Value-Directed Human Behavior Analysis from Video Using Partially Observable Markov Decision Processes

Jesse Hoey and James J. Little, Member, IEEE

Abstract—This paper presents a method for learning decision theoretic models of human behaviors from video data. Our system learns relationships between the movements of a person, the context in which they are acting, and a utility function. This learning makes explicit that the meaning of a behavior to an observer is contained in its relationship to actions and outcomes. An agent wishing to capitalize on these relationships must learn to distinguish the behaviors according to how they help the agent to maximize utility. The model we use is a partially observable Markov decision process, or POMDP. The video observations are integrated into the POMDP using a dynamic Bayesian network that creates spatial and temporal abstractions amenable to decision making at the high level. The parameters of the model are learned from training data using an a posteriori constrained optimization technique based on the expectation-maximization algorithm. The system automatically discovers classes of behaviors and determines which are important for choosing actions that optimize over the utility of possible outcomes. This type of learning obviates the need for labeled data from expert knowledge about which behaviors are significant and removes bias about what behaviors may be useful to recognize in a particular situation. We show results in three interactions: a single player imitation game, a gestural robotic control problem, and a card game played by two people.

Index Terms—Face and gesture recognition, video analysis, motion, statistical models, clustering algorithms, machine learning, parameter learning, control theory, dynamic programming.

1 INTRODUCTION

This paper describes a model of human behaviors that unifies computer vision and decision theory through a framework of probabilistic modeling. The motivation is that computational agents will need capabilities for learning, recognizing, and using the extensive range of human nonverbal communication skills. The perceiver of a nonverbal signal must not only recognize the signal, but must understand what it is useful for. The signal’s usefulness will be defined by its relationship to both signaler and receiver, their actions, their possible futures together, and the individual ways they assign value to these futures. This paper describes a method for the automatic learning and analysis of purposeful, context-dependent, human nonverbal behavior. No prior knowledge about the structure of behaviors or the number of behaviors is necessary. The method learns which behaviors (and how many) are conducive to achieving value in the context. An important aspect of our work is the explicit modeling of uncertainty in both the observations and in the temporal dynamics of the system. Taking this uncertainty into account allows the system to better optimize over possible outcomes based on noisy visual data. This paper will focus on human nonverbal communicative behaviors, including facial expressions and hand gestures. We will use the term display to refer to face or hand behaviors.

There has been a growing body of work in the past decade on the communicative function of the face [24], [46] and of the hands [38]. This psychological research has drawn three major conclusions. First, displays are often purposeful communicative signals [24]. Second, the purpose is not defined by the display alone, but also depends on the context in which the display was emitted [46]. Third, the signals are not universal, but vary between individuals in their physical appearance, their contextual relationships, and their purpose [46]. We believe that these three considerations should be used as critical constraints in the design of computational communicative agents able to learn, recognize, and use human behavior. First, context dependence implies that the agent must model the relationships between the displays and the context. Second, the agent must be able to compute the utility of taking actions in situations involving purposeful displays. Third, the agent needs to adapt to new partners and new situations.

These constraints can be integrated into a decision-theoretic, vision-based model of human nonverbal signals. The model can be used to predict human behavior or to choose actions that maximize expected utility. The basis for this model is a partially observable Markov decision process, or POMDP [1]. A POMDP describes the effects of an agent’s actions upon its environment, the utility of states in the environment, and the relationship between the observations, the actions, and the states. A POMDP model allows an agent to predict the long term effects of its actions upon its environment and to choose valuable actions based on these predictions. The model can be acquired from data and can be used for decision making based, in part, on the nonverbal behavior of a human through observation.
Our work is distinguished from other work on recognizing human nonverbal behavior primarily because it automatically learns what behaviors are distinguishable in the data and useful in the task. We do not train classifiers for predefined behaviors and then base decisions upon the classifier outputs. Instead, the training process discovers categories of behaviors in the data. This method is general in that it removes the need to reengineer a model for each task. This is especially important in the design of human-interactive systems since the human behaviors that will be observed are typically poorly known at the time of model specification. Instead, our method does not require expert knowledge about which displays are important nor extensive human labeling of data. Expert data labeling is not only time consuming, but also unnecessarily constrains the resulting models to the types of behaviors believed to be important by the experts.

In contrast, the Facial Action Coding System (FACS) has become the standard for psychological inquiries into facial expression. Computer vision researchers have made significant progress toward automatic FACS coding [2], [21], [55]. However, although the recognition of facial action units may give the ability to discriminate between very subtle differences in facial motion, or “microactions,” it requires extensive training and domain specific knowledge [2]. Further, the importance of such a detailed level of representation is not clear for computer vision systems that intend to take actions based on observations of humans [42] and much simpler representations will be sufficient in many tasks. Our method takes a more task-oriented approach, automatically finding and modeling behaviors that are sufficient for performance.

There will be little discussion of speech recognition and natural language understanding in this paper. Our work focuses solely on recognizing and using nonverbal communicative acts. However, it is well known that gesture and facial expression are intimately tied to speech [14], [38] and one might object to our omission. Nevertheless, our models are theoretically well grounded and models of the same type have been used extensively in speech recognition [45] and dialogue management [41], [57]. Therefore, we believe that our models are ideally suited for integration with dialogue modeling work at some time in the future.

This paper is organized as follows: We first review previous work in modeling human behaviors from both labeled and unlabeled data. Section 3 then describes a general POMDP model of human behavior modeling within a task and Section 4 goes into more detail of a specific POMDP model. Finally, Section 5 discusses the results of learning the model in three situations.

## 2 Previous Work

Learning and solving POMDPs is a well-studied problem in the artificial intelligence literature. One of the main areas of research is into learning, solving, and using a POMDP simultaneously, online, using reinforcement learning [52]. Although these approaches carry optimality guarantees, they do not scale well to real-world domains. On the other hand, recent developments in efficient approximate solution techniques for specific POMDPs (where the model is known) have shown promise for solving large, realistic problems [9], [29], [50]. However, these methods do not approach the problem of learning the models and typically assume that sensing yields simple observations that are easily quantified.

This is a problem in domains that require interactions with humans as the inputs to the decision maker will be very complex. For example, if human facial expressions need to be distinguished for performance in some task, then the observations will be entire sequences of video. Therefore, the decision making process needs to be coupled with some hierarchical modeling that takes care of spatio-temporally abstracting these complex observations. This problem is well studied in computer vision [7], [11], [12], [40], [51], where the goal is to train a classifier to recognize some predefined set of behaviors using supervised learning. Our work instead addresses the coupling between the behavior models and the decision making process: We automatically learn which behaviors are important to recognize.

Representation of human behavior in video is usually done by first estimating some quantity of interest at the pixel level and then spatially abstracting this to a low-dimensional feature vector. Optical flow [6], [20], [21], color blobs [12], [17], deformable models [35], motion energy [7], and filtered images [2] are the more well-used pixel-level features. Spatial abstraction is usually approached using either a model-based or a view-based representation of a body part. Model-based approaches are often three-dimensional wire-frame models [2], sometimes including musculature [21]. View-based approaches spatially abstract video frames by projecting them to a low-dimensional subspace, using, for example, principal components [6], [23], [56], Fisher linear discriminants [3], or independent component analysis [2]. Other representations of faces and bodies use templates [7], feature points [35], or “blobs” [12], [51].

Our work uses Zernike basis functions [44] for holistic representation of the face and facial motion. The Zernike polynomial basis provides a rich and independent description of optical flow fields and gray-scale images. When applied to optical flow, the Zernike basis can be seen as an extension of the standard affine basis [27]. The Zernike representation differs from approaches such as Eigen-analysis [56], or facial action unit recognition [55] in that it makes no commitment to a particular type of motion, leading to a transportable classification system (e.g., usable for clustering different types of behaviors), which is necessary for a general modeling technique such as we are pursuing. Zernike polynomials have been used for recognizing hand poses [32], handwriting and silhouettes [53], and optical flow fields [28]. It is important to note, however, that our learning method does not rely on this particular choice of basis function and others could be investigated within the same method.

Once features are computed for each frame, their temporal progression must be modeled. Spatio-temporal templates [21], dynamic time warping [18], and hidden Markov models [51] are popular approaches. HMMs have been applied to many recognition problems in computer vision, such as hand gestures [47], American Sign Language [51], and facial action units [2].

There are many DBN extensions of HMMs, including the coupled hidden Markov model [11]. Hierarchical models [22] are particularly interesting as they incorporate temporal abstractions and have been used for modeling full body motions [12]. However, learning a hierarchy is a notoriously difficult problem. Seer [40] implements a real-time hierarchical model using a layered HMM, where each layer represents events at a different temporal granularity. However, in Seer, each layer is trained independently from labeled data. The DBN underlying a POMDP is known as the input-output hidden Markov model [5]. POMDPs have
been used in realistic domains, including robot control [54] and spoken dialogue management [57]. Darrell and Pentland used POMDPs for control of an active camera [17]. Their POMDP model was trained to foveate regions containing information of interest, such as hands during gesturing. However, their work is focused on computing policies in a reinforcement learning setting. They do not learn the number of behaviors and they separate visual recognition from decision making.

Most of the methods we have been describing are trained with labeled data, which requires human intervention and makes adaptivity more difficult. The alternative is to develop with labeled data, which requires human intervention and maximizes some notion of utility. In this section, we first give an overview of POMDPs in Section 3.1, followed by a discussion in Section 3.2 of how to learn the parameters of a POMDP from data with the expectation-maximization algorithm. We show in Section 3.3 how to approximately solve the POMDP by solving the associated MDP model.

3 LEARNING AND SOLVING POMDPS

A partially observable Markov decision process (POMDP) is a probabilistic temporal model of an agent interacting with the environment. POMDPs can be learned from data, and, once specified, can be used to compute policies of action that maximize some notion of utility. In this section, we first give an overview of POMDPs in Section 3.1, followed by a discussion in Section 3.2 of how to learn the parameters of a POMDP from data with the expectation-maximization algorithm. We show in Section 3.3 how to approximately solve the POMDP by solving the associated MDP model. The solution to the POMDP gives indications about the structure of the model and about how useful this structure is for achieving value.

In this paper, we use standard notation for variables, where capital letters denote random variables, while small letters denote an instantiation of that variable. Subscripts on small letters denote a particular value of a variable. Bold-faced letters represent sets of variables, usually referring to sets that extend over time (e.g., sequences of data). Thus, \( X = \{X_1, \ldots, X_N\} \) is a set of \( N \) random variables, \( x = \{x_1, \ldots, x_N\} \) is an assignment of values to those variables, and \( x_i \) is an assignment to \( X_i \). We will also write \( x_{i,j} \) as the particular assignment \( X_i = j \).

3.1 Overview

A POMDP is a tuple \( \langle S, A, \Theta_S, R, O, \Theta_O \rangle \), where \( S \) is a finite set of (possibly unobservable) states of the environment, \( A \) is a finite set of agent actions, \( \Theta_S : S \times A \rightarrow S \) is a transition function that describes the effects of agent actions upon the world states, \( R : S \times A \rightarrow \mathbb{R} \) is a reward function that gives the expected reward for taking action \( A \) in state \( S, O \) is a set of observations, and \( \Theta_O : S \times A \rightarrow O \) is an observation function that gives the probability of observations in each state-action pair. Fig. 1a shows two time slices of a POMDP as a dynamic Bayesian network (DBN). Shaded nodes denote observables, unshaded nodes denote unobservable variables. Parameter random variables are denoted by \( \Theta \), prior hyperparameters by \( \alpha \), and the diamond is the reward.

When POMDPs are used by a decision maker interacting with another agent (possibly a human), then the state, \( S \), includes some descriptions of the behaviors of this other agent. That is, we factor the state, \( S \), into two parts, \( S = \{C, D\} \), as shown in Fig. 1b. Now, \( D \) is a high-level description of the observed behaviors of other agent(s), while \( C \) contains the remainder of the state, including observable actions of the other agent(s). In order to focus on learning models of behaviors, we assume that only \( D \) in this model is unobservable directly, while \( C \) and \( A \) are always fully observed. The transition function is also factorized in two parts: \( \Theta_C = P(C_0|C_{t-1}, D_{t-1}, A_t) \) gives the state dynamics given the observed behaviors and the action taken, while \( \Theta_D = P(D_t|D_{t-1}, C_t, A_t) \) gives the expected behaviors given the state and action. We assume that \( C \) and \( D \) are discrete random variables, so the associated parameters \( \Theta_C \) and \( \Theta_D \) are multinomial distributions. We denote the complete set of parameters \( \Theta = \{\Theta_C, \Theta_D, \Theta_O\} \). We also include fixed prior hyperparameters in Fig. 1: \( \alpha_C, \alpha_D \) are Dirichlet prior parameters, while \( \alpha_O \) is a more complex prior explored in detail in Section 4.

The observations at time step \( t \), \( O_t \), are a sequence of \( T \) observations (e.g., video frames), \( o_1 \ldots o_T \). In domains with human behaviors as observations, the rate at which decisions are made at the high level is slower than the rate of
observations (the video frame rate) and, therefore, $T_i \geq 1$. This difference in temporal scales between decision making and observations means that the observation function must be capable of spatio-temporally abstracting a video stream into a set of high level descriptors of behaviors, $D$. That is, it must be capable of computing $P(O_t|D_i)$ and of computing the gradient of this function. We describe one particular such function in Section 4. We assume throughout this paper that the boundaries of the observation sequences will be given by the changes in the fully observable context state, $C$ and $A$. There are other approaches to the temporal segmentation problem, ranging from the complete Bayesian solution in which the segmentation is parameterized [22] to specification of a fixed segmentation time [40].

A complete set of POMDP parameters, $\theta$, allows us to compute the likelihood of a sequence of data, $O = \{o, c, a\}$, $P(o, c, a; \theta)$, by summing over the unobserved values of $d$ using the standard forward algorithm for input-output hidden Markov models [5], [39]. Denoting $T$ as the number of POMDP transitions in the sequence, then $a = a_1, \ldots, a_T$ and $c = c_1, \ldots, c_T$ are the inputs, $O = o_1, \ldots, o_T$ (where each $o_i$ is a sequence of video frames) are the outputs.

### 3.2 Learning POMDP Parameters

This section describes how to learn the parameters of a POMDP, which can be formulated as a constrained optimization of the likelihood of the observations, given the model, over the (constrained) model parameters. We will show how the expectation-maximization, or EM, algorithm can be used to find a locally optimal solution [19]. Learning takes place over the entire model simultaneously: Both the output distributions, $\Theta_D$, and the high-level POMDP transition functions, $\Theta_C, \Theta_P$, are learned from data during the process. Since the behaviors, $D$, are unobservable, this means that the learning will perform both clustering of the observation sequences into a set of behavior descriptors, $D$, and learning of the relationship between these behavior descriptors, the observable context, $C$, and the action, $A$. In this section, we discuss the general learning method. Section 4.4 shows how to learn the parameters of a hierarchical observation function.

Learning the POMDP parameters is to find the set of parameters, $\Theta = \theta^*$ that maximize the posterior density of all observations and the model, $P(o, c, a; \Theta)$, subject to constraints on the parameters. The EM algorithm starts from an estimate of the parameter values, $\theta^*$, and computes

$$\theta^* = \arg \max_{\theta} \left\{ \sum_{D=d} P(d|o, c, a, \theta)^* \log P(d, o, c, a|\theta) + \log P(\theta) \right\}$$

The “E” step of the EM algorithm is to compute expectation over the hidden state, $P(d|o, c, a, \theta^*)$, given the observations $o, c, a$ and a current guess of the parameter values, $\theta^*$. The “M” step is then to perform the maximization which, in this case, can be computed analytically by taking derivatives with respect to each parameter, setting to zero, and solving for the parameter. The resulting update equations for the parameters of the POMDP transition functions are the same as for an input-output hidden Markov model [5].

The update equation for the $D$ transition parameter, $\theta_{D_{i,j;k,l}} = P(d_{i,j}|d_{i-1,j}, c_{i,k}, a_{i,l})$, is then

$$\theta_{D_{i,j;k,l}} = \frac{\alpha_{D_{i,j;k,l}} + \sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}{\sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}$$

where the sum over the temporal sequence is only over time steps in which the observed values of $o_t$ and $a_t$ in the data $d_t$ are $k$ and $l$, respectively, and $\alpha_{D_{i,j;k,l}}$ is the parameter of the Dirichlet smoothing prior. The summand can be factored as

$$P(d_{i,j}, d_{i-1,j}|o, c, a, \theta) = \beta_{i,j} P(c_{i}|d_{i-1,j}, c_{i-1}, a_i)$$

Learning the $POMDP$ parameters is to find the set of $\beta_{i,j}$, $P(c_{i}|d_{i-1,j}, c_{i-1}, a_i)$, $P(o_t|d_{i,j})$, which the segmentation is parameterized [22] to specification.

The resulting update equations for the parameters of the POMDP transition functions are the same as for an input-output hidden Markov model [5].

The update for the $D$ transition parameter, $\theta_{D_{i,j;k,l}} = P(d_{i,j}|d_{i-1,j}, c_{i,k}, a_{i,l})$, is then

$$\theta_{D_{i,j;k,l}} = \frac{\alpha_{D_{i,j;k,l}} + \sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}{\sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}$$

where the sum over the temporal sequence is only over time steps in which the observed values of $o_t$ and $a_t$ in the data $d_t$ are $k$ and $l$, respectively, and $\alpha_{D_{i,j;k,l}}$ is the parameter of the Dirichlet smoothing prior. The summand can be factored as

$$P(d_{i,j}, d_{i-1,j}|o, c, a, \theta) = \beta_{i,j} P(c_{i}|d_{i-1,j}, c_{i-1}, a_i)$$

Learning the $POMDP$ parameters is to find the set of $\beta_{i,j}$, $P(c_{i}|d_{i-1,j}, c_{i-1}, a_i)$, $P(o_t|d_{i,j})$, which the segmentation is parameterized [22] to specification.

The resulting update equations for the parameters of the POMDP transition functions are the same as for an input-output hidden Markov model [5].

The update for the $D$ transition parameter, $\theta_{D_{i,j;k,l}} = P(d_{i,j}|d_{i-1,j}, c_{i,k}, a_{i,l})$, is then

$$\theta_{D_{i,j;k,l}} = \frac{\alpha_{D_{i,j;k,l}} + \sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}{\sum_{t \in \{1, \ldots, N\}} P(d_{t+1,j}|o_{t+1}, c_{t+1}, a_{t+1})}$$

where the sum over the temporal sequence is only over time steps in which the observed values of $o_t$ and $a_t$ in the data $d_t$ are $k$ and $l$, respectively, and $\alpha_{D_{i,j;k,l}}$ is the parameter of the Dirichlet smoothing prior. The summand can be factored as

$$P(d_{i,j}, d_{i-1,j}|o, c, a, \theta) = \beta_{i,j} P(c_{i}|d_{i-1,j}, c_{i-1}, a_i)$$
useful to make. The ability of the value function to provide this information is intimately connected with its final ability to choose correct actions. However, we can use the simplest possible approximation for a particular domain that still yields sufficient information to guide our learning process. In the domains we have investigated, we can consider the POMDP as a fully observable MDP (the MDP approximation). The state, $S_t$, is assigned its most likely value in the belief state, $s^* = \arg \max_b b(s)$. Solving the resulting fully observable MDP using dynamic programming consists of computing value functions, $V^n$, where $V^n(s)$ gives the expected value of being in state $s$ with a horizon of $n$ stages to go, assuming optimal actions at each step. The actions that maximize $V^n$ are the policy with $n$ stages to go. The value functions are computed by setting $V^0 = R$ (the reward function), and iterating [4]

$$V^{n+1}(s) = R(s) + \max_{a,n} \left\{ \sum_{s' \in S} Pr(s'|a,s) \cdot V^n(s') \right\}, \quad (3)$$

where $Pr(s'|a,s)$ is the transition function from $s$ to $s'$ on action $a$. The actions that maximize (3) form the optimal (with regard to the MDP) $n$-stage-to-go policy, $\pi^n(s)$. We will only consider finite horizon policies in this paper. We use the SPUDD MDP solver, which exploits the structure inherent in a factored representation for efficient solutions [31].

We again stress the fact that, although we are solving the model using a strong approximation, the full POMDP model is still being learned and the solution is only being used in this work to guide our structure learning. For the domains we consider in this paper, this approximation still preserves the information about the state space distinctions necessary to learn the smallest model. It is also important to note that the solution technique we describe here could be replaced with other approximate model-based POMDP solution algorithms. The resulting changes to the value-directed structure learning technique would not alter the basic premise: that state space distinctions which are not useful will be apparent in any reasonable approximate solution.

### 3.4 Value-Directed Structure Learning

The value function, $V(s)$, gives the expected value for the decision maker in each state. However, there may be parts of the state space that are indistinguishable (or nearly so) insofar as decisions go. Eliminating the distinctions between them by merging states can lead to efficiency gains without compromising decision quality. Such state merging is a form of structure learning (for model order) based upon the utility of states. While this idea has been explored in the machine learning literature [15], we apply it here to the task of learning the minimum number of behaviors that need to be distinguished in our learned POMDP. This value-directed structure learning can be contrasted with data-dependent structure learning, in which the complexity of the model is traded off against the quality of fit to the data. For example, Bayesian inference gives rise to a manifestation of Occam’s razor by assigning higher probability to simpler models in many cases. However, this only considers utility based on data prediction. In our case, we explicitly look for models that are the simplest for achieving value within a task and these may not be the same as those given by Bayesian model selection.

There are two problems that must be solved: first, finding the parts of the state space that can be merged and, second, actually performing the merging, which involves combining the (learned) observation functions for different behaviors. We address each of these in turn. To merge states, we use the fact that parts of the state space with similar values in the value function can be aggregated to form abstract states when computing a policy without sacrificing much in terms of value. We can also look at the policy for a given model and find states that map to the same action. This process is repeated until no more merges are possible or until the number of behaviors becomes 1 (in which case, recognition of the behaviors is deemed useless).

To see how this is implemented, recall that the state space is represented in a factored POMDP as a product over a set of variables. Therefore, the value function can be split into $N_d$ pieces, $V_i$, one for each value (behavior class) of the variable $D = d_i$. Each such $V_i$ gives the values of being in any state in which $D = d_i$. A similar split occurs for the policy, yielding subpolicies, $\pi_i$, giving the actions to take for each $D = d_i$. The $V_i$ can be compared by computing the difference between them, $d_{ij} = \|V_i - V_j\|$, where $\|X\| = \max\{x : x \in X\}$ is the supremum norm. This value difference is an indication of how much value will potentially be lost if we merge the states $i$ and $j$. Two subpolicies, $\pi_i$ and $\pi_j$, are compared by dividing the number of states for which they agree, $n_{ij} = |\pi_i \cap \pi_j|$, by the size of the state space spanned by all variables except $D$, written $|S_{-D}|$. This fraction, $p_{ij} = n_{ij}/|S_{-D}|$, is the amount of the state space over which the policies agree. The algorithm shown in Fig. 2 uses these measures, beginning with $N_d$ as large as the training data will support. The thresholds $\omega_p$ and $\omega_d$ govern how aggressively this method merges states. These parameters are not learned and must be tuned manually. We discuss particular values for these thresholds in Section 5.

Once two states have been found, we must initialize a new POMDP model with one state representing the observation space that both original states did (Step 6). We have used two methods for doing this. In the first, we simply delete one of the two states. We use this method in Sections 5.2 and 5.3.2. This may be dangerous, however, as a large part of the observation space may not be modeled and the next iteration of learning (Step 1) may account for it in unpredictable ways. A more principled method is to reinitialize an observation function $P(O | d_i)$, where $d_i$ is the new merged state based on all the data that was classified into states with $D = d_i \vee d_j$ (the old states to be merged). We use this method in Section 5.3.1.

![Fig. 2. Value-directed structure learning algorithm.](image)

```plaintext
initialise $N_d$ as large as possible
repeat
1. learn the POMDP model
2. compute $V_i$ and $\pi_i \forall i$
3. compute $d_{ij}$ and $p_{ij} \forall (i,j), i \neq j$
4. $\{i,j\} = \arg \min_{\{i,j\}} (d_{ij} V_i(\mathbf{k}, \mathbf{l}) \mid p_{ij} > \omega_p)$
5. if $d_{ij} < \omega_d \land N_d > 1$
6. merge states $i$ and $j$
7. $N_d \leftarrow N_d - 1$
end
until $N_d$ stops changing
```
In what follows, we describe a model for applied to different types of behaviors without modification. Finally, we wish to sample from this distribution to find POMDP solutions [29], we require a generative model. Further, since we will eventually describe instantaneous dynamics and configuration of the face to a smile. Section 5.1 shows how configuration and dynamics together outperform either one separately in a recognition task.

We assume that a region of interest in each frame is selected by some independent tracking process (details in [26]). The observations are video image regions, \( f_t \), and the spatio-temporal derivatives, \( \nabla f_t \), between pairs of images over these regions. The observations are spatially abstracted using projections, \( Z \), to an an a priori basis of feature-weighted 2D polynomials, as shown in Fig. 3b. The basis functions are fixed in order to ensure the model is generalizable to different behaviors, but we show in Section 4.3 how feature weights can be learned such that only those basis functions that are necessary are used.

As a generative model, it can be described as follows: A high-level behavior at time \( t \), \( d_t \), generates a sequence of values, \( w_{t-1} \ldots w_{T-1} \), and \( x_{t-1} \ldots x_{T-1} \), for the discrete configuration and dynamics variables, \( W \) and \( X \), respectively. The dynamics in the hidden chains are given by the transition functions \( \Theta_X = P(X_t | X_{t-1}, W_{t-1}, D_t) \) and \( \Theta_W = P(W_t | W_{t-1}, X_{t-1}, D_t) \) and the initialization functions \( \Pi_X = P(X_0 | W_{0-1}, D_0) \) and \( \Pi_W = P(W_0 | D_0) \). The dynamics and configuration variables at time \( t' \), \( X_{t'}, W_{t'} \), generate an image, \( f_{t'} \) and a spatio-temporal derivative, \( \nabla f_{t'} \), through the conditional distributions, \( \Theta_{t'} = P(f_{t'} | W_{t'}) \) and \( \Theta_{t'} = P(\nabla f_{t'} | X_{t'}) \). We discuss these last two functions in more detail in Sections 4.1 and 4.2, respectively.

This mixture model can be used to compute the likelihood of a video sequence given the display descriptor, \( P(o_1 \ldots o_T | d_t) \), where \( o_t \equiv \{ f_t, \nabla f_t \} \), using the recursive equations:

\[
P(o_1 \ldots o_T | d_t) = \sum_{ij} P(\nabla f_i | x_{i,1}) P(f_i | w_{i,j}) P(x_{i,1} | x_{i-1,j}, w_{i,j}, d_T) \\
\sum_{kl} P(w_{l,j} | w_{l-1,k}, x_{t-1,j}, d_T) P(x_{l-1,k} | w_{t-1,j} o_1 \ldots o_{T-1} | d_t) \\
p(o_1 | d_t) = \sum_{ij} P(\nabla f_i | x_{i,1}) P(f_i | w_{i,j}) P(x_{i,1} | w_{i,j}, d_t) P(w_{1,j} | d_t).
\]

Table 1 shows the parameters in the model. On the left are parameters that have fixed values. These numbers are set manually based on prior expectations. The right table shows the parameters that are learned using EM, the initialization heuristics, or the value-directed learning methods.

The following sections give more details on the observation function. Sections 4.1 and 4.2 give overviews of the likelihood computations for dynamics, \( P(\nabla f_t | X_t) \), and configuration, \( P(f_t | W_t) \), respectively. Feature weighting is described in Section 4.3. Section 4.4 then discusses learning the observation function.

### 4.1 Dynamics

Fig. 3b shows an expanded version of the dynamics vertical chain from Fig. 3a. We wish to derive the likelihood of a derivative, \( \nabla f_t \), given the high-level dynamics class, \( X \). Since we wish to classify optical flow fields, we expand the likelihood as
The distribution over \( f \) given \( z_X \) is parameterized using a projection of \( v \) to the basis of Zernike polynomials, \( P(v|z_X) = \sum_{j} P(v|z_X)P(z_X|X) \), where \( Z_X \) is the feature vector in the polynomial basis space. We parameterize the distribution over \( Z_X \) given \( X \) with a normal, \( P(Z_X|X) = \mathcal{N}(Z_X; \mu_{z,X}, \Sigma_{z,X}) \). We are expecting flow fields to be normally distributed in the space of the basis function projections.

The distribution over \( v \) given \( z_X \) is given by projections to the basis of Zernike polynomials, which have useful properties for modeling flow fields [27] and images [53]. Our method is not restricted to this basis set, but gains independence from the data by using an a priori set of basis functions, leading to a more generally applicable observation function. Zernike polynomials are an orthogonal set of complex polynomials defined on a 2D elliptical region as follows [44]:

\[
\begin{align*}
A_n^m(x, y) & = \frac{1}{2\pi} \sum_{l=0}^{\infty} \frac{1}{2l+1} \left[ \frac{\Gamma(n+l)}{\Gamma(n+1)} \right] \cos((n-l)\phi) \\
B_n^m(x, y) & = \frac{1}{2\pi} \sum_{l=0}^{\infty} \frac{1}{2l+1} \left[ \frac{\Gamma(n+l)}{\Gamma(n+1)} \right] \sin((n-l)\phi),
\end{align*}
\]

where \( \phi = \arctan(y/x) \), \( \rho = \sqrt{x^2 + y^2} \leq 1 \), and \( x, y \) are measured from the region center. The lowest two orders of Zernike polynomials correspond to the standard affine basis, and higher orders (higher values of \( n \) and \( m \)) represent higher spatial frequencies. The basis is orthogonal such that each order can be used as an independent characterization of a 2D function and each such function has a unique decomposition in the basis. Define an \( N \times N \) matrix \( B \) whose columns are the \( N \) Zernike basis functions (with pixels arranged rowwise), alternating between \( A_n^m \) and \( B_n^m \) such that columns 0, 1, 2, 3... are Zernike polynomials \( A_0^0, A_1^1, B_1^0, A_2^2, \ldots \). Then, a 2D function, \( f \), with \( N \) pixels is projected to the basis using \( z = \epsilon f \) and can be reconstructed from the \( N \times 1 \) column vector of coefficients, \( z \), using \( f = Bz \). The normalization constant \( \epsilon = \epsilon_m(n+1)/\pi \), where \( \epsilon_m = 1 \) if \( m = 0 \) and \( \epsilon_m = 2 \) otherwise. Each of the vertical and horizontal components of a flow field can be written as a linear combination of the basis functions and, so, we can write \( v = Mz_X \), where

\[
v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \quad M = \begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix}, \quad z_X = \begin{bmatrix} z_x \\ z_y \end{bmatrix},
\]

in which \( z_x, z_y \) are the Zernike coefficients for horizontal and vertical flow, respectively. In practice, \( M \) will be some subset of the Zernike basis vectors, the remaining variance in the flow fields being attributed to zero-mean Gaussian noise. Thus, we write \( v = Mz_X + \eta_v \), where \( \eta_v \sim \mathcal{N}(0, \Sigma_v) \) and, so, \( P(v) = \mathcal{N}(v; Mz_X, \Sigma_v) \). The noise, \( \eta_v \), is a combination of three noise sources: the reconstruction error (energy in the higher order moments not in \( M \)), the geometric error (due to discretization of a circular region), and the numerical error (from discrete integration) [37]. The choice of a subset of basis elements to use will depend on what the projections are being used for (see Section 4.3).

Since all of the terms in the integration (5) are Gaussian distributions, we can integrate over \( v \) and \( z_X \) analytically by successively completing the squares in \( v \) and \( z_X \) to obtain

\[
P(f_z|Xf_z) = \frac{1}{\sqrt{\|A_z\|^2}} \exp \left[ -\frac{1}{\rho^2} (\mu_{z,x} - \mu_{z,x})^2 - \mu_{z,y}^2 - \mu_{z,y}^2 - \epsilon \right],
\]

where \( \epsilon \) is an approximation of the reconstruction error.

### Table 1: List of Fixed and Learned Parameters in the Model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Used for</th>
<th>Value</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1, \sigma_2 )</td>
<td>optical flow</td>
<td>0.08, 1.0</td>
<td></td>
</tr>
<tr>
<td>( \sigma_{p,x}, \sigma_{p,w} )</td>
<td>projection error</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>( N_{x,z}, N_{z,w} )</td>
<td>number of features for ( X, W )</td>
<td>16, 32</td>
<td></td>
</tr>
<tr>
<td>( \kappa_x, \alpha_x, \beta_w )</td>
<td>feature weights ( (X) )</td>
<td>( N_x + 2, 1, 0.01 )</td>
<td></td>
</tr>
<tr>
<td>( \kappa_w, \alpha_w, \beta_w )</td>
<td>feature weights ( (W) )</td>
<td>( N_x + 2, 1, 0.01 )</td>
<td></td>
</tr>
<tr>
<td>( \alpha_x, \alpha_w )</td>
<td>transition priors ( (X, W) )</td>
<td>1.25</td>
<td></td>
</tr>
<tr>
<td>( \alpha_x^<em>, \alpha_w^</em> )</td>
<td>initialisation priors ( (X, W) )</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>( \alpha_D, \alpha_C )</td>
<td>transition priors ( (D, C) )</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>( \alpha_D^<em>, \alpha_C^</em> )</td>
<td>initialisation priors ( (D, C) )</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>( N_a, N_c )</td>
<td>number of actions, states</td>
<td>( \infty, 0.9 )</td>
<td></td>
</tr>
<tr>
<td>( \omega_1, \omega_p )</td>
<td>value-directed thresholds</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
where
\[ \hat{A}_{z,x} = \left( A_{z,x}^{-1} + M' \left( A_p + (f_a A^-1 f_s)^{-1} \right)^{-1} M \right)^{-1}, \]
\[ \hat{\mu}_{z,x} = \hat{A}_{z,x} \left( A_{z,x}^{-1} \mu_{z,x} - M' \mu_p^{-1} A_w w \right), \]
\[ \mu_w = \left( f_a A^-1 f_s + A_p^{-1} \right)^{-1}, \]
\[ \epsilon = f_a A^-1 f_s + w' A_w w' \]

The brightness constancy assumption fails if the velocity \( v \) is large enough to produce aliasing. Therefore, a multiscale pyramid decomposition of the optical flow field must be used. This results in distribution over the flow vectors, \( P(v|\nabla f) \sim N(v; \mu_v, \Lambda_v) \), where \( \mu_v = (f_a A^-1 f_s)^{-1} \) and \( \mu_v = -A_p A^{-1} f_s \) [48]. Using these coarse-to-fine estimates, (9) become
\[ \hat{A}_{z,x} = \left( A_{z,x}^{-1} + M' \left( A_p + A_v \right)^{-1} M \right)^{-1}, \]
\[ \hat{\mu}_{z,x} = \hat{A}_{z,x} \left( A_{z,x}^{-1} \mu_{z,x} + M' \mu_p^{-1} \mu_v \right). \]

The mean of this distribution, \( \hat{\mu}_{z,x} \), is a weighted combination of the mean Zernike projection from the data (\( M' \mu_v \)) and the model mean, \( \mu_{z,x} \).

4.2 Configuration

The classification of image configurations proceeds analogously to the classification of temporal derivatives. We use the same set of basis functions, but the measurements are now the images, \( I \), the subspace projections onto image regions are \( \tilde{H} \) and the basis feature vectors are \( Z_{w} \). The distribution over \( Z_{w} \) given \( W \) is parameterized with a normal \( P(Z_{w}|W) = N(Z_{w}; \mu_{z,w}, \Lambda_{z,w}) \). The distribution over the image regions given the Zernike projection is normal, \( P(H|Z_{w}, \Theta) = N(H; BZ_{w}, \Lambda_{z,w}) \). The distribution over images given the subspace image region, \( h \), \( P(f|H, \Theta) \), can be approximated using a normal distribution at each pixel, \( P(f|H) \sim N(f; \tilde{H}, \Lambda_{z}) \).

The integrations give results similar to that for the dynamics (8):
\[ P(f|W, \Theta) \propto e^{\epsilon(\Lambda_{z}^{-1} \mu_{z,w} - e^{-1} \mu_{z,0} \Lambda_{z}^{-1} \mu_{z,0})}, \]

where
\[ \mu_{z,w} = A_{z}^{-1} \left[ A_{z}^{-1} \mu_{z,w} + \Lambda_{z}^{-1} f_s \right], \]
\[ \Lambda_{z,w} = \left( B' \Lambda_{z}^{-1} B - B' A_{z}^{-1} \left( \Lambda_{z}^{-1} + \Lambda_{z}^{-1} \right) - \Lambda_{z}^{-1} \right)^{-1}. \]

In this case, however, there are no data-dependent variances and we approximate \( P(f|W, \Theta) \approx P(z_{w}|W, \Theta) \), where \( z_{w} = B f \) is the projection of the image region to the basis.

4.3 Feature Weighting

In general, we will not know which basis coefficients are the most useful for our classification task: which basis vectors should be included in \( M \) and should be left out (as part of \( \epsilon \)). We use the feature weighting techniques of [13] that characterize the relevance of basis vectors by examining how the cluster means, \( \mu_{z,x} \), are distributed along each basis dimension, \( k = 1 \ldots N_{z} \). Relevant dimensions will have well-separated means (large interclass distance along that dimension), while irrelevant dimensions will have means that are all similar to the mean of the data, \( \mu' \).

To implement these notions, we place a conjugate normal prior on the cluster means, \( \mu' \sim N(\mu', T) \), where \( T \) is diagonal with elements \( \tau_{z}^{2} \ldots \tau_{z}^{2} \), and \( \tau_{z}^{2} \) is the feature weight for dimension \( k \). The prior biases the model means to be close to the data mean along dimensions with small feature weights (small variance of the means), but allows them to be far from the data mean along dimensions with large feature weights (large variance of the means). Thus, \( \tau_{z}^{2} \) will be large if \( k \) is a dimension relevant to clustering, while \( \tau_{z}^{2} \rightarrow 0 \) if the dimension is irrelevant. Feature selection occurs if we allow \( \tau_{z}^{2} = 0 \) for some \( k \). We do not select features in this work.

Conjugate priors are placed on the feature weights, \( \tau_{z}^{2} \), and on the model covariances, \( \Lambda \). Each feature weight is univariate and, so, an inverse gamma distribution is the prior on each \( \tau_{z}^{2} \):
\[ P(\tau_{z}^{2}|a, b) \propto \frac{1}{\tau_{z}^{a-1} e^{-b/\tau_{z}^{2}}} \]

This prior allows some control over the magnitude of the learned feature weights, \( \tau_{z}^{2} \). The model covariances are multivariate, for which the conjugate prior is an inverse-Wishart prior:
\[ P(\Lambda_{z}|\alpha, \Lambda^{*}) \propto |\Lambda_{z}|^{-(\alpha + N_{z} + 1)/2} e^{-\frac{1}{2} tr(\Lambda^{*} \Lambda_{z}^{-1})} \]

where \( \Lambda^{*} \) is the covariance of all the data and \( \kappa \) is a parameter that dictates the expected size of the clusters (the intraclass distance). This prior stabilizes the cluster learning.

4.4 Learning the Observation Function

Recall from Section 3.2 that the updates (M step) to the \( j \)-th component of the output distribution, \( P(O|D) \), will be weighted by \( P(d_{ij}|o, a, c) = \alpha_{ij} \beta_{ij} \), where \( \alpha \) and \( \beta \) are the forward and backward variables ((1) and (2)), respectively, that require the computation of \( P(o|d_{ij}) \) as given by (4). This means that we can use the standard equations for updating the transition functions in the \( X \) and \( W \) chains as would be used for a normal CHMM, except the evidence from the data is weighted by \( P(d_{ij}|o, a, c) \).

The updates to the output distributions in the configuration process, \( P(f|W) \), and in the dynamics process, \( P(\nabla f|X) \), are as they would be in a mixture model, except that the feature weights bias the updates toward the prior distributions, and the prior weights are given by the state likelihoods in the POMDP (high-level) chain. The update equation for the mean of the \( j \)-th Gaussian output distribution in the \( j \)-th model (a component of \( P(o|x, d_{ij}) \), \( \mu_{z,ij} \), is
\[ \mu_{z,ij} = (\xi_{ij} A_{z,ij} + T_{ij}^{-1})^{-1} \left[ A_{z,ij}^{-1} \mu_{z,ij} + T_{ij}^{-1} \right], \]

where \( \xi_{ij} = \sum_{t=1}^{T} \xi_{t,ij} \), \( \mu_{z,ij} \) is given by (10) using the parameters of the \( j \)-th Gaussian output in the \( j \)-th model, \( \mu_{z,ij}, A_{z,ij} \), and
\[ \xi_{t,ij} = P(x_{t}, d_{ej}|o, a, c') \]

The first term is given by the the usual forward-backward equations in a CHMM [11], while the second is given by the forward-backward equations in the high-level POMDP ((1) and (2)). Thus, the most likely mean for each state \( x \) is the weighted sum of the most likely values of \( z \) as given by (10).
Dimensions of the means, $\mu_{z,j}$, with small feature weights, $\tau_{k}^2$, will be biased toward the data mean, $\mu^*$, in that dimension. This is reasonable because such dimensions are not relevant for clustering and, so, should be the same for any cluster, $X$.

The updates to the feature weights are

$$\tau_{k}^2 = \frac{b}{a + N_{k}/2 + 1} \frac{1}{2a + N_{k} + 2} \sum_{i=1}^{N} (\mu_{z,i,k} - \mu_{k})^2$$

and show that those dimensions, $k$, with $\mu_{z,i,k}$ very different from the data mean, $\mu^*$, across all states, will receive large values of $\tau_{k}^2$, while those with $\mu_{z,i,k} \sim \mu^*$ will receive small values of $\tau_{k}^2$. Intuitively, the dimensions along which the data is well separated (large interclass distance) will be weighted more. The complete derivation, along with the updates to the output distributions of the CHMMs, including to the feature weights, can be found in [26].

Model initialization proceeds in a bottom-up fashion. First, the dynamics mixture model with $N_{d}$ classes is initialized from a set of single (time-independent) spatio-temporal derivative fields, $\nabla f$, by first computing the expected most likely values of $z_{i}$ for each frame using a single zero-mean model with constant diagonal covariance 0.001 and then fitting a Gaussian mixture to the result of K-means clustering with $K = N_{d}$ [28]. While the K-means algorithm uses the Euclidean distance in the space of $Z$, the Gaussian fits use the Mahalanobis distance. All of the feature weights, $\tau_{k}^2$, are initialized to 1 and state assignment probability $\Theta_{k}$ is initialized evenly. The mixture model over the configurations is initialized in a similar way, using the projections of image regions to the Zernike basis.

The entire model is initialized using the estimates of dynamics and configuration mixtures by first classifying all of the data using the two mixture models and finding the largest $N_{d}$ sets of sequences whose sets of visited $X$ states match exactly. Second, find the set of $X (W)$ states visited by all the sequences in set $i: N_{X}(N_{d})$. Third, initialize a CHMM for each set, $i$, by assigning the output distributions to be those in the simple mixture models visited by the sequences in the cluster. The transition and initial state probabilities are initialized randomly. Finally, training each CHMM, keeping the output distributions fixed and initializing the feature weights for the mixture of CHMMs evenly for each context state.

### 4.5 Complexity

The complexity of parameter learning is dominated by the computation of (10) in the “E” step, which is $O(N_{d}[N_{d}N_{d} + N_{d}^{2}T^{"}})$, where $N_{d}$ is the number of high-level display states, $T^{"}$ is the length of the entire sequence of data, $N_{d}$ is the maximum number of pixels in the region being tracked, and $N_{d}$ is the maximum dimensionality of the feature vectors $Z_{z}, Z_{w}$. The computation is repeated until EM converges, usually some small number of iterations $n < 10$.

The worst-case complexity of solving the POMDP using value iteration over the associated MDP is $O(N_{d}^{2}N_{d}H)$, where $N_{d}$ is the number of states in the POMDP, $N_{d}$ is the number of actions, and $H$ is the horizon. In typical problems, $N_{d}$ is very large (exponential in the number of variables in the POMDP) and the complexity of the entire system, learning plus solution, will be dominated by the solution term. In the experiment described in Section 5.3, the number of features is $N_{d} = 36$ (the major factor in the learning complexity), but the number of states is $N_{d} > 6 \times 10^{3}$. Note, however, that we use a structured approach to solving this MDP [31], which can substantially reduce the solution average case complexity.

Once a policy has been found, the complexity of using the model online is composed of updating the belief state ($O(N_{d}N_{d}^{2})$) and consulting the policy ($O(N_{d}N_{d})$). The belief state inference will dominate for any reasonably sized model, but techniques that leverage structure can substantially reduce the running time so as to make this possible in near-real-time. The other major computation is optical flow, which can be done in near-real-time as well.

### 5 Experiments

We present three sets of experiments in this section, each designed to address a particular issue in the learning method we have presented. In the first (imitation game), we explore the representational power of our computer vision modeling techniques by examining how they can be used to learn fairly complex facial expressions. The second experiment demonstrates the value-directed learning technique using a simple set of hand gesture sequences. The third experiment then shows how the model can be used to learn a more complex interaction during a card matching game. We demonstrate on both synthetic and real data in this third experiment.

#### 5.1 Imitation Game

In this experiment, human subjects imitated the facial expressions of an animated character. We learn a mixture of CHMMs model of their facial expressions and use this model to predict the animation that caused it. To play the imitation game, a player watches a computer animated face on a screen and is told to imitate the actions of the face. The animations start from a neutral face ($n$) and warp to one of four poses $(a_{1}, a_{2}, a_{3}, a_{4})$, as shown in Fig. 4. The pose is held for one second and the face then warps back to neutral, where it remains for another second.

The simplified model is shown in Fig. 4b. The cartoon facial expressions are $A = \{a_1, a_2, a_3, a_4\}$, the observations of the human’s actions, $O$, are sequences of video images and the spatio-temporal derivatives between subsequent video frames and the descriptor of the human’s facial expression is $D \in \{d_1, \ldots, d_{6}\}$. The key here is the ability to learn both the output distribution $P(O|D)$ and the relationship between the player’s behavior, $D$, and the cartoon display, $A$, $P(D|A)$. Thus, we train the model on a set of training data in which the cartoon display labels are observed, then we attempt to predict the cartoon facial expression on a set of test data in which the labels are hidden. That is, on the test data, we compute $a^* = \arg \max_{a} P(A = a | o) = \arg \max_{a} \sum_{d} P(o|d) P(d|A = a)$.

Three subjects performed the task 40 times each. Video frames were recorded at $160 \times 120$ with a Sony EVI-D30 color camera (framerate 28 fps) mounted above the screen. The subjects’ faces were located in each frame using an optical flow and exemplar-based tracker [26]. The videos were temporally segmented using the onset times of the cartoon facial expressions and the resulting sequences were input to the mixture of coupled HMM clustering and training algorithm using four clusters ($N_{d} = 4$, the number of expressions the subjects were trying to imitate).

We did a leave-one-out cross validation experiment for each of the subjects to verify the prediction accuracy. There were 40 sequences for each subject, one of which was
removed. The remaining 39 sequences were used to train the POMDP. The learned model was used to infer $A$ from the
remaining sequence (in which it is hidden) and the most likely value was compared to the actual display. This process was
repeated for all 40 left-out sequences, giving the best unbiased estimated of performance on the whole set of data. This
process was repeated three times for each subject with different random initializations, with average success rates of
78, 74, and 62 percent. However, these results ignore the
model’s explicit representation of uncertainty, only reporting
success if the actual display is the peak of $P(A|O)$. In some
cases, there is a second display that is nearly as likely as the
best one. The success rates rise to 95, 93, and 84 percent if we
count those classifications as correct where the probability of
the most likely display is less than 0.5 and the second most
likely display is the correct one.

We also used this game to evaluate the modeling of both
the dynamics and the configuration in the observation
CHMMs. We performed the same leave-one-out experiment
on the first subject’s data with 10 random initializations, and
for each seed, we trained one model with only dynamics ($X$)
states, one with only configuration ($W$) states, and one with
both. Results showed that modeling both outperformed
either separately: 76 percent for both compared to 68 percent
for dynamics alone and 68 percent for configuration alone.

We now show some details of the learned $D = 1$
(“smiling”) model for one subject. Feature weights are shown
in Fig. 5. The dynamics chain has four significant features: two
in the horizontal flow, $A_{1}, B_{3}$, and three in the vertical flow,$A_{2}, B_{1}$, and $A_{2}$. The three most significant features in
the configuration chain are $B_{1}, B_{3}$, and $A_{1}$. The output distributions of the four dynamics states ($X$) are shown in Fig. 6a,
plotted along the two most significant feature dimensions,$A_{1}$ and $B_{1}$. Two states ($X = 2, 4$) correspond to no motion,
while the other two correspond to expansion upward and
outward in the bottom of the face region ($X = 1$) and
contraction downward and inward in the bottom of the face
region ($X = 3$). These states correspond to the expansion and
relaxation phase of smiling. The output distributions of the
four configuration states are shown in Fig. 6b. There are two
states ($W = 2$ and $W = 4$) which describe the face in a fairly
relaxed pose, while $W = 1$ and $W = 3$ describe “smiling”
configurations, as evidenced by the darker patches near the
sides at the bottom.

Fig. 7 shows the model’s explanation of a sequence in
which $D = 1$. We see the high level distribution over $D$ is
peaked at $D = 1$. Distributions over dynamics and config-
uration chains show which state is most likely at each frame.
The expected pose, $H$, and flow field, $V$, are shown
conditioning the image, $f$, and the temporal derivative, $f_t$,
respectively.

### 5.2 Robot Control Gestures

This “game” involves a human operator issuing navigation
commands to a robot using hand gestures. The robot has
four possible actions, $A$: go left, go right, stop, and go forward
and the operator uses four hand gestures corresponding to
each command. The state, $C$, is the operator’s action: A
Boolean indicator of whether the robot took the right action
or not. The reward function is 1 if the robot took the correct
action ($C = 1$); otherwise, it is 0. It is important to state that
these experiments do not demonstrate gesture recognition,
but only the value-directed learning. Realistic robotic
control requires more complex tracking to deal with
moving platforms and the high variability in gestures.

We recorded 12 examples of each of four gestures
performed by a single subject in front of a stationary camera.
Video was grabbed from a IEEE 1394 (Firewire) camera at
$150 \times 150$. The region of interest was the entire image, so
no tracking was required. Sequences were taken of a fixed length
of 90 frames. From the original 12 data sequences, we selected...
and constructed a training set in which each $A$ was tried for each possible gesture sequence (and the resulting $C$ response was simulated), giving a total of 44 training sequences. We used an initial $N_d = 6$ states, which is as many as we can expect to learn given the amount of training data. The value-directed structure learning algorithm used $w_p = 0.9$ and $w_d = \infty$.

Once the model is trained on the 44 training sequences, we evaluate how well it chooses an action on the four remaining sequences (one for each gesture). This leave-one-out cross-validation is repeated for 12 different sets of four test sequences and the total rewards gathered give an unbiased indication of how well the model performs on unseen data. The model chose the correct action 47 out of a total of 12 \times 4 = 48 times, for a total success rate of 47/48 or 98 percent. The one failure was due to a misclassification of a “left” gesture as a “right” gesture due to a large rightward motion of the hand at the beginning of the stroke. More importantly, the final POMDP models learned that there were $N_a = 4$ states in all 12 cases.

5.3 Card Matching Game

The card matching game is a simple cooperative two-player game in which the players must send signals to each other through a video link in order to win. At the start of a round, each player is dealt three cards: a heart, a diamond, and a spade. Each player can only see his own set of cards. The players must each play a single card simultaneously and, if the suits match, the players win a function of the amount on the cards; otherwise, they incur a fixed penalty. Thus, the goal of the game is to agree on which suit to play to maximize the return. In order to facilitate this, one player (the bidder) can send a bid to their partner (the ally), indicating a card suit, and can see (but not hear) the ally
HOEY AND LITTLE: VALUE-DIRECTED HUMAN BEHAVIOR ANALYSIS FROM VIDEO USING PARTIALLY OBSERVABLE MARKOV DECISION... 1129

Fig. 8. Average reward gathered over 20 trials in simulation, for simulated models with (top row) \( N_d = 3 \) and (bottom row) \( N_d = 5 \) (three and five behaviors to be recognized, respectively), shown as a function of the number of training sequences. The three plots in each row show the results for \( R_f = 100, 20, \) and \( 10 \), from left to right. The dashed line in each plot shows the mean reward achieved when the model is given as the simulated model and, so, is optimal for the given simulation. The values of \( N_d \) shown in the plot are the averages learned by the algorithm for \( N_t = 200 \).

through a real-time video link. Thus, the expectation is that the ally will develop a gesturing strategy to indicate agreement with the bid. For example, a simple strategy involving two head gestures is for the ally to nod or shake her head in agreement or disagreement with the bid. A more complex strategy is to develop a set of three head or hand gestures for each card suit. This game is similar to games used in psychology to study the emergence of language [25], [34]. We use it because the number and form of the gestures are not specified as rules of the game, but instead are decided upon by the players at game time.

There are nine variables that describe the state of the game. The suit of each of the three cards can be one of \( \heartsuit, \diamondsuit, \clubsuit \). The bidder’s actions, \( A \), can be null (no action) or sending a confidential bid (\( bid_\heartsuit, bid_\diamondsuit, bid_\clubsuit \)) or committing a card (\( comm_\heartsuit, comm_\diamondsuit, comm_\clubsuit \)). The ally’s action, \( C \), can be null or committing a card (\( comm_\heartsuit, comm_\diamondsuit, comm_\clubsuit \)). The behavior variable, \( D = d_1, \ldots, d_{N_d} \), describes the ally’s communication through the video link. The other six observable variables in the game are more functional for the POMDP, including the values of the cards (\( v1, v2 \) for bidder and ally, respectively) and whether a match occurred or not. The reward is a function of fully observable variables only and is \( [v1 + v2][1 + (v1 > 1 \land v2 > 1)] \) if the suits match and \(-10\) otherwise. The number of display states, \( N_d \), is learned using the value-directed structure learning technique in Fig. 2. The number of states in the MDP is 20, 736 \( N_d \).

5.3.1 Simulated Results

Since our model is generative, we can use it to simulate the game and to generate data. We first specified two prior models for the card matching game: The first, \( M_1 \), has \( N_d = 3 \), while the second, \( M_2 \), has \( N_d = 5 \). We randomly specify a complete set of output distribution parameters, \( P(O | D) \), as follows: Transition matrices were drawn randomly in \([0, 1]\) but with added weight of 30.0 on the diagonal (followed by a renormalization). Feature weights for all output distributions were drawn randomly in \([0, R_f]\). The means were drawn from the model priors given these feature weights. Covariance matrices were set to be diagonal with a variance in each dimension of \((10.0 + c)^2\), where \( c \) is normally distributed with variance 1.0. We used \( N_{s,z} = N_{s,w} = 4 \). The parameter \( R_f \) varies the “overlap” between output distributions. The larger \( R_f \) is, the more distinguishable the behavior models will be. We can estimate the degree of “overlap” of the output distributions by using them to simulate data and then measuring the maximum-likelihood error rates, \( R_f \), on this data given the models. We used three values of \( R_f = \{10, 20, 100\} \), which gave error rates on 50,000 simulated sequences of 1.6, 0.02, and 0.0 percent, respectively, for model \( M_1 \) and 3.6, 0.04, and 0.0 percent, respectively, for model \( M_2 \).

Once the model was specified, we simulated a set of \( N_t \) training sequences from it, each of length \( T = 100 \), with output sequences of a fixed length of \( T = 20 \), using a random selection of actions, \( A \). We only simulate the values of \( Z_W, Z_X \), not the full observation set, \( f, \nabla f \), since this is sufficient to demonstrate the method. We then applied the training process in Fig. 2 on these \( N_t \) training sequences, using \( w_p = 0.6 \) and \( w_d = 20.0 \), and starting with \( N_d = 8 \). The learned model and policy was tested for 20 trials of length 100 and we record the average reward gathered over the 20 trials. The whole process is repeated (including random specification of new output distributions) 20 times and the means and standard deviations of the averages are recorded. We also estimate the optimal values that can be achieved by performing the same simulation experiment, but using the original simulated model instead of a learned model.

Fig. 8 shows the results. Each plot shows the average reward and standard deviation for each value of \( N_t \) as well as the optimal value (dashed line), the average number of behavior models (for \( N_t = 200 \)), \( N_d \), and the degree of overlap, \( R_f \). The average number of states of \( D \) learned for \( N_t = 200, R_f = 100 \) was 4.0 and 6.1 for \( N_d = 3 \) and \( N_d = 5 \), respectively, showing that, although the state merging was not overly aggressive, it managed to reduce the model order close to that of the true models in both cases. The results show that, if the actual output models are distinguishable
(Rf = 100), then the resulting learned model is nearly optimal for Nd ≥ 50, even using the approximate solution technique for the POMDP policy. This is what we expect to happen if the learning technique is able to recover a set of behaviors that distinguish at least those behaviors that are important to the task, since then the state is (nearly) fully observable and the MDP policy is (nearly) optimal for the POMDP. On the other hand, when the output models become less easily distinguishable (for Rf = 20 and 10), the learning is more difficult and, so, the gap between the optimal solution and the learned one widens. Nevertheless, the performance remains close to optimal in all but the most difficult case (with Nd = 5 and Rf = 10).

5.3.2 Real Data

The card matching game was played by two users through a computer interface in our laboratory. Each player viewed their partner through a link from their workstation to a Sony EVI S-video camera mounted atop their partner’s screen. The average frame rate at 320 × 240 resolution was over 28 fps. The rules were explained to the subjects and they played four games of five rounds each. The players had no chance to discuss strategies before the game, but were given time to practice. The player’s faces were tracked using the same tracker as in Section 5.1, described in [26]. In this experiment, the partner used the obvious communication strategy of “nodding” and “shaking” their head in response to good and bad bids, respectively.

The model was trained with four display states, which is as large as we think is possible to learn reliable models given the training set size. The structure learning algorithm was applied with ωp = 0.9 and ωd = ∞, which finds values of D for which the subpolicies match over 90 percent of the state space. Two values of D had policies that were in p01 = 0.96 agreement (96 percent of the state space) and were merged. The result was, as expected, one “null” state (d1), one “nodding” state (d2), and one “shaking” state (d3). No further merges were found. The associated policy correctly predicted 14/20 actions in the training games and 5/7 actions in the test game.

In order to attenuate the effects of the lack of training data, we use symmetry arguments to fill in the model without having to explicitly explore those situations. In particular, we may assume that players do not have any particular preference over card suits such that the conditional probability tables should be symmetric under permutation of suits. Therefore, we can “symmeterize” the probability distributions by simply averaging over the six card suit permutations. The policy for the symmetrized POMDP correctly predicts all but one (19/20) action in the training game, for an error rate of 5 percent. The misclassification was due to the subject looking to one side of the screen, yielding significant horizontal head motion and a classification as d3.

The symmetrized policy correctly predicts all but one (6/7) action in the test game. The misclassified sequence is longer than usual (over 300 frames) and includes some horizontal head motion in the beginning that appears as shaking in the model. This misclassification may expose a weakness of the temporal segmentation method we use, which is based entirely on the observable actions and game states. Although this sequence is long, it is only the (fairly vigorous) head nod at the very end that is the important display. This situation could be dealt with by incorporating an explicit model of human attention, for example [41].

We performed a second experiment in which the role of bidder was exchanged between the same two players and found similar results. The symmetrized policy in this case predicted all 13 actions in the training game, but only 3/5 in the test game. The mispredictions in this case were not due to misclassifications of the behaviors in the sense that the classifications were consistent with others in the training set. They arose instead due to the lack of training data for the POMDP and would be expected to disappear once more data was incorporated.

5.4 Discussion

The experiments we have presented have demonstrated three things. First, the imitation game showed how we can learn models of complex facial expressions with little prior knowledge and no labeled data. This was done by observing causes of the facial expressions in training data only and then predicting the likely causes with over 80 percent accuracy in the test data. The second experiment then showed how, when behaviors are distinct and the task is simple, we can learn the number of behaviors that occur in the data and that are useful to the task being modeled. In 12 experiments, the correct number of gestures was correctly learned in every case. Third, the card matching game demonstrated that we can apply the same learning and solution techniques to a task of realistic size and learn what behaviors (and how many) are being exhibited and what their relationship is to the task. Using synthetic data from simulations, we demonstrated that we can learn a complex transition function and a complex observation function simultaneously and that the resulting model performs close to optimally in simulation and has a structure (number of displays) close to that of the correct model. We then demonstrated how the learning technique on real data from humans playing the card matching game showed how the learned model can predict the actions of humans, and can learn the correct number of displays being used.

In all three of the experiments we have presented, there was never a need to specify what behaviors were to be recognized. The system learned the behaviors that were used within each task and how these behaviors were related to the utility. Thus, by specifying only some of the fully observable aspects of the domain being modeled (such as the rules of the card game), the system can learn the interactions automatically and a new set of behavior models does not need to be reengineered for each new task. For example, suppose the rules of the card game were changed such that the players had to communicate with their hands alone. While a traditional computer vision approach would have to start afresh by building a recognition system for all possible gestures thought (by some experts) to be those that could be used by the participants, our approach could be applied directly and would learn what gestures were being used and why they were being used. This is the primary benefit of this type of learning: No prior knowledge about human behaviors needs to be included, but, rather, can be learned directly from data.

While the results we have presented show the validity of our method from a technical standpoint, a thorough evaluation of the domains that are impacted by this work remains. A significant limitation is that only simulations were used to select actions and gather rewards online. In theory, correct action selection is the only accurate method for validating that the model was learned and solved correctly. Simply predicting actions taken by a human is not sufficient since the learned model could implement a different, yet still optimal, policy.
than that used by the human. A second limitation of our current experiments is the use of a fully observed state space (apart from the displays). Additional unobserved variables increase learning complexity and makes temporal segmentation more difficult. A third limitation of our current experiments is that we only use a small number of displays and only allow for merging states during learning. We would like the system to learn a larger number of displays by starting from a small number and splitting states based on, e.g., their predictive power.

6 Conclusion

This paper has shown how partially observable Markov decision processes, or POMDPs, can be used to allow an agent to incorporate actions and utilities into the sensing and representation of visual observations. We have shown how this model provides top-down value-based evidence for the learned probabilistic models and allows us to learn models most conducive for achieving value in a task. One of the key features of this technique is that it does not require labeled data sets. That is, the model makes no prior assumptions about the form or number of nonverbal behaviors used in an interaction, but, rather, discovers this from the data during training.

There are three major open questions that remain to be addressed for POMDP modeling of human interactions. First, most interaction tasks will involve state spaces that are significantly larger than we have experimented with in this paper and may be partially or fully unobservable. For example, an assistive system for handwashing used an MDP with over 20 million states [8] and the scalability of the learning method remains to be validated for such larger models. This involves ensuring that we can learn the model parameters for a larger state space and that we can compute good approximate POMDP policies efficiently [29], [50]. However, it is precisely the combination of value-directed learning methods with POMDP solution techniques that will enable the solution of very large POMDPs [43]. The second open question concerns the interplay between a solution for the POMDP and the value-directed structure learning as described in Section 3.3. The strong approximation we used was appropriate for the examples we have examined, but should be relaxed (using, e.g., [29]). More complex structure learning techniques would be required for this learning. The third open question involves the representational power of the observation function. This should be sufficient to distinguish what is necessary to recognize within a particular task. That is, the features used for modeling video sequences of human displays must be able to distinguish what is needed for performance in the task. It remains to be verified if our CHMM-based observation function is sufficient for a wide range of tasks.

Another interesting avenue for future research is the use of the POMDP models in active vision systems [17]. The trade-off between behavior recognition and resource use can be explicitly addressed by a POMDP model and, coupled with the behavior learning techniques we have presented in this paper, would form a possible solution to the active vision problem.

We are currently applying POMDP models to assisted living tasks in which a POMDP-based system helps a cognitively disabled person complete activities of daily living [30]. POMDP models are well suited to this environment since they model the stochastic nature of user behavior, the need to trade off various objective criteria (e.g., task completion, caregiver burden, user frustration, and independence) and the need to tailor guidance to specific individuals and circumstances [8].

Acknowledgments

The authors would like to thank Don Murray, Pantelis Elinas, Pascal Poupart, David Lowe, and David Poole for invaluable help and suggestions. This work was supported by grants from the Natural Sciences and Engineering and Research Council of Canada and from the Institute for Robotics and Intelligent Systems, a Canadian Network of Centres of Excellence.

References
