ثمین مهدی پور ۹۸۳۹۰۳۹

گزارش پروژه سیستم های فازی – اتومبیل خودران

فاز۱) فازی سازی:

```
def fuzzify_left_dist(self, left_dist):
    close_l = 0
    moderate_l = 0
    far_l = 0
    if 50 >= left_dist >= 0:
        close_l = (-(1 / 50) * left_dist) + 1
    if 50 >= left_dist >= 35:
        moderate_l = (1 / 18) * left_dist + ((-1) * 32 / 18)
    if 65 >= left_dist >= 50:
        moderate_l = ((-1) * (1 / 15) * left_dist) + (65 / 15)
    if 100 >= left_dist >= 50:
        far_l = (1 / 50) * left_dist + 1
    vector_membership = [close_l, moderate_l, far_l]
    return vector_membership
```

```
def fuzzify_right_dist(self, right_dist):
    close_r = 0
    moderate_r = 0
    far_r = 0
    if 50 >= right_dist >= 0:
        close_r = (-(1 / 50) * right_dist) + 1
    if 50 >= right_dist >= 35:
        moderate_r = (1 / 18) * right_dist + ((-1) * 32 / 18)
    if 65 >= right_dist >= 50:
        moderate_r = ((-1) * (1 / 15) * right_dist) + (65 / 15)
    if 100 >= right_dist >= 50:
        far_r = (1 / 50) * right_dist + 1
    vector_membership = [close_r, moderate_r, far_r]
    return vector_membership
```

با توجه به نمودار داده شده ورودی فاصله تا چپ و راست فازی سازی شده و به صورت یک بردار برگردانده میشود.

فاز۲) Interface

با توجه به بردارهای تولید شده در مرحله قبل برحسب قواعد خروجی فازی تولید میشود:

```
fuzzy_right_dist={}
fuzzy_right_dist['close']=fuzzy_right[0]
fuzzy_right_dist['moderate'] = fuzzy_right[1]
fuzzy_right_dist['far'] = fuzzy_right[2]

# Perform inference using fuzzy rules
fuzzy_output = []

# Rule 1: IF (d_L IS close_L) AND (d_R IS moderate_R) THEN Rotate IS low_right
fuzzy_output.append(("low_right", min(fuzzy_left_dist['close'], fuzzy_right_dist['moderate'])))

# Rule 2: IF (d_L IS close_L) AND (d_R IS far_R) THEN Rotate IS high_right
fuzzy_output.append(('high_right', min(fuzzy_left_dist['close'], fuzzy_right_dist['far'])))

# Rule 3: IF (d_L IS moderate_L) AND (d_R IS close_R) THEN Rotate IS low_left
fuzzy_output.append(('low_left',min(fuzzy_left_dist['moderate'], fuzzy_right_dist['close'])))

# Rule 4: IF (d_L IS far_L) AND (d_R IS close_R) THEN Rotate IS high_left
fuzzy_output.append(('high_left', min(fuzzy_left_dist['far'], fuzzy_right_dist['close'])))

# Rule 5: IF (d_L IS moderate_L) AND (d_R IS moderate_R) THEN Rotate IS nothing
fuzzy_output.append(('nothing', min(fuzzy_left_dist['moderate'], fuzzy_right_dist['moderate'])))
```

ثمین مهدی پور ۹۸۳۹۰۳۹

امتیازی:

در صورتی که در یک مسئله چندین قانون با مجموعه نهایی یکسان فعال شوند، روش‌های مختلفی برای محاسبه مقدار تعلق نهایی وجود دارد. برخی از این روش‌ها عبارتند از:

- Summation در این روش، مقادیر تعلق فعال شده در هر قانون با هم جمع می‌شوند و مقدار تعلق نهایی برابر با مجموع این مقادیر می‌شود.
- Maximum در این روش، مقدار تعلق نهایی برابر با بیشترین مقدار تعلق فعال شده در میان قوانین است.
- Product در این روش، مقادیر تعلق فعال شده در هر قانون با هم ضرب می‌شوند و مقدار تعلق نهایی برابر با حاصل ضرب این مقادیر می‌شود.

بهترین روش محاسبه مقدار تعلق نهایی بستگی به ماهیت و نیازهای خاص مسئله دارد. ممکن است در برخی موارد مقدار تعلق نهایی با جمع مقادیر تعلق، بیشینه‌گیری یا حاصل ضرب محاسبه شود. در هر صورت، روش انتخابی باید با منطق مسئله و تصمیم‌گیری‌ها سازگاری داشته باشد.

فاز۳) Defuzzification

برای پیاده سازی این بخش ابتدا باید اجتماع برخورد نمودارهای خروجی را حساب کنیم:

```
    # -50,-20
    rang_number.append((-50, -20, (a[0], b[0]),name[0]))
    # -20 , -10
    if self.get_y(fuzzy_output, name[1]) < self.get_y(fuzzy_output,name[2]):
        rang_number.append((-20, -10, (a[2], b[2]),name[2]))
    else:
        rang_number.append((-20, -10, (a[1], b[1]),name[1]))
    # -10, -5
    max_y = max(self.get_y(fuzzy_output, name[1]), self.get_y(fuzzy_output,name[3]),self.get_y(fuzzy_output, name[4]))
    for i in range(1, 5):
        if i == 2:
            continue
        if max_y == self.get_y(fuzzy_output, name[i]):
            rang_number.append((-10, -5, (a[i], b[i]),name[i]))
    # -5,0
    if self.get_y(fuzzy_output, name[3]) < self.get_y(fuzzy_output, name[4]):# -5,0
        rang_number.append((-5, 0, (a[4], b[4]),name[4]))
    else:
        rang_number.append((-5, 0, (a[3], b[3]),name[3]))
```

```
    # -5,0
    if self.get_y(fuzzy_output, name[3]) < self.get_y(fuzzy_output, name[4]):# -5,0
        rang_number.append((-5, 0, (a[4], b[4]),name[4]))
    else:
        rang_number.append((-5, 0, (a[3], b[3]),name[3]))
    # 0,5
    if self.get_y(fuzzy_output, name[5]) < self.get_y(fuzzy_output, name[6]):
        rang_number.append((0, 5, (a[6], b[6]),name[6]))
    else:
        rang_number.append((0, 5, (a[5], b[5]),name[5]))
    # 5 , 10
    max_y = max(self.get_y(fuzzy_output, name[5]), self.get_y(fuzzy_output, name[6]), self.get_y(fuzzy_output, name[8]))
    for i in range(5, 8):
        if i == 7:
            continue
        if max_y == self.get_y(fuzzy_output, name[i]):
            rang_number.append((5, 10, (a[i], b[i]), name[i]))
    # 10,20
    if self.get_y(fuzzy_output, name[7]) < self.get_y(fuzzy_output, name[8]): #10,20
        rang_number.append((10, 20, (a[8], b[8]),name[8]))
    else:
        rang_number.append((10, 20, (a[7], b[7]),name[7]))
    # 20, 50
    rang_number.append((20, 50, (a[9], b[9]),name[9]))
    return rang_number
```

سپس با توجه به شکل نهایی مرکز ثقل را محاسبه کنیم:

```
def CoG_Finder(self,line_equations, fuzzy_output):
    range_number = self.aggregator(fuzzy_output, line_equations)
    sigma_tot = 0
    sigma_m_tot = 0
    for min_num, max_num, (a, b), name in range_number:
        s, s_ = self.integrate(min_num , max_num , (name, (a,b)), fuzzy_output)
        sigma_tot += s
        sigma_m_tot += s_
    CoG = float(sigma_tot) / float(sigma_m_tot)
    print(f'fuzzy controller - CoG :  {CoG}')
    if CoG < 0:
        print("fuzzy controller - CoG is negative")
    return CoG
```

مرکز ثقل به عنوان جواب نهایی برگردانده خواهد شد.

ثمین مهدی پور ۹۸۳۹۰۳۹

بخش امتیازی:

- **فازی سازی:**

```
def fuzzify_center_dist(self, center_dist):
    close = 0
    moderate = 0
    far = 0
    if 50 >= center_dist >= 0:
        close = ((-1) * (1 / 50) * center_dist) + 1
    if 50 >= center_dist >= 40:
        moderate = ((1 / 10) * (center_dist)) - 4
    if 100 >= center_dist >= 50:
        moderate = (-1) * (1 / 50) * center_dist + 2
    if 200 >= center_dist >= 90:
        far = ((1 / 110) * center_dist) - (90 / 110)
    if center_dist >= 200:
        far = ((1 / 110) * center_dist) - (90 / 110)
    vector_membership = [close, moderate, far]
    return vector_membership
```

- اینترفیس:

```
def decide(self, center_dist):
    print('Deciding..')
    fuzzify_center = self.fuzzify_center_dist(center_dist)

    fuzzy_center_dist={}
    fuzzy_center_dist['close']=fuzzify_center[0]
    fuzzy_center_dist['moderate'] = fuzzify_center[1]
    fuzzy_center_dist['far'] = fuzzify_center[2]

    # Perform inference using fuzzy rules
    fuzzy_output = []

    # Rule 1: IF (center_dist IS close )  THEN  gas IS low
    fuzzy_output.append(("low", fuzzy_center_dist['close']))

    # Rule 2: IF (center_dist IS moderate )  THEN  gas IS medium
    fuzzy_output.append(('medium',fuzzy_center_dist['moderate']))

    # Rule 3: IF (center_dist IS far )  THEN  gas IS high
    fuzzy_output.append(('high',fuzzy_center_dist['far']))

    line_equations = self.get_lines()
    CoG = self.CoG_Finder(line_equations, fuzzy_output)
```

- دیفازی سازی:

```
    #0,5
    if self.get_y(fuzzy_output, name[0]) < self.get_y(fuzzy_output, name[2]):
        rang_number.append((0, 5, (a[2], b[2]), name[2]))
    else:
        rang_number.append((0, 5, (a[0], b[0]), name[0]))
    #5,10
    if self.get_y(fuzzy_output, name[1]) < self.get_y(fuzzy_output, name[2]):
        rang_number.append((5, 10, (a[2], b[2]), name[2]))
    else:
        rang_number.append((5, 10, (a[1], b[1]), name[1]))
    #10,15
    rang_number.append((10, 15, (a[2], b[2]), name[2]))
    #15,25
    rang_number.append((15, 25, (a[3], b[3]), name[3]))
    #25,30
    if self.get_y(fuzzy_output, name[3]) < self.get_y(fuzzy_output, name[4]):
        rang_number.append((25, 30, (a[4], b[4]), name[4]))
    else:
        rang_number.append((25, 30, (a[3], b[3]), name[3]))
    #30,90
    rang_number.append((30, 90, (a[5], b[5]), name[5]))
    return rang_number
```

```
def CoG_Finder(self, line_equations,fuzzy_output):
    range_number = self.aggregator(fuzzy_output, line_equations)
    sigma_tot = 0
    sigma_m_tot = 0
    for min_num, max_num, (a, b), name in range_number:
        s, s_ = self.integrate(min_num, max_num, (name, (a, b)), fuzzy_output)
        sigma_tot += s
        sigma_m_tot += s_
    if sigma_m_tot != 0:
        CoG = float(sigma_tot) / float(sigma_m_tot)
    else:
        return 0
    print(f'additional controller - CoG : {CoG}')
    if CoG < 0:
        print("additional controller - CoG is negative")
    return CoG
```

ثمین مهدی پور ۹۸۳۹۰۳۹

خروجی نهایی: