TDT4137 - Exercise 3

Lars Moe Ellefsen

A) Hva slags aksjon gjør roboten med Mamdani resonering?

Mamdani resonnering har flere steg.

Steg 1: Fuzzification

Vi tar *Crisp Input'*ene **Distance** og **Delta**, og avgjør graden de hører til i sine respektive sett.

Distance:
VerySmall = 0
Small = **0,6**
Perfect = **0,1**
Big = 0
VeryBig = 0

Delta:
ShrinkingFast = 0
Shrinking = 0
Stable = **0,3**
Growing = **0,4**
GrowingFast = 0

Steg 2: Rule Evaluation

1. **IF distance is Small AND delta is Growing THEN action is None**
   None = min(Small(0,6) Growing(0,4)) = 0,4

2. **IF distance is Small AND delta is Stable THEN action is SlowDown**
   SlowDown = min(Small(0,6) Stable(0,3)) = 0,3

3. **IF distance is Perfect AND delta is Growing THEN action is SpeedUp**
   SpeedUp = min(Perfect(0,1) Growing(0,4)) = 0,1

4. **IF distance is VeryBig AND (delta is NOT Growing or delta is NOT GrowingFast) THEN action is FloorIt**
   FloorIt = min(0, max(NOT(0,4) NOT(0)) = 0

5. **IF distance is VerySmall THEN action is BrakeHard**
   Breakhard = VerySmall(0) = 0

Steg 3: Aggregation of the rule outputs

Kombiner outputten til et fuzzy sett..

Action = {BreakHard(0) , SlowDown(0,3), None(0,4), SpeedUp(0,1), FloorIt(0)}

Steg 4: Defuzzification

Bruker COG (center of gravity) til å regne ut *crisp output*en.

$$\text{COG} = \frac{0{,}3 * ((-6) + (-5) + (-4) + (-3) + (-2)) + 0{,}4 * ((-1) + 0 + 1) + 0{,}1 * (2+3+4+5+6)}{0{,}3 * 5 + 0{,}4 * 3 + 0{,}1 * 5}$$

$$\text{COG} = \frac{-4}{3.2} = -1{,}25$$

Den vil da velge None som action.

B)
Her er koden, som også vil ende opp med å velge None som action:

```
from sys import maxint

def triangle(position, x0, x1, x2, clip):
    value = 0.0
    if position >= x0 and position <= x1:
        value = (position - x0) / (x1 - x0)
    elif position >= x1 and position <= x2:
        value = (x2 - position) / (x1 - x0)
    if value > clip:
        value = clip
    return value


def grade(position, x0, x1, clip):
    value = 0.0
    if position >= x1:
        value = 1.0
    elif position <= x0:
        value = 0.0
    else:
        value = (position - x0) / (x1 - x0)
    if value > clip:
        value = clip
    return value

def reverse_grade(position, x0, x1, clip):
    value = 0.0
    if position <= x0:
        value = 1.0
    elif position >= x1:
        value = 0.0
    else:
        value = (x1 - position) / (x1 - x0)
    if value > clip:
        value = clip
    return value

def fuzzy_and(x, y):
    return min(x, y)
```

```
def fuzzy_and(x, y):
    return min(x, y)

def fuzzy_or(x, y):
    return max(x, y)

def fuzzy_not(x):
    return 1.0 - x

def generate_fuzzy_set(value, graph):
    fuzzy_set = [0.0 for _ in range(len(graph))]
    for index, set_ in enumerate(graph):
        if set_[0] == 'g':
            fuzzy_set[index] = grade(value, set_[1], set_[2], set_[3])
        elif set_[0] == 't':
            fuzzy_set[index] = triangle(value, set_[1], set_[2], set_[3], set_[4])
        elif set_[0] == 'r':
            fuzzy_set[index] = reverse_grade(value, set_[1], set_[2], set_[3])
    return fuzzy_set

def calculate_action_values(fuzzy_distance, fuzzy_delta):
    action_values = [
        fuzzy_distance[0],  # BrakeHard
        fuzzy_and(fuzzy_distance[1], fuzzy_delta[2]),  # SlowDown
        fuzzy_and(fuzzy_distance[1], fuzzy_delta[3]),  # None
        fuzzy_and(fuzzy_distance[2], fuzzy_delta[3]),  # SpeedUp
        fuzzy_and(fuzzy_distance[4],  # FloorIt
                  fuzzy_or(
                      fuzzy_not(fuzzy_distance[3]),
                      fuzzy_not(fuzzy_distance[4]))
                  )
    ]
    return action_values


def calculate_centroid(data):
    # Sample intervals
    intervals = [
        [-10, -9, -8, -7],
        [-6, -5, -4, -3],
        [-2, -1, 0, 1],
        [2, 3, 4, 5],
        [6, 7, 8, 9, 10]
    ]
    sum_ = 0.0
    fraction = 0.0
    for index, x in enumerate(data):
        for y in intervals[index]:
            sum_ += x * y
        fraction += x * len(intervals[index])
    return sum_ / fraction

def find_action_by_centroid(value):
    actions = ["BrakeHard", "SlowDown", "None", "SpeedUp", "FloorIt"]
    centers = [-8.0, -4.0, 0.0, 4.0, 8.0]
    lowest_distance = maxint
    action_index = 0
    for index, center in enumerate(centers):
        distance = abs(center - value)
        if distance < lowest_distance:
            lowest_distance = distance
            action_index = index
    return actions[action_index]
```

```
def main():
    distance_value = float(raw_input("Distance value: "))
    delta_value = float(raw_input("Delta value: "))
    print "-" * 19
    print "Calculating action"
    print "-" * 19

    distance_graph = [
        ["r", 0, 2.5, 1],  # VerySmall
        ["t", 1.5, 3, 4.5, 1],  # Small
        ["t", 3.5, 5, 6.5, 1],  # Perfect
        ["t", 5.5, 7, 8.5, 1],  # Big
        ["g", 7.5, 10, 1]  # Very Big
    ]

    delta_graph = [
        ["r", -5, -2.5, 1],  # ShrinkingFast
        ["t", -3.5, -2, -0.5, 1],  # Shrinking
        ["t", -1.5, 0, 1.5, 1],  # Stable
        ["t", 0.5, 2, 3.5, 1],  # Growing
        ["g", 2.5, 5, 1]  # GrowingFast
    ]

    fuzzy_distance = generate_fuzzy_set(distance_value, distance_graph)
    fuzzy_delta = generate_fuzzy_set(delta_value, delta_graph)
    action_values = calculate_action_values(fuzzy_distance, fuzzy_delta)
    centroid = calculate_centroid(action_values)
    action = find_action_by_centroid(centroid)
    print "Centroid %.2f" % centroid
    print "Action = %s" % action


Distance value: 3.7
Delta value: 1.2
-------------------
Calculating action
-------------------
Centroid -0.83
Action = None
```