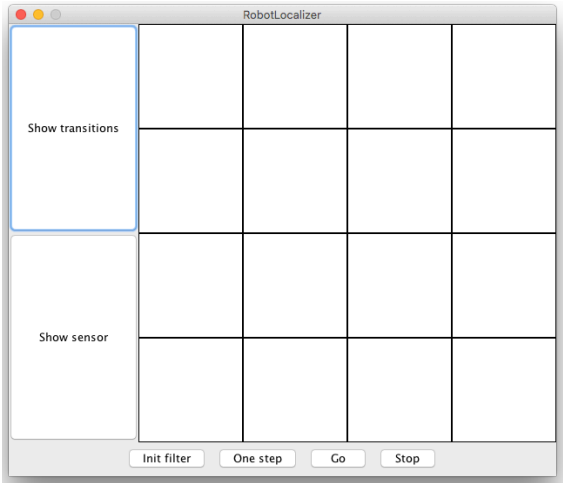
Using the RobotLocalisationViewer:

Note that this description is based on the Java-version of the handout; colours / visual experience in the Python version might differ slightly.

The RobotLocalisationViewer assumes a state coding 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, with x=0 being the top row, y=0 the leftmost column and h running around the compass from 0 to 3 as NORTH-EAST-SOUTH-WEST.

The provided implementation provides tools for conversion between numbered states and this representation as poses.
For the sensor readings it assumes to receive n*m probabilities, one for each position (x, y), to have caused the reading r = (rX, rY) or r = (-1, -1), which means "nothing". Make sure that you provide the data from the observation and transition matrices that you actually **use in your filtering calculations** in the respective methods, the handout does this for you, but if you change in the code, avoid changes in the wrong place.

1 Starting:

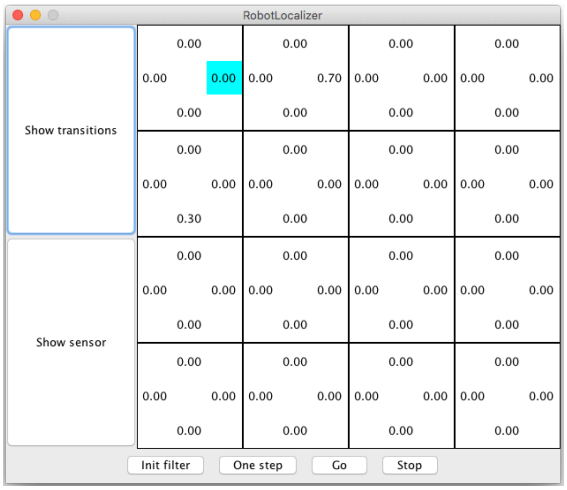


The viewer starts up with an empty grid according to the dimension specifications retrievable from the StateModel, see also Main.java.

The figure shows the viewer for a 4x4-field grid. Please observe: The dimensions shown here (4x4) are ONLY an example, easy to fit in the document. Your implementation should consider BIGGER layouts!

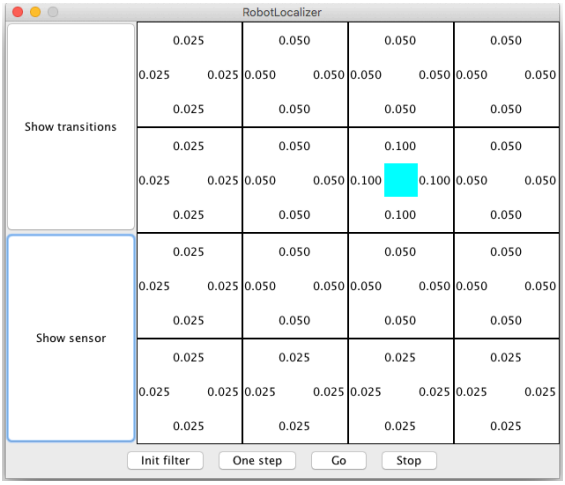
The Python viewer can actually be controlled with two sliders to change the grid size interactively, the Java version needs to be changed offline in Main().

2 Checking the matrices



2.1 Checking the transition matrix

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.



2.2 Checking the observation matrix

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

The Python viewer shows the numbers also as heat map, so "all white" only refers to the center cells in the visualisation.

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 state (black) at (1, 0, 1) and the probability distribution (normally equally distributed). A light grey "ring" marker is on the field with highest probability (i.e. on all fields, as everything is the same). Please note: clicking "Init filter" again does not do anything apart from skipping the sensor reading marker in one step - the viewer is currently meant to be started for a new run from scratch.

3.2 Stepping through

With "One step" you advance one step in the filtering process (hence the update() method for one step). Black: true position/state, cyan: sensor reading, light grey: highest probability (if several fields have the same highest probability, all of them are marked grey). White: assumed impossible (p = 0), yellow: low probability (0.0<p<=0.1), orange: higher probabilities (0.1< p <=0.3), red: "high" (p>0.3). In the upper part of each field the actual probability is shown (truncated to four decimals).

Please note: The numbers 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

With a click on "Go" you start the loop over the steps (delay according to time parameter for the driver thread). "Stop" interrupts the loop, it is possible to go stepwise again - or loop again.