Solving Escape Roomba

Joan Chen
joanchen@stanford.edu

This paper explores the problem of the escape Roomba. In this problem, the Roomba must guide itself out of a room without knowledge of its current state. However, it is aware of the room layout and can gather observations that come from its bumper sensors. This problem can be approached as a POMDP, and as such, we use both a baseline policy and POMCP solver to guide the Roomba safely out of the room.

I. Introduction

Our Roomba is on the run, attempting to escape its current room. After a long party the night before, the Roomba has been left with a large mess to clean up. Now, it needs to get out of its current room and move onto the adjacent room which is in tremendous disarray. Unfortunately, the task of escaping its current room is not as easy as it might initially sound.

The Roomba has no idea where it starts off at in the room. Without knowledge of where it currently is, the Roomba must utilize its bumper sensors, which indicate when there is contact with the the Roomba and the wall, to lead itself out of the room. Additionally, it must avoid too many damaging collisions against the wall or a lethal fall down the stairs.

With only basic information on the Roomba's current state from a limited bumper sensor and knowledge of the room, this problem is a great example of utilizing and collecting information to manage and make decisions in uncertainty. The Roomba must balance the need to collect information through wall contacts against the goal of safety and efficient navigation out of the room.

This paper explores various approaches to solving the problem of getting the Roomba out of the room based on the limited observations and knowledge. In our research, we introduce both a policy that follows the wall and a POMCP solver.

II. Related Work

A. MDPs

In a Markov decision process, or MDP, the environment's dynamics are fully determined by the agent's current state. This means that an agent choose action a_t at time t based on its observation of state s_t. As a result, the agent is then to receive reward r_t. For any state and action pair, the transition probabilities determine the next state distribution and the reward function determines the expected reward.

From the MDP, we want to derive a policy that determines what action to select given the past history of states and actions. Using the Markov assumption which states that the next state depends only on the current state and action, we can focus on policies that are reliant only on the current state when solving MDPs. [1]

B. POMDPs

Expanding upon the Markov decision process, we have the partially observable Markov decision process, or POMDP, which is an extension of an MDP. The Roomba problem we are attempting to solve in this paper is one such example of a POMDP.

A POMDP's system dynamics are determined by an MDP. However, what differentiates a POMDP from an MDP is that the agent cannot directly observe the underlying state. Due to this, the model must track the probability of making a particular observation given the current state. [1]

One way to look at a POMDP is to view it as an MDP with an observation model. We utilize the observation model to determine the probability of observing o given state s. When making decisions on which action to take, it is common to keep track of the belief state which is a distribution over states. When designing a policy in a POMDP, we create a mapping from belief states to actions. Thus, we can also say that a POMDP is actually an MDP in which the states are belief states. [1]

Using the Roomba problem as our base, we will further explore some of the various ways to solve a POMDP in this paper.

C. POMCP Solver

One approach to solving a POMDP is through a Partially Observable Monte Carlo Planning (POMCP) solver. A POMCP solver is an online algorithm that combines a Monte-Carlo update of the agent's belief state with a Monte-Carlo tree search from the current belief state.

The algorithm constructs, online, a search tree of histories where each node of the search tree estimates the value of a history by Monte-Carlo simulation. It combines a Monte-Carlo update of the agent's belief state with a Monte-Carlo tree search from the current belief state. For each simulation, the start state is sampled from the current belief state, and state transitions and observations are sampled from a black box simulator, instead of explicit probability distributions. As a result, POMCP can plan effectively in significantly larger POMDPs. [2]

III. Methods

A. Particle Filter

In order to solve a POMDP, we must track our belief state and update it given a new observation and action. To do so, we utilize a particle filter that computes the posterior distributions of the states of some Markov process, given some noisy and partial observations.

To implement our particle filter, we first instantiate a resampler, which updates the belief state given an observation. Next, we instantiate a SimpleParticleFilter, which enables us to perform our belief updates. Lastly, we pass this particle filter into a custom struct called a RoombaParticleFilter, which takes two additional arguments. These arguments specify the noise in the velocity and turn-rate, used when propagating particles according to the action taken. [3]

B. Follow the Wall Policy

Looking at the given Lidar ToEnd policy, we realized a reasonable baseline for the Bumper Roomba would be a policy that has some sort of initial exploration for localization and then utilizes a proportional controller to navigate directly to the goal, using the mean belief state.

Thus, we implemented the following policy:

- To start, the Roomba goes straight in whichever direction it is initially facing at `RoombaAct(2., 0.)` until it hits a wall.
- For the first three time the Roomba makes contact with a wall, it turns right and arcs away from and then back to the wall. This allows the Roomba to follow the wall. With wall contact, the Roomba will take action `RoombaAct(1., -3.)` and arc away at `RoombaAct(2., 0.02)`.
- After three bounces off the wall, we finish our localization portion of the policy. Now, we utilize the proportional controller to navigate to the goal. If we hit the wall, we still turn right at `RoombaAct(1., -3.)` until we no longer have contact with the wall. Then, we continue using the proportional controller.

C. POMCP Solver

The second approach to the Roomba problem attempted to use a POMCP solver to create a policy for the POMDP. To utilize a POMCP solver, we must define a heuristic policy for estimating the value of a state. Given the true state, we do the following:

- If the Roomba's state has contact with the wall, it will turn away from the wall at `RoombaAct(1., -3.)`
- Otherwise, the Roomba determines the angle towards the goal from the current state and travles at `RoombaAct(5., 3.` $\cdot$ `angle)`

IV. Results and Analysis

A. Evaluation Method

To evaluate the methods, we built upon the provided evaluation metric. Our evaluation metric goes through the three possible room configurations and for each configuration, initializes the robot with five random seeds and finds the total reward of each run. From all fifteen total reward values, we report the total mean and standard deviation.

B. Metrics

	Config 1 Mean	Config 2 Mean	Config 3 Mean	Mean Total Reward	StdErr Total Reward
Follow the Wall	3.24	-18.38	3.24	-3.967	4.887
POMCP Solver	-4.22	-11.76	-4.22	-6.733	5.293

C. Analysis

Due to the relatively simple heuristic used by the POMCP solver, it actually performs worse than the Follow the Wall policy. This can be attributed to a few things.

Firstly, the Follow the Wall policy attempts to perform a large arc away from the wall to reduce the number of collisions the Roomba has against the wall. The POMCP solver heuristic does not account for this beyond angling itself towards the goal. This works if there is no wall in the way; however, in the case that there is in fact a wall in the way of the direct path to the goal, this policy results in multiple, unnecessary hits against the wall. Thus, we are increasing the damage sustained by the Roomba and decreasing our reward.

Secondly, the Follow the Wall policy we implemented does a pretty good job of localizing with its approach of first doing three bounces against the wall. The POMCP solver localizes as it goes which works, but seems to take a bit longer than the Follow the Wall policy.

For both policies implemented, we notice that the Roomba does significantly worse in configuration 2 over the other two room configurations. To investigate this a bit further, we notice that when running the simulation, the belief updater ends up ruling out the Roomba's actual current state. This leads to situations like when the Roomba incorrectly believes it has hit the goal already and moves aimlessly in a circle. To address this, further investigation and improvements of the belief updater will need to be done.

V. Future Work

A. Further Investigation on Belief Updater

As mentioned above, the belief updater sometimes rules out the Roomba's actual current state. This leads to an inability for both policies to correctly guide the Roomba to the goal. A deeper dive into why this might be happening and addressing the issue would be helpful in improving the success of the policies.

B. Account for Stairs Location

Currently, our policies do not account for the stairs' location when determining the Roomba's angle and path. Since hitting the stairs results in immediate termination of the Roomba's journey, it is quite important to avoid contact with the stairs.

To further improve upon both policies, we could take into account the additional knowledge of the stairs location and attempt to avoid the lethal stairs. For the POMCP heuristic policy, this could mean turning in place if we are within some distance of the stairs.

C. Tune Hyperparameters

With both policies, there exist a couple of hyperparameters. By doing additional tuning, we may be able to increase our policies' rewards.

For the Follow the Wall policy, we have the following hyperparameters: number of bumps before completing localization, angle to arc away from the wall at, fixed velocity to move, and timesteps to delay after hitting a wall before angling back to goal.

For the POMCP heuristic policy, we have the following hyperparameters: velocity and angle to turn away from the wall at, fixed velocity to move, and proportional control mutliplier used to compute the turn-rate.

We did some runs with different parameter values to determine which are better; however, a more thorough investigation of these hyperparameters values would certainly assist in improving the policies.

D. Improve POMCP Heuristic Policy

The current heurstic policy is quite simple in that it just angles itself towards the goal. This is works well for when the Roomba is quite near the goal; however, lesser so when it is further away. Perhaps, we could further improve upon this policy by accounting for things like where the stairs are, how close to the wall it is, and which part of the room the Roomba is in.

VI. Conclusion

The Roomba problem addressed in this paper is certainly a challenging one for which we must utilize various policies and solvers to address. We approach this problem as a POMDP in which the current state of the Roomba is unknown; however, we have knowledge of the room layout and can gather observations from the bumper sensors. The goal of the Roomba is to reach the room exit while avoiding both the lethal stairs and too many collisions with the wall. With this scenario, we are able to generate our belief state at each time-step through a particle filter and then implement policies to safely guide the Roomba during its escape out of the room. To tackle this problem, we experimented with two different policies.

First, we implemented a baseline policy that we name 'Follow the Wall'. The policy works by intially following the wall for three bounces. With each bounce off the wall, the Roomba performs a long arcs. Through these initial bounces, the Roomba is allowing itself to localize. It then uses the mean of the belief particles to determine what angle it should act at to orient itself towards the goal. Improving upon this policy, we tuned the hyperparameters for the arc angle and initial velocity to increase the policy's performance.

Second, we explored using a POMCP solver to approach the problem. This solver utilizes a heuristic policy that returns an action based on a true state. For our heuristic policy, we oriented ourselves towards the goal. In the case where we are in contact with the wall, we turn away. This simple solver performs relatively well; however, it is a bit more basic and further improvements can be made to avoid too many unnecessary collisions with the wall and to sharply turn away from the stairs and/or the wall when the state is too close.

Through the escape Roomba, we are able to analyze and experiment with the importance of balancing information gathering and reward optimization when solving a POMDP. It also demonstrates the importance of utilizing all possible information when generating a policy, such as the location of the goal, the location of the stairs, and all of the Roomba's sensor observations. With these in mind, we generate two policies to solve this problem in our paper.

References

[1] Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., Üre, N. K., and Vian, J., *Decision Making Under Uncertainty: Theory and Application*, 1st ed., The MIT Press, 2015.

[2] Silver, D., and Veness, J., "Monte-Carlo Planning in Large POMDPs," *Advances in Neural Information Processing Systems 23*, edited by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Curran Associates, Inc., 2010, pp. 2164–2172. URL http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf.

[3] 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, No. 26, 2017, pp. 1–5. URL http://jmlr.org/papers/v18/16-300.html.