2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Anticipatory Robot Control for a Partially Observable Environment Using Episodic Memories

Yoichiro Endo

Abstract—**This paper explains an episodic-memory-based approach for computing anticipatory robot behavior in a partially observable environment. Inspired by biological findings on the mammalian hippocampus, here, episodic memories retain a sequence of experienced observation, behavior, and reward. Incorporating multiple machine learning methods, this approach attempts to help reducing the computational burden of a partially observable Markov decision process (POMDP) problem. In particular, proposed computational reduction techniques include: 1) abstraction of the state space via temporal difference learning; 2) abstraction of the action space by utilizing motor schemata; 3) narrowing down the state space in terms of goals through instance-based learning; 4) elimination of the value-iteration by assuming a unidirectional-linear-chaining formation of the state space; 5) reduction of the state-estimate computation by exploiting the property of the Poisson distribution; and 6) trimming the history length by imposing a cap on the number of episodes that are computed. Claims 5) and 6) were empirically verified, and it was confirmed that the state estimation can be in fact computed in an *$O(n)$* time (where *n* is the number of the states), more efficient than a conventional Kalman-filter based approach of *$O(n^2)$*.**

I. Introduction

As the robotic technologies keep advancing and start interweaving into our lifestyle, it is inevitable that some robots will be soon required to make instant decisions in life-or-death situations for humans. For example, the robots deployed in the domains such as military [1, 2], nursing [3, 4], and search-and-rescue [5, 6] are the obvious candidates. These robots will have to behave in an anticipatory manner. In other words, they have to be able to assess a current situation, predict the future consequence of the situation, and execute an action to have desired outcome based on the assessment and the prediction. For humans, such critical decisions are made by experts, relying on their experiences. Similarly, for robots, the premise here is that experience does matter as well. The question is then how to store the experience into the robot's memory and, when necessary, utilize it without delay.

We have previously investigated an anticipatory robot navigation method [7], in which a robot constructs a cognitive map while simultaneously localizing itself relative to it. The cognitive map consists of episodic memories. Inspired by the notion proposed by neuroscientists [8], the basic idea of episodic memory is that it stores a temporal sequence of events where each event consists of sensory and behavioral information. While retaining the core concept, in this paper, we extend the episodic memory from a mere "map" notion into a framework to solve partially observable Markov decision process (POMDP) problems efficiently.

The objective of an MDP problem is to find the best action for the current state that maximizes expected rewards. While solving a standard (stochastic) MDP problem itself suffers from a computational complexity as the state space broadens, solving a POMDP problem is known for its severe computational burden because a current state cannot be assessed directly and therefore has to be estimated first. Unfortunately, when dealing with real robots, the assumption of complete observability cannot be guaranteed because various types of uncertainties influence the robot's state. Hence, the challenge for the robotics researchers has been to find a computationally tractable solution while working in a partially observable environment.

II. Related Work

A standard approach for solving POMDP problems is to apply Bayes' rule. Most notably, Cassandra et al. [9] laid out one of the first Bayesian-based frameworks for the artificial intelligence community. In robotics, Koenig and Simmons [10] have developed a computational architecture for robot navigation that incorporates POMDP. Representing the environment with a topological map, in their method, the optimal policies were refined (offline) through the Baum-Welch algorithm.

Various attempts have been made to reduce the computational load associated with the POMDP computation. One way to accomplish such reduction is to represent the state space hierarchically. For example, in Theocharous and Mahadevan's approach [11], the state space was abstracted based on spatial granularities. Through their experiment using a real robot, the hierarchical dissection of the state space was proven effective especially when covering a large area. Likewise, Pineau et al. [12] tackled a POMDP problem by decomposing the action space hierarchically. The application of their method on a real robot in nursing homes has successfully provided necessary assistances to the elderly residents. It should be noted that our method presented here also utilizes the notion of abstract action (behavior) that is composed with lower level motor schemata (which themselves can be represented

Y. Endo is with Georgia Tech Mobile Robot Laboratory, Atlanta, GA 30332 USA (corresponding author to provide phone: 404-385-7139; fax: 404-894-0673; e-mail: endo@cc.gatech.edu).

978-1-4244-1647-9/08/$25.00 ©2008 IEEE.

hierarchically).

Another way to reduce the state space is via sampling. Thrun [13] has demonstrated that Monte Carlo sampling over belief space can attain solutions that are near optimal. On the other hand, instead of sampling based on the belief distribution, Pineau et al. [14] proposed a sampling method based on the shape of the value function. More specifically, a finite set of sampling points is selected, which is enough to recover the value function through a piecewise linear function. For each computational cycle, a new set of sampling points is selected by stochastically simulating the trajectory of the previous points; hence, the old points are thrown away if found irrelevant (i.e., trimming the history length). Correspondingly, our episodic-memory-based method can be viewed as a form of trajectory sampling [15]; instead of exhausting computational effort on sweeping the entire state space, state parameters are updated only for those residing along the trajectory of performing a task.

There is also an alternative to the Bayesian-based approach for solving POMDP problems. McCallum [16] applied an instance-based learning method to estimate the current state. More specifically, from its memory, the robot retrieves the *k* nearest neighboring states that best represent the current state based on the current sequence of action, perception, and reward. The state parameter (Q-value), which is used to obtain an optimal policy, is determined by averaging the values from those *k* states. Ram and Santamaria [17] also took a similar approach to identify the current state in the context of continuous case-based reasoning. In their method, however, the retrieved case was used to directly alter behavioral parameters in order to obtain desired behavioral effects. Our method presented here also utilizes the instance-based learning. However, instead of directly identifying the current state, it was employed to help narrow down the state space based on the goals.

III. Anticipatory Robot Control

The diagram in Figure 1 shows our proposed computational steps that integrate multiple machine learning methods to compute anticipatory behavior for a robot. While we have proposed in [18] that these steps can be employed to compute improvisational behavior as well, in this paper, we will limit our discussion to the anticipatory behavior aspect only.

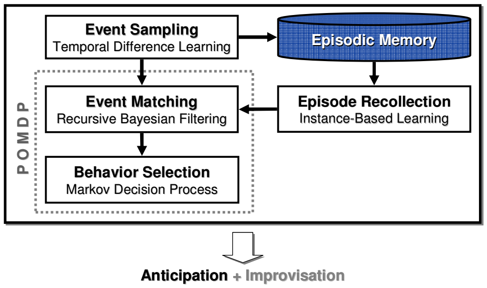


Figure 1: Computational steps for the anticipatory robot control

A. Episodic Memory

Our computational steps utilize episodic memory, whose biological inspiration comes from the mammalian hippocampus [7]. As shown in Equation 1, an episodic memory (*E*) consists of a temporal sequence of events (*e*), where *n* is the number of events in the episode, and a goal (*g*), which the robot was pursuing during the episode:

$$E = \{(e_1, e_2, ..., e_n), g\} \tag{1}$$

The event can be considered as a snapshot of the world at a certain instance during the episode. More specifically, the event consists of a set of observation (*o*), behavior (*b*), and reward (*r*):

$$e = \{o, b, r\} \tag{2}$$

o (observation) is an *m*-length vector of sensor readings (*z*) where *m* is the number of sensors that the robot is integrated with:

$$o = \{z_1, z_2, \ldots, z_m\} \tag{3}$$

b (behavior) is defined as a set of motor schemata [19] (σ) that are instantiated at the instance:

$$b = \{\sigma_1, \sigma_2, \ldots, \sigma_\beta\} \tag{4}$$

r (reward) is a value of the reward signal at the instance, which is modulated by a separate function (Subsection III.C). Finally, *g* (goal) is a particular perceptual state that the robot was attempting to reach during the episode (Subsection III.C). As for the observation above, the goal is denoted with the *m*-length vector of sensor readings:

$$g = \{z_1^g, z_2^g \ldots, z_m^g\} \tag{5}$$

Note that episodes are partitioned based on goals. In other words, a new episode starts when the robot starts pursuing a new goal and ends when the robot stops pursuing it. Hence, the number of events in each episode varies depending on how long the particular goal was pursued by the robot.

B. Anticipatory Behavior Computation

As proposed in [18], anticipatory robot behavior is computed by the following four steps: *event sampling*, *episode recollection*, *event matching*, and *behavior selection* (Figure 1). Each step employs a different machine learning method: namely, temporal difference learning, instance-based learning, recursive Bayesian filtering, and MDP, respectively. Note that the combination of the recursive Bayesian filtering and MDP is used to provide the solution for the POMDP problem in our case.

1) Event Sampling: The goal of event sampling is to construct a model of the world in terms of episodic memories (Equation 1). More specifically, given a continuous stream of sensor readings, discrete states are temporally abstracted in this step. In order to abstract an event from the input data stream, a simple (model-free) reinforcement learner, namely, *TD*(λ) [20], is used. In this case, the sole purpose of the learner is to predict the current observation based on the previous observation as fast as possible. The assumption here is that, at the instance when the learner fails to predict the observation, the robot must be entering a new state;

hence the state parameters are remembered. The observation is learned at the individual sensor reading (z) level. At an instance t, based on the previous sensor reading (z_{t-1}), each current sensor reading is predicted by a simple linear equation*:

$$z'_t = w_t \; z_{t-1} \tag{6}$$

where w is a weight. At each time cycle, w is updated with the $TD(\lambda)$ update rule [20]:

$$w_{t+1} = w_t + \alpha(z_t - z'_t)\sum_{k=1}^{t} \lambda^{t-k} \nabla z'_k \tag{7}$$

where α is a learning rate, λ^k is an exponential weighting factor (eligibility trace), and the gradient $\nabla z'_k$ is a partial derivative of z'_k with respect to the weights†.

The error of the prediction is monitored at each time cycle in order to decide when to sample an event. The error is measured in terms of a *root-mean-squared* (RMS) difference of the predicted and actual observations. If the error at t is larger than the one before and after, a new event is sampled (Equation 8):

$$f_{\text{sample}}(t) = \begin{cases} \text{true} & \text{if } f_{\text{rms}}(o'_t - o_t) \ge f_{\text{rms}}(o'_{t-1} - o_{t-1}) \text{ and} \\ & f_{\text{rms}}(o'_t - o_t) > f_{\text{rms}}(o'_{t+1} - o_{t+1}) \\ \text{false} & \text{otherwise} \end{cases} \tag{8}$$

where f_{rms} is a function that returns a RMS of a vector. Figure 2 shows the error between predicted and actual observations when a simulated robot (integrated with sonar sensors) proceeds along a corridor of a typical office building. Each tip of the spikes represents the occurrence of an event, and, it shows how events are clustered around salient features of the environment such as open doors and a corridor junction.

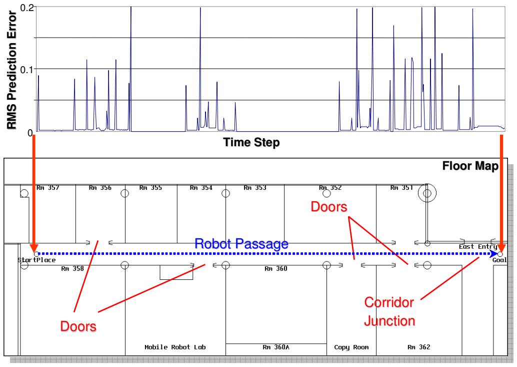


Figure 2: Comparison between the prediction errors and the passage of the (simulated) robot.

2) Episode Recollection: One way to compute the best behavior is to consider all the episodes collected by the robot to find the best policy (as we did in [7]). However, as the number of episodes increases, the computational power that is necessary to process all of them also increases. In this step, in order to allocate the computational power to only those relevant to the current situation, the episodes are preprocessed, and irrelevant episodes are filtered out by an instance-based learning method.

The core of instance-based learning algorithms is a set of similarity and classification functions [21]. Taking the current goal (g_{cur}) as a query point, our similarity function is implemented with a Gaussian-based likelihood function:

$$\rho_E = f_{\text{L}}(g_{\text{cur}}, g_{[E]}) \tag{9}$$

The likelihood function (f_{L}) returns the similarity value (ρ_E) in terms of the likelihood of a sample (the first input parameter) given a measurement (the second input parameter). In this case, we examine the similarities between the current goal (g_{cur}) and the goal of the querying episode ($g_{[E]}$) that we wish to evaluate.

Once the similarity is computed, the classification function determines whether the episode is relevant to the current goal or not. More specifically, for any episode that is in the robot's memory (C), if ρ_E of the episode is above a predefined threshold (θ_ρ), the episode will be classified as *relevant* and added to the collection of relevant episodes (M_{Rel}). Note that, in order to reduce the computation time in the event matching step (below), the size of M_{Rel} is restricted to a predefined number, K. In other words, the K latest episodes that meet the similarity condition are selected:

$$M_{\text{rel}} = \{E_{1:K} \mid \{E_{1:K}\} \subseteq C \wedge \rho_{E_{1:K}} \ge \theta_\rho\} \tag{10}$$

The effect of K with respect to the computation time of the event matching step is reported in Section IV.

3) Event Matching: This step is invoked whenever a new event is captured by the event sampler. It is an equivalent of the state estimation process in POMDP. From a collection of relevant episodes computed above, events that best represent the current state are determined by a recursive Bayesian filter, the probabilistic method commonly used for solving the simultaneous localization and mapping (SLAM) problem [22]. At first, for each relevant episode, the posterior probabilities (belief) of being at some event (e_q) in the episode given the history of the observations (o^τ) and executed behaviors (b^τ) are solved by the following recursive equation‡:

$$p(e_q | o^\tau, b^\tau) = \eta \, p(o_\tau | e_q) \sum_{e_{\tau-1} \in E} p(e_q | b_\tau, e_{\tau-1}) \, p(e_{\tau-1} | o^{\tau-1}, b^{\tau-1}) \tag{11}$$

where η is a normalization factor, $p(o_\tau \mid e_q)$ is a sensor model, $p(e_q | b_\tau, e_{\tau-1})$ is a motion model, and $p(e_{\tau-1} | o^{\tau-1}, b^{\tau-1})$ is the belief of the previous computational cycle.

To implement the sensor model, which is the conditional probability of observing o_τ given the query event (e_q), we employ the same Gaussian-based likelihood function used in Equation 9. More specifically, the similarity (ρ_{sensor}) of the current observation (o_τ) and the ones recorded in the querying event ($o_{[e_q]}$) is computed as our sensor model:

$$p(o_\tau \mid e_q) = \rho_{sensor} = f_{\text{L}}(o_\tau, o_{[e_q]}) \tag{12}$$

The motion model is the transition probability of the robot arriving at the target event (e_q) if the previous event is $e_{\tau-1}$

[*] The subscript denotes a temporal index, different from the subscript used to denote the integrated sensor types in Equation 3.

[†] $\nabla z'_k$ is z_{k-1} because of Equation 6.

[‡] See [7] for derivation.

and behavior b_τ is executed. In the certainty equivalence approach [23], the transition probabilities may be estimated by taking the statistic of the transitions while exploring the environment [24]. On the other hand, in our episodic-memory-based approach, since events are formed in a unidirectional linear chain (Equation 1), the motion model can be computed in terms of how many events the robot needs to advance in order to reach e_q from $e_{\tau-1}$. Let e_j be $e_{\tau-1}$, Equation 13 shows our implementation of the motion model:

$$p(e_q \mid b_\tau, e_j) = \begin{cases} f_{\mathrm{P}}(e_j, e_q) + \varepsilon_m & \text{if } q > j \text{ and } b_q = b_\tau \\ \kappa_m \, f_{\mathrm{P}}(e_j, e_q) + \varepsilon_m & \text{else if } q > j \text{ and } b_q \neq b_\tau \\ \varepsilon_m & \text{otherwise} \end{cases} \tag{13}$$

where ε_m is some small number to ensure that the probability does not become absolutely zero, κ_m is a discount factor, and f_{P} is a function that returns the probability of the robot reaching e_q from e_j based on the Poisson distribution. Let us define $d_{j:q}$ to denote the distance between e_j and e_q in terms of event numbers and $\bar{d}$ to denote the average number of events that the robot advances within one computational cycle. The Poisson-based function is then implemented as:

$$f_{\mathrm{P}}(e_j, e_q) = \mathrm{Poisson}(d_{j:q}, \bar{d}) = \frac{\exp(-\bar{d})\, \bar{d}^{(d_{j:q})}}{d_{j:q}!} \tag{14}$$

In other words, the motion model is the probability from the Poisson distribution if the index of e_q is greater than the index of $e_{\tau-1}$ and b_τ is the same behavior that is stored in e_q. If the behaviors mismatch, the probability is discounted. Since the posterior probabilities are computed whenever the event sampling step captures a new event, the value of $\bar{d}$ is assumed to be 1.0. Note that, as shown in Figure 3, the probability of this Poisson distribution becomes near-zero when the distance from e_j to e_q becomes 6. This property can be in fact exploited to reduce the computational burden of the event matching (state estimate) step for each episode from $O(n^2)$ to $O(n)$ by computing the motion model (Equation 11) for only 5 events (instead of n events). The empirical result of this optimization is reported in Section IV.

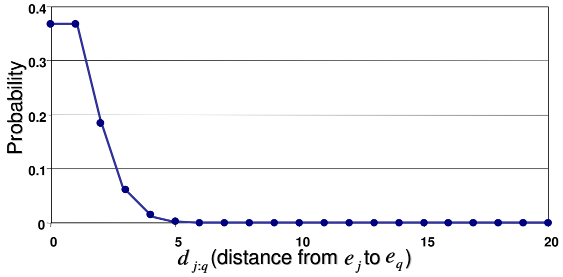


Figure 3: The probability mass function (Equation 14) for the Poisson distribution ($\bar{d}$ = 1.0).

After the posterior probabilities for all of the events in the episode are computed, the one with the highest probability is considered as the the event that best represents the current state. However, it is possible that the current state is novel, and none of the events could correspond to the current state. Hence, we introduce an assumption here that, if the posterior probability distribution is spread evenly around the average value rather than having a distinct peak, the current state is considered to be novel. One way to check such novelty is to compare the highest probability value with a predefined threshold as we did in [7]. Another approach is, as suggested by Tomatis et al. [25], to use the entropy of the posterior probability distribution. More specifically, the entropy (H) of the posterior probability distribution for an episode (E) is computed by:

$$H(E) = -\sum_{e_i \in E} p(e_i \mid o^\tau, b^\tau) \log_2 p(e_i \mid o^\tau, b^\tau) \tag{15}$$

Having a high entropy value infers that the probability distribution is close to uniform. Thus, only if $H(E)$ is below the predefined threshold, the event with the highest posterior probability in the episode is considered to be *matched* ($\hat{e}_{[E]}$) to the current state:

$$\hat{e}_{[E]} = \begin{cases} \underset{e_i \in E}{\mathrm{argmax}}\; p(e_i \mid o^\tau, b^\tau) & \text{if } H(E) \leq \theta_H \\ \varnothing & \text{otherwise} \end{cases} \tag{16}$$

Note that, if the previous step (episode recollection) yields k episodes as relevant, there will be at most k matched events. Here, the set of all relevant episodes that contain valid matched events is denoted as $\hat{M}_{\mathrm{rel}}$:

$$\hat{M}_{\mathrm{rel}} = \{\forall E \mid E \in M_{\mathrm{rel}} \wedge H(E) \leq \theta_H\} \tag{17}$$

4) Behavior Selection: Based on the matched events found in the above step, the most appropriate behavior for anticipation will be selected in this step. At first, the utility (U) of each event is computed using a Bellman equation:

$$U(e_i) = r_i - \sum_{e' \in E} p(e' \mid b_{i+1}, e_i)\, U(e') \tag{18}$$

where r_i is the reward value stored in e_i. Note that $p(e' \mid b_{i+1}, e_i)$ is the same transition probability computed for the motion model (Equation 13). Generally, in MDP problems, the Bellman equation has to be iterated for a number of times to obtain converged utility values (value iteration). On the other hand, in our case, because events are formed in a unidirectional linear chain[§], from the end event to the start event, the utility value can be computed by a recursive (dynamic programming) fashion but without any iteration.

Next, we define a new function, $\Gamma^+(b)$, which returns a subset of episodes from $\hat{M}_{\mathrm{rel}}$. More specifically, given a behavior (b), Γ^+ returns a special case of episodes in $\hat{M}_{\mathrm{rel}}$ where the localized events ($\hat{e}_{[E]}$) in these episodes are followed by events storing b:

$$\Gamma^+(b) = \left\{ \forall E \;\middle|\; \begin{array}{c} E \in \hat{M}_{\mathrm{rel}} \wedge \{e_i, e_{i+1}\} \subseteq E \wedge \\ e_i = \hat{e}_{[E]} \wedge b \in e_{i+1} \end{array} \right\} \tag{19}$$

Finally, based on the utility values and $\Gamma^+(b)$, we select the

[§] ε_m in the transition probability (Equation 13) is zero in this case.

best behavior (b^*) by a maximization function:

$$b^* = \arg\max_b \frac{1}{|\Gamma^+(b)|} \sum_{E\in\Gamma^+(b)} \sum_{e'\in E} p(e' \mid b, \hat{e}_{[E]})\, U(e') \qquad (20)$$

where $p(e'|b_{i+1}, e_i)$ is the same transition probability used in Equations 13 and 18. Note that Equation 20 is equivalent of how an optimal policy is computed in a standard MDP problem. However, while a standard MDP assumes only one state that is representing the current state, in our case, as much as the number of episodes returned by $\Gamma^+(b)$ there are events that represent the current state. Hence, the expected utility of executing b is averaged over the number of those events.

C. *Goal and Reward*

As mentioned above, episodes are partitioned based on the goals, and the goals are used as the keys to retrieve relevant episodes from the memory at the episode recollection step. Let G be a set of all possible goals. A unique goal (g_{cur}) for the current instance is chosen by the robot based on a motivation function:

$$g_{\text{cur}} = \arg\max_{g\in G} f_{\text{motiv}}(g, o, \mu) \qquad (21)$$

where f_{motiv} is a motivation function that returns the degree of motivation for pursuing a particular goal (g) given the current observation (o) and the internal state (μ). The use of motivation has been exploited by many robotics researchers, especially in behavior-based robotics [26-30]. In those cases, motivation influences behaviors directly by adjusting behavior parameters such as the activation level. On the other hand, in our case, motivation influences behaviors by setting a goal, and the goal influences behaviors by recalling right episodes. It should be noted, however, that our implementation of f_{motiv} is still preliminary at this point.

Furthermore, based on the goal, the robot modulates a single reward signal. Being saved in each event, the reward signal influences the choice of behaviors by providing their utilities. In our implementation, the reward signal is determined by three factors: 1) the similarity between the current goal and the current observation; 2) the similarity between the predicted observation ($o''_{[E]}$) and the actual observation; and 3) the innate rewarding states (ω) and the current observation. These similarities are computed by the same likelihood function (f_{L}) used in Equations 9 and 12, and they are weighted by predefined constants (κ_g, κ_o, and κ_ω):

$$r_{\text{cur}} = \kappa_g f_{\text{L}}(g_{\text{cur}}, o) + \kappa_o \max_{E\in\Gamma^+(b^*)} f_{\text{L}}(o''_{[E]}, o) + \sum_{\omega\in\Omega} \kappa_\omega f_{\text{L}}(\omega, o) \qquad (22)$$

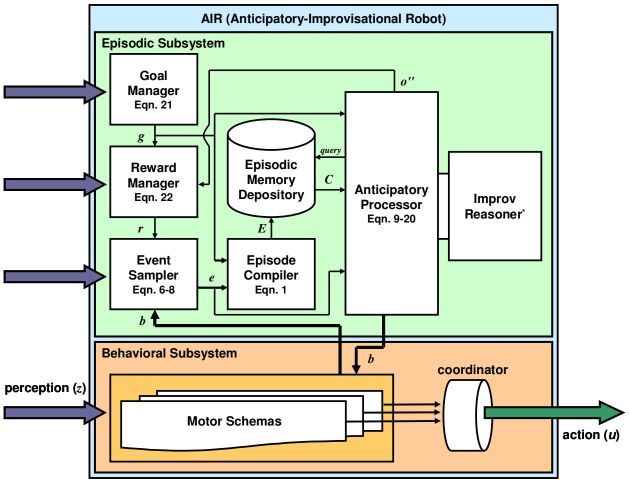
Note that, here, the predicted observation is not the same observation predicted by $TD(\lambda)$ above (Equation 6); in this case, the observation is predicted based on the matched events obtained by Equation 16. More specifically, given an episode (E), $o''_{[E]}$ is the observation stored in the event right after the matched event ($\hat{e}_{[E]}$):

$$o''_{[E]} = \{ o_i \mid \{e_i, e_{i-1}\} \subseteq E \wedge o_i \in e_i \wedge e_{i-1} = \hat{e}_{[E]} \} \qquad (23)$$

The innate rewarding states are particular perceptual states that are inherently important for the robot. For example, a voltage meter indicating that the battery is fully charged may be one of the innate rewarding states. The importance of such states is appropriately weighted by corresponding weights (κ_ω). Note that κ_ω can have a positive or negative value. For example, a reading from a tactile sensor indicating that the robot is violently hitting some object can be considered as a negative innate rewarding state.

IV. Optimization and Empirical Results

One of the most computationally expensive parts of a Bayesian-based POMDP approach is the state estimation (*event matching* in our case). Given n states (events), it requires an $O(n^2)$ computation time to compute the full posterior probabilities by the recursive Bayesian filter because the transition probability (motion model) has to be computed n times for each of the n states (Equation 11). Incidentally, localization using a Kalman filter (also Bayesian) requires an $O(n^2)$ computation time [31]. If implemented naively, the event matching step of our computational method proposed here requires $O(kn^2)$ where k is the number of relevant episodes (Equation 10) and n is the number of the events in each episode. However, by imposing k to be a constant and assuming the transition probability to be from the Poisson distribution, event matching can be done in an $O(n)$ time. The following experimental results verify the claim.



* To be implemented

Figure 4: AIR Architecture

A. *Implementation*

The anticipatory behavior computational method proposed in Section III was implemented within a two-layer architectural framework, AIR (Figure 4), consisting of the episodic subsystem (deliberative layer) and the behavior subsystem (reactive layer). The episodic subsystem takes the current sensor readings, identifies the current goal, modulates the reward value, samples events, compiles episodes, saves/retrieves the episodes, and computes the anticipatory behavior. The behavior subsystem retains the repertory of motor schemata and executes the ones specified by the episodic subsystem.

AIR (executed as a Java program) interacts with the environment simulated in Gazebo [32] (a high fidelity 3D simulator developed by University of Southern California). More specifically, AIR receives sensor readings of ActiveMedia Pioneer 2 DX emulated in Gazebo and sends back the control commands. The sensor readings and the motor commands are relayed by HServer [33], which communicates with AIR (running on Dell Latitude X200 with Pentium III; 933 MHz) and Gazebo (running on Dell Dimension 4700 with Pentium 4; 3.00 GHz) through the shared memory and a socket connection, respectively (Figure 5).

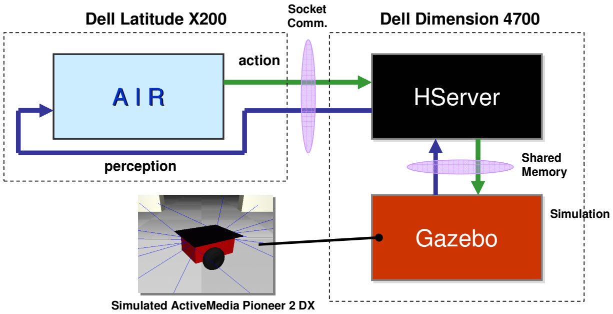


Figure 5: Communications among AIR, HServer, and Gazebo

B. *Limited Transitions vs. Full Transitions*

As mentioned above, since the events in an episode are formed in a unidirectional linear chain, our claim here is that, if we exploit the property of the Poisson distribution, the event matching of each episode can be computed in an $O(n)$ time. In this experiment, we compared the cases between computing the event matching steps when the property of the Poisson distribution was exploited (limited transitions) and not exploited (full transitions). For the limited-transitions case, the motion model in Equation 11 was computed for only 5 relevant events.

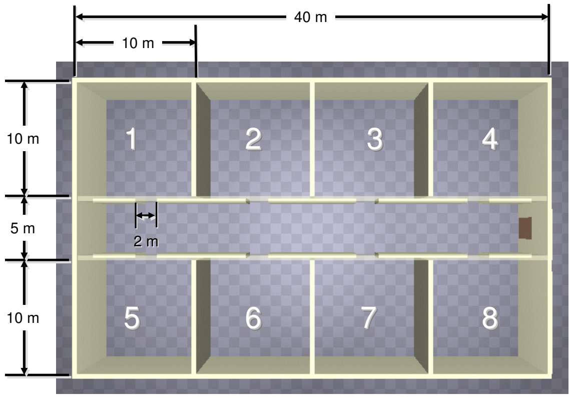


Figure 6: The experimental indoor environment simulated in Gazebo

The average computation time for the event matching was recorded while the Pioneer 2 DX robot, autonomously driven by AIR, navigated the hallway in a simulated indoor environment (Figure 6). The robot was equipped with 16 sonar sensors and 16 bumper sensors. Note that no odometry information was ever used. AIR computed the anticipatory behavior based on a single training episode stored in the memory. The training episode was constructed by manually instantiating a combination of AvoidObstacle, MoveForward, and SwirlObstacle schemata and assigning a reward at the end of the episode. For each case, the size of the training episode in the memory was varied from 20 events to 200 events with the increment of 10 events (i.e., 19 different sizes). For each condition, the testing was lasted 10 event-matching cycles, and it was repeated 20 times. Hence, the computation time of the each data point was averaged over 200 measurements.

The result, the average event matching computation time of each condition with respect to the number of the events in the episode, is plotted in Figure 7. Expectedly, when all of the possible transitions were taken into account upon computing the motion model, the computation time increased quadratically with respect to the number of events. When the computation was broken down to the sensor model and motion model parts, the motion model computation did indeed exhibit the quadratic increase while the increase of the sensor model computation remained linear. On the other hand, in the limited-transitions case, the overall event matching time was increased only linearly with respect to the number of events, consistent with the $O(n)$ claim. The computations for both sensor and motion models were evidently also linear.

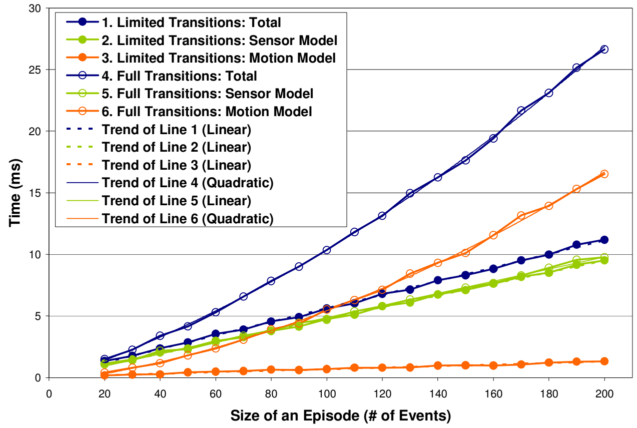


Figure 7: The average computation time required for the event matching step with respect to the size of the episode (the size of the history is fixed)

C. *Limited History vs. Full History*

One of the main differences between the conventional POMDP approaches and our method here is that, in our method, there could be multiple events that are considered to be the current states. If there are k relevant episodes retrieved by the episode collection step (Equation 10), the posterior probabilities have to be computed for at most the k episodes. Naturally, if the robot increases the experience, the k value also increases. As mentioned above, our hypothesis here is that we can impose a cap on the number episodes that are considered to be relevant without compromising the quality of the performance. To test this hypothesis, two cases, the event matching with an imposed cap on the number of the relevant episodes (limited history) and without imposing the cap (full history) were evaluated. For the limited-history case, the latest 5 episodes that meet the goal condition (Equation 9) were selected.

The experiment was conducted in the same indoor environment as the previous experiment using the same robot and the sensor configuration. During the training, the robot was dispatched from Room 8 (see Figure 6), the combination of AvoidObstacle, EnterOpening, MoveForward, SwirlObstacle, TurnLeft, and TurnRight schemata were manually instantiated in order to navigate the robot into Room 2 via the hallway. The robot received a reward upon arriving Room 2. For each case, there were initially 5 training episodes in the memory, and the size of the history were accumulated up to 15 episodes during the testing. Each testing was repeated four times. To reach Room 2, each run generally required over 300 event-matching cycles; hence the event matching computation time for each condition was averaged over more than 1200 measurements. Furthermore, the quality of the performance was measured in terms of the total distance the robot traveled (path length) and the time the robot took to reach the goal.

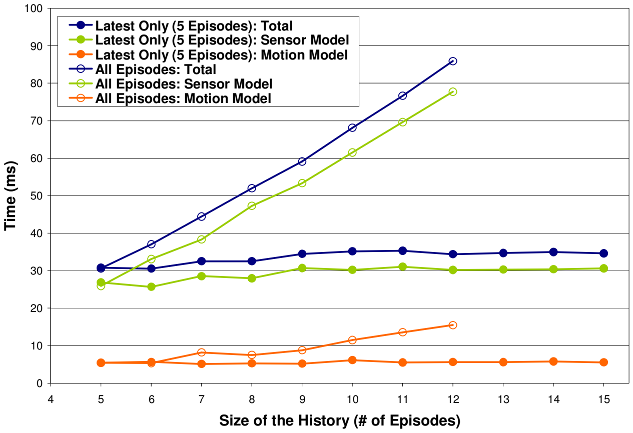


Figure 8: The average computation time required for the event matching step with respect to the size of the history

The graphs in Figure 8 shows the averaged computation time required for the event matching step with respect to the number of episodes in the robot's memory. It can be observed that, if all episodes in the memory were taken into consideration, the overall computation time increased linearly (same for both sensor model and motion model computations). On the other hand, when the cap was imposed on the number of the relevant episodes, those computation times remained constant (with minor variances[**]). Note that, for the limited-history case, the experiment was able to be carried out even when the size of the history reached 15 without any problem. On the other hand, for the full-history case, the robot could not reach the goal after the size of the history reached to 13 because the increased event matching time seemed to have started interfering with other parts of the computation (e.g., event sampling). As shown in Figure 9, even if only a limited history was taken into account, the performance in terms of the path-length did not seem to have been compromised. Similarity, the time to reach the goal (Figure 10) did not seem to have been affected by the imposed cap[††].

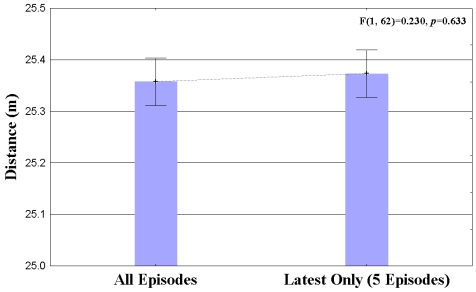


Figure 9: Comparison of the performances in terms of the path length of the robot (the vertical whiskers indicate the 95% confidence)

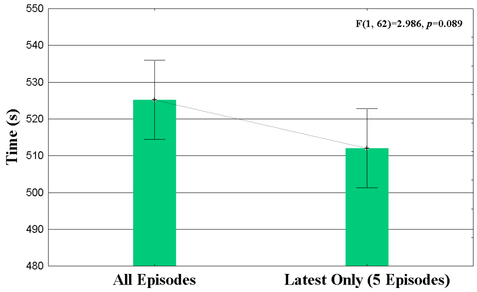


Figure 10: Comparison of the performances in terms of the time the robot took to reach the goal (the vertical whiskers indicate the 95% confidence)

V. Conclusion

In this paper, a biologically inspired episodic-memory-based approach for anticipatory behavior computation was explained. Forming episodic memories in a unidirectional-linear-chaining fashion, this approach incorporates multiple machine learning methods: namely, temporal difference learning, instance-based learning, recursive Bayesian filtering, and MDP. This approach attempts to solve the computational burden of the POMDP through: 1) abstraction of the state space via temporal difference learning; 2) abstraction of the action space by utilizing motor schemata; 3) narrowing down the state space in terms of the goals by employing instance-based learning; 4) eliminating the value-iteration by assuming unidirectional-linear-chaining formation of the states; 5) reducing the state-estimate computation by exploiting the property of the Poisson distribution; and 6) trimming the history length by imposing the cap on the number of episodes that are computed. In

[**] The variances most likely came from the different numbers of events in the different episodes.

[††] In fact, the mean value for the limited-history case was less than the full-history one even though the difference was not statistically significant ($p = 0.09$).

particular, claims 5) and 6) were empirically verified, confirming that the state estimation can be computed in an $O(n)$ time (where n is the number of the states).

Acknowledgment

The author would like to thank Prof. Ronald Arkin for his helpful comments and advises on this study as well as Michael Kaess, Zsolt Kira, Keith O'Hara, and Ananth Ranganathan for engaging in very inspirational discussions.

References

[1] L. Chaimowicz, A. Cowley, D. Gomez-Ibanez, B. Grocholsky, M. A. Hsieh, H. Hsu, J. F. Keller, V. Kumar, R. Swaminathan, and C. J. Taylor, "Deploying Air-Ground Multirobot Teams in Urban Environments," presented at *Multirobot Workshop*, Washington, D.C., 2005.

[2] Y. Endo, D. C. MacKenzie, and R. C. Arkin, "Usability Evaluation of High-Level User Assistance for Robot Mission Specification," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 34, 2004, pp. 168-180.

[3] M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma, "Experiences with a Mobile Robotic Elderly Guide for the Elderly," presented at *AAAI National Conference on Artificial Intelligence*, 2002.

[4] K. Wada, T. Shibata, T. Saito, and K. Tanie, "Robot Assisted Activity for Elderly People and Nurses at a Day Service Center," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2002, pp. 1416-1421.

[5] J. L. Burke, R. R. Murphy, M. Coovert, and D. Riddle, "Moonlight in Miami: A Field Study of Human-Robot Interaction in the Context of an Urban Serach and Rescue Disaster Response Training Exercise," *Human-Computer Interaction*, vol. 19, 2004, pp. 85-116.

[6] J. Casper and R. R. Murphy, "Human-Robot Interactions During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center," *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 33, 2003, pp. 367-385.

[7] Y. Endo and R. C. Arkin, "Anticipatory Robot Navigation by Simultaneously Localizing and Building a Cognitive Map," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems* 2003, pp. 460-466.

[8] H. Eichenbaum, P. Dudchenko, E. Wood, M. Shapiro, and H. Tanila, "The Hippocampus, Memory, Review and Place Cells: Is It Spatial Memory or a Memory Space?," *Neuron*, vol. 23, 1999, pp. 209-226.

[9] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman, "Acting Optimally in Partially Observable Stochastic Domains," *Proc. Nat'l Conf. Artificial Intelligence*, 1994, pp. 1023 - 1028.

[10] S. Koenig and R. Simmons, "Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models," in *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. Bonasso, and R. Murphy, eds., MIT Press, 1998, pp. 91 - 122.

[11] G. Theocharous and S. Mahadevan, "Approximate Planning with Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation," *Proc. IEEE Int'l Conf. Robotics and Automation*, 2002, pp. 1347-1352.

[12] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards Robotic Assistants in Nursing Homes: Challenges and Results," *Robotics and Autonomous Systems*, vol. 42, 2003, pp. 271-281.

[13] S. Thrun, "Monte Carlo POMDPs," *Proc. Advances in Neural Information Processing Systems 12*, MIT Press, 2000, pp. 1064-1070.

[14] J. Pineau, G. Gordon, and S. Thrun., "Point-Based Value Iteration: An Anytime Algorithm for POMDPs," *Proc. Int'l Joint Conf. Artificial Intelligence*, 2003, pp. 1025-1032.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction*. MIT Press, Cambridge, Mass., 1998.

[16] R. A. McCallum, "Hidden State and Reinforcement Learning with Instance-Based State Identification," *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 26, 1996, pp. 464-473.

[17] A. Ram and J. C. Santamaria, "Continuous Case-Based Reasoning," *Artificial Intelligence*, vol. 90, 1997, pp. 25-77.

[18] Y. Endo, "Anticipatory and Improvisational Robot via Recollection and Exploitation of Episodic Memories," *Proc. 2005 AAAI Fall Symp.: From Reactive to Anticipatory Cognitive Embodied Systems*, 2005, pp. 57-64.

[19] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *Int'l J. Robotics Research*, vol. 8, 1989, pp. 92-112.

[20] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, 1988, pp. 9-44.

[21] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, 1991, pp. 37-66.

[22] S. Thrun, "Robotic Mapping: A Survey," in *Exploring Artificial Intelligence in the New Millenium*, Morgan Kaufmann, 2002.

[23] P. R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification and Adaptive Control*. Prentice-Hall, Upper Saddle River, N.J., 1986

[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, 1996, pp. 237-285.

[25] N. Tomatis, I. Nourbakhsh, and R. Siegwart, "Hybrid Simultaneous Localization and Map Building: a Natural Integration of Topological and Metric," *Robotics and Autonomous Systems*, vol. 44, 2003, pp. 3-14.

[26] L. E. Parker, "ALLIANCE: an Architecture for Fault Tolerant Multirobot Cooperation," *IEEE Trans. Robotics and Automation*, vol. 14, 1998, pp. 220-240.

[27] C. Breazeal, "A Motivational System for Regulating Human-Robot Interaction," *Proc. Nat'l Conf. Artificial intelligence*, 1998, pp. 54-62.

[28] A. Stoytchev and R. C. Arkin, "Combining Deliberation, Reactivity, and Motivation in the Context of a Behavior-Based Robot Architecture," *Proc. IEEE Int'l Symp. Computational Intelligence in Robotics and Automation*, Banff, Alberta, 2001, pp. 290-295.

[29] T. Sawada, T. Takagi, Y. Hoshino, and M. Fujita, "Learning Behavior Selection Through Interaction Based on Emotionally Grounded Symbol Concept," *Proc. Int'l Conf. Humanoid Robots*, Los Angeles, 2004, pp. 450-469.

[30] R. C. Arkin, "Moving Up the Food Chain: Motivation and Emotion in Behavior-based Robots," in *Who Needs Emotions: The Brain Meets the Robot*, J. Fellous and M. Arbib, eds., Oxford University Press, 2005, pp. 245-270.

[31] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, Cambridge, Mass., 2005.

[32] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator," *Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, 2004, pp. 2149- 2154.

[33] *MissionLab: User Manual for MissionLab 7.0*. Georgia Tech Mobile Robot Laboratory, College of Computing, Georgia Institute of Technology, Atlanta, Ga, 2006.