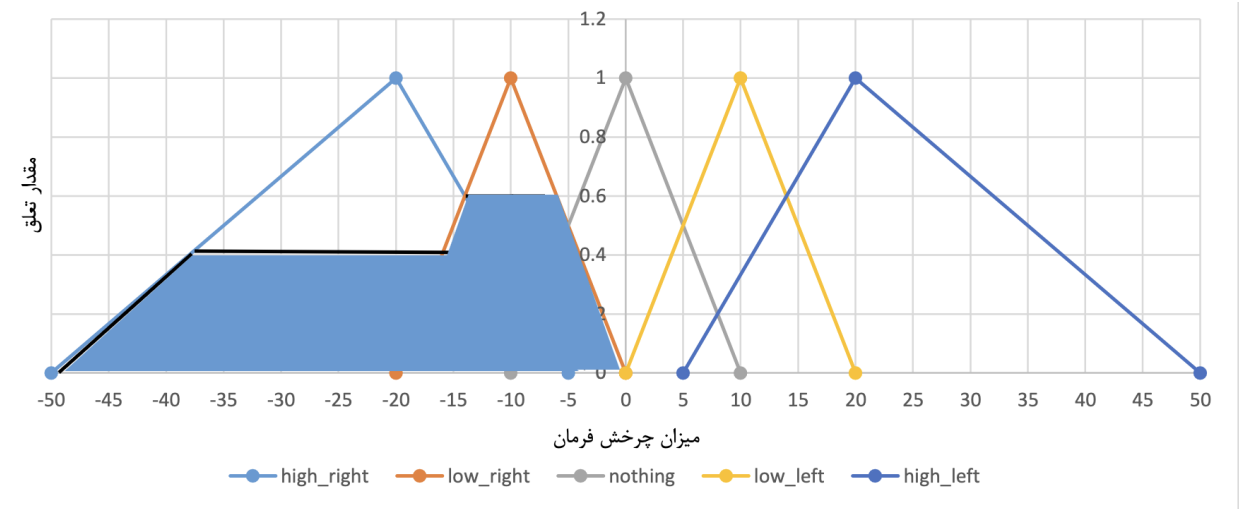
فرهاد امان ۹۹۳۱۰۰۶

```python
def calculate_close_L(x):
    if 0 <= x <= 50:
        return 1 - (x / 50)
    return 0


2 usages  Farhad Aman
def calculate_close_R(x):
    if 0 <= x <= 50:
        return 1 - (x / 50)
    return 0


1 usage  Farhad Aman
def calculate_far_L(x):
    if 100 >= x >= 50:
        return (x / 50) - 1
    return 0


```

توابع calculate یک ورودی x می‌گیرند و میزان تعلق x به آن مجموعه فازی را محاسبه می‌کنند.
این توابع براساس معادله خطهای داده شده به صورت نمودار داخل دستور کار پیاده‌سازی شده اند.

```python
self.low_left = min(calculate_moderate_L(left_dist), calculate_close_R(right_dist))
self.low_right = min(calculate_close_L(left_dist), calculate_moderate_R(right_dist))
self.high_left = min(calculate_far_L(left_dist), calculate_close_R(right_dist))
self.high_right = min(calculate_close_L(left_dist), calculate_far_R(right_dist))
self.nothing = min(calculate_moderate_L(left_dist), calculate_moderate_R(right_dist))
```

در بخش بعدی rule های داده شده در دستور کار را پیاده‌سازی می‌کنیم.
همانطور که می‌دانیم And در منطق فازی همان Min گرفتن است.
در واقع ما در اینجا خطوطی را به دست آوردیم که با استفاده از آن‌ها می‌توان نمودار مربوط به مجموعه‌های فازی را محاسبه کرد.



می‌دانیم در نقاطی که نمودار تو در تو هست باید Max بگیریم که این موضوع در تابع max-min هندل شده است.

```
def max_min(self, x):
    return max(min(self.low_right, calculate_low_right(x)),
               min(self.high_right, calculate_high_right(x)),
               min(self.low_left, calculate_low_left(x)),
               min(self.high_left, calculate_high_left(x)),
               min(self.nothing, calculate_nothing(x)))
```

این تابع در هر نقطه x میزان واقعی نمودار نهایی را محاسبه می‌کند.

برای به دست آوردن مرکز جرم طبق فرمول باید دو انتگرال را حساب کرده و آن‌ها را بر هم تقسیم کنیم برای این کار تابع integral را نوشتیم

```
def integral(self, start, limit, interval):
    x = start
    num = 0
    den = 0
    while x < limit:
        den += self.max_min(x) * interval
        num += self.max_min(x) * interval * x
        x += interval
    if den != 0:
        return float(num) / float(den)
    return 0
```

در این تابع انتگرال به صورت تقریبی محاسبه می‌شود. در واقع با استفاده از جمع مساحت مجموعه‌ای از مستطیل‌های کوچک که زیر نمودار هستند.
میزان interval هرچه کمتر باشد محاسبات دقیق‌تر اما طولانی‌تر خواهد بود.

بخش امتیازی:
پیاده‌سازی این بخش تا حد زیادی شبیه به بخش اول است.

```
def calculate_close(x):
    if 0 <= x <= 50:
        return (-x / 50) + 1
    return 0


1 usage  Farhad Aman
def calculate_moderate(x):
    if 40 <= x <= 50:
        return (x / 10) - 4
    if 50 <= x <= 100:
        return (-x / 50) + 2
    return 0


1 usage  Farhad Aman
def calculate_far(x):
    if 90 <= x <= 200:
        return (x / 110) - (9 / 11)
    if x >= 200:
        return 1
    return 0
```

دوباره توابعی برای محاسبه میزان تعلق می‌نویسیم.

```
self.low = calculate_close(center_dist)
self.medium = calculate_moderate(center_dist)
self.high = calculate_far(center_dist)
```

سپس میزان تعلق به هر مجموعه را محاسبه می‌کنیم.

```
def max_min(self, x):
    return max(min(self.low, calculate_low_speed(x)),
               min(self.high, calculate_high_speed(x)),
               min(self.medium, calculate_medium_speed(x)))
```

با استفاده از تابع max-min نمودار نهایی را به دست می‌آوریم.

```
def integral(self, start, limit, interval):
    x = start
    num = 0
    den = 0
    while x < limit:
        den += self.max_min(x) * interval
        num += self.max_min(x) * interval * x
        x += interval
    if den != 0:
        return float(num) / float(den)
    return 0
```

در نهایت از تابع انتگرال نوشته شده در بخش قبل استفاده می‌کنیم.