"Robot" localisation with HMM based forward-filtering

Aliya Ospanova & Ómar Bessi Ómarsson

0. Prefix & peer review statement

A peer review was conducted with Janna Osama & Vahid Faraji

Ospanova did implementation (task #2), whilst Ómarsson did evaluation and report writing (tasks #3 & #6). Both researchers shared responsibility in understanding the base code and models that were given (task #1), conducting the peer review (task #4), and reading the research paper by *Fox et al.* (task #5).

1. Task summary

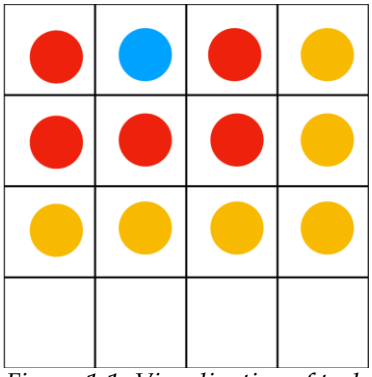

Figure 1.1: Visualization of task

The task is to track an agent (meant to replicate a robot) through a grid-based evironment without any landmarks. To do this, we will use *forward filtering,* and *smoothing* based on a **H**idden **M**arkov **M**odel (HMM). The agent's true location is hidden, therefore we must use unreliable sensor data to try to track the agent through the environment. *Figure 1* shows how the agent's location and the sensors interact. The sensors will show 3 approximations: the true location of the agent (blue), the direct surroundings of the agent (red), and the wider surroundings (yellow). The agent's location can be reported with three parameters, {x, y, θ}, where θ connotates the heading of the robot (south, east, north, west).

2. Explanation of models

Researchers worked with 4 models to create the Forward Filtering and Smoothing algorithms:

1. A **State** model that contains information on the state of the robot, with a *state* being the index coding of a *pose* (a pose being the {x, y, θ} mentioned in section 1). For instance, state 1 is the pose <0, 0, 1>. It also contains *readings*, which are the index codings of *positions* (x, y). States are to poses what readings are to positions are vice-versa.

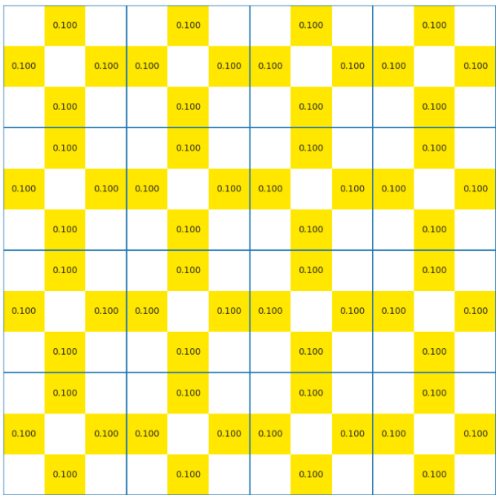

Figure 2.1: Observation model with uniform sensor failure with no sensor readings.

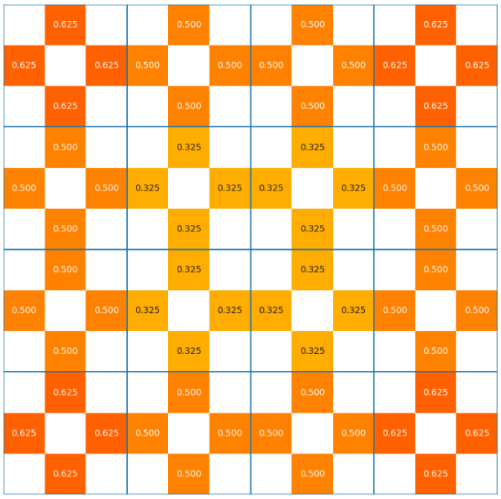

Figure 2.2: Observation model with non-uniform sensor failure with no sensor readings.

2. Two **Observation** models. The observation models calculate for each position in the grid the probability that the agent is at that position. There are a couple of key differences in the two models:

One of the models, ObservationModel_UF, calculates the probability of the sensor reporting a surrounding state based on the number of surrounding positions (the number of red and yellow dots in Figure 1.1, here on out named *n_Ls*, and *n_Ls2*, respectively). The red dots get a probability of *0.4/n_Ls*, and the yellow dots get a probability of *0.4/n_Ls2*, it sort of normalizes the states, making them uniform. When this model gets no sensor data, it fixes the probability of all states to 0.1, see Figure 2.1

The other model, ObservationModel_NUF, fixes the probabilities in the sensor based on proximity. Red dots have a probability of *0.05*, yellow dots have a probability of *0.025*. When this model gets no sensor data, it computes for each position *1.0 - 0.1 - n_Ls*0.05 - n_Ls2*0.025*. See Figure 2.2, states at grid edges have fewer neighbours, leading to higher probabilities.

During localisation, the NUF model adapts more effectively to the environment than its UF counterpart. This difference is visualised in Figures 2.1 and 2.2, where the probability distributions in corners and edges under the NUF model are significantly more pronounced.

3. A **Transition** model, defining how the agent moves on the grid. The model defines and works with a matrix representation of the agent's movement system. The agent picks a random start heading (south, east, …) h_0 and then picks a new heading h_t+1 based on the current heading h_t according to:

P(h_t+1 = h_t | not encountering a wall) = 0.7	P(h_t+1 != h_t | not encountering a wall) = 0.3
P(h_t+1 = h_t | encountering a wall) = 0.0	P(h_t+1 != h_t | encountering a wall) = 1.0

3. Program Results

Below is a comparison of different versions of the algorithm against each other for evaluation purposes: Forward Filtering (NUF) vs. the Sensor output, Forward Filtering with Uniform Sensor Failure vs. Forward Filtering with **Non**-Uniform Sensor failure, and Forward Filtering vs. Forward-Backward Smoothing. Plots are referred to as plots 1-4, read from left to right, top to bottom. Plots were made with the help of DeepseekAI.

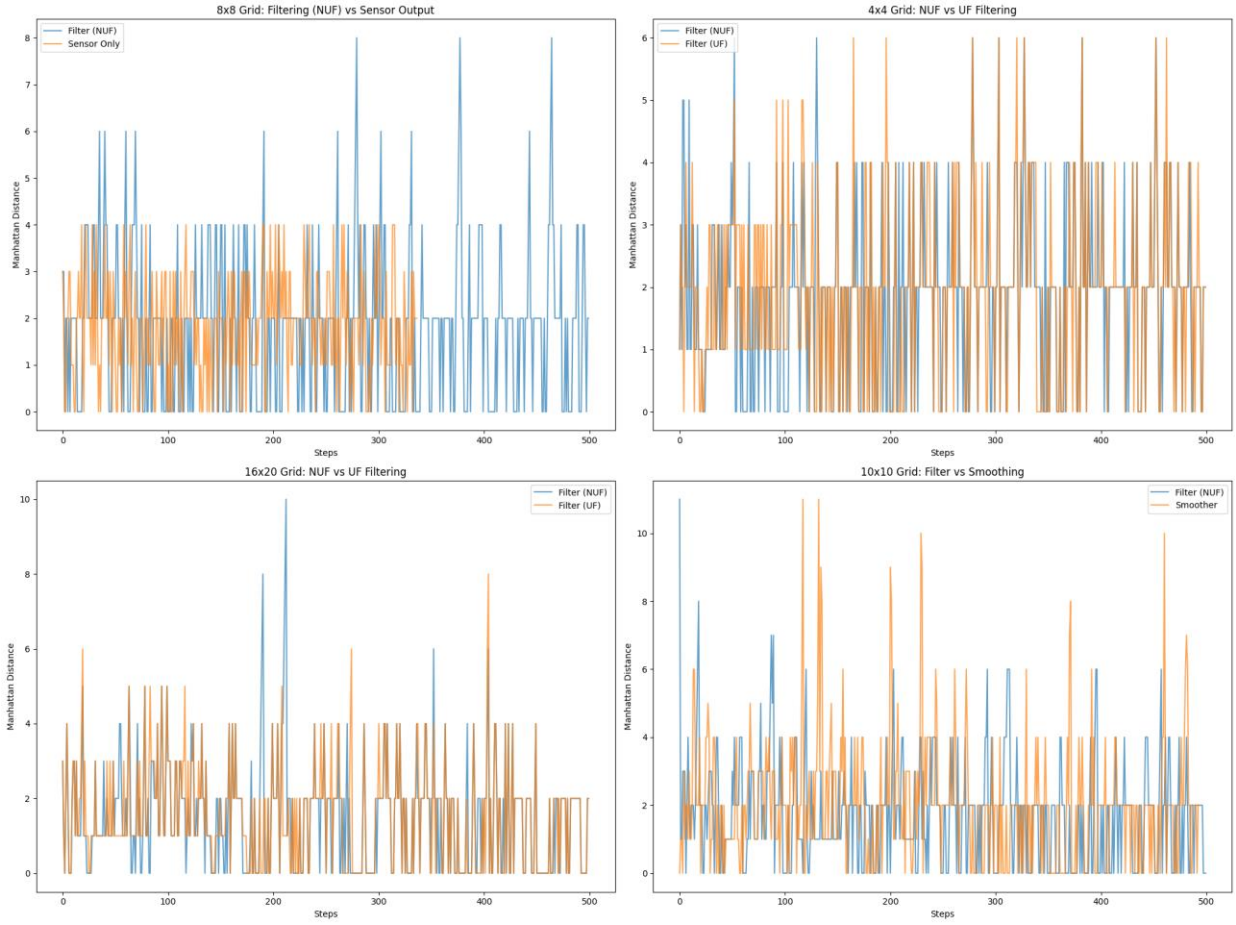

Figure 3.1: Results from testing

Plot 1 average errors: Filtering: 1.78 Sensor: 1.786	Plot 2 average errors: Non-uniform filter: 1.868 Uniform filter: 1.956
Plot 3 average errors: Non-uniform filter: 1.502 Uniform filter: 1.502	Plot 4 average errors: Non-uniform filter: 1.68 Non-uniform smoother: 2.012

Figure 3.2: Table of average error rates from Figure 3.1

4. Results Discussion

The results from the first plot show that the filtering function follows sensor error very closely (the average error is nearly the same, see Figure 3.2), indicating that the filtering function performs well given correct data. When comparing the uniform and non-uniform observation models, we see that the non-uniform model performs slightly better than its uniform counterpart.

When comparing forward filtering to forward-backward smoothing, we observe an increase in error rate. This suggests that the smoothing step may introduce errors due to incorrect assumptions about past states. This might be due to program error, rather than an issue with the algorithm itself. The trade-off between filtering and smoothing depends on the specific application and the nature of the sensor noise.

Overall, forward filtering seems to provide a reliable method for tracking the agent's position. We see even in larger grids (Figure 3.1 / Figure 3.2, plot 3), the average error remains reasonably low.

5. Comparison to other studies

The research by Fox et al. presents a method known as **M**onte **C**arlo **L**ocalisation, which improves computational efficiency whilst maintaining accuracy. Fox et al. mention many reasons why it was advantageous over other work in the field, citing higher accuracy, reduced memory usage, and ease of implementation.

One of the main advantages of MCL over HMM-based methods is that MCL can dynamically adjust the number of particles to focus resources when needed, unlike HMM-based methods, which try to keep track of all states at all times. This difference leads to significant improvements in computational efficiency.

However, HMM-based filtering still has practical applications. Our implementation is more well-suited for scenarios where a fully probabilistic sampling method like MCL is unnecessary. Additionally, HMMs offer a deterministic framework that might be easier to work with for predefined motion models.

Given the results presented in this report, we find that; even though HMM-based localisation is capable of tracking a robot agent with reasonable accuracy, it does not match the efficiency of MCL.

In clean, empty environments, HMM-based approaches are definitely viable, however, for real-time applications in dynamic environments, MCL provides a more efficient and effective solution.