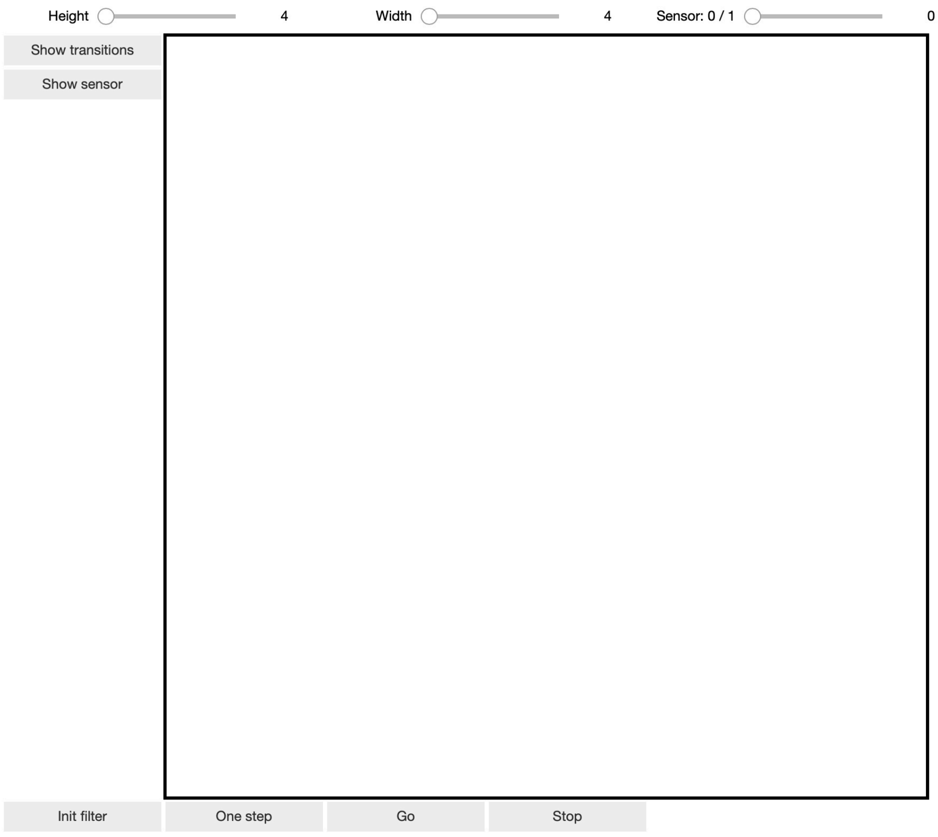
Using the RobotLocalisationViewer:

The RobotLocalisationViewer assumes a *state* encoding based on triplets (x,y,h), with x being the row, y the column (together the position) and h the heading of the robot in the grid world, where x=0 is the top row, y=0 is the leftmost column and h runs around the compass from 0 to 3 as SOUTH-EAST-NORTH-WEST. This combination of position and heading is also called *pose* and the setup follows “robotics conventions”. Try to view the grid as a “rotated” coordinate system, as “x” describes the axis going “forward” in a mobile robot’s world (“down” in this grid), while “y” is pointing “left” for a robot (“right” in the grid). Angles (headings) are then given in the mathematical sense, i.e. counter-clockwise.

1 Starting:

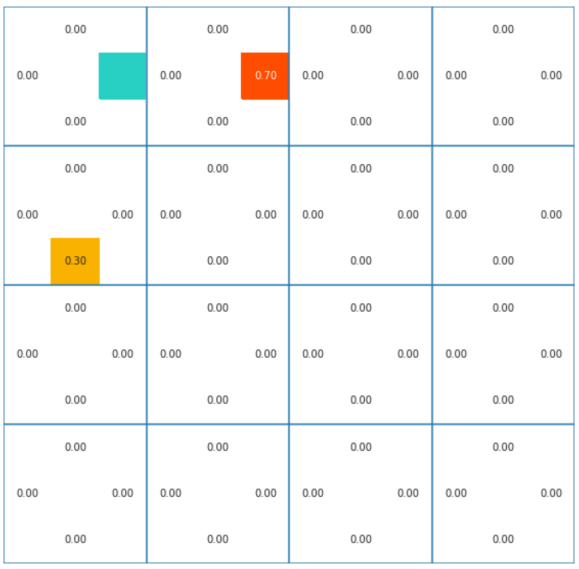

Height 4
Width 4
Sensor: 0 / 1 0

The viewer starts up with an empty field, two sliders for the dimensions of the grid, one “toggle” slider for the sensor type and six control buttons.

The figure shows the viewer with settings for a 4x4-field grid with non-uniformly failing sensor (type 0).

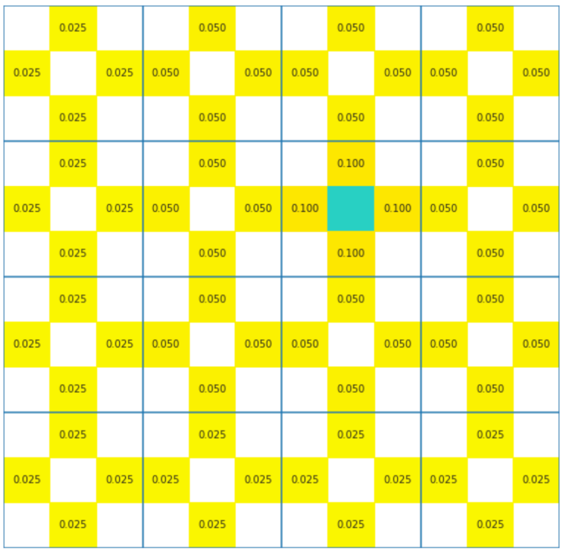
Please observe: The dimensions shown here and in the following figures (4x4) are ONLY an example, easy to fit in the document. Your implementation should consider BIGGER layouts and it is possible to have a rectangular grid, not only squares!

2 Checking the models (transition and observation)



2.1 Checking the transition model

Clicking on the “Show transitions”-button shows the probabilities for the different poses (x, y, h) to be reached after having been in the given state (marked in cyan). Each further click steps through the states and wraps in the end. The colour coding for the probabilities follows the idea of a heat map - the higher the value, the darker the marker.

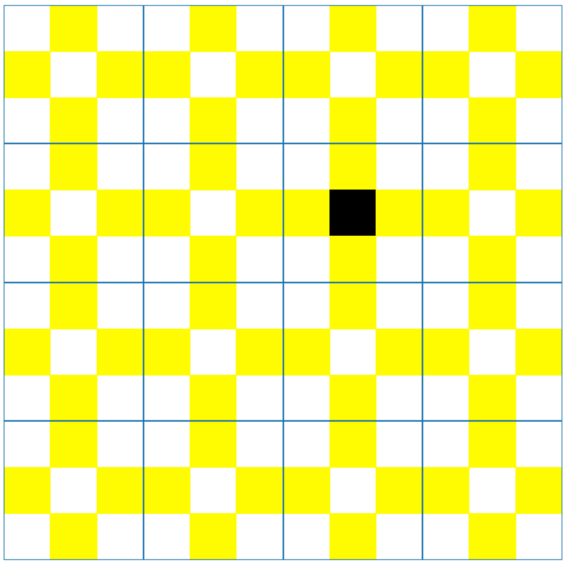


	0.025			0.050			0.050			0.050	
0.025		0.025	0.050		0.050	0.050		0.050	0.050		0.050
	0.025			0.050			0.050			0.050	
	0.025			0.050			0.100			0.050	
0.025		0.025	0.050		0.050	0.100		0.100	0.050		0.050
	0.025			0.050			0.100			0.050	
	0.025			0.050			0.050			0.050	
0.025		0.025	0.050		0.050	0.050		0.050	0.050		0.050
	0.025			0.050			0.050			0.050	
	0.025			0.025			0.025			0.025	
0.025		0.025	0.025		0.025	0.025		0.025	0.025		0.025
	0.025			0.025			0.025			0.025	

2.2 Checking the observation model
Clicking on the "Show sensor"-button shows the probabilities with which the sensor reports the given position (marked cyan) or "nothing" (all centers white), when the robot is actually in the other states (poses). Each further click steps through the sensor readings and wraps in the end.

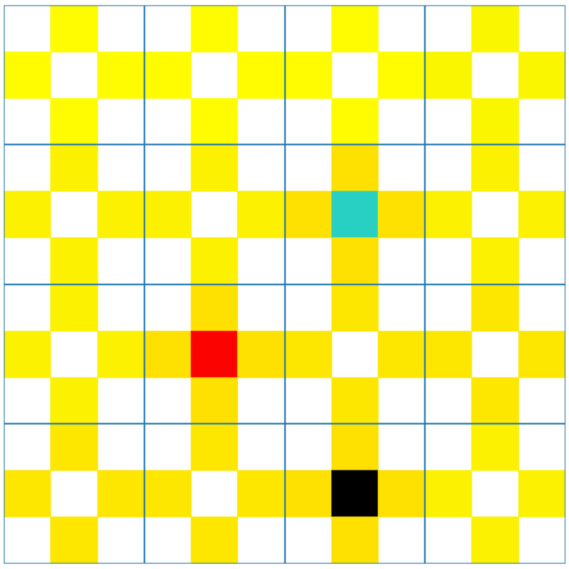
The viewer shows again all the numbers based on a heat map coding.

3 Visualising the filtering steps and results



3.1 Initialising
Clicking on "Init filter" initialises the viewer / localiser. This step is necessary to get further steps running properly (and it shows you the initial state of the grid). The figure shows the starting position (black) at (1, 0) and the probability distribution you start out with.
Please note: clicking "Init filter" again resets everything except the size of the grid and the sensor type, that is done with the sliders.

Note that when you just start the Viewer from the handout code, it does not actually work with a proper filter, as there is only a stub implementation in *Filters.py.* As also explained in the instructions: If you want your filtering results to be shown in the Viewer, you need to implement your algorithm in the stub method "*filter(...)*". The implementation will then default to use the NUF-model, if you want any changes there, you have to change that in *update()* in *Localizer.py.*



3.2 Stepping through
With "One step" you advance one step in the filtering process (i.e. cause a call to the *update()* method in the *Localizer* once). Black: true position, cyan: sensor reading, red: estimated (guessed) position. The heat map shows the probabilities for each position in all four states (poses) belonging to this position.

Please note: The results shown in the last figure are only **examples** for the visualisation - **do not assume them** as the final result for validation of your results!

3.3 Running continuously / stopping
With a click on "Go" you start a loop over the steps (delay according to time parameter for the driver thread in Dashboard). "Stop" interrupts the loop, it is possible to go stepwise again - or loop again. Not recommended for extensive evaluation, just for visual inspection!