Planning with Continuous Actions in Partially Observable Environments

Matthijs T. J. Spaan and Nikos Vlassis
Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{*mtjspaan,vlassis*}*@science.uva.nl*

Abstract— **We present a simple randomized POMDP algorithm for planning with continuous actions in partially observable environments. Our algorithm operates on a set of reachable belief points, sampled by letting the robot interact randomly with the environment. We perform value iteration steps, ensuring that in each step the value of all sampled belief points is improved. The idea here is that by sampling actions from a continuous action space we can quickly improve the value of all belief points in the set. We demonstrate the viability of our algorithm on two sets of experiments: one involving an active localization task and one concerning robot navigation in a perceptually aliased office environment.**

I. Introduction

Planning is a central problem in robotics: it involves computing a sequence of actions that accomplish a task as effectively as possible. Classical motion planning [1] for instance computes a series of motor commands which will move a robot to a desired location. In this paper we specifically focus on planning for robots which have a continuous set of actions at their disposal. Example scenarios include navigating to an arbitrary location, or rotating a pan-and-tilt camera at any desired angle. Moreover, in the type of domains we are considering the robot is uncertain about the exact effect of its action; it might for instance end up at a different location than it aimed for, because of errors in the robot motion. Furthermore, in most realistic scenarios the robot cannot determine with full certainty the true state of the environment with a single sensor reading, i.e., the environment is only partially observable to the robot. Not only are its sensors likely to suffer from noise, the robot's environment is often "perceptually aliased". This phenomenon occurs when different parts of the environment appear similar to the sensor system of the robot, but require different actions.

Partially observable Markov decision processes (POMDPs) provide a rich mathematical framework for acting optimally in such partially observable environments [2]. The POMDP defines a sensor model specifying the probability of observing a particular sensor reading in a specific state and a stochastic transition model which captures the uncertain outcome of executing an action. The robot's task is defined by the reward it receives at each time step, and its goal is to maximize the discounted cumulative reward it gathers. Assuming a discrete state space, the POMDP framework allows for capturing all uncertainty introduced by the transition and observation model by defining and operating on the *belief state* of a robot. A belief state is a probability distribution over all states and summarizes all information regarding the past.

Computing optimal planning solutions for POMDPs is an intractable problem for any reasonably sized task [3], [4], calling for approximate solution techniques [5], [6]. A recent line of research on approximate POMDP algorithms focuses on the use of a sampled set of *belief points* on which planning is performed [6], [7], [8], [9], [10], [11]. The idea is that instead of planning over the complete belief space of the robot (which is intractable for large state spaces), planning is carried out only on a limited set of prototype beliefs that have been sampled by letting the robot interact (randomly) with the environment. The computed policy generalizes over the complete belief space via a set of hyperplanes that cause nearby beliefs to share the same policy.

In previous work [10] we developed a simple randomized point-based POMDP algorithm called PERSEUS. We applied it to robotic planning problems featuring large state spaces and high dimensional observations, but with discrete action spaces. In PERSEUS we perform value iteration steps, ensuring that in each step the value of all sampled belief points is improved, by only updating the value and its gradient of a randomly selected subset of points. Extending this scheme, we propose in this paper CA-PERSEUS, an algorithm for planning with continuous actions in partially observable environments, which is particularly relevant in robotics. Obviating the need to discretize a robot's action space allows for more precise control. Most work on POMDP solution techniques targets discrete action spaces. Exceptions include the application of a particle filter to a continuous state and action space [12] and certain policy search methods [13]. We report on experiments in an abstract active localization domain in which a robot can control its range sensors to influence its localization estimate, and on results from a navigation task involving a mobile robot with omnidirectional vision in a perceptually aliased office environment. We will demonstrate that CA-PERSEUS can compute successful policies for these domains.

II. POMDP PLANNING

We start by reviewing briefly the POMDP framework from a robotic point of view, followed by techniques to (approximately) solve POMDPs. The POMDP framework allows one to define a task for a robot with noisy sensors and stochastic actions. At any time step the environment is in a state $s \in S$. A robot inhabiting this environment takes an action $a \in A$ and receives a reward $r(s,a)$ from the environment as a result of this action. The environment switches to a new state s' according to a known stochastic transition model $p(s'|s,a)$. The transition model is often assumed Gaussian when a is a movement action, reflecting errors in the robot motion. The robot then perceives an observation $o \in O$ that depends on its action. This observation provides the robot with information about the state s' through a known stochastic observation model $p(o|s,a)$. The observation model captures the sensory capabilities of the robot. The sets S, O and A are assumed discrete and finite here, but we will generalize to continuous A in Section III.

In order for a robot to choose its actions successfully in such a partially observable environment some form of memory is needed. In belief based POMDP solving we maintain a discrete probability distribution $b(s)$ over states s that summarizes all information about the past. This distribution, or belief, is a Markovian state signal and can be updated using Bayes' rule each time the robot takes an action a and receives an observation o, as follows:

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

where $p(o|a,b) = \sum_{s'\in S} p(o|s',a) \sum_{s\in S} p(s'|s,a)b(s)$ is a normalizing constant.

Given such a belief space, the planning task becomes one of computing an optimal *policy*, a mapping from beliefs to actions that maximizes the expected discounted future reward of the robot $E[\sum_t^\infty \gamma^t r(s_t,a_t)]$, where γ is a discount rate, $0 \le \gamma < 1$. A policy can be defined by a value function, which defines the expected amount of future discounted reward for each belief state. The value function of an optimal policy is called the optimal value function and is denoted by V^*, which satisfies the Bellman optimality equation $V^* = HV^*$, or

$$V^*(b) = \max_{a\in A}\Big[\sum_{s\in S} r(s,a)b(s) + \gamma \sum_{o\in O} p(o|a,b)V^*(b_a^o)\Big], \tag{2}$$

with b_a^o given by (1), and H is the Bellman backup operator [14]. When (2) holds for every b in the belief space we are ensured the solution is optimal.

A. Value iteration in POMDPs

A classical method for solving POMDPs is value iteration. This method iteratively builds better estimates of V^* by applying the operator H to an initially piecewise linear and convex value function V_0 [15]. The intermediate estimates $V_1, V_2, \ldots$ will then also be piecewise linear and convex. We will throughout assume that a value function V_n at step n is represented by a finite set of vectors (hyperplanes) $\{\alpha_n^1, \alpha_n^2, \ldots\}$. Additionally, with each vector an action $a(\alpha_n^i) \in A$ is associated, which is the optimal one to take in the current step, assuming optimal actions are executed in following steps. Given a set of vectors $\{\alpha_n^i\}_{i=1}^{|V_n|}$ in V_n, the value of a belief b is given by

$$V_n(b) = \max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i, \tag{3}$$

where $(\cdot)$ denotes inner product. The gradient of the value function at b is given by the vector $\alpha_n^b = \arg\max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i$, and the policy at b is given by $\pi(b) = a(\alpha_n^b)$.

The main idea behind many value iteration algorithms for POMDPs is that for a given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg\max_{\{\alpha_{n+1}^i\}_i} b \cdot \alpha_{n+1}^i \tag{4}$$

where $\{\alpha_{n+1}^i\}_i$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha = \mathtt{backup}(b)$. It computes the optimal vector for a given belief b by back-projecting all vectors in the current horizon value function one step from the future and returning the vector that maximizes the value of b. In particular, defining $r_a(s) = r(s,a)$, it is straightforward to show that combining (1), (2), and (3) gives [11]:

$$V_{n+1}(b) = \max_a \Big[b \cdot r_a + \gamma \sum_o \max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i\Big], \text{with} \tag{5}$$

$$g_{a,o}^i(s) = \sum_{s'} p(o|s',a)p(s'|s,a)\alpha_n^i(s'). \tag{6}$$

Applying the identity $\max_j b\cdot\alpha_j = b\cdot\arg\max_j b\cdot\alpha_j$ in (5) twice, we can compute the vector $\mathtt{backup}(b)$ as follows:

$$\mathtt{backup}(b) = \arg\max_{\{g_a^b\}_{a\in A}} b \cdot g_a^b, \quad \text{with} \tag{7}$$

$$g_a^b = r_a + \gamma \sum_o \arg\max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i. \tag{8}$$

Note that the backup operator in (7) requires enumerating all actions, which is not feasible in practice when the action space is continuous.

Although computing the vector $\mathtt{backup}(b)$ for a given b is straightforward, locating all vectors $\cup_b \mathtt{backup}(b)$ of HV_n typically requires linear programming and is therefore costly in high dimensions [2]. A promising way to sidestep this issue is to sample in advance a set of belief points from the belief simplex, and then perform value updates on these points only [5], [6], [7], [9], [10], [11]. Point-based solution techniques are justified by the fact that in most robotic problem settings the belief simplex is sparse, in the sense that only a limited number of belief points can ever be reached when the robot directly interacts with the environment. In these cases, one would like to plan only for those reachable beliefs instead of planning over the complete belief simplex.

In [10] we described PERSEUS, a simple approximate point-based algorithm for solving POMDPs which we have applied to robotic planning. The key idea is that, in each value iteration step, we can improve the value of all points in the belief set by only updating the value and its gradient of a randomly selected subset of the points. We performed experiments in benchmark problems from literature, and PERSEUS turns out to be very competitive to state-of-the-art methods in terms of solution quality and computation time. We show here that this scheme of 'partial' value updates is also very well suited for handling problems with continuous action spaces.

III. PLANNING WITH CONTINUOUS ACTIONS

In [10] our PERSEUS POMDP algorithm assumed discrete action sets. Here we extend PERSEUS to handle problems with continuous action spaces. Instead of considering a finite and discrete action set A we parameterize the robot's actions on a set of k, problem-specific parameters $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$. These parameters are real valued and can for instance denote the angle by which the robot rotates. Computing a policy containing such actions requires modifying the `backup` operator defined in Section II-A, since A now contains an infinite numbers of actions (and therefore maximization over these is not straightforward). The idea here is that instead of maximizing over all $a \in A$, we sample actions at random from A, compute their backed up vectors, and check whether one of the resulting vectors improves the value of the corresponding belief point. The `backup` operator as defined in (7) is replaced by a backup operator $\alpha = \texttt{backup}'(b)$:

$$\texttt{backup}'(b) = \arg\max_{\{g_a^b\}_{a \in A_b'}} b \cdot g_a^b, \quad \text{with} \tag{9}$$

$$A_b' = \{a_i : a_i \text{ is drawn from } A\}, \tag{10}$$

and g_a^b as defined in (8). We draw at random a set A_b' from the continuous set A, in particular specific θ vectors which define actions and which in turn define the g_a^b vectors. We can easily incorporate such a backup operator in PERSEUS as we will show next.

We first let the robot randomly explore the environment and collect a set B of reachable belief points. Our CA-PERSEUS algorithm performs a number of backup stages. In each backup stage, given a value function V_n, we compute a value function V_{n+1} that improves the value of all $b \in B$. Often, a small number of vectors will be sufficient to improve $V_n(b)\ \forall b \in B$ (especially in the first steps of value iteration), and we compute these vectors in a randomized greedy manner by sampling from B. We initialize the value function V_0 as a single vector with all its components equal to $\frac{1}{1-\gamma} \min_{s,a} r(s,a)$ [16]. Starting with V_0, we perform a number of value function update stages until convergence (which is guaranteed [11], [7]). In particular, given V_n, a backup stage is as follows:

```
CA-PERSEUS backup stage
1) Set $V_{n+1} = \emptyset$. Initialize $\tilde{B} = B$.
2) Sample a belief point $b$ uniformly at random from $\tilde{B}$
   and compute $\alpha = \texttt{backup}'(b)$.
3) If $b \cdot \alpha \geq V_n(b)$ then add $\alpha$ to $V_{n+1}$, otherwise add
   $\alpha' = \arg\max_{\alpha \in V_n} b \cdot \alpha$ to $V_{n+1}$.
4) Compute $\tilde{B} = \{b \in B : V_{n+1}(b) < V_n(b)\}$. If
   $\tilde{B} = \emptyset$ then stop, otherwise go to 2.
```

The algorithm tries in each backup stage to improve the value of all points in B (step 4). If some points are not improved yet (set $\tilde{B}$), one of them is selected at random and a corresponding vector is generated (step 2). If the vector improves the value of the selected point b we add it to V_{n+1} and update $V_{n+1}(b)$ for all $b \in B$. If not, we ignore the vector and insert a copy of the maximizing vector of b from V_n in V_{n+1}. Adding a vector improves the value of hopefully many non-improved points, and the process continues until all points are improved. Essentially our value update scheme defines an *approximate* Bellman backup operator that improves (instead of maximizes) the value of all belief points in each iteration, and turns out to be very successful in practice. It is exactly this "improve-only" principle that justifies the use of the randomized operator `backup'` (9) in step 2 of the algorithm instead of the deterministic operator `backup` (7).

An alternative to sampling for handling continuous action spaces is to discretize the action space. A computational advantage of reducing the action space to a set of discrete actions is the fact that when A is small enough one can cache in advance the explicit tabular representation of the transition, observation and reward models for all $a \in A$. In contrast, when we sample a real-valued action we have to generate from a parametric description of these models their tabular representation, necessary for computing (6). However, discretization has its limits, particularly when considering scalability. The number of discrete actions grows exponentially with k, the number of dimensions of θ. For instance, consider a robotic arm with a large number of joints or, as in one of the experiments below, a robot which can control a number of sensors at the same time: discretization would require a number of bins that is exponential in the number of joints or sensors, respectively. Furthermore, the discretization can lead to worse control performance, as demonstrated in the second set of experiments. Clearly, working directly with continuous actions allows for more precise control.

IV. EXPERIMENTS

We applied our algorithm in two domains: an abstract active localization domain in which a robot can control its

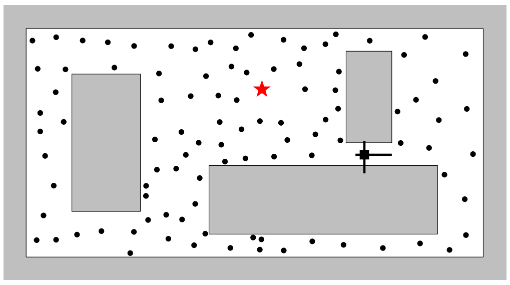


Fig. 1. Environment of the ALH problem. The dots indicate the states, the star depicts the goal state and the black square represents the robot. The four lines show the range of its sensors when they are set to $\theta_{n,e,s,w} = \{0.61, 1.12, 0.81, 0.39\}$.

range sensors to influence its localization estimate and a navigation task involving a mobile robot with omnidirectional vision in a perceptually aliased office environment.

A. Active localization

We first tested our approach on a navigation task in a simulated environment. The Active Localization Hallway (ALH) environment represents a 20×10 m hallway which is highly perceptually aliased (see Fig. 1). The robot inhabiting the hallway is equipped with four range sensors, each observing one compass direction. The robot can set the range of each sensor, up to a certain limit. We assume a sensor can only detect whether there is a wall within its range or not (but with perfect certainty), resulting in a total number of 16 possible observations. The task is to reach a goal location located in an open area where there are no walls near enough for the robot to detect. We would like the robot also to take into account its energy consumption. Moving as well as using the sensor above its default range requires energy and is penalized. The robot is initialized at a random state in the hallway with no knowledge regarding its location, i.e., its belief is uniform. By moving through the hallway and adjusting its sensors at each step the robot receives information indicating its location. The better it controls the range of its sensors, the more accurate it can localize itself and the easier it is to find a path to the goal. Thus, the robot should not only learn what movement actions to take in order to reach the goal, but also how to set its sensors.

The robot's actions are defined by the parameters $\theta = \{\theta_m, \theta_n, \theta_e, \theta_s, \theta_w\}$. At each time step the robot has to set θ_m to one out of four basic motion commands $\{$*north, east, south, west*$\}$ which transports it according to a Gaussian distribution centered on the expected resulting position (translated one meter in the corresponding direction). It sets the range of each its sensors $\{\theta_n, \theta_e, \theta_s, \theta_w\}$ to a real value on the interval $[0, 2]$m. We assume that setting a sensor's range higher than its default range of 0.5m costs energy and we penalize with a reward of -0.01 per meter, resulting in a reward of -0.06 if all sensors are fired at maximum range. Each movement is also penalized, with a reward of -0.12 per step. The reward obtainable at the goal location is 10. As our algorithm assumes a finite and discrete set S we need to discretize the state space (and consequently the transition model), which is defined as the robot's location. For discretizing the positions in the map of the environment we performed a straightforward k-means clustering on a random subset of all possible positions, resulting in a grid of 100 positions, depicted in Fig. 1.

To test the feasibility of our algorithm, i.e., whether it can compute successful policies by sampling actions at random, we ran it with several different action sampling strategies. At each backup a single action is sampled uniformly at random ($|A'_b| = 1$ in (10)). The sampling strategy determines from what range each parameter in θ is sampled, and here it defines how many sensors the robot can control. Strategy 0 restricts the robot to only setting θ_m, with $\theta_{n,e,s,w}$ fixed at the default range, while strategy 4 allows full control of all sensors. Table I summarizes the five strategies we tested. Note that strategy 0 in fact reduced the action space to a discrete set of four actions. For each strategy a belief set B of $10,000$ points was gathered by simulating a random walk of the robot through the hallway. We ran our algorithm 5 times (with different random seeds) for each strategy and the plots are averaged over these five runs. To evaluate the computed value function estimates we collected rewards by sampling 10 trajectories from 100

TABLE I

ACTION SAMPLING STRATEGIES TESTED IN THE ALH DOMAIN.

strategy	θ_m	θ_n	θ_e	θ_s	θ_w
0	$\{n, e, s, w\}$	0.5	0.5	0.5	0.5
1	$\{n, e, s, w\}$	$[0, 2]$	0.5	0.5	0.5
2	$\{n, e, s, w\}$	$[0, 2]$	$[0, 2]$	0.5	0.5
3	$\{n, e, s, w\}$	$[0, 2]$	$[0, 2]$	$[0, 2]$	0.5
4	$\{n, e, s, w\}$	$[0, 2]$	$[0, 2]$	$[0, 2]$	$[0, 2]$

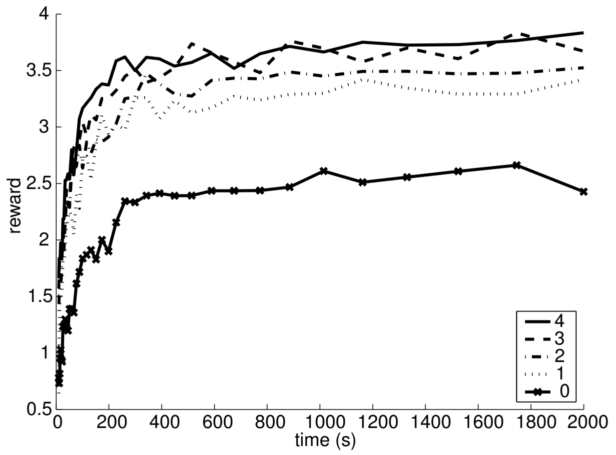


Fig. 2. Performance in ALH domain for 5 different strategies, averaged over 5 runs. The y-axis depicts the expected discounted reward (estimated by sampling $1,000$ trajectories) and the x-axis indicates CPU time used by CA-PERSEUS in seconds.

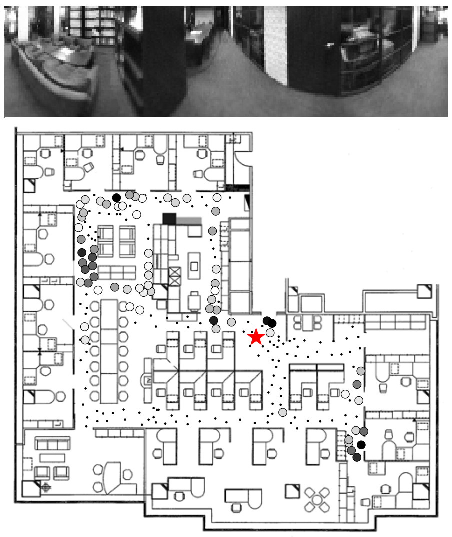


Fig. 3. cTRC Problem: Panoramic image corresponding to a prototype feature vector $o_k \in O$, and below its induced $p(s|o_k)$. The darker the dot, the higher the probability.

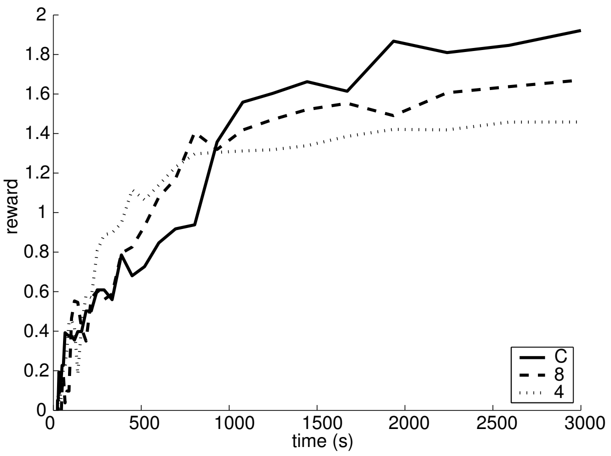
random starting locations. In our experiments we used a discount factor $\gamma = 0.95$ and each trajectory was stopped after a maximum of 100 steps (if the robot had not reached the goal by then).

Fig. 2 shows the expected discounted cumulative reward for each of the strategies. We see that allowing the robot to control more sensors improves its performance. The algorithm does not seem to be hampered by the increased dimensionality of the action space, as it computes better policies in the same amount of time (using roughly the same amount of vectors). It learns that the advantage of a more accurate localization outweighs the cost of increasing the range of its sensors. We can see that the discrete strategy 0 performs poorly, because of its limited range of sight, even though we can cache the tabular representation of its transition, observation and reward models. Visual inspection of the computed policies suggests that the robot learns that repeated switching between long and short range settings allows it to more accurately determine the distance to the corresponding wall.

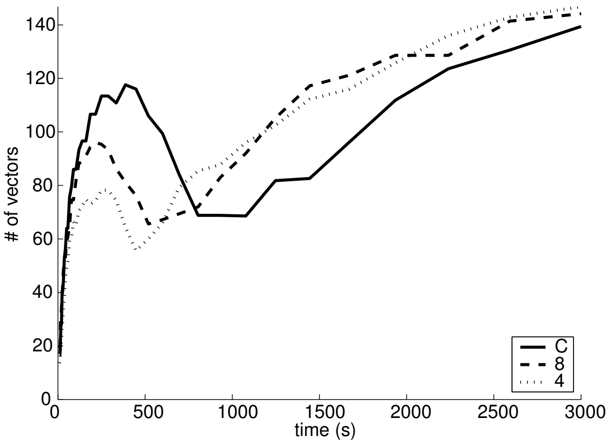
Depending on the problem to be solved, more informed schemes could be devised than sampling only a single action uniformly at random at each backup. One option would be to expand A'_b to include actions sampled from a Gaussian distribution centered on the best known action so far (for the particular belief b). Preliminary results suggest that such a sampling scheme improves the control quality in the ALH domain.

B. Arbitrary heading navigation

To evaluate our algorithm on a more realistic problem and compare against discretized action sampling we also include the cTRC domain. In this problem a mobile robot with omnidirectional vision has to navigate a highly



(a) Reward.



(b) Number of vectors.

Fig. 4. Performance in cTRC domain, averaged over 5 runs. Top figure depicts the expected discounted reward and the bottom shows the number of vectors in the value function. In both figures the x-axis indicates CPU time used by CA-PERSEUS in seconds.

perceptually aliased office environment (see Fig. 3). It is a variation of the TRC problem introduced in [10], but with a continuous action space. The robot can decide to move 5 meters in an arbitrary direction, i.e., its actions are parameterized by $\theta = \theta_\alpha$ ranging on $[0, 2\pi]$. We assume a Gaussian error on the resulting position.

For our observation model we used the MEMORABLE[1] robot database that contains a set of approximately 8000 panoramic images collected manually by driving the robot around in a 17×17 meters office environment. As in [10], we compressed the images with Principal Component Analysis and applied k-means clustering to create 10 three-dimensional prototype feature vectors $\{o_1, \ldots, o_{10}\}$. Fig. 3 shows the inverse of the observation model $p(o|s)$ for one

[1]The MEMORABLE database has been provided by the Tsukuba Research Center in Japan, for the Real World Computing project.

observation, together with the image in the database closest to this particular prototype observation. We used the same technique as in the ALH domain to grid our state space in 200 states. The task is to reach a certain goal state at which a reward of 10 can be obtained; each action yields a reward of -0.12. We used belief sets of $10,000$ points, $\gamma = 0.95$ and other parameters are the same as in ALH.

We compared CA-PERSEUS to two discretized versions of this problem, in which we allowed the robot to sample actions from a set of 4 or 8 headings with equal separation (offset with a fixed random angle to prevent any bias resulting from the orientation of the corridors in the map). Fig. 4 displays results for our algorithm, sampling a continuous A ("C") and the two discretized A ("4" and "8"). In particular, Fig. 4(a) shows the superior control quality of the continuous A, accumulating more reward than the discrete cases. Even after 3000s strategy 4 does not reach the goal in 100% of the cases, while the other two strategies do. However, CA-PERSEUS when employing strategy C exploits its ability to move in an arbitrary angle to find a better policy than both discrete cases. Fig. 4(b) plots the number of vectors in the value function for each strategy. Typically this number grows with time, reflecting the need for a more complex plan representation when the planning horizon increases. Interestingly, this figure shows that our algorithm can discover that after a certain planning horizon, a smaller number of vectors suffice to improve all points in the belief set.

V. DISCUSSION AND CONCLUSIONS

We presented CA-PERSEUS, a simple randomized POMDP algorithm for planning with continuous actions in partially observable environments. CA-PERSEUS operates on a set of reachable belief points, sampled by letting the robot interact randomly with the environment. We perform value iteration steps, ensuring that in each step the value of all sampled belief points is improved by only updating the value (and its gradient) of a random subset of the points. The key point of our algorithm is that we can sample actions from a continuous action space and use them to improve the value of all beliefs in a belief set.

Relatively little work has been done on the topic of planning with continuous actions in POMDPs. Policy search methods like [13] and particle filters [12] have been applied to POMDP domains with a continuous state and action space. As the continuous state space precludes the computation of a traditional belief state, many nice properties (e.g., known shape of the value function) are lost. While we acknowledge the importance of planning over continuous state spaces, in this work we chose to discretize our state space. This allows us to take advantage of the successful point-based belief solution machinery. It could be a subject of further study how Monte Carlo techniques can be combined with (point-based) value iteration, or how to generalize value iteration to continuous state POMDPs.

We evaluated our algorithm on two sets of experiments: one involving an active localization task and the other set involved a robot navigating a perceptually aliased office environment. We demonstrated that our algorithm can compute successful policies for these domains while sampling from a continuous set of actions. By obviating the need to discretize the robot's action space more precise control can be achieved. To our knowledge, CA-PERSEUS is the first (point-based) value iteration algorithm that can directly handle continuous actions in POMDPs.

Acknowledgments

This research is 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.

REFERENCES

[1] J. Latombe, *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[3] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[4] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems," in *Proc. 16th National Conf. on Artificial Intelligence*, Orlando, Florida, July 1999.

[5] W. S. Lovejoy, "Computationally feasible bounds for partially observed Markov decision processes," *Operations Research*, vol. 39, no. 1, pp. 162–175, 1991.

[6] M. Hauskrecht, "Value function approximations for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–95, 2000.

[7] K.-M. Poon, "A fast heuristic algorithm for decision-theoretic planning," Master's thesis, The Hong-Kong University of Science and Technology, 2001.

[8] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Advances in Neural Information Processing Systems 15.* Cambridge, MA: MIT Press, 2003.

[9] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. Int. Joint Conf. on Artificial Intelligence*, Acapulco, Mexico, Aug. 2003.

[10] M. T. J. Spaan and N. Vlassis, "A point-based POMDP algorithm for robot planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, Louisiana, 2004, pp. 2399–2404.

[11] ——, "Perseus: randomized point-based value iteration for POMDPs," Informatics Institute, University of Amsterdam, Tech. Rep. IAS-UVA-04-02, Nov. 2004.

[12] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems 12*, S. Solla, T. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1064–1070.

[13] A. Y. Ng and M. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proc. of Uncertainty in Artificial Intelligence*, 2000.

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

[15] E. J. Sondik, "The optimal control of partially observable Markov decision processes," Ph.D. dissertation, Stanford University, 1971.

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