TDT4137 – Exercise 4

Fuzzy Reasoning

- Lars Liverød Andersen

Task a)

Step 1 - Fuzzification

Crisp input x1:

μ(Distance = Small) = 0.6
μ(Distance = Perfect) = 0.1

Crisp input y1:

μ(Delta = Stable) = 0.3
μ(Delta = Growing) = 0.4

Step 2 – Rule evaluation

Rule 1: IF Distance is Small (0.6)
AND Delta is Growing (0.4)
THEN Action is None (0.4)

Rule 2: IF Distance is Small (0.6)
AND Delta is Stable (0.3)
THEN Action is SlowDown (0.3)

Rule 3: IF Distance is Perfect (0.1)
AND Delta is Growing (0.4)
THEN Action is SpeedUp (0.1)

Rule 4: IF Distance is VeryBig (0.0)
AND (Delta is NOT Growing (1 – 0.4 = 0.6) OR
(Delta is NOT GrowingFast (0.0))
THEN Action is FloorIt (0.0)

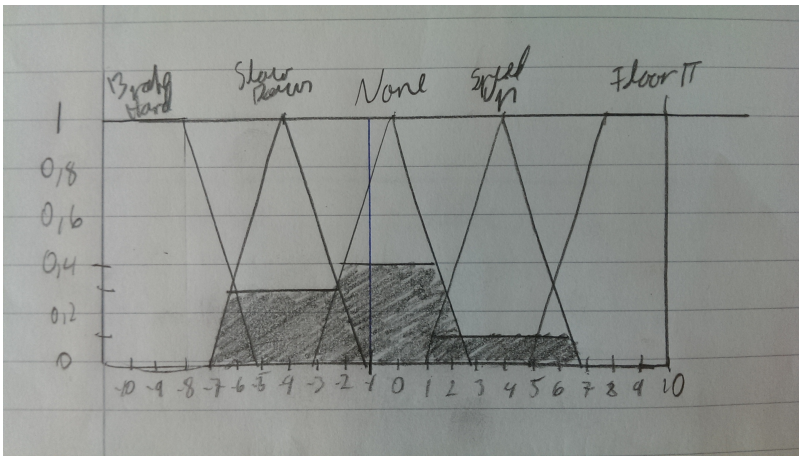
Rule 5: IF Distance is VerySmall (0.0)
THEN Action is BrakeHard (0.0)

Step 3 – Aggregation of the rule outputs

Action is None (0.4)
Action is SlowDown (0.3)
Action is SpeedUp (0.1)
Action is FloorIt (0.0)
Action is BrakeHard (0.0)

Step 4 - Defuzzification

Based on the fuzzy set created in step 3, I drew the following graph:



Using the values from this graph with the mathematical expression for finding Centre of Gravity, I got this equation:

COG =
*(-10-9-8-7-6-5) * 0.0 + (-7-6-5-4-3-2-1) * 0.3 + (-3-2-1+0+1+2+3) * 0.4 + (1+2+3+4+5+6+7) * 0.1 + (5+6+7+8+9+10) * 0.0 /*
*0.0 * 6 + 0.3 * 7 + 0.4 * 7 + 0.1 * 7 + 0.0 * 6 =*

-5.6/5.6 = -1

The centre of gravity is -1, which seems to be quite fitting.

Task b)

Running the same values as in a), I get very similar results in my Mamdani Reasoner.
Here are some sample outputs, where the first one is with similar values as in a).

Input: x=3.9, y=1.4: Action → None

```
COG: 1.09

CRISP_X: 3.9
CRISP_Y: 1.4

DISTANCE: {'verysmall': 0.0, 'small': 0.4000000000000001, 'verybig': 0.0, 'perfect': 0.2666666666666666, 'big': 0.0}

DELTA: {'shrinkingfast': 0.0, 'growing': 0.6, 'shrinking': 0.0, 'growingfast': 0.0, 'stable': 0.06666666666666672}

ACTIONS: {'none': 0.4000000000000001, 'slowdown': 0.06666666666666672, 'floorit': 0.0, 'speedup': 0.2666666666666666, 'brakehard': 0.0}
```

Input: x=4.8, y=2.6: Action → SpeedUp

```
COG: 4.00

CRISP_X: 4.8
CRISP_Y: 2.6

DISTANCE: {'verysmall': 0.0, 'small': 0.0, 'big': 0.0, 'perfect': 0.8666666666666666, 'verybig': 0.0}

DELTA: {'stable': 0.0, 'shrinkingfast': 0.0, 'growingfast': 0.06666666666666672, 'growing': 0.6, 'shrinking': 0.0}

ACTIONS: {'speedup': 0.6, 'floorit': 0.0, 'slowdown': 0.0, 'brakehard': 0.0, 'none': 0.0}
```

By looking at the COG, and the degree of membershit for each action, we can decide an action to do by placing the COG value on the x-axis of the action graph.

Following is the code for my Mamdani Reasoner, implemented in Python: (Step 3 is disregarded, as step 2 (rule-evaluation) does both steps at the same time in the program)

```
class Reasoner:
    def __init__(self, crisp_x, crisp_y):
        self.crisp_x = crisp_x
        self.crisp_y = crisp_y
        # Dictionary to keep track of the intersection value of the distance graphs and the crisp x
        self.distance = {'verysmall': 0, 'small': 0, 'perfect': 0, 'big': 0, 'verybig': 0}
        # Dictionary to keep track of the intersection value of the delta graphs and crisp y
        self.delta = {'shrinkingfast': 0, 'shrinking': 0, 'stable': 0, 'growing': 0, 'growingfast': 0}
        # Dictionary to keep track of the outputs of each action
        self.actions = {'none': 0, 'slowdown': 0, 'speedup': 0, 'floorit': 0, 'brakehard': 0}

    def __repr__(self):
        return '\n' + "CRISP_X: " + str(self.crisp_x) + \
               '\n' + "CRISP_Y: " + str(self.crisp_y) + \
               '\n'*2 + "DISTANCE: " + str(self.distance) + \
               '\n'*2 + "DELTA: " + str(self.delta) + '\n'*2 \
               + "ACTIONS: " + str(self.actions) + '\n'

    def fuzzification(self):
        self.distance['verysmall'] = self.reverse_grade(self.crisp_x, 0.0, 2.5, 1.0)
        self.distance['small'] = self.triangle(self.crisp_x, 1.5, 3.0, 4.5, 1.0)
        self.distance['perfect'] = self.triangle(self.crisp_x, 3.5, 5.0, 6.5, 1.0)
        self.distance['big'] = self.triangle(self.crisp_x, 5.5, 7.0, 8.5, 1.0)
        self.distance['verybig'] = self.grade(self.crisp_x, 7.5, 10.0, 1.0)
        self.delta['shrinkingfast'] = self.reverse_grade(self.crisp_y, -4.0, -2.5, 1.0)
        self.delta['shrinking'] = self.triangle(self.crisp_y, -3.5, -2, -0.5, 1.0)
        self.delta['stable'] = self.triangle(self.crisp_y, -1.5, 0, 1.5, 1.0)
        self.delta['growing'] = self.triangle(self.crisp_y, 0.5, 2, 3.5, 1.0)
        self.delta['growingfast'] = self.grade(self.crisp_y, 2.5, 4, 1.0)

    def rule_evaluation(self):
        self.actions['none'] = min(self.distance['small'], self.delta['growing'])       # Rule 1
        self.actions['slowdown'] = min(self.distance['small'], self.delta['stable'])    # Rule 2
        self.actions['speedup'] = min(self.distance['perfect'], self.delta['growing'])  # Rule 3
        self.actions['floorit'] = min(self.distance['verybig'],                         # Rule 4
                                      max(1 - self.delta['growing'], 1- self.delta['growingfast']))
        self.actions['brakehard'] = self.distance['verysmall']                          # Rule 5

    def defuzzification(self):
        above = (((-10 - 9 - 8 - 7 - 6 - 5) * self.actions['brakehard'])
                 + ((-7 - 6 - 5 - 4 - 3 - 2 - 1) * self.actions['slowdown'])
                 + ((-3 - 2 - 1 + 0 + 1 + 2 + 3) * self.actions['none'])
                 + ((1 + 2 + 3 + 4 + 5 + 6 + 7) * self.actions['speedup'])
                 + ((5 + 6 + 7 + 8 + 9 + 10) * self.actions['floorit']))
        below = (self.actions['brakehard'] * 6
                 + self.actions['slowdown'] * 7
                 + self.actions['none'] * 7
                 + self.actions['speedup'] * 7
                 + self.actions['floorit'] * 6)
        print("COG: " + '%.2f' % (above / below))

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

    def grade(self, 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(self, 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

if __name__ == '__main__':
    crisp_x_value = 2.6
    crisp_y_value = 0.8
    reasoner = Reasoner(crisp_x_value, crisp_y_value)
    # Step 1
    reasoner.fuzzification()
    # Step 2
    reasoner.rule_evaluation()
    # Step 4
    reasoner.defuzzification()
    # Printing rest of values in the reasoner
    print(reasoner)
```