

BURSA TEKNİK ÜNİVERSİTESİ

Mühendislik ve Doğa Bilimleri Fakültesi – Bilgisayar Mühendisliği Bölümü



BURSA TEKNİK ÜNİVERSİTESİ

BLM0468 Esnek Hesaplamaya Giriş

Bahar 2025

Bulanık Mantık Dönem Projesi Ödevi

**Kablosuz Sensör Ağları (WSN) içindeki düğüm lokalizasyon problemi için
Mamdani tipi bulanık çıkarım sistemi (FIS)**

**Esra İLBOĞA
21360859063**

**Emre GÜÇLÜ
20360859013**

İçindekiler

1.	Giriş :.....	3
2.	Veri Seti Hakkında Bilgi :	3
3.	Yöntem ve Uygulama:.....	3
3.1.	Kullanılan Yöntemler :.....	4
3.2.	Kod Blokları ve Açıklamaları :	4
3.3.	Sonuç ve Değerlendirme :.....	15
4.	Kaynakça :.....	17

1. Giriş :

Kablosuz Sensör Ağları (Wireless Sensor Networks – WSNs), uzaktan algılama, çevresel izleme, askeri uygulamalar ve akıllı şehirler gibi birçok alanda kritik bir rol oynamaktadır. Bu sistemlerin temel yapı taşlarından biri, düğümlerin konum bilgilerinin doğru bir şekilde belirlenmesidir.

Düğüm lokalizasyonu, konumu bilinmeyen düğümlerin (unknown nodes), konumu bilinen çapalı düğümler (anchor nodes) yardımıyla konumlarının tahmin edilmesi sürecidir. Bu sürecin doğruluğu, **ALE (Average Localization Error)** metriği ile değerlendirilir. Bu çalışmada, verilen UCI veri seti kullanılarak, **Mamdani tipi bulanık çıkarım sistemi (FIS)** ile ALE değerinin tahmin edilmesi amaçlanmıştır.

2. Veri Seti Hakkında Bilgi :

- Veri Seti :**

[https://archive.ics.uci.edu/dataset/844/average+localization+error+\(ale\)+in+sensor+node+localization+process+in+wsns](https://archive.ics.uci.edu/dataset/844/average+localization+error+(ale)+in+sensor+node+localization+process+in+wsns)

Veri setinin genel özellikleri aşağıdaki tablo ve açıklamalarla detaylandırılmıştır :

Sütun	Açıklama
1	Anchor Ratio (Çapa Oranı)
2	Transmission Range (İletim Aralığı)
3	Node Density (Düğüm Yoğunluğu)
4	Iterations (Yineleme Sayısı)
5	ALE (Average Localization Error - hedef)
6	Standart Sapma (X Bu projede kullanılmaz)

Toplam 107 örnek, 4 giriş değişkeni, 1 hedef değişken ALE kullanılarak proje gerçekleştirilecektir. Yani veri kümesinde bulunan **standart sapma sütunu kullanılmamış**, yalnızca ilk 5 sütun değerlendirmeye alınmıştır.

3. Yöntem ve Uygulama:

Sensör düğümlerinin konumlarının tahmin edilmesine yönelik olarak geliştirilen bulanık mantık tabanlı tahmin modeli detaylandırılmıştır. Mamdani bulanık çıkarım sistemi temel

alınarak geliştirilen bu modelde, farklı üyelik fonksiyonu türleri ve berraklaştırma yöntemleri kullanılmıştır.

3.1. Kullanılan Yöntemler :

- **Bulanık Mantık Tipi:** Mamdani
- **Bulanıklaştırma (Fuzzification):**
 - Üçgensel (Triangular)
 - Gauss (Gaussian)
- **Berraklaştırma (Defuzzification):**
 - centroid (Ağırlıklı Ortalama)
 - som (Sum of Maximums - Maksimumların ToplAMI)
 - mom (Mean of Maximum – Maksimumların Ortalaması)
 - bisector (Alanın Ortası – Bisector of Area)
- **Uygulanan Kombinasyonlar:**

Bu çalışmada 2 farklı üyelik fonksiyonu (MF) ve 4 farklı defuzzification yöntemi kullanılarak **toplamda 8 farklı model kombinasyonu** test edilmiştir:

MF Tipi	Defuzzification Yöntemi
Triangular	Centroid
Triangular	SOM
Triangular	MOM
Triangular	Bisector
Gaussian	Centroid
Gaussian	SOM
Gaussian	MOM
Gaussian	Bisector

3.2. Kod Blokları ve Açıklamaları :

- Python ortamına **scikit-fuzzy** adlı kütüphaneyi yüklenir. **scikit-fuzzy**, Python'da bulanık mantık sistemleri (fuzzy logic) oluşturmak ve çalıştmak için kullanılan bir kütüphanedir.

```
!pip install scikit-fuzzy
```

Kod Blogu 3.2.1

- Veri kümesi ile çalışmak, bulanık mantık modeli kurmak, sonuçları analiz edip grafiklerle sunmak için gerekli kütüphaneler yüklenir.

```
from IPython import get_ipython
from IPython.display import display
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from sklearn.model_selection import train_test_split
```

Kod Bloğu 3.2.2

- pandas:** Veri analizi ve düzenlemesi için kullanılır.
 - numpy:** Sayısal işlemler ve dizilerle çalışma sağlar.
 - matplotlib.pyplot ve seaborn:** Veri görselleştirme işlemleri için kullanılır.
 - sklearn.metrics:** Modelin başarı düzeyini ölçmek için çeşitli hata metrikleri sunar.
 - skfuzzy:** Bulanık mantık sistemlerini oluşturmak ve kontrol etmek için kullanılır.
 - train_test_split (sklearn):** Veri setini eğitim ve test olmak üzere bölmek için kullanılır.
- Veri seti projeye dahil edilir ve veri setinin ilk 5 satırı incelenmek üzere görüntülenir.

```
df = pd.read_csv('esnek.csv')
df.head()
```

Kod Bloğu 3.2.3

	anchor_ratio	trans_range	node_density	iterations	ale	sd_ale
0	30	15	200	40	0.773546	0.250555
1	15	15	100	70	0.911941	0.498329
2	30	15	100	50	0.814867	0.255546
3	15	20	100	20	1.435332	0.394603
4	30	15	100	40	1.265909	0.302943

Görsel 3.2.1

Görsel 3.2.1'de Kod Bloğu 3.2.3'teki kodun çıktısı görüntülenmektedir.

- Standart sapmayı barındıran ‘sd_ale’ sütunu kullanılmayacağından df veri çerçevesinden silinir.

```
df = df.drop(columns=['sd_ale'], errors='ignore')
```

Kod Bloğu 3.2.4

- Veri setindeki **sayısal sütunların istatistiksel özeti** (ortalama, std, min, max, çeyrekler) yazdırılır. Bu özellikle üyelik fonksiyonlarını tanımladığımız süreçte yardımcı olacak bilgiler taşır.

```
print(df.describe())
```

Kod Bloğu 3.2.5

	anchor_ratio	trans_range	node_density	iterations	ale
count	107.000000	107.000000	107.000000	107.000000	107.000000
mean	20.523364	17.878505	159.813084	47.887850	0.983471
std	6.739556	3.107235	71.189109	24.668874	0.408313
min	10.000000	12.000000	100.000000	14.000000	0.394029
25%	15.000000	15.000000	100.000000	30.000000	0.655368
50%	18.000000	17.000000	100.000000	40.000000	0.899102
75%	30.000000	20.000000	200.000000	70.000000	1.196418
max	30.000000	25.000000	300.000000	100.000000	2.568407

Görsel 3.2.2

Görsel 3.2.2 ile **Kod Bloğu 3.2.5**’in çıktısı görüntülenmektedir. Böylece istatistiksel veriler net ve analize açık bir şekilde görülebilmektedir.

- Veri dağılımlarını, aykırı değerleri daha rahat anlayabilmek ve analiz etmek üzere boxplot grafikleri çizilir.

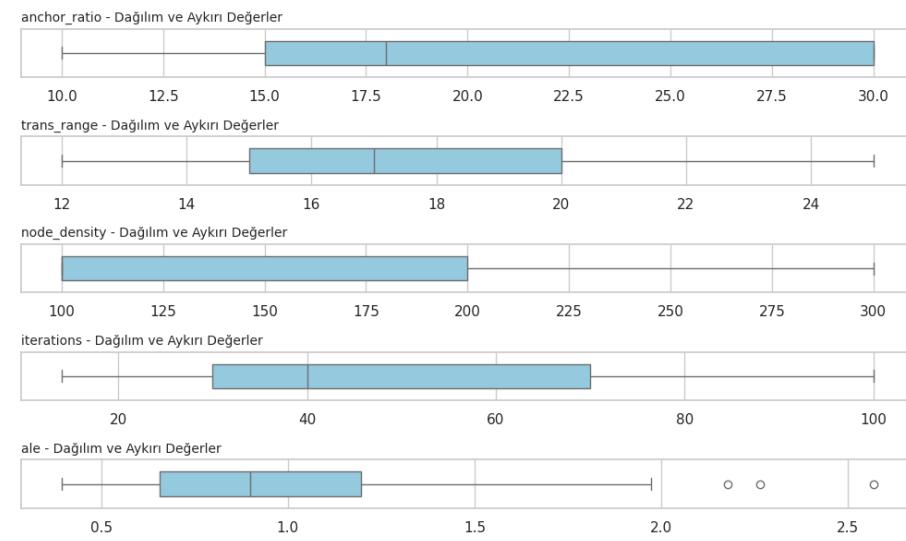
```
sns.set(style="whitegrid", palette="pastel")
plt.figure(figsize=(10, 6))
for i, column in enumerate(df.columns):
    plt.subplot(len(df.columns), 1, i + 1)
    sns.boxplot(x=df[column], color="skyblue", width=0.5)
    plt.title(f'{column} - Dağılım ve Aykırı Değerler", fontsize=10, loc="left")
    plt.xlabel("") # x-label'leri kaldır
    plt.tight_layout(pad=1.0)
plt.show()
```

Kod Bloğu 3.2.6

Kod Bloğu 3.2.6 ile veri setindeki her sayısal sütun için **yatay kutu grafiği (boxplot)** oluşturarak verinin dağılımını ve aykırı değerlerini görselleştirir.

- sns.set() ile grafik stili belirlenir.
- plt.figure() çizim alanını tanımlar.
- Döngü ile her sütun için sns.boxplot() ile kutu grafiği çizilir

- plt.tight_layout() boşlukları düzenler, plt.show() tüm grafikleri ekrana getirir.



Görsel 3.2.3

Kod Bloğu 3.2.6 ile görselleştirilen **yatay kutu grafikleri (boxplot)** Görsel 3.2.3'te görülmektedir.

- Bulanık sistemdeki giriş ve çıkış değişkenleri değer aralıkları ile tanımlanır. Yani üyelik fonksiyonu tanımlanır.

```
# Girdiler
anchor_ratio = ctrl.Antecedent(np.arange(10, 30.1, 0.5), 'anchor_ratio')
trans_range = ctrl.Antecedent(np.arange(12, 25.1, 0.5), 'trans_range')
node_density = ctrl.Antecedent(np.arange(100, 301, 10), 'node_density')
iterations = ctrl.Antecedent(np.arange(15, 101, 5), 'iterations')

# Çıktı
ale = ctrl.Consequent(np.arange(0.4, 2.57, 0.01), 'ale')
```

Kod Bloğu 3.2.7

Kod Bloğu 3.2.7 ile dört giriş ve bir çıkış değişkeni bulanık mantık sistemi için tanımlar ve her bir değişkenin evreni (değer aralığı) belirtilir. Örnek olarak np.arange(10, 30.1, 0.5) ile bir değişkenin 10'dan 30'a kadar olan aralıkta 0.5 adımlarla değer alabileceği belirtilir. Bu değerler daha sonra **düşük, orta ve yüksek** gibi üyelik fonksiyonlarıyla sınıflandırılacaktır. ctrl.Antecedent sınıfı, bir değişkenin bulanık sistemde bir **giriş (input)** olduğunu tanımlar.

- Giriş ve çıkış değişkenleri bulanıklaştmak için farklı sekillerde üçgensel (triangular) ve gauss (gaussian) üyelik fonksiyonları tanımlanır. Sistem, bu tanımlara göre verileri

düşük, orta, yüksek gibi bulanık kümelere ayırır.

```
# Triangular üyelik fonksiyonları

def define_triangular_mfs():

    anchor_ratio['low'] = fuzz.trimf(anchor_ratio.universe, [10, 10, 18])
    anchor_ratio['medium'] = fuzz.trimf(anchor_ratio.universe, [15, 20, 25])
    anchor_ratio['high'] = fuzz.trimf(anchor_ratio.universe, [22, 30, 30])

    trans_range['low'] = fuzz.trimf(trans_range.universe, [12, 12, 16])
    trans_range['medium'] = fuzz.trimf(trans_range.universe, [14, 17, 20])
    trans_range['high'] = fuzz.trimf(trans_range.universe, [18, 25, 25])

    node_density['low'] = fuzz.trimf(node_density.universe, [100, 100, 150])
    node_density['medium'] = fuzz.trimf(node_density.universe, [120, 180, 220])
    node_density['high'] = fuzz.trimf(node_density.universe, [200, 300, 300])

    iterations['low'] = fuzz.trimf(iterations.universe, [10, 10, 30])
    iterations['medium'] = fuzz.trimf(iterations.universe, [20, 50, 70])
    iterations['high'] = fuzz.trimf(iterations.universe, [60, 100, 100])

    ale['low'] = fuzz.trimf(ale.universe, [0.4, 0.4, 0.9])
    ale['medium'] = fuzz.trimf(ale.universe, [0.7, 1.0, 1.3])
    ale['high'] = fuzz.trimf(ale.universe, [1.2, 2.0, 2.0])

# Gauss üyelik fonksiyonları

def define_gaussian_mfs():

    anchor_ratio['low'] = fuzz.gaussmf(anchor_ratio.universe, 12, 2)
    anchor_ratio['medium'] = fuzz.gaussmf(anchor_ratio.universe, 20, 2)
    anchor_ratio['high'] = fuzz.gaussmf(anchor_ratio.universe, 28, 2)

    trans_range['low'] = fuzz.gaussmf(trans_range.universe, 13, 1.5)
    trans_range['medium'] = fuzz.gaussmf(trans_range.universe, 17, 2)
    trans_range['high'] = fuzz.gaussmf(trans_range.universe, 23, 2)

    node_density['low'] = fuzz.gaussmf(node_density.universe, 120, 20)
    node_density['medium'] = fuzz.gaussmf(node_density.universe, 180, 30)
    node_density['high'] = fuzz.gaussmf(node_density.universe, 260, 20)

    iterations['low'] = fuzz.gaussmf(iterations.universe, 25, 8)
    iterations['medium'] = fuzz.gaussmf(iterations.universe, 50, 10)
    iterations['high'] = fuzz.gaussmf(iterations.universe, 85, 10)

    ale['low'] = fuzz.gaussmf(ale.universe, 0.6, 0.1)
```

```
ale['medium'] = fuzz.gaussmf(ale.universe, 1.0, 0.15)
ale['high'] = fuzz.gaussmf(ale.universe, 1.8, 0.2)
```

Kod Bloğu 3.2.8

Üçgensel üyelik fonksiyonları, üç noktayla tanımlanan basit üçgen şekillerinden oluşur; bu sayede her değişken için “düşük”, “orta” ve “yüksek” gibi bulanık kümeler oluşturulmuştur. Diğer yandan **Gauss üyelik fonksiyonları** daha yumuşak geçişler sağlar ve çan eğrisi şeklinde dir; **fuzz.gaussmf()** fonksiyonu kullanılarak merkez ve genişlik belirlenir. Bu iki yaklaşım sayesinde, sistemin giriş ve çıkış değişkenleri bulanık kümelere aitlik derecelerine göre modellenir ve karar kuralları oluşturulabilir.

- Tanımlanan üçgensel (triangular) ve gauss (gaussian) üyelik fonksiyonları görselleştirilerek eğrileri çizilir.

# Triangular MF'leri tanımla define_triangular_mfs() # .view() ile göster anchor_ratio.view() trans_range.view() node_density.view() iterations.view() ale.view()	# Gaussian MF'leri tanımla define_gaussian_mfs() # .view() ile göster anchor_ratio.view() trans_range.view() node_density.view() iterations.view() ale.view()
--	--

Kod Bloğu 3.2.9

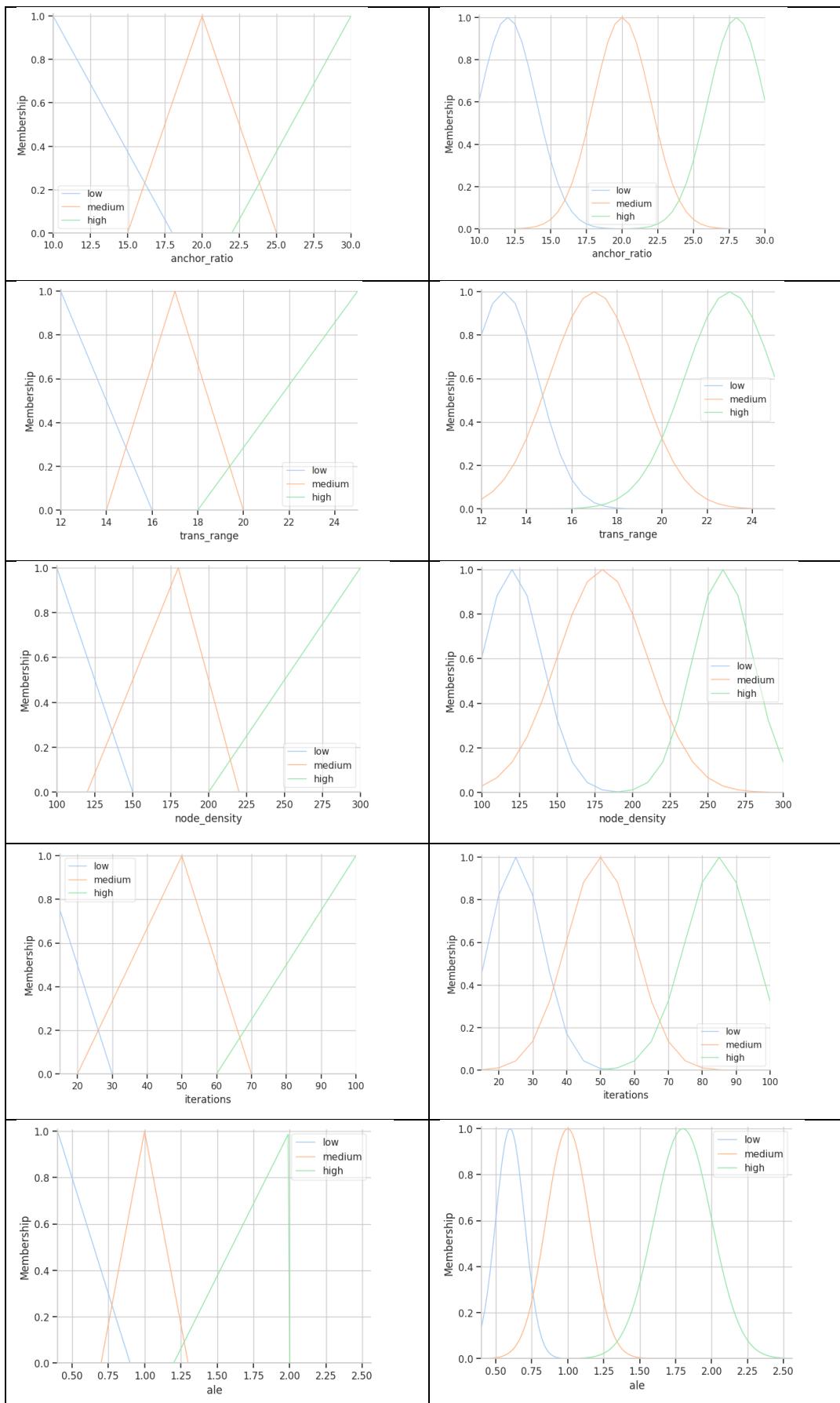
Kod bloğu 3.2.9, daha önce tanımlanan **üçgensel (triangular) ve gauss (gaussian)** üyelik fonksiyonlarını kullanarak tüm giriş ve çıkış değişkenlerinin **bulanık kümelerini grafiksel olarak görselleştirir**.

İlk olarak define_triangular_mfs() fonksiyonu çağrılarak her değişken için "low", "medium" ve "high" seviyelerinde üçgensel üyelik fonksiyonları oluşturulur. Ardından .view() komutları ile bu fonksiyonlar **matplotlib** kullanılarak çizilir.

Benzer şekilde, define_gaussian_mfs() fonksiyonu çağrıldığında gauss üyelik fonksiyonları tanımlanır ve aynı şekilde görselleştirilir.

Grafiklerde:

- **X ekseni**: Değişkenin değer aralığını,
- **Y ekseni**: İlgili değerin üyelik derecesini (0–1 arası) temsil eder.



Görsel 3.2.4

Görsel 3.2.4 ile bulanık kümelerin grafikleri görsel olarak gösterilmektedir.

- Bulanık mantık sisteminde, farklı giriş değerlerine göre çıkış olan Ortalama Lokalizasyon Hatası (ALE) için kurallar tanımlanır.

```
def define_rules():
    return [
        ctrl.Rule(anchor_ratio['low'] & trans_range['low'] & node_density['low'] & iterations['low'], ale['high']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['low'] & iterations['medium'], ale['medium']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['medium'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['low'] & iterations['medium'], ale['medium']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['medium'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['medium'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['high'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['low'] & node_density['high'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['high'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['high'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['medium'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['medium'] & iterations['high'], ale['medium']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['high'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['high'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['medium'] & node_density['high'] & iterations['high'], ale['medium']),
        ctrl.Rule(anchor_ratio['low'] & trans_range['high'] & node_density['low'] & iterations['high'], ale['medium']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['medium'] & node_density['medium'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['low'] & node_density['high'] & iterations['high'], ale['medium']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['high'] & node_density['high'] & iterations['medium'], ale['high']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['medium'] & node_density['high'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['low'] & node_density['medium'] & iterations['high'], ale['medium']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['high'] & node_density['low'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['low'] & node_density['high'] & iterations['low'], ale['high']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['high'] & node_density['low'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['medium'] & trans_range['low'] & node_density['medium'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['low'] & node_density['high'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['high'] & node_density['low'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['low'] & node_density['medium'] & iterations['high'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['high'] & node_density['low'] & iterations['high'], ale['high']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['low'] & node_density['medium'] & iterations['low'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['high'] & node_density['medium'] & iterations['medium'], ale['low']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['high'] & node_density['high'] & iterations['high'], ale['high']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['medium'] & node_density['low'] & iterations['high'], ale['high']),
        ctrl.Rule(anchor_ratio['high'] & trans_range['medium'] & node_density['high'] & iterations['low'], ale['low']),
        ctrl.Rule(node_density['low'] & iterations['low'], ale['high']),
        ctrl.Rule(trans_range['medium'] & node_density['low'], ale['medium']),
    ]
```

Kod Bloğu 3.2.10

Veriler değerlendirilerek Kod Bloğu 3.2.10'de verilen fonksiyon **ctrl.Rule** objeleri

halinde **bulanık mantık kurallarını** tanımlar. Dört giriş değişkeninin (anchor_ratio, trans_range, node_density, iterations) üyelik fonksiyonlarındaki durumlarına göre (örn. 'low', 'medium', 'high'), çıkış olan ALE (Average Localization Error) için bir üyelik değeri atanır. Kurallar, girişlerin farklı kombinasyonlarına karşılık çıkışın nasıl olacağını belirtir:

- Örneğin; ctrl.Rule(anchor_ratio['low'] & trans_range['low'] & node_density['low'] & iterations['low'], ale['high']), sensörlerin çapa oranı düşük, iletim aralığı düşük, düğüm yoğunluğu düşük ve iterasyon sayısı düşük ise, ALE değeri yüksek olur.

Bazı kurallar 4 değişkeni birden kullanırken, en alta yer alan 2 kural sadece 2 değişkeni içerir; bu şekilde daha genel durumlar da dikkate alınmıştır.

- Örneğin; ctrl.Rule(trans_range['medium'] & node_density['low'], ale['medium']), sensörün iletim aralığı orta seviyede ve düğüm yoğunluğu düşük olduğunda, Ortalama Lokalizasyon Hatası (ALE) değeri orta olur.

Bu kurallar, sisteme verilen girişlere göre ALE değerini bulanık mantıkla tahmin etmesini sağlar ve sistemin karar mekanizmasının temelini oluşturur.

- Bulanık mantık sistemini simüle ederek kuralları kontrol etmek ve düzenleyebilmek adına verilen giriş değerleri için iki farklı üyelik fonksiyonu türü (üçgen ve gauss) kullanarak bulanık çıkarım sistemi ile ALE tahmin sonuçları görselleştirilir.

```
def run_and_visualize(mf_type='triangular'):  
    print(f"\n== {mf_type.upper()} Üyelik Fonksiyonu ==")  
    if mf_type == 'triangular':  
        define_triangular_mfs()  
    elif mf_type == 'gaussian':  
        define_gaussian_mfs()  
    else:  
        raise ValueError("mf_type 'triangular' veya 'gaussian' olmalıdır.")  
    rules = define_rules()  
    ale_ctrl = ctrl.ControlSystem(rules)  
    ale_simulator = ctrl.ControlSystemSimulation(ale_ctrl)  
    ale_simulator.input['anchor_ratio'] = 15  
    ale_simulator.input['trans_range'] = 20
```

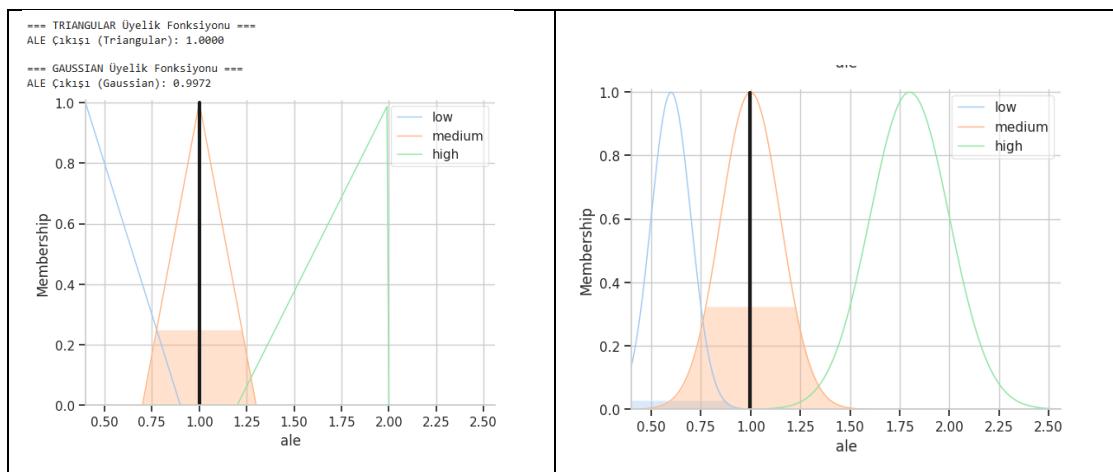
```

ale_simulator.input['node_density'] = 100
ale_simulator.input['iterations'] = 70
ale_simulator.compute()
print(f"ALE Çıkışı ({mf_type.title()}): {ale_simulator.output['ale']:.4f}")
ale.view(sim=ale_simulator)
run_and_visualize('triangular')
run_and_visualize('gaussian')

```

Kod Bloğu 3.2.11

Kod Bloğu 3.2.11 ile **ControlSystemSimulation** ile simülatör oluşturulur. Giriş değerleri atanır ve **compute()** ile bulanık kurallar uygulanarak ALE çıktıları hesaplanır. Hesaplanan değer hem sayısal olarak yazdırılır hem de **ale.view()** ile iki farklı üyelik fonksiyonu ve sonuç grafiği görselleştirilir. (15, 20, 100, 70 değerleri çıktıları simüle edilmeye çalışılan bir veri sadece bu değiştirilebilir.)



Görsel 3.2.5

Görsel 3.2.5 ile (15, 20, 100, 70) için simüle edilen çıktı değerleri ve üyelik fonksiyonları görüntülenmiştir.

- Farklı **üyelik fonksiyonları (triangular ve gaussian)** ve farklı **defuzzification yöntemleri (centroid, som, bisector, mom)** kullanılarak bulanık mantık sistemi çalıştırılır ve tahminler yapılır. Sonrasında gerçek değerlerle tahmin edilen değerler karşılaştırılarak **MAE** ve **RMSE** hata metrikleri hesaplanır ve sonuçlar yazdırılır.

```

def run_fuzzy_system(X, y_true, mfs='triangular', defuzz='centroid'):
    if mfs == 'triangular':
        define_triangular_mfs()

```

```

else:
    define_gaussian_mfs()
ale.defuzzify_method = defuzz
rules = define_rules()
system = ctrl.ControlSystem(rules)
simulator = ctrl.ControlSystemSimulation(system)
predictions = []
for i in range(len(X)):
    simulator.input['anchor_ratio'] = X.iloc[i, 0]
    simulator.input['trans_range'] = X.iloc[i, 1]
    simulator.input['node_density'] = X.iloc[i, 2]
    simulator.input['iterations'] = X.iloc[i, 3]
    simulator.compute()
    predictions.append(simulator.output['ale'])
return np.array(predictions)

from sklearn.metrics import mean_absolute_error, mean_squared_error
def evaluate(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    return mae, rmse

X = df.drop(columns=['ale'])
y_true = df['ale']
for mf_type in ['triangular', 'gaussian']:
    for method in ['centroid', 'som', 'bisector', 'mom']: # scikit-fuzzy default centroid,
        som (sum of max), bisector, mom diğer seçenek için custom gereklidir
        print(f"==> MF: {mf_type.upper()}, Defuzz: {method}")
        preds = run_fuzzy_system(X, y_true, mfs=mf_type, defuzz=method)
        mae, rmse = evaluate(y_true, preds)
        print(f'MAE: {mae:.4f}, RMSE: {rmse:.4f}')

```

Kod Bloğu 3.2.12

Kod Bloğu 3.2.12, verilen giriş verileri (**X**) için seçilen üyelik fonksiyonu tipi ve **defuzzification** yöntemi ile bulanık mantık tabanlı **ALE** tahminleri üretir. **run_fuzzy_system** fonksiyonu, her bir gözlem için sensör parametrelerini sisteme

verip, **compute()** ile bulanık çıkarımı çalıştırarak net (sayısal) sonuçlar elde eder. Daha sonra, **evaluate** fonksiyonu gerçek **ALE değerleri (y_true)** ile **tahmin edilen değerleri (y_pred)** karşılaştırıp, performansı **MAE** ve **RMSE** ile ölçer. Tüm kombinasyonlar denenerek hangi **üyelik fonksiyonu** ve **defuzzification** yönteminin en iyi performansı verdiği net bir şekilde görülebilir. Bu yapı, bulanık mantık sisteminin farklı parametrelerle nasıl davranışını deneyel olarak değerlendirmeye olanak sağlar.

3.3. Sonuç ve Değerlendirme :

Bu çalışmada, 2 farklı üyelik fonksiyonu (**Üçgensel (Triangular)**, **Gauss (Gaussian)**) ve 4 farklı defuzzification yöntemi(**centroid (Ağırlıklı Ortalama)**, **som (Sum of Maximums - Maksimumların ToplAMI)**, **mom (Mean of Maximum – Maksimumların Ortalaması)**, **bisector (Alanın Ortası – Bisector of Area)**) kullanılarak 8 farklı model kombinasyonu oluşturulmuştur.

Kod Bloğu **3.2.12** ile **MAE (Ortalama Mutlak Hata)** ve **RMSE (Kök Ortalama Kare Hata)** kullanılarak bu 8 model kombinasyonunun performansları ölçülmüştür.

```
==> MF: TRIANGULAR, Defuzz: centroid  
MAE: 0.2116, RMSE: 0.2975  
==> MF: TRIANGULAR, Defuzz: som  
MAE: 0.3370, RMSE: 0.4053  
==> MF: TRIANGULAR, Defuzz: bisector  
MAE: 0.2114, RMSE: 0.2949  
==> MF: TRIANGULAR, Defuzz: mom  
MAE: 0.2322, RMSE: 0.3274  
==> MF: GAUSSIAN, Defuzz: centroid  
MAE: 0.2216, RMSE: 0.3111  
==> MF: GAUSSIAN, Defuzz: som  
MAE: 0.3335, RMSE: 0.4431  
==> MF: GAUSSIAN, Defuzz: bisector  
MAE: 0.2179, RMSE: 0.3082  
==> MF: GAUSSIAN, Defuzz: mom  
MAE: 0.2365, RMSE: 0.3505
```

Görsel 3.2.6 Performans değerlendirmesi çıktıları.

Bahsi geçen performans çıktıları **Görsel 3.2.6**'da gösterilmiştir. Elde edilen sonuçlar, **triangular** üyelik fonksiyonunun, özellikle **centroid** ve **bisector** defuzzification yöntemleri ile en iyi performansı verdienen ortaya koymaktadır. Bu kombinasyonlar, diğer yöntemlere kıyasla daha düşük hata değerleri (**MAE ~0.21**, **RMSE ~0.29**) sunarak tahmin doğruluğunu artırmaktadır.

Gaussian üyelik fonksiyonu da benzer performans sergilemekle birlikte, **triangular** fonksiyonun hafif bir gerisinde kalmıştır. Öte yandan, her iki üyelik fonksiyonu için **som**

(sum of maxima) yöntemi diğer **defuzzification** tekniklerine göre belirgin şekilde daha yüksek hata oranlarına sahiptir ve bu nedenle tercih edilmemelidir.

Bu analiz, bulanık mantık sistemlerinde doğru üyelik fonksiyonu ve **defuzzification** yönteminin seçiminin model doğruluğu üzerinde kritik etkisi olduğunu göstermektedir. Modelin genel başarısı, sensör çapa oranı, iletim aralığı, düğüm yoğunluğu ve iterasyon sayısı gibi parametreler için bulanık kuralların etkinliğine ve seçilen parametrelerin optimizasyonuna bağlıdır.

Sonuç olarak, ALE tahmininde **triangular** üyelik fonksiyonu ile **centroid** veya **bisector defuzzification** yöntemlerinin kullanılması, tahmin hatalarının minimize edilmesi açısından en uygun seçeneklerdir. Bu bulgu, uygulamada daha doğru ve güvenilir tahminler yapılmasına olanak sağlamaktadır.

4. Kaynakça :

- <https://www.geeksforgeeks.org/fuzzy-logic-introduction/>
- <https://www.datacamp.com/tutorial/fuzzy-logic-in-ai>
- <https://medium.com/deep-learning-turkiye/python-ile-bulan%C4%B1k-mant%C4%B1k-modellemesi-74459dc27308>
- <https://github.com/AdelAKA/FuzzyLogic-Robot-Controller>
- <https://github.com/BipinRimal314/fuzzylogic>