Assignment 2

Probabilistic reasoning over time

Filippa Hansen

EDAP01

February 17, 2023

1 Peer review

The peer review was done with Abdulrahman Husari (stil-id ab8233hu-s).

2 Summary of the task

The task was to implement a tracking approach which should be based on a Hidden Markov Model with forward filtering in order to track a robot. The task also included writing a class ”RobotSim” to control the movement of the robot using the predetermined rules of the movement provided in the task. If the robot for example was heading at one direction in the last step the probability of the robot continuing in that direction if it was not heading a wall should be 0.7. It was also of essence that the RobotSim class was to be implemented in a correct way for the predictions to be accurate, since the feature vector holding the probabilities for the different states is based on these probabilities. The provided classes StateModel, ObservationModel and TransitionModel where to be used in the implementation of the movement and the tracking of the robot.

3 Implementation

As previously indicated, the handout provided three classes - StateModel, TransitionModel, and ObservationModel - for use in the implementation of the RobotSim, HMMFilter, and Localizer classes. The Localizer class controls the Robot Simulation, and the first step is to retrieve the current pose from the current state using the methods provided in the StateModel class.
In the RobotSim class, a ”move” method has been implemented, which moves the robot one step either vertically or horizontally based on probabilities determined by the direction of the robot's previous movement. The Localizer class calls the move method, causing the robot to move one step, after which the get_current_state method in the RobotSim class is called to obtain the new state.
Once the state is updated, the ”sense” method in the RobotSim class is called to obtain an updated reading based on the ObservationModel. The probabilities for the sensor to produce any of the possible readings while in the current state are then calculated, and a random reading weighted over the probabilities is returned if the sensor did not sense ”nothing”. If the last reading is chosen, it indicates that the sensor produced no reading because the last vector in the ObservationModel contains the probabilities for the sensor to produce ”None” as a reading.
After the reading is updated in the Localizer, the ”update” method in the HMMFilter class is called. This function uses a Hidden Markov model with a forward filtering approach. The update method first obtains the transposed transition matrix with the probabilities from the TransitionModel class, and then the diagonal matrix with the probabilities of the states from the ObservationModel class. It then calculates the product of the two matrices and multiplies it by the feature vector to obtain the new feature vector. The resulting matrix is divided by the sum of the matrix to calculate the new feature vector.
The estimate of the robot's position is determined by selecting the state with the highest value in the feature vector. This state is then converted to a position using the StateModel. The estimated position is updated to the obtained position and is returned in the update function along with the new feature vector.

4 Changes in the implementation

When Abdulrahman reviewed my code my estimated position was based on the true position and therefore accurate 100 % of the time.
Regarding the comments on the robots movements it is true that I used an unnessasarily complex approach where I could have made use of the TransitionModel, it has since then become clearer of how this could be done. The choice of heading should be correct since the movement of the robot follows “robotics conventions”, and this was evaluated before implementing the filter. Although the task was to use the TransitionModel and the code has therefore been updated with this approach. The HMMFilter did produce a feature vector that looked fair, but something was still off in the implementation. And by using the initial belief state vector fVec, the transition matrix, and the observation matrix as suggested by Abdulrahman a more accurate prediction could be made. I also updated my sense method in the RobotSim class when I noticed that the accuracy was way to high and that there was a reading in all cases.

5 The models

In the visualisation of the transition model the probabilities for the different poses (x, y, h) to be reached after having been in the given state (marked in cyan) can be seen. The red box is the estimated position. In the visualisation of the sensor model the given state is visualized with the probabilities surrounding the current state. While running the filtering process the current position is visualised in black, the estimated position (based on the probabilities that are visualised in the cells) in red and the sensed position in cyan. When the sensed position is none there will be no cyan-colored cell since it is only visualised if the value of sense is True.

5.1 StateModel

The StateModel provides methods to transform between a pose, position, reading and state. It also provides methods to get the number of states, readings and the dimensions of the grid.

5.2 ObservationModel

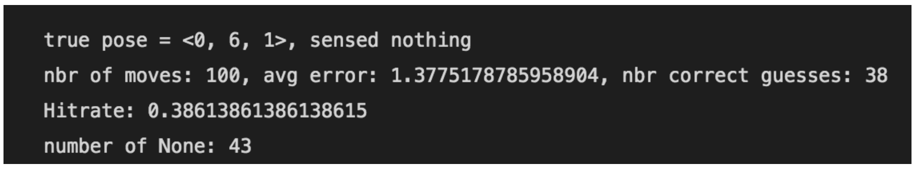
The observation model, stored as a vector, contains the diagonal of observation matrices for each sensor reading, including the probability of the sensor producing ”None”. It calculates probabilities for the sensor reporting a reading in the current state, surrounding fields, and secondary surrounding fields. The ”None” probability is calculated as 1.0 minus the sum of the other probabilities. The class also provides methods to retrieve the number of readings, probability for a reading in a state, O_reading diagonal matrix, and plot vectors as heat maps.

5.3 TransitionModel

The Transition Model is responsible for providing the probability of moving from one state of the robot to another. The probabilities are determined based on the rules defined in the Assignment description, such as the probability of maintaining the same heading or changing the heading and the effect of encountering a wall. The Transition Model uses a matrix to store the transition probabilities between states.

6 Discussion

After 100 steps in the grid the average error is 1.38 and the hitrate/success rate 0.39. The error is calculated using Manhattan distance. Out of 100 moves 43 of the moves has a Non-value on the reading. So answering the question ”How accurately can you track the robot?”, the accuracy will lie around 30% when using a 8x8 grid. And the average error will lie around 1.4 steps.



```
true pose = <0, 6, 1>, sensed nothing
nbr of moves: 100, avg error: 1.3775178785958904, nbr correct guesses: 38
Hitrate: 0.38613861386138615
number of None: 43
```

It can also be seen that in the cases when the reading is not None the accuracy is 0.67.

7 Monte Carlo Localization: Efficient Position Estimation for Mobile Robots

In this article, a method called Monte Carlo Localization (MCL) is introduced as a probabilistic approach to estimate the position of a mobile robot in an environment. MCL utilizes a particle filter that maintains a set of samples, with each sample representing a possible location of the robot. The samples are distributed throughout the environment uniformly at first, but as the robot moves and takes sensor measurements, the samples are updated to reflect the robot’s new location estimate. This is achieved by assigning weights to each sample based on its correspondence to the sensor measurements and resampling the samples accordingly to generate a new set of samples.
The authors demonstrate that MCL can handle highly nonlinear and non-Gaussian models of robot motion and sensing because it samples from the robot’s belief distribution, which can be complex and non-Gaussian. The particle filter also allows the robot to maintain a multi-modal belief distribution, which is often necessary in highly uncertain and variable environments.

8 Relation between implementation and article

MCL represents the robot’s location using weighted particles and is suitable for environments with high uncertainty. HMMs estimate the probability distribution of the robot’s position based on sensor readings and are suitable for low uncertainty environments. However, the HMM approach may not work well in non-Gaussian environments, while MCL can handle highly nonlinear models and non-Gaussian noise distributions. The study presented in the paper demonstrates that the HMM approach implemented is not suitable for solving robot localization in non-Gaussian environments, while MCL has been shown to be more accurate and faster in such environments.
The probabilities assigned to each possible movement in the implementation are not Gaussian, since they are not normally distributed around a mean. Instead, they are based on a set of probabilities assigned to each possible movement given the current state of the robot. Therefore, the movement can be considered non-Gaussian. Which means that the HMM approach as implemented is not suitable for solving the problem of robot localisation which can be seen while stepping through the grid. The average ”hitrate”/how often the prediction is correct is around 0.3 when a grid of the size 8x8 is applied which is a strong indication that this approach is not suitable.

9 Appendix

9.1 Peer Review

Upon launching the Viewer, I observed that the robot's movements were consistently accurate, with the sensor registering a 100% success rate in detecting its position and the filter providing equally precise location estimates.
The robot's movement adheres to the specifications outlined in the assignment. However, it appears that you opted not to utilise the TransitionModel provided by the course and instead created your own from the ground up. While this approach is not necessarily incorrect, your choice of headings may be inaccurate, as h_0 = 0 should correspond to a downward direction rather than a rightward one.
The primary issue pertains to the sensor implementation. In your solution, you employ the StateModel to emulate sensor readings, thus avoiding the generation of false data and relying solely on actual readings. However, the sense() function in the RobotSimAndFilter class is not utilised and seems to be misspelled. This function should employ the ObservationModel to generate sensor readings that conform to the assignment's specifications and do not provide a 100% accuracy.
Regarding the filter, it is challenging for me to assess its correctness as the sensor consistently provides the robot's precise location. However, the filter does not require the use of the state model, and only three matrices are necessary: the initial belief state vector fVec, the transition matrix, and the observation matrix. By utilising these three matrices, one can obtain the information needed for the filter. I suggest reviewing the first section of the Probabilistic Robotics lecture, as it offers a comprehensive explanation of the filter's functionality and computation.