Simultaneous Localization and Mapping: A Simulated Environment

Isaac Lee
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA, United States of America
ilee05@stanford.edu

David Li
Department of Computer Science
Stanford University
Stanford, CA, United States of America
davidwl@cs.stanford.edu

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a core problem in robotics in which an autonomous agent must estimate its own position while simultaneously constructing a map of an unknown environment. In this work, we formulate a simplified SLAM problem as a Partially Observable Markov Decision Process (POMDP) in a discretized 10-by-10 grid world containing sparse, fixed landmarks and noisy lidar-based observations. To address the computational intractability of exact planning in POMDPs, we implement an online planning algorithm based on lookahead with rollouts, which approximates optimal action selection through simulation over a finite horizon. We evaluate the performance of this approach against a random-action baseline across 100 time-steps. These results demonstrate that the planning-based policy consistently converges to accurate estimates of both agent position and landmark locations, achieving a significant improvement in total reward and landmark estimation error compared to the baseline. These results suggest that lookahead with rollouts provides an effective and computationally efficient solution for online planning in simplified SLAM environments, and offers a promising framework for extension to more realistic and stochastic settings.**

***Index Terms*—SLAM, simulated environment, robotics**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is the problem in which an autonomous agent attempts to estimate its pose within an environment while concurrently trying to create a map of its surroundings. In a typical SLAM algorithm, the agent localizes itself based on landmarks it observes in its environment, repeatedly adjusting the estimated poses of both itself and the landmarks based on data from sensors such as cameras, radar, and lidar. Because SLAM algorithms are commonly deployed in domains such as self-driving cars, unmanned aerial vehicles, and autonomous robotics, a successful SLAM algorithm will minimize the error between the agent's internally stored pose and map of the environment with the ground truth.

In this paper, we model a simplified version of the SLAM problem as a Partially Observable Markov Decision Process (POMDP) in which a simulated autonomous agent navigates a discretized 2D environment with sparse landmarks. The agent moves around the map and uses simulated lidar measurements to both localize itself and identify the locations of the landmarks. As is typical of SLAM problems, our goal is to minimize the error between the estimated agent and landmark positions with their real values. To do this, we generate policies using lookahead with rollouts and compare their performance to a random benchmark policy.

II. Prior Work

The SLAM problem has been widely studied over the past few decades and extensive literature exists on both formulations of the problem as well as potential solutions. Here, we outline relevant details from a few pieces of prior work that provided guidance and inspiration for this paper.

When deploying SLAM algorithms, the following quantities are usually defined [1].

- $x_{0:k}$: the position of the agent at each timestep from $t = 0$ to $t = k$.
- $u_{0:k}$: the control inputs (movement commands) at each timestep from $t = 0$ to $t = k$.
- m: the map (set of landmarks).
- $z_{0:k}$: the measurements the agent receives (typically of landmarks).

Typically, the relationships between these different quantities are represented using the below conditional probabilities [1]:

- $P(x_k, m|z_{0:k}, u_{o:k}, x_o)$: the probability that the agent is at position x_k and the map is configured as described in m given the received measurements and control inputs up to $t = k$, as well as the agent's initial position.
- $P(z_k|x_k, m)$: the probability of observing measurement z_k, given the agent position x_k and map m at time $t = k$.
- $P(x_k|x_{k+1}, u_k)$: the probability of the agent being at position x_k, given it was at position x_{k-1} and received control input u_k.

The SLAM problem described by the probability distributions above is known as the "online SLAM problem," in which we estimate our belief in the agent's momentary position. It is also possible to model the "full SLAM problem," in which the posterior for the agent's whole trajectory is calculated [2]. Our simulated SLAM problem will be largely akin to the online version of the problem in order to limit computational cost.

III. Problem and Model

We choose to model a simplified version of the online SLAM problem as a POMDP. Below, we include descriptions

of the state and action spaces, the transition and observation models, the reward function, and other relevant details.

A. State Spaces

Our agent operates in an enclosed 10-by-10 2-dimensional grid world, meaning there are 100 positions indexed between $(0, 0)$ and $(9, 9)$. 4 of the cells $((2, 2), (3, 6), (6, 3), (7, 7))$ are occupied by landmarks and cannot be traversed by the agent, which starts in a random grid cell not occupied by a landmark. We create two state spaces for our problem formulation: $\mathcal{S}_\mathcal{P}$ for the agent position and $\mathcal{S}_\mathcal{L}$ for the landmark positions. States will be indexed by their x and y coordinates; an example agent position state is $s_{p_{6,6}}$ and the landmarks are in states $s_{l_{2,2}}, s_{l_{3,6}}, s_{l_{6,3}}$, and $s_{l_{7,7}}$. Fig. 1 illustrates the locations of the landmarks and an example agent starting position, as well as how the x and y coordinates correspond to positions in the grid.

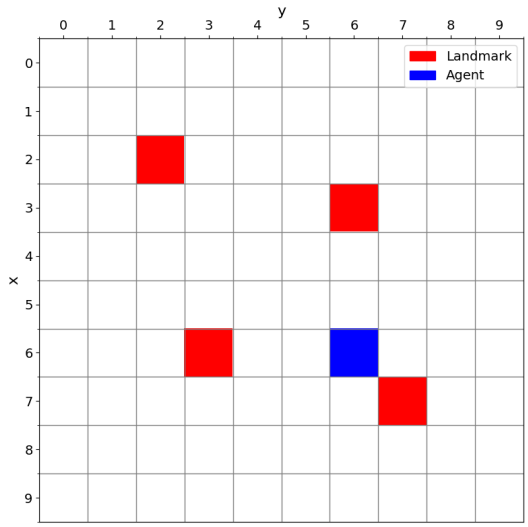

Fig. 1. 10-by-10 grid world before any agent movement if the agent starts at (6, 6).

Because we are modeling the SLAM problem as a POMDP, we also store the agent's belief distributions as two 10-by-10 arrays whose entries are nonnegative and sum to 1. We use one array to store the belief distribution of the agent's position, and one array to store the belief distribution of the four landmark positions. Since the agent has no knowledge of its environment before any measurements are taken, we initialize both with a uniform prior; for all possible x, y coordinate pairs between $(0, 0)$ and $(9, 9)$, we begin with $b(s_{p_{x,y}}) = b(s_{l_{x,y}}) = 0.01$.

Note that our state space and belief distribution for the landmark positions differs from traditional state spaces and belief distributions in that we allow multiple states to be occupied at once and we maintain a single distribution for four different variables. This was done to reduce computational cost and emphasize the fact that there is no inherent order to which the agent perceives the landmarks. However, this also means that the agent can only possess relatively lower beliefs in the presence of a landmark at a given position, compared to beliefs of its own position.

B. Action Space and Transition Model

Because the landmarks are stationary, our model of the SLAM problem only has an action space and transition model associated with $\mathcal{S}_\mathcal{P}$. Our action space $\mathcal{A}$ contains four actions: moving one grid cell North (a_0), South (a_1), West (a_2), or East (a_3).

Transitions are deterministic, meaning the agent moves exactly as it is controlled with probability 1. However, in the case where moving at it is commanded would cause the agent to collide with a landmark or move off the grid, the agent stays in its current position. We do this because we assume that the grid is surrounded by walls on all sides, and the agent will try and fail to drive into a wall or landmark when it is controlled to do so.

C. Observation Models

The agent makes observations of its environment through simulated lidar readings in each of the four cardinal directions. Each lidar reading is implemented as a virtual ray that emanates from the agent in one of the four directions until it reaches either a landmark or an edge (wall) of the grid world. The agent uses the lidar readings that hit the walls to update its belief of its own position, then uses any lidar readings that hit landmarks to update its belief of the overall map layout. Because we have positioned the landmarks such that none of them share a row or column of the grid, at least one horizontal lidar and one vertical lidar will reach a wall, ensuring that the belief distribution of the agent position gets fully updated at every reading.

Since we use the simulated lidar to observe both the position of the agent and the positions of the landmarks, we also have two observation spaces: $\mathcal{O}_\mathcal{P}$ for the agent position and $\mathcal{O}_\mathcal{L}$ for the landmark positions. Similar to states, observations are also indexed by their x and y coordinates; for example, $o_{p_{6,6}}$ means that the lidar reading indicates the agent is in position $(6, 6)$ while $o_{l_{2,2}}$ means that the lidar detects a landmark at position $(2, 2)$.

Our observation model for $\mathcal{O}_\mathcal{P}$ is formulated as follows: given a lidar estimate of the agent's x or y coordinate, we assign a probability of 0.8 that the lidar estimate of the coordinate is equal to the true value, and a probability of 0.1 each that the lidar reading is one tile too low or one tile too high in its estimate of the coordinate. Lidar observations of the agent's position are taken in both the horizontal and vertical directions; the probabilities of the lidar estimates being inaccurate in either direction are considered independent. Fig. 2 illustrates how the observation probabilities for each potential agent position are evaluated when the agent is at position $(6, 6)$.

Our observation model for $\mathcal{O}_\mathcal{L}$ is formulated in a similar fashion, except we assume the agent is currently at the position with the highest belief and extrapolate our estimated lidar positions off of that assumption. For each of the lidar readings

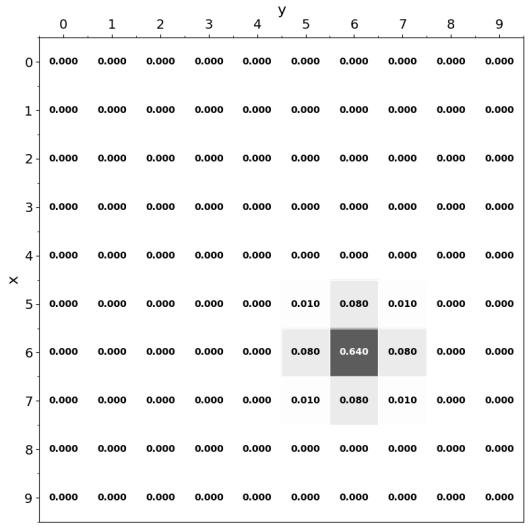

Fig. 2. Observation probabilities for each potential agent position when the agent is at (6, 6).

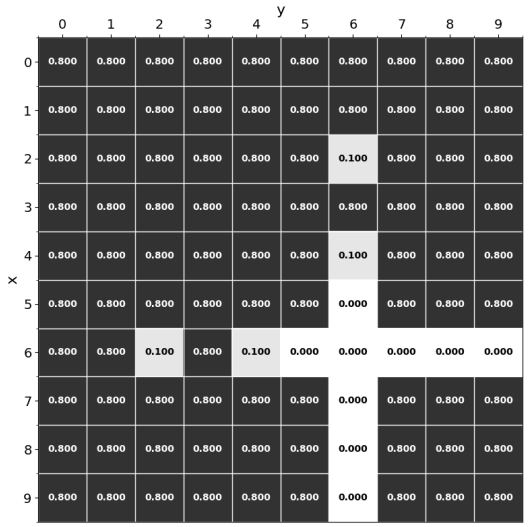

Fig. 3. Observation probabilities for each potential landmark position when lidar readings are taken from (6, 6). The lidar hits the landmarks at positions (3, 6) and (6, 3).

that emanate from the agent in one of the four cardinal directions and hit a landmark, we assign a probability of 0.8 that the lidar estimate of the landmark location is equal to the true value, and a probability of 0.1 each that the lidar reading is one tile too low or one tile too high in its estimate of the location along the axis it emanates across. We assign a probability of 0 for any other grid cells that the lidar readings pass through to indicate our confidence that these cells are empty. For all remaining cells in the grid, we assign an observation probability of 0.8. Although the lidar does not provide any evidence against the possibility of a landmark being located in those cells, we select a decay factor of 0.8 to represent the fact that cells that have not been confirmed to contain a landmark despite many lidar scans are less likely to have one. Fig. 3 illustrates how the observation probabilities for each potential landmark position are evaluated when the agent is at position $(6, 6)$.

The one exception to the above rules is if the lidar observes that a measured position is along the edge of the grid world (i.e. one of the indices is 0 or 9). In this scenario, we only have one case in which the lidar estimate is off by one grid cell since the other case would cause the potential position to move off the grid, and the 0.1 probability that would have been given to the other case is "folded back" into the observation probability that the lidar estimate is correct.

D. Belief Updates

After each lidar observation is taken, we use simplified versions of the standard discrete state filter to update our belief distributions over the possible agent and landmark positions, as shown below in Expressions (1) and (2). This is done to reflect the reduced sources of uncertainty in our modeling of the SLAM problem as a POMDP. Note that $s_p^* = \arg\max_{s_p} b'(s_p)$, reflecting the fact that the estimations of the landmark positions assume that the agent is located at the position with the highest belief, as explained above.

$$b'(s'_p) \propto O_P(o_p|s'_p) \sum_{s_p} T(s'_p|s_p, a) b(s_p) \tag{1}$$

$$b'(s_l) \propto O_L(o_l|s_p^*, s_l) b(s_l) \tag{2}$$

After updating our belief distributions, we normalize the beliefs so that the sum of the beliefs over possible agent positions and possible landmark positions each sum to 1.

E. Reward Function

For the reward function, we would like to penalize belief distributions that do not accurately reflect the true locations of the agent or landmarks. We would also like to penalize the agent for colliding with landmarks. Therefore, for a given belief b and agent position state $s_{p_{x,y}}$, we define the reward $R(b, s_{p_{x,y}})$ as shown in Equation (3), where the variable collision $= 1$ if the latest action caused the agent to collide with a landmark and collision $= 0$ if not.

$$\begin{aligned} R(b, s_{p_{x,y}}) = -10000 \Bigg(& (\text{collision}) + \big((1 - b(s_{p_{x,y}})\big)^2 \\ & + \big(0.25 - b(s_{l_{2,2}})\big)^2 + \big(0.25 - b(s_{l_{3,6}})\big)^2 \\ & + \big(0.25 - b(s_{l_{6,3}})\big)^2 + \big(0.25 - b(s_{l_{7,7}})\big)^2 \Bigg) \end{aligned} \tag{3}$$

We use the value 0.25 because in an ideal belief distribution over the possible landmark locations, we would have a belief of 0.25 for the actual positions of the four landmarks and a belief of 0 for all other grid cells. This would reflect the fact that we are equally confident in the positions of all four

landmarks, while adhering to the fact that all the beliefs over the potential landmark locations must sum to 1.

IV. Approach

Though fully solving this simplified SLAM problem is theoretically possible, it is computationally inefficient. Instead, if we planned only a limited depth into the future from the current belief state, the agent could make effective decisions in real time while avoiding the exponential complexity associated with solving the full POMDP exactly. As a result, we choose to implement a lookahead with rollouts algorithm, which approximates the optimal action at each time-step by simulating possible actions.

Lookahead with rollouts is a simple online strategy that involves choosing a greedy action with respect to values estimated through future simulation to an arbitrary finite depth [3]. In order to simulate our problem, we generate rollout policies by randomly sampling from our belief state and our possible actions. Our algorithm combines these rollout policies with one-step lookahead to approximate a more optimal action in the current state. We include the following pseudocode to better illustrate our implementation where $\mathcal{K}$ and $\mathcal{H}$ are hyperparameters that represent the number of simulations we execute and the depth to which we simulate, respectively. When running the algorithm as described in the below experiments, we set values of $\mathcal{K} = 5$, $\mathcal{H} = 5$, and a discount factor of $\gamma = 0.95$.

Algorithm 1 Lookahead with rollouts

```
for action in A do
    for s in range(K) do
        start ← sample(pos_belief)
        next ← simulate(start, action)
        for d in range(H) do
            next ← simulate(next, random(A))
        end for
    end for
end for
```

V. Analysis and Results

To test the effectiveness of our approach, we ran our algorithm and compared it to a simple baseline that generates random moves at each time-step. Both algorithms were run for 100 time-steps starting from the same randomized position.

A. Baseline: Randomized Actions

After 100 time-steps, unsurprisingly, the random policy was not effective at solving our problem. The final position of the agent can be seen in Figure 4. The randomized policy performs well in approximating the position of the agent with the final position belief state shown in Figure 5. However, when it comes to approximating the position of the landmarks, the random policy performs significantly worse, unable to even identify that there are four different landmarks as shown in Figure 6.

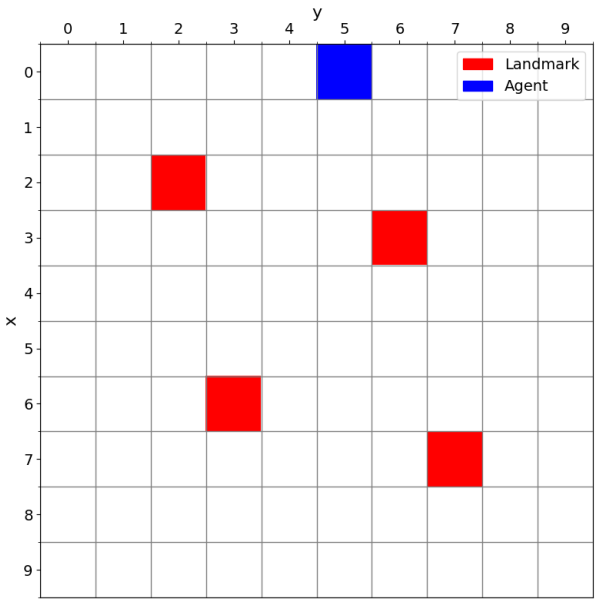

Fig. 4. Random policy: final agent position

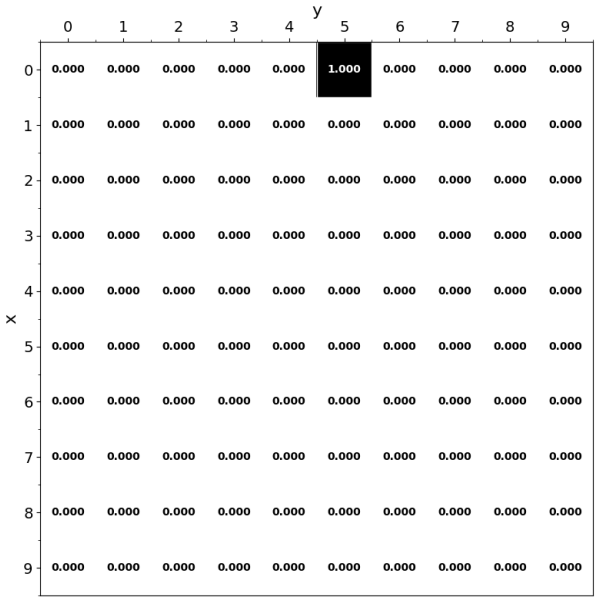

Fig. 5. Random policy: final agent position belief distribution

B. Planning: Lookahead with Rollouts

After 100 time-steps, our planning algorithm was able to effectively solve our problem in an efficient and accurate manner. Our algorithm was able to converge to an optimal solution within 60 steps. The final position of the agent can be seen in Figure 7. Our planning algorithm also correctly identifies the position of the agent with extremely high precision as shown in Figure 8 and is able to correctly identify the locations of the landmarks as shown in Figure 9, a significant improvement compared to the baseline policy.

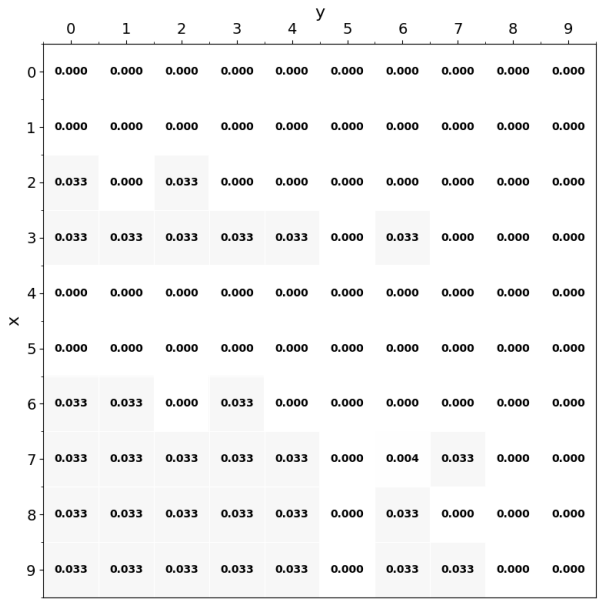

Fig. 6. Random policy: final landmark position belief distribution

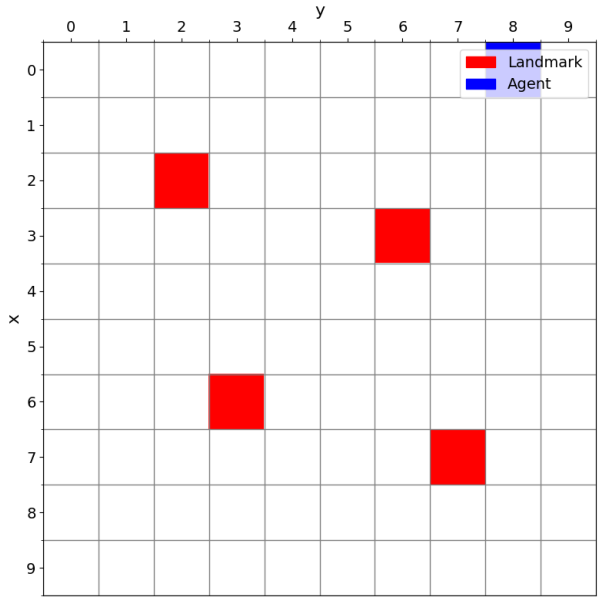

Fig. 7. Planning policy: final agent position

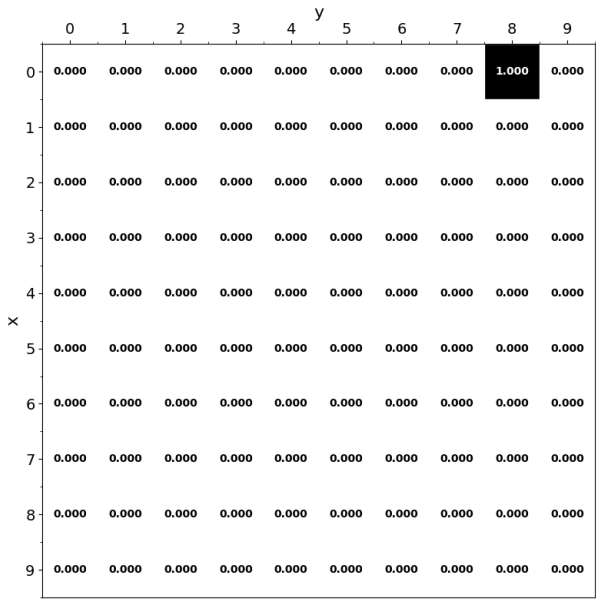

Fig. 8. Planning policy: final agent position belief distribution

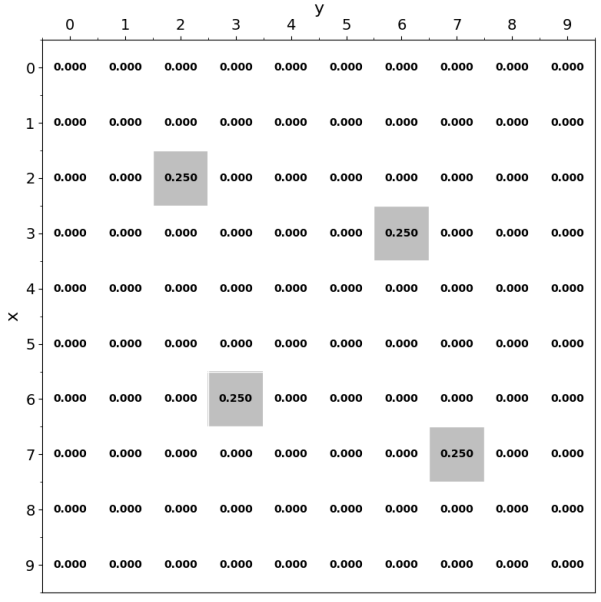

Fig. 9. Planning policy: final landmark position belief distribution

C. Comparison

Table I below includes more running metrics for both policies: the total reward across all 100 time-steps, the number of landmark collisions, and well as the final agent position error (defined in Expression (4)) as the final landmark position error (defined in Expression (5)). As evident, there is a significant decrease in the total reward ($+59.58\%$) and the final landmark position error ($+100.00\%$) of the planning policy as compared to the random one. However, we observe no decrease in the number of collisions; this is likely not because of a flaw in our lookahead with rollouts algorithm, but instead due to the fact that the random policy generated in this experiment happened to result in only 1 collision.

TABLE I

Metric	**Random Policy**	**Planning Policy**
Total Reward	-194484	-78601
Collisions	1	1
Final Position Error	0	0
Final Landmark Error	1880	0

$$-10000\big((1-b(s_{p_{final}})\big)^2 \tag{4}$$

$$\begin{aligned}-10000\Bigg(&\big(0.25-b(s_{l_{2,2}})\big)^2+\big(0.25-b(s_{l_{3,6}})\big)^2\\&+\big(0.25-b(s_{l_{6,3}})\big)^2+\big(0.25-b(s_{l_{7,7}})\big)^2\Bigg)\end{aligned} \tag{5}$$

To prove the consistency of the lookahead with rollouts algorithm in generating robust policies, Table II includes the same metrics averaged over 10 runs for both types of policies. Again, we observe a significant decrease in the average total reward ($+67.68\%$) and final landmark position error ($+96.52\%$) of the planning policies as compared to the random one. We also observe a $+100.0\%$ decrease in the average number of collisions, confirming that the earlier run of lookahead with rollouts was likely an aberration and that our algorithm really does generate policies that reduce collisions with obstacles as desired.

TABLE II

Metric	Random Policy	Planning Policies
Average Total Reward	-180176	-58227
Average Collisions	3.50	0.00
Average Final Position Error	0	0
Average Final Landmark Error	1318	46

VI. Conclusion

In this paper, we demonstrated the effectiveness of a lookahead with rollouts algorithm in generating policies that optimize for map and localization accuracy in a simplified SLAM problem. Despite the inherent paradoxical nature of SLAM due to the fact that the agent must locate its position in an environment that it is concurrently trying to map out, the implemented algorithm proved to be a computationally inexpensive solution that was able to correctly identify the location of the agent and landmarks with comparatively insignificant error. Overall, we have shown that lookahead with rollouts is a viable solution in a situation for which an autonomous agent must judiciously select actions in real time with the goal of precisely identifying its own location as well as the layout of the environment in which it operates.

As noted earlier, we observed an instance where a policy generated by the lookahead with rollouts algorithm did not present a decrease in the number of times the agent collided with a landmark as compared to the random policy, despite the additional negative reward resulting from landmark collisions. Even though our subsequent tests indicated that this was likely an anomaly and lookahead with rollouts is able to reliably produce policies that avoid collisions, we may increase the negative reward for collisions and observe if this causes any further improvements in the future.

There are inherent limitations in the work presented above, the most obvious being our use of a simulated environment for testing. Real-world environments in which SLAM algorithms need to be deployed are often not as straight-forward as a simple grid and may contain unforeseeable challenges in problem formulation that we did not account for here. Given more time and computational resources, we may relax some of these simplifications in the future to test if our lookahead with rollouts algorithm can still accommodate a more intricate problem formulation. Increasing the number of lidar directions, making our transition model stochastic instead of deterministic, randomizing landmark locations, increasing grid-size, and removing the assumption that the agent is located at the position with the highest belief when estimating landmarks are all examples of iterative degrees of complexity we might add.

Contributions

Isaac Lee worked on the problem formulation, project proposal and update, researching prior work, designing and implementing the POMDP model, and writing up the final report. David Li worked on the problem formulation, project proposal, project update, implementing the online planning solution, and writing up the final report. We also used Cursor, ChatGPT, and Claude to generate and debug code, as well as to create the figures in this paper.

References

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99–110, June 2006, doi: 10.1109/MRA.2006.1638022.

[2] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. Cambridge, MA, USA: The MIT Press, 2005.

[3] M. Kochenderfer, T. Wheeler, and K. Wray, Algorithms for Decision-Making. Cambridge, MA, USA: The MIT Press, 2022.