Particle Filter Localization with MCMC

Andrew Luo Huang, Parth Patel, Adnan Sherief, Albert Wilcox, Brian Yu

December 6, 2019

1 Introduction

Inference and estimation over large state spaces play a crucial role in many fields of engineering, ranging from Robotics to Information Theory. We decided to use the Metropolis-Hastings algorithm to localize an agent in a large, fine grid of space and walls. We demonstrate that the agent's position can be quickly determined and continuously tracked with high accuracy. A demonstration of our algorithm can be found at `https://www.youtube.com/watch?v=TyqL8eDl5uo`.

2 Methods

The setup of our localization problem is as follows. The agent's world consists of a grid of variable size, where every location in the agent's world is a square within the grid. Each of these square contains either a free space where the agent can move to or a wall where the agent can't move to. At each time step, our agent travels in a random direction for a given distance plus binomial error centered around 0, stopping if it hits a wall.

We use a particle filter to determine the probability distribution of the agent's location. We initialize a fixed number of particles uniformly on the grid. At each time step, we sample new particles by applying the agent's movement function to each particle.

Then, we use eight noisy sensors to measure the agent's distance to walls along cardinal and ordinal directions. We use these observations to weight particles. The general particle filtering algorithm weights each particle by the its likelihood in the target distribution divided by the probability of proposing it. However, because we are in the special case where our target distribution is

$$\mathbb{P}(observations \mid new\_location) * \mathbb{P}(new\_location \mid old\_location)$$

and our proposal distribution is

$$\mathbb{P}(new\_location \mid old\_location)$$

we can simply weight each particle by the probability of seeing our observations at that location, which is the product of the probabilities of seeing the sensor

readings. However, we also chose to smooth our weight according to the parameter sensor_trust, a measure of the agent's fidelity of its sensor readings. In the event that our sensor readings are extremely off, we don't want to eliminate likely positions due to our poor sensor readings i.e. all our sensor readings are taken with a grain of salt.

Lastly, we re-sample particles using MCMC over the weighted distribution, maintaining our total probability mass and clustering our particles in regions of high probability.

3 Experiments

After deciding on the more detailed graph style, we wanted to see how our simulation would perform with different hyperparameter configurations, so we decided to vary these and compare its performance. To quantify performance, a given hyperparameter configuration was run ten times, where each run went for 30 iterations and counted for how many of those all the particles were within 30 squares of the agent. This gave a good indicator for the success of the algorithm in tracking the agent for that run.

The hyperparameters we experimented on were sensor noise, movement noise, and sensor trust. Sensor and movement noise were parameters passed into the binomial distribution adding noise to the agent's observations and actions where higher values mean noisier systems. Sensor trust was a parameter the agent used to determine how much it should trust its sensors, where 1 means complete trust and 0 means no trust.

To test, we fixed the other hyperparameters and varied the one we were testing on. The defaults we used for the ones not being tested are $sensor\_noise = 8$, $movement\_noise = 12$ and $sensor\_trust = 0.9$.

4 Results and Analysis

The outcome from the experiments can be seen in figures 1, 2, and 3 below.

As the experiments show, our simulation performed very well. For situations that had reasonable amounts of noise, it managed to correctly approximate the position about 80% of the time.

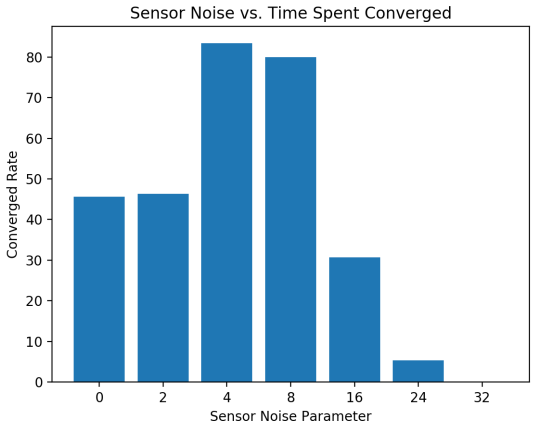

Figure 1: Graph showing how percentage of time spent converged varies with sensor noise. As the noise for our observations increases, we get increasingly inaccurate measurements, making localization more challenging. Interestingly, our agent performs relatively poorly when we have low sensor noise, indicating that the quality of our model of sensor noise heavily impacts our agent performance.

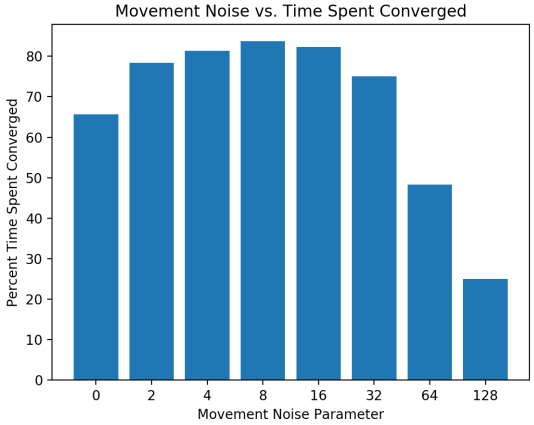

Figure 2: Graph showing how percentage of time spent converged varies with movement noise. Since our performance remains relatively in the same ballpark for variations of our movement noise parameter, our model of movement noise does not critically impact our ability to localize, implying that the sensor noise is the more impactful parameter. As movement noise increases to high values compared to our board size, our agent understandably performs poorly, as its movements now result in extremely large changes in position, making localization extremely difficult.

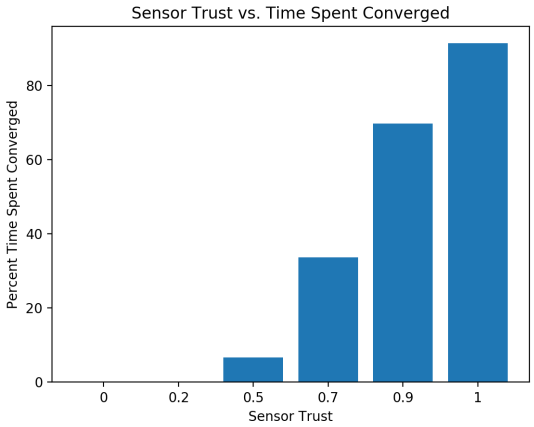

Figure 3: Graph showing how percentage of time spent converged varies with trust. Since this was run with relatively low noise, it did best when it completely trusted its sensors. Low values of sensor trust yield nearly uniform probabilities, making our observation probabilities nearly random, explaining why we have no convergence at all at the lower end of the graph.

5 Limitations

Particle filters suffer if the sensor readings are uninformative, either due to the layout of the map or due to poor exploration of the map. If our agent repeatedly moves in the same general area it started in, the observations it collects are not distinct enough to differentiate its current position from other areas that happen to have similar surrounding wall characteristics.

To deal with these uninformative situations, we can resample our particles only when our observations are different enough, allowing us to avoid converging to incorrect locations and maintaining a balanced sample of particles. Another solution is to simply add more particles, but after a certain number of additional particles, our particle filter will be no more efficient than maintaining a belief distribution over our entire board. Finally, we can resample from the entire distribution over our entire board, not just over our current distribution of samples, with smoothing in place.

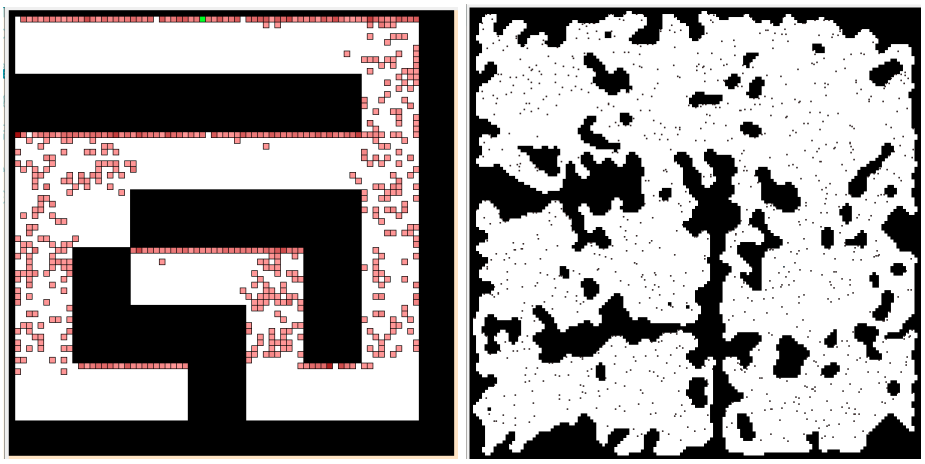

Figure 4: The maze (left) is an example of a layout of the map that produces uninformative sensor readings. Most of the corridors in the maze yield similar noisy distance observations, making it hard to pinpoint the location of the agent. Compare this to the highly differentiable and organic cave system (right), where each position is highly differentiable from other positions.

6 Conclusion

This project was an interesting first experience into the world of Markov Chain applications. We found it interesting that with no knowledge about the environment and little trust in the sensors of the agent, we were still able to find its exact position after a finite number of iterations. We hope to continue to experiment with our current model and add other features, such as mapping.

Mapping is an interesting problem that is surprisingly similar to the localization problem, except now our samples would reflect the presence of a wall at a certain square rather than the agent position!