Roomba Escape! An Exploration of POMDP Solvers

Sarah Radzihovsky*[1] Heather Shen*[1] Darren Mei*[1]

Abstract— **Partially observable markov decision processes (POMDPs) have been a broad and important area of study for many years. However, the challenges of large and continuous state, action, and observation spaces continue to make solving POMDPs a pressing area of research. In this project, we investigate different offline and online policies for helping an autonomous robotic vacuum cleaner, or Roomba, escape a room with minimal damage. We consider the challenge of real-time applications as well as discretization of a continuous space. In our results and discussion, we discuss how each policy performed and analyze the strengths and weaknesses of each method in the context of this problem.**

I. INTRODUCTION

With advances in robotics, companies are increasingly investing in simple robots with reactive architectures to perform everyday tasks: in-store customer service, managing warehouses, cleaning large spaces, and much more [8].

To remain affordable, these robots must use inexpensive sensors to navigate an unknown area and must avoid obstacles to reduce damage. We model these considerations with the following Roomba model: the Roomba initially has no idea where it is in the room, but luckily the robot has a bump sensor that informs it when it contacts the surrounding walls. The Roomba decides it can use these sensors to figure out its current location, but it has to be careful — collisions with the wall are potentially damaging and one wrong move could cause the robot to take a lethal tumble down a staircase. Thus, the Roomba must balance the need to collect information about its current position with the goal of safely and efficiently navigating its way out of the room.

We can model this problem as a POMDP (partially observable Markov decision process), and like other POMDP problems, there are several challenging factors. The first is the continuous action and state space. The Roomba can move at any velocity (up to a defined maximum) and in any direction. We must also balance our goal of navigating with the fear of damage. In the following sections, we will discuss various methods for solving the POMDP problem.

*Stanford University
[1]Sarah Radzihovsky `sradzi13@stanford.edu`
[1]Heather Shen `hcshen@stanford.edu`
[1]Darren Mei `dmei@stanford.edu`

II. PRIOR WORK

Partially observable markov decision processes, or POMDPs, are frequently used to model and solve real world problems such as air traffic navigation, robotic action sequences, food distributions, and much more. As a result, POMDPs have been an active area of research for many years.

We can consider a POMDP as an MDP in which the states are belief states. Given an initial belief state, we update our belief state using recursive Bayesian estimation based on the last observation and action executed. For general problems with continuous state spaces, we often have to rely on approximation methods, such as sampling with particle filters. We discuss belief updaters further in following subsections.

Even given a reliable belief updater, however, it remains a challenge to scale POMDP planning and achieve reasonable, real-time results for POMDPs with complex dynamics and large state spaces intrinsic to these practical applications. In general, computing exact solutions to POMDPs is intractable [3]. At the root of these scaling issues is the curse of history and the curse of dimensionality. The curse of history causes the number of hyper-planes to grow exponentially with the planning horizon. Simultaneously, the curse of dimensionality causes the volume of the space to increase too quickly with increasing dimensionality: the available data becomes sparse and makes it difficult to create a policy based on beliefs.

Several different offline and online methods have since been derived to tackle these challenges.

A. *Belief Updater*

Given a large or continuous state space, there are two methods we can use: 1) a linear-Gaussian model or 2) a sampling-based approach.

1) Linear Gaussian Model: This method allows us to perform exact belief updates using a linear-Gaussian filter, more commonly known as a Kalman filter. The dynamics and observations can be modeled with the following equations:

$$T(\mathbf{z}|\mathbf{s},\mathbf{a}) = \mathcal{N}(\mathbf{z}|\mathbf{T}_s\mathbf{s} + \mathbf{T}_a\mathbf{a}, \mathbf{\Sigma}_s)$$

$$O(\mathbf{o}|\mathbf{s}) = \mathcal{N}(\mathbf{o}|\mathbf{O}_s\mathbf{s}, \mathbf{\Sigma}_o)$$

Assuming that the initial belief state is represented by a Gaussian, we can update the belief state as follows:

$$\Sigma_b \leftarrow T_s(\Sigma_b - \Sigma_b O_s^T (O_s \Sigma_b O_s^T + \Sigma_o)^{-1} T_s^T + \Sigma_s$$

$$K \leftarrow T_s \Sigma_b O_s^T (O_s \Sigma_b O_s^T + \Sigma_o)^{-1}$$

$$\mu_b \leftarrow T_s \mu_b + T_a a + K(o - O_s \mu_b)$$

2) Particle Filter: When dynamics are not well approximated by a linear-Gaussian model, we can use a sampling-based approach to perform belief updates. The belief state is represented as particles, or samples from the state space. Updating our belief b is based on a generative model which gives the next state s' and observation o'. There are two models commonly used: particle filter with rejection (Alg. 1) and particle filter without rejection (Alg. 2).

Algorithm 1 Particle filter with rejection

```
function UpdateBelief(b,a,o)
$b' \leftarrow \emptyset$
for $i = 1$ to $|b|$ do
    $s \leftarrow$ random state in $b$
    repeat
        $(s', o')$   $G(s, a)$
    until $o' = o$
    Add $s'$ to $b'$
end for
return $b'$
```

As we increase the number of particles, the distribution should approach the true posterior distribution. However, in practice, particle filters can fail: it is possible that random sampling fails to produce any particles near the true state. To mitigate this, we introduce additional noise to the particles.

B. Offline

Offline methods are strategies that compute the policy prior to execution in the environment. In practice, because of high dimensional state and action spaces, we only find approximately optimal solutions. Shani, Pineau, and Kaplow have focused on surveying point-based approximation techniques [5]. Two of the best algorithms, Heuristic Search Value Iteration (HSVI) and Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) involve building search trees through belief space and maintaining upper and lower bounds on the value function [7][6].

Algorithm 2 Particle filter without rejection

```
function UpdateBelief(b,a,o)
$b' \leftarrow \emptyset$
for $i = 1$ to $|b|$ do
    $s_i \leftarrow$ random state in $b$
    $s'_i$   $G(s_i, a)$
    $w_i \leftarrow O(o|s'_i, a)$
end for
for $i = 1$ to $|b|$ do
    Randomly select $k$ with probability proportional to $w_k$
    Add $s'_k$ to $b'$
end for
return $b'$
```

Our main focus will be on two popular offline approximation methods, QMDP and FIB. We discuss the algorithms in further detail in Section III.

C. Online

Of the online methods, some of the most popular are POMCP, DESPOT, and the novel POMCPOW.

1) POMCP: POMCP is a method that was first introduced by Silver and Veness in 2010 [1]. It stands for Partially Observable Monte Carlo Planning. POMCPs address the issues of the curse of dimensionality and the curse of history by using Monte-Carlo sampling. It is an extension of the Monte Carlo tree search that implicitly uses an unweighted particle filter to represent beliefs in the search tree to find a policy for continuous state space and discrete action space problems. Researchers have already proven that POMCPs can be used to solve large game problems, such as Battleship and Pacman [1]

2) DESPOT: DESPOT stands for Determinized Sparse Partially Observable Tree and is a method which approximates the standard belief tree. DESPOT planning demonstrates comparable results with some of the best online POMDP algorithms available and has been used in many applications including autonomous driving systems for real-time vehicle control. [2].

A DESPOT is constructed by applying a deterministic simulator to all possible action sequences under

K sampled scenarios. Each randomly sampled scenario uses a sequence of random numbers that makes the execution of a policy deterministic under uncertainty, and generates a unique trajectory of states and observations given an action sequence. DESPOT encodes the execution of the policies that fit under the fixed set of K sampled scenarios. When K is small, DESPOTs are much sparser than a standard belief tree. However, as K grows, DESPOTs are near optimal, converging to the result of standard belief trees while searching fewer nodes than a standard belief tree would.

Online planning with DESPOTs has two main steps in each iteration: (1) action selection and (2) belief update.

For action selection, Ye et. al. suggests using a heuristic search to find a policy that optimizes a regularized objective function, balancing the estimated values of a policy with the policy size to avoid overfitting [2]. More specifically, the heuristic algorithm constructs a DESPOT incrementally, which can scale to very large POMDPs in practice.

The strength of DESPOT is that it converges to a result that is near optimal while exploring fewer nodes than a standard belief tree would.

3) POMCPOW: The recently published method POMCPOW (Partially Observable Monte Carlo Planning with Observation Widening) can solve problems in continuous state and action spaces without requiring discretization. The main issue with continuous state and action spaces is that it is extremely unlikely to sample the same real number twice from a continuous random variable. This causes the width of the planning tree to explode on the first step, resulting in the curse of dimensionality. POMCPOW solves this by using Double Progressive Widening (DPW) to limit the number of new children sampled from any node in the tree. Although DPW allows for solvers of continuous state and action spaces, POMCPOW also has a different approach to representing beliefs. The belief representations used in other methods to solve continuous action and state space problems collapse to a single state particle, causing the problem to resemble a QMDP policy. POMCPOW instead uses the observation model to weight the particles used to represent beliefs. With these two modifications, POMCPOW has been successful in solving problems where other solvers failed.

III. METHODS

While there are many options to try to solve our Roomba problem, we focus on three: QMDP, FIB, and POMCP as we describe below.

A. Baseline

For our baseline policy, we implemented a period of random exploration followed by a series of greedy actions towards the goal state. For 50 time-steps, the Roomba moves quickly in a random direction, learning about the environment as it is inflicted with stair and collision penalties. Unless the Roomba has randomly found its way to a goal state, the Roomba now uses the mean belief state to follow a proportional controller to navigate directly to the goal. More specifically, we find the difference between the believed and goal position as well as the difference between the believed and goal angle to compute the angle we wish to turn by in the next action step. Because of the initial exploration, the believed state is reliable enough that the Roomba is able to find the goal state.

B. QMDP

In this method, we create a set of alpha vectors, one for each action, using value iteration. Initializing all alpha vectors to zero, we can then iterate:

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

Each iteration requires $O(|A|^2|S|^2)$ operations. The approximately optimal action is given by $argmax_a \alpha_a^T b$.

We chose to run the QMDP policy for 50 iterations. Because we know that complexity is directly linked with the size of the state and action space, we tried to reduce the state and action space without sacrificing dynamics of the problem. We divided the state space in a 50x50x20 space (corresponding to 50 steps in the x and y direction and 20 steps along theta) and the action space varies from velocity = [3,4] and angle = $[\frac{-3\pi}{4}, \frac{-\pi}{2}, \frac{-\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$.

C. FIB

Unlike QMDP, Fast Informed Bound (FIB) takes into account partial observability. We iterate over our alpha vectors with:

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

Each iteration requires $O(|A|^2|S|^2|O|)$ operations, but provides a tighter bound on the optimal value function than QMDP.

We similarly chose to run FIB policy for 50 iterations with the same discretization of the state and action space as described above.

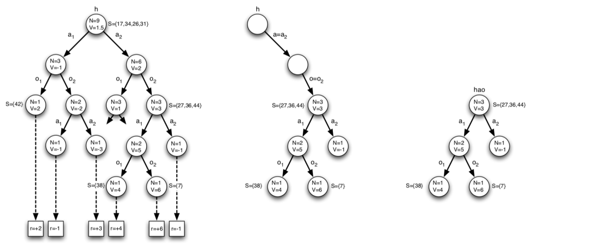


Fig. 1. An illustration of POMCP in an environment with 2 actions, 2 observations, 50 states, and no intermediate rewards. The agent constructs a search tree from multiple simulations and evaluates each history by its mean return (left). The agent uses the search tree to select a real action a and observes a real observation o (middle). The agent then prunes the tree and begins a new search from the updated history hao (right) [1]

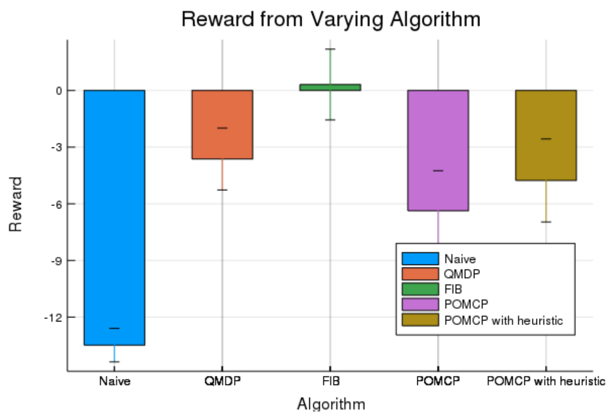


Fig. 2. The average reward from Naive, QMDP, FIB, POMCP, and POMCP with heuristic. Note that these algorithms used the same action space discretizations for comparison.

D. POMCP

The complete POMCP algorithm is described in Figure 1. Within the algorithm, there are several parameters that can be tuned:

- *Max depth*: Rollouts and tree expansion will stop when this depth is reached. Default: 20
- *Exploration constant*: The upper confidence bound (UCB) constant specifies the trade off between exploration and exploitation. Default: 1.0
- *Discretization*: POMCP works with continuous state, but discrete action spaces. Varying the discretization of the action space could improve or hurt the policy.

We explore these three tunable parameters and report our findings in Section IV, Results.

E. POMCP with heuristic

With this method, we incorporated a heuristic policy with the POMCP to estimate the value of a sate. Given the true state, we define what the action should be: the difference between the current and goal position as well as the difference between the current and goal angle tells us how much to turn by in the next action step. Since the action space is discrete in a POMCP, the turn rate of the Roomba is rounded to the nearest discrete value. We always set the velocity to a number within the Roomba's velocity range (0-10).

IV. EVALUATION AND RESULTS

To evaluate our various methods, we collect final rewards from 20 runs with different random seeds and then got the average and standard deviation.

A. Performance of different methods

1) Rewards: In general, FIB had the best average reward among the algorithms we explored (see Figure 2). This could be because we compared methods using the same action space discretization between offline and online methods. In the next sections, we discuss how to improve POMCP performance by tuning various parameters.

2) Time: Reward is not the only indicator of performance though; execution and processing time is also an important factor. We compared the average time per step that the Roomba needed to decide its next action for all methods (see Table I). By evaluating it against our baseline implementation, we wanted to find the optimal tradeoff of speed and performance.

As expected, the Naive baseline policy was the fastest method. QMDP and FIB had very similar time per step because the majority of the computation occurs prior to execution. Therefore, the time between steps is mainly to choose the best action given the current belief state. However, FIB required a significantly longer pre-execution time to create its alpha-vector policy than QMDP.

The POMCP implementation with a heuristic is the best combination of both speed and performance. Not only does it perform better than our baseline and standard POMCP implementation, it also spends less time between steps than the POMCP policy without a heuristic.

B. Tuning POMCP Parameters

1) Max Depth: As we see in Figure 3, varying the maximum depth for a rollout affects the average reward of the POMCP policy. Although we expected

Planning Method	Execution Time Per Step (s)	Pre-Execution Time
Baseline	0.246	N/A
QMDP	0.400	$\sim$ 29 min
FIB	0.408	$>$ 8 hours
POMCP w/ heuristic	0.803	N/A
POMCP	1.180	N/A

TABLE I
AVERAGE TIME PER STEP FOR PLANNING METHODS

that increasing the rollout depth would also increase the reward, the actual results proved otherwise. The best depths, in order, are: 20, 100, 50, and 10. There's no clear pattern to what makes a depth better than the next. Given that there is a large unknown gap between testing depth of 50 to 100, our next step would be to more finely discretize the depths that we are testing and see if that gives more insight.

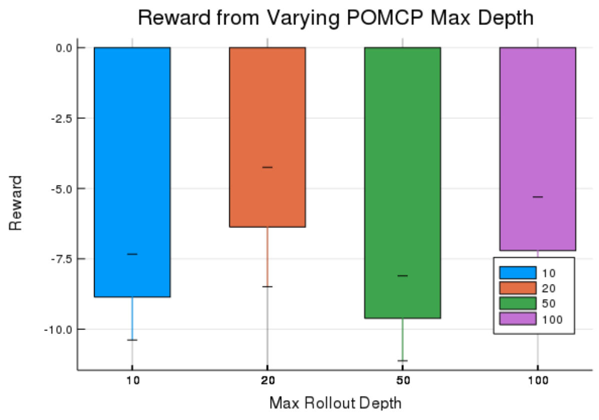


Fig. 3. The effect of varying the max depth for POMCP policies

2) UCB Exploration Constant: The best exploration constant is 1.0, according to Figure 4. This represents the ideal exploration and exploitation constant. Again, we would want to sweep more finely between 1.0 and 5.0 to see if there is an even better exploration constant.

3) Experimenting with Discretization: When experimenting with discretization, we selected various step lengths to sweep over the velocity and angle range. For velocity, we know that the Roomba can choose to either not move at all (which corresponds to a 0) or go at maximum speed (which we map to 10). We also know that the Roomba must be able to turn in any direction, from $-\pi$ to π. Thus, we have our ranges for velocity and omega.

To test how discretization of the velocity range affects POMCP policy results, we held the omega step size constant and swept over a range of values for velocity step size. As we see in Figure 5, there is no

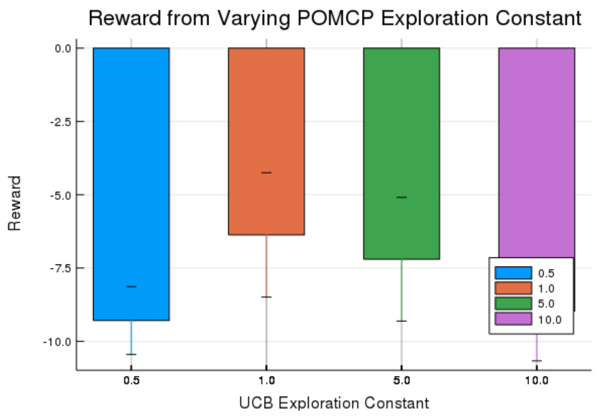


Fig. 4. The effect of varying the exploration constant for POMCP policies

clear optimal velocity step size. We have a few peaks when we discretize the range into 30 and 60 steps, but there is no clear indicator that a finer step size is better.

Next, we examined how omega step size affects POMCP policy reward. Similarly, we held the velocity step size constant and swept over a range of omega step size values. In contrast to velocity, more finely discretizing the angle seems to have a positive influence on average reward (see Figure 5). When the omega value range is divided into 70 steps, the reward can be comparable to the reward we saw using the FIB policy earlier. Thus, it appears that having more options to turn is more important than speed settings.

C. Selecting the Velocity for the Heuristic

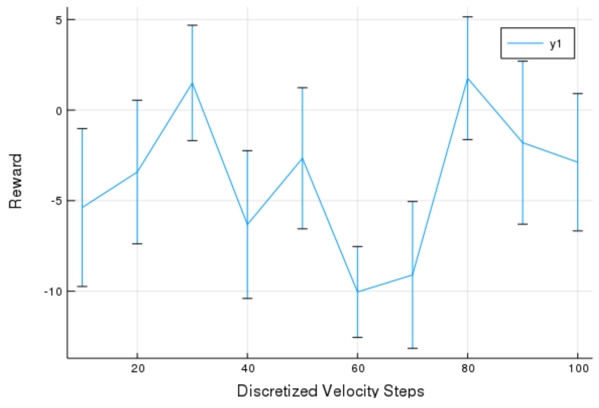
As mentioned above, the heuristic is an important improvement to POMCP performance; in the heuristic policy, we select a velocity within the Roomba's velocity range to use for all heuristic actions. We can tune this velocity to select the one that has the best average performance. See Figure 6 for results from experimenting with various heuristic velocities.

V. DISCUSSION

Although initial results showed that FIB had the highest average rewards, further investigation proved that POMCP can be as good, if not better, than the offline method. A fuller action space allows POMCP to shine where offline methods fall flat. We will discuss some of the tradeoffs between the methods tested.

A. Time

When comparing offline and online methods, time is an important consideration. For offline methods, the majority of processing work is done prior to execution.



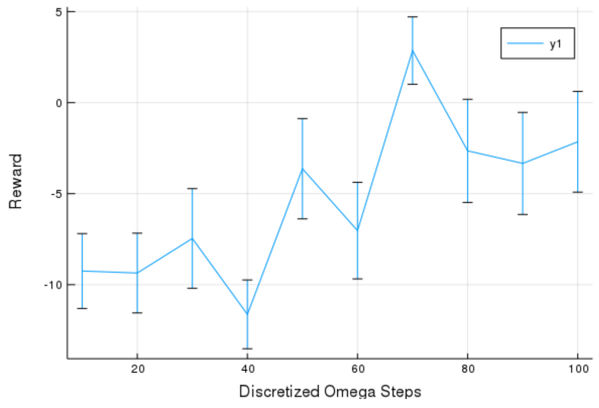


Fig. 5. Top: We sweep over different steps for discretizing velocity values from 0 to 10 (holding the omega steps constant) and get average rewards for the POMCP policy. Bottom: We sweep over different steps for discretizing omega values from -pi to π (holding the velocity steps constant)

Thus, each decision or action of the Roomba occurs with very little lag. Online methods, on the other hand, do the majority of their computation at every time step. With a large state or action space, there can be considerable lag, which would not be suitable for real-time applications. When thinking about the Roomba problem, we ask ourselves: is it acceptable if our Roomba takes a second to think before it moves? Or is time of the essence?

In this instance, our Roomba is in no hurry. But suppose that it is trying to clean as fast as possible and reach the goal. Then we would want to sacrifice some benefits of online methods and choose an offline method.

B. Action and State Space

At the crux of this entire problem is the issue of complexity as the state and action space increases. Even with our large step sizes for the state and action space, the total time evaluating QMDP and FIB models was significantly longer than that of online methods.

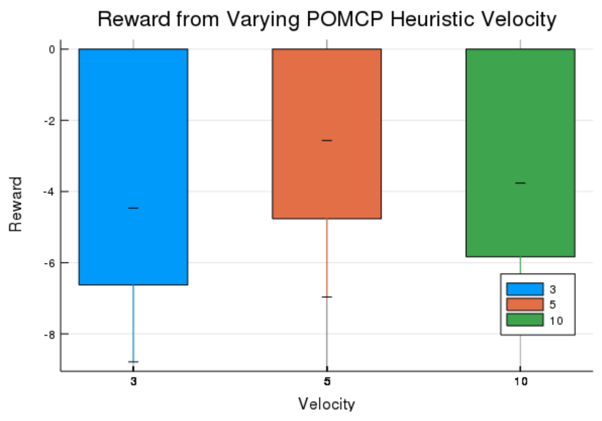


Fig. 6. The effect of varying the velocity for POMCP heuristic policies

As we decrease the step sizes to make the model more continuous and aligned with the original problem, offline methods may take so long to converge that it would be impossible to produce a good policy. Thus, POMCP or another online method would be preferable to offline methods.

VI. FUTURE WORK

Based on our current findings with QMDP, FIB, and POMCP, we are interested in further exploring online methods such as POMCPOW and ARDESPOT. With a continuous action and state space in this Roomba problem, these online methods seem better equipped to combat the problem's complexity. Although we were able to get ARDESPOT to run for several trials, the algorithm did not generalize and failed on specific random seeds. For future work, we would like to isolate and solve why ARDESPOT does not work well consistently and then also survey this algorithm for an additional point of comparison.

Future work might also include making the Roomba problem more involved. For example, our new objective could be to visit as much area in the room before exiting to the goal, just as we would hope a true Roomba would do. Or perhaps we could simulate ”trash” in the room. For a high reward, the Roomba should navigate to pick up the trash before exiting to the goal state.

VII. CONCLUSIONS

In this project, we surveyed various POMDP solvers including QMDP, FIB, and POMCP to understand their abilities and limitations. As with any problem, we must tradeoff certain factors for each method. We found that offline methods, especially Fast Informed Bound, had the highest average reward when all methods were discretized similarly. However, when we tested different action space discretizations, POMCP could reach about the same performance as FIB.

Further testing showed that replacing our initial naive solution with POMCP with a heuristic vastly improved our baseline results. Perhaps more novel solvers like POMCPOW have the potential to improve our results by closely modeling the representation of the problem with continuous state and action space.

In conclusion, there are various factors in choosing a Roomba policy. If we wanted to reduce execution time, we would opt for an offline method as most computation happens prior to running the simulation. However, to ensure that an offline method will converge, we must limit how finely we can discretize the state and action space. If instead we prioritized having a more continuous state or action space, we would want to use online methods such as POMCP and tune parameters to achieve the best performance. Ultimately, our exploration of Roomba policies has given us a greater appreciation and understanding of POMDPs and the challenges and tradeoffs they face.

ACKNOWLEDGMENT

We would like to thank Jeremy and Louis for their mentorship throughout this class and project via office hours and problem sessions. In addition, Jayesh Gupta helped us debug a Julia FIB version issue. We would also like to thank Professor Mykel Kochenderfer for a phenomenal class that equipped us with the understanding and tools necessary to delve into decision making under uncertainty.

We would also like to acknowledge that this project idea and backbone starter code was provided by the CS238 teaching staff.

Ode to Roombas

From POMCP to the decision tree,
We know how to deal with uncertainty.
With QMDP or even FIB,
Begone, indecision! We say with glee.
So many distributions, but we like Dirichlet.
It helps us decide what to do everyday.
Ranging from roombas to playing catch at night,
The problems we've solved have been such a delight.
And with that we must say goodbye 228,
We won't forget Bayes Rule, we promise, it's great!

REFERENCES

[1] Silver, David, and Joel Veness. ”Monte-Carlo planning in large POMDPs.” Advances in Neural Information Processing Systems vol. 23 (NIPS), 2010.

[2] ”DESPOT: Online POMDP Planning with Regularization”, JAIR vol. 58 (NIPS), pp. 231-266, 2017.

[3] C. Papadimitriou and J. Tsitsiklis, “The Complexity of Markov Decision Processes,” Mathematics of Operation Research, vol. 12, no. 3, pp. 441–450, 1987. doi: 10.1287/moor.12.3.441.

[4] M. Hauskrecht, “Value-Function Approximations for Partially ObservableMarkov Decision Processes,” Journal of Artificial Intelligence Research, vol. 13, pp. 33–94, 2000. doi: 10.1613/jair.678.

[5] G. Shani, J. Pineau, and R. Kaplow, “A Survey of Point-Based POMDP Solvers,” Autonomous Agents and Multi-Agent Systems, pp. 1–51, 2012. doi: 10.1007/s104 58-012-9200-2.

[6] H. Kurniawati, D. Hsu, andW.S. Lee, “SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces,” in Robotics: Science and Systems, 2008.

[7] T. Smith and R.G. Simmons, “Heuristic Search Value Iteration for POMDPs,” in Conference on Uncertainty in Artificial Intelligence (UAI), 2004.

[8] “Robotic Industries Association.” Robotics Online, www.robotics.org/service-robots/customer-service-robots.