Semestral project 2: Robot in maze

Matěj Boxan, Can Gundogdu

***Abstract*— In this paper, our aim is to predict the robot position in a maze based on its initial position and sensor data with known emission and transition probabilities, while the environment is modelled as a Hidden Markov Model (HMM). The capabilities of algorithms such as Filtering, Smoothing and Viterbi are evaluated. The naive implementation was enhanced with a matrix multiplication, leading to a significant increase in computational efficiency. Underflow issues that arise from the multiplication of small floating-point numbers were solved by scaling in case of filtering and smoothing and by log-probabilities for the Viterbi algorithm. An alternative robot model, which considers the robot's orientation, was evaluated and compared to the original model.**

I. Assignment

The assignment is to estimate the robot position while considering its initial position and the data received from the direction sensors. Observations made on the results of the original implementation lead to several improvements of performance in terms of speed and accuracy of the estimated position. The results will be shown and discussed at the end of the paper.

II. Introduction

Hidden Markov Models are probabilistic frameworks where the observed data are modelled as a series of outputs (or emissions) generated by one of several (hidden) internal states. It is a significant probabilistic concept that can be used in diverse machine learning methods such as reinforcement learning, face expression, speech or handwriting recognition. In this paper, we will focus on an application of HMM where the goal is to predict the robot position based on sensor data and known probabilities. The output of such a prediction, obtained from Filtering, Smoothing and Viterbi algorithms, is the most likely robot position and possibly also the probabilities. The ground truth position is known from the simulation environment. This allows a comparison with the estimate and further evaluation of results.

In the Methodology section of this report, we will introduce the algorithms used to estimate the robots position in the maze and the enhancements we made to improve their performance in terms of both speed and accuracy. The Experiments section introduces the environment where the experiments were performed, together with the experimental protocol. Finally, in Discussion, we compare the evaluated approaches with regard to their correctness and speed.

III. Methodology

In this section, we focus on describing the methods used for the solution. Firstly, we briefly introduce Filtering, Smoothing and the Viterbi algorithm. Then a matrix representation of the transition and emission model is presented. Next, we describe a method to help with underflow issues caused by the multiplication of small numbers. Finally, we introduce and describe an alternative robot.

A. Filtering

The method is based on the forward algorithm which computes the probabilities in forward series. The goal is to estimate the last hidden state given all the observations. The forward probability $P(X_t|e_1^t)$ can be be found by the formula

$$P(X_t|e_1^t) = f_t = \alpha \cdot P(e_t|X_t) \cdot \sum_{X_{t-1}} P(X_t|x_{t-1}) \cdot f_{t-1}, \quad (1)$$

where f_t is the belief distribution in time t, α is the scaling factor, f_{t-1} is the previous belief distribution and $P(X_t|X_{t-1})$ is the probability of transition of X_t with respect to X_{t-1} and $P(e_t|X_t)$ is the probability emission X_t with the evidence e_t [2].

B. Smoothing

The goal of Smoothing is to compute the probability distribution over the past given the evidence up to the present. It is solved by the forward-backward algorithm [4], where the probability $P(X_t|e_1^t)$ can be obtained as

$$P(X_t|e_1^t) = \alpha \cdot f_k \cdot b_k. \quad (2)$$

The normalisation factor α and forward probability f_k were defined in the Filtering subsection and backward probability b_k can be computed as

$$b_k = \sum_{X_{k+1}} P(e_{k+1}|X_{k+1}) \cdot P(X_{k+1}|X_k) \cdot b_{k+1}. \quad (3)$$

C. Viterbi

The Viterbi algorithm finds a single best sequence of states, which is the most likely sequence. It works as taking the maximal argument among all states and find result on the next steps in the sequence where max message is estimated from previous message recursively [3]. The max message

$$m_t(X_t) = \max_{x_1^{t-2}} P(x_1^{t-2}, X_t, e_1^t) \quad (4)$$

can be for $t \geq 2$ expressed as

$$m_t(X_t) = P(e_t|X_t) \cdot \max_{X_{t-1}}[P(X_t|x_{t-1}) \cdot m_{t-1}(x_{t-1})], \quad (5)$$

where the recursion is initialised as $m_1 = P(X_1, e_1)$.

D. Matrix Multiplication

As can be examined from the equations above, the probabilities can be assigned iteratively. It is possible to assign emission and transition probabilities in the form of a matrix and use matrix and vector multiplication instead of for loops. The transition matrix is

$$A = [a_{ij}] = [P(X_t = x_j|X_{t-1} = X_i] \quad (6)$$

and the emission model is

$$B = [b_{ik}] = [P(E_t = e_k|X_t = X_i]. \quad (7)$$

The implementation profits from numpy library [1] and can be found in the `hmm_inference_Matrix.py` file and `robot_Matrix.py` file.

E. Scaling

For all the methods mentioned above, underflow problems occur for long sequences of incrementally decreasing numbers. An approach we took to solve this issue is called Scaling. By normalising results for estimated probabilities, we can avoid very low values that cause the underflow. The fixed implementation can be found in `hmm_inference_scaling.py`, which expands the matrix implementation `hmm_inference_Matrix.py` and should be thus used with `robot_Matrix.py`.

F. Modified Viterbi algorithm

Same underflow issues arise also for the Viterbi algorithm. Rabiner [3] provides a modified algorithm, which replaces multiplication with sum of logarithms of probabilities. Initially, we set

$$m_1 = \log(P(e_t|X_t)) + \sum_{X_{t-1}} \log(P(X_t|x_{t-1})) + \log(P(X_{t-1}|e_1^{t-1})) \quad (8)$$

and the recursion step is

$$m_t(X_t) = \log(P(e_t|X_t)) + \max_{X_{t-1}}[\log(P(X_t|x_{t-1})) + \log(m_{t-1}(x_{t-1}))]. \quad (9)$$

G. A modified robot model

The original robot model does not take into account robot orientation. The robot can thus move freely in any direction, and the sensor readings are not restricted by its heading. A modified robot model which uses the concept of orientation is implemented in `robot2.py`, and we will further describe it in the following paragraphs.

While for the original model a state was unambiguously defined by a position $[x, y]$ in the map, this approach needs to be extended to account for robot's orientation. Our solution uses robot pose, defined as $[x, y, o_x, o_y]$, where $[o_x, o_y] \in \{[-1, 0], [0, 1], [1, 0], [0, -1]\} = \{\text{NORTH, EAST, SOUTH, WEST}\}$. This approach allows for easy motion in the robot's direction, where the next state in time $t + 1$ is $s^{t+1} = [[x, y]^t + [o_x, o_y]^t, [o_x, o_y]^t]$. Implementation of this can be found in function `add_movement()`. Apart from motion, the other default actions are left and right turn. The default probabilities are 0.5 for the motion and 0.25 for both left and right turn.

Since the modified robot model sensors are limited to the left, right and front side, we need to take this into account when observing the environment. For example, if the robot faces NORTH, the left side relates to WEST and the right to the EAST direction. This is handled in function `add_sensor_dir()`. The possible states of the modified robot in a map are equal to all robot orientations for all free cells of the map.

IV. Experiments

The experiments was conducted to to emphasise two characteristics of evaluated algorithms - speed and accuracy, measured as Hit and Miss rate and the Manhattan distance. The Hit and Miss rate V_{hm} [%] is defined as

$$V_{hs} = 100 \cdot \frac{V_h}{V_h + V_m}, \quad (10)$$

where V_h and V_m are the numbers of correctly and incorrectly estimated positions (or poses for the modified robot model). The Manhattan distance md of two points $p_1 = [x_1, y_1]$, $p_2 = [x_2, y_2]$ in a 2D grid is defined as $md(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$. The figures in this section display an increasing value of sums of Manhattan distances d, for which in time t the value $d_t = d_{t-1} + md_t$.

The experiments were run on 5 different mazes (Figures 1-5). The random seed was set to a fixed value between different experiments to enable a cross experiment comparison. In each experiment, all 5 mazes were tested several times and the results were then averaged - this will be further referred to as *iterations*. The number of steps *n_steps* corresponds to a length of sequence of time slices the robot performed in the maze. Based on the purpose of performed experiments, we offer a graph of Manhattan distance, a table of Hit and Miss rate, a table of time durations or a combination of those results.

5 different configurations for experiments were identified. Firstly, the original, naive implementation of Filtering, Scaling and Viterbi algorithm was evaluated. This

corresponds to the file `hmm_inference.py`. This naive representation was then applied to the modified robot model (`robot2.py`). As the next step, we evaluated time related improvements rising from the use of matrix multiplication (file `hmm_inference_Matrix.py`), together with the default robot model). To demonstrate the benefits of scaling, the experiments were then performed on long sequences (file `hmm_inference_scaling.py`). Finally, the implementation of Viterbi algorithm using log-probabilities was evaluated (file `hmm_inference2.py`).

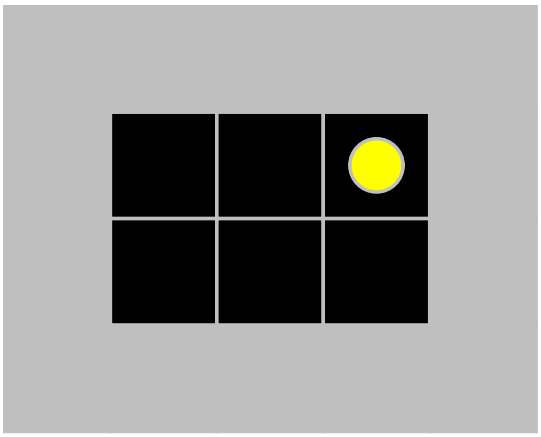


Fig. 1: Maze 1 - Empty grid with dimensions 3x2

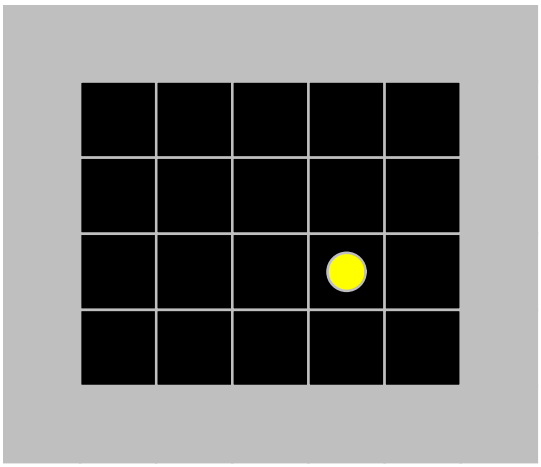


Fig. 2: Maze 2 - Empty grid with dimensions 5x4

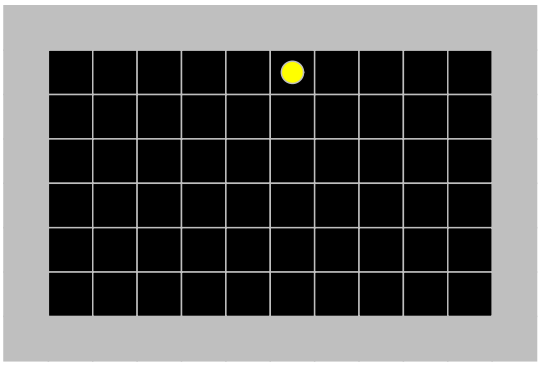


Fig. 3: Maze 3 - Empty grid with dimensions 6x10

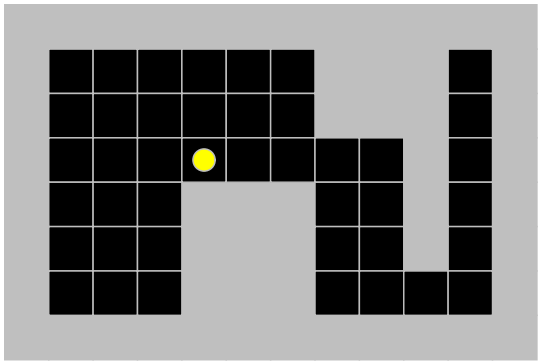


Fig. 4: Maze 4 - Large obstacles in a grid with dimensions 6x10

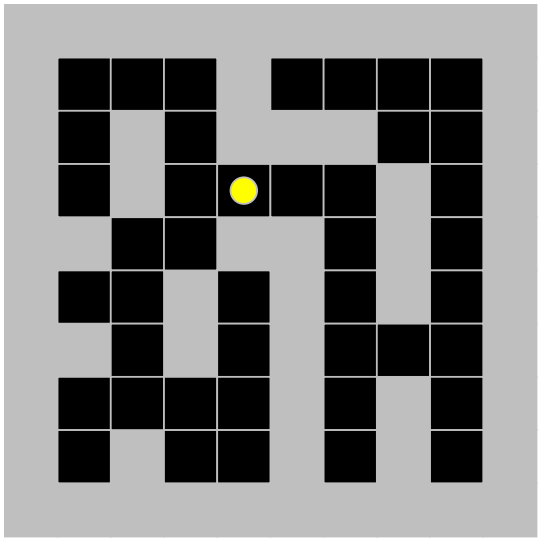


Fig. 5: Maze 5 - Maze with dimensions 8x8

A. Original implementation

The comparisons of the initial default algorithms should be conducted to overview their performance. The parameters used in this experiment were: number of iterations 20, number of steps 200. From Figures 6-10, it is obvious that Viterbi and smoothing (forwardbackward) algorithms are performing better than the forward algorithm in terms of accuracy for both Hit and Miss and Manhattan distance. In terms of speed (Table II), forward and Viterbi are faster than smoothing, while Viterbi is the fastest algorithm by a small margin. Each method performed the slowest on Maze 3 (Fig. 3). This is due to the fact that it contains the largest number of states the robot is allowed to reach, since it does not contain any obstacles.

For the sake of simplicity, further experiment results will not contain figures for all the mazes. Instead, we chose 3 which we think contain most of information. Among those, we opted for Maze 2, as it is a grid with no obstacles. Maze 4, which is partially filled with obstacles, but still contain large empty space. Finally, Maze 5 is defined by narrow corridors.

TABLE I: Hit and Miss score for the original implementation

maze	forward	forwardbackward	viterbi
Maze 1	63.75 %	68.40 %	66.62 %
Maze 2	34.42 %	50.08 %	39.42 %
Maze 3	23.15 %	35.50 %	27.73 %
Maze 4	56.80 %	73.95 %	72.45 %
Maze 5	77.85 %	88.28 %	87.47 %

TABLE II: Time durations for the original implementation

maze	forward [ms]	forwardbackward [ms]	viterbi [ms]
Maze 1	65.5	184.1	63.2
Maze 2	615.6	2270.6	614.6
Maze 3	5466.2	25126.5	5345.5
Maze 4	2682.5	10470.0	2666.5
Maze 5	2628.8	9704.9	2598.4

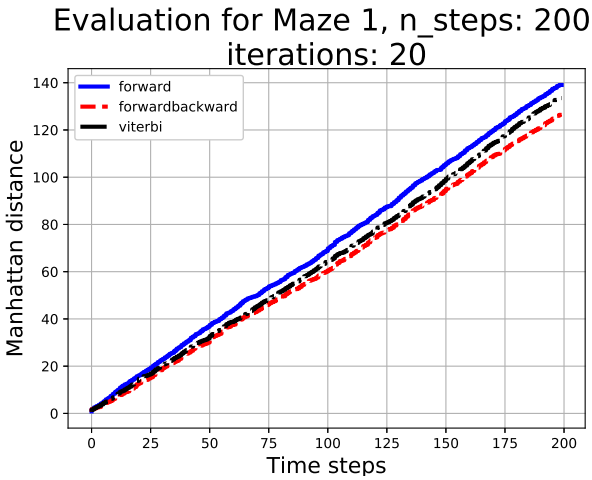


Fig. 6: Maze 1 - Evaluation of the original implementation

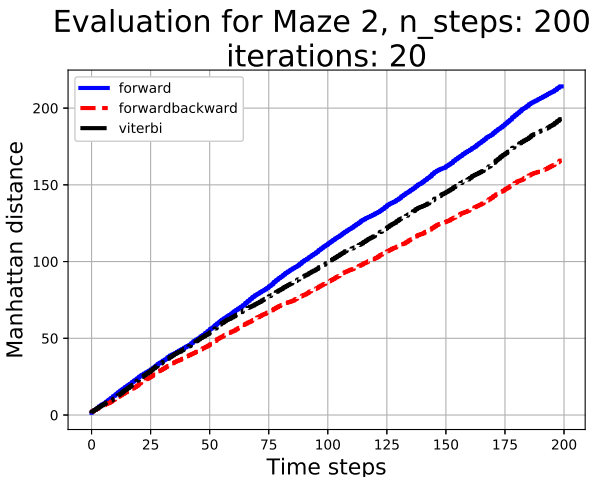


Fig. 7: Maze 2 - Evaluation of the original implementation

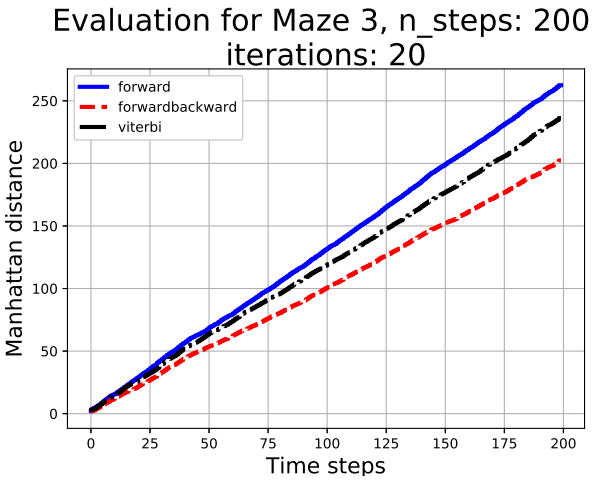


Fig. 8: Maze 3 - Evaluation of the original implementation

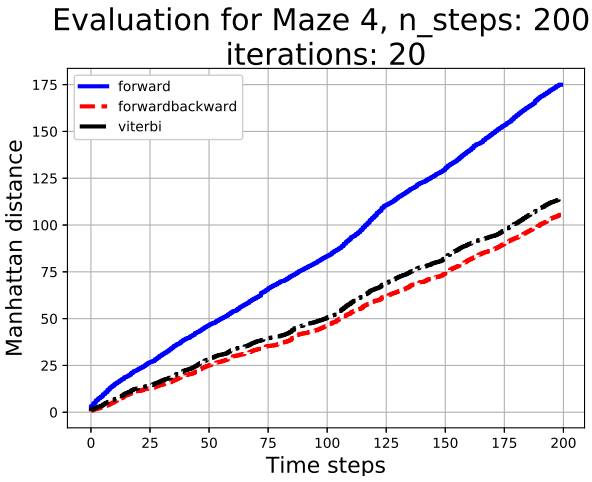


Fig. 9: Maze 4 - Evaluation of the original implementation

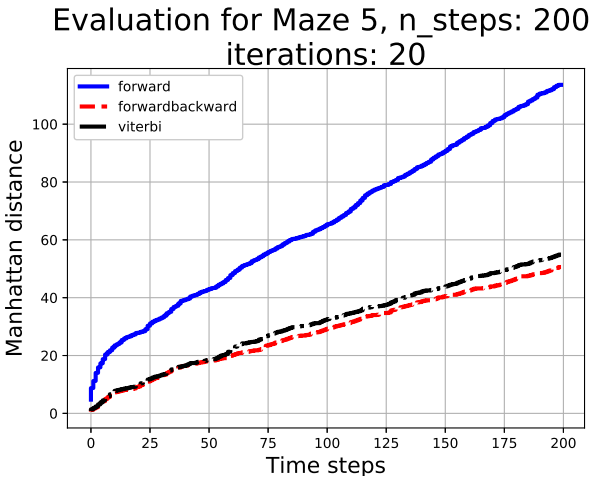


Fig. 10: Maze 5 - Evaluation of the original implementation

B. Modified robot model

The modified robot model `robot2.py` was tested with the same configuration, as the original one. The number of averaged samples *iterations* was set to 20 and *n_steps* to 200. The overall performance in terms of Manhattan distances (Figs. 11, 12, 13) is without exception worse than the original implementation. The Hit and Miss rate given in Table III confirms this. The number of possible states and thus the search space is much larger for this model, so it is not surprising that the computation is approximately 10 times slower (Table IV).

TABLE III: Hit and Miss score for the modified robot model

maze	forward	forwardbackward	viterbi
Maze 1	32.48 %	43.48 %	36.58 %
Maze 2	44.95 %	68.62 %	66.12 %
Maze 3	50.73 %	84.22 %	83.62 %
Maze 4	63.80 %	84.03 %	83.25 %
Maze 5	57.30 %	77.75 %	77.42 %

TABLE IV: Time durations for the modified robot model

maze	forward [ms]	forwardbackward [ms]	viterbi [ms]
Maze 1	721.2	2295.6	711.8
Maze 2	7876.3	28338.0	7721.5
Maze 3	68411.4	294882.0	66409.6
Maze 4	32652.7	122324.9	31764.4
Maze 5	32458.0	114298.9	31378.9

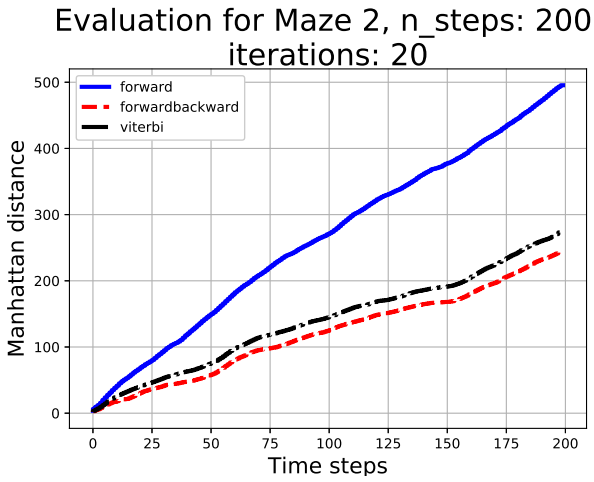


Fig. 11: Maze 2 - Evaluation of the modified robot model

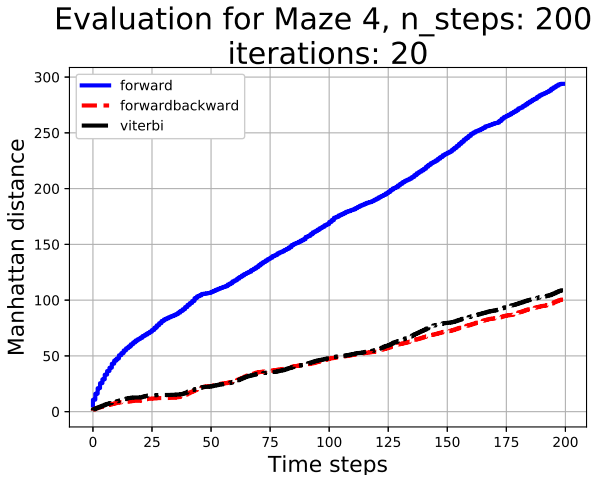


Fig. 12: Maze 4 - Evaluation of the modified robot model

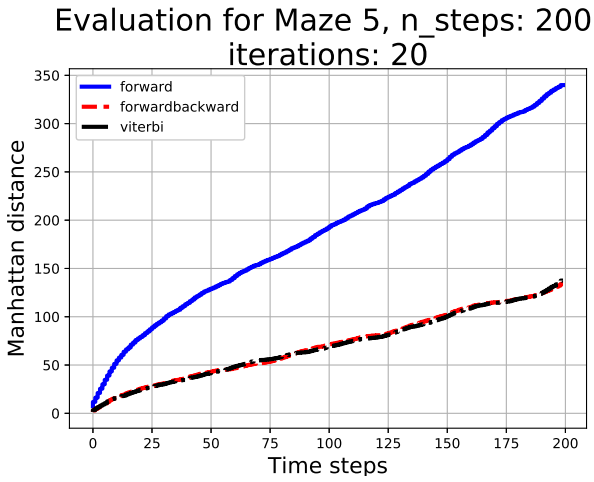


Fig. 13: Maze 5 - Evaluation of the modified robot model

C. Matrix multiplication

The aim of the use of matrix multiplication is to improve the speed performance of the algorithms, by avoiding the necessity to use for loops. To test this attribute we will compare the time results of the matrix multiplication implementation of inference file `hmm_inference_Matrix.py` with default inference file `hmm_inference.py`. The experiment was firstly performed with 200 steps and 20 iterations. The results in Table VII show that the matrix multiplication approach is far more faster than the naive implementation version which is presented in Table II. The Hit and Miss rate, presented in Table V, is similar to the results obtained for the original algorithm (Table I. In Table VI the reader can observe a radical deterioration of results. The same phenomenon, visible also on Figures 14, 15 and 16, is caused by underflow errors and will we further discussed in the next subsection.

TABLE V: Hit and Miss score for the implementation using Matrix multiplication for 200 steps

maze	forward	forwardbackward	viterbi
Maze 1	60.60 %	66.50 %	59.45 %
Maze 2	31.00 %	52.50 %	29.00 %
Maze 3	20.05 %	33.65 %	15.90 %
Maze 4	50.55 %	70.05 %	48.80 %
Maze 5	74.15 %	88.65 %	73.60 %

TABLE VI: Hit and Miss score for the implementation using Matrix multiplication for 500 steps

maze	forward	forwardbackward	viterbi
Maze 1	51.66 %	14.19 %	43.36 %
Maze 2	29.68 %	3.34 %	22.82 %
Maze 3	25.91 %	16.03 %	14.90 %
Maze 4	41.99 %	1.65 %	35.53 %
Maze 5	65.04 %	2.59 %	60.92 %

TABLE VII: Time durations for the implementation using Matrix multiplication

maze	forward [ms]	forwardbackward [ms]	viterbi [ms]
Maze 1	31.9	57.8	43.0
Maze 2	68.5	135.1	107.2
Maze 3	181.8	355.4	295.3
Maze 4	129.1	259.5	209.2
Maze 5	132.8	265.5	214.1

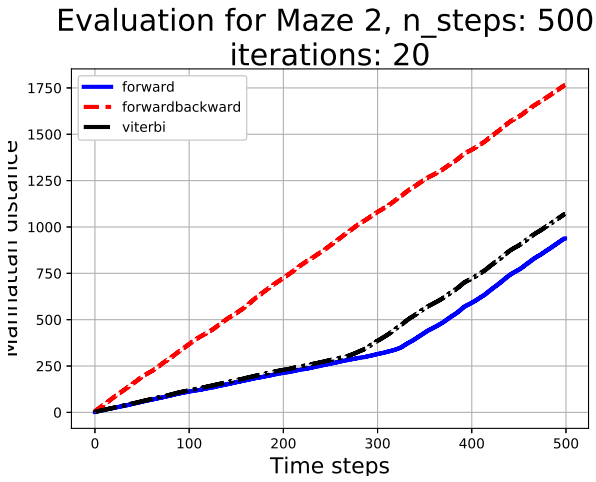


Fig. 14: Maze 2 - Evaluation of the implementation using Matrix multiplication

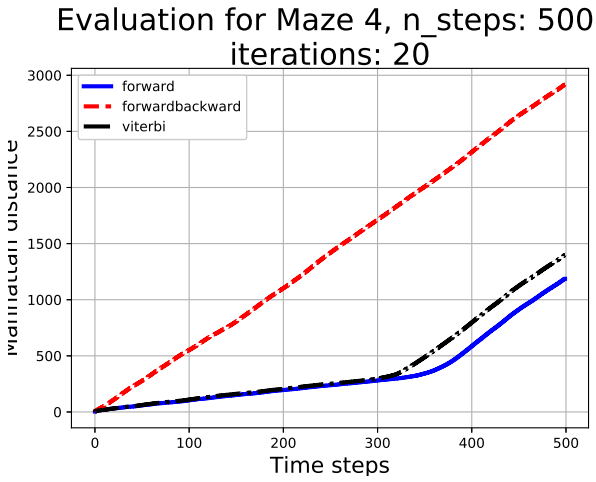


Fig. 15: Maze 4 - Evaluation of the implementation using Matrix multiplication

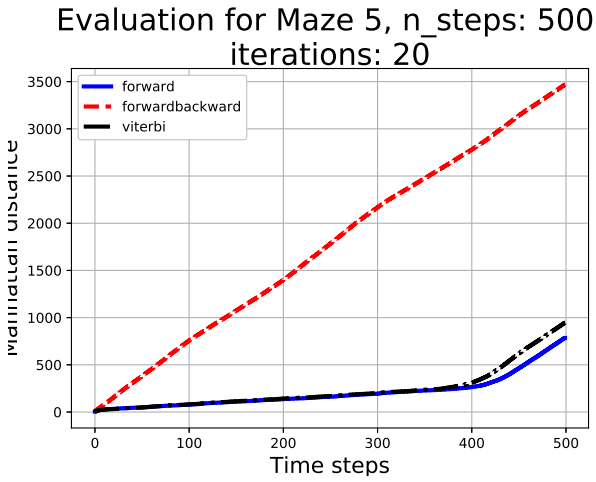


Fig. 16: Maze 5 - Evaluation of the implementation using Matrix multiplication

D. Scaling

In this experiment, we once again used 500 steps and 20 iteration. We expect to see that Scaling prevented underflow errors and improved performance significantly. For this we compared inference file with matrix implementation (file `hmm_inference_Matrix.py`) and inference file with both matrix and scaling implementation (file `hmm_inference_scaling.py`). From Table VI we can observe that Hit and Miss score improved in comparison to Table I throughout all mazes. The highest improvement is for smoothing (forwardbackward) algorithm. If we take a look at the Figures 7, 9, 10 we can notice that after the number of steps exceeds 300-400 the Manhattan distance increases dramatically. On the other hand, from Figures 14, 15, 16 we can deduct that this phenomenon disappeared. The total Manhattan distance decreased for all the algorithms. In terms of speed comparison the scaling is a bit slower than the default matrix implementation but it is still way faster than the original naive implementation.

TABLE VIII: Hit and Miss score for the implementation using Scaling

maze	forward	forwardbackward	viterbi
Maze 1	60.61 %	67.45 %	57.33 %
Maze 2	46.05 %	58.48 %	41.55 %
Maze 3	26.62 %	42.95 %	19.48 %
Maze 4	57.99 %	71.59 %	55.85 %
Maze 5	76.21 %	87.84 %	75.47 %

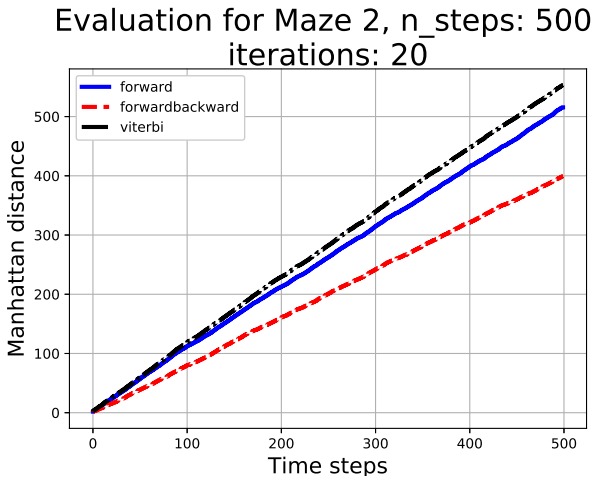


Fig. 17: Maze 2 - Evaluation of the implementation using Scaling

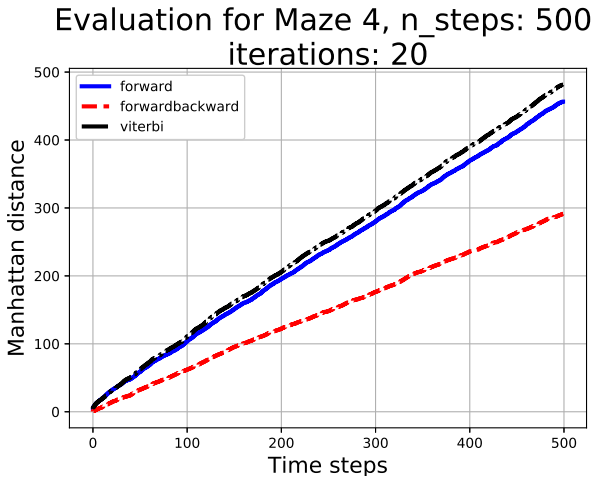


Fig. 18: Maze 4 - Evaluation of the implementation using Scaling

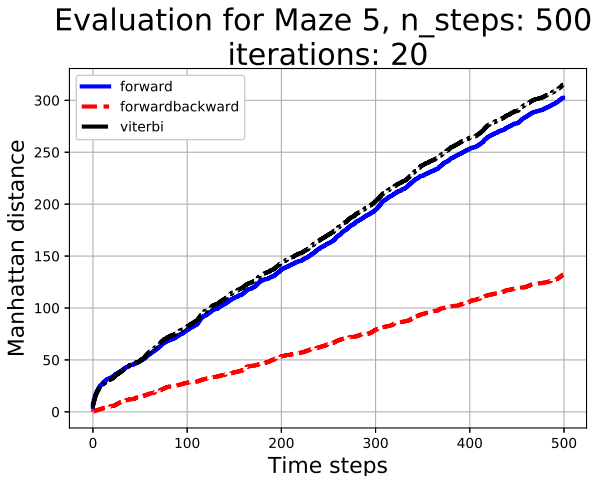


Fig. 19: Maze 5 - Evaluation of the implementation using Scaling

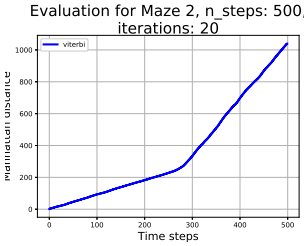
E. Viterbi and log-probabilities

In this experiment, we will demonstrate how the use of log-probabilities improved the accuracy of Viterbi algorithm. The experiments were performed with 20 iterations and sequences of length 500. Table IX shows a Hit and Miss score of the two approaches. The reader can observe significant improvement in the score for all of the 5 mazes.

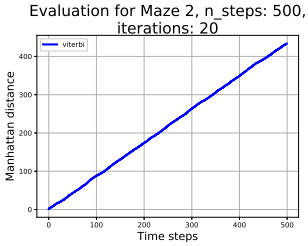
The computation of Manhattan distance provide similar results. On figures 20-22, each row consists of two images. The left one is the result of the original Viterbi algorithm, the right one is the output of enhanced implementation using log probabilities. These figures clearly show that around step 300, the error starts to rise significantly in the case of the original Viterbi algorithm. The use of log probabilities reduced the error from 1000 to just over 400 for Maze 2, from 2500 to 300 for Maze 3 and from 200 to about 140 for Maze 4.

TABLE IX: Hit and Miss score iter: 20, steps: 500

maze	viterbi	viterbi log probs
Maze 1	48.16 %	62.61 %
Maze 2	27.41 %	55.13 %
Maze 3	11.26 %	29.58 %
Maze 4	26.27 %	70.12 %
Maze 5	43.60 %	87.43 %

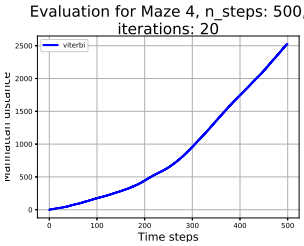


(a) Original

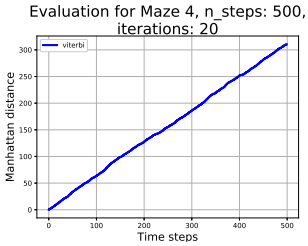


(b) Log probabilities

Fig. 20: Maze 2 - Evaluation of accuracy improvement of Viterbi algorithm

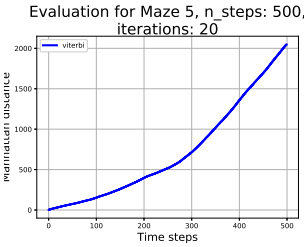


(a) Original

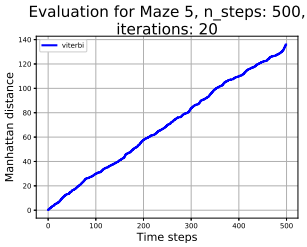


(b) Log probabilities

Fig. 21: Maze 4 - Evaluation of accuracy improvement of Viterbi algorithm



(a) Original



(b) Log probabilities

Fig. 22: Maze 5 - Evaluation of accuracy improvement of Viterbi algorithm

V. Discussion

The 5 tested configurations provided several findings. Generally, the most difficult and time-demanding maze turned to be Maze 3. The alternative robot model proved to be more time-demanding than the original one. This is not a surprise since the space of possible states is much larger in its case since it considers robot orientation. Matrix multiplication improved the speed of the computation significantly. Underflow errors which start to arise between step 300 and 400 can be suppressed by scaling in case of Filtering and Smoothing, or with the use of log probabilities in case of Viterbi algorithm. The algorithm that profited the most from scaling was Smoothing, even as the results for the other also improved significantly. Thus the algorithm the most severely affected by floating-point errors was Smoothing.

VI. Conclusion

In this semestral task, we applied the knowledge of Hidden Markov Models on a problem of robot position estimation in a maze. The original naive implementation was enhanced with matrix multiplication, which greatly improved the computational speed. Further changes concerning scaling and log-probabilities lead to a significant improvement of stability of results for sequences longer than 300 time slices. An alternative robot model was introduced and compared to the original one. It was shown that the modification of the robot model led to a radical increase in computation time. The use of matrix multiplication and scaling sped the execution time tenfold and improved the Hit and Miss rate by tens of per cent.

References

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] B. H. Juang Lawrence R. Rabiner. "An introduction to hidden Markov models". In: *IEEE ASSP Magazine* (1986), pp. 4–15.

[3] L.R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[4] Stuart Russell and Peter Norvig. "Artificial Intelligence A Modern Approach 3rd Edition". In: *Pearson Education/Prentice-Hall* (2010).