M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 367
Appeared 13th IEEE Intl. Conference on Pattern Recognition (ICPR '96), Vienna, Austria.

Active Gesture Recognition using Partially Observable Markov Decision Processes

Trevor Darrell and Alex Pentland
Perceptual Computing Group, MIT Media Lab

Abstract

We present a foveated gesture recognition system that guides an active camera to foveate salient features based on a reinforcement learning paradigm. Using vision routines previously implemented for an interactive environment, we determine the spatial location of salient body parts of a user and guide an active camera to obtain images of gestures or expressions. A hidden-state reinforcement learning paradigm based on the Partially Observable Markov Decision Process (POMDP) is used to implement this visual attention. The attention module selects targets to foveate based on the goal of successful recognition, and uses a new multiple-model Q-learning formulation. Given a set of target and distractor gestures, our system can learn where to foveate to maximally discriminate a particular gesture.

1 Introduction

The use of vision for communication and interaction with people is an interesting new domain for computer vision. We have been developing methods which combine low-level vision and learning to implement interfaces which can directly respond to a human user via visual perception.

Previously, we presented a method for view-based recognition of spatio-temporal hand gestures [3] and a similar mechanism for the analysis/real-time tracking of facial expressions [5]. These methods offered real-time performance and a relatively high level of accuracy, but required foveated images of the object performing the gesture and were thus of limited usefulness in unconstrained domains, such as "intelligent rooms" or interactive virtual environments. In [6] and [7], we expanded our gesture recognition method to include an active component, utilizing an active image sensor that can foveate a person's hand or face at high resolution. The active camera was guided by vision routines which could track people and identify head/hand locations as they walk about a room based on coarse-scale (wide field-of-view) images from a second, static camera [4, 17].

In this paper we address the problem of where to foveate in an active recognition framework, e.g., how to plan observations given a particular recognition task. We explore a reinforcement learning approach, in which foveation actions are based on prior supervised training experience. We implement this visual attention using learning techniques specialized to deal with the hidden state problem. Using a simple reward schedule the attention system learns the appropriate object (hand, head) to foveate in order to maximize recognition performance.

2 Visual Attention for Recognition

Visual routines can be used to track the head and hands of a user, and an active camera guided to provide foveated images for gesture recognition. If we knew *a priori* which body parts to foveate to detect the gesture of interest, or if we had a sufficient number of active cameras to track all body parts, then there is no problem. Of course, in practice there are more possible loci of gesture performance than there are active cameras, and we have to address the problem of selectively observing parts of the scene, i.e., attention. We desire a method for perceptual action selection that can learn from experience and model the fact that we only have partial observations of the actual state in the world. 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 use a hidden state learning model with both action and perception: the Partially Observable Markov Decision Process (POMDP). In our system we use a POMDP formalism to define perceptual action selection in a recognition task, and solve for foveation policies using reinforcement learning methods.

We define an Active Gesture Recognition (AGR) task as follows. First we assume there is some state representation of the world describing the person configuration (or more generally, the scene configuration). Second, we assume that portions of the state of the world are only revealed via a moving fovea, and that a set of actions exist to perform that foveation. Some portion of the world state (e.g., a low-resolution view) is fully observable. The position of the camera constitutes the perceptual state of the system; we define the full "state" to be the concatenation of world state and perceptual state. Third, we assume that, in addition to actions for foveation, there is also a special action labeled `accept`, and that the execution of this action by the AGR system signifies detection of a target world state (or target sequence of states). Finally, the goal of the AGR task is to execute `accept` whenever a target pattern is present, and not to perform it when any other pattern (e.g. distractor) is present. A pattern is simply a certain world state, or sequence of world states. The AGR system should use the foveation actions to selectively reveal the hidden state needed to discriminate the target pattern.

In the interactive interface/room domain, we assume primitive routines exist to provide the continuous valued control and tracking of the different body parts that contain hidden state (i.e., are not observable in the coarse static view, only with a foveated view). We also assume that body pose and hand/face state is represented as a feature set output by our body tracker and view-based recognition system.

2.1 POMDP formalism

An important problem in applying reinforcement learning to this task is that our perceptual observations may not provide a complete description of the user's state. Indeed, because we have a foveated image sensor in our interactive interface environment we know that the true world state is hidden as the camera can only foveate on a single body part of the user at any given moment in time. 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.

The AGR task can be considered as a Partially Observable Markov Decision Process (POMDP), which is essentially a Markov Decision Process (MDP) without direct access to state [13, 11]. A POMDP consists of a set of states in the world $\mathcal{S}$, a set of observations $\mathcal{O}$, a set of actions $\mathcal{A}$, and a reward function R. After executing an action a, the likelihood of transitioning between two states s, s' is given by $T(s, a, a')$, and an observation o is generated with probability $O(s, a, o)$. In practice, T and O are not easily obtainable, and we use methods which do not require them *a priori*. Rather than attempt to solve the POMDP explicitly, we look to techniques for hidden state reinforcement learning to find a solution [12, 10, 8, 1].

Our state is defined by the users pose, facial expression, and hand configurations, expressed in nine variables Three of these are 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 world is defined by a state $s \in \mathcal{S}$, which 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 are 4 foveation commands: `look-body`, `look-head`, `look-left-hand`, and `look-right-hand` plus the special `accept` action. Each foveation command causes the active camera to follow the respective body part, and sets the internal foveation feature bits accordingly.

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 pattern is being presented to the system. The parameter α expresses the trade-off between false alarms and misses; for the results presented here we have taken a conservative approach and set $\alpha = 10$. (This is similar to the idea of disproportionally penalizing the Q-value of perceptually aliased states, in Whitehead's Lion algorithm [16].) Zero reward is given whenever a foveation action is performed.

We thus 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. Given the reward function defined in the AGR task, this will correspond to a successful recognition.

2.2 Hidden-State Learning

To find policies for AGR tasks we have implemented an instance-based method for hidden state reinforcement learning, based on earlier work by McCallum [12]. This method performs Q-learning [14, 15] (see Appendix A), but replaces the absolute state with a distributed memory-based state representation. 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, $q[t]$, and Q-learning is performed by evaluating the similarity of recently observed tuples with sequences farther back in the history chain. Q-values are computed, and the Q-learning update rule applied, maintaining this distributed, memory-based representation of Q-values.

As in traditional Q-learning, at each time step the utility of each action in the current state is evaluated. 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 in a POMDP we do not have full access to state. Using a variation on the instance-based approach employed by McCallum's Nearest Sequence Memory (NSM) algorithm, we instead find the K nearest neighbors in the history list relative to the current time point, and compute their average Q value. For each element on the history list, we compute the sequence match criteria with the current time point, $M(i,T)$, where $M(i,j) = S(i,j) + M(i-1,j-1)$ if $S(i,j) > 0$ (and $i,j > 0$), otherwise $M(i,j) = 0$. 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. Using a superscript in parentheses to denote the action index of a Q-value,

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

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

$$\pi[T] = \arg\max_{a \in \mathcal{A}} Q^{(a)}[T] \; . \qquad (2)$$

The new action $a[T+1]$ is chosen either according to this policy or based on an exploration strategy. 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, $q[T+1] = Q^{(a[T+1])}[T]$. The update step of Q learning is then computed, evaluating

$$U[T+1] = \max_{a \in \mathcal{A}} Q^{(a)}[T+1] \; , \qquad (3)$$

$$q[i] \leftarrow (1-\beta) q[i] + \beta (r[i] + \gamma U[T+1]) \; , \qquad (4)$$

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

3 Multiple Model Q-Learning

In general, we have found the simple, instance-based hidden state reinforcement learning described above to be an effective way to perform action selection for foveation when the task is recognition of a single object from a set of distractors. However, this system performed poorly when the AGR task was extended to include more than one target gesture. When multiple `accept` actions were added to enumerate the different targets, we were not able to find exploration strategies that would converge in reasonable time.

This is not unexpected, since the addition of multiple causes of positive reward makes the Q-value space considerably more complex. To remedy this problem, we propose a multiple model Q-learning system. In a multiple model approach to the AGR problem, separate learning agents model the task from each target's perspective. A separate Q-learning agent exists for each target, maintains it's own Q-value and history structure, and is coupled to the other agents via shared observations. Since we can interpret the Q-value of an individual AGR agent as a confidence value that its target is present, we can mediate among the actions predicted by the different agents by selecting the action from the agent with highest Q-value.

Formally, in our multiple model Q-learning system all agents share the same observation and selected action, but have different reward and Q-values. Thus they can be considered a single Q-learning system, with vector reward and Q-values. Our multiple model learning system is thus obtained by rewriting Eqs. (1)-(4) with vector $q[t]$ and $r[t]$. Using a subscript j to indicate the target index, we have

$$Q_j^{(a)}[T] = (1/K) \sum_{i=0}^{T} v^{(a)}[i] q_j[i] \; ; \; \pi[T] = \arg\max_{a \in \mathcal{A}} \left(\max_j Q_j^{(a)}[T] \right) . \qquad (5)$$

Rewards are computed with: if $a[T] =$ `accept` then $r_j[T] = R(j,T)$ else $r_j[T] = 0$; $R(j,T) = 1$ if gesture j was present at time T, else $R(j,T) = -\alpha$. Further,

$$U_j[T+1] = \max_{a \in \mathcal{A}} Q_j^{(a)}[T+1] \; , \qquad (6)$$

$$q_j[i] \leftarrow (1-\beta) q_j[i] + \beta (r_j[i] + \gamma U_j[T+1]) \; \forall \, i \text{ s.t. } v^{(a[T+1])}[i] = 1 \; . \qquad (7)$$

Note that our sequence match criteria, unlike that in [12], does not depend on $r[t]$; this allows considerable computational savings in the multiple model system since $v^{(a)}$ need not depend on j.

4 Experimental Results

We experimented with our recognition system using a real-time implementation connected to the perceptual outputs of our person tracking and gesture analysis systems [17, 3], as described in Section 2.1.

To train our system, we follow a three stage procedure. First, we provide the system with deterministic experience whereby the system selects actions according to to a pre-scheduled list which enumerates action chains up to a predetermined depth (typically 2, in our experiments) followed by an `accept` terminal. Each action chain is executed d (a given parameter) times for each target/distractor, with the intention being the system should have sufficient initial experience with both target and distractor to determine the true utility of each action. Throughout the training phase, we randomly switch between target and distractors,

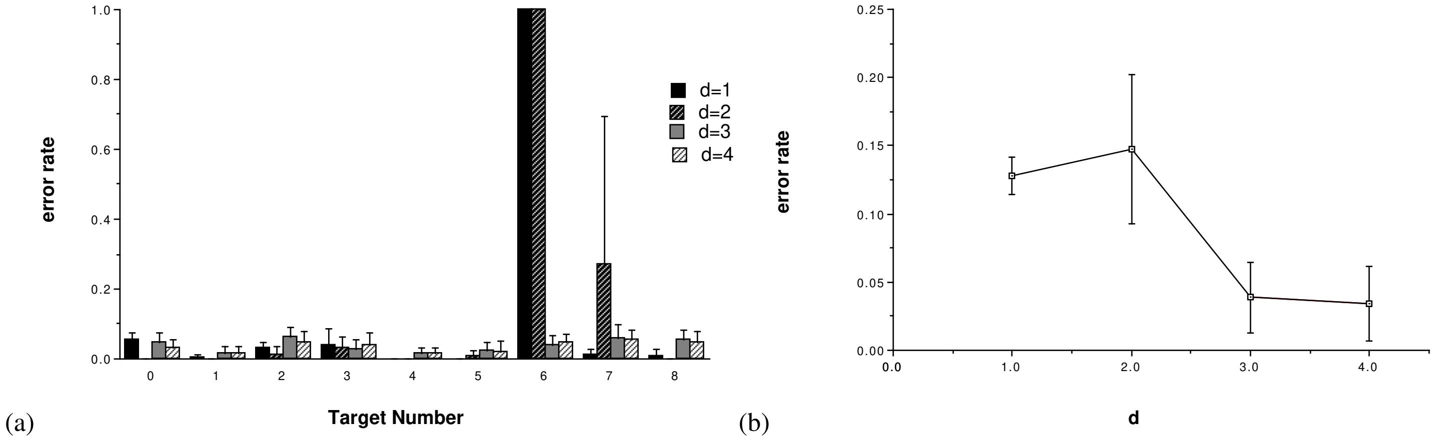

Figure 1: Results of multiple model recognition experiment for increasing amounts of initial training (d=1..4): (a) for each target, and (b) averaged across all targets

so that the utility of actions will be averaged over trials with each. Second, we run a batch version of the Q-learning update rule until the utility values converge. We cycle through the memory structure, updating the utility of each q value as given in Eq. 4. When the cumulative change is less than a fixed value or stops decreasing, we consider the utility values to have converged. Finally, we let the system run according to policy, (i.e. select the action with maximal Q-value) and evaluate the recognition performance. During testing we randomly switch between target and distractors, and record trials when the system executes `accept` on a distractor, or fails to execute `accept` on a target. These trials are marked as errors; all other trials are considered correct. When the time-averaged performance rate stops increasing, we freeze all utility values, re-measure recognition performance, and record this performance measure as the final value.

We collected example output from a user interacting with the perceptual systems, where the user defined nine different target patterns. The first two patterns set the `left-arm-extended`, `left-hand-point`, and `left-arm-extended`, `left-hand-open` bits, respectively. (All patterns also included the `person-present` bit.) The next two patterns were the mirror image on the right side. Two more targets (4,5) had no arms extended, and were differentiated by `face-smile` or `face-surprise`. The last three patterns had both arms extended; target 6 had `left-hand-open` and `right-hand-open`, target 7 `left-hand-open` and `right-hand-point`, and target 8 `left-hand-point` and `right-hand-open`.

We ran off-line experiments to test the recognition performance of the learning system under different amounts of initial experience. (We kept constant the parameters $\beta = 0.1, \gamma = 0.5, \alpha = 10$.) Figure 1 shows results for each target and cumulative across all targets, plotted for different amounts of initial exploration experience according to the parameter d. This parameter represents the average number of times the system initially explored a given action chain, divided by the number of targets and distractors.

Targets which are easily discriminable (e.g., 0-5) have good performance even with little initial training, but the more complicated targets (6,7,8) require considerable amounts of initial training ($d > 2$) to obtain adequate performance. This is due to the fact that the policies formed for each Q-learning module for the last three targets "overlap" in the sense that they call for different actions in the same perceptual condition. With a deterministic policy, when there are competing (different) actions with substantial utility in different modules the system will pick one action and not the other, causing the second policy to not be executed (and thus the corresponding target not recognized).

However, the system does learn to recognize these targets, with enough experience. With a strict deterministic policy, it is still possible for the system to learn to act differently after it tries an action, fails to recognize the target, and resets to the initial condition (when the maximum utility falls below zero the system resets). The variable length match voting scheme described above allows the length of an history chain matching to compute utility to vary for different actions. Thus, after resetting, the utility of repeating the action taken prior to resetting will ideally be lower. The system can infer this from previous experience with repeating the action twice, if this appropriate prior experience is available.

The system will naturally experience the effect of multiply repeating the particular action with an intervening reset, since this is the "pathological" behavior described above. The utility of the alternate (overlapped) action will remain the same, since it will still be matching against the same experience after the reset as it did at the first timestep with the given trial. The second action will this get chosen after the system explores the first action and resets; the "prior experience" matched to compute the utility of the first action will then consist of prior trials where the system tried a dual repeat of the first action.

In this way, deterministic multiple-model policies can overcome the action overlap problem. However, in practice it can take a considerable amount of time to learn this solution strategy. We are currently investigating an alternative means of dealing with this problem, using an algorithm for persistent and exclusive multiple-model policies. This latter algorithm guarantees the policy in each Q module will fully execute (persistence), and prevents a module which had control and failed to find non-zero reward to regain control until either all modules have had a chance to execute or the world state changes (exclusivity). Our initial results show that multiple-model policies with action overlap can be learned much quicker with persistent and exclusive Q-learning.

Appendix A: Q-learning

Given absolute state, recovering an optimal policy for a Markov Decision Process (MDP) is a well-studied problem. One wishes to find an action-selection policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which if followed will maximize the *expected discounted future reward*,

$$Z = E\left[\sum_{t=t_0}^{\infty} \gamma^{t-t_0} r[t]\right] ,$$

where γ is the discount factor. Q-Learning is one widely used method based on a reinforcement learning paradigm for finding an optimal 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 Q values can be computed on-line using a *Temporal Differences* method [14], which is an incremental form of a full Dynamic Programming approach [2]. With of deterministic state transitions, the optimal Q function must satisfy

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

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 this equation is minimized:

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

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 [15].

In the case of instance-based POMDP reinforcement learning, we do not use a table representation, i.e. $Q(s, a)$. Time plays the role of state, in essence, and so we store a Q function over time, i.e. $Q[T]$. At each time step we compute and compare (but do not store) the Q-value of several different actions, $Q^{(a)}[T]$. Finally, in the multiple-model formulation we have separate Q-values for each target pattern, and thus the notation $Q_j^{(a)}[T]$.

References

[1] A. Cassandra, L. P. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proc. AAAI-94*, 1994.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh. Real-time learning and control using asynchronous dynamic programming. C.S. T.R. 91-57, U. Mass, 1991.

[3] T., Darrell, and A., Pentland, Space-Time Gestures. *Proceedings IEEE CVPR-93*, New York, IEEE Comp. Soc. Press, 1993.

[4] T. Darrell, P. Maes, B. Blumberg, and A. P. Pentland, A Novel Environment for Situated Vision and Behavior, *Proc. IEEE Workshop for Visual Behaviors*, IEEE Comp. Soc. Press, 1994

[5] T. Darrell, I. Essa, and A. P. Pentland, Correlation and Interpolation Networks for Real-time Expression Analysis/Synthesis, In *Advances NIPS 7*, MIT Press, 1995.

[6] T. Darrell and A. Pentland, A., Attention-driven Expression and Gesture Analysis in an Interactive Environment, in *Proc. IWAFGR '95*, Zurich, Switzerland, 1995.

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

[8] T. Jaakkola, S. Singh, and M. Jordan. Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems. In *Advances NIPS 7*, MIT Press, 1995.

[9] T. Poggio and F. Girosi, A Theory of Networks for Approximation and Learning. MIT AI Lab TR-1140, 1989.

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

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

[12] R. A. McCallum. Instance-based State Identification for Reinforcement Learning. In *Advances NIPS 7*, MIT Press, 1995.

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

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

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

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

[17] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, Pfinder: Real-Time Tracking of the Human Body, MIT Media Lab Perceptual Computing Group Technical Report No. 353, 1994