

ROS Problem Set 4 Report

Foundations of Robotics

Problem 1

Script Breakdown:

The `BayesFilter` class is defined in which some prerequisite variables are initialised and a function is defined which sets the action and measurement attributes for the class:

```
class BayesFilter:
    def __init__(self, initial_belief: Dict) -> None:
        self.initial_belief = initial_belief
        self.states = ["open", "closed"]
        self.action = None
        self.measurement = None

    def set_action_measurement(self, action: str, measurement: str) -> None:
        self.action = action
        self.measurement = measurement
```

The class is further extended by defining functions for the system model and the sensor model as shown in the following figure and the next:

```
# dynamic model/system model. Order: x(t), u(t), x(t-1)
@staticmethod
def get_system_prediction(curr:str, act:str, prev:str) -> float:
    if curr == "open" and act == "push" and prev == "open":
        return 1
    if curr == "closed" and act == "push" and prev == "open":
        return 0
    if curr == "open" and act == "push" and prev == "closed":
        return 0.8
    if curr == "closed" and act == "push" and prev == "closed":
        return 0.2
    if curr == "open" and act == "do_nothing" and prev == "open":
        return 1
    if curr == "closed" and act == "do_nothing" and prev == "open":
        return 0
    if curr == "open" and act == "do_nothing" and prev == "closed":
        return 0
    if curr == "closed" and act == "do_nothing" and prev == "closed":
        return 1
```

```
# sensor model
@staticmethod
def get_sensor_prediction(sensed_state: str, actual_state: str) -> float:
    if sensed_state == "open" and actual_state == "open":
        return 0.6
    if sensed_state == "closed" and actual_state == "open":
        return 0.4
    if sensed_state == "open" and actual_state == "closed":
        return 0.2
    if sensed_state == "closed" and actual_state == "closed":
        return 0.8
```

Function for the prediction step of the filter logic:

```
def get_bel_prediction(self, state: str) -> float:
    prediction = 0
    for prev_state in self.states:
        # sys_p(open)*init_bel(open) + sys_prob(closed)*init_bel(closed)
        sys_prediction = self.get_system_prediction(state, self.action, prev_state)
        initial_belief = self.initial_belief[prev_state]
        prediction += sys_prediction * initial_belief
    return prediction
```

The `filter` function encodes the bayes filter logic complete with the `prediction` and the `correction` steps and calculating the final probabilities. It suffices to call this one for any individual (action, measurement) pair:

```
def filter(self) -> Tuple[float, float]:
    bel_bar_open = self.get_bel_prediction("open")
    bel_open = self.get_sensor_prediction(self.measurement, "open") * bel_bar_open

    bel_bar_closed = self.get_bel_prediction("closed")
    bel_closed = self.get_sensor_prediction(self.measurement, "closed") * bel_bar_closed

    eta = 1 / (bel_open + bel_closed)

    return eta * bel_open, eta * bel_closed
```

An extension to its precursor, the following function takes care of iteratively calculating the state probabilities given a list of consecutive action, measurement pairs.

```

def filter_consecutive(self, inputs: List[Tuple[str, str]]) -> Tuple[float, float]:
    bel_open, bel_closed = 0.0, 0.0
    for (action, measurement) in inputs:
        bayes.set_action_measurement(action, measurement)
        bel_open, bel_closed = bayes.filter()

        # Update initial belief based on observation
        self.initial_belief = {"open": bel_open, "closed": bel_closed}

    return bel_open, bel_closed

```

Finally, everything culminates in the main function to obtain results for the cases as described in the Problem Set PDF.

(Individual):

```

if __name__ == "__main__":
    iterations = [
        # ("do_nothing", "closed"),
        # ("do_nothing", "closed"),
        ("push", "closed"),
        # ("do_nothing", "closed"),
        # ("push", "open"),
        # ("do_nothing", "open"),
    ]

    # Test individual (Comment out when testing consecutive inputs)
    bayes = BayesFilter(initial_belief={"open": 0.5, "closed": 0.5})
    for (act, measure) in iterations:
        bayes.set_action_measurement(act, measure)
        p_open, p_closed = bayes.filter()
        print("Probability that the door is open:", p_open)
        print("Probability that the door is closed", p_closed)

    # Test consecutively (Comment out when testing individual inputs)
    # p_open, p_closed = bayes.filter_consecutive(iterations)
    # print("Probability that the door is open:", p_open)
    # print("Probability that the door is closed", p_closed)

```

(Consecutive):

```
if __name__ == "__main__":
    iterations = [
        ("do_nothing", "closed"),
        ("do_nothing", "closed"),
        ("push", "closed"),
        ("do_nothing", "closed"),
        ("push", "open"),
        ("do_nothing", "open"),
    ]

    # Test individual (Comment out when testing consecutive inputs)
    bayes = BayesFilter(initial_belief={"open": 0.5, "closed": 0.5})
    # for (act, measure) in iterations:
    #     bayes.set_action_measurement(act, measure)
    #     p_open, p_closed = bayes.filter()
    #     print("Probability that the door is open:", p_open)
    #     print("Probability that the door is closed", p_closed)

    # Test consecutively (Comment out when testing individual inputs)
    p_open, p_closed = bayes.filter_consecutive(iterations)
    print("Probability that the door is open:", p_open)
    print("Probability that the door is closed", p_closed)
```

Results:

("do_nothing", "closed"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: do_nothing
Measurement: closed
Probability that the door is open: 0.3333333333333333
Probability that the door is closed 0.6666666666666666
```

("do_nothing", "closed"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: do_nothing
Measurement: closed
Probability that the door is open: 0.3333333333333333
Probability that the door is closed 0.6666666666666666
```


("push", "closed"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: push
Measurement: closed
Probability that the door is open: 0.81818181818182
Probability that the door is closed 0.181818181818182
```

("do_nothing", "closed"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: do_nothing
Measurement: closed
Probability that the door is open: 0.3333333333333333
Probability that the door is closed 0.6666666666666666
```

("do_nothing", "closed"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: push
Measurement: open
Probability that the door is open: 0.9642857142857143
Probability that the door is closed 0.03571428571428572
```

("do_nothing", "open"):

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: do_nothing
Measurement: open
Probability that the door is open: 0.75
Probability that the door is closed 0.25
```

Consecutive iterations:

```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8
radha@radha-hp-nb:~/WPI/RBE-500-Foundations/HW8$ python bayes_filter.py
Action: do_nothing
Measurement: closed
Action: do_nothing
Measurement: closed
Action: push
Measurement: closed
Action: do_nothing
Measurement: closed
Action: push
Measurement: open
Action: do_nothing
Measurement: open
Probability that the door is open: 0.9895901106050748
Probability that the door is closed 0.010409889394925181
```

Problem 2

Script Breakdown:

The motion is modelled as a random walk. System matrices are deduced:

```
# Random walk
#  $x_k = x_{k-1} + R$ 
#  $z_k = x_k + Q$ 

A = [1] # State matrix
B = [0] # Input matrix
C = [1] # Output matrix
I = [1] # Identity

R = [4] # system_noise
Q = [1] # sensor_noise

u = [0] # Control input
z1 = [22]
z2 = [23]
```

The following function encodes the kalman filter logic:

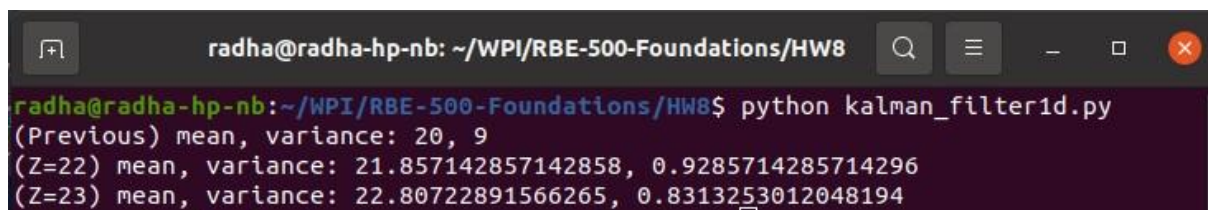
```
def kalman_filter(prev_mean, prev_var, control_action, sensor_measure):
    mew_bar = np.dot(A, prev_mean) + np.dot(B, control_action)
    sigma_bar = np.dot(np.dot(A, prev_var), np.transpose(A)) + R

    i1 = np.dot(C, sigma_bar)
    i2 = np.dot(i1, np.transpose(C))
    i3 = 1/(i2 + Q) # np.linalg.inv(a2 + Q)
    innovation = np.dot(np.dot(sigma_bar, np.transpose(C)), i3)

    mew = mew_bar + np.dot(innovation, (sensor_measure - np.dot(C, mew_bar)))
    sigma = (I - innovation * C) * sigma_bar

    return mew, sigma
```

Result:



```
radha@radha-hp-nb: ~/WPI/RBE-500-Foundations/HW8$ python kalman_filter1d.py
(Previous) mean, variance: 20, 9
(Z=22) mean, variance: 21.857142857142858, 0.9285714285714296
(Z=23) mean, variance: 22.80722891566265, 0.8313253012048194
```