Report for Assignment 2, EDAP01

Sophia Kovalenko [so4816ko-s]

21 February 2024

1 Introduction

This report is about the second assignment for the class Artificial Intelligence, EDAP01. I will first summarise the task, explain the models given, evaluate and discuss the results of my implementation and finally discuss the relation between an article about Monte Carlo Localisation and my implementation.

2 Statement

The peer review was made with my fellow partner Vivian Bai [vi0713ba-s]. We also discussed the assignment together during the implementation phase.

3 Summary of the task

In this assignment, the task was to implement the theory on probabilistic reasoning over time using Hidden Markov Models (HMM) and filters. The task consisted of estimating the position of a simulated robot which moves in a grid without any landmarks. The models (observation, transition and state) as well as the simulation of the robot were given by the course staff. The goal was to implement a HMM, with a forward filtering approach first. The last task consisted of using a smoothing algorithm to improve the estimated localisation.

4 Changes in implementation after the peer review

Vivian's review of my implementation can be found in the Appendix.
My implementation was a working one at the time of the peer review, and the only implementation error she pointed at was about the main method in the Jupyter Notebook. I was in fact not averaging the Manhattan distances while going over the 500 steps in the graphs. I was simply plotting the errors in Manhattan distance at each step, which did not show any useful information like trends.

5 Explanation of the models

5.1 Transition model

The transition model contains the transition matrix. This matrix represents the probability of the next state, given the previous state: $P(S_i|S_{i-1})$. Here, it is the robot's step direction which is encoded, according to whether or not a wall was encountered.

GUI Pressing the "show transition" of the GUI gives us a visualisation of the transition model. We can see the transition probabilities at every step.

5.2 Observation models

The observation models contain the diagonals of the observation matrices for each possible sensor reading. The observation matrix encodes the probability of the robot's estimated position given a sensory input.

GUI In the GUI, the "show sensor" button gives us a visualisation of the observation models. We can see a heat map which shows probabilities of the robot's location given the sensor's output.

Difference between the 2 models

Probability distribution The difference between the two models is the distribution of the probability for a sensor failure. In the non-uniform model, the probability is adjusted based on the relationship between the robot's true location and the estimated one. The uniform model, on the other hand, assumes a uniform failure probability distribution for all the readings.

GUI In the GUI, the first image shows us the probability for the None readings. We can see that in the uniform case, the heat map is uniform so this shows how the probability of failure is uniformly distributed. This contrasts with the non-uniform's heat map, which has more probability of having a sensor failure on the edges and corners.

Performance The uniform model performs better in the smaller grid. But the non-uniform model has more accuracy in a bigger grid setting. This is due to the fact that, as the distances are bigger, the spatial sensitivity of the probability distribution becomes more important.
The non-uniform model requires more knowledge about the environment and the sensor characteristics, whereas the uniform one is more general and does not depend on any sensor's or environment's characteristics.
Note: I have used the initial version of the uniform model for the entire assignment.

6 Discussion of the results

Guessing and sensor output only Using a pure guessing approach, basic probabilities tell us that we would get a correct guesses percentage of $500/(ROWS*COLS)$. So for an 8x8 grid, we would get around 8% and the average Manhattan distance is of around 5.5, which is past the halfway point of the grid in any direction.
Keeping only the sensor outputs, gives us approximately the same error distance (around 5) as the pure guessing. It is sometimes even worse because of the sensor failures.

Filtering Filtering improves the localisation and has an average error of 1.5 Manhattan distance, with a percentage of correct guesses between 30 and 40 %.

Smoothing The smoothing improves the results even more. It gets an average error of around 1.1 with a correct results percentage of around 40-50%.

In conclusion, a hidden Markov model works well in this situation. It can estimate quite well the robot's position and the average error is relatively small, especially for a 16x20 grid. Moreover, it is a simple straight-forward implementation which improves drastically the results. The smoothing, however, improves the average error but does not show a significant difference. The drawback of it is that it cannot provide live localisation, as it only sees five steps back at each point. The question is now if it is worth doing it for this small improvement.

7 Discussion of the article

The article is "Monte Carlo Localisation: Efficient Position Estimation for Mobile Robots". It talks about another algorithm for robot localisation: Monte Carlo Localisation (MCL). The relation between it and my implementation is that they both solve the same problem: robot localisation.
MCL is an innovative approach, which applies sampling based methods to approximate probability distributions. It adapts the number of samples to reduce the computation costs, which gives it the ability to obtain higher resolution. As we can read in the article, the algorithm has shown great results in large spaces. In our task, we needed to implement a localisation in a small grid. MCL too big and cumbersome for this task, so our approach fits perfectly in it. For bigger and more complex spaces, it would of course be more efficient and precise to implement Monte Carlo Localisation.

Appendix

Peer review

For this assignment, my peer review partner is Sophia Kovalenko (so4816ko-s). Regarding the implementation in filters.py, it is evident that she has a deep understanding of how the forward filter and backward smoothing algorithms work. Both formulas used in the algorithm appear to be correctly written, as they closely resemble the logic I employed when writing my own implementation for these functions.

In terms of evaluations, Sophia's code seems fairly accurate. Her Manhattan distances for each case are below 2, and in terms of correctness rates, each case surpasses 25%. Additionally, for the evaluation of the smoothing function, she clearly demonstrates the difference between just filtering and filtering with smoothing, where the latter exhibits a notably higher correctness rate in the output messages.

However, there are slight differences in logic between her evaluation and mine. In my evaluation functions, I attempted to utilize the localizer class to minimize code bloat and unnecessary repetition. Sophia, on the other hand, followed the code provided in the localizer.py update function and used it as inspiration to write her evaluation functions. Despite these differences, both methods are logically similar.

In terms of improvement, the way Sophia made her graphs in the evaluations may need to be refactored as they appear to not be accurate. As far as I recall, as the number of steps increases, the average Manhattan distance should decrease accordingly, where a downward trend is observed in the graphs. However, the graphs Sophia has produced are scatterplots where the data points are difficult to discern, making it challenging to compare the different methods effectively. It is recommended that she address this issue before finalizing her implementation.

Overall, Sophia has done a great job with this assignment and outside of the graphs, seemed to have fulfilled the requirements of this assignment