Exploring Localization and Pathfinding with Monte Carlo and A* Algorithm

Frances Raphael
Stanford University
fraphael@stanford.edu

December 1, 2024

1 Introduction

This project combines two essential robotics concepts—localization and pathfinding—into a single framework. Localization helps a robot estimate its position within an environment, while pathfinding enables it to plan and navigate a path to a specified goal. These capabilities are crucial for autonomous systems operating in dynamic or unfamiliar settings.

The simulation uses a grid-based environment where the robot employs Monte Carlo localization to determine its position, even with noisy sensor data. Particle filtering refines this estimate by maintaining and updating a set of weighted particles. For navigation, the A* algorithm computes the shortest path to the goal while avoiding obstacles. Visualized using Pygame, the system showcases real-time localization, pathfinding, and obstacle interaction. Details of the implementation are provided below.

2 Project Breakdown

2.1 Grid-based Game Setup

I began by creating a grid-based environment in Pygame, where each cell could be empty, contain an obstacle, or represent the goal. This approach was inspired by the robot paths question in problem set 1. To enhance the visual aspect, I sourced icons for the robot, obstacle, and goal from the internet, which allowed me to set up the graphical representation of the grid and its elements.

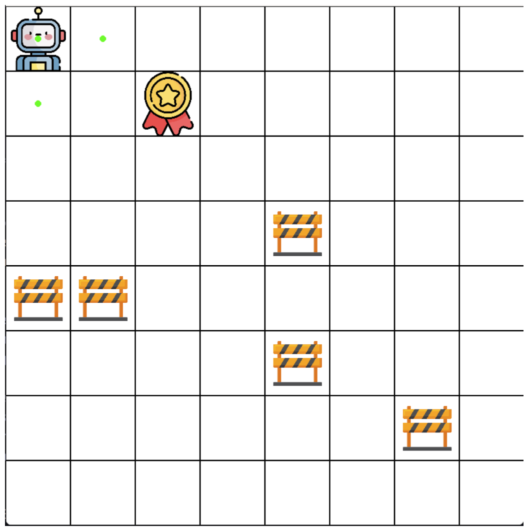
Figure 1: Pictured above is the grid with beautiful icons

2.2 Monte Carlo Localization

Monte Carlo Localization (MCL) is implemented using a particle filter to estimate the robot's position within the grid. The key steps in the algorithm are initialization, motion update, measurement update, and resampling. Each particle represents a possible state of the robot, defined as $\mathbf{s}_t = [x, y]$, where x and y are the robot's coordinates on the grid at time t. Check the appendix for the rest of the derivations.

2.3 A* Pathfinding Algorithm

The A* algorithm is implemented to navigate the robot from its current position to the goal. The algorithm uses a heuristic based on Manhattan distance and explores the grid to find the shortest path. Once the path is found, the robot follows it autonomously by selecting the next step along the path.

2.4 Probability

To visualize the localization process, I generate probability distributions of particle positions using both 2D and 3D plots. The 2D plot illustrates the likelihood of the robot being at each grid cell, while the 3D surface plot offers a more detailed view of the particle distribution as it evolves over time. The plots below demonstrate the Bayesian probability of the robot's location, providing a clearer understanding of its position within the grid.

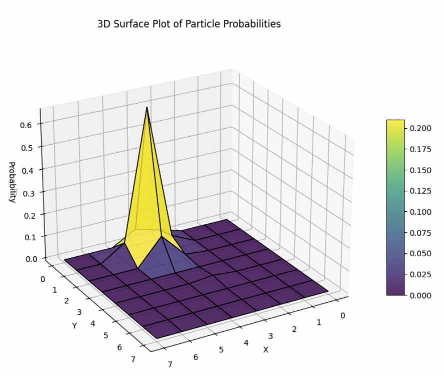

Figure 2: 3D plot showing the probability distribution of the robot's location.

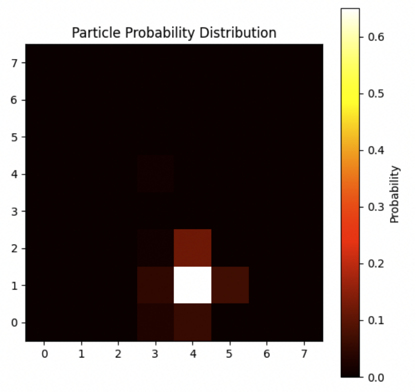

Figure 3: 2D heatmap displaying the likelihood of the robot's position in the grid.

3 Conclusion

This project was a fun and rewarding exploration of robotics and probability. By combining Monte Carlo localization and A* pathfinding, I created a robot that autonomously navigates a map, estimating its position while avoiding obstacles. It was incredible to see how probability, particularly Monte Carlo localization, enables a robot to track its location and navigate dynamically. The visualizations helped clarify the robot's process of localization and navigation. Future improvements could include adding dynamic obstacles, experimenting with other localization algorithms, or enhancing decision-making for more complex environments.

4 Appendix

Initialization

We initialize N particles, $\{\mathbf{s}_t^{[i]}\}_{i=1}^N$, uniformly distributed across the grid. Each particle's initial weight is assigned uniformly:

$$w_t^{[i]} = \frac{1}{N}, \quad \forall i = 1, 2, \ldots, N.$$

This ensures an uninformative prior, as the robot's position is equally likely anywhere in the grid at the start.

Motion Update

At each timestep, particles are updated according to a motion model. Given a movement $\mathbf{u}_t$, which corresponds to one of four possible grid movements ($[\pm 1, 0], [0, \pm 1]$), the state of each particle is updated:

$$\mathbf{s}_{t+1}^{[i]} = \mathbf{s}_t^{[i]} + \mathbf{u}_t + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ represents motion noise.

The motion is constrained by the grid boundaries and obstacles:

$$\mathbf{s}_{t+1}^{[i]} \in \{(x, y) \mid 0 \leq x, y < \text{GRID\_SIZE and } (x, y) \notin \text{obstacles}\}.$$

Measurement Update

For each particle, a weight is computed based on the likelihood of the robot's actual position given the particle's state:

$$w_{t+1}^{[i]} \propto P(\mathbf{z}_{t+1} \mid \mathbf{s}_{t+1}^{[i]}),$$

where $\mathbf{z}_{t+1}$ is the observation (e.g., the robot's measured position). A simple measurement model compares the particle's state to the robot's actual position:

$$P(\mathbf{z}_{t+1} \mid \mathbf{s}_{t+1}^{[i]}) = \begin{cases} \alpha, & \text{if } \mathbf{s}_{t+1}^{[i]} = \text{robot\_pos}, \\ \beta, & \text{otherwise}, \end{cases}$$

where α is a high likelihood (e.g., $\alpha = 1$) and β is a low likelihood (e.g., $\beta = 0.1$).

The weights are normalized to ensure they form a probability distribution:

$$w_{t+1}^{[i]} = \frac{w_{t+1}^{[i]}}{\sum_{j=1}^N w_{t+1}^{[j]}}.$$

Resampling

After updating the weights, particles are resampled using a multinomial distribution to focus on higher-weight particles. The probability of selecting particle $\mathbf{s}_{t+1}^{[i]}$ is proportional to its weight:

$$P(\text{select } \mathbf{s}_{t+1}^{[i]}) = w_{t+1}^{[i]}.$$

Convergence

As the algorithm progresses, the particles converge toward the robot's true position. The expected position of the robot can be estimated as the weighted mean of the particles:

$$\hat{\mathbf{s}}_{t+1} = \sum_{i=1}^{N} w_{t+1}^{[i]} \mathbf{s}_{t+1}^{[i]}.$$