Journal of Machine Learning Research 7 (2006) 2329-2367 Submitted 6/05; Revised 5/06; Published 11/06

Point-Based Value Iteration for Continuous POMDPs

Josep M. Porta PORTA@IRI.UPC.EDU
Institut de Robòtica i Informàtica Industrial
UPC-CSIC
Llorens i Artigas 4-6, 08028, Barcelona, Spain

Nikos Vlassis VLASSIS@SCIENCE.UVA.NL
Matthijs T.J. Spaan MTJSPAAN@SCIENCE.UVA.NL
Informatics Institute
University of Amsterdam
Kruislaan 403, 1098SJ, Amsterdam, The Netherlands

Pascal Poupart PPOUPART@CS.UWATERLOO.CA
David R. Cheriton School of Computer Science
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1

Editors: Sven Koenig, Shie Mannor, and Georgios Theocharous

Abstract

We propose a novel approach to optimize Partially Observable Markov Decisions Processes (POMDPs) defined on continuous spaces. To date, most algorithms for model-based POMDPs are restricted to discrete states, actions, and observations, but many real-world problems such as, for instance, robot navigation, are naturally defined on continuous spaces. In this work, we demonstrate that the value function for continuous POMDPs is convex in the beliefs over continuous state spaces, and piecewise-linear convex for the particular case of discrete observations and actions but still continuous states. We also demonstrate that continuous Bellman backups are contracting and isotonic ensuring the monotonic convergence of value-iteration algorithms. Relying on those properties, we extend the PERSEUS algorithm, originally developed for discrete POMDPs, to work in continuous state spaces by representing the observation, transition, and reward models using Gaussian mixtures, and the beliefs using Gaussian mixtures or particle sets. With these representations, the integrals that appear in the Bellman backup can be computed in closed form and, therefore, the algorithm is computationally feasible. Finally, we further extend PERSEUS to deal with continuous action and observation sets by designing effective sampling approaches.

Keywords: planning under uncertainty, partially observable Markov decision processes, continuous state space, continuous action space, continuous observation space, point-based value iteration

1. Introduction

Automated systems can be viewed as taking inputs from the environment in the form of sensor measurements and producing outputs toward the realization of some goals. An important problem is the design of good control policies that produce suitable outputs (e.g., actions) based on the inputs received (e.g., observations). When the state of the environment is only partially observable through noisy measurements, and actions have stochastic effects, optimizing the course of action is a non-trivial task. Partially Observable Markov Decision Processes, POMDPs (Åström, 1965;

Dynkin, 1965) provide a principled framework to formalize and optimize control problems fraught with uncertainty. Such problems arise in a wide range of application domains including assistive technologies (Montemerlo et al., 2002; Boger et al., 2005), mobile robotics (Simmons and Koenig, 1995; Cassandra et al., 1996; Theocharous and Mahadevan, 2002; Pineau et al., 2003b), preference elicitation (Boutilier, 2002), spoken-dialog systems (Roy et al., 2000; Zhang et al., 2001; Williams et al., 2005), and gesture recognition (Darrell and Pentland, 1996).

Policy optimization (i.e., optimization of the course of action) can be done with or without a model of the environment dynamics. Model-free techniques such as neuro-dynamic programming (Bertsekas and Tsitsiklis, 1996), and stochastic gradient descent (Meuleau et al., 1999; Ng and Jordan, 2000; Baxter and Bartlett, 2001; Aberdeen and Baxter, 2002) directly optimize a policy by simulation. These approaches are quite versatile since there is no explicit modeling of the environment. On the other hand, the absence of explicit modeling information is compensated by simulation which may take an unbearable amount of time. In practice, the amount of simulation can be reduced by restricting the search for a good policy to a small class.

In contrast, model-based approaches assume knowledge about a transition model encoding the effects of actions on environment states, an observation model defining the correlations between environment states and sensor observations, and a reward model encoding the utility of environment states. Even when sufficient a priori knowledge is available to encode a complete model, policy optimization remains a hard task that depends heavily on the nature of the model. To date, most existing algorithms for model-based POMDPs assume discrete states, actions and observations. Even then, optimization is generally intractable (Papadimitriou and Tsitsiklis, 1987; Madani et al., 1999; Lusena et al., 2001) and one must resort to the exploitation of model-specific structural properties to obtain approximate scalable algorithms for POMDPs with large state spaces (Boutilier and Poole, 1996; Roy and Gordon, 2003; Poupart and Boutilier, 2003, 2005) and complex policy spaces (Pineau et al., 2003a; Spaan and Vlassis, 2005; Smith and Simmons, 2004; Poupart and Boutilier, 2004, 2005).

Many real-world POMDPs are naturally modeled by continuous states, actions and observations. For instance, in a robot navigation task, the state space may correspond to robot poses (x, y, θ), the observations may correspond to distances to obstacles measured by sonars or laser range finders, and actions may correspond to velocity and angular controls. Given the numerous optimization techniques for discrete models, a common approach for continuous models consists of discretizing or approximating the continuous components with a grid (Thrun, 2000; Roy et al., 2005). This usually leads to an important tradeoff between complexity and accuracy as we vary the coarseness of the discretization. More precisely, as we refine a discretization, computational complexity increases. Clearly, an important research direction is to consider POMDP solution techniques that operate directly in continuous domains, which would render the discretization of the continuous components superfluous.

Duff (2002) considered a special case of continuous POMDPs in the context of model-based Bayesian reinforcement learning, in which beliefs are maintained over the space of unknown parameters of the transition model of a discrete-state MDP. As those parameters are probabilities, the corresponding POMDP is defined over a continuous domain. Duff (2002) demonstrated that, for this special case, the optimal value function of the POMDP is parameterized by a set of functions, and for finite horizon it is piecewise-linear and convex (PWLC) over the belief space of multinomial distributions. Independently, Porta et al. (2005) considered the case of robot planning under uncertainty, modeled as a continuous POMDP over the pose (continuous coordinates) of the robot.

They also proved that the optimal value function is parameterized by an appropriate set of functions called α-functions (in analogy to the α-vectors in discrete POMDPs). Moreover, they demonstrated that the value function for finite horizon is PWLC over the robot belief space for any functional form of the beliefs. In addition, Porta et al. (2005) provided an analytical derivation of the α-functions as linear combinations of Gaussians when the transition, reward, and observation models of the POMDP are also Gaussian-based.

In this paper, we generalize the results of Duff (2002) and Porta et al. (2005), and describe a framework for optimizing model-based POMDPs in which the state space and/or action and observation spaces are continuous. We first concentrate on the theoretical basis on which to develop a sound value-iteration algorithm for POMDPs on continuous spaces. We demonstrate that the value function for arbitrary continuous POMDPs is in general convex, and it is PWLC in the particular case when the states are continuous but the actions and observations are discrete. We also demonstrate that Bellman backups for continuous POMDPs are contracting and isotonic, which guarantees the monotonic convergence of a value-iteration algorithm.

Functions defined over continuous spaces (e.g., beliefs, observation, action and reward models) can have arbitrary forms that may not be parameterizable. In order to design feasible algorithms for continuous POMDPs, it is crucial to work with classes of functions that have simple parameterizations and that yield to closed belief updates and Bellman backups. We investigate Gaussian mixtures and particle-based representations for the beliefs and linear combinations of Gaussians for the models. Using these representations, we extend the PERSEUS algorithm (Spaan and Vlassis, 2005) to solve POMDPs with continuous states but discrete actions and observations. We also show that POMDPs with continuous states, actions, and observations can be reduced to POMDPs with continuous states, discrete actions, and discrete observations using sampling strategies. As such, we extend PERSEUS to handle general continuous POMDPs.

The rest of the paper is structured as follows. Section 2 introduces POMDPs. Section 3 includes the proofs of some basic properties that are used to provide sound ground to the value-iteration algorithm for continuous POMDPs. Section 4 reviews the point-based POMDP solver PERSEUS. Section 5 investigates POMDPs with Gaussian-based models and particle-based representations for belief states, as well as their use in PERSEUS. Section 6 addresses the extension of PERSEUS to deal with continuous action and observation spaces. Section 7 presents some experiments showcasing the extended PERSEUS algorithm on a simulated robot navigation task. Section 8 gives an overview of related work on planning for continuous POMDPs. Finally, Section 9 concludes and highlights some possibilities for future work.

2. Preliminaries: MDPs and POMDPs

The Markov Decision Process (MDP) framework is a well-known planning paradigm that can be applied whenever we have an agent making decisions in a system described by

- a set of system states, S,
- a set of actions available to the agent, A,
- a transition model defined by $p(s'|s,a)$, the probability that the system changes from state s to s' when the agent executes action a, and

- a reward function defined as $r_a(s) \in \mathbb{R}$, the reward obtained by the agent if it executes action a when the system is in state s.

The dynamics of a discrete-time MDP is the following: at a given moment, the system is in a state s and the agent executes an action a. As a result, the agent receives a reward r and the system state changes to s'. The state contains enough information to allow to plan optimally, and thus a *policy* is a mapping from states to actions. To assess the quality of a given policy, π, the *value function* condenses the immediate and delayed reward that can be obtained from a given state s_0

$$V^{\pi}(s_0) = E\Big[\sum_{t=0}^{n} \gamma^t r_{\pi(s_t)}(s_t)\Big],$$

where the state evolves according to the transition model $p(s_{t+1}|s_t, \pi(s_t))$, n is the planning horizon (possibly infinite), and $\gamma \in [0,1)$ is a discount factor that trades off the importance of the immediate and the delayed reward.

The objective of MDP-based planning is to determine an *optimal policy*, π^*, that is, a policy that assigns to each state the action from which the most future reward can be expected. In the literature, there are several algorithms for computing an optimal policy for any MDP. When the transition and the reward model are known in advance, we can use planning algorithms mainly developed within the *operations research* field. Algorithms also exist for the case where the transition and reward models must be learned by the agent as it interacts with the environment. These learning algorithms are typically developed within the *reinforcement learning* community (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998).

Three popular planning algorithms are *value iteration*, *policy iteration*, and *linear programming*. We will focus on the first one, value iteration. This algorithm computes a sequence of value functions departing from an initial value function V_0 and using the following recursion

$$V_n(s) = \max_{a \in A} Q_n(s,a),$$

with

$$Q_n(s,a) = r_a(s) + \gamma \sum_{s' \in S} p(s'|s,a)\, V_{n-1}(s').$$

The above recursion is usually written in functional form

$$Q_n^a = H^a V_{n-1},$$

$$V_n = H V_{n-1}, \tag{1}$$

and it is known as the *Bellman recursion* (Bellman, 1957). This recursion converges to V^*, from which we can define an optimal policy π^* as

$$\pi^*(s) = \arg\max_a Q^*(s,a).$$

For each value function, V_n, we can readily derive an approximation to the optimal policy. Bounds on the quality of this approximation are given by Puterman (1994) in Theorem 6.3.1.

The MDP framework assumes the agent has direct knowledge of the system state. In many realistic situations, however, the agent can not directly access the state, but it receives an *observation*

that stochastically depends on it. In these cases, the system can be modeled as a Partially Observable Markov Decision Process (POMDP). This paradigm extends the MDP framework by incorporating a set of observations O, and an observation model defined by $p(o|s)$, the probability that the agent observes o when the system reaches state s. In a POMDP, the agent typically needs to infer the state of the system from the sequence of received observations and executed actions. A usual representation for the knowledge about the system state is a *belief*, that is, a probability distribution over the state space. The initial belief is assumed to be known and, if b is the belief of the agent about the state, the updated belief after executing action a and observing o is

$$b^{a,o}(s') = \frac{p(o|s')}{p(o|b,a)}\, p(s'|b,a), \tag{2}$$

with $p(s'|b,a)$ the propagation of the belief b through the transition model. For a continuous set of states S, this propagation is defined as

$$p(s'|b,a) = \int_{s\in S} p(s'|s,a)\, b(s)\, ds, \tag{3}$$

and, for a discrete set S, the integral is replaced by a sum. Under the Markov assumption, the belief carries enough information to plan optimally (see Bertsekas, 2001). Thus, a belief-based discrete-state POMDP can be seen as an MDP with a continuous state space that has one dimension per state. In the case of continuous-state POMDPs, the corresponding belief space is also continuous, but with an infinite number of dimensions since there are infinitely many physical states. This additional complexity is one of the reasons why most of the POMDP research focuses on the discrete-state case.

The belief-state MDP defined from a POMDP has a transition model

$$p(b'|b,a) = \begin{cases} p(o|b,a) & \text{if } b' = b^{a,o}, \\ 0 & \text{otherwise,} \end{cases}$$

and its policy and value function are defined on the space of beliefs. The Bellman recursion is defined as

$$V_n(b) = \sup_{a\in A} \Big\{ \langle r_a, b\rangle + \gamma \int_o p(o|b,a)\, V_{n-1}(b^{a,o})\, do \Big\}. \tag{4}$$

For discrete observation and action spaces, the integral over the observation space is replaced by a sum and the sup over the action space by a max operator. In Eq. 4, the $\langle f, b\rangle$ operation is used to express the expectation of the function f in the probability space defined by sample space S, the σ-algebra on S, and the probability distribution b. For continuous-state POMDPs, this operator is computed with an integral over S

$$\langle f, b\rangle = \int_{s\in S} f(s)\, b(s)\, ds,$$

and for discrete-state POMDPs, it corresponds to the inner product

$$\langle f, b\rangle = \sum_{s\in S} f(s)\, b(s).$$

Note that, in both cases and for a fixed f, the expectation operator is a linear function in the belief space since we have

$$\langle f, k\,b\rangle = k\,\langle f, b\rangle,$$
$$\langle f, b + b'\rangle = \langle f, b\rangle + \langle f, b'\rangle,$$

for any k independent of the integration/sum variable.

At first sight computing the POMDP value function seems intractable, but Sondik (1971) has shown that, for discrete POMDP, this function can be expressed in a simple form

$$V_n(b) = \arg\max_{\{\alpha_n^i\}_i} \langle \alpha_n^i, b\rangle,$$

with $\{\alpha_n^i\}_i$ a set of vectors. Each α-vector is generated for a particular action, and the optimal action with planning horizon n for a given belief is the action associated with the α-vector that defines V_n for that belief. Thus, the set of α_n^i vectors encodes not only the value, but also the optimal policy.

Since the $\langle \cdot, \cdot \rangle$ function is linear, the value function V_n computed as a maximum of a set of such expectations is piecewise-linear convex (PWLC) in the belief space. Using this formulation, value iteration algorithms for discrete state POMDPs typically focus on the computation of the α_n-vectors. Two basic strategies for POMDP value iteration are found in the literature. In the first one, the initial value function (i.e., at planning horizon 0) is a set of α-vectors directly defined from the reward function (Sondik, 1971; Monahan, 1982; Cheng, 1988; Kaelbling et al., 1998; Cassandra et al., 1997; Pineau et al., 2003a). In the second strategy, the initial value function is a single α-vector that lower bounds the value function for any possible planning horizon (Zhang and Zhang, 2001; Spaan and Vlassis, 2005). In both cases, exact value iteration converges to the same fixed point, but the second strategy may be more effective in approximate value iteration schemes.

3. Properties of Continuous POMDPs

In the previous section, we saw that algorithms for discrete POMDPs rely on a representation of the value function as a PWLC function based on a discrete set of supporting vectors. In this section, we show that this representation can be generalized to continuous-state POMDPs, while still assuming a discrete set of actions and observations. In Section 6, we discuss how to tackle POMDPs with continuous actions and observations via sampling.

First, we prove that the value function for a continuous POMDP is convex and, next, that it is PWLC for the case of continuous states, but discrete observations and actions. In this last case, the value function can be represented as a set of α-functions that play the same role as α-vectors in a discrete POMDP. We also prove that the continuous POMDP value-function recursion is an isotonic contraction. From these results, it follows that this recursion converges to a single fixed point corresponding to the optimal value function V^*. The theoretical results presented in this section establish that there is in principle no barrier in defining value iteration algorithms for continuous POMDPs.

3.1 The Optimal Value Function for Continuous POMDPs is Convex

To prove that the optimal value function for continuous POMDPs is convex, we first prove the following lemma.

Lemma 1 *The n-step optimal value function V_n in a continuous POMDP can be expressed as*

$$V_n(b) = \sup_{\{\alpha_n^i\}_i} \langle \alpha_n^i, b \rangle,$$

for appropriate continuous set of α*-functions* $\alpha_n^i : S \to \mathbb{R}$.

Proof The proof, as in the discrete case, is done via induction. In the following we assume that all operations (e.g., integrals) are well-defined in the corresponding spaces. For planning horizon 0, we only have to take into account the immediate reward and, thus, we have that

$$V_0(b) = \sup_{a \in A} \langle r_a, b \rangle,$$

and, therefore, if we define the continuous set

$$\{\alpha_0^i\}_i = \{r_a\}_{a \in A}, \tag{5}$$

we have that, as desired

$$V_0(b) = \sup_{\{\alpha_0^i\}_i} \langle \alpha_0^i, b \rangle.$$

For the general case, we have that, using Eq. 4

$$V_n(b) = \sup_{a \in A} \left\{ \langle r_a, b \rangle + \gamma \int_o p(o|b,a)\, V_{n-1}(b^{a,o})\, do \right\}$$

and, by the induction hypothesis,

$$V_{n-1}(b^{a,o}) = \sup_{\{\alpha_{n-1}^j\}_j} \langle \alpha_{n-1}^j, b^{a,o} \rangle.$$

From Eq. 2 and the definition of the $\langle \cdot, \cdot \rangle$ expectation operator,

$$V_{n-1}(b^{a,o}) = \frac{1}{p(o|b,a)} \sup_{\{\alpha_{n-1}^j\}_j} \int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|b,a)\, ds'.$$

With the above

$$\begin{aligned}
V_n(b) &= \sup_{a \in A} \left\{ \langle r_a, b \rangle + \gamma \int_o \sup_{\{\alpha_{n-1}^j\}_j} \int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|b,a)\, ds'\, do \right\} \\
&= \sup_{a \in A} \left\{ \langle r_a, b \rangle + \gamma \int_o \sup_{\{\alpha_{n-1}^j\}_j} \int_{s'} \alpha_{n-1}^j(s')\, p(o|s') \left[\int_s p(s'|s,a)\, b(s)\, ds \right] ds'\, do \right\} \\
&= \sup_{a \in A} \left\{ \langle r_a, b \rangle + \gamma \int_o \sup_{\{\alpha_{n-1}^j\}_j} \int_s \left[\int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|s,a)\, ds' \right] b(s)\, ds\, do \right\} \\
&= \sup_{a \in A} \left\{ \langle r_a, b \rangle + \gamma \int_o \sup_{\{\alpha_{n-1}^j\}_j} \left\langle \int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|s,a)\, ds', b \right\rangle do \right\}.
\end{aligned}$$

At this point, we can define

$$\alpha^j_{a,o}(s) = \int_{s'} \alpha^j_{n-1}(s')\, p(o|s')\, p(s'|s,a)\, ds'. \tag{6}$$

Note that these functions are independent of the belief point b for which we are computing V_n. With this, we have that

$$V_n(b) = \sup_{a\in A} \Big\{ \langle r_a, b\rangle + \gamma \int_o \sup_{\{\alpha^j_{a,o}\}_j} \langle \alpha^j_{a,o}, b\rangle \, do \Big\},$$

and we define

$$\alpha_{a,o,b} = \arg\sup_{\{\alpha^j_{a,o}\}_j} \langle \alpha^j_{a,o}, b\rangle. \tag{7}$$

The $\alpha_{a,o,b}$ set is just a subset of the $\alpha^j_{a,o}$ set defined above. Using this subset, we can write

$$\begin{aligned} V_n(b) &= \sup_{a\in A} \Big\{ \langle r_a, b\rangle + \gamma \int_o \langle \alpha_{a,o,b}, b\rangle \, do \Big\} \\ &= \sup_{a\in A} \Big\langle r_a + \gamma \int_o \alpha_{a,o,b}\, do, b \Big\rangle. \end{aligned}$$

Now

$$\{\alpha^i_n\}_i = \bigcup_{\forall b} \{r_a + \gamma \int_o \alpha_{a,o,b}\, do\}_{a\in A}, \tag{8}$$

is a continuous set of functions parameterized in the continuous action set. Intuitively, each α_n-function corresponds to a plan and, the action associated with a given α_n-function is the optimal action for planning horizon n for all beliefs that have such function as the maximizing one.

With the above definition, we have that V_n can be put in the desired form

$$V_n(b) = \sup_{\{\alpha^i_n\}_i} \langle \alpha^i_n, b\rangle, \tag{9}$$

and, thus, the lemma holds. ■

Using the above lemma we can directly prove the convex property for the value function on continuous POMDPs. Recall that, as mentioned in Section 2, for a fixed α^i_n-function the $\langle \alpha^i_n, b\rangle$ operator is *linear* in the belief space. Therefore, the convex property is given by the fact that V_n is defined as the supreme of a set of convex (linear) functions and, thus, we obtain a convex function as a result. The optimal value function, V^* is the limit for V_n as n goes to infinite and, since all V_n are convex functions so is V^*.

Lemma 2 *When the state space is continuous but the observation and action sets are discrete, the finite horizon value function is piecewise-linear convex (PWLC).*

Proof First, we have to prove that the $\{\alpha^i_n\}_i$ sets are discrete for all n. Again, we can proceed via induction. For discrete actions, $\{\alpha^i_0\}_i$ is discrete from its definition (see Eq. 5). For the general case, we have to observe that, for discrete actions and observations and assuming $M = |\{\alpha^j_{n-1}\}|$, the sets $\{\alpha^j_{a,o}\}$ are discrete: for a given action a and observation o we can generate at most M

$\alpha^j_{a,o}$-functions. Now, using a reasoning parallel to that of the enumeration phase of the Monahan's algorithm (Monahan, 1982), we have at most $|A|\,M^{|O|}$ different α^i_n-functions (fixing the action, we can select one of the M $\alpha^j_{a,o}$-functions for each one of the observations) and, thus, $\{\alpha^i_n\}_i$ is a discrete set.

From the previous lemma, we know the value function to be convex. The *piecewise-linear* part of the property is given by the fact that, as we have just seen, the $\{\alpha^i_n\}_i$ set is of finite cardinality and, therefore, V_n is defined as a finite set of linear functions. ■

When the state space is discrete, the α-functions become α-vectors and the above proof is equivalent to the classical PWLC demonstration first provided by Sondik (1971).

3.2 The Continuous POMDP Bellman Recursion is a Contraction

Lemma 3 *For the continuous POMDP value recursion H and two given value functions V and U, it holds that*

$$\|HV - HU\| \leq \beta \|V - U\|,$$

with $0 \leq \beta < 1$ and $\|\cdot\|$ the supreme norm. That is, the continuous POMDP value recursion H is a contractive mapping.

Proof The H mapping can be seen as

$$HV(b) = \max_a H^a V(b),$$

with

$$H^a V(b) = \langle r_a, b\rangle + \gamma \int_o p(o|b,a)\, V(b^{a,o})\, do.$$

Assume that $\|HV - HU\|$ is maximum at point b. Denote as a_1 the optimal action for HV at b and as a_2 the optimal one for HU

$$HV(b) = H^{a_1} V(b),$$
$$HU(b) = H^{a_2} U(b).$$

Then it holds

$$\|HV(b) - HU(b)\| = H^{a_1} V(b) - H^{a_2} U(b),$$

assuming, without loss of generality that $HV(b) \leq HU(b)$. Since a_1 is the action that maximizes HV at b we have that

$$H^{a_2} V(b) \leq H^{a_1} V(b).$$

Therefore, we have that

$$\begin{gathered}
\|HV - HU\| = \\
\|HV(b) - HU(b)\| = \\
H^{a_1} V(b) - H^{a_2} U(b) \leq \\
H^{a_2} V(b) - H^{a_2} U(b) = \\
\gamma \textstyle\int_o p(o|a_2,b)\, [V(b^{a_2,o}) - U(b^{a_2,o})]\, do \leq \\
\gamma \textstyle\int_o p(o|a_2,b)\, \|V - U\|\, do = \\
\gamma \|V - U\|.
\end{gathered}$$

Since γ is in $[0,1)$, the lemma holds. ■

The space of value functions define a *vector space* (i.e., a space closed under addition and scalar scaling) and the contraction property ensures this space to be *complete* (i.e., all Cauchy sequences have a limit in this space). Therefore, the space of value functions together with the supreme norm form a *Banach space* and the *Banach fixed-point theorem* ensures (a) the existence of a single fixed point, and (b) that the value recursion always converges to this fixed point (see Puterman, 1994, Theorem 6.2.3 for more details).

3.3 The Continuous POMDP Bellman Recursion is Isotonic

Lemma 4 *For any two value functions V and U, we have that*

$$V \leq U \Rightarrow HV \leq HU$$

that is, the continuous POMDP value recursion H is an isotonic mapping.

Proof Let us denote as a_1 the action that maximizes HV at point b and a_2 the action that does so for HU

$$HV(b) = H^{a_1}V(b),$$
$$HU(b) = H^{a_2}U(b).$$

By definition, the value for action a_1 for HU at b is lower (or equal) than that for a_2, that is

$$H^{a_1}U(b) \leq H^{a_2}U(b).$$

From a given b we can compute $b^{a_1,o}$, for an arbitrary o and, then, the following holds

$$\begin{aligned}
&V \leq U \Rightarrow \\
&\forall b,o,\ V(b^{a_1,o}) \leq U(b^{a_1,o}) \Rightarrow \\
&\int_o p(o|a_1,b)\,V(b^{a_1,o})\,do \leq \int_o p(o|a_1,b)\,U(b^{a_1,o})\,do \Rightarrow \\
&\langle r_{a_1},b\rangle + \gamma \int_o p(o|a_1,b)\,V(b^{a_1,o})\,do \leq \langle r_{a_1},b\rangle + \gamma \int_o p(o|a_1,b)\,U(b^{a_1,o})\,do \Rightarrow \\
&H^{a_1}V(b) \leq H^{a_1}U(b) \Rightarrow \\
&H^{a_1}V(b) \leq H^{a_2}U(b) \Rightarrow \\
&HV(b) \leq HU(b) \Rightarrow \\
&HV \leq HU.
\end{aligned}$$

Since b and, from it $b^{a_1,o}$, can be chosen arbitrarily, the value function is isotonic. ■

The isotonic property of the value recursion ensures that value iteration converges *monotonically*.

4. PERSEUS: A Point-Based POMDP Solver

Eqs. 6 to 8 constitute the value-iteration process for continuous POMDPs since they provide a constructive way to define the α-elements (α-functions for the continuous-state case and α-vectors for the discrete one) defining V_n from those defining V_{n-1}. The implementation of this value iteration, however, will be only computationally feasible if all the involved integrals can be either derived in closed form or approximated numerically. Moreover, the $\{\alpha_n^i\}_i$ are continuous sets and this makes the actual implementation of the described value-iteration process challenging. In this section, we concentrate on continuous-state POMDPs (POMDPs with continuous states, but discrete actions and observations). In this case, the $\{\alpha_n^i\}_i$ sets contain finitely many elements and the value function is PWLC. This allows us to adapt POMDP solving algorithms designed for the discrete case. In particular, we describe the point-based value-iteration algorithm PERSEUS (Spaan and Vlassis, 2005) which has been shown to be very efficient for discrete POMDPs. The description shown in Table 1 is generic so that it can be used for either discrete or continuous-state POMDPs. Extensions of PERSEUS to deal with continuous action and observation spaces are detailed in Section 6.

The computation of the mapping H (Eq. 1) for a given belief point b is called a *backup*. This mapping determines the α-element (α-function for continuous POMDPs and α-vector for discrete-state POMDPs) to be included in V_n for a belief point under consideration (see Eqs. 6 to 8). A full backup, that is, a backup for the whole belief space, involves the computation of all relevant α-elements for V_n. Full backups are computationally expensive (they involve an exponentially growing set of α-elements), but the backup for a single belief point is relatively cheap. This is exploited by recent point-based POMDP algorithms to efficiently approximate V_n on a fixed set of belief points (Pineau et al., 2003a; Spaan and Vlassis, 2005). The α-elements for this restricted set of belief points generalize over the whole belief space and, thus, they can be used to approximate the value function for any belief point.

The backup for a given belief point b is

$$\texttt{backup}(b) = \arg\max_{\{\alpha_n^i\}_i} \langle \alpha_n^i, b \rangle,$$

where $\alpha_n^i(s)$ is defined in Eqs. 7 and 8 from the $\alpha_{a,o}$-elements (Eq. 6). Using the backup operator, the value of V_n at b (Eq. 9) is simply

$$V_n(b) = \langle \texttt{backup}(b), b \rangle.$$

If this point-backup has to be computed for many belief points, the process can be speeded up by computing the set $\{\alpha_{a,o}^j\}_j$ for all actions, observations, and elements in V_{n-1} (see Eq. 6) since these α-elements are independent of the belief point and are the base components to define the $\alpha_{a,o,b}$ for any particular belief point, b.

Using this backup operator, PERSEUS is defined as follows. First (Table 1, line 2), we let the agent randomly explore the environment and collect a set B of reachable belief points. Next (Table 1, lines 3-5), we initialize the value function V_0 as a constant function over the state space. The value for V_0 is the minimum possible accumulated discounted reward, $\min\{R\}/(1-\gamma)$ with R the set of possible rewards. In line 3, u denotes a function on S so that

$$\langle u, b \rangle = 1,$$

for any possible belief, b and, in particular, for the beliefs in B. The exact form for u depends on the representation we use for the α-elements. For instance, for a discrete set of states, u is a constant

Perseus
Input: A POMDP.
Output: V_n, an approximation to the optimal value function V^*.

1: **Initialize**
2: $B \leftarrow$ A set of randomly sampled belief points.
3: $\alpha \leftarrow \frac{\min\{R\}}{1-\gamma}\, u$
4: $n \leftarrow 0$
5: $V_n \leftarrow \{\alpha\}$
6: **do**
7: $\forall b \in B$,
8: $\texttt{Element}_n(b) \leftarrow \arg\max_{\alpha \in V_n} \langle \alpha, b \rangle$
9: $\texttt{Value}_n(b) \leftarrow \langle \texttt{Element}_n(b), b \rangle$
10: $V_{n+1} \leftarrow \emptyset$
11: $\tilde{B} \leftarrow B$
12: **do**
13: $b \leftarrow$ Point sampled randomly from $\tilde{B}$.
14: $\alpha \leftarrow \texttt{backup}(b)$
15: **if** $\langle \alpha, b \rangle < \texttt{Value}_\texttt{n}(\texttt{b})$
16: $\alpha \leftarrow \texttt{Element}_n(b)$
17: **endif**
18: $\tilde{B} \leftarrow \tilde{B} \setminus \{b' \in \tilde{B} \,|\, \langle \alpha, b' \rangle \geq \texttt{Value}_n(b')\}$
19: $V_{n+1} \leftarrow V_{n+1} \cup \{\alpha\}$
20: **until** $\tilde{B} = \emptyset$
21: $n \leftarrow n+1$
22: **until** convergence

Table 1: The PERSEUS algorithm: a point-based value-iteration algorithm for planning in POMDPs.

vector of $|S|$ ones, and for a continuous state space, u can be approximated by a properly scaled Gaussian with a large covariance in all the dimensions of the state space.

Starting with V_0, PERSEUS performs a number of approximate value-function update stages. The definition of the value-update process can be seen on lines 10–20 in Table 1, where $\tilde{B}$ is a set of non-improved points: points for which $V_{n+1}(b)$ is still lower than $V_n(b)$. At the start of each update stage, V_{n+1} is set to $\emptyset$ and $\tilde{B}$ is initialized to B. As long as $\tilde{B}$ is not empty, we sample a point b from $\tilde{B}$ and compute the new α-elements associated with this point using the `backup` operator. If this α-element improves the value of b, that is, if $\langle \alpha, b \rangle \geq V_n(b)$, we add α to V_{n+1}. The hope is that α improves the value of many other points, and all these points are removed from $\tilde{B}$. Often, a small number of α-elements will be sufficient to improve $V_n(b)$ $\forall b \in B$, especially in the first steps of value iteration. As long as $\tilde{B}$ is not empty we continue sampling belief points from it and trying to add their α-elements to V_{n+1}.

If the α computed by the `backup` operator does not improve at least the value of b (i.e., $\langle \alpha, b \rangle < V_n(b)$, see lines 15–17 in Table 1), we ignore α and insert a copy of the maximizing element of b

from V_n in V_{n+1}. Point b is now considered improved and is removed from $\tilde{B}$, together with any other belief points that had the same function as maximizing one in V_n. This procedure ensures that $\tilde{B}$ shrinks at each iteration and that the value update stage terminates.

PERSEUS stops when a given convergence criterion holds. This criterion can be based on the stability of the value function, on the stability of the associated policy, or simply on a maximum number of iterations or maximum planning time.

5. Representations for Continuous-State POMDPs

PERSEUS can be used with different representations for the beliefs, the α-functions, and the transition, observation and reward models. The selected representations should fulfill three minimum requirements. First, the belief update has to be closed, that is, the representation used for the beliefs must be closed under the propagation through the transition model (Eq. 3) and the multiplication with the observation model (Eq. 2). The second requirement is that the representation for the α-functions must be closed under addition and scaling (to compute Eq. 8), and closed for the integration after the product with the observation and the action models (see Eq. 6). Finally, the third requirement is that the $\langle \alpha, b \rangle$ expectation operator must be computable.

For discrete-state POMDP, the belief and the α-functions are represented by vectors and the models by matrices. In this case, all operations are linear algebra that produce closed form results. Next, we describe two alternative representations for continuous-state POMDPs. The first one uses linear combinations of Gaussian distributions to represent α-functions and mixtures of Gaussian distributions to represent belief states. The second one also uses linear combinations of Gaussian distributions to represent α-functions, but uses sets of particles to represent beliefs.

5.1 Models for Continuous-State POMDPs

For POMDPs with continuous states and discrete observations, a natural observation model $p(o|s)$ would consist of a continuum of multinomial distributions over o (i.e., one multinomial for each s). Unfortunately, such an observation model will not keep α-functions in closed form when multiplied by the observation model in a Bellman backup. Instead, we consider observation models such that $p(o|s)$ is approximated by a mixture of Gaussians in s for a given observation o.

We define the observation model $p(o|s)$ indirectly by specifying $p(s|o)$. More specifically, for a fixed observation o, we assume that $p(s|o)$ is a mixture of Gaussians on the state space defined non-parametrically from a set of samples $T = \{(s_i, o_i) \,|\, i \in [1, N]\}$ with o_i an observation obtained at state s_i. The training set can be obtained in a supervised way (Vlassis et al., 2002) or by autonomous interaction with the environment (Porta and Kröse, 2004). The observation model is

$$p(o|s) = \frac{p(s|o)\, p(o)}{p(s)},$$

and, assuming a uniform $p(s)$ in the space covered by T, and approximating $p(o)$ from the samples in the training set we have

$$p(o|s) \propto \left[\frac{1}{N_o} \sum_{i=1}^{N_o} \lambda_i^o \, \phi(s|s_i^o, \Sigma_i^o)\right] \frac{N_o}{N} = \sum_{i=1}^{N_o} w_i^o \, \phi(s|s_i^o, \Sigma_i^o),$$

where s_i^o is one of the N_o points in T with o as an associated observation, ϕ is a Gaussian with mean s_i^o and covariance matrix Σ_i^o, and $w_i^o = \lambda_i^o / N$ is a weighting factor associated with that training point.

The sets $\{\lambda_i^o\}_i$ and $\{\Sigma_i^o\}_i$ should be defined so that

$$\sum_{i=1}^{N_o} \lambda_i^o = N_o,$$
$$\lambda_i^o \geq 0,$$

and so that

$$p(s) = \sum_o p(s|o)\, p(o) = \sum_o \sum_{i=1}^{N_o} w_i^o\, \phi(s|s_i^o, \Sigma_i^o),$$

is (approximately) uniform in the area covered by T.

As far as the transition model is concerned, we assume it is linear-Gaussian

$$p(s'|s,a) = \phi(s'|s+\Delta(a), \Sigma^a), \tag{10}$$

with ϕ a Gaussian centered at $s+\Delta(a)$ with covariance Σ^a. The function $\Delta(\cdot)$ is a mapping from the action space to the state space and encodes the changes in the state produced by each action. For discrete action sets, this function can be seen as a table with one entry per action.

Finally, the reward model $r_a(s)$ is defined by a linear combination of (a fixed number of) Gaussians

$$r_a(s) = \sum_i w_i\, \phi_i(s|\mu_i^a, \Sigma_i^a),$$

where μ_i^a and Σ_i^a are the mean and covariance of each Gaussian.

5.2 α-Functions Representation

As mentioned above, we require an α-function representation that allow us to get a closed expression for the $\alpha_{a,o}^j$ (Eq. 7). With the above models, the α-functions can be represented by a linear combination of Gaussians as stated in the following lemma.

Lemma 5 *The functions $\alpha_n^i(s)$ can be expressed as linear combinations of Gaussians, assuming the observation, transition and reward models are also linear combinations of Gaussians.*

Proof This lemma can be proved via induction. For $n=0$, $\alpha_0^i(s) = r_a(s)$ for a fixed a and thus it is indeed a linear combination of Gaussians. For $n>0$, we assume that

$$\alpha_{n-1}^j(s') = \sum_k w_k^j\, \phi(s'|s_k^j, \Sigma_k^j).$$

Then, with our particular models, $\alpha_{a,o}^j(s)$ in Eq. 6 is the integral of three linear combinations of Gaussians

$$\begin{aligned}
\alpha_{a,o}^j(s) &= \int_{s'} \Big[\sum_k w_k^j\, \phi(s'|s_k^j, \Sigma_k^j)\Big] \Big[\sum_l w_l^o\, \phi(s'|s_l^o, \Sigma_l^o)\Big]\, \phi(s'|s+\Delta(a), \Sigma^a)\, ds' \\
&= \int_{s'} \sum_{k,l} w_k^j\, w_l^o \phi(s'|s_k^j, \Sigma_k^j)\, \phi(s'|s_l^o, \Sigma_l^o)\, \phi(s'|s+\Delta(a), \Sigma^a)\, ds' \\
&= \sum_{k,l} w_k^j\, w_l^o \int_{s'} \phi(s'|s_k^j, \Sigma_k^j)\, \phi(s'|s_l^o, \Sigma_l^o)\, \phi(s'|s+\Delta(a), \Sigma^a)\, ds'.
\end{aligned}$$

To compute this equation, we have to perform the product of two Gaussians and a closed formula is available for this operation

$$\phi(x|a,A)\,\phi(x|b,B) = \delta\,\phi(x|c,C),$$

with

$$\begin{aligned}\delta &= \phi(a|b,A+B) = \phi(b|a,A+B),\\ C &= (A^{-1}+B^{-1})^{-1},\\ c &= C\,(A^{-1}\,a + B^{-1}\,b).\end{aligned}$$

In the above case, we have to apply this formula twice, once for $\phi(s'|s_k^j,\Sigma_k^j)$ and $\phi(s'|s_l^o,\Sigma_l^o)$ to get $(\delta_{k,l}^{j,o}\,\phi(s'|s_1,\Sigma_1))$ and once more for $(\delta_{k,l}^{j,o}\,\phi(s'|s_1,\Sigma_1))$ and $\phi(s'|s+\Delta(a),\Sigma^a)$ to get $(\delta_{k,l}^{j,o}\,\beta_{k,l}^{j,o,a}(s)\,\phi(s'|s,\Sigma))$. The scaling terms $\delta_{k,l}^{j,o}$ and $\beta_{k,l}^{j,o,a}(s)$ can be expressed as

$$\begin{aligned}\delta_{k,l}^{j,o} &= \phi(s_l^o|s_k^j,\Sigma_k^j+\Sigma_l^o),\\ \beta_{k,l}^{j,o,a}(s) &= \phi(s|s_{k,l}^{j,o}-\Delta(a),\Sigma_{k,l}^{j,o}+\Sigma^a),\end{aligned}$$

with

$$\begin{aligned}\Sigma_{k,l}^{j,o} &= [(\Sigma_k^j)^{-1}+(\Sigma_l^o)^{-1}]^{-1},\\ s_{k,l}^{j,o} &= \Sigma_{k,l}^{j,o}\,[(\Sigma_k^j)^{-1}\,s_k^j+(\Sigma_l^o)^{-1}\,s_l^o].\end{aligned}$$

With this, we have

$$\begin{aligned}\alpha_{a,o}^j(s) &= \sum_{k,l} w_k^j\,w_l^o \int_{s'} \delta_{k,l}^{j,o}\,\beta_{k,l}^{j,o,a}(s)\,\phi(s'|s,\Sigma)\,ds'\\ &= \sum_{k,l} w_k^j\,w_l^o\,\delta_{k,l}^{j,o}\,\beta_{k,l}^{j,o,a}(s)\int_{s'}\phi(s'|s,\Sigma)\,ds'\\ &= \sum_{k,l} w_k^j\,w_l^o\,\delta_{k,l}^{j,o}\,\beta_{k,l}^{j,o,a}(s).\end{aligned}$$

Using Eqs. 7 and 8, we define the elements in $\{\alpha_n^i\}$ as

$$\alpha_n^i = r_a + \gamma\sum_o \arg\max_{\{\alpha_{a,o}^j\}_j}\langle\alpha_{a,o}^j,b\rangle.$$

Since the result of the arg max is just one of the members of the set $\{\alpha_{a,o}^j\}_j$, all the elements involved in the definition of α_n^i are linear combinations of Gaussians and so is the final result. ■

One point that deserves special consideration is the explosion of the number of components in the linear combinations of Gaussians defining the α-functions. If N_o is the number of components in the observation model and C_r is the average number of components in the reward model, the number of components in the α_n-functions scales with $O((N_o)^n\,C_r)$. Appendix A details an algorithm to bound the number of components of a mixture while losing as little information as possible.

5.3 Belief Representation

To get a belief update and an expectation operator that are computable, we consider two possible representations for the beliefs. The first one is Gaussian-based and the second one is particle-based.

5.3.1 GAUSSIAN-BASED REPRESENTATION

In this first case, we will assume that belief points are represented as Gaussian mixtures

$$b(s) = \sum_j w_j \, \phi(s|s_j, \Sigma_j), \tag{11}$$

with ϕ a Gaussian with mean s_j and covariance matrix Σ_j and where the mixing weights satisfy $w_j > 0$, $\sum_j w_j = 1$. In the extreme case, Gaussian mixtures with an infinite number of components would be necessary to represent a given point in the infinite-dimensional belief space of a continuous-state POMDP. However, only Gaussian mixtures with few components are needed in practical situations.

The belief update on Eq. 2 can be implemented in our model taking into account that it consists of two steps. The first one is the application of the action model on the current belief state. This can be computed as the propagation of the Gaussians representing $b(s)$ (Eq. 11) through the transition model (Eq. 10)

$$p(s'|b,a) = \int_s p(s'|s,a)\, b(s)\, ds = \sum_j w_j \, \phi(s|s_j + \Delta(a), \Sigma_j + \Sigma^a).$$

In the second step of the belief update, the prediction obtained with the action model is corrected using the information provided by the observation model

$$\begin{aligned} b^{a,o}(s') &\propto \Big[\sum_i w_i^o \, \phi(s'|s_i^o, \Sigma_i^o)\Big] \Big[\sum_j w_j \, \phi(s|s_j + \Delta(a), \Sigma_j + \Sigma^a)\Big] \\ &= \sum_{i,j} w_i^o \, w_j \, \phi(s'|s_i^o, \Sigma_i^o) \, \phi(s|s_j + \Delta(a), \Sigma_j + \Sigma^a). \end{aligned}$$

As mentioned, the product of two Gaussian functions is a scaled Gaussian. Therefore, we have that

$$b^{a,o}(s') \propto \sum_{i,j} w_i^o \, w_j \, \delta_{i,j}^{a,o} \, \phi(s'|s_{i,j}^{a,o}, \Sigma_{i,j}^{a,o}),$$

with

$$\begin{aligned} \delta_{i,j}^{a,o} &= \phi(s_j + \Delta(a) \,|\, s_i^o, \Sigma_i^o + \Sigma_j + \Sigma^a), \\ \Sigma_{i,j}^{a,o} &= ((\Sigma_i^o)^{-1} + (\Sigma_j + \Sigma^a)^{-1})^{-1}, \\ s_{i,j}^{a,o} &= \Sigma_{i,j}^{a,o}((\Sigma_i^o)^{-1} \, s_i^o + (\Sigma_j + \Sigma^a)^{-1} \, (s_j + \Delta(a))). \end{aligned}$$

Finally, we can rearrange the terms to get

$$b^{a,o}(s') \propto \sum_k w_k \, \phi(s'|s_k, \Sigma_k),$$

with $w_k = w_i^o \, w_j \, \delta_{i,j}^{a,o}$, $s_k = s_{i,j}^{a,o}$, and $\Sigma_k = \Sigma_{i,j}^{a,o}$ for all possible i, j. The proportionality in the definition of $b^{a,o}(s')$ implies that the weights (w_k, $\forall k$) should be scaled to sum to one

$$b^{a,o}(s') = \frac{1}{\sum_k w_k} \sum_k w_k \, \phi(s'|s_k, \Sigma_k).$$

An increase in the number of components representing a belief occurs when computing the belief update just detailed. If b_0 has C_b components and $p(o|s)$ is represented with an average of C_o components, the number of components in the belief b_t scales with $O(C_b(C_o)^t)$. As in the case of the α-functions, the procedure detailed in Appendix A could be used to bound the number of components in the beliefs.

Taking into account that the α-functions are also Gaussian-based, the expectation operator $\langle \cdot, \cdot \rangle$ can be computed in closed form as

$$\begin{aligned}
\langle \alpha, b \rangle &= \int_s \Big[\sum_k w_k \, \phi(s|s_k, \Sigma_k) \Big] \Big[\sum_l w_l \, \phi(s|s_l, \Sigma_l) \Big] \, ds \\
&= \sum_{k,l} w_k \, w_l \int_s \phi(s|s_k, \Sigma_k) \, \phi(s|s_l, \Sigma_l) \, ds \\
&= \sum_{k,l} w_k \, w_l \, \phi(s_l|s_k, \Sigma_k + \Sigma_l) \int_s \phi(s|s_{k,l}, \Sigma_{k,l}) \, ds \\
&= \sum_{k,l} w_k \, w_l \, \phi(s_l|s_k, \Sigma_k + \Sigma_l).
\end{aligned}$$

5.3.2 PARTICLE-BASED REPRESENTATION

An alternative to parameterize the belief densities using Gaussian mixtures is to represent the belief using N random samples, or particles, positioned at points s_i and with weights w_i. Thus, the belief is

$$b_t(s) = \sum_{i=1}^{N} w_i \, d(s - s_i),$$

where $d(s - s_i)$ is a Dirac's delta function centered at 0. Particle-based representations have been very popular in recent years, and they have been used in many applications from tracking to Simultaneous Localization and Mapping, SLAM, (see Doucet et al., 2001, for a review).

A particle-based representation has many advantages: it can approximate arbitrary probability distributions (with an infinite number of particles in the extreme case), it can accommodate non-linear transition models without the need of linearizing the model, and it allows several quantities of interest to be computed more efficiently than with the Gaussian-based belief representation. In particular, the integral in the belief update equation becomes a simple sum

$$b^{a,o}(s') \propto p(o|s') \sum_{i=1}^{N} w_i \, p(s'|s_i, a).$$

The central issue in the particle filter approach is how to obtain a set of particles to approximate $b^{a,o}(s')$ from the set of particles approximating $b(s)$. The usual Sampling Importance Re-sampling (SIR) approach (Dellaert et al., 1999; Isard and Blake, 1998) samples particles s'_i using the motion model $p(s'|s_i, a)$, then it assigns a new weight to each one of these particles proportional to the likelihood $p(o|s'_i)$, and finally it re-samples particles using these new weights in order to make all particles weights equal. The main problem of the SIR approach is that it requires many particles to converge when the likelihood $p(o|s')$ is too peaked or when there is only a small overlap between the prior and the posterior likelihood.

In the auxiliary particle filter (Pitt and Shephard, 1999) the sampling problem is addressed by inserting the likelihood inside the mixture

$$b^{a,o}(s') \propto \sum_{i=1}^{N} w_i \, p(o|s') \, p(s'|s_i, a).$$

The state s' used to define the likelihood $p(o|s')$ is not observed when the particles are resampled and we have to resort to approximations

$$b^{a,o}(s') \propto \sum_{i=1}^{N} w_i \, p(o|\mu_i) \, p(s'|s_i, a).$$

with μ_i any likely value associated with the *i-th* component of the transition density $p(s'|s_i, a)$, for example its mean. In this case, we have that $\mu_i = s_i + \Delta(a)$. Then $b^{a,o}(s')$ can be regarded as a mixture of the N transition components $p(s'|s_i, a)$ with weights $w_i \, p(o|\mu_i)$. Therefore, sampling a new particle s'_j to approximate $b^{a,o}(s')$ can be carried out by selecting one of the N components, say i_j, with probability $w_i \, p(o|\mu_i)$ and then sampling s'_j from the corresponding component $p(s'|s_{i_j}, a)$. Sampling is performed in the intersection of the prior and the likelihood and, consequently, particles with larger prior and larger likelihood (even if this likelihood is small in absolute value) are more likely to be used.

After the set of states for the new particles is obtained using the above procedure, we have to define their weights. This is done using

$$w'_j \propto \frac{p(o|s'_j)}{p(o|\mu_{i_j})}.$$

Using the sample-based belief representation the averaging operator $\langle \cdot, \cdot \rangle$ becomes

$$\begin{aligned}
\langle \alpha, b \rangle &= \int_s \Big[\sum_k w_k \, \phi(s|s_k, \Sigma_k) \Big] \Big[\sum_l w_l \, d(s - s_l) \Big] \, ds \\
&= \sum_k w_k \int_s \phi(s|s_k, \Sigma_k) \sum_l w_l \, d(s - s_l) \, ds \\
&= \sum_k w_k \sum_l w_l \, \phi(s_l|s_k, \Sigma_k) \\
&= \sum_{k,l} w_k \, w_l \, \phi(s_l|s_k, \Sigma_k).
\end{aligned}$$

Other re-sampling strategies such as those proposed by Fox (2003) that on-line adapt the number of sampled particles can also be applied here.

Given the common features between beliefs and α-functions in value iteration (i.e., beliefs and α-functions are both continuous functions of the state space), the α-functions also admit a particle representation. Note however that we cannot have both beliefs and α-functions represented by particles since the computation of $\langle \alpha, b \rangle$ requires that either b or α be in functional form to generalize over the entire state space.

Perseus
Input: A POMDP.
Output: V_n, an approximation to the optimal value function, V^*.

Initialize
  $B \leftarrow$ A set of randomly sampled belief points.
  $\alpha \leftarrow \frac{\min\{R\}}{1-\gamma} U$
  $n \leftarrow 0$
  $V_n \leftarrow \{\alpha\}$
do
  $\forall b \in B$,
    $\texttt{Element}_n(b) \leftarrow \arg\max_{\alpha \in V_n} \langle \alpha, b \rangle$
    $\texttt{Value}_n(b) \leftarrow \langle \texttt{Element}_n(b), b \rangle$
  $V_{n+1} \leftarrow \emptyset$
  $\tilde{B} \leftarrow B$
  do
    $b \leftarrow$ Point sampled randomly from $\tilde{B}$.
    $A \leftarrow \texttt{SampleActions}(b)$
    $\alpha \leftarrow \texttt{backup}(b)$
    if $\langle \alpha, b \rangle < \texttt{Value}_n(b)$
      $\alpha \leftarrow \texttt{Element}_n(b)$
    endif
    $\tilde{B} \leftarrow \tilde{B} \setminus \{b' \in \tilde{B} \mid \langle \alpha, b' \rangle \geq \texttt{Value}_n(b')\}$
    $V_{n+1} \leftarrow V_{n+1} \cup \{\alpha\}$
  until $\tilde{B} = \emptyset$
  $n \leftarrow n+1$
until convergence

Table 2: Modification of the PERSEUS algorithm in Table 1 to deal with large or continuous action spaces.

6. Extensions to Continuous Action and Observation Spaces

In Section 4, we presented a point-based value iteration algorithm to deal with continuous-state POMDPs. Now, we describe how to extend the presented framework to deal with continuous sets of actions and observations so that fully continuous POMDPs can also be addressed. The basic idea is that general continuous POMDPs can be cast in the continuous-state POMDP paradigm via sampling strategies.

6.1 Dealing with Continuous Actions

The backup operator in continuous-state value iteration requires computing a set of α-functions in Eq. 8, one function for each action $a \in A$, and then choosing the best function to back up. When the action space A is finite and small, the above optimization can be carried out efficiently by enumerating all possible actions and choosing the best, but in very large discrete action spaces this is

computationally inefficient. In this case, or when actions are continuous, one can resort to sampling-based techniques. As proposed by Spaan and Vlassis (2005), we can replace the full maximization over actions with a *sampled max* operator that performs the maximization over a subset of actions randomly sampled from A. One may devise several sampling schemes for choosing actions from A, for example, uniform over A or using the best action up to a given moment. Actions sampled uniformly at random can be viewed as exploring actions, while the latter can be viewed as exploiting current knowledge. Spaan and Vlassis (2005) provide more details on this point.

The use of a sampled max operator is very well suited for the point-based backups of PERSEUS, in which we only require that the values of belief points do not decrease over two consecutive backup stages. However, some modifications need to be introduced in the action and reward models described in Section 5.1. The action model described can be easily extended to continuous actions defining a continuous instead of a discrete function $\Delta : A \rightarrow S$ and evaluating it for the actions in the newly sampled A. As far as the reward model is concerned, we simply need to evaluate it for the sampled actions. Table 2 describes a modification of the basic PERSEUS algorithm to deal with large or continuous action spaces. Observe that, before computing the `backup` for the randomly selected belief point b (line 14b), we have to sample a new set of actions, A (line 14a) and the transition and reward models have to be modified accordingly, since they depend on the action set. Beside the action sampling and the on-line models computation, the rest of the algorithm proceeds the same as the one in Table 1. Note however, that the actions are sampled specifically for each belief b and, therefore we can not compute something similar to the $\alpha^j_{a,o}$-elements (see Eq. 6) that are common for all beliefs.

6.2 Dealing with Continuous Observations

In value iteration, the backup of a belief point b involves computing the expectation over observations

$$V_n(b) = \arg\max_a \Big\{ \langle r_a, b\rangle + \gamma V_{n-1}(b^a) \Big\},$$

with

$$V_{n-1}(b^a) = \int_o p(o|b,a)\, V_{n-1}(b^{a,o})\, do.$$

Using the definition of value function, the above reads

$$V_{n-1}(b^a) = \int_o p(o|b,a) \max_{\{\alpha^j_{n-1}\}_j} \langle \alpha^j_{n-1}, b^{a,o}\rangle \, do. \tag{12}$$

Building on an idea proposed by Hoey and Poupart (2005), assuming a finite number of α-elements α^j_{n-1}, observation spaces can always be discretized without any loss of information into regions corresponding to each α-element. In Eq. 12, all observations that lead to belief states $b^{a,o}$ with the same maximizing α-element can be aggregated together into one meta observation $O^j_{a,b}$ defined as follows

$$O^j_{a,b} = \{o \,|\, \alpha^j_{n-1} = \arg\max_{\{\alpha^j_{n-1}\}_j} \langle \alpha, b^{a,o}\rangle\}.$$

Using $O^j_{b,a}$, we can rewrite Eq. 12 as a sum over α-elements

$$V_{n-1}(b^a) = \sum_j \int_{o \in O^j_{a,b}} p(o|b,a)\, \langle \alpha^j_{n-1}, b^{a,o}\rangle \, do.$$

Rewriting the observation probabilities $p(o|b,a)$ in terms of s', we obtain

$$V_{n-1}(b^a) = \Big\langle \sum_j \int_{s'} \alpha^j_{n-1}(s') \Big[\int_{o \in O^j_{a,b}} p(o|s')\, do \Big]\, p(s'|s,a)\, ds', b \Big\rangle.$$

Hoey and Poupart (2005) assume a discrete state space, in which case the above quantity can be simplified by accumulating probability masses over observations in $O^j_{a,b}$, that is, defining $p(O^j_{a,b}|s') = \int_{o \in O^j_{a,b}} p(o|s')\, do$ for each state s', and then approximating $p(O^j_{a,b}|s')$ by sampling observations from $p(o|s')$. When the variable s' is continuous we can sample observations by *importance sampling* from some proposal distribution $q(o)$. With this we have

$$p(O^j_{a,b}|s') \simeq \frac{1}{N} \sum_{i=1}^{N_j} \frac{p(o^j_i|s')}{q(o^j_i)},$$

with $O^j_{a,b} = \{o^j_1, \ldots, o^j_{N_j}\}$ the set of observations for which α^j_{n-1} is maximal. The proposal distribution $q(o)$ can be, for instance, $p(o|b')$ with b' uniform in S or $p(o|b')$ with b' the current belief point. In our experiments we simply used a uniform distribution in O.

When working with continuous observations, the model given in Section 5.1 is no longer valid. However, we can assume the observation model to be defined using kernel smoothing from a training set including state-observation tuples. From those samples we can define

$$p(o,s) = \sum_{i=1}^{N} \lambda_i\, \phi(o|o_i, \Sigma^o_i)\, \phi(s|s_i, \Sigma^s_i),$$

and, using that,

$$p(o|s) = \frac{p(o,s)}{p(s)}.$$

Assuming a uniform $p(s)$, we have that $p(o|s)$ for a fixed observation is a Gaussian in s', which guarantees that α-functions remain closed under Bellman backups (see Section 5.1). With the observation model in the above form, we can further simplify $p(O^j_{a,b}|s')$ since we have that

$$\begin{aligned}
p(O^j_{a,b}|s') &\simeq \frac{1}{N} \sum_{i=1}^{N_j} \frac{p(o^j_i|s')}{q(o^j_i)} \\
&= \frac{1}{N} \sum_{i=1}^{N_j} \frac{1}{q(o^j_i)} \Big[\sum_{k=1}^{N} \lambda_k\, \phi(o^j_i|o_k, \Sigma^o_k)\, \phi(s|s_k, \Sigma^s_k) \Big] \\
&= \sum_{k=1}^{N} \Big[\frac{1}{N} \sum_{i=1}^{N_j} \frac{1}{q(o^j_i)} \lambda_k\, \phi(o^j_i|o_k, \Sigma^o_k) \Big]\, \phi(s|s_k, \Sigma^s_k) \\
&= \sum_{k=1}^{N} \rho^j_k\, \phi(s|s_k, \Sigma^s_k)
\end{aligned}$$

with

$$\rho^j_k = \frac{\lambda_k}{N} \sum_{i=1}^{N_j} \frac{\phi(o^j_i|o_k, \Sigma^o_k)}{q(o^j_i)}.$$

With the discretized observation model we can define

$$\alpha^j_{a,b} = \sum_j \int_{s'} \alpha^j_{n-1}(s')\, p(O^j_{a,b}|s')\, p(s'|s,a)\, ds',$$

that plays the same role in the Bellman backup as the $\alpha^j_{a,o,b}$-functions introduced in Eq. 7. With the above we have

$$V_{n-1}(b^a) = \langle \alpha^j_{a,b}, b \rangle,$$

and the α-elements for V_n at belief b are defined as

$$\{\alpha^i_n\}_i = \{r_a + \gamma \alpha^j_{a,b}\}_{a \in A}.$$

Thus, as far as implementation is concerned, continuous observation spaces introduce a modification in the `backup`, but this modification is independent of the rest of the algorithm. Therefore this new operator can be used both in PERSEUS with either discrete or sampled continuous actions (see Table 1 and Table 2, respectively).

Note that if we work with continuous observation spaces, the $\alpha^j_{a,b}$-functions are computed specifically for each belief and, therefore no precomputation similar to those of the $\alpha^j_{a,o}$-elements is possible.

7. Experiments and Results

To demonstrate the viability of our method we carried out some experiments in a simulated robotic domain. The simulation was programmed in Matlab 7.1 using a Pentium Xeon at 3 GHz running under Linux. In the simulated problem (see Fig. 1-a), a robot is moving along a corridor with four doors, where the state space is the continuous interval $[-21, 21]$. The target for the robot is to locate the second door from the right and enter it. The robot only receives positive reward when it enters the target door (see Fig. 1-c). When the robot tries to move beyond the end of the corridor (either right or left), or when it tries to enter a door at a wrong position, it receives negative reward. The reward function is represented using a linear combination of nine Gaussian functions. Three Gaussians are placed at each extreme of the corridor to represent the negative reward for trying to move beyond the end of the corridor (with means $\pm 21, \pm 19, \pm 17$, covariance 0.05, and weight -2). Two Gaussians represent the negative reward for trying to enter a door at the wrong position (with means ± 25, covariance 12.5, and weight -10). Finally, one Gaussian is used for the positive reward associated with entering the correct door (with mean 3, covariance 0.15, and weight 2).

In all reported experiments, the set of beliefs B used in the PERSEUS algorithm contains 500 unique belief points collected using random walks departing from a uniform belief, the latter being approximated with a Gaussian mixture with four components. The walks of the robot along the corridor are organized in episodes of 30 actions (thus, for instance, the robot can repetitively try to enter the correct door accumulating positive reward). In all experiments we set $\gamma = 0.95$.

In the first experiment we assume discrete observations and actions. There are four distinct observations, *left-end*, *right-end*, *door*, and *corridor*. The observation model, shown in Fig. 1-b, is approximated using a training set of 22 samples evenly placed every two space units from -21 to 21 (with $\Sigma^o = 4$). The five right/left-most samples correspond to observations *right-end* and *left-end*, respectively, each sample taken in front of a door corresponds to observation *door*, and the rest of the samples correspond to observation *corridor*. There are three distinct actions: the robot can

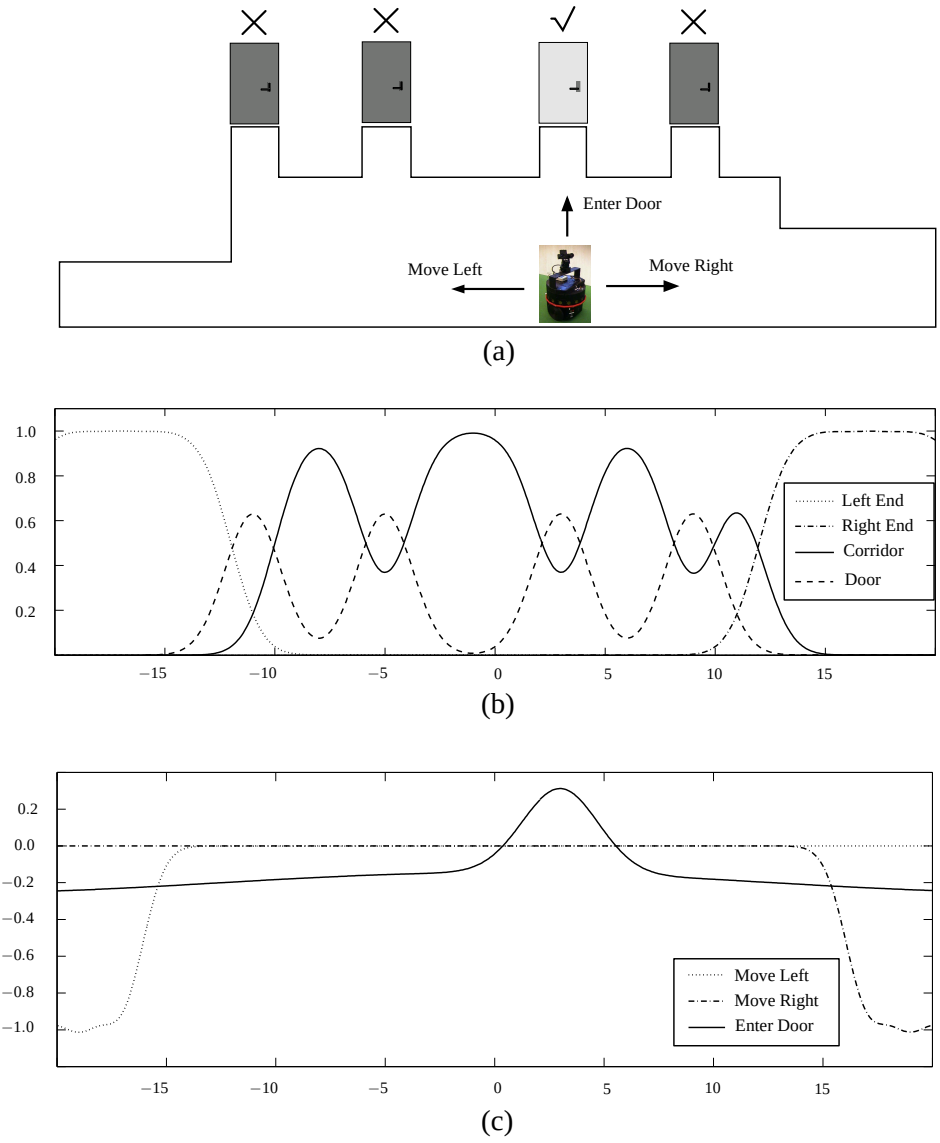

Figure 1: A pictorial representation of the test problem (a), the corresponding observation model (b), and the reward model (c).

move two units either to the left or to the right (with $\Sigma^a = 0.05$), or it can try to enter a door at any point. In this experiment we used Gaussian mixtures to represent the beliefs, compressing them, if necessary, to a maximum of four components, and similarly we used α-functions with a maximum of nine Gaussian components.

Fig. 2 shows the average results obtained after 10 runs of the version of PERSEUS described in Section 4. The first plot (top-left) shows the convergence of the value computed as $\sum_b V(b)$. The second plot (top-right) shows the expected discounted reward computed by running for 50 episodes the policy available at the corresponding time slice. The fact that this plot converges to a positive

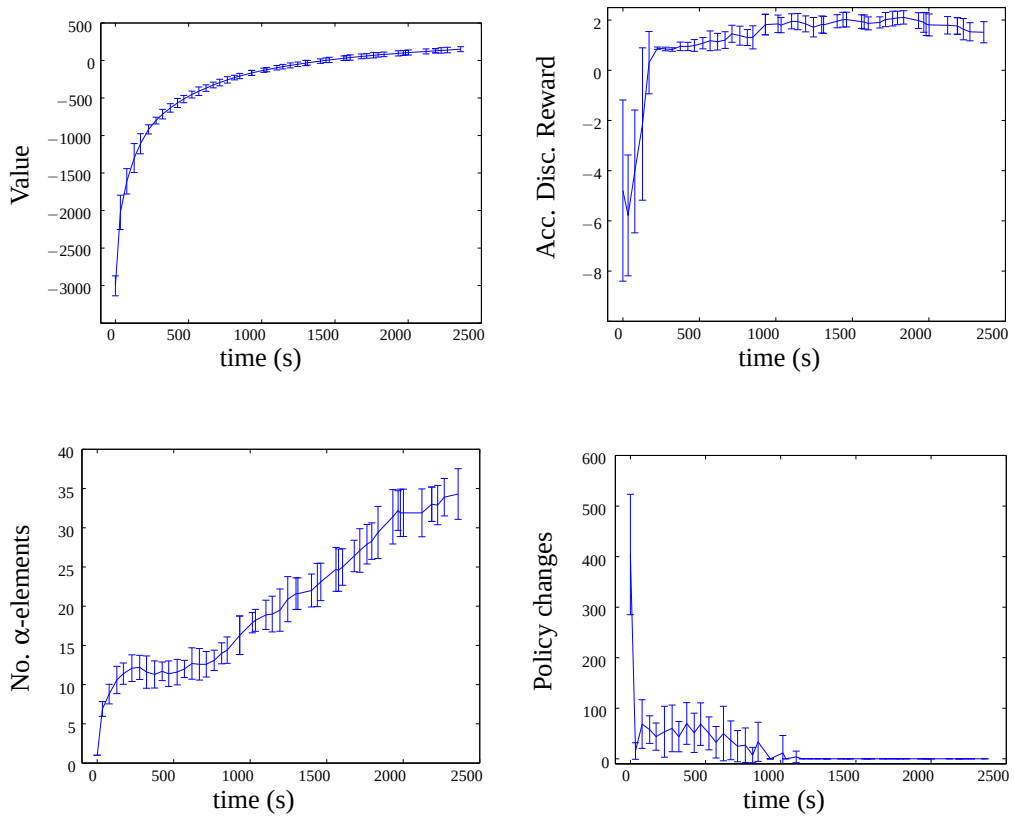

Figure 2: Results for the simulated robotic problem using continuous states but discrete actions and observations. Top: Evolution of the value for all the beliefs in B and the average accumulated discounted reward for 10 episodes. Bottom: Number of elements in V_n and the number of policy changes. Results are averaged for 10 repetitions and the bars represent the standard deviation.

value indicates that the robot successfully learns to avoid collisions, to find out its position, and to identify the target door. Next plot (bottom-left) shows the number of α-functions used to represent the value function. We can see that the number of α-functions increases, but it remains far below 500, the maximum possible number of α-functions (in the extreme case we would use a different α-function for each point in B). Finally, the bottom-right plot shows the number of changes in the policy from one time step to another. The changes in the policy are computed as the number of beliefs in B with a different optimal action from one time slice to the next. The number of policy changes drops to zero, indicating convergence with respect to the particular B. In Fig. 3 we show a typical trajectory of the robot when executing a policy found at convergence of PERSEUS. The snapshots show the evolution of the belief of the robot, and the actions taken, from the beginning of the episode (the robot starts at location 7) until the target door is entered.

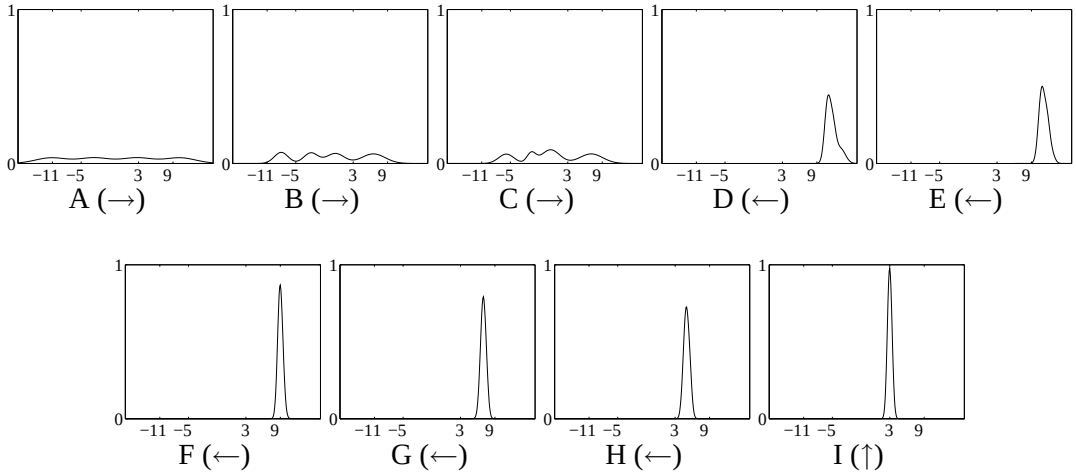

Figure 3: Evolution of the belief when following the discovered policy. The arrows under the snapshots represent the actions: → for moving right, ← for moving left, and ↑ for entering the door. The four numbers on the x-axis indicate the locations of the four doors.

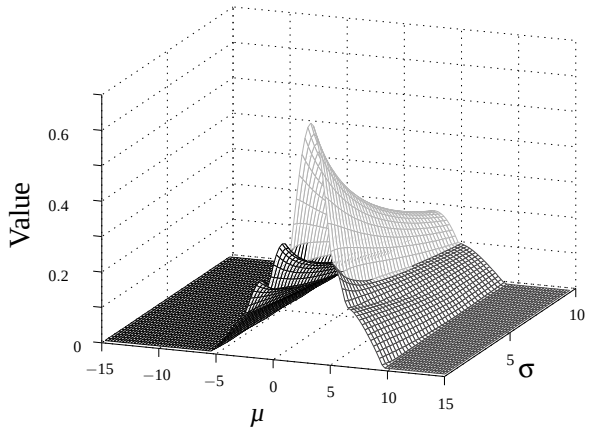

Figure 4: Value function for single-component beliefs as a function of the mean μ and the standard deviation σ.

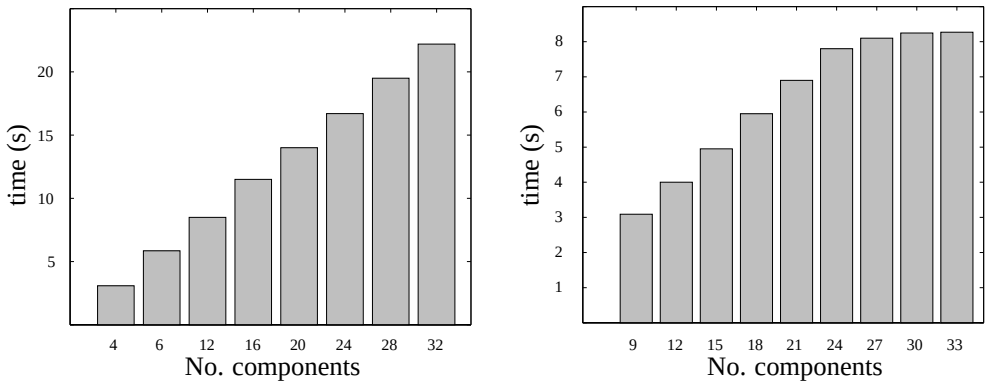

Figure 5: Execution time in seconds for the first iteration of PERSEUS as the number of components representing the beliefs increase (left) and as the number of components representing the α-functions increase (right).

Since the state space is one-dimensional in this example, beliefs with a single (Gaussian) component can be fully characterized by their sufficient statistics, that is, the mean μ and the variance σ^2. In Fig. 4 we plot the value of single-Gaussian beliefs for different μ and σ. We note that, as the uncertainty about the position of the robot grows (i.e., the σ is larger), the value of the corresponding belief decreases. The colors/shadings in the figure correspond the different actions: black for moving to the right, light-gray for entering the door, and dark-gray for moving to the left. This plot demonstrates that a value function that is convex over the belief space may not necessarily be convex over the space of sufficient statistics of the beliefs.

Fig. 5-left shows the increase in the execution time as more components are used to represent the beliefs. The plotted data correspond to the time in seconds for the first PERSEUS value update stage, that is, for the computation of the first `backup` (line 14 in Table 1) and the new value for all the beliefs in B (line 18 in the algorithm). The cost of executing the first iteration is an indicator of the computational complexity of the system that is independent of the problem at hand; the cost of later stages of PERSEUS scales with the number of elements in the previous value function approximation, V_{n-1}, and the number of elements to be generated for the new approximation, V_n, and both quantities are problem-dependent. We can see that the increase in the execution time is rather linear with the number of components in the belief. In all the experiments summarized in Fig. 5-left, we used nine components to represent the α-functions. To assess the effects of increasing the number of components in the α-functions, in Fig. 5-right we show the increase in the execution time for the first PERSEUS iteration when beliefs are represented with four components and the α-functions are represented with an increasing number of elements. We can observe that after about 24 components the execution time is almost constant. This is due to the fact that, for the problem at hand, no more components are needed to represent the α-functions. The Gaussian mixture condensation algorithm detailed in Appendix A has the property of discarding some components from the output if these are not necessary.

The effect on the quality of the solution when reducing the number of components for the beliefs and the α-functions can be seen in Fig. 6 where we depict the average accumulated discounted

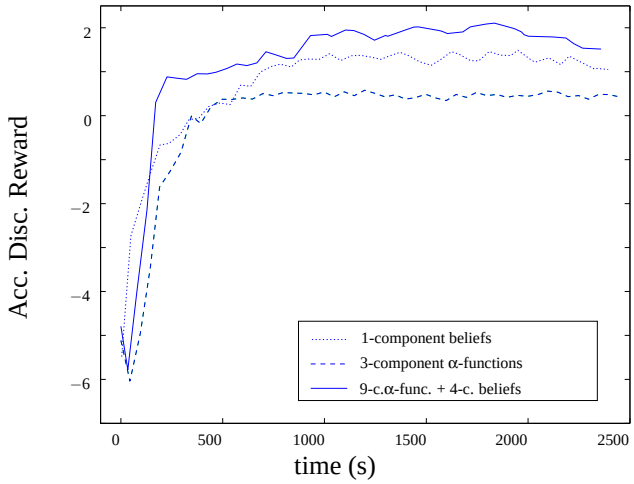

Figure 6: Reduction in the obtained average accumulated discounted reward when reducing the number of components in the beliefs to just one (dotted line) and in the α-functions to three (dashed line). The solid line is the average accumulated reward when using 4 components for the beliefs and 9 for the α-functions.

reward when representing beliefs with one component and α-functions with three components. We observe that when using fewer components for the α-functions, the algorithm may converge to a suboptimal policy. We also noticed that when using more than nine components, the improvements in the final policy were marginal. When representing the beliefs with just one component, the quality of the obtained policy also decreases. This is due to the fact that the problem at hand presents some degree of perceptual aliasing (i.e., states for which different actions are required but where the same observation is obtained). This aliasing can only be solved properly when using a multi-modal belief representation, which is not the case for single Gaussians.

We note that the advantage of using a continuous state space is that we obtain a scale-invariant solution. If we have to solve the same problem in a longer corridor, we can just scale the Gaussians used in the problem definition and we will obtain the solution with the same cost as we have now. The only difference is that more actions would be needed in each episode to reach the correct door.

Another way to solve this problem would be to discretize the state space and then apply the PERSEUS algorithm for discrete POMDPs. When discretizing the environment, the granularity has to be in accordance with the size of the actions taken by the robot (± 2 left/right) and, thus, the number of states and, consequently, the cost of the planning grows as the environment grows. Fig. 7-left shows the execution time in seconds for the first stage of PERSEUS in a discretized version of the problem as the number of states grows. The discretization is performed by selecting n states uniformly sampled on the state space and then using the continuous models to define the discrete ones. As we can see in the figure, the increase in the execution time with respect to the number of states is higher than linear. With more than 100 states the execution is slower than that for the continuous version when using 4 components for the beliefs and 9 for the α-functions (the dashed line in Fig. 7-left is the time for the execution of the first iteration of PERSEUS in this case).

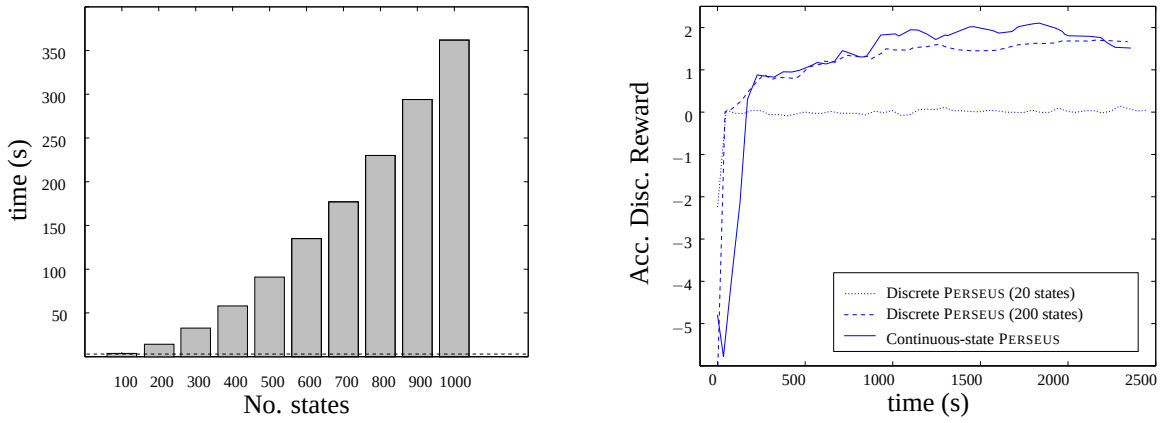

Figure 7: Left: Execution time in seconds for the first iteration of PERSEUS in a discretized version of the problem as the number of states grows. The dashed line is the time for the first iteration in the continuous-state version of the same problem. Right: Average accumulated discounted reward obtained with the continuous-state version of PERSEUS with 4 components for the beliefs and 9 for the α-functions (solid line) compared with the one obtained with the discrete version of PERSEUS using only 20 states (dotted line) and using 200 states (dashed line) .

Note that the discrete version of PERSEUS relies on linear algebra operations that can be sped up by taking advantage of the sparsity of the matrices and vectors defining the models and the beliefs, however, such speedups are not implemented in the version of PERSEUS we use for the experiments. A remarkable difference between the continuous and the discrete-state version is that the first one spends most of the time in the computation of the value for all beliefs (i.e., in the $\langle \cdot, \cdot \rangle$ operator) while the second one spends most of the time in the computation of the $\alpha^j_{a,o}$ vectors that are later on used in the backup. Fig. 7-right shows the average accumulated discounted reward obtained with the discrete version of PERSEUS working on different number of states compared with the one obtained in the first experiment (see Fig. 2). We can see that, when using a too coarse discretization (only 20 states) the discrete version of the problem does not capture all the features of the continuous one and, therefore, we observe convergence to a sub-optimal solution. Only when using enough states in the discretization the discrete version of PERSEUS delivers a plan that is as good as the one obtained with the continuous PERSEUS. The average accumulated discounted reward with a discretization with 200 states is shown in Fig. 7-right.

In the following experiment, the same problem was solved using particles to represent the beliefs instead of Gaussian mixtures. In this case, the α-functions are still represented as Gaussian mixtures, with 9 components. The results obtained using 75 particles are shown in Fig. 8. Note that the results are similar to those obtained when using Gaussian mixtures to represent the beliefs (see Fig. 2) but they are obtained in about 5 times more execution time. This is reasonable since, although the basic operations implementing the expectation operator $\langle \cdot, \cdot \rangle$ are more efficient when using particle-based beliefs, this is compensated by the fact that, in general, we have to use a large amount of particles. Therefore, the use of particles might only be advantageous when the belief cannot be represented

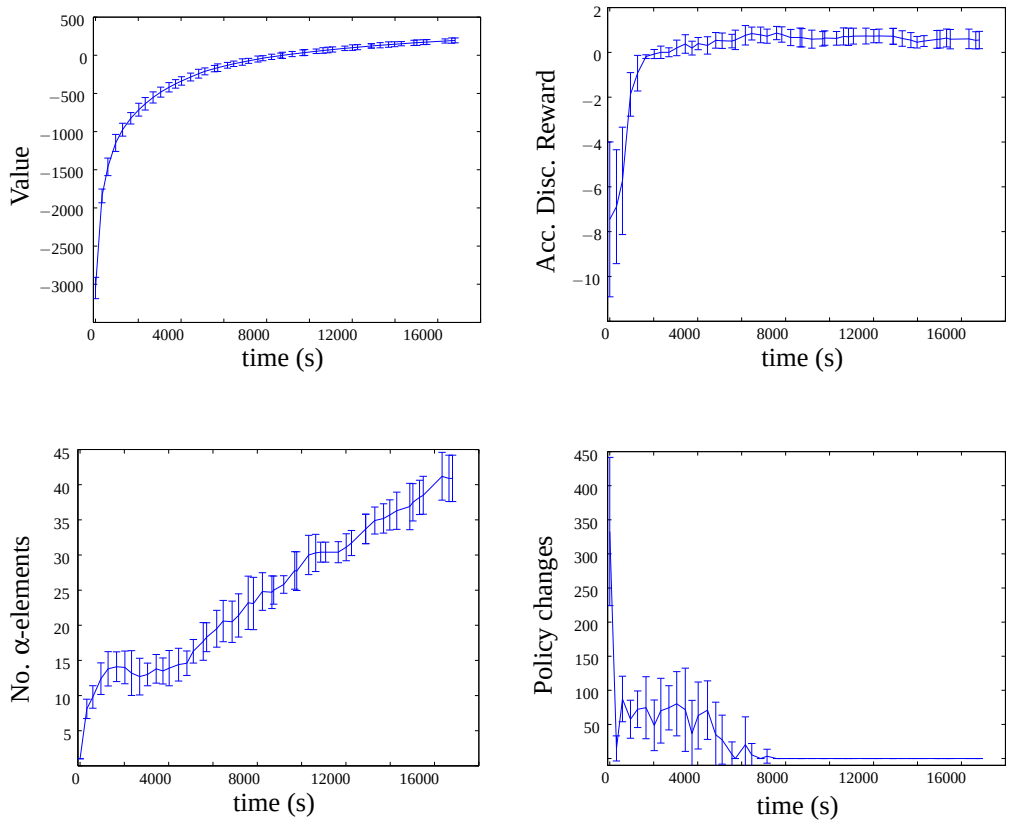

Figure 8: Results when using 75 particles to represent the beliefs. Top: Evolution of the value for all the beliefs in B and the average accumulated discounted reward for 10 episodes. Bottom: Number of elements in V_n and the number of policy changes. Results are averaged for 10 repetitions and the bars represent the standard deviation.

with a few-components Gaussian mixture, when the action model is not linear, or when using an on-line mechanism to dynamically adjust the number of particles (Fox, 2002, 2003).

Fig. 9 shows the average accumulated discounted reward using two different sets of actions. When using an action set including short robot movements (± 1), the number of steps to reach the target increase and, since positive reward is only obtained at the end of the run when entering the door, the average accumulated reward decreases. When using a set of actions with too large movements (± 4) the robot has problems aiming the correct door and the average accumulated reward also decreases. Since the appropriate set of actions for each problem is hard to forecast, it would be nice to have a planning system able to determine a proper set of actions by itself. For this purpose in the next experiment we let the robot execute actions in the continuous range $[-6, 6]$, where an action can be regarded here as a measure of velocity of the robot. When the robot is almost stopped (i.e., its velocity is below 5% of the maximum one) we interpret this as trying to enter a door. In each backup at planning stage n, we consider the optimal action according to V_{n-1} and three more

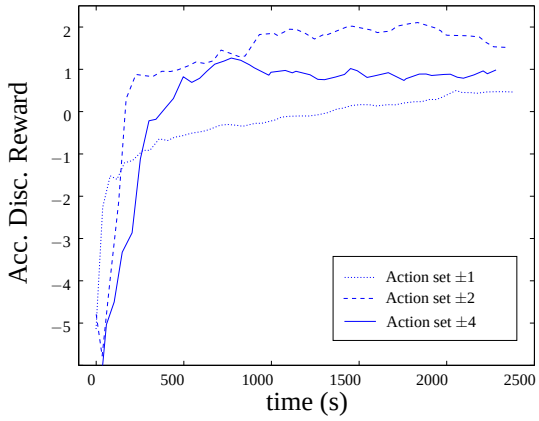

Figure 9: Average accumulated discounted reward using different sets of actions.

actions selected at random with uniform distribution in the range $[-6,6]$. Fig. 10 shows the average results obtained by 10 repetitions. The policy change in the bottom-right plot is computed as the sum squared difference of the actions in two consecutive PERSEUS iterations for all beliefs. The fact that this norm goes to zero means that policy gets stable and, observing the plot for the reward, we can see that the discovered policy is better than the one in Fig. 2, meaning that the algorithm is able to determine better motion actions than the ones we manually fixed in the initial version of the problem (± 2), and that is able to select *enter door* actions when necessary.

Finally, we modified the initial problem so that it is formalized with continuous state, action, and observation spaces. Here we assume that the robot observations are obtained with a noisy sensor that measures the width of the corridor. The observations are noise-perturbed versions of four nominal integer values: 1 for the right extreme, 2 for the left one, 4 for the doors, and 3 for the rest of positions (see Fig. 1-a). The sensor noise is assumed white Gaussian with covariance equal to 0.3, resulting in a continuous set of observations in the range $[0,5]$. For each backup, we still use four actions (the optimal one up for V_{n-1} and three randomly selected ones) and we discretize the observation space by uniformly sampling 100 observations. Fig. 11 shows the results obtained in this case. We see a performance similar to the one obtained with continuous states, continuous actions, and discrete observations, implying that the observation-space discretization does not affect the quality of the final policy. An interesting observation is that the algorithm converges faster and that the optimal value is approximated with fewer α-functions than when using discrete observations. This is probably due to the fact that the observation discretization takes advantage (and relies on) the structure of the previous approximation to the value function.

With this experiment we conclude our demonstration that the full continuous POMDP case can be addressed with the techniques proposed in this paper.

8. Related Work

The literature on POMDPs with continuous states is still relatively sparse. A common approach is to assume a discretization of the state space, which can be a poor model of the underlying system. However, when the system is linear and the reward function is quadratic, an exact solution for continuous-state POMDPs is known that can be computed in closed form (Bertsekas, 2001). While

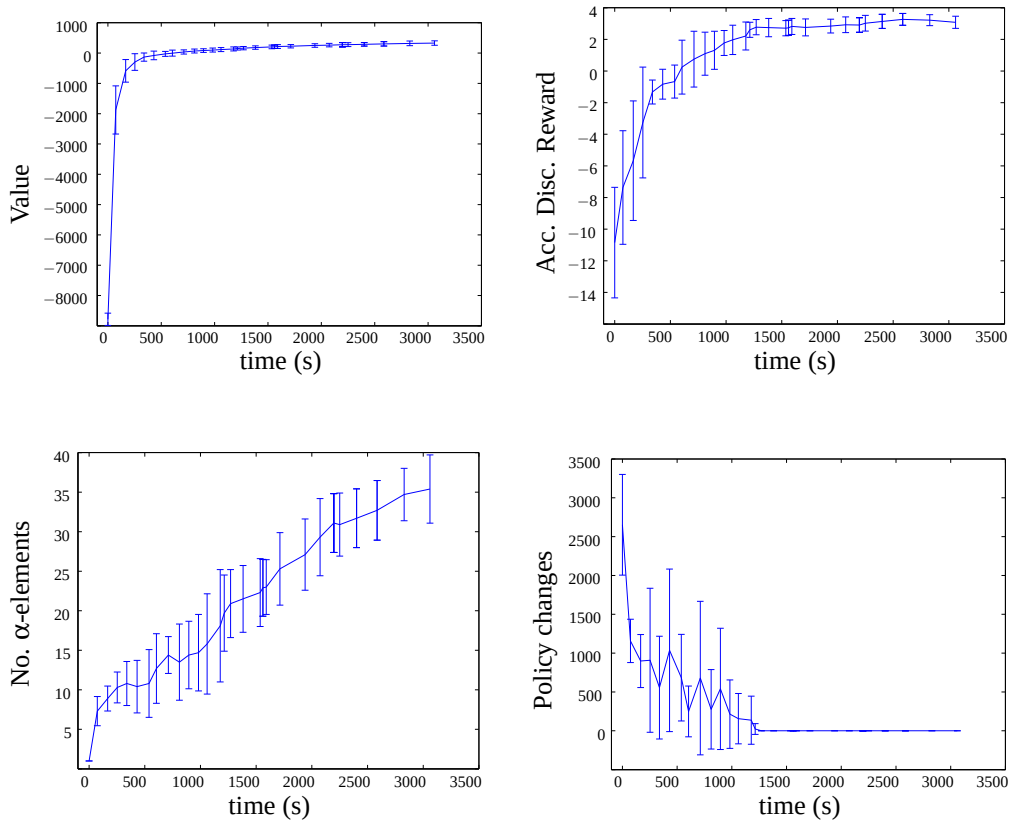

Figure 10: Results when the problem is modeled with continuous states and actions. Top: Evolution of the value for all the beliefs in B and the average accumulated discounted reward for 10 episodes. Bottom: Number of elements in V_n and the average policy change. Results are averaged for 10 repetitions and the bars represent the standard deviation.

such an assumption on the reward function can be reasonable in certain control applications, it is a severe restriction for the type of AI applications that we consider.

Roy (2003) has proposed compression techniques for handling POMDPs with large (discrete) state spaces, one of which compresses beliefs to two parameters: the state with maximum likelihood and the belief's entropy. Such a representation may lead to poor performance when multi-modal beliefs are likely to occur in a particular application. Recently, Brooks et al. (2006) have proposed a related parameterization of the beliefs using the sufficient statistics of an appropriately chosen parametric family (e.g., Gaussians). Both methods compute an approximate value function on a grid in their low-dimensional parameter spaces, and do not use the PWLC property of the POMDP value function. In contrast, we exploit the known shape of the value function, which offers an attractive potential for generalization through the use of α-functions, analogous to the effective exploitation of α-vectors in discrete-state POMDPs.

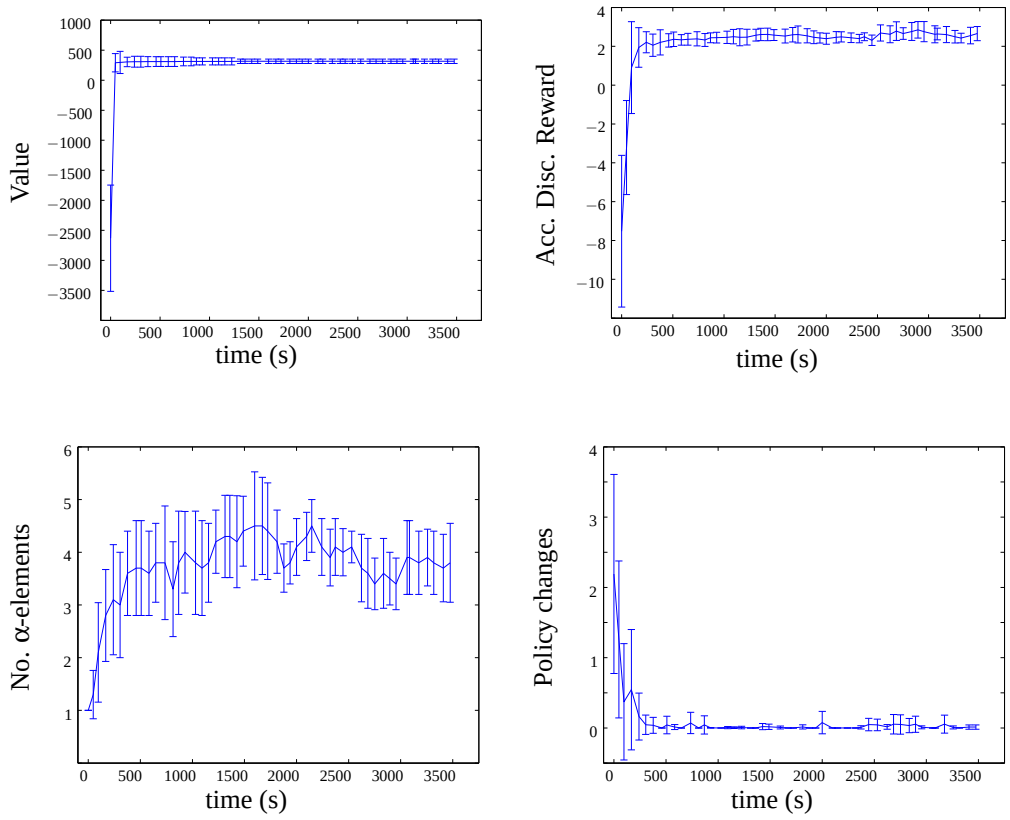

Figure 11: Results when the problem is modeled with continuous states, observations and actions. Top: Evolution of the value for all the beliefs in B and the average accumulated discounted reward for 100 episodes. Bottom: Number of elements in V_n and the number of policy changes. Results are averaged for 10 repetitions and the bars represent the standard deviation.

An approach to continuous-state POMDPs that is closely related to ours is the Monte Carlo POMDP (MC-POMDP) method of Thrun (2000), in which real-time dynamic programming is applied on a POMDP with a continuous state and action space. In that work beliefs are represented by sets of samples drawn from the state space, while $Q(b,a)$ values are approximated by nearest-neighbor interpolation from a (growing) set of prototype values and are updated by online exploration and the use of sampling-based Bellman backups. The MC-POMDP method approximates the Bellman backup operator by sampling from the belief transition model, whereas in our case, we compute the Bellman backup operator analytically given the particular value-function representation. In the MC-POMDP algorithm nearest-neighbor interpolation is used to approximate the value of beliefs outside the set. This is in contrast with our Gaussian-mixture representation, in which the value function achieves generalization through a set of α-functions. When the value function maintained by MC-POMDP does not contain enough neighbors within a certain distance for an encountered belief, the belief is added to the value function. PERSEUS operates on a fixed set of

beliefs, and does not require such an online expansion. Furthermore, the PERSEUS value function is likely to generalize better over the belief space through the use of α-functions. In contrast with PERSEUS, the MC-POMDP method does not exploit the piecewise linearity and convexity of the value function.

Duff (2002) considered the problem of Bayesian reinforcement learning, in which the parameters of the transition model of an MDP are treated as random variables. Experience in the form of observed state transitions and received rewards is used to estimate the unknown MDP models. In contrast with straightforward exploration strategies such as ε-greedy, Bayesian reinforcement-learning techniques try to identify the action that will maximize long-term reward. Such an optimally-exploring action might sacrifice expected immediate payoff for refining the model estimates, thus facilitating better control in the future. Duff (2002) models the Bayesian reinforcement-learning problem as a POMDP, in which the parameters of the transition model form the state of the system, and experienced transition tuples (s,a,s') are the possible observations. Such a POMDP has a continuous state space as the transition probabilities can be any real number between zero and one. A Monte Carlo algorithm is proposed for learning a (stochastic) finite-state controller for this particular class of POMDPs, where the required integrals are approximated by sampling and numerical methods. Recently, Poupart et al. (2006) demonstrated that the optimal value function in Bayesian reinforcement learning can be represented by a set of multivariate polynomials, in direct analogy to the α-function representations for Gaussian-based POMDPs in this paper.

In the case of continuous action spaces only few methods exist that can handle continuous action spaces directly (Thrun, 2000; Ng and Jordan, 2000; Baxter and Bartlett, 2001). Certain policy search methods tackle continuous actions, for instance Pegasus (Ng and Jordan, 2000), which estimates the value of a policy by simulating trajectories from the POMDP using a fixed random seed, and adapts its policy in order to maximize this value. Pegasus can handle continuous action spaces at the cost of a sample complexity that is polynomial in the size of the state space (Ng and Jordan, 2000, Theorem 3). Baxter and Bartlett (2001) propose a policy gradient method that searches in the space of randomized policies, and which can also handle continuous actions. The main disadvantages of policy search methods are the need to choose a particular policy class and the fact that they are prone to local optima.

Traditional POMDP methods assume discretized observation spaces. POMDPs with continuous observation spaces have mainly been studied in model-free settings, for instance to learn policies for a partially observable version of the classic pole-balancing task (Whitehead and Lin, 1995; Meuleau et al., 1999; Bakker, 2003). Rudary et al. (2005) extend Predictive State Representations (PSRs) to the linear-Gaussian case, which allows them to learn a PSR model of a linear dynamical system with a continuous observation space. Finally, the analytic solution for the quadratic reward case mentioned above can also handle continuous observations with a linear noise model (Bertsekas, 2001).

9. Conclusions and Future Work

In this paper we described an analytical framework for optimizing POMDPs with continuous states, actions, and observations. For POMDPs with continuous states, we demonstrated the piecewise linearity and convexity of value functions defined over infinite-dimensional belief states induced by continuous states. We also demonstrated that continuous Bellman backups are isotonic and contracting, allowing value iteration to be adapted to continuous POMDPs. In particular, we extended

the PERSEUS algorithm with linear combinations of Gaussians and particle-based representations for belief states. These are expressive representations that are closed under Bellman backups and belief updates. Finally, we also extended PERSEUS to continuous actions and observations by particular sampling strategies that reduce the problem to a continuous state POMDP that can be tackled by the PERSEUS algorithm.

In the near future, we plan to investigate reinforcement learning approaches for scenarios where the POMDP model is unknown. The particle-based approach may be adaptable to reinforcement learning since particles may be thought as sampled values. Conversely, note that discrete Bayesian reinforcement learning can be cast as a POMDP with continuous states (Duff, 2002). Poupart et al. (2006) recently developed a similar technique to optimize policies in environments with (partially) unknown transition dynamics modeled by multinomials. It would be interesting to follow up on this work by tackling Bayesian reinforcement learning problems with Gaussian-based dynamics.

Another interesting research direction would be to investigate which families of functions (beyond mixtures of Gaussians) are closed under Bellman backups and belief updates for different types of transition, observation and reward models. In particular, mixtures of log-linear distributions provide an expressive parameterization that is likely to possess the necessary properties.

Acknowledgments

We would like to thank Jakob Verbeek and Wojtek Zajdel for their contributions to the work reported here. Josep M. Porta has been partially supported by a *Ramón y Cajal* contract from the Spanish government and by the EU PACO-PLUS Project FP6-2004-IST-4-27657. Nikos Vlassis and Matthijs Spaan are supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414. Pascal Poupart is supported by the Canada's National Science and Engineering Research Council.

Appendix A.

As a large number of components representing beliefs and α-functions slows down the basic operations of the algorithm, an efficient implementation of the algorithm is required to keep the number of components reasonably bounded.

We use the procedure described by Goldberger and Roweis (2005) that transforms a given Gaussian mixture with k components to another Gaussian mixture with at most m components, $m < k$, while retaining the initial component structure. The algorithm is detailed in Table 3.

The algorithm uses the Kullback-Leibler, KL, distance between to Gaussian distributions $f_i = N(\mu,\Sigma)$, $g_j = N(\mu',\Sigma')$ that is

$$KL(f_i \| g_j) = \frac{1}{2}\left(\log\frac{|\Sigma'|}{|\Sigma|} + Tr((\Sigma')^{-1}\Sigma) + (\mu-\mu')^\top(\Sigma')^{-1}(\mu-\mu') - c\right),$$

with c the dimensionality of the space where the Gaussians are defined.

Observe that the above procedure is defined for Gaussian mixtures (with positive weights that sum to 1), but our α-functions are linear combinations of Gaussian (with possibly negative weights). Therefore, for the α-function compression, we use a modified version of the procedure just described where the weights are normalized after taking its absolute value. This way, the distance

Gaussian Mixture Condensation(f, m)
Input: A Gaussian mixture $f = \sum_{i=1}^{k} w_i \, f_i(x|\mu_i, \Sigma_i)$.
The maximum number of components
in the output mixture, m, $m < k$.
Output: A Gaussian mixture $g = \sum_{i=1}^{m} w'_i \, g_i(x|\mu'_i, \Sigma'_i)$ that
locally minimizes $\sum_{i=1}^{k} w_i \, min_{j\in[1,m]} KL(f_i \| g_j)$

1: **Initialize**
2: **for** $j = 1$ **to** m
3: $w'_j \leftarrow w_j$
4: $\mu'_j \leftarrow \mu_j$
5: $\Sigma'_j \leftarrow \Sigma_j$
6 : $d \leftarrow \sum_{i=1}^{k} w_i \min_{j\in[1,m]} KL(f_i \| g_j)$
7: **do**
8: **Compute the mapping from** f **to** g
9: **for** $i = 1$ **to** k
10: $\pi(i) \leftarrow \arg\min_{j\in[1,m], w'_j>0} KL(f_i \| g_j)$
11: **Define a new** g
12: **for** $j = 1$ **to** m
13: $I_j \leftarrow \{i \,|\, \pi(i) = j, \, i \in [1,k]\}$
14: $w'_j \leftarrow \sum_{i\in I_j} w_i$
15: $\mu'_j \leftarrow \frac{1}{w'_j} \sum_{i\in I_j} w_i \, \mu_i$
16: $\Sigma'_j \leftarrow \frac{1}{w'_j} \sum_{i\in I_j} w_i \, (\Sigma_i + (\mu_i - \mu'_j)(\mu_i - \mu'_j)^\top)$
17: $d' \leftarrow d$
18: $d \leftarrow \sum_{i=1}^{k} w_i KL(f_i \| g_{\pi(i)})$
19: **until** $d < \varepsilon$ **or** $\frac{|d-d'|}{d} < \varepsilon$

Table 3: Gaussian mixture condensation algorithm where ε is a sufficiently small threshold (10^{-5} in our implementation).

(locally) minimized by the algorithm in Table 3 is

$$d = \sum_{i=1}^{k} |w_i| \min_{j\in[1,m]} KL(f_i \| g_j).$$

Therefore, the algorithm tries to preserve the relevant peaks (either positive or negative) in the original mixture. After the compression, the weights are re-computed taken into account the original weights and the map π provided by the algorithm above.

References

K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

D. Aberdeen and J. Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the International Conference on Machine Learning*, pages 3–10, Sydney, Australia, 2002.

B. Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems 15 (NIPS-2002)*, pages 1475–1482. MIT Press, 2003.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

R. E. Bellman. *Dynamic Programming*. Princenton University Press, 1957.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2001. 2nd Edition.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1293–1299, Edinburgh, Scotland, 2005.

C. Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 239–246, Edmonton, AB, 2002.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1168–1175, Portland, OR, 1996.

A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte. Planning in continuous state spaces with parametric POMDPs. In *Reasoning with Uncertainty in Robotics, Workshop of the International Joint Conference on Artificial Intelligence*, pages 40–47, 2006.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: discrete Bayesian models for mobile-robot navigation. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 54–61, 1997.

H. T. Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, 1988.

T. Darrell and A. P. Pentland. Active gesture recognition using partially observable Markov decision processes. In *IEEE International Conference on Pattern Recognition*, pages 984–988, Vienna, Austria, 1996.

F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte-Carlo localization for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1322–1328, 1999.

A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo in Practice*. Springer-Verlag, New York, 2001.

M. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massassachusetts Amherst, 2002.

E. B. Dynkin. Controlled random sequences. *Theory of probability and its applications*, 10(1): 1–14, 1965.

D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14 (NIPS-2001)*, pages 713–720. MIT Press, 2002.

D. Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22(10-11):985–1004, 2003.

J. Goldberger and S. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17 (NIPS-2004)*, pages 505–512. MIT Press, 2005.

J. Hoey and P. Poupart. Solving pomdps with continuous or large discrete observation spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1332–1338, 2005.

M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

C. Lusena, J. Goldsmith, and M. Mundhenk. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:83–103, 2001.

O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 541–548, Orlando, FL, 1999.

N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 427–436, Stockholm, 1999.

G. E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the National Conference on Artificial Intelligence*, pages 587–592, Edmonton, AB, 2002.

A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 406–415, Stanford, CA, 2000.

C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1025–1032, 2003a.

J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, 2003b.

M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statististical Association*, 94(446):590–599, 1999.

J. M. Porta and B. J. A. Kröse. Appearance-based concurrent map building and localization using a multi-hypotheses tracker. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 3424–3429, Sendai, Japan, 2004.

J. M. Porta, M. T. J. Spaan, and N. Vlassis. Robot planning in partially observable continuous domains. In *Robotics: Science and Systems I*, pages 217–224, MIT, Cambridge, MA, 2005.

P. Poupart and C. Boutilier. Value-directed compressions of POMDPs. In *Advances in Neural Information Processing Systems 15 (NIPS-2002)*, pages 1547–1554. MIT Press, 2003.

P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16 (NIPS-2003)*, pages 823–830. MIT Press, 2004.

P. Poupart and C. Boutilier. VDCBPI: an approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems 17 (NIPS-2004)*, pages 1081–1088. MIT Press, 2005.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 697–704, 2006.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, Inc., 1994.

N. Roy. *Finding Approximate POMDP Solutions Through Belief Compression*. PhD thesis, Carnegie Mellon University, 2003.

N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems 15 (NIPS-2002)*, pages 1635–1642. MIT Press, 2003.

N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

N. Roy, J. Pineau, and S. Thrun. Spoken dialog management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 93–100, Hong Kong, 2000.

Matt Rudary, Satinder Singh, and David Wingate. Predictive linear-gaussian models of stochastic dynamical systems. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 501–508, 2005.

R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.

T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 520–527, Banff, Alberta, 2004.

E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1347–1352, 2002.

S. Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12 (NIPS-1999)*, pages 1064–1070. MIT Press, 2000.

N. Vlassis, B. Terwijn, and B.J.A. Kröse. Auxiliary particle filter robot localization from high-dimensional sensor observations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 7–12, 2002.

Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, 1995.

J. Williams, P. Poupart, and S. Young. Using factored Markov decision processes with continuous observations for dialogue management. Technical Report CUED/F-INFEG/TR.520, Cambridge University, Engineering Department, 2005.

B. Zhang, Q. Cai, J. Mao, and B. Guo. Planning and acting under uncertainty: a new model for spoken dialogue systems. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 572–579, Seattle, WA, 2001.

N. L. Zhang and W. Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.