To appear in Proceedings of the AAAI Fall Symposium, November 2013

Combining a POMDP Abstraction with Replanning to Solve Complex, Position-Dependent Sensing Tasks

Devin K. Grady and **Mark Moll** and **Lydia E. Kavraki**
Dept. of Computer Science, Rice University
Houston, TX 77005, USA
{devin.grady, mmoll, kavraki}@rice.edu

Abstract

The Partially-Observable Markov Decision Process (POMDP) is a general framework to determine reward-maximizing action policies under noisy action and sensing conditions. However, determining an optimal policy for POMDPs is often intractable for robotic tasks due to the PSPACE-complete nature of the computation required. Several recent solvers have been introduced that expand the size of problems that can be considered. Although these POMDP solvers can respect complex motion constraints in theory, we show that the computational cost does not provide a benefit in the eventual online execution, compared to our alternative approach that relies on a policy that ignores some of the motion constraints. We advocate using the POMDP framework where it is critical – to find a policy that provides the optimal action given all past noisy sensor observations, while abstracting some of the motion constraints to reduce solution time. However, the actions of an abstract robot are generally not executable under its true motion constraints. The problem is addressed offline with a less-constrained POMDP, and navigation under the full system constraints is handled online with replanning. We empirically demonstrate that the policy generated using this abstracted motion model is faster to compute and achieves similar or higher reward than addressing the motion constraints for a car-like robot as used in our experiments directly in the POMDP.

Introduction

POMDPs are a structured representation of a planning problem where, at each discrete time point, an agent performs an action and obtains an observation through sensing. The challenge lies in addressing noise in both action and sensing. This creates uncertainty in the state of the robot and in the outcome of actions. The solution of a POMDP is an action policy for an agent that maximizes an arbitrary reward function. The POMDP encapsulates a well-defined noise model for action and sensing. Finding an optimal policy for a POMDP ("solving" the POMDP) can be thought of as a search in a continuous belief space of dimension equal to the number of states in the underlying state space. Solving POMDPs is PSPACE-complete (Papadimitriou and Tsitsiklis 1987) due to the curse of history and the curse of dimensionality. Although solving a POMDP is computationally challenging, POMDPs are useful because all computation is offline, and a globally optimal policy is computed. To describe the class of robotic tasks we would like to solve, a reward function is constructed such that an optimal policy to increase reward over time will solve the task optimally.

The computational burden prevents solving large problem instances in reasonable time or space. Recent belief point-based solvers (e.g., Smith and Simmons 2005; Kurniawati, Hsu, and Lee 2008) have extended the size of solvable POMDP instances such that they are useful for some robotic tasks involving action and sensing noise. The state spaces considered in these solvers tend to be on the order of 10×10 grids with some uncertain world features (e.g., Lee and Kim 2013; Hsiao, Kaelbling, and Lozano-Perez 2007); this a small state space.

This paper does not present a new POMDP solver, instead it proposes an *abstraction* of a POMDP, which is combined with online planning to address the simplifications made in the abstraction. We demonstrate problem instances in which the POMDP model of a car-like robot can not be solved with modern techniques, but a solution to the abstraction can be found. Online planning does not incorporate the observations from sensing in a globally optimal way as the POMDP policy does. Combining a POMDP abstraction with online planning yields a best of both worlds solution in our experiments.

This paper considers the task of navigating a car-like robot with a partially-known map and noisy sensors. We propose that the natural level of abstraction for this task is a holonomic robot operating in $\mathbb{R}^2$. This abstraction reduces the computational burden by ignoring the complex motion constraints. Intuitively, this abstraction is effective because it attacks the computational complexity by reducing the state space, and the curse of history is eased by using simpler constraints. Although the problems we consider have a similarly sized state space as prior work, we show that added motion constraints can lead to unsolvable POMDPs. Our experimental results indicate that our POMDP abstraction with replanning is an effective technique. Executing a policy constructed in an abstract motion model is not trivial, and is discussed in the Methodology section. Details connecting the theoretical method with our evaluation platform appear in Experimental Setup. Our evaluation shows that, counter-intuitively, incorporating the true constraints takes more time, but may also generate hard to execute policies, reducing reward even when given time to converge to an optimal policy.

Problem Definition

The task we set out to solve is an escape problem, where a robotic agent must avoid detection. This robot has a radio signal strength sensor that estimates the distance to a radio source emanating from an adversarial detection outpost. The problem is to determine a sensor-based policy that a car-like robot can follow without detection to a goal region. There are some existing, known obstacles in the workspace, as well as one possible unknown obstacle. The range to the unknown obstacle can be sensed, however the sensor is noisy and noise increases with range. Additionally, only noisy position sensing is available, preventing the agent from ever fully localizing.

Our problem is similar in construction to RockSample, introduced on 7×7 grids (Smith and Simmons 2004), since expanded and modified (Zhang and Chen 2010; Bai et al. 2010). Similar to RockSample, the environment is defined as a grid, and there exists an exit region that is rewarding and terminal. Unlike RockSample, costly and terminal obstacles exist. RockSample, unlike our problem, has perfect localization. In addition, our variables for sensing the environment are not boolean values of rocks with known positions but, instead, are range measurements to a terminal obstacle with unknown position. Overall, these modifications make this new problem much more difficult.

The POMDP representing this task is defined as $P_0 = \langle S, A_0, O_0, T_0, \Omega_0, R, \gamma \rangle$, and is the input to our algorithm.

$$
\begin{aligned}
S &= \text{Continuous State Space} \\
A_0 &= \text{Continuous Action Space} \\
O_0 &= \text{Continuous Observation Space} \\
T_0 &= \text{Transition Probabilities } (S \times A_0 \times S) \Rightarrow \mathbb{R} \\
\Omega_0 &= \text{Observation Probabilities } (S \times A_0 \times O_0) \Rightarrow \mathbb{R} \\
R &= \text{Reward Function} (S \times A) \Rightarrow \mathbb{R} \\
\gamma &= \text{Discount Factor } (0.95)
\end{aligned}
$$

The output is a sequence of states (execution trace) that follow the propagation of a simulated car-like robot. An execution trace has a reward calculated using R and γ from above. We define success as when a trace ends in a terminal exit region, and failure is all other traces.

Overall Approach

P_0 is useful to describe the underlying continuous problem, however, continuous action spaces have only recently been addressed using techniques from sampling-based planning to find useful action samples (Kurniawati, Bandyopadhyay, and Patrikalakis 2012). We directly define a solvable POMDP abstraction as $P = \langle S, A, O, T, \Omega, R, \gamma \rangle$ where A and O are discretized approximations of A_0 and O_0. T and Ω are then functions over these discretized spaces. Modern solvers, such as Monte-Carlo Value Iteration (MCVI) (Lim, Hsu, and Lee 2011), can solve a POMDP in this form.

The stealth escape problem will be solved in two stages:

1. Offline POMDP policy generation, and
2. Online replanning agent policy execution.

To represent a car-like model, the state space S is the continuous space of rigid body poses in the plane, denoted as $SE(2)$. Action space A will be discretized for computational efficiency into 5 actions as shown in Figure 2. Sensor observations consist of $(x, y, range)$ tuples, discretized to integral values. T will be a Gaussian distribution centered around the expected outcome from the motion model. Ω will implement a similar Gaussian noise model for sensing. The current orientation, θ, is noise-free. We will define the reward function R to be independent of A and only depend on S. The discount factor, γ, encourages short policies as in all known prior finite-time horizon POMDP planning.

As we will show, computing even an approximate policy for P is intractable, despite the discretization of sensing and action. Therefore, this paper proposes an abstracted POMDP model, $P' = \langle S', A', O, T', \Omega, R, \gamma \rangle$. Although the motion model has changed (S', A', T'), the components of the task description that will not be addressed with replanning are unchanged (O, Ω, R, γ). In P', $S' = \mathbb{R}^2$, $A' = \{\text{North}, \text{South}, \text{East}, \text{West}, \text{Stop}\}$, and T' is the same Gaussian, but centered on the state returned by this simpler motion model.

For comparison purposes, we solve both P and P' using the powerful solver MCVI, and the policies generated will be used in online execution. The task is complete, either in success or failure, when the policy update of the online execution indicates that the robot has entered a terminal state.

Related Work

The two-phase method we propose may appear similar to hierarchical POMDP methods (Foka and Trahanias 2007; Pineau, Roy, and Thrun 2001; Pineau et al. 2003). However, these methods generate POMDP policies "all the way down" to the full system detail through decomposition of the state, action, and sensor spaces. Our proposal instead uses an abstracted POMDP, and then applies replanning to incorporate the constraints.

Point-Based Policy Transformation (PBPT) (Kurniawati and Patrikalakis 2012) and online POMDP methods (Wang and Dearden 2012; Ross et al. 2008) propose a similar idea to this paper, by limiting offline computation and then fixing the policy when necessary to decrease the required offline computation time. However, we are breaking out of the POMDP framework entirely and not changing the input policy. In our work, the input POMDP policy is for an abstracted model, therefore replanning is necessary to execute the policy. PBPT uses a bijection $S \Leftrightarrow S'$ and $A \Leftrightarrow A'$. We will be operating in state spaces that differ in dimensionality, and our action spaces have very different constraints, precluding the applicability of PBPT. These changes in state and action spaces are precisely what drives the computational benefits of our proposed framework.

There has been significant interest recently in problems where a belief can be approximated by a Gaussian distribution (Bai, Hsu, and Lee 2013; Van den Berg, Patil, and Alterovitz 2012). Although we use Gaussian distributions in our sensor model, the belief state of the world cannot effectively be collapsed into a unimodal distribution due to the uncertainty

of the location of obstacles. Therefore, these extremely fast and effective techniques are not applicable to our problem.

In (Candido and Hutchinson 2011), a method similar to MCVI is proposed for an explicit belief variance minimization and navigation task. Local policies are explicitly user-designed and given as input. These local policies are designed to decrease uncertainty in the current belief and drive the system to a goal state. This will allow the use of high-level abstract actions similar to our method, however, our technique builds per-action policies online instead of using user defined policies, and we use MCVI, which converges to optimal policies.

Recent efforts to improve POMDP solution times by decreasing the number of states (Agha-mohammadi, Chakravorty, and Amato 2011; Kurniawati, Bandyopadhyay, and Patrikalakis 2012; Kaplow, Atrash, and Pineau 2010) and actions (Grady, Moll, and Kavraki 2013) are also relevant to our goal of decreased computation time without sacrificing reward. In general, these methods will produce abstractions of navigation and/or carve out "interesting" areas of the spaces to focus on. Unlike in this paper, the discretizations are altered while the underlying motion constraints and state spaces are unchanged.

Methodology

The proposed algorithm, Algorithm 1: POMDP+Online, operates in two phases. Line 1 is the entire offline solution phase, taking the input POMDP model and getting an optimal policy. Lines 2–4 set up the initial state of the replanning system. Line 5 initializes the output of the execution trace. Lines 6–12 perform the replanning loop until a terminal observation is sensed, appending to the execution trace after each action. Finally, on line 13, the execution trace is returned as output.. The policy is represented as a deterministic policy graph, where each node is the optimal action and edges are taken dependent on observations. The policy node update on line 11 thus depends on the observation (from line 10) This observation may be critical to the overall reward, as noisy sensing plays a pivotal role in the task. Therefore, the execution of each action is planned independently (line 7). This ensures that policy node updates are done correctly and the agent follows the reward-maximizing policy from line 1.

Algorithm 1 POMDP+Online

Require: *P* is a POMDP model
1: *policy* ← Solve(P) ▷ Offline Planning
2: *state* ← SampleInitState(P) ▷ Initialization
3: *policyNode* ← *policy*.root(P)
4: *obs* ← SampleObs(*state*, *policyNode*, P)
5: *trace* ← {*state*}
6: **repeat** ▷ Online replanning loop
7: *path* ← Plan(*obs*.state(), *policy*, *policyNode*, P)
8: *state* ← *state*.IntegrateAlong($path$)
9: *trace*.append(*state*)
10: *obs* ← SampleObs(*state*, *policyNode*, P)
11: *policyNode* ← *policyNode*.child(*obs*)
12: **until** TerminalObservation(*obs*)
13: **return** *trace*

POMDP+Online is a high-level view, and wraps up a several important details in the functions called Solve(), Plan(), and SampleObs(). Many of these implementation details will be covered in the Methodology and Experimental Setup sections. Critical to our combination of an offline policy and online replanning is having the correct goal state for Plan(), and we will expand that function here.

Algorithm 2 Plan() from line 7 of POMDP+Online

Require: *start* is a valid continuous state for *planner*
Require: *policyNode* is a node in *policy*
Require: *P* is a POMDP model
1: *maxReward* ← P.minReward()
2: *bestFuture* ← NULL
3: **for** $i = 1 \dots N$ **do**
4: *future*, *reward* ← Simulate(*policyNode*, P)
5: **if** *reward* $\geq maxReward$ **then**
6: *bestFuture* ← *future*
7: *maxReward* ← *reward*
8: **end if**
9: **end for**
10: *goal* ← GoalSelect(*bestFuture*)
11: *path* ← *planner*.plan(*start*, *goal*)
12: **return** *path*

A brief walkthrough of Plan() starts at the initialization on lines 1–2. The loop on lines 3–9 finds the best possible future trajectory over N simulations. At the end of this loop, *bestFuture* is the set of states along this best possible future trajectory. This is used on line 10 to access *bestFuture*$[1]$, the next state from the best future. Line 10 selects a goal – when P and *planner* have different state spaces this is non-trivial and will be discussed later. Finally, on line 11 the planner is actually called to solve the motion planning problem, and the path found is returned on line 12.

The call to *planner*.plan on line 10 of Algorithm 2 must generate collision-free trajectories that respect the system constraints. The online system is also computationally constrained for each cycle. During online execution, the same noise model is applied as in the offline stage. Due to the computation budget and noise, even simple problems and environments fail occasionally. This occurs when Plan() returns an empty path because the planner could not connect the start and goal states given the constraints of the system.

Experimental Setup

In this section, we will first give the details of the offline models, P and P'. We will proceed to provide the details of our implementation of online replanning. These details will give the algorithms presented earlier some concrete structure.

Both P and P' share a common noise model for sensing, shown in Figure 1. Position sensing noise is applied by drawing a Gaussian sample around the true state, with $\sigma = 0.2$, and then only the integer part is returned. The range sensor observations follow the same general noise model, except that σ is multiplied by the current true range. Therefore, the range sensor samples from a discretized Gaussian centered around the true range to the unknown obstacle. Because our

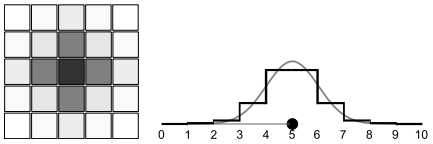

Figure 1: Position (left) and range (right) sensing noise models are Gaussian distributions truncated to integral values.

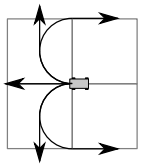

Figure 2: The action model of P.

sensing model does not depend on orientation, P' still captures the essential characteristics of our task, but under fewer motion constraints.

For offline planning, MCVI is used. MCVI is a limited time horizon offline planner. We set the horizon to 100 actions. For our problems, the policies produced by MCVI always terminated significantly before this cutoff.

The action space A is shown in Figure 2. A is a discretized-action first-order robot with bounded curvature. This robot model is a discretized version of the online system that will be discussed later in this section. A does not allow the system to back up since, in the task we are solving, reverse movement is of little use at the POMDP level. A' consists of the 4 cardinal directions with unit length, plus staying still.

The reward function R is assumed constant over unit squares of $\mathbb{R}^2$, and R is exactly the definition of the environment. Obstacles and exit regions are sets of terminal states. P or P' is solved with MCVI, and a policy mapping observations to actions is returned. This reward-maximizing policy will be followed one action at a time with an online system that does respect all the constraints of the system.

The online planner uses locally-optimal Dubins (Dubins 1957) and Reeds-Shepp (Reeds and Shepp 1990) paths. These first-order, bounded-curvature paths are time-independent, kinematically valid paths for many car and UAV models. Controllers can easily track these paths under the full system dynamics. The Reeds-Shepp model is almost identical to the Dubins model with the additional ability to go backward. Reeds-Shepp paths clearly are not valid for fixed-wing UAVs, but are more applicable to car-like UGVs. The difficulty of online planning is in satisfying the motion constraints while executing the input policy. Being able to reverse allows the Reeds-Shepp car to easily approach a target orientation and correct for position noise. The effect of this will be seen in the solution percentages.

Transition-based RRT (Jaillet, Cortes, and Siméon 2008) was chosen for online replanning because it plans in a space with an arbitrary reward function to produce low-cost paths. TRRT reward is based on the executions of the policy and the POMDP reward function R. This directly couples the online replanning with the offline policy at a low level. Online replanning is implemented using the open-source OMPL package (Şucan, Moll, and Kavraki 2012). Dubins and Reeds-Shepp models with a minimum turning radius of 0.5 are

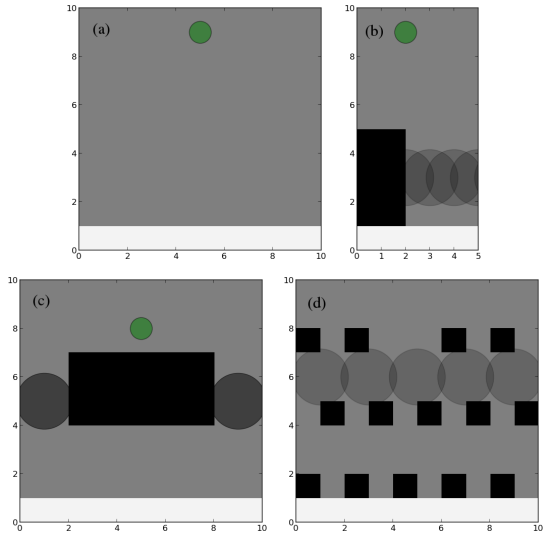

Figure 3: Environments used in trials. They are: (a) empty, (b) single stealth, (c) double stealth, and (d) slalom.

instantiated. $N = 1000$ runs of the policy are executed from the current state, given all previous sensor measurements.

A benefit of our technique is that the same offline policy can be utilized for multiple online systems. In this paper, we will take advantage of this abstraction and compute one policy offline, and then execute it under two different vehicle models, each with its own set of constraints. This represents the potential for massive computational savings, as it indicates that a policy does not need to be re-calculated for small differences in system constraints.

The noise model in the online stage is the same as in P and P'. As we are not testing the effectiveness of online localization algorithms, we have omitted implementing such algorithms. Omitting localization at the online stage is not a limitation of the method, but will negatively affect success rates. Because it affects all experimental conditions equally, it changes the overall results by a uniform reduction in reward and success rates.

As discussed earlier, execution of the policy is done one POMDP action at a time, by computing an intermediate goal that accomplishes each action (Algorithm 2, line 10). Each intermediate goal is a state (x, y, θ), where θ is taken directly from the P policy, and is predicted with a 2-action lookahead when using the P' policy. This prediction considers the direction of the action that joins the next and the next-next state, and then assigns that direction as the goal orientation. If there is only 1 state in the future of either the P or P' policy, then it is assumed to be a final state and an escape goal is used that does not constrain orientation or position within the exit region.

Experimental Evaluation

The environments used are depicted in Figure 3. Most of the space has reward of -1 (gray), terminal escape regions have reward $+900$ (white), and the terminal obstacle regions have a reward of -1000 (black). The large grey circles are the pos-

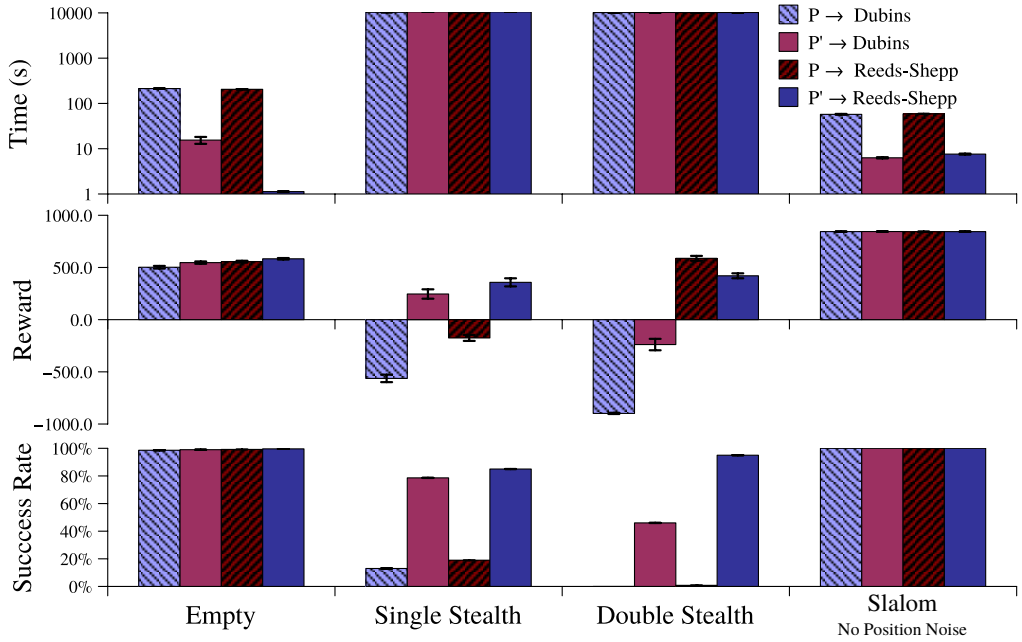

Figure 4: Experimental results are presented in this summary figure. Striped bars used P, solid bars used P'.

sible locations of the unknown obstacle. There are also large grey circles and small green circles. The small green circle is an initial belief 95% confidence circle. No green circle appears in the slalom environment because this environment was run without position noise. Possible obstacle positions are placed so that, without sensing the unknown obstacle's location, reaching a state with positive reward is unlikely.

Metrics tested are total solution time, policy execution reward, and task success rate. Each metric is computed across 500 trials in each environment and the mean value is shown with a 95% confidence interval in Figure 4. Although solution time is inclusive of both the offline POMDP solution and online replanning, the online time is negligible in all but the simplest tests. To compare between the policies generated from P and P', reward is considered at the online level. This is the reward the robot is actually getting under the true constraints. Each set of bars shows a comparison of the policy generated by solving P and the policy from solving P', when each are executed on the Dubins and Reeds-Shepp models.

The first test, Figure 3(a), was an empty environment, only limited by the bounds and kinematics of the system. In this test, we see that the total computation time for P is more than an order of magnitude greater than for P' (note the log scale). The success rate and reward shows that the policies for P and P' were equally optimal.

In the stealth single passage problem in Figure 3(b), neither policy is optimal within time limits, however, the policy for the simpler POMDP abstraction P' achieves significantly higher reward for both robot models. This test shows that the policy for the more complex POMDP was not just somewhat worse, but performed no better than chance. We conclude that all of the time was spent in determining a policy for only the navigational aspects of the problem, without being able to build a large enough policy to localize the unknown obstacle.

For the stealth double passage in Figure 3(c), making the wrong navigation decision based on the noisy sensing is catastrophic. This makes the problem harder than the single passage problem. If P' is used to generate a policy, the Dubins model gets negative reward. The same policy executed under the Reeds-Shepp car model achieves a very high success rate and a positive reward. From this we can conclude that the policy was good, but actually following it was difficult for the Dubins model. When simulations were visualized, it was apparent that turning corners was often problematic for the Dubins model, as it would often compute paths that were believed to be barely collision free, and, due to the position sensing noise, were actually in collision.

In the slalom environment in Figure 3(d), the noise model was too challenging for either P or P' to produce a policy that had a non-zero success rate. We expect this is due to the large number of turns that are made, a difficulty discussed above. Results from this environment were therefore obtained *without* position noise. In this case, MCVI can find an optimal policy for both P and P'. Note the system does not reduce to an MDP because the unknown obstacle is only partially observable. Due to the increased clutter, the difficulty of achieving specific orientations is highlighted by the reward and success degradation when using the policy from P.

When MCVI converges for both models, P converges significantly faster and the reward of both policies are approximately equal. When MCVI times out, it is clear that the policy from P' is closer to optimal and yields significantly higher reward. The generality of online replanning is hinted at by using a single offline policy for two online models.

Discussion

To address sensing-based robotic tasks with a complex reward structure, we propose that the correct level of abstrac-

tion is to remove complex motion constraints from a POMDP. Removed constraints can be incorporated online with replanning. Our results show that, for the problem instances we have considered, our proposed algorithm is superior.

It is clear from our results that complex constraints may cause an explosion in the computation time such that the policy returned may have extremely poor success rates – 0% in some cases. Delaying the consideration of these constraints is shown to be an effective strategy to generate better policies and improve success rates.

For the future, it is clear that a localization algorithm will be required. Implementing this addition may make the results harder to analyze, but will help improve success rates. We are also interested in investigating more complex search-and-rescue tasks where the goal may be unknown.

Acknowledgements This work was partially supported by NSF 1139011, NSF DUE 0920721, CCF 1018798 and the U. S. Army Research Laboratory and the U. S. Army Research Office under grant number W911NF-09-1-0383. Equipment funded in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097. D. Grady was additionally supported by an NSF Graduate Research Fellowship.

References

Agha-mohammadi, A.-a.; Chakravorty, S.; and Amato, N. M. 2011. FIRM: Feedback controller-based information-state roadmap - A framework for motion planning under uncertainty. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, volume 1, 4284–4291.

Bai, H.; Hsu, D.; Lee, W. S.; and Ngo, V. 2010. Monte Carlo value iteration for continuous-state POMDPs. In *Workshop on the Algorithmic Foundations of Robotics*, 175–191.

Bai, H.; Hsu, D.; and Lee, W. S. 2013. Planning How to Learn. In *IEEE Intl. Conf. on Robotics and Automation*.

Van den Berg, J.; Patil, S.; and Alterovitz, R. 2012. Efficient approximate value iteration for continuous Gaussian POMDPs. In *AAAI Conf. on Artificial Intelligence*, 1832–1838.

Candido, S., and Hutchinson, S. 2011. Minimum uncertainty robot navigation using information-guided POMDP planning. In *IEEE Intl. Conf. on Robotics and Automation*, 6102–6108.

Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79(3):497–516.

Foka, A., and Trahanias, P. 2007. Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics and Autonomous Systems* 55(7):561–571.

Grady, D.; Moll, M.; and Kavraki, L. E. 2013. Automated Model Approximation for Robotic Navigation with POMDPs. In *IEEE Intl. Conf. on Robotics and Automation*.

Hsiao, K.; Kaelbling, L. P.; and Lozano-Perez, T. 2007. Grasping POMDPs. In *IEEE Intl. Conf. on Robotics and Automation*, 4685–4692.

Jaillet, L.; Cortes, J.; and Siméon, T. 2008. Transition-based RRT for path planning in continuous cost spaces. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2145–2150.

Kaplow, R.; Atrash, A.; and Pineau, J. 2010. Variable resolution decomposition for robotic navigation under a POMDP framework. In *IEEE Intl. Conf. on Robotics and Automation*, 369–376.

Kurniawati, H., and Patrikalakis, N. M. 2012. Point-Based Policy Transformation : Adapting Policy to Changing POMDP Models. In *Workshop on the Algorithmic Foundations of Robotics*, 1–16.

Kurniawati, H.; Bandyopadhyay, T.; and Patrikalakis, N. M. 2012. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots*.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*.

Lee, T., and Kim, Y. J. 2013. GPU-based Motion Planning under Uncertainties using POMDP. In *IEEE Intl. Conf. on Robotics and Automation*.

Lim, Z.; Hsu, D.; and Lee, W. S. 2011. Monte Carlo Value Iteration with Macro-Actions. In *Advances in Neural Information Processing Systems*.

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

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

Pineau, J.; Roy, N.; and Thrun, S. 2001. A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning*.

Reeds, J. A., and Shepp, L. A. 1990. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145(2):367–393.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-Draa, B. 2008. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32(2):663–704.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Conf. on Uncertainty in Artificial intelligence*, 520—-527.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Uncertainty in Artificial Intelligence*.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72—-82.

Wang, M., and Dearden, R. 2012. Improving Point-Based POMDP Policies at Run-Time. In *Workshop of the UK Planning And Scheduling SIG*.

Zhang, Z., and Chen, X. 2010. Accelerating Point-Based POMDP Algorithms via Greedy Strategies. *Simulation, Modeling, and Programming for Autonomous Robots* 6472:545–556.