1. [50] A robot is moving on a horizon grid of size 10 as shown in Fig. 1. The state of the robot at each time step k is given by its location with respect to the grid, i.e. $\xi_k \in \{1,2,\dots,10\}$. The robot can only move to the right by 1 unit or stay in the current cell at each time step through its control action $u_k \in \{1,0,-1\}$ where 1 for "move right" action, 0 for "stay" and -1 for "move left" action. The robot has a vision sensor to detect whether the wall, i.e. $z_k \in \{1,0\}$ where 1 for "facing wall" and 0 for "facing empty space". The system model and the sensor model are given by the following probabilities.

i) The initial position of the robot is located at 4, 5, or 6 with equal probability:

$$bel(\xi_0 = i) = \begin{cases} \frac{1}{3}, & i \in \{4,5,6\} \\ 0, & otherwise \end{cases}$$

ii) The measurement probability of the vision sensor with respect to the wall is given as:

- Correct positive: $p(z_k = \mathbf{1} | \xi_k \in \{2,6,7\}) = 0.9$,
- False negative: $p(z_k = \mathbf{0} | \xi_k \in \{2,6,7\}) = 0.1$,
- False positive: $p(z_k = \mathbf{1} | \xi_k \notin \{2,6,7\}) = 0.05$,
- Correct negative: $p(z_k = \mathbf{0} | \xi_k \notin \{2,6,7\}) = 0.95$

iii) The robot motion model:

- The "move right" command will cause it to move right at 90% rate (if $i \neq 10$), and stay otherwise; $p(\xi_{k+1} = i+1 | \xi_k = i, u_k = \mathbf{1}) = 0.9$, $p(\xi_{k+1} = i | \xi_k = i, u_k = \mathbf{1}) = 0.1$ (for $i < 10$)
- The "move left" command will cause it to move to left at 80% rate (if $i \neq 1$), and stay otherwise; $p(\xi_{k+1} = i-1 | \xi_k = i, u_k = -\mathbf{1}) = 0.8$, $p(\xi_{k+1} = i | \xi_k = i, u_k = -\mathbf{1}) = 0.2$ (for $i > 1$)
- If $\xi_k = 1$ or $\xi_k = 10$, then the robot will stay for any command to move it out of state range;

  $p(\xi_{k+1} = 1 | \xi_k = 1, u_k = -\mathbf{1}) = p(\xi_{k+1} = 10 | \xi_k = 10, u_k = \mathbf{1}) = 1$
- The "stay" command keeps the robot still, i.e. $p(\xi_{k+1} = i | \xi_k = i, u_k = \mathbf{0}) = 1$, $\forall i \in \{1,\dots,10\}$.

Assume that the robot control action sequence is given by $u_{0:6} = \{1,1,1,-1,0,-1,0\}$. Using Matlab, implement the Bayes filter for localization of the robot, i.e. compute $bel(\xi_k = i)$ with $k = 0,1,2,3,4,5,6$ for each location $i = 1,2,\dots,10$. Construct $bel(\xi_k)$ as a 8×10 matrix of the following form;

$$bel = \begin{bmatrix} bel(\xi_0 = 1) & bel(\xi_0 = 2) & \cdots & bel(\xi_0 = 10) \\ bel(\xi_1 = 1) & bel(\xi_1 = 2) & \cdots & bel(\xi_1 = 10) \\ \vdots & \vdots & \vdots & \vdots \\ bel(\xi_7 = 1) & bel(\xi_7 = 2) & \cdots & bel(\xi_7 = 10) \end{bmatrix}$$

Get the plot of each row of bel matrix. Based on the result, find the most-likely location of the robot at time step $k = 6$ and $k = 7$. The skeleton code ("HW4Q1skeleton.m") is provided and you can fill up the commented section.

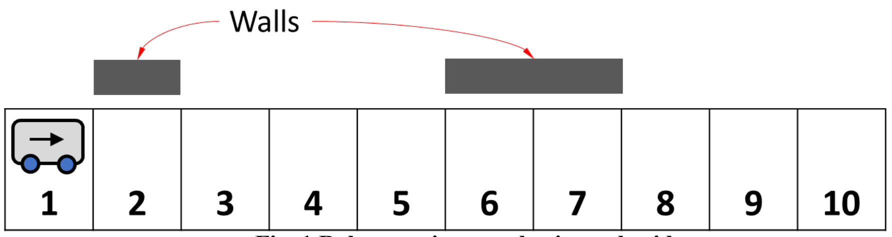

Fig. 1 Robot moving on a horizontal grid

2. [50] You are given the map of an environment as occupancy grid (Fig. 2), obtained from range sensors, the values of which are between [0,1], indicating the likelihood of having an obstacle in a cell of the grid. Dijkstra algorithm can be used to find a feasible path (collision-free) from a starting point to a goal point, by assuming that the cells of the occupancy grid are vertices connected to their neighboring cells. Complete the following functions in the given Matlab code:
   a) get_neighbors, which has arguments of the current cell (as coordinates [x,y]), and the occupancy grid map. This function should return all the neighbors of your current cell as a cell array of coordinates;
   b) get_edge_cost, which has arguments of the prior node and the current node (as coordinates [x,y]), and the occupancy grid map. This function should return the edge cost between the prior node and the current node. The costs should allow you to plan the path using Dijkstra's algorithm, by taking into account the occupancy information (i.e. give priority to the cells with lower probabilities);
   c) complete the update step in the while loop (which implements Dijkstra's algorithm), by checking the neighbors (using get_neighbors) of the prior node and calculating the costs (using get_edge_cost).

   The code might be quite slow, you can comment the 'pause' in the plotting functions.

   In your report, attach the code segments you've implemented, and the figures resulting from the code for the given start = [40,15] and goal = [21,37] points. [Optional] You can also try other starting and goal points and see how things change.

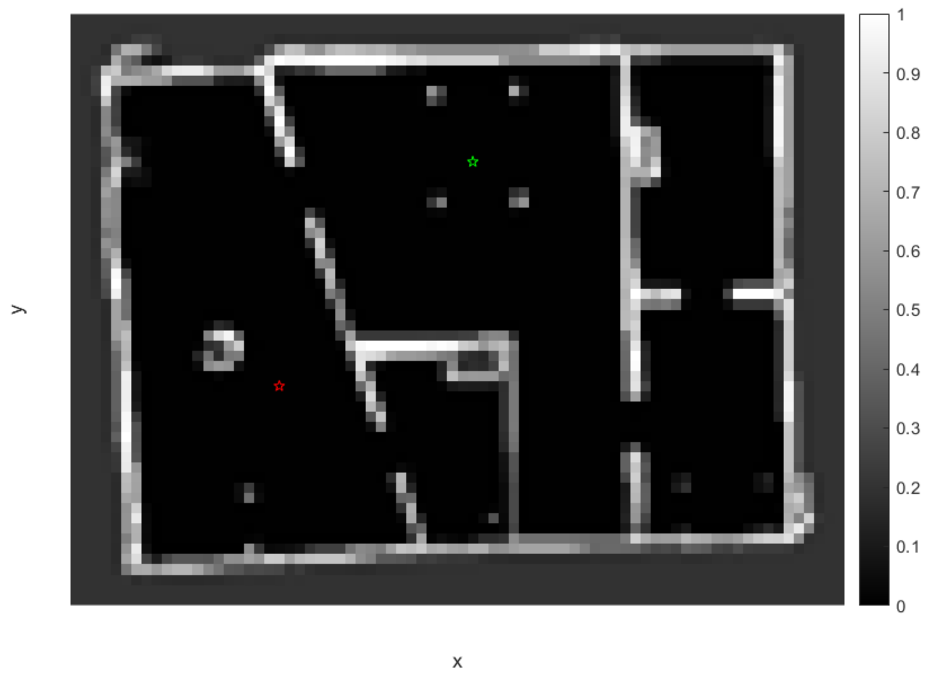

Fig. 2 Occupancy grid map with start and goal