# “Robot” localisation with HMM based forward-filtering

This task relies on the explanations for matrix based forward filtering operations according to section 14.3.1 of the book (15.3.1 in the previous, 3rd, edition, 14.3.1 in the 2nd edition).

## The purpose of this assignment

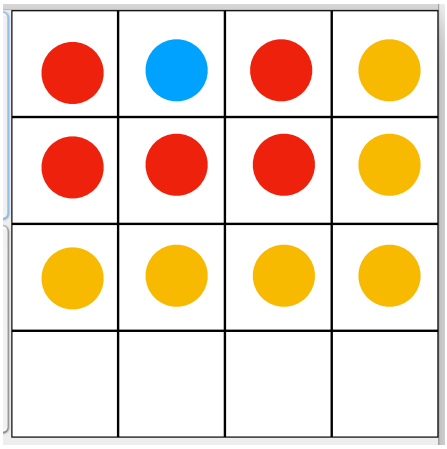
The idea with this assignment is to make you reflect about what you can do with a toy example implementation and how this is related to real world problems and applications. The idea is not to have you simply tick off a box when your implementation works, or to reach a specific number or other measure of quality. We expect you to read (both the instruction, but also the theory behind the task in the book), discuss, and think for yourself. There is no absolute recipe to follow!

## Task description

Please do not start out with the implementation, do the thinking and understanding first, it helps!

### *1) The scenario and overall problem to handle*

You are supposed to track an agent (robot) (with forward filtering based on an HMM) in an environment without any landmarks. Consider this agent to live *in an empty room*, represented by an *n x m* rectangular grid, that can move only along the rows and columns in the grid. The robot’s location is hidden; the only evidence available to you (the observer) is a noisy sensor that gives a direct, but not necessarily correct, approximation to the robot’s location. The sensor gives approximations $S = (x',y')$ for the (true) location $L = (x, y)$, the directly surrounding fields *Ls,* or the “second surrounding ring” *Ls2* according to the specifications below. Here, *n_Ls* is the number of directly surrounding fields for *L* (this can be 3, 5, or 8, depending on whether *L* is in a corner, along a “wall” or at least 2 fields away from any “wall”) and *n_Ls2* is the number of secondary surrounding fields for *L* (this can be 5, 6, 7, 9, 11, or 16, depending on where *L* is located relative to “walls” and corners).



The sensor reports

- the true location *L* (blue) with probability 0.1
- any of the $n\_Ls \in \{3, 5, 8\}$ existing surrounding fields *Ls* (red, here $n\_Ls = 5$) with probability 0.05 each
- any of the $n\_Ls2 \in \{5, 6, 7, 9, 11, 16\}$ existing “secondary” surrounding fields *Ls2* (yellow, here $n\_Ls2 = 6$) with probability 0.025 each
- "nothing" with probability 1.0 - 0.1 - *n_Ls**0.05 - *n_Ls2**0.025.

- This means that the sensor is more likely to fail entirely and produce “nothing” when the robot’s true location is less than two steps from a wall or in a corner.

The robot moves according to the following strategy:

Pick random start heading h_0 among the four possible headings. For any new step pick new heading h_t+1 based on the current heading h_t according to:

P( h_t+1 = h_t | not encountering a wall) = 0.7
P( h_t+1 != h_t | not encountering a wall) = 0.3
P( h_t+1 = h_t | encountering a wall) = 0.0
P( h_t+1 != h_t | encountering a wall) = 1.0

It then moves in the direction h_t+1 by one direct step in the grid. This means essentially that a) it will always move one step and b) it can only move straight.

In case a new heading is to be found, the new one is randomly chosen from the possible ones (facing a wall somewhere along the wall leaves three, facing the wall in a corner leaves two options for where to turn).

## *2) Understanding the given models and viewer tool (Your TODO #1!)*

Essentially, you are supposed to implement a localisation approach for the robot using the models described above. To do that, you are given a handout (.zip-file) with the following files that implement the above described specifications for the robot (transition and observation models) and the glue around them. Inspect these, consult the documentation ("Viewer Guide") and find your way through the handout, in particular work to understand the models in relation to the theory described in the book chapter and how you can make use of them later.
You are given:

- A Jupyter notebook that can be used to run and visualise the application (see "Viewer Guide" for some description of the GUI in the last cell). Note that the two cells with code are independent of each other, the first shows how to work with the models without the graphical interface, the second (last in the notebook) invokes the graphical interface.
- A viewer class (Dashboard) in which the visualisation is implemented (you do not need to change anything in there).
- *Localizer.py* with the stubb implementation of the respective class *Localizer*, which controls the localisation process. In this file, you need to make adaptations, see the comments in the file.
- Three files containing the models that implement the above described scenario in form of a state model, and the transition and observation models for the HMM:
    - *StateModel.py* contains the overall information of how states (poses) are encoded and provides a number of methods to transform the different representations.
    - *ObservationModel.py* holds the observation model and provides some methods to access the probabilities stored in the respective matrices.
    - *TransitionModel.py* holds the transition model in form of the respective matrix.

  Please consult the documentation INSIDE the model files for more details.
  One task in the writing part (report) is to explain the models, hence you should be able to show that you have understood what they are good for and what the information in them actually means!

## *3) Implementation (Your TODO #2!)*

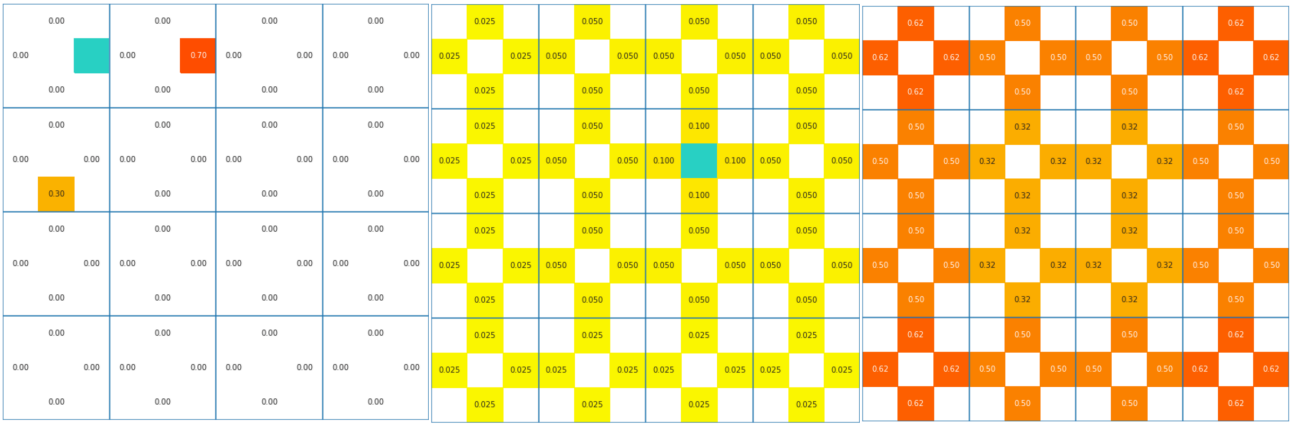
**Implement** a localisation / tracking approach based on an HMM and apply simple **forward filtering** to track the robot (according to the matrix-vector notation suggested in section 14.3.1 of the course book and the respective lecture slides, i.e. make use of the given models!).
This requires obviously a two-part implementation, as you first need to simulate the robot with its movement to have some ground truth to evaluate your tracking against, and simulate a sensor reading from this robot for the HMM-based tracking algorithm. **You should make use of the given models both for the simulation and the filtering!** Consider the general idea of randomised selection from a set or array based on a probability distribution for the simulation.

Your algorithm should then basically loop over the following three steps:

1. Move (simulated) robot to new pose according to transition model;
2. obtain (simulated) sensor reading based on its true, current, position given the observation model;
3. update the position estimate (based on the probability distribution given as vector f) with the forward-algorithm based on the sensor reading from step 2, using (again) the known sensor and transition models.

   Note that a sensor reading of "nothing" normally means to do the forward step without update, i.e. it boils down to mere prediction in theory. However, here, you should not go the normal way, but always do a complete update.
   **Thus, even a "nothing" reading from the sensor should entail a proper prediction + update step!**

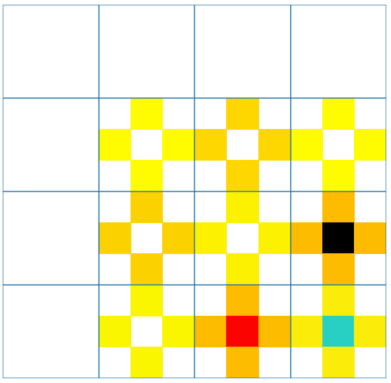
When starting to work with the handout, make sure you get the following results when visualising the transition model (left) and the sensor model (centre and right) for a 4x4-grid with the GUI.



Make sure that your robot moves correctly (no "jumps", no diagonal moves, no "sitting put") and that the sensor works within reasonable boundaries. Use the visualisation for that, see "Viewer guide" for details regarding how to do that.

## *4) Evaluation (Your TODO #3!)*

**Evaluate** your implementation. In terms of robot localisation it is often not relevant to know how often you are 100% correct with your estimate, but rather, how far "off" your estimate is from reality on average / how often. Measure the distance between true location and estimate by using the Manhattan distance (how many robot steps off) and report it as suggested in *Localizer.update()*.
**Compare** this result with what you would get without any filtering (pure guessing, or "filtering" based on the transition model and some random initial guess only), and with using the sensor reports only (base average Manhattan distance only on the steps when the sensor reports something, but count the "nothing" readings as a reference).
Use a grid size of preferably 6x8 (at least, however, 5x6) to base your evaluation on.
Decide whether you want to estimate based on the highest probability for one single state (pose), or based on the sum over the state probabilities for one grid cell (position).
If you use an 8x8 grid, you should observe something like 25% (probably more) of correct estimates rather quickly, roughly 100 steps should already get you there safely - note that this is only given as a *reference* for you. The average Manhattan distance should then be somewhere around 2.0 (probably below). If summed-up probabilities are sent to the viewer (i.e. all four state entries belonging to one position (grid cell) get the same value, which is the sum over the actual state probabilities in this cell), it is quite common to observe a "checker-board" pattern in the visualisation image with lower probabilities in every other position in the grid (in the image you see this also with the most likely position (red), the true position (black) and the sensed position (cyan)).



## *5) Peer review (Your TODO #4!) - OBS, initial submission required!*

Find a peer for a review and discussion when you have a working implementation (it does not need to be perfect, but it should not produce any blatant errors).
You should check that the robot simulation is correct, produces reasonable sensor readings (i.e. never further away than the "second ring" if not "nothing"), produces "nothing" reasonably often and that the overall results are within boundaries (see above). An indicator for something being really off is if the probabilities collapse after a while (gives a division-by-zero error).
You can also make comments on the efficiency of the implementation, e.g. if the models are not used to simulate movement and sensor readings, but these are computed according to the scenario / model description "from scratch", this should be pointed out as unnecessarily complex. Note though that this is not a coding competition and that there are peers with different backgrounds in the course, so the code does not have to follow the harshest rules or conventions!

Give (and receive) feedback in written form and coordinate for an actual discussion with your peer.

Submit three items in one text entry in Canvas according to the instructions for the first submission:
1) Relevant code snippets (methods you wrote code in) together with the
2) review you received and the
3) review you provided.

Improve your implementation if necessary. After the peer review is done, it should not be necessary for any TA to debug your code!

## *6) Reading article (Your TODO #5!)*

Read the article "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots" by Dieter Fox et al., which you can find (for example) at http://robots.stanford.edu/papers/fox.aaai99.pdf.
It was published in the proceedings of the 16th AAAI Conference on Artificial Intelligence (AAAI-99) in 1999, and received the AAAI Classic AI Paper Award in 2017. Examples relating to this paper will be part of the lecture on Probabilistic Robotics.

## *7) Writing report (Your TODO #6!) - OBS, final submission required!*

Write a brief (max 3 filled A4-pages excluding images, and excluding the appendix with the peer review you received) report that must follow the structure below and enclose all the listed points. Make use of the language check list below to avoid making the reader angry!

- ***Statement*** of who you did the ***peer review*** with, and (if so) who you discussed with during the implementation phase.
  If you could not do a peer review for some reason, inform Elin about that and write a respective statement instead. Expect your submission to be treated with low priority, which might result in your not being able to write the exam or get your credits in time for other courses, exceptions in cases with valid reasons are of course possible.
- ***Summary*** of the ***task*** in your own words (this should not be a description of your implementation, but it should explain for a random reader what problem you are supposed to solve and with which methods you tackled the task).
- ***Brief overview*** over your implementation, in particular if you did not follow the handout.
- ***Statement*** of what you ***changed*** in your ***implementation*** after the peer review of the implementation, either to fix problems or to simply improve it after reflecting more.
  Explain here what your peer commented upon, and how that had effect / impact on your thoughts about your own implementation.
- ***Explanation*** of the ***models*** given with the handouts, including an answer to this *question:* "***What does one see in the visualisation of the transition model, sensor model and for the running filtering process when using the GUI in the last cell of the notebook?"***. Discuss / explain especially the "no reading" visualisation of the observation model (also called sensor)! You can base this on the images above, or on your own experiments with the viewer.
- ***Discussion*** of your ***results***, answering the *core question:* ***"How accurately can you track the robot?"***. To do this, compare your results to pure guessing (no filtering or sensing at all applied) and to using only the sensor readings ("sensed"). Use error/success rates and average Manhattan distance to the true position as metrics.
- ***Summary*** of the ***article*** you read.
- ***Discussion*** of the ***relation*** between your ***implementation*** and the work described in the ***article***, answering the *core question:* ***"Is the HMM approach as implemented suitable for solving the problem of robot localisation?"***. Give clear evidence / motivation for your statement.
- ***Appendix*** with the comments you received from your peer for your initial implementation.

If you experience difficulties with understanding the task, getting the implementation to work, etc, please, CONTACT me (Elin, elin_anna.topp@cs.lth.se) or a TA well BEFORE the deadline!

Language check list:

- *Write entire sentences! Do this even though it seems now common practice in newspapers and other "official" texts to avoid this antique technique!*
  The second "sentence" of "The grass is wet. This because it rained until 15 minutes ago." or even "The grass is wet. Because it rained until 15 minutes ago." is not a sentence, as it is lacking a predicate (verb). Reading such constructs is tiring for the reader, who would be in search for an overlooked word or starts wondering whether the "." was accidentally placed instead of ",", which would have been correct in the second version, but would still not have helped in the first.

- *Check your grammar regarding the correct numerus of verb forms!*
  Rule of thumb: if the noun is in plural form (has an "s" in most cases[1]), the verb should NOT have an "s" or "es" and vice versa, as in "One bird flies, two birds fly". Respective mistakes should be even possible to catch with automated tools by now, at least the most blatant ones!

- *Decide on British or American English and stick to it!*
  British: colour, neighbour, visualise, e.g. ..., i.e. ...
  American: color, neighbor, visualize, e.g., ..., i.e., …

- *Keep track of possessive forms in comparison to plural forms and think about exceptions.*
  "The robot's arm was stuck because one of its motors broke."

- *Sort out "to" vs. "two" vs. "too", as well as "where" vs. "were"!*
  A typo here alters the meaning of the text (and yes, using the wrong form often is just a typo and just happens... but try to catch mix-ups), or makes it at least more difficult to understand.

- *Consider this to be a formal text!*
  Do not use short forms like "don't", "isn't", "we're", use the complete expressions "do not", "is not", "we are" / "we were". OBS, exception: the negation of *can* is *cannot*, it is not *can not*.

[1] One criterion, two criteria; one child, two children; one fish, two fish