Interval Research Technical Report 1997-045. Portions of this paper previously appeared in *Advances in Neural Information Processing Systems 8*, (NIPS '95), pp. 858-864, MIT Press, and *Intelligent Robotic Systems*, M. Vidyasagar ed., pp. 73-80, Tata Press, 1998.

Reinforcement Learning of Active Recognition Behaviors

T. Darrell
Interval Research Corp.
1801 Page Mill Road Bl. C
Palo Alto, CA 94304
trevor@interval.com
trevor@interval.com
http://www.interval.com/papers/1997-045

Abstract

We show how a concise representation of active recognition behavior– what observations to make to detect a given object– can be derived from hidden-state reinforcement learning techniques. These learning techniques can solve decision process tasks which include perceptual observations, defined formally as Partially Observable Markov Decision Processes (POMDP). We define recognition within a POMDP context, with an action indicating recognition of the target as well as actions for adjusting the perceptual apparatus or other effectors. An explicit supervised reward signal is provided to the decision process whenever the accept action is performed. With sufficient experience, a memory-based approach to reinforcement learning can find optimal policies which discriminate target from distractor patterns despite considerable perceptual aliasing at any given instant. To avoid perceptual aliasing while learning, all similar experiences are combined when computing the utility of a possible action, including experiences with both target and distractor patterns. By discarding the representation of negative regions of the utility space when learning is complete, and collapsing duplicate representations of positive regions, a representation similar to an augmented Finite State Machine is obtained. We show application of our method for the task of recognizing human gesture performance that occurs at multiple spatial scales.

1 Introduction

Visual phenomena take place over a wide range of scales, and often can not be captured by a single passive camera view. This is especially true in the domain of observing people, since gestures performed by the face or hand require detailed observation while recognition of overall body pose requires a relatively coarse scale view. Confounding matters further is that the significance of a particular detailed gesture can depend on the coarse context in which it is performed.

Techniques for observing people have been a topic of increasing interest in the machine vision literature recently. Progress has been reported by several independent groups on methods for discriminating faces and facial expressions, hand gestures, and tracking body pose. With the exception of face detection methods, the techniques reported for hand and face processing generally assume that high-resolution views (greater than 50x50) of the hand or face are available. With fixed cameras, these methods are limited to domains where face/hand location is stationary and known a priori: for face processing this is possible only in certain domains such as automobile driving or interaction with bank automated teller machines. For hand processing, this usually restricts the class of applications to those in which more traditional interface methods such as mouse or touch-screen are also available (and are often preferred by users). When camera views are fixed, the application of face and hand processing is limited.

A solution problem to this lies in the use of Active Vision techniques [1, 2, 3]. Active visual observations are used both in biological and machine vision systems to overcome the limitation of fixed view; actuators are added to the the perceptual apparatus together with foveal sensors that have (or approximate) non-uniform photodetector sampling. The technology for active visual observation has become increasingly accurate and available to the average researcher: high-performance visual tracking systems whose performance rivals biological systems are being developed, while basic pan/tilt/zoom cameras have become readily available at low cost[1], driven by teleconferencing applications.

The addition of active methods to the perception process raises several interesting questions, chief among them how to determine optimal methods for controlling the active apparatus to achieve perceptual goals such as recognition, i.e., how to perform visual attention. We choose not to focus on the data-driven / bottom-up aspect of attention; our approach instead explores top-down influences in attentive control by attempting to find active recognition strategies to recognize particular objects. We use a decision processes formalism and reinforcement learning solution technique for top-down control, and assume a data-driven, bottom-up module provides candidate foveation locations that appear interesting due to a prior feature model, local statistical content or change over time.

Several methods have been proposed in the machine perception literature for the active control of a camera to reduce the uncertainty that a particular object is present [6, 13]. In contrast to these methods, which assume a model of the object, we explore the case when no model of the object or environment is available. Our active recognition system has no access to the underlying state of the world, only to actions and observations made by the system, and an externally provided reinforcement signal that indicates whether assertions of object identity are correct.

To accomplish this we use a hidden-state decision process paradigm for active recognition. Since these decision process formalisms explicitly describe both action and perception in a statistical framework, they are potentially useful for modeling aspects of visual attention. As we shall see in the following sections, hidden-state reinforcement learning can solve decision process tasks which are provided only with observations of actual state, overcoming the perceptual aliasing problem by modeling state as sequences of actions and observations. Reinforcement learning has been explored for use in active visual tasks by several authors recently [21, 4, 22, 23], but none address the task of hidden-state recognition without a prior model.

In this paper we show how these techniques can be used to solve an active recognition task, and how a concise, visual behavior-like representation can be extracted from the learned action-selection policies. The following section presents our definition of an active recognition task within a partially observable decision process formalism. We then present the hidden-state reinforcement learning methods with which we learn action-selection policies to solve these tasks. Finally we present a method for transforming the learned action-selection policy into a simple augmented state-machine behavior, and conclude with a discussion of the properties and limitations of our current method.

2 Active-Perception Recognition Tasks

We desire a method for perceptual action selection that can learn from experience and be applied to a recognition task. Because we have a foveated image sensor which can only observe a single portion of the user at high resolution, at any given moment the full world state is hidden from our system. We thus need an action-selection method that can learn from only partial observations of state. By definition, a system for perceptual action-selection must not assume a full observation of state is available, otherwise there would be no meaningful perception taking place. Inspired by the success of statistical methods for hidden state learning in the domain of static perception (e.g., Hidden Markov Models), for active tasks we have chosen to explore the use of a hidden state learning model with both

[1] Such as the Sony EVI-D30 and the Cannon VCC1.

action and perception: the Partially Observable Markov Decision Process (POMDP).

A Partially Observable Markov Decision Process (POMDP) is essentially a Markov Decision Process without direct access to state [24, 18]. Formally, a POMDP is defined as a tuple, $< \mathcal{S}, \mathcal{O}, \mathcal{A}, Tr(\cdot), Ob(\cdot), R(\cdot) >$, where $\mathcal{S}$ is a finite set of states, $\mathcal{O}$ and $\mathcal{A}$ a set of observations and actions, $Tr(\cdot)$ a model of state transition probabilities, and R a function giving the reward associated with executing a particular action in a particular state. After executing a particular action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, the world transitions to a new state s' with probability $Tr(s, a, s')$, and the agent receives a reward $R(s, a)$ and observation $o \in \mathcal{O}$ with probability $Ob(s, a, o)$. We model state in the POMDP as the cross-product of world state and and the perceptual state of the camera, $\mathcal{S} = \mathcal{W} \times \mathcal{P}$. In our work we do not assume the system has access to $\mathcal{W}$, nor does it know the transition likelihood between states $Tr(\cdot)$ or the likelihood function mapping states to observations $Ob(\cdot)$; we leave these unspecified.

We formulate an "Active Gesture Recognition" (AGR) task using this POMDP framework, and have found that instance-based reinforcement learning is a feasible means for finding good foveation policies. The set of states $\mathcal{W}$ describes the various person or object configurations possible in the scene. Since we have a foveated sensor, we assume that portions of the state of the world are only revealed via a moving fovea, and that the set of actions exist to perform that foveation. Some portion of the world state (e.g., the low-resolution view) may be fully observable and always present in the observation. The set $\mathcal{A}$ contains actions for foveation, a special action labeled `accept` and a null action. By definition, execution of the `accept` action by the AGR system signifies detection of the target pattern in the world. The goal of the AGR task is therefore to execute the `accept` action whenever a target pattern is present, and not to perform that action when any other pattern (e.g. distractor) is present. A pattern is simply a certain world state, or more generally a sequence of world states when targets are dynamic. The AGR system should use the foveation actions to selectively reveal the hidden state needed to discriminate the target pattern.

In the AGR task we define the reward function to provide a unit positive reward whenever the `accept` action is performed and the target pattern is present (as defined by an oracle, external to the AGR system), and a fixed negative reward of magnitude α when `accept` is performed and a distractor (non-target) pattern is being presented to the system.[2] Zero reward is given whenever a foveation action is performed.

In reinforcement learning problems we wish to find a *policy*, a mapping from state (in the case of an MDP) or some function on observations (in the case of a POMDP) to action which maximizes the expected future reward, suitably discounted to bias towards timely performance. An optimal policy maximizes

$$Z = \sum_{t=t_0}^{\infty} \gamma^{(t-t_0)} r[t] \qquad (1)$$

where $r[t]$ is the reward obtained at time t, γ is the discount parameter (we used $\gamma = 0.8$). Given the reward function in the AGR task, this will correspond to a policy which successfully recognizes the target pattern.

2.1 Active Recognition Tasks

We have experimented with AGR tasks in two domains: an extended image domain used for algorithm evaluation and pedagogical purposes, and an interactive interface domain features from person-tracking and gesture analysis routines.

In the interactive interface domain, we have implemented the AGR task using primitive routines to provide the continuous valued control and tracking of the different body parts that represent/contain hidden state. We represent body pose and hand/face state using a simple feature set, based on the representation produced by a body tracker [28] and an appearance-based recognition gesture system [9, 10]. (See Figure 1(a,b)). We define the world state (which we also call user state in this domain) to be a configuration of the user in the scene. $\mathcal{W}$ is defined by body pose, facial expression, and hand configurations, expressed in nine variables (see Table 1). Three of these are

[2]The parameter α expresses the trade-off between the two types of recognition errors, false alarms and misses; for the all the results presented in this paper we have taken a conservative approach and set $\alpha = 10$. This is somewhat similar to the idea of disproportionately penalizing perceptually aliased states in Whitehead's Lion algorithm [27].

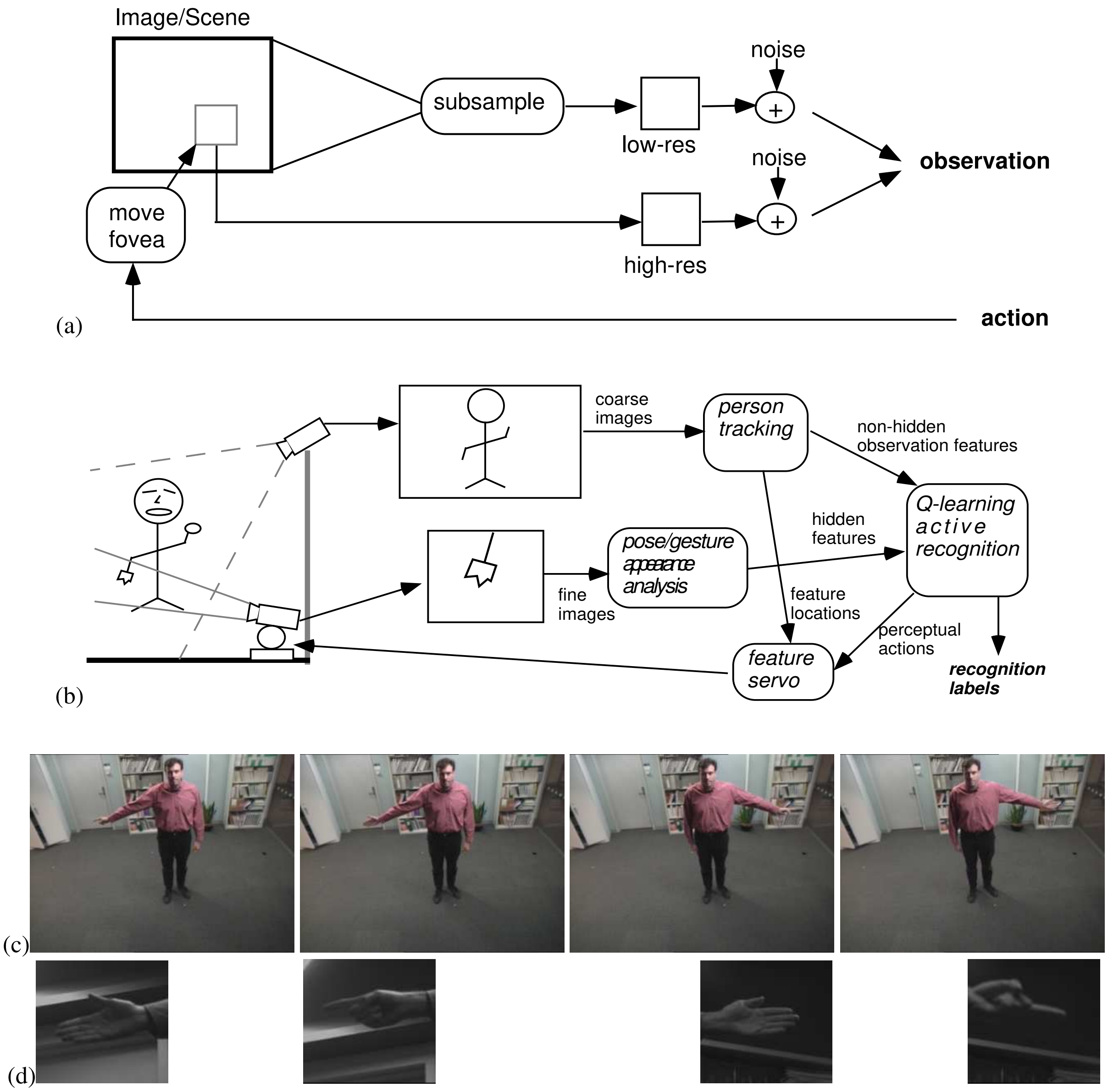

Figure 1: (a) Active observation framework; static low-resolution observations are combined with active, high-resolution observations. Explicit actions guide fovea region. (b) Overview of AGR task in interactive domain. Real-time vision modules track body pose and hand/face gestures, providing input to hidden-state reinforcement learning module which chooses next observation and outputs recognition labels. (c,d) Four gesture patterns in interactive interface domain which require foveated images for discrimination: (c) Output from wide field of view camera; (d) output from narrow field-of-view active camera.

feature	*values*	*observability precondition*
`person-present`	(true, false)	(always observable)
`left-arm-extended`	(true, false)	(always observable)
`right-arm-extended`	(true, false)	(always observable)
`face-foveated`	(true, false)	(always observable)
`left-hand-foveated`	(true, false)	(always observable)
`right-hand-foveated`	(true, false)	(always observable)
`face`	(neutral, smile, surprise, ...)	`face-foveated` == true
`left-hand`	(neutral, point, open, ...)	`left-hand-foveated` == true
`right-hand`	(neutral, point, open, ...)	`right-hand-foveated` == true

Table 1: Set of features used in POMDP formulation of Active Gesture Recognition task in interactive interface domain. This representation is computable in real-time using person tracking and gesture recognition routines described in [28, 9, 10].

boolean, `person-present`, `left-arm-extended`, and `right-arm-extended`, and are provided directly by the person tracker. Three more are provided by the foveated gesture recognition system, `face`, `left-hand`, `right-hand`, and take on an integer number of values according to the number of view-based expressions/hand-poses: in our first experiments `face` can be one of `neutral`, `smile`, or `surprise`, and the hands can each be one of `neutral`, `point`, or `open`. In addition, three boolean features represent the internal state of the vision system: `head-foveated`, `left-hand-foveated`, `right-hand-foveated`.

At each time-step, the full state $s \in \mathcal{S}$ is defined by these features. An observation, $o \in \mathcal{O}$, consists of the same feature variables, except that those provided by the foveated gesture system (e.g., head and hands) are only observable when foveated. Thus the `face` variable is hidden unless the `head-foveated` variable is set, the `left-hand` variable hidden unless the `left-hand-foveated` variable set, and similarly with the right hand. Hidden variables are set to a `undefined` value. The set of actions, $\mathcal{A}$, available to the AGR system in this example are 4 foveation commands: `look-body`, `look-head`, `look-left-hand`, and `look-right-hand` plus the special `accept` action and a `null` action. Each foveation command causes the active camera to follow the respective body part.

We also have implemented a version of the AGR task with target patterns which are defined as simple images. In this extended image domain, world state is simply a single high-resolution image, and the observation consists of a sub-sampled version of the entire image, plus a full-resolution window over a foveated region of the image. The fovea is a fixed size rectangle and can be moved by executing a set of foveation actions. Gaussian noise with variance σ^2 is added to the sub-sampled and windowed images to yield the low and high resolution observations.

In this domain we use a data-driven process to determine possible actions. Given a target image and a distribution (or set) of distractors, we sample distractor images, and compare with the target to determine locations which can possibly discriminate perceptually aliased pairs. We first normalize the center of mass of each object image so that actions will be computed in object-relative coordinates. Each pair of images which is not discriminable[3] using the low-resolution (fully-observable) portion of the observation is passed to a high resolution comparison stage. In this stage the images are compared and all points which differ are marked into a candidate foveation mask. The marked points in the mask are then clustered, yielding a set of final foveation targets. These target locations are converted to foveation actions to fixate at the given coordinate in an image of a new object, relative to the new

[3] In these examples observation differences were determined using an ad-hoc threshold θ (we set $\theta = 2$) on the variance normalized average pixel differences between images in the two observations.

object's coordinate frame.

In both domains, when evaluating the performance of an AGR system we present test and training patterns to the learning system interleaved with blank fields. Typically we switched patterns and started a new trial after 10-15 time steps, or after `accept` was generated.

3 Finding optimal action- selection policies

Given a POMDP problem, one wishes to construct a *policy*, π which maps a function on observations to actions, and whose execution leads to maximizing the criterion function in Equation (1). Methods for constructing a policy based on a POMDP model can be divided into two approaches: indirect and direct. Indirect methods attempt to estimate the actual state of the world, and then treat the problem as a conventional Markov Decision problem (MDP) with full state access. Direct methods forgo recovery of the actual state and attempt to characterize optimal behavior given only actions and observations. We will discuss examples of each of these, and then a hybrid approach using memory-based approaches to model world state.

Indirect methods combine a predictive forward model that recovers an estimate of absolute state with a conventional MDP learning method. Given absolute state, recovering an optimal policy for a MDP is a well-studied problem. Q-Learning is one widely used reinforcement learning method for finding an optimal MDP policy. The Q function maps state and action pairs to a *utility* value, which is the expected discounted future reward given that state and action. Optimal utility values can be computed on-line using a *Temporal Differences* method [25], which is an incremental form of a full Dynamic Programming approach [5]. In the case of deterministic state transitions, to maximize Eq. (1) the optimal Q function must satisfy

$$Q(s,a) = r + \gamma \max_{a \in \mathcal{A}} Q(s',a) \tag{2}$$

where s' is the next state after executing action a in state s. To find an optimal Q, the difference between the two sides of Eq. (2) is minimized, using the well known update rule:

$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(r + \gamma \max_{a \in \mathcal{A}} Q(s',a)), \tag{3}$$

where β is the learning rate parameter. This has been shown to converge to optimal policies when the environment is Markovian and the Q-function is represented literally as a lookup table [26].

Several approaches have been proposed which combine a predictive model to transform a POMDP to a MDP and then use Q-learning to construct a policy function. The Perceptual Distinctions Approach (PDA) developed by Chrisman [8] uses a state-splitting approach to construct a model of the domain. A modified Expectation-Minimization algorithm was used to alternate between maximizing the model probability given a fixed number of internal states, and splitting perceptually aliased states into new states. A similar method was developed by McCallum, the Utile Distinction Memory approach (UDM), which split states based on the predicted reward [19]. Given a state estimator learned via PDA or UDM, or simply computed from T and O, an optimal policy can be constructed [7]. However these indirect methods have been criticized as being computationally intractable in realistic environments [14]. Indeed, the empirical results suggest that a prohibitive amount of training time is required with these algorithms [20].

Rather than convert a POMDP to a MDP through state estimation, direct methods update a value function over observations without recovering absolute state. Jaakkola et. al. present a Monte-Carlo algorithm for policy evaluation and improvement using Q-values defined over actions and observations [14]. They provide a recursive formulation of the Monte-Carlo algorithm, and prove the policy improvement method will converge to at least a local maximum.

A hybrid approach sharing both direct and indirect qualities has been explored by Lin [16] and McCallum [20]. These methods use a memory-based approach to identifying state. Lin's *window-Q* algorithm supplies a history of the N most recent observations and actions to a conventional Q-learning algorithm. The intuition is that perceptually aliased states can be discriminated by indexing utility over a vector of recent experience. A significant drawback to this approach is the use of fixed window size. McCallum's *Instance-Based* state identification method extends this approach to use windows of varying length, using a sequence matching criteria with a K-nearest neighbor utility representation.

The memory-based approach to hidden-state reinforcement learning can be successfully applied to our AGR task, once we generalize the the nearest-neighbor algorithm. As described in Section 3.2, we modify the nearest neighbor computation to pool all experience whose similarity (e.g., sequence match length) is approximately equal to the K'-th percentile rank, rather than simply using the K largest similarity scores. This is important in domains where the world is non-stationary, such as with rapidly changing target and distractor patterns; in this case we need to average *all* experiences with a particular sequence of observations when computing utility. We first describe the basic memory-based, hidden-state reinforcement learning algorithm, and then present our extensions.

3.1 Memory-based Reinforcement Learning of Hidden-state Control Policies

Instance-based Q-learning replaces the state-action table with a distributed utility representation [20]. Given a history of action, reward, and observation tuples, $(a[t], r[t], o[t])$, $0 \leq t \leq T$, a Q-value is also stored with each time-step, $q[t]$. Utility is computed by averaging Q-values of the most similar previous experiences, defined using a sequence match criteria. Q-values are computed, and the Q-learning update rule applied, maintaining this distributed, memory-based representation.[4]

As in traditional Q-learning, at each time-step the utility of performing each action in the current state is computed. If full access to the state was available and a table used to represent Q values, this would simply be a table look-up operation, but we do not have full access to state, nor a table representation. Instead, utility is associated with the implicit state the system was in at previous time point. Using McCallum's Nearest Sequence Memory (NSM) algorithm, we find the K nearest neighbors in the history list for a given action, and compute their average Q value.

[4]Note that instance-based Q-learning does not assume any knowledge of the set of states $\mathcal{S}$, or the likelihood functions $Tr(\cdot)$, $Ob(\cdot)$.

Nearest neighbors are computed using a sequence match criteria; we look to prior experiences that are most similar to the current time point, in terms of having had the same preceding sequence of actions and observations. For each element on the history list, we compute the sequence match criteria with the current time point, $M(i, T)$, where $M(i, j)$ indicates the number of action and observations that overlap between the sequence ending at time i and the sequence ending at time j:

$$M(i,j) = S(i,j) + M(i-1, j-1) \text{ if } S(i,j) > 0 \quad (4)$$

$$0 \quad \text{otherwise} .$$

We modify McCallum's algorithm slightly and ignore reward in computing matches: we define $S(i, j)$ to be 1 if $o[i] = o[j]$ or $a[i] = a[j]$, 2 if both are equal, and 0 otherwise; we also define $S(i, j) = 0$ if the low-resolution component of $o[i]$ and $o[j]$ both indicate an empty field.

For each candidate action, we compute the K most similar previous experiences that were then followed by the candidate action. Using a superscript in parentheses to denote the action index of a Q-value, we then compute

$$Q^{(a^*)}[T] = (1/K) \sum_{i=0}^{T} v^{(a^*)}[i] q[i] \,, \quad (5)$$

where $v^{(a^*)}[i]$ indicates whether the history tuple at time-step i votes when computing the Q-value of a new action a^*: $v^{(a^*)}[i]$ is set to 1 when $a[i] = a^*$ and $M(i-1, T)$ is among the K largest match values for all k which have $a[k] = a^*$, otherwise it is set to 0.

Given Q values for each action the optimal policy is simply

$$\pi[T] = \arg\max_{a^* \in \mathcal{A}} Q^{(a^*)}[T] \,. \quad (6)$$

The new action $a[T+1]$ is chosen either according to this policy or based on an exploration strategy; a random action is taken whenever maximum utility is zero.[5] In either case, the action is executed yielding an observation and reward, and a new tuple added to the history. The new Q-value is set to be the Q value of the chosen action,

[5]Maximum utility can never be negative, since the `null` action will always lead to zero or greater utility (it can be repeated infinitely with zero reward).

$q[T+1] = Q^{(a[T+1])}[T]$. The update step of Q learning is then computed, evaluating

$$U[T+1] = \max_{a^* \in \mathcal{A}} Q^{(a^*)}[T+1] \,, \tag{7}$$

$$q[i] \leftarrow (1-\beta)q[i] + \beta(r[i] + \gamma U[T+1]) \,, \tag{8}$$

for each i such that $v^{(a[T+1])}[i] = 1$.

3.2 Variable-K Nearest Neighbor Algorithm

The key to successful application of instance-based Q-learning algorithm is the identification of the appropriate instances (e.g. memories) on which to base estimates of the utility of a particular new action. In our AGR task, it is especially important that the utility of a particular action given a particular observation history be estimated from previous experience *on both target and distractor trials*. Estimates of utility which are based only trials with the target present will greatly overestimate the utility of all actions, since with the target present by definition any action chain followed by `accept` will yield positive reward. Conversely, estimates of utility based only on trials with distractors will grossly underestimate the true utility. Effective learning occurs in our system only when the utility of a particular action is considered over trials with both target and distractor.

This poses a problem for the instance-based algorithm defined above, which finds voting instances using a strict K-nearest neighbor method, since it depends critically on choosing the correct value of K. A value which is too small will miss some relevant experience, and yield an unreliable utility estimate; a value which is too large will merge together instances which are actually different states, returning us to the problem of perceptual aliasing. Since the number of instances (memories) which are relevant to a particular utility estimate may vary greatly depending on the distribution of target and distractor trials, no fixed value of K will suffice.

A simple example illustrates the problem: consider the estimation of the utility of `accept` when the previous observation/action sequence was to foveate the face and observe a neutral expression. Assume that the target and all distractors all have a face with neutral expression, so this should be of no use in discriminating target from distractor. During the training phase the system will likely have explored this sequence several times, some with the target and some with the distractor. When match similarities are computed, the memories with maximum match value will indeed all correspond to the relevant previous experience. There may be 20 such previous experiences, but a strict K-nearest neighbor algorithm with K=4 will ignore 16 of them. If these 4 memories happen to miss experiences with the target, which is not unlikely, then the computed utility will be incorrect.

Rather than set K very large and risk merging truly different states, we instead propose a new algorithm, which we call *Variable-K Nearest Neighbors*. Variable-K differs from Strict-K in that all memories with a given match length are included in the voting set, e.g., are pooled together when estimating utility.

In the Variable-K algorithm, we assert a minimum percentage kp of experience to be included. (If there is no noise in the match function, this can be effectively set to zero; in our experiments we used $kp = 5\%$) We histogram the match lengths of all memories, and sort them from largest to smallest. We then find the match length value m' such that the $kp\%$ of memories have larger match lengths than m'. We then subtract a fixed amount kt to this match length value, and include all memories with match value greater than $m' - kt$ into the voting set.

The effect of this algorithm is to find a percentage of nearest neighbors with largest match values, as does strict-K, but then to include all other memories with roughly the same match length. In the above example, all 20 experiences would be included in the voting set, and utility accurately estimated. In practice we have used $kt = 1$, and found this algorithm greatly improved the performance of instance based Q-learning given the multiple trial (e.g. non-stationary world) structure of our AGR task.

With the Variable-K enhancement, instance-based Q-learning can find optimal policies to solve AGR tasks. Figure 2 shows the results using this hidden-state Q-learning algorithm on an AGR task using image-based world state. In this example four different world states were used for the target and distractors; varying amounts of noise were added to the actual observations at run-time. We ran the system by randomly selecting one of

Figure 2: AGR results for simple imagery targets: (a) set of four target patterns, where the first and last pair are indistinguishable at low-resolution (b) recognition performance (percentage of correct trials) for varying amounts of observation noise.

the patterns as target, and using the other three as distractors. This Q-learning system easily learns the correct foveation policies to actively recognize these targets; e.g., to foveate down to recognize one of the square targets, and to foveate up to recognize one of the circle targets. For moderate levels of noise, each pattern can be easily discriminated from all the others, however, as σ exceeds 20.0[6], the high-resolution information in the signal becomes insufficient to discriminate the two square targets and the two circle target, so performance drops by half. As σ approaches 60.0, the low-resolution information becomes indistinguishable, and all the targets are confused, leading to a recognition rate of zero.

Successful policies were also found for the interactive interface domain. Figure 3 shows the recognition performance on the set of gestures shown in Figure 1(c,d). These patterns were quite simple; and convergence to an accurate recognition policy is straightforward. For more complicated targets, it is possible for the random policy search to converge prematurely to a local maxima of Equation (1), which can correspond to good performance but still have errors. While there are many strategies for focusing search in a structured manner and overcoming local minima, we have implemented a simple strategy: we detect local minima and restart the learning process with a new random seed when premature convergence is found. We detect convergence by examining average recognition error over a window of ($N_c = 100$) recent of trials: if during the entire window average the error rate has remained fixed, was above an upper threshold (99%) or below a lower threshold (1%), we declare the algorithm converged. We then test N_c subsequent trials; if an error is detected we determine that this was a local minima, and restart the learning process, clearing experience memory and beginning again with a new random seed (which affects both the sequence of target/distractor patterns and the exploration actions chosen.) Additionally if we exceed a fixed maximum memory length ($M_l = 10000$), we restart learning. This algorithm is suboptimal in terms of learning time, but does find correct recognition policy solutions. In the following section we address the issue of how to extract a more efficient run-time representation from these learned policies.

4 Extracting Stable Behavior from Learned Policy

The hidden-state reinforcement learning procedure described above can learn to solve active recognition problems. However, as is true with all reinforcement learning approaches it can take considerable exploration to find the optimal solution. Since we use a memory based approach, storing all previous experience, this has the disadvantage that the learned representation can grow quite large. (And keeps on growing, if the algorithm is implemented literally.)

[6] The image range was $[0, 255]$.

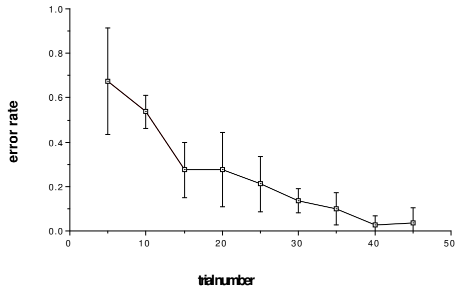

Figure 3: Performance with random exploration on AGR task with targets shown in Figure 1(c,d). For each run, one gesture was randomly chosen as target, and the other three used as distractors. Results were averaged over 100 runs; average error rate is plotted as a function of trial number during each run. Recognition converges to correct performance on average after approximately 40 trials.

We ran experiments to measure the amount of experience and random search needed by the hidden-state reinforcement learning system to find active recognition policies in the interactive interface domain. In each experiment we ran the learning system with a random user state to be the target, and used the remaining states in $\mathcal{W}$ to be distractors. While correct performance was eventually achieved in each case, it can take a considerable number of steps to find a solution. Averaged over 22 experiment trials, there were approximately $T = 2973$ experience time-steps in the system before a solution was found, even though the most complicated recovered recognition strategy contained no more than four actions! This is not entirely unsurprising, since the system needs to average reward across many trials to avoid perceptual aliasing, and perform considerable random exploration to find solutions.

A typical solution executes a small number of actions in only few different perceptual conditions. I.e., to detect a target gesture where the user is smiling and pointing with her right hand, the system waits until it sees a form with extended arm, then looks to the face, if it sees a smile expression it then looks to the right-hand, and then if it sees a point gesture it generates the `accept` action. Since the actual behavior of the algorithm after convergence can be quite simple, it seems desirable to attempt to capture just this simple behavior, and eliminate experience not needed for its performance.

A further problem is that the Q-learning policy is sensitive to the distribution of target vs. non-target trials. In Q-learning the utility of actions is constantly updated; if the target stops appearing, or if the distractors which are perceptually aliased with the target stop appearing, then the system will evolve towards a new policy which is optimal to the distribution of targets at the current moment, but may not be the desired policy. This plasticity is of course the central feature of general learning methods, but at some point we may wish to fix utilities and remove this dependence. We may need to have a statistically representative training set during learning, but it is desirable to have no such requirements for the testing set.

Our innovation to overcome these problems is to devise a method for extracting a visual behavior-like representation from the learned policy. We observe that after convergence of the learning algorithm, the utility values represent the average expected reward propagated backwards through time and discounted appropriately, (i.e. a time-discounted approximation to the confidence that we will detect the target). We thus no longer need the actual experience containing the raw rewards, if we can extract or reconstruct the relevant representation of utility. We further observe that the experience representation used in our hidden-state reinforcement learning method shares certain similarities to a representation that is simply a series of state transitions, with links augmented with actions as well as observation pre-conditions. Indeed, we can transform our learned policy into a state transition representation, and eventually into a Finite State Machine augmented with action transition labels.[7]

Conversion to an augmented Finite State Machine (aFSM) has several advantages for representing run-time recognition behavior. It is more stable, compact, and does not require any of the matching or learning machinery presented in the previous section. Indeed, the states that are implicitly represented in the instance-based method are made explicit in the aFSM representation. The aFSM is defined by a set of recognition states $\mathcal{RS}$ and a set of directed links $\mathcal{L}$ each labeled with an associated observation and action pair. Terminal nodes are labeled with target identity, and transition into them indicate detection of the target. We use three steps to convert our reinforcement policy to aFSM: utility space pruning, conversion to a recognition state chain, and finally integration of multiple target strategies. We will discuss each of these in turn. We also will show how recognition policies learned for multiple targets can be merged during conversion to the aFSM representation.

4.1 Run-time action-selection and utility-space pruning

We first note that once correct behavior has been learned, the utility values contain all the relevant information for action selection at run-time. I.e., the selection of new actions in Equations (6,5) depends only on $q[i]$, and not on $r[i]$. Rewards only have influence in the Q-learning update step, Equation (8). Thus when the utility values have converged to optimal behavior for a given target and distribution of distractors, we no longer need to perform this update step; this renders the reward values themselves no longer of use, and freezes the utility value space. This has the immediate potential benefit that the system will now behave independent of the future distribution of targets and distractors.

For many if not most recognition tasks, a small subset of the learned utility space is sufficient to drive successful recognition. Since the null action is always available at zero utility or greater, regions of the utility space with negative utility will, as can be seen in Equation (6), never influence action selection. We can thus set to zero any negative utility values, and further prune the memory of any experience tuple with zero utility assuming the observation and action of that experience are never used in the nearest sequence matching process when a positive utility value is returned.

Most powerfully, we observe that after utility space convergence, recognition behavior for our task can be driven solely from utility associated with experience with the target. When utility values have properly converged the representation of regions of utility space defined only by experiences with distractors is irrelevant: any instances of positive utility values in distractor trials will occur over subsequences of experience that are perceptually aliased with target trials, and the copy of those experiences in the target trials is sufficent to drive correct recognition behavior.

Our final experience pruning observation is that any target trials which are redundant due to equivalent observation and action sequences can be merged together without affecting action selection.[8] Based on these observations, we perform the following steps to prune the action-selection policy computed from reinforcement learning once the policy has converged to correct performance:

- Utility values are frozen; the Q-learning update step (Equation 8) is no longer applied.
- Execution continues for fixed number of trials (sufficient such that several experiences with the target pattern will be observed), discarding any new trial which either has uniformly negative utility or which is known to be a distractor trial.
- The memory of all trials collected prior to pruning is discarded, retaining only those trials collected while learning was frozen.
- Any two trials which have equivalent observation and action sequences are merged.

[7] Ideally, this would eventually lead to a behavior representation that integrated discrete and continuous control, such as in [15].

[8] When merging two sequences we perform a weighted average of utility, setting the weight of the merged sequence to be the sum of the individual weights (which are initialized to 1.) We do modify the Nearest Sequence Match algorithm such that this weight value is considered when averaging utility and computing match length histograms.

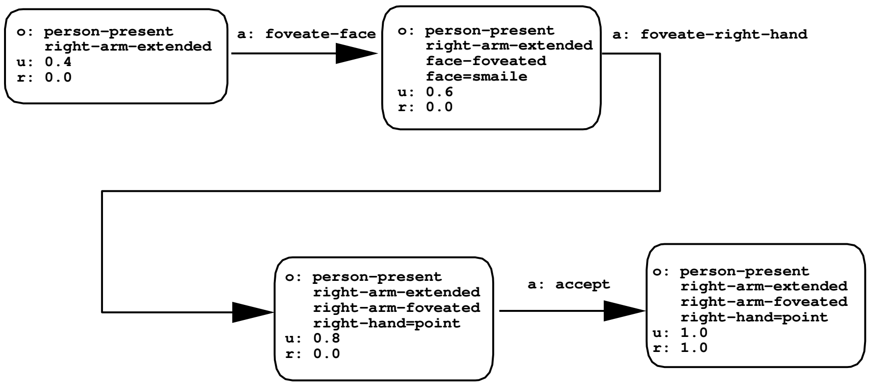

Figure 4: Active recognition policy used in hidden-state reinforcement learning method, after pruning algorithm described in Section 4.1 is applied.

We have found that this pruning method can extract representations of utility that capture recognition behavior for our tasks in an intuitive and efficient way. Typically, one or two chains are recovered, which simply list the steps required to discriminate target from distractor. Figure 4 shows the memory-based action selection policy for one of the randomly selected targets in the experiment described above. The solution to these recognition problems generally involves first checking to see if the coarse pose is the same as the target, then observing the face; if the face pose matches the target then one or both hands is observed. If all hidden features match the target, accept is generated.

4.2 Conversion to recognition-state chain

To convert from hidden-state action-selection policy to an aFSM representation, we first compute a set of recognition states $\mathcal{RS}$ corresponding to the implicit states of the active recognition system expressed in the learned policy. Recall that the nearest sequence matching algorithm associates utility with sequences of action/observation tuples of varying length, so each subsequence of a recovered experience chain is a possible recognition state.

The set of actual recognition states used by a particular policy is found by marking the "nearest neighbor" sequences which lead to computation of positive utility in Equation 5 when the pruned policy is executing. I.e., whenever a positive utility value is returned by Equation 5, for each voting time-step $v[i]$ we define a recognition state to be associated with the $M(i-1,T)$ previous time-steps (recall $M(i-1,T)$ is the match length associated with $v[i]$).

Each recognition state will thus correspond to an action/observation sequence of varying length in the learned policy. We look in the policy to determine what transitions are defined between recognition states. For each recognition state whose policy sequence "follows" another recognition state we note the action and observation linking the two, and add a link to $\mathcal{L}$ (the set of labeled links in the aFSM). Each link is labeled with the observation as a precondition of traversal, and the action to take when traversing the link. "Follows" here is defined to mean that the endpoint of the subsequence corresponding to one recognition state is one time-step earlier than the endpoint of the subsequence corresponding to the following recognition state.

For the example in Figure 4, three recognition states are found, not surprisingly corresponding to the progressive revelation of the hidden features of the target pattern. Figure 5(a) shows the recognition state chain corresponding to this example.

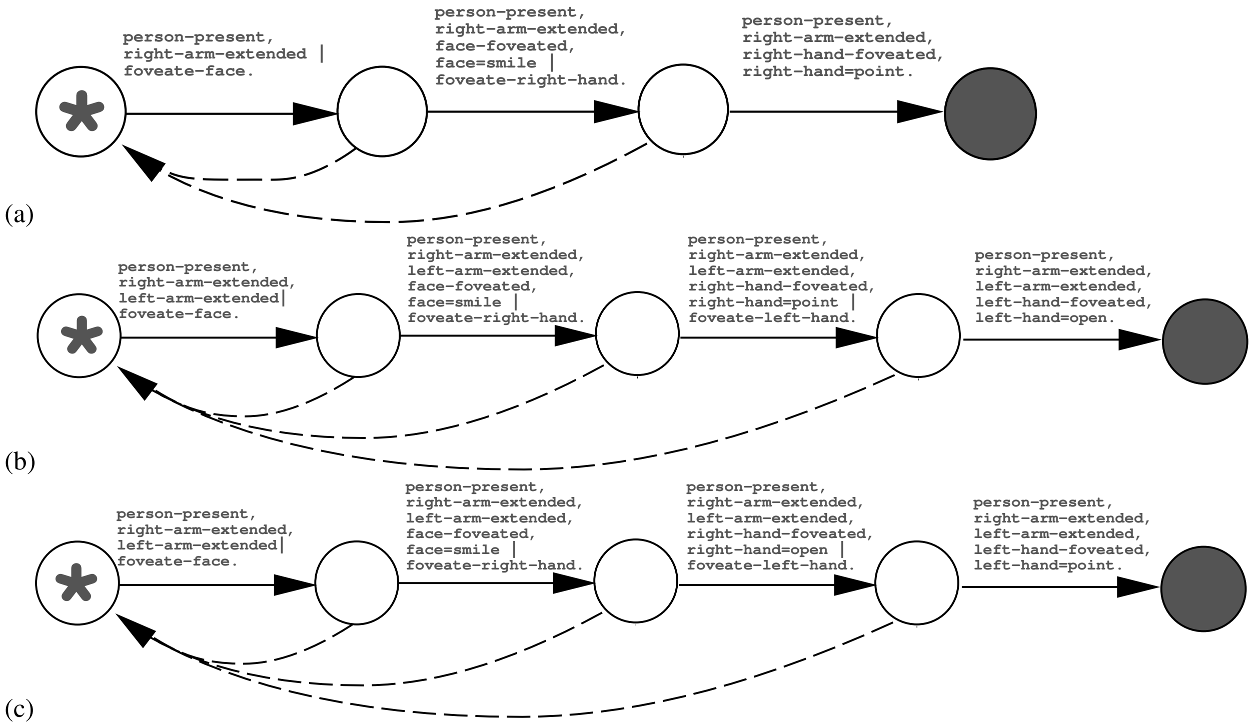

Figure 5: Augmented Finite State Machine representation of three different active recognition policies. Circles indicate state, arcs are labeled transitions marked with "$o|a$", where o is the list of observations that are a precondition for link traversal, and a is an action to execute when traversing the link. Dashed lines inidicate the default traversal when no precondition is met. An asterisk marks the initial state, and the shaded circle marks the terminal state indicating recognition. (a) recognizes the gesture where the user is smiling and pointing with his right hand with his left arm down, (b) the gesture where the user is smiling, pointing with the right hand, and has his left hand open, and (c) the gesture where the user is smiling, has the right hand open and is pointing with his left hand.

4.3 Integration of multiple target strategies

Finally we convert the recognition state chain into an aFSM. We add default links returning the system to the initial state when no perceptual condition is satisfied, and we define recognition states entered following the `accept` action to be terminals.

It is also straightforward to integrate strategies for recognizing different targets at this point. We separately train policies corresponding to different target patterns, and then use syntactic heuristics to merge them at the recognition state level.[9] Equivalent recognition states in each chain (or aFSM) are merged, taking the union of links to subsequent states. We also merge the initial states from each separate aFSM.

When merging states, we check to see whether duplicate perceptual conditions are present for any recognition state, and if so we re-arrange the links appropriately. If two links l_1, l_2 exiting state s_0 leading to new states s_1, s_2

[9] Policies can also be trained simultaneously for multiple goals, using the approach presented in [11]. For the purposes of pruning and conversion to representation space, however, each learned policy is considered separate.

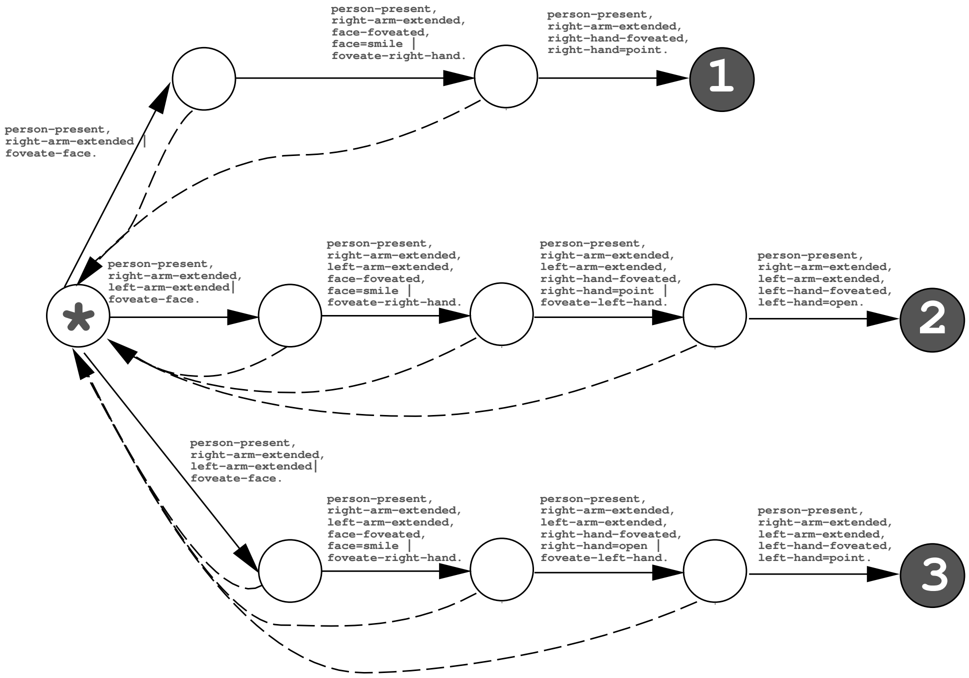

Figure 6: Initial stage of merging the three different recognition strategies presented in previous figure. Terminal nodes are marked with object / gesture identity.

have the same observation precondition, and they have the same action a_1, a_2, we delete l_2, and merge s_1 with s_2. If a_1 and a_2 are different, we set the default action from s_1 to point to s_2, and label it with the action on l_2. (If s_1 already had a labeled default action, we follow this default action link forward iteratively until we come to a state with an unlabeled default action, and then set it to point to s_2 with the action on l_2.)

Figure 5 shows three recognition state chains found for different targets in our example. Simply connecting together the initial state of these three chains yields the (invalid) aFSM shown in Figure 6. The initial state has links with common observation precondition; we remove them as described above, yielding the final (correct) aFSM recognition behavior shown in Figure 7. This aFSM will direct foveation to detect any of its three target patterns when present, and will not be confused by any other possible user configuration.

5 Discussion

The examples we have shown demonstrate the ability of hidden-state Q-learning methods, using an instance-based utility representation, to learn where to look to discriminate target from distractor patterns. Conceptually, the system constructs an action-selection mechanism which operates by considering prior experiences which had similar recent action/observation history compared to the most recent time point. Since the observation/action history capture both the state of the user and the state of the perception system (e.g. where the active camera is looking), this means that the system builds predictive models that can combine both what the user will do next, and where to look to confirm the relevant part of the hidden state.

Because our system is based on reinforcement learning, it has the ability to learn from delayed rewards. This is important, since reward is only distributed on the per-

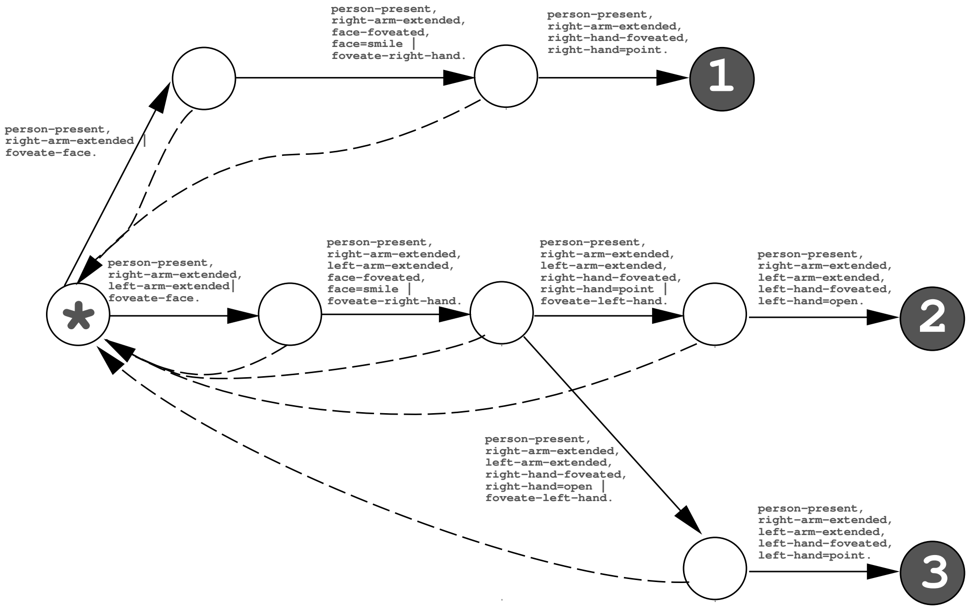

Figure 7: Final active recognition aFSM after duplicate preconditions have been eliminated from the aFSM in the previous figure, using algorithm described in Section 4.3.

formance of the recognition action, which happens at the end of a sequence of foveation actions. Learning the correct utility values corresponding to possible foveation actions during the states before reward is generated is the key problem faced by the system; were the learning system only to model the expected instantaneous reward, it could never learn foveation behavior.

The major limitation of the our system as presented lies in the use of unstructured, random search. This leads to a large amount of initial training time. We are exploring ways to structure search to focus new experience towards likely goal locations in the utility landscape. We also note that it would be straightforward to allow a teacher to give examples of correct performance and use them as starting points for learning (or to improve learning in progress); this possibility remains a topic of future work.

The use of a reinforcement/reward paradigm offers considerable flexibility; in addition to the application of active pattern recognition, one can envision a range of interaction regimes that are applicable to this learning framework. Overt actions in an interface could be included into the POMDP action set, and reward directly conditioned on high-level task performance. Much in the way that general-purpose optimization frameworks have proven powerful for scene description and structure recovery, it is our belief that reinforcement protocols hold promise for modeling a wide range of performance in interactive systems. Our results demonstrate initial progress on this path.

6 Conclusion

Gesture recognition systems often require high-resolution images of hands and faces, but the issue of how to choose which body features to observe is often left unresolved or only addressed with hard-coded solutions. In this paper we have presented a mechanism to learn to selectively foveate salient body parts in an active gesture recognition task. We adopt a Partially Observable Markov Decision Process formalism, using an action set comprised of foveation actions as well as a special recognition action. Execution of this action is rewarded based on whether the target is in the scene. We use instance-based hidden state reinforcement learning to learn a policy which models both when to execute the recognition action and what foveation commands to execute to properly discriminate the target gesture. To accurately pool experience when estimating the utility of a new action, we implement a variable-K nearest neighbor algorithm which includes all experience with a given action chain. These methods learn successful recognition policies but can consume large amounts of memory, which leads to inefficient run-time performance. To overcome this we demonstrated how to transform the learned action-selection policy representation into a concise visual behavior based on an augmented Finite State Machine representation.

References

[1] Aloimonos, Y., ed., Active Perception, Lawrence Erlbaum Associates, 1993.

[2] Bajcsy, R., Active Perception, Proceedings of the IEEE, 76(8):996-1005, August 1988

[3] Ballard, D., Reference Frames for Animate Vision, in Proc. 11th IJCAI, pp. 1635-1641, August 1989.

[4] Bandera, C., Vico, J., Bravo, J., Harmon, M., Baird, L., Residual Q-Learning Applied to Visual Attention, Proc. 13th Intl. Conf. on Machine Learning, Bari, Italy, July 1996.

[5] Barto, A. G., Bradtke, S. J., and Singh, S. P., Real-time learning and control using asynchronous dynamic programming. Computer Science Technical Report 91-57, University of Massachusetts, August 1991.

[6] Callari, F.G., Ferrie, F.P., Autonomous Recognition: Driven by Ambiguity, Proc. CVPR-96, pp. 701-707. 1996.

[7] Cassandra, A., Kaelbling, L. P., and Littman, M., Acting optimally in partially observable stochastic domains. In Proc. AAAI-94, pages 1023–1028. Morgan Kaufmann, 1994.

[8] Chrisman, L., Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In Proc. AAAI-92, pages 183–188. Morgan Kaufmann, Los Altos, California, 1992.

[9] Darrell, T., and Pentland, A., Attention-driven Expression and Gesture Analysis in an Interactive Environment, in Proc. Intl. Workshop on Automatic Face and Gesture Recognition (IWAFGR '95), Zurich, Switzerland, 1995.

[10] Darrell, T., Moghaddam, B., and Pentland, A., Active Face Tracking and Pose Estimation in an Interactive Room, Proc. CVPR-96. 1996.

[11] Darrell, T., and Pentland, A., Active Gesture Recognition using Learned Visual Attention, in D. S. Touretzky, M. Mozer, and M. Hasselmo, eds., *Advances in Neural Information Processing Systems (NIPS) 8* , MIT Press, Cambridge MA, 1996.

[12] Darrell, T., and Pentland, A., Active Gesture Recognition using Partially Observable Markov Decision Processes, in *Proc. IEEE International Conference on Pattern Recognition*, IEEE Computer Society Press, Vienna, 1996.

[13] Dickinson, S.J., Christensen, H.I., Tsotsos, J., and Olofsson, G., Active Object Recognition Integrating Attention and Viewpoint Control, Proc. ECCV-94, 1994.

[14] Jaakkola, T., Singh, S., and Jordan, M., Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems. In Advances In Neural Information Processing Systems 7, MIT Press, 1995.

[15] Kosecka, J., Christensen, H.I., Bajcsy, R., Discrete-Event Modeling of Visually Guided Behaviors, IJCV(14), No. 2, pp. 179-191, March 1995.

[16] Lin, L., and Michell, T., Reinforcement learning with hidden states. In Proc. AAAI-92. Morgan Kaufmann, 1992.

[17] Loeve, M.M., Probability Theory, Van Nostrand, Princeton, 1955.

[18] Lovejoy, W., A survey of algorithmic methods of partially observed markov decision processes. Annals of Operation Reserach, 28:47–66, 1991.

[19] McCallum., R. A., Overcoming incompleat perception with utile distinction memory. In Proceedings Tenth Machine Learning Conference. Morgan Kaufmann, 1993.

[20] McCallum, R. A., Instance-based State Identification for Reinforcement Learning. In Advances In Neural Information Processing Systems 7, MIT Press, 1995.

[21] Nakamura, N., and Asada, M.,, Motion Sketch: Acquisition of Visual Motion Guided Behaviors, in Proc. of Int. Joint Conference on Aritificial Intelligence, pp.126-132, 1995.

[22] Peng, J., Bhanu, B., Delayed Reinforcement Learning for Closed-Loop Object Recognition, Proc. Intl. Conf. Pattern Recognition '96, Vienna, Austria. 1996.

[23] Rong, S., Bhanu, B., Reinforcement Learning for Integrating Context with Clutter Models for Target Detection, Proc. ARPA IU Workshop '96, pp. 1389-1394, 1996.

[24] Sondik, E. J. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. Operations Reserach, 26(2):282–304, 1978.

[25] Sutton, R. S., Learning to predict by the method of temporal differences. Machine Learning, 3:9–44, 1988.

[26] Watkins, C., and Dayan, P., Q-learning. Machine Learning, 8:279–292, 1992.

[27] Whitehead, S., Active perception and reinforcement learning. In Proc. 7th Intl. Conf. ML, June 1990.

[28] Wren, C., Darrell, T., Starner, T., Johnston, M., Russell, K., Azarbayejani, A., and Pentland, A. pfinder: A Real-Time System for Tracking People, SPIE Conference on Real-Time Vision, M. Bove, Ed., Philadelpia, PA, July 1995.