به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده‌ی مهندسی کامپیوتر

پروژه فازی

هوش محاسباتی بهار ۱۴۰۲

اتومبیل خودران فازی

اروند درویش

۹۸۳۱۱۳۷

این پروژه شامل سه مرحله اصلی است که در ادامه به توضیح و تفسیر هر کدام می پردازیم.

مرحله اول : Fuzzification

```
def fuzzification(self, left_dist, right_dist):
    close_L, moderate_L, far_L, close_R, moderate_R, far_R = 0,0,0,0,0,0

    if left_dist < 50 :
        close_L = -0.02*left_dist + 1

    if 35 < left_dist < 50 :
        moderate_L = (left_dist - 35)/15
    if 50 <= left_dist < 65 :
        moderate_L = (65 - left_dist)/15

    if 50 < left_dist <= 100 :
        far_L = 0.02*left_dist - 1



    if right_dist < 50 :
        close_R = -0.02*right_dist + 1

    if 35 < right_dist < 50 :
        moderate_R = (right_dist - 35)/15
    if 50 <= right_dist < 65 :
        moderate_R = (65 - right_dist)/15

    if 50 < right_dist <= 100 :
        far_R = 0.02*right_dist - 1

    return close_L, moderate_L, far_L, close_R, moderate_R, far_R
```

در این مرحله با استفاده از دو نمودار توابع تعلق فاصله از راست و فاصله از چپ ، ۶ مقادیر فازی را مقدار دهی می‌کنیم یعنی در حقیقت ما با ورودی گرفتن فاصله خودرو از چپ و راست به صورت crisp و دقیق آن را به مقادیر فازی تبدیل می کنیم با میزان تعلق یه این مقادیر.

بنابراین همان گونه که در کد مشاهده می کنید ابتدا ما نمودارهای توابع تعلق این دو متغیر را (یعنی فاصله از راست و فاصله از چپ) به صورت معادله خط مینویسیم و با ورودی ای که گرفتیم مقایسه می کنیم که ببینیم در کدام بازه قرار دارد تا معادله خط مربوط به آن را استفاده کنیم و مقدار میو یا همان تعلق را برای مقدار فازی مورد نظرمان را به دست آوریم.

مرحله دوم : Inference

set of rules :

```
1   IF (d_L IS close_L ) AND (d_R IS moderate_R)  THEN  Rotate IS low_right
2   IF (d_L IS close_L ) AND (d_R IS far_R)  THEN  Rotate IS high_right
3   IF (d_L IS moderate_L ) AND (d_R IS close_R)  THEN  Rotate IS low_left
4   IF (d_L IS far_L ) AND (d_R IS close_R)  THEN  Rotate IS high_left
5   IF (d_L IS moderate_L ) AND (d_R IS moderate_R)  THEN  Rotate IS nothing
```

code :

```python
def inference(self, left_dist, right_dist) :
    close_L, moderate_L, far_L, close_R, moderate_R, far_R = self.fuzzification(left_dist, right_dist)
    low_right = min(close_L, moderate_R)
    high_right = min(close_L, far_R)
    low_left = min(moderate_L, close_R)
    high_left = min(far_L, close_R)
    nothing = min(moderate_L, moderate_R)

    return high_right, low_right, nothing, low_left, high_left
```

در این بخش با استفاده از مجموعه قوانینی که به ما داده شده است سعی بر این داریم تا مقادیر جدید خروجی را به صورت فازی به دست آوریم به عنوان مثال قانون شماره یک بیان بر این می‌دارد که اگر فاصله از سمت چپ نزدیک باشد و فاصله سمت راست متوسط باشد در نتیجه خروجی ما که همان چرخش است باید کمی به سمت راست باشد همچنین برای مدل کردن این قوانین ما از روش ماکسیمم و مینیمم استفاده کردیم یعنی AND را به صورت مینیمم و OR را به صورت ماکسیمم تلقی کرده ایم.

امتیازی : اگر در یک مسئله چندین قانون با مجموعه ی نهایی یکسان فعال شوند برای محاسبه ی مقدار تعلق نهایی این مجموعه چه باید کرد؟

اگر در حالتی از مسئله چندین قانون با مجموعه نهایی یکسان فعال شوند برای محاسبه مقدار تعلق نهایی این مجموعه فازی میتوان مقدار تعلق به دست آمده از چند قانونی که فایر شده‌اند را با یکدیگر جمع کرد (در حالت جمع و ضرب جبری برای or و and) یا ماکسیمم (در حالت مینمم و ماکسیمم برای اشتراک و اجتماع) گرفت. به عنوان مثال در اینجا اگر دو قانون به ما مقدار تعلق برای مثلا مجموعه فازی high_right دهند، برای محاسبه مقدار تعلق نهایی این مجموعه مقادیر بدست امده را ماکسیمم می‌گیریم.

مرحله سوم : Defuzzification

max rotate function and defuzzifying inputs to give crisp outputs :

```
def defuzzification(self, left_dist, right_dist) :

    def max_rotate(x):
        high_right_new, low_right_new, nothing_new, low_left_new, high_left_new = 0,0,0,0,0
        high_right, low_right, nothing, low_left, high_left = self.inference(left_dist, right_dist)

        if x <= -20:
            high_right_new = min(high_right, (x + 50)/30)
        if -20 < x < -5 :
            high_right_new = min(high_right, (-5 -x)/15)

        if -20 < x <= -10:
            low_right_new = min(low_right, (x + 20)/10)
        if -10 < x < 0 :
            low_right_new = min(low_right, (-x)/10)

        if -10 < x <= 0 :
            nothing_new = min(nothing, (x + 10)/ 10)
        if 0 < x < 10:
            nothing_new = min(nothing, (-x + 10)/10)

        if 0 < x <= 10:
            low_left_new = min(low_left, x/10)
        if 10 < x < 20:
            low_left_new = min(low_left, (-x + 20)/10)

        if 5 < x <= 20 :
            high_left_new = min(high_left, (x - 5)/15)
        if 20 < x < 50:
            high_left_new = min(high_left, (-x + 50)/30)
```

center of gravity calculation :

```
soorat = 0.0
makhraj = 0.0
X = linspace(-50, 50, 1000)
for i in X:
    U = max_rotate(i)
    soorat += U * i
    makhraj += U
center = 0.0
if makhraj != 0:
    center = 1.0 * float(soorat)/float(makhraj)

return center
```

در این بخش منطق غیر فازی سازی را پیاده سازی می کنیم که در آن با استفاده از نمودار داده شده برای rotate یا همان خروجی مسئله ما و فرمول مرکز ثقل مقدار خروجی نهایی را به صورت دقیق (crisp) محاسبه میکنیم و بر می گردانیم.

فرمول محاسبه مرکز ثقل استفاده شده در اینجا به صورت گسسته است :

$$x^* = \frac{\sum_{i=1}^{n} \mu_{\bar{C}}(x_i) \,.\, x_i}{\sum_{i=1}^{n} \mu_{\bar{C}}(x_i)}$$

فاز دوم : امتیازی

در این بخش همانند فاز قبل تمام مراحل گفته شده را این بار برای متغیر فاصله از جلو تکرار می کنیم یعنی ابتدا مقدار فاصله از جلو را با استفاده از سنسور به صورت دقیق می گیریم سپس آن را با استفاده از نمودار داده شده در دستور کار به صورت فازی در می آوریم، در مرحله بعد با استفاده از قوانینی که داریم مقادیر فازی خروجی را به دست می آوریم و در نهایت این مقادیر به دست آمده را دوباره به صورت crisp برمی‌گردانیم و غیره فازی سازی را انجام می‌دهیم.

در نهایت کد ما به این شکل خواهد بود :

```
def fuzzification(self, center_dist):
    close, moderate, far = 0, 0, 0

    if 0 <= center_dist < 50:
        close = (50 - center_dist) / 50

    if 40 < center_dist <= 50:
        moderate = (center_dist - 40) / 10
    if 50 <= center_dist < 100:
        moderate = -(center_dist - 100) / 50

    if 90 < center_dist <= 200:
        far = (center_dist - 90) / 110
    if center_dist > 200:
        far = 1

    return close, moderate, far



def inference(self, center_dist):
    close, moderate, far = self.fuzzification(center_dist)
    low = close
    medium = moderate
    high = far
    return low, medium, high


def defuzzification(self, center_dist) :
    def max_speed(x) :
        low2, medium2, high2 = 0, 0, 0
        low, medium, high = self.inference(center_dist)
        if 0 < x <= 5:
            low2 = min(low, x / 5)
        if 5 <= x < 10:
            low2 = min(low, -(x - 10) / 5)

        if 0 < x <= 15:
            medium2 = min(medium, x / 15)
        if 15 <= x < 30:
            medium2 = min(medium, -(x - 30) / 15)

        if 25 < x <= 30:
            high2 = min(high, (x - 25) / 5)
        if 30 <= x < 90:
            high2 = min(high, -(x - 90) / 60)
```