Escape Roomba : Policy Evaluation using POMDP

Gao Xian Peh[1]

Christina Lin[2]

and

Won Sup Song[3]

Stanford University, Stanford, CA, 94305, USA

I. Abstract

In this paper, we explored multiple techniques to enable a Roomba robot to efficiently navigate out of a room without tumbling down staircases. The readings from the bumper sensor on the Roomba robot are fed into a particle filter which performs object localization to identify the precise location of the object. To identify the best performing technique, we calculated the average effectiveness of each technique over multiple trials over three different configurations of the goal and staircase in the room.

II. Introduction

Home services is an example of an industry which could be further developed via the use of robots. The Roomba robot was first introduced in September 2002 by iRobot [1]. With more than 6 million home robots sold worldwide, the Roomba robot is becoming increasingly pervasive in the typical household. The Roomba robot is equipped with multiple sensors that enable it to navigate the floor area of a home and clean it. However, there are some technical challenges that the Roomba robot needs to overcome.

The challenge for the Roomba to effectively exit a target doorway while avoiding the undesirable doorway that leads to falling down the staircase can be modelled as a Partially Observable Markov Decision Process (POMDP). In a POMDP, a model specifies the probability of making a particular observation given the current state. The decisions in a POMDP at time t can only be based on the history of observations $o_{1:t}$. Instead of keeping track of arbitrarily long histories, we keep track of the belief state which is a distribution over states. In a POMDP, actions are obtained from the policy evaluated at the current belief state. Our current belief state, represented as a collection of particles, is updated each time we receive a new observation and reward on each timestep. A particle is a sample from the state space. This belief update is carried out via a particle filter, which is an algorithm that adjusts our collection of particles based on the observation received.

In this paper, we model an action to be a (velocity, angular velocity) pair and a state to be a tuple consisting of: (x-coordinate of Roomba in meters, y-coordinate of Roomba in meters, orientation of Roomba in radians, boolean indicator of whether Roomba robot has reached goal or stairs).

Our paper proceeds as follows. We first provide the background for the various techniques that we have used to solve the POMDP that we have previously modelled in *Section III: Relevant Work*. Subsequently, we describe the

[1] M.S. Candidate, Computer Science, School of Engineering
[2] NDO Student, Stanford Center for Professional Development
[3] M.S. Candidate, Mechanical Engineering, School of Engineering

techniques that we have used in *Section IV: Methods.* Next, we evaluate and discuss the performance of each of our methods in solving the Roomba challenge in *Section V: Evaluation* and *Section VI: Results and Discussion*. Finally, we conclude and discuss our next steps in *Section VII: Conclusion and Next Steps.*

III. Relevant Work

While there are a number of frameworks that solve fully observable decision making problems, few are able to accommodate partially observable domains. POMDPs.jl introduces an interface that allows solving partially observable Markov decision processes (POMDPs) by defining problems, creating solvers, and running experiments [2].

One natural approach to finding Q functions for POMDP's is to make use of the Q values of the underlying MDP. In other words, the QMDP (s,a) values could be found temporarily ignoring the observation model and finding QMDP consisting of the transitions and rewards only. For QMDP, all the QMDP values for each action is treated as a single linear function so that the action whose long-term reward from all states is largest will be the one chosen at each step [3]. Also, policies based on this approach can be remarkably effective but these policies will not take actions to gain information. This can lead to situations in which the agent loops forever without changing belief state [4].

Monte-Carlo Tree Search (MCTS) is an approach to online planning that breaks the curse of dimensionality by sampling state transitions instead of considering all possible state transitions. It requires a black box simulator and uses random simulations to estimate the potential for long-term reward, so that it plans over large horizons. This approach is extended to POMDPs, resulting in Partially Observable Monte Carlo Process (POMCP) where Monte-Carlo sampling is used to break both curse of dimensionality and curse of history by sampling start states from the belief state, and by sampling histories using a black box simulator [5].

Determinized Sparse Partially Observable Tree (DESPOT) focuses online planning on a set of randomly sampled scenarios and compactly captures the "execution" of all policies under these scenarios. The construction of the DESPOT leverages a set of randomly sampled scenarios. A DESPOT encodes the execution of all policies under a fixed set of K sampled scenarios. It alleviates the "curse of dimensionality" by sampling states from a belief and alleviates the "curse of history" by sampling observations [6].

IV. Methods

Recall that we model an action to be an (velocity, angular velocity) pair and a state to be a tuple consisting of (x-coordinate of Roomba in meters, y-coordinate of Roomba in meters, orientation of Roomba in radians, boolean indicator of whether Roomba robot has reached goal or stairs). In addition, we calculate rewards as such:

Description	Reward
Reward for non-consecutive wall contact	- 1.0
Reward per timestep	- 0.1
Reward for reaching goal	+ 10
Reward for reaching stairs	- 10

Fig. 1 Table for Rewards

For context, we provide a visualization of the enclosing room with the Roomba below. The goal is highlighted in green while the staircase is highlighted in red.

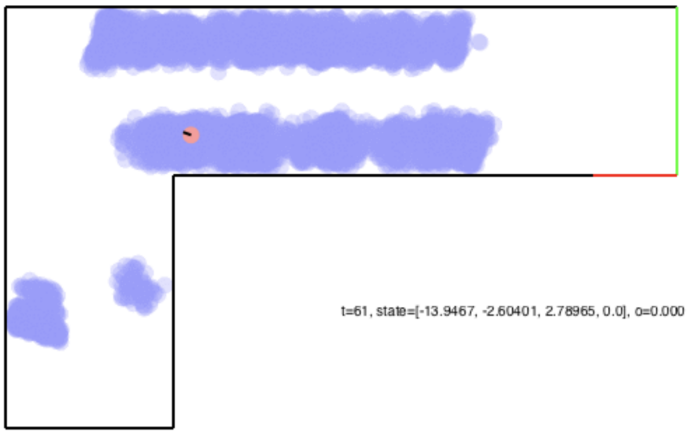

Fig. 2 Depiction of the Roomba environment

For our particle filter, we instantiate a particle filter as implemented in ParticleFilters.jl with a noise coefficient for velocity of 2.0 and a noise coefficient for turn rate of 0.5.

We will now describe the various techniques that we have used to generate a policy that attempts to navigate the Roomba to the goal. For each POMDP solver, we conduct a greedy hyperparameter search as detailed in the appendix to find the optimal values for the parameters of each POMDP solver and the optimal discretization of the state space. The hyperparameter search was done over configuration 3 of the room.

A. Baseline Policy

The bumper sensors on the Roomba learns about the position that it is in after bumping into the walls of the room for several times. Our policy begins by moving in the angle defined by two points: the goal and the (x,y) coordinates of the mean of our beliefs. We named this angle as angle_to_goal. Upon bumping into a wall, the Roomba will move with reduced velocity and begin to rotate clockwise so as to "bump" off from the wall.

The Roomba will then continue by moving in the angle defined by angle_to_goal. As the Roomba continues to bump into different walls, the Bumper Particle Filter will resample our belief states, which further refines the accuracy of the (x,y) coordinates of the mean of our beliefs to the actual Roomba location. Eventually, our model correctly finds the actual Roomba location and the Roomba makes its way to the goal state. As part of a potential next step to further improve our baseline policy, we could add logic to ensure that the Roomba avoids moving in the direction towards the stairs.

B. QMDP

QMDP uses the following parameters which are described in the appendix: num_x_pts, num_y_pts, num_th_points, vlist, omlist, max_iterations, tolerance. We used JuliaPOMDP/QMDP.jl as our solver.

For QMDP value method, we temporarily ignore the observation model and find the Q(s,a) values for the MDP consisting of the transitions and rewards only. We can treat all the QMDP values for each action as a single linear function and estimate the Q value for a belief state b as $Q_a(b) = \sum_s b(s)Q(s,a)$. This estimate amounts to assuming that any uncertainty in the agent's current belief state will be gone after the next action. Thus, the action whose long-term reward from all states (weighted by the probability of occupying the state) is largest will be the one chosen at each step.

$$\hat{V}(b) = \max_{a \in A} \sum_{s \in S} b(s) Q^*_{MDP}(s, a),$$

$$Q^*_{MDP}(s, a) = \rho(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*_{MDP}(s')$$

Fig. 3 QMDP update equation

C. POMCP

POMCP uses the following parameters which are described in the appendix: num_x_pts, num_y_pts, num_th_points, vlist, omlist, max_depth, c, tree_queries. We used JuliaPOMDP/BasicPOMCP.jl as our solver.

Partially Observable Monte-Carlo Planning (POMCP) has two properties. First, Monte-Carlo sampling is used to break the curse of dimensionality both during belief state updates and during planning. Second, only a black box simulator of the POMDP is required, rather than explicit probability distributions. These properties enable POMCP to plan effectively in significantly larger POMDPs than has previously been possible. POMCP consists of an Upper Confidence Bound applied to trees (UCT) search that selects actions at each time-step; and a particle filter that updates the agent's belief state. POMCP combines Monte-Carlo belief state updates with PO–UCT, and shares the same simulations for both Monte-Carlo procedures.

D. AR-DESPOT

AR-DESPOT uses the following parameters which are described in the appendix: num_x_pts, num_y_pts, num_th_points, vlist, omlist, **e**psilon_0, xi, K , D, lambda. We used JuliaPOMDP/ARDESPOT.jl as our solver.

Among the Monte Carlo sampling algorithms, one unique feature of DESPOT is the use of regularization over all policies trees T to avoid overfitting to sampled scenarios: it balances the estimated performance value of a policy and the policy size during the online search, improving overall performance for suitable tasks. Forward search is performed to interleave DESPOT construction and optimization.

V. Evaluation

We evaluated each of the generated policies with five metrics:

1. Average number of time steps taken for Roomba to reach a terminal state (either goal state or stairs)
2. Average number of bumps that Roomba took before reaching a terminal state (either goal state or stairs)
3. Average reward obtained
4. Runtime for building the policy
5. Average Runtime for executing the policy

We conducted our evaluation over the three different configurations provided, with each configuration corresponding to a different location for the goal and the stairs. The averages were taken over 5 iterations of each of the 3 different configurations, for a total of 15 iterations. On each iteration, the Roomba was initialized at a different randomly allocated location. Our results can be reproduced as we have allocated a seed for this random process. The maximum number of time steps allowed for each iteration is 500.

The table below summarizes the results of the evaluation of our baseline policy.

	Average number of time steps taken for Roomba to reach terminal state	Average Number of Bumps	Average Reward	Runtime for building the policy (in seconds)	Average Runtime for executing the policy (in seconds)
Baseline Policy	109.27 (Std Error: 46.37)	10.07 (Std Error: 4.57)	-10.55 (Std Error: 7.8)	1.32	18.50 (Std Error: 1.4)
QMDP	155.2 (Std Error: 40.746)	1.6 (Std Error: 0.4)	-19.72 (Std Error: 5.341)	2738.421099	25.822895 (Std Error: 13.85972321)
POMCP	28.8 (Std Error: 8.163)	2 (Std Error: 0.447)	0.32 (Std Error: 4.254)	0.195930	9.6358686 (Std Error: 5.830739146)
AR-DESPOT	15.4 (Std Error: 3.043)	1.2 (Std Error: 0.2)	6.46 (Std Error: 0.602)	0.455958	6.14511 (Std Error: 4.523304321)

Fig. 4 Evaluation metrics of each optimized POMDP solver

VI. Results and Discussion

We observe that the baseline policy usually allows the Roomba to reach the goal. The average reward of -10.55 comes from taking a penalty of -10.927 on for taking 109.27 timesteps to reach a terminal state, -10.07 for bumping into the wall, and a +10 reward for reaching the goal. However, the baseline policy takes a large number of timesteps and bumps before getting there since the baseline policy is for the Roomba to keep bumping into walls randomly until it finds its exact location. Thus, we expect the other algorithmic solutions to successfully navigate the Roomba to the goal with fewer timesteps and fewer wall bumps.

First, we consider the offline policy, QMDP. Compared to our baseline policy, QMDP runs for 50% more time steps and bumps into walls 90% less. Based on average reward, we clearly observe that QMDP performs the worst out of all the four techniques. Its earns an average reward of -19.72, which may be explained by a time penalty of -15.52, -1.6 for bumps, and an additional penalty of -1, which suggests that it has a mixed rate of exiting the room and falling down the stairs. One key difference between QMDP and the other policies, including the baseline policy, is that it does not take bumps into account. Recall that QMDP assumes full observability at the next step and completely ignores the observation model. This may explain why QMDP performs worse on average reward, even compared to that of the baseline policy.

The two online POMDP solvers, POMCP and AR-DESPOT, take significantly fewer timesteps than baseline policy and bump into the walls less often. Compared to QMDP, POMCP and AR-DESPOT both do not assume full observability and both do take into account the observations received by the Roomba robot. As such, POMCP and AR-DESPOT should perform better on average reward than QMDP. Comparing the average reward between POMCP and AR-DESPOT, POMCP seems to perform worse than AR-DESPOT. This may be because POMCP can be sometimes misguided by the upper confidence bound (UCB) heuristic of the UCT algorithm and be overly greedy [6]. On the other hand, AR-DESPOT uses regularization to avoid overfitting to sampled scenarios. This way it balances the estimated performance value of a policy and the policy size during the online search, improving the overall performance of the AR-DESPOT algorithm over the POMCP algorithm.

VII. Limitations, Conclusion and Next Steps

Due to time limitations, we were only able to try out 3 different POMDP solvers along with our baseline policy. As next steps, we could try out other POMDP solvers or reinforcement learning methods. We could also develop tougher room configurations with obstacles, multiple Roomba robots, goals and staircases.

In reality, a Roomba with only a Bumper sensor alone will not be practical because the Roomba will need to bump into walls for several times before being able to find its exact location. This frequent bumping could be annoying to the user and could also lead to a short life-span of the Roomba. As such, the Roomba could be improved further by using Lidar sensors that measure distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. Localization can then be carried out by instructing the Roomba to rotate at a stationary position for a couple of times. Subsequently, we can then apply our techniques to generate an optimum policy for the Roomba to exit the room without falling down the stairs.

VIII. Contributions

Gao Xian and Christina worked on the code for the various techniques implemented and for evaluating each of these techniques. Sup Song worked on finding new techniques to be implemented and on the literature review for the final project paper. Everyone contributed to this final project paper.

Appendix

Policy	Parameters	Description
QMDP, POMCP, AR-DESPOT	num_x_pts and num_y_pts	Discretization of our state space in terms of (x, y) coordinates in meters
QMDP, POMCP, AR-DESPOT	num_th_points	Discretization of our state space in terms of the orientation of the Roomba robot in radians
QMDP, POMCP, AR-DESPOT	vlist	List of possible velocities (meters per second) that the Roomba robot can take
QMDP, POMCP, AR-DESPOT	omlist	List of possible angular velocities (radians per second) that the Roomba robot can take
QMDP	max_iterations	The maximum number of iterations that the QMDP algorithm can run
QMDP	tolerance	Tolerance, which is also known as the bellman residual, of the QMDP algorithm
POMCP	max_depth.	Rollouts and tree expansion will stop when this depth is reached
POMCP	c	UCB Exploration constant which specifies how much the solver should explore
POMCP	tree_queries	Number of iterations during each action() call
AR-DESPOT	**e**psilon_0	Current gap at a particular node in the tree
AR-DESPOT	xi	Node on the path from node b to the root of D
AR-DESPOT	K	Number of Scenarios
AR-DESPOT	D	Depth of the belief tree
AR-DESPOT	lambda	Lambda parameter of objective function

Fig. 5 Description of parameters used in each POMDP solver

POMCP Solver

num_x_pts, num_y_pts	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
(25, 25)	-6.48	6.951	3.6	1.249	82.8	28.824
(50, 50)	-5.94	5.101	1	0	21.4	9.212
(100, 100)	-9.98	7.787	4.2	1.393	73.8	23.265
(150, 150)	-6.94	6.015	3.4	0.51	53.4	17.371

num_th_points	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
6	-27.74	9.5	0.8	0.2	205.4	120.278
18	-16.6	6.964	3	0.949	110	40.728
30	-15.68	5.835	2.4	0.245	68.8	29.078
60	-5.94	5.101	1	0	21.4	9.212
90	-13.58	3.331	2	0.707	47.8	14.627
180	-7.72	4.156	4.2	1.281	89.2	24.6
360	-17.1	2.595	3.4	1.288	77	19.013

v_list	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
[5]	-19.68	2.721	4.6	0.98	84.8	20.906
[10]	0.32	4.254	2	0.447	28.8	8.163
[5, 10]	-5.94	5.101	1	0	21.4	9.212
[1, 5, 10]	ERROR: LoadError: ArgumentError: range must be non-empty					
[0, 1, 5, 10]	-20.56	5.501	2.2	1.2	75.6	42.499
[-1, 0, 1, 5, 10]	-21.26	11.151	5.4	1.965	134.6	57.319
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	-21.26	11.151	5.4	1.965	134.6	57.319

omlist	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
[-1, 0, 1]	0.32	4.254	2	0.447	28.8	8.163
[-1, -0.5, 0, 0.5, 1]	-13.18	4.141	1.8	0.374	53.8	19.515
[-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1]	-14.12	6.734	1.6	0.927	57.2	37.995

max_depth	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
1	0.32	4.254	2	0.447	28.8	8.163
5	0.32	4.254	2	0.447	28.8	8.163
10	0.32	4.254	2	0.447	28.8	8.163
20 (default)	0.32	4.254	2	0.447	28.8	8.163
40	0.32	4.254	2	0.447	28.8	8.163
60	0.32	4.254	2	0.447	28.8	8.163
80	0.32	4.254	2	0.447	28.8	8.163
100	0.32	4.254	2	0.447	28.8	8.163
500	0.32	4.254	2	0.447	28.8	8.163

c (exploration constant)	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
0.01	0.32	4.254	2	0.447	28.8	8.163
0.1	0.32	4.254	2	0.447	28.8	8.163
0.5	0.32	4.254	2	0.447	28.8	8.163
1 (default)	0.32	4.254	2	0.447	28.8	8.163
5	0.32	4.254	2	0.447	28.8	8.163
10	0.32	4.254	2	0.447	28.8	8.163
10000	0.32	4.254	2	0.447	28.8	8.163

tree_queries	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
100	0.32	4.254	2	0.447	28.8	8.163
500	0.32	4.254	2	0.447	28.8	8.163
1000 (default)	0.32	4.254	2	0.447	28.8	8.163
2000	0.32	4.254	2	0.447	28.8	8.163
5000	0.32	4.254	2	0.447	28.8	8.163

QMDP Solver

max_iterations	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
10	error					
20 (default)	-24.44	7.315	1.6	0.4	222.4	80.32
50	-20.24	5.103	1.8	0.49	158.4	40.985
100	-19.72	5.341	1.6	0.4	155.2	40.746
200	-19.72	5.341	1.6	0.4	155.2	40.746

tolerance	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
1.00E-08	-19.72	5.341	1.6	0.4	155.2	40.746
1.00E-07	-19.72	5.341	1.6	0.4	155.2	40.746
1.00E-05	-19.72	5.341	1.6	0.4	155.2	40.746
1e-3 (Default)	-19.72	5.341	1.6	0.4	155.2	40.746
1	-24.08	7.441	1.6	0.4	218.8	80.561

AR-DESPOT Solver

epsilon_0	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
0 (default)	-6.46	3.805	0.8	0.374	68.6	40.643
0.2	0.46	3.06	1.4	0.4	35.4	10.515
0.4	-4.26	7.792	2	0.548	116.6	74.296
0.6	-3.88	4.882	2.4	0.51	68.8	23.735
0.8	-12.26	4.944	2	0.707	36.6	13.655
1	2.14	3.423	1	0.316	18.6	6.202

xi	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
0	1	5.002	2	0.775	60	42.576
0.2	-2.26	3.819	2.6	0.927	88.6	33.369
0.4	-3.84	6.235	2	0.447	32.4	12.632
0.9	-3.6	3.858	1.4	0.4	34	12.748
0.95 (default)	-10.7	4.629	1.6	0.4	27	12.771
1	ERROR: LoadError: ArgumentError: range must be non-empty					

K (number of scenarios) (tuned with xi=0)	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
250	-5.84	5.417	1.8	0.374	54.4	39.828
500 (default) [running]	-4.8	4.732	2.8	0.86	32	11.653

D (tuned with xi=0.2, k=250)	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
0	2.9	1.72	2.8	0.663	37	12.325
90 (default)	-0.58	4.647	1	0.316	7.8	1.562
100	-7.78	12.225	2.4	0.98	127.8	94.852

lambda (tuned with xi=0.2, k=250, d=0)	Mean Total Reward	StdErr Total Reward	Mean Bumps	StdErr Bumps	Mean Time Steps	StdErr Time Steps
0	-1.76	4.401	1.6	0.4	17.6	10.405
0.01 (default)	6.46	0.602	1.2	0.2	15.4	3.043
0.1	-4.46	5.049	0.8	0.374	8.6	2.96
0.5	-5.22	4.688	1.2	0.583	14.2	5.739
1	-5.56	4.397	1.2	0.374	15.6	6.675

Fig. 6 Hyperparameter Search for POMCP, QMDP and AR-DESPOT

Acknowledgments

We thank Professor Mykel Kochenderfer and the AA228 Teaching Staff for their guidance and assistance with this project.

References

[1] IRobot Corporation, "Our History" *iRobot Corporation: Our History*. [online] URL: https://web.archive.org/web/20120103091646/http://www.irobot.com/sp.cfm?pageid=203 [retrieved 24 November 2018].

[2] Egorov, M. Sunberg, Z. N. Balaban, E. Wheeler, T. A. Gupta, J. K. and Kochenderfer, M. J. "POMDPs.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research*, vol. 18, iss. 26, p. 1–5, 2017.

[3] Littman, M. L., Cassandra, A. R., and Kaelbling, L. P., "Learning policies for partially observable environments: Scaling up," *Machine Learning Proceedings 1995*, 1995, pp. 362–370.

[4] Hauskrecht, M., "Value-Function Approximations for Partially Observable Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 13, Jan. 2000, pp. 33–94.

[5] Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems* (pp. 2164-2172).

[6] Somani, A., Ye, N., Hsu, D., & Lee, W. S. (2013). DESPOT: Online POMDP planning with regularization. In *Advances in neural information processing systems* (pp. 1772-1780).