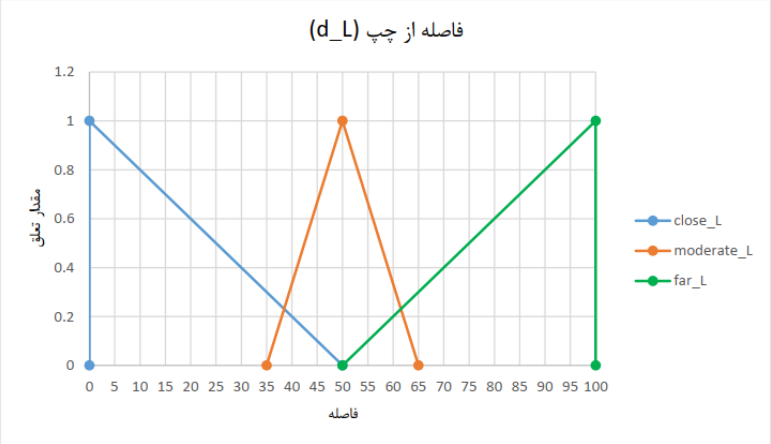
بخش اول) در فایل fuzzy_controller پیاده سازی شده است.

بخش اول از سه بخش تشکیل شده است. ابتدا فازی سازی سپس محاسبه خروجی فازی و در نهایت تبدیل خروجی فازی به مقداری مطلق است.

فازی سازی:

```
def mem_colse_L(self, x):...

def mem_far_L(self, x):...

def mem_moderate_L(self, x):...

def mem_colse_R(self, x):...

def mem_far_R(self, x):...

def mem_moderate_R(self, x):...
```

هر یک از توابع بالا میزان تعلق ورودی x به آن مجموعه را محاسبه میکند. برای مثال تابع mem_close_L مقدار تعلق x به دسته ی نزدیک به چپ برمیگرداند. پیاده سازی توابع از روی نمودار های داده شده صورت گرفته. برای مثال نمودار moderate_L را در نظر بگیرید.



تابع 3 بخش مجزا دارد. قبل از 35 و بعد از 65 میزان تعلق 0 است. بین 35 تا 50 میزان تعلق x/15 – 7/3 است و بین 50 تا 65 میزان تعلق x/15 – 13/3 است. طبق روابط بیان شده کد این تابع را به شکل زیر پیاده سازی میکنیم.

```
def mem_moderate_L(self, x):
    if 35 < x < 50:
        return x / 15 - 7 / 3
    if 65 > x >= 50:
        return 13 / 3 - x / 15
    return 0
```

سایر توابع تعلق نیز مشابه آنچه برای moderate_L توضیح داده شد پیاده میشوند.

برای میزان چرخش فرمان نیز توابع تعلق به دسته های nothing، low_right،high_right high_left،low_left را نیز پیاده سازی میکنیم.

```
def mem_high_right(self, x):...

def mem_high_left(self, x):...

def mem_low_right(self, x):...

def mem_low_left(self, x):...

def mem_nothing(self, x):...
```

هر یک از توابع بالا نشان میدهند یک x چقدر به یک دسته از میزان چرخش فرمان تعلق دارد.

محاسبه خروجی فازی:

برای محاسبه خروجی یکسری قوانین داریم.

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

AND در منطق فازی معادل minimum است. بنابراین برای مثال در قانون اول، مقدار low_right برابر با مینیمم بین میزان تعلق فاصله تا چپ به close_L و میزان تعلق فاصله تا راست به moderate_R است. یعنی اگر فاصله از چپ کم بود و فاصله از راست معمولی بود میزان چرخش فرمان در حد low_right است.

قوانین بالا را به شکل زیر پیاده سازی میکنیم.

```
low_right = min(self.mem_colse_L(left_dist), self.mem_moderate_R(right_dist))
high_right = min(self.mem_colse_L(left_dist), self.mem_far_R(right_dist))
low_left = min(self.mem_moderate_L(left_dist), self.mem_colse_R(right_dist))
high_left = min(self.mem_far_L(left_dist), self.mem_colse_R(right_dist))
nothing = min(self.mem_moderate_L(left_dist), self.mem_moderate_R(right_dist))
```

Left_dist و right_dist فاصله از گاردریل چپ و راست هستند که به عنوان ورودی به ما داده میشوند.

سوال امتیازی)

معمولاً از عملگرهای تجمیع (aggregation) استفاده می‌شود. عملگرهای تجمیع فازی، نقشی مشابه با عملگرهای تجمیع در منطق کلاسیک دارند و برای ترکیب نتایج فعالیت چندین قاعده در یک استنتاج مورد استفاده قرار می‌گیرند. یکی از روش‌های معمول برای تجمیع مقادیر تعلق در چندین قانون فعال، استفاده از عملگر ماکزیمم (maximum) است. به این صورت که مقدار تعلق نهایی برابر با بیشترین مقدار تعلق در میان قواعد فعال می‌شود. به عنوان مثال، اگر قاعده ۱ مقدار تعلق ۰٫۸ و قاعده ۲ مقدار تعلق ۰٫۶ داشته باشند، مقدار تعلق نهایی با استفاده از عملگر ماکزیمم برابر با ۰٫۸ خواهد بود.

تبدیل خروجی فازی به مقدار مطلق:

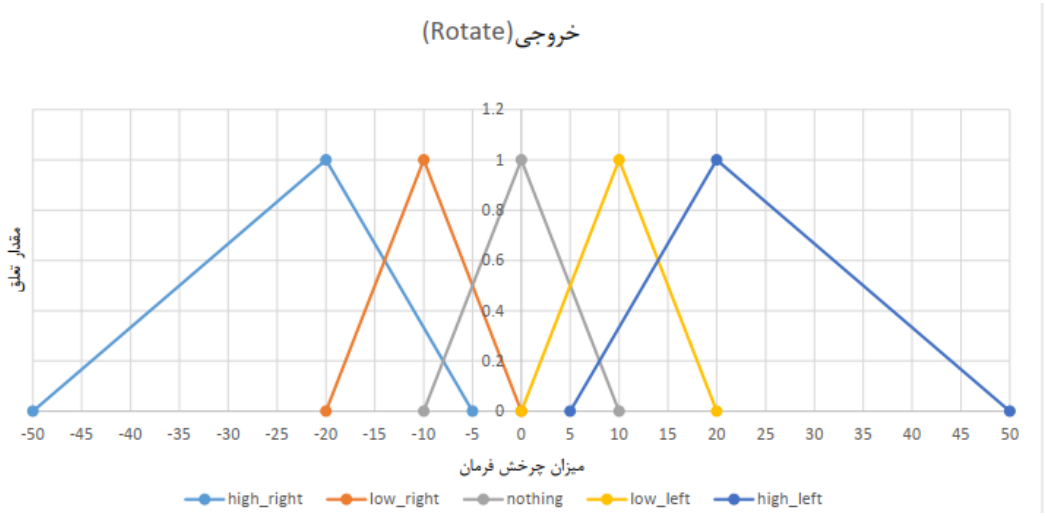
در حال حاضر خروجی ما مجموعه ای از مقادیر تعلق ها به دسته های low_right و low_left و ... است. برای استفاده از این خروجی نیاز داریم آنرا به عددی مطلق تبدیل کنیم که آن عدد در اینجا درجه چرخش فرمان است.

یکی از روش های غیرفازی سازی روش مرکز جرم است که طبق فرمول زیر حاصل میشود.

$$x^* = \frac{\int \mu_{\bar{C}}(x) \,.\, x \, dx}{\int \mu_{\bar{C}}(x) \, dx}$$

برای استفاده از روش مرکز جرم و انتگرال گیری تابع خروجی را ابتدا تعیین میکنیم.

توابع تعلق زیر را در نظر بگیرید.



در بخش قبل برای تعلق به هر دسته یک عدد به دست آوردیم. فرض کنید اعداد به شکل زیر باشند.

High_left=0.7

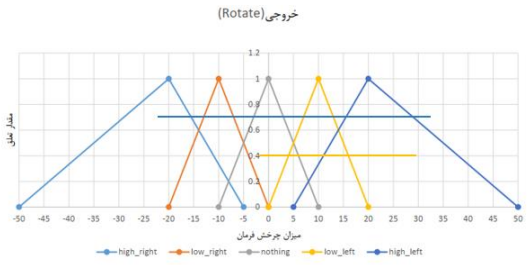
Low_left = 0.4

Nothing = 0

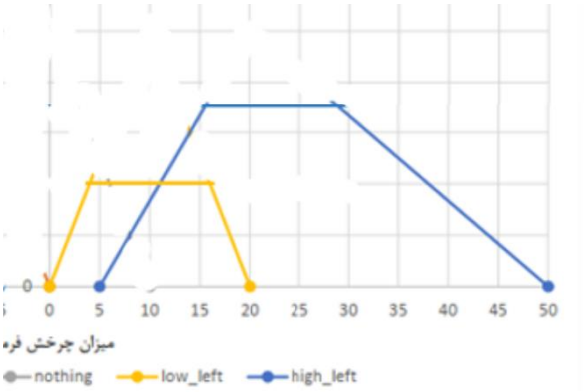
Low_right = 0

High_right = 0

خطوط موازی محور ایکس ها به اندازه های بالا میکشیم . هر خط متناظر نمودار خودش است و آنرا در دو نقطه قطع میکند.

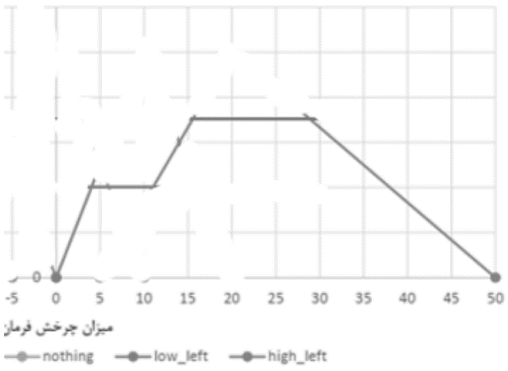


در نمودار هر تابع بخش بالایی این تقاطع حذف میشود (بین خط افقی و مقدار قبلی مینیمم باقی میماند) پس نمودار ها به شکل زیر میشوند.



دقت کنید نمودار سایر دسته ها 0 شده و دیگر نیستند.

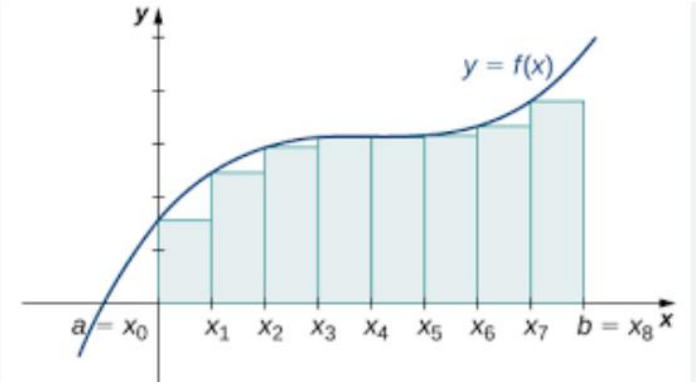
تابع خروجی ماکسیمم این توابع باقی مانده و به شکل زیر است.



مراحل بالا را در محاسبه تابع خروجی فازی به شکل زیر پیاده سازی میکنیم.

```
def max_function(x):
    return max(min(low_right, self.mem_low_right(x)),
               min(high_right, self.mem_high_right(x)),
               min(low_left, self.mem_low_left(x)),
               min(high_left, self.mem_high_left(x)),
               min(nothing, self.mem_nothing(x)))
```

در مرحله بعد نیاز به محاسبه دو انتگرال داریم. از انتگرال گیری تقریبی استفاده میکنیم که تابع را به شکل مجموعه ای از ستون ها در نظر میگیرد و مساحت ستون ها را با یکدیگر جمع میکند. ستون ها عرض برابر 0.1 دارند. میتوانیم برای افزایش دقت این عرض را کمتر کنیم اما سرعت محاسبات کاهش میابد.



شکل بالا نحوه محاسبه را نشان میدهد. از ابتدا تا انتها بازه بر حسب عرض مستطیل ها تعدادی نقطه به دست می آوریم. F(x) این نقاط همان ارتفاع مستطیل هاست. مساحت ها را حساب کرده و باهم جمع میکنیم.

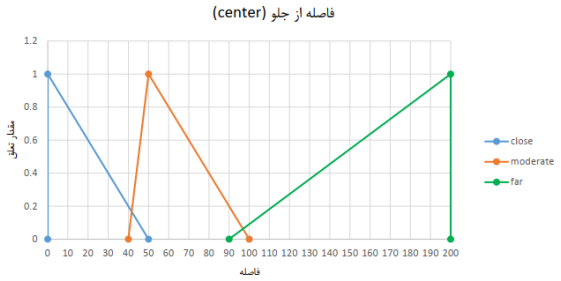
```
x = -50
b = +50
makhraj = 0.0
soorat = 0.0
while x < b:
    x = x + 0.1
    makhraj = makhraj + max_function(x) * 0.1
    soorat = soorat + max_function(x) * 0.1 * x
if makhraj != 0:
    return float(soorat) / float(makhraj)
return 0
```

X شروع بازه انتگرال گیری و b انتهای بازه است. هر بار با در نظر گرفتن اینکه max_function(x) ارتفاع یک مستطیل است، ارتفاع را در 0.1 که عرض است ضرب میکنیم و با مساحت های قبلی (makhraj) جمع میکنیم. به طور موازی برای تابع max_function(x).x محاسبه میکنیم. محاسبات را انقدر ادامه میدهیم تا x که هر دور به آن 0.1 افزوده شده به b برسد. در نهایت طبق فرمول مرکز جرم این دو انتگرال را بر هم تقسیم میکنیم تا عدد درجه چرخش فرمان به دست آید.

بخش امتیازی)

این بخش نیز مشابه بخش قبلی انجام میشود.

ابتدا توابع تعلق تعریف میشوند.




```
def mem_close(self, x):...

def mem_moderate(self, x):...

def mem_far(self, x):...

def mem_low_speed(self, x):...

def mem_medium_speed(self, x):...

def mem_high_speed(self, x):...
```

فقط در نمودار تابعبرای far ، فاصله بیشتر از 200، 0 لحاظ شده بود که من 1 در نظر گرفتم وگرنه در فواصل بیشتر از 200 تا دیوار جلویی، اصلا ماشین حرکت نمیکند. بعد از تعریف توابع ، قوانین موجود را تعریف میکنیم.

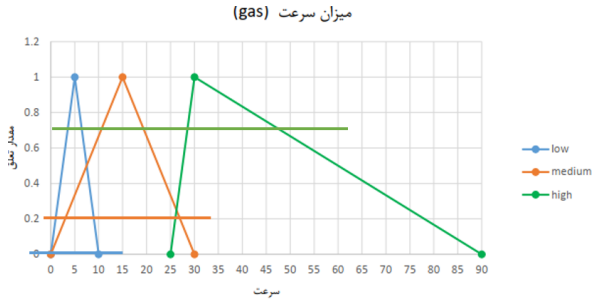
```
low = self.mem_close(center_dist)
medium = self.mem_moderate(center_dist)
high = self.mem_far(center_dist)
```

یعنی اگر فاصله مان نزدیک باشد سرعت باید کم باشد، اگر فاصله مان متوسط باشد سرعت باید متوسط باشد و اگر فاصله مان دور بود سرعت میتواند زیاد باشد و میزان کم ، متوسط یا زیاد بودن سرعتمان متناسب با فاصله مان است.

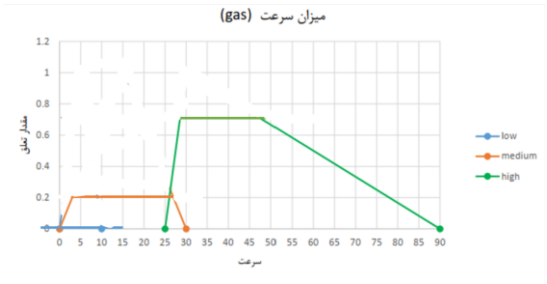
حال ما نیاز داریم مقادیر فازی low و medium و high را به مقدار مطلق تبدیل کنیم. این تبدیل را با روش مرکز جرم انجام میدهیم که نیاز به دو انتگرال گیری دارد. این انتگرال ها روی تابع ماکس صورت میگیرند.

نحوه ساخت تابع به شکل زیر است.

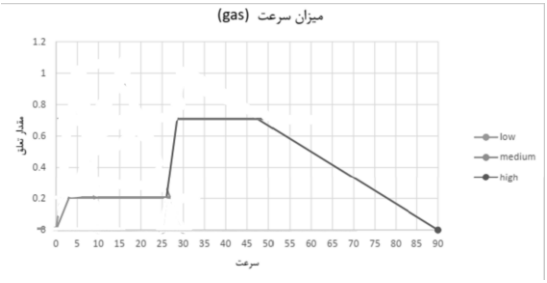
فرض کنید low=0 medium=0.2 high=0.7 باشد. خطوط موازی محور ایکس با اندازه هایlow و high و medium میکشیم.



هر خط متناظر نمودار خودش است و آنرا در دو نقطه قطع میکند. از نقطه تقاطع به بالای نمودار را حذف میکنیم در واقع بین خط متناظر با نمودار و خود نمودار مینیمم میگیریم.



حال ماکسیمم بین توابع باقی مانده را میگیریم.



مراحل بالا را در محاسبه تابع خروجی فازی به شکل زیر پیاده سازی میکنیم.

```
def max_function(x):
    return max(min(low, self.mem_low_speed(x)),
               min(high, self.mem_high_speed(x)),
               min(medium, self.mem_medium_speed(x)))
```

به روش عددی فاز 1 انتگرال گیری را انجام میدهیم.

```
x = 0
upper_bound = +90
makhraj = 0.0
soorat = 0.0
while x < upper_bound:
    x = x + 0.1
    makhraj = makhraj + max_function(x) * 0.1
    soorat = soorat + max_function(x) * 0.1 * x
if makhraj != 0:
    return float(soorat) / float(makhraj)
return 0
```

x شروع بازه انتگرال گیری و upper_bound انتهای بازه است. هر بار با در نظر گرفتن اینکه max_function(x) ارتفاع یک مستطیل است، ارتفاع را در 0.1 که عرض است ضرب میکنیم و با مساحت های قبلی (makhraj) جمع میکنیم. به طور موازی برای تابع max_function(x).x محاسبه میکنیم. محاسبات را انقدر ادامه میدهیم تا x که هر دور به آن 0.1 افزوده شده به upper_bound برسد. در نهایت طبق فرمول مرکز جرم این دو انتگرال را بر هم تقسیم میکنیم تا عدد سرعت به دست آید.