To begin with the given problem, first I modeled all the probabilities related to the measurements as seen in the code snippet –

#### initializing all the measurement probabilities ####  
def measurement\_probability(measurement,state):  
 if (measurement == 'open') and (state == 'open'):  
 prob = 0.6  
 elif (measurement == 'closed') and (state == 'open'):  
 prob = 0.4  
 elif (measurement == 'open') and (state == 'closed'):  
 prob = 0.2  
 elif (measurement == 'closed') and (state == 'closed'):  
 prob = 0.8  
 return prob

Next the probabilities related to the actions taken and the current and next state of the door are modeled as follows –

#### initializing all the action probabilities ####  
def action\_probability(next\_state,action,state):  
 if (next\_state == 'open') and (action == 'push') and (state == 'open'):  
 prob = 1  
 elif (next\_state == 'closed') and (action == 'push') and (state == 'open'):  
 prob = 0  
 elif (next\_state == 'open') and (action == 'push') and (state == 'closed'):  
 prob = 0.8  
 elif (next\_state == 'closed') and (action == 'push') and (state == 'closed'):  
 prob = 0.2  
 elif (next\_state == 'open') and (action == 'do\_nothing') and (state == 'open'):  
 prob = 1  
 elif (next\_state == 'closed') and (action == 'do\_nothing') and (state == 'open'):  
 prob = 0  
 elif (next\_state == 'open') and (action == 'do\_nothing') and (state == 'closed'):  
 prob = 0  
 elif (next\_state == 'closed') and (action == 'do\_nothing') and (state == 'closed'):  
 prob = 1  
 return prob

Now, I wrote general function for the **Action Step –**

#### function definition for action step and calculation bel\_bar ####  
def action\_step(action, prev\_bel\_open, prev\_bel\_closed):  
 bel\_open = action\_probability('open',action,'open')\*(prev\_bel\_open) + action\_probability('open',action,'closed')\*(prev\_bel\_closed)  
 bel\_closed = action\_probability('closed', action, 'open') \* (prev\_bel\_open) + action\_probability('closed', action,'closed') \* (prev\_bel\_closed)  
 return bel\_open, bel\_closed

Next, I wrote a general function for the **Measurement Step** –

#### function definition for measurement step and calculation bel ####  
def measurement\_step(measurement, bel\_open\_bar, bel\_closed\_bar):  
 temp\_bel\_open = measurement\_probability(measurement,'open')\*(bel\_open\_bar)  
 temp\_bel\_closed = measurement\_probability(measurement, 'closed') \* (bel\_closed\_bar)  
 bel\_open = temp\_bel\_open/(temp\_bel\_closed+temp\_bel\_open)  
 bel\_closed = temp\_bel\_closed/(temp\_bel\_closed+temp\_bel\_open)  
 return bel\_open, bel\_closed

Now as per the given question, I made two arrays - One for storing the **actions** in the given order and another for storing the **measurements** received in the corresponding iterations.

actions = ['do\_nothing','do\_nothing','push','do\_nothing','push','do\_nothing']   
measurements = ['closed','closed','closed','closed','open','open']

As the robot does not have any prior information on the state of the door, therefore:

bel(open) = 0.5

bel(closed) = 0.5

The iterative implantation of Bayes Filter is as follows –

##### final Bayes Filter Implementation #####  
for i in range(len(actions)):  
  
 bel\_open\_bar, bel\_closed\_bar = action\_step(actions[i], bel\_open, bel\_closed)  
   
 bel\_open, bel\_closed = measurement\_step(measurements[i], bel\_open\_bar, bel\_closed\_bar)

Text

Description automatically generatedThe observed output of my implementation is as follows –

Text

Description automatically generated