Assignment 3 - Probabilistic Reasoning over Time

Theodora Gaiceanu

Filtering and Hidden Markov Models (HMM)

According to [2], filtering computes the belief state $P(X_t|e_{1:t})$ (the posterior distribution) over the most recent state given all evidence to date. The process is recursive because one wants to calculate the estimation for the next step $t+1$ from the new observation e_{t+1} when having the result of the filtering up to moment t. An HMM is a temporal probabilistic model where the probability of being in a certain state depends only on the last state (here the state is a single, discrete random variable [2]). As we are dealing in this assignment with a very noisy sensor, trying to do robot localisation based on forward filtering with HMM is a good and efficient way of solving the given task.

Summary of the implementation

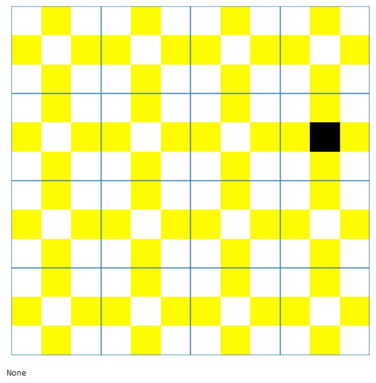
The implementation for the robot simulator can be observed in the *RobotSim* class. This class has 3 private properties (an instance of the state model, an instance of the transition model and an instance of the observation model). The movements of the robot are implemented in the *movingStrategyNew* method of this class. The method has one argument, the current true state of the robot. First, one gets the transition matrix and selects only the probabilities that are not 0, as they represent the probabilities of the possible next transition states. Then one does random sampling in order to get just one state according to the probabilities. This state is returned as it represents the next state of the robot. The sensor readings are implemented in the *sensor* method. This function has again the current state as argument (it will be the current state returned by the robot movement). One iterates through all the sensor readings produced and gets the probabilities for the sensor to have produced the current reading given the current true state. Then one does a random sampling and chooses one reading according to the before mentioned probabilities. If the sensor does not produce any reading, the returned reading should be None, otherwise it is the reading from the random sampling.

The forward filtering (implemented as a method in the *Localizer* class) produces a new prior based on the returned sensor reading, transition and sensor models. So, first, one stores the values of the priors, of the observation model for the current sensor reading and of the transposed transition model. Then one updates the priors according to the equation $f_{1:t+1} = \alpha * O_{t+1} * T^T * f_{1:t}$ from [2]. The estimate is computed based on the highest probability for one single state. The HMM filter returns the priors and the estimate.

One last thing to do is to compute the Manhattan distance between the current true state and the current estimate. This is computed easily as $|eX - tsX| + |eY - tsY|$, where eX and eY are the coordinates of the estimates and tsX and tsY are the coordinates of the true state.

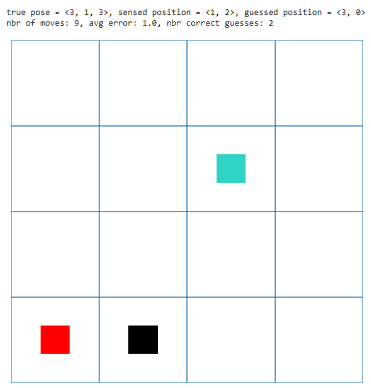
Peer Review

I did the peer review with Rasmus Beck. He commented upon my methods for the robot movements and for the sensor readings. He pointed out that it is not necessary to normalize the probabilities for the next possible states, as they already sum to one. So I am not using the normalization for that anymore. Also, after peer review-ing him, I noticed that my previous implementation for the sensor reading was not working well (thing that he also mentioned) and was not treating good the cases when the sensor does not report a reading. Before, my sensor function converted the current state given by the robot movement to a reading, then it saved the diagonal elements of the observation matrix corresponding to the reading. The zeros were changed with the probabilities for a None reading and this new array of probabilities was saved. Then it followed the random sampling, that chose one possible state according to the array with probabilities. Then the state was converted to a reading.



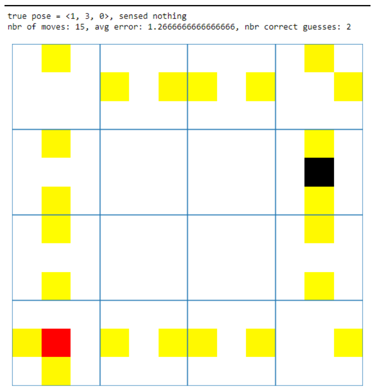
```
None
```

Figure 1: Initialization



```
true pose = <3, 1, 3>, sensed position = <1, 2>, guessed position = <3, 0>
nbr of moves: 9, avg error: 1.0, nbr correct guesses: 2
```

Figure 2: Running the filter



```
true pose = <1, 3, 0>, sensed nothing
nbr of moves: 15, avg error: 1.2666666666666666, nbr correct guesses: 2
```

Figure 3: Running the filter - no sensor reading

Models from the handout

The transition model implements the transition matrix with the probabilities for the next transitions and some methods like: getting the number of states represented and the probability to go from a current state to another one, getting the entire matrix and the transposed matrix, plotting the matrix as a heat map. The observation model contains the diagonals of the observation matrices for each possible sensor reading. The probabilities are computed according to the instructions in the constructor. The model has methods for getting the number of possible readings, the probability to produce a certain reading when in a particular state, the diagonal matrix with the state probabilities for a certain reading or for no reading, plotting the probabilities as a heat map. The state model contains the dimensions of the grid and conversion methods (from a state to a pose and vice versa, from a state to a position, from a reading to a position and vice versa, from a state to a reading and vice versa).

The transition matrix has the dimensions (rows*columns*4) x (rows*columns*4), each element representing a probability for the possible transitions (next movements). The transition model can be seen in the instructions (page 2, left Figure from the bottom), where it shows the probabilities for the two possible next poses (x,y,h) (yellow and orange) given the current state (cyan). The observation matrices are diagonal matrices with the dimensions (rows*columns*4) x (rows*columns*4). There are (rows*columns+1) observation matrices, one for each sensor reading. The sensor model can be seen in the instructions (page 2, centre Figure from the bottom), where it reports a reading (cyan). The probabilities for each reading can be seen in the yellow squares (0.1 for the true location, 0.05 for the directly surrounding fields, 0.025 for the secondary surrounding fields). The sensor can also produce no reading. This situation is represented at page 2, in the instructions, right Figure (it can be observed that there is no cyan square in the middle of any cell). This observation model shows the probabilities for the case when the sensor reports nothing according to the formula $1.0 - 0.1 - n\_Ls * 0.05 - n\_Ls2 * 0.025$. Also, from the observation model figure without a sensor reading, it can be seen that in case of a None sensor reading, the probability of the robot being in a corner or near a wall is very high (this is because of the sensor model, which make it more prone to errors near a wall or in a corner).

One can view also the filtering steps. Figure 1 shows the initialization step. The black square is the initial state and the probabilities in the yellow squares are the initial priors f. Then, as the filtering process is running, one can see the true state (with black), the sensor reading (cyan) and the estimated position (red). This situation is illustrated in Figure 2. Then one can have the situation when there is no sensor reading, so one can only observe the true position and the estimated one (Figure 3).

Evaluation and results

The evaluation was done by writing my version of *main* in the notebook. This one takes into consideration a 8x8 board and creates an instance of the state model and the localizer classes. It iterates through 300 trials and for each one it computes the Manhatten distance for the HMM filter and the Manhatten distance for the case when using only the sensor readings. It counts the correct results for the HMM filter and the sensor readings. After computing everything for each trials the accuracy and the Manhatten distances for both cases are printed. The filter accuracy is somewhere between 22% and 30%, while the accuracy of the sensor readings is just 10%. The Manhatten distance for the filter is 1.86 (or 1.63 in some cases), while the Manhatten distance for the sensor readings is near 4. These results are in accordance to the instructions. It is obvious that the HMM filter improved the performances of the robot. Also, sometimes it may take some iterations before the filter gives some good results. This is because in the beginning we do not know where the robot may be.

Article summary

In [1], it is presented a Monte Carlo based localization (MCL) for mobile robots. The robot's belief are represented using efficient sampling methods. More samples are used in the global localization phase and less in the tracking phase, as one has a hint about the location of the robot in the tracking phase. MCL has advantages as: the ability to globally find the location of a robot, a significant reduction in the amount of memory used when compared to grid-based Markov localization, an increased accuracy.

MCL is using sampling/ importance re-sampling (SIR). Here, the posterior probability is represented as particles (a set of weighted, random samples), where a sample set is a discrete approximation of a probability distribution. Similar to Markov localization, MCL has also 2 phases of the algorithm. The first one is called robot motion and it constitutes of a random generation of new samples (drawn from the previous sample set) with the purpose of estimating the robot's location after the movement command. The second phase is represented by the sensor readings. This is done by re-weighting the sample set in a manner that computes Bayes rule in Markov localization. The authors also used a few uniformly distributed random samples after every estimation step, as it was practically helpful. This extra samples were very useful for relocalization, in case of a rare event where the robot does not know where it is anymore.

If one wants to put MCL in a category of algorithms, a suitable category would be online algorithms. It is also an any-time implementation (though one should keep in mind that more accurate results can be obtained in a longer time). However, the sampling step can be terminated in any moment. Moreover, the size of the sample set is decided on the fly. In general, the variance or the approximation error is higher when the sampling distribution and the approximated distribution with the weighted sample disagree more. The authors compensated the variance using larger sample set sizes in order to get an almost uniform error.

The MCL performance was evaluated in public spaces using multiple kinds of sensors. The accuracy of MCL was proven to be higher than the highest accuracy for a Markov localization (grid-based Markov localization), being at the same time less memory and computational expensive. In the majority of cases adaptive sampling performed similar to fixed sample sizes (but in cases where there are more kinds of uncertainties, adaptive sampling is better). Anyway, small sample sets are not good as they are infer a high error in the approximation. Large sample sets are not suitable as well, because they are computational expensive and not so many sensor items are processed in real-time. The best sample set size was proven to be between 1000 and 5000 samples (the accuracy of this being similar to the accuracy when using 4 cm resolution for grid localization, but that was infeasible for the computer capabilities at that time).

Conclusions

There are some similarities between the HMM approach for robot localization and the Monte Carlo approach. In [1], they also have a robot motion phase and a sensor reading phase. But, unlike in the HMM approach, the Monte Carlo localization is used in both phases, in the first phase to generate new samples that approximate the robot's position after the motion command, and in the second phase the sensor readings are computed by re-weighting the sample set.

The main drawback of the HMM approach is its computational cost. According to [2], if the state contains n discrete variables with d values each, the transition matrix would have size $O(d^{2n})$, leading to a computation time of $O(d^{2n})$ per update. Meanwhile, according to [1], the Monte Carlo approach brings a significant reduction in the amount of memory required. Also, according to [1], the Monte Carlo approach is more accurate because the state represented in the samples is not discretized.

Considering all of these, I would say that the HMM approach is good for the given task from the assignment, but in a more complex robot localization problem it would not be a suitable solution, as it requires a big amount of memory and it is not so accurate.

References

[1] Dellaert F. Fox D. Burgard W. and Thrun S. “Monte Carlo Localization: Efficient Position Estimation for Mobile Robots”. In: *AAAI-99 Proceedings* (1999).

[2] Norvig P. Russel S. *Artificial Intelligence - A Modern Approach.* Pearson Education, 2021. Chap. 14. Probabilistic Reasoning over Time. ISBN: 9780134610993.