Proceedings of the 9th International
Workshop on Discrete Event Systems
Göteborg, Sweden, May 28-30, 2008

Monte-Carlo-Based Partially Observable Markov Decision Process Approximations for Adaptive Sensing

Edwin K. P. Chong, Christopher M. Kreucher, and Alfred O. Hero III

Abstract— **Adaptive sensing involves actively managing sensor resources to achieve a sensing task, such as object detection, classification, and tracking, and represents a promising direction for new applications of discrete event system methods. We describe an approach to adaptive sensing based on approximately solving a partially observable Markov decision process (POMDP) formulation of the problem. Such approximations are necessary because of the very large state space involved in practical adaptive sensing problems, precluding exact computation of optimal solutions. We review the theory of POMDPs and show how the theory applies to adaptive sensing problems. We then describe Monte-Carlo-based approximation methods, with an example to illustrate their application in adaptive sensing. The example also demonstrates the gains that are possible from nonmyopic methods relative to myopic methods.**

I. Introduction

In its broadest sense, *adaptive sensing* has to do with actively managing sensor resources to achieve a sensing task. As an example, suppose our goal is to determine the presence or absence of an object, and we have at our disposal a single sensor that can interrogate the scene with any one of K waveforms. Depending on which waveform is used to radiate the scene, the response may vary greatly. After each measurement, we can decide whether to continue taking measurements using that waveform, change waveforms and take further measurements, or stop and declare whether or not the object is present. In adaptive sensing, this decision making is allowed to take advantage of the knowledge gained from the measurements so far. In this sense, the act of sensing "adapts" to what we know so far. What guides this adaptation is a performance objective that is determined beforehand—in our example above, this might be the average number of interrogations needed so that we can declare the presence or absence of the object with a confidence that exceeds some threshold (say, 90%). Adaptive sensing problems arise in a variety of application areas: medical diagnostics; nondestructive testing; sensor scheduling for target detection, identification, and tracking; waveform selection for radar imaging; and laser pulse shaping. Adaptive sensing represents a promising direction for new applications of discrete event system methods.

Adaptive sensing is fundamentally a *resource management* problem, in the sense that the main task is to make decisions over time on the use of sensor resources to maximize sensing performance. It is informative to distinguish between *myopic* and *nonmyopic* (also known as *dynamic* or *multistage*) resource management, a topic of much current interest (see, e.g., [4], [15], [12], [7], [1], [8], [10], [16]). In myopic resource management, the objective is to optimize performance on a per-decision basis. For example, consider the problem of *sensor scheduling* for tracking a single target, where the problem is to select, at each decision epoch, a single sensor to activate. An example sensor-scheduling scheme is *closest point of approach*, which selects the sensor that is perceived to be the closest to the target. Another (more sophisticated) example is the method described in [13], where the authors present a sensor scheduling method using alpha-divergence (or Rényi divergence) measures. Their approach is to make the decision that maximizes the expected information gain (in terms of the alpha-divergence).

Myopic adaptive sensing may not be ideal when the performance is measured over a horizon of time. In such situations, we need to consider schemes that trade off short-term for long-term performance. We call such schemes *nonmyopic*. Several factors motivate the consideration of nonmyopic schemes, easily illustrated in the context of sensor scheduling for target tracking:

Heterogeneous sensors. If we have sensors with different locations, waveform characteristics, usage costs, and/or lifetimes, the decision of whether or not to use a sensor, and with what waveform, should consider the overall performance, not whether or not its use maximizes the current performance.

Sensor motion. The future location of a sensor affects how we should act now. To optimize a long-term performance measure, we need to be opportunistic in our choice of sensor decisions.

Target motion. If a target is moving, there is potential benefit in sensing the target before it becomes unresolvable (e.g., too close to other targets or to clutter, or shadowed by large objects). In some scenarios, we may need to identify multiple targets before they cross, to aid in data association.

Environmental variation. Time-varying weather patterns affect target visibility in a way that potentially benefits from nonmyopic decision making. In particular, by exploiting models of target visibility maps, we can achieve improved sensing performance by careful selection of waveforms and beam directions over time.

Edwin K. P. Chong is with Colorado State University. Email: edwin.chong@colostate.edu

Christopher M. Kreucher is with Integrity Applications Incorporated in Ann Arbor, MI. Email: ckreuche@umich.edu

Alfred O. Hero III is with the University of Michigan. Email: hero@umich.edu

This material is based in part upon work supported by the Air Force Office of Scientific Research under Award FA9550-06-1-0324 and by DARPA under Award FA8750-05-2-0285. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the Air Force or of DARPA. Approved for Public Release, Distribution Unlimited.

978-1-4244-2593-8/08/$25.00 ©2008 IEEE

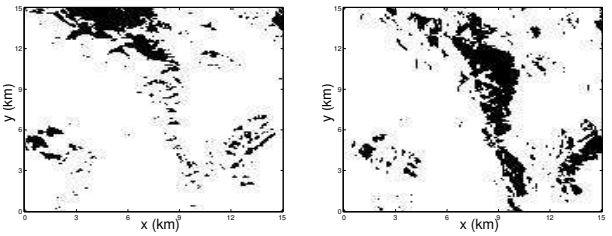
The main focus of this paper is on nonmyopic adaptive sensing. The basic methodology presented here consists of two steps: (1) Formulating the adaptive sensing problem as a partially observable Markov decision process (POMDP); and (2) applying an approximation to the optimal policy for the POMDP, because computing the exact solution is intractable.

II. Motivating Example

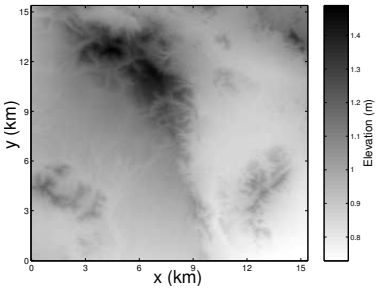
We now present a concrete motivating example involving a remote sensing application where the goal is to learn the contents of a surveillance region via repeated interrogation. (See [9] for a more complete exposition of adaptive sensing applied to such problems.)

Consider a single airborne sensor which is able to image a portion of a ground surveillance region to determine the presence or absence of moving ground targets. At each time epoch, the sensor is able to direct an electrically scanned array so as to interrogate a small area on the ground. Each interrogation yields some (imperfect) information about the small area. The objective is to choose the sequence of pointing directions that lead to the best ability to describe the entire contents of the surveillance region.

Further complicating matters is the fact that at each time epoch the sensor position causes portions of the ground to be unobservable due to the terrain elevation between the sensor and the ground. Given its position and the terrain elevation, the sensor can compute a visibility mask which determines how well a particular spot on the ground can be seen. As an example, in Figure 1 we give binary visibility masks computed from a sensor positioned (a) south and (b) to the west of the topologically nonhomogeneous surveillance region (these plots come from real digital terrain elevation maps). As can be seen from the figures, sensor position causes "shadowing" of certain regions. These regions, if measured, would provide no information to the sensor. A similar target masking effect occurs with atmospheric propagation attenuation from disturbances such as fog, rain, sleet, or dust. This example illustrates a situation where nonmyopic adaptive sensing is highly beneficial. Using a known sensor trajectory and known topological map, the sensor can predict locations that will be obscured in the future. This information can be used to prioritize resources so that they are used on targets that are predicted to become obscured in the future. Extra sensor dwells immediately before obscuration (at the expense of not interrogating other targets) will sharpen the estimate of target location. This sharpened estimate will allow better prediction of where and when the target will emerge from the obscured area. This is illustrated graphically with a six time-step vignette in Figure 2.



(a) Visibility mask: south (b) Visibility mask: west



(c) Elevation map of the surveillance region

Fig. 1. Visibility masks for a sensor positioned south and west of the surveillance region. We show binary visibility masks (non-visible areas are black and visible areas are white). In general, visibility may be between 0 and 1 indicating areas of reduced visibility, e.g., regions that are partially obscured by foliage.

III. Formulating Adaptive Sensing Problems

A. Partially Observable Markov Decision Processes

An adaptive sensing problem can be posed formally as a *partially observable Markov decision process* (POMDP), a model that should be familiar to the discrete event system community. For completeness, we will introduce POMDPs in sufficient detail to allow the reader to see how an adaptive sensing problem can be posed as a POMDP, and to explore methods to approximate optimal solutions. For a full treatment of POMDPs and related background, see [2].

A POMDP is specified by the following ingredients:

- A set of states (the state space) and a distribution specifying the random initial state.
- A set of possible actions.
- A state-transition law specifying the next-state distribution given an action taken at a current state.
- A reward function specifying the reward (real number) received given an action taken at a state.
- A set of possible observations.
- An observation law specifying the distribution of observations given an action taken at a state.

As usual, for technical formality, we will assume that the state space, the action set, and the observation set are all finite.

A POMDP is a controlled dynamical process in discrete time. The process begins at time $k = 0$ with a (random) initial state. At this state, we perform an action and receive a reward, which depends on the action and the state. At the same time, we receive an observation, which again depends on the action and the state. The state then transitions to some random next state, whose distribution is specified by the state-transition law. The process then repeats in the same way—at each time, the process is at some state, and the action taken at that state determines the reward, observation, and next state. As a result, the state evolves randomly over time in response to actions, generating observations along the way.

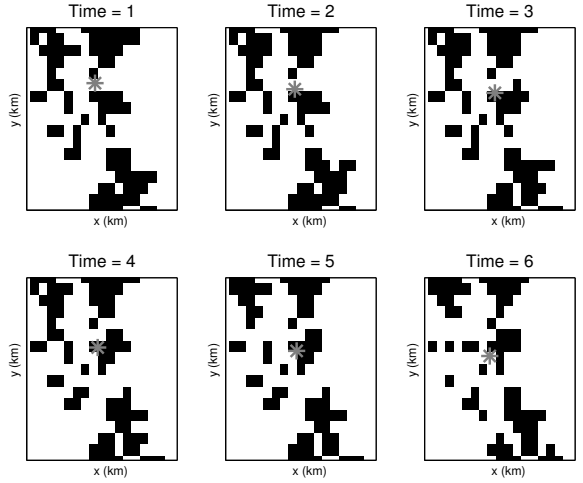


Fig. 2. A six time step vignette where a target moves through an obscured area. Other targets are present elsewhere in the surveillance region. The target is depicted by an asterisk. Areas obscured to the sensor are black and areas that are visible are white. Extra dwells just before becoming obscured (time = 1) aid in localization after the target emerges (time = 6).

B. Belief State

As a POMDP evolves over time, we do not have direct access to the states that occur. Instead, all we have are the observations generated over time, providing us with clues of the actual underlying states (hence the term *partially observable*). These observations might, in some cases, allow us to infer exactly what states actually occurred. However, in general, there will be some uncertainty in our knowledge of the states that actually occurred. This uncertainty is represented by the *belief state* (or *information state*), which is the *a posteriori* distribution of the underlying state given the history of observations.

Let $\mathcal{X}$ denote the state space (the set of all possible states in our POMDP), and let $\mathcal{B}$ be the set of distributions over $\mathcal{X}$. Then a belief state is simply an element of $\mathcal{B}$. Just as the underlying state changes over time, the belief state also changes over time. At time $k = 0$, the (initial) belief state is equal to the given initial state distribution. Then, once an action is taken and an observation is received, the belief state changes to a new belief state, in a way that depends on the observation received and the state-transition and observation laws. This change in the belief state can be computed explicitly using Bayes' rule.

To elaborate, suppose that the current time is k, and the current belief state is $b_k \in \mathcal{B}$. Note that b_k is a probability distribution over $\mathcal{X}$—we use the notation $b_k(x)$ for the probability that b_k assigns to state $x \in \mathcal{X}$. Suppose that we take action a_k and, as a result, we receive observation z_k. Denote the transition law by P_{trans}, so that the probability of transitioning to state y given that action a is taken at state x is $P_{\text{trans}}(y|x,a)$. Similarly, denote the observation law by Z, so that the probability of receiving observation z given that action a is taken at state x is $P_{\text{obs}}(z|x,a)$. Then, the next belief state given action a_k is computed using the following two-step update procedure:

1. Compute the "updated" belief state $\hat{b}_k$ based on the observation y_k of the state x_k at time k, using Bayes' rule:
$$\hat{b}_k(x) = \frac{P_{\text{obs}}(y_k|x,a_k)b_k(x)}{\sum_{y\in\mathcal{X}} P_{\text{obs}}(y_k|y,a_k)b_k(y)}, \quad x \in \mathcal{X}.$$
2. Compute the belief state b_{k+1} using the state-transition law:
$$b_{k+1}(x) = \sum_{y\in\mathcal{X}} \hat{b}_k(y)P_{\text{trans}}(x|y,a_k), \quad x \in \mathcal{X}.$$

This two-step procedure is commonly realized in terms of a Kalman filter or a particle filter [9].

It is useful to think of a POMDP as a random process of evolving belief states. Just as the underlying state transitions to some random new state with the performance of an action at each time, the belief state also transitions to some random new belief state. So the belief state process also has some "belief-state-transition" law associated with it, which depends intimately on the underlying state-transition and the observation laws. But, unlike the underlying state, the belief state is fully accessible.

Indeed, any POMDP may be viewed as a *fully observable* Markov decision process (MDP) with state space $\mathcal{B}$, called the *belief-state MDP* or *information-state MDP* (see [2]). To complete the description of this MDP, we sill show how to write its reward function, which specifies the reward received when action a is taken at belief-state b. Suppose $b \in \mathcal{B}$ is some belief state and a is an action. As before, let $R(x,a)$ be the reward received if action a is taken at underlying state x. Then let $r(b,a) = \sum_{x\in\mathcal{X}} b(x)R(x,a)$ be the expected reward with respect to belief-state b, given action a. This reward $r(b,a)$ then represents the reward function of the belief-state MDP.

C. Optimization Objective

Given a POMDP, our goal is to select actions over time to maximize the expected cumulative reward (we take expectation here because the cumulative reward is a random variable). To be specific, suppose we are interested in the expected cumulative reward over a time horizon of length H: $k = 0, 1, \ldots, H-1$. Let x_k and a_k be the state and action at time k, and let $R(x_k, a_k)$ be the resulting reward received. Then, the cumulative reward over horizon H is given by
$$V_H = \mathrm{E}\left[\sum_{k=0}^{H-1} R(x_k, a_k)\right],$$
where E represents expectation. It is important to realize that this expectation is with respect to $x_0, x_1, \ldots$; i.e., the random initial state and all the subsequent states in the evolution of the process, given the actions $a_0, a_1, a_2, \ldots$ taken over time. The goal is to pick these actions so that the objective function is maximized.

Note that we can also represent the objective function in terms of r (the reward function of the belief-state MDP)

instead of R:

$$V_H(b_0) = \mathrm{E}\left[\sum_{k=0}^{H-1} r(b_k, a_k) \middle| b_0\right].$$

where $\mathrm{E}[\cdot|b_0]$ represents conditional expectation given b_0.

D. Optimal Policy

In general, the action chosen at each time should be allowed to depend on the entire history up to that time (i.e., the action at time k is a random variable that is a function of all observable quantities up to time k). However, it turns out that if an optimal choice of such a sequence of actions exists, then there is an optimal choice of actions that depends only on "belief-state feedback" (this result is due to [18]). In other words, it suffices for the action at time k to depend only on the belief-state b_k at time k. So what we seek is, at each time k, a mapping $\pi_k^* : \mathcal{B} \to \mathcal{A}$ such that if we perform action $a_k = \pi_k^*(b_k)$, then the resulting objective function is maximized. As usual, we call such a mapping a *policy*. So, what we seek is an *optimal policy*.

E. Bellman's Principle and Q-values

The key result in Markov decision theory relevant here is Bellman's principle. Let $V_H^*(b_0)$ be the optimal objective function value (over horizon H) with b_0 as the initial belief state. Then, *Bellman's principle* states that

$$V_H^*(b_0) = \max_a (r(b_0, a) + \mathrm{E}[V_{H-1}^*(b_1)|b_0, a])$$

where b_1 is the random next belief state (with distribution depending on a), and $\mathrm{E}[\cdot|b_0, a]$ represents conditional expectation with respect to the random next state b_1, whose distribution depends on b_0 and a. Moreover,

$$\pi_0^*(b_0) = \arg\max_a (r(b_0, a) + \mathrm{E}[V_{H-1}^*(b_1)|b_0, a])$$

is an optimal policy.

Define the *Q-value* of taking action a at state b_k as

$$Q_{H-k}(b_k, a) = r(b_k, a) + \mathrm{E}[V_{H-k-1}^*(b_{k+1})|b_k, a],$$

where b_{k+1} is the random next belief state (which depends on the observation z_k at time k, as described in Section III-B). Then, Bellman's principle can be rewritten as

$$\pi_k^*(b_k) = \arg\max_a Q_{H-k}(b_k, a)$$

i.e., the optimal action at belief-state b_k (at time k, with a horizon-to-go of $H - k$) is the one with largest Q-value at that belief state. This principle, called *lookahead*, is the heart of POMDP solution approaches.

F. Stationary policies

In general, an optimal policy is a function of time k. If H is sufficiently large, then the optimal policy is approximately *stationary* (independent of k). This is intuitively clear: if the end of the time horizon is a million years away, then how we should act today given a belief-state is the same as how we should act tomorrow with the same belief state. Said differently, if H is sufficiently large, the difference between Q_H and Q_{H-1} is negligible. Henceforth we will assume for convenience there is a stationary optimal policy, and this is what we seek. We will use the notation π for stationary policies (with no subscript k)—this significantly simplifies the notation. Our approach is equally applicable to the short-horizon, nonstationary case, with appropriate notational modification (to account for the time dependence of decisions).

G. Receding horizon

Assuming H is sufficiently large and that we seek a stationary optimal policy, at any time k we write:

$$\pi^*(b) = \arg\max_a Q_H(b, a).$$

Notice that the horizon is taken to be fixed at H, regardless of the current time k. This is justified by our assumption that H is so large that at any time k, the horizon is still approximately H time steps away. This approach of taking the horizon to be fixed at H is called *receding horizon control*. For convenience, we will also henceforth drop the subscript H from our notation (unless the subscript is explicitly needed).

IV. Q-value Approximation Methods

A. Basic approach

By Bellman's principle, knowing the Q-values allows us to make optimal control decisions. In particular, if we are currently at belief-state b, we need only find the action a with the largest $Q(b, a)$. This principle yields a basic control framework.

Recall the definition of the Q-value,

$$Q(b, a) = r(b, a) + \mathrm{E}[V^*(b')|b, a], \tag{1}$$

where b' is the random next belief state (with distribution depending on a). In all but very special problems, it is impossible to compute the Q-value exactly. In this section, we describe methods to approximate the Q-value, focusing on Monte-Carlo-based methods. Because the first term on the right-hand side of (1) is usually straightforward to compute, most approximation methods focus on the second term. It is important to realize that the quality of an approximation to the Q-value is not so much in the accuracy of the actual values obtained, but in the *ranking* of the actions reflected by their *relative* values.

B. Monte Carlo sampling

In general, we can think of Monte Carlo methods simply as the use of computer generated random numbers in computing expectations of random variables through averaging over many samples. With this in mind, it seems natural to consider using Monte Carlo methods to compute the value function directly based on Bellman's equation:

$$V_H^*(b_0) = \max_{a_0} (r(b_0, a_0) + \mathrm{E}[V_{H-1}^*(b_1)|b_0, a_0]).$$

Notice that the second term on the right-hand side involves expectations (one per action candidate a_0), which can be

computed using Monte Carlo sampling. However, the random variable inside each expectation is itself an objective function value (with horizon $H-1$), and so it too involves a max of an expectation via Bellman's equation:

$$V_H^*(b_0) = \max_{a_0} \left(r(b_0, a_0) + \mathrm{E} \left[\max_{a_1} (r(b_1, a_1) + \mathrm{E}[V_{H-2}^*(b_2)|b_1, a_1]) \middle| b_0, a_0 \right] \right).$$

Notice we now have two "layers" of max and expectation, one "nested" within the other. Again, we see the inside expectation involves the value function (with horizon $H-2$), which again can be written as a max of expectations. Proceeding this way, we can write $V_H^*(b_0)$ in terms of H layers of max and expectations. Each expectation can be computed using Monte Carlo sampling. The remaining question is how computationally burdensome is this task?

Kearns, Mansour, and Ng [11] have provided a method to calculate the computational burden of approximating the value function using Monte Carlo sampling as described above, given some prescribed accuracy in the approximation of the value function. Unfortunately, it turns out that for practical POMDP problems this computational burden is prohibitive, even for modest degrees of accuracy. So, while Bellman's equation suggests a natural Monte Carlo method for approximating the value function, the method is not useful in practice. For this reason, we seek alternative approximation methods. In the next few subsections, we explore some of these methods.

C. Hindsight optimization

Let us write the value function (optimal objective function value as a function of belief state) as

$$V^*(b) = \max_{\pi} \mathrm{E} \left[\sum_{k=0}^{H-1} r(b_k, \pi(b_k)) \middle| b, \pi(b) \right] = \mathrm{E} \left[\max_{a_0, \ldots, a_{H-1} : a_k = \pi(b_k)} \sum_{k=0}^{H-1} r(b_k, a_k) \middle| b \right],$$

where the notation $\max_{a_0,\ldots,a_{H-1}:a_k=\pi(b_k)}$ means maximization subject to the constraint that each action a_k is a (fixed) function of the belief state b_k. If we relax this constraint on the actions and allow them to be arbitrary random variables, then we have an upper bound on the value function:

$$\hat{V}_{\mathrm{HO}}(b) = \mathrm{E} \left[\max_{a_0, \ldots, a_{H-1}} \sum_{k=0}^{H-1} r(b_k, a_k) \middle| b \right].$$

In some applications, this bound provides a suitable approximation to the value function. The advantage of this method is that in certain situations the computation of the "max" above involves solving a relatively easy optimization problem. This method is called *hindsight optimization* [5], [19].

An implementation of particular interest here involves averaging over many Monte Carlo simulation runs to compute the expectation above. In this case, the "max" is computed for each simulation run by first generating all the random numbers for that run, and then applying a static optimization algorithm to compute optimal actions $a_0, \ldots, a_{H-1}$. It is easy now to see why we call the method "hindsight" optimization: the optimization of the action sequence is done after knowing all uncertainties over time, as if making decisions in hindsight.

D. Rollout

Next, we describe the method of *rollout* [3]. The basic idea is simple. First let $V^\pi(b_0)$ be the objective function value corresponding to policy π. Recall that $V^* = \max_\pi V^\pi$. In the method of rollout, we assume that we have a candidate policy π_{base} (called the *base policy*), and we simply replace V^* in (1) by $V^{\pi_{\mathrm{base}}}$. In other words, we use the following approximation to the Q-value:

$$Q^{\pi_{\mathrm{base}}}(b, a) = r(b, a) + \mathrm{E}[V^{\pi_{\mathrm{base}}}(b')|b, a].$$

We can think of $V^{\pi_{\mathrm{base}}}$ as the performance of applying π_{base} in our system. In many situations of interest, $V^{\pi_{\mathrm{base}}}$ is relatively easy to compute, either analytically, numerically, or via Monte Carlo simulation.

It turns out that the policy π defined by

$$\pi(b) = \arg\max_a Q^{\pi_{\mathrm{base}}}(b, a) \tag{2}$$

is at least as good as π_{base} (in terms of the objective function); in other words, this step of using one policy to define another policy has the property of *policy improvement*. This result is the basis for a method known as *policy iteration*, where we iteratively apply the above policy-improvement step to generate a sequence of policies converging to the optimal policy. However, policy iteration is difficult to apply in problems with large belief-state spaces, because the approach entails explicitly representing a policy and iterating on it (remember that a policy is a mapping with the belief-state space $\mathcal{B}$ as its domain).

In the method of rollout, we do not explicitly construct the policy π in (2). Instead, at each time step, we use (2) to compute the output of the policy at the current belief-state. For example, the term $\mathrm{E}[V^{\pi_{\mathrm{base}}}(b')|b, a]$ can be computed using Monte Carlo sampling. To see how this is done, observe that $V^{\pi_{\mathrm{base}}}(b')$ is simply the mean cumulative reward of applying policy π_{base}, a quantity that can be obtained by Monte Carlo simulation. The term $\mathrm{E}[V^{\pi_{\mathrm{base}}}(b')|b, a]$ is the mean with respect to the random next belief-state b' (with distribution that depends on b and a), again obtainable via Monte Carlo simulation. We provide more details in Section IV-E. In our subsequent discussion of rollout, we focus on implementation of rollout using Monte Carlo simulation. For an application of the rollout method to sensor scheduling for target tracking, see [7], [8], [10], [16]. For an extension involving multiple base policies, see [6].

As a further approximation, suppose we use a delta distribution to approximate belief states in our simulation of the future. In other words, in our lookahead simulation, we do away with keeping track of belief states altogether and instead simulate only a *completely observable* version

of the system. In this case, we need only consider a base policy that maps underlying states to actions—we could simply apply rollout to this policy, and not have to maintain any belief states in our simulation. We call this method *completely observable (CO) rollout*. It turns out that in certain applications, such as in sensor scheduling for target tracking, a CO-rollout base policy is naturally available (see [7], [8], [10], [16]). Note that we will still need to keep track of (or estimate) the actual belief state of the system, even if we use CO rollout. The benefit of CO rollout is that it allows us to avoid keeping track of (simulated) belief states in our *simulation* of the future evolution of the system.

E. Control architecture

Figure 3 shows the basic control architecture. The topmost block represents the sensing system, which we treat as having an input and two forms of output. The input represents actions (external control commands) we can apply to control the sensing system. Actions usually include sensor-resource controls, such as which sensor(s) to activate, at what power level, where to point, what waveforms to use, and what sensing modes to activate. Actions may also include communication-resource controls, such as the data rate for transmission from each sensor.

The two forms of outputs from the sensing system represent: (1) Fully observable aspects of the internal state of the sensing system (called *observables*), and (2) measurements (observations) of those aspects of the internal state that are not directly observable (which we refer to simply as *measurements*). We assume that the underlying state-space is the Cartesian product of two sets, one representing unobservables and the other representing observables. Target states are prime examples of unobservables. So, measurements are typically the outputs of sensors, representing observations of target states. Observables include things like sensor locations and orientations, which sensors are activated, battery status readings, etc. In the remainder of this section, we describe the components of our control framework. Our description starts from the architecture of Figure 3 and progressively fills in the details.

At each decision epoch, the *controller* takes the outputs (measurements and observables) from the sensing system and, in return, generates an action that is fed back to the sensing system. This basic closed-loop architecture is familiar to mainstream control system design approaches.

The controller has two main components. The first is the particle filter, which takes as input the measurements, and provides as output samples of unobservable internal states (henceforth called *unobservables*), representing the posterior distribution of the unobservables. This posterior distribution, together with the observables, form the belief state.

The second component is the *action selector*, which takes the belief state and computes an action (the output of the controller). As shown in Figure 4, the action selector consists of a search (optimization) algorithm that optimizes an objective function, the *Q-function*, with respect to an action. In other words, the Q-function is a function of the

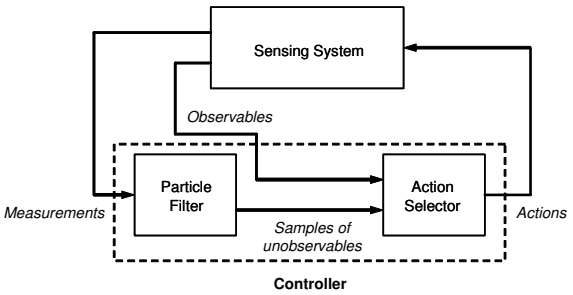


Fig. 3. Basic control architecture.

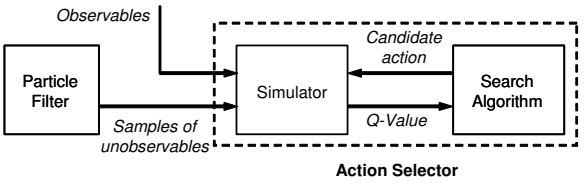


Fig. 4. Components of the action selector.

action—it maps each action, at a given belief state, to its Q-value. The action that we seek is one that maximizes the Q-function. The search algorithm iteratively generates a candidate action and evaluates the Q-function at this action (this numerical quantity is the Q-value), searching over the space of candidate actions for one with the largest Q-value. Search algorithms that are suitable here include the method of [17], which is designed for such problems, dovetails well with a simulation-based approach, and accommodates heuristics to guide the search within a rigorous framework.

To further explain the components of the control architecture, consider applying the method of rollout. In this case, the evaluation of the Q-value for any given candidate action relies on a simulation model of the sensing system with some base policy. This simulation model is a "dynamic" model in that it evaluates the behavior of the sensing system over some horizon of time (specified beforehand). The simulator requires as inputs the current observables and samples of unobservables from the particle filter (to specify initial conditions) and a candidate action. The output of the simulator is a Q-value corresponding to the current measurements and observables, for the given candidate action. The output of the simulator represents the mean performance of applying the base policy, depending on the nature of the objective function. For example, the performance measure of the system may be the negative mean of the sum of the cumulative tracking error and the sensor usage cost over a horizon of H time steps, given the current system state and candidate action.

To elaborate on exactly how the Q-value approximation using rollout is implemented, suppose we are given the current observables and a set of samples of the unobservables (from the particle filter). The current observables together with a single sample of unobservables represent a candidate current underlying state of the sensing system. Starting from this candidate current state, we simulate the application of

the given candidate action (which then leads to a random next state), followed by application of the base policy for the remainder of the time horizon—during this time horizon, the system state evolves according to the dynamics of the sensing system as encoded within the simulation model. For this single simulation run, we compute the performance of the system (e.g., the negative of the sum of the cumulative tracking error and sensor usage cost over that simulation run). We do this for each sample of the unobservables, and then average over the performance values from these multiple simulation runs. This average is what we output as the Q-value.

The samples of the unobservables from the particle filter that are fed to the simulator (as candidate initial conditions for unobservables) may include all the particles in the particle filter (so that there is one simulation run per particle), or may constitute only a subset of the particles. In principle, we may even run multiple simulation runs per particle.

The Monte Carlo method for approximating POMDP solutions has some beneficial features. First, it is flexible in that a variety of adaptive sensing scenarios can be tackled using the same framework. This is important because of the wide variety of sensors encountered in practice. Second, the method does not require analytical tractability; in principle, it is sufficient to simulate a system component, whether or not its characteristics are amenable to analysis. Third, the framework is modular in the sense that models of individual system components (e.g., sensor types, target motion) may be treated as "plug-in" modules. Fourth, the approach integrates naturally with existing simulators. Finally, the approach is inherently nonmyopic, allowing the trade-off of short-term gains for long-term rewards.

V. Simulation Results

In this section, we illustrate the performance of several of the strategies discussed in this paper on a common model problem. The model problem has been chosen to have the characteristics of the motivating example given earlier, while remaining simple enough so that the workings of each method are transparent. The defining elements and challenges of the problem include a continuous, large state space, a large action space when considering the multi-step optimization, and time-varying dynamics of the sensor and target.

In the model problem, there are two targets, each of which is described by a one-dimensional position (see Figure 5). The state is therefore a 2-dimensional real number describing the target locations plus the sensor position. Targets move according to a pure diffusion model, and the belief state is propagated using this model. Computationally, the belief state is estimated by a multi-target particle filter, according to the algorithm given in [14].

The sensor may measure any one of 16 cells, which span the possible target locations (again, see Figure 5). The sensor is capable of making three (not necessarily distinct) measurements per time step, receiving binary returns independent from dwell to dwell. In occupied cells, a detection is received with probability $P_d = 0.9$. In cells that are unoccupied a detection is received with probability $P_f = 0.01$.

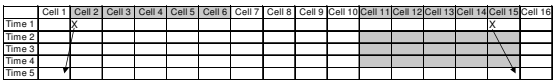


Fig. 5. The model problem. At the onset, the belief state for target 1 is uniformly distributed across cells $\{2, \ldots, 6\}$ and the belief state for target 2 is uniformly distributed across cells $\{11, \ldots, 15\}$. At time 1 all cells are visible. At times 2, 3, and 4, cells $\{11, \ldots, 15\}$ are obscured. This is a simple case where a target is initially visible, becomes obscured, and then reemerges.

At the onset, positions of the targets are known only probabilistically. The belief state for the first target is uniform across sensor cells $\{2, \ldots, 6\}$ and for the second target is uniform across sensor cells $\{11, \ldots, 15\}$. The particle filter used to estimate the belief state is initialized with this uncertainty.

Visibility of the cells changes with time as follows. At time 1, all cells are visible. At times 2, 3, and 4, cells $\{11, \ldots, 15\}$ become obscured. At time 5, all cells are visible again. This time varying visibility map is known to the sensor management algorithm and should be exploited to best choose sensing actions.

Sensor management decisions are made by using the belief state to predict which actions are most valuable. In the following paragraphs, we contrast the decisions made by a number of different strategies that have been described earlier.

At time 1 a myopic strategy, using no information about the future visibility, will choose to measure cells uniformly from the set $\{2, \ldots, 6\} \cup \{11, \ldots, 15\}$ as they all have the same expected immediate reward. As a result, target 1 and target 2 will on the average be given equal attention. A nonmyopic strategy, on the other hand, will choose to measure cells from $\{11, \ldots, 15\}$ as they are soon to become obscured. That is, the policy of looking for target 2 at time 1 followed by looking for target 1 is best.

Figure 6 shows the performance of several of the on-line strategies discussed in this paper on this common model problem. The performance of each scheduling strategy is measured in terms of the mean squared tracking error at each time step. The curves represent averages over $10,000$ realizations of the model problem. Each realization has randomly chosen initial positions of the targets and measurements corrupted by random mistakes as discussed above. The five policies are as follows.

- A **random** policy that simply chooses one of the 16 cells randomly for interrogation. This policy provides a worst-case performance and will bound the performance of the other policies.
- A **myopic** policy that takes the action expected to maximize immediate reward. Here the surrogate reward is information gain (see [13]), so the value of an action is estimated by the amount of information it gains. The myopic policy is sub-optimal because it does not consider the long term ramifications of its choices. In particular, at time 1

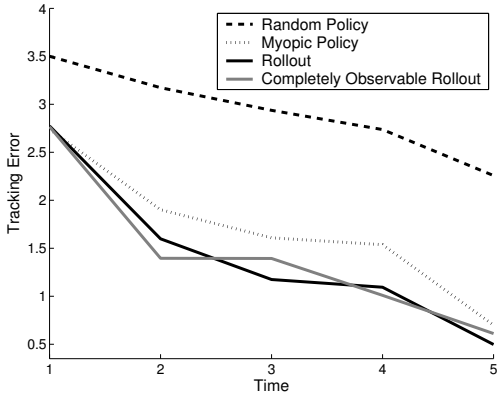


Fig. 6. The performance of the five policies discussed above. Performance is measured in terms of mean squared tracking error at each time step, averaged over 10^4 Monte Carlo trials.

the myopic strategy has no preference as to which target to measure because both are unobscured and have uncertain position. Therefore, half of the time, target 1 is measured, resulting in an opportunity cost because target 2 is about to disappear.

• The **rollout** policy described in Section IV-D. The base policy used here is to point the sensor where the target is expected to be. This expectation is computed using the predicted future belief state, which requires the belief state to be propagated in time. This is done using a particle filtering. We again use information gain as the surrogate metric to evaluate policies. The computational burden of this method is on the order of NH times that of the myopic policy, where H is the horizon length and N is the number of Monte Carlo trials used in the approximation (here $H = 5$ and $N = 25$).

• The **CO-rollout** policy described in Section IV-D. The base policy here is to point the sensor where the target is expected to be, but enforces the criterion that the sensor should alternate looking at the two targets. This slight modification is necessary due to the delta-function representation of future belief states. Since the completely observable policy does not predict the posterior into the future, it is significantly faster than standard rollout (an order of magnitude faster in these simulations). However, it requires a different surrogate reward (one that does not require the posterior like the information gain surrogate metric). Here we have chosen as a surrogate reward to count the number of detections received, discounting multiple detections of the same target.

Our main intent here is simply to convey that, from Figure 6, the nonmyopic policies perform similarly, and are better than the myopic and random policies, though at the cost of additional computational burden. The nonmyopic techniques perform similarly since they ultimately choose similar policies. Each one prioritizes measuring the target that is about to dissapear over the target that is in the clear. On the other hand, the myopic policy is "losing" the target more often, resulting in higher mean error as there are more catastrophic events.

REFERENCES

[1] D. P. Bertsekas, "Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC," in *Proc. of the Joint 44th IEEE Conference on Decision and Control and European Control Conference*, Seville, Spain, December 12–15, 2005.

[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, Vol. I, 3rd Ed., 2005; Vol. II, 2nd Ed., 2001.

[3] D. P. Bertsekas and D. A. Castanon, "Rollout algorithms for stochastic scheduling problems," *Journal of Heuristics*, vol. 5, pp. 89–108, 1999.

[4] D. Castanon,"Approximate dynamic programming for sensor management," *Proceedings of the 36th IEEE Conference on Decision and Control*, San Diego, December 1997, pp. 1202–1207.

[5] E. K. P. Chong, R. L. Givan, and H. S. Chang, "A framework for simulation-based network control via hindsight optimization," *Proc. of the 39th IEEE Conf. on Decision and Control*, Sydney, Australia, Dec. 12–15, 2000, pp. 1433–1438.

[6] H. S. Chang, R. L. Givan, and E. K. P. Chong, "Parallel rollout for online solution of partially observable Markov decision processes," *Discrete Event Dynamic Systems*, vol. 14, no. 3, pp. 309–341, 2004.

[7] Y. He and E. K. P. Chong, "Sensor scheduling for target tracking in sensor networks," in *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC'04)*, December 14–17, 2004, pp. 743–748.

[8] Y. He and E. K. P. Chong, "Sensor scheduling for target tracking: A Monte Carlo sampling approach," *Digital Signal Processing*, vol. 16, no. 5, pp. 533–545, September 2006.

[9] A. Hero, D. Castanon, D. Cochran, and K. Kastella, Eds., *Foundations and Applications of Sensor Management*, Springer, 2007.

[10] L. W. Krakow, Y. Li, E. K. P. Chong, K. N. Groom, J. Harrington, and B. Rigdon, "Control of perimeter surveillance wireless sensor networks via partially observable Markov decision process," in *Proceedings of the 2006 IEEE International Carnahan Conference on Security Technology (ICCST)*, Lexington, Kentucky, October 17–20, 2006.

[11] M. J. Kearns, Y. Mansour, and A. Y. Ng, "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes," *Proc. of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 1324–1331.

[12] C. M. Kreucher, A. O. Hero, K. Kastella, and D. Chang, "Efficient Methods of Non-myopic Sensor Management for Multitarget Tracking," in *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC'04)*, December 14–17, 2004.

[13] C. M. Kreucher, K. Kastella, and A. O. Hero III, "Sensor Management Using An Active Sensing Approach," *Signal Processing*, vol. 85, no. 3, pp. 607–624, March 2005.

[14] C. M. Kreucher, K. Kastella, and A. O. Hero III, "Multitarget Tracking using the Joint Multitarget Probability Density," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, no. 4, pp. 1396–1414, October 2005.

[15] V. Krishnamurthy and R. J. Evans, "Hidden Markov Model Multiarm Bandits: A Methodology for Beam Scheduling in Multitarget Tracking," *IEEE Transactions on Signal Processing*, vol. 49, no. 12, pp. 2893–2908, December 2001.

[16] Y. Li, L. W. Krakow, E. K. P. Chong, and K. N. Groom, "Approximate stochastic dynamic programming for sensor scheduling to track multiple targets," *Digital Signal Processing*, 2008, to appear.

[17] L. Shi and C.-H. Chen, "A new algorithm for stochastic discrete resource allocation optimization," *Discrete Event Dynamic Systems*, vol. 10, pp. 271–294, 2000.

[18] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, pp. 1071–1088, no. 5, 1973.

[19] G. Wu, E. K. P. Chong, and R. L. Givan, "Burst-level congestion control using hindsight optimization," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 979–991, June 2002.