به نام خدا

ایلیا نورانی ۹۹۳۱۰۹۲

گزارش پروژه فازی درس هوش محاسباتی

به طور کلی این پروژه دارای دو فاز است:

فاز اول که در آن تنها از دو سنسور راست و چپ برای جلوگیری از برخورد به مانع استفاده می‌شود و فاز دوم که در آن علاوه بر سنسورهای راست و چپ، از سنسور جلو نیز برای کنترل سرعت استفاده می‌شود.

فاز اول:

این فاز شامل سه مرحله است:

۱- Fuzzification

۲- Inference

۳- Defuzzification

در مرحله‌ی اول (Fuzzification)، مقادیر از حالت مطلق به حالت فازی تبدیل می‌شوند. برای انجام این مرحله، سه تابع d_L ، d_R و rotate را پیاده‌سازی می‌کنیم که به ترتیب برای تبدیل مقادیر مطلق فاصله از چپ، فاصله از راست و میزان چرخش فرمان به مقادیر فازی می‌باشند. نحوه عملکرد این توابع به این صورت است که ورودی با مقدار مطلق به تابع داده می‌شود و با استفاده از تابع تعلق مربوط به هر بخش، مقادیر تعلق محاسبه می‌شوند و به عنوان خروجی بازگردانده می‌شوند.

```
10      def d_L(self, x):
11          membership_func_close_L = 0
12          membership_func_moderate_L = 0
13          membership_func_far_L = 0
14
15          if x <= 0:
16              membership_func_close_L = 1
17          elif x > 0 and x <= 50:
18              membership_func_close_L = -0.02 * x + 1
19          # close_L -- done
20
21          if x >= 35 and x <= 50:
22              membership_func_moderate_L = 0.0666666 * x - 2.3333333
23          elif x > 50 and  x <= 65:
24              membership_func_moderate_L = -0.0666666 * x + 4.3333333
25          # moderate_L --- done
26
27          if x >= 50 and x <= 100:
28              membership_func_far_L = 0.02 * x - 1
29          elif x > 100:
30              membership_func_far_L = 1
31          # far_L --- done
32
33          return [membership_func_close_L, membership_func_moderate_L, membership_func_far_L]
34
```

```python
def d_R(self, x):
    membership_func_close_r = 0
    membership_func_moderate_r = 0
    membership_func_far_r = 0
    if x <= 0:
        membership_func_close_r = 1
    elif x >= 0 and x <= 50:
        membership_func_close_r = -0.02 * x + 1
    # close_L -- done

    if x >= 35 and x <= 50:
        membership_func_moderate_r = 0.0666 * x - 2.3333
    elif x >= 50 and  x <= 65:
        membership_func_moderate_r = -0.0666 * x + 4.3333
    # moderate_L --- done

    if x >= 50 and x <= 100:
        membership_func_far_r = 0.02 * x - 1
    elif x >= 100:
        membership_func_far_r = 1
    # far_L --- done

    return [membership_func_close_r, membership_func_moderate_r, membership_func_far_r]
```

```python
def rotate(self, x):
    membership_func_high_right = 0
    membership_func_low_right = 0
    membership_func_nothing = 0
    membership_func_low_left = 0
    membership_func_high_left = 0

    if x >= -50 and x <= -20:
        membership_func_high_right = 0.0333 * x + 1.6666
    elif x >= -20 and x < -5:
        membership_func_high_right = -0.0666 * x - 0.3333
    # high_right --- done

    if x >= -20 and x <= -10:
        membership_func_low_right = 0.1 * x + 2
    elif x >= -10 and x <= 0:
        membership_func_low_right = -0.1 * x + 0
    # low_right --- done

    if x >= -10 and x <= 0:
        membership_func_nothing = 0.1 * x + 1
    elif x >= 0 and x <= 10:
        membership_func_nothing = -0.1 * x + 1
    # nothing --- done

    if x >= 0 and x <= 10:
        membership_func_low_left = 0.1 * x + 0
    elif x >= 10 and x <= 20:
        membership_func_low_left = -0.1 * x + 2
    # low_left --- done

    if x >= 5 and x <= 20:
        membership_func_high_left = 0.0666 * x - 0.3333
    elif x >= 20 and x <= 50:
        membership_func_high_left = -0.0333 * x + 1.6666
    # high_left --- done

    return[membership_func_high_right, membership_func_low_right, membership_func_nothing,
           membership_func_low_left, membership_func_high_left]
```

در مرحله دوم (Inference)، مقادیر بدست آمده از توابع d_L و d_R یعنی میزان فاصله از چپ و راست به صورت فازی را به عنوان ورودی به تابع inference می‌دهیم. سپس این

تابع با استفاده از قوانین تعریف شده، میزان گردش فرمان (به صورت فازی) را به عنوان خروجی برمیگرداند.

```
101     def inference(self, left_dist, right_dist):
102         low_right = min(left_dist[0], right_dist[1])
103         high_right = min(left_dist[0], right_dist[2])
104         low_left = min(left_dist[1], right_dist[0])
105         high_left = min(left_dist[2], right_dist[0])
106         nothing = min(left_dist[1], right_dist[1])
107 
108         rotation = [high_right, low_right, nothing, low_left, high_left]
109 
```

در مرحله سوم (Defuzzification) نیز با استفاده از خروجی تابع inference و تابع rotate، مقادیر را از حالت فازی به حالت مطلق در می‌آوریم. برای تبدیل مقادیر فازی به مقادیر مطلق، از روش مرکز جرم استفاده شده است.

```
113     def defuzzify(self, rotation):
114         soorat = 0.0
115         makhraj = 0.0
116         X=np.linspace(-50, 50, 1000)
117         delta = X[1] - X[0]
118         for i in X:
119             temp = self.rotate(i)
120             m = max(min(temp[0], rotation[0]), min(temp[1], rotation[1]), min(temp[2], rotation[2]),
121                     min(temp[3], rotation[3]), min(temp[4], rotation[4]))
122             soorat = soorat + m * i * delta
123             makhraj = makhraj + m * delta
124         center = 0.0
125         if not makhraj == 0:
126             center = 1.0 * float(soorat) / float(makhraj)
127 
128         return center
129 
130 
```

در نهایت خروجی تابع defuzzify، میزان چرخش فرمان و همان جواب نهایی است.

```python
    def decide(self, left_dist, right_dist):
        """
        main method for doin all the phases and returning the final answer for rotation
        """
        # fuzzify

        d_l = self.d_L(left_dist)
        d_r = self.d_R(right_dist)

        # inference

        rot = self.inference(d_l,d_r)

        # defuzzify

        result = self.defuzzify(rot)

        return result
```

فاز دوم (امتیازی):

این فاز نیز مانند فاز قابل شامل سه مرحله‌ی Fuzzification، Inference و Defuzzification است. در مرحله اول مقادیر مطلق فاصله از جلو و سرعت به مقادیر فازی تبدیل می‌شوند.

```python
    # fuzzify

    def front_dist(self, x):
        membership_func_close_f = 0
        membership_func_moderate_f = 0
        membership_func_far_f = 0

        if x <= 0:
            membership_func_close_f = 0
        elif x > 0 and x <= 50:
            membership_func_close_f = -0.02 * x + 1
        # close -- done

        if x >= 40 and x <= 50:
            membership_func_moderate_f = 0.1 * x - 4
        elif x > 50 and  x <= 100:
            membership_func_moderate_f = -0.02 * x + 2
        # moderate --- done

        if x >= 90 and x <= 200:
            membership_func_far_f = 0.0090 * x - 0.8181
        elif x > 200:
            membership_func_far_f = 1
        # far --- done

        return [membership_func_close_f, membership_func_moderate_f, membership_func_far_f]

```

```python
    def gas(self, x):
        membership_func_low = 0
        membership_func_med = 0
        membership_func_high = 0

        if x <= 0:
            membership_func_low = 0
        elif x > 0 and x <= 5:
            membership_func_low = 0.2 * x + 0
        elif x > 5 and x <= 10:
            membership_func_low = -0.2 * x + 2
        # close -- done

        if x >= 0 and x < 15:
            membership_func_med = 0.0666 * x + 0
        elif x >= 15 and  x <= 30:
            membership_func_med = -0.0666 * x + 2
        # moderate --- done

        if x >= 25 and x <= 30:
            membership_func_high = 0.2 * x - 5
        elif x > 30 and x <= 90:
            membership_func_high = -0.0166 * x + 1.5
        elif x > 90: membership_func_high = 0
        # far --- done

        return [membership_func_low, membership_func_med, membership_func_high]

```

سپس در مرحله دوم مقادیر فازی بدست آمده از تابع front_dist به تابع inference داده می‌شوند و با استفاده از قوانین تعریف شده، خروجی این تابع که همان میزان تغییر سرعت (به صورت فازی است) بازگردانده می‌شود.

```python
    # inference

    def inference(self, forward_dist):
        gas_low = forward_dist[0]
        gas_mid = forward_dist[1]
        gas_high = forward_dist[2]

        return [gas_low, gas_mid, gas_high]

```

در نهایت در مرحله سوم خروجی از مقادیر فازی به مقادیر مطلق تبدیل می‌شوند.

```python
    # defuzzify

    def defuzzify(self, speed):
        soorat = 0.0
        makhraj = 0.0
        X=np.linspace(0, 200, 1000)
        delta = X[1] - X[0]
        for i in X:
            temp = self.gas(i)
            m = max(min(temp[0], speed[0]), min(temp[1], speed[1]), min(temp[2], speed[2]))
            soorat = soorat + m * i * delta
            makhraj = makhraj + m * delta
        center = 0.0
        if not makhraj == 0:
            center = 1.0 * float(soorat) / float(makhraj)

        return center

```

```python

    def decide(self, center_dist):
        """
        main method for doin all the phases and returning the final answer for gas
        """

        #fuzzify
        dist = self.front_dist(center_dist)

        #inference
        inf = self.inference(dist)

        res = self.defuzzify(inf)

        return res

```

پاسخ سوال امتیازی:

اگر در یک مسئله چندین قانون با مجموعه نهایی یکسان فعال شوند، به ناچار مقدار تعلق نهایی در یکی از مجموعه ها قرار می‌گیرد و در نهایت یکی از حرکت‌ها (مانند چرخش به چپ زیاد) انتخاب می‌شود. اما پس از آن برای محاسبه گام بعد دوباره این مقادیر محاسبه می‌شوند و ممکن است این بار مجموعه‌های نهایی متفاوت باشند. به طور کلی هر چقدر تعداد سنسورهای ما بیشتر و قوانین ما دقیق‌تر باشند، احتمال یکسان شدن مجموعه‌های نهایی کمتر می‌شود.