbol. Figure 2.7a shows a discrete HMM with two states and two output symbols. As shown, state A is more likely to output a "0" than a "1". Conversely, state B is more likely to produce a "1" than a "0".

In practice, most processes do not output discrete symbols. Instead, the continuous outputs are quantized to fit in one of a set of predefined classes. Thus, we can think of the finite set of discrete symbols as quantization levels in a vector quantization (VQ) codebook. Quantization, however, can lead to serious quantization errors which would distort subsequent hidden Markov modeling. Another disadvantage comes from the fact that HMMs and the VQ codebook are modelled separately, which may not result in an optimal combination for pattern classification. Also, discrete distributions can only model events seen in the training data. Thus, to obtain a good discrete model, a large amount of training data is needed.

In the continuous case, the observations come from a continuum of values in some multi-dimensional Euclidean space, instead of a finite set. Thus, each state of the continuous HMM has a mixture of probability density functions to describe the likelihood that the state would produce a particular value. Mixtures of Gaussian densities are typically used to model the output probabilities. Figure 2.7b shows an example of a continuous HMM. Again, there are two states, but there are an infinite number of possible observations (all values between "0" and "1").

Using continuous probability densities allow for interpolation between seen events, thus alleviating the sparse training data problem. However, continuous HMMs require a considerable increase in computational complexity compared to the discrete case. The number of free parameters to be estimated from the training data is also much higher.

The semi-continuous HMM incorporates features from both the discrete and continuous HMMs. Like the discrete HMM, the observation vector is quantized into a set of finite classes, thus reducing the number of free parameters. But like the continuous HMM, the classes are modelled by multivariate Gaussian densities. This helps remove distortion due to quantization as well as avoid the sparse data problem. Figure 2.7c shows a two-state semi-continuous HMM with its corresponding codebook. In this thesis, we will use continuous HMMs with output probabilities modelled by mixtures of Gaussian densities.

2.4 Previous Work

Using HMMs for vision is a relatively new approach. A few works related to visual applications of HMMs and hand gesture recognition are summarized below. Most early vision work with HMMs dealt with handwriting recognition, such as [8]. In work related to human motions, Yamato et al [18] used discrete HMMs to recognize six different tennis swings performed by three different subjects. The observation or feature vector used was the quantized, subsampled, 25x25 pixel image. Respectable recognition rates were reported, although the system required the swings to be temporally segmented first.

Due to the expressiveness of hands, several systems have been developed specifically for recognizing hand gestures. One such system, developed by Darrell and Pentland, used a set of view models to represent the hand [4]. Typical sequences of hand views for the different gestures were determined and stored. Recognition of a novel hand gesture was performed by matching its space-time pattern to the stored gesture templates using dynamic time warping and normalized correlation. An accuracy rate of 96% was reported for the gestures “hello” and “good-bye”.

A slightly more ambitious system was developed by Cui et al [3] for recognizing 28 different hand signs. Here, the features were essentially the unit eigenvectors and eigenvalues obtained from principal component analysis. The authors used learned decision boundaries within the feature space to distinguish between the different hand signs.

A system which used continuous HMMs specifically for hand gesture recognition was developed by Starner and Pentland for recognizing simple sentences in American Sign Language [13]. The feature vector in this case consisted of the position, orientation, and eccentricity of the bounding ellipse of each hand. The system achieved high recognition accuracy rate but imposed a very strict grammar on possible sentences. Our system is similar to this system since we also use continuous HMMs, and our feature vector also includes the hand’s position, orientation, and bounding ellipse measurements.

Wilson and Bobick also implemented an HMM-based system for gesture recognition [16]. However, instead of having one feature set, they used multiple independent representations. At each state, a state-membership measure was used to combine the different representations or feature subspaces. Training involved the concurrent reestimation of the model subspaces and the traditional HMM parameters via the Baum-Welch algorithm.

In a separate paper, Bobick and Wilson defined a gesture as a sequence of states in a measurable feature space [2]. The states were aligned along a prototype trajectory obtained from example trajectories of the same gesture. This approach is significantly different from hidden Markov modeling in that it allows continuous tracking of a gesture within and between states. Thus, smooth gesture reconstruction from a prototype trajectory may be feasible.

Chapter 3

System Description

Our goal is to determine whether knowledge of hand gestures will lead to more efficient coding for transmission purposes, such as video teleconferencing. To this end, we implement a dual-system framework, which includes a knowledge-free system and a knowledge-based system. The knowledge-free system will also be called the PCA system since it uses principal component analysis to accomplish both coding and reconstruction of the hands. Similarly, the knowledge-based system will be alternately referred to as the HMM system because it uses hidden Markov modeling to carry out gesture coding. The intention is for the HMM system to handle the bulk of the transmission, and for the PCA system to serve as a substitute when the HMM fails to recognize the current hand gesture with reasonable confidence. A visualization of the combined framework is shown in Figure 3.1.

As seen from Figure 3.1, video images are first preprocessed at the transmitter. This preprocessing involves segmenting the hand from the scene and extracting the necessary hand features. The segmented hand I and extracted feature vector x are then fed to the PCA and HMM coders, which run in parallel. Note that the HMM coder does not require the segmented hand as an input since in the HMM system, coding and reconstruction depend on the hand's features rather than the actual hand image. At the switch, the output Ω_{HMM} from the HMM coder is selected unless the HMM confidence level is lower than a specified threshold. In that case, the output Ω_{PCA} from the

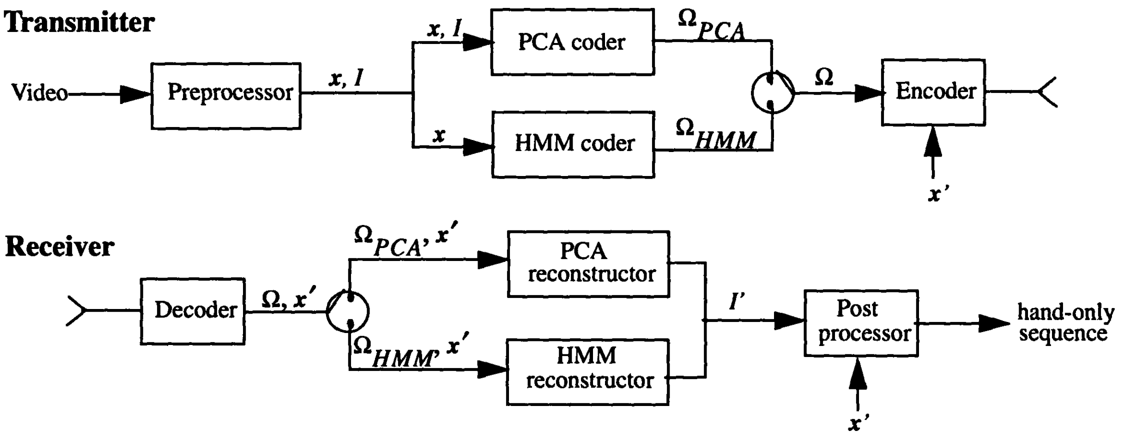


Figure 3.1 Schematic of the coding and transmission system. The HMM coder exploits knowledge of hand gestures to accomplish more efficient transmission.

PCA system is chosen. A subset of the feature vector x (denoted by x') and the selected output Ω along with its 1-bit system-identifying label are encoded and transmitted using a standard encoder.

At the receiver end, the transmitted data is decoded. Using the ID label, the switch determines which system should perform the reconstruction. The chosen system reconstructs the hand image I', hopefully with little distortion. Finally, postprocessing is applied to embed the reconstructed hand at its original location in an empty (black background) frame. This leads to a hand-only reconstructed video sequence, which can be used with the original sequence in a range of possible coding and reconstruction schemes.

The success of the system clearly depends on the ability to reliably recognize and predict hand gestures and the ability to intelligently incorporate gesture knowledge into a coding scheme. Below we detail the main components of the framework shown in Figure 3.1. Specifically, we describe the designs of the image preprocessor, the knowledge-free PCA gesture coder and reconstructor, and the knowledge-based HMM gesture coder and reconstructor. Implementation-specific issues are discussed in Appendix A.

3.1 Image Preprocessing

The purpose of the preprocessor is to extract the hand and its corresponding feature vector from an

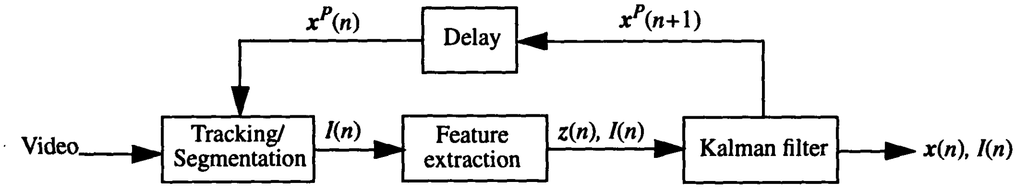


Figure 3.2 Schematic of the image preprocessor. Kalman filtering is used for tracking the hand as well as "smoothing out" the hand's feature measurements.

input video sequence. The schematic of the preprocessor is shown in Figure 3.2. First, the hand $I(n)$ is tracked and segmented from the original image. A set of feature measurements $z(n)$ is then extracted from the hand. Because of the inherent noise within the system, Kalman filtering is used to refine the feature estimates $x(n)$ as well as to predict the state of the feature vector at the next time step. The prediction $x^P(n+1)$ serves as a guide for the tracking and segmentation of the next frame. The refined feature vector is also used to normalize the segmented hand with respect to location, scale, and orientation. Below we discuss issues dealing with hand segmentation and feature extraction in more detail.

3.1.1 Hand Tracking and Segmentation

Tracking of the hand in a natural setting is difficult. Most successful hand tracking systems are constrained in some way. Some require a simple background or more than one camera. Others require the cameras to be at close range to the hand. Still others impose the use of wired sensors or colored gloves. The limitation of these systems is that they are either obtrusive to the user or they limit the user's natural movements. Most of these systems are also reconstructive; they try to recover the full 3D information of the hand.

For our application, complete 3D information is unnecessary. We need only an approximate location of the hand in order to segment it from the scene for further processing. Upon starting, the system needs to be initialized with the approximate centroid of the hand in the first frame. Kal-

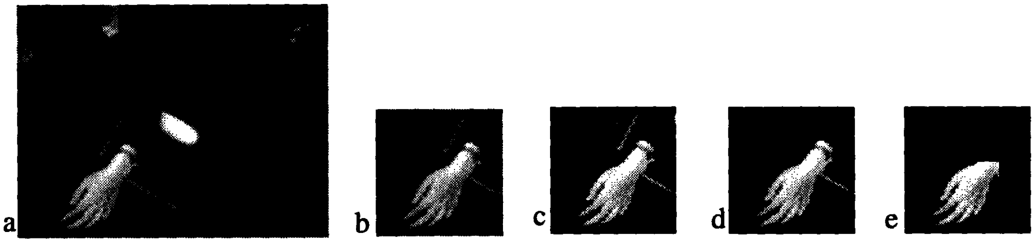


Figure 3.3 Typical segmentation results. (a) original frame; (b) cropped area around predicted centroid; (c) area after thresholding; (d) area after imposing 6-connectedness; and (e) area after imposing frame differencing.

man filtering is then used to predict the hand's centroid in successive frames, thus guiding the tracking of the hand.

From the centroid information, the system crops out an area of interest. Segmentation of the hand from this area is accomplished using thresholding, 6-connectedness constraint, and frame differencing. The outputs at the end of these segmentation steps are shown in Figure 3.3. It was found that a threshold of 115 works well for the 8-bit grayscale image sequences we have. The success of thresholding in this case is mainly due to the user's choice of dark-colored clothing. In a less favorable situation, we can think of using color images and thresholding for flesh color, which is the approach taken in [17]. Since thresholding the cropped area usually yields many unconnected regions, we impose a 6-connectedness constraint (as defined in [6]) to find the largest connected region, which in most cases includes the hand. Finally, we use frame differencing to define a "motion blob", which helps remove extraneous stationary objects that are connected to the hand. By using frame differencing, we assume the background is stationary and the user is relatively stationary except for his hand and arm movements. This is a valid assumption within the video teleconferencing scenario.

Our segmentation algorithm performs satisfactorily in most cases. The algorithm fails when the hand moves significantly from one frame to the next, due to the non-constant frame rate of our video capture program. The Kalman filter, however, can generally recover within one or two frames. Figure 3.4a shows a case where the algorithm fails due to a big "jump" in the hand's loca-

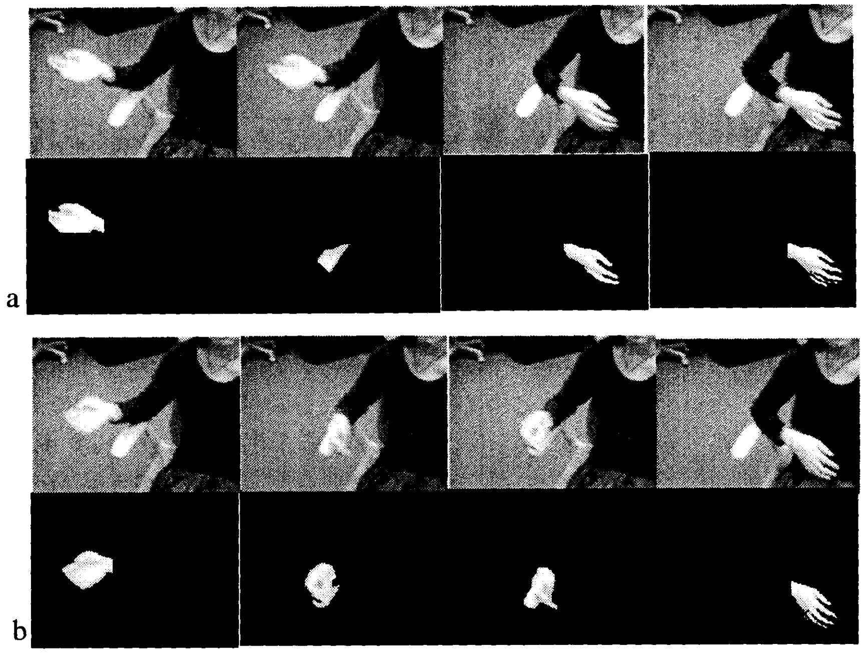


Figure 3.4 Examples of where the segmentation algorithm fails. Failure due to (a) significant hand motion and (b) hand occluding stationary objects.

tion. Notice how the system compensates in the third frame. Another place where the algorithm breaks down is when the hand occludes a stationary object in one frame but not the next, causing the algorithm to mistake the object as being part of the motion blob. An example of this is shown in Figure 3.4b.

Once segmentation of the hand is obtained, the next stage is to extract relevant features from the segmented hand.

3.1.2 Feature Extraction

The features we select to extract from each hand include the centroid, orientation, and scale. Calculations of these features are performed on a binarized version of the segmented hand. Finding the centroid (x_0, y_0) is straightforward:

$$x_0 = \frac{\sum_x \sum_y x b(x, y)}{\sum_x \sum_y b(x, y)}$$

$$y_0 = \frac{\sum_x \sum_y y b(x, y)}{\sum_x \sum_y b(x, y)}$$

where $b(x, y)=1$ for pixels belonging to the segmented hand and 0 otherwise.

The orientation and scale of the hand may be determined from the bounding ellipse around the hand. The major axis of the ellipse corresponds to the axis of least inertia and can be determined from the covariance matrix of the x and y coordinates of the hand pixels [6]. Mathematically, if the covariance matrix is

$$\begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

then a, b, and c are defined as

$$a = \sum_x \sum_y (x')^2 b(x, y)$$

$$b = 2\sum_x \sum_y x'y'b(x, y)$$

$$c = \sum_x \sum_y (y')^2 b(x, y)$$

where x' and y' are the x and y coordinates normalized with respect to the centroid (x_0, y_0).

The primary eigenvector of the covariance matrix corresponds to the normalized major axis of the bounding ellipse. Conversely, the secondary eigenvector corresponds to the normalized minor axis. The square roots of the eigenvalues yield the respective half-lengths l_M and l_m of the axes, which provide a measure of the size or scale of the hand. In addition, the angle of orientation θ of the hand (measured clockwise from the vertical) can also be computed from the primary eigenvector. In summary,

$$l_M = \sqrt{\frac{a + c + \sqrt{(a-c)^2 + b^2}}{2}}$$
$$l_m = \sqrt{\frac{a + c - \sqrt{(a-c)^2 + b^2}}{2}}$$
$$\theta = \operatorname{atan}\left(\frac{\sqrt{\sqrt{(a-c)^2 + b^2} - (a-c)}}{\sqrt{\sqrt{(a-c)^2 + b^2} - (c-a)}}\right)$$

Note however that the formulas given above are actually for the ellipse that best approximates the binarized hand, which may not necessarily coincide with what we think of perceptually as a "bounding" ellipse. Also, for "roundish" hand shapes, which have no dominating axis, the computed value for θ is highly unreliable.

All the feature measurements listed above are obtained directly from video input and may be corrupted by noise. Thus, we apply Kalman filtering to estimate the true values of the feature vector. The measurement vector in this case consists of $z = (x_0, y_0, \theta, l_M, l_m)^{\mathrm{T}}$, and the state vector x includes the same features as well as their velocities and accelerations. That is, $x = \left(x_0, y_0, \theta, l_M, l_m, \dot{x}_0, \dot{y}_0, \dot{\theta}, \dot{l}_M, \dot{l}_m, \ddot{x}_0, \ddot{y}_0, \ddot{\theta}, \ddot{l}_M, \ddot{l}_m\right)^{\mathrm{T}}$, where "$\cdot$" denotes velocity and "$\cdot\cdot$" denotes acceleration.

Thus, the measurement model H and the dynamic model Φ are

$$H = \begin{bmatrix} I & 0 & 0 \end{bmatrix} \qquad \Phi = \begin{bmatrix} I & \Delta t I & \frac{1}{2}(\Delta t)^2 I \\ 0 & I & \Delta t I \\ 0 & 0 & I \end{bmatrix}$$

where I is a 5x5 identity matrix. For the corresponding measurement noise v and the dynamic model noise w, we use simple Gaussians with zero means and variances

$$R = I \qquad Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I \end{bmatrix}$$

respectively, where I again is a 5x5 identity matrix.

In determining the values for the different hand features, the "smoothed" updated estimate vector $\hat{x}(+)$ is used instead of the actual measurement vector z. A subset of this state vector and the segmented hand obtained from preprocessing are passed to the next stage, which is gesture coding.

3.2 Gesture Coding

Thus far we have used the term "gesture" loosely. Rather than restricting a gesture to mean an action which has definite meaning in human communication, we define gesture in a broader sense. In this thesis, a hand gesture is a "repeated" sequence or trajectory of measurements through a feature space. The feature space is created from examples of different hand movements frequently used while speaking.

Our definition of hand gesture makes several assumptions. First, the feature space must be comprehensive in the sense that it includes all gestures of interest. Second, each gesture within this space must describe a set of trajectories that can be interpreted by a human as being examples of that gesture. Finally, the sets of trajectories for the different gestures must be mutually exclusive so that gestures are distinguishable.

Gesture coding is handled by either the knowledge-free PCA coder or the knowledge-based HMM coder. The PCA coder exploits the redundancy of information found in a video sequence by using the last estimated hand to predict the appearance of the new hand. Principal component analysis is applied to the difference image or *innovation* between the predicted hand and the actual hand. The output of the PCA coder is thus a vector of projection coefficients. In contrast, the HMM system uses the learned (hence the term "knowledge-based") transition matrix and output probability density matrix to identify the current hand gesture. The HMM outputs are a gesture ID label and a score of how confident the system is of its decision.

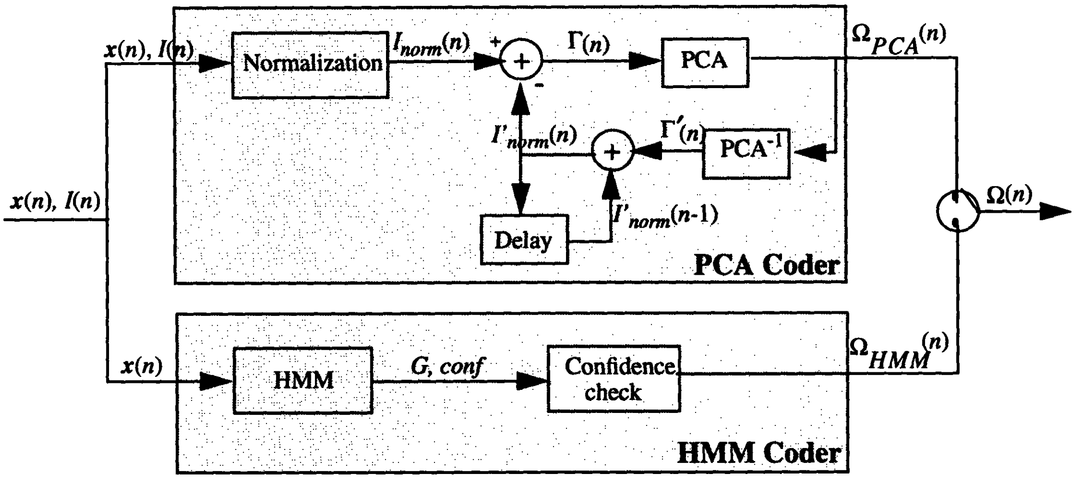


Figure 3.5 Schematic of the dual-system gesture coder. At the switch, output of the HMM system is selected unless the HMM's confidence level is low.

Figure 3.5 shows the schematic of the dual-system framework for gesture coding. As mentioned, the output of the HMM coder is selected unless the HMM's confidence level is lower than a specified threshold. In this case, the switch selects the output of the PCA coder. The chosen output and the feature vector are then encoded and transmitted. Below we describe the PCA and HMM coding systems more thoroughly.

3.2.1 Knowledge-free PCA Coding

As shown in Figure 3.5, the knowledge-free approach uses the feature vector $x(n)$ to normalize the segmented hand with respect to intensity, scale, location, and orientation. Scaling and rotation are accomplished using bilinear interpolation. The resulting normalized hand image $I_{norm}(n)$ is then differenced by a predicted image $I'_{norm}(n)$. Principal component analysis is applied to the innovation $\Gamma(n)$ between the predicted hand and the normalized hand to obtain the vector of projection coefficients Ω_{PCA} . The predicted hand in this case is simply the PCA reconstruction of the innovation plus the last predicted hand image.

Note that for this scheme to work, the system must be initialized at both the transmitter and receiver with the first hand image of the video sequence. Another underlying assumption of this approach is that the innovation takes less amount of bits to code than the actual image. Also, we select principal component analysis for image compression since it can transform an entire image to a small set of coefficients in a reversible process. PCA has been successfully used for face coding, detection, and recognition [15],[9]. Moghaddam and Pentland reported using only 105 bytes to represent a normalized face image along with its original location, scale, and contrast information.

Since PCA is highly sensitive to variations in location, scale, orientation, and lighting contrast, the innovation must be normalized in all these respects. In this case, the hand images from which the innovations were derived have already been normalized. Thus, we only need to normalize the intensity of the innovation. From the set of example gesture sequences, we construct several eigenspaces and select the best ones for the final implementation. The specifics of the eigenspaces and eigenvectors used are given in Chapter 4.

3.2.2 Knowledge-based HMM Coding

The schematic of the knowledge-based coding system is shown in Figure 3.5. At each time step, the system takes as input only the feature vector $x(n)$. However, the HMM component takes into account the history of all the previous inputs in deciding the identity of the current gesture. The output of the HMM component includes a gesture ID code G and a confidence score *conf,* which is the log probability of how certain the system is of its decision. If *conf* is at least equal to a preset threshold, the system output $\Omega_{HMM}=G$. Otherwise, Ω_{HMM}=INVALID, which tells the switch to pass control to the PCA system. For the remainder of this section, we discuss the issues dealing with gesture recognition and prediction using HMMs.

Gesture Recognition: Automatic identification or recognition of natural hand gestures is difficult because human gestures are inherently variable in both spatial and temporal configuration. However, we can safely assume that by observing enough examples of a gesture, we can determine which aspects of the gesture are reliably repeated and which are unreliable. The reliable features can then be used to characterize and identify the gesture.

To achieve this, during training the system must determine a prototype trajectory through the feature space from a group of example trajectories for a given gesture. The system must also determine the local variances along the prototype trajectory. At the end of the training phase, the system should have a unique prototype trajectory for each gesture of interest. Recognition of a gesture is then simply finding the prototype that is most consistent with the trajectory of the gesture. This is the approach taken by Bobick and Wilson to characterize a wave gesture [2].

The above approach, however, does not account for temporal variability. One solution is to use dynamic time warping (DTW). In this thesis, we instead use hidden Markov modeling since HMMs allow the temporal variability to be learned at the same time as the other features. Using HMMs for gesture recognition is valid in a historical sense since the HMM methodology is related to standard statistics techniques such as expectation maximization (EM), Q-learning, and dynamic time warping. HMMs have also been used successfully by the speech community for the past decade.

In this thesis, we shall train the HMMs starting with a small subset of the extracted hand features since a large feature set would require more training as well as recognition time. On the other hand, if the feature set is too small, the system would not be able to distinguish between the different gestures. Hence, our approach is to continually add measurements to the feature vector until the system can reliably differentiate between the gestures. The initial feature vector used consists of the change in the centroid position, the angle of the axis of least inertia, and the half-lengths of the major and minor axes of the bounding ellipse, or $(\dot{x}_0, \dot{y}_0, \theta, l_M, l_m)$.

Similarly, the topology of the HMMs are kept as simple as possible. We model each gesture with a continuous left-to-right HMM and allow no skipping of states. The number of states of the

HMMs is varied from 3 to 7. Each state output probability density is modelled by a single Gaussian instead of a mixture of Gaussians.

The initial model $\lambda_0 = [A_0, B_0, \pi]$ used for training the different HMMs are identical,. The transition matrix A_0 is defined such that the probability of staying in the current state and the probability of going to the next state are 0.5, with all other transition probabilities set to 0. Next, the initial state output matrix B_0 is set such that all states have the same zero-mean, unit-variant Gaussian density. From this initial model, we apply the Baum-Welch algorithm until the parameters for all the HMMs converge.

Gesture Prediction: Most implemented gesture recognition systems identify the gesture after seeing the complete action sequence. In contrast, our system must decide the gesture identity as early as possible. Thus, during the recognition phase, we force the HMM to make a decision at each time step. Then, using the corresponding log probability score, we can determine whether the decision is a valid one. Hence, in training the HMMs, we care about how soon the models can make a reliable prediction, not just the final result. Using a set of example gesture sequences, we can determine the confidence threshold beforehand.

3.3 Gesture Reconstruction

Since the input hands may be coded using either the PCA approach or the HMM approach, we need two different ways to reconstruct the hand. Figure 3.6 shows the schematic of the twofold module for handling gesture reconstruction. Again, the knowledge-free system is called the PCA reconstructor, and the knowledge-based system is called the HMM reconstructor. Note that Ω_{PCA} contains a vector of eigen projection coefficients, whereas Ω_{HMM} contains only a gesture ID code.

Briefly, the knowledge-free system reconstructs the hand in each frame using inverse principal component analysis. The HMM system, however, uses the transmitted gesture ID code to retrieve

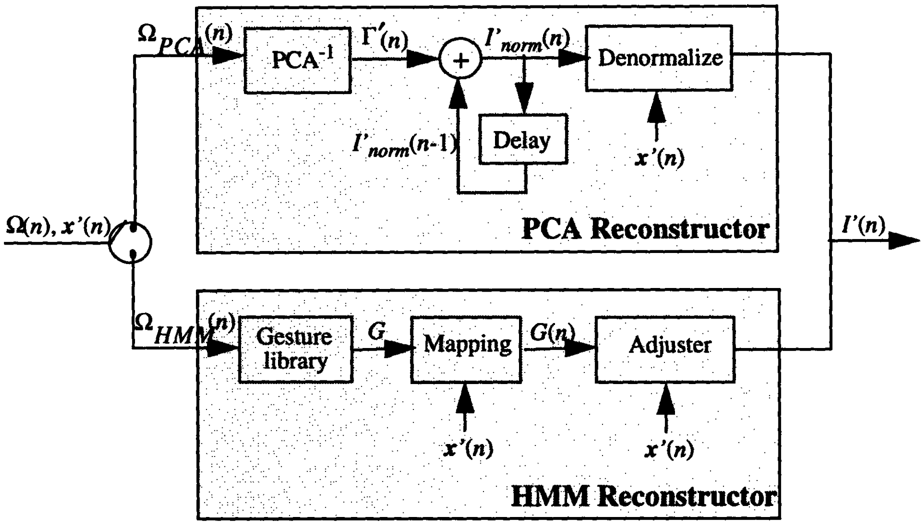


Figure 3.6 Schematic of the twofold gesture reconstructor. Based on the decoded "message", the switch determines which subsystem should handle reconstruction.

a particular gesture prototype sequence from a library. HMM reconstruction is accomplished using the stored prototype gesture. The detailed design of the PCA and HMM reconstructors are described below.

3.3.1 Knowledge-free PCA Reconstruction

The success of any reconstruction approach depends on its ability to simulate the knowledge available at the coding end. In the knowledge-free case, we need to obtain a normalized hand image from the transmitted projection coefficients of the innovation, As shown in Figure 3.6, the PCA system has two layers of reconstruction in both systems. The first layer reconstructs the innovation Γ' from the decoded transmission via inverse principal component analysis. Since PCA reconstruction may not be exact, Γ' is not necessarily the same as the Γ before transmission. The second layer of reconstruction adds the innovation Γ' to a predicted hand to produce the normalized hand image I'_{norm}. At any given time n, the predicted hand is simply the last reconstructed normalized hand image $I'_{norm}(n-1)$. The last step in the reconstruction process is to denormalize I'_{norm}. Using the decoded x', the system converts the normalized hand image back to its original intensity, size, and orientation. Again, scaling and rotation are done using bilinear interpolation.

Note that the reconstruction of future hand estimates is based on previously reconstructed images. Clearly, the error will propagate from frame to frame. One solution is to send the original (normalized) image at regular intervals for the reconstructor to recalibrate its estimate such as in the MPEG scheme. This would limit the duration as well as the extent of the error accumulation. The inherent trade-off here is between quality and efficiency. For the system to reliably regenerate the original motion, it needs to send frequent update information. But for maximum data compression, we like for the system to transmit as little as possible. The trick is to find the optimal balance between quality and efficiency. PCA reconstruction results and analysis are discussed in Chapter 4.

3.3.2 Knowledge-based HMM Reconstruction

The schematic of the HMM reconstructor is given in Figure 3.6. As shown, the inputs to the system are only the feature vector x' and the gesture ID code Ω_{HMM}. From these, we need a way to reconstruct the gesture.

Although HMMs have been demonstrated to work well for gesture recognition, their structure makes them unsuitable for gesture generation. The underlying assumption of hidden Markov modeling is that the time-varying process (e.g. the sequence of feature measurements corresponding to a gesture) to be modelled is usually held in steady state, except for minor fluctuations, for a certain period of time before changing to another steady state. HMMs therefore can not model motion within a state, and smooth motion estimation via hidden Markov modeling is not possible.

The prototype trajectory approach of Bobick and Wilson [2] may provide a solution to the problem of gesture reconstruction. In this scheme, each gesture is characterized by a prototype trajectory through feature space. Suppose the system recognizes the user's current motion as being the start of a certain gesture, and suppose the system also knows how far along the corresponding prototype trajectory the motion is at the moment of recognition, then the system would simply move along the rest of the prototype trajectory to generate future hand estimates.

Two main issues must be resolved for this estimation-from-prototype approach to work, namely timing and hand shape correction. Both of these deal with "personalizing" the reconstructed hand. Embedded in the prototype trajectory is an average gesture execution time, which in general will not match the execution time of a random user's gesture. To ensure that the predicted gesture will be in sync with the observed gesture, we can incorporate velocity information of the observed hand. If we think of the prototype trajectory as a continuous curve in feature space, then the velocity information determines the sampling rate at which we generate the curve. High velocity indicates coarse sampling, and low velocity indicates a finer sampling.

The second issue, hand shape discrepancy, is harder to resolve. However, in most cases of interest, the hand is moving fast enough such that the motion is more important than the details of the hand. Of course, the system can also be made user-dependent. In this case, the system can learn the hand shapes and motion characteristics of the different gestures of a specific user or set of users. Since the system knows more about the users, the discrepancy between the reconstructed hand and the actual hand should be small. In addition, it may be acceptable to not reconstruct the actual motion exactly. Since the stored prototype is characteristic of the user, reconstruction from the prototype with no further correction would still appear normal to the receiver.

Reconstruction from prototype gestures is the approach we undertake for this thesis. Specifically, we implement a user-dependent system for one user. For each gesture of interest, we build a hand-only prototype from training examples of the gesture. All the prototypes are stored in a "gesture library" at the receiver end as shown in Figure 3.6. Using the ID label Ω_{HMM}, we retrieve the corresponding gesture G from the library.

Next, we have to determine where along the prototype gesture we are to retrieve the appropriate hand image for display. To achieve this, we need to find a mapping between the transmitted feature vector $x'(n)$ and the prototype. In this implementation, we compute a sequence of feature vectors x_P for each prototype. Then, by keeping track of how $x'(n)$ changes with time, we can dynamic time warp x' to x_P to find the mapping. The trick is to determine which features to compute for x' and x_P. For the current task, a set of good features must be chosen such that the trajectory of the gesture through the feature space is smooth and follows a general pattern. Note that different sets of features may be used for different gestures.

Finally, after we retrieve the appropriate hand image $G(n)$ from the stored prototype, we need to adjust its intensity and scale (using x') to fit the current situation. The results of HMM reconstruction is one of the topics discussed in the next chapter on system evaluation.

Chapter 4

System Evaluation

The work described here is a feasibility study to determine whether knowledge of gestures can be used to improve coding for the transmission of hand gestures. Hence, although we use the number of bytes needed to be transmitted at each time step to compare between different approaches, we are more interested in discovering when the system works and when it fails. To this end, we evaluate each component of the system first before testing the combined system.

In this chapter, we first describe the data used to train and test the system. Next, the specifics of the PCA system and the results obtained using only the PCA approach are given. Similarly, the specifics of the HMM approach and its results are also provided. Finally, we present the analysis of the combined PCA-HMM framework.

4.1 Gesture Data

The gesture data used in this thesis was obtained using an SGI Indy video capture program with instant playback. The camera was mounted on top to the display, and the user was seated facing the setup. This is intended to simulate a video teleconferencing situation via the computer network. Note that due to storage limitations, we recorded only grayscale 160x120 video sequences. Also, the built-in digitizer and playback program do not have a constant frame rate, resulting in jerkiness and blurriness in some sequences. The average frame rate is about 15Hz.

Since the system implemented is user-dependent, gesture examples from only one user (the author) were recorded. Currently, the system only knows of two gestures: "self" (hand going from resting position up to chest and back, used when referring to oneself) and "side" (hand swinging from resting position to side and back). These gestures were chosen because they are the most natural for the particular user and thus most frequently repeated gestures.

Thirty-five examples of each gesture were recorded. Of these, we set aside 25 examples for training and 10 for evaluating the system. The assignment is mostly arbitrary, although we made sure that both sets have slow as well as fast examples. The average length for the "self" gesture is 29 frames (approximately 2sec), and the average for the "side" gesture is 24.5 frames (1.6sec).

The training data was used in all system design stages, including determining the parameters of the segmentation algorithm, constructing the different eigenspaces, training the HMMs, setting the confidence threshold, and building the gesture prototype sequences. The testing set was used solely for evaluating the system.

4.2 PCA Approach

The success of the PCA approach depends on the selection of an eigenspace. The ideal eigenspace allows an image to be encoded and reconstructed using only a small number of projection coefficients. Below we present a comparison between the different spaces tried, followed by a discussion of the final PCA system design and results.

4.2.1 Eigenspace Selection

In designing the PCA system, our assumption was that the innovation takes less amount of bits to code than the actual hand. To test this assumption, we construct eigenspaces using innovation images as well as actual hand images. From the set of example innovations, we build 3 different eigenspaces: one using 500 frames from the gesture "self" only; another using 500 frames from the gesture “side” only; and a third using 500 frames equally chosen from both gestures. Similarly, for the actual hand images, we build a “self” space, a “side” space, and a combination space using 500 frames each.

Table 4.1 shows the average reconstruction residual obtained from projecting all the training sequences onto the 50 most significant eigenvectors of the six spaces described. The last row “best” of the table shows the average obtained when we select the space (either “self”, “side”, or combination) which yields the smallest reconstruction residual for each image. Surprisingly, the innovation spaces produce worse results than the spaces constructed from actual hands. Figure 4.1 shows typical reconstruction results using 50 projection coefficients. In Figure 4.1, (a) shows the original image, (b) shows the reconstructed image using the best of the innovation spaces, and (c) shows the reconstructed image using the best of the hand spaces.

Spaces	***Innovation***	***Hand***
“self”	1291.09	693.50
“side”	1021.33	528.53
combo	909.49	383.59
best	641.07	250.36

Table 4.1: Average reconstruction residual from different eigenspaces.

One possible explanation for the poor performance of the innovation spaces is that PCA reconstruction tends to “smooth out” or low-pass the original image. This is not as significant for hand images as for innovation images, which contain more high-frequency components. Figure 4.2 shows the low-pass effect of PCA reconstruction on innovation images. Here, the original innovation image is displayed in (a), and the reconstructed innovation image is shown in (b).

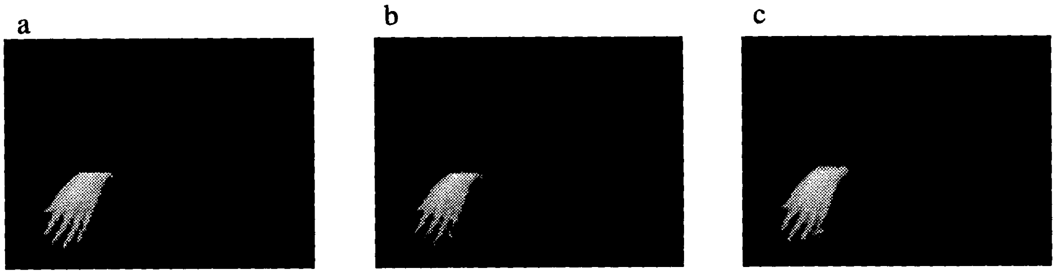


Figure 4.1 Comparison of innovation space and hand space. (a) original segmented image, (b) reconstructed image using the best innovation space, and (c) reconstructed image using the best hand space.

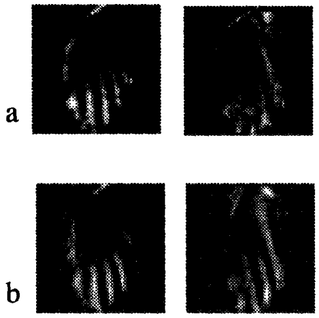


Figure 4.2 Low-pass effect of PCA reconstruction. (a) original innovation and (b) reconstructed innovation with less sharp edges.

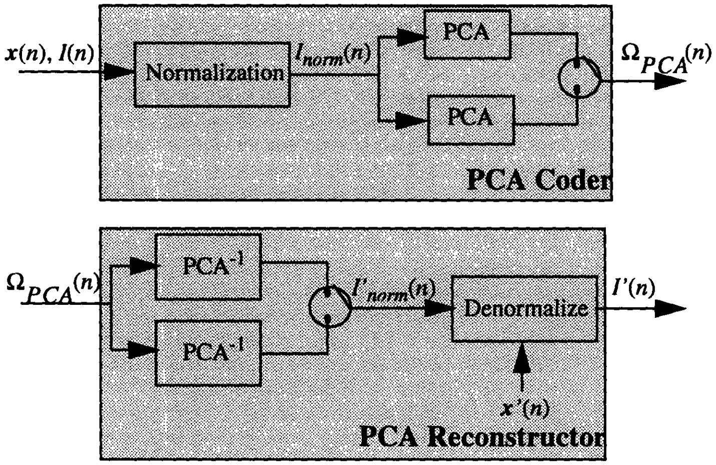


Figure 4.3 Modified schematic of the PCA coding and reconstruction system.

Based on the above analysis, we modify the design of the PCA system to that shown in Figure 4.3. The eigenspaces used in the system are constructed from training hand images rather than innovation images. Thus, we no longer need to keep the previously reconstructed images. However, the system now includes two different spaces. We have found that the "self" space works best for "self" gestures, and similarly the "side" space works best for "side" gestures. This is illustrated in Figure 4.4, where (a) displays the average residual (as a function of the number of eigenvectors) obtained by projecting the 25 training "self" gestures onto the "self", "side", and combination spaces, and (b) shows the average residual obtained by projecting the 25 training "side" gestures onto the same set of spaces. Since the PCA system has no knowledge of gestures, we need to use both the "self" and "side" spaces and select the one which yields the smaller residual. Note that adding an extra space to the system yields better overall results, but it also increases the computational requirement. The number of projection coefficients is still kept at 50 since this accounts for 90% of the total variance in both the "self" and "side" spaces.

4.2.2 Final PCA Results

At each time step, the PCA system needs to transmit 50 projection coefficients, the eigenspace ID

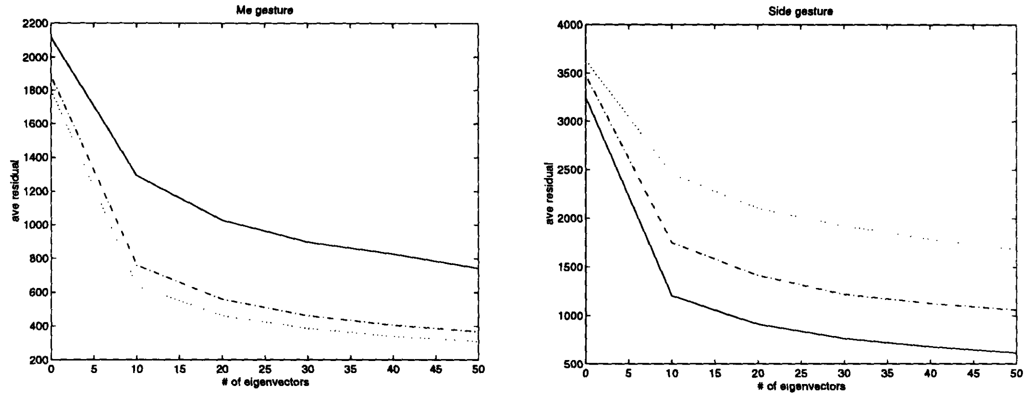


Figure 4.4 Comparison of the "self", "side", and combination spaces. Average residuals as a function of the number of eigenvectors for (a) the gesture "self" and (b) the gesture "side". Dotted curve represents the "self" space, solid curve indicates the "side" space, and dash-dot curve represents the combination space.

code, and the feature vector x', which includes the hand centroid (x_0, y_0), the angle of rotation θ, and the original intensity and size information. Thus, the number of bytes needed to be transmitted per frame is about 212. This is twice the figure reported by Moghaddam and Pentland for coding faces using principal component analysis [9]. However, the human head is for the most part a rigid object, whereas the human hand is highly deformable.

Using the 10 test sequences for each gesture, we found that the average reconstruction residual grows from 250.4 for the training sequences to 461.6 for the testing data. The dramatic increase in average reconstruction residual indicates that we may have overfit the training data. Perhaps fewer eigenvectors should have been used to approximate the spaces.

Of the two gestures, the system performs worse for the gesture "side". This is partly because the "side" gesture is executed much faster than the "self" gesture, resulting in more motion blurs and jerkiness. Also, the hand's appearance in the "side" gesture as seen from the camera deforms much more than in the "self" gesture. One possible improvement is to construct more "specialized" spaces for the gesture "side", with each space capturing a set of similar hand poses.

In general, the PCA reconstructed images are blurry and tend to contain hands with very thin fingers. However, in images where the segmentation algorithm fails and the segmented hand includes extra parts, the PCA reconstruction of the image tends to make the extra parts less noticeable.

4.3 HMM Approach

In this approach, the system tries to recognize the ongoing gesture and to use this knowledge to reconstruct the rest of the gesture from a stored prototype. Below, we first discuss the hidden Markov model selected to handle gesture prediction. Next, we describe how we determine the mapping between the current frame and the stored prototype. Finally, we present the analysis of the HMM system's performance.

4.3.1 Model Selection

Our search for the right HMM starts with a simple feature set and topology as described in Chapter 3. This simple setup, however, yields 100% recognition accuracy when trained and tested on the full-length training examples. We vary the number of states from 3 to 7, and the system still performs consistently well.

However, as mentioned previously, we are more interested in how quickly and reliably the system can predict the identity of the current gesture before its completion. Thus, as a basis for comparison, we define "wait" to be the amount of time (in terms of number of frames) before the system correctly recognizes the gesture. The worst wait is the longest wait over all instances within a given gesture set, and the average wait is the average over all the gestures of the same set.

Table 4.2 shows the worst waits for the "self" and "side" HMMs as we vary the number of states. The values in parentheses indicate the average waits over all training examples. As shown, the 6-state case yields the shortest worst-wait, whereas the 7-state case has the shortest average wait. In general, increasing the number of states seems to favor the "side" gesture.

HMM	*Self*	*Side*
3-State	5 (1.28)	3 (2.56)
4-State	5 (1.28)	3 (2.56)
5-State	3 (1.08)	4 (2.68)
6-State	2 (1.04)	4 (2.68)
7-State	5 (1.56)	3 (1.12)

Table 4.2: Waits" for the different HMMs. First value is the wait in the worst case; value in parentheses is the average wait over all training examples.

The average log probability (or confidence score) per frame for the different cases are shown in Table 4.3. As the number of states increases, the confidence score also increases. However, we did not try to raise the number of states beyond 7 since we did not want to saturate the model. Some gesture examples are only 19 frames in duration, and ideally we would like each gesture instance to spend on average two to three frames in each state.

Gesture	*Self*	*Side*
3-State	-7.138	-11.123
4-State	-6.844	-10.227
5-State	-6.393	-9.987
6-State	-6.182	-9.708
7-State	-5.999	-9.018

Table 4.3: Average log probability per frame for the different HMMS.

Based on the above analysis, we select 7 to be the number of states for the HMMs. The final HMMs for the "self" and "side" gestures are shown in Figure 4.5. Note that at each state, we have one Gaussian density for each of the 5 output features.

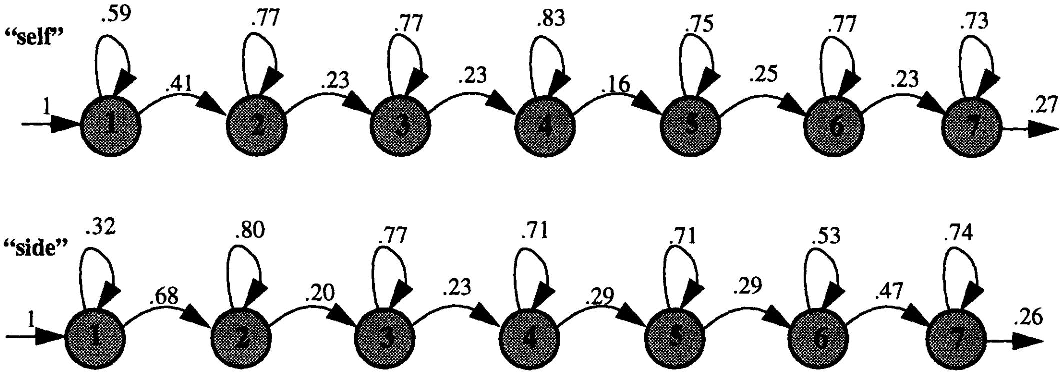


Figure 4.5 Transition diagrams for the "self" and "side" HMMs.

For checking the validity of the HMM's prediction, we use an average log probability threshold and a consistency test. Hence, for a "self" prediction, if the average log probability is at least −5.0 or the current prediction is consistent with the previous prediction, then the current prediction is valid. Similarly, for a "side" prediction, if the average log probability is at least −0.3 or the current prediction is consistent with the four previous predictions, then the system accepts the current prediction. Note that in establishing these conditions, we did not account for the null case. We assume that at any given time, the ongoing action is either gesture "self" or gesture "side". (The system will be extended in the very near future to account for unseen gestures as well.)

Applying these validity tests to the training set results in the rejection of 45 out of 1316 predictions. Of these rejections, 31 (or 2.36%) are misses. This reflects our conservative view which favors rejections over misclassifications.

4.3.2 Prototype Gesture Mapping

Our approach to finding the mapping between a new gesture and the stored prototype is to dynamic time warp their corresponding feature vectors. The key issue is which feature(s) should be included in this mapping. To start, we consider all the features in the state vector x as possible candidates and weed out the ones that do not satisfy our mapping condition. The mapping condition requires that all example trajectories of the same gesture through the feature space must be

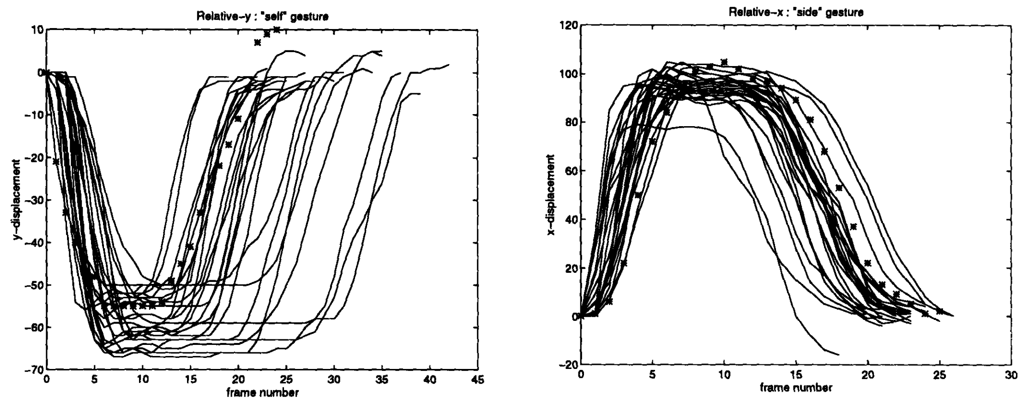


Figure 4.6 Mapping trajectories. (a) shows the relative-y trajectories of instances of gesture “self” and (b) shows the relative-x trajectories of instances of gesture “side”. The asterisk curve in both represents the respective prototype trajectory.

smooth and must follow a general pattern.

Following the above paradigm, we find the relative *y* position of the hand to be the ideal mapping feature for the “self” gesture, and the relative *x* position of the hand to be ideal for the “side” gesture. Figure 4.6a shows the relative-*y* trajectories of the all the training examples for the gesture “self”. The blue curve represents the trajectory of the prototype “self”. Similarly, Figure 4.6b shows the relative-*x* trajectories for the “side” examples, again with the blue curve depicting the prototype curve.

For most of the training examples, the mapping scheme discussed above performs nicely. The system is able to “stretch” and “shrink” the prototype sequence to fit the current gesture. Most of the reconstructed sequences appear natural to the human observer. Only 2 out of the 25 training examples for the gesture “side” appear unnatural. In both cases, the problem occurs on the “coming-back” part of the gesture where the hand’s appearance changes from palm facing the camera to backside forward. The system mistakenly maps the backside of the hand too early, thus making the reconstructed gesture appear artificial.

4.3.3 Final HMM Results

In evaluating the HMM approach, we analyze each component of the system separately to deter-

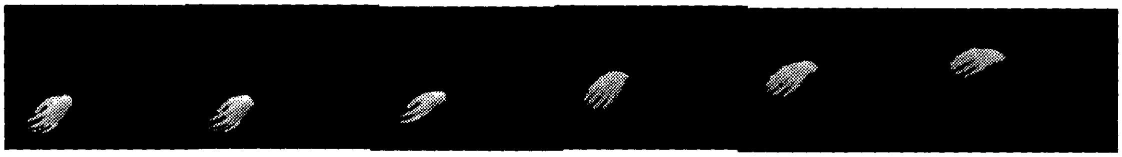


Figure 4.7 A "self" sequence reconstructed using prototype gestures. The first 3 frames were mistakenly taken from the "side" prototype.

mine which parts are satisfactory and which need improvement. Hence, we first discuss the performance of the HMMs, followed by an analysis of the preset confidence check.

On the test set, the 7-state HMMs again achieve 100% recognition accuracy. The worst waits for the "self" and "side" HMMs are 4 frames and 3 frames, respectively. The corresponding average waits are 2.2 and 1.2 frames. The average log probabilities for the "self" and "side" gestures are slightly lower than the for the training set at -6.591 and -9.292 respectively.

Imposing the validity conditions, the system rejects 8 correct predictions (1.41% miss rate) and accepted 7 wrong predictions (1.24% false alarm rate) out of 566. The total number of rejections is 14, thus 14 hand images must be handled by the PCA system. In cases where a wrong prediction passes the confidence test, the reconstruction may still look relatively natural since some parts of the "side" gesture do appear very much like parts of the "me" gesture.

Figure 4.7 shows an example where the wrong predictions pass through. The actual gesture is "self"; however, the first 3 frames are from the prototype of the gesture "side". Note that only the third frame appears out of place. If the sequence is played instead of analyzed frame by frame, the mistake is barely noticeable.

In general, the reconstruction of the 20 test sequences appears natural. The mapping problem present in the training phase was not encountered. The HMM-reconstructed hands not only seem clearer than the PCA-reconstructed hand, but also clearer than the actual hand in most cases. Also, since the reconstruction is from stored prototypes, even when the segmentation includes extra part, the reconstructed images are still hands only, as shown in Figure 4.8.

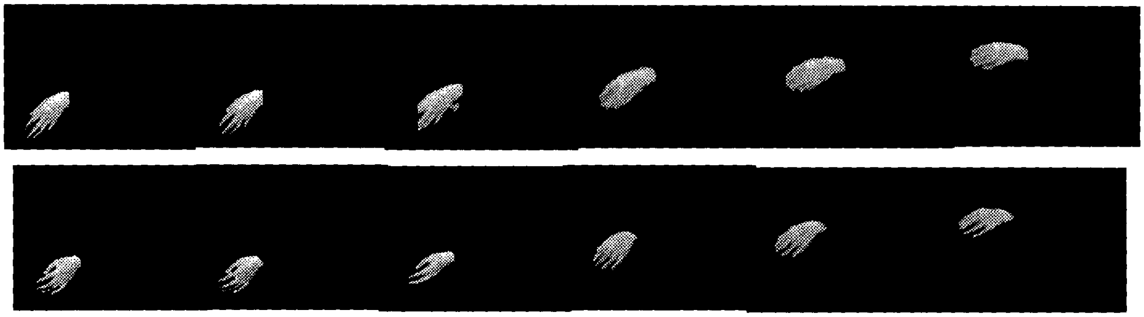


Figure 4.8 Example of the higher quality of a HMM-reconstructed sequence. The top line is the original segmented sequence, and the second line is the reconstructed sequence (which is the same as the one shown in Figure 4.7)

For each HMM-reconstructed frame, we need to transmit the gesture ID code, the centroid location, and the original intensity and size information, which takes approximately 8 bytes.

4.4 Combined System

As mentioned above, when tested on unseen sequences of the two gestures, the HMM system rejects only 14 out of 566 testing frames. Hence, the PCA system is in use only 2.5% of the time. Note that the results reported are valid only under the assumption that there is no null case (i.e. the ongoing action is always either the gesture "self" or the gesture "side"). In general, the PCA system handles the first one or two frames before the HMM system takes over. Since the duration of the PCA reconstruction is relatively short, no visible jump or discontinuity is observed. Figure 4.9 shows an example of a reconstructed sequence using both methods. The PCA system reconstructs the first 3 frames, and the HMM system reconstructs the last 3 frames.

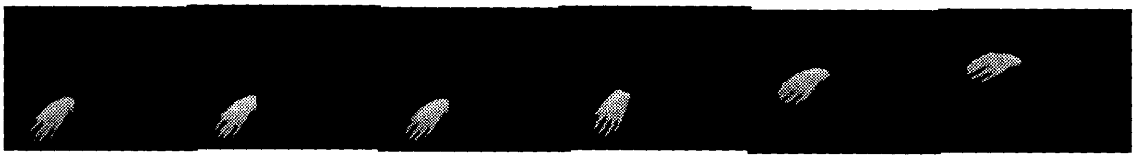


Figure 4.9 Sequence reconstructed using the combined PCA-HMM system. The first 3 frames are handled by the PCA system and the last 3 are reconstructed using the HMM approach.

Although the results reported here are encouraging, it is important to keep in mind the constraints and limitations of the system. Part of the success of the system is due to the particular selection of the gestures. Although unintended, the gestures chosen in this implementation are easily distinguishable. In the next chapter, we summarize the work done and discuss the conditions under which our coding approach can be successfully applied and extended.

Chapter 5

Conclusion

Although the task performed by our coding system is only modestly complicated, the results obtained are encouraging enough to justify future research on knowledge-based coding systems for the transmission of hand gestures. In this chapter, we summarize the important results obtained and provide suggestions for future work.

5.1 Summary

In order to determine the feasibility of using gesture knowledge to improve hand coding for transmission purposes, we have designed and implemented a complete coding system. The system converts an input video sequence into a sequence of variable-length feature vectors. From the feature vectors, the system reconstructs a corresponding hand-only video sequence.

The two main components of the system are the PCA coding module and the HMM coding module. The PCA system uses principal component analysis to compress and reconstruct the sequence of hand images, whereas the HMM system uses hidden Markov modeling to guess the identity of the ongoing hand gesture. Actual hand reconstruction in the HMM system is accomplished by dynamic time warping the stored prototype gestures to fit the current situation. In our implementation, the PCA system has to transmit 50 projection coefficients whereas the HMM system only needs to transmit a gesture ID code (along with a few update feature parameters for both systems).

Due to its greater compression power, we intend for the HMM system to handle the bulk of the transmission, with the PCA system "helping out" only when the HMM component is not reasonably confident about the identity of the ongoing action. In the ideal case, the HMM system would have knowledge of most of the hand gestures of a particular user or set of users. Transmission would then be handled almost completely by the HMM system. In reality, it is most likely that the PCA system will do most of the work since the number of possible hand gestures is too great for the system to learn.

One feasible plan is to make the system user-dependent and to train it only on the most frequently used hand gestures of a particular user. This is the approach we took to evaluate our system design. We trained the system on two frequently used gestures of one user. When tested on different examples of the two gestures, the HMM system handles the coding and reconstruction of the hand sequences 97.5% of the time. In general, the PCA system processes only the first one or two frames of each gesture instance.

The hand-only reconstruction of the test gestures appears natural to the human eye. There is no observable "glitches" even in gesture examples where both the PCA and HMM systems were used to reconstruct different parts of the sequence. In addition, since the stored prototypes were constructed from "good" gesture examples in the training set, the quality of the HMM-reconstructed hand images is much higher than that of the PCA-reconstructed images. In some cases, the quality of the HMM-reconstructed hand is even higher than that of the actual segmented hand, which may suffer from motion blur or segmentation errors.

Even when the HMM system makes the wrong choice (which occurred 1.24% of the time), the reconstructed sequence still appears natural. This is because the hand's appearance in certain parts of the two gestures is similar. Hence, a more interesting and meaningful evaluation would be to test the system using video sequences containing the two gestures that the system was trained on as well as other yet-unseen gestures. It may turn out that the system can reliably reconstruct parts of the unlearned gestures from the stored prototypes. However, this evaluation is beyond the time frame of the thesis.

5.2 Future Work

One immediate extension is to evaluate the system using sequences of known as well as unknown gestures, and to adjust the system parameters accordingly. This may require that we make the confidence test more robust, possibly by setting the rejection threshold higher.

Since this work is meant to be a feasibility study, we trained the system on only two different gestures. It would be insightful to see how the system performance changes as we add more gestures. The gestures "self" and "side" used are easily distinguishable from each other. With a less disparate set of gestures, the system probably would not be able to make a prediction as quickly. In this case, we would need to improve the efficiency of the PCA system. Particularly, we can try using different "specialized" spaces.

The system was also designed to be user-dependent, with the stored prototypes being gesture examples performed by a particular user beforehand. It would be interesting to extend the system to many different users. In this case, the design of the HMM coder would remain mostly the same. However, instead of storing prototypes, the system can learn them from the current user. For instance, the first time the system sees and recognizes a gesture from the user, it transmits a code to instruct the other side to store the sequence. The next time the same gesture is executed, the system would simply tell the reconstructor to play back (after dynamic time warping if necessary) what was just previously received.

Since the HMM system and especially the PCA system depends on good segmentation, more time could be spent on improving the segmentation algorithm. One obvious extension is to use color images and threshold for flesh color to find the hand. We can also readjust the Kalman filter's parameters, particularly the noise models R and Q.

Finally, the grand goal is, of course, to make the whole system perform in real time.

Bibliography

[1] Ballard, D.H. and C.M. Brown. *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

[2] Bobick, A.F. and A.D. Wilson. "Using configuration states for the representation and recognition of gesture." *Proc. Fifth International Conf on Computer Vision.* IEEE Press, 1995.

[3] Cui Y. and J. Weng. "Learning-based hand sign recognition." *Proc. of the International Workshop on Automatic Face and Gesture Recognition.* Zurich, 1995.

[4] Darrell, T.J. and A.P. Pentland. "Space-time gestures." *Proc. IEEE Conf. on Computer Vision and Pattern Recognition.* 1993.

[5] Gelb, A. *Applied Optimal Estimation.* Cambridge, MA: The MIT Press, 1974.

[6] Horn, B.K.P. *Robot Vision.* New York: McGraw-Hill Book Company, 1986.

[7] Huang, X.D., Y. Ariki, and M.A. Jack. *Hidden Markov Models for Speech Recognition.* Edinburgh: Edinburgh University Press, 1990.

[8] Makhoul, J., T. Starner, R. Schwartz, and G. Chou. "On-line cursive handwriting recognition using hidden Markov models an statistical grammars." *IEEE Conf. on Acoustics, Speech, and Signal Processing.* Adelaide, Australia, April 1994.

[9] Moghaddam B. and A. Pentland. "An automatic system for model-based coding of faces." *IEEE Data Compression Conference.* Snowbird, Utah, March 1995.

[10] Rabiner, L.R. "A tutorial on hidden Markov models and selected applications in speech recognition." *Proc. IEEE*, pp. 257-85, February 1989.

[11] Rabiner L.R. and B.H. Juang. "An introduction to hidden Markov models." *IEEE ASSP Magazine*, pp. 4-16, January 1986.

[12] Rangarajan, A., G.L. Cash, D.L. Duttweiler, H.M. Hang, B.G. Haskell, and A. Puri. "Image and video coding standards." *AT&T Technical Journal.* January-February, 1993.

[13] Starner T.E. and A. Pentland. "Visual recognition of American Sign Language using hidden Markov models." *Proc. of the International Workshop on Automatic Face and Gesture Recognition.* Zurich, 1995.

[14] Tanguay, D.O. "Hidden Markov models for gesture recognition." Master thesis. MIT, 1995.

[15] Turk, M. and A. Pentland. "Eigenfaces for recognition." *Journal of Cognitive Neuroscience.* Vol. 3 No. 1, pp. 71-86.

[16] Wilson, A.D. and A.F. Bobick. "Learning visual behavior for gesture analysis." *Proc. IEEE Symposium on Computer Vision.* Coral Gables, Florida, November 1995.

[17] Wren, C. and A. Azarbayejani, T. Darrel, and A. Pentland. "Pfinder: Real-Time Tracking of the Human Body." MIT Media Laboratory Perceptual Computing Section TR#353.

[18] Yamato, J., J. Ohya, and K. Ishii. "Recognizing human action in time-sequential images using hidden Markov model." *Proc. IEEE Conf on Computer Vision and Pattern Recognition.* IEEE Press, 1992.

[19] Young, S.J. *HTK: Hidden Markov Model Toolkit V1.5* Washington, D.C.: Cambridge University Engineering Department Speech Group and Entropic Research Laboratories, Inc., 1993.

Appendix A

Implementation

Implementation of the complete coding system was done by the author in the C++ programming language, except for two modules. First, computation of the different eigenspaces was accomplished using Baback Moghaddam's principal component analysis program. Second, we used a commercially available software package called *HMM Toolkit* (*HTK*) [19] developed by the Entropic Research Laboratory to handle hidden Markov modeling.