Probabilistic reasoning over time

Harald Haglund <ha4384ha-s>

February 19, 2024

1 Statement

Peer review was made with Erik Ersmark, <er5612er-s>. The assignment was discussed with Daniel Eklund, <da6604ek-s>in order to gain a comprehensive understanding of its requirements.

2 Summary of the task

The point of this assignment was to use probabilistic reasoning over time based on the Hidden Markov Model in order to predict the position of a robot in a grid. In order to accurately predict the position, two types of filters were used, previously derived from the theory. The filters are based on forward filtering and a naïve form of fixed-lag smoothing (also known as forward-backward filtering with smoothing), and look as follows:

$$f_{1:i} = \alpha \mathbf{O}(d_i)^T f_{1:i-1} \tag{1}$$

$$b_{k+1:i} = \mathbf{T}^T \mathbf{O}(d_{k+1}) b_{k+2:i} \tag{2}$$

$$P(T_k|D_{1:i}) = \alpha f_{1:k} \cdot b_{k+1:i} \tag{3}$$

Here, equation (1) represents the forward part. The the backwards part and smoothing are defined in (2), (3) respectively. [1]

3 Peer-review

During the peer-review, my peer noticed three different ways to improve my code. The first one was a bug i had inside my smoothing, where i used the Transposed T-Matrix by accident. This resulted in my smoothing function performing very poorly, and when i changed it to align with the "correct" formula, my smoothing worked as expected.

The second issue i had was calculating the "None" frequency. This was quite a simple bug to fix, but something i just didn't realize was present until we compared our frequency's. My frequency was correctly adjusted from 70% to 30%.

The third tip i got from Erik was to call the forward filter function inside the smoothing function. This essentially just reduces the use of duplicate code, and makes the code more readable.

Lastly, we briefly discussed if there were any improvements that both of us could do. One subject that came up was the concept of using dynamic programming. By saving values for each calculated sequence, we can take advantage of the fact that some sequences may appear again, and thus use the old stored values to save a lot of time. Whether or not this is feasible in our case is unclear, since the sequences are of length five and thus requires a lot of memory.

4 Explanation of the models

The models given in this assignment were "ObservationModel__UF (Uniform distribution)", "ObservationModel__NUF (Non-Uniform distribution)", "StateModel" and "TransitionModel".

The state model describes a certain state. This includes the dimensions of a grid and help-methods to transform a pose or a position into a state or a sensor reading and/or vice versa. This class was mostly used inside the main method, in order to convert readings to positions.

The transition model contains the transition matrix, where each element represents the probability of transitioning from one state to another. Inside the transition class we also have an important help-method to fetch the transposed matrix.

The observation models connect the hidden states of the system to the observed sensor readings. The observation matrix represents the probabilities of observing each sensor reading given the current state. This leads us to the next question: What is the actual difference in the two sensor/observation models, and how does this difference affect the performance in different grids?

Simply stated, the difference lies in how the different models assign probabilities to the sensor readings. In a uniform distribution, the probabilities for reporting sensor readings are distributed uniformly across all scenarios. The model is very simple and generic. For the non-uniform distribution the probabilities for reporting sensor readings vary. Different probabilities are assigned for readings in surrounding fields, *Ls* and "secondary" surrounding fields, *Ls2*. This model is more complex. We have now established that the complexity of the different models differ, but how does the complexity of a model explain its performance?

The answer is that different models are good for different applications. When picking a model, there is always a trade-off between variance error and bias error.[2] By increasing the model complexity we decrease the bias error, but we also increase the variance error.[1] The opposite happens if

[1] By decreasing the bias error we enhance the accuracy and reliability of the model, leading to more precise predictions, but by increasing the variance error, the model becomes less robust against noise and worse at generalizing unseen data.

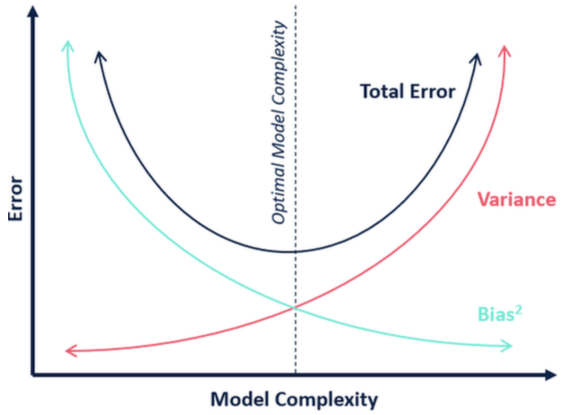
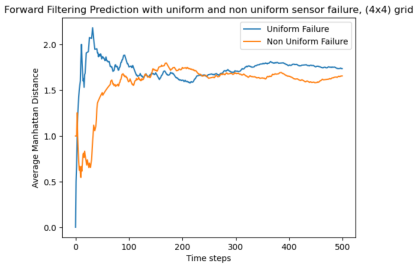
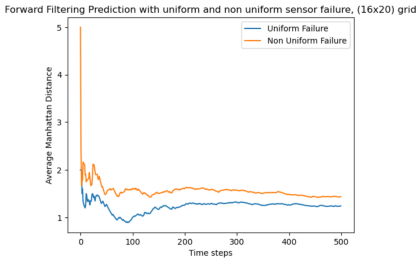


Figure 1: Illustration of how model complexity affect the error [2]



(a) Results in the (4x4) grid



(b) Results in the (16x20) grid

Figure 2: Uniform vs non-uniform failure, results 2) and 3) from the notebook

we lower the model complexity, so we want to use a model that has an optimal model complexity, i.e minimum error 1. This explains why the simple model gives better results for a small, simple grid (4x4) but the non-uniform distribution model gives better results for the larger (16x20) grid 3. These concepts are also called under-fitting (Having a too generic model in a complex environment) or over-fitting (Having a too complex model in a simple environment). In our case the uniform model is under-fitting in the (16x20) grid, and the non-uniform model is over-fitting in the (4x4) grid, furthermore explaining why the results turned out like they did.

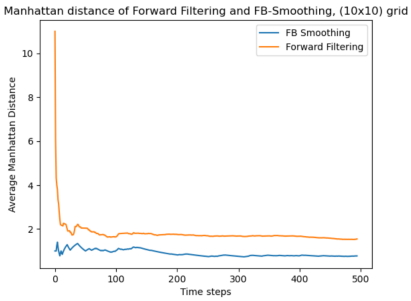
5 Discussion

5.1 Discussion of my results

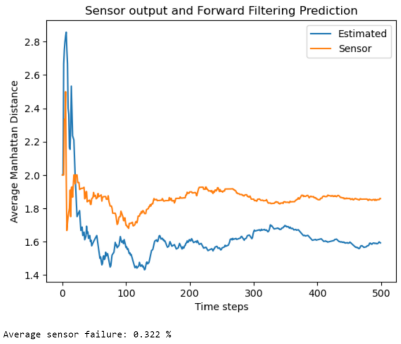
These are the Manhattan distances, as well as the correct number of guesses, that i ended up with[2] for the (10x10) grid, non-uniform sensor failure:

- Random guessing

[2]After computing the average value for 500 iterations



(a) Plot of the FB-Smoothing vs Forward Filtering



(b) Plot of the Sensor vs Forward Filtering

Figure 3: Results 1) and 4) from the notebook

  - Manhattan distance: 6.7
  - Number of correct guesses: 10

- Sensor Reading
  - Manhattan distance: 1.9
  - Number of correct guesses: 80

- Forward Filtering
  - Manhattan distance: 1.7
  - Number of correct guesses: 128

- FB-Smoothing
  - Manhattan distance: 0.9
  - Number of correct guesses: 580

The value for random guessing was determined empirically after introducing a "random" function inside the filters class. With these results we can conclude that both Forward Filtering and FB-Smoothing are providing significant improvements in tracking accuracy compared to random guessing. Furthermore, FB-Smoothing outperforms Forward Filtering, implying that incorporating information from future measurements in the smoothing process contributes to better localization accuracy. Comparing just the sensor reading with the forward filtering shows us that there is not a large increase in accuracy, suggesting that introducing forward filtering only is not optimal for this task. Another filter that would possibly perform better is a Kalman filter, since its good at handling linear system dynamics and Gaussian noise.

5.2 Relation between my implementation and the work described in the article

After reading the article "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots" [3] i aim to answer the question: Is the HMM approach

as implemented suitable for solving the problem of robot localisation? In the article, an alternative location tracking method is introduced, called Monte Carlo Localization (MCL for short). This method is designed to address the uncertainty associated with a robot's sensor readings. In a Monte Carlo Localization, you represent the uncertainty through a set of locations, each denoted by a particle. As the robot collects sensor data, these particles are constantly updated to present the changing uncertainty about the robot's position.

By looking at the presented tests, we see that MCL outperforms HMM by far, providing better estimates of the robot's position. By reducing the computation-time and memory consumption, MCL can operate at a higher frequency and incorporate more sensor data, which in turn implies much higher accuracy.

After looking at this data, i believe its safe to say that MCL is a better localisation method than HMM, especially in complex environments (which is usually the case for robot localization). That doesn't mean that HMM are useless though, since they are still highly effective in capturing uncertainties, making them particularly valuable for applications like speech recognition, machine translation, financial economics and many other fields.[4]

6 Appendix: Comments from my peer

Firstly, Harald had implemented a filter function (forward) and a smoothing function (backward-smoothing). The smoothing function has 2 redundant lines of code that can simply be replaced by calling the forward function, which makes it more readable.

Secondly, Harald had identified an error with the frequency counting of "None" reports from the sensor. I suggested changing the code such that you would count the "None" elements after the for-loop and dividing the count with the number of steps.

Lastly, there was an error in the smoothing function as it performed worse than the filter (forward). After analyzing and in comparison with my own smoothing function, I found the error to simply be a minor detail in the B calculation, using a transposed T matrix instead of regular (order matters).

We also discussed plotting for the average sensor Manhattan distance, whether to plot over all steps or just the ones not reading "None". Harald had done the first ("None" read would give the previous average at that step) while I did the latter. It was to us, unclear from the task description which was the correct way but overall our plots had similar shape still.

References

[1] Simon Kristoffersson Lind. "Artificial Intelligence Lecture Notes - Probabilistic Reasoning over Time". In: (2024).

[2] Microsoft Pragati Baheti. *Overfitting vs. Underfitting: What's the Difference?* URL: https://www.v7labs.com/blog/overfitting-vs-underfitting.

[3] Dieter Fox et al. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: (1999).

[4] Stuart Russell and Peter Norvig. "Artificial Intelligence: A Modern Approach". In: (2010), pp. 497–498.