Hidden Markov Models

for Gesture Recognition

by

Donald O. Tanguay, Jr.

S.B. Computer Science
Massachusetts Institute of Technology, Cambridge, MA
June 1993

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
August 1995

Author _____
Department of Electrical Engineering and Computer Science
August 25, 1995

Certified by _____
Aaron F. Bobick
Assistant Professor of Computational Vision
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by _____
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Hidden Markov Models for Gesture Recognition

by

Donald O. Tanguay, Jr.

Submitted to the
Department of Electrical Engineering and Computer Science

August 25, 1995

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

Abstract

Understanding human motions can be posed as a pattern recognition problem. Humans express time-varying motion patterns (gestures), such as a wave, in order to convey a message to a recipient. If a computer can detect and distinguish these human motion patterns, the desired message can be reconstructed, and the computer can respond appropriately. This thesis describes an approach to recognize domain-dependent gestures using the statistical pattern recognition tool, the Hidden Markov Model (HMM). Through several experiments with two-dimensional mouse gestures, this thesis analyzes the behavior of HMM training and reports some important insights towards better HMM performance.

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

Acknowledgments

First and foremost, I want to thank Aaron Bobick for giving me the opportunity to work on this fascinating and fun thesis. I have thoroughly enjoyed even the most painful moments of learning the theory of Hidden Markov Models and the secrets of the covariance matrix. For, though my confused state was nonemitting, I knew there was a nonzero transition probability to the success state. This thesis and his course have opened my eyes to the world of computer vision.

I am thankful to Gilbert Leung for helpful discussions on mathemagics and interesting late-night conversations. I especially thank him for choosing to go on vacation just as I was searching for a comfortable couch to call home. I will be forever thankful to him for giving me a place to sleep.

I must thank NSF for supporting me while making the transition from computer architecture to computer vision.

Finally, I thank Katerina Nguyen for proofreading this document, staying up until 5 AM to run experiments, and being patient with my excitement. Most importantly, I thank her for remaining strong.

Contents

List of Figures

Chapter 1

Introduction

This thesis is a study of how Hidden Markov Models can be applied to recognize gestures. In order to understand the behavior of Hidden Markov Models (HMMs), the work consists of the design, implementation, and experimentation of a system for creating gestures, training HMMs, and recognizing gestures with the HMMs. Before describing those aspects, this section motivates the thesis topic and outlines its presentation.

1.1 Motivation

One person raises his flattened, vertical palm toward another, as if to assure the other that his hands conceal nothing harmful. The other person replies similarly, and they smile. This wave gesture has been learned from childhood to mean extension of friendship.

Understanding human motions can be posed as a pattern recognition problem. In order to convey visual messages to a receiver, a human expresses motion patterns. Loosely called gestures, these patterns are variable but distinct and have an associated meaning. The wave gesture is variable because even the same person's hand position may be several inches away from the position in a previous wave. It is distinct because it can be readily distinguished from a different gesture, such as a beckoning or a shrug. Finally, it has the agreed meaning of "hello."

Research on gesture recognition has many motivations, all of which are related to improving the interface between humans and computers. If a computer can detect and recognize a set of gestures, it can infer the sender's message and respond appropriately. For example, a conductor can control a "virtual orchestra" by gesturing commands to a video camera. The system responds by appropriately varying the volume and tempo of the prerecorded music being played. As another example, a system can annotate video clips of athletic events with meaningful descriptions. When requested for an example of a triple salchow, another system responds by quickly finding the figure skating jump among an annotated database of video sequences. As a final example, a karate instruction system can visually evaluate the performance of a student's kick.

This thesis attempts to help bridge the visual communication gap between computers and humans by designing and implementing a gesture recognition system based on the semicontinuous Hidden Markov Model. The HMM framework models the temporal behavior of gesture, and the use of a global codebook allows the discovery of shared atomic pieces among different gestures, leading to an "understanding" of gesture by identifying their similarities and differences.

1.2 Overview

Before describing the work of this thesis, Chapter 2 presents necessary background material. First, Section 2.1 presents several topics in pattern recognition in order to provide context for the gesture recognition problem. This exposition includes discussion of the general pattern recognition approach, the clustering process, and the Hidden Markov Model. General pattern recognition provides terminology, clustering associates the observations, and Hidden Markov Models provide the recognition framework. Second, Section 2.2 describes the gesture recognition problem as posed in this thesis. An understanding is established by first defining gestures and gesture recognition and then by reviewing previous work on the subject.

Chapter 3 presents the core of the thesis, Hidden Markov Models for gesture recognition. First, Section 3.1 describes the approach of this thesis for recognizing gestures. Next, Section 3.2 details how this work relates to the previous work on the subject, including a description of influential papers. Section 3.3 describes the experiments performed, presents the empirical results of each, and interprets the results by identifying and rationalizing the behaviors involved.

Chapter 4 concludes the thesis presentation. First, Section 4.1 summarizes the major results and insights of the study. Then, Section 4.2 describes future extensions to the system, including future experiments for further HMM study and modifications for improving system performance and accuracy.

Chapter 2

Background

Before presenting the core of this thesis, it is necessary to establish both terminology and an understanding of the problems encountered in HMM gesture recognition. This background material is divided into two major areas: pattern recognition and gesture recognition. First, the section on pattern recognition gives the HMM gesture recognition problem a solid context by presenting the pattern recognition approach, the clustering of data, and the Hidden Markov Model. Next, the description of gesture recognition completes the background material and reviews some related gesture recognition research.

2.1 Pattern Recognition

Pattern recognition forms the mathematical basis of gesture recognition in this thesis. First, a definition of pattern recognition establishes the necessary terminology. Next, clustering, a process for grouping similar objects together, is described. Finally, the foundation of this thesis, the Hidden Markov Model is presented.

2.1.1 Definition

Pattern recognition is a mathematically rigorous field with the purpose of classifying objects into one of a number of classes. These objects of interest are generically termed *patterns* and include

printed characters, speech waveforms, textures, "states" of a system, and anything else one wishes to classify. The pattern recognition process is generally implemented in a manner that allows automatic recognition without human intervention. For example, a system may tell the credit card company which transactions are likely the result of unauthorized credit card use.

Construction of a pattern recognition system involves learning from a set of example patterns. This learning process has two forms. If the classes of the example patterns are already known, the learning process is termed *supervised pattern recognition*. In such a case, the correct classification of an individual pattern is used to evaluate the performance of the system. This feedback allows the system to iteratively improve itself. On the other hand, if the classes are not known *a priori*, then the *unsupervised pattern recognition* system must not only produce a classification procedure but also define the classes themselves. Though this type of pattern recognition is much more difficult, useful algorithms have been developed that allow successful systems to be constructed.

The actual pattern recognition process is performed in two phases, the first of which is *feature extraction*, where the observation $\hat{x}$ of a pattern is transformed into a vector $\hat{y}$, whose components are called *features*. $\hat{y}$ is generally much more tractable for the system than $\hat{x}$, but should contain most of the information necessary for classification of the patterns. The procedures for feature extraction may be based on intuition or physical considerations of the problem, or they may be purely mathematical techniques for simply reducing the dimensionality of the observations.

The second phase of pattern recognition is *classification* of the feature vectors. A classifier partitions the feature space of $\hat{y}$ into disjoint regions, each corresponding to a pattern class. If the feature vector of a specific observation lies in the region R_k the observation is assigned to class C_k. Thus, the partition specifies the class membership of the observations. Constructing the classifier in a supervised pattern recognition system is relatively simple because the class membership of a set of example patterns are known. As mentioned above, this knowledge is used to both train

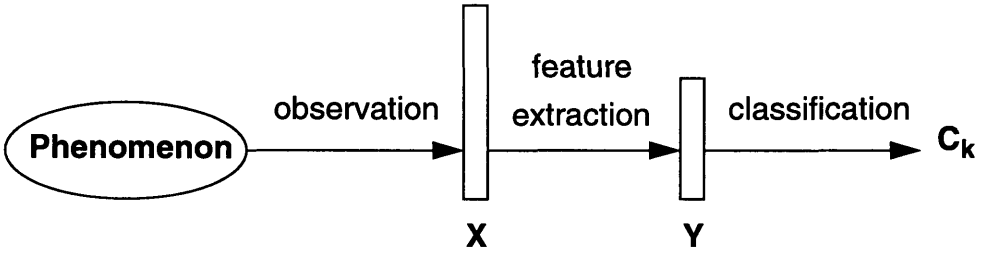

Figure 2.1 The pattern recognition process.

and test the classifier. If class membership is unknown, however, construction is much more difficult and involves *clustering*, which is described in the next section.

Figure 2.1 illustrates the pattern recognition process. First, an observation vector $\hat{x}$ is recorded while observing a phenomenon. Feature extraction from $\hat{x}$ produces a smaller vector $\hat{y}$ which encapsulates the salient features of the original observation. Finally, classification determines the best class label associated with $\hat{y}$. For example, $\hat{x}$ can be a video sequence, $\hat{y}$ can be a sequence of hand parameters or other measured feature, and C_k can be the class of waving gestures. In this way, a system can identify the phenomenon's class. In other words, the system *recognizes* the phenomenon.

In the context of pattern recognition, this thesis is supervised pattern recognition of gestures using an HMM-based classifier. First, example gestures (a sequence of observations) are transformed into a sequence of feature vectors. Each sequence and its known class is used to train a Hidden Markov Model for classification. The result is a system that classifies observation sequences into gesture classes.

Because the observation is transformed into a *sequence* of feature vectors rather than a single feature vector, the work presented here is actually a two-level pattern recognition system. While the above describes the temporal pattern recognition (performed on a sequence), the Hidden Markov Model component itself contains a subsystem for static, unsupervised pattern recognition

(performed on individual observations). Training of the Hidden Markov Model includes clustering, which is described in the next section.

2.1.2 Clustering

Clustering is the process of constructing a classifier for unsupervised pattern recognition. Here, the problem is not only to classify the given data, but also, at the same time, to define the classes. In the general sense, *clusters* are defined as groups of similar points according to some measure of similarity. Usually similarity is defined as proximity of the points as measured by a distance function, such as the Euclidean distance, of feature vectors in the feature space. However, measures of other properties, such as vector direction, can also be used. The method of finding the clusters may have a heuristic basis or may be dependent on minimization of a mathematical clustering criterion.

In the field of digital signal processing, *vector quantization* is clustering using the Euclidean distance measure; however, many new terms are used. The clusters of a classifier are now called the quantization levels of a VQ *code book*. Furthermore, the distance of each sample to the mean of its enclosing cluster is no longer a measure of similarity but rather a measure of *distortion*. The goal of vector quantization is to find the set of quantization levels that minimizes the average distortion over all samples. However, finding the codebook with the minimal average distortion is intractable. Nevertheless, given the number of clusters K, convergence to a local minimum can be achieved through the simple *K-means* algorithm:

1. Randomly assign samples to clusters.
2. Compute the sample mean of each cluster.
3. Reassign each sample to the cluster with the nearest mean.
4. If classification of all samples is unchanged, stop.
   Else, go to Step 2.

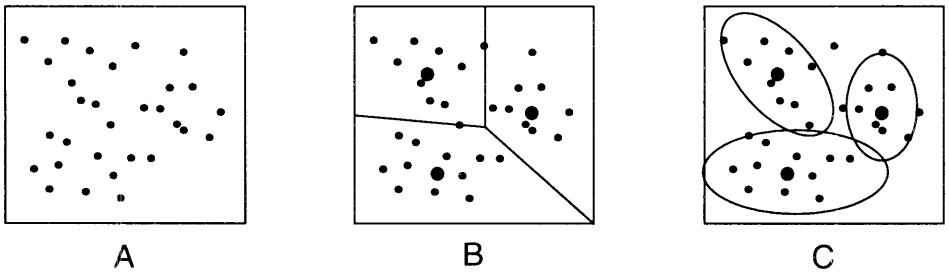

Figure 2.2 Clustering of data. (a) collection of data, (b) vector quantization showing the clusters and associated mean, and (c) estimation of Gaussian distributions.

In this thesis, the clustering method is as follows. First, all the data are collected into a single set of observation vectors. Next, vector quantization groups the data points into clusters. Finally, assuming a Gaussian distribution in each cluster, the estimated mean and covariance of each cluster are computed. The individual classes are then multivariate Gaussians with the estimated parameters. Because these distributions will overlap, classifying is no longer a one-to-one mapping of pattern to class. Rather, a pattern is mapped to a set of classes, each with a probability that it produced the sample. Figure 2.2 illustrates the clustering process. Section B.2 describes the implementation of the code book in more detail.

2.1.3 Hidden Markov Models

Before describing the Hidden Markov Model, it is necessary to describe its foundation, the Markov process. In any pattern there is usually sufficient structure to influence the probability of the next event. For example, in the English language, the probability of detecting the letter u depends very much on whether the letter q was last detected, since u almost always follows q. A stochastic process is called a j^{th}-order Markov process if the conditional probability density of the current event, given all past and present events, depends only on the j most recent events.

A *Hidden Markov Model* (denoted λ) is a doubly stochastic process. The first stochastic layer is the underlying first-order Markov process, represented by the state transition diagram of Figure 2.3a. Each state is a possible observation of the Markov process, and a transition probability from

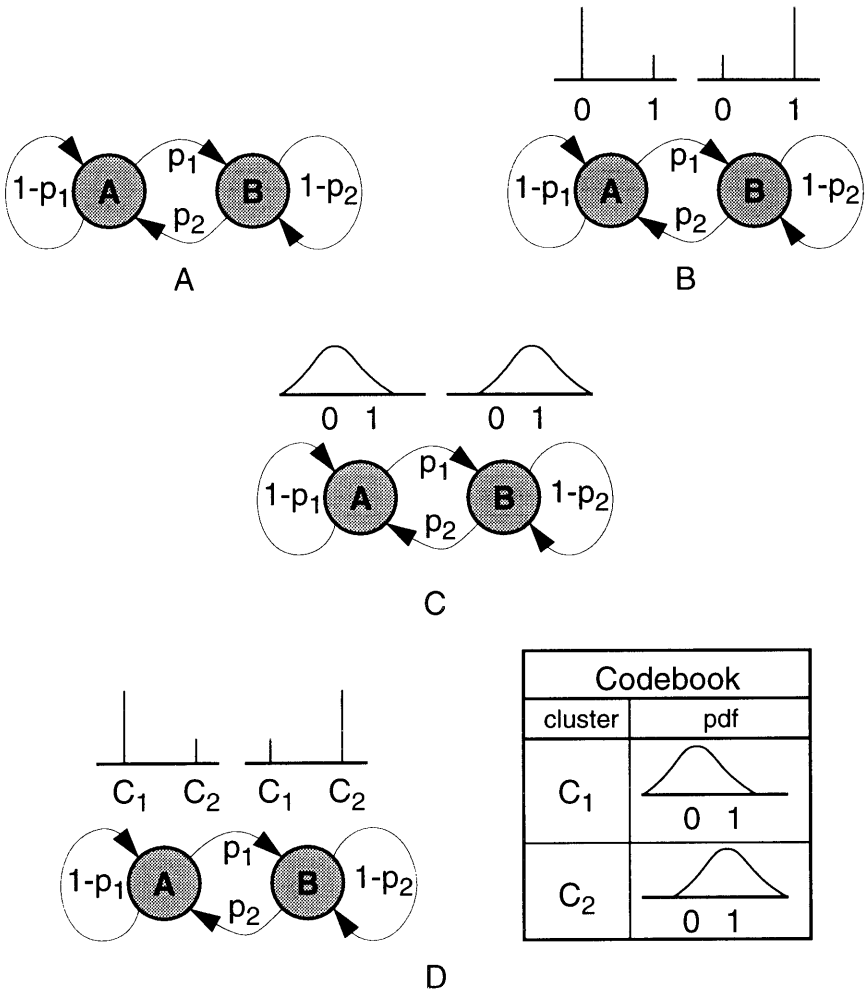

Figure 2.3 Four types of Markov Models. (a) First-order Markov process, (b) Discrete Hidden Markov Model (HMM), (c) Continuous HMM, and (d) Semi-continuous HMM.

state A to state B is $P(s_{t+1} = B | s_t = A)$, the probability of going to state B at time $t+1$ given that the state at time t is A. The second stochastic layer of the HMM is the set of output probabilities for each state. For example, the output probabilities of state A specifies the likelihood of seeing certain observations, given the HMM is actually in state A. This second layer of probabilities creates a veil so that, given a sequence of observations, the actual sequence of states is ambiguous; it is "hidden" from the observer.

Algorithms exist for both training and testing the HMMs. The goal of HMM training is to lift the veil so that, with good probability, the actual sequence of states S can be determined from the

sequence of observations X. However, enough training data must be provided so that a good internal statistical model can be built. Proven to converge, the Baum-Welch reestimation procedure can find locally optimal HMM parameters for a given set of training data. Reasonable initial estimates can help the procedure find the globally optimal solution. For testing and recognition, the Viterbi algorithm determines the state sequence S with the highest probability, given a particular observation sequence $\hat{X}$ (i.e., it maximizes $P(S|\hat{X}, \lambda)$). Appendix A explains these procedures in more detail.

In the literature, there are three basic types of HMMs, differentiated by their method of modeling output probabilities. The observations of the *discrete HMM* are discrete symbols of a finite alphabet that typically correspond to quantization levels (classes) of a vector quantization codebook. Each state has a discrete probability mass function (PMF) for describing the probability that the state would produce a certain symbol. Figure 2.3b shows an HMM with two output symbols; state A is more likely to produce a "0" symbol than a "1", and B is more likely to produce a "1" than a "0". The states of the *continuous HMM* each have a mixture of probability density functions (pdf's) to represent the probability of observing certain multidimensional, continuous data. Mixtures of Gaussian (normal) pdf's are typically used to accurately model the state's membership in the space of observation vectors. Figure 2.3c shows an HMM where each state has a single 1-D pdf; state A is still more likely to produce a "0" than a "1", but quantization no longer distorts the continuous 1-D observations.

The *semicontinuous HMM* is a hybrid of the discrete and continuous HMMs. Like the discrete HMM, the observation vectors are quantized into one of a finite set of classes, reducing the number of free parameters. However, like the continuous HMM, the observation classes are modeled by a multivariate Gaussian pdf, removing the distortion due to quantization and effectively modeling the variance of an observation class. This formulation is similar to a continuous HMM with parameter tying in which states are forced to share the same pdf's. Initially formed by clustering

the example data, the classes are reestimated along with the HMM parameters to form an integrated model. Figure 2.3d shows a two-state HMM with a two-cluster code book; while the observations are still modeled by 1-D pdf's, these pdf's are shared by all states[1]. The work presented here uses the semicontinuous Hidden Markov Model.

2.2 Gesture Recognition

The goal of this thesis is recognition of gesture. This section provides the final background material needed before presenting the procedure, experiments, and results of this thesis. First, gesture and gesture recognition are defined. Then, previous works related to gesture recognition are summarized.

2.2.1 Definition

Avoiding the semantic issues of gesture, this thesis defines gesture in a purely scientific manner. Rather than defining a gesture based on its meaning to a receiver, this work bases a gesture only on the behavior of sequences of scientific measurements obtained from gesture examples. The definition of gesture in the system then becomes a probabilistic definition relying on higher knowledge outside of the system for defining gesture[2]. Explicitly, within the system gestures are defined as sets of trajectories through a measurable gesture space.

The above definition relies on several severe assumptions. First, the feature space of the measurements represents the entire space of gestures. Thus, the expressiveness of gesture space depends highly on the features being used; if the features are inappropriate to the problem at hand or if too few features are used, the system will not be able to distinguish different gestures. Second, gestures are modal. Because an instance of a gesture is a trajectory through feature space, modality means the set of trajectories for each gesture is mutually exclusive. This assumption also

1. In a multiple-HMM recognition system, the states of all HMMs share a single code book.
2. This higher knowledge is usually a human.

depends on the features chosen. Third, gestures can be statistically modeled as a sequence of transitions between perceptual states (states defined by feature measurements). The perceptual states are clusters of observation vectors in the feature space. This assumption is due to the use of HMMs, and is more of a problem for generating examples of gestures than for recognizing examples. For example, when trying to determine the type of gesture an HMM represents, a small transition probability may generate a prototype completely different from the typical example. This problem is examined in Section 3.2.

2.2.2 Previous Work

There exist many reported research projects related to learning and recognizing visual behavior. However, due to its recent introduction to the vision community, only a small number have been reported which use Hidden Markov Models. A few of the interesting works related to this thesis are summarized below.

Most of the early vision work using HMMs was limited to handwriting recognition, such as [8]. More recently, Starner, *et al* [11] explicitly used an established HMM speech recognition product for a real-time handwriting recognition system.

In work related to human gesture, Yamato, *et al* [13] used discrete HMMs to successfully recognize six different tennis swings among three subjects. Though the system used only 25x25-pixel images as the feature vector, it could decently distinguish the different motions. However, in order to use the discrete HMM, the domain was constrained enough to avoid the effects of vector quantization distortion. Furthermore, the system performed only isolated gesture recognition, in which gestures have already been temporally segmented from video.

Probably the most explicit example of human gesture is sign language, which has well-defined vocabulary and grammar. Starner [10] reapplied the speech community's HMM methods to recognize American Sign Language (ASL). Starner required the signer to sit in a particular position,

wear colored gloves, and refrain from finger signing. These requirements led to a single 8-dimensional feature vector consisting of each hand's x and y position, angle of axis of least inertia, and eccentricity of bounding ellipse. In order to achieve 97% accuracy with this simple feature vector, he used a strong grammar as well.

The system described by Cui, Swets, and Weng [5] was "a self-organizing framework... for learning and recognizing spatiotemporal events (or patterns) from intensity image sequences." They applied this system to hand sign recognition of a single hand. Though they wisely distinguished between the most expressive and most discriminating features, all features were essentially eigenvector coefficients. The example gestures were used to partition the visual space, where each partition had its own eigenvectors. The gesture was assumed to have been isolated in a temporal window from which 5 frames were sampled to represent the gesture. They claimed 96% recognition rate of a simple vocabulary; however, they also admitted to hand-tweaking a threshold to achieve this. Though it is not based on HMMs, it is instructive to see how much complexity HMMs incorporate by examining a system similar to this thesis that does not use them.

In an interesting application of HMMs, Bregler and Omohundro [4] presented a technique for lip reading tasks. The significance of their work was the incorporation of both auditory and visual features in an HMM system. To keep the vision task simple, their method was view-based. From the images, the human lip was modeled by smooth nonlinear manifolds. They reported significant improvements of continuous speech recognition in noisy environments by using the additional visual information.

Two papers describe the research leading to the conception of this thesis. In the first paper, Bobick and Wilson [3] defined a gesture to be a sequence of states in a measurement space. Unlike HMMs, this model allowed the calculation of a prototype trajectory to represent the gesture. This prototype was used to align the state pdf's along the most-likely direction by explicitly defining two types of variance, the along-trajectory variance and the across-trajectory variance. This

approach is significantly different from HMMs because the prototype allowed tracking of the gesture within a state, whereas HMMs cannot model the motion within a state.

In the second paper, Wilson and Bobick [12] used a state-membership measure to combine multiple representations at each state. Reestimation of the HMM parameters via the Baum-Welch algorithm was interleaved with reestimation of the representation parameters. By using the measure of state membership instead of the typical pdf's in HMMs, they were given more freedom in defining the representations, such as the use of eigenimages. However, though it was not observed, they state that this formulation may preclude convergence of the reestimated HMM parameters.

2.3 Summary

Pattern recognition forms the mathematical and procedural basis of this thesis. Feature extraction reduces raw observation vectors to feature vectors; clustering divides the feature space among perceptual states; and Hidden Markov Models represent the temporal behavior between those states.

Gesture recognition depends on the definition of gesture. In a scientific manner, this thesis defines gesture solely on the basis of feature measurements. This causes system performance to rely heavily on the quality of the features extracted but also allows reduction of gesture recognition to a purely computational form.

Chapter 3

Hidden Markov Models for Gesture Recognition

This chapter describes the core of the thesis. First, the system used to recognize gesture is described, followed by a discussion of how this work relates to previous research. Next, the experimental results and their interpretation are presented.

3.1 Description

Studying the domain of 2-dimensional mouse gestures, the experimental setup for this thesis is simple. First, a tool has been created for easily generating and modifying test suites of mouse data. In a window environment, the user can add and remove gestures and examples of gestures. The data, a four-feature vector of window-scaled position and velocity components sampled at approximately 20 Hz, is saved to a file in a generic test suite format. This format allows test suites created from other input devices, such as a program for extracting features from video, to be treated in the same way by the HMM system.

After the training data has been generated and stored as a test suite, the HMM recognizer program instantiates a new code book of specified size and a set of HMMs (one per gesture) with a specified number of states. The code book is clustered on the observation vectors using a modified

K-means algorithm, and the HMMs are created with a choice of left-right or ergodic transition matrix. The codebook implementation is described in more detail in Section B.2. After initialization, both the HMM and code book parameters are iteratively improved with a modified Baum-Welch algorithm, described in Section A.1. Upon convergence of the system parameters, the HMMs are immediately tested on the training data to verify the accuracy of training. At this point, the recognizer can also be tested with previously unseen gesture examples. After a successful recognizer has been created, it can easily be applied to examples of unknown type — the ultimate goal of pattern recognition.

3.2 Relation to Previous Work

This section describes how the work presented here is related to the previous work on the subject. Not only is the system here compared to other systems, but also the ideas from the other works that inspired this thesis are described.

Because the recognition system here is newly implemented with only initial experimentation, the application of this system most closely resembles the efforts in handwriting recognition. The domain of two-dimensional gestures is easy to implement but interesting and useful enough to be valuable. However, on-line recognition, as in [11] is not currently supported. The most significant difference between the methods of this thesis and these other 2-D gesture systems is the amount of training data required for good results. This thesis asserts that the large, painfully-obtained amounts of training data typical of the speech recognition methods is not required. Through future work, we hope to show that modification of the codebook can lead to good results with much smaller training sets.

Although the features are different and the observation vectors are reduced to discrete symbols, the training philosophy here is similar to that of Yamato *et. al.* [13]. First, only a few (6) distinct gestures are recognized. Second, a small amount of data is used to train the system. Three

people performed the 6 tennis strokes 10 times each; while five examples of each gesture were used for training, the other five were used for testing. The first major difference in the systems is their use of a hand-clustered vector quantization code book; this thesis uses the automatic clustering algorithm of Section 2.1.2. The most significant difference is the large number of states (36) they used to temporally represent each gesture. Considering that gesture examples had between 23 and 70 time steps, 36 states seems absurd. In the examples with 23 observation symbols, only two-thirds of the states can possibly be visited; in the larger examples, each state has too little duration to be significant. This paper reminds us that the results of HMM training do not necessarily make intuitive sense. This thesis attempts to force training of HMMs that make intuitive sense by reasonably limiting the number of states.

This thesis is different from Starner and Pentland [10] in two ways. First, for successful recognition in the domain of American Sign Language, they used a strong grammar. This thesis does not use a grammar to avoid both the dependency on any specific domain and the task of defining a grammar. Second, while their system used continuous HMMs in which each HMM was independently trained, this thesis uses semicontinuous HMMs, where the output classes must be shared by all HMMs. We feel that the independent training method is an inaccurate model of gesture when gestures are composed of similar atomic parts and that finding those common parts would lead to an understanding of gesture. We study the effects of sharing output classes in this thesis.

The work by Bobick and Wilson ([3] and [12]) greatly influenced the ideas in this thesis. First, they defined gesture as a trajectory in feature space and modeled it as such. This thesis uses the same definition but instead models the trajectory as a transition sequence among control states, allowing the use of established HMM procedures. Second, their use of the (observation) state-membership measure allowed the use of very different models in the same framework. This led to the ideas presented in Section 4.2 in which the feature space is divided into multiple independent subspaces.

3.3 Experiments

In order to understand the behavior and limitations of Hidden Markov Models, a number of initial experiments have been performed in the mouse gesture domain. This section describes the experiments and interprets their results.

Because we worked in the domain of 2-D mouse gestures, the observation vectors were mouse coordinates sampled at 20 Hz. Three types of feature vectors were used: position, velocity, and a single vector of position and velocity. While position was specified in scaled absolute window coordinates, velocity was represented by the difference of two temporally-adjacent position samples. The use of the absolute position feature depends on the presence of little positional variance in the training and testing sets; we readily satisfied this criterion in order to remove the effects of feature quality on HMM performance. However, for a real recognition system required to recognize widely varied gestures, better features would be used. As examples, a relative position feature allows recognition of gestures in very different positions, and normalized relative position allows gestures to have both differing positions and sizes. Because we felt gestures should be modeled as a pure sequence of conceptual states, the HMM transition matrices were forced to be strictly left to right. Each state had only two transitions: a self-loop and a transition to the next state.

Artificial Experiments. The first set of experiments were artificially created, non-mouse test suites designed to assure correct implementation of the reestimation procedures. The first test suite had three clusters in a one-dimensional feature space. Three gestures were defined that traversed the clusters in different orders. The code book clustering easily found the three distinct output spaces, and each HMM was given three states. Given 5 variable-length examples of each gesture, the recognizer converged in 8 iterations (2 seconds total execution time) and correctly recognized 100% of the examples. However, because of the random selection of initial codebook means, many executions of this test occasionally resulted in a single misrecognized example (93.3% cor-

rect). The second test suite contained two clusters at opposite corners of a finite two-dimensional feature space. 100% recognition of the training examples was always achieved on repeated executions.

Two conclusions were made regarding these experiments. First, we assumed the system was implemented correctly after convergence of a successful recognizer. Second, and more importantly, we realized the inherent dependency of success on the initial random means of the clustering algorithm. Although reestimation of the codebook parameters alongside the HMM parameters ensures an integrated system, the initial randomly selected means still influence the results on even the simplest of cases. This serves as a reminder to perform several independent training runs for any HMM.

Straight Lines. The first mouse-gesture test suite was a set of 8 different line gestures: horizontal, vertical, and both diagonals, each with a twin having opposite direction. The recognizer was given 8 code book clusters and 5 states per HMM and trained on five examples of each gesture, where the examples averaged 35 feature vectors (time samples) each. Each feature vector had 4 components to represent position and velocity. Testing on the training data resulted in 39 correct labels (97.5% correct), verifying successful training. A separate test suite was devised to show applicability to previously unseen gestures. These gestures had 10 examples each and, unlike the training examples, were in differing positions of the window. Still, 71 of the 80 examples were correctly labeled (88.75% correct).

The experiments on the 8 straight lines are significant for several reasons. First, this was the first experiment with "real" data, showing that the system worked for more complicated (and interesting) gestures. Second, the surprising success of verification demonstrated that our simple features (scaled window position and velocity) still gave the recognizer enough expression to distinguish the gestures. Lastly, the respectable performance on previously unseen gestures in significantly different positions from the training data showed that, as expected, in this experiment

position was not as important a feature as change in position. This leads to consideration of a system with independent feature subspaces, as described in Section 4.2.

A and D. The third set of experiments consisted of two gestures, a lowercase "a" and a lowercase "d." These two letters are very similar since "d" is really an "a" with an elongated vertical stroke. A 5-cluster recognizer with two 5-state HMMs was trained on 10 examples of each letter. Verification of the training showed that the Viterbi algorithm produced very similar scores for each HMM. Regardless of those similarities, the system was still able to correctly distinguish all 20 letters in the training set. When applied to an independent test set with 20 examples of each letter, only a single letter is misclassified (97.5% correct).

This simple experiment is significant because it demonstrates the difference between *representative* features and *discriminating* features. When the lowercase "a" and "d" are written in the same manner (started at the top of a counter-clockwise loop), the only discriminating feature is the length of the final vertical rise and fall. The trained HMM correctly found this difference and was able to recognize the test set very well. However, unlike the previous experiment, the letters were written with little positional variance. Because the Viterbi scores were so similar, if positional variance were present the slight effects of having or not having that feature would be lost in the deteriorated Viterbi scores, resulting in poor recognition. Further experimentation supported this claim. Figure 3.1 shows that the discriminating feature corresponded to an output class used exclusively by the "d" HMM, as one would hope. However, the strength of the feature was weakened because the single "d" state using the output class had a mixture of three output classes. The other two classes valid at that state blurred the distinction between an "a" and a "d". Furthermore, because the first three states of each HMM are identical, the 5-state HMMs can be reduced to 3-state HMMs in a canonical left-right form. It seems contradictory that the first three states are redundant while the last state is overloaded. Repeated runs with the same number of clusters and states but different features resulted in virtually the same HMM, suggesting that the Baum-Welch procedure

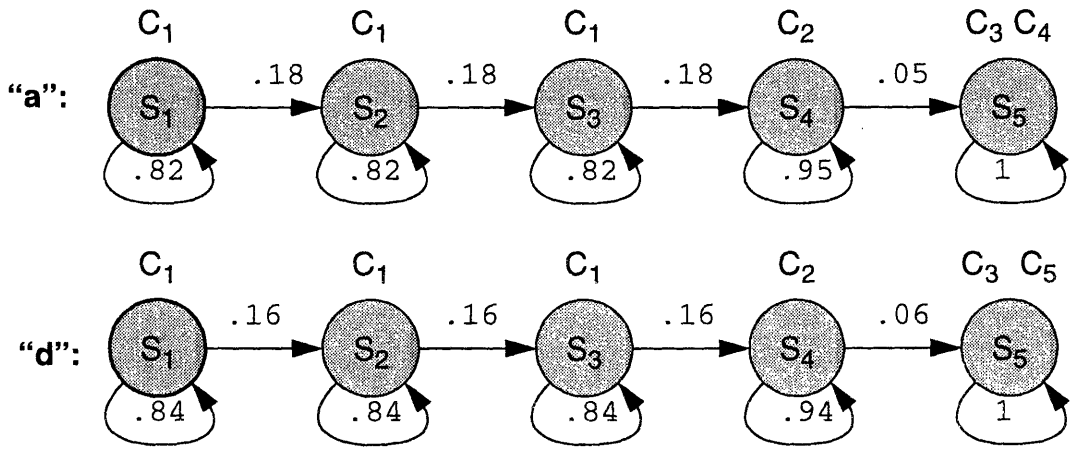

Figure 3.1 Inaccurate modeling of gesture. A mixture of Gaussians on the last state implies an equivalence of the output clusters where a sequential relationship is actually present. While the mixture coefficients on the last state of the "a" HMM are 0.85 and 0.15, the coefficients in the last state of the "d" HMM are 0.4 and 0.6.

kept finding the same local extremum. Randomization of initial HMM parameters may help to avoid this problem. This result is a reminder that HMMs statistically model patterns in any way they can, many times finding an accurate recognition representation that is nevertheless a disturbingly inaccurate model. The interesting behavior of this experiment led to further experimentation described later.

A, B, C, D, and E. The next experiment involved distinction of the lowercase letters "a", "b", "c", "d", and "e". Because the recognition task was difficult, we decided to try several recognizer architectures by varying the number of code book clusters and HMM states. Each experiment used five examples of each letter for training and verification, and 10 other examples for recognition of previously unseen observations. Table 3.1 shows the results of these experiments, where N is the

number of HMM states, K is the number of code book clusters, and the verification and test scores are the percentage of correctly labeled examples.

Table 3.1: Recognition of A, B, C, D, & E

N	K	Verification	Testing
4	5	88%	56%
4	10	96	66
5	4	76	54
5	5	100	72
5	6	80	56
5	7	84	64
5	10	96	62
6	5	100	58

The current system could not handle this experiment well. Though the recognizer learned the example gestures fairly well, it had only mediocre performance on independent test sets. Though we would like the HMMs to find the intuitively correct states and output classes, we see here that in fact the HMMs only do their best at describing the training data. In this example, the distinction between letters was very critically balanced so that the slightest variation in the test set resulted in misclassification. This problem is likely caused by too few training examples (only five of each letter were used here) and inadequacy of the features. From Table 3.1, we see that changing the HMM structure and code book size does little to improve performance on the test set.

A and D, revisited. In order to understand the behavior of the earlier experiment for distinguishing A and D, the experiment was repeated with different HMM architectures. Because we wanted to remove the dependency of HMM performance on the quality of features while ensuring that the features were expressive enough, the only feature used here was absolute scaled position, and each example was written in approximately the same position at the same scale. The goal of this exper-

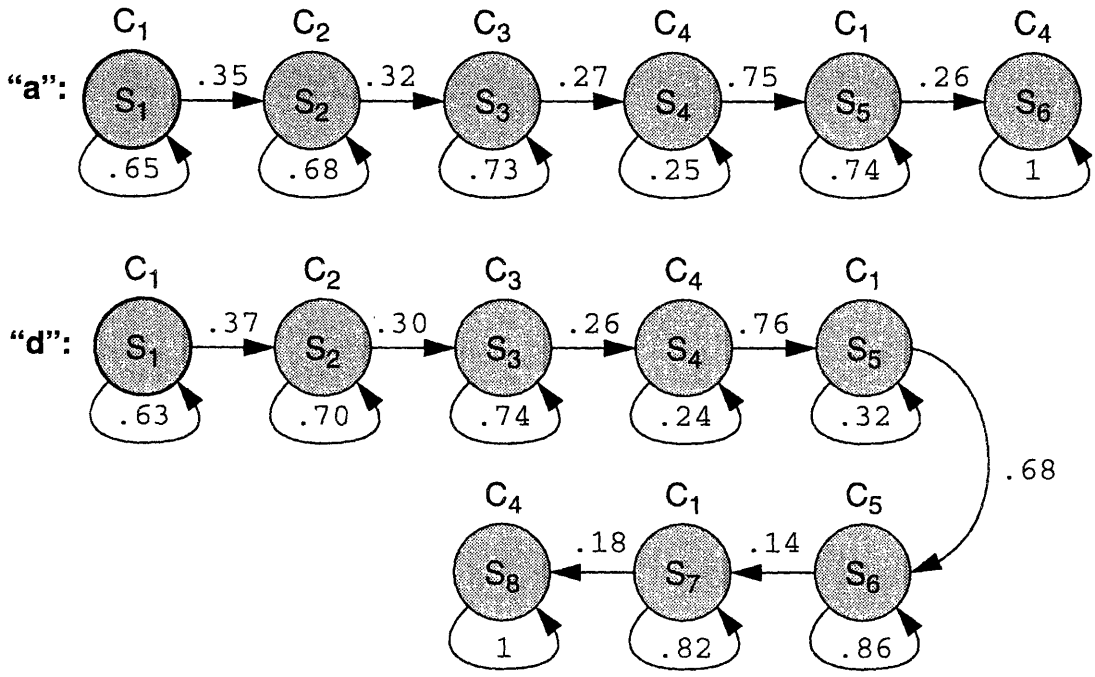

Figure 3.2 Accurate modeling of gesture. The last few strokes of the gestures have been divided among separate control states, accurately modeling the sequential behavior of the strokes. C_5 is the output cluster corresponding to the distinguishing feature.

iment was to identify a way of forcing the HMM to find the sequential relationship in the last few strokes of the gestures, *i.e.*, to find an accurate modeling of the gesture, rather than clumping the critical sequential relationship on a single state. Varying the number of states from 4 to 10 was unsuccessful in finding a better-converged canonical HMM. Finally, an HMM with 20 states found the sequential relationship. The underlying canonical HMMs are pictured in Figure 3.2. However, though the critical relationship was found and all tests were 100% accurate, the "a" HMM still could not distinguish the letters as well as the "d" HMM.

This experiment provided significant insight into the training of left-right HMMs. First, in order to find the important temporal relationships enough states must be used to avoid "undersampling" of the pattern's temporal behavior. Second, the left-right model has a simple reduction to a canonical form. Thus, a good strategy for ensuring accurate modeling is to train with a large number of states, followed by reduction to canonical form. Furthermore, the same performance of the

"a" HMM shows that not only does accurate recognition not imply accurate modeling but also accurate modeling does not imply robust recognition, as previously thought.

3.4 Summary

A system for training and testing semicontinuous HMMs has been developed and applied to the domain of 2-D mouse gestures. This system was compared to related works by other authors, showing the ideas derived from previous work.

Initial experimentation has helped to identify several interesting and important behaviors of HMMs. First, we are reminded that the initial random clustering of the code book forces us to consider several training sessions before making conclusions. Second, we see that the system critically depends on the quality of the chosen features. In the simple experiments, absolute position and velocity were sufficient for recognition; however, the slightly more complicated 5-letter recognition task could not be successfully applied to very similar independent test suites. Furthermore, the important distinction between representative and distinctive features requires us to reevaluate the modeling of state output in the HMM formulation. In addition, we have seen that, although an HMM system may have learned to recognize the training set, it has not necessarily learned any underlying, intuitive traits for robustly classifying other observations. Instead, it may be critically balanced on the assumption that there will be no variance it has not seen before. Finally, even though the intuitively satisfying HMM may have been found, it may still not have the robust performance we hope to find.

Chapter 4

Conclusion

This chapter summarizes the important empirical results and describes further experiments and future improvements to the current gesture recognition system.

4.1 Summary

A system for gesture recognition has been designed and implemented. Though initial experimentation was simple, poor performance was observed on a modestly complicated task, and simple tasks led to sufficient but inaccurate models. Rather than discouraging further studies, these experiments demonstrate that many factors contribute to the success of an HMM recognition system. Further experimentation will aid in identifying those factors.

In the statistically-based Hidden Markov Model, accurate recognition does not imply accurate modeling. Oftentimes, the trained HMM does not make intuitive sense yet successfully recognizes the training set and similar testing sets. However, these HMMs not only recognize many absurd patterns as well as the real patterns but also fail miserably on slightly different test sets because of a critical balance that fails with previously unseen variances. In order to stabilize this critical balance, many systems rely on large training sets for enumerating the variances; however, collection and labeling of all the data is costly. Originally motivated by reducing the number of free HMM parameters, Huang's semicontinuous HMM makes the similarities among patterns explicit. The

approach of this thesis has two key advantages over normal continuous HMMs. First, the accurate modeling creates a robust, variance-insensitive HMM without the need for large amounts of data. Second, the global codebook makes the similarities among gestures explicit, allowing an "understanding" of the gesture relationships. Through several experiments with 2-dimensional mouse gestures, this thesis analyzed the behavior of HMM training and reported some important insights.

Mixtures of Gaussians provide a better representation of a control state's observation membership; at the same time, it gives the Baum-Welch reestimation procedure more freedom to choose an inaccurate model. Figure 4.1 shows three distinctly-behaving gesture segments that may be confused as a mixture of Gaussians during training. In a left-right HMM, we found that we could prevent this problem by providing enough states for training, followed by state merging to reduce the HMM to a canonical form. Further thought should be directed at preventing this problem for other HMM architectures.

In the course of this thesis, we have identified several important ideas on how HMMs should be used. First, because training is a very expensive procedure, we feel every attempt should be taken to minimize the amount of training data needed to obtain good results. This can be achieved by better features and removal of similar examples. We would like to see experiments use smarter data selection methods to show the minimal data set for a successful recognizer. Second, as gestures become more complicated, inspecting the values of the system parameters becomes unmanageable. Observation of the system behavior will be facilitated by visualization, described in Section 4.2. Finally, the quality of the features can sometimes solely determine the performance of a recognition system. Instead of forcing the system to used bad features, we suggest a scheme for breaking up the feature space into independent feature subspaces. This can be accomplished by either hardcoding zeros in the covariance matrices of the code book to force statistical independence, or by creating a separate code book for each feature subspace. To the author, the latter approach seems most promising.

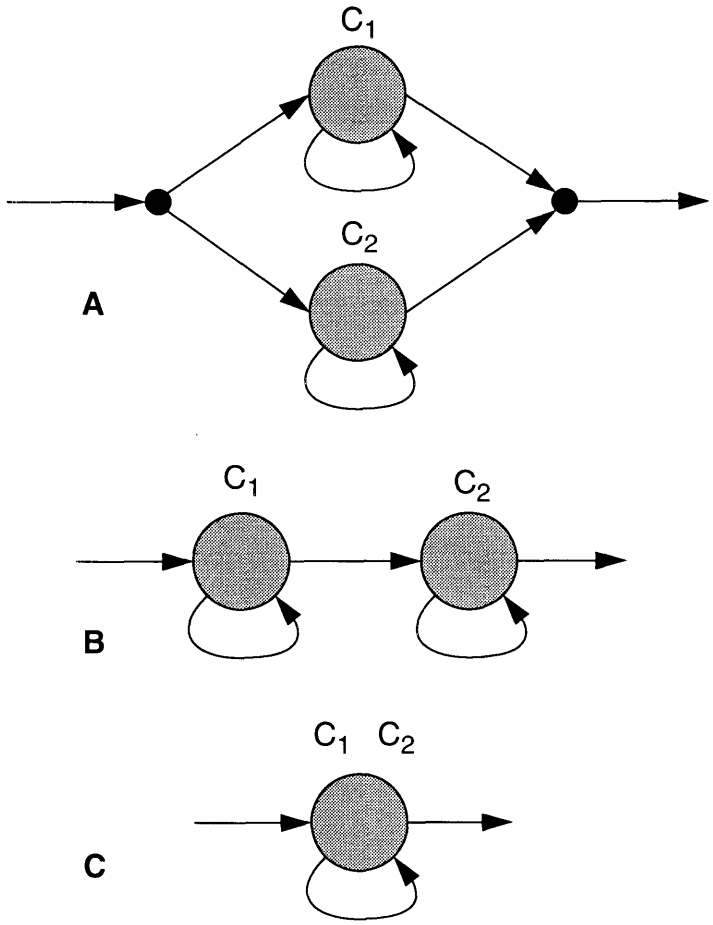

Figure 4.1 Confusion of inequivalent HMM constructs. All three constructs, each with very different behavior, can be confused as a single state with a mixture of Gaussians. (a) shows a parallel (exclusive-or) relationship, (b) shows a sequential (juxtaposition) relationship, and (c) shows a nondeterministic (equivalence) relationship. Only for (c) is such a model accurate.

HMMs show potential, but they lack several useful traits. First, they have no notion of a prototype, disallowing the summarization of gesture. Rather than identifying an HMM with its given name, it would be very useful to identify it with a typical gesture. Second, in the experiments here we see a trade off between finding the representative features, resulting in an understanding of gesture similarities, and finding the distinguishing features, resulting in distinction of gestures. Further research should attend this problem and suggest ways to modify the HMM. Lastly, the probabilistic nature of the state transitions allows gross errors, such as omission, to go undetected. For example, an "a" is a "d" with a short stem, causing the "d" HMM to score well by simply tra-

versing a state transition quickly. Perhaps this type of problem can be avoided by introducing the notion of state duration, as described in [9].

4.2 Future Work

As with any research project, many enhancements and extensions have become apparent through the course of the work. First, in the domain of 2-D mouse gestures, more experiments should be performed. Most importantly, more robust and useful features should be extracted, such as angle, change in angle, and relative position of the mouse instead of absolute position. This will improve recognition results. Furthermore, a very interesting extension to the system would allow independence of feature subspaces. For example, keeping the position and velocity features separate will reduce the number of system parameters (by removing entries in the code book's covariance matrices) and perhaps lead to better convergence. Of course, asserting that two subspaces of the feature space are orthogonal will have major repercussions, inviting experimentation on the subject. This idea can be extended by creating two separate code books, one for each feature subspace. Finally, other common extensions to the HMM framework may help performance, such as time duration modeling and corrective training.

Second, to allow more interesting studies, the experimental setup should contain data collection more complex than 2-D mouse gestures. For example, by using video sequences, the system can be applied to the more typical vision domains, allowing many more applications of the recognition system. In addition, each NxM video frame is an NM-element observation vector, allowing many more operators for feature extraction. This gives the system more freedom in choosing the relevant features. Of course, larger feature vectors demand better execution performance.

Independent of processor speeds, execution performance can be improved in two ways. The first method is to directly optimize the C++ program and use the compiler options for generating optimized code. A second method is to simplify the problem. For example, large feature vectors

can be replaced by smaller ones; or observation sequences can be undersampled to halve the number of observation vectors. Another way of simplifying the problem is to use the discrete HMM instead of the semicontinuous HMM. The discrete HMM reestimation equations are considerably simpler than those for the semicontinuous. An interesting study would be to compare the trade-offs between discrete HMM simplicity and semicontinuous HMM expressiveness.

A more practical extension with immediate need is to create tools for visualization of the HMM procedures. Independent of the domain for which HMMs are being applied, visualization would promote a much better understanding of the system dynamics. For example, if an experimenter notices that the HMM parameters have already converged, he can halt the training process. This type of visualization can be achieved by displaying the parameter matrices as images, where the brightness of an image location corresponds to the relative weight of an entry in the matrix. As another example, when a human sees not only the strongest HMM for describing an example but also its strength relative to other HMMs, the label assigned to the example in recognition can be qualified. The code book, too, can be better understood through visualization. When a two-dimensional code book can be used, a window can display the 2-D pdf's in the feature space. For example, a two-codebook mouse gesture feature space can be represented in two windows, one with a 2-D subspace for position and one with a 2-D subspace for velocity. When 2-D features are not present, simply displaying the covariance matrices can aid in deciding if an independent feature subspace exists, where a separate code book may then be created.

Bibliography

[1] L. E. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of markov processes. *Inequalities*, 3:1-8, 1972.

[2] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Stat.*, 41:164-171, 1970.

[3] A. F. Bobick and A. D. Wilson. Using configuration states for the representation and recognition of gesture. In *Proc. Fifth International Conf. on Computer Vision*, pages 631-636. IEEE Press, 1995.

[4] C. Bregler and S. M. Omohundro. Nonlinear manifold learning for visual speech recognition. In *Proc. Fifth International Conf. on Computer Vision*, pages 494-499. IEEE Press, 1995.

[5] Y. Cui, D.L. Swets, and J. J. Weng. Learning-based hand sign recognition using SHOSH-LIF-M. In *Proc. Fifth International Conf. on Computer Vision*, pages 631-636. IEEE Press, 1995.

[6] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, Inc., New York, 1973.

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

[8] R. Nag, K. H. Wong, and F. Fallside. Script recognition using hidden markov models. In *ICASSP 86*, 1986.

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

[10] T. E. Starner and A. Pentland. Visual recognition of American Sign Language using hidden markov models. In *Proc. of the Intl. Workshop on Automatic Face- and Gesture-Recognition*, Zurich, 1995.

[11] Thad Starner, John Makhoul, Richard Schwartz, and George Chou. On-line cursive handwriting recognition using speech recognition methods. In *ICASSP 94*, 1994.

[12] A. D. Wilson and A. F. Bobick. Learning Visual Behavior for Gesture Analysis. To appear in Proc. IEEE Symposium on Computer Vision, Coral Gables, Florida, Nov. 20-22, 1995.

[13] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov models. In *Proc. 1992 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 379-385. IEEE Press, 1992.

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

Appendix A

HMM Algorithms

This appendix provides a simplified description of the algorithms used for training and testing the semicontinuous Hidden Markov Model. The first algorithm, the Baum-Welch procedure, is used to reestimate the model parameters. The second algorithm, the Viterbi procedure, is used to evaluate the ability of an HMM at describing a particular observation. For a much more detailed discussion of these algorithms, [7] is recommended.

A.1 Parameter Reestimation

Given a set of examples $\vec{X}$ for a particular gesture, an ideal system will create the HMM λ that most likely (as opposed to other HMMs) generated those examples. In other words, it finds λ such that $P(\lambda|\vec{X})$ is maximized. Such a system seems intractable, however. Instead, given the examples and a particular HMM, the Baum-Welch algorithm attempts to change the HMM parameters so that the given HMM most likely generated the examples (as opposed to other examples), *i.e.*, it maximizes $P(\vec{X}|\lambda)$. Although this procedure is far from ideal, it has been proven to converge to locally optimal parameters. This section describes the extended Baum-Welch algorithm used to reestimate both the HMM and codebook parameters. Table A.1 lists the variables used to describe the models and procedures.

Table A.1: Semicontinuous HMM Notation

Symbol	Description
$\vec{X}$	observation sequence
T	number of feature vectors in observation sequence
$\vec{x}_t$	feature vector at time t
M	number of clusters in the codebook
C_k	kth cluster of the codebook
μ_k	estimated mean of the cluster C_k
Σ_k	estimated covariance matrix of the cluster C_k
$f(\vec{x}_t \mid C_k)$	probability that $\vec{x}_t$ was produced by cluster C_k
N	number of states in the HMM
s_t	state at time t
S	set of states in the HMM
a_{ij}	transition probability from state i to state j
A	$N \times N$ matrix of state transition probabilities
$b_j(k)$	probability that state j produced an observation from cluster C_k
B	$N \times M$ matrix of output probabilities

In order to derive the equations for reestimating the model parameters, it is convenient to define several intermediate probabilities. First, the *forward probability* $\alpha_t(j)$ is defined as the joint probability of observing the first t vectors and being in state j at time t. In other words, $\alpha_t(j) = P(\vec{x}_1, \ldots, \vec{x}_t, s_t = j \mid \lambda)$. In an HMM model in which states 1 and N are nonemitting, the forward probability is recursively calculated by the equation

$$\alpha_t(j) = \left[\sum_{i=2}^{N-1} \alpha_{t-1}(i)\, a_{ij}\right] b_j(\vec{x}_t) \tag{A.1}$$

with initial conditions $\alpha_1(1) = 1$ and $\alpha_1(j) = a_{1j} b_j(\vec{x}_1)$ for $1 < j < N$ and final condition $\alpha_T(N) = \sum_{i=2}^{N-1} \alpha_T(i)\, a_{iN}$. This recursion asserts that the probability of being in state j at time t and seeing observation $\vec{x}_t$ can be calculated by adding the forward probabilities for all possible predecessor states i weighted by the transition probability a_{ij}. In a similar but opposite manner,

the *backward probability* $\beta_t(i) = P(\vec{x}_{t+1}, \ldots, \vec{x}_T | s_t = i, \lambda)$ can be recursively computed using the equation

$$\beta_t(i) = \sum_{j=2}^{N-1} a_{ij} b_j(\vec{x}_{t+1}) \beta_{t+1}(j) \tag{A.2}$$

with the initial condition $\beta_T(i) = a_{iN}$ for $1 < i < N$ and the final condition $\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(\vec{x}_1) \beta_1(j)$. Together, the two procedures form the *forward-backward algorithm*. The forward and backward probabilities lead to a convenient method of finding the likelihood of state occupation, $\gamma_t(j) = P(s_t = j | \vec{X}, \lambda)$. The equation is simply $\gamma_t(j) = \frac{1}{P} \alpha_t(j) \beta_t(j)$, where P is the probability of observing the sequence $\vec{X}$ given the model λ: $P = P(\vec{X}|\lambda) = \alpha_T(N)$. These intermediate probabilities are used to compute the reestimated parameters. The details can be found in [9] and [7].

A.2 Recognition Algorithm

While training the HMM is an involved process, performing recognition is much simpler. Given an observation, all HMMs are scored based on how well they describe the sequence. The HMM with the highest score is chosen as the likely generator of the observation, resulting in a label of that HMM's gesture.

The Viterbi algorithm finds the single best state sequence $\hat{s} = (s_1, s_2, \ldots, s_T)$ for the observation sequence $\vec{X} = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T)$. In this thesis, we only need the score of the state sequence rather than the actual state sequence. The intermediate score is defined recursively as

$$\delta_t(j) = \max_i [\delta_{t-1}(i) a_{ij}] b_t(j) \tag{A.3}$$

with initial conditions $\delta_1(1) = 1$ and $\delta_1(j) = a_{1j} b_1(j)$ for $1 < j < N$ and final condition $\delta_T(N) = \max_i [\delta_T(i) a_{iN}]$. The score of the HMM for a particular observation is simply $P(\vec{X}|\lambda) = \delta_T(N)$. To decide the gesture label of a particular observation, all HMMs go through the Viterbi procedure. The example's classification is the gesture corresponding to the highest

scoring HMM. The implementation considerations of this procedure are detailed in Section B.1, which describes the implementation of the Hidden Markov Model algorithms.

Appendix B

Implementation

The entire system has been implemented in the C++ programming language for several reasons. First, the ANSI C++ standards make the system (*sans* user interfaces) portable to other platforms with an ANSI C++ compiler. The applications with user interfaces can be compiled for any machine using the X Window System. Second, the C++ compilers generate highly efficient code, reducing execution time of the training and recognition procedures. Finally, C++ classes provide a very effective environment for managing system complexity and reusing code.

The bulk of the system relies on two portable libraries. The Matrix library provides a useful matrix abstraction for all normal types (`char`, `int`, `float`, `double`). It has been optimized for both matrix operations and memory management. Where possible, matrix operations are performed with highly efficient pointer arithmetic. In a program with many allocations and deallocations of large objects, the program's memory segment becomes fragmented, causing significant execution time to be spent on copying the objects to fit compactly in memory. In the Matrix class, normal allocation and deallocation of matrices has been overridden to allow reuse of pointers to large matrix objects, easing the memory management task of the operating system. Both optimizations have demonstrated noticeable improvement in execution speeds.

The second portable library is the MotifApp library. Built on top of the X and Motif libraries, the MotifApp library provides many classes for user interface components. For example, the

OptionMenu class allows for easy construction and use of an option menu. The MotifApp library has proven to greatly simplify the complicated task of constructing a user interface. The Mouse Gesture Creator has been implemented using this library.

The following sections describe in more detail the current implementations of the Hidden Markov Models and Codebooks.

B.1 Hidden Markov Models

The `Hmm` class instantiates the semicontinuous Hidden Markov Models. Using the Matrix library, the `Hmm` class represents each HMM by the three matrices A, B, and π. Note that this is different from the more general formulation described in Section A.1, where each HMM has a single initial and final state that produces no output. Because the more general representation facilitates continuous recognition of gestures (recognition without requiring segmentation of gestures), the current implementation will be replaced in the future.

Reestimating the HMM parameters is performed using the specifications described in Appendix A. However, naively implementing the equations directly leads to very inefficient code. Instead, the reestimation procedures have been partially optimized by removing needless intermediate values. For example, the denominator terms in the reestimation equations are used to enforce the stochastic constraint that individual probabilities sum to 1. This computation is removed by simply normalizing all the individual probabilities after all numerators have been accumulated.

Another problem with directly implementing the equations of Section A.1 is caused by the finite precision of a computer's internal representation of numbers. In the forward-backward algorithm, the intermediate probabilities fall quickly to zero. Even a gesture observation with just 50 time steps can result in arithmetic underflow. In severe situations, an entire row of the accumulator becomes zero and normalization of that row leads to division by zero. Two solutions avoid this problem. The first solution introduces a new intermediate value $c(t) = \sum_{i=1}^{N} \alpha_t(i)$, which is

used to scale the $\alpha_t(i)$ and $\beta_t(i)$ values. The new intermediate probabilities become $\bar{\alpha}_t(i) = \frac{\alpha_t(i)}{c(t)}$ and $\bar{\beta}_t(i) = \frac{\beta_t(i)}{c(t)}$. When the scaling is performed during the forward-backward procedure, all intermediate values are safe from underflow. The downfall with this approach is that normal computation of the scoring probability $P(\vec{X}|\lambda)$ still leads to underflow, resulting in a logarithmic computation. The second solution computes all probabilities in a log manner. All initial values are replaced by their logarithm, and all multiplications become simple additions. The serious downfall with this approach is an inability to easily handle addition, which becomes a complicated computation. In order to improve the addition performance, a complex procedure involving lookup tables is described in [7]. Because the training process does not necessarily need to compute the scoring probability, the current implementation uses the former approach for its simplicity. Only for the Viterbi algorithm is the computation done with logarithms (an easy modification) to prevent underflow.

Finally, the equations of Section A.1 reestimate the parameters given only a single observation sequence. To obtain any worthwhile results from an HMM, the accumulation process must be extended. First, many example gestures must be used. Fortunately, directly extending the accumulation process to all examples accomplishes multiple-example training. Second, any useful system will require recognition of multiple gestures. The codebook accumulators are changed to accumulate values over all HMMs. Thus, for a multiple-gesture, multiple-example system the reestimation procedure becomes

```
do until little change in parameters {
    clear accumulators for code book;
    for each gesture G_i {
        clear accumulators for HMM λ_i;
        for each observation sequence X of G_i {
            for each observation vector x_t {
                accumulate values for λ_i;
                accumulate code book values;
            }
        }
        reestimate λ_i parameters;
    }
```

```
    reestimate code book parameters;
}
```

B.2 Code Book

The `CodeBook` class implements the code book abstraction used for probabilistically classifying the observation vectors.

Given a set of observation vectors, the code book is constructed in two steps: clustering and estimating. The clustering procedure is a modified K-means algorithm. First, to ensure the initial means are on the correct scale, the initial means are randomly selected observation vectors. All observations are grouped with their closest mean, new means are computed for each group, and the process is repeated until the computed means have converged. During these iterations, whenever a cluster has less than `min` elements, it is removed. In this way, the user-specified K is an upper limit on the actual number of clusters in the code book. For all of the experiments presented in this thesis, the value of `min` was 2, which is the smallest number of vectors for which a cluster will have nonzero covariance. The second step, estimating the code book classes, assumes that each cluster of observations can be probabilistically modeled by a multivariate Gaussian distribution. Using the means computed above as the estimated mean $\hat{m}_i$ of cluster C_i, the estimated covariance matrix of each distribution is found by $\Sigma_i = E\left[(\hat{x} - \hat{m}_i) (\hat{x} - \hat{m}_i)^T \right], \forall \hat{x} \in C_i$, where $E[x]$ denotes the expectation of x.

After the code book has been constructed, the μ_i and Σ_i parameters are reestimated alongside the HMM parameters using the algorithm described in Section A.1. Theoretically, this reestimation customizes the code book for the HMMs, implying better recognition performance. However, in the experiments presented here, this reestimation sometimes led to a cluster with an entire row and column of zeros in the covariance matrix. This was caused by shifting, scaling, and rotating of the pdf's until a dimension of the feature space no longer influenced representation of the cluster. This problem manifests itself when computing the determinant of the singular covariance matrix

causes an error. It may be possible to avoid this by requiring *min* vectors in all clusters at all times, removing those that do not, as in the initial code book clustering.