Report of assignment 3

Joel Bäcker (jo4383ba-s) of EDAP01

March 16, 2022

1 Summary of implementation

My implementation of the assignment were done in the files `RobotSimAndFilter.py` and `Localizer.py`. In the first file two classes, `RobotSim` and `HMMFilter`, are implemented. The first of the two simulates the behaviour of a robot using two method `next_state` and `sense`. The `next_state` method requires a true state, i.e. current position, to find the next position the robots moves to, this is done by extracting all possible moves and then using the transition model to calculate the probability to move between the current position and all the positions, the move returned is based on that probability. The second function, returns a sensor reading in a similar manor to the previous method, but instead of finding the probabilities using the Transition Model the probabilities for the different scenarios is found in the implementation specification. The probabilities are used to guess what the most likely path the robot has taken. In the specifications it says that probability to move between two states where the heading is the same and not encountering a wall has probability 0.7, and not moving in the same direction, i.e turning, and not encountering a wall has the probability 0.3. Moving in the same direction and encountering a wall has probability 0, and turning away from the wall probability 1.

The `HMMFilter` class does what the name suggests, applies a Hidden Markov Model Filter. It works by calling the method with a sensed position and probability vector, denoted as $\mathbf{f}$. The forward-equation can be written as matrix and vector operations using the a transition model and oberservation model:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{e_{t+1}} \mathbf{T}^T \mathbf{f}_{1:t} \text{ where } \alpha = \frac{1}{\text{sum}(\mathbf{f}_{1:t+1})}$$

The alpha terms ensures that $\mathbf{f}_{1:t+1}$ summarizes to 1, i.e. a probability vector.

In the last file, `Localizer`, the update cycle takes place. The method `update` performs the steps of the cycle, i.e. robot moves and generates new state, sensor produces a reading based on the state and filtering creates a probability distribution. These steps are done by first calling on `next_state` then `sense` in `RobotSim` and lastly `forward_filter` in `HMMFilter`.

2 Peer review

I peer reviewed with Christoffer Svenningsson, and expect for him I discussed the assignment with Lisa Af Klint, Linus Åbrink, Hugo Mattsson, Fredrik Voigt,

Victor Winkelmann, Arvid Bengtsson, Filip Tran, Hannes Östergren, Cecilia Huang, Evelina Danielsson, Marcus Hedebark.

My peer commented on two things, that I should implement a main method that evaluate the performance of the implementation and normalize the probability vector in the HMM filter. The fix for this where quite simple to implement, the norm of the vector where calculated by dividing the probability vector with the sum of it, i.e. `f /= sum(f)`. It is important to note that when we normalize the vector it needs to summarize to 1 for it to be a probability distribution. To evaluate the performance of the program I created a main that creates an instance of `Localizer`, `StateModel`, `TransitionModel` and `ObservationModel`, and in a for-loop call `update` on the `Localizer`. From `update` we get values to create the mean error and the number of correct guesses.

3 Handouts

In the visualization given in the notebook shows two buttons at the top, "Show Transition" and "Show Sensor". Pressing the first one will show the probabilities of moving from a current state to all the other states, pressing the button multiple times will traverse over all the sates. The number of state that are possible to move to are at most 4, so all other states will have probability 0.

The other button displays the probability for the sensor to report a given position on the board. The probability in the current state is 0.1, one step away has probability 0.05 and two steps has 0.025 as stated in the assignment instruction.

If a new visualization is created without creating a initial filter and pressing "Show Sensor" a grid with probabilities that shows for each state the probability that "nothing" were detected. The probability is equal to $1.0 - 0.1 - \text{n\_Ls} \cdot 0.05 - \text{n\_Ls2} \cdot 0.025$, where n_Ls is the number of reachable states one step away and n_Ls2 is the same but for two steps.

4 Discussion

As said by my peer, I should implement a evaluation method to compare the performance of choosing a move by random. The performance of the HMM filter (on a 8×8 grid) made about 30% – 40% correct guesses and average error where less than or equal to 2 while the random choices only making ca. 2% correct guesses and average error were larger than 5. If the sensor readings were used the accuracy slightly increases from just random guessing, but the accuracy just reaches, at most 12% and a mean error of about 4–6. We should expect that using sensor readings gives a better result than just random choice since the sensor readings are actually based on possible moves and not just random guesses. This illustrates that the robot can make qualified guesses that are far better than just making random choices and solely using the sensor reading.

5 Article

The article to discuss for this assignment were *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots* by D. Fox et al., it analyse the

performance of Monte Carlo Localization (MCL) and compare it to other popular localization methods, including HMM filtering. The idea behind MCL is it samples/sensing the environment around a robot to estimate its position and orientation, given a map of the setting. Some of the advantages over HMM stated by the authors is MCL drastically reduces the memory requirement compared to HMM, and it being more accurate than HMM when a fixed cell size it used, as well as it being much more easier to implement. MCL makes samples that are proportional to the likelihood, thus more samples from where the it really matters. The article also states that adding random samples is necessary for the robot if it needs to be relocated after it loses its position.

The different methods were tested in a couple of real life scenarios, one of those were that the robots should navigate the Smithsonian's Museum of Natural History. The experiment resulted in that the grid based localization method (HMM) failed to locate the robot only using its vision because of the computationally heavy task of analysing the surrounding. On the other hand were the MCL successful in both tracking and localizing the robot.

The answer to the question of whether or not HMM is suitable to for solving the problem of localization, depends on the premises of the localization. As said above, a HMM-like method failed during the visual part of the experiment, but for simpler task it were able the complete the task of both localizing and tracking. But, MCL were virtually better in all categories tested by the researchers. For a task such as this assignment where the grid size is small and stay fixed its performance is adequate. If however, a harder task is to be done, MCL would be the method to be chosen.