

Maximizing Reward in a Non-Stationary Mobile Robot Environment

Dani Goldberg

*Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA
15213-3891*

danig@cs.cmu.edu

Maja J Matarić

*Computer Science Department, University of Southern California
Los Angeles, CA 90089-0781*

mataric@cs.usc.edu

Abstract.

The ability of a robot to improve its performance on a task can be critical, especially in poorly known and non-stationary environments where the best action or strategy is dependent upon the current state of the environment. In such systems, a good estimate of the current state of the environment is key to establishing high performance, however quantified. In this paper, we present an approach to state estimation in poorly known and non-stationary mobile robot environments, focusing on its application to a mine collection scenario where performance is quantified using reward maximization. The approach is based on the use of *augmented Markov models* (AMMs), a sub-class of semi-Markov processes. We have developed an algorithm for incrementally constructing arbitrary-order AMMs on-line. It is used to capture the interaction dynamics between a robot and its environment in terms of behavior sequences executed during the performance of a task. For the purposes of reward maximization in a non-stationary environment, multiple AMMs monitor events at different timescales and provide statistics used to select the AMM likely to have a good estimate of the environmental state. AMMs with redundant or outdated information are discarded, while attempting to maintain sufficient data to reduce conformation to noise. This approach has been successfully implemented on a mobile robot performing a mine collection task. In the context of this task, we first present experimental results validating our reward maximization performance criterion. We then incorporate our algorithm for state estimation using multiple AMMs, allowing the robot to select appropriate actions based on the estimated state of the environment. The approach is tested first with a physical robot, in a non-stationary environment with an abrupt change, then with a simulation, in a gradually shifting environment.

Keywords: mobile robots, reward maximization, on-line modeling, collection tasks, non-stationary environments

1. Introduction and Motivation

In certain classes of mobile robot tasks, a robot may be required to perform optimally with respect to the information it possesses about the structure of its environment. Reward maximization may be used as a framework for quantifying performance, with the robot receiving reward in proportion to its success in executing the task. Reward maximization in a non-stationary environment requires the robot to be able to maintain a good estimate of the state of the changing environment and use that estimate to help select appropriate actions. There are a number of issues that can compound the difficulty of this problem:



© 2002 Kluwer Academic Publishers. Printed in the Netherlands.

- The robot may have no *a priori* knowledge of the environment and thus also lack a baseline for gauging the non-stationarity of that environment.
- Given only local sensing capabilities, the robot may require a significant amount of time to estimate the state of the environment. Any estimate of state, however, may be outdated in a non-stationary system.
- The nature of the task may be stochastic, with uncertainties large enough to preclude an effective predictive model of environmental state, or dynamics too complex to make the development of such a model feasible or tractable. Alternatively, however potentially simple the system, there may be no *a priori* data with which to instantiate a model.
- Further, in a stochastic system, the variability associated with performing a task (or elements thereof) may be enormous and effectively mask gradual shifts in the environment. Conversely, in a system with very low variability, even minute shifts may be easily detected. Thus, effective estimation of environmental state requires an understanding of the system’s variability.
- Depending on the task or environment, the timescale at which the non-stationarity manifests, and thus can be detected, may differ. For example, in one task, the environmental change may be almost instantaneous (e.g., inside, lights being turned on/off), detectable over a short period of time. In another task, the change may be slow and incremental (e.g., day turning into night), requiring the examination of a large time interval for detection. Hard-coding the robot with a specific timescale to use for state estimation can be problematic. A timescale that is too small makes the robot incapable of detecting the change. Conversely, a timescale that is unnecessarily large delays detection of changes and may be undesirable in time-critical situations.

As a concrete example, consider the task of collecting undetonated land mines in a field. Assume that there are two types of mines, large and small, with destructive power proportional to their size. The robot’s goal is to minimize the total destructive power of the mine field as much as possible during a given period of time. When the robot is rewarded in proportion to the destructive power of the mines it collects, the goal becomes equivalent to reward maximization. To accomplish its goal, the robot must have enough data about its environment (the field) to intelligently decide whether it is best to collect large mines or small ones each time a mine is encountered (we assume that the robot can only collect one mine at a time). The difficulty of this task is compounded when the issues mentioned above apply. The task is likely stochastic, with unknown variability (for instance, finding a mine is a probabilistic process, and the time required to do so may vary greatly from one mine to the next). The robot may have no *a priori* information about the numbers of large and small mines in the field, their distributions, or relative proportions. The robot may also lack global sensing of the mines in the field.

These limitations relegate the robot to estimating the environmental state while performing the task. With only an estimate, however, the robot may not perform in a globally optimal manner. The goal, therefore, is to find a good estimate of environmental state (i.e., one close to the true state) given the limitations of the system.

If the environment is stationary, then all of the data the robot gathers may be used to estimate the state, with more data presumably providing a better estimate. Conversely, for non-stationary environmental state estimation, some mechanism must exist for discarding old data. This is a tricky proposition. If too much data are discarded, the estimate may be susceptible to noise and variability; if too little are discarded, the estimate may be skewed and not accurately represent the current state. This is analogous to the issues of overfitting and underfitting generally encountered in machine learning (Mitchell, 1997). The appropriate amount of data to be kept is not necessarily static and pre-determinable, but rather, depends on the variances of the system and the type of non-stationarity exhibited. Low variances require less data (i.e., a smaller timescale) to characterize, as does non-stationarity exemplified by abrupt shifts. Both high variances and gradually shifting non-stationarity require greater amounts of data (i.e., a larger timescale) to characterize.

As an example, consider a normally distributed random variable (time required to find a small mine) in two different systems (one with low small mine density and the other with high density). In one system (the one with high density), the random variable has a theoretical mean of 5.0 (minutes) and variance of 1.0, in the other, a mean of 20.0 (minutes) and variance of 5.0. Using a straightforward confidence interval calculation it is easy to see that establishing identical bounds on sample means requires 5 times (i.e., the ratio of their variances) as much data for the latter system as for the former. Similarly, a shift in the mean requires less data to detect when it is large compared to the variance (i.e., abrupt), than when it is small (i.e., gradual). Thus, a mechanism for estimating environmental state must accommodate both the variances and the type of non-stationarity exhibited by the system. Additionally, since multiple types of relevant non-stationarity and variances may exist in the system, a state estimation procedure capable of dynamically adjusting the amount of data considered is desirable.

We propose an algorithm that provides a moving average estimate of the state of a non-stationary system. The algorithm dynamically adjusts the window size used in the moving average to accommodate the variances and type of non-stationarity exhibited by the system, while discarding outdated and redundant data. Our focus is the application of the algorithm to the problem of reward maximization in a non-stationary environment.

The algorithm uses *augmented Markov models* (AMMs), and builds upon our previous work with AMMs (Goldberg and Matarić, 1999; Goldberg and Matarić, 2001) to demonstrate another, potentially concurrent, application of the model. Multiple AMMs are learned, capturing in real time the dynamics of a robot interacting with its environment in terms of the behaviors it performs. AMMs are created and maintained at different timescales, and statistics about

the environment at those timescales are derived from them. The state of the environment is thus estimated indirectly through the robot’s interaction with it. Similar to many Markov models, individual AMMs cannot explicitly accommodate non-stationary systems because transition dynamics are assumed to be stationary. Maintaining multiple AMMs allows the modeling of average system behavior over different intervals of time (i.e., different data sets), and thus captures the changing nature of the system. As task execution progresses, AMMs are dynamically generated to accommodate new time intervals. Sets of statistics from the models are used to determine whether old AMMs contain redundant/outdated data and thus should be discarded. This approach requires no *a priori* knowledge, uses only local sensing, and captures the notion of timescale. Additionally, it works naturally with stochastic task domains where variations between trials may change the most appropriate amount of data for state estimation.

The rest of the paper is organized as follows. In Section 2, we describe the relationship between Markov chains, semi-Markov processes and AMMs. This motivates the details of our AMM representation presented in Section 3. Section 3 also presents an AMM construction algorithm that is able to dynamically restructure an AMM to maintain a consistent first-order representation of a higher-order system. We follow by considering the application of AMMs to mobile robots in Section 4. Section 5 presents the Markov chain theory applicable to AMMs and derives the statistics that will be used for reward maximization and the dynamic moving average algorithm we present in Section 6. In Section 7, we validate our approach using an implementation of the mine collection task both with a real mobile robot and in simulation. Sections 8 through 10 present the validation results and experimental results for abruptly and gradually changing non-stationary environments. In Section 11, we discuss related work and Section 12 concludes the paper.

2. Markov Chains, SMPs, and AMMs

In this section, we develop the relationship between augmented Markov models (AMMs), Markov chains, and semi-Markov processes in order to provide theoretical context for our use of AMMs. We begin with Markov chains and work towards AMMs.

A discrete, first-order Markov chain is a stochastic process $\{X_m, m = 1, 2, 3, \dots\}$ with a finite or countable state space adhering to the following property:

$$\begin{aligned} P\{X_{m+1} = j \mid X_m = i, X_{m-1} = i_{m-1}, \dots, X_2 = i_2, X_1 = i_1\} \\ = P\{X_{m+1} = j \mid X_m = i\}, \end{aligned} \quad (1)$$

for all states $i_1, i_2, \dots, i_{m-1}, i, j$, and all $m \geq 1$ (Ross, 1992). In other words, the probability that the next state X_{m+1} is j , given the current state ($X_m = i$) and any past state ($X_1 = i_1, \dots, X_{m-1} = i_{m-1}$), is dependent only upon the current state i . In general, a stochastic process that satisfies Equation 1 is said

to be first-order Markovian. If Equation 1 is independent of m , the Markov chain has stationary transition probabilities. The models described in this section all assume stationary transition probabilities.

In an n th-order Markov chain, Equation 1 takes the following form:

$$\begin{aligned} P\{X_{m+1} = j \mid X_m = i, X_{m-1} = i_{m-1}, \dots, X_2 = i_2, X_1 = i_1\} \\ = P\{X_{m+1} = j \mid X_m = i, X_{m-1} = i_{m-1}, \dots, X_{m-n+1} = i_{m-n+1}\}. \end{aligned} \quad (2)$$

We assume that Equation 2 holds for state j and some $n = n_1 \leq m$, and that for all states other than j the equation holds for some $n \leq n_1$. In other words, the probability of the next state is dependent on the current state and at most $n - 1$ previous states.

In a Markov chain, the time spent in each state before a transition to a different state follows a geometric distribution. A semi-Markov process (SMP) is a generalization of a Markov chain allowing arbitrary state durations (Ross, 1992). We let $Q_{ij}(t)$ be the probability of remaining in state i for time $\leq t$ before transitioning to state j . If we let $P_{ij} = Q_{ij}(\infty)$, the P_{ij} define the transition probabilities of the *embedded Markov chain* (Ross, 1992) and it follows that $\sum_j P_{ij} = 1$. We let $F_{ij}(t) = Q_{ij}(t)/P_{ij}$ be the conditional probability of remaining in state i for time $\leq t$ given the system has just entered state i and will transition to state j . (Ross (1992) provides further details on Markov chains and semi-Markov processes.)

An AMM (Goldberg, 2001) is a sub-class of SMP in which the time spent in a particular state is not dependent upon the next state, i.e., $F_{ij}(t) = F_{ik}(t)$ for all states i, j , and k such that $j, k \neq i$. In addition, before a self-transition, the system remains in the current state for exactly one time step, giving:

$$F_{ii}(t) = \begin{cases} 0, & \text{if } t < 0 \\ 1, & \text{if } t \geq 0 \end{cases}$$

Though this formulation makes no assumptions about underlying distributions, the statistical tests we employ later in the paper do assume normal distributions. This constrains $F_{ij}(t)$ to be $\Phi(\mu = 1/(1 - P_{ii}), \sigma^2)$, where Φ is the normal cumulative distribution function (with mean and variance determined empirically).

AMMs provide a compromise between the generality of SMPs and the computational simplicity of Markov chains. They allow standard expectation calculations from Markov chain theory to be easily combined with statistical hypothesis tests such as t and F tests. Though clearly not applicable to every problem, we will see that AMMs work naturally in modeling the interaction dynamics between a mobile robot and its environment in terms of the behaviors executed during a task. Now that we have placed AMMs in the context of Markov chains and SMPs, we provide further detail on the representation used and on our AMM construction algorithm.

3. Augmented Markov Models

We have shown that AMMs are closely related to semi-Markov processes. Unlike perhaps a straightforward SMP representation, our AMM representation incorporates additional statistics that are used during construction and are available for applications. These statistics allow the AMM construction algorithm to dynamically restructure a model in order to capture second-order, or higher, Markovian systems in a first-order form. These statistics are used in conjunction with state splitting to “unfurl” the higher-order transitions into first-order transitions. Maintaining a first-order representation greatly simplifies many expectation calculations, allowing standard Markov chain theory to be employed.

Before continuing, we clarify what an AMM is not. Unlike a Markov decision process (MDP), an AMM is not capable of capturing a decision process, i.e., there is no explicit representation of actions, or a reward signal to indicate the value of an action taken in a particular state. It also does not capture partial observability, as does a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998). In an AMM, the state of the world is assumed to be known, with one exception: the AMM construction algorithm does not decide a priori on the structure (order) of the Markov model best capable of capturing the system. Instead, the order of the model is incrementally generated and state splitting is used to capture hidden state arising from the higher-order nature of the system. Unlike POMDPs and MDPs which can learn controllers, AMMs are perhaps best utilized in gathering statistics about controller execution to be used for improving performance. That is how AMMs are utilized in this work. In the next section we present the representational components of AMMs.

3.1. REPRESENTATION OF AMMs

An AMM is defined as a four-tuple $\langle V, S, A, T \rangle$:

1. V , a set of observation symbols $\{v_1, v_2, \dots, v_M\}$, each associated with at least one state.
2. S , a set of states $\{s_1, s_2, \dots, s_N\}$. Each s_i has three attributes:
 - s_i^v , the symbol that the state recognizes, an element of V ;
 - s_i^μ , the average number of time steps that the system remains in s_i whenever it enters that state;
 - $s_i^{\sigma^2}$, the variance associated with s_i^μ ;
3. A , an $N \times N$ matrix of state transition probabilities. Each element, a_{ij} , is the probability of a transition from state s_i to state s_j , with $\sum_{j=1}^N a_{ij} = 1$.
4. T , a set of structures $\{T^1, \dots, T^{n_{\max}}\}$, each with elements $\{t_1^n, t_2^n, \dots, t_{Q_n}^n\}$ storing information on a particular n -step transition sequence, where $1 \leq n \leq n_{\max}$ and n_{\max} is a user-specified maximum order for the model,

and where self-transitions are not considered. Each element t_i^n has $n + 4$ attributes:

- $t_i^{n,1}, \dots, t_i^{n,n+1}$, the $n + 1$ states defining the transition sequence, excluding self-transitions, i.e., adjacent states in the sequence must be different;
- $t_i^{n,\delta}$, the number of times the transition sequence has been traversed;
- $t_i^{n,\Sigma}$, the total number of time steps that the system has spent in the state $t_i^{n,1}$, after first having traversed the full sequence, $t_i^{n,1:n+1}$;
- t_i^{n,Σ^2} , the sum of the squares of all the durations that comprise $t_i^{n,\Sigma}$.

Given this AMM representation, the transition probabilities of the embedded Markov chain (i.e., the Markov chain generated by assuming that AMM state durations follow a geometric distribution) are given by A . The addition of s^μ and s^{σ^2} provides the more general state duration capabilities of an SMP. T is used in incremental model generation and dynamic model reconfiguration with state splitting. We present the model construction algorithm next.

3.2. GENERATION OF AMMS

The data used for constructing an AMM consist of a stream of symbols belonging to V . In our implemented robot behavior model, the symbols are the names of behaviors that the robot performs, sampled at some appropriately high frequency to avoid aliasing. Construction of an AMM proceeds according to the following algorithm:

1. Initialize the model by creating an initial state, s_\emptyset . This state provides a starting point for model construction. No transitions are ever made to s_\emptyset . After initialization, V is empty (no symbols have been seen yet) as are A and T (no transitions have occurred).
2. Designate the current input symbol to the system as v_c , and the current state as s_c .
3. If v_c has not been seen before, i.e., $v_c \notin V$:
 - $V \leftarrow V \cup \{v_c\}$.
 - Create a new state, s_n , with $s_n^v \leftarrow v_c$, $s_n^\mu \leftarrow 0$, and $s_n^{\sigma^2} \leftarrow 0$. Set $a_{cn} \leftarrow 1 / \sum t_*^{1,\delta}$, where the applicable values from T are those with $t_*^{1,1} = s_c$. Readjust all other transition probabilities, a_{c*} . (Note: ‘*’ represents all values.) Create all of the necessary new transitions in T having $t_*^{*,1} = s_n$, initializing $t_*^{*,\delta} \leftarrow 1$, $t_*^{*,\Sigma} \leftarrow 0$, and $t_*^{*,\Sigma^2} \leftarrow 0$.
 - Update the information about s_c , including all $t_*^{*,\Sigma}$ and t_*^{*,Σ^2} that have $t_*^{*,1} = s_c$. Use these updated values to recompute s_c^μ and $s_c^{\sigma^2}$.

according to the formulae:

$$\begin{aligned} s_c^\mu &= \Sigma/\delta, & \text{if } \delta > 0 \\ s_c^{\sigma^2} &= \frac{\Sigma^2 - (\Sigma)^2/\delta}{\delta - 1}, & \text{if } \delta > 1. \end{aligned}$$

where δ is the total number of transitions to s_c , Σ is the total time steps spent in s_c , and Σ^2 is the sum of the squares of the individual values comprising Σ , as obtained from T . Note that $(\Sigma)^2$ is different from Σ^2 .

- Go to step 6 to check consistency of the model.
4. If v_c is the same as the last input symbol:
 - Remain in the current state, i.e., $s_n = s_c$.
 - Readjust all of the transition probabilities, a_{c*} , to account for the increase in a_{cc} .
 - Go to step 7.
 5. If the current input symbol has been seen before, i.e., $v_c \in V$, but is different from the last symbol:
 - Transition to a state, s_n , that accepts v_c .
 - Update the appropriate values in S , A , and T , similarly to what is done in step 3, except that no new state is created.
 6. State splitting:

When about to transition from s_c to s_n , do the following for each order n , $2 \leq n \leq n_{max}$:

 - Calculate the total number of traversals ($\sum t_*^{n,\delta}$) made along all $t_*^{n,*}$ that have $t_*^{n,2} = s_c$ and $t_*^{n,3:n+1}$ equal to the previous $n - 1$ states. Based on this total, calculate the transition probabilities for s_c and their associated binomial confidence intervals (Blyth, 1986).
 - If the actual number of n th-order traversals does not fall outside of the confidence intervals, then there is no n th-order inconsistency in the model. If all orders have been checked, go to step 7.
 - If the actual number of n th-order traversals does fall outside of the confidence interval, then there is an n th-order inconsistency in the model and the current state s_c as well as the previous $n-2$ states must be *split*. Create the $n - 1$ new states and re-attribute the transition and duration information in T pertaining to these states. In addition, create all necessary new elements in T and adjust the transition and duration data in these and other elements of T that are affected. Expand matrix A to account for the new states, and calculate the new transition probabilities.

7. Transition to the new state, setting $s_c \leftarrow s_n$.
8. Go to step 2, repeating until there are no more input symbols.

Step 6 of the algorithm describes *state splitting* and deserves further explanation. Since an AMM is constructed incrementally from incoming data, it is important that there be some mechanism for model modification when new data invalidate the current structure of the model. This mechanism is provided by state splitting, utilizing data from T to ensure that the model remains consistent. State splitting allows an n th-order traversal sequence to be represented in a first-order model, making the model intuitively easier to understand and simplifying model calculations. A first-order model is easy to understand because its behavior is fully determined by the current state, i.e., one is not required to keep in mind the history of transitions. Calculations are simplified for a similar reason, i.e., statistical dependencies are first-order.

One issue to note is the potential for erroneous state splitting due to Type I and Type II errors in our hypothesis test using binomial confidence intervals (step 6). Given a fixed amount of data, there is a tradeoff between incorrectly splitting states (Type I error) and not splitting states that should be split (Type II error) (Freund, 1992). In the extreme, being 100% certain that no erroneous state splitting occurs means performing no state splitting at all.

The computational complexity of the algorithm per input symbol is at most $O(N^n)$ (for a fully connected n th-order model of N states) with state splitting, and at most $O(N)$ otherwise. The space complexity is $O(N^n)$ for N fully connected states. Though certainly dependent on the application, we have noticed that, in practice, graphs tend to be fairly sparse and thus computational and space complexity tends towards $O(N)$. We now present an example of AMM construction.

3.3. EXAMPLE OF AMM CONSTRUCTION

Consider the sequence of input symbols $\{3\ 2\ 1\ 4\ 2\ 1\ 3\ 2\ 1\ 4\ 2\ 1\ 3\ 2\ 1\ 4\ 2\ 1\ \dots\}$ that alternates between occurrences of $\{3\ 2\ 1\}$ and $\{4\ 2\ 1\}$. Figure 1 gives an example of an AMM, capable of capturing up to second-order transitions, generated with 100 symbols from this sequence. The key item to note in the figure is that from state #4 (accepting symbol 1) there is a 0.5 probability of transitioning to state #2 or state #5, and thus generating $\{3\ 2\ 1\}$ or $\{4\ 2\ 1\}$. We know, however, that this is an inaccurate representation of the system since there can never be two consecutive occurrences of either $\{3\ 2\ 1\}$ or $\{4\ 2\ 1\}$. Because the next state after #4 depends on two states before, the system is third order. In contrast, Figure 2 shows the first-order representation generated by a third-order AMM constructed with the same sequence. One third-order state split gives two additional states: one that accepts symbol 1 and one that accepts symbol 2. This new first-order representation accurately represents the symbol sequence.

This illustration of third-order AMM construction demonstrates the case where there is an inconsistency just in the link traversals, whereas a more subtle case might also involve inconsistencies in the traversal probabilities.

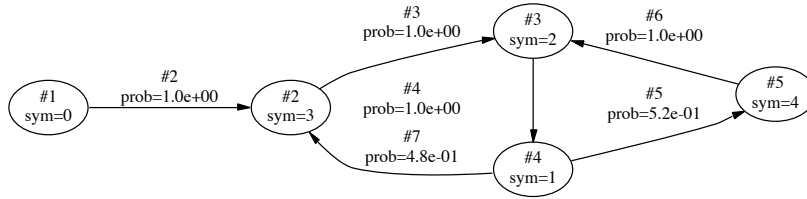


Figure 1. A example of a first-order or second-order AMM generated with 100 input symbols from the sequence $\{3\ 2\ 1\ 4\ 2\ 1\ 3\ 2\ 1\ 4\ 2\ 1\ \dots\}$

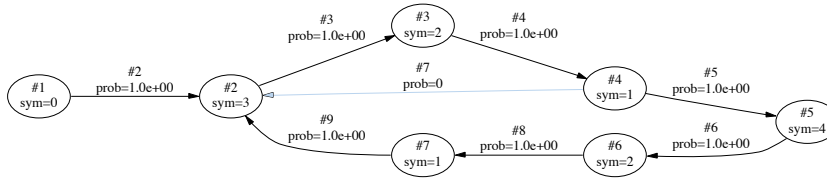


Figure 2. A example of a third-order AMM generated with 100 input symbols from the sequence $\{3\ 2\ 1\ 4\ 2\ 1\ 3\ 2\ 1\ 4\ 2\ 1\ \dots\}$

Next, we consider in more detail the application of AMMs to the mobile robotics domain.

4. Model Use with Mobile Robots

AMMs can be constructed and modified as a mobile robot is performing a task, in order to capture the dynamics of its interaction with the environment. The ability of our AMM construction algorithm to capture higher-order dynamics provides part of the expressiveness required to model the richness of interaction. The use of behavior-based control (BBC) as a substrate for AMM construction provides the remaining representational expressiveness.

Behavior-based control (BBC) is a paradigm for constructing controllers for situated agents (Brooks, 1991; Matarić, 1992). In BBC, a controller is organized as a collection of processing modules, called *behaviors*, that receive input from sensors and/or other behaviors, process the input (possibly modifying internal state), and send output to effectors and/or other behaviors (Figure 3). Each behavior generally serves some coherent, independent goal-achieving or goal-maintaining function, such as *avoiding* obstacles or *homing* to a destination. All behaviors in a controller are executed asynchronously and in parallel, simultaneously receiving input and producing output. An action selection mechanism prevents conflicts when signals are simultaneously sent to the same actuators or behaviors (Pirjanian, 1998). Behavior-based control has

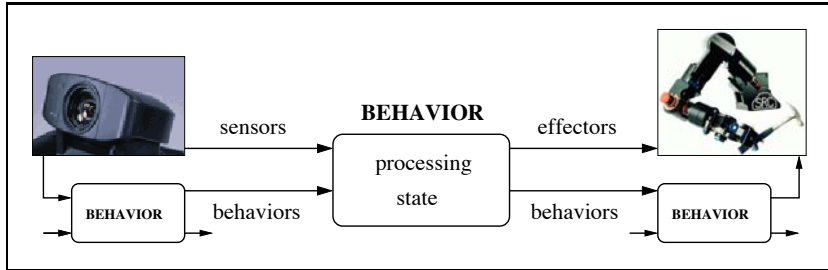


Figure 3. The basic structure of a behavior-based controller. Behaviors receive input from sensors and other behaviors, process the input possibly changing internal state, and send outputs to effectors and other behaviors. (Sensors are represented by the Sony pan-tilt-zoom camera on the left, and effectors by the Sarcos Dextrous Arm on the right.)

proven to be an effective paradigm for developing single-robot and multi-robot controllers (Matarić, 1997; Arkin, 1998).

AMMs with BBC are the synergistic combination integral to the work in this paper. AMMs provide the ability for on-line, real-time model construction in higher-order Markovian systems, while BBC provides the representational richness for capturing interaction dynamics. In addition, because BBC abstracts low-level sensing and action into behaviors with coherent functions, it both provides a parsimonious space for AMM construction and facilitates interpretation of the models. In this paper, we *use the states of an AMM to represent the execution of individual behaviors of a controller*. One caveat in using BBC is that, because the controllers can carry state with extended history that impacts behavior execution, the interaction dynamics captured in terms of behaviors may not be (first-order) Markovian (Whitehead and Lin, 1995). The ability of our AMM construction algorithm to represent higher-order Markovian systems, however, compensates for this.

In order to use BBC with AMMs, one must determine an appropriate *behavior space*, i.e., a set of behaviors that do not overlap in execution, that continuously describe the robot’s activity, and that are uniquely labeled. One of these labels or symbols is sent to the model generation algorithm at each time step as an indication of the behavior that is currently active. It is important to note that the algorithm only accepts one symbol per time step. If the robot’s control system is structured so as to utilize simultaneous execution of two or more behaviors, the AMM algorithm will only be able to consider one of their symbols as input. Unless that one symbol is consistently used to represent the parallel execution of both behaviors, the result will be a model unrepresentative of the actual behavior dynamics. If no behavior is executing, a special symbol representing a “null” behavior should be sent to the algorithm.

Thus, the symbols of the behavior space should represent the non-overlapping execution of a collection of behaviors that account for all of the robot’s time/activity. More precisely, in behavior space $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_K\}$, no two behaviors \mathcal{B}_i and \mathcal{B}_j are simultaneously active. (It is possible, however, for

an individual behavior, \mathcal{B}_i , to represent the concurrent execution of multiple behaviors in the controller.) In addition, if \mathcal{T}_{tot} is the total time that the robot is active, and $\mathcal{T}_{\mathcal{B}_i}$ is the total time that behavior \mathcal{B}_i is active, then $\mathcal{T}_{\text{tot}} = \sum_{i=1}^K \mathcal{T}_{\mathcal{B}_i}$. The behavior space \mathcal{B} need not contain all of the behaviors that the robot may exhibit, only a subset that adheres to these criteria and is appropriate to the problem under consideration.

In the unlikely worst case scenario where all of the behaviors on a robot are executed in parallel for \mathcal{T}_{tot} , or where there is only one behavior, \mathcal{B} also consists of exactly one behavior. The AMM generated in this degenerate case has one non-initial state that it never leaves, and consequently is of no practical use. Fortunately, the large majority of current approaches to robot control, ranging from hybrid to behavior-based, utilize sequences and priorities over the actions and behaviors executed on the robot (Matarić, 1997; Gat, 1998). As long as the behavior of the robot can be decomposed into a behavior space, AMMs apply as a modeling tool.

In the next section, we describe how to derive AMM statistics used later in our moving average algorithm for state estimation.

5. AMM Statistics

We derive several useful statistics from an AMM that we will subsequently use in our reward maximization criterion and our dynamic moving average algorithm using multiple AMMs.

One such statistic is the expected number of time steps the system takes to reach a destination state from a given start state. This is known as the *mean first passage* (Roberts, 1976). Markov chain theory provides tools for easily calculating such expectations. We apply these tools to AMMs, then use the results to calculate two other statistics: the total variance associated with the mean first passage, and the accompanying degrees of freedom. (A more detailed treatment of Markov chain theory, including proofs and derivations, is available in Roberts (1976) and Kemeny et al. (1966)).

5.1. MEAN FIRST PASSAGE

The first step in calculating the mean first passage is to extract the $N \times N$ transition matrix P of an AMM M with N non-initial states. P is identical to the transition matrix A except that all entries involving s_0 are removed. Each element p_{ij} of P is thus the probability of transitioning directly to state s_j given that the system is in state s_i .

In order to be able to continue the calculation of the mean first passage, P must represent a *regular* Markov chain, i.e., every state in the underlying directed graph of P must reach every other state in some constant integer, κ , number of steps. If P is *ergodic* (i.e., every state can reach every other state) and it has a non-zero main diagonal, then it is also regular (Roberts, 1976). We can determine that a directed graph is ergodic by showing that it has one strongly connected component (see Cormen et al. (1990) for an algorithm).

Given that P is regular, we calculate the stationary vector $w = w_1, \dots, w_N$ giving the probability of being in each state s_1, s_2, \dots, s_N of M in the limit. w is found by solving the system of equations $wP = w$ with the constraint that $w_1 + w_2 + \dots + w_N = 1$. Using w , we calculate the *fundamental matrix* $Z = [I - (P - W)]^{-1}$, where W is the square matrix with w as each row. Z now enters the calculation of the mean first passage matrix $E = (I - Z + JZ_{\text{dg}})D$, where J is a matrix of 1's, Z_{dg} is the diagonal matrix containing the main diagonal of Z , and D is the diagonal matrix with $d_{ii} = 1/w_i$. Each element e_{ij} of E represents the mean first passage from state s_i to s_j .

In terms of AMMs, we are interested in the mean first passage from an input symbol v_i to another input symbol v_j . Since each input symbol may be recognized by several states in the AMM, the mean first passage between two input symbols may not be unique. In general, if n states recognize v_i and m states recognize v_j , then nm entries in E represent the mean first passage between these symbols. In such a case, our interest is in the *minimum mean first passage between two input symbols* and the corresponding states that give this value in E . For symbols v_i and v_j , let ε_{ij} be this minimum value from E . Let s_α and s_β be states such that $e_{\alpha\beta} = \varepsilon_{ij}$.

Once we know the states associated with the minimum mean first passage, we can calculate the expected amount of time spent in each state before reaching s_β . To do so, we first convert our regular Markov chain into an absorbing chain with s_β as an absorbing state. Essentially, this means modifying P so that once the system enters state s_β it does not leave it, i.e., $a_{\beta\beta} = 1.0$. The transition matrix P is converted into a new matrix Q containing only transitions to and from the $N - 1$ non-absorbing states by simply deleting the β -th row and column from P . Note that $\alpha \neq \beta$. The fundamental matrix R for the absorbing chain is given by $R = (I - Q)^{-1}$, where each element r_{ij} represents the expected amount of time spent in the j -th non-absorbing state, given that the system starts in the i -th non-absorbing state, and

$$e_{\alpha\beta} = \begin{cases} \sum_{j=1}^{N-1} r_{\alpha j}, & \text{if } \alpha < \beta \\ \sum_{j=1}^{N-1} r_{(\alpha-1)j}, & \text{if } \alpha > \beta. \end{cases}$$

5.2. VARIANCE OF MEAN FIRST PASSAGE

In order to calculate the variance associated with a value in the mean first passage matrix, we first note that the embedded Markov chain of an AMM may be interpreted as a set of random variables, one for each state of the chain. Generally, it is assumed that these random variables are independent, identically distributed, and follow a normal distribution. Our calculation of variance also assumes independence and a normal distribution, but to be more general it allows non-identical distributions. Given a set of independent random variables $\{X_1, X_2, \dots, X_N\}$ with each X_i having an associated population mean μ_i and variance σ_i^2 , we wish to calculate the mean and variance

for the linear combination $Y = c_1X_1 + c_2X_2 + \dots + c_nX_n$. Basic results from statistics tell us that the mean of Y is $\mu_Y = \sum_{i=1}^N c_i\mu_i$ and the variance of Y is $\sigma_Y^2 = \sum_{i=1}^N c_i^2\sigma_i^2$. In our case, we do not know the population means and variances and instead use the sample means and variances calculated for each state of the AMM, i.e., s_i^μ and $s_i^{\sigma^2}$.

Given $e_{\alpha\beta}$, s_α , s_β , and the fundamental matrix R generated by making s_β an absorbing state, we calculate the variance associated with $e_{\alpha\beta}$. As shown previously, $e_{\alpha\beta}$ is equivalent to the sum of the row of R associated with s_α . Equivalently, $e_{\alpha\beta}$ may be expressed as a linear combination of the s_i^μ extracted from the AMM:

$$e_{\alpha\beta} = \sum_{i=0}^N c_i s_i^\mu$$

where

$$c_i = \begin{cases} r_{\alpha i}/s_\alpha^\mu, & \text{if } \alpha < \beta \text{ and } i < \beta \\ r_{\alpha-1,i}/s_\alpha^\mu, & \text{if } \alpha > \beta \text{ and } i < \beta \\ r_{\alpha,i-1}/s_\alpha^\mu, & \text{if } \alpha < \beta \text{ and } i > \beta \\ r_{\alpha-1,i-1}/s_\alpha^\mu, & \text{if } \alpha > \beta \text{ and } i > \beta \\ 0, & \text{if } i = \beta \end{cases}$$

Now that we have expressed $e_{\alpha\beta}$ as a linear combination of the means of the random variables composing the AMM, we can do similarly for the variance of $e_{\alpha\beta}$:

$$\text{var}(e_{\alpha\beta}) = \sum_{i=0}^N c_i^2 s_i^{\sigma^2}$$

with the c_i 's calculated as above.

5.3. DEGREES OF FREEDOM

The number of degrees of freedom associated with the linear combination $Y = \sum_{i=1}^N c_i X_i$ may be expressed as

$$\frac{\left[\sum_{i=1}^N \frac{c_i^2 \text{var}(X_i)}{V_{X_i}} \right]^2}{\sum_{i=1}^N \frac{[c_i^2 \text{var}(X_i)/V_{X_i}]^2}{V_{X_i} - 1}}$$

where V_{X_i} is the sample size used in the calculation of $\text{var}(X_i)$. This is an expanded version of the formula found in Press et al. (1992). To apply this to our calculation of the variance of $e_{\alpha\beta}$ we simply use $c_i^2 s_i^{\sigma^2}$ in place of $c_i^2 \text{var}(X_i)$, with the c_i 's calculated as above, and with

$$V_{X_i} = \sum_{\{i|t_i^{1,1}=s_i\}} t_i^{1,\delta}$$

where V_{X_i} is the total number of transitions made to state s_i .

5.4. THE t -TEST AND F -TEST

Given the mean first passage values in E and their associated variances and degrees of freedom, we can perform two standard tests of statistical significance: the t -test and F -test (Freund, 1992). The t -test uses Student's t distribution to determine whether the difference between two means is significant. Calculation of this test requires the two mean values and their combined degrees of freedom derived above. The F -test uses the F distribution to determine if the variances of samples from two normal distributions are significantly different given their degrees of freedom, i.e., the amount of data in the samples. These two tests can be used to compare values for mean first passage and variance within the same AMM and across AMMs. Press et al. (1992) provide more details on the computation of these tests. See Peizer and Pratt (1968) and Ling (1978) for computationally efficient approximations to F and t tail probabilities for use in these tests.

In the next section, we use AMMs and the statistics derived above in our dynamic moving average algorithm.

6. Dynamic Moving Average Algorithm

To manage the amount of data used to estimate the state of a non-stationary environment, we present an algorithm that essentially computes a moving average with a dynamic window size, increasing and reducing the amount of data used in the moving average as necessary. The algorithm maintains multiple AMMs for different time intervals, and uses a t -test and F -test on comparable values from the different AMMs to determine which AMM provides the best information. These tests allow the algorithm to adjust the window size of the moving average to accommodate both the amount of variance in the system and the type of non-stationarity (ranging from abrupt to very gradual). The window size of the moving average is allowed to grow by maintaining and expanding old AMMs, and is shrunk by deleting AMMs. We now present the algorithm, followed by a discussion of some of its features.

1. Let \mathcal{L} be a queue of AMMs, initialized to contain one model. We designate the first AMM in the queue as \mathcal{L}_0 , and subsequent AMMs as $\mathcal{L}_1, \mathcal{L}_2, \dots$. When \mathcal{L}_0 is removed from the queue, the designations shift so that \mathcal{L}_1 becomes \mathcal{L}_0 , and so forth.
2. Let ϖ_t and ϖ_F be constants specifying the significance levels for the t -test and F -test, respectively.
3. Let *mode* be a variable designating the two modes of the algorithm.
4. For each new input symbol do the following:

5. Update each AMM in \mathcal{L} with the new input.
6. If a new AMM is required, then:
 7. Create a new AMM and add it to the end of \mathcal{L} .
 8. Update the new AMM with the input.
 9. Compute the mean first passage matrix for \mathcal{L}_0 . Extract the desired values and calculate their associated variances and degrees of freedom.
 10. Do the same for \mathcal{L}_1 .
 11. Perform an F -test between the variances calculated for \mathcal{L}_0 and \mathcal{L}_1 .
 12. If the significance level returned by the F -test is less than ϖ_F (i.e., the variances are different) then set $mode=1$, else set $mode=2$.
 13. If $mode=1$, then let i be the index of the first AMM in \mathcal{L} after \mathcal{L}_0 that has either significantly different variances or significantly different means (i.e., significance level $< \varpi_F, \varpi_t$) as compared to \mathcal{L}_0 . If such an i exists, delete \mathcal{L}_0 through \mathcal{L}_{i-2} and use the new \mathcal{L}_0 as the best estimate of the state.
 14. If $mode=2$, then let i be the index of the first AMM in \mathcal{L} after \mathcal{L}_0 that has neither significantly different variances nor means (i.e., significance levels $> \varpi_F, \varpi_t$) as compared to \mathcal{L}_0 . If such an i exists, delete \mathcal{L}_0 through \mathcal{L}_{i-1} and use the new \mathcal{L}_0 as the best estimate of the state.
 15. If no such i exists for either value of $mode$, then do not delete any AMMs and use the current \mathcal{L}_0 as the best estimate.

There are several characteristics of the algorithm worth noting. Each AMM in \mathcal{L} provides an (average) estimate with a different window size. \mathcal{L}_0 has the largest window size because it is generated with the most data and provides the current best guess for the state of the system. When \mathcal{L}_0 is deleted, the average “moves” to the next AMM and the window size decreases. When \mathcal{L}_0 is not deleted and is instead grown to incorporate more data, the window size of the average expands, but the average does not “move.” The key to maintaining a good estimate of the state of the system in \mathcal{L}_0 is to allow the AMM to incorporate enough data to provide a good average, but not so much that the average becomes skewed with old data.

The algorithm attempts to do this by adjusting the amount of data in the moving average to accommodate the variance in the system. This is accomplished by considering deletion of \mathcal{L}_0 , the AMM representing the largest time window of data, only when its variance is comparable to (i.e., not significantly different from) that of another AMM. When the variability in the system is

high, the AMMs require more data (i.e., larger windows) to acquire comparable variances. When variability is low, less data are required to accurately characterize the variances of the system.

The algorithm has two distinct modes. In the first mode ($mode=1$), the algorithm removes redundant/old data. In systems with very gradual non-stationarity, this mode effectively maintains a good moving average estimate of the state. When there is an abrupt change in the system, the comparable means of adjacent AMMs in \mathcal{L} may become statistically different as more data are collected, causing the first mode to stall (i.e., not delete old AMMs). The second mode ($mode=2$) solves this problem by comparing non-adjacent AMMs to find two that are not significantly different. The second mode then “jumps over” the intervening AMMs by deleting them.

The decision criterion used to create a new AMM (line 6) is very general. For example, a new AMM might be created after a certain period of time, a certain number of input symbols, or when a particular input symbol is observed. In the experiments described below, a new AMM is created every time the robot finds a mine (puck) to collect. The decision criterion for creating new AMMs, in conjunction with the characteristics of the system (variability, type of non-stationarity) determine how many AMMs are maintained by the algorithm, which determines the computational complexity of the algorithm. In the experiments described later, we observed the algorithm to maintain a small number of AMMs (e.g., between 5 and 15).

One final item of note is the importance of the ϖ_t and ϖ_F thresholds for the significance level. Both values must be in the interval $[0, 1.0]$, with a significance level of 0.05, or less, generally considered significant. The effect of ϖ_t and ϖ_F is to adjust the size of the moving average window. Extremely large values of both thresholds produce a very large window with excessive smoothing and a potentially skewed estimate of state. Very small values of the thresholds result in a small window and state estimation that is prone to overfitting. Empirical tests suggest that values in the interval $[0.01, 0.1]$ tend to work well, with relatively little sensitivity. It should also be noted that the experiments described later use this algorithm in real time.

In Section 8, we present our reward maximization criterion, and in Sections 9 and 10, we show how the use of the dynamic moving average algorithm just presented improves performance in non-stationary environments. First, we describe our experimental mine collection task and the associated behavior space.

7. Mine Collection Task

To validate our dynamic moving average algorithm and its application to reward maximization, we use a task analogous to the mine collection example of the Introduction. The experiments are implemented on a real mobile robot and in simulation.

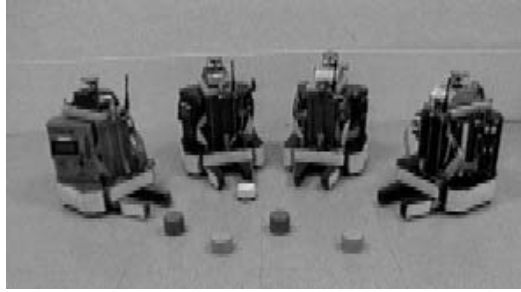


Figure 4. The four IS Robotics R2e robots used in the experiments. Each robot is approximately 1 foot (25 cm) along each dimension.

7.1. THE ROBOT

We used four IS Robotics R2e robots (Figure 4) in the experiments, not concurrently, but one at a time. Each robot is a differentially-steered base equipped with two drive motors and a two-fingered gripper. The sensing capabilities include piezo-electric contact sensors around the base and in the gripper, five infrared (IR) sensors around the body and one on each finger, a color sensor in the gripper, a radio transmitter/receiver for communication and data gathering, and an ultrasound triangulation system for positioning. The robot is programmed in the Behavior Language (Brooks, 1990). Experiments were performed in an 11 × 14 foot rectangular enclosure (the Corral). The Corral had up to 36 small plastic cylinders (pucks) of two different colors: clear (representing large mines) and black (representing small ones). The pucks were evenly distributed throughout the Corral, except in the drop-off area, called Home, a ninety degree sector of a circle with a radius of two feet, located in one corner. (Figure 5).

7.2. BEHAVIORS

The Behavior Language, used for programming the robots, provides a natural structure for implementing behavior-based controllers. Behaviors are defined as sets of concurrent, real-time, asynchronous rules receiving inputs from sensors and other behaviors, and sending outputs to actuators and other behaviors. A set of these behaviors can easily map to the behavior space for an AMM, as with our experimental mine collection task. The following 9 behaviors constituted the behavior space for our task:

- **wandering**: move forward and, at random intervals, turn left or right through some random arc.
- **avoiding**: avoid any object (detected by IR and contact sensors) deemed to be in the path of the robot.
- **puck detecting**: if avoiding is not active, and if an object is detected by the front IRs, lift up the gripper fingers to determine whether the object

is short enough to be a puck. If it is, approach the object and try to place it between the fingers and pick it up. If unsuccessful, perform **avoiding**.

- **color detecting**: if **puck detecting** has picked up a puck, detect the color of the puck. If it is the desired color, then perform **homing**, else perform **leave puck**.
- **leave puck**: drop the puck and continue searching for more, using **avoiding**, **wandering** and **puck detecting**.
- **homing**: if carrying a puck, move towards the designated goal, Home.
- **creeping**: when near Home, perform a slower, more accurate homing behavior.
- **exiting**: if in the Home region, drop puck and exit Home.
- **reverse homing**: move away from the Home region.

The **color detecting**, **homing**, **avoiding**, and **creeping** behaviors are all parametrized to indicate the color of puck that the robot has found, so that a different input symbol is fed to the AMM construction algorithm depending on that color/type. This qualification allows us to calculate the expected time for finding and delivering each type of puck/mine from the AMM statistics. Control of the robot’s drive motors was the basis for selecting the constituent members of this behavior space. When active, each of the behaviors has exclusive control of the motors, and together they account for all activity (or inactivity) of the motors for the duration of the task. We now present the validation experiment and results for the reward maximization criterion.

8. Experiment 1: Validation of the Reward Maximization Criterion

Before demonstrating reward maximization in a non-stationary version of the mine collection task, we first validate the reward maximization criterion in a stationary environment. This validation is necessary to ensure that our subsequent results are not biased by an invalid assumption about the value of reward maximization. If it were shown that our reward maximization criterion did not improve performance over random behavior, then the utility of our dynamic moving average algorithm in the non-stationary version of the task would be suspect.

We now more formally define the reward maximization criterion. Let \mathcal{R}_s and \mathcal{R}_l be the rewards for small and large mines, respectively. Let τ_s^f be the expected time required for the robot to find a small mine, and let τ_s^d be the expected time to deliver the mine to the goal location once it has been found. Similarly, τ_l^f and τ_l^d represent these times for large mines. The robot maximizes its reward by deciding for each mine found whether to deliver it

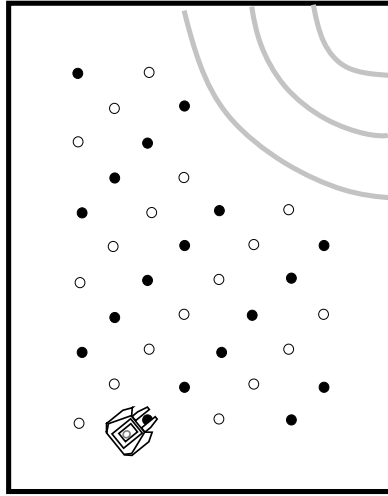


Figure 5. The mine collection task: setup for validation of the reward maximization criterion.

or leave it in search of a higher-valued one. The action chosen is the one that maximizes the expected reward per unit time (and thus the overall expected reward). If the robot finds a small mine, and the inequality

$$\frac{\mathcal{R}_s}{\tau_s^d} > \frac{\mathcal{R}_l}{\tau_l^f + \tau_l^d}$$

holds, then delivering the small mine maximizes reward. The inequality states that the reward per unit time for delivering the small mine that has been found should be greater than the reward per unit time for leaving the small mine to find and deliver a large one. If this inequality does not hold, the small mine should be left and a large mine sought. The complementary inequality

$$\frac{\mathcal{R}_l}{\tau_l^d} > \frac{\mathcal{R}_s}{\tau_s^f + \tau_s^d}$$

is used when a large mine is found.

The main issue is calculating τ^d and τ^f for each mine type. One could maintain internal variables that record these values. Our approach, however, is to calculate these values from the robot's AMM. As discussed previously, each element, e_{ij} , of E gives the mean first passage from state i to state j . E also contains the values for τ^f and τ^d . τ^f is simply the entry in E associated with the minimum mean time from a **wandering** state to a **color detecting** state, and τ^d is the minimum mean time from a **color detecting** state to a **wandering** state. In other words, if v_1 is the input symbol for the **wandering**

behavior and v_2 is the input symbol for the **color detecting** behavior when a large mine is found, then $\tau_i^d = \varepsilon_{21}$ and $\tau_i^f = \varepsilon_{12}$.

We performed two sets of experiments: one control set where the robot collected both types of pucks without discrimination, and one set with reward maximization allowing the robot to decide which pucks to collect. For these experiments, reward values were set in proportion to a mine’s explosive power, thus making reward maximization identical to minimizing the mine field’s destructive potential. The setup for both sets of experiments was identical, with 18 clear pucks (large mines) and 18 black pucks (small mines) evenly distributed in the Corral (Figure 5). Each clear puck had a reward of 4 points, while each black had a reward of 1 point. In addition, the environment was kept stationary by replenishing collected pucks. We performed five one-hour trials for each experiment. Using the reward maximizing criterion, the robot accrued an average of 46.6 points, while without reward maximization the robot averaged 37 points. A hypothesis test based on Student’s t indicates that the means are different at a significance level of 5%, and validates our reward maximization criterion. One issue of importance in this experiment and the following ones is the designation of reward values for pucks. Because of the time required to find a puck, setting reward values too close (e.g., 1 point and 2 points) could preclude ever satisfying the reward maximization criterion, making reward maximization a moot issue. Setting reward values too far apart (e.g., 1 point and 1000 points) could trivialize the problem. Thus, reward values in our laboratory experiments were set to a compromise that allowed for statistically significant results within a reasonable number of experimental trials. In a real world setting, reward values would likely be set to meaningful values without concern for their relative magnitudes.

We now present experimental results from applying the dynamic moving average algorithm to an abruptly changing non-stationary version of this task.

9. Experiment 2: Abruptly Changing Environment

In this set of experiments, we aim to show that, in an abruptly changing non-stationary version of the mine collection task, reward maximization with the addition of the dynamic moving average algorithm is superior to reward maximization alone. The hypothesis is that when the environment changes, average values of τ^f and τ^d become inaccurate and thus not effective for reward maximization. Using a dynamic moving average allows us to maintain effective estimates of τ^f and τ^d and to adapt to environmental changes relatively quickly.

The environment was first initialized to contain only 18 clear pucks (Figure 6). We ran one trial of approximately 20 minutes in this environment, during which the robot collected 4 clear pucks. This result is extremely variable: the actual time to collect 4 pucks could easily range between 10 and 30 minutes. In order to reduce the variability, the data from this one trial were used as a primer for all of the subsequent experiments, in which the white pucks (large mines) were replaced with black (small) ones. Doing so allowed

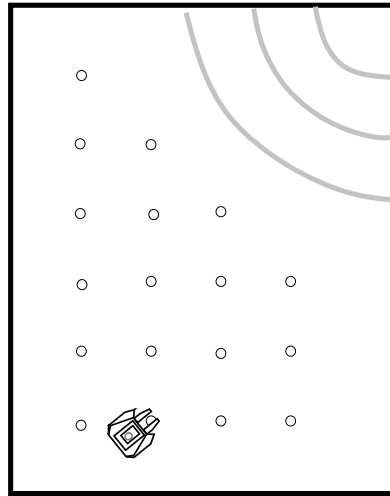


Figure 6. The mine collection task: setup for reward maximization in a non-stationary environment.

us to focus on the key experimental parameter: the time required to adapt to the new environment. We considered the robot to have adapted to the new environment when it consistently began collecting black pucks.

We ran two experiments using reward maximization with the dynamic moving average algorithm and three experiments using reward maximization alone. The reward for a white puck was 7 points and the reward for a black puck was 1 point. For reward maximization alone, the mean time to adaptation was 47 minutes, while the mean time with the algorithm was 18.3 minutes. A *t*-test indicates that these means are different at a significance level of 0.01. This strongly supports our hypothesis that the dynamic moving average algorithm allows faster adaptation to abrupt non-stationarities.

10. Experiment 3: Gradually Shifting Environment

In this set of experiments, we test the effectiveness of the dynamic moving average algorithm for reward maximization in a gradually shifting environment. Instead of abruptly changing the color of the pucks, as in the previous experiment, here the environment shifts slowly as the robot collects pucks. Due to the high degree of variance in the mine collection task using physical robots, we anticipated that the number of experiments we would have had to conduct in order to obtain statistical significance in the gradually shifting environment would have posed a practical impossibility. The experiments described in this section were therefore conducted in simulation.

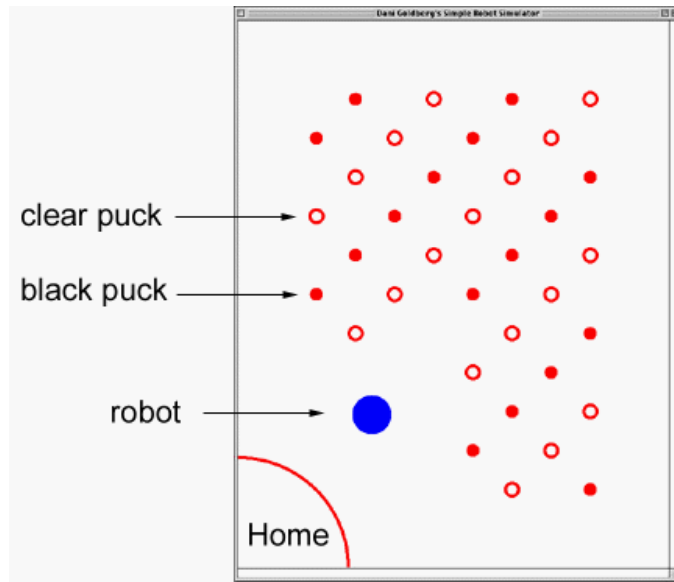


Figure 7. The simulated mine collection task: setup for reward maximization in a gradually shifting non-stationary environment.

The simulation was initialized to contain 18 clear pucks worth 10 points each, and 18 black pucks worth 1 point each (Figure 7). Three experimental scenarios were examined:

1. **random**: the robot collects any puck it encounters regardless of color.
2. **control**: the robot uses the reward maximization criterion, but does not employ the dynamic moving average algorithm used to compensate for non-stationarity.
3. **algorithm**: the robot uses both reward maximization and the dynamic moving average algorithm for state estimation.

We conducted trials of 50,000 simulation steps for each scenario: 200 trials of the **random** scenario, 40 of **control**, and 100 of **algorithm**, with the actual number determined by the desired level of statistical significance. The data gathered included the time at which each puck was collected, allowing us to calculate the accrued number of reward points. Figure 8 presents the average number of reward points accrued at 1000-time-step intervals for each of the three scenarios. The maximum possible accrued reward is 198 points, corresponding to the collection of all 36 pucks. Both the **random** and **algorithm** scenarios essentially reach this maximum by the end of each trial (with average accrued points of 197.8 and 197.5, respectively). The **control** scenario outperforms the other two until around 20,000 time steps, then quickly declines, ending with an average reward of 184.4. Because the

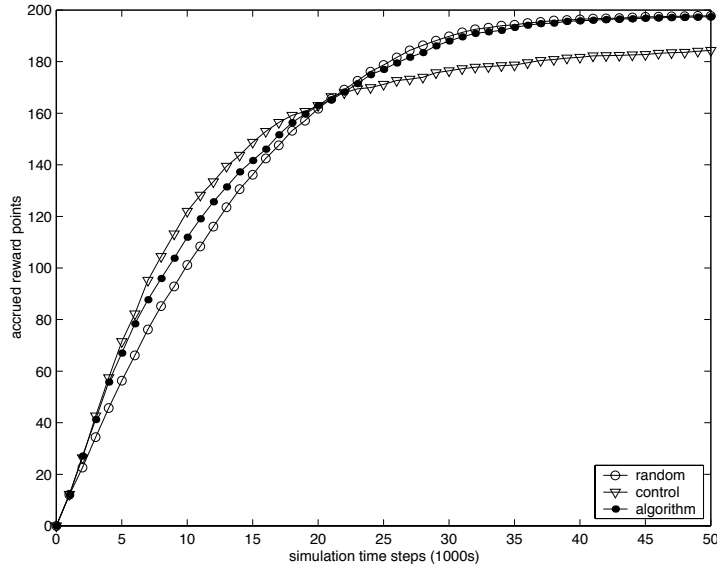


Figure 8. Average accrued reward for the three experimental scenarios (puck point values: black=1, clear=10).

control scenario blindly collects only clear pucks, it does somewhat well at the beginning of the trials, but its inability to adapt (as the **algorithm** scenario can) to the changing environment causes large performance degradation once clear pucks become depleted. The discrepancy between the **algorithm** and **control** cases illustrates the importance of eliminating outdated information, and the effectiveness of our algorithm in doing so. As a further comparison, we calculate the number of reward points that the robot is expected to have accrued on average during the course of a trial: 151.6 for **random**, 149.3 for **control**, and 153.8 for **algorithm**. These data are significantly different at $p\text{-value} < 0.02$ using a two-tail version of Student's t test. The superiority of the **algorithm** case illustrates the effectiveness of our moving average algorithm for reward maximization in a gradually shifting environment.

It should be noted that the **random** scenario used above lies along a continuum of possible experiments distinguished by the probability with which the robot collects (or discards) each puck it encounters. For comparison with the **control** and **algorithm** scenarios, we used the random case in which the robot collects 100% of the pucks it encounters. Intuitively, one would expect this to be the best performing of the random cases. When the probability of collecting a found puck is less than 1.0, the robot wastes more time searching for pucks but does not improve its reward since it discards both high-valued and low-valued pucks with equal probability. To verify our intuition, we conducted 80 trials for each of three lower probabilities of collection: 0.75, 0.50, 0.25. As expected, the data show that there is a continual significant decrease in accrued reward as the probability of collection decreases. The expected

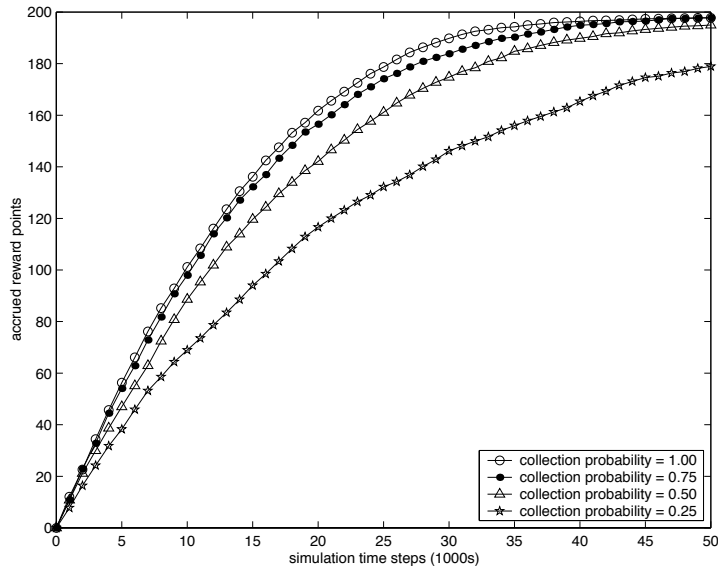


Figure 9. Average accrued reward for the four versions of the random scenario with different probabilities for collecting pucks (puck point values: black=1, clear=10).

reward points accrued on average during the course of a trial (in order of decreasing collection probability) are: 151.6, 148.5, 140.3, 118.3. Pairwise t -tests indicate that the four random scenarios are indeed different at a significance level of 0.0001, and Figure 9 visually demonstrates this difference. Thus, it might initially seem that the random cases with lower collection probabilities are more appropriate for comparison with the **algorithm** and **control** scenarios. It is, however, the collection probability of 1.0 that is the most difficult challenge for our algorithm, and consequently the one we used for the comparisons described here as well as those in Section 8.

We also tested our algorithm in a more challenging set of experiments where the point values of pucks were closer together (black=1, clear=4). Close point values make the difference in reward for collecting the “wrong” versus the “right” colored puck very small, and thus further sensitize the performance of the robot to the accuracy of the estimate of environmental state. In these experiments, we compared data from 140 trials of the **control** and **algorithm** scenarios, and 200 trials of the **random** scenario (all trials being run to 50,000 simulation steps). The expected reward accrued on average *during* these scenarios was, respectively: 67.99, 69.67, 69.02. Note that this is different from the average reward *at the end of* the scenarios, which is approximately 90.0. A t -test shows these results to be different at a significance level of 0.02. The superior performance of the **algorithm** scenario given the closeness of the data (Figure 10) helps illustrate the effectiveness of our AMM-based, moving-average state estimation algorithm using dynamic windowing.

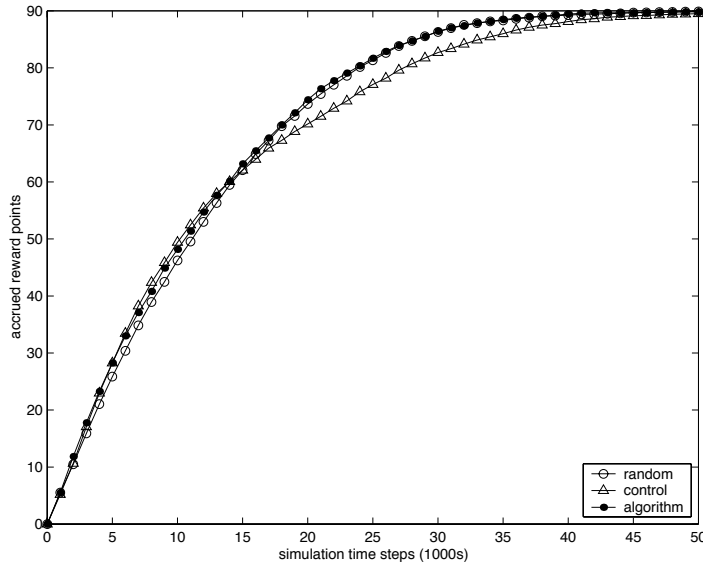


Figure 10. Average accrued reward for the three experimental scenarios with close point values for pucks (black=1, clear=4).

We have also performed tests of the algorithm in environments with other initial puck configurations, e.g., having the clear pucks close to the goal location and black pucks further away, and vice versa. Results from preliminary trials indicate that our algorithm continues to provide superior performance, but that the magnitude of the improvement diminishes. It is clear that in certain environments the **algorithm** scenario does not significantly improve performance over **random** and **control**, or that the improvement is so slight as to be inconsequential. The superiority of the **algorithm** scenario is most evident when the difference between reward values (for the puck types) is large (i.e., the importance of the decision to collect a puck increases) and when the environment shifts abruptly. Currently, we have only investigated systems where the non-stationarity is either gradual or abrupt. In future work, we intend to consider systems where gradual and abrupt changes both occur, e.g., there is a gradual change in the system that eventually triggers a catastrophic/abrupt event. An example might be gradual sensor degradation followed by catastrophic sensor failure.

11. Related Work

We confine our review to a selection of most relevant work in mobile robotics and statistical modeling. Various models have been employed on mobile robot platforms to date, a few examples of which we consider here. Cassandra et al. (1996) use a partially observable Markov decision process (POMDP) to model

uncertainty of location in a robot navigation task for an office environment. This work is similar to ours in that both explore model construction on, and use by, a mobile robot. The high computational complexity and large data requirements for POMDP generation, however, make it necessary to use a number of heuristics to reduce the problem size and facilitate learning in addition to often running numerous trials, neither of which is the case with our approach.

Koenig and Simmons (1996) use a POMDP to model sensor, actuator, and metric uncertainties in a similar office navigation task. For tractability, the learning system on the robot is initialized with a POMDP compiled off-line using sensor models and a topological map with metric uncertainties. A modified POMDP learning algorithm is used to passively fine-tune the probability distributions. Somewhat similarly, our work uses AMMs passively to capture statistics about the interaction dynamics between a robot and its environment, with the data used to influence the robot's behavior. Both POMDPs and AMMs represent uncertainties using probability distributions, but POMDPs, being decision processes, also enable learning of control policies. Such policies may require sensor and environment state information, making the search space large and requiring more (possibly heuristic) computation. The number of states in a POMDP is usually pre-specified, whereas in an AMM it is learned and depends on the number of input symbols and the order of the system.

Michaud and Mataric (1998) present a learning technique that uses a behavioral model of a robot's interaction with the environment. The model takes the form of a tree capturing the history of behavior use, i.e., specific sequences of behaviors, with nodes representing executed behaviors, and links representing transitions between them. Their approach and our AMMs are both constructive, being generated and modified as needed. Unlike AMMs, which can form graphs with alternative behavior choices represented by probability distributions over transitions, their approach stores potentially long linear sequences of behaviors in a tree with branches indicating alternative choices. The result is that the tree representation may require many trials to collect useful statistics, whereas AMMs can be generated during the course of one trial. This, however, is understandable given the goal of their work is for the robot to learn how to perform a task efficiently. By contrast, in our work the robot has a controller for the task, but must make intelligent decisions about how to proceed given its experience in the environment.

Košecá and Bajcsy (1993) present Discrete Event Systems (DESs) with emphasis on applications to mobile robotics. The structure of this DES approach is also related to AMMs, both being represented as directed graphs with behaviors as states. Unlike AMMs, DESs require *a priori* specification of all possible states, events, and the full transition function. This specification endows DES with control theoretic properties, though a practical specification of these parameters for mobile robots would seem to require heuristic engineering. Mahadevan and Theodorou (1998) have applied the notion of discrete (though time-extended) events to a Markov decision process for modeling

industrial manufacturing. The goal is to optimize production using reinforcement learning. Other work has also used such hybrid SMP/MDP models, or *semi-Markov decision processes* (Sutton et al., 1999; Wang and Mahadevan, 1999), as well as dynamical systems approaches (Beer, 1993; Smithers, 1995) to model the interaction between an agent (robot) and its environment.

The basic structure of augmented Markov models is very similar to that of hidden Markov models (HMMs) (Rabiner, 1989). The difference is that in an AMM, there is only one observation symbol per state, as opposed to a probability distribution over observation symbols in an HMM. In addition, an AMM assumes that the state of the system is known, thereby removing the HMM hidden state assumption. Our construction algorithm, however, is able to capture hidden state associated with higher-order transitions. Han and Veloso (2000) use HMMs to represent robot behaviors in a real-time behavior recognition system employed in a robot soccer domain. We chose AMMs for the work in this paper because of our need to construct higher-order Markov models incrementally and on-line, which is not as easily accomplished with HMMs. In addition, we did not require the hidden state capabilities of HMMs.

The notion of splitting AMM states based on local estimates of mean and variance is related to work by Hanson on the stochastic delta rule used in updating mean and variance estimates on the weights of feed-forward neural networks. Meiosis Networks use these mean and variance estimates to decide when to split states in the hidden layer (Hanson, 1990).

A complementary notion is that of state merging, explored by Stolcke and Omohundro (1993) for learning the number of states and topology of an HMM. The approach begins by constructing a specialized model exactly representing all of the data. Successive merging operations are used to generalize the model while making certain that the Bayesian posterior probability of the model given the data does not decrease. This approach is similar to our AMM construction algorithm in being data-driven, but differs in, among other ways, its use of state merging rather than state splitting, and its non-incremental processing style. Seymore et al. (1999) also use Bayesian model merging for learning HMM structure.

Other similar state splitting approaches have been explored. McCallum (1996) presents Utile Distinction Memory (UDM), an algorithm that splits the states of a POMDP using a method for storing statistics that is almost identical to the statistics in an AMM. The state splitting tests used in the two approaches are also very similar. UDM performs state splitting based on reward distinctions and not perceptual distinctions. This limits the growth of the POMDP to the complexity of the task and not the complexity of the perceptual space. (In contrast, Chrisman (1992) explored reinforcement learning with state splitting based on perceptual distinctions.) Our use of behaviors (instead of actions, precepts, etc.) serves a similar function — the AMM is only as complicated as the interaction between the robot, controller, and environment, for a particular task. UDM, however, is quite slow to learn and not appropriate to the type of incremental, on-line construction explored in this paper and provided by our AMM construction algorithm. U-Tree, the

culminating algorithm in McCallum (1996), allows for higher-order Markovian systems, as well as perceptual and reward distinctions. While quite useful in the reinforcement learning framework, it is not well suited to the problems explored in our work. We have focussed on modeling the execution of a controller (for the mine collection task) as a robot interacts with its environment, not on constructing the controller.

Much research has also been conducted on the performance and properties of robot collection tasks similar to the one we use (Arkin et al., 1993; Arkin and Ali, 1994; Parker, 1994; Fontán and Matarić, 1998; Balch, 2000). The reader is encouraged to see Cao et al. (1997) and Matarić (1995) for a more complete set of references from Artificial Intelligence, Robotics, Distributed AI, and Artificial Life.

While not the focus of this review, much work has been done in the statistics community on sequential changepoint detection. Lai (1995) provides a review of some of the literature. Our dynamic moving average approach is related to various changepoint detection schemes, particularly those that employ moving averages. Our approach, however, differs from much of the statics literature in one significant way. Rather than, for example, estimating the mean of a particular distribution (Chernoff and Zacks, 1964), it functions at more of a meta-level, selecting the model (AMM) that best represents the current state of the system.

12. Conclusions

We have presented an algorithm for moving average state estimation in a non-stationary environment. The algorithm learns multiple augmented Markov models (AMMs) and uses statistics from these to dynamically adjust the window size of the moving average. It requires no *a priori* knowledge of the environment, and is designed to compensate for both the variability and the manifested types of non-stationarity (from abrupt to gradual shifts) in a stochastic system. We have applied this algorithm to the problem of reward maximization in a situated mine collection task for robots, and shown that it improves performance in environments with abrupt and gradual non-stationarities.

Acknowledgments

The research reported here was conducted at the Interaction Lab, part of the Robotics Research Labs at the University of Southern California Computer Science Department. The work was supported in part by the Office of Naval Research Grant N00014-95-1-0759, by the National Science Foundation Infrastructure Grant CDA-9512448, by the Office of Naval Research Grant N00014-00-1-0140, and by the Sandia National Labs Grant 3065. Special thanks to Sridhar Mahadevan for help in relating SMPs and AMMs.

References

- Arkin, R. C.: 1998, *Behavior-Based Robotics*. The MIT Press: Cambridge, Massachusetts.
- Arkin, R. C. and K. S. Ali: 1994, 'Reactive and Telerobotic Control in Multi-Agent Systems'. In: *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. Brighton, England, pp. 473–478.
- Arkin, R. C., T. Balch, and E. Nitz: 1993, 'Communication of Behavioral State in Multi-Agent Retrieval Tasks'. In: *IEEE International Conference on Robotics and Automation*. Atlanta, pp. 588–594.
- Balch, T.: 2000, 'Hierarchical Social Entropy: An Information Theoretic Measure of Robot Group Diversity'. *Autonomous Robots* **8**, 209–237.
- Beer, R. D.: 1993, 'A Dynamical Systems Perspective on Agent-Environment Interaction'. *Artificial Intelligence* **72**, 173–215.
- Blyth, C. R.: 1986, 'Approximate Binomial Confidence Limits'. *Journal of the American Statistical Association* **81**(395), 843–855.
- Brooks, R. A.: 1990, 'The Behavior Language; User's Guide'. Technical Report AIM-1227, MIT AI Lab.
- Brooks, R. A.: 1991, 'Intelligence Without Reason'. In: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*. Sydney, Australia, pp. 569–590.
- Cao, Y. U., A. S. Fukunaga, and A. B. Kahng: 1997, 'Cooperative Mobile Robotics: Antecedents and Directions'. *Autonomous Robots* **4**, 1–23.
- Cassandra, A. R., L. P. Kaelbling, and J. A. Kurien: 1996, 'Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation'. In: *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2. pp. 963–972.
- Chernoff, H. and S. Zacks: 1964, 'Estimating the Current Mean of a Normal Distribution which is Subjected to Changes in Time'. *Annals of Mathematical Statistics* **35**(3), 999–1018.
- Chrisman, L.: 1992, 'Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach'. In: W. Swartout (ed.): *Proceedings of the 10th National Conference on Artificial Intelligence*. San Jose, CA, pp. 183–188.
- Cormen, T. H., C. E. Leiserson, and R. L. Rivest: 1990, *Introduction to Algorithms*. McGraw-Hill Book Company.
- Fontán, M. S. and M. J. Mataric: 1998, 'Territorial Multi-Robot Task Division'. *IEEE Transactions on Robotics and Automation* **14**(5), 815–822.
- Freund, J. E.: 1992, *Mathematical Statistics*. Prentice Hall, fifth edition.
- Gat, E.: 1998, 'On Three-Layer Architectures'. In: D. Kortenkamp, R. P. Bonasso, and R. Murphy (eds.): *Artificial Intelligence and Mobile Robotics: Case Studies of Successful Robot Systems*. AAAI Press, pp. 195–210.
- Goldberg, D.: 2001, 'Evaluating the Dynamics of Agent Environment Interaction'. Ph.D. thesis, University of Southern California.
- Goldberg, D. and M. J. Mataric: 1999, 'Coordinating Mobile Robot Group Behavior Using a Model of Interaction Dynamics'. In: *Proceedings, The Third International Conference on Autonomous Agents (Agents '99)*. Seattle, Washington, pp. 100–107.
- Goldberg, D. and M. J. Mataric: 2001, 'Detecting Regime Changes with a Mobile Robot using Multiple Models'. In: *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Maui, Hawaii, USA, pp. 619–624.
- Han, K. and M. Veloso: 2000, 'Automated Robot Behavior Recognition Applied to Robotic Soccer'. In: *Robotics Research: the Ninth International Symposium*. Snowbird, Utah, pp. 249–256.
- Hanson, S. J.: 1990, 'Meiosis Networks'. In: D. S. Touretzky (ed.): *Advances in Neural Information Processing Systems 2*. San Mateo, CA, pp. 533–541.
- Kaelbling, L. P., M. L. Littman, and A. R. Cassandra: 1998, 'Planning and Acting in Partially Observable Stochastic Domains'. *Artificial Intelligence* **101**(1–2), 99–134.
- Kemeny, J. G., J. L. Snell, and A. W. Knapp: 1966, *Denumerable Markov Chains*. D. Van Nostrand Company, Inc.

- Koenig, S. and R. G. Simmons: 1996, 'Unsupervised Learning of Probabilistic Models for Robot Navigation'. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3. pp. 2301–2308.
- Košecká, J. and R. Bajcsy: 1993, 'Discrete Event Systems for Autonomous Mobile Agents'. In: *Proceedings of the First Workshop on Intelligent Robotic Systems*. Zakopane, Poland, pp. 21–31.
- Lai, T. L.: 1995, 'Sequential Change-point Detection in Quality Control and Dynamical Systems'. *Journal of the Royal Statistical Society, Series B (Methodological)* **57**(4), 613–658.
- Ling, R. F.: 1978, 'A Study of the Accuracy of Some Approximations for t , χ^2 , and F Tail Probabilities'. *Journal of the American Statistical Association* **73**(362), 274–283.
- Mahadevan, S. and G. Theodorou: 1998, 'Optimizing Production Manufacturing using Reinforcement Learning'. In: *Proceedings of the Eleventh International FLAIRS Conference*. Sanibel Island, Florida, pp. 372–377.
- Matarić, M. J.: 1992, 'Behavior-Based Systems: Key Properties and Implications'. In: *IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*. Nice, France, pp. 46–54.
- Matarić, M. J.: 1995, 'Issues and Approaches in the Design of Collective Autonomous Agents'. *Robotics and Autonomous Systems* **16**(2–4), 321–331.
- Matarić, M. J.: 1997, 'Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior'. *Journal of Experimental and Theoretical Artificial Intelligence* **9**(2–3), 323–336. Special issue on Software Architectures for Physical Agents.
- McCallum, A. K.: 1996, 'Reinforcement Learning with Selective Perception and Hidden State'. Ph.D. thesis, University of Rochester, Department of Computer Science, Rochester, New York.
- Michaud, F. and M. J. Matarić: 1998, 'Learning from History for Behavior-Based Mobile Robots in Non-stationary Conditions'. *Autonomous Robots* **5**(3–4), 335–354.
- Mitchell, T. M.: 1997, *Machine Learning*. The McGraw-Hill Companies, Inc.
- Parker, L. E.: 1994, 'Heterogeneous Multi-Robot Cooperation'. Ph.D. thesis, MIT.
- Peizer, D. B. and J. W. Pratt: 1968, 'A Normal Approximation for Binomial, F , Beta, and Other Common, Related Tail Probabilities, I'. *Journal of the American Statistical Association* **63**(324), 1416–1456.
- Pirjanian, P.: 1998, 'Multiple Objective Action Selection & Behavior Fusion using Voting'. Ph.D. thesis, Institute of Electronic Systems, Alborg University, Denmark.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery: 1992, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.
- Rabiner, L. R.: 1989, 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition'. *Proceedings of the IEEE* **77**(2), 257–285.
- Roberts, F. S.: 1976, *Discrete Mathematical Models: With Applications to Social, Biological, and Environmental Problems*. Prentice-Hall, Inc.
- Ross, S. M.: 1992, *Applied Probability Models with Optimization Applications*. New York: Dover Publications, Inc.
- Seymore, K., A. McCallum, and R. Rosenfeld: 1999, 'Learning hidden Markov model structure for information extraction'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence: Workshop on Machine Learning for Information Extraction*. Orlando, FL, pp. 37–42.
- Smithers, T.: 1995, 'What the Dynamics of Adaptive Behavior and Cognition Might Look Like in Agent-Environment Interaction Systems'. In: *Practice and Future of Autonomous Agents*. Mt. Verita, Switzerland.
- Stolcke, A. and S. Omohundro: 1993, 'Hidden Markov Model Induction by Bayesian Model Merging'. In: S. J. Hanson, J. D. Cowan, and C. L. Giles (eds.): *Advances in Neural Information Processing Systems*, Vol. 5. pp. 11–18.
- Sutton, R. S., D. Precup, and S. Singh: 1999, 'Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning'. *Artificial Intelligence* **112**, 181–211.
- Wang, G. and S. Mahadevan: 1999, 'Hierarchical Optimization of Policy-Coupled Semi-Markov Decision Processes'. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. Bled, Slovenia, pp. 464–473.

Whitehead, S. D. and L.-J. Lin: 1995, 'Reinforcement Learning of Non-Markov Decision Processes'. *Artificial Intelligence* **73**(1-2), 271-306.