B(E)3M33UI — Semestral project 2: Robot localization in a maze

Petr Pošík

April 24, 2017

1 Introduction

This is a **team project**; students shall work in groups of two. The goal is to apply the knowledge and algorithms of Hidden Markov Models to tasks in somewhat more realistic domain: robot localization in a maze!

Your task is to evaluate various approaches to the robot localization problem when the environment is modeled as HMM, extend/improve the basic solution in a chosen way, and describe your solution and results in a report.

2 Robot in a maze

Until now, we have mostly played with a very simple Weather-Umbrella domain. This project aims at a bit more realistic domain, robot in a maze, see Fig. 1.

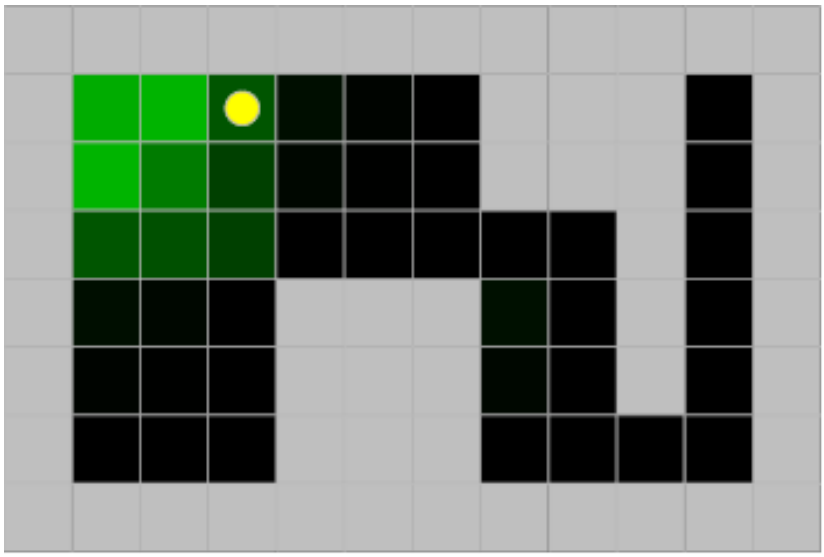

Figure 1: Example of the maze environment with the robot in it. The maze is a rectangular grid, robot is shown as a yellow circle, and the beliefs over the states are shown in shades of green.

Imagine you have a patrolling robot which moves in an indoor environment (in a "maze"). You have a map of the maze, but neither the maze, nor the robot is equipped with sensors of the absolute position of the robot. You have to estimate its position, i.e. estimate its hidden state, from a sequence of observations.

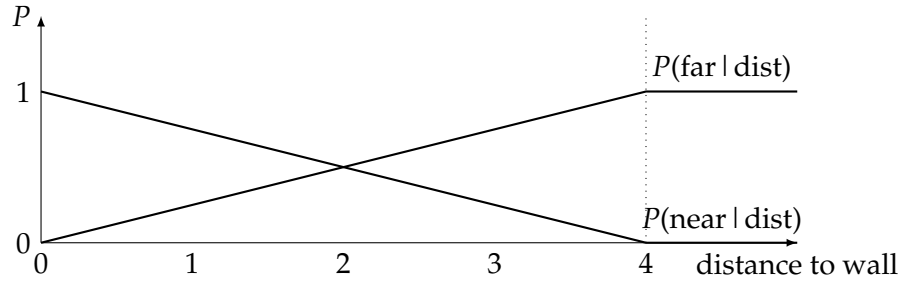

Figure 2: The dependence of sensor emission probabilities on the distance to wall

2.1 Robot movements

Because you do not want the robot movements to be predictable, they are chosen stochastically. The robot can move one step north, east, south, or west. Each time slice, the robot issues a *move* command, but the actual movement direction is chosen randomly. The probability distribution over these directions is known and does not have to be uniform. Moreover, in many positions, only some of these directions are admissible. In that case, the robot chooses only among the admissible directions. (This information shall be sufficient to compute the transition model $P(X_t|X_{t-1})$ where X_t is the position of the robot at time t.)

2.2 Robot observations

The robot is equipped with four "near-far sensors" which provide a very rough information about the distance to the nearest walls in specified directions. The output of each of the four sensors is either `'n'` or `'f'`, meaning that the wall is *near* or *far*. Moreover, the sensor information is noisy in the sense that the probability of returning symbol `'n'` grows with decreasing distance to wall and vice versa for symbol `'f'` (see Fig. 2). The four sensors are independent. There are $2^4 = 16$ possible observations: from (n,n,n,n) to (f,f,f,f). (This shall be sufficient to compute the emission model $P(E_t|X_t)$.)

Example: Imagine the robot is situated at position x_t where the distances to the closest wall in the north, east, south, and west directions are (0, 1, 2, 5). According to the functions for $P(near)$ and $P(far)$ in Fig. 2, the probabilities of individual observations are as follows:

Observation e_t				Probability
North	East	South	West	$P(e_t\|x_t)$
*	*	*	n	0 (because *near* in western direction is impossible)
n	n	n	f	$P(n\|0)P(n\|1)P(n\|2)P(f\|5) = 1 \cdot 0.75 \cdot 0.5 \cdot 1 = 0.375$
n	n	f	f	$P(n\|0)P(n\|1)P(f\|2)P(f\|5) = 1 \cdot 0.75 \cdot 0.5 \cdot 1 = 0.375$
n	f	n	f	$P(n\|0)P(f\|1)P(n\|2)P(f\|5) = 1 \cdot 0.25 \cdot 0.5 \cdot 1 = 0.125$
n	f	f	f	$P(n\|0)P(f\|1)P(f\|2)P(f\|5) = 1 \cdot 0.25 \cdot 0.5 \cdot 1 = 0.125$
f	*	*	*	0 (because *far* in northern direction is impossible)

3 Software

You are provided with several modules that should help you start. The two main modules are

- `robot.py`, which implements the "robot in a maze" environment as a HMM, and
- `gui.py`, which provides a graphical user interface useful for interactive work with the codes and for studying interesting cases.

3.1 Class `Maze`

Class `Maze` in module `robot.py` represents the maze a robot will be situated in. The representation of a maze is just a text file using `'#'` as the character for a wall, so that you can easily create your own mazes (you can find them in subdirectory `mazes`). A maze can be loaded from a text file and printed:

```
>>> from robot import *
>>> m = Maze('mazes/rect_6x10_obstacles.map')
>>> print(m)
############
#      ### #
#      ### #
#        # #
#   ###  # #
#   ###  # #
#   ###    #
############
```

Among other things, you can also ask a maze object, whether a certain position is free or contains a wall, and what is the distance from certain position to the closest wall in certain direction (`NORTH`, `EAST`, `SOUTH`, and `WEST` are just symbolic names for direction vectors defined in `robot.py`):

```
>>> m.is_free((0,0)), m.is_wall((0,0))
(False, True)
>>> m.is_free((1,1)), m.is_wall((1,1))
(True, False)
>>> NORTH, EAST
((-1, 0), (0, 1))
>>> m.get_dist_to_wall((1,1), NORTH)
0
>>> m.get_dist_to_wall((1,1), EAST)
5
```

3.2 Class `Robot`

Class `Robot` in module `robot.py` implements our `HMM` interface. It represents a robot that knows in what maze it is situated, knows its own position,[1] and provides its transition and emission models.

An instance of a Robot can be created like this:

```
>>> from robot import *
>>> robot = Robot()
```

This code creates a default robot, i.e. robot with near-far sensors in all 4 directions, and with 0.25 probability of moving to each of the 4 directions. You can create a modified robot e.g. with less sensors and with different movement probabilities like this:

```
>>> from robot import *
>>> move_probs = {NORTH: 0.5, EAST: 0.5, SOUTH: 0, WEST: 0}
>>> robot = Robot(sensor_directions=(NORTH, WEST), move_probs=move_probs)
```

A robot created this way, however, is not placed in any maze. To finish its initialization, we have to set the maze and place the robot to certain position.

[1] You may wonder why we should estimate the position if the robot knows it. Robot is a HMM which serves us as a data generator. We then will pretend that we do not know the correct positions and will try to estimate them. After the estimation, we can compare the estimated positions with the true ones.

```
>>> robot.maze = Maze('mazes/rect_6x10_obstacles.map')
>>> robot.position = (1, 1)
```

The robot instance will serve us as

- a **data generator**:

```
>>> robot = Robot()
>>> robot.maze = Maze('mazes/rect_6x10_obstacles.map')
>>> robot.position = (1, 1)
>>> states, observations = robot.simulate(n_steps=5)
>>> for i, (s, o) in enumerate(zip(states, observations)):
...     print('Step:', i+1, '| State:', s, '| Observation:', o)
Step: 1 | State: (1, 2) | Observation: ('n', 'f', 'f', 'n')
Step: 2 | State: (1, 3) | Observation: ('n', 'f', 'f', 'f')
Step: 3 | State: (1, 2) | Observation: ('n', 'f', 'f', 'n')
Step: 4 | State: (2, 2) | Observation: ('n', 'f', 'f', 'n')
Step: 5 | State: (2, 3) | Observation: ('n', 'f', 'f', 'f')
```

- a source of the transition and emission probabilities:

```
>>> robot.pt((1,1), (1,2)), robot.pt((1,1), (2,1))
(0.5, 0.5)
>>> robot.pe((1,1), ('n','n','n','n')), robot.pe((1,1), ('n','f','f','n'))
(0.0, 1.0)
```

Hint: Further examples using `Robot` objects can be found in module `test_robot.py`.

3.3 Graphical user interface

Module `gui.py` contains a graphical user interface[2] you can use to get an intuition of the behavior of the robot and the inference algorithms, and to explore some special cases in detail. It contains some control widgets in the left part and a maze display in the right part, see Fig. 3

The interface shall allow you to:

- load a maze from a text file,
- generate a random path, i.e. sequence of states and observations, from a user-specified or randomly selected starting point, save and load generated paths,
- load a solution, pressumably file `hmm_inference.py` or any other Python file containing at least one of functions `forward()`, `forwardbackward()`, and `viterbi()`,
- choose the inference algorithm which results shall be shown in shades of green in the grid,
- move in time, i.e. to choose the time frame for which the state is shown in the grid,
- set the size of the grid cell,
- save the grid state as an EPS image (these can be converted to PDF or other image formats e.g. by using `convert` from ImageMagick free library),
- etc.

In general, you shall load the maze, generate random path, and load solution. Afer that, you can explore the results of the chosen inference algorithm, or compare the results of the inference algorithms in a single time frame.

[2]The GUI is build using `tkinter` module which shall be part of Python standard library. But, on some systems it may be missing. If you have troubles running it, try Google, or contact your teacher.

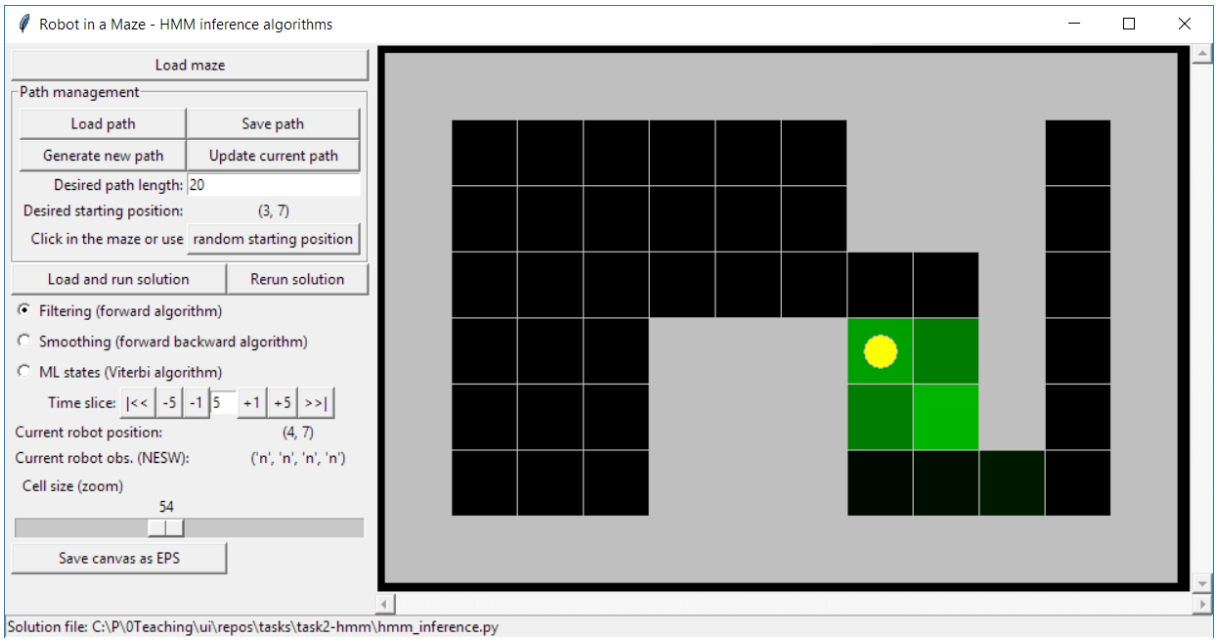

Figure 3: GUI for robot-in-a-maze domain. For illustration, the actual form may change as new features are added.

4 HMM project tasks

Although we provide a GUI for this task, it is meant only as a support tool. You should not solve the tasks listed below just by playing with the GUI; you shall create Python scripts which directly show without any human interaction what you have analyzed/achieved.

4.1 Mandatory part

In the two lab exercises aimed at HMMs, your task was to implement 3 algorithms as functions:

- `forward()`, which implements the forward algorithm for filtering,
- `forwardbackward()`, which implements the forward-backward algorithm for smoothing, and
- `viterbi()`, which implements the Viterbi algorithm for finding the most likely sequence of states.

Now, you shall be in a state when you have naive implementations of these functions working for the Weather-Umbrella domain.

Task 1: Check that your implementations of the above 3 functions in `hmm_inference.py` also work for the "Robot in a maze" domain. Update the implementation, if necessary, such that it works for both domains. Describe the implementations in your report, if you think there is something special about them the reader shall be aware of.

Hints:

- If you strictly coded to the general HMM interface and did not use any specifics of the Weather-Umbrella domain in `hmm_inference.py`, you shall have no problems applying the functions to a new HMM instance.

- Especially for longer observation sequences, the algorithms may suffer from numerical errors (underflow issues). Do not be surprised by that, rather observe when and in what circumstances do these issues emerge.

All 3 functions can be used to estimate the position of the robot in a maze. For filtering and smoothing, you get for each time slice a belief distribution over states (which expresses the probability that the robot is at certain position given the evidence). For the most likely sequence of states, you have a single state (a degenerate distribution over states) for each time slice which is part of the most likely sequence. It thus makes sense to compare these 3 approaches regarding their precision of their estimates.

Task 2: Design your own experimental protocol for assessing and comparing the precision of the estimates provided by the three functions, and perform their comparison. Describe the protocol and the results in the report.

Hints:

- This is not an easy task and you should think about it a while. (But the task is very close to an engineering practice.) You have to decide a number of things, most importantly:
- What set (class) of mazes will you consider? Think about maze size, maze type (empty, isolated obstacles, narrow corridors, etc).
- What will be your precision measure? Using `Robot.simulate()`, you can generate sequence of true positions and observations for certain maze. Using one of the considered estimation functions, you can get estimated positions (or distribution over positions). There are many ways how to define your measure of accuracy. For each time slice, you can use hit/miss, manhattan or Euclidean distance between the estimated and true positions, or you can compare the whole estimated distribution over states with the desired distribution (having only a single 1 at the true position, and 0s elsewhere).
- How many mazes will you use? How many simulations for each maze?

The above tasks are the only mandatory parts for this project. They will not bring you the full points, but they should be sufficient to fulfill partial requirements for the assessment.

4.2 Optional parts

All the following tasks are meant as an optional addition to the mandatory part. You can choose any number of them, solve them and describe them in your report for some additional points. You should assess the effects or contribution of the optional tasks, and the design of an experimental procedure for such an assessment is part of the task.

4.2.1 A modified robot model

The robot model as described in Sections 2 and 3.2 assumes that the states of the robot are given just by its position, i.e. it completely ignores robot orientation (because the robot, as described, can freely move to all directions without changing its orientation). The goal of this task is to change the robot model such that it would consider also the robot orientation, limit its actions e.g. to "move forward", "turn left", "turn right", and may also limit its sensors to left, front, and right, which may be a more appropriate model for a patrolling robot.

Task 3: Create a new HMM of the robot such that it considers also the robot orientation. Perform similar comparison as in the mandatory part, and report your results.

Hints:

- Describe the design of the robot that you consider. Define the possible states and observations. Implement methods `pt()` and `pe()` correctly!
- Create and submit module `robot2.py` with your implementation of the modified HMM.
- The original HMM in `robot.py` and your HMM in `robot2.py` are simply two different models, and their direct comparison is questionable. Nevertheless, you can comment on the estimation accuracy of individual methods, discuss reasons why the estimation for one HMM is more accurate on average than for the other HMM, etc.

4.2.2 Implementation of HMM using matrices

The HMM interface we use (from `hmm.py`) is quite general, yet it may lack efficiency. We use methods `HMM.pt()` and `HMM.pe()` to access individual transition and observation probabilities, respectively. All these probabilities may be collected in matrices. The transition matrix is then

$$A = [a_{ij}] = [P(X_t = x_j | X_{t-1} = x_i)]$$

and the observation matrix (emission model) is

$$B = [b_{ik}] = [P(E_t = e_k | X_t = x_i)].$$

When working on functions `forward()`, `forwardbackward()`, and `viterbi()`, you surely noticed that they actually implement matrix-vector multiplications, elementwise vector-vector multiplications, etc. This can be done much more efficiently using linear algebra library (like `numpy`), than using nested `for`-loops as done in our naive implementations.

Task 4: Enhance our HMM model and the inference algorithms to work with matrix representations of transition and emission models and to use matrix computations. Concentrate on replicating the same results coming from your naive solution. Describe your design in the report and evaluate the efficiency gains compared to the naive implementation.

Hints:

- Matrix-vector computations (with a slightly different notation) can be seen e.g. at [2, 4].
- The modification probably requires extending our `HMM` interface with methods like
  - `get_transition_matrix()` and
  - `get_observation_matrix()`

  which shall return `numpy.ndarray` objects. The ordering of states and observations shall respect the ordering of `get_states()` and `get_observations()`.
- The default implementation of these methods may just precompute the matrices by calling `pt()` and `pe()` methods, respectively, iteratively for each needed pair of states, or pair of state-observation. In a particular subclass, the matrices may be returned directly, if known.
- A modified `hmm_inference2.py` will then use these matrices to perform the computation efficiently.
- It would be nice if we could use your matrix-oriented `hmm_inference2.py` as a drop-in replacement for our naive `hmm_inference.py`, e.g. in the graphical user interface.

4.2.3 An interface to an existing HMM library

During the exercises, we actually created our own limited HMM library. Of course, there are several other more advanced HMM libraries, more efficient, and with more/different capabilities.

Task 5: Design and develop an adapter for an existing HMM library that would allow us to use the inference algorithms implemented therein with our HMM interface. Describe the design in the report and assess the contributions of using the third-party library from the point of view of capabilities, time requirements, etc.

Hints:

- Choose one of the following existing third-party HMM libraries for Python:
  - hmmlearn, a HMM library with API similar to scikit-learn. Source code: `https://github.com/hmmlearn/hmmlearn`, documentation `http://hmmlearn.readthedocs.io/en/latest/`
  - pomegranate. Source code: `https://github.com/jmschrei/pomegranate`, documentation `http://pomegranate.readthedocs.io/en/latest/`.
  - ghmm `http://ghmm.org/` and hmmus `https://pypi.python.org/pypi/hmmus/0.3.1` do not work in Python 3.
  - There are other options. If you find one suitable for this task, use it.
- The adapter will have to
  - transform our description of HMM to the form understood by the third-party library, and
  - transform the outputs provided by the third-party library inference algorithms to the format we expect.

4.2.4 Scaling to help with underflow issues

You may run into numerical issues when executing some of the naive implementations of the inference algorithms, especially for longer observation sequences. These are usually caused by the fact that during the updates, many probabilities (numbers lower than 1) are multiplied, which may cause underflow errors.

Task 6: Learn about the modification of the inference algorithms called *scaling*, implement it, and assess the effect of the modification.

Hints:

- Scaling is described e.g. on the Wikipedia page of forward-backward algorithm [4], in [3] or in the highly respected tutorial [2], Sec. V.A. Note that there is no standard notation used for HMMs, so be sure to check twice you understand the meaning of the symbols used in the used reference.
- Note that scaling in forward algorithm is actually equal to applying normalization, i.e. it is possible that you already do scaling in your `forward()` implementation. To implement it completely also in `forwardbackward()`, however, you will have to enhance the implementation a bit.
- The assessment of the contributions of scaling shall contain
  - a check that functions `forward()`, `forwardbackward()`, and `viterbi()` provide the same results without and with scaling,
  - whether and to what extent scaling helps to remedy the underflow issues.

4.2.5 Viterbi algorithm using log probabilities

If you have underflow issues with the Viterbi algorithm (for the same reason, multiplying a lot of probabilities), you can try to implement its version that uses log-probabilities.

Task 7: In `hmm_inference2.py`, implement a new version of function `viterbi()` that will use log-probabilities to prevent numerical issues. Describe the design in the report, and assess the contribution of the modification, i.e. show that it helped with your issues.

Hints:

- The modified algorithm is described e.g. in [2], Sec. V.A, Eqs. 104-105.
- You shall identify some sequences which cause numerical problems for the unmodified naive implementations. Then you shall show that Viterbi algorithm with log-probabilities has no issues with them.

4.2.6 Training from data using Baum-Welch

In all the above tasks, we assumed that the HMM is known to us. (For both Weather-Umbrella and Robot-in-maze domains, the models were prespecified.) Baum-Welch is an instance of EM algorithm for learning the HMM parameters (initial distribution, transition probabilities, emission probabilities) from sample sequences.

Task 8: In `hmm_inference`, implement function `baumwelch()` for training HMM from data. Design its interface, describe the design in the report, and assess its accuracy.

Hints:

- The Baum-Welch algorithm is described e.g. in [2], Sec. III.C, page 8, or in [1, 3]. Its version for working with multiple observation sequences is described in Sec. V.B, page 17 of [2]. Sections V.C and V.D are also relevant for the implementation of Baum-Welch.
- To assess the accuracy of the learning procedure, you shall start with an HMM with known parameters, and generate many observation sequences from it. Than you shall apply Baum-Welch on these observation sequences and you shall get the estimates of HMM parameters. Then you shall compare the estimates with the original parameters values used for generating the data. You shall also evaluate the likelihood of the original model, with the likelihood of the model updated by Baum-Welch, given the dtaa.

5 Requirements

You shall submit the Python scripts which demonstrate what you have achieved, and a report describing the methods and results, i.e. submit

1. module `hmm_inference.py` (mandatory), containing working naive implementations of forward, forward-backward, and Viterbi algorithms.
2. PDF file containing the report (mandatory), and
3. (optionally) other Python scripts, data, etc. used to create the report, especially the code that generates the results and figures used in the report, e.g. when comparing several approaches. The code shall be organized such that it is clear what parts of the code are related to what parts of the report. These files may not be evaluated, they serve mainly as a reference: if any part of the report would rise any doubts, these files will be consulted. If they are missing, the questionable part of the report may be evaluated with 0 points.

Because this project is a team work, you shall submit only one solution for the whole team. BRUTE allows for creation of teams, so find a partner!

Remember to submit your own work, i.e. work of your team! Do not commit plagiarism! The plagiarism detection feature will be on for this task, and any confirmed plagiat may be a reason to assign 0 points from this task! This warning applies to both the PDF reports and Python scripts.

5.1 Code

Your code shall be clean, readable, and understandable. This will form one criterion of the evaluation. Please, take the time to come up with meaningful names of variables, functions, constants, etc. Keep the functions short, spanning several lines only, if possible. This corresponds especially to the parts of your work which are a "free programming". Of course, in those parts where you have to comply to a specified interface there is much less for you to design.

5.2 Report

The report can be elaborated in Czech or in English. It shall have a form of a scientific article. You can assume the reader has a general background in Hidden Markov Models. The report shall contain an adequate description of the data, principles, methods used and results achieved in your work. By adequate description we mean a **concise description** sufficient to **understand and replicate** the work done by you. You should not only present the results, but also try to **interpret and discuss** them. You can use the provided templates in LaTeX or MS Word.

6 Scoring

The final score for the task will be composed of the components found in Tab. 1. The score for

Table 1: Scoring breakdown

Component (mandatory part)	**Points**
Correct naive implementation of `forward()`	0-3
Correct naive implementation of `forwardbackward()`	0-3
Correct naive implementation of `viterbi()`	0-3
Code quality (readability, understandability)	0-2
Report in LaTeX	0-1
Adequate description of the implementation	0-2
Adequate description of the comparison of estimation approaches	0-4
	Subtotal: 0-16
Component (optional parts)	**Points**
A modified robot model	0-6
Implementation of HMM using matrices	0-6
An interface to an existing HMM library	0-6
Scaling to help with underflow issues	0-6
Viterbi algorithm using log probabilities	0-6
Training from data using Baum-Welch	0-6
Total	Max. 20 regular points + max. 5 bonus points

optional parts is composed of evaluation of several aspects: readability of the implementation, adequacy of the description in the report, etc.

The **necessary** condition for the assessment is to get **at least 10 points from the mandatory part** of this project. If you solve more than 1 optional parts, their score will be added up to the maximal score of 25 points (points over 20 will be denoted as bonus points).

7 Have fun!

References

[1] Jeff Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, International Computer Science Institute, Berkeley CA, 1998. `http://melodi.ee.washington.edu/people/bilmes/mypapers/em.pdf`.

[2] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989. Can be found on web, e.g. at `http://www.cs.ubc.ca/~murphyk/Software/HMM/rabiner.pdf`.

[3] Mark Stamp. A revealing introduction to hidden markov models, 2015. `https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf`.

[4] Wikipedia. Forward-backward algorithm. `https://en.wikipedia.org/wiki/Forward-backward_algorithm`.