IMPLEMENTATION OF ROBOT STATE ESTIMATION USING HIDDEN MARKOV MODELS

Mikhail Ivanov, Timur Uzakov

***Abstract*— The Hidden Markov Models is a key probabilistic concept that can be used in various machine learning (reinforcement learning) applications such as speech, handwriting and gesture recognition. In this work we tried to use the model in robotics plant by prediction of robot position in a maze based on initial position, sensor data (perceptions), and known transition and emission probabilities. Smoothing, filtering and Viterbi algorithms have been used in the work. We also applied linear algebraic approach to speed up computations, probabilities scaling to avoid underflow errors. The estimation turned out to be successful, the algorithms are working, and comparison was implemented. The average values of Manhattan distance mismatch are fluctuating around one maze cell distance, while the percent of accuracy is approximately 60%.**

Keywords – Hidden Markov Model, Smoothing, Filtering, Viterbi, Scaling

I. Introduction

The semestral task is aimed to teach us of calculating the hidden state probabilities using filtering, smoothing and Viterbi algorithms. Those algorithms are standalone tools to predict robot positions based on observations and known probabilities. The output of each algorithm is the most likely robot position or sequence of positions and its probabilities. For academical purpose, we know real robot position and generated possible sensor output. The real position was laterally compared with algorithms' outputs in terms of prediction evaluation. The comparison of algorithm's accuracy is presented in Experimental part (III). Initial realization of algorithms could be improved in terms of accuracy and speed. We applied matrix multiplication for speed improvement and achieved up to 60 times speed increase. The scaling helps to avoid underflow error for low probabilities on large step sequences. There is possibility to implement the task using an external library and we will provide an example how.

II. Methodology

All used methods rely on initial robot position, observations (sensor output), transition and emission probabilities.

A. Filtering

The filtering is interpretation of forward algorithm, where we compute probabilities in the forward series. Forward probability $P(X_t|e_1^t)$ can be found in following manner:

$P(X_t|e_1^t) = f_t = \alpha * P(e_t|X_t) * \sum_{x_{t-1}} P(X_t|X_{t-1}) * f_{t-1}$, where f_t is belief distribution in time, f_{t-1} is the previous belief distribution, α is scaling factor, $P(e_t|X_t)$ is probability emission X_t with evidence e_t, $P(X_t|X_{t-1})$ is probability of transition of X_t based on X_{t-1}

B. Smoothing

The smoothing is based on forward-backward algorithm computations. The probability $P(X_t|e_1^t)$ can be found as $P(X_t|e_1^t) = \alpha * f_k * b_k$, where f_k and b_k are forward and backward probabilities respectively. Backward probability can be found as

$b_k = \sum_{x_{k+1}} P(e_{k+1}|x_{k+1}) * P(x_{k+1}|X_k) * b_{k+1}$

C. Viterbi algorithm

Viterbi algorithm is based on computation of most likely sequence of states. It takes maximal argument among all possible states and apply the result on next steps in the sequence. So-called max message is recursively computed from previous message as

$m_t = P(e_t|X_t) * \max_{x_{t-1}} P(X_t|X_{t-1}) * m_{t-1}$

Algorithms above are presented and can be integrated using class `Robot.py` and `hmm_inference.py`

D. Matrices multiplication

From the equations above we can see that probabilities can be assigned iteratively. Thus, we can assign transition and emission probabilities to the matrix form and use matrix multiplication for probability calculations. The updated approach is presented in `RobotM.py` and `hmm_inferenceM.py`.

E. Scaling

We can observe underflow error on long sequences of incrementally descanting numbers. This can be fixed by scaling - we can use result normalization for obtained probabilities and hence avoid extremely low values. The approach can be run using `RobotM.py` and `hmm_inferenceMS.py` files.

F. External libraries

There are several options for HMM evaluation available online. However, we have chosen hmmlearn, since it is easy to install and work with. MultinominalHMM model was chosen that could solve the task provided with start probability, transition and emission probability matrices. The implementation of the procedure can be found in `robot_alg_lib.py` script.

III. Experiments

For the algorithm comparison and evaluation, we test every algorithm separately. Detailed experiment description could be found in Experimental Protocol (please see attachments). We started with evaluation of accuracy of different algorithms – Filtering, Smoothing and Viterbi. For this test we conducted 10 separate runs on 5 maps for 100 steps each. Secondly, we compared speed and performance of solution that use matrix multiplication with solution that use for-loops instead. We used time profiling function of PyCharm and conduct 5 runs of each on the same map of 100 steps each. Next, we tested performance of scaling, we conducted 5 runs on the same map of 1000 steps each. We have implemented pose estimation using hmmlearn and compared the sequences of estimations by Viterbi from hmmlearn and our initial implementation of the algorithm.

IV. Evaluation

A. Filtering, Smoothing, Viterbi

Algorithms evaluation is shown on the following figures(1-15). In the same conditions, the Smoothing shows better accuracy in average. We can also observe better performance on maze with obstacles. It is explained by a smaller number of squares of the same properties – sensor perceives most of the neighboring squares with the same output.

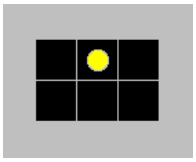

Fig. 1. Maze 1

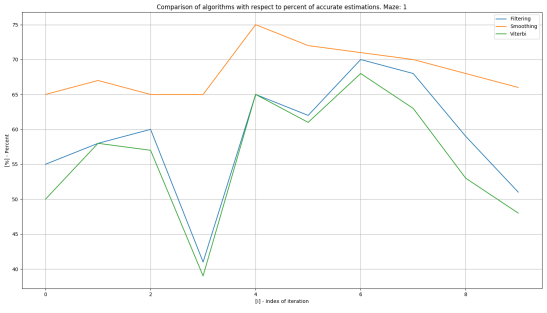

Fig. 2. Evaluation in terms of accuracy: maze 1

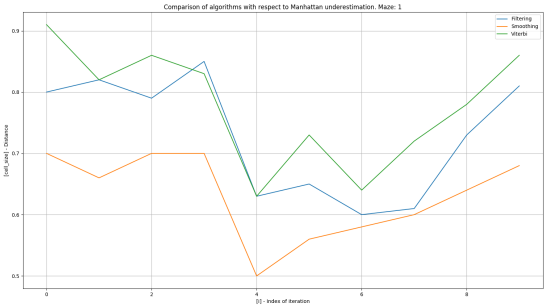

Fig. 3. Evaluation in terms of distance: maze 1

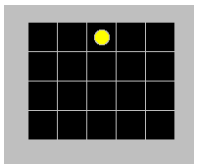

Fig. 4. Maze 2

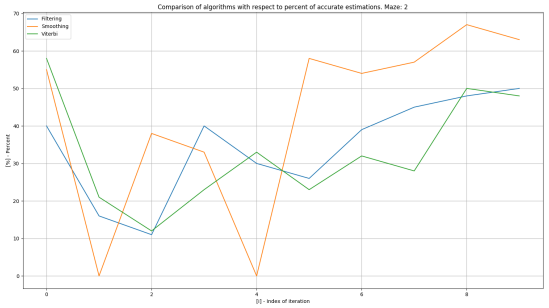

Fig. 5. Evaluation in terms of accuracy: maze 2

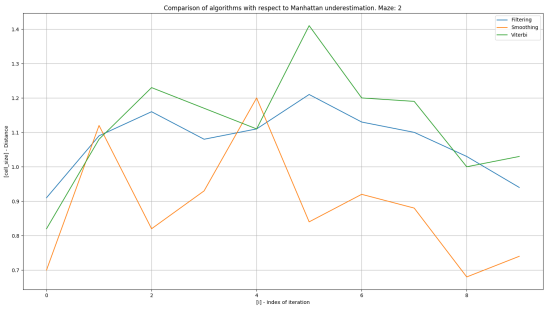

Fig. 6. Evaluation in terms of distance: maze 2

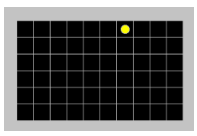

Fig. 7. Maze 3

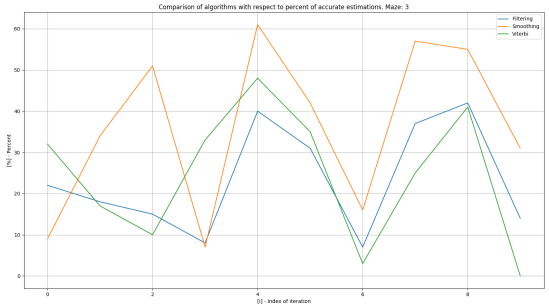

Fig. 8. Evaluation in terms of accuracy: maze 3

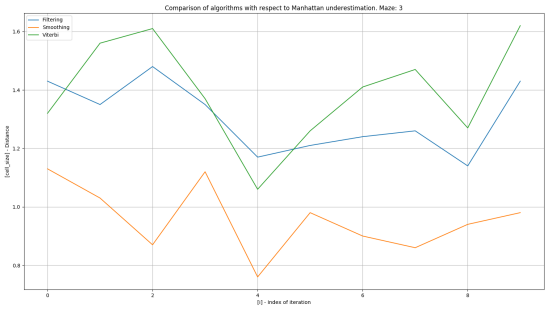

Fig. 9. Evaluation in terms of distance: maze 3

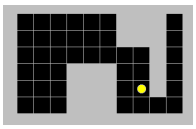

Fig. 10. Maze 4

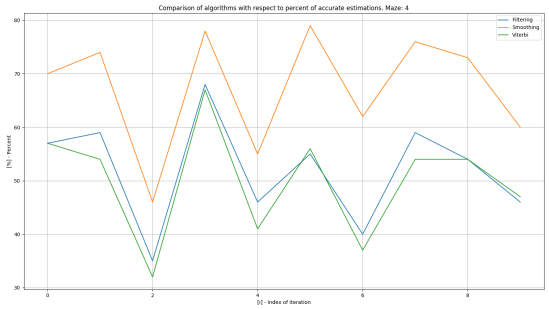

Fig. 11. Evaluation in terms of accuracy: maze 4

B. Matrix multiplication

Using matrix multiplication, we can cover nested for-loops by a single multiplication of matrices. Obtained speed increase can be observed in Table 1 (the runs are made on the same PC in a sequence, speed of computation will be different on different machines, data is suitable only for relative comparison).

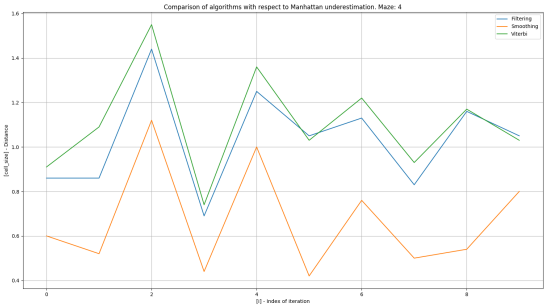

Fig. 12. Evaluation in terms of distance: maze 4

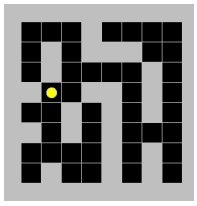

Fig. 13. Maze 5

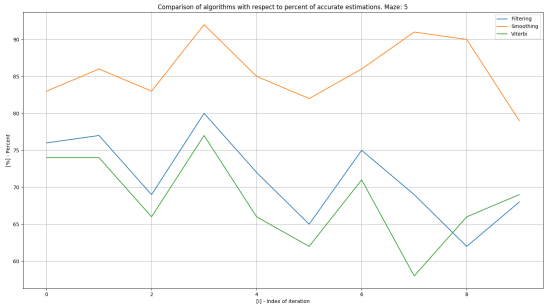

Fig. 14. Evaluation in terms of accuracy: maze 5

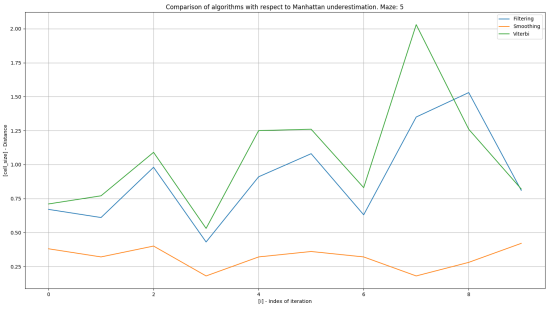

Fig. 15. Evaluation in terms of distance: maze 5

C. Scaling

It is important to notice that influence of scaling has become distinguishable due to the increase of number of steps. The experiment conducted with 1000 movements explicitly showed that scaling has little effect on filtering, due to little changes in already almost scaled algorithm. A slight improvement

TABLE I

PROFILING OF ALGORITHM WITH MATRIX MULTIPLICATION (MM) AND WITHOUT MATRIX MULTIPLICATION (W/O MM).

	Filtering run [ms]		Smoothing run [ms]		Viterbi run [ms]	
#	w/o MM	MM	w/o MM	MM	w/o MM	MM
1	5129	327	18832	379	5345	324
2	5157	324	18989	381	5432	324
3	5103	327	18702	382	5383	322
4	5126	321	18797	386	5197	322
5	5164	325	18762	390	5098	325
Mean	5136	325	18816	384	5291	323

can be seen in the smoothing algorithm, which is incredibly inaccurate in large number of steps. However, significant error reduction has been achieved by scaling in Viterbi algorithm: the accuracy doubled, while Manhattan error decreased four times.

TABLE II

ALGORITHM PERFORMANCE COMPARISON WITH SCALING AND WITHOUT SCALING. MAZE 4. RUN 1

	Manhattan error [cell size]		Accuracy [%]	
	No scalling	Scalling	No scalling	Scalling
Filtering	0.88	0.89	59	58
Smoothing	5.74	3.12	2	26
Viterbi	4.65	0.96	17	56

D. External library

The evaluation by external library required transfer of transition and emission Counters into matrix forms. After that, it is necessary to transform input sequence of observations into understandable by the model array of indexes. To do this, all combinations of observations are collected and numerated from 0 to 15, that allows to convert from combination such as "f,f,f,n" into 1. Next, this sequence of indexes is transposed and added to the model's decoder with specification: "Viterbi". The last step is to convert sequence of position indexes into positions themselves. To do this, it is essential to obtain a collection of all possible states, enumerate them and retrieve the position accordingly. We have compared the results of hmmlearn to our viterbi and there are slight deviations between the two and the accuracy is roughly the same - refer Table 3.

TABLE III

ROBOT POSITION ESTIMATIONS BY HMMLEARN AND OUR VITERBI ALGORITHM, TESTED ON MAZE 4

Real state	Hmmlearn	Our viterbi
(1, 2)	(1, 1)	(1, 1)
(2, 2)	(2, 1)	(1, 1)
(2, 3)	(3, 1)	(2, 1)
(2, 2)	(3, 2)	(3, 2)
(3, 2)	(3, 1)	(3, 1)
(3, 1)	(3, 2)	(4, 1)
(3, 2)	(2, 2)	(3, 3)
(2, 2)	(1, 2)	(1, 2)
(2, 3)	(1, 3)	(1, 3)
(3, 3)	(1, 2)	(1, 2)

V. CONCLUSIONS

Finally, the behavior of the three algorithms has been assesed, their performance has been observed and compared. According to our estimations, the Smoothing (or forward-backward) algorithm is considered to be the most accurate provided with smaller number of steps. However, the algorithm shows significantly larger number of mistakes when 1000 or more steps are due to compute. In this case, we would recommend referring to Filtering and/or Viterbi algorithms.

Matrix multiplication is essential, because of significant runtime reduction.

Scaling contributes to error reduction, which is especially observed in Viterbi algorithm performance.

External library has been also applied, it may take some time to get familiar with the API, nevertheless the estimations are not significantly different.

REFERENCES

[1] Forward–backward algorithm. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-13]. Available from: https://en.wikipedia.org/wiki/Forward%E2%80%93backward_algorithm , 2019

[2] Hidden Markov model. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-13]. Available from: https://en.wikipedia.org/wiki/Hidden_Markov_model , 2019

[3] Viterbi algorithm. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-13]. Available from: https://en.wikipedia.org/wiki/Viterbi_algorithm , 2019

[4] HMMlearn. Unsupervised learning and inference of Hidden Markov Models [cit. 2019-05-14]. Available from: https://hmmlearn.readthedocs.io/en/latest/, 2019