Robotics: Science and Systems 2005
Cambridge, MA, USA, June 8-11, 2005

Robot Planning in Partially Observable Continuous Domains

Josep M. Porta
Institut de Robòtica i
Informàtica Industrial (UPC-CSIC)
Llorens i Artigas 4-6, 08028, Barcelona
Spain
Email: porta@iri.upc.edu

Matthijs T. J. Spaan
Informatics Institute
University of Amsterdam
Kruislaan 403, 1098SJ, Amsterdam
The Netherlands
Email: mtjspaan@science.uva.nl

Nikos Vlassis
Informatics Institute
University of Amsterdam
Kruislaan 403, 1098SJ, Amsterdam
The Netherlands
Email: vlassis@science.uva.nl

***Abstract*— We present a value iteration algorithm for learning to act in Partially Observable Markov Decision Processes (POMDPs) with continuous state spaces. Mainstream POMDP research focuses on the discrete case and this complicates its application to, e.g., robotic problems that are naturally modeled using continuous state spaces. The main difficulty in defining a (belief-based) POMDP in a continuous state space is that expected values over states must be defined using integrals that, in general, cannot be computed in closed from. In this paper, we first show that the optimal finite-horizon value function over the continuous infinite-dimensional POMDP belief space is piecewise linear and convex, and is defined by a finite set of supporting α-functions that are analogous to the α-vectors (hyperplanes) defining the value function of a discrete-state POMDP. Second, we show that, for a fairly general class of POMDP models in which all functions of interest are modeled by Gaussian mixtures, all belief updates and value iteration backups can be carried out analytically and exact. A crucial difference with respect to the α-vectors of the discrete case is that, in the continuous case, the α-functions will typically grow in complexity (e.g., in the number of components) in each value iteration. Finally, we demonstrate PERSEUS, our previously proposed randomized point-based value iteration algorithm, in a simple robot planning problem with a continuous domain, where encouraging results are observed.**

I. INTRODUCTION

A popular formalism for decision making under uncertainty is the Markov Decision Process (MDP) framework [1]. In this paradigm, an agent interacts with a given system by executing actions that change the state of the system stochastically and that provide rewards or penalties to the agent. The objective of the learning agent is to identify for each state the action that produces the most reward in the long term. When the decision making has to performed based on uncertain information about the state of the system, the task is naturally formalized as a Partially Observable Markov Decision Process (POMDP) [2]–[6]. POMDPs have often been used as a framework for planning in robotics [7]–[10]. In general, computing the exact solution of a POMDP is an intractable problem [11], [12], even for the discrete case (i.e., discrete sets of states, actions, and observations). Two main factors cause this high computational cost [13]. The first one is the *curse of history*: the number of action-observation sequences to be considered increases exponentially as we extend the planning horizon. Fortunately the curse of history can be minimized by limiting ourselves to approximate solutions [13], [14]. The second factor that makes POMDP algorithms inefficient is the *curse of dimensionality*: the computational cost of discrete state POMDP algorithms scales with the number of states. Therefore, the finer the granularity of the state space discretization, the higher the cost of solving the POMDP. One insight we can extract from this fact is that it would be desirable to avoid the discretization of the state space. Moreover, real world problems are naturally formalized using continuous spaces. For instance, in a robot navigation problem, the state to be estimated is the pose of the robot that, for a robot moving on a planar surface, is naturally defined in the continuous space of the Cartesian coordinates of the robot and its orientation. Linear POMDPs with continuous states and quadratic reward functions have a closed solution [15]. However, this is a too restrictive case for many practical purposes. Existing algorithms for continuous-state POMDPs with general reward functions are based on policy search [16], [17] or approximate (grid-based) value iteration [18], [19]. For discrete-state POMDPs, recent promising algorithms are based on point-based value iteration [13], [14].

In this paper, we present a novel approach to solve POMDPs in continuous state spaces via value iteration. The main difficulty of working in continuous state spaces is that expected values over states must be defined using integrals. These integrals cannot be computed in closed form for general functions and, therefore, only approximation techniques can be used [19]. In our approach, we restrict all functions defined on the state space to a particular, although highly expressive, family of functions: linear combinations of Gaussians. This allows us to evaluate all integrals involved in the value iteration POMDP formulation in closed form. Using this fact, we can adapt to the continuous case the rich machinery developed for discrete-state POMDP value iteration, in particular the point-based algorithms.

This paper is organized as follows. First, in Section II, we review the POMDP framework and the value iteration process for discrete-state POMDPs. In Section III, we generalize the value function representation commonly used in discrete-state POMDPs to continuous-state ones. This allows us to do value iteration for the continuous case. In Section IV, we derive closed formulas for the elements involved in the value iteration

framework introduced in Section III, assuming a Gaussian-based representation for the beliefs and the models defining the POMDP. In Section V, we use these closed formulas to define a point-based algorithm for Gaussian-based POMDPs. In Section VI, we present some results with the proposed algorithm and, in Section VII, we summarize our work and point to directions for further research.

II. Preliminaries: POMDPs

A POMDP models an agent interacting with a system using the following elements

- A set of system states, S.
- A set of agent actions, A.
- A set of observations, O.
- An action (or transition) model defined by $p(s'|a,s)$, the probability that the system changes from state s to s' when the agent executes action a.
- An observation model defined by $p(o|s)$, the probability that the agent observes o when the system reaches state s.
- A reward function defined as $r_a(s) \in \mathbb{R}$, the reward obtained by the agent if it executes action a when the the system is in state s.

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, the system state changes to s' and, then, the agent observes o. The knowledge of the agent about the system state is represented as a *belief*, i.e., a probability distribution over the state space. The initial belief is assumed to be known and, for a discrete set of states, if b is the belief of the agent about the state, the belief after executing action a and observing o is

$$b^{a,o}(s') = \frac{p(o|s')}{p(o|a,b)} \sum_{s \in S} p(s'|s,a)\, b(s). \tag{1}$$

A function mapping beliefs to actions is called a *policy*. An optimal policy is one that, on the average, generates as much reward as possible in the long term. The *value function* condenses the immediate and delayed reward that can be obtained from a given belief. This function can be expressed in a recursive way

$$V_n(b) = \max_a Q_n(b,a), \tag{2}$$

with

$$Q_n(b,a) = \sum_{s \in S} r_a(s)\, b(s) + \gamma \sum_o p(o|b,a)\, V_{n-1}(b^{a,o}), \tag{3}$$

where n is the planning horizon, S and O are assumed discrete and $\gamma \in [0,1)$ is a discount factor that trades off the importance of the immediate and the delayed reward. The above recursion is usually written in functional form

$$V_n = H\, V_{n-1} \tag{4}$$

and it is known as the *Bellman recursion* [20]. This recursion converges to a fixed point V^* that is the optimal value function [1] An optimal policy π^* can be defined as

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

for Q^* the Q-function associated with the optimal value function, V^*.

Value iteration for POMDPs [2], [6], [21] generates a sequence of functions V_i using the recurrence in Eq. 4 that progressively approach V^* and computes an approximately optimal policy from the final V_i.

At first sight the value function seems intractable, but it can be expressed in a simple form [2]

$$V_n(b) = \max_{\{\alpha_n^i\}_i} \sum_s \alpha_n^i(s)\, b(s),$$

with $\{\alpha_n^i\}_i$ a set of vectors. Using this formulation, value iteration algorithms typically focus on the computation of the α_n-vectors.

III. POMDPs in Continuous State Spaces

In this section, we generalize POMDPs to continuous state spaces, while still assuming discrete action and observation spaces. With this formulation, we avoid the necessity of discretizing the state space and, thus, we reduce the chance of being affected by the curse of dimensionality.

In the discrete case, expectations for a given belief are computed by summing over the state space (see Eqs. 1 and 3). The generalization to the continuous case amounts to computing these expected values by integrating instead of summing. Thus we have

$$b^{a,o}(s') = \frac{p(o|s')}{p(o|a,b)} \int_s p(s'|s,a)\, b(s), \tag{5}$$

and

$$Q_n(b,a) = \int_s r_a(s)\, b(s) + \gamma \sum_o p(o|b,a)\, V_{n-1}(b^{a,o}), \tag{6}$$

where $r_a : S \to \mathbb{R}$ is a continuous reward function for action a.

With a continuous state space, the belief space is also continuous, as in the discrete case, but now with an infinite number of dimensions. However, there are several properties typical of value functions for discrete state spaces that still hold in the continuous case. Namely, we can prove [22] that (1) the optimal finite-horizon value function is piecewise linear and convex (PWLC) in the belief space, (2) the value function recursion is isotonic, and (3) this recursion is also a contraction (and thus, the iterative computation of the value function for increasing horizons will converge to the optimal value function V^*).

The PWLC is a basic property since it allows to represent the value function using a small set of supporting elements. This kind of representation is the key element to define the value iteration process. To prove this property, we first need to prove the following lemma.

Lemma 1: The value function in a continuous-state POMDP can be expressed as

$$V_n(b) = \max_{\{\alpha_n^i\}_i} \int_s \alpha_n^i(s)\, b(s),$$

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

Proof: The proof, as in the discrete case, is done via induction. For planning horizon 0, we only have to take into account the immediate reward and, thus, we have that

$$V_0(b) = \max_a \int_s r_a(s)\, b(s),$$

and, therefore, if we define

$$\{\alpha_0^i(s)\}_i = \{r_a(s)\}_{a\in A},$$

we have that, as desired

$$V_0(b) = \max_{\{\alpha_0^i\}_i} \int_s \alpha_0^i(s)\, b(s).$$

For the general case, we have that, using Eqs. 2 and 6,

$$V_n(b) = \max_a \Big\{ \int_s r_a(s)\, b(s) + \gamma \sum_o p(o|a,b)\, V_{n-1}(b^{a,o}) \Big\},$$

and, by the induction hypothesis,

$$V_{n-1}(b^{a,o}) = \max_{\{\alpha_{n-1}^j\}_j} \int_{s'} \alpha_{n-1}^j(s')\, b^{a,o}(s').$$

From Eq. 5,

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

and, therefore,

$$V_n(b) = \max_a \Big\{ \int_s r_a(s)\, b(s) \quad + \gamma \sum_o \max_{\{\alpha_{n-1}^j\}_j} \int_{s'} \alpha_{n-1}^j(s')\, p(o|s') \int_s p(s'|s,a)\, b(s) \Big\}$$
$$= \max_a \Big\{ \int_s r_a(s)\, b(s) \quad + \gamma \sum_o \max_{\{\alpha_{n-1}^j\}_j} \int_s \Big[\int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|s,a) \Big] b(s) \Big\}.$$

At this point, we define

$$\alpha_{a,o}^j(s) = \int_{s'} \alpha_{n-1}^j(s')\, p(o|s')\, p(s'|s,a). \tag{7}$$

With this, we have that

$$V_n(b) = \max_a \Big\{ \int_s r_a(s)\, b(s) + \gamma \sum_o \max_{\{\alpha_{a,o}^j\}_j} \int_s \alpha_{a,o}^j(s)\, b(s) \Big\},$$

and we define

$$\alpha_{a,o,b} = \arg\max_{\{\alpha_{a,o}^j\}_j} \int_s \alpha_{a,o}^j(s)\, b(s). \tag{8}$$

Observe that, for a given a and o, $\alpha_{a,o,b}$ is just one of the M elements in the set $\{\alpha_{a,o}^j\}_j$. Using a reasoning parallel to that of the enumeration phase of the Monahan's algorithm [3], we can have, at most, $|A|M^{|O|}$ different $\alpha_{a,o,b}$-functions. The finite cardinality of this set is a crucial point since it proves that we can represent $V_n(b)$ with a finite set of supporting α-functions, despite the infinite dimensionality of the belief space.

Using the above, we can write

$$V_n(b) = \max_a \Big\{ \int_s r_a(s)\, b(s) + \gamma \sum_o \int_s \alpha_{a,o,b}(s)\, b(s) \Big\}$$
$$= \max_a \Big\{ \int_s \Big[r_a(s) + \gamma \sum_o \alpha_{a,o,b}(s) \Big] b(s) \Big\}.$$

If we define

$$\{\alpha_n^i(s)\}_i = \{r_a(s) + \gamma \sum_o \alpha_{a,o,b}(s)\ \}_{a\in A}, \tag{9}$$

we have V_n in the desired form

$$V_n(b) = \max_{\{\alpha_n^i\}_i} \int_s \alpha_n^i(s)\, b(s), \tag{10}$$

and, thus, the lemma holds. ■

Lemma 2: The value function is PWLC in the belief space.

Proof: It holds that

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

with

$$V_n^i(b) = \int_s \alpha_n^i(s)\, b(s).$$

For a particular V_n^i clearly holds

$$V_n^i(\kappa\, b_1 + \lambda b_2) = \kappa\, V_n^i(b_1) + \lambda\, V_n^i(b_2),$$

for arbitrary κ and λ. Therefore, each V_n^i is a linear function in b.

The *piecewise linearity* part of the property is given by the fact that the $\{\alpha_n^i\}_i$ set is of finite cardinality and, as shown above, V_n is linear, for each individual α_n^i. Finally, the convexity is given by the fact that we take the maximum of convex (linear) functions when computing the value function and, thus, we obtain a convex function as a result. ■

Eqs. 7 to 9 constitute the value iteration process for continuous state POMDP since they provide a constructive way to determine the elements (i.e., the α-functions) defining V_n from those defining V_{n-1}.

IV. Gaussian-based POMDPs

In previous section, we left as an open point how to actually compute the belief update (Eq. 5), the steps in the value iteration process (Eqs. 7 to 9), and the value for a given belief point (Eq. 10). In this section, we show how these computations are possible assuming that the beliefs as well as the observation, action, and reward models are represented as linear combinations of Gaussians. We first formally introduce our assumptions on the models (Section IV-A) and then we define the belief update (Section IV-B) and the basic value iteration steps (Section IV-C) for Gaussian-based POMDPs.

Note that other families of integrable functions could be used to determine the α-functions in closed form, but Gaussian-based models provide a high degree of flexibility and are of common use in many applications, including robotics [23], [24].

A. Models for Gaussian-based POMDPs

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 continuous, infinite-dimensional belief space. However, only Gaussian mixtures with few components are needed in practical situations.

We assume that our observation model is defined non-parametrically from a set of samples $T = \{(s_i, o_i) \mid i \in [1, N]\}$ with o_i an observation obtained at state s_i. Using these samples, the observation model can be defined as

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

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

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

with s_i^o one of the N_o points in T with o as an associated observation and where $w_i^o = \lambda_i^o / N$ and Σ_i^o are, respectively, a weighting factor and a covariance matrix associated with that training point. The sets $\{\lambda_i^o\}_i$ and $\{\Sigma_i^o\}_i$ are defined 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 action model is concerned, we assume it is linear-Gaussian

$$p(s'|s, a) = \phi(s'|s + \Delta(a), \Sigma^a). \tag{12}$$

Non-linear action models can be approximated as it is done, for instance, in the extended Kalman filter or in the unscented Kalman filter [25]. The function $\Delta : A \to S$ implements the transition model of the system.

Finally, the reward can be seen as an observation with an associated scalar value. Therefore, assuming a finite set of possible rewards $R = \{r_i \mid i \in [1, M]\}$, the reward model $p(r|s, a)$ for each particular a can be represented in the same way as the observation model

$$p(r|s, a) \approx \sum_{i=1}^{M_r} w_i^r \, \phi(s|s_i^r, \Sigma_i^r).$$

With that, we have that

$$r_a(s) = \sum_{r \in R} r \, p(r|s, a) \approx \sum_{r \in R} r \sum_{i=1}^{M_r} w_i^r \, \phi(s|s_i^r, \Sigma_i^r),$$

that is an unnormalized Gaussian mixture.

B. Belief update for Gaussian-Based POMDPs

The belief update on Eq. 5 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 convolution of the Gaussians representing $b(s)$ (Eq. 11) with the Gaussian representing the action model (Eq. 12). This convolution results in

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

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

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

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

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

The proportionality in the definition of $b^{a,o}(s')$ implies that the weights $(w_i^o \, w_j \, \delta_{i,j}^{a,o}, \forall i, j)$ should be scaled to sum to one.

C. Backup Operator for Gaussian-Based POMDPs

The computation of the mapping H (Eq. 4) for a given belief point b is called a *backup*. This mapping determines the α function (or α-vectors in the discrete case) to be included in V_n for a belief point under consideration (see Eqs. 7 to 9). A full backup, i.e., a backup for the whole belief space, involves the computation of all relevant α-functions for V_n. Full backups are computationally expensive (in the discrete case they involve the use of linear programming in order to determine a sufficient set of points on which to backup), but the backup for a single belief point is relatively cheap. This is exploited by the point-based POMDP algorithms to efficiently approximate V_n on a fixed set of belief points [13], [14]. Next, we describe the backup operator on a continuous state space that we will use later in the PERSEUS algorithm.
The backup for a given belief point b is

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

where $\alpha_n^i(s)$ is defined in Eqs. 8 and 9 from the $\alpha_{a,o}$-functions (Eq. 7).

Lemma 3: The functions $\alpha_n^i(s)$ can be expressed as linear combinations of Gaussians, assuming the sensor, action and reward models are also Gaussian-based.

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 an unnormalized Gaussian mixture. 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. 7 is the integral of three linear combinations of Gaussians

$$\alpha_{a,o}^j(s) = \int_{s'} \Big[\sum_k w_k^j \, \phi(s'|s_k^j, \Sigma_k^j)\Big]\Big[\sum_l w_l^o \, \phi(s'|s_l^o, \Sigma_l^o)\Big] \times \phi(s'|s + \Delta(a), \Sigma^a)$$

$$= \sum_{k,l} w_k^j \, w_l^o \int_{s'} \phi(s'|s_k^j, \Sigma_k^j)\phi(s'|s_l^o, \Sigma_l^o)\phi(s'|s + \Delta(a), \Sigma^a).$$

In this case, we have to perform the product of two Gaussians 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 terms $\delta_{k,l}^{j,o}$ and $\beta_{k,l}^{j,o,a}(s)$ can be expressed as

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

with

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

With this, we have

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

Once we have the $\alpha_{a,o}^j$-functions, we can compute the α_n^i-functions. To do that, we need to determine the $\alpha_{a,o}^j$ for which

$$\int_s \alpha_{a,o}^j(s) \, b(s)$$

is maximized. Since the integral of the product of two Gaussian mixtures (in particular an α-function and a belief point) is a rather common operation in the continuous state POMDP framework we will denote it by

$$\langle \alpha, b \rangle = \int_s \alpha(s) \, b(s).$$

This operator can be computed as

$$\langle \alpha, b \rangle = \int_s \Big[\sum_k w_k \, \phi(s|s_k, \Sigma_k)\Big]\Big[\sum_j w_j \, \phi(s|s_j, \Sigma_j)\Big]$$
$$= \sum_{k,j} w_k \, w_j \int_s \phi(s|s_k, \Sigma_k) \, \phi(s|s_j, \Sigma_j)$$
$$= \sum_{k,j} w_k \, w_j \, \phi(s_j|s_k, \Sigma_k + \Sigma_j).$$

Using this operator and Eqs. 8 and 9, we define

$$\{\alpha_n^i(s)\}_i = \{r_a(s) + \gamma \sum_o \arg\max_{\{\alpha_{a,o}^j\}_j} \langle \alpha_{a,o}^j, b \rangle\}_{a \in A}.$$

Since all elements involved in the definition are linear combination of Gaussians so is the final result. ∎

Using the above lemma, the `backup` function is

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

and the value of V_n at b (Eq. 10) is simply

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

V. Continuous-State Perseus

In this section, we use the `backup` operator to extend to the continuous case the point-based value iteration algorithm Perseus [14], [26], which has been shown to be very efficient for discrete state POMDPs. The continuous-state Perseus algorithm is shown in Table I. Point-based POMDP algorithms focus on identifying the α-functions (α-vectors in the discrete case) for a set of likely belief points. The α-functions 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 result is an approximation of the value function with less error in regions of the belief space where decisions are more likely to be taken.

The value update scheme of Perseus implements a randomized approximate value function recursion $V_n = \tilde{H} V_{n-1}$ for a set of randomly sampled belief points B. First (Table I, line 2), we let the agent randomly explore the environment and collect a set B of reachable belief points. Next (Table I, lines 3-5), we initialize the value function V_0 as a single weighted Gaussian with large covariance and with weight $\min\{R\}/(1-\gamma)$, with R the set of possible rewards.

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 I, 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-function associated with this point using the `backup` operator (see Section IV-C and line 14 in Table I). If this α-function improves the value of b (i.e., if $\langle \alpha, b \rangle \geq V_n(b)$, line 15), we add α to V_{n+1} (line 18). The hope is that α improves the value of many other points, and all these points are removed from $\tilde{B}$ (line 19). Often, a small number of vectors will be sufficient

Perseus
Input: A continuous state 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: $\alpha \leftarrow \frac{\min\{R\}}{1-\gamma}\, \phi(s|0, \Sigma_\infty)$
4: $n \leftarrow 0$
5: $V_n \leftarrow \{\alpha\}$
6: **do**
7: $\forall b \in B$,
8: $\texttt{Function}_n(b) \leftarrow \arg\max_{\alpha \in V_n} \langle \alpha, b \rangle$
9: $\texttt{Value}_n(b) \leftarrow \langle \texttt{Function}_n(b), b \rangle$
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: $\alpha \leftarrow \texttt{Function}_n(b)$
17: **endif**
18: $V_{n+1} \leftarrow V_{n+1} \cup \{\alpha\}$
19: $\tilde{B} \leftarrow \tilde{B} \setminus \{b' \in \tilde{B} \mid \langle \alpha, b' \rangle \geq \texttt{Value}_n(b')\}$
20: **until** $\tilde{B} = \emptyset$
21: $n \leftarrow n + 1$
22: **until** convergence

TABLE I
THE PERSEUS ALGORITHM. THE backup FUNCTION IS DESCRIBED IN SECTION IV-C.

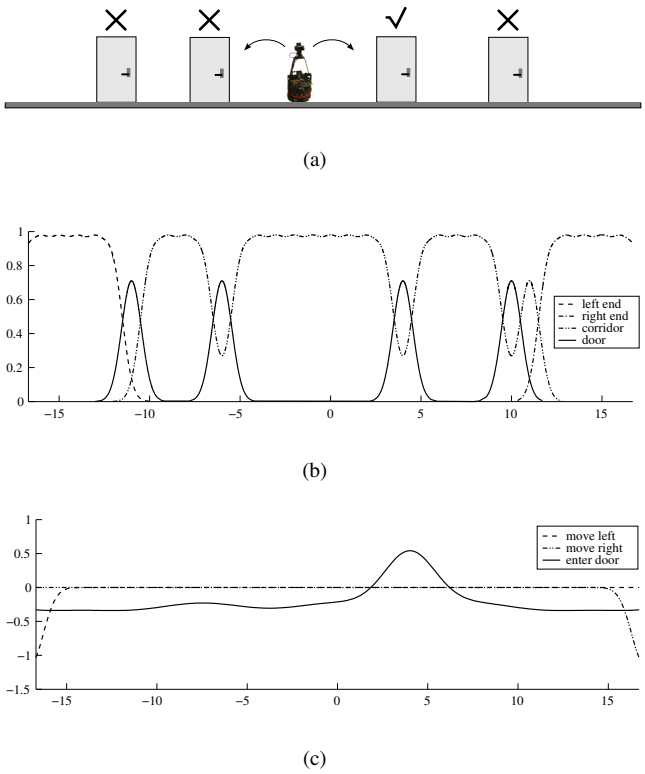

Fig. 1. A pictorial representation of the test problem (a), the corresponding observation model (b) and the reward model (c).

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-functions 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 in Table I), we ignore α and insert a copy of the maximizing function of b from V_n in V_{n+1} (lines 16 and 18). Point b is now considered improved and is removed from $\tilde{B}$, together with any other belief points that have the same function as maximizing one in V_n (line 19). 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.

One point that deserves special consideration when implementing the PERSEUS algorithm is the possible explosion of the number of components in the Gaussian mixtures defining the α-functions for increasing n's and on the number of components in the belief representation when the belief update (see Section IV-B) is repeated for many time steps. The larger the number of components the slower the basic operations of the algorithm. To keep the number of components bounded, we adapted the procedure described in [27] 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.

VI. EXPERIMENTS AND RESULTS

To demonstrate the viability of our method we carried out an experiment in a simulated robotic domain. In this problem (see Fig. 1-a), a robot is moving in a corridor with four doors. The robot can detect when it is in front of a door and when it is at the left or right end of the corridor. In any other situation, the robot just detects that it is in a corridor (see Fig. 1-b). The robot can move 2 units to the left or to the right (with $\Sigma^a = 0.05$) and can try to enter a door at any point (even when not in front of a door). The target for the robot is to locate the second door from the right and to enter it. The robot only gets positive reward when it enters the target door (see Fig. 1-c). When the robot tries to move further than the end of the corridor (either at the right or at the left) or when it tries to enter the door at a wrong position it gets negative reward.

The set of beliefs B used in the PERSEUS algorithm contains 1000 unique belief points. Those belief points are collected using random walks departing from a belief including 4 components that approximate a uniform distribution on the whole corridor. The walks of the robot along the corridor are organized in episodes where the robot executes actions until it tries to enter a door or until it executes 25 (movement) actions.

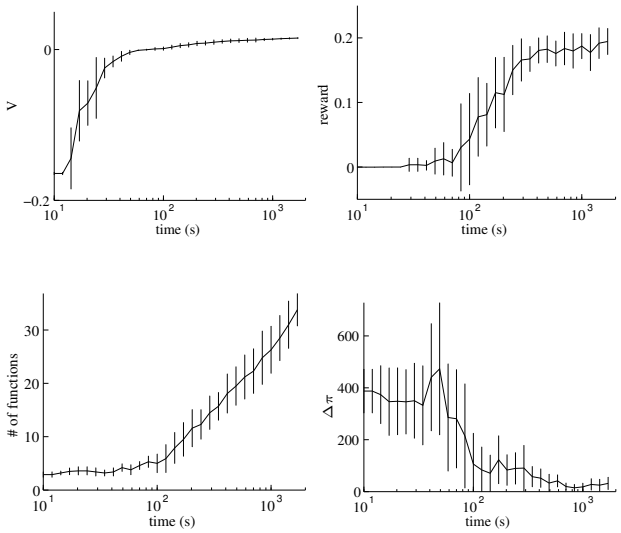

Fig. 2. Top: Evolution of the value for all the beliefs in B and the average accumulated discounted reward for 100 episodes. Bottom: Number of vectors in V_n and the number of policy changes. Results are averaged for 10 repetitions and the bars represent the standard deviation.

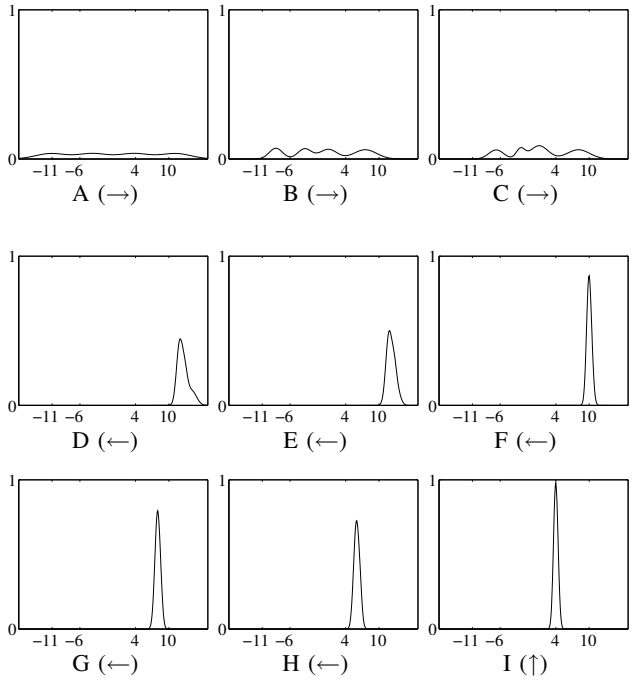

Fig. 3. Evolution of the belief when following the discovered policy. The arrows under the snapshots represent the actions: → for moving right, ← for moving left and ↑ for entering the door. On the x-axis the four door locations are indicated.

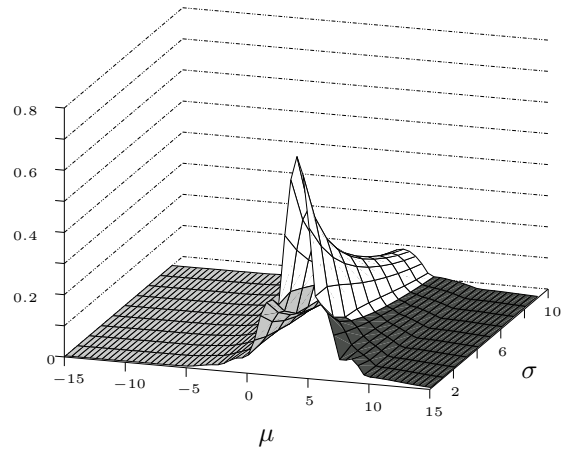

Fig. 4. Value function for single component beliefs as a function of the mean and the covariance.

The experimental setup is completed by setting γ to 0.95, compressing beliefs so that they never contain more than 4 components (i.e., the number of components of the initial belief) and compressing α-functions so that they never have more components than those used to represent the reward function (11 components).

Fig. 2 shows the average results obtained after 10 runs of the PERSEUS algorithm on this problem. The first plot (top-left) shows that the value computed as $\sum_{b \in B} V(b)$ converges. The second plot (top-right) shows the expected discounted reward averaged for 100 episodes with the policy available at the corresponding time slice. The plot indicates that the robot successfully learns to find out its position and to distinguish between the four doors. The 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 is far below 1000, the maximum possible number of α-functions (if we would back up each point in B). In the final plot (bottom-right) we show the number of changes in the policy from one time step to the next one. The changes in the policy are computed as the number of elements in B with a different action from one time slice to the next. The number of policy changes drops to close to zero, indicating convergence with respect to the particular B.

Following the learned policy the robot moves to one of the ends of the corridor to determine its position and then towards the correct door to enter it. The snapshots A to I in Fig. 3 show the evolution of the belief of the robot and the executed action in each case from the initial stage of the episode to the point at which the target door is reached.

In Fig. 4 we plot the value of beliefs that have only one component, parametrized by the mean and the covariance of this component. We can see that, as the uncertainty about the position of the robot grows (i.e., as the covariance is larger) the value of the corresponding belief decreases. The colors/shadings in the figure correspond to the different actions: light-gray for moving to the right, white for entering the door, and dark-gray for moving to the left.

Observe 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. 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, grow as the environment grows.

VII. Conclusions and Future Work

In this paper, we have shown how to generalize value iteration to continuous-state POMDPs and, in particular, for the case of Gaussian-based beliefs and models. This allowed us to define an efficient point-based value iteration algorithm that seems to be appropriate for planning problems that are often encountered in robotics.

An approach to continuous-state POMDPs that is closely related to ours is presented in [19]. In that work, a belief is represented by a set of weighted samples, which can be regarded as a degenerate version of our Gaussian mixture representation. Additionally, the value function is approximated by nearest-neighbor interpolation, whereas in our case the value function achieves generalization through a set of α-functions. Also, in the above work a real-time dynamic programming approach is used for updating the value function, with the Bellman backup operator being approximated by sampling from the belief transition model. In our case, value iteration applies on a pre-collected set of beliefs, while the Bellman backup operator is analytically computed given the particular value function representation. Although we have not directly compared our method to the method presented in [19], we expect our method to be faster (since it plans on a fixed set of belief points) and the value function to generalize better over the belief space (through the use of α-functions).

Ongoing work involves extending our framework to continuous action [26] and observation spaces [28], as well as defining approximate belief representations using Monte Carlo techniques [19].

Acknowledgments

We would like to thank J.J. Verbeek and W. Zajdel for their contributions to the work reported here, and the four reviewers for their detailed comments. J.M. Porta has been partially supported by Ramón y Cajal contract from the Spanish Ministry for Science and Technology. M.T.J. Spaan and N. Vlassis 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. Authors are listed in alphabetical order.

References

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

[2] E. J. Sondik, "The Optimal Control of Partially Observable Markov Processes," Ph.D. dissertation, Stanford University, 1971.

[3] G. E. Monahan, "A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[4] H. T. Cheng, "Algorithms for Partially Observable Markov Decision Processes," Ph.D. dissertation, University of British Columbia, 1988.

[5] 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 the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, 1997.

[6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[7] R. Simmons and S. Koenig, "Probabilistic Robot Navigation in Partially Observable Environments," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[8] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, "Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1996, pp. 963–972.

[9] G. Theocharous and S. Mahadevan, "Approximate Planning with Hierarchical Partially Observable Markov Decision Processes for Robot Navigation," in *IEEE International Conference on Robotics and Automation*, 2002, pp. 1347–1352.

[10] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards Robotic Assistants in Nursing Homes: Challenges and Results," in *Robotics and Autonomous Systems*, vol. 42, no. 3-4, 2003, pp. 271–281.

[11] C. Papadimitriou and J. N. Tsisiklis, "The Complexity of Markov Decision Processes," *Mathematical and Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[12] 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 Sixteenth National Conference on Artificial Intelligence (AAAI)*, 1999, pp. 541–548.

[13] J. Pineau, G. Gordon, and S. Thrun, "Point-based Value Iteration: An Anytime Algorithm for POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[14] N. Vlassis and M. T. J. Spaan, "A Fast Point-Based Algorithm for POMDPs," in *In Proceedings of Annual Machine Learning Conference of Belgium and the Netherlands, Brussels, Belgium*, 2004, pp. 170–176.

[15] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmont, MA: Athena Scientific cop, 2001.

[16] A. Y. Ng and M. Jordan, "PEGASUS: A Policy Search Method for Large MDPs and POMDPs," in *Proceedings of the 16th Conference on Uncertainty in Artificial Inteligence (UAI)*, 2000, pp. 406–415.

[17] D. Aberdeen and J. Baxter, "Scalable Internal-State Policy-Gradient Methods for POMDPs," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2002, pp. 3–10.

[18] N. Roy, G. Gordon, and S. Thrun, "Finding Approximate POMDP Solutions Through Belief Compression," *Journal of Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[19] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems (NIPS)*, S. Solla, T. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1064–1070.

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

[21] M. Hauskrecht, "Value Function Approximations for Partially Observable Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–95, 2000.

[22] J. M. Porta, M. T. J. Spaan, and N. Vlassis, "Value Iteration for Continuous-State POMDPs," IAS Technical Report, University of Amsterdam, Tech. Rep. IAS-UVA-04-04, 2004.

[23] J. J. Leonard and H. F. Durrant-Whyte, "Mobile Robot Localization by Tracking Geometric Beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

[24] P. Jensfelt and S. Kristensen, "Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, 2001.

[25] J. Julier and J. K. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems," in *In Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*, 1997, pp. 1628–1632.

[26] M. T. J. Spaan and N. Vlassis, "Perseus: Randomized Point-based Value Iteration for POMDPs," *Journal of Artificial Intelligence Research*, 2005.

[27] J. Goldberger and S. Roweis, "Hierarchical Clustering of a Mixture Model," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[28] J. Hoey and P. Poupart, "Solving POMDPs with Continuous or Large Discrete Observation Spaces," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.