퍼지 추론 알고리즘

목표 속력, 현재 속력, 도로 경사 3가지를 한번에 고려하는 정속주행 퍼지 알고리즘

1. 알고리즘 구조

-멤버쉽 함수 정의

-규칙 베이스 정의

-크리스프(모호하지 않고 명확한) 입력

-멤버쉽 함수를 통한 룰 마다의 퍼지화(룰 각각의 퍼지 집합에 어느 정도 속하는지 계산)

-출력 룰에 따라 denominator, numerator 계산 및 출력

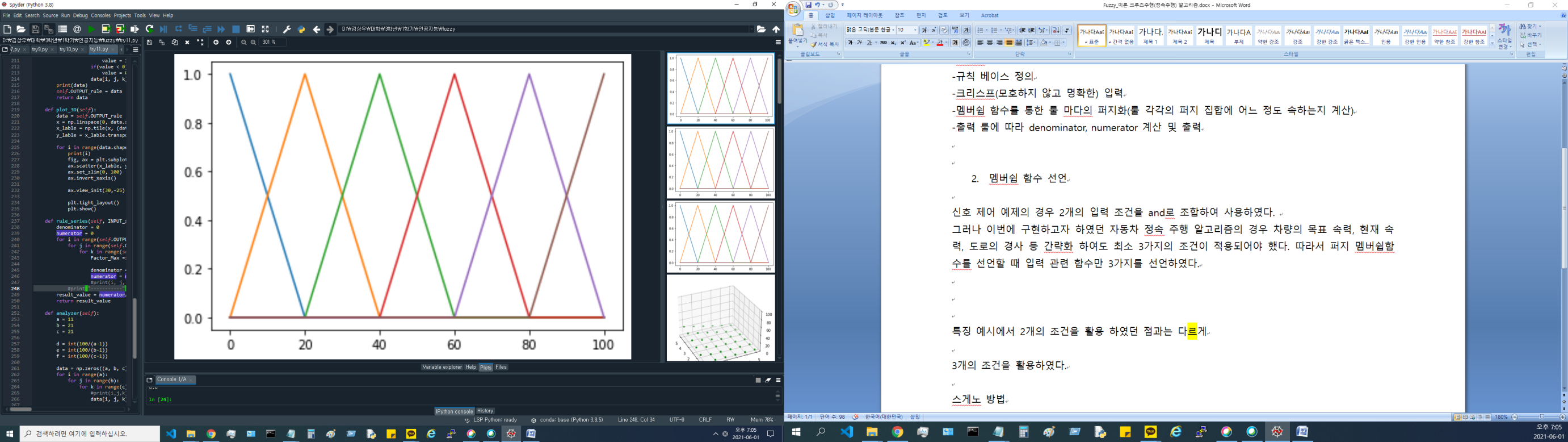
1. 멤버쉽 함수 선언

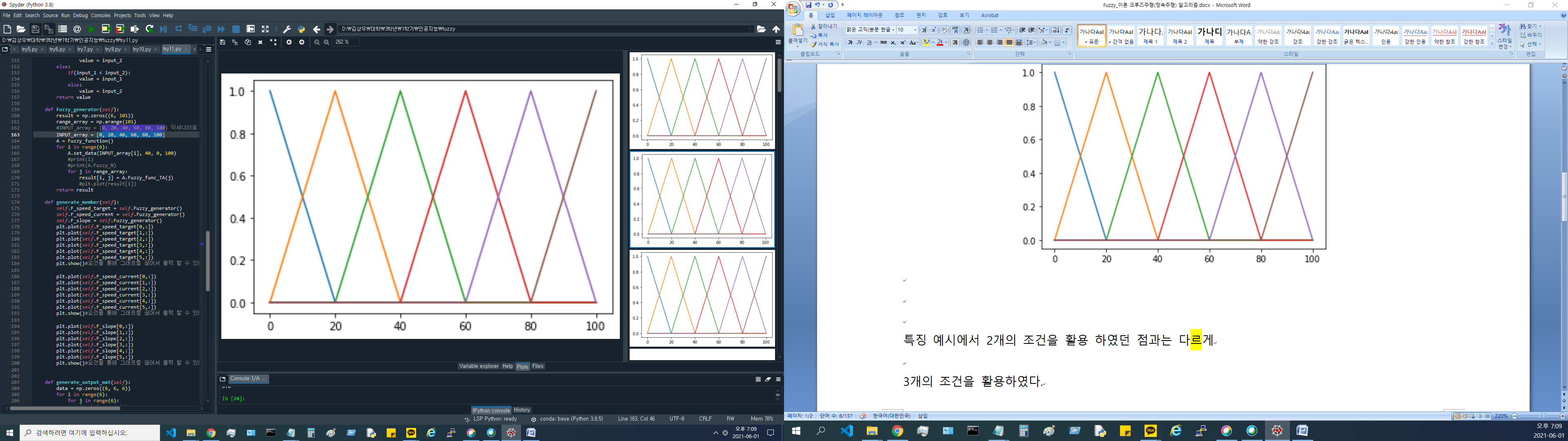
신호 제어 예제의 경우 2개의 입력 조건을 and로 조합하여 사용하였다.

그러나 이번에 구현하고자 하였던 자동차 정속 주행 알고리즘의 경우 차량의 목표 속력, 현재 속력, 도로의 경사 등 간략화 하여도 최소 3가지의 조건이 적용되어야 했다. 따라서 퍼지 멤버쉽함수를 선언할 때 입력 관련 함수만 3가지 카테고리로 선언하였다.

선언한 함수의 형상은 아래와 같이

0, 1, 2, 3, 4, 5 단계로 구분하였으며 각각 함수들의 퍼지수는 0, 20, 40, 60, 80, 100 으로 정하였다.

형상은 삼각형으로 정의하였으며 목표 속력, 현재 속력, 도로 경사 3가지 모두 동일한 형태로 선언하였다



3가지 카테고리(목표 속력, 현재 속력, 도로 경사)의 함수를 모두 확인한 모습

1. 규칙 베이스 선언

규칙 베이스는 3가지 카테고리 모두에 대한 규칙을 모두 직접 정의하기에는 100개 이상의 규칙을 모두 선언하기가 어려움으로 각각 카테고리의 성질에 맞게 수식으로 표현하고 자동으로 생성되도록 하였다.

조건 1 : 목표 속력 조건

-목표 속력이 클수록 출력이 커져야 한다.

조건 2 : 현재 속력 조건

-목표 속력보다 모자란 차이 만큼에 비례하여 출력이 커지되 현재 속력이 목표 속력보다 크다면 출력을 줄여야한다.

조건 3 : 경사 조건

-같은 목표 속도, 현재 속도 조건에서도 경사가 증가하면 경사 0~5단계에서 1단계당 출력이 10%p씩 더 커야 한다.

규칙의 간격은 0~5단계씩 3가지 카테고리이므로 6\*6\*6 = 216 개의 규칙이 생성된다.

각 규칙은 이름으로 호출할 수 없을 정도로 수가 너무 많아서, numpy로 생성한 3차원 배열인 OUTPUT\_rule에 정의하였다.

조건을 수식으로 표현하면

def generate\_output\_mat(self):

data = np.zeros((6, 6, 6))

for i in range(6):

for j in range(6):

for k in range(6):

if(j <= i):

value = 5\*i + 10\*j + 5\*k

else:

value = 15\*i + -5\*(j-i) + 5\*k

if(value < 0):

value = 0

data[i, j, k] = value

print(data)

self.OUTPUT\_rule = data

return data

위와 같다.

4차원 자료(목표 속력, 현재 속력, 도로 경사 + 출력)이기 때문에 3차원료 여러 개로 나누어서 시각화하면

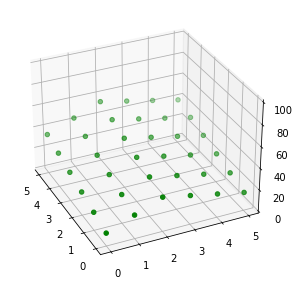
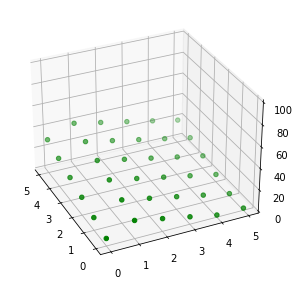
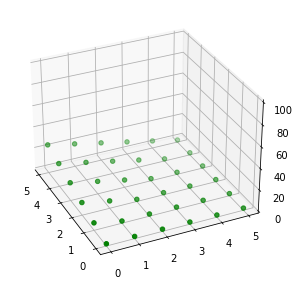
목표 속력 0~5 단계에 대해서 표의 개수를 나누고 가로축(좌우)을 현재 속력, 세로축(상하)를 도로 경사로 나타내면 아래와 같이 나타난다.

세로축을 따라 나있는 산등 모양의 경계를 기준으로 기준보다 작은 값에서는 속력이 증가하다가 기준값 이상에서는 속력이 감소하며, 세로축값이 증가할수록 출력 값이 증가하는 양상을 보인다.

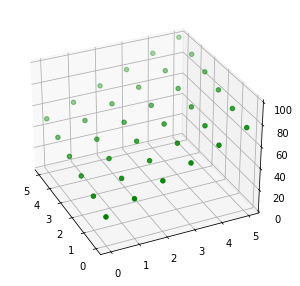
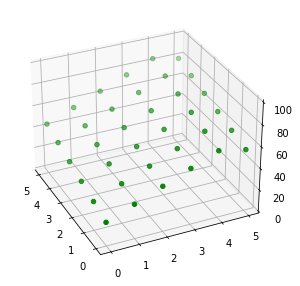
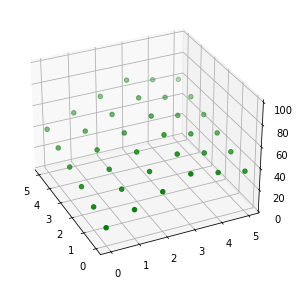
이는 경사가 심할수록 항상 출력이 점점 더 커져야 한다는 규칙과 현재 속력이 목표 속력을 추가하면 감속한다는 규칙을 잘 따르고 있다.

또한 목표속도가 증가함에 따라 그래프 출력의 전체적인 평균이 증가하는 양상도 잘 드러난다.

#목표 속력 : 0 #목표 속력 : 1 #목표 속력 : 2



#목표 속력 : 3 #목표 속력 : 4 #목표 속력 : 5



1. 퍼지화 및 비퍼지화(출력 계산)

퍼지화 및 비퍼지화 하여 결과를 출력하는 함수를 하나로 묶어서 구현 하였다. 이때, 조건의 종류를 3가지나 적용한 결과 규칙 배이스의 양이 216개로 일일이 계산하기 어려울 정도가 되었다. 따라서 다중 for문을 통하여서 계산하는 것이 가장 합리적이었다

또한 교차로 신호 제어기 예제와 같이 스게노 방법으로 비퍼지화를 구현하였다.

함수는 다음과 같은 구조로 작성 되었다.

1. 퍼지화 : 비퍼지화 역할을 하는 함수는 각각의 맴버쉽 함수에 입력된 크리스프한 값이 어느 정도 포함되는지 비교하고 각 일치도 값을 계산
2. 퍼지연산 : 3가지 조건 목표 속력, 현재 속력, 도로 경사는 모두 and 로 계산하였다 따라서 각 조건들의 최소 값을 찾는 함수를 선언하였다. 이중에서 선택된 최소값이 스게노 방법의 가중평균의 가중치로 적용되도록 하였다.
3. 비퍼지화 : 가중치와 출력값을 곱한 것의 총합을 분자로 가중치의 총합을 분모로 하여 계산하고 결과 값으로써 출력한다.

코드는 아래와 같다.

def rule\_series(self, INPUT\_speed\_target, INPUT\_speed\_current, INPUT\_slope\_current):

denominator = 0

numerator = 0

for i in range(self.OUTPUT\_rule.shape[0]):

for j in range(self.OUTPUT\_rule.shape[1]):

for k in range(self.OUTPUT\_rule.shape[2]):

Factor\_Min =self.MIN\_3(self.F\_speed\_target[i, INPUT\_speed\_target], self.F\_speed\_current[j, INPUT\_speed\_current], self.F\_slope[k, INPUT\_slope\_current])

denominator = denominator + Factor\_Min

numerator = numerator + Factor\_Min\*self.OUTPUT\_rule[i, j, k]

#print(i, j, k, Factor\_Max, self.OUTPUT\_rule[i, j, k])

#print("-----------")

result\_value = numerator/denominator

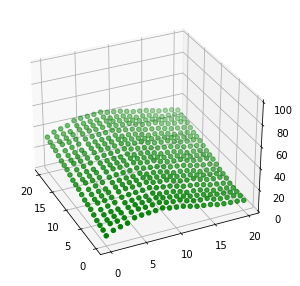
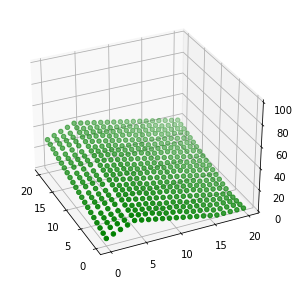
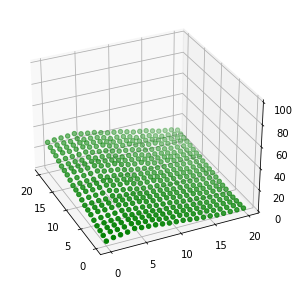
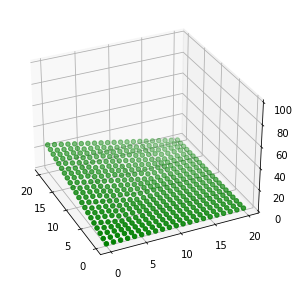
return result\_value

1. 결과 확인

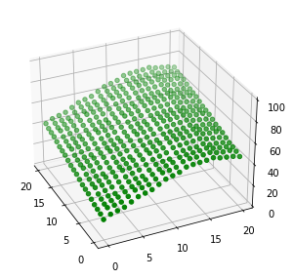
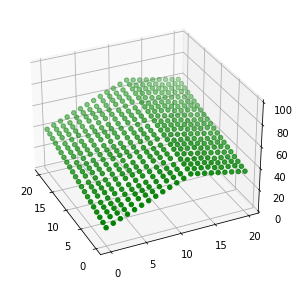
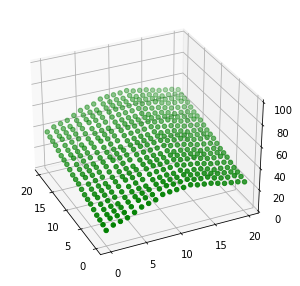
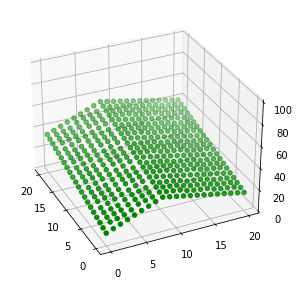
주어진 조건보다 세분화된 입력을 넣어서 퍼지이론이 규칙으로써 미리 주어지지 않은 모호한 입력에 대한 결과 값을 제대로 생성하는지 확인해 보면 아래의 표와 같이 기존에 주어지지 않은 지점의 값까지 촘촘하게 나타내는 표를 생성함을 통해 확인할 수 있다.

(각 레벨은 0일대 0, 5일 때 100의 실제 값을 가진다.)

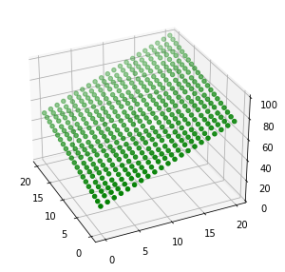
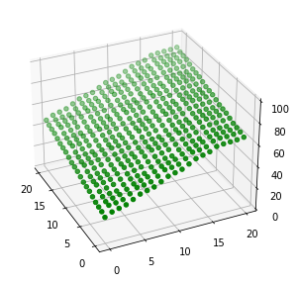
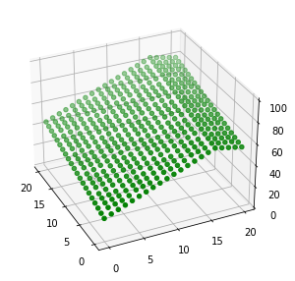
#목표 속력 0 lv #목표 속력 0.5 lv #목표 속력 1 lv #목표 속력 1.5 lv



#목표 속력 2 lv #목표 속력 2.5 lv #목표 속력 3 lv #목표 속력 3.5 lv



#목표 속력 4 lv #목표 속력 4.5 lv #목표 속력 5 lv



미리 입력한 데이터 보다 촘촘한 위치까지 데이터가 잘 생성됨을 확인할 수 있다.

1. 느낀점

3차원 같이 다변수 시스템에서도 굉장히 잘 작동한다는 특징을 알 수 있었다 드러나 3차원의 경우 값의 양상이 미리 주어진 값 주변과 미리 알려주지 않은 값 사이에서의 값이 출렁이듯이 규칙적인 잡신호가 발생하는 현상이 나타났다. 위의 결과 확인에서 3.5나 4.5의 결과 도표에서도 FUzzy연산으로 새롭게 생성된 값들이 만들어내는 표면이 울퉁불퉁해지는 현상이 생긴다. Fuzzy 시스템 중 다변수 시스템의 한계나 주의점 대해서 궁굼해졌다. 또한 이러한 자동제어의 경우 Fuzzy 인공지능 이론과 PID라는 비 인공지능 이론을 조합한 Fuzzy-Pid 제어기라는 것이 있다는 것을 알게 되는 등 Fuzzy이론과 다른 이론사이의 조합에 대해여 흥미가 생겼다.

1. 참고 자료

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=jerrypoiu&logNo=221178162267

1. 전체 코드

import numpy as np

import matplotlib.pyplot as plt

class Fuzzy\_function:

def set\_data(self, N, W, F, L):

self.Fuzzy\_N = N

self.Width = W

self.range\_first = F

self.range\_last = L

def Fuzzy\_func\_TA(self, input\_data):

if(input\_data > self.range\_last):

input\_data = self.range\_last

if(input\_data < self.range\_first):

input\_data = self.range\_first

if(input\_data == self.Fuzzy\_N):

result\_value = 1

elif((self.Fuzzy\_N - 0.5\*self.Width)< input\_data < self.Fuzzy\_N):

result\_value = 1-(2/self.Width)\*(self.Fuzzy\_N - input\_data)

elif(self.Fuzzy\_N < input\_data < (self.Fuzzy\_N + 0.5\*self.Width)):

result\_value = 1-(2/self.Width)\*(-1)\*(self.Fuzzy\_N - input\_data)

else:

result\_value = 0

if(result\_value < 0):

result\_value = 0

if(result\_value > 1):

result\_value = 1

return result\_value

class throtle\_contorll\_Fuzzy:

def \_\_init\_\_(self):

self.F\_speed\_target = 0

self.F\_speed\_current = 0

self.F\_slope = 0

self.OUTPUT\_rule = 0

def MAX\_3(self, input\_0, input\_1, input\_2):

if(input\_0 > input\_1):

if(input\_0 > input\_2):

value = input\_0

else:

value = input\_2

else:

if(input\_1 > input\_2):

value = input\_1

else:

value = input\_2

return value

def MIN\_3(self, input\_0, input\_1, input\_2):

if(input\_0 < input\_1):

if(input\_0 < input\_2):

value = input\_0

else:

value = input\_2

else:

if(input\_1 < input\_2):

value = input\_1

else:

value = input\_2

return value

def Fuzzy\_generator(self):

result = np.zeros((6, 101))

range\_array = np.arange(101)

INPUT\_array = [0, 20, 40, 60, 80, 100]

A = Fuzzy\_function()

for i in range(6):

A.set\_data(INPUT\_array[i], 40, 0, 100)

for j in range\_array:

result[i, j] = A.Fuzzy\_func\_TA(j)

return result

def generate\_member(self):

self.F\_speed\_target = self.Fuzzy\_generator()

self.F\_speed\_current = self.Fuzzy\_generator()

self.F\_slope = self.Fuzzy\_generator()

plt.plot(self.F\_speed\_target[0,:])

plt.plot(self.F\_speed\_target[1,:])

plt.plot(self.F\_speed\_target[2,:])

plt.plot(self.F\_speed\_target[3,:])

plt.plot(self.F\_speed\_target[4,:])

plt.plot(self.F\_speed\_target[5,:])

plt.show()

plt.plot(self.F\_speed\_current[0,:])

plt.plot(self.F\_speed\_current[1,:])

plt.plot(self.F\_speed\_current[2,:])

plt.plot(self.F\_speed\_current[3,:])

plt.plot(self.F\_speed\_current[4,:])

plt.plot(self.F\_speed\_current[5,:])

plt.show()

plt.plot(self.F\_slope[0,:])

plt.plot(self.F\_slope[1,:])

plt.plot(self.F\_slope[2,:])

plt.plot(self.F\_slope[3,:])

plt.plot(self.F\_slope[4,:])

plt.plot(self.F\_slope[5,:])

plt.show()

def generate\_output\_mat(self):

data = np.zeros((6, 6, 6))

for i in range(6):

for j in range(6):

for k in range(6):

if(j <= i):

value = 5\*i + 10\*j + 5\*k

else:

value = 15\*i + -5\*(j-i) + 5\*k

if(value < 0):

value = 0

data[i, j, k] = value

print(data)

self.OUTPUT\_rule = data

return data

def plot\_3D(self):

data = self.OUTPUT\_rule

x = np.linspace(0, data.shape[1]-1, data.shape[0])

x\_lable = np.tile(x, (data.shape[2], 1))

y\_lable = x\_lable.transpose()

for i in range(data.shape[0]):

print(i)

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.scatter(x\_lable, y\_lable, data[i, :, :], c='green')

ax.set\_zlim(0, 100)

ax.invert\_xaxis()

ax.view\_init(30,-25)

plt.tight\_layout()

plt.show()

def rule\_series(self, INPUT\_speed\_target, INPUT\_speed\_current, INPUT\_slope\_current):

denominator = 0

numerator = 0

for i in range(self.OUTPUT\_rule.shape[0]):

for j in range(self.OUTPUT\_rule.shape[1]):

for k in range(self.OUTPUT\_rule.shape[2]):

Factor\_Min =self.MIN\_3(self.F\_speed\_target[i, INPUT\_speed\_target], self.F\_speed\_current[j, INPUT\_speed\_current], self.F\_slope[k, INPUT\_slope\_current])

denominator = denominator + Factor\_Min

numerator = numerator + Factor\_Min\*self.OUTPUT\_rule[i, j, k]

result\_value = numerator/denominator

return result\_value

def analyzer(self):

a = 11

b = 21

c = 21

d = int(100/(a-1))

e = int(100/(b-1))

f = int(100/(c-1))

data = np.zeros((a, b, c))

for i in range(a):

for j in range(b):

for k in range(c):

data[i, j, k] = self.rule\_series(d\*i, e\*j, f\*k)

x = np.linspace(0, data.shape[1]-1, data.shape[1])

x\_lable = np.tile(x, (data.shape[2], 1))

y\_lable = x\_lable.transpose()

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})

ax.scatter(x\_lable, y\_lable, data[i, :, :], c='green')

ax.set\_zlim(0, 100)

ax.invert\_xaxis()

ax.view\_init(30,-25)

plt.tight\_layout()

plt.show()

B = throtle\_contorll\_Fuzzy()

B.generate\_member()#퍼지 멤버 집합을 생성함

B.generate\_output\_mat()#출력 규칙을 생성함

B.plot\_3D()

B.analyzer()