# bayes_filter_Dheeraj

February 12, 2024

## 1 Task 1

The probabilities derived from the measurement model indicate higher reliability (80%) when the sensor measures a closed door, as opposed to a lower certainty (60%) when measuring an open door.

Mathematical representations of the propagation model are given below:

1. If $x_{t-1} = 1$ (door open) and $u_t = 1$ (Push): > $p(X_t = 1|U_t = 1, X_{t-1} = 1) = 1$ > $p(X_t = 0|U_t = 1, X_{t-1} = 1) = 0$
2. If $x_{t-1} = 0$ (door closed) and $u_t = 1$ (Push): > $p(X_t = 1|U_t = 1, X_{t-1} = 0) = 0.8$ > $p(X_t = 0|U_t = 1, X_{t-1} = 0) = 0.2$
3. If $x_{t-1} = 1$ (door open) and $u_t = 0$ (Do Nothing): > $p(X_t = 1|U_t = 0, X_{t-1} = 1) = 1$ > $p(X_t = 0|U_t = 0, X_{t-1} = 1) = 0$
4. If $x_{t-1} = 0$ (door closed) and $u_t = 0$ (Do Nothing): > $p(X_t = 1|U_t = 0, X_{t-1} = 0) = 0$ > $p(X_t = 0|U_t = 0, X_{t-1} = 0) = 1$

This model indicates that if the door is closed and the robot pushes, there's a 0.8 probability that it opens and a 0.2 probability that the door remains closed. If the robot takes no action, the door's state remains the same. Similarly if the robot pushes and the door is already open, the door stays open.

## 2 Task 2

Bayes Filter Implementation

[103]:
```
import numpy as np

def prediction(prior_belief,action):
    """
    Perform the prediction step of the Bayes Filter algorithm.

    Input:
    - prior_belief (numpy array): Prior belief about the state probabilities␣
  ↪[P(x_t=0), P(x_t=1)].
    - action (int): Action taken by the robot (0 for do nothing, 1 for push).

```

```
    Returns:
    - predicted_belief (numpy array): Predicted belief after the prediction step␣
↪[P(x_t=0), P(x_t=1)].
    """
    # define propagation model probabilities
    p_xt_ut_xt_1 = np.array([[[1,0]    , [0,1]],     # [ p(xt | u=0, xt_1=0) ,␣
↪p(xt | u=0, xt_1=1)
                            [[0.2,0.8], [0,1]]]);    #   p(xt | u=1, xt_1=0) ,␣
↪p(xt | u=1, xt_1=1) ]

    # predict belief based on action
    predicted_belief =  np.dot(p_xt_ut_xt_1[action].transpose(),prior_belief)

    return predicted_belief
```

[104]:
```
def update(predicted_belief,measurement):
    """
    Perform the measurement update step of the Bayes Filter algorithm.

    Input:
    - predicted_belief (numpy array): Predicted belief before the measurement␣
↪update.
    - measurement (int): Measurement received from the sensor (0 for closed, 1␣
↪for open).

    Returns:
    - updated_belief (numpy array): Updated belief after the measurement update.
    """
    # define sensor model probabilities
    p_zt_xt = np.array([[0.8,0.4],     #[ p(zt=0 | xt=0) , p(zt=0 | xt=1)
                        [0.2,0.6]])    #  p(zt=1 | xt=0) , p(zt=1 | xt=1) ]

    # update belief using the measurement
    updated_belief = p_zt_xt[measurement]*predicted_belief
    updated_belief /= np.sum(updated_belief)

    return updated_belief
```

[112]:
```
def bayes_filter(initial_belief,actions,measurements):
    """
    Bayes Filter algorithm to estimate the probability that the door is open.

    Parameters:
    - initial_belief (numpy array): Initial belief about the state probabilities␣
↪[P(x_t=0), P(x_t=1)].
```

```
    - actions (list): List of actions taken by the robot (0 for do nothing, 1 for␣
↪push).
    - measurements (list): List of measurements received from the sensor (0 for␣
↪closed, 1 for open).

    Returns:
    - final_belief (float): Final estimated probability that the door is open␣
↪(P(x_t=1)).
    """
    # Initialize the belief with the initial values
    current_belief = initial_belief.copy()

    # Iterate over the sequence of actions and measurements
    for action, measurement in zip(actions, measurements):
        # Prediction Step
        predicted_belief = prediction(current_belief, action)

        # Measurement Update Step
        current_belief = update(predicted_belief, measurement)
        # print(current_belief)

    # The final belief is the probability that the door is open (P(x_t=1))
    final_belief = current_belief[1]

    return final_belief
```

Q1. If the Action is always 0 and measurement is always 1, it takes 9 iterations for the belief that door is open to be greater than 99.99%

[113]:
```
# initilaize belief
initial_belief = np.array([0.5,0.5])
# belief that door is open
door_open = initial_belief[1]
# define action and measurment values
action = [0]
measurement = [1]
# counter for number of iterations
n = 0
# calculate belief and count iterations until belief that door is open >=0.9999
while(door_open<0.9999):
  door_open = bayes_filter(np.array([1-door_open,door_open]),action,measurement)
  n += 1

# display results
print("Belief that door is open: ",door_open)
print("Number of iterations: ",n)
```

```
Belief that door is open:  0.9999491973176183
Number of iterations:  9
```

Q2. If the Action is always 1 and measurement is always 1, it takes 4 iterations for the belief that door is open to be grater than 99.99%

[114]:
```python
# initilaize belief
initial_belief = np.array([0.5,0.5])
# belief that door is open
door_open = initial_belief[1]
# define action and measurment values
action = [1]
measurement = [1]
# counter for number of iterations
n = 0
# calculate belief and count iterations until belief >=0.9999
while(door_open<0.9999):
  door_open = bayes_filter(np.array([1-door_open,door_open]),action,measurement)
  n += 1

# display results
print("Belief that door is open: ",door_open)
print("Number of iterations: ",n)
```

```
Belief that door is open:  0.9999893637388586
Number of iterations:  4
```

Q3. If the action is always 1 and measurement is always 0, the belief that the door is open reaches max value of '1'. For 1000 action/measurment values, I printed the beliefs for each iteration. I observed that eventually the belief converged to a value of 1 after approx. 22 iterations.

The sensor has a 40% chance to sense a false negative , $p(z_t = 0|x_t = 1) = 0.4$. Since the action is always 1, the filter might consider these measurements to be noisy and the beleif converges to 1.

[115]:
```python
# initilaize belief
initial_belief = np.array([0.5,0.5])

# counter for number of iterations
n = 1000
# define action and measurment values
action = [1]*n
measurement = [0]*n

door_open = bayes_filter(initial_belief,action,measurement)

# display results
print("Belief that door is open: ",door_open)
print("Steady state belief: ",[1-door_open,door_open])
```

```
Belief that door is open:  1.0
```

```
Steady state belief:  [0.0, 1.0]
```