Offline and Online POMDP Solution Approaches for Roomba Navigation

Brad Cash [bradcash@stanford.edu]
Joel Persson [joelpe@stanford.edu]

Abstract

This paper compares the implementation of three unique POMDP solution methods to a common agent and operating environment. The paper begins with a detailed introduction of the common environment by describing the agent, a Roomba, its physical operating environment, a single room, and the resulting state, action, and observable space. One offline and two online POMDP solution methods are tested with an explanation of their implementation in relation to the environment. QMDP is the offline solution method and POMCP and POMCPOW are the two online solution methods implemented. Solution method results are compared according to a performance metric introduced in the paper. We find that QMDP is outperformed by both POMCP and POMCPOW, while POMCPOW has a slightly better performance measure than POMCP.

1 Introduction

An autonomous, self-driving vacuum cleaner, referred to in our paper as the Roomba, is powered on inside a room. The Roomba knows the layout of the room but is not aware of its initial position and heading within the room. In order to begin its cleaning duties in the adjacent room, the Roomba needs to safely and efficiently navigate its way outside of the room by passing through the room's single doorway. The room the Roomba currently occupies, explained in further detail in the Operational Environment section below, also has an entrance into a descending staircase. Entering the staircase spells almost certain destruction for the Roomba and mission failure for cleaning the adjacent room. The Roomba is equipped with one sensor for observing the environment, a bump sensor. The bump sensor alerts the Roomba when it has come in contact with a wall. In addition to the bump sensor, the Roomba maintains a history of its preceding movements relative to its current state. While the bump sensor serves as the primary instrument for learning the environment as the Roomba explores the room, limiting contact with the wall is valued as repeated contact may be damaging. In addition, repeated contact with the wall, or edges of the room, may increase the probability of total failure with an unwanted excursion down the stairs. With these considerations, the methods implemented to optimally maneuver through our partially observable space need to balance efficiency and safety, or commonly referred to in decision making in a partially observable Markov decision process (POMDP) as environment, exploration and exploitation.

A collection of both offline and online POMDP solution methods are implemented to varying degrees of success. Success, for our implementation, is measured by simple aggregation of values associated with the final state of the Roomba when each trial is complete. We take the mean for all trials to directly compare iterations between applied solution methods. Each trial will be conducted for a maximum period of 100 time steps. 100 time steps is our determined threshold for a Roomba to efficiently navigate into the adjacent room. The trial ends when the Roomba enters the doorway, enters the stairs, or 100 time steps have lapsed. For a trial, if the Roomba reaches the doorway, the resulting value is 1. If the Roomba enters the stairwell, the value for the trial is -1. Lastly, if 100 time steps lapse and the Roomba is still in the room, the resulting value is 0. The mean value for all trials will serve as our comparable metric of success for a tested solution method. Three solution methods were tested in this paper: one offline and two online methods. The offline method is QMDP, Q Markov decision process, with the Q representing the value function associated with a state/action pair. The online methods utilized are POMCP, partially observable Monte Carlo planning, and POMCPOW, partially observable Monte Carlo planning with observation widening.

The state and action space within the room is continuous. For the application of the QMDP solution method, the state and action spaces are discretized. For the application of the POMCP and POMCPOW solution methods, only the action space is discretized. Observations are limited to contact or no contact with the wall, entering the doorway, or entering the staircase. POMCP and POMCPOW solution methods utilize particle filtering to represent belief for the Roomba's current state.

2 Background

POMDPs are an extension of Markov decision processes (MDPs). Unlike a strict MDP, the agent of a POMDP is not able to observe the entire state space and its relative position therein. The agent must rely on what is observable in the current and preceding time steps to generate beliefs about existing in a collection of possible states. The solution methods incorporate and updates these beliefs to effectively plan for the optimal actions to take in the succeeding time steps. To address high dimensionality of planning through a large, continuous state space, several sampling methods are implemented to include particle filtering and Monte Carlo tree search (MCTS). This paper assumes familiarity with POMDPs and the solution methods applied. This paper will not explain in detail the algorithms associated with the POMDP solution methods but will rather focus on their application and results when applied in this specific testing environment. Detailed information with regards to the solution methods can be found by exploring the references in the References section of the paper. This paper will utilize standard POMDP lexicon to include state, action, observation, rewards, value, belief, and particles. Clear and extensive definitions for this verbiage, as they relate to the study of POMDPs, may be found in [1] and [4].

3 Operational Environment

For the purpose of our experimentation, the room may be configured in three different ways. For each configuration, the dimensions of the room remain the same. The dimensions of the room are indicated by the adjacent number. The only variation in the configurations is the location of the doorway into the room and the entrance into the staircase. The doorway is indicated in green and the entrance to the staircase is indicated in red. Ideally, a successful policy solver will safely navigate the Roomba through the doorway in any configuration while avoiding the entrance to the stairs. The three configurations are depicted below.

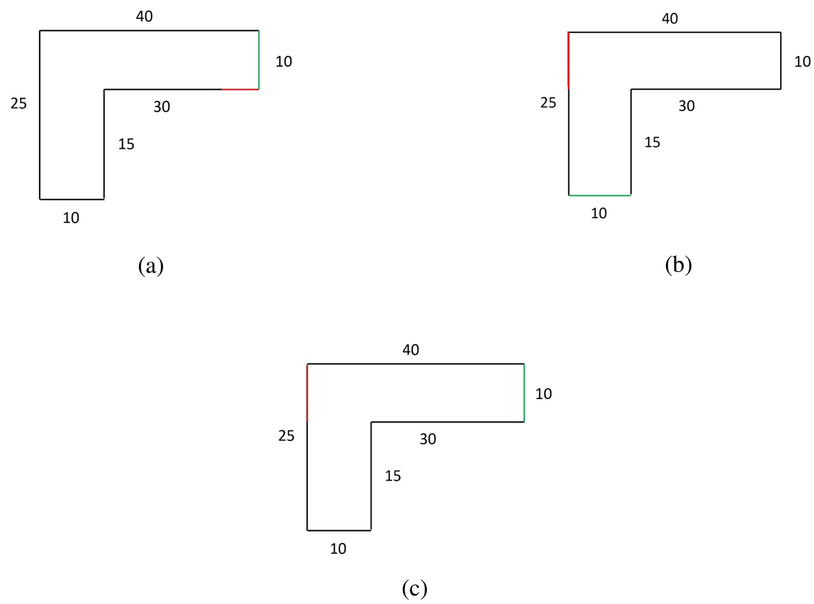

Figure 1: The three different room configurations in the Roomba environment.

4 Methodology

Detailed below are the three POMDP solution methods implemented to navigate the Roomba out of the room and their respective results in accordance with our unique value metric explained in the Introduction. We implemented the solution methods using the Julia coding language leveraging the POMDP library and associated packages for QMDP, POMCP, and POMCPOW.

4.1 QMDP

The first POMDP solution method we implemented was QMDP, the method computes an alpha vector for each action assuming we have full observability of the model [1]. We find the alpha vectors by initializing them to zero and then iterating over:

$$\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) \max_{a'} \alpha_{a'}^{(k)}(s')$$

When we have our set of alpha vectors we find the optimal action, given a belief state, by taking $\max_a \alpha_a^\top \mathbf{b}$. One obvious flaw with this approach is that assumes that all uncertainty regarding the state will disappear in the next time step. Given this information we anticipated that it would not perform very well in the Roomba environment where the agent needs to take multiple actions in a row with the sole purpose of eliminating, or at least reduce, the state uncertainty. Our hypothesis was immediately confirmed as the Roomba did not decide to make any moves that would serve to reduce its uncertainty. However, if we gave the Roomba full observability it would quickly find its way out of the room successfully which demonstrates that QMDP is an efficient algorithm when the state uncertainty has little impact on the optimal actions.

4.2 Partially Observable Monte-Carlo Planning

The second policy solver we implemented was Partially Observable Monte-Carlo Planning (POMCP). The method performs a Monte-Carlo tree search from the current belief states and updates the belief during the tree search using another layer of Monte-Carlo update [2]. The method produces two apparent advantages: using Monte-Carlo update we reduce the curse of dimensionality in the Roomba environment where you have a large (or even continuous) state space combined with a large action space. The second advantage is that we do not need to encode the parameters and the model governing the Roomba. We can let the POMCP solver interact with the environment using a black-box simulation. To find efficient and satisfying results we needed to tune three different hyper-parameters: number of tree queries, exploration coefficient, and the maximum depth for each search. In addition to this the POMCP uses rollout to evaluate the value of each path in the tree. Instead of using a random policy for evaluation we defined a policy that is close to optimal given knowledge of its current state, rollout uses states rather than belief states for evaluation. Utilizing an optimal policy in comparison to a random policy for evaluation resulted in a significant improvement in our evaluation metric. Using an "optimal" policy our Roomba had a success rate of 100% and an evaluation score of 1.0, if we instead used a random policy for rollout the corresponding numbers were 70% and 0.7 since the Roomba did not enter the stairs in any simulation. We evaluated the planners in 10 different simulations and calculated our evaluation metric described in earlier sections. Given the relatively small sample size we can not draw any major conclusions but our limited computational power made large sample sizes intractable.

4.3 Partially Observable Monte-Carlo Planning with Observation Widening

The third planning algorithm applied to the Roomba environment was Partially Observable Monte-Carlo Planning with Observation Widening (POMCPOW). This algorithm is essentially an extension of POMCP where we use a weighted particle filtering in the tree search compared to an unweighted particle filtering used in POMCP [3]. While POMCP supports operating on continuous state spaces it requires discrete action and observation spaces, in contrast, POMCPOW allows us to plan in environments having continuous state, action and observation spaces. Similar to when implementing POMCP, we needed to tune hyper-parameters to find satisfying results. In addition to the hyper-parameters from POMCP, we also needed to determine the criterion and its exploration coefficient used for choosing an action at each node during the Monte-Carlo search.

5 Results

In Table 1 we can observe how our evaluation metric differs across the three implemented POMDP solvers and for the various room configurations. We see that POMCP and POMCPOW performs similar to each other across the configurations, although POMCPOW appears to perform slightly better. The results confirm our expectations since the POMCPOW algorithm uses weighted belief sampling when searching the tree while POMCP uses unweighted belief states. For this environment we anticipated weighted sampling to perform better than unweighted sampling since it encourages the Roomba to take actions that are better aligned with the observations.

Based on the results above we concluded that POMCPOW was the better of the three methods. The evaluation was done on the same 10 seeds for all the methods making

Room configuration	QMDP	POMCP	POMCPOW
#1	0	1.0	1.0
#2	0	-0.1	0.1
#3	0	-0.2	0.5

Table 1: Results for 3 different POMDP solvers.

the comparisons justifiable. However, the hyper-parameter tuning was performed on the same 10 seeds making the results in the table an optimistic estimate of the true score. To get an understanding of how good the method performs out of sample we simulated POMCPOW using the first configuration on 10 new seeds that it had not been trained on before, i.e. these seeds had not been used for hyper-parameter tuning or evaluation. The evaluation score on these 10 new seeds were 0.5 compared to 1.0 in sample. This suggests that the hyper-parameters might have been overfit, or the first samples just showed a high score due to randomness. However, a score of 0.5 is still satisfying and the Roomba had a failure rate of 0% and a success rate of 100% in the out of sample test.

6 Discussion

While our results largely confirmed our expectations of offline versus online solver performance and POMCP versus POMCPOW performance, we did not expect the large degree in performance variation with respect to different room configurations. Both the POMCP and POMCPOW algorithm had superior performance in the first configuration compared to the other settings. Judging by Figure 1 it is logical that the first room has a higher score than the two other environments since the first configuration has both a smaller area/length for the stairs and a less critical placement of the stairs than the two other environments. We say less critical, since there are fewer states were the Roomba might actually be facing the stairs when it has a large state uncertainty. In future work, we would devote more time to identifying the source of this discrepancy.

To better differentiate the degree of performance between POMCP and POMCPOW, future work would include executing simulations through a larger number of trials while utilizing greater values for the depth of search and number of tree queries hyper-parameters. We hypothesize this work would result in a more pronounced difference in performance between POMCP and POMCPOW. An effort would also be made to further debug the simulation environment. An intermittent error occurred across both discretized and continuous state space simulations when the Roomba contacted the one convex corner of the room. The Roomba appeared to be stuck while believing it was actually moving through the room. In addition to more rigorous simulations for the POMCP and POMCPOW solution methods, we would like to implement the Determinized Sparse Partially Observable Trees (DESPOT) POMDP solution method and the Regularized DESPOT (ARDESPOT) method [5].

References

[1] M. J. Kochenderfer. Decision Making Under Uncertainty: Theory and Application. MIT Press, 2015.

[2] David Silver, Joel Vaness. "Monte-Carlo planning in large POMDPs." *Advances in Neural Information Processing Systems 23*, 2010.

[3] Zachary Sunberg and Mykel Kochenderfer. "Online Algorithms for Continuous State, Action, and Observation Spaces." *ICAP*, 2018.

[4] Peigen Liu, Jeng Chen, Hongfu Liu. "An Improved Monte Carlo POMDPs Online Planning Algorithm Combined with RAVE Heuristic." *ICSESS*, 2016.

[5] Adhiraj Somani, Nan Ye, David Hsu, Wee Sun Lee. "DESPOT: Online POMDP Planning with Regularization." *NIPS*, 2013.