YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NÖRAL – GENETİK TABANLI OPTİMAL BULANIK KONTROLÖRÜN GERÇEKLENMESİ ve DC SERVOMOTORA UYGULANMASI

Elektrik Yük. Müh. İbrahim Beklan KÜÇÜKDEMİRAL

F.B.E Elektrik Müh. Anabilim Dalında
Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi	: 24. 05. 2002
Tez Danışmanı	: Prof. Dr. Galip CANSEVER (YTÜ)
İkinci Tez Danışmanı	:
Jüri Üyeleri	: Prof. Dr. Oruç BİLGİÇ (YTÜ)
	: Prof. Dr. Çingiz HACIYEV (İTÜ)
	: Doç. Dr. Canbolat UÇAK (İTÜ)
	: Doç. Dr. Tülay YILDIRIM (YTÜ)

İSTANBUL, 2002

İÇİNDEKİLER

SİMGE LİSTESİ

$\mu_F\{.\}$	Bulanık üyelik fonksiyonu
Q	Bulanık bağıntı
P	Bulanık bağıntı
FP	Bulanık önerme
p	Bulanık önerme
q	Bulanık önerme
$\dot{+}$	Bulanık bileşim yöntemi
*	Bulanık kesişim yöntemi
$\Re$	Reel sayılar uzayı
$\bar{y}^l$	*l.* bulanık kümenin merkezi
M	Kural tabanındaki toplam kural sayısı
N	Popülasyonu oluşturan kromozom sayısı
P_i	*i.* kromozomun seçilme olasılığı
f_i	*i.* kromozomun uyumu
P_m	Mutasyon oranı
P_c	Çaprazlama oranı
$\Psi\{.\}$	Aktivasyon fonksiyonu
d_i	*i.* nöronun çıkışının istenen değeri
S_i	*i.* nöronun girişinde oluşan net toplam
θ	Eşik
w_i	*i.* ağırlık
o_i	*i.* nöronun çıkışı
J	Maliyet fonksiyonu
η	Öğrenme oranı
K_P	Oran katsayısı
K_I	İntegral katsayısı
K_D	Türev katsayısı
K'_D	GA ile bulunan optimal türev katsayısı
K'_P	GA ile bulunan optimal oran katsayısı
K'_I	GA ile bulunan optimal integral katsayısı
$K_{P_{\max}}$	Oran katsayısının alabileceği en büyük değer
$K_{P_{\min}}$	Oran katsayısının alabileceği en küçük değer
$K_{I_{\max}}$	İntegral katsayısının alabileceği en büyük değer
$K_{I_{\min}}$	İntegral katsayısının alabileceği en küçük değer
$K_{D_{\max}}$	Türev katsayısının alabileceği en büyük değer
$K_{D_{\min}}$	Türev katsayısının alabileceği en küçük değer
T_i	İntegral zaman sabiti
e	Hata işareti
T_s	Örnekleme periyodu
S_e	Hata işaretine ait giriş ölçekleme faktörü
S_d	Hatanın değişimine ait giriş ölçekleme faktörü

S_u	Çıkış ölçekleme faktörü
U_m	Normalize edilmiş hata işaretine ait uzayın genişliği
U_{dm}	Normalize edilmiş hata işaretinin değişimine ait uzayın genişliği
U_{out}	Normalize edilmiş kontrol işaretine ait uzayın genişliği
$F\{.\}$	Bulanık fonksiyon
NB	Negatif Büyük üyelik fonksiyonu
NM	Negatif Orta üyelik fonksiyonu
NS	Negatif Küçük üyelik fonksiyonu
Z	Sıfır Üyelik fonksiyonu
PS	Pozitif Küçük üyelik fonksiyonu
PM	Pozitif Orta üyelik fonksiyonu
PB	Pozitif Büyük üyelik fonksiyonu
VS	Çok küçük üyelik fonksiyonu
S	Küçük üyelik fonksiyonu
SB	Biraz küçük üyelik fonksiyonu
MB	Orta büyük üyelik fonksiyonu
B	Büyük üyelik fonksiyonu
VB	Çok büyük üyelik fonksiyonu
α	Öz uyarlama mekanizması çıkışı
β	Sürtünme katsayısı
K_m	Moment sabiti
E_b	Ters EMK
K	Ters EMK katsayısı
n	Motorun devir sayısı
ω	Motor açısal hızı
t_r	Yükselme zamanı
t_s	Yerleşme zamanı
Δe	Hatanın değişimi

KISALTMA LİSTESİ

CHR	Cien-Hrones-Reswick
BMK	Bulanık Mantık Kontrol
NGTOPI	Nöral-Genetik Tabanlı Optimal Oran İntegral
NGTOFPI	Nöral-Genetik Tabanlı Optimal Bulanık Oran İntegral
NGTOFPID	Nöral-Genetik Tabanlı Optimal Bulanık Oran İntegral Türev
NGTOSTFPI	Nöral-Genetik Tabanlı Optimal Öz-Uyarlamalı Bulanık Oran İntegral
NGTOSTFP-ID	Nöral-Genetik Tabanlı Optimal Öz-Uyarlamalı Bulanık Oran-İntegral Türev
PD	Oran Türev
PI	Oran İntegral
PID	Oran İntegral Türev
TS	Takagi-Sugeno
GA	Genetik Algoritma
IAE	Integral of Absolute Error
ITAE	Integral of Time Weighted Absolute Error
PWM	Pulse Width Modulation (Darbe Genişlik Modülasyonu)
ADC	Analog Digital Converter
YSA	Yapay Sinir Ağları

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Doktora tez çalışması oldukça zahmetli, özveri isteyen zaman zaman insanı çıkmazlara sokan, bazen karamsarlığa iten; ancak bir şeyler elde ettikçe ve ortaya attığınız fikirler bir bir meyve vermeye başlayınca heyecan ve azimle içinizi dolduran dünyanın ender çalışmalarından biridir. Yapılan çalışmanın bir eşinin daha olmadığını bilmek, dünya bilimine yeni bir şeyler eklediğinizi hissetmek ise inanın tarifi olmayan bir duygudur.

Bu çalışmanın seyiri esnasında bir çok kişinin katkıları olmuştur. İlk olarak lisans öğrenimimin üçüncü sınıfında tanıdığım ve o günden beri danışmanlığımı yapan değerli hocam Prof. Dr. Galip Cansever'e teşekkür etmek isterim. Bana yaşattığı, sıradan bir öğrenci hoca ilişkisi değil; bir ağabey-kardeş ilişkisidir. Benim için çizdiği bilim rotası, ve bu rotadan sapmadan ilerlememi sağlayan itici gücü zannediyorum kolay kolay her doktoranta nasip olmaz. Yaptığımız tartışmalar, bildiri ve makale çalışmaları, sıkıcı anları ortadan kaldıran sohbetlerimiz bu çalışmanın yürümesini sağlayan en önemli unsurlardı. Teşekkürler hocam.

Anne ve babam; sizlerin karşılıksız maddi ve manevi desteği, bir akademisyen olmam için yapmış olduğunuz her türlü fedakarlık olmasaydı bugünleri ancak rüyamda görürdüm. Her ikinize de sonsuz teşekkürler.

Ve Sevgili Neslihan, nişanlım, hayat arkadaşım; tez çalışmamın başından sonuna kadar hep yanımdaydın. Dertlerimi paylaştın, sorunlarımı dinledin; bir şeyler başardığımda mutluluğuma ortak oldun. Çıkmaza düştüğümde karamsarlığımı silmeme yardımcı oldun. Tezin metin kısımlarını okudun, düzeltmeler yaptın. Bana vermiş olduğun motivasyon olmasaydı bu çalışma nasıl ilerlerdi bilemiyorum. Gösterdiğin sabır ve her türlü manevi destek için sana da kucak dolusu teşekkür ederim.

Elektrik Makineleri Anabilim Dalı teknisyeni Habib abi (Özak), çalışmaya ait deney düzeneğinin kurulmasında defalarca yardım ettin, ayrıca her türlü arıza durumunda imdadıma koştun, her şey için teşekkürler. Ayrıca, Anabilim Dalımız öğretim üyelerinden Yrd. Doç. Dr. Şeref Naci Engin ve Dr. Yük. Müh. A. Faruk Bakan'a da değerli görüş ve yardımları için teşekkür etmek isterim. Bu kısıtlı satırlarda ismini sayamadığım birçok kişinin bu çalışmada az veya çok emeği geçmiştir. Emeği geçen herkese teşekkür ederim.

Bu proje sizlerin desteği olmadan hiç bir zaman gerçekleşemezdi.

ÖZET

Bulanık mantık kontrolörlerinin (BMK) tasarım işlemi genellikle dört evreden oluşur. Bunlar: giriş ve çıkış üyelik fonksiyonlarının belirlenmesi ve ayarı; giriş ve çıkış ölçekleme faktörlerinin belirlenmesi ve ayarı; kural tabanının tasarımı ve çıkarım ile netleştirme yöntemlerinin belirlenmesidir. Bunların arasında, giriş ve çıkış ölçekleme faktörlerinin belirlenmesine ayrı bir önem gösterilmelidir. Zira, çıkış ölçekleme faktörü sistemin kararlılığı ve osilasyon eğilimi üzerinde oldukça etkiliyken; giriş ölçekleme faktörleri kontrolörün iyi seçilmiş bir çalışma bölgesi için temel hassasiyetini belirlemektedir. Ancak günümüzde bile ölçekleme faktörlerinin BMK sistemleri üzerindeki etkileri tam olarak belirlenebilmiş değildir.

Bu çalışmada giriş ve çıkış ölçekleme faktörlerinin optimal değerlerinin belirlenmesi için yeni ve sistematik bir yöntem sunulmuştur. Sunulan yöntem Genetik Algoritma (GA) ve Yapay Sinir Ağları (YSA) kullanmaktadır. GA, optimal ölçekleme faktörlerinin bulunması esnasında kullanılırken, YSA, her bir jenerasyonda uyum değerlerinin hesaplanmasında kullanılmaktadır.

Çalışmada ayrıca, yeni bir öz-uyarlamalı bulanık mantık kontrolörü, NGTOSTFP-ID (Nöral-Genetik Tabanlı Optimal Bulanık P-ID) sunulmaktadır. Burada, çıkış ölçekleme faktörü, bağımsız bir bulanık kontrolör tarafından ayar edilmektedir. Çıkış ölçekleme faktörünü ayarlayan bulanık kontrolör, kontrol edilen proses değişkenine ait hata (e) ve hatanın değişimini (Δe) giriş olarak kabul etmektedir. Ayrıca, ana kontrolörün kural tabanının belirlenmesinde uzman operatörün görüşleri yerine, gradyen düşüm tekniğine dayanan bir yöntem kullanılmıştır. Diğer taraftan, bulanık mantık kontrolörünün geçici zaman cevaplarında gösterdiği olumsuzlukları ortadan kaldırmak amacı ile kontrolör mimarisine ileri sürümlü bir integratör ve geri besleme yolu üzerine türev kontrolörü eklenmiştir.

Ortaya atılan optimizasyon metodunun geçerliliğini göstermek için mevcut yöntem sırası ile, geleneksel PI, geleneksel bulanık PI, hibrid tipte bulanık PID, öz-uyarlamalı bulanık PI ve NGTOSTFP-ID kontrolörlerine uygulanmıştır. Ayrıca ele alınan kontrolörlerin performans cevapları sabit mıknatıslı DC motor ve yük içeren bir servo sistem üzerinde gerçek zamanlı olarak karşılaştırılmıştır. Yapay sinir ağı tabanlı sistem tanıyıcılar ve genetik optimizasyon işlemini gerçekleştiren hesaplayıcılar dahil tüm bulanık kontrolörler, C++ dili ile programlanıp, bir Pentium PC aracılığı ile sisteme uygulanmıştır. Ortaya atılan öz-uyarlamalı bulanık kontrolör dahil tüm kontrolörlerin performanslarının karşılaştırılması esnasında, aşım, yerleşme ve yükselme zamanları, mutlak hatanın integrali (IAE) ve mutlak hatanın zaman ağırlıklı integrali (ITAE) ile, sistemin rampa şeklinde değişen bir yörüngeye verdiği cevap gibi değişik performans kriterleri kullanılmıştır. Son olarak, bahsi geçen performans kriterleri göz önüne alındığında, sunulan yöntem ile ayarlanan NGTOSTFP-ID kontrolörü, yukarıda adı geçen diğer kontrolörlere göre dikkate değer bir performans artışı sergilemektedir.

Anahtar Kelimeler: Bulanık mantık kontrol, genetik algoritma, yapay sinir ağları, sabit mıknatıslı DC motor, ölçekleme faktörü, öz-uyarlamalı kontrol.

ABSTRACT

The design process of a fuzzy logic controller (FLC) can be divided usually into four stages. These are determining and tuning the input and output-membership functions, determining and tuning the input and output-scaling factors, designing the rule base, and choosing the implication, inference and defuzzification methods. Among them, much more attention should be paid for determination of the input and output scaling factors since the output-scaling factor has the most influence on stability and oscillation tendency of the system, whereas the input-scaling factors have a significant influence on basic sensitivity of the controller with respect to the optimal choice of the operating regions of the input signals. However, relative importance of the input and output scaling-factors to the performance of a FLC system is yet to be fully established.

In this work a new and systematic method for the determination of the optimal values of input and output scaling factors is proposed. The method makes use of Genetic Algorithms (GAs) and Artificial Neural Networks (ANNs). The method uses the GA for searching the optimal values of scaling factors while the ANN is used for the computation of fitness function in each generation.

On the other hand a new self-tuning FLC architecture NGTOSTFP-ID, which stands for Neuro-Genetic Based Optimal Self-tuning Fuzzy P-ID is proposed. Here, the output-scaling factor is adjusted on-line by an independent fuzzy controller according to the trends of the controlled process. The rule base for tuning the output-scaling factor is defined on error (e) and change of error (Δe) of the controlled variable. Also, the rule base of the controller is designed by a gradient decent technique without using any experience of a skilled operator. Furthermore, in order to eliminate the drawbacks e.g. poor transient performance of the FLC, a feedforward integrator and a feedback derivative controller blocks are added to the controller architecture.

In order to demonstrate the validity of the proposed optimization method, the method is applied to conventional -PI, conventional fuzzy -PI, hybrid type fuzzy -PID, self-tuning type fuzzy -PI and NGTOSTFP-ID controllers. Also the performances of these controllers are compared on a real-time servo system including a permanent magnet DC motor and a load. All the controllers, neural network system identifier and genetic optimizers are coded under C++, and implemented into the system via a Pentium PC. The performances of the controllers including the proposed self-tuning FLC are compared in terms of several performance measures such as peak overshoot, settling time, rise time, integral of absolute error (IAE) and integral of the time weighted absolute error (ITAE), in addition to the responses due to ramp type trajectory. In each case the proposed scheme, tuned by the introduced optimization method, showed a remarkably improved performance over the other above-mentioned controllers.

Keywords: Fuzzy logic control, genetic algorithms, artificial neural networks, permanent magnet DC motor, scaling factor, self-tuning control.

1. GİRİŞ

Son yirmi yıldır, bulanık mantık kontrol (BMK), bulanık kümeler teorisinin en çok üzerinde çalışılır ve en çok meyve veren dalı haline gelmiştir. Bulanık kontrol üzerine ilk çalışmalar Mamdani ve arkadaşları tarafından yapıldı. Onları bulanık kontrol sistemleri üzerinde çalışmaya teşvik eden ise Zadeh'in "*Bulanık kümeler teorisine dayanan dilsel yaklaşım ve sistem analizi*" adlı makalesi olmuştur. Daha sonraları bu konudaki çalışmalar peşi sıra geldi. Su kalitesinin kontrolünde bulanık yaklaşım (Yagishita vd., 1985); otomatik tren işletim sistemi (Yasunobu vd., 1983), otomatik konteynır vinci sistemi (Yasunobu vd., 1986), asansör sistem kontrolü (Fujitec, 1988), nükleer reaktör kontrolü (Bernard, 1988), otomobil transmisyon kontrolü (Kasai vd., 1988) bu çalışmalardan bazılarıdır. Bu çalışmalar, iyi tanımlanamamış sistemlerin, uzman kişilerin tecrübelerinden yararlanılarak bulanık kontrol yardımı ile kontrol edilebileceğinin kanıtıdır.

Son yıllarda bulanık kontrol sistemleri o derece gelişti ki; bu alanda yapılan çalışmaları birkaç sayfa ile özetlemek nerede ise imkansız bir hal aldı. Yine de, bulanık kontrol sistemlerinin gelişmesinde önemli yer tutan ve bu konunun kilometre taşlarını oluşturan bazı çalışmalar, Çizelge 1.1'de sunulmuştur.

Çizelge 1.1 Bulanık kontrolün gelişmesinde pay sahibi olan önemli çalışmalar

1973	Zadeh	Dilsel yaklaşım
1974	Mamdani ve Assilian	Buhar makinesinin kontrolü
1976	Rutherford	Kontrol algoritmalarının analizi
1977	Willaeys	Optimal bulanık kontrol
1980	Tong	Atık su arıtma sistemleri
1983	Hirota ve Pedrycz	Rastlantısal bulanık kümeler (kontrol)
1983	Takagi ve Sugeno	Bulanık kontrol kurallarının türetilmesi
1983	Yasunobu, Miyamoto	Öngörülü bulanık kontrol
1984	Sugeno ve Murakami	Model arabanın park kontrolü
1985	Kiszka, Grupta	Bulanık sistemlerin kararlılığı
1985	Togai ve Watanabe	Bulanık çipi
1986	Yamakawa	Bulanık kontrol donanım sistemi
1988	Dubois ve Prade	Yaklaşık nedensellik

Bulanık kontrolün dayandığı bulanık mantık, insanoğlunun düşünme sistemini ve dilsel izah tarzını temel alır. Bulanık mantık, yeryüzündeki olayların kesin taraflarından çok, yaklaşıklıklar üzerinde durur. Aslında bulanık kontrol sistemleri, uzman kişinin görüşlerine dayanan dilsel bir kontrol stratejisidir. BMK, özellikle kontrol edilen sistemin analizinin çok

zor olduğu karmaşık durumlarda veya sistem bilgilerinin yetersiz olduğu durumlarda klasik kontrol sistemlerinden daha üstün cevaplar vermektedir. Bir BMK'nın temeli, genellikle uzman bir kişinin bilgisinden yararlanılarak oluşturulan kural tabanına dayanır. İlgili giriş-çıkış uzayına ait dilsel değişkenler üyelik fonksiyonları ile önceden tanımlanır. Genellikle, başarılı bir BMK tasarımı için; giriş ve çıkış ölçekleme faktörlerinin seçimi, kural tabanının oluşturulması, üyelik fonksiyonlarının tasarımı, kuralların icra edilmesi oldukça önemlidir. Tasarım esnasında belirlenmesi gereken bu değişkenlerin fazlalığı, uygulamadan uygulamaya farklılıklar göstermesi, kuralların icrasındaki hesaplama güçlükleri nedeni ile günümüzde bile bulanık kontrolörlerin tasarımında tam bir sistematik yöntem ortaya konulamamıştır. Yukarıda adı geçen tasarım parametrelerinden en önemlisi hiç kuşkusuz ölçekleme faktörlerinin düzgün belirlenmesidir. Zira ölçekleme faktörleri sistem performansını oldukça fazla etkilemektedir. Kural tabanlarının yapıları genellikle birbirlerine benzerlik gösterir. Ayrıca sistem cevabı üzerindeki etkileri de oldukça kısıtlıdır. Üyelik fonksiyonlarının şekilleri genellikle hesaplamada kolaylık sağlanması amacı ile üçgensel biçimde seçilir. Üyelik fonksiyonlarının ilgili uzaydaki yerlerinin tespiti ise ölçekleme faktörünün değiştirilmesi ile aynı anlama gelir. Yani, üyelik fonksiyonları sabit tutularak ölçekleme faktörleri üzerinde yapılacak değişiklikler ile; ölçekleme faktörleri sabit tutularak üyelik fonksiyonlarının ilgili uzay üzerinde yerlerinin değiştirilmesi şeklinde yapılacak ayar işleminin birbirinden farkı yoktur. Bu nedenle ölçekleme faktörlerinin ayarı ile ilgili olarak birçok araştırmacı tarafından çeşitli yöntemler geliştirilmiştir.

He vd.; geleneksel Oran – İntegral – Türev (PID) kontrolörlerinin katsayılarının ayarlanması işlemi için bulanık kurallar kullanmışlardır. Bu çalışmada, oran katsayısı (K_p), integral zaman sabiti (T_i) ve türev zaman sabiti (T_d), Ziegler – Nichols ayar yöntemi ile ayar edilmiş, daha sonra gerçek zamanlı olarak hata ve hatanın değişimine dayanan kural tabanı ile güncellenmiştir. Sonuç olarak; sunulan yöntemin, geleneksel yöntemlere göre üstünlükleri rapor edilmiştir. Genellikle ölü zamana sahip ikinci mertebeden sistemlerde, aşımların azaltılması yönünde dikkate değer başarılı sonuçlar elde edilmiştir. Ancak aşımlar azaltılırken, yükselme zamanlarının artması, bu yöntemin bir dezavantajıdır (He vd., 1993).

Nomura vd.; daha çok öz uyarlamalı BMK sistemleri üzerinde durmuşlar ve bulanık değişkenlerin optimal değerlerini elde etmek için gradyen düşüm tekniğini kullanmışlardır. Bu çalışmada kontrolör parametreleri; BMK çıkışı ile, arzu edilen çıkış arasındaki farktan ortaya çıkan hatanın karesini minimize edecek şekilde, iteratif olarak ayar edilmiştir. Ayar işleminin yapıldığı parametreler; keskin değerler, üyelik fonksiyonlarının genişlikleri ve

merkezleridir (Nomura vd., 1991).

Zheng, Oran – İntegral (PI) tipindeki BMK'ların ayar edilebilecek parametreleri üzerinde durmuştur. Yaptığı çalışmada, öncelikle etkisi global olan parametrelerin ayarı üzerinde durmuş, daha sonra etkisi lokal olan parametrelerin ayarına değinmiştir. Kendisi yaptığı çalışmada sistematik bir ayar yöntemi sunmaktan çok, sistem performansını etkileyen bir çok faktörden bahsetmiş, bunların birbirleri ile olan ilişkileri üzerinde durmuş ve ayar yöntemleri ile ilgili önerilerde bulunmuştur. Zheng, bu çalışmada; BMK'nın parametrelerinin ayar edilmesinde temel olarak geleneksel PI kontrolörlerin oran ve integral katsayılarından yararlanılması gerektiğinden söz etmiş, bu sabitlerin belirlenemediği durumlarda ise deneme ve yanılma yönteminden yararlanılması gerektiğini önermiştir (Zheng, 1992).

Yoshida, tüm prosesleri ölü zamana sahip birinci mertebeden sisteme indirgemiş; giriş ve çıkış ölçekleme faktörlerinin ayarı için sistem parametrelerinin de içinde bulunduğu deneysel bir formül ortaya atmıştır. Bu yöntemle yüksek mertebeden sistemler için başarılı bir performans elde edilememiştir (Yoshida, 1990).

Hayashi, otomatik ayarlamalı bulanık kontrolörler için iki ayrı ayar fonksiyonu ortaya atmıştır. Bunlardan birincisi, geleneksel PI tipi kontrolörlerin ayarında kullanılan Cien-Hrones-Reswick (CHR) metodunun giriş ve çıkış ölçekleme faktörlerinin belirlenmesi üzerine yapılan uyarlamadır. İkincisi ise, keskin değerlerin iyi bir aşım ve yükselme zamanı sağlayacak yönde değiştirilmesi üzerinedir (Hayashi, 1991).

Palm, giriş ölçekleme faktörlerinin optimal değerlerinin belirlenmesinde giriş – çıkış çapraz korelasyon fonksiyonundan yararlanmıştır. Ancak, burada giriş parametrelerinin Gauss dağılımını izlediği ve belirsiz oldukları kabul edilmiştir. Optimal giriş ölçekleme faktörleri, bu giriş – çıkış arasındaki istatistiksel ölçütü veren çapraz korelasyon fonksiyonunun maksimize edilmesi ile bulunmuştur (Palm, 1995).

Palm ve yukarıda adı geçen diğer bilim adamlarının yaptığı gibi; Li ve Gatland; giriş ve çıkış ölçekleme faktörlerinin ayarı üzerinde durmuş; buna karşın, kural tabanı ve üyelik fonksiyonlarının ayar işlemine daha az önem vermiştir. Giriş ve çıkış ölçekleme faktörlerinin ayarı için sunulan yöntemin temelleri deneme – yanılma yöntemine dayanmaktadır. Ele aldıkları kontrolör ise bir bulanık PID kontrolörüdür (Li ve Gatland, 1996).

Maeda ve Murakami, giriş ve çıkış ölçekleme faktörlerinin ayarı için kural tabanlı bir şema geliştirmişlerdir. Ayrıca benzer yöntemleri, Takagi-Sugeno (TS) modeli için de

oluşturmuşlardır. Bulanık kural tabanlı ayar mekanizması üç tip kural içermektedir. Bu kurallar; aşım, yükselme zamanı ve genlik işaretlerine dayanmaktadır. Ölçekleme faktörlerinin ayar işleminden sonra kontrol kurallarının keskin değerleri her bir örnekleme zamanında, performans ölçütü ve arzu edilen çıkıştan sapma işaretine göre yeniden ayar edilmektedir (Maeda ve Murakami, 1992).

Lee, PI tipi bulanık kontrolörlerde kontrol işaretinin depolanması sonucu ortaya çıkan aşımların yok edilmesi için klasik bulanık-PI kontrolörlere sıfırlama mekanizması ilave etmiştir. Bu çalışmada kullanılan bulanık kontrolörlerden birincisi, hata ve hatanın değişimini giriş olarak alıp, çıkışta sıfırlama faktörünü belirlemektedir. İkinci bulanık kontrolör ise, giriş olarak, sistem çıkış hatası ile kontrol işaretini kullanmaktadır. Simülasyon sonuçları, kontrolörün ikinci mertebeden entegratör içeren lineer sistemlerin geçici rejim cevaplarını oldukça başarılı bir düzeyde etkilediğini göstermektedir (Lee, 1993).

Mudi ve Pal, çalışmalarında, çıkış ölçekleme faktörünün önemine değinmişler ve hızlı, dayanıklı bir bulanık-PI tipinde kontrolör meydana getirmek için çıkış ölçekleme faktörünü gerçek zamanlı olarak ayarlayan bir mekanizma ortaya atmışlardır. Ortaya attıkları öz uyarlamalı mekanizmayı, değişik türden lineer ve lineer olmayan sistemler üzerinde, simülasyon ortamında denemişlerdir. Elde ettikleri simülasyon sonuçları, aşım ve yerleşme zamanı açısından oldukça başarılıdır (Mudi ve Pal, 1999).

Li ve Shieh, Genetik Algoritma (GA) tabanlı bir bulanık-PID kontrolör ile minimum fazlı olmayan bir sistemin alt aşım sorunlarını gidermişler ve sistemin geçici rejim cevaplarını düzeltmişlerdir. Yaptıkları çalışmada minimum fazlı olmayan sistemin sorunlarını iki farklı bulanık kontrolör içeren bir mimari ile çözmüşlerdir. Bunlardan birincisi, bulanık-PI; ikincisi bulanık-PD tipinde kontrolörlerdir. Bulanık-PI tipindeki kontrolörün görevi, kararsız sıfırların etkilerini ortadan kaldırmak; bulanık-PD tipindeki kontrolörün görevi ise, aşımları engellemektir. Her iki kontrolörün de kuralları GA ile tespit edilmiştir. Simülasyon sonuçları kontrolörün başarısının oldukça yüksek olduğunu göstermiştir (Li ve Shieh, 2000).

Belarbi ve Titel'in çalışmalarında, bulanık kontrol sistemi yapay sinir ağları şeklinde modellenmekte ve sistem parametreleri bu yapay sinir ağları modeli üzerinden optimize edilmektedir. Ortaya atılan algoritmanın gücünü göstermek için ters sarkaç sisteminin kontrolünü ve Van der Pool sisteminin kontrolünü simülasyon ortamında denemişlerdir (Belarbi ve Titel, 2000).

Yeh, 1999 yılında yayınladığı çalışmasında, büyük ölçekli lineer olmayan sistemlerin

kontrolünde GA destekli bulanık kontrolör tasarımına yer vermiştir. Bu çalışmada Yeh; giriş ve çıkış ölçekleme faktörlerini GA kullanarak sisteme ait matematiksel model üzerinden optimize etmiştir. Kural tabanı ise gradyen düşüm algoritması yardımı ile tasarlanmıştır. Tasarladığı Bulanık kontrolör mekanizmasını Puma 560 robotunun ve iki ters sarkaçlı sistemin matematiksel modellerini kullanarak simülasyon ortamında denemiştir (Yeh, 1999).

Zhou ve Lai, GA'ları, bulanık sistemlerin üyelik fonksiyonlarının belirlenmesinde kullanmışlar ve tasarladıkları kontrolörü üçüncü mertebeden zamanla değişmeyen lineer bir sistem üzerine simülasyon ortamında uygulamışlardır. Yaptıkları çalışma; GA tabanlı bulanık kontrolörlerin maksimum aşım ve yerleşme zamanları bakımından, klasik PID (Oran – İntegral – Türev) kontrolörlerden üstün olduklarını ortaya koymaktadır (Zhou ve Lai, 2000).

Hu vd., çalışmalarında, PID tipi bulanık kontrolörlerin tasarlanmasında yarı sistematik metotlar üzerinde durmuşlardır. Yaptıkları çalışmada en önemli yenilik, ortaya attıkları kontrolörün tek girişli, 3 kurallı ve en fazla 6 ayar parametresi içermesidir. Tasarladıkları kontrolörü ölü zamanlı ve doyumlu, birinci, ikinci ve beşinci mertebeden sistemlere uygulamışlar ve klasik yöntemlerle tasarlanan sistemlere göre daha iyi sistem yanıtları, kararlılık marjı açısından da yüksek performanslar elde etmişlerdir (Hu, vd., 1999).

Chiaberge, Bene, Pascoli, Lazzerini, Maggiore ve Reyneri 1999'da yaptıkları ve "Mixing Fuzzy Nöral and Genetic Algorithms in an Integrated Design Environment for Intelligent Controllers" adını verdikleri çalışmada, bulanık sistemlerin, genetik algoritmaların ve yapay sinir ağlarının zeki kontrol sistemlerinde beraberce nasıl kullanılabileceklerini açıklamışlar ve bu üç yöntemi karşılaştırarak birbirlerinin eksik yönlerini hibrid bir sistemde nasıl çözdüklerini göstermişlerdir (Chiaberge vd., 1999).

Yukarıda adı geçen çalışmalar, BMK'ların ayar metotlarının geliştirilmesi üzerine yapılan güncel çalışmalardan bazılarıdır. Bunca çalışmaya rağmen, BMK'lar için, hala sistematik bir ayar yöntemi geliştirilememiştir. Bazı çalışmalarda bulanık olmayan ayar yöntemleri, bazılarında bulanık ayar yöntemleri kullanılmıştır. Bunun asıl nedeni BMK'ların; PI, PD ve PID kontrolörlerindekine benzer sabit bir yapıya sahip olmayışıdır. Ayrıca; üyelik fonksiyonlarının tanımlanması, dilsel değişkenlerin sayısının belirlenmesi, standart kural tabanının belirlenmesi, bulanıklaştırma ve netleştirme stratejilerinin belirlenmesi için hala iyi tanımlanmış bir kriter mevcut değildir.

Bu çalışmada; yukarıda bahsi geçen eksiklikler ortadan kaldırılmaya çalışılmış ve optimal bir bulanık kontrolörün tasarımı için sistematik bir yöntem sunulmuştur. Optimizasyon işlemi,

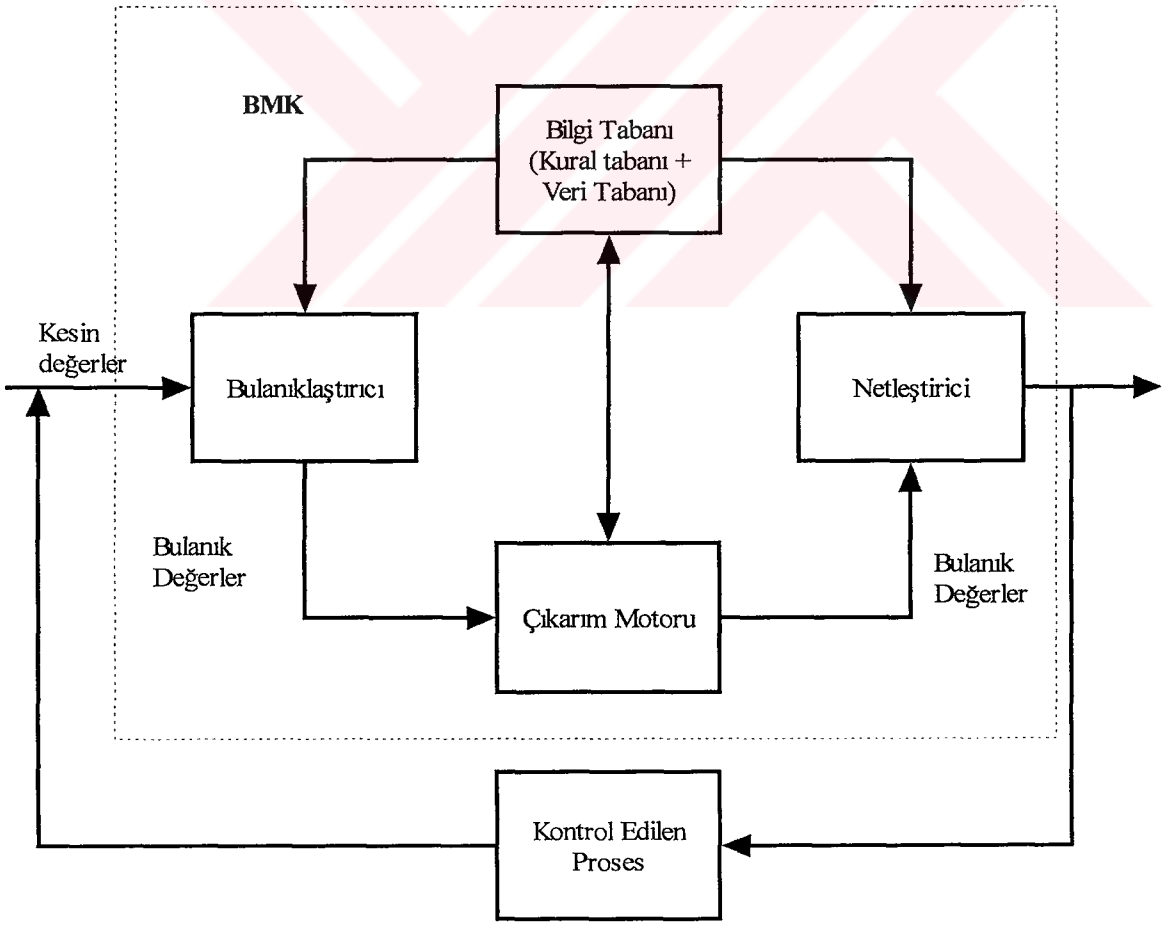
bulanık kontrolörlerin performanslarını en çok etkileyen giriş ölçekleme faktörleri için yapılmıştır. Bunun için öncelikle, Nöral-Genetik tabanlı optimal PI kontrolörü (NGTOPI) tasarlanmıştır. Daha sonra sırası ile Nöral-Genetik tabanlı optimal bulanık PI kontrolörü (NGTOFPI), Nöral-Genetik tabanlı optimal bulanık PI – bulanık PD tipi bulanık PID (NGTOFPID) kontrolörü, Nöral-Genetik tabanlı optimal öz uyarlamalı bulanık PI (NGTOSTFPI) ve Nöral-Genetik tabanlı optimal öz uyarlamalı bulanık P-ID kontrolörü (NGTOSTFP-ID) tasarlanmıştır. NGTOSTFP-ID kontrolörü dışındaki tüm kontrolörlerin kural tabanları hata faz düzlemi yöntemi ile optimize edilmiştir. NGTOSTFP-ID kontrolörünün kural tabanı ise diğer bulanık kontrolörlerden farklı olarak gradyen düşüm tekniğine dayanan optimizasyon yöntemi ile belirlenmiştir. Tezde yukarıda bahsi geçen çalışmalardan farklı bir yenilik olarak; genetik optimizasyon işlemi, sistemin matematiksel modeli kullanılmadan; bir Yapay Sinir Ağı (YSA) modeli üzerinden gerçekleştirilmiştir. Ayrıca her bir kontrolör mekanizması, C++ dili ile programlanarak ve gerçek zamanda bir PC aracılığı ile Engel firmasına ait doğru akım servomotor düzeneği (52V, 2.2A, 92W, 3000d/dk'lık 3 adet motordan oluşan düzenek) üzerinde test edilmiştir. Sistemi yüklemek amacı ile lineer olmayan bir yük sistemi kullanılmıştır. Mevcut kontrolör mekanizmalarını birbirleri ile karşılaştırmak için; *IAE* (mutlak hatanın integrali), *ITAE* (mutlak hatanın zaman ağırlıklı integrali) ve mutlak aşım (*%Aşım*) kriterleri performans ölçütü olarak alınmıştır. Bu çalışmada sunulan bulanık kontrolör tasarım stratejisi; tamamı ile sistemin matematiksel modelinden bağımsız, sistematik bir yöntemdir.

Tez çalışmasının geri kalan kısmı ise şu şekildedir: İkinci bölümde, yapay zeka uygulamalarının önemli kısımlarını oluşturan bulanık mantık, genetik algoritmalar ve yapay sinir ağlarına kısaca değinilmiştir. Üçüncü bölümde, tez çalışmasının ana hedefini oluşturan optimal bulanık kontrolörlerin tasarımı üzerinde durulmuştur. Bunun için öncelikle endüstride sıkça kullanılan PI tipi kontrolör tasarlanmıştır. Ancak literatürden farklı olarak, optimal PI katsayıları Neuro-Genetik bir işlem ile taranmıştır. Daha sonra sırası ile NGTOFPI, NGTOF-PID, NGTOSTFPI ve NGTOSTFP-ID kontrolörlerinin tasarımları ele alınmış, son bölümde ise, yukarıda adı geçen kontrolörler gerçek zamanlı olarak motor düzeneği üzerinde test edilerek performansları karşılaştırılmış ve optimal bulanık kontrolör belirlenmiştir.

2. BULANIK MANTIK KONTROL, GENETİK ALGORİTMALAR, YAPAY SİNİR AĞLARI

2.1 Bulanık Mantık Kontrol

Bulanık Mantık Kontrol (BMK), bulanık mantık temellerine dayanan, kesin değerler yerine yaklaşık değerler üzerinde duran ve genellikle uzman kişinin görüşlerine dayanarak çalışan, kontrol ettiği sistemin matematiksel modelinden bağımsız bir kontrol yaklaşımıdır. Yapısı Şekil 2.1'den de görülebileceği gibi dört ana kısımdan oluşur. Bunlar sırası ile, gerçek dünyanın fiziksel değerlerini bulanık uzayına taşıyan, fiziksel değerlere [0,1] arası bulanık değerler atayan bulanıklaştırıcı (bu çalışmada üçgensel tipte kümeler kullanılmıştır); uzman kişinin bilgi ve tecrübeleri doğrultusunda hazırlanan bilgi tabanı; bulanık giriş değerlerini kural tabanına göre yorumlayan çıkarım motoru ve son olarak BMK'nın çıkışında elde edilen bulanık değerleri fiziksel dünyada kullanılabilecek forma çeviren netleştiricidir.



Şekil 2.1 Bulanık Mantık Kontrol sisteminin genel yapısı

2.1.1 Bulanık Kümeler ve Terminoloji

Tanım 1: Bulanık Küme: Bir bulanık küme F; bahsi geçen uzay U da, [0,1] aralığında değerler alabilen μ_F üyelik fonksiyonu ile karakterize edilir. $\mu_F : U \rightarrow [0,1]$. Bir bulanık küme, yalnızca {0,1} değerlerini alabilen klasik bir kümenin genelleştirilmiş halidir. Bundan dolayı U uzayında tanımlı bir F bulanık kümesi,

$$F = \{(u, \mu_F(u)) \mid u \in U\} \tag{2.1}$$

şeklinde tanımlanabilir. Eğer U sürekli bir uzay ise, bu uzayda tanımlı F bulanık kümesi;

$$F = \int_U \mu_F(u)/u \tag{2.2}$$

şeklinde ifade edilir. U uzayının ayrık olduğu durumlarda ise;

$$F = \sum_{i=1}^{n} \mu_F(u_i)/u_i \tag{2.3}$$

şeklinde ifade edilir.

Tanım 2: Destek (Support): U uzayında tanımlı bir bulanık F kümesinin desteği, üyelik değerleri sıfırdan farklı olan U uzayının ayrık elemanlarıdır ve

$$supp\ (F) = \{\ u \in U \mid \mu_F(u) > 0\} \tag{2.4}$$

şeklinde gösterilir.

Tanım 3: Geçiş Noktası (Crossover Point): Bir bulanık kümenin geçiş noktası; üyelik değeri 0.5 olan U tanım uzayındaki veridir.

Tanım 4: Yükseklik (Height): Bir bulanık kümeye ait üyelik fonksiyonunun aldığı en büyük değer, o kümenin yüksekliği olarak adlandırılır.

Tanım 5: α - Kesiti (α - Cut): Bir bulanık kümenin α kesiti; üyelik değerleri α dan büyük veya eşit, U tanım bölgesinin tüm elemanlarından oluşan kümedir ve

$$F_\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\} \tag{2.5}$$

şeklinde ifade edilir.

2.1.2 Bulanık Kümeler Üzerinde İşlemler

Eşitlik (Equality): A ve B aynı tanım uzayı, U da tanımlı iki bulanık küme olsun. $A = B \Leftrightarrow \mu_A(u) = \mu_B(u)\ \forall\ u \in U$.

Ters (Complement): U uzayında tanımlı bir A bulanık kümesinin tersi,

$$\mu_{\overline{A}}(u) = 1 - \mu_A(u) \tag{2.6}$$

şeklinde ifade edilir.

Birleşim (Union): A ve B, U uzayında tanımlı bulanık kümeler olsun. Bu kümelerin bileşimi,

$$\mu_{A \cup B}(u) = \max[\mu_A(u), \mu_B(u)] \tag{2.7}$$

şeklinde ifade edilir.

Kesişim (Intersection): U uzayında tanımlı A ve B bulanık kümelerinin kesişimi,

$$\mu_{A \cap B}(u) = \min[\mu_A(u), \mu_B(u)] \tag{2.8}$$

şeklinde ifade edilir.

2.1.3 Bulanık Kümelerde Bağıntılar (Relations)

Bir bulanık bağıntı, $U_1, U_2, \ldots, U_n$ kesin kümelerinin kartezyen çarpımından oluşan uzay üzerinde tanımlı kümedir. O halde, $U_1 \times U_2 \times \ldots \times U_n$ uzayı üzerinde tanımlı bir Q bağıntısı;

$$Q = \{((u_1, u_2, \ldots, u_n), \mu_Q(u_1, u_2, \ldots, u_n)) \mid (u_1, u_2, \ldots, u_n) \in U_1 \times U_2 \times \ldots \times U_n\} \tag{2.9}$$

şeklinde temsil edilebilir. Burada, μ_Q: $U_1 \times U_2 \times \ldots \times U_n \rightarrow [0,1]$ şeklindedir.

2.1.3.1 Bulanık Bağıntıların Kompozisyonu

$P(U,V)$ ve $Q(V,W)$ ortak bir V kümesini kullanan ikilik (binary) bağıntılar olsun. P ve Q nun kompozisyonu $P \circ Q$ şeklinde gösterilir ve $U \times W$ uzayında tanımlıdır.

$(x,z) \in P \circ Q \Leftrightarrow \exists\ y \in V \mid (x, y) \in P$ ve $(y, z) \in Q$. Aynı ifade bulanık kümelerle;

$$\mu_{P \circ Q}(x,z) = \max_{y \in V} \min[\mu_P(x,y), \mu_Q(y,z)] \tag{2.10}$$

şeklinde ifade edilebilir. Eğer max-çarpım kompozisyonu kullanılırsa,

$$\mu_{P\circ Q}(x,z) = \max_{y\in V}[\mu_P(x,y)\mu_Q(y,z)] \quad (x,z) \in U \text{ x } W \tag{2.11}$$

şeklinde ifade edilir.

2.1.4 Dilsel Değişkenler ve Bulanık IF – THEN Kuralları

Günlük hayatta sıklıkla değişkenleri tanımlamak için kelimeler kullanılır. Örneğin, "Bugün hava sıcak" veya eşdeğer olarak, "Bugün sıcaklık çok yüksek" cümlelerinde yüksek kelimesi, günün sıcaklığına bir değer atamaktadır. Burada günün sıcaklığı bir değişken; buna karşılık, yüksek kelimesi bu değişkenin bir değeridir. Tanım olarak, eğer bir değişken dilsel olarak bir değer alıyor ise bu **dilsel değişken**'dir.

Bir dilsel değişken (*X, T, U, M*) ile tanımlanır. Burada:

- *X* dilsel değişkenin ismidir. Örneğin *X*, arabanın hızı olabilir.
- *T*, *X* in alabildiği değerlerin oluşturduğu kümedir. Örneğin, *T* = {yavaş, orta hızlı, hızlı, çok hızlı}
- *U*, *X* in fiziksel dünyada tanımlı olduğu aralığı işaret eder. Örneğin, $U = \{0, V_{max}\}$. Burada, V_{max} arabanın ulaşabileceği maksimum fiziksel hızdır.
- *M*, *T* deki her dilsel değeri *U* daki bir bulanık kümeye karşı düşüren kuraldır. Örneğin yavaş, orta hızlı, çok hızlı gibi değişik gruplara ait öz üyelik fonksiyonlarıdır.

2.1.4.1 Bulanık Önermeler

Bulanık sistemlerde uzman kişinin bilgisi *IF – THEN* kuralları ile tanımlanır. Yapısı kabaca,

IF < bulanık önerme >, THEN < bulanık önerme > (2.12)

şeklindedir. Bulanık önermeler;

x is A

x is M veya x is not S şeklindeki yapılardır.

Bu türden önerme yapılarındaki "*ve*" bağlaçları için bulanık kesişim işlemi kullanılacaktır. Sırası ile *x* ve *y*, *U* ve *V* fiziksel uzaylarındaki değişkenler olsunlar. Ayrıca *A* ve *B*, *U* ve *V* uzaylarında tanımlı bulanık kümeler ise;

x is A ve y is B

şeklindeki bir birleşik önerme U x V uzayında tanımlı $A \cap B$ bulanık bağıntısı ile

$$\mu_{A\cap B}(x,y) = t[\mu_A(x), \mu_B(x)] \tag{2.13}$$

şeklinde tanımlanır. Burada t: [0,1] x [0,1] → [0,1] şeklindedir ve t herhangi bir kesişim işlemini ifade eder. Bu çalışmada kesişim işlemi için *min* operatörü kullanılmıştır.

veya bağlacı için bulanık bileşim operatörünü kullanılacaktır. Örneğin;

x is A veya y is B

şeklindeki bir bulanık önerme U x V uzayında

$$\mu_{A\cup B}(x,y) = s[\mu_A(x), \mu_B(x)] \tag{2.14}$$

şeklinde tanımlanır. Burada s:[0,1] x [0,1] → [0,1] şeklindedir ve s herhangi bir bileşim işlemini ifade eder. Bu çalışmada bileşim işlemleri için *max* operatörü kullanılmıştır.

2.1.4.2 IF – THEN Kurallarının Gerçeklenmesi

Klasik mantıkta, *IF p THEN q* işlemi, $p \rightarrow q$ şeklinde gösterilir. Burada p ve q sırası ile; doğru ve yanlış türünden değerler alabilen önermelerdir. Klasik mantıkta $p \rightarrow q$ şeklindeki bir işlemin eşdeğeri, $\overline{p} \vee q$ olarak tanımlanır. Ayrıca aynı işlem, $(p \wedge q) \vee \overline{p}$ şeklinde de tanımlanabilir. Klasik mantıktaki bu işlemden yola çıkarak, bulanık mantıkta da benzer gerçeklemeler oluşturmak mümkündür. Bulanık kümelerde birden fazla türden kesişim, bileşim ve ters alma operatörü bulunduğu için birçok *IF THEN* gerçeklemeleri ortaya atılmıştır. Bunlardan bazıları şunlardır: FP_1 ve FP_2 bulanık önermeler olmak üzere *IF* < FP_1 > *THEN* <FP_2 > türünden bir kural için;

Dienes-Rescher Gerçeklemesi :

$$\mu_{Q_D}(x,y) = \max[1 - \mu_{FP_1}(x), \mu_{FP_2}(y)] \tag{2.15}$$

Burada, Q_D, U x V uzayında tanımlı bir bağıntıdır.

Lukasiewicz Gerçeklemesi:

$$\mu_{Q_L}(x,y) = \min[1, 1 - \mu_{FP_1}(x) + \mu_{FP_2}(y)] \tag{2.16}$$

Burada Q_D, U x V uzayında tanımlı bir bağıntıdır.

Zadeh Gerçeklemesi:

$$\mu_{Q_Z}(x,y) = \max[\min(\mu_{FP_1}(x), \mu_{FP_2}(y)), 1 - \mu_{FP_1}(x)] \tag{2.17}$$

Gödel Gerçeklemesi:

$$\mu_{Q_G}(x,y) = \begin{cases} 1 & if\ \mu_{FP_1}(x) \le \mu_{FP_2}(y) \\ \mu_{FP_2}(y) & \text{diger durumlar} \end{cases} \tag{2.18}$$

Mamdani Gerçeklemesi:

$$\mu_{Q_{MM}}(x,y) = \min[\mu_{FP_1}(x), \mu_{FP_2}(y)] \tag{2.19}$$

veya,

$$\mu_{Q_{MP}}(x,y) = \mu_{FP_1}(x)\mu_{FP_2}(y) \tag{2.20}$$

şeklindedir. Mamdani gerçeklemeleri, bulanık kontrol sistemlerinde sıkça kullanılan bir yöntemdir.

2.1.5 Bulanık Mantık ve Yaklaşık Nedensellik

Klasik mantıkta üç çeşit çıkarım kuralı bulunur. Bunlar sırası ile Modus Ponens, Modus Tollens ve Varsayımsal Kıyas (Hypothetical Syllogism) olarak adlandırılırlar.

2.1.5.1 Modus Ponens

Bu çıkarım kuralına göre, p ve $p \rightarrow q$ şeklinde koşullar verildiğinde q çıkışının doğruluğu belirlenebilir. Sembolik olarak;

$$(p \wedge (p \rightarrow q)) \rightarrow q \tag{2.21}$$

şeklinde gösterilir. Diğer bir ifade ile:

Koşul 1:*x is A*

Koşul 2:IF x is A THEN y is B (2.22)

Sonuç:y is B.

2.1.5.2 Modus Tollens

Bu çıkarım kuralına göre, $\overline{q}$ ve $p \rightarrow q$ şeklinde koşullar verildiğinde $\overline{p}$ girişinin doğruluğu belirlenebilir. Sembolik olarak;

$$(\overline{q} \wedge (p \rightarrow q)) \rightarrow \overline{p} \tag{2.23}$$

şeklinde gösterilir. Diğer bir ifade ile:

Koşul 1: *y is not B*

Koşul 2: *IF x is A THEN y is B* (2.24)

Sonuç: *x is not A.*

2.1.5.3 Varsayımsal Karşılaştırma (Hypothetical Syllogism)

Bu çıkarım kuralına göre, $p \rightarrow q$ ve $q \rightarrow r$ şeklinde iki önerme verildiğinde, $p \rightarrow r$ önermesinin doğruluğu çıkarılabilir. Sembolik olarak;

$$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r) \tag{2.25}$$

şeklinde gösterilebilir. Diğer bir ifade ile,

Koşul 1: *IF x is A THEN y is B*

Koşul 2: *IF y is B THEN z is C* (2.26)

Sonuç: *IF x is A THEN z is C.*

2.1.5.4 Genelleştirilmiş Modus Ponens

Klasik mantıktaki benzerinden esinlenerek ortaya atılmış çıkarım kuralıdır. Bir A' bulanık kümesi (*x is* A' koşulu) verildiğinde ve U x V de tanımlı bir $A \rightarrow B$ (*IF x is A THEN y is B*) koşulu verildiğinde, V uzayında tanımlı bir B' (*y is* B') bulanık kümesi çıkarılabilir.

$$\mu_{B'}(y) = \sup_{x \in U} t[\mu_{A'}(x), \mu_{A \rightarrow B}(x, y)] \tag{2.27}$$

Burada *t*, herhangi bir kesişim yöntemi olabilir.

2.1.5.5 Genelleştirilmiş Modus Tollens

Klasik mantıktaki benzerinden esinlenerek ortaya atılmış çıkarım kuralıdır. Bir B' bulanık kümesi (y *is* B' koşulu) verildiğinde ve U x V de tanımlı bir $A \rightarrow B$ (*IF* x *is* A *THEN* y *is* B) koşulu verildiğinde U uzayında tanımlı bir A' (x *is* A') bulanık kümesi çıkarılabilir.

$$\mu_{A'}(y) = \sup_{y \in V} t[\mu_{B'}(y), \mu_{A \rightarrow B}(x, y)] \tag{2.28}$$

Burada t herhangi bir kesişim yöntemi olabilir.

2.1.5.6 Genelleştirilmiş Varsayımsal Karşılaştırma

U x V uzayında tanımlı $A \rightarrow B$ bulanık bağıntısı (*IF* x *is* A *THEN* y *is* B) verildiğinde ve V x W uzayında tanımlı $B' \rightarrow C$ bulanık bağıntısı (*IF* y *is* B' *THEN* z *is* C) verildiğinde, U x W uzayında tanımlı $A \rightarrow C'$ bulanık bağıntısı (*IF* x *is* A *THEN* z *is* C') çıkarılabilir.

$$\mu_{A \rightarrow C'}(x, z) = \sup_{y \in V} t[\mu_{A \rightarrow B}(x, y), \mu_{B' \rightarrow C}(y, z)] \tag{2.29}$$

Burada t herhangi bir kesişim yöntemi olabilir.

Bir bulanık kural tabanlı sistemde birden çok *IF-THEN* şeklinde kurallar bulunur. Bu kurallar bulanık sistemin kalbi niteliğindedir. Genellikle kurallar şu şekildedir:

$$Ru^{(l)} : IF\ x_1\ is\ A_1^l\ and \ldots and\ x_n\ is\ A_n^l,\ THEN\ y\ is\ B^l. \tag{2.30}$$

Burada A_1^l ve B^l , sırası ile $U \subset \Re$ ve $V \subset \Re$ uzaylarının birer bulanık kümeleridir. $\mathbf{X} = (x_1, x_2, \ldots, x_n)^T \in U$ ve $y \in V$ sırası ile, bulanık sistem girişlerine ait vektör ve sistem çıkışını temsil eder. $l = 1 \ldots M$ kural tabanındaki kuralları ifade eder. Genellikle bulanık sistemler çok girişli - tek çıkışlı sistemler olarak bilinir. Zira, çok girişli - çok çıkışlı her sistem, çok girişli – tek çıkışlı sistemlerin topluluğu olarak da ifade edilebilir.

2.1.6 Bulanık Çıkarım Motorları

Bir bulanık çıkarım motorunun görevi; kural tabanındaki *IF – THEN* tipindeki kuralları birleştirerek, U uzayında tanımlı bir A' bulanık kümesinden, V uzayında tanımlı bir B' kümesine geçiş oluşturmaktır. Bundan önce bir çok *IF- THEN* kuralının nasıl bulanık olarak icra edildiğini açıklamıştık. Eğer kural tabanında sadece bir adet kural olsaydı, genelleştirilmiş modus ponens yöntemi ile U uzayında tanımlı bir A' bulanık kümesinden, V

uzayında tanımlı bir B' kümesine geçiş sağlanabilirdi. Fakat kural tabanında birden fazla kural bulunduğu zaman bu geçişi sağlamak için kuralların birleştirilmeleri ve adeta tek bir kural olarak çalıştırılmaları gerekmektedir. Bunun icrası için iki farklı yöntem ortaya atılmıştır. Bunlardan birincisi kompozisyon tabanlı çıkarım (composition based inference), ikincisi ise bireysel kural tabanlı çıkarımdır (individual – rule based inference).

2.1.6.1 Kompozisyon Tabanlı Çıkarım

Kompozisyon tabanlı çıkarım yönteminde, kural tabanındaki tüm kurallar birleştirilerek, U x V uzayında tek bir bağıntı gibi ele alınır. Bunu gerçeklemek için iki alt yöntem belirlenmiştir. Birinci yöntem, kural tabanındaki tüm kuralların bağımsız olduğu ve birbirleri ile ilişki içinde bulunmadığı varsayımı ile ortaya atılmıştır. Bu nedenle bu yöntemde kurallar, tek bir kural şekline indirgenirken *bileşim* işlemi uygulanır. Bu yöntem Mamdani tarafından ortaya atıldığından, Mamdani çıkarım yöntemi olarak bilinir.

$$Q_M = \bigcup_{l=1}^{M} Ru^{(l)} \tag{2.31}$$

Eğer $\dot{+}$ herhangi bir bileşim ifadesi olarak kabul edilirse;

$$\mu_{Q_M}(\mathbf{x}, y) = \mu_{Ru^{(1)}} \dot{+} \mu_{Ru^{(2)}} \dot{+} \ldots \dot{+} \mu_{Ru^{(M)}}(\mathbf{x}, y) \tag{2.32}$$

yazılabilir. Benzer şekilde, $*$ herhangi bir kesişim işlemini ifade ederse;

$$\mu_{Q_G}(\mathbf{x}, y) = \mu_{Ru^{(1)}} * \mu_{Ru^{(2)}} * \ldots * \mu_{Ru^{(M)}}(\mathbf{x}, y) \tag{2.33}$$

yazılabilir. Burada M, kural tabanındaki toplam kural sayısıdır. Diğer bir yöntem Gödel tarafından ortaya atılmıştır ve bu yaklaşıma göre kural tabanını oluşturan tüm kurallar birbirleri ile çok kuvvetli bağlar içerir. Bu yaklaşıma göre, kuralların her birinin diğeri üzerinde etkisi olacağı için, ara-kesitlerinden bahsedilmelidir. Buna göre;

$$Q_G = \bigcap_{l=1}^{M} Ru^{(l)} \tag{2.34}$$

tanımı geçerlidir. O halde, n girişli – tek çıkışlı bir sistem için Mamdani kombinasyonu dikkate alınırsa,

$$\mu_{B'}(y) = \sup_{\mathbf{x} \in U} t[\mu_{A'}(\mathbf{x}), \mu_{Q_M}(\mathbf{x}, y)] \tag{2.35}$$

bulanık çıkış kümesi; Gödel kombinasyonu ele alınırsa,

$$\mu_{B'}(y) = \sup_{\mathbf{x}\in U} t[\mu_{A'}(\mathbf{x}), \mu_{Q_G}(\mathbf{x}, y)] \tag{2.36}$$

bulanık çıkış kümesi elde edilir.

2.1.6.2 Bireysel Kural Tabanlı Çıkarım

Bireysel kural tabanlı çıkarım yönteminde, kural tabanını oluşturan her bir kural, bir çıkış bulanık kümesi belirler ve tüm çıkarım motorunun çıkışı ise M adet bireysel bulanık kümesinin kombinasyonu olarak tanımlanır. Yani U uzayında tanımlı bir A' kümesi çıkarım motorunun giriş kümesi ise; her bir kural $Ru^{(l)}$ için, V uzayında tanımlı B_l' çıkış bulanık kümesi,

$$\mu_{B_l'}(y) = \sup_{\mathbf{x}\in U} t[\mu_{A'}(\mathbf{x}), \mu_{Ru^{(l)}}(\mathbf{x}, y)] \qquad l = 1,2,...,M \tag{2.37}$$

şeklinde tanımlanır. Tüm bulanık çıkarım motorunun çıkışı M adet bulanık kümenin $\{B_1', ..., B_M'\}$ birleşimi olarak;

$$\mu_{B'}(y) = \mu_{B_1'}(y) \dot{+} \cdots \dot{+} \mu_{B_M'}(y) \tag{2.38}$$

şeklinde ifade edilebilir. Buna karşılık M adet bulanık kümenin kesişimi olarak,

$$\mu_{B'}(y) = \mu_{B_1'}(y) * \cdots * \mu_{B_M'}(y) \tag{2.39}$$

şeklinde ifade edilebilir.

2.1.7 Bazı Çıkarım Motorları

Daha önceki kısımlarda da belirtildiği gibi bulanık çıkarım motorlarının değişik tipleri mevcuttur. Örneğin, kompozisyon tabanlı çıkarım, bireysel kural tabanlı çıkarım, Mamdani çıkarımı, Gödel çıkarımı gibi çeşitli gerçeklemeler yapmak mümkündür. Bunlardan bazıları, Dienes-Rescher gerçeklemesi, Lukasiewicz gerçeklemesi, Zadeh gerçeklemesi, Gödel gerçeklemesi ve Mamdani gerçeklemesidir. Ayrıca kesişim ve birleşim operatörleri için de çok sayıda seçenek literatürlerde sunulmaktadır. (Lee, 1990), (Wang, 1997), (Klir ve Yuan, 1995).

2.1.7.1 Çarpım Tipi Çıkarım Motoru

Bu tipdeki çıkarım motorunda bireysel kural tabanlı çıkarım yöntemi, Mamdani'nin çarpım tipindeki çıkarımı, tüm kesişim operatörleri için cebirsel çarpım ve tüm bileşim operatörleri için *max* kullanılır. Eğer, A', U uzayında tanımlı bir bulanık küme ise ve B', V uzayında tanımlı bir bulanık küme ise,

$$\mu_{B'}(y) = \max_{l=1}^{M}[\sup_{\mathrm{x}\in U}(\mu_{A'}(\mathrm{x})\prod_{i=1}^{n}\mu_{A_i^l}(x_i)\mu_{B^l}(y))] \tag{2.40}$$

bağıntısı ile tanımlanır.

2.1.7.2 Minimum Tipi Çıkarım Motoru

Minimum tipindeki çıkarım motorunda Mamdani'nin minimum çıkarım metodu; bütün kesişim işlemleri için *min* operatörü, tüm bileşim işlemleri için *max* operatörü kullanılır. Eğer, A', U uzayında tanımlı bir bulanık küme ise ve B', V uzayında tanımlı bir bulanık küme ise,

$$\mu_{B'}(y) = \max_{l=1}^{M}[\sup_{\mathbf{x}\in U}\min(\mu_{A'}(\mathbf{x}),\mu_{A_1^l}(x_1),\ldots,\mu_{A_n^l}(x_n),\mu_{B^l}(y))] \tag{2.41}$$

bağıntısı ile tanımlanır.

Çarpım tipi çıkarım motoru ve minimum tipi çıkarım motoru, bulanık sistemlerde en çok kullanılan tiplerdir. Özellikle çarpım tipi çıkarım motoru, kolay hesaplanabildiği için sık tercih edilir. Ancak, bu iki yöntemin de dezavantajı; küçük $x \in U$ için, küçük $\mu_{A_i^l}(x_i)$ değerleri oluşturması ve dolayısı ile çıkış üyelik fonksiyonları olan $\mu_{B'}(y)$ lerin çok küçük değerler almasıdır. Bu dezavantajı ortadan kaldırmak amacı ile kimi zaman; **Lukasiewicz çıkarım motoru**, **Zadeh çıkarım motoru** veya **Dienes-Rescher çıkarım motoru** kullanılır.

2.1.7.3 Lukasiewicz Çıkarım Motoru

Bu tipteki bir çıkarım motorunda, bireysel kural tabanlı çıkarım ve kesişim tipinde kombinasyon işlemi, Lukasiewicz gerçeklemesi, tüm kesişim işlemleri için *min* operatörü kullanılır. Eğer, A', U uzayında tanımlı bir bulanık küme ise ve B', V uzayında tanımlı bir bulanık küme ise;

$$\mu_{B'}(y) = \min_{l=1}^{M}\{\sup_{\mathbf{x}\in U}\min[\mu_{A'}(\mathbf{x}),1-\min_{i=1}^{n}(\mu_{A_i^l}(x_i))+\mu_{B^l}(y)]\} \tag{2.42}$$

bağıntısı ile tanımlanır.

2.1.7.4 Zadeh Çıkarım Motoru

Bu tip bir çıkarım motorunda, kesişim tipinde kombinasyon içeren bireysel kural tabanlı çıkarım, Zadeh gerçeklemesi, tüm kesişim işlemleri için *min* operatörü kullanılır. Eğer, A', U uzayında tanımlı bir bulanık küme ise ve B', V uzayında tanımlı bir bulanık küme ise,

$$\mu_{B'}(y) = \min_{l=1}^{M}\{\sup_{\mathbf{x}\in U} \min[\mu_{A'}(\mathbf{x}), \max(\min(\mu_{A_1^l}(x_1),\ldots,\mu_{A_n^l}(x_n), \mu_{B^l}(y)), 1 - \min_{i=1}^{n}(\mu_{A_i^l}(x_i)))]\} \quad (2.43)$$

bağıntısı ile ifade edilir.

2.1.7.5 Dienes-Rescher Çıkarım Motoru

Bu tip bir çıkarım motorunda kullanılan yöntemler tamamı ile Zadeh'in çıkarım motorundaki yöntemlerin aynıdır. Tek fark ise, Zadeh'in gerçeklemesi yerine Dienes-Rescher gerçeklemesinin kullanılmasıdır. Eğer, A', U uzayında tanımlı bir bulanık küme ise ve B', V uzayında tanımlı bir bulanık küme ise;

$$\mu_{B'}(y) = \min_{l=1}^{M}\{\sup_{\mathbf{x}\in U} \min[\mu_{A'}(\mathbf{x}), \max(1 - \min_{i=1}^{n}(\mu_{A_i^l}(x_i)), \mu_{B'}(y))]\} \quad (2.44)$$

bağıntısı ile ifade edilir.

2.1.8 Netleştirici

Bulanıklaştırma işleminin tersini yerine getirir. Yapılan işlem, V uzayında tanımlı bir B' kümesinden, $y^* \in V$ keskin değerine iz düşüm oluşturmak ile eşdeğerdir. Diğer bir deyiş ile; V uzayında tanımlı B' kümesini en iyi sembolize eden çıkışı bulmaktır. En çok kullanılan netleştirme yöntemleri: Ağırlık merkezini bulma (center of gravity defuzzification), ortalama merkezi bulma (center average defuzzification) ve maksimum değer yöntemleridir. Bunlardan ağırlık merkezi yöntemi,

$$y^* = \frac{\int_V y\mu_{B'}(y)dy}{\int_V \mu_{B'}(y)dy} \quad (2.45)$$

şeklinde hesaplanır. Burada $y^* \in V$ değeri, B' üyelik fonksiyonu tarafından örtülen alanın

merkezidir. Hesaplanmasındaki güçlükler nedeni ile gerçek zamanlı sistemlerde tercih edilmeyen bir yöntemdir. Ortalama merkezini bulma yönteminde ise y^* çıkışı,

$$y^* = \frac{\sum_{l=1}^{M} \bar{y}^l \omega_l}{\sum_{l=1}^{M} \omega_l} \tag{2.46}$$

şeklinde hesaplanır. Burada, $\bar{y}^l$, l ninci bulanık kümenin merkezini; ω_l, yüksekliğini; M, kural tabanındaki toplam kural sayısını göstermektedir. Kolay hesaplanması ve gerçek sonuca çok yakın değerler üretmesi nedeni ile tercih edilen bir yöntemdir. Netleştirme yöntemleri arasında sık kullanılan diğer bir yöntem ise maksimum değer yöntemidir. Burada, maksimum netleştiricisi, $y^* \in V$ değerini (2.47) ye göre seçer.

$$hgt(B') = \{y \in V \mid \mu_{B'}(y) = \sup_{y \in V} \mu_{B'}(y)\}. \tag{2.47}$$

Eğer çıkış bulanık kümesinin birden çok maksimum değeri varsa, bu durumda ya rastgele bir değer maksimumlar arasından seçilir, ya da daha önceden belirlenen bir karar yapısı ile maksimumların en küçüğünü veya maksimumların en büyüğünü sağlayan y^* noktası çıkış olarak seçilir (Wang, 1997), (Lee, 1990).

2.2 Genetik Algoritmalar (Genetic Algorithms)

Genetik algoritmalar, doğal genetik ve doğal seçim yöntemlerine dayanan, basit, güçlü, genel amaçlı, cebirsel türev işleminden bağımsız, stokastik, global optimizasyon (tarama) algoritmalarıdır. Darwin'in evrim teorisinden (en güçlünün varlığını sürdürmesi) esinlenilerek ortaya atılmıştır. Bir amaç fonksiyonunu minimize veya maksimize ederken türev işlemini kullanmaması bu algoritmanın diğer optimizasyon algoritmalarına göre üstün tarafıdır. Ancak türev işlemine dayanılarak yapılan optimizasyon işlemlerine göre daha yavaştır. Bir genetik tabanlı tarama algoritmasında, her tasarım değişkeni sonlu uzunluğa sahip ikilik formda tanımlanır; buna **gen** adı verilir. Tüm değişkenler bu yapıda arka, arkaya sıralanarak tek bir ikilik diziyi oluşturur; buna **kromozom** adı verilir. Olası çözümler kodlanarak bir ikilik dizi popülasyonu elde edilir. Biyomantık üreme ve evrime benzer genetik dönüşümler peş peşe kullanılarak kodlanmış çözümler geliştirilir. Genetik algoritmaların dayandığı en önemli unsur, problemin çözümüne uygun bir maliyet (uyum) fonksiyonunun belirlenmesidir (Fitness Function). Genetik algoritmanın basamakları şunlardır: Rasgele ilk popülasyonun oluşturulması, popülasyonu oluşturan bireylerinin uyumlarının hesaplanması ve yeni

jenerasyonun oluşturulmasıdır. Yeni jenerasyonun oluşturulması işlemi de üç alt evreden oluşur. Bunlar: Reproduction (seçim), crossover (çaprazlama) ve mutation (mutasyon) işlemleridir (Vas, 1999).

2.2.1 İlk Popülasyonun Rastgele Oluşturulması

Genetik algoritmanın ilk adımında popülasyon, taramanın yapıldığı uzaydan seçilen rasgele N adet değerden oluşur. Yani çözüm olabilecek N adet kromozom ilk popülasyonu meydana getirir.

2.2.2 İlk Popülasyonun Uyumunun Hesaplanması

Rastgele oluşturulan popülasyonun tüm bireyleri (kromozomları) daha önceden belirlenen amaç fonksiyonundan geçirilerek uyumları (fitness values) hesaplanır. Amaç fonksiyonu, problemin tanımına göre minimize veya maksimize edilmesi istenen değişkenlerden oluşur.

2.2.3 Yeni Popülasyonun Oluşturulması

Darwin'in evrim teorisine dayanan biyomantık evrimsel proseste, üç ayrı işlem yapılır. Bunlar en kuvvetli ebeveyn kromozomların seçilmesi, seçilen kromozomların oluşturduğu havuzdaki bireylerin çaprazlanması ve genler üzerinde değişiklik yapılması (mutasyon) işlemleridir. Amaç bireyleri çok güçlü olan popülasyonlar yaratmaktır.

2.2.3.1 Seçim Aşaması

Genetik işlemlerden çaprazlama ve mutasyon işlemlerinin öncesinde yapılan işlemdir. Popülasyon içersinden N adet kromozom, bireylerin uyum değerlerine göre rasgele seçilir. Popülasyon içersinde en yüksek uyuma sahip kromozomların seçilme şansı daha yüksektir. Yani, yeni jenerasyonun toplam uyumunun, bir önceki jenerasyonun toplam uyumundan büyük olma olasılığı daha yüksektir. Seçim işlemi için genellikle **rulet tekerleği** adı verilen yöntem kullanılır. Bu yöntemde, N adet kromozoma sahip bir popülasyon içerisindeki i. kromozomun seçilme olasılığı,

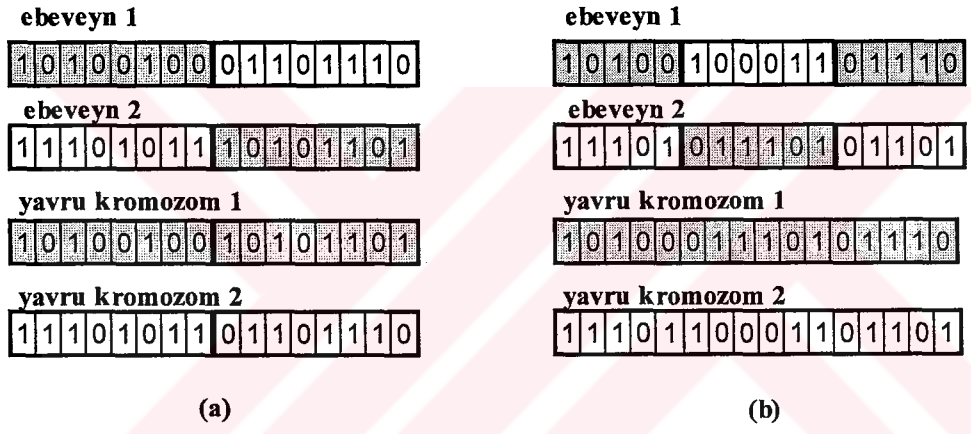
$$P_i = \frac{f_i}{\sum_{k=1}^{N} f_k} \qquad (k = 1,2,\ldots,N) \tag{2.48}$$

şeklinde hesaplanır. Burada f_i, i. kromozomun uyumudur. Seçim işlemini takiben yeni bir N

şeklinde hesaplanır. Burada f_i, i. kromozomun uyumudur. Seçim işlemini takiben yeni bir N elemanlı havuz oluşur. Buna **çiftleşme havuzu** (mating pool) adı verilir.

2.2.3.2 Çaprazlama Aşaması (Crossover Stage)

Seçim aşamasını takiben yeni oluşturulan çiftleşme havuzunda yapılan işlemdir. Havuzdaki elemanlardan seçilen belli sayıda kromozom, birbirleri ile çaprazlanır. Çaprazlama yapılacak kromozom sayısını, P_c çaprazlama oranı belirler. Çaprazlanacak elemanlar ise rasgele seçilir. Genellikle iki tip çaprazlama işlemi kullanılır. Birincisi **sabit noktalı çaprazlama**; diğeri **değişken noktalı** çaprazlamadır. Sabit noktalı çaprazlama işlemi de tek noktalı veya iki noktalı olarak kendi arasında ikiye ayrılır. Sabit noktalı çaprazlama yöntemi Şekil 2.2 de bir örnek üzerinde gösterilmiştir. Bu çalışmada tek sabit noktalı çaprazlama işlemi kullanılmıştır.



Şekil 2.2 Sabit noktalı çaprazlama işlemi (a) tek noktalı çaprazlama işlemi (b) çift noktalı çaprazlama işlemi

2.2.3.3 Mutasyon Aşaması (Mutation Stage)

Çaprazlama aşamasını takiben yapılan işlemdir. Kromozom dizilerinde bazı bitler üzerinde yapılan mantıksal ters alma işlemidir. Amaç, popülasyonu oluşturan kromozomlara yeni genetik nitelikler kazandırmaktır. Bu yeni nitelik ebeveyn kromozomlarda bulunmayan bir niteliktir. Optimal çözüm arayışı esnasında global minimumlar yerine lokal minimumlara yakalanma olasılığını azaltır. Çözüm uzayının tamamının taranmasında oldukça etkili bir yöntemdir. Bir jenerasyonda yapılması gereken mutasyon miktarı,

$$n_m = round(P_m * N * l) \tag{2.49}$$

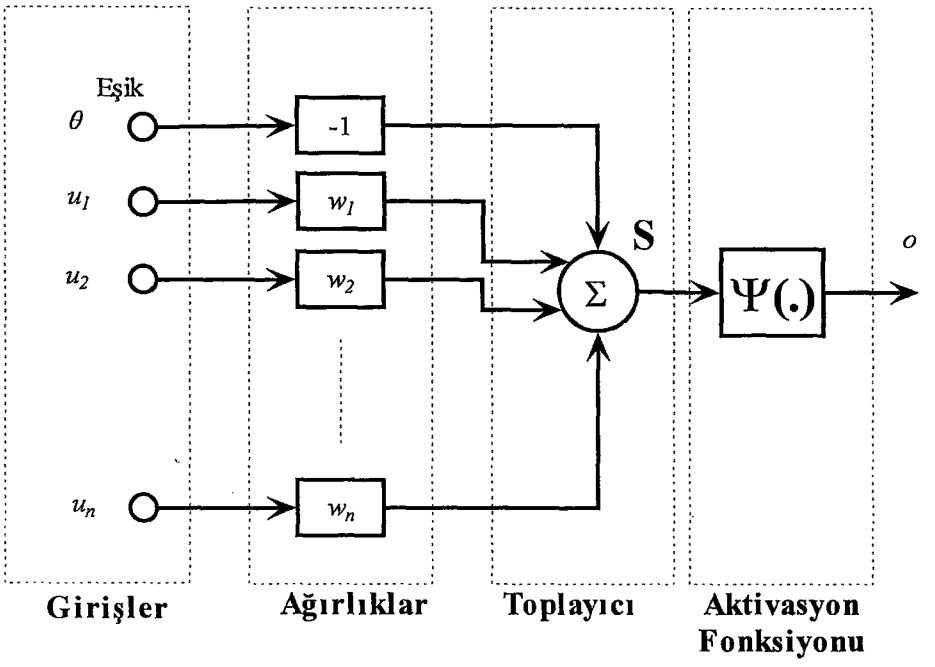
jenerasyondaki toplam kromozom sayısını ve l, bir kromozomdaki bit sayısını göstermektedir.

2.3 Yapay Sinir Ağları (Artificial Neural Networks)

Yapay sinir ağları kavramı beynin çalışma ilkelerinin sayısal bilgisayarlar üzerinde taklit edilmesi fikri ile ortaya çıkmış ve ilk çalışmalar beyni oluşturan biyolojik hücrelerin, yada literatürdeki ismi ile nöronların matematiksel modellenmesi üzerinde yoğunlaşmıştır. Bu çalışmaların ortaya çıkardığı bulgular, her bir nöronun komşu nöronlardan bazı bilgiler aldığı ve bu bilgilerin biyomantık nöron dinamiğinin öngördüğü biçimde bir çıktıya dönüştürüldüğü şeklinde idi. Bugün yapay sinir ağları olarak isimlendirilen alan, bir çok nöronun belirli şekillerde bir araya getirilip bir işlevin gerçeklenmesi üzerindeki yapısal olduğu kadar matematiksel ve felsefi sorunlara yanıt arayan bir bilim dalı olmuştur (Efe ve Kaynak, 2000).

Bir sistemin davranışının matematiksel olarak ifade edilmesi, fen ve mühendislik alanında çalışan kişiler için önemli bir nokta oluşturur. Ancak günümüzde gelişen teknoloji ile birlikte sistemlerin karmaşıklığı da artmıştır. Bu ise sistemlerin tam olarak cebirsel biçimde ifade edilmesini zorlaştırmaktadır. Bu noktada yapay sinir ağları veya öğrenen türdeki diğer yapılar imdadımıza yetişmektedir. Yapay sinir ağları, sistemin matematiksel modelinden bağımsız tanımlamalar yapabilmektedir. Tanıma ve denetim, endüstride sıkça karşılaşılan bir problemdir. Örneğin bir robot sisteminin davranışını karakterize eden denklem kümesi tüm ayrıntıları ile bilinmese dahi kaba bir model ve sistemden elde edilen nümerik verilerle sistem davranışı modellenebilmektedir. Benzer şekilde, bir robot sisteminin denetimi için de sistemin tanılanması sonucu elde edilen model kullanılarak bir kontrolör tasarlanabilmektedir.

Bir yapay sinir ağının en küçük birimi nörondur. Nöronlar, sinir ağlarını oluşturan, tek başlarına ele alındıklarında çok basit işleve sahip işlemcilerdir. Bir nöron yapısı içersinde üç ana bölüm bulunur. Bunlar sırası ile sinapslar, toplayıcı ve aktivasyon fonksiyonlarıdır. Şekil 2.3'de bir nöronun modeli gösterilmektedir. Bu şekilden de görülebileceği gibi, nöron girdileri sinaptik bağlantılar üzerindeki ağırlıklar ile çarpılarak bir toplayıcıya uygulanmakta ve elde edilen toplam, nöron aktivasyon fonksiyonundan geçirilerek çıkışlar hesaplanmaktadır. (2.50) denkleminde ağırlıklı toplamın oluşturulması, (2.51) denkleminde ise nöron çıkışının hesaplanması gösterilmektedir.



Şekil 2.3 Bir nöronun yapısı

$$S = \sum_{i=1}^{n} w_i u_i - \theta \tag{2.50}$$

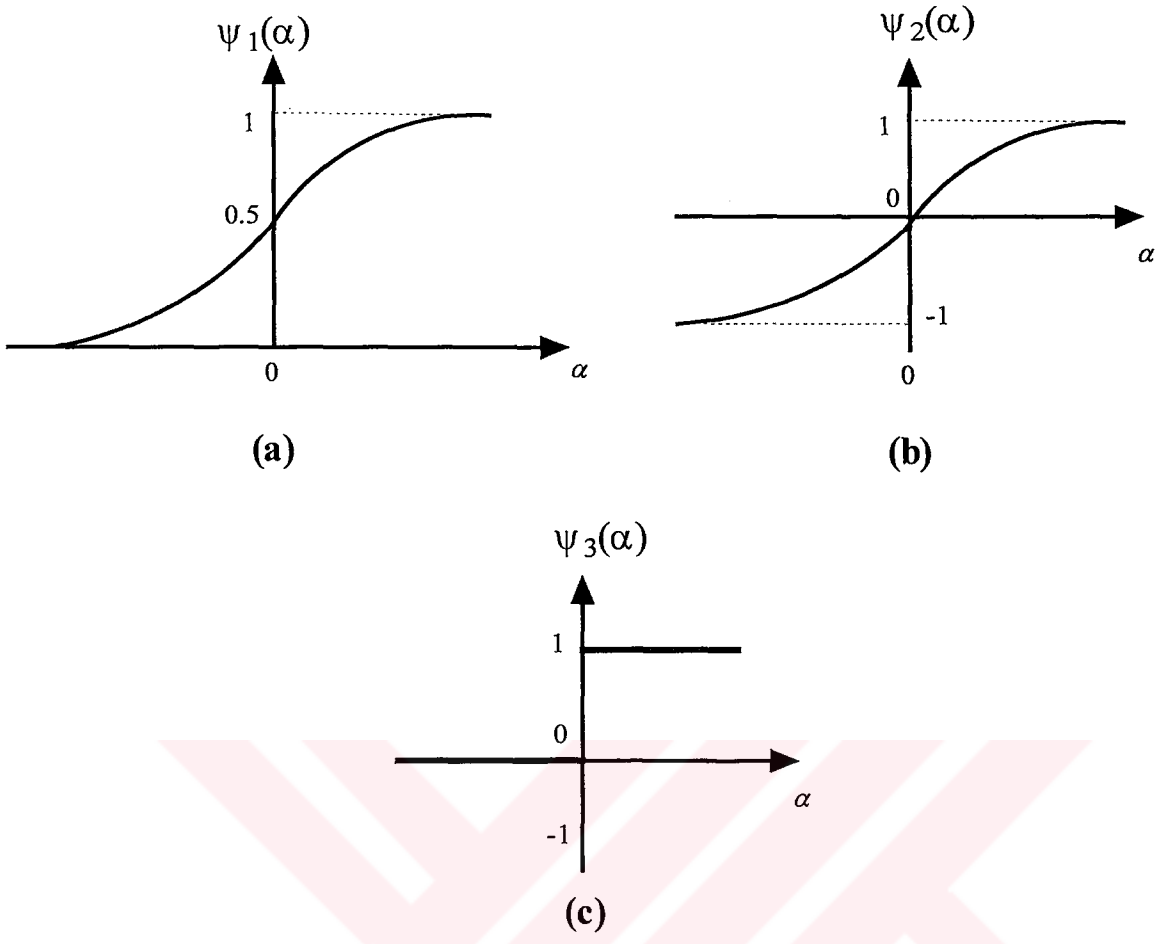
$$o = \Psi(S) \tag{2.51}$$

Her bir girdideki değişim, nöron çıkışında belirli bir değişime neden olmakta ve bu değişimin genliği, girdinin etki derecesini belirleyen bağlantı kazançlarına, toplayıcının eşik değerine ve aktivasyon fonksiyonunun tipine bağlı olmaktadır. Burada, w_i ile gösterilen kazançlar ağırlık olarak; θ değeri eşik olarak; Ψ fonksiyonu da, nöron aktivasyon fonksiyonu olarak isimlendirilmektedir.

Nöronun davranışını belirleyen önemli etmenlerden biri nöron aktivasyon fonksiyonudur. Biyolojik nöronlarda, S ile gösterilen toplam, belli bir değeri aştığında ilgili nöronun kısa süreli bir darbe ürettiği bilinmektedir. Bu davranışa benzer bir davranışı yapay nöronlarla da elde etmek için kullanılan aktivasyon fonksiyonlarından üçü Şekil 2.4'de gösterilmiş, matematiksel ifadeleri ise (2.52)-(2.54) denklemlerinde verilmiştir.

$$\Psi_1(\alpha) = \frac{1}{1+e^{-\alpha}} \quad \text{Sigmoid tipi aktivasyon fonksiyonu} \tag{2.52}$$

$$\Psi_2(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}} \quad \text{Hiperbolik tanjant tipi aktivasyon fonksiyonu} \tag{2.53}$$



Şekil 2.4 (a) Sigmoid tipi aktivasyon fonksiyonu (b) Hiperbolik tanjant tipi aktivasyon fonksiyonu (c) Sert geçişli aktivasyon fonksiyonu

$$\Psi_3(\alpha) = \begin{cases} 0 & S \le 0 \\ 1 & S > 0 \end{cases}$$ Sert geçişli tipte aktivasyon fonksiyonu (2.54)

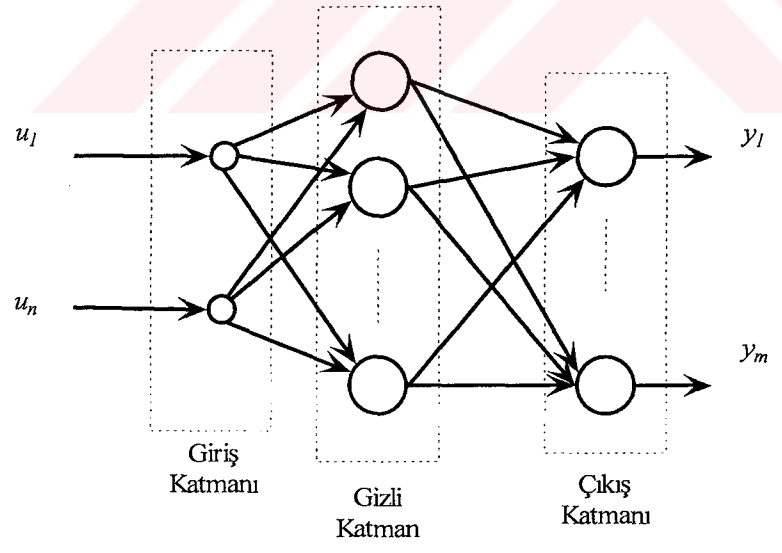
Uygulamalarda nöron cevabının, girdilerin sürekli bir fonksiyonu olmasını gerektiren durumlarda (2.52) ya da (2.53) denklemlerinde verilen aktivasyon fonksiyonları kullanılırken, ikili karar mekanizması gerektiren durumlarda sert geçişli aktivasyon fonksiyonları tercih edilir.

Ağ mimarisi olarak genellikle iki tip mimari karşımıza çıkar. Bunlar ileri beslemeli ağlar ve geri beslemeli ağlardır. Bu çalışmada ileri beslemeli ağlar kullanılacağı için, geri beslemeli ağlar üzerinde durulmayacaktır. İleri beslemeli ağların eğitiminde sıklıkla gradyen düşüm yöntemine dayanan hatanın geriye yayılımı (error back propagation) algoritması kullanılır. Bu algoritmada amaç, ele alınan bir hata performans ölçütünü her bir ağ değişkeni doğrultusunda minimize etmektir.

2.3.1 İleri Beslemeli Ağlar ve Hata Geriye Yayma Yöntemi ile Eğitim

İleri beslemeli sinir ağı yapılarının en tipik şekli bir önceki bölümde ele alınan nöron modeli ile oluşturulan katmanların ardışıl biçimde biraraya getirilmesi ile kurulabilir. Şekil 2.5'de 3 katmanlı n girişli, m çıkışlı ileri sürümlü bir ağ yapısı gösterilmiştir. Şekil 2.5'den de görüleceği üzere, girdilerin uygulandığı katmana giriş katmanı, çıkışların alındığı katmana çıkış katmanı adı verilir. Bu katmanlara fiziksel dünyadan erişilebilir. Giriş ve çıkış katmanlarının arasında gizli katmanlar bulunur. Bu katmanlardaki nöronların lineer olmayan davranışları, sinir ağının toplam davranışındaki lineersizliğindeki kaynağı teşkil ederler. Giriş / çıkış katmanlarındaki nöronların sayısı ele alınan problemin gerekliliğine göre belirlenir. Ancak gizli katman(lar)daki nöron sayısının optimallik anlamında doğru sayısını veren herhangi bir analitik yöntem şu ana kadar geliştirilememiştir. Dolayısı ile gizli katman sayısındaki ve bu katmanların nöron sayılarındaki belirsizlikleri aşabilmenin tek yolu deneme yanılma yöntemidir.

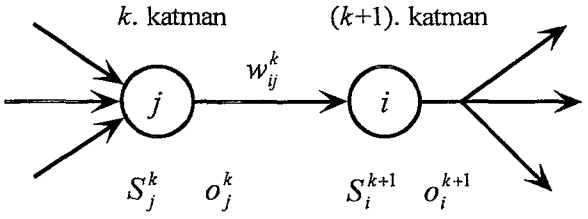
Şekil 2.5'de gösterilen ağ yapısı, geri besleme bağlantılarının olmaması dolayısı ile, veri akışı yönünden ileri beslemeli bir ağ yapısıdır. Bu yapıda giriş katmanı, giriş vektörünü gizli katmanlara ulaştırmakla yükümlüdür ve lineer olmayan bir davranışa sahip değildir. Dolayısı ile giriş katmanını oluşturan her bir nöronun çıkışında giriş değeri görülür.



Şekil 2.5 n girişli, m çıkışlı 1 gizli katmana sahip ileri beslemeli sinir ağı yapısı

Ağ üzerindeki katmanlar k indisi ile sıralansın, ve L adet gizli katman olduğu varsayılsın. Eğer k+1'inci katmanın i'inci nöronunu, k'ıncı katmanın j'inci nöronuna bağlayan bağlantıların ağırlık değerleri w_{ij}^{k} sembolü ile gösterilirse (Şekil 2.6) ve k'ıncı katmanın i'inci

nöronunun çıkışı o_i^k ile gösterilirse, k+1'inci katmandaki i'inci nöronun net toplamı S_i^{k+1} ve çıkış değeri o_i^{k+1}, (2.55) ve (2.56) denklemlerinden elde edilir. (2.55) denkleminde görülen n_k değişkeni, k'ıncı katmandaki nöron sayısını simgelemektedir.



Şekil 2.6 Ağ içerisinde katmanların ve nöronların sıralanması

$$S_i^{k+1} = \sum_{j=1}^{n_k} w_{ij}^k o_j^k \tag{2.55}$$

$$o_i^{k+1} = \Psi(S_i^{k+1}) \tag{2.56}$$

Yukarıdaki denklemlerde eşik değerleri ve aktivasyon fonksiyonlarının türü karmaşıklığı engellemek amacı ile gösterilmemiştir. Aktivasyon fonksiyonu olarak bir önceki bölümde tanıtılan fonksiyonlardan herhangi biri kullanılabilir. Önemli olan husus, bu fonksiyonların türevlenebilir olmasıdır. Aksi takdirde bu bölümde ele alınan ve aşağıda açıklanan hata geriye yayma yöntemi kullanılamaz.

Yapay sinir ağlarının parametrelerinin güncellenmesi için literatürde en çok kullanılan yöntem hata geriye yayma yöntemidir. Ses tanıma problemlerinden, lineer olmayan sistem tanıma ve denetimi problemlerine kadar yapay sinir ağları ile çözüm üretilen birçok alanda başarı ile kullanılan bu yöntem, kuadratik bir maliyet fonksiyonunun zaman içerisinde, ağ parametrelerinin uyarlanması ile minimizasyonuna dayanmaktadır. Temel felsefesi eğim düşümü yöntemine, yani (2.57) denklemi ile verilen tek parametreli bir (ϕ) maliyet fonksiyonunun en küçük değerini aldığı noktanın (2.58) bağıntısı ile verilen kural ile iteratif olarak bulunabilmesine dayalıdır.

$$J_r = \frac{1}{2}\phi^2, \tag{2.57}$$

$$\Delta\phi = -\eta \frac{\partial J_r}{\partial \phi}. \tag{2.58}$$

Burada önem kazanan bir nokta, η değişkeninin değeridir. Pratikte öğrenme katsayısı yada adım büyüklüğü olarak bilinen bu değer çok küçük ise, hata uzunca bir sürede sıfıra doğru yakınsarken, büyük bir değer sıfır etrafında salınımlara hatta ıraksamalara neden olabilmektedir. Katman ve nöron sayısının seçimi gibi, adım büyüklüğünün de seçimi bir çok uygulamada deneme yanılma yöntemi ile yapılır.

Hata geriye yayma yönteminin türetimi için (2.59) ile verilen maliyet fonksiyonu minimize edilmelidir. Bu amaçla, (2.60) bağıntısı ile verilen parametre güncelleme formülü benimsenecektir. (2.60) denkleminde ∇_w sembolü, w parametresine göre kısmi türevi göstermektedir.

$$J_r = \frac{1}{2}\sum_{i=1}^{n_{k+1}}(d_i - o_i)^2 \tag{2.59}$$

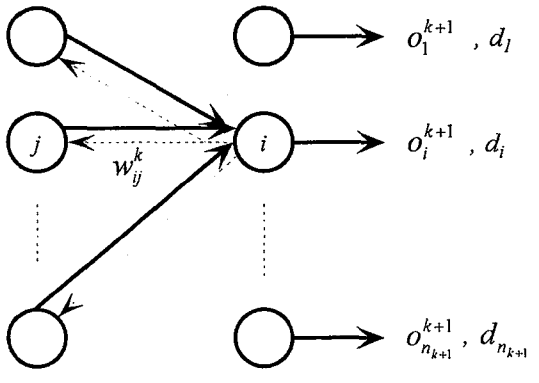
$$\Delta w = -\eta \nabla_w J_r \tag{2.60}$$

Yöntemin türetimi, çıkış katmanlarındaki nöronların parametreleri için farklı, gizli katmanlardaki nöronların parametreleri için farklı bir formülasyon ortaya çıkarır. Bu nedenle aşağıda bu iki durum birbirinden ayrı olarak ele alınmıştır. Türetim için Şekil 2.7 ile verilen çıkış katmanı göz önüne alınsın. Bu katmanın k+1'inci katman olduğu ve n_{k+1} adet nöron içerdiği varsayılsın. Türetim esnasında aşağıda verilen değişkenler kullanılacaktır:

J_r : Maliyet fonksiyonu,

d_i : Ağın i'inci çıkışı için istenen çıkış değeri,

o_i^{k+1} : k+1'inci katmanın i nöronunun çıkışında gözlenen değer,



Şekil 2.7 Çıkış katmanında hatanın geriye yayılması

w_{ij}^{k} : k+1'inci katmandaki i nöronu ile k'ıncı katmandaki j nöronunu birleştiren ağırlık,

S_i^{k+1} : k+1'inci katmanındaki i nöronunun girişinde oluşan net toplam.

(2.60) bağıntısında kullanılan türev, zincir kuralının uygulanmasıyla (2.61) denkleminde verildiği üzere üç çarpandan oluşacak şekilde açılabilir. Bu çarpanların açılımı sırası ile (2.62)-(2.64) denklemlerinde verilmiştir.

$$\frac{\partial J_r}{\partial w_{ij}^{k}} = \frac{\partial J_r}{\partial o_i^{k+1}} \frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} \frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}} \tag{2.61}$$

$$\frac{\partial J_r}{\partial o_i^{k+1}} = -(d_i - o_i^{k+1}) \tag{2.62}$$

$$\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} = \frac{d\Psi\left(S_i^{k+1}\right)}{dS_i^{k+1}} = \Psi'\left(S_i^{k+1}\right) \tag{2.63}$$

$$\frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}} = \frac{\partial}{\partial w_{ij}^{k}} \left[\sum_{j=1}^{n_k} w_{ij}^{k} o_j^{k} \right] = o_j^{k} \tag{2.64}$$

Eğer (2.65) ile verilen kısmi türev, delta değeri olarak tanımlanırsa, çıkış katmanındaki nöronlar için delta değerinin genel hali (2.66) denkleminde verilen biçimde, parametrelerin değişim miktarı ise (2.67) denkleminde verilen biçimde olacaktır.

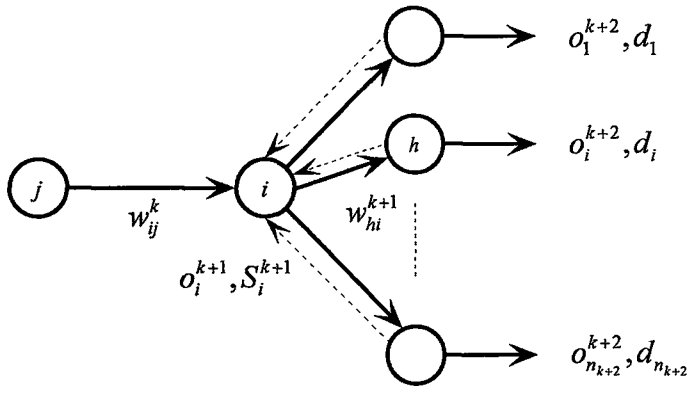
$$\delta_i^{k+1} = -\frac{\partial J_r}{\partial S_i^{k+1}} \tag{2.65}$$

$$\delta_i^{k+1} = \left(d_i - o_i^{k+1}\right)\Psi'\left(S_i^{k+1}\right) \tag{2.66}$$

$$\Delta w_{ij}^{k} = \eta \delta_i^{k+1} o_j^{k} \tag{2.67}$$

Gizli katmanlardaki nöronların parametrelerinin nasıl güncellenmesi gerektiğini gösterebilmek için (2.61) denklemini tekrar ele alalım.

$$\frac{\partial J_r}{\partial w_{ij}^{k}} = \frac{\partial J_r}{\partial o_i^{k+1}} \frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} \frac{\partial S_i^{k+1}}{\partial w_{ij}^{k}} \tag{2.68}$$



Şekil 2.8 Gizli katmanlar boyunca hatanın yayılımı

(2.68) denklemini oluşturan terimler Şekil 2.8'de gösterildiği gibi değişik yollardan gelebilirler. Bu durum (2.68) denklemindeki zincir kuralının ilk teriminin açık hali olan (2.69) denkleminde de görülmektedir. Aynı terimin daha açık ifadeleri (2.70) ve (2.71) denklemlerinde de verilmiştir.

$$\frac{\partial J_r}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial J_r}{\partial S_h^{k+2}} \frac{\partial S_h^{k+2}}{\partial o_i^{k+1}} \tag{2.69}$$

$$\frac{\partial J_r}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}} \left[\frac{\partial J_r}{\partial S_h^{k+2}} \frac{\partial}{\partial o_i^{k+1}} \left(\sum_{i=1}^{n_{k+1}} w_{hi}^{k+1} o_i^{k+1} \right) \right] \tag{2.70}$$

$$\frac{\partial J_r}{\partial o_i^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial J_r}{\partial S_h^{k+2}} w_{hi}^{k+1} \tag{2.71}$$

Çıkış katmanı için yapıldığı gibi, gizli katmanlar için de (2.72) ile verilen delta değerleri tanımlanabilir.

$$\delta_i^{k+1} = -\frac{\partial J_r}{\partial S_i^{k+1}} \tag{2.72}$$

Bu tanımın kullanılması ile (2.68) denkleminin ilk terimi (2.73) denkleminde gösterilen biçimde yazılabilir. (2.74) ve (2.75) denklemleri ise (2.68) denkleminde görülen ikinci ve üçüncü terimlerin daha açık ifade edilmiş durumlarıdır. Elde edilen terimler birleştirilirse, (2.76)'daki delta değerine ve (2.77)'deki parametre güncelleme kuralına ulaşılır.

$$\frac{\partial J_r}{\partial o_i^{k+1}} = -\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{hi}^{k+1} \tag{2.73}$$

$$\frac{\partial o_i^{k+1}}{\partial S_i^{k+1}} = \frac{d\Psi\left(S_i^{k+1}\right)}{dS_i^{k+1}} = \Psi'\left(S_i^{k+1}\right) \tag{2.74}$$

$$\frac{\partial S_i^{k+1}}{\partial w_{ij}^k} = o_j^k \tag{2.75}$$

$$\delta_i^{k+1} = \left(-\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{hi}^{k+1} \right) \Psi'\left(S_i^{k+1}\right) \tag{2.76}$$

$$\Delta w_{ij}^k = \eta \delta_i^{k+1} o_j^k \tag{2.77}$$

Hata geriye yayma yöntemi geniş çapta uygulama alanı bulmasına rağmen yöntemin uygulamadaki başarımı ve güvenilirliği konusunda bazı sorunlar vardır. İlk olarak öğrenme hızına değinmek gerekir. Bir eşleştirmeyi gerçekleştirmek üzere ele alınan bir sinir ağı yapısında, öğrenme süreci boyunca, N parametreli bir ağ için N+1 boyutlu bir uzayda, N değişkenli bir yüzey üzerinde gezen bir noktanın, maliyeti en aza indiren noktayı araması gerekmektedir. Burada değinilen yüzeyin her bir parametre yönünde kısmi türevleri hesaplanmakta ve parametre güncelleme işlemi yapılmaktadır. Bu işlem parametre vektörünün bulunduğu noktayı, yüzey üzerinde bir başka noktaya kaydırmaktadır. Eğer parametreye göre alınan türevler çok küçük genlikte ve parametre vektörü optimal noktaya çok uzak ise öğrenme işlemi çok uzun zaman alacaktır. Uygulanacak bir yöntem, maliyet fonksiyonundaki değişime göre adım büyüklüğünün değiştirilmesidir.

İkinci önemli sorun anlık sıçramalardır. Parametre uzayında oluşan yüzey, eğitim çiftlerinde bulunabilecek gürültüden yada başka çevresel etkilerden dolayı küçük genlikli iniş ve çıkışlar içerebilir. Bu iniş ve çıkışlar kısmi türevlerin hesaplanması esnasında parametrelerin optimal noktaya çok yakın olduğu durumlarda bile büyük genliklere neden olabilir. Bu tür ani sıçramalar, güncelleme kuralında momentum terimi olarak bilinen bir terimin kullanılması ile önlenebilir. Bu durumda parametre güncelleme kuralı (2.78) deki şekli alacaktır.

$$\Delta w_{ij}^k(K+1) = \mu \Delta w_{ij}^k(K) + \eta \delta_i^{k+1} o_j^k \tag{2.78}$$

burada μ, momentum katsayısı olup (0, +1) aralığından seçilen reel bir sayıdır. Bu çalışmada momentum katsayısı, μ = 0.9; öğrenme oranı η= 0.7 ve aktivasyon fonksiyonu olarak sigmoid tipi fonksiyon kullanılmıştır (Bakınız Ek 2).

3. NÖRAL-GENETİK TABANLI OPTİMAL PI KONTROLÖRÜ TASARIMI

Bu bölümde yapay sinir ağları ve genetik algoritmalar kullanarak, klasik PI tipindeki kontrolörlerin optimizasyonunun nasıl gerçekleştirileceği ele alınacaktır.

Kontrol sistemlerinde sıkça karşımıza çıkan ve endüstride en çok uygulama alanı bulan kontrolör yapısı PID kontrolör yapısıdır. Ancak, türev kontrolörünün sisteme direkt olarak bir sıfır ilave etmesi kararlılık üzerinde olumsuz bir tesir bırakır. Bu etkiyi gidermek için genellikle türev kontrolörü kullanılmaz. En genel halde ayrık zamanlı kontrol sistemlerinde PI kontrolör yapısı aşağıdaki gibidir.

$$u(k) = K_p e(k) + K_I T_s \sum_{i=0}^{k} e(i) \tag{3.1}$$

Bu çalışmada, kontrolörün direkt olarak sisteme uygulanması yerine PWM sinyalinin artımı kontrol edileceğinden; yukarıda tanımlanan (3.1) ifadesinin zamana göre birinci türevini almak gereklidir.

$$\frac{u(k)-u(k-1)}{T_s} = K_p \frac{e(k)-e(k-1)}{T_s} + K_I e(k) \tag{3.2}$$

$$u(k) = u(k-1) + K_p[e(k)-e(k-1)] + K_I e(k) T_s \tag{3.3}$$

İlgili katsayıların belirlenmesinde literatürde en çok Ziegler-Nichols ayar yöntemleri kullanılır. Fakat bu yöntemin hassasiyeti görsel algıya dayandığı için, kesin optimal ayarı vermekte zorlanmaktadır. Bu çalışmada, bu katsayıların belirlenmesinde Genetik Algoritmaların gücünden yararlanılacaktır. Genetik Algoritma ile yapılacak optimal katsayıların aranması işlemi sistemin yapay sinir ağı modeli üzerinden gerçeklenecektir. Sistematik optimizasyon yöntemi aşağıdaki gibidir.

Öncelikle sistemin katsayıları deneme – yanılma yöntemi ile ayarlanarak belli bir çalışma aralığı elde edilir. Bu aralıklar K_p için $[K_{P_{\max}}, K_{P_{\min}}]$, K_I için $[K_{I_{\max}}, K_{I_{\min}}]$ şeklindedir. Yukarıda tanımlanan aralıklardan seçilen belli sayıda rasgele K_P ve K_I çiftleri için (3.4) kullanılarak *IAE* değeri, (3.5) kullanılarak *ITAE* değeri ve (3.6) kullanılarak mutlak aşım değerleri hesaplanır.

$$IAE(k) = \sum_{i=0}^{k} e(i) \qquad i=[0,1,...,k] \tag{3.4}$$

$$ITAE(k) = \sum_{i=0}^{k} ie(i) \quad i=[0,1,...,k] \tag{3.5}$$

$$\%Aşım = \frac{\text{maksimum hız - referans hız}}{\text{referans hız}} \times 100 \tag{3.6}$$

Elde edilen *IAE* ve *ITAE* değerleri yapay sinir ağında kullanılmak üzere [0, 1] aralığına normalize edilir. Normalizasyon işlemi için (3.7) ve (3.8) bağıntıları kullanılır. Aşım değeri zaten [0, 1] aralığında olduğu için herhangi bir normalizasyon işlemine ihtiyaç duymaz.

$$IAE_N = \frac{IAE_{gerçek} - IAE_{\min}}{IAE_{\max} - IAE_{\min}} \tag{3.7}$$

$$ITAE_N = \frac{ITAE_{gerçek} - ITAE_{\min}}{ITAE_{\max} - ITAE_{\min}} \tag{3.8}$$

(3.7) ve (3.8) bağıntıları yardımı ile izdüşürülen *IAE* ve *ITAE* değerleri ile aşım değerleri bir dosyaya kaydedilir. Bu işlem, yapay sinir ağını eğitmede kullanılacak değer çifti sayısınca tekrarlanır. Bir sonraki adımda Genetik Algoritmanın uyum fonksiyonu seçilir. Bu çalışmada (3.9) bağıntısında belirtilen uyum fonksiyonunu kullanılacaktır.

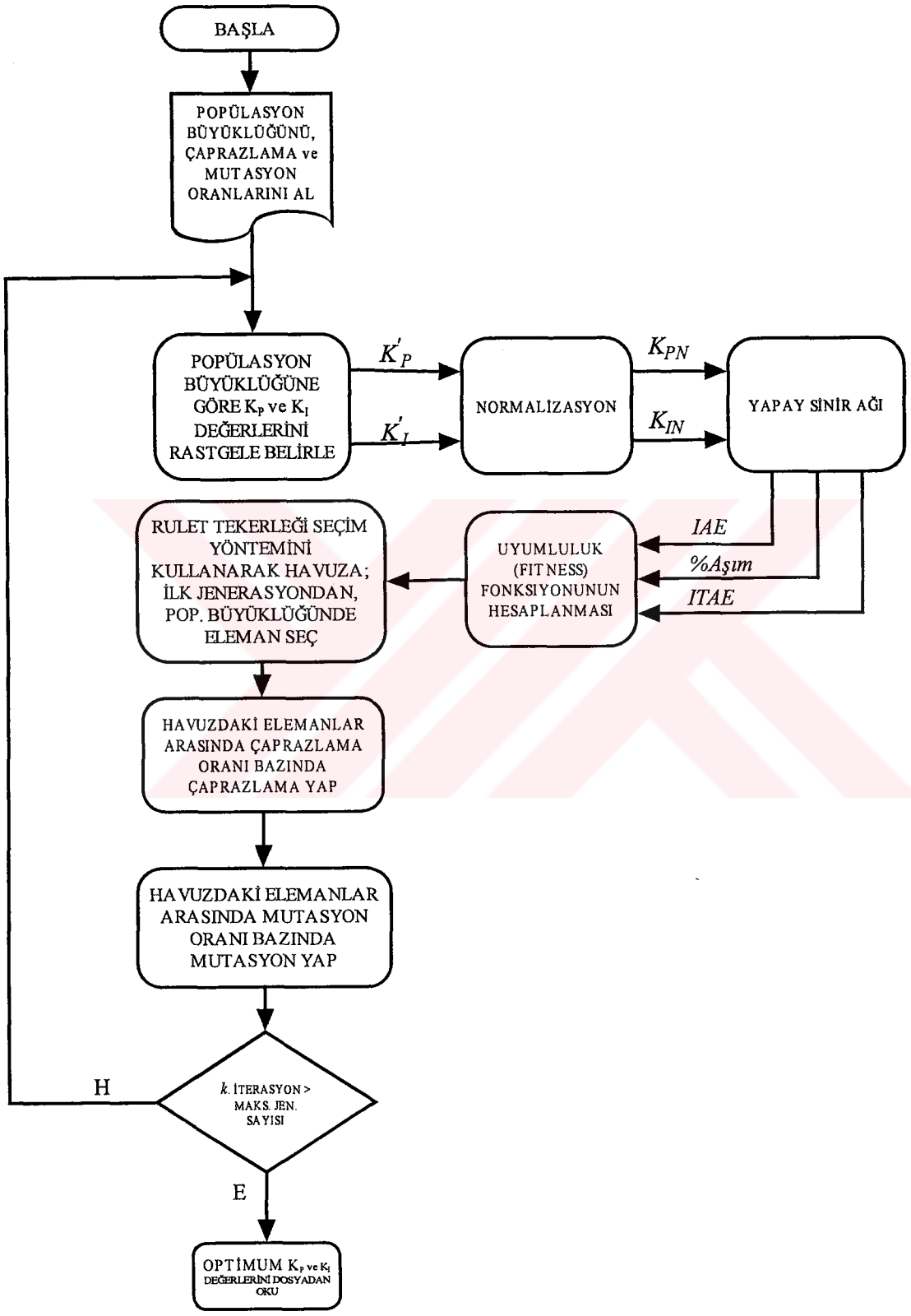
$$F(IAE, ITAE, \%Aşım) = \frac{100}{0.001 + 10 * IAE + 20 * \%Aşım + 10 * ITAE}. \tag{3.9}$$

(3.9) bağıntısında, *%Aşım* değerinin diğer faktörlere (*IAE* ve *ITAE*) göre iki kat daha fazla ağırlaştırılmasının nedeni maksimum aşımın [0, 0.5] aralığında olmasındandır. İki kat daha fazla ağırlaştırma bu uzayın boyutlarını [0, 1] aralığına çekecektir.

Son aşamada ise, yapay sinir ağı şeklinde modellenen sistem, genetik optimizasyon işleminde kullanılarak optimal değerlere yakın kontrolör katsayıları bulunur. Algoritmanın akış diyagramı Şekil 3.1'de gösterilmiştir. Genetik optimizasyon işlemi esnasında kontrolör katsayılarını temsil eden kromozomlar tam sayı değerler şeklinde belli bir aralıkta değişmektedir. Sistemin yapay sinir ağı modeli ise gerçek kontrolör katsayılarını giriş olarak kullanmaktadır. Bu nedenle kromozom değerlerinin gerçek değerlere ait uzaya iz düşürülmesi için (3.10) ve (3.11) bağıntılarından yararlanılacaktır.

$$K_{PN} = \frac{(K_{P_{\max}} - K_{P_{\min}})}{(K'_{P_{\max}} - K'_{P_{\min}})}(K'_P - K'_{P_{\min}}) + K_{P_{\min}} \tag{3.10}$$

$$K_{IN} = \frac{(K_{I_{max}} - K_{I_{min}})}{(K'_{I_{max}} - K'_{I_{min}})}(K'_I - K'_{I_{min}}) + K_{I_{min}} \tag{3.11}$$



Şekil 3.1 Nöral-Genetik tabanlı optimizasyon işlemi

4. NÖRAL-GENETİK TABANLI OPTİMAL BULANIK PI KONTROLÖRÜ TASARIMI

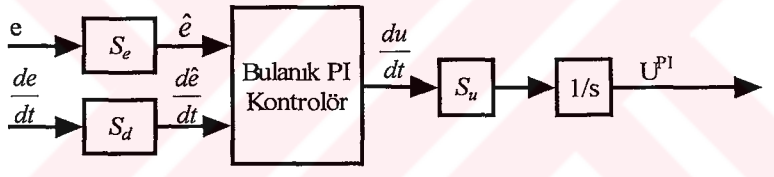
Bilindiği gibi en genel anlamda bir PI kontrolörü aşağıdaki gibi ifade edilir.

$$U^{PI} = K_p e + K_I \int e dt \tag{4.1}$$

Ya da diğer bir ifade ile aynı denklem,

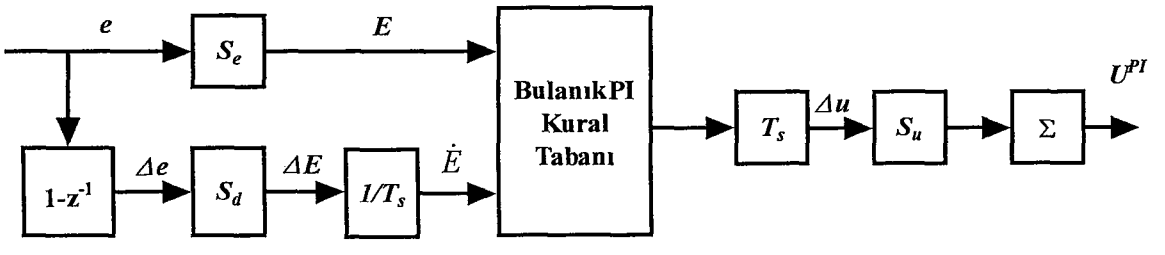
$$U^{PI} = K_p \int \{\dot{e} + \frac{e}{T_i}\} dt \tag{4.2}$$

şeklinde ifade edilebilir. Burada e, referans işaret ile sistem çıkışı arasındaki fark olan hatayı ifade etmektedir. $T_i = K_I / K_p$ şeklinde tanımlanır ve integral zaman sabiti olarak adlandırılır. Eğer e ve $\dot{e}$ bulanık değişkenler olursa, yukarıda tanımlanan denklemler bulanık -PI tipinde kontrolörü belirler. Sürekli zaman iki terimli Bulanık PI kontrolörünün blok diyagramı Şekil 4.1'de gösterilmiştir.



Şekil 4.1 Sürekli zaman bulanık PI kontrolörünün blok diyagramı

Sürekli zaman eşdeğerinden yola çıkarak aynı kontrolörün ayrık zamanlı eşdeğeri ise Şekil 4.2'de gösterilmiştir.



Şekil 4.2 Ayrık zamanlı bulanık PI kontrolörünün blok diyagramı

Burada S_e ve S_d giriş vektörüne ait ölçekleme faktörlerini tanımlarken, S_u çıkış ölçekleme faktörüdür. T_s ise örnekleme periyodunu belirleyen sabittir. Bulanık tabanlı PI kontrolörünün

çıkış fonksiyonu

$$U^{PI} = S_u \sum F\{S_e e, \frac{S_d}{T_s}\dot{e}\}.T_s \tag{4.3}$$

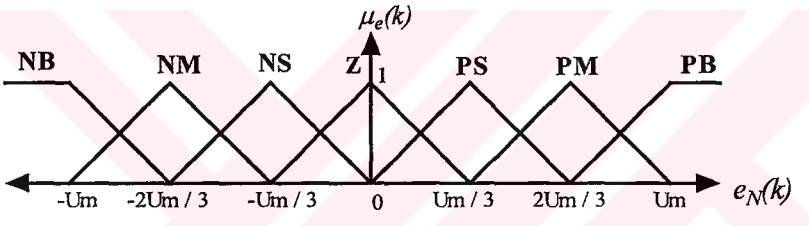
şeklinde tanımlanabilir. Eğer bulanık -PI kontrolörü ile, bulanık olmayan klasik PI kontrolörü arasında benzetim yapılırsa, bulanık K_P ve bulanık K_I sırası ile;

$$K_P = S_u F\{\frac{S_d}{T_s}\}T_s \tag{4.4}$$

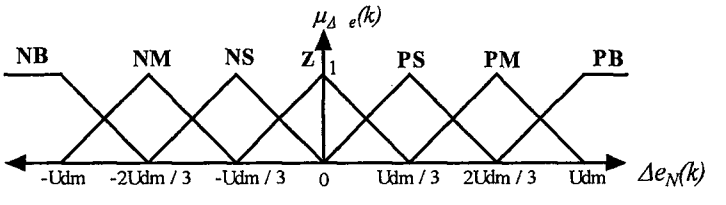
ve

$$K_I = S_U F\{S_e\}T_s \tag{4.5}$$

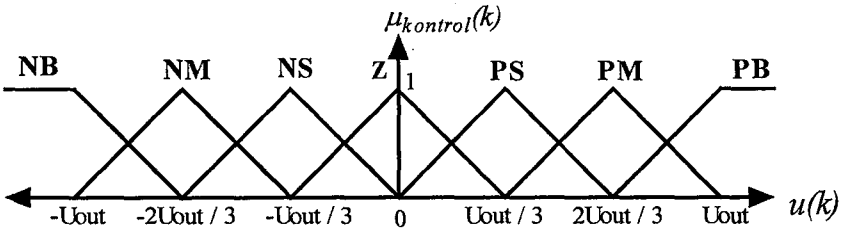
şeklinde tanımlanabilir. Burada $F\{.\}$ bulanık işlemlerin fonksiyonunu ifade eder.



Şekil 4.3 Normalize edilmiş hata işaretine ait bulanıklaştırma üyelik fonksiyonları



Şekil 4.4 Normalize edilmiş hatanın hızına ait bulanıklaştırma üyelik fonksiyonları



Şekil 4.5 Kontrol işaretine ait netleştirme üyelik fonksiyonları

Bu çalışmada kullanılan bulanıklaştırma ve netleştirme işlemleri için tanımlanan üyelik fonksiyonları Şekil 4.3, Şekil 4.4 ve Şekil 4.5'de gösterilmiştir.Burada NB = Negatif Büyük, NM = Negatif Orta, NS = Negatif Küçük, Z = Sıfır, PS = Pozitif Küçük, PM = Pozitif Orta, PB = Pozitif Büyük anlamındadır. U_m normalize edilmiş hata uzayının genişliğini; U_{dm} , normalize edilmiş hatanın değişimine ait uzayın genişliğini ve U_{out} çıkış uzayının genişliğini belirler. Şekil 4.3, 4.4 ve 4.5'den de görülebileceği gibi, giriş ve çıkış bulanık üyelik fonksiyonları hesaplamada kolaylık yaratması amacı ile ikiz-kenar üçgen şeklinde seçilmişlerdir. Çıkarım işlemi olarak Kısım 2.1.7.2 de tanımlanan minimum tipi çıkarım motoru kullanılmıştır. Ayrıca netleştirme işlemi için (2.46) bağıntısı ile tanımlanan ortalama merkezi yöntemi kullanılmıştır. Kısaca bulanık kontrolöre ait kısmı (4.6) ile ifade edebiliriz.

$$f(x) = \frac{\sum_{l=1}^{M} \bar{y}^l \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i)\right)}{\sum_{l=1}^{M} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i)\right)} \tag{4.6}$$

Burada $x \in U \subset \Re^n$ bulanık sistemin girişini; $f(x) \in V \subset \Re$ bulanık sistemin çıkışını tanımlar. Bu çalışmada 2 girişli ve tek çıkışlı bulanık kontrolörler kullanıldığından ve her bir giriş değişkeni için 7 farklı bulanık küme tanımlandığından, (4.6) ifadesi (4.7) biçimini alır.

$$f(x) = \frac{\sum_{l=1}^{49} \bar{y}^l \left(\prod_{i=1}^{2} \mu_{A_i^l}(x_i)\right)}{\sum_{l=1}^{49} \left(\prod_{i=1}^{2} \mu_{A_i^l}(x_i)\right)} \tag{4.7}$$

4.1 NGTOFPI Kontrolörüne ait Kural Tabanı Tasarımı

Kural tabanının tasarımında hata durum uzayından yararlanılacaktır. Kural tabanının oluşturulması işlemine, sistem basamak cevabının Şekil 4.6'daki gibi 4 ayrı alana bölünmesi ile başlanır. Bu alanlar; A_1, A_2, A_3 ve A_4 dür. Ayrıca sistem basamak cevabı üzerinde $\{b_1, b_2\}$ geçiş noktaları ile $\{c_1, c_2\}$ tepe ve vadi noktaları belirlenir. Sistemin denge noktası Şekil 4.7'de gösterilen faz düzleminin merkezidir. Daha sonra aşağıdaki adımlar izlenerek Çizelge 4.1 de gösterilen kural tabanı elde edilir (Li ve Gatland, 1996).

1. Hata ve hatanın türevi sıfır ise mevcut kontrol işareti korunacak,
2. Eğer hata işareti sıfıra belli bir hızda yakınsıyor ise mevcut kontrol işareti korunacak,
3. Eğer hata işareti kendi kendine düzelemeyecek durumda ise aşağıdaki 5 alt kriter yardımı ile kural işareti belirlenecek.

a) $\{b_1, b_2\}$ geçiş noktaları için tanımlanan kurallar A_2 ve A_4 alanlarında aşımları engelleyecek şekilde seçilmelidir.

b) Tepe ve vadi noktaları olan $\{c_1, c_2\}$ için tanımlanan kurallar sistem cevabını hızlandıracak şekilde seçilmelidir.

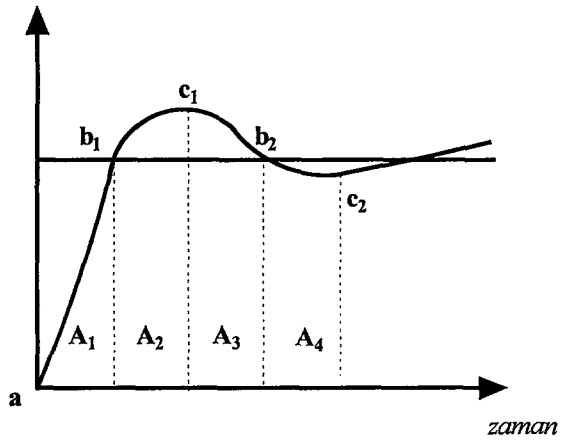
c) *e*, hata işareti büyükken, A_1 ve A_3 bölgeleri için tanımlanan kurallar sistemi hızlandırmalı; *e*, hata işareti küçükken, A_2 ve A_4 bölgeleri için tanımlanan kurallar aşımları engellemelidir.

d) A_2 bölgesi için tanımlanan kurallar tepe civarında aşımı azaltmalıdır.

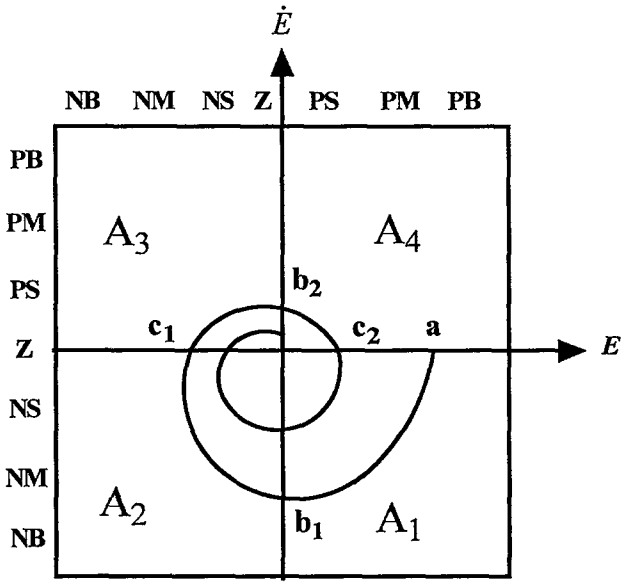
e) A_4 bölgesi için tanımlanan kurallar vadi civarında aşımı azaltmalıdır.

Çizelge 4.1 Bulanık PI kontrolörü için tasarlanan kural tabanı

Δ e/e	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NB	NM	NS	Z
NM	NB	NB	NB	NM	NS	Z	PS
NS	NB	NB	NM	NS	Z	PS	PM
Z	NB	NM	NS	Z	PS	PM	PB
PS	NM	NS	Z	PS	PM	PB	PB
PM	NS	Z	PS	PM	PB	PB	PB
PB	Z	PS	PM	PB	PB	PB	PB



Şekil 4.6 Sistem basamak cevabının bölümlere ayrılması



Şekil 4.7 Bulanık PI kontrolörü için hata faz düzlemi

4.2 Giriş Ölçekleme Faktörlerinin Nöral-Genetik Tabanlı Optimizasyonu

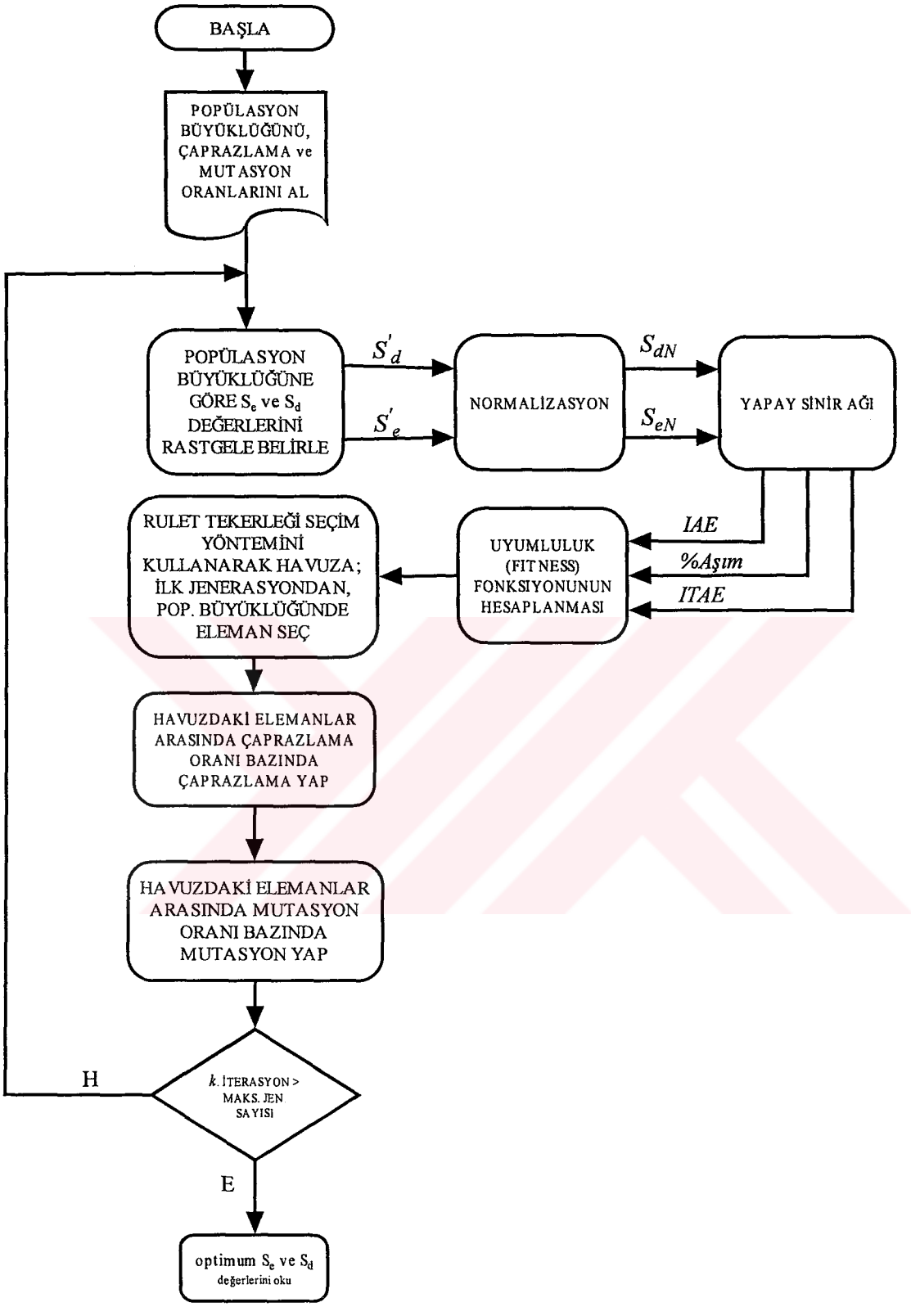
Kural tabanının oluşturulması işlemini ölçekleme faktörlerinin optimizasyonu takip eder. Zira ölçekleme faktörlerinin seçimi sistem performansını diğer bulanık değişkenlere göre çok daha fazla etkiler. Bu çalışmada NGTOFPI çıkış ölçekleme faktörü deneme yanılma yöntemiyle ayar edilecektir. Giriş ölçekleme faktörleri ise Nöral-Genetik tabanlı optimizasyon yöntemi ile ayar edilecektir.

Nöral-Genetik tabanlı optimizasyon işleminde öncelikle sistemin giriş ölçekleme faktörleri sabit tutularak, çıkış ölçekleme faktörleri kabul edilebilir bir geçici rejim cevabı verecek şekilde ayar edilir.

Daha sonra S_e ve S_d için çalışma aralıkları deneme–yanılma yöntemi ile belirlenir. Bu aralıklar S_e için $[S_{e_{\max}}, S_{e_{\min}}]$; S_d için $[S_{d_{\max}}, S_{d_{\min}}]$ şeklindedir.

Bu aralıklardan seçilen belli sayıda rasgele S_e ve S_d çiftleri için (3.4) kullanılarak *IAE* değeri, (3.5) kullanılarak *ITAE* değeri ve (3.6) kullanılarak mutlak aşım değeri hesaplanır. Elde edilen *IAE* ve *ITAE* değerleri yapay sinir ağında kullanılmak üzere [0, 1] aralığına iz düşürülür. İz düşürme işlemi için (3.7) ve (3.8) bağıntıları kullanılır. Aşım değeri zaten [0,1] aralığında olduğu için herhangi bir normalizasyon işlemine ihtiyaç duymaz.

(3.7) ve (3.8) bağıntıları yardımı ile izdüşürülen *IAE* ve *ITAE* değerleri ile aşım değerleri bir



Şekil 4.8 NGTOFPI kontrolörüne ait ölçekleme faktörlerinin optimizasyon işlemi

dosyaya kaydedilir. Bu işlem yapay sinir ağını eğitmede kullanılacak değer çifti sayısınca tekrarlanır.

Bir sonraki adımda Genetik Algoritmanın uyum fonksiyonu seçilir. Bu çalışmada (3.9) bağıntısında belirtilen uyum fonksiyonu kullanılacaktır. (3.9) bağıntısında, %Aşım değerinin diğer faktörlere (*IAE* ve *ITAE*) göre iki kat daha fazla ağırlaştırılmasının nedeni maksimum aşımın [0, 0.5] aralığında olmasındandır. İki kat daha fazla ağırlaştırma bu uzayın boyutlarını [0, 1] aralığına çekecektir.

Son aşamada ise, yapay sinir ağı şeklinde modellenen sistem genetik optimizasyon işleminde kullanılarak optimal değerlere yakın kontrolör katsayıları bulunur. Algoritmanın akış diyagramı Şekil 4.8'de gösterilmiştir.

Genetik optimizasyon işlemi esnasında kontrolör katsayılarını temsil eden kromozomlar tam sayı değerler şeklinde belli bir aralıkta değişmektedir. Sistemin yapay sinir ağı modeli ise gerçek kontrolör katsayılarını giriş olarak kullanmaktadır. Bu nedenle kromozom değerlerinin gerçek değerlere iz düşürülmesi için (4.8) ve (4.9) bağıntılarından yararlanılacaktır.

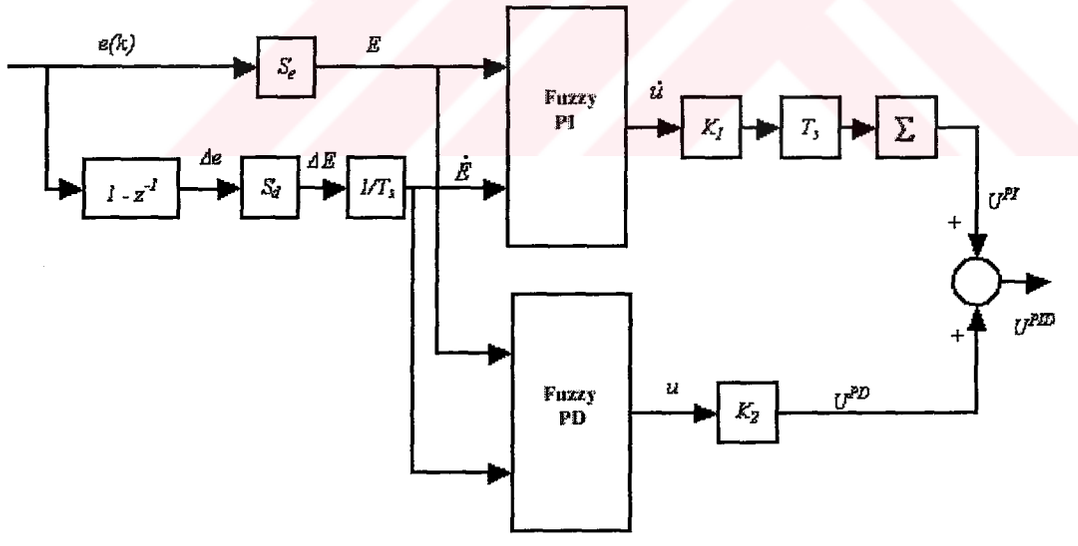
$$S_{eN} = \frac{(S_{e_{\max}} - S_{e_{\min}})}{(S'_{e_{\max}} - S'_{e_{\min}})}(S'_e - S'_{e_{\min}}) + S_{e_{\min}} \tag{4.8}$$

$$S_{dN} = \frac{(S_{d_{\max}} - S_{d_{\min}})}{(S'_{d_{\max}} - S'_{d_{\min}})}(S'_d - S'_{d_{\min}}) + S_{d_{\min}} \tag{4.9}$$

5. NÖRAL-GENETİK TABANLI OPTİMAL BULANIK-PID KONTROLÖRÜ TASARIMI

Bulanık–PI tipi kontrolörler, yüksek mertebeden sistemlerde, geçici rejim bakımından düşük performanslar gösterirler. Diğer taraftan PD tipindeki bulanık kontrolörler ise büyük kararlı hal hatalarına neden olurlar. PD tipindeki bulanık kontrolörlerin geçici rejim performanslarındaki başarılarını, PI tipi bulanık kontrolörlerin kararlı hal performanslarındaki başarıları ile birleştirmek amacı ile üç terimli bulanık kontrolörler olan bulanık-PID tipi kontrolörler ortaya atılmıştır.

Normal bir bulanık-PID kontrolöründe, üç giriş ve tek çıkış bulunmaktadır. Girişlerden birincisi hata, ikincisi hatanın hızı ve üçüncüsü hatanın ivmesidir. Ancak böyle bir yapıya sahip bulanık kontrolörünün kural tabanının üç boyutlu olması gerekmektedir. Bu da uygulanabilirlik açısından sorunlar ortaya koymaktadır. Ayrıca hatanın ivmesine ait terim, performans üzerinde oldukça az etkilidir. İşte bu dezavantajları ortadan kaldırabilmek amacı ile hibrid tipte bir bulanık-PID kontrolörü ortaya atılmıştır. Hibrid tipte bulanık-PID kontrolörünün blok diyagramı Şekil 5.1'den görülmektedir.



Şekil 5.1 Hibrid Bulanık PID kontrolörünün blok diyagramı (Li ve Gatland, 1996)

Hibrid bulanık -PID kontrolörünün çıkışı aslında bulanık -PI ve bulanık -PD tipindeki iki ayrı kontrolörün süperpozisyonudur. Matematiksel olarak *k.* örnek için kontrolör çıkışı,

$$U^{PID}(k) = U^{PI}(k) + U^{PD}(k) \tag{5.1}$$

şeklinde tanımlanabilir. Burada bulanık -PI kontrolörünün çıkışı,

$$U^{PI}(k) = U^{PI}(k-1) + \Delta U^{PI}(k) \tag{5.2}$$

şeklindedir. Burada tanımlanan $\Delta U^{PI}(k)$ ise,

$$\Delta U^{PI}(k) = \dot{u}(k).T_s \tag{5.3}$$

şeklinde tanımlanır. $\dot{u}(k)$ ise,

$$\dot{u}(k) = K_I e(k) + K_P \dot{e}(k) \tag{5.4}$$

şeklindedir. $U^{PD}(k)$ ise klasik PD tipi kontrolörlere benzetilerek,

$$U^{PD}(k) = K_p e(k) + K_D \dot{e}(k) \tag{5.5}$$

şeklinde tanımlanabilir. O halde hibrid bir bulanık -PID kontrolörünün K_P, K_I ve K_D katsayıları,

$$K_P = T_s K_1 F\{\frac{S_d}{T_s}\} + K_2 F\{S_e\} \tag{5.6}$$

$$K_I = T_s K_1 F\{S_e\} \tag{5.7}$$

$$K_D = K_2 F\{\frac{S_e}{T_s}\} \tag{5.8}$$

şeklindedir (Li ve Gatland, 1996). (5.6), (5.7) ve (5.8) denklemlerinden de görülebileceği gibi, oran, türev ve integral etkileri bulanık -PID tipi kontrolörlerde iç içe girmiştir. Çıkış ölçekleme faktörleri olan K_I ve K_2, bulanık oran katsayısı K_p üzerinde oldukça etkilidir. Giriş ölçekleme faktörlerinin etkileri ise gizlidir. Yani bu çalışmada giriş ölçekleme faktörlerinin optimizasyonu ile bulanık kontrolörün performansları direkt olarak kontrol edilebilecektir. Çıkış ölçekleme faktörlerinin ayarı ise sadece sistemin performansını klasik kontrolörlerdekine benzer şekilde lineer olarak etkilemektedir. Giriş ölçekleme faktörlerinin ayarı ile yapılan işlem direkt olarak $F\{.\}$ üzerinde etkili olacağı için non-lineer bir kontrolör tasarımı gerçeklenmiş olacaktır.

Bu çalışmada PI tipi bulanık kontrolörü ve PD tipi bulanık kontrolörlerinin bulanıklaştırma ve netleştirme işlemleri için kullanılan üyelik fonksiyonu tipleri ve tanımlı uzayların boyutları

birbirinin aynı olarak seçilmiştir. Bulanıklaştırma ve netleştirme işlemleri için tanımlanan üyelik fonksiyonları Şekil 4.3, Şekil 4.4 ve Şekil 4.5'de gösterilmiştir. Burada NB = Negatif Büyük, NM = Negatif Orta, NS = Negatif Küçük, Z = Sıfır, PS = Pozitif Küçük, PM = Pozitif Orta, PB = Pozitif Büyük anlamındadır. U_m normalize edilmiş hata uzayının genişliğini; U_{dm}, normalize edilmiş hatanın değişimine ait uzayın genişliğini ve U_{out} çıkış uzayının genişliğini ifade eder. Şekil 4.3, 4.4 ve 4.5'den de görülebileceği gibi giriş, ve çıkış bulanık üyelik fonksiyonları hesaplamada kolaylık yaratması amacı ile ikiz-kenar üçgen şeklinde seçilmişlerdir. Çıkarım işlemi olarak Kısım 2.1.7.2 de tanımlanan, minimum tipi çıkarım motoru kullanılmıştır. Ayrıca netleştirme işlemi için (2.46) bağıntısı ile tanımlanan ortalama merkezi yöntemi kullanılmıştır.

5.1 NGTOFPID Kontrolörüne ait Kural Tabanı Tasarımı

Bulanık-PI kontrolörünün kural tabanı Kısım 4.1'deki kurallar göz önüne alınarak Çizelge 4.1 şeklinde oluşturulmuştur. Bulanık-PD kontrolörünün kural tabanının tasarımında da Kısım 4.1'deki kurallardan yararlanılmıştır. Bulanık-PD kontrolörü için elde edilen kural tabanı Çizelge 5.1'de gösterilmiştir.

Çizelge 5.1 Bulanık-PD kontrolörünün kural tabanı

Δ e/e	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NS	PS	PS	PS
NM	NB	NB	NB	NS	PS	PS	PS
NS	NB	NB	NM	NS	PS	PS	PM
Z	NB	NM	NS	Z	PS	PM	PB
PS	NM	NS	NS	PS	PM	PB	PB
PM	NS	NS	NS	PS	PB	PB	PB
PB	NS	NS	NS	PS	PB	PB	PB

5.2 Giriş Ölçekleme Faktörlerinin Nöral-Genetik Tabanlı Optimizasyonu

Kural tabanının oluşturulması işlemini ölçekleme faktörlerinin optimizasyonu takip eder. Bu çalışmada, NGTOFPID kontrolörünün çıkış ölçekleme faktörleri olan K_1 ve K_2 deneme yanılma yöntemiyle ayar edilecektir. Buna karşın, giriş ölçekleme faktörleri ise Nöral-Genetik tabanlı optimizasyon yöntemi ile ayar edilecektir.

Nöral-Genetik tabanlı optimizasyon işleminde, öncelikle sistemin giriş ölçekleme faktörleri sabit tutularak, çıkış ölçekleme faktörleri kabul edilebilir bir geçici rejim cevabı verecek

şekilde kabaca ayar edilir.

Daha sonra S_e ve S_d için çalışma aralıkları deneme – yanılma yöntemi ile belirlenir. Bu aralıklar S_e için $[S_{e_{max}}, S_{e_{min}}]$; S_d için $[S_{d_{max}}, S_{d_{min}}]$ şeklindedir.

Bu aralıklardan seçilen belli sayıda rasgele S_e ve S_d çiftleri için (3.4) kullanılarak *IAE* değeri, (3.5) kullanılarak *ITAE* değeri ve (3.6) kullanılarak mutlak aşım değeri hesaplanır. Elde edilen *IAE* ve *ITAE* değerleri yapay sinir ağında kullanılmak üzere [0, 1] aralığına iz düşürülür. İz düşürme işlemi için (3.7) ve (3.8) bağıntıları kullanılır. Aşım değeri, zaten [0, 1] aralığında olduğu için herhangi bir normalizasyon işlemine ihtiyaç duymaz.

(3.7) ve (3.8) bağıntıları yardımı ile izdüşürülen *IAE* ve *ITAE* değerleri ile aşım değerleri bir dosyaya kaydedilir. Bu işlem yapay sinir ağını eğitmede kullanılacak değer çifti sayısınca tekrarlanır.

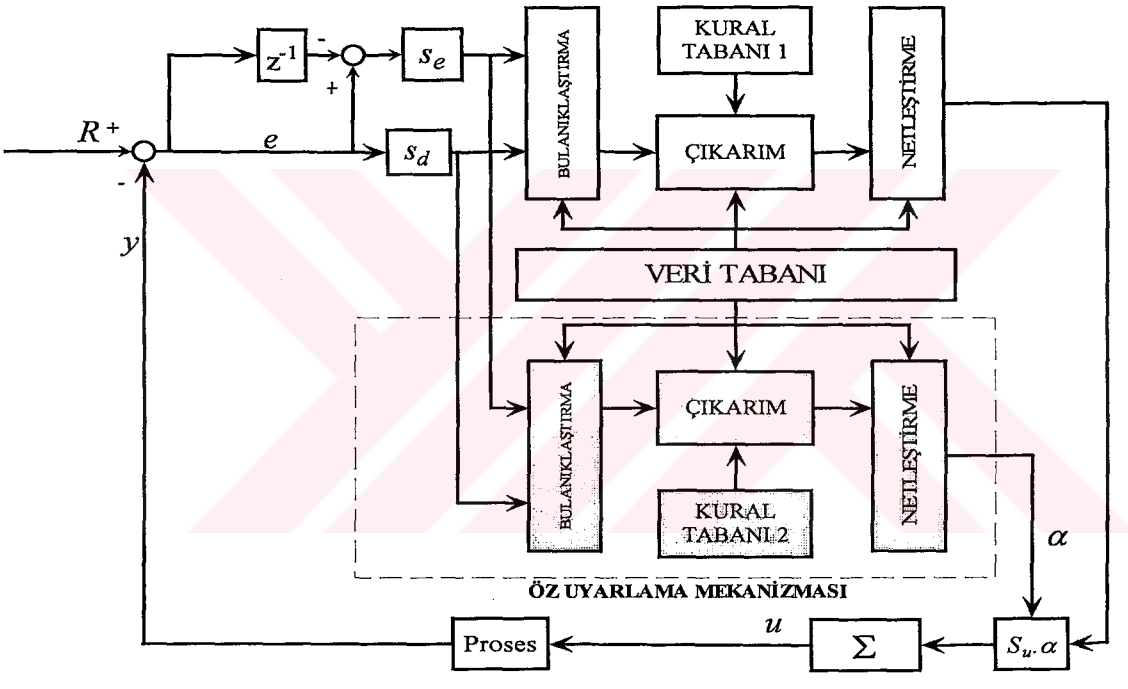
Bir sonraki adımda Genetik Algoritmanın uyum fonksiyonu seçilir. Bu çalışma için de (3.9) bağıntısında belirtilen uyum fonksiyonunu kullanılır.

Son aşamada ise, yapay sinir ağı şeklinde modellenen sistem genetik optimizasyon işleminde kullanılarak optimal değerlere yakın kontrolör katsayıları bulunur. Algoritmanın akış diyagramı, Şekil 4.8'de tanımlanan NGTOFPI kontrolörünün giriş ölçekleme faktörlerinin ayarında kullanılan yapının aynısıdır.

Genetik optimizasyon işlemi esnasında kontrolör katsayılarını temsil eden kromozomlar tam sayı değerler şeklinde belli bir aralıkta değişmektedir. Oysaki sistemin yapay sinir ağı modeli gerçek kontrolör katsayılarını giriş olarak kullanmaktadır. Bu nedenle kromozom değerlerinin gerçek değerlere iz düşürülmesi için (4.8) ve (4.9) bağıntılarından yararlanılır.

6. NÖRAL-GENETİK TABANLI OPTİMAL ÖZ-UYARLAMALI BULANIK -PI KONTROLÖRÜ TASARIMI

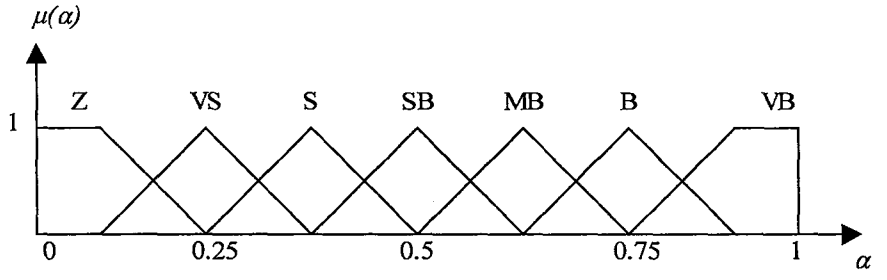
Dört ve beşinci bölümlerde ele alınan bulanık-PI ve bulanık-PID kontrolörlerinin çıkış ölçekleme faktörleri olan S_u, K_1 ve K_2 değişkenleri manuel olarak tasarım esnasında ayar edilmekte ve kontrolörün prosese uygulanması esnasında değerleri sabit tutulmaktaydı. Ancak, bilinen bir gerçektir ki, çıkış ölçekleme faktörlerinin sabit tutulması kontrolörün performansının belli bir aralıkta sabit tutulması ile eşdeğerdir. Bu sakıncayı ortadan kaldırmak amacı ile bu bölümde, PI tipi bulanık kontrolörün çıkış ölçekleme katsayısını ayar eden bir öz-uyarlama yapısı kontrolör mimarisine eklenmiştir. Yeni yapı Şekil 6.1 de görülmektedir.



Şekil 6.1 Öz uyarlamalı bulanık-PI tipi kontrolör

Öz-uyarlama mekanizmasının da kontrolöre ilave edilmesi sonucunda, yapı içersinde manuel olarak ayarlanması gereken birim kalmamıştır. PI tipi bulanık kontrolörün ve öz-uyarlama mekanizmasının bulanıklaştırma üyelik fonksiyonları ile bulanık PI tipi kontrolörün çıkış üyelik fonksiyonu Şekil 4.3, Şekil 4.4 ve Şekil 4.5'deki gibi seçilmiştir. Öz-uyarlama mekanizmasının çıkışını temsil eden α ayar parametresi ise Şekil 6.2'den de görülebileceği gibi [0,1] aralığında değişen değerler almaktadır. NGTOSTFPI kontrolörünün giriş üyelik fonksiyonları ise sistemin YSA modeli üzerinden genetik ortamda optimize edilecektir. Şekil 6.2'de tanımlanan ve öz-uyarlama yapısının çıkışını oluşturan üyelik fonksiyonları ise şu

şekildedir: Z = Sıfır, VS = Çok Küçük, S = Küçük, SB = Biraz Küçük, MB = Orta Büyük, B = Büyük ve VB = Çok Büyük.



Şekil 6.2 α ayar parametresine ait çıkış üyelik fonksiyonları

Çıkarım işlemi olarak kısım 2.1.7.2 de tanımlanan minimum tipi çıkarım motoru kullanılmıştır. Ayrıca netleştirme işlemi için (2.46) bağıntısı ile tanımlanan ortalama merkezi yöntemi kullanılmıştır.

6.1 NGTOSTFPI Kontrolörüne ait Kural Tabanı Tasarımı

Ana kontrolörü oluşturan Bulanık-PI yapısındaki kurallar,

R_{PI} : IF e is $\boldsymbol{E}$ and Δe is $\boldsymbol{\Delta E}$ THEN Δu is $\boldsymbol{\Delta U}$ (6.1)

şeklindedir. Benzer şekilde öz-uyarlama mekanizmasını oluşturan kurallar,

R_{α} : IF e is $\boldsymbol{E}$ and Δe is $\boldsymbol{\Delta E}$ THEN α is $\boldsymbol{\alpha}$ (6.2)

şeklindedir.

Çizelge 6.1 (a) NGTOSTFPI kontrolörünün bulanık -PI kısmına ait kural tabanı. (b) NGTOSTFPI kontrolörüne ait öz-uyarlama mekanizmasının kural tabanı

Δ e/e	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NM	NS	NS	Z
NM	NB	NM	NM	NM	NS	Z	PS
NS	NB	NM	NS	NS	Z	PS	PM
Z	NB	NM	NS	Z	PS	PM	PB
PS	NM	NS	Z	PS	PS	PM	PB
PM	NS	Z	PS	PM	PM	PM	PB
PB	Z	PS	PS	PM	PB	PB	PB

(a)

Δ e/e	NB	NM	NS	Z	PS	PM	PB
NB	VB	VB	VB	B	SB	S	Z
NM	VB	VB	B	B	MB	S	VS
NS	VB	MB	B	VB	VS	S	VS
Z	S	SB	MB	Z	MB	SB	S
PS	VS	S	VS	VB	B	MB	VB
PM	VS	S	MB	B	B	VB	VB
PB	Z	S	SB	B	VB	VB	VB

(b)

Bulanık-PI tipi kontrolörün kural tabanı Çizelge 6.1.(a)'da gösterilmiştir. Yapısal olarak Çizelge 4.1'deki kural tabanının üzerinde yapılan ufak değişiklikler neticesinde ortaya çıkmıştır. Bu kural tabanının amacı, hata işaretini kayan kipli moda sokarak orada tutmaktır. Bulanık-PI kontrolörünün genel performansını arttırmak için ana kontrolöre paralel çalışan öz-uyarlama mekanizması dahil edilmiştir. Öz-uyarlama mekanizmasına ait kural tabanının tasarımında aşağıdaki hususlar dikkate alınmıştır:

a) Kontrolörün daha az aşım ve daha kısa yerleşme zamanı üretebilmesi için (yükselme zamanından ödün vermeden), hata büyük değerlerde iken kontrolör kazancı küçük değerlerde tutulmuştur. Örneğin: If e is PB and Δe is NS then α is VS; veya If e is NM and Δe is PM then α is S. Ayrıca hata büyük iken ve e ve Δe aynı işarete sahipken (bu durumda hata sadece referanstan çok uzakta değil aynı zamanda giderek uzaklaşmaktadır) kazanç işareti durumu daha da kötüleştirmemek amacı ile çok büyük değerlere çıkartılır. Bu olay If e is PB and Δe is PS then α is VB; veya If e is NM and Δe is NM then α is VB, şeklindeki kurallar ile gerçeklenir.

b) Proses ihtiyacına göre referans nokta etrafında büyük kazanç değişimleri istenebilir. Örneğin, hata küçük değerlerde iken üst ve alt aşımları engellemek amacı ile If e is Z and Δe is NM then α is B tipinde kurallar kullanılmıştır. Bu kural şu anlama gelir: Proses çıkışı referans noktaya varmış ve üst aşım yaratacak şekilde referans noktadan uzaklaşmaktadır. Bu durumda, büyük kazanç işareti yukarı doğru yönlenmiş çıkışı çok sert bir şekilde engelleyerek küçük aşımlara neden olmaktadır. Benzer şekilde If e is NS and Δe is PS then α is VS şeklindeki kural ile de alt aşımlar engellenmeye çalışılmıştır. Referans nokta etrafındaki bu tarz geniş kazanç değişimleri sayesinde çatırtılar önlenmiş olur ve yerleşme zamanları da küçülür. Referans nokta etrafındaki bu tarz büyük genlik değişimleri ancak bu mimaride kullanılan öz-uyarlama mekanizmasına benzer yöntemler ile sağlanabilir.

c) Çoğu endüstriyel sistem çalışma ortamında yük bozucuları tarafından etki altındadır. İyi bir kontrolör mekanizması yük değişimlerinden etkilenmez. Diğer bir deyişle, iyi bir kontrolör ani bir yük değişimi meydana geldiğinde, sistem çıkışını kısa bir süre içersinde tekrar kararlı duruma taşır. Bu, kontrolör kazancının çıkabilecek en yüksek noktaya çıkarılması ile sağlanır. Bu nedenle, yük değişimlerine karşı kontrolör performansı korumak amacı ile, If e is PS and Δe is PM then α is B; veya If e is NS and Δe is NM then α is B, şeklindeki kurallar kural tabanına eklenmiştir. Burada dikkat edilmesi gereken nokta şudur: Ani bir yük değişimi meydana geldiği anda e küçük bir

değere sahipse Δe çok büyük (e ile aynı işarette) değerler alır. Bu durumda yapılacak iş α kazancını çok büyük değerlere çıkartmaktır. Kararlı hal durumunda iken ($e \approx 0$ ve $\Delta e \approx 0$) kontrolör kazancı çok küçük tutularak referans nokta etrafındaki çatırtı problemi önlenmeye çalışılır.

6.2 Giriş Ölçekleme Faktörlerinin Nöral-Genetik Tabanlı Optimizasyonu

Kural tabanının oluşturulması işlemini ölçekleme faktörlerinin optimizasyonu takip eder. Bu çalışmada NGTOSTFPI kontrolörünün çıkış ölçekleme faktörü olan S_u deneme yanılma yöntemiyle ayar edilecektir. Buna karşın, giriş ölçekleme faktörleri ise Nöral-Genetik tabanlı optimizasyon yöntemi ile ayar edilecektir.

Nöral-Genetik tabanlı optimizasyon işleminde öncelikle sistemin giriş ölçekleme faktörleri sabit tutulur; öz-uyarlama mekanizması kontrolör yapısı dışına çıkartılır ve çıkış ölçekleme faktörleri kabul edilebilir bir geçici rejim cevabı verecek şekilde ayar edilir. Daha sonra öz uyarlama mekanizması sisteme dahil edilir. Bu yapı sisteme dahil edildiğinde sistemin çıkış ölçekleme faktörü yaklaşık 1/3 oranında zayıflatılır.

Bir sonraki adımda S_e ve S_d için çalışma aralıkları deneme – yanılma yöntemi ile belirlenir. Bu aralıklar S_e için $[S_{e_{\max}}, S_{e_{\min}}]$; S_d için $[S_{d_{\max}}, S_{d_{\min}}]$ şeklindedir.

Bu aralıklardan seçilen belli sayıda rasgele S_e ve S_d çiftleri için (3.4) bağıntısı kullanılarak *IAE* değeri, (3.5) bağıntısı kullanılarak *ITAE* değeri ve (3.6) bağıntısı kullanılarak mutlak aşım değeri hesaplanır. Elde edilen *IAE* ve *ITAE* değerleri yapay sinir ağında kullanılmak üzere [0, 1] aralığına iz düşürülür. İz düşürme işlemi için (3.7) ve (3.8) bağıntıları kullanılır. Aşım değeri zaten [0, 1] aralığında olduğu için herhangi bir normalizasyon işlemine ihtiyaç duymaz.

(3.7) ve (3.8) bağıntıları yardımı ile izdüşürülen *IAE* ve *ITAE* değerleri ile aşım değerleri bir dosyaya kaydedilir. Bu işlem yapay sinir ağını eğitmede kullanılacak değer çifti sayısınca tekrarlanır.

Bir sonraki adımda Genetik Algoritmanın uyum fonksiyonu seçilir. Bu çalışma için de (3.9) bağıntısında belirtilen uyum fonksiyonunu kullanılacaktır.

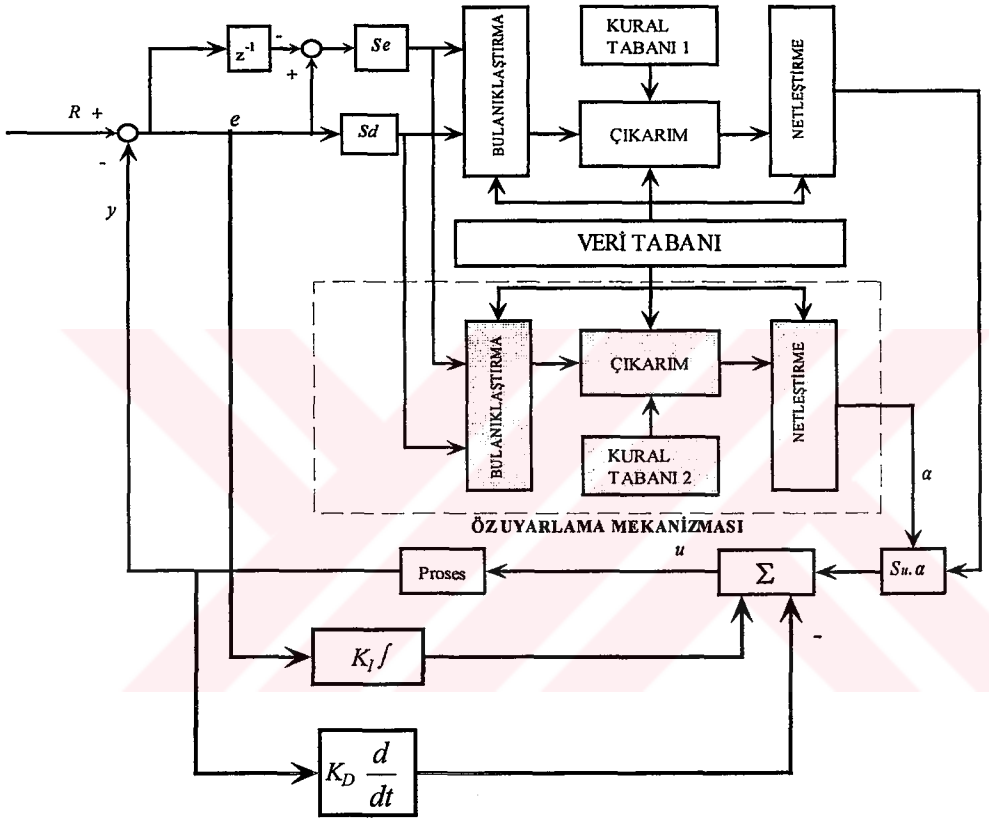
Son aşamada ise, yapay sinir ağı şeklinde modellenen sistem genetik optimizasyon işleminde kullanılarak optimal değerlere yakın kontrolör katsayıları bulunur. Algoritmanın akış diyagramı Şekil 4.8'de gösterilen NGTOFPI kontrolörünün giriş ölçekleme faktörlerinin

ayarında kullanılan yapının aynısıdır.

Genetik optimizasyon işlemi esnasında kontrolör katsayılarını temsil eden kromozomlar tam sayı değerler şeklinde belli bir aralıkta değişmektedir. Sistemin yapay sinir ağı modeli ise gerçek kontrolör katsayılarını giriş olarak kullanmaktadır. Bu nedenle kromozom değerlerinin gerçek değerlere iz düşürülmesi için (4.8) ve (4.9) bağıntılarından yararlanılacaktır.

7. NÖRAL-GENETİK TABANLI OPTİMAL ÖZ-UYARLAMALI BULANIK P-ID KONTROLÖRÜ TASARIMI

Altıncı bölümde tanımlanan öz uyarlamalı kontrolör yapısının performansını arttırmak amacı ile bu bölümde mevcut öz-uyarlamalı kontrolör yapısına bulanık kontrolörden bağımsız türev ve integral kontrolörleri eklenmiştir. Elde edilen yeni kontrolör yapısı Şekil 7.1'de gösterilmiştir.



Şekil 7.1 NGTOSTFP-ID kontrolörüne ait blok diyagram

Burada dikkate edilecek nokta, türev işleminin hata işareti yerine, direkt olarak çıkış işaretinin bir fonksiyonu şeklinde tanımlanmasıdır. Bu tarz bir türev işleminin kullanılmasının nedeni referans nokta sıçramalarının* türev işlemini kararsızlaştırmasını engellemektir. Bilindiği üzere türev işlemi standart olarak hata işareti üzerine uygulanır. Hata işareti ise referans işaret ile çıkış işareti arasındaki farktan ibarettir. Referans işaretin basamak fonksiyonu tarzında seçildiği durumlarda ise türev işlemi darbe şeklinde ani sıçramalar yaratır. Çıkış işaretinde ise

---

* Bu olaya literatürde "set point kick phenomenon" adı verilir. Daha geniş bilgi için: "Modern Control Engineering", 3th Edition, K. Ogata, Prentice Hall, 1997

bu tarz ani değişimler fiziksel sistemlerin yapısı gereği meydana gelmez. İşte bu dezavantajı ortadan kaldırmak amacı ile türev işlemi sadece çıkış işareti üzerine uygulanmıştır. Aynı zamanda analitik integral işleminin gücünü kontrolör mekanizmasına aktarmak için kural tabanından bağımsız bir entegratör sisteme dahil edilmiştir. Kontrol işaretinin analitik ifadesinin elde edilmesi için ilk olarak standart analog bir PI-D kontrolörü ele alınmalıdır. PI-D kontrolörünün genel yapısı,

$$u(t) = K_p e(t) + K_I \int e(t)dt - K_D \frac{d}{dt} y(t) \tag{7.1}$$

şeklindedir. Bizim sistemimiz ise bulanık P-ID tipinde bir kontrolör olduğu için (7.1) bağıntısı

$$u(t) = S_u \alpha F\{e, \dot{e}\} + K_I \int e(t)dt - K_D \frac{d}{dt} y(t) \tag{7.2}$$

şeklini alır. Pratik uygulamada kullanacağımız sistem bir artımsal sürücü içerdiği için (7.2) ifadesinin türevi alınarak ayrıklaştırılması gerekmektedir. (7.2) bağıntısının türevi alınırsa,

$$\frac{d}{dt}u(t) = \frac{d}{dt} S_u \alpha F\{e, \dot{e}\} + K_I e(t) - K_D \frac{d^2 y(t)}{dt^2} \tag{7.3}$$

bağıntısı elde edilir. Gerekli ayrıklaştırma işlemleri yapılırsa ve bulanık kontrolörün kural tabanı PI tipi seçilir ise,

$$\Delta u(k) = u(k) - u(k-1) = S_u \alpha F\{e, \dot{e}\} + K_I T_s e(k) - K_D \frac{y(k) - 2y(k-1) + y(k-2)}{T_s} \tag{7.4}$$

elde edilir. PI tipi bulanık kontrolörün ve öz-uyarlama mekanizmasının bulanıklaştırma üyelik fonksiyonları ile bulanık PI tipi kontrolörün çıkış üyelik fonksiyonu Şekil 4.3, Şekil 4.4 ve Şekil 4.5'deki gibi seçilmiştir. Öz-uyarlama mekanizmasının çıkışını temsil eden α ayar parametresi ise Şekil 6.2'den de görülebileceği gibi [0,1] aralığında değişen değerler almaktadır.

7.1 NGTOSTFP-ID Kontrolörüne ait Kural Tabanı Tasarımı

NGTOSTFP-ID kontrolörünün kural tabanı, optimal kontrol teorisine dayanan bir yaklaşımla tasarlanmıştır. En genel halde 2 girişli bir bulanık kontrolörü

$$u = F\{e, \dot{e}\} \tag{7.5}$$

şeklinde tanımlanır. Yani bir bulanık kontrolörün çıkışı giriş işaretleri olan hata ve hata hızının lineer olmayan bir fonksiyonudur.

Burada hata işaretini $e = y - r$ şeklinde tanımlamış olalım ve kural tabanını oluşturan kurallar,

$$IF\ e \text{ is } A_{I_0} \text{ and } \dot{e} \text{ is } A_{I_1}\ THEN\ u \text{ is } B_{F(I_0,I_1)} \tag{7.6}$$

şeklinde olsun. Burada $F : I^2 \rightarrow I$ kural türetme fonksiyonu; $I = \{-n,...,-1,0,1,...n\}$ giriş bulanık kümelerinin indisleridir. Burada amaç F kural türetme fonksiyonunun belirlenmesidir. Kontrol işareti sınırlı olarak kabul edilirse, sistem için performans ölçütü,

$$P = \sum_{k=1}^{n} \sqrt{e^2(k) + \dot{e}^2(k)} \tag{7.7}$$

şeklinde tanımlanabilir. Burada P performans ölçüt değerini; k, k. örneği ve n, toplam örnek sayısını ifade eder. Kontrolörün amacı ise sistem hatalarını orijine taşımak olsun. Bunun gerçeklenebilmesi için yapılması gereken şey ise P performans ölçütünün e ve $\dot{e}$ için minimize edilmesidir. P nin e ve $\dot{e}$ için kısmi türevleri sırası ile,

$$\frac{\partial P}{\partial e(k)} = \frac{e(k)}{\sqrt{e^2(k) + \dot{e}^2(k)}} \tag{7.8}$$

$$\frac{\partial P}{\partial \dot{e}(k)} = \frac{\dot{e}(k)}{\sqrt{e^2(k) + \dot{e}^2(k)}} \tag{7.9}$$

şeklindedir. Kontrolörün, optimal performansı verebilmesi için, performans ölçütünün negatif gradyeni doğrultusunda davranışını sürdürmesi gerekmektedir. Bunun için P nin negatif gradyeni alınırsa,

$$-|\nabla P| = \left[-\frac{|e(k)|}{\sqrt{e^2(k) + \dot{e}^2(k)}} \quad -\frac{|\dot{e}(k)|}{\sqrt{e^2(k) + \dot{e}^2(k)}} \right]^T \tag{7.10}$$

elde edilir. Optimal kontrol teorisine dayanılarak bulanık kontrolörün üreteceği kontrol işareti k. örnek için,

$$u(k) = S_u(-|\nabla P|)^T \begin{bmatrix} e(k) \\ \dot{e}(k) \end{bmatrix} = S_u \left(-\frac{|e(k)|e(k)}{\sqrt{e^2(k) + \dot{e}^2(k)}} - \frac{|\dot{e}(k)|\dot{e}(k)}{\sqrt{e^2(k) + \dot{e}^2(k)}} \right) \tag{7.11}$$

şeklini alır. Eğer (7.11) bağıntısında $F\{.\}$ fonksiyonu için tanımlanan kısımda, hata ve hatanın

değişimleri yerine, bu işaretlerin indislerini kullanacak olursa ve kontrol işaretinin sınırlı olduğunu kabul edilirse $F\{.\}$ fonksiyonu için,

$$F(I_0, I_1) = -sat\left(n, -\frac{|I_0|I_0 + |I_1|I_1}{\sqrt{I_0^2 + I_1^2}}\right) \tag{7.12}$$

ifadesi kullanılabilir. Burada $sat(n, x)$ fonksiyonu aşağıdaki özelliklere sahiptir. "–" işaretinin (7.12) bağıntısında kullanılmasının nedeni, hatanın bu çalışmada $e = r - y$ şeklinde tanımlanmasıdır.

$sat(n,x) = n$ Eğer $x \geq n$

$sat(n,x) =$ yuvarla (x) Eğer $-n < x < n$

$sat(n,x) =$ -n Eğer $x \leq$ -n . (7.13)

Burada, n uzayın boyutunu gösteren uç indistir. Bu çalışmada giriş ve çıkış değişkenlerinin herbirisi için 7 adet bulanık alt küme tanımlanmıştır. Bunlar NB = Negatif Büyük, NM = Negatif Orta, NS = Negatif Küçük, Z = Sıfır, PS = Pozitif Küçük, PM = Pozitif Orta, PB = Pozitif Büyüktür.

Yukarıda tanımlanan optimizasyon işlemini gerçeklemek amacı ile her bulanık kümeye birer indis karşı düşürülecektir. Bunlar NB = -3, NM = -2, NS = -1, Z = 0, PS = 1, PM =2 ve PB = 3 şeklindedir. 7 * 7 = 49 adet kural için tüm olasılıklar (7.12)'de yerine konularak Çizelge 7.1'deki kural tabanı elde edilir. Örneğin IF e is NB and Δe is PM THEN u is NS şeklindeki bir kural şu şekilde elde edilir (Kirk, 1970), (Yeh, 1999).

$$F(-3,2) = -sat\left(3, -\frac{|-3|*(-3) + |2|*2}{\sqrt{(-3)^2 + 2^2}}\right) = -1 \approx NS \tag{7.14}$$

Çizelge 7.1 NGTOSTFP-ID kontrolörünün kural tabanı

Δ e/e	NB	NM	NS	Z	PS	PM	PB
NB	NB	NB	NB	NB	NB	NS	Z
NM	NB	NB	NM	NM	NS	Z	PS
NS	NB	NM	NS	NS	Z	PS	PB
Z	NB	NM	NS	Z	PS	PM	PB
PS	NB	NS	Z	PS	PS	PM	PB
PM	NS	Z	PS	PM	PM	PB	PB
PB	Z	PS	PB	PB	PB	PB	PB

7.2 NGTOSTFP-ID Kontrolörünün Ayar Parametrelerinin Optimizasyonu

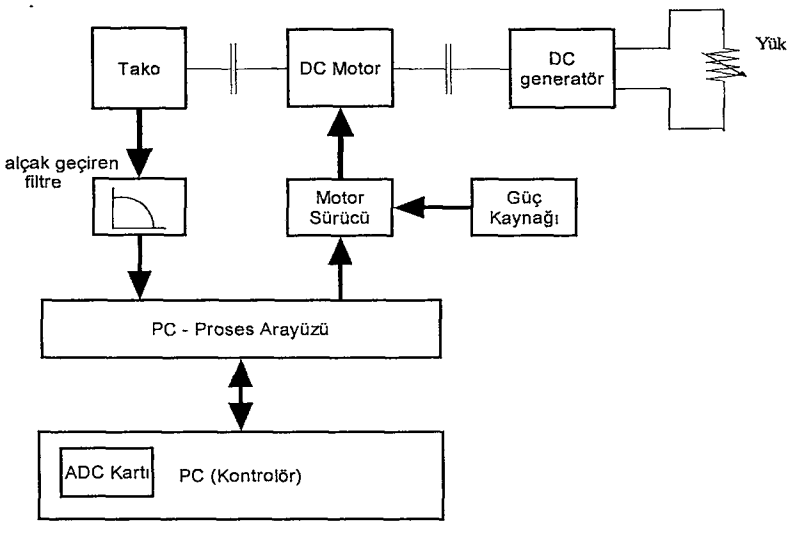
NGTOSTFP-ID kontrolörünün giriş ölçekleme faktörleri ile türev ve integral katsayıları, sistemin YSA modeli üzerinden genetik ortamda optimize edilecektir. Bunun için öncelikle türev ve integral işlemlerine ait bloklar devre dışı bırakılır. Geriye kalan kontrolör aslında altıncı bölümde ele alınan NGTOSTF-PI kontrolörü ile aynıdır. Buradaki tek fark ise kural tabanının optimal kontrol teorisine dayanılarak tasarlanmış olmasıdır. Ayar işlemi öncelikle giriş ölçekleme faktörleri üzerinde yapılır. Çıkış ölçekleme faktörü olan S_u değeri kabul edilebilir bir geçici rejim cevabı verecek şekilde kabaca ayarlanır. Daha sonra giriş ölçekleme faktörleri olan S_e ve S_d için çalışma bölgeleri belirlenir. Bu işlemi takiben sistemin bu ölçekleme faktörü uzayına göre eğitilebilmesi için YSA modeli oluşturulması kısmına geçilir. Giriş ölçekleme faktörlerine ait uzaydan seçilen bir grup değer çifti için performans ölçütleri bir dosyada saklanır. Bu değerler sistemin YSA modelinin eğitilmesi esnasında kullanılır. Bir sonraki aşamada sistemin YSA modeli üzerinden optimal değerlere yakın giriş ölçekleme faktörleri Kısım 4.2'de açıklanan yöntem yardımı ile belirlenir.

İkinci aşama, NGTOSTFP-ID kontrolörünün integral ve türev katsayılarının optimizasyonu aşamasıdır. Bu aşamada, sisteme integral ve türev kontrolörleri dahil edilir. Giriş ölçekleme faktörleri için yukarıda açıklanan yöntem ile bulunan optimale yakın ölçekleme faktörleri kullanılır. Çıkış ölçekleme faktörü olan S_u değeri değiştirilmeden integral ve türev katsayılarının çalışma uzayları deneme yanılma yöntemi ile belirlenir. Belirlenen uzaydan seçilen değer çiftleri ile sistemin YSA modeli elde edilir. Elde edilen ağ yapısı türev ve integral katsayılarını giriş; *IAE*, *ITAE* ve %Aşım değerlerini çıkış olarak kabul etmektedir. Üçüncü bölümde anlatılan PI kontrolörüne ait katsayılarının genetik optimizasyonu esnasında kullanılan algoritmaya benzer bir algoritma (P katsayısı yerine K_d kullanılarak) ile optimale yakın integral ve türev katsayıları belirlenir.

8. PRATİK ÇALIŞMA

Bu bölümde, 3, 4, 5, 6 ve 7. bölümlerde tasarımları anlatılan değişik Nöral-Genetik tabanlı optimal kontrolörlerin performanslarını karşılaştırabilmek amacı ile tasarlanan deneysel amaçlı kontrol sisteminden bahsedilecektir.

Deneysel amaçlı test sistemi 3 adet DC servomotordan oluşmaktadır. Sistemin blok diyagramı Şekil 8.1'de, kullanılan motorlara ait parametreler ise Çizelge 8.1'de gösterilmiştir. Bu motorlar aynı mil ekseni üzerine gelecek şekilde kaplinler yardımı ile monte edilmiştir. Motorlardan bir tanesi kontrol ettilen prosesi temsil ederken, diğer ikisi sırası ile yük ve tako-generatör olarak kullanılmaktadır. Yük olarak kullanılan motorun çıkışı herbiri 12V / 21W'lık 4 adet ampulden oluşan seri bağlanmış bir yük sistemini beslemektedir. Yükler, üzerlerinden geçen akımın karesi ile orantılı güç çektikleri için lineer olmayan bir yükü temsil etmektedir. Tako-generatör olarak kullanılan motorun çıkışı nominal devir sayısında yaklaşık 50V seviyesinde bir gerilim ürettiği için bu değer bir direnç takımı üzerinden bölünmekte ve alçak geçiren birinci mertebeden bir filtre üzerinden ADC'ye ulaştırılmaktadır. DC motor sistemini besleyen gerilim, sürücü devresinden bağımsız bir doğrultucu yardımı ile elde edilmektedir. DC motorun besleme gerilimini sağlayan güç devresi Şekil Ek 8.1'de gösterilmiştir. DC Servomotor hızı ise bir PWM (Pulse Width Modulation) kontrolörü tarafından ayarlanmaktadır. PWM kontrolörü, Şekil Ek 8.1'de gösterilmiş olan Mosfeti iletime sokup-çıkararak DC motor üzerine düşen gerilimin ortalama değerini ayar etmektedir.



Şekil 8.1 Deneysel kontrol sisteminin blok diyagramı

PWM sinyali ise tasarlanan kontrolör tarafından sisteme her örnekleme periyodunun başında

aktarılmaktadır. Örnekleme frekansı olarak 2kHz seçilmiştir. Tüm kontrol algoritmaları Borland C++ V3.1 ile Pentium 200MHz bir bilgisayarda programlanmıştır. Kontrolör için geri besleme elemanı olarak bir ADC ISA kart bilgisayara monte edilmiştir. Bilgisayarın ürettiği kontrol işareti paralel port üzerinden sürücü devreye ulaştırılmaktadır. DC Servomotor sürücü devresini bilgisayardan yalıtmak amacı ile opto kuplörler kullanılmıştır. Sürme ve bilgisayar – sürücü ara yüz devresi Şekil Ek 8.3'de gösterilmiştir. 2kHz örnekleme frekansını ayarlayan zamanlayıcı devresi ise Şekil Ek 8.2'de gösterilmiştir. Zamanlayıcı devresi, örnekleme anlarını PC'ye paralel port üzerinden kesme şeklinde iletmektedir.

Çizelge 8.1 Kullanılan DC Servomotorlara ait parametreler

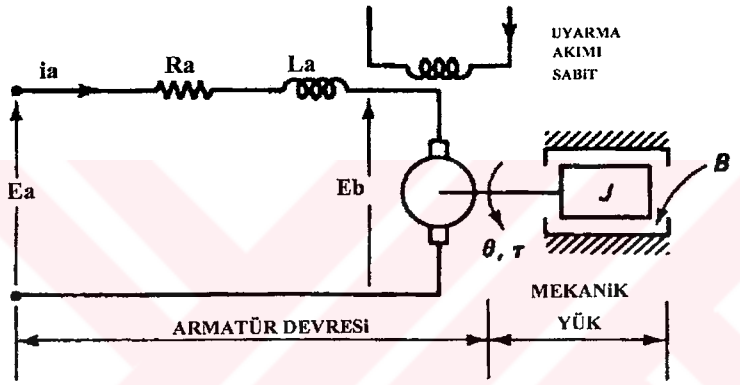
Nominal Gerilim	U	52V DC
Nominal Akım	I	2.2 A
Nominal Devir	n	3000 d/dk
Atalet momenti	J	0.0000421kgm^2
Sürtünme Kat.	β	0.000111408Nmsn/rad
Moment Katsayısı	K_m	0.14Nm/A
Hız Katsayısı	K	0.14Vsn/rad
Rotor Direnci	R_a	2.9Ω

8.1 DC Servomotorlar ve Matematiksel Modeli

Sanayide çeşitli doğru akım motoru kullanılır. Bu motorlardan, servo sistemlerde kullanılanlarına DC servomotorlar adı verilir. DC servomotorlarda rotor eylemsizlik momenti ile zaman sabitleri oldukça küçüktür. Düşük güçlü DC servomotorlar piyasada genellikle bilgisayar kontrollü cihazlarda (disket sürücüler, teyp sürücüleri, yazıcılar, kelime işlemciler, tarayıcılar vs.) kullanılır. Orta ve büyük güçlü servomotorlar ise sanayide genellikle robot sistemler ile sayısal kontrollü hassas diş açma tezgahlarında kullanılır.

DC servomotorlarda alan sargıları rotor sargılarına seri veya paralel bağlanır. Endüi sargılarından bağımsız olarak uyartılan alan sargılarının akısı, endüi sargılarından geçen akımın fonksiyonu değildir. Bazı DC servomotorlarda manyetik alan sargısı yerine sabit bir mıknatıs bulunur. Bu tip motorlarda manyetik akı sabittir. Uyarma sargıları endüiden bağımsız olan veya sabit mıknatısla uyartılan motorlarda hız kontrolü, endüi gerilimi ile yapılabilir. Bu tip kontrol yöntemine **endüi kontrol yöntemi** denir.

Uyarma sargılarının yarattığı akı ile yapılan kontrollerde ise endüi akımı sabit tutulur. Statorda bulunan uyartım sargılarının yarattığı akının kontrolü ile hız ayarlanır. Bu tip motorlara **alan kontrollü** motorlar denir. Fakat rotor sargılarından geçen akımın sabit tutulabilmesi ise ciddi bir problemdir. Zira rotor akımı yükün ve kaynağın birer fonksiyonudur. Endüi kontrollü motorlara göre, alan kontrollü motorların zaman sabitleri daha büyüktür. DC servomotorlar genellikle elektronik hareket kontrolörleri adı verilen servo sürücüler ile kontrol edilirler. Servo sürücüler servomotorun hareketini kontrol eder. Kontrol edilen büyüklükler çoğu zaman noktadan noktaya konum kontrolü, hız kontrolü ve ivme kontrolüdür. PWM tekniği adı verilen darbe genişlik modülasyonu genellikle robot kontrol sistemlerinde, sayısal kontrol sistemlerinde ve konum kontrolörlerinde kullanılırlar.



Şekil 8.2 DC Servomotorun elektro-mekanik eşdeğer devresi (Phillips ve Harbor, 1988)

Şekil 8.2'deki servomotor sistemini göz önüne alırsak, uyarma akımının sabit olduğu varsayımı altında motorun indüklediği moment,

$$T = K_m i_a \tag{8.1}$$

şeklindedir. Burada K_m motor moment sabitidir ve i_a rotor akımıdır. Dikkat edilecek olursa akım ifadesinin işareti değişecek olursa moment ifadesinin de yönü değişecektir. Rotor hareket halindeyken manyetik akı ve açısal hızla orantılı olarak rotor sargılarında belli bir gerilim indüklenir. Bu gerilim ifadesi;

$$E_b = K\frac{d\theta}{dt} \tag{8.2}$$

şeklindedir. Burada E_b ifadesi ters elektromotor kuvveti; θ, motorun dönme açısını; K, ise ters elektromotor kuvvet katsayısını yada diğer bir deyişle hız katsayısını sembolize etmektedir. DC servomotorunun hızı, endüi gerilimi E_a nin ayarlanması sonucu elde

edilmektedir. Rotor devresi için diferansiyel denklem takımı;

$$L_a \frac{di_a}{dt} + R_a i_a + E_b = E_a, \tag{8.3}$$

$$L_a \frac{di_a}{dt} + R_a i_a + K\frac{d\theta}{dt} = E_a, \tag{8.4}$$

şeklindedir. Aynı motora ait mekanik sistem için denklem takımı ise;

$$J_0 \frac{d^2\theta}{dt^2} + \beta_0 \frac{d\theta}{dt} = T = K_m . i_a, \tag{8.5}$$

şeklindedir. Yukarıda (8.3), (8.4), (8.5) bağıntılarında adı geçen terimler:

L_a (H) = Rotor devresi toplam endüktansı,

J_0 (kgm^2) = Yük, motor ve tako generatörün toplam eylemsizlik momenti,

β_0 (*Nmsn/rad*) = Yük, motor ve tako generatörün toplam sürtünme katsayısı,

R_a (Ω) = Rotor devresi toplam direnci,

i_a (A) = Rotor devresi akımı

şeklindedir. Yukarıda tanımlanan (8.1)-(8.5) ifadeleri düzenlenirse ve çıkış büyüklüğü olarak motor açısal hızı, giriş büyüklüğü olarak da besleme gerilimi alınırsa, motor transfer fonksiyonu aşağıdaki şekli alır.

$$\frac{\omega_{motor}(s)}{E_a(s)} = \frac{K_m}{J_0 L_a s^2 + (\beta_0 L_a + J_0 R_a)s + (\beta_0 R_a + K_m K)}. \tag{8.6}$$

Elektriksel zaman sabitlerinin mekaniksel zaman sabitlerine oranla çok çok küçük olduğu kabul edilirse (8.7) bağıntısı elde edilir. (Phillips ve Harbor, 1988).

$$\frac{\omega_{motor}(s)}{E_a(s)} = \frac{K_m}{J_0 R_a s + \beta_0 R_a + K_m K} \tag{8.7}$$

9. SONUÇLAR ve TARTIŞMA

9.1 Sonuçlar

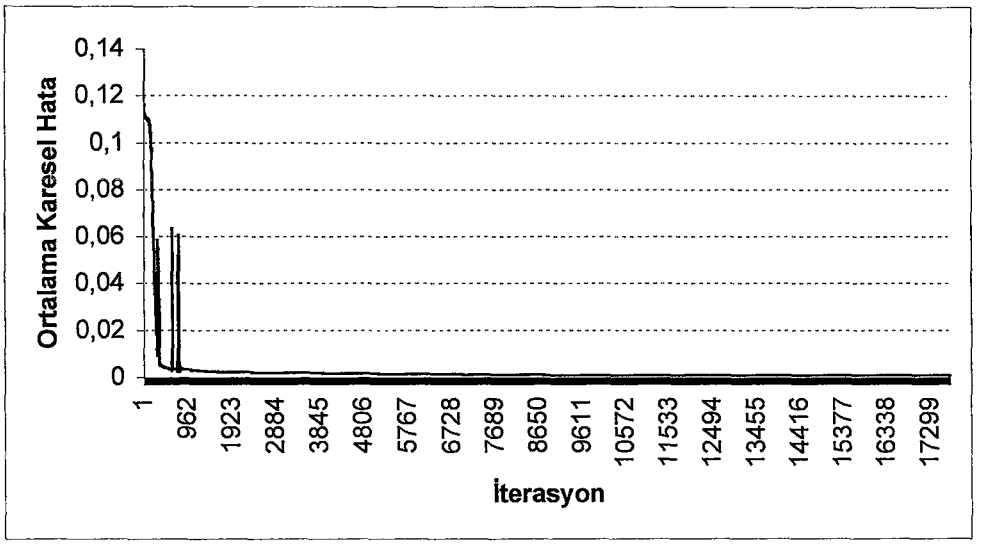
Bu bölümde, önceki kısımlarda tasarımları yapılan değişik tipteki kontrolörlerin 8. bölümde ayrıntıları verilen sistem üzerindeki performansları değerlendirilmektedir. Kontrolörler birbirleri ile karşılaştırılmadan önce bağımsız olarak performansları sunulmaktadır. Daha sonra birbirleri ile karşılaştırmalı performans analizleri yapılarak, parametreleri bilinmeyen bir DC servo sistem için optimal bulanık kontrolör belirlenecektir. Çalışmada, tüm performans karşılaştırmaları 3000 örnek üzerinden tam yük altında yapılmaktadır. Performans kriterleri basamak cevabı, rampa cevabı, IAE, ITAE ve %Aşım'dır. Basamak cevabı, sistemin ani referans değişikliklerine karşı duyarlılığını; rampa cevabı, sistem çıkışının referansı izleme başarısını; IAE, geçici rejim performansını ve ITAE, kararlı hal performansını sorgulamaktadır.

9.1.1 NGTOPI Kontrolörüne ait Deneysel Sonuçlar

Yapay sinir ağı 24 giriş / çıkış çifti ile eğitilmiştir. Eğitme esnasında 2 girişli – 3 çıkışlı 2 gizli katmana ve her gizli katmanda 5 hücreye sahip bir ağ yapısı kullanılmıştır. 15000. iterasyondan sonra eğitme durdurulmuştur. 15000. iterasyonda elde edilen ortalama karesel hata 0.000742 dir. Ortalama karesel hatanın iterasyonlara göre değişimi Şekil 9.1'de, YSA mimarisi ise Şekil 9.2'de gösterilmiştir. Daha sonra eğitilen ağ üzerinden optimal K_p ve K_I değerleri Genetik Algoritma yardımı ile bulunmuştur. Genetik optimizasyon işlemi esnasında elde edilen jenerasyon başına toplam uyumluluk değerinin değişimi Şekil 9.3'de gösterilmiştir.

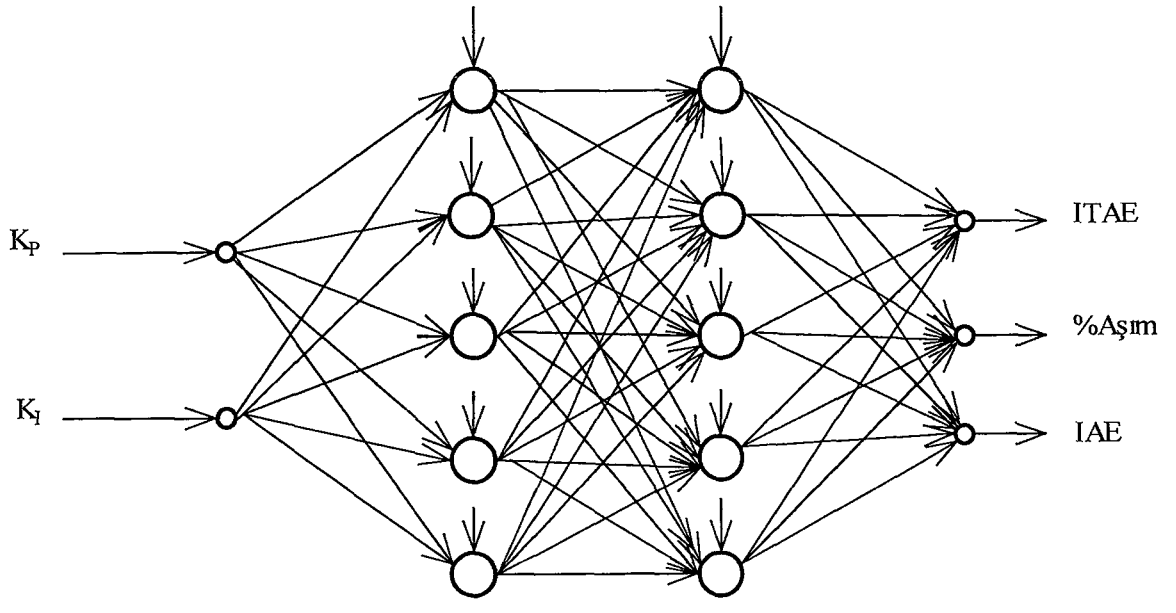
Çizelge 9.1 NGTOPI kontrolörüne ait parametreler ve deneysel sonuçlar

P_c	0.95
P_m	0.01
Popülasyon büyüklüğü	30
Maksimum Jenerasyon sayısı	1000
Optimal K_p	0.096
Optimal K_I	19
IAE	90007
ITAE	9610
% Aşım	10

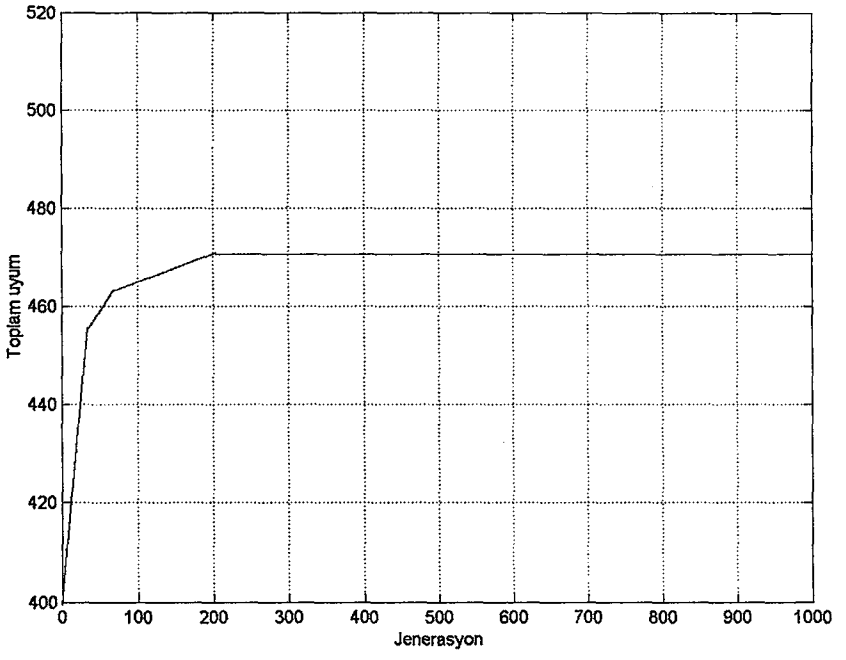


Şekil 9.1 Sistemin YSA modelinin eğitimine ait ortalama karesel hatanın iterasyonlara göre değişimi

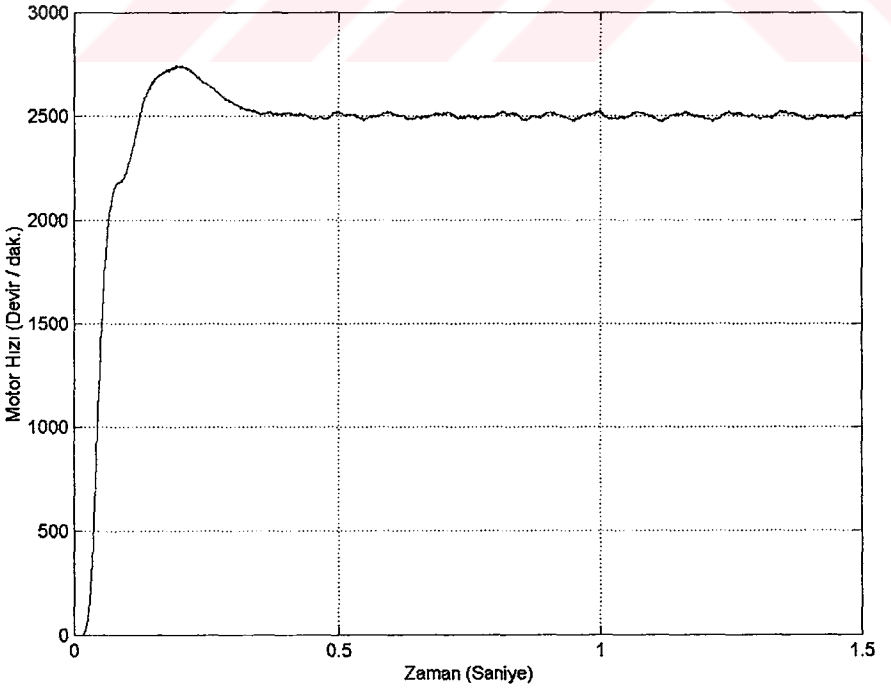
Genetik optimizasyon işlemi esnasında kullanılan parametreler, optimal K_p ve K_I değerleri ile bu değerlere karşılık gelen IAE, ITAE ve %Aşım değerleri Çizelge 9.1'de sunulmuştur. Bulunan optimal kontrolör parametreleri ile gerçekleştirilen deneylerde sistemin 2500 d/dk.'lık basamak şeklindeki referansa cevabı Şekil 9.4'de; hata işaretinin değişimi Şekil 9.5'de; kontrolör işaretinin değişimi Şekil 9.6'da ve rampa şeklindeki referansa karşılık motor çıkış hızının değişimi Şekil 9.7'de gösterilmiştir.



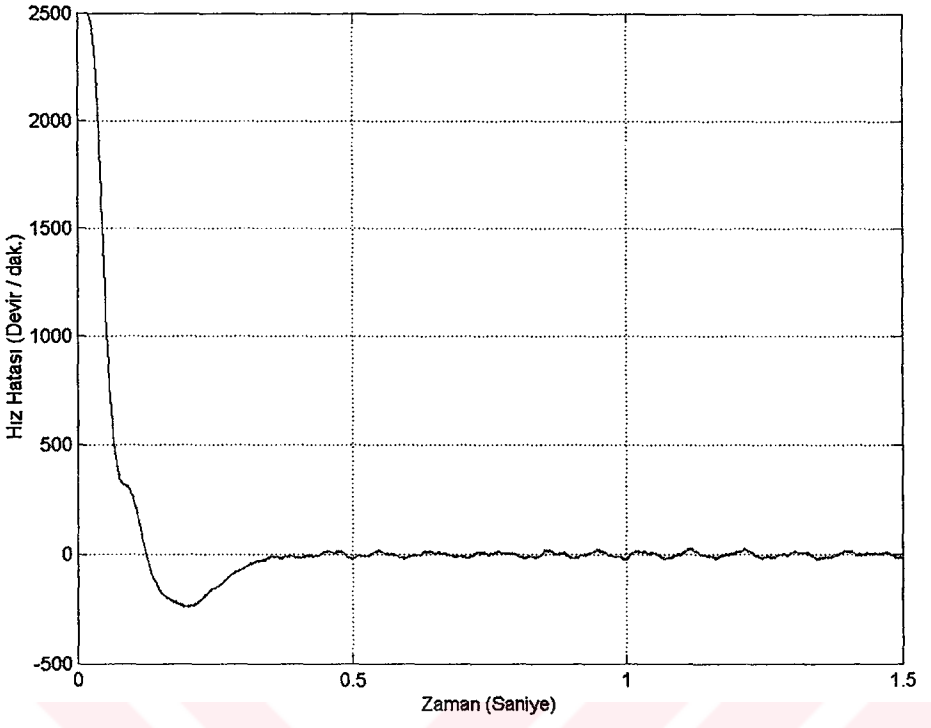
Şekil 9.2 Sistemin YSA mimarisi



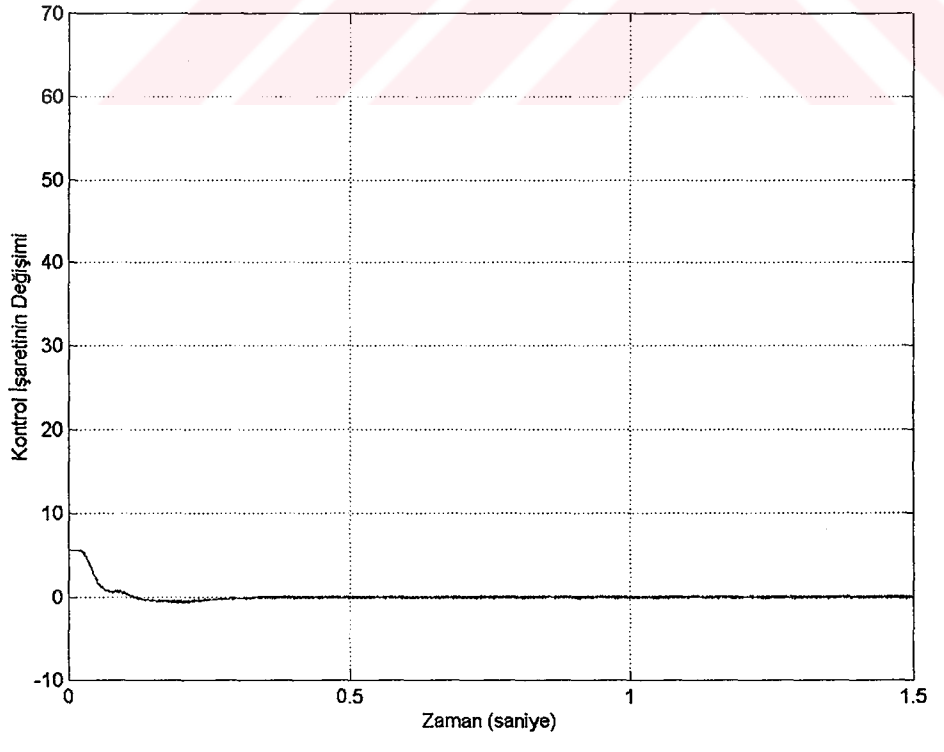
Şekil 9.3 NGTOPI kontrolörüne ait genetik optimizasyon aşamasında jenerasyon başına toplam uyumluluk değişimi



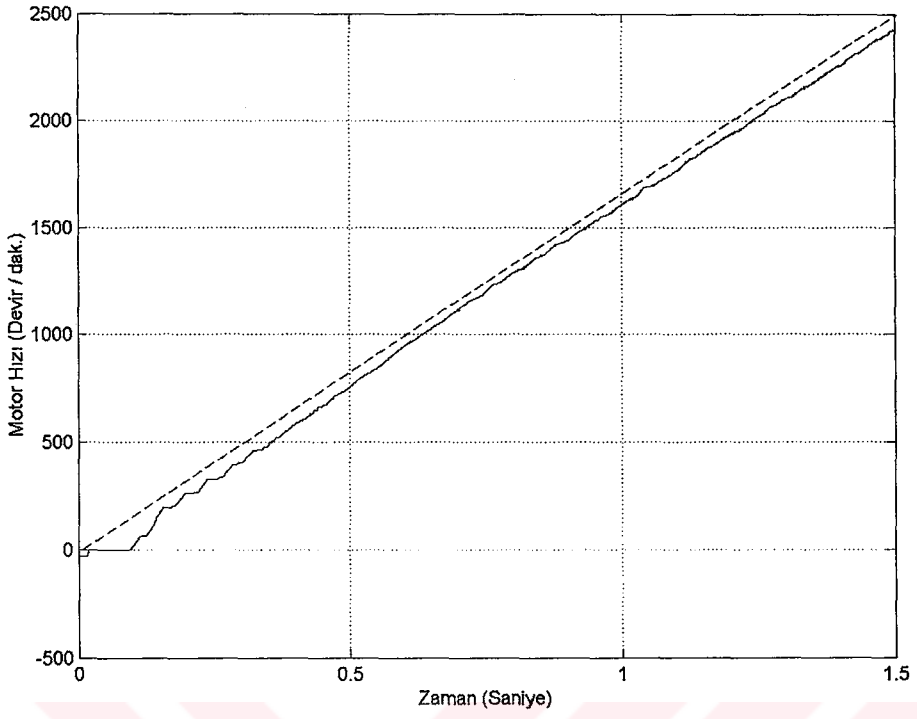
Şekil 9.4 NGTOPI kontrolörünün basamak şeklindeki referans cevabı



Şekil 9.5 NGTOPI kontrolörüne ait hız hatasının değişimi



Şekil 9.6 NGTOPI kontrolörüne ait kontrol işaretinin değişimi

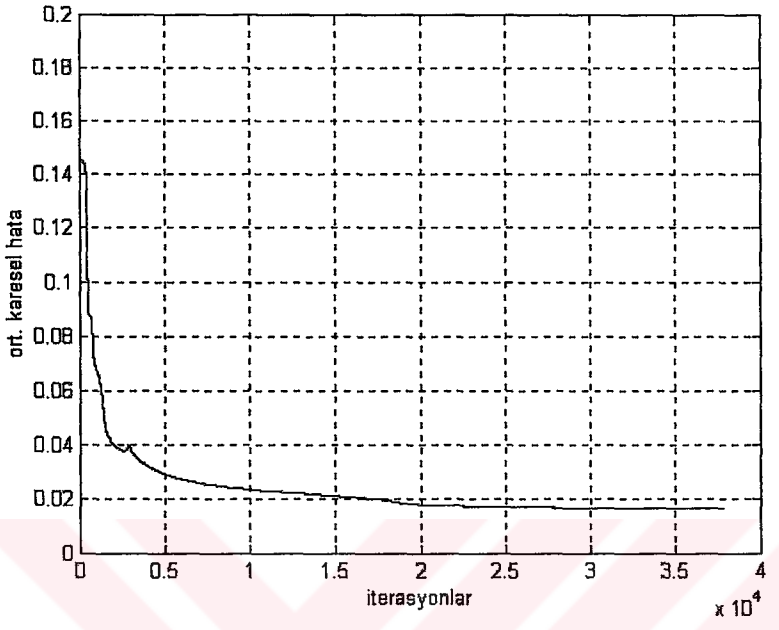


Şekil 9.7 NGTOPI kontrolörü için motorun rampa şeklinde değişen referansa (---) verdiği cevap (—)

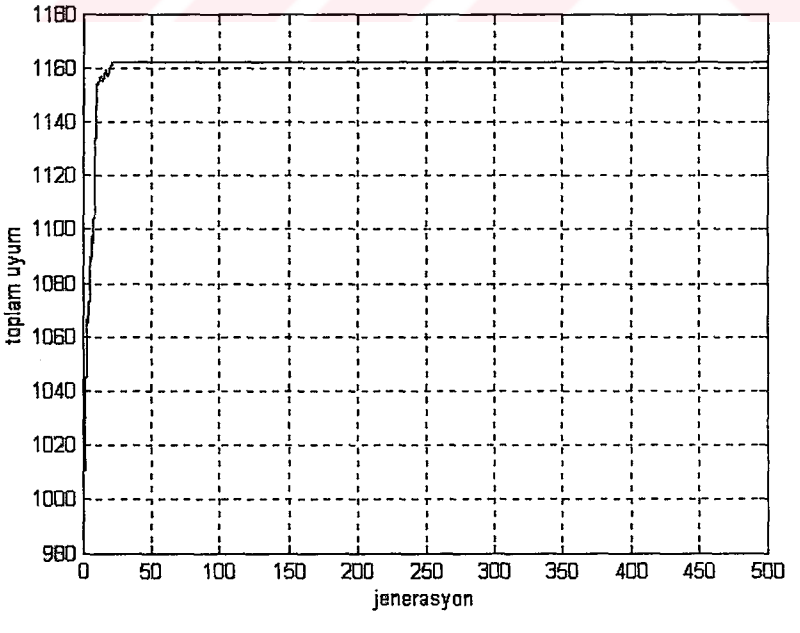
9.1.2 NGTOFPI Kontrolörüne ait Deneysel Sonuçlar

Sistemi modelleyen YSA, 25 giriş ölçekleme faktörü çifti için uygun ITAE, IAE ve %Aşım değerlerini verecek şekilde eğitildi. Eğitim işlemi sırasında 2 girişli 3 çıkışlı 1 gizli katman ve bu gizli katmanda 5 hücresi bulunan bir ağ yapısı kullanıldı. Eğitme işlemi için momentum kuralına dayanan hatanın geriye yayılımı algoritması kullanıldı. Eğitme işlemi 30000 iterasyon sürdürüldü ve ortalama karesel hatanın %1.8 civarına yakınsadığı görüldü. Eğitme süreci esnasında elde edilen ortalama karesel hata değişimi Şekil 9.8'de gösterilmiştir. Daha sonra eğitilen ağ yardımı ile giriş ölçekleme faktörlerinin ayarı işlemine geçildi. Optimal ölçekleme faktörlerinin belirlenmesinde özellikleri Çizelge 9.2'de belirtilen Genetik Algoritma kullanıldı. Genetik Algoritmanın optimal ölçekleme faktörlerini yaklaşık 20. iterasyonda belirlediği Şekil 9.9'dan görülmektedir. Jenerasyon başına toplam uyum değerleri Şekil 9.9'da gösterilmiştir. Genetik Algoritma yardımı ile bulunan optimal ölçekleme faktörlerinin değerleri; bu değerlerin kullanılması ile elde edilen *IAE, ITAE* ve *%Aşım* değerleri ile NGTOFPI kontrolörüne ait diğer parametreler yine Çizelge 9.2'de sunulmuştur. Bulunan optimal ölçekleme faktörleri kullanılarak gerçekleştirilen deneylerde sistemin

2500d/dk.'lık basamak fonksiyonu şeklindeki referansa verdiği cevap Şekil 9.10'da; hata işaretinin değişimi Şekil 9.11'de; kontrolör işaretinin değişimi Şekil 9.12'de ve rampa şeklindeki referansa karşılık motor çıkış hızının değişimi Şekil 9.13'de gösterilmiştir.



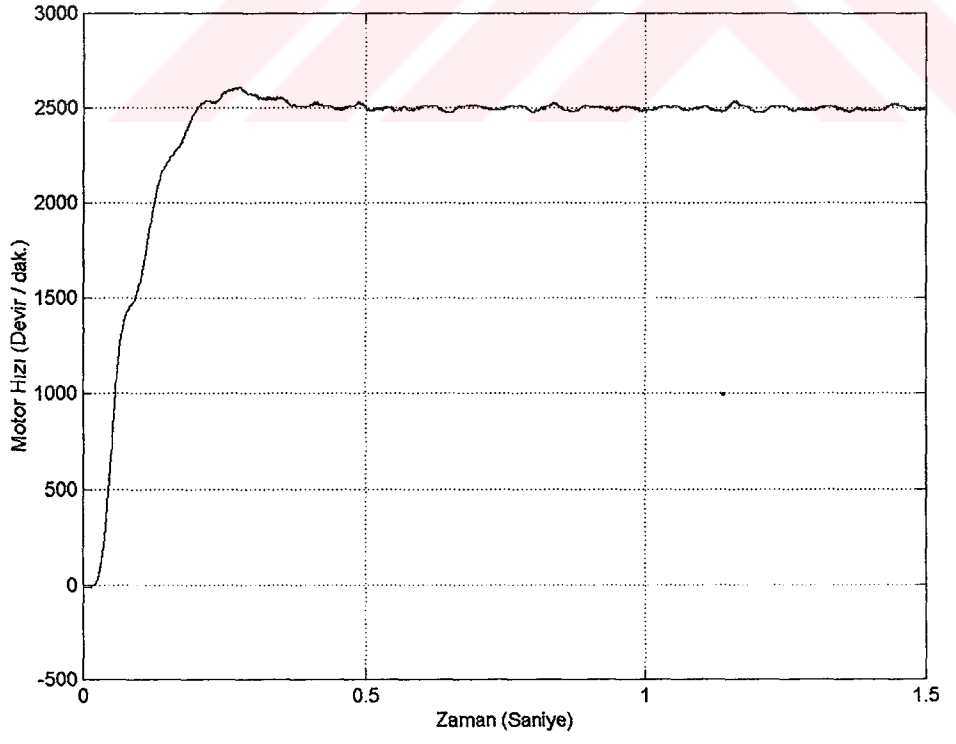
Şekil 9.8 NGTOFPI kontrolörünün tasarımı esnasında,YSA eğitimi süresince ortalama karesel hatanın değişimi



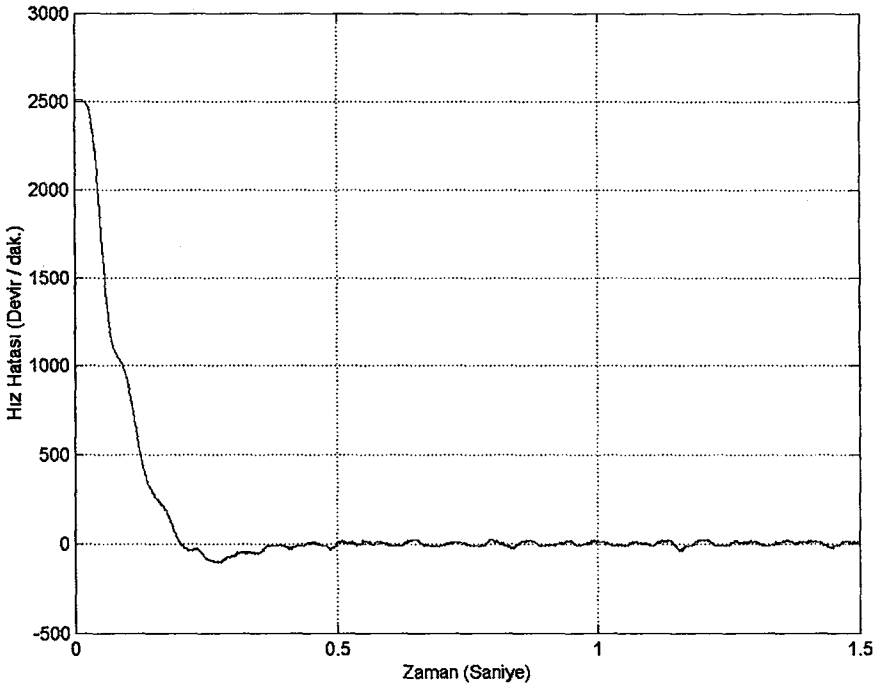
Şekil 9.9 NGTOFPI kontrolörünün genetik optimizasyon işlemi esnasında jenerasyonlara göre toplam uyum değişimi

Çizelge 9.2 NGTOFPI kontrolörüne ait parametreler ve performans sonuçları

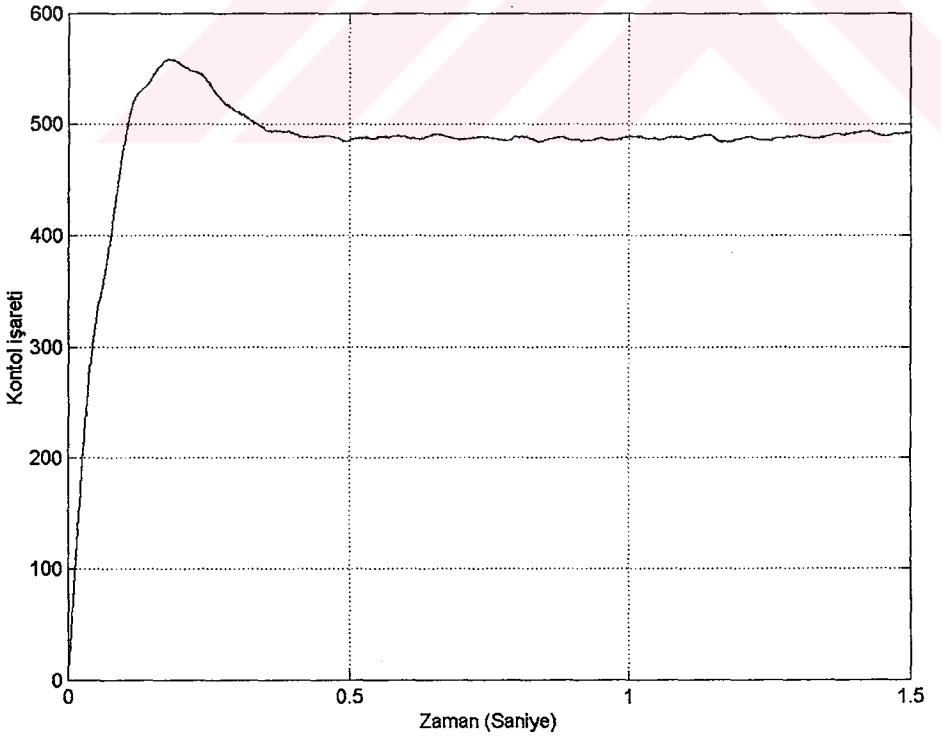
P_c	0.87
P_m	0.001
Popülasyon Büyüklüğü	30
Maksimum Jenerasyon Sayısı	500
Optimal S_e	0.0433
Optimal S_d	0.000865
S_u	400
U_m	80d/dk
U_{dm}	80d/dk
U_{out}	20
T_s	0.0005sn
IAE	111618
ITAE	15033
%Aşım	%4.8



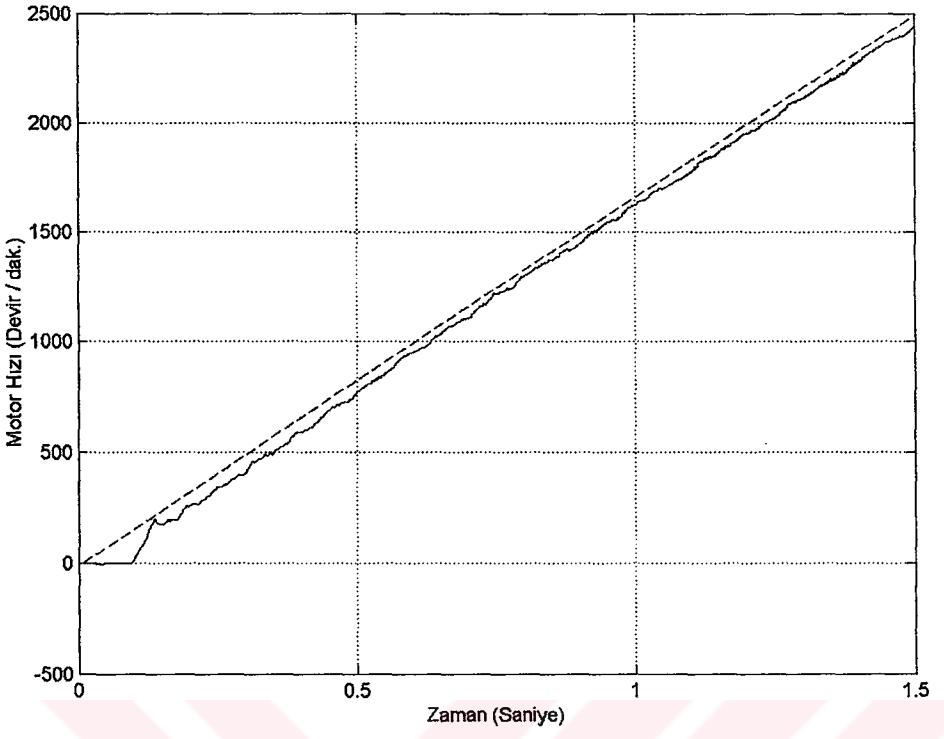
Şekil 9.10 NGTOFPI kontrolörünün basamak şeklindeki referansa verdiği cevap



Şekil 9.11 NGTOFPI kontrolörüne ait hız hatasının değişimi



Şekil 9.12 NGTOFPI kontrolörüne ait kontrol işaretinin değişimi



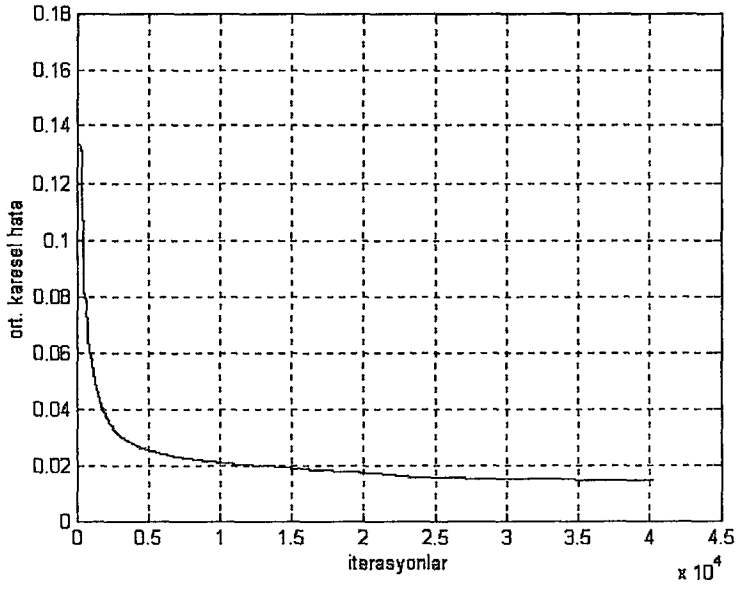
Şekil 9.13 NGTOFPI kontrolörü için, motorun rampa şeklinde değişen referansa (---) verdiği cevap (—)

9.1.3 NGTOFPID Kontrolörüne ait Deneysel Sonuçlar

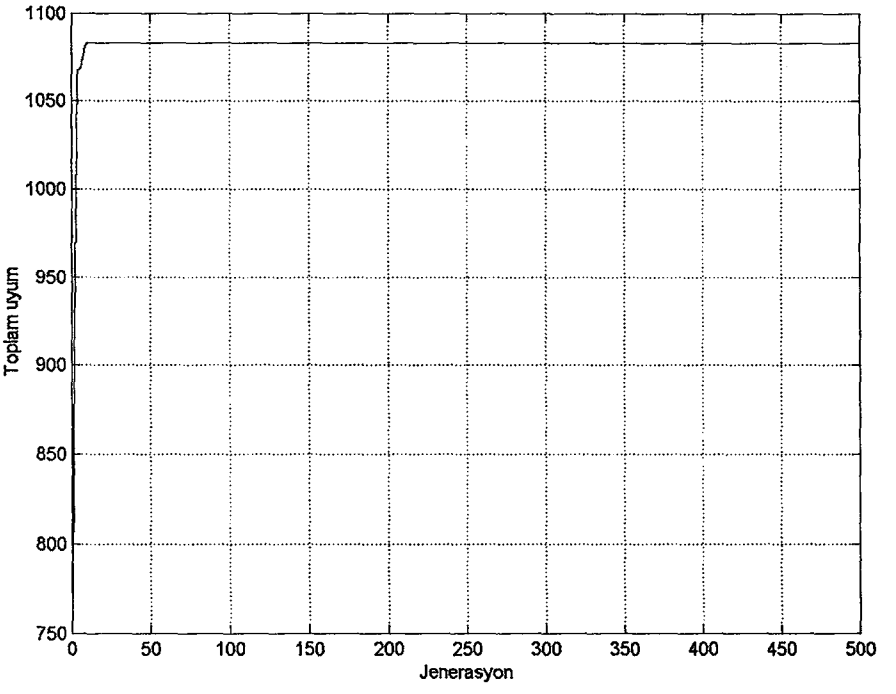
Sistemin YSA modeli, 25 giriş / çıkış çifti ile eğitilmiştir. Eğitme esnasında 2 girişli – 3 çıkışlı, 1 gizli katmana ve ilgili gizli katmanda 5 hücreye sahip ağ yapısı kullanılmıştır. Eğitim algoritması olarak momentum yöntemine dayanan hatanın geriye yayılımı algoritması kullanılmıştır. YSA 40000 iterasyon boyunca eğitilmiş ve ortalama karesel hatanın ilgili ağ mimarisi için 0.015 civarında seyrettiği gözlemlenmiştir. Ortalama karesel hatanın iterasyonlara göre değişimi Şekil 9.14'de gösterilmiştir. Daha sonra giriş ölçekleme faktörlerinin optimizasyonu işlemine geçilmiştir. Bu işlem için Çizelge 9.3'de parametreleri verilen Genetik Algoritma kullanılmıştır. Genetik optimizasyon işlemi esnasında elde edilen jenerasyon başına toplam uyumluluk değişimi Şekil 9.15'de; optimal ölçekleme faktörlerinin kullanılması sonucunda elde edilen performans ölçütleri Çizelge 9.3'de, sistem çıkışının basamak şeklindeki 2500d/d.'lık referansa verdiği cevap Şekil 9.16'da; hata işaretinin değişimi Şekil 9.17'de, kontrol işaretinin değişimi Şekil 9.18'de ve rampa şeklindeki referansa karşılık motor çıkış hızının değişimi Şekil 9.19'da gösterilmiştir.

Çizelge 9.3 NGTOFPID kontrolörüne ait parametreler ve deneysel sonuçlar

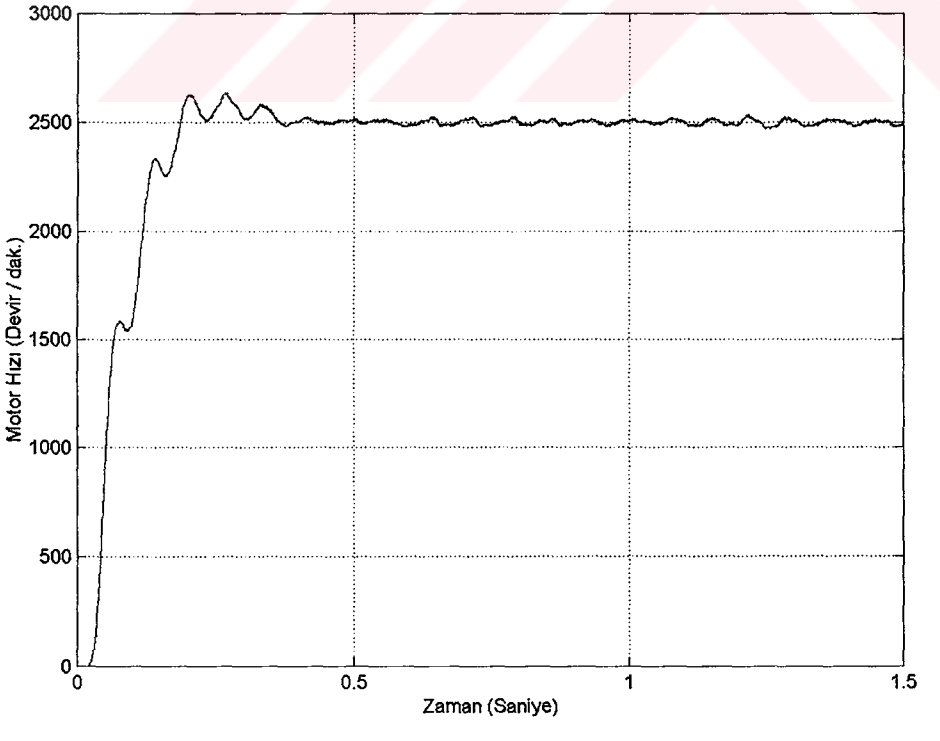
P_c	0.87
P_m	0.001
Popülasyon Büyüklüğü	30
Maksimum Jenerasyon Sayısı	500
Optimal S_e	0.0401
Optimal S_d	0.00090
K_1	450
K_2	0.1
U_m	80d/dk
U_{dmt}	80d/dk0
$U_{out\ 1}$	20
$U_{out\ 2}$	10
T_s	0.0005sn
IAE	100946
ITAE	11294
%Aşım	%4.5



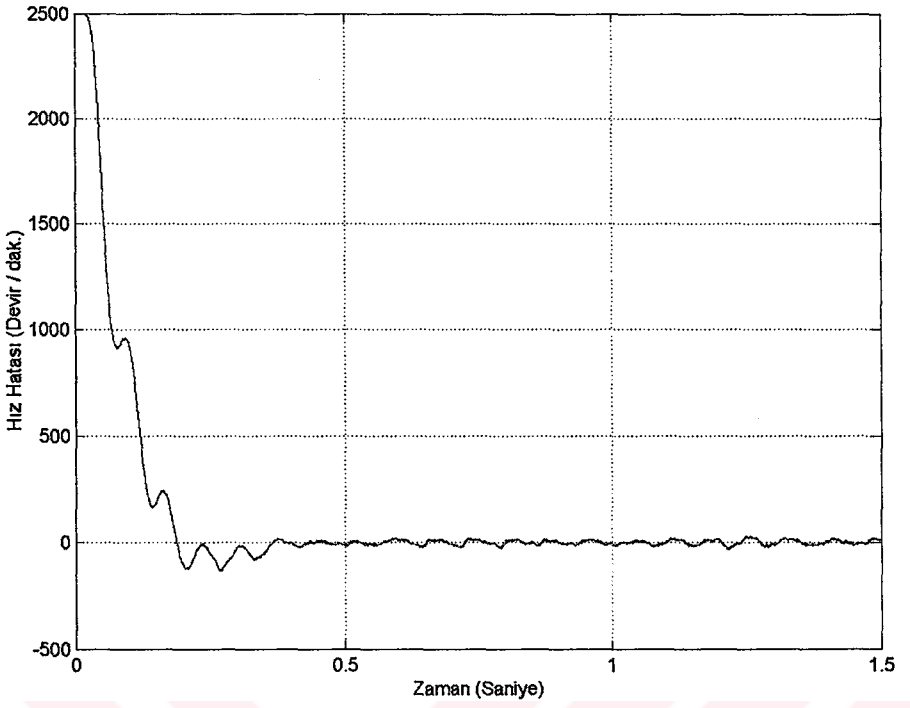
Şekil 9.14 NGTOFPI kontrolörünün tasarımı esnasında,YSA eğitimi süresince ortalama karesel hatanın değişimi



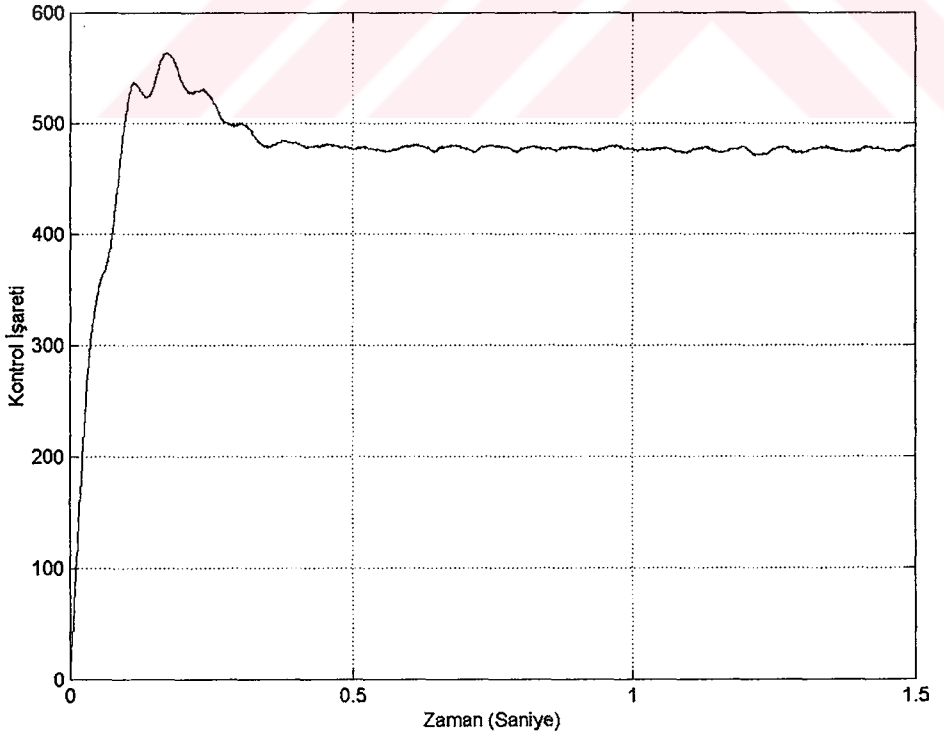
Şekil 9.15 NGTOFPID kontrolörünün genetik optimizasyon işlemi esnasında jenerasyonlara göre toplam uyum değişimi



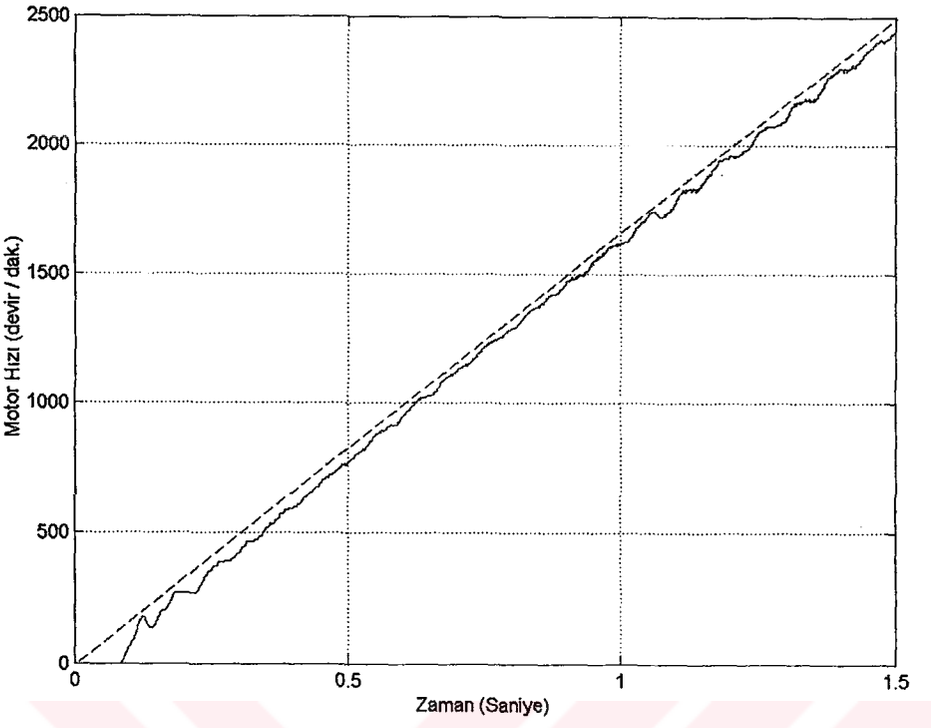
Şekil 9.16 NGTOFPID kontrolörünün basamak şeklindeki referansa verdiği cevap



Şekil 9.17 NGTOFPID kontrolörüne ait hız hatasının değişimi



Şekil 9.18 NGTOFPID kontrolörüne ait kontrol işaretinin değişimi



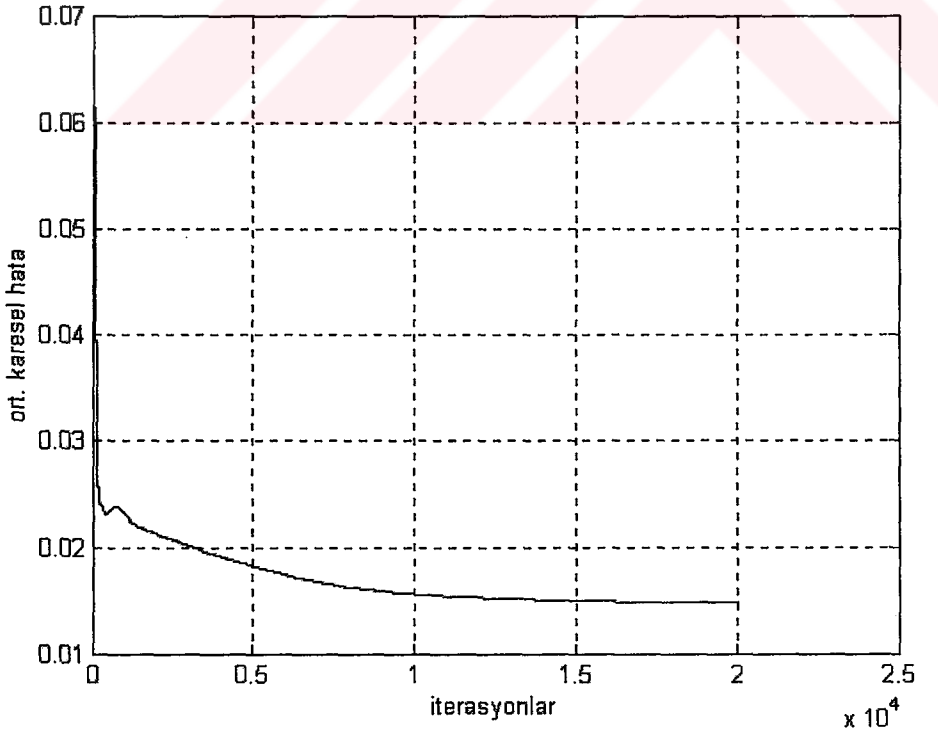
Şekil 9.19 NGTOFPID kontrolörü için, motorun rampa şeklinde değişen referansa (---) verdiği cevap (—)

9.1.4 NGTOSTFPI Kontrolörüne ait Deneysel Sonuçlar

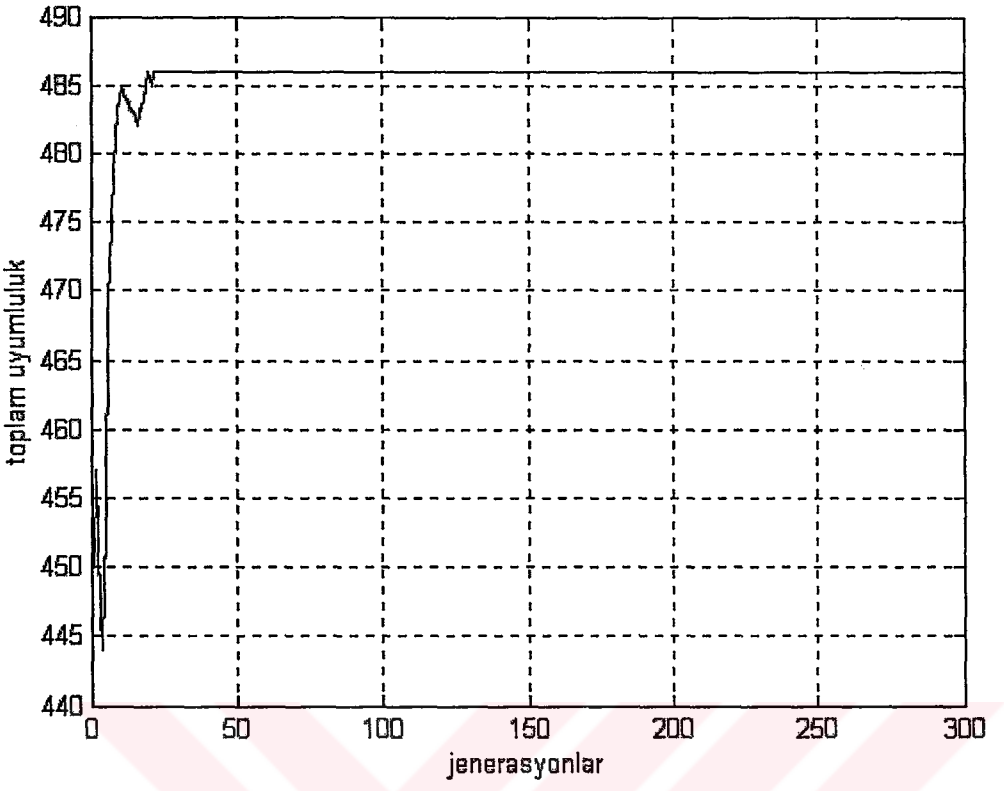
Sistemin YSA modeli 35 giriş / çıkış çifti ile eğitilmiştir. Eğitme esnasında 2 girişli – 3 çıkışlı, 1 gizli katmana ve ilgili gizli katmanda 5 hücreye sahip ağ yapısı kullanılmıştır. Eğitim algoritması olarak momentum yöntemine dayanan hatanın geriye yayılım algoritması kullanılmıştır. YSA 20000 iterasyon boyunca eğitilmiş ve ortalama karesel hatanın ilgili ağ mimarisi için 0.015 civarında seyrettiği gözlemlenmiştir. Ortalama karesel hatanın iterasyonlara göre değişimi Şekil 9.20'de gösterilmiştir. Daha sonra optimal giriş ölçekleme faktörlerinin taranması işlemine geçilmiştir. Bu işlem için Çizelge 9.4'de parametreleri verilen Genetik Algoritma kullanılmıştır. Genetik optimizasyon işlemi esnasında elde edilen jenerasyon başına toplam uyumluluk değişimi Şekil 9.21'de; optimal ölçekleme faktörlerinin kullanılması sonucunda elde edilen performans ölçütleri Çizelge 9.4'de, sistem çıkışının basamak şeklinde değişen 2500d/dk.'lık referansa verdiği cevap Şekil 9.22'de; hata işaretinin değişimi Şekil 9.23'de, kontrol işaretinin değişimi Şekil 9.24'de; öz-uyarlama mekanizmasının çıkışı olan α ayar parametresinin değişimi Şekil 9.25'de ve rampa şeklindeki referansa karşılık motor çıkış hızının değişimi Şekil 9.26'da gösterilmiştir.

Çizelge 9.4 NGTOSTFPI kontrolörüne ait parametreler ve deneysel sonuçlar

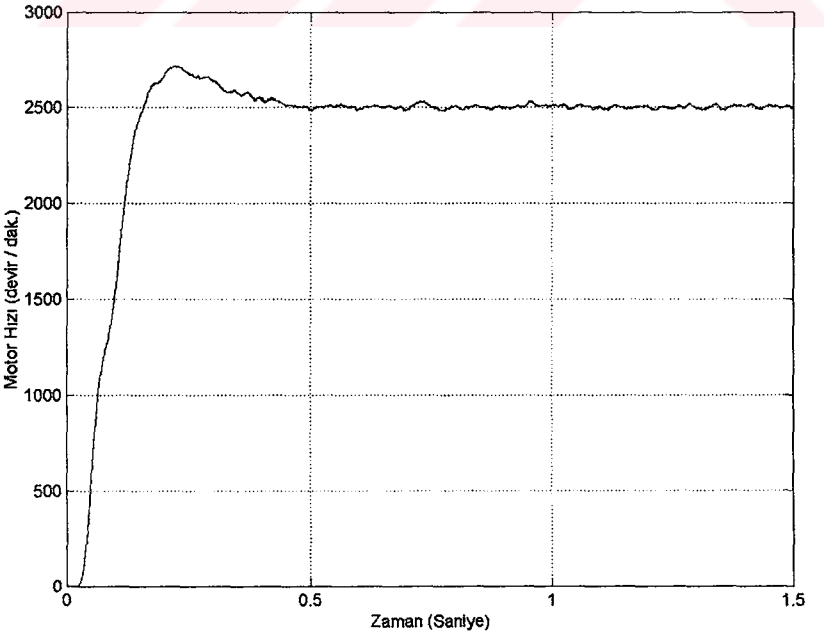
P_c	0.9
P_m	0.001
Popülasyon Büyüklüğü	40
Maksimum Jenerasyon Sayısı	300
Optimal S_e	0.1562
Optimal S_d	0.000855
U_{out}	50
U_α	1
U_m	50
U_{dm}	50
T_s	0.0005sn
IAE	118987
ITAE	12509
%Aşım	%8.6



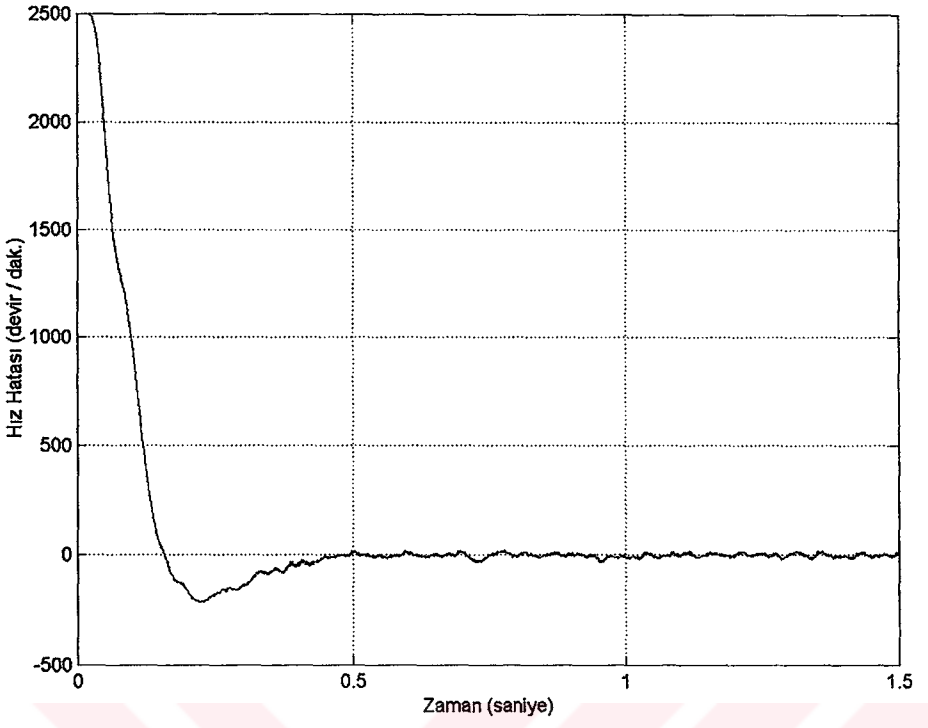
Şekil 9.20 NGTOSTFPI kontrolörünün tasarımı esnasında,YSA eğitimi süresince ortalama karesel hatanın değişimi



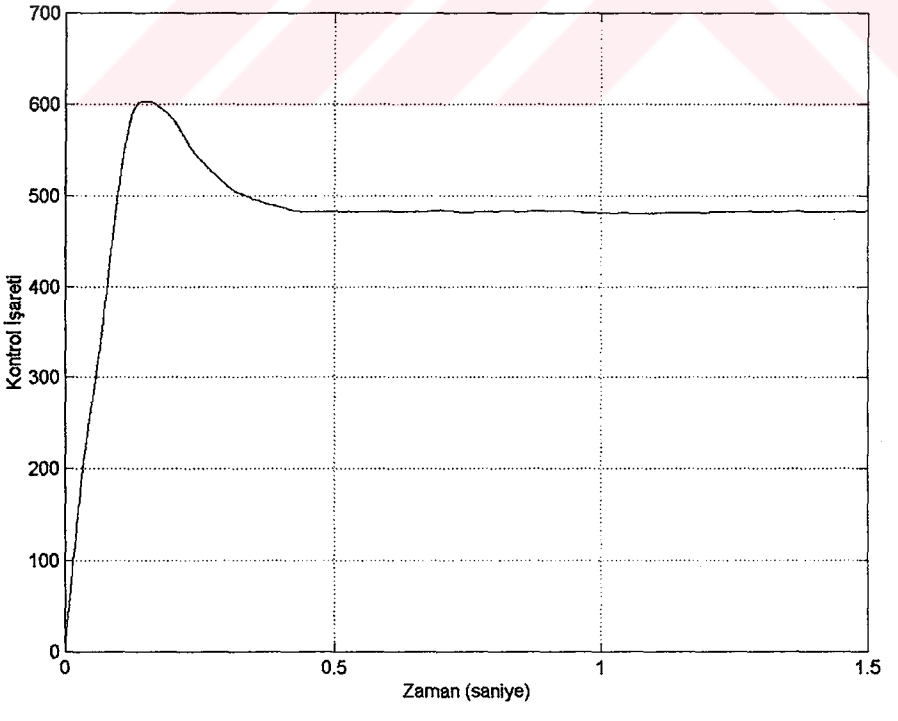
Şekil 9.21 NGTOSTFPI kontrolörünün genetik optimizasyon işlemi esnasında jenerasyonlara göre toplam uyum değişimi



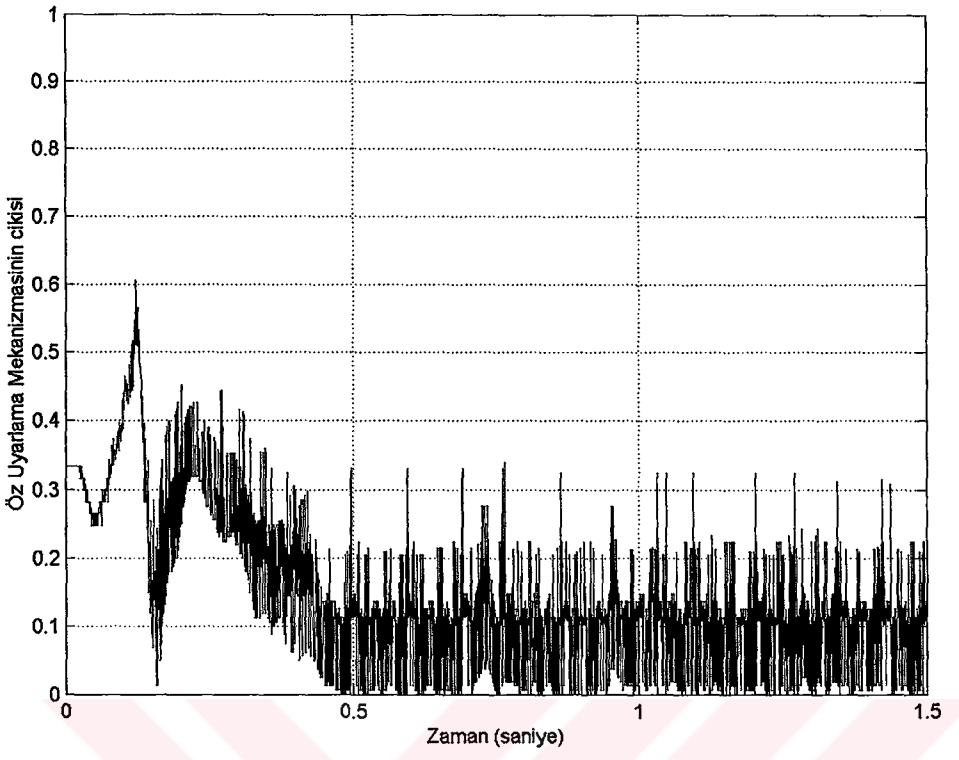
Şekil 9.22 NGTOSTFPI kontrolörünün basamak şeklindeki referansa verdiği cevap



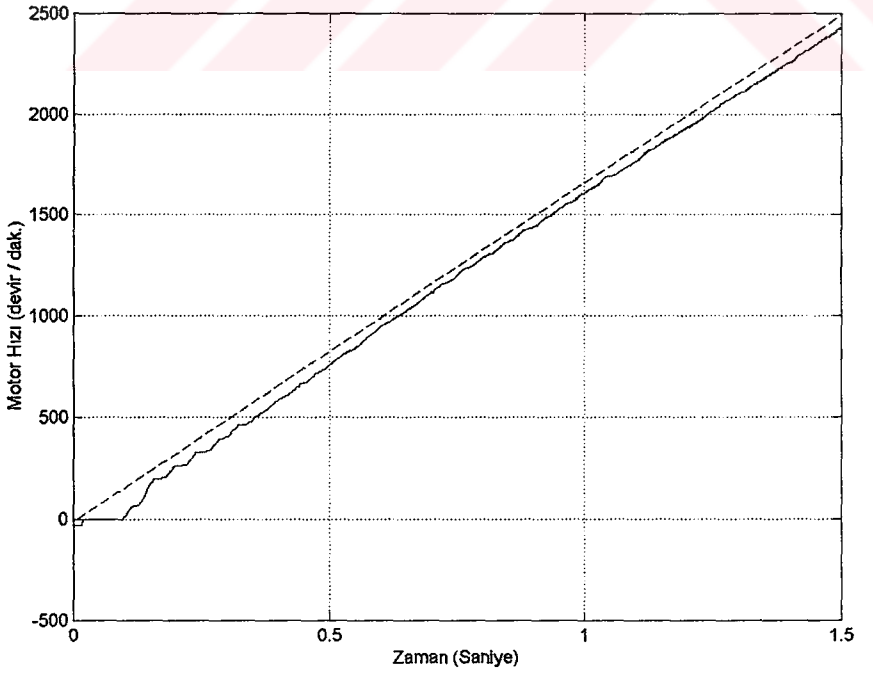
Şekil 9.23 NGTOSTFPI kontrolörüne ait hız hatasının değişimi



Şekil 9.24 NGTOSTFPI kontrolörüne ait kontrol işaretinin değişimi



Şekil 9.25 NGTOSTFPI kontrolörüne ait öz uyarlama mekanizmasının çıkışı



Şekil 9.26 NGTOSTFPI kontrolörü için, motorun rampa şeklinde değişen referansa (---) verdiği cevap (—)

9.1.5 NGTOSTFP-ID Kontrolörüne ait Deneysel Sonuçlar

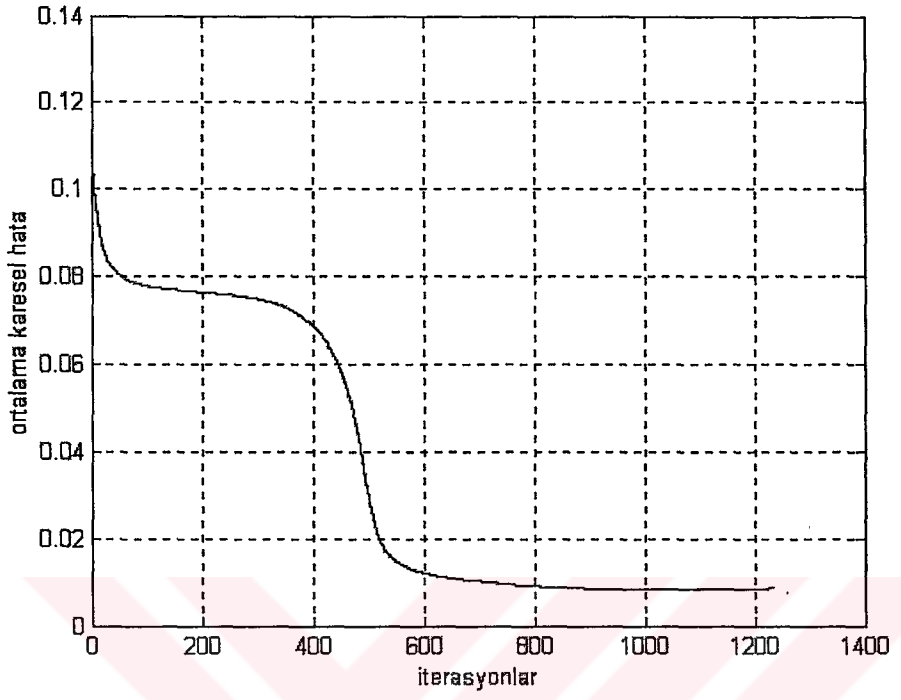
Öncelikle sistemin K_I ve K_d katsayıları sıfırlanarak öz-uyarlamalı bulanık kontrolörü gibi çalışan sistem ele alınır. Sistemin giriş ve çıkış ölçekleme faktörleri kabul edilebilir geçici rejim cevabı verecek şekilde kabaca ayarlanır. Ayarlanan bu değerlerden çıkış ölçekleme faktörü olan S_u değeri sabit tutularak belli bir aralıkta değişen giriş ölçekleme çiftleri için sistemin *IAE, ITAE* ve *%Aşım* cevapları bir dosyaya kaydedilir. Kaydedilen bu değerler sistemin YSA modelinin oluşturulması esnasında kullanılır. Sistemin, giriş ölçekleme faktörlerine verdiği performans cevapları 36 veri çifti için denenmiş ve bu değerler sistemin YSA modelinin eğitilmesi esnasında kullanılmıştır. YSA modelinin eğitimi esnasında momentum kuralına dayanan hatanın geriye yayılımı algoritması kullanılmıştır. Eğitmenin başlangıcından yaklaşık 1200 iterasyon sonra hatanın ortalama değerinin 0.01 civarına yakınsadığı ve burada kaldığı görülmüştür (Şekil 9.27). Daha sonra eğitilen ağ yapısı kullanılarak Genetik optimizasyon işlemine geçilmiştir. Optimizasyon işlemi için seçilen parametreler ile eğitme esnasında kullanılan bulanık kontrolörün parametreleri Çizelge 9.5'de gösterilmiştir.

Çizelge 9.5 NGTOSTFP-ID kontrolörünün giriş ölçekleme faktörlerinin optimizasyonu esnasında kullanılan parametreler

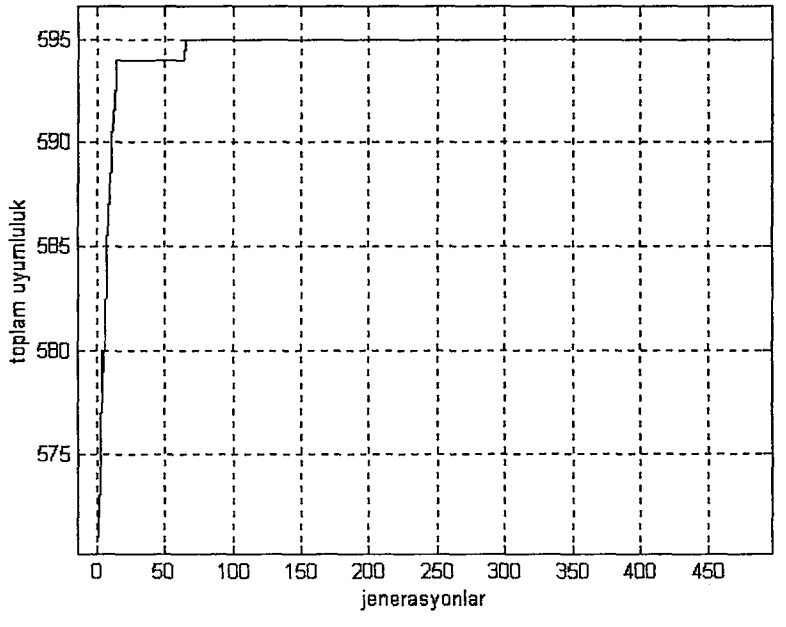
P_c	0.87
P_m	0.002
Popülasyon Büyüklüğü	30
Maksimum Jenerasyon Sayısı	500
Optimal S_e	0.0321
Optimal S_d	0.000091
U_{out}	20
U_α	1
U_m	20
U_{dm}	20
T_s	0.0005sn

Sistemin YSA modelinin oluşturulması esnasında 2 girişli- 3 çıkışlı, 1 gizli katmana ve ilgili gizli katmanda 10 hücreye sahip bir ağ yapısı kullanılmıştır. Ağın eğitimi esnasında ortalama karesel hatanın iterasyonlara göre değişimi Şekil 9.27'de; optimizasyon işlemi esnasında

jenerasyon başına düşen toplam uyumluluk ifadesinin değişimi ise Şekil 9.28'de gösterilmiştir.

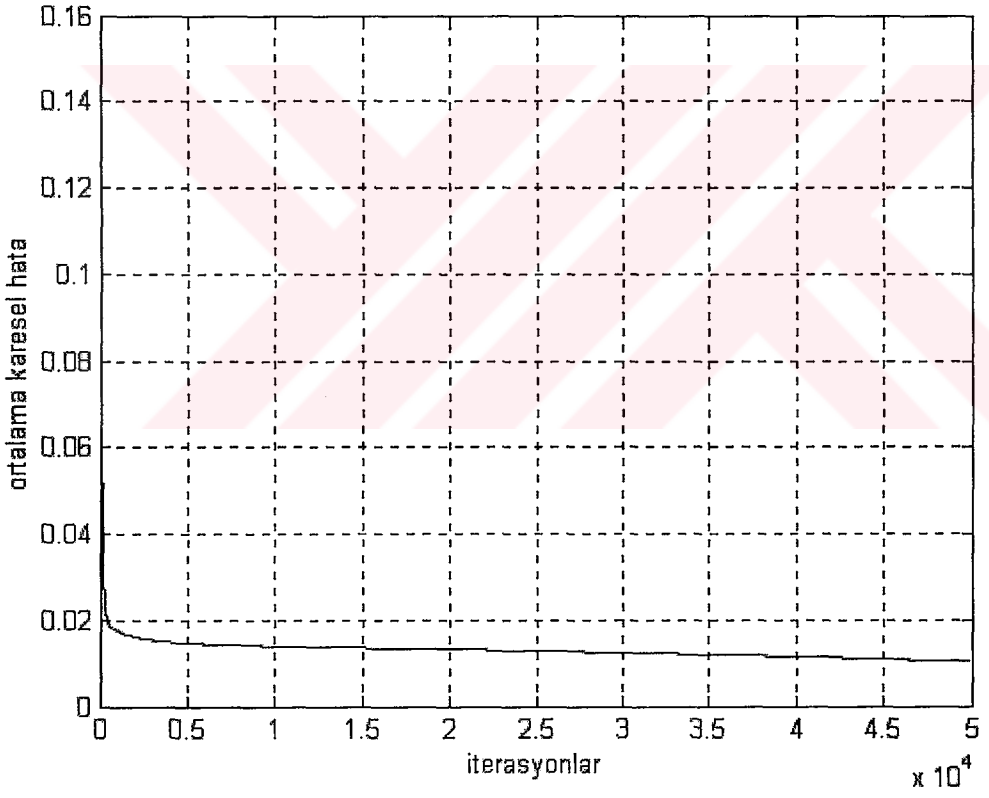


Şekil 9.27 NGTOSTFP-ID kontrolörünün tasarımının birinci aşaması esnasında,YSA eğitimi süresince ortalama karesel hatanın değişimi



Şekil 9.28 NGTOSTFP-ID kontrolörüne ait genetik optimizasyon işleminin birinci aşaması esnasında jenerasyonlara göre toplam uyum değişimi

NGTOSTFP-ID kontrolörüne ait parametrelerin optimizasyonunun ikinci aşaması K_I ve K_d katsayılarının ayarı ile ilgilidir. Bu aşamada bulanık kontrolörün giriş ölçekleme faktörleri olarak yukarıda bulunan optimale yakın değerler kullanılır. Çıkış ölçekleme faktörü ise değiştirilmez. Belli bir çalışma aralığından seçilen 16 adet K_I ve K_d katsayısı için performans ölçütleri IAE, ITAE ve %Aşım değerleri bir dosyaya kaydedilir. Daha sonra bu giriş – çıkış çiftleri kullanılarak sistemin katsayılara verdiği cevaba ait bir YSA modeli elde edilir. Kullanılan ağ, 2 girişli, 3 çıkışlı tek gizli katmana ve ilgili gizli katmanda 10 hücreye sahip bir mimariye sahiptir. Eğitme esnasında momentum kuralına dayanan hatanın geriye yayılımı algoritması kullanılmıştır. Eğitme işlemi yaklaşık 50000 iterasyon boyunca sürdürülmüş ve ortalama karesel hatanın 0.01 civarına yakınsadığı gözlemlenmiştir. Eğitme esnasında hatanın ortalama değerinin iterasyonlara göre değişimi Şekil 9.29'da gösterilmiştir.



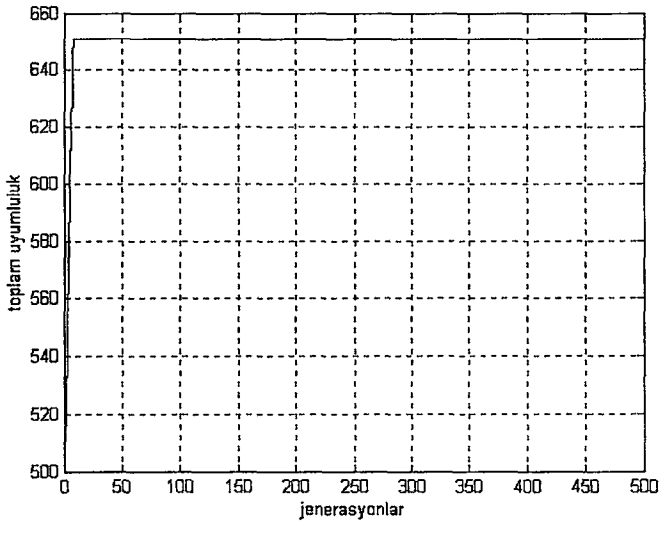
Şekil 9.29 NGTOSTFP-ID kontrolörünün tasarımının ikinci aşaması esnasında,YSA eğitimi süresince ortalama karesel hatanın değişimi

Daha sonra eğitilen ağ üzerinden Genetik Algoritma yardımı ile optimale yakın K_I ve K_d değerleri tespit edilmiştir. K_I ve K_d değerlerinin optimizasyonu esnasında kullanılan genetik algoritma parametreleri, bulunan optimal K_I ve K_d değerleri ile bulanık kontrolöre ait parametreler ve performans sonuçları Çizelge 9.6'da; optimizasyon işlemi esnasında toplam

uyumluluk ifadesinin jenerasyonlara göre değişimi ise Şekil 9.30'da gösterilmiştir.

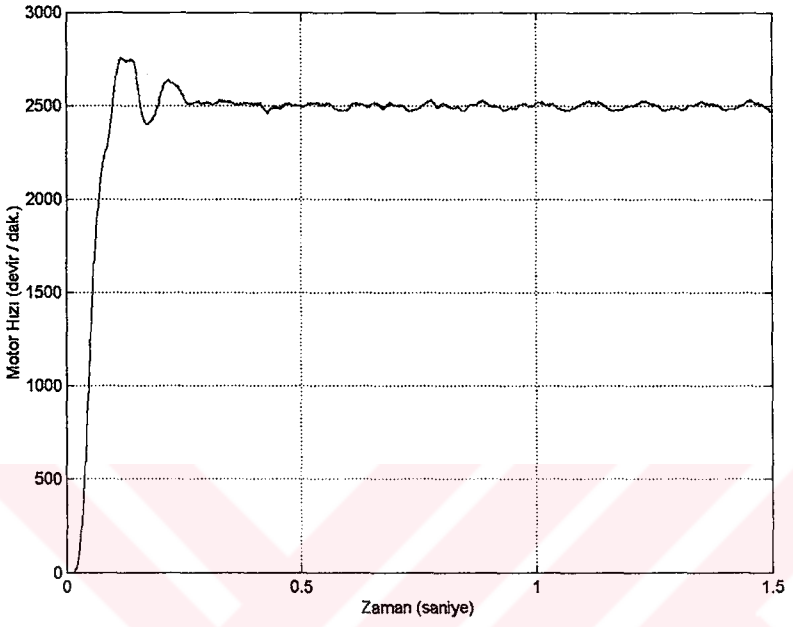
Çizelge 9.6 NGTOSTFP-ID kontrolörüne ait parametreler ve performans sonuçları

P_c	0.87
P_m	0.002
Popülasyon Büyüklüğü	30
Maksimum Jenerasyon Sayısı	500
Optimal S_e	0.0321
Optimal S_d	0.000091
U_{out}	20
U_α	10
U_m	20
U_{dm}	20
T_s	0.0005sn
IAE	83184
ITAE	9193
%Aşım	10
K_I	20
K_d	0.000048

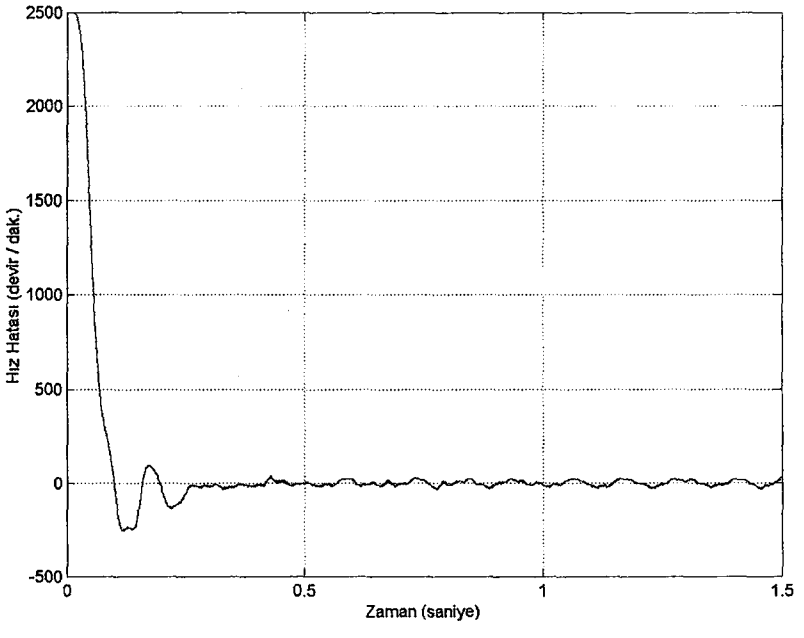


Şekil 9.30 NGTOSTFP-ID kontrolörüne ait genetik optimizasyon işleminin ikinci aşaması esnasında jenerasyonlara göre toplam uyum değişimi

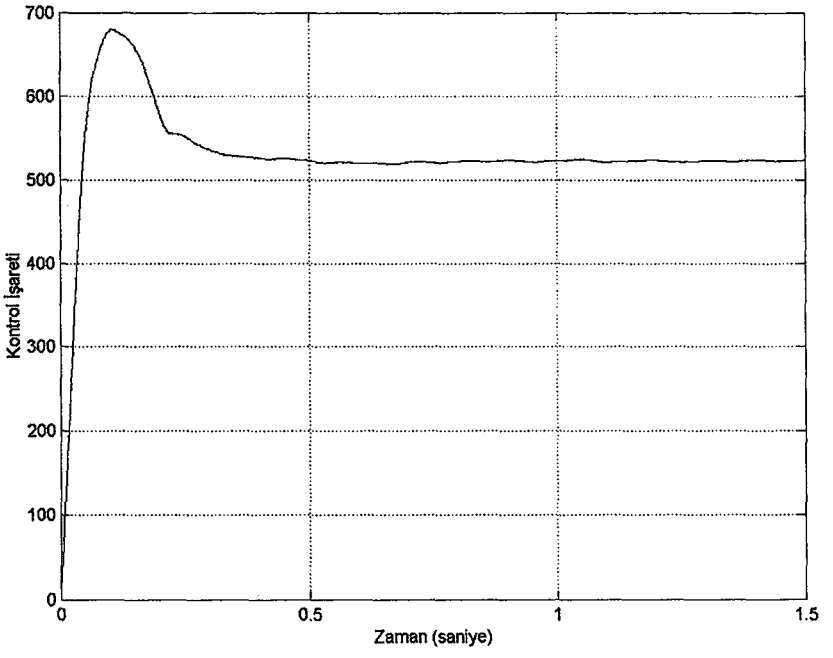
Şekil 9.31, sistemin basamak şeklinde değişen 2500d/dk.'lık referans işarete verdiği yanıtı; Şekil 9.32, sisteme ait hız hatasının değişimini; Şekil 9.33, kontrol işaretinin değişimini; Şekil 9.34, sistemin rampa şeklinde değişen referans işarete verdiği yanıtı ve son olarak Şekil 9.35 öz-uyarlama mekanizmasının çıkışını göstermektedir.



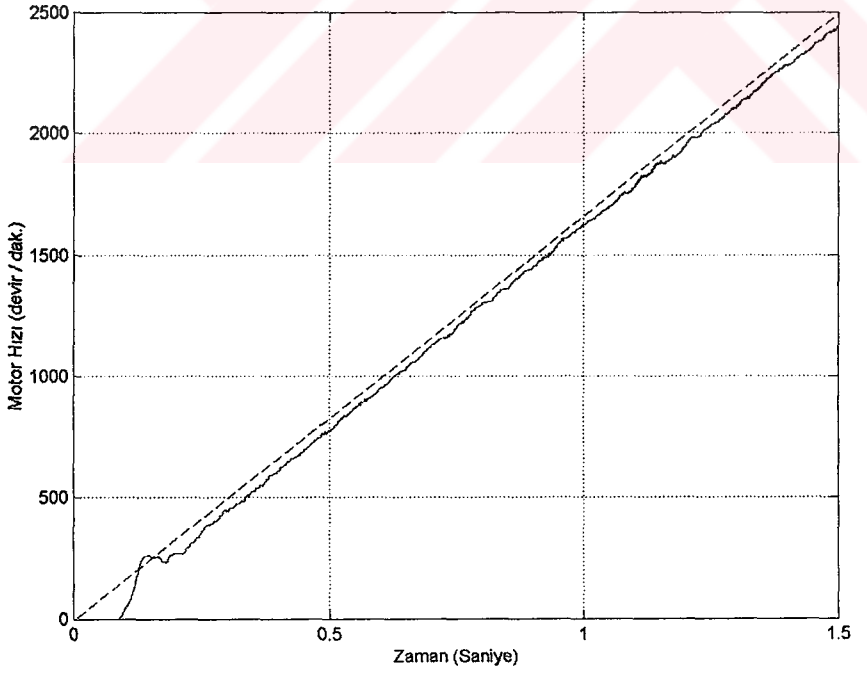
Şekil 9.31 NGTOSTFP-ID kontrolörünün basamak şeklindeki referansa verdiği cevap



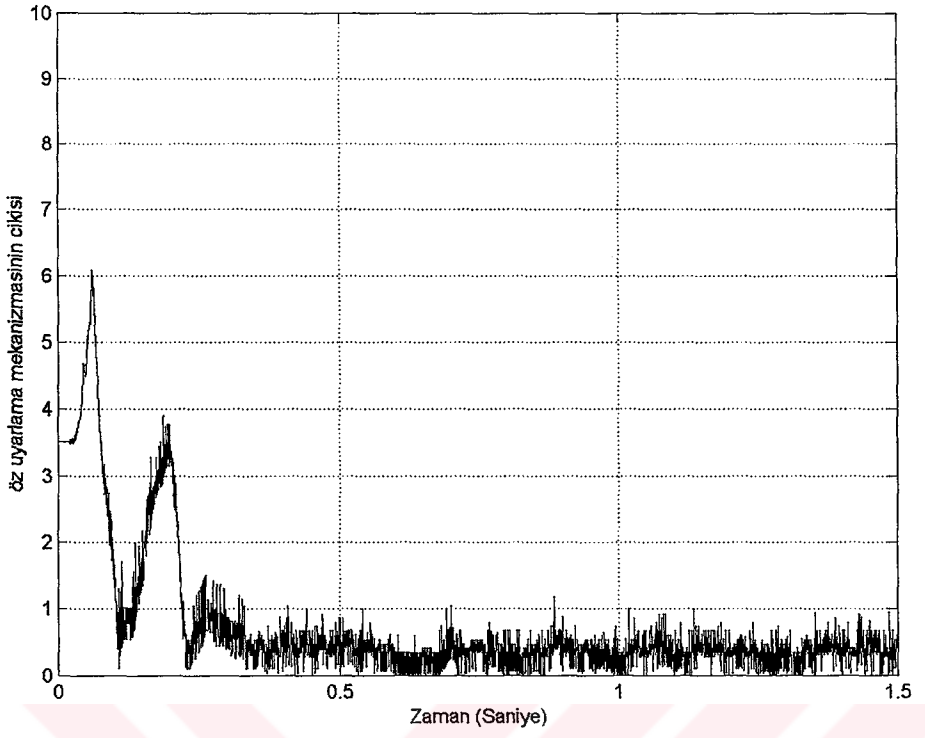
Şekil 9.32 NGTOSTFP-ID kontrolörüne ait hız hatasının değişimi



Şekil 9.33 NGTOSTFP-ID kontrolörüne ait kontrol işaretinin değişimi



Şekil 9.34 NGTOSTFP-ID kontrolörü için, motorun rampa şeklinde değişen referansa (---) verdiği cevap (—)



Şekil 9.35 NGTOSTFP-ID kontrolörüne ait öz-uyarlama mekanizmasının çıkışı

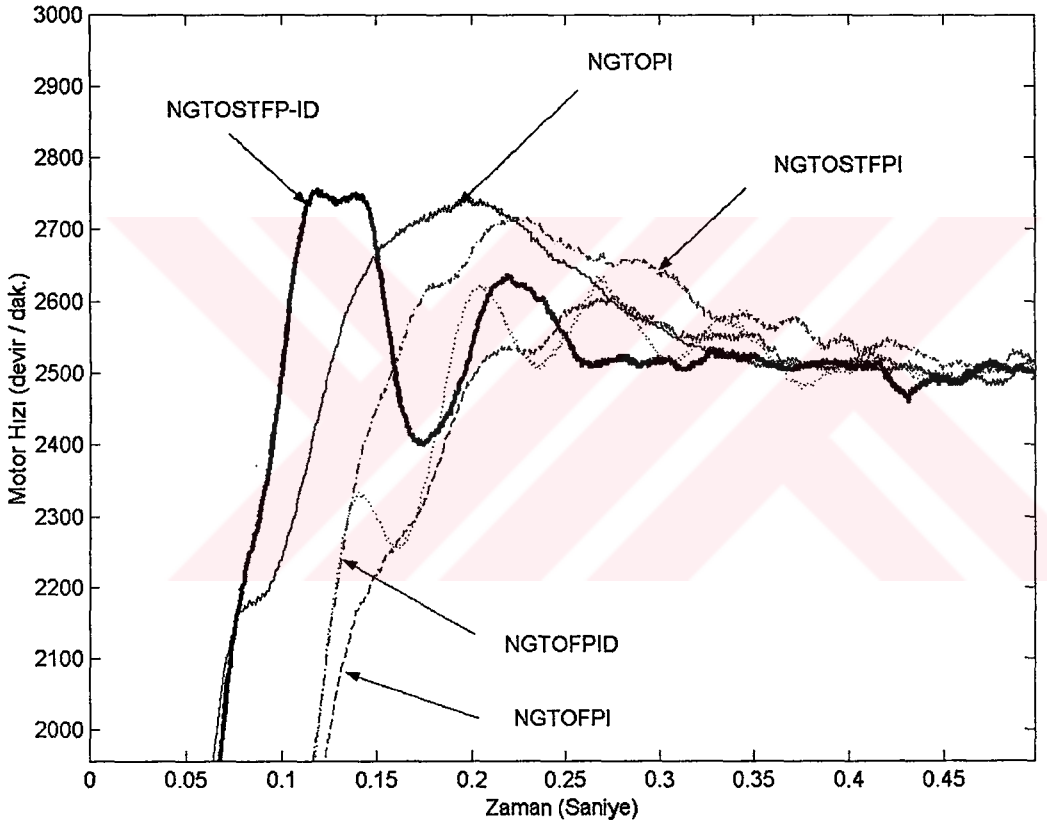
9.2 Sonuç, Tartışma ve Öneriler

Şekil 9.36'da da görülebileceği gibi en iyi basamak cevabını NGTOSTFPI kontrolörü sağlamakta; ona en yakın cevabı ise NGTOPI kontrolörü vermektedir.

Çizelge 9.7 Çalışmada ele alınan kontrolörlerin performans cevapları

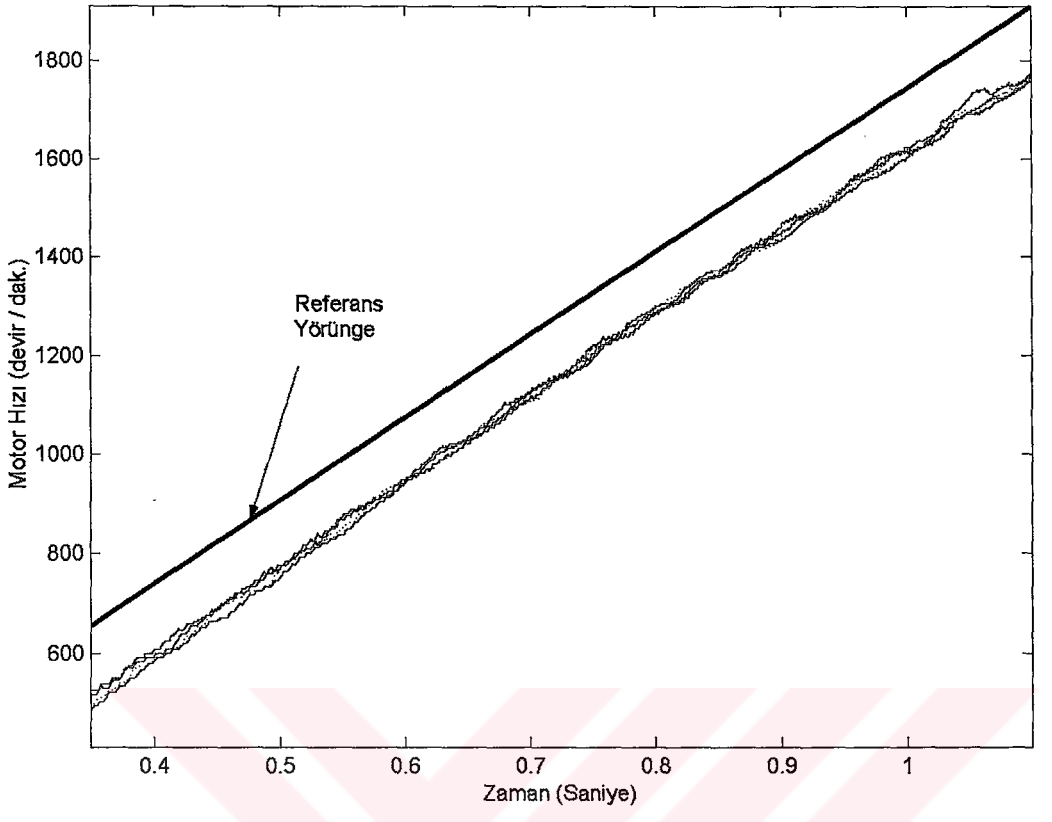
KONTROLÖRLER	IAE	ITAE	%AŞIM	Yük. Zamanı t_r (ms)	Yer. Zamanı t_s (ms)
NGTOPI	90007	9610	10	125	335
NGTOFPI	111618	15033	4.8	200	350
NGTOFPID	100946	11294	4.5	195	375
NGTOSTFPI	118987	12509	8.6	200	450
NGTOSTFP-ID (Önerilen Kontrolör)	83184	9193	10.1	100	250

Bu iki kontrolör basamak şeklinde değişen referans fonksiyona aynı aşımı göstermekte buna karşın NGTOSTFP-ID kontrolörü, NGTOPI kontrolörüne göre çok daha hızlı yükselmekte ve referans hıza yerleşmektedir. Ayrıca NGTOSTFP-ID kontrolörü kararlı hal hatalarının giderilmesi bakımından diğer kontrolörlere nazaran çok daha başarılı olmaktadır. NGTOFPI kontrolörünün çıkış ölçekleme faktörünü güncelleyen versiyonu olan NGTOSTFPI kontrolörü kararlı hal hatalarının yok edilmesi bakımından NGTOFPI kontrolörüne göre daha başarılıdır. Ancak yükselme ve yerleşme zamanları bakımından olumsuzluklar göstermektedir.



Şekil 9.36 Çalışmada ele alınan kontrolörlerin basamak cevaplarına ait geçici rejim davranışları

Ayrıca tez çalışmasında ele alınan kontrolörlerin rampa şeklinde değişen referans işarete verdikleri yanıtlar ise Şekil 9.37'de gösterilmiştir. İlgili şekilden de görülebileceği gibi tüm kontrolörler hemen, hemen aynı rampa cevabını sergilemekte ve referans işareti çok küçük bir hata ile takip etmektedirler.



Şekil 9.37 Çalışmada ele alınan kontrolörlerin rampa şeklinde değişen referansa verdiği yanıtlar

Günümüzde bulanık kontrol sistemlerin parametrelerinin ayarlanmasında hala deneme yanılma yöntemlerinden, uzman kişilerin görüşlerinden veya sistem üzerindeki tecrübelerinden yararlanılmaktadır. Ancak, uzman kişiler, sistemin her durumu için görüş ortaya koyamayabilirler. İşte bu gibi durumlarda yapılması gereken işlem kontrolör tasarımını daha dayanıklı temeller üzerine oturtmak, hata toleranslı kontrolörler tasarlamaktır. Bunu yapabilmenin yolu ise kontrolör tasarımlarını uzman kişinin de görüşlerinden faydalanılacak şekilde daha çok analitik tekniklerle desteklemektir. Bu çalışmada bulanık kontrolörlerin ayar işlemlerini bir sistematiğe bağlayan ve deneme yanılma yönteminin kullanımını en aza indirgeyen bir ayar yöntemi ortaya atılmıştır. Ayar mekanizması tamamı ile sistemin matematiksel modelinden bağımsız, gradyen düşüm tekniklerine dayanmayan sistematik bir yöntemdir. Matematiksel modellerden bağımsızlığı yapay sinir ağı; gradyen düşüm yönteminden bağımsızlığı ise genetik algoritmalar sağlamaktadır. Kontrolörün ayar parametreleri olarak genellikle giriş ölçekleme faktörleri üzerinde durulmuştur. Zira bu parametreler kontrolör duyarlılığını değiştirerek, kontrolörün performansını önemli ölçüde

etkilemektedir. Optimizasyon işlemi için önerilen yöntem iki aşamadan oluşmaktadır. Birinci aşama matematiksel olarak modellenemeyen veya modellenmesi oldukça zor ve zahmetli olan sistemin yapay sinir ağları ile tanınması; ikinci aşama ise ilgili YSA modeli üzerinden genetik algoritma yardımı ile kontrolör katsayılarının optimizasyonu işlemidir. Çalışmada bir yenilik olarak ortaya atılan bu yöntem sadece bulanık kontrolörlerin parametrelerinin ayarlanması için ortaya atılan bir yöntem değildir. Genellikle tüm kontrolör tiplerinde rahatlıkla kullanılabilecek bir yöntemdir. Ortaya atılan optimizasyon yöntemi vasıtası ile bulanık kontrolörlerin kural tabanları, çıkış ölçekleme faktörleri, değişik mimariye sahip PID kontrolör katsayıları rahatlıkla optimize edilebilmektedir. Tez çalışmasının yedinci bölümünde ortaya atılan NGTOSTFP-ID kontrolörü ise, yapısında manuel olarak ayar edilen parametre barındırmayan bir bulanık tabanlı kontrolör mekanizmasıdır. Bu kontrolörün kural tabanının tasarımı ise tamamen analitik bir yöntem yardımı ile tasarlanmıştır ve uzman kişi veya kişilerin görüşlerini içermemektedir. Son olarak, NGTOSTFP-ID tip kontrolör mimarisi Çizelge 9.7'den de görülebileceği gibi birçok yönden diğer kontrolör mimarilerine göre üstünlük sağlamaktadır. Deneysel sonuçlar, NGTOSTFP-ID kontrolörünün hassas hız ayarlarının istendiği mekanik sistemlerin kontrolünde (Robot kontrol sistemleri, hassas diş açma tezgahları, ölçü düzenekleri v.b.) başarı ile kullanılabileceğini ortaya koymaktadır.

Bundan sonra yapılabilecek çalışmalara ışık tutmak amacı ile, mevcut kontrolör mimarileri değişik proseslere uygulanabilir. Özellikle, yüksek mertebeden lineer sistemler ile matematiksel olarak modellenmesi zor olan lineer olmayan sistemlerde, ortaya atılan kontrolör mekanizmasının gücü sınanabilir. Diğer bir taraftan mevcut kontrolör mimarisine adaptif kontrolörler ilave edilebilir. NGTOSTFP-ID kontrolöründe bulunan çıkış ölçekleme faktörünün ayarlanmasında kullanılan kural tabanlı öz-uyarlama mekanizmasına benzer bir yapı, giriş ölçekleme faktörlerinin gerçek zamanlı olarak ayarlanmasında kullanılabilir. Ayrıca, sadece ölçekleme faktörlerinin GA tabanlı optimizasyonunu içeren işlem, tüm kural tabanını da içine alacak şekilde geliştirilebilir. Fakat bu durumda GA tarafından global minimumu aranan yüzeyin boyutları inanılmaz bir oranda artacaktır. Bu ise, global minimumun bulunmasını zorlaştıracak, optimizasyon işlemi için harcanan süreyi arttıracaktır.

Tezin bu konuda araştırma yapan araştırıcılara yararlı olmasını temenni ederim.

KAYNAKLAR

Belarbi K.ve Titel F., (2000), "Genetic Algorithm for the Design of a Class of Fuzzy Controllers: An Alternative Approach", IEEE Trans. Fuzzy Syst., 8(4): 398-404.

Bernard J. A., (1988), "Use of Rule-based System for Process Control", IEEE Contr. Syst. Mag., 8(5): 3-13.

Chao C. T. ve Teng C. C., (1997), "A PD-Lile Self-Tuning Fuzzy Controller without Steady-State Error", 87 :141-154.

Chen C. Ve Chang M. H., (1998), ":Optimal Design of Fuzzy Sliding-Mode Control: A comparative Study", Fuzzy Sets and Systems, 93: 37-48.

Chiaberge M., Bene D., Pascoli S., Lazzerini B., Maggiore A.ve Reyneri L. M., (1996), "Mixing Fuzzy, Neural and Genetic Algorithms in an Integrated Design Environment for Intelligent Controllers", Tech. Report, Torino, Italy.

Choi B. J., Kwak S. W. ve Kim B. K., (2000), "Design and Stability Analysis of Single-Input Fuzzy Logic Controller", IEEE Trans. Syst. Man Cyber., Part B, 30(2): 303-309.

Chung H. Y., Chen B. C. ve Lin J. J., (1998), "A PI-type Fuzzy Controller with Self-Tuning Scaling Factors", Fuzzy Sets Syst., 93: 23-28.

Efe M. Ö.ve Kaynak O., (2000), Yapay Sinir Ağları ve Uygulamaları, Boğaziçi Üniv. Yay., İstanbul.

Eminoğlu İ. ve Altaş İ. H., (1996), "A Method to Form Fuzzy Logic Control Rules for a PMDC Motor Drive System", Fuzzy Sets and Systems, 39:81-87.

Fujitec F., (1988), FLEX-8800 Series Elevator Group Control System, Fujitec Co. Ltd., 1988, Osaka, Japan.

Hayashi S., (1991), "Auto-tuning Fuzzy PI Controller", 41-44, Proc. Int. Fuzzy Syst. Assoc., Brussel, Belgium.

Haykin S., (1994), Neural Networks A comprehensive Foundation, MacMillan Pub., NJ.

He S. Z., Tan S., Xu F. L.ve Wang P. Z., (1993), "Fuzzy Self-tuning of PID Controller", Fuzzy Sets Syst., 56:37-49.

Horng J. H., (1999), "Neural Adaptive Tracking Control of a DC Motor", Information Sciences, 118:1-13.

Hu B., Mann K. I. ve Gosine R. G., (1999), "New Methodology for Analytical and Optimal Design of Fuzzy PID Controllers", IEEE Trans. Fuzzy Syst., 7(5): 521-539.

Karr C. L. ve Gentry E. J., (1993), "Fuzzy Control of pH Using Genetic Algorithms", IEEE Trans. Fuzzy Syst., 1(1): 46-53.

Kasai Y.ve Morimoto Y., (1988), "Electronically Controlled Continuously Variable Transmission", Proc. Int. Congress on Transportation Electronics, 1988, Dearborn, MI.

Kirk D. E., (1970), Optimal Control Theory, An Introduction, Prentice Hall, NJ.

Klir G. J.ve Yuan B., (1995), Fuzzy Sets and Fuzzy Logic, Theory and Applications, Prentice

Hall, NJ.

Koo T. J., (2001), "Stable Model Reference Adaptive Fuzzy Control of a Class of Nonlinear Systems", IEEE Trans. Fuzzy Syst., Vol.9(4): 624-636.

Kukolj D., Kulic F. ve Levi E., (2000), "Design of the Speed Controller for Sensorless Electric Drives Based on AI Techniques: A Comparative Study", Arttificial Intelligence in Eng., 14: 165-174.

Kuo Y. ve Li T., (1999), "GA-Based Fuzzy PI/PD Controller for Automotive Active Suspension System", IEEE Trans. Industrial Electronics, 46(6):1051-1056.

Lafore R., (1987), Turbo C, Programming for the IBM, Macmillan Inc., Indianapolis, Indiana.

Lee C. C., (1990), "Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Pat I", IEEE Trans. Syst. Man Cyber., 20(2): 404-415.

Lee C. C., (1990), "Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Pat II", IEEE Trans. Syst. Man Cyber., 20(2): 419-435.

Lee J., (1993), "On Methods for Improving Performance of PI-Type Fuzzy Logic Controllers", IEEE Trans. Fuzzy Syst., (1): 198-301.

Li H. X.ve Gatland H. B., (1996), "Conventional Fuzzy Control and its Enhancement", IEEE Trans. Syst. Man, Cybern., 26(5): 791-797.

Li T. S.ve Shieh M., (2000), "Design of a GA-based Fuzzy PID Controller for non-minimum Phase Systems", Fuzzy Sets and Syst., 111: 183-197.

Li W., (1997), "A Method for Design of a Hybris Neuro-Fuzzy Control System Based on Behavior Modeling", IEEE Trans. Fuzzy Syst., 5(1): 128-137.

Li W., (1998), "Design of a Hybrid Fuzzy Logic Proportional Plus Conventional Integral-Derivative Controller", IEEE Trans. on Fuzzy Systems, 6(4): 449-463.

Lyshevski S. E., (1999), "Nonlinear control of Mechatronic Systems with Permanent-Magnet DC Motors", Mechatronics, 9: 539-552.

Maeda M.ve Murakami S., (1992), "A Self-Tuning Fuzzy Controller", Fuzzy Sets Syst., 51:29-40.

Malki H. A. ve Li H., Chen G., (1994), "New Design and Stability of Fuzzy Proportional-Derivative Control Systems", IEEE Trans. Fuzzy Syst., 2(4): 245-254.

Mann G. K. I., Hu B. G. ve Gosine R., (1999), "Analysis of Direct Action Fuzzy PID Controller Structures", IEEE Trans. Syst. Man, Cybern., Part B, 29(3): 371-388.

Minkova M. D., Minkov D., Roggerson J. L. ve Harley R. G., (1998), "Adaptive Neural Speed Controller of a DC Motor", Electric Power Research, 47 : 123-132.

Mudi R. K.ve Pal N. R., (1999), "A Robust Self-tuning PI- and PD- Type Fuzzy Controllers", IEEE Trans. Fuzzy Syst., 7(1): 2-16.

Mudi R. K. ve Pal N. R., (2000), "A self-tuning PI Controller", Fuzzy Sets Syst., 115:327-338.

Nomura H., Hayashi I. ve Wakami N., (1991), "A Self-tuning Method of Fuzzy Control by

Decent Method", 155-158, Proc. Int. Fuzzy Syst. Assoc., Brussels, Belgium.

Palm R., (1995), "Scaling of Fuzzy Controller Using Cross-Corellation", IEEE Trans. Fuzzy Syst., 3:116-123.

Phillips C. L. ve Harbor R. D., (1988), Feedback Control Systems, Prentice Hall, NJ.

Su M. C. ve Chang H. T., (2000), "Application of Neural Networks Incorporated with Real-Valued Genetic Algorithms in Knowlegde Acquisition", Fuzzy Sets Syst., 112: 85-97.

Tarng Y. S., Yeh Z. M. ve Nian C. Y., (1996), "Genetic Synthesis of Fuzzy Controllers in Turning", Fuzzy Sets and Systems, 83: 301-310.

Vas P., (1999), Artificial-Intelligence-Based Electrical Machines and Drives, Oxford University Press, New-York.

Wang J., Rad A. B. ve Chan P. T., (2001), "Indirect Adaptive Fuzzy Sliding Mode control: Part I: Fuzzy Switching", Fuzzy Sets Syst., 122 : 21-30.

Wang L. X., (1997), A Course in Fuzzy Systems and Control, Prentice Hall, NJ.

Wang L. X., (1999), "Automatic Design of Fuzzy Controllers", Automatica, 35:1471-1475.

Yagishita O., Itoh O. ve Sugeno M.., (1985), "Application of Fuzzy Reasoning to the Water Purification Process", 19-40, Industrial Applications of Fuzzy Control, 1985, Holland.

Yasunobu S., Miyamoto S. ve Ihara H., (1983), "Fuzzy Control for Automatic Train Operation System", Proc. 4th IFAC / IFIP / IFORS Int. Congress on Control in Transportation Systems, Baden – Baden.

Yasunobu S. ve Hasegawa T., (1986), "Evaluation of an Automatic Container Crane Operation System Based on Predictive Fuzzy Control", Control Theory Adv. Technol., 2(3): 419-432.

Yeh Z., (1999), "A Systematic Method for Design of Multivariable Fuzzy Logic Control Systems", Dec. 1999, 7(6): 741-752.

Yoshida M., Tsutsumi Y. ve Ishida T., (1990), "Gain Tuning for Design of Fuzzy Control Systems", Proc. Int. Conf. Fuzzy Logic Neural Networks, Fukuoka, Japan, 405-408.

Zadeh L. A., (1999), "From Computing with Numbers to Computing with Words – From Manipulation of Measurements to Manipulation of Perceptions", IEEE Trans. Syst., Man, Cybern., Part I, 45(1):105-119.

Zhao Z. Y., Tomizuka M. ve Isaka S., (1993), "Fuzzy Gain Scheduling of PID Controllers", IEEE Trans. Syst., Man, Cybern., Part I, 23(5): 1392-1398.

Zheng L., (1992), "A Practical Guide to Tune of Proportional and Integral (PI) Like Fuzzy Controllers", Proc. Fuzz. IEEE, Mar. 1992, San Diego, CA, 633-641.

Zhou Y. ve Lai L., (2000), "Optimal Design for Fuzzy Controllers by Genetic Algorithms", IEEE Trans. Industry App., 36(1): 93-97.

EKLER

Ek 1 Nöral-Genetik Tabanlı Optimizasyon Programı

```
/*    GENETIK ALGORITMA ILE SISTEM GIRIS OLCEKLEME FAKTORLERININ
                  OPTIMIZASYONUNU YAPAN PROGRAM
            programmed by ibrahim Beklan KUCUKDEMIRAL
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#include <process.h>
#include <alloc.h>
#include<ctype.h>
#include<string.h>
#include<alloc.h>

#define NMXUNIT   10
#define NMXHLR    10
#define NMXOATTR  128
#define NMXINP    120
#define NMXIATTR  120
#define SEXIT     3
#define RESTRT    2
#define FEXIT     1
#define CONTNE    0

#define MAXPOP    200
#define MAXGEN    5000

/* yapi bildirimleri */

struct CHROMOSOME {
      unsigned int string;
      unsigned int ustbyte;
      unsigned int altbyte;
};

struct ELEMAN {
      struct CHROMOSOME Ke;
      struct CHROMOSOME Kde;
      double fitness;
};

/* fonksiyon prototipleri*/

void select(struct ELEMAN *pop_old, struct ELEMAN *pop_new, int popsize);
void crossover(int popsize, struct ELEMAN *pop_new, double pcross);
void mutation(double pmutation, struct ELEMAN *pop_new, int popsize, int
n_generation);
void Fitness(struct ELEMAN *pop_old, int popsize);
void initialize (struct ELEMAN *pop_old, int popsize);
void statistics(struct ELEMAN *pop_old,int popsize);
void sorting_popold(struct ELEMAN *pop_old, int popsize);
void sorting_popnew(struct ELEMAN *pop_new, int popsize);
void Fitness_new(struct ELEMAN *pop_new, int popsize);
void init(void);
void forward(struct ELEMAN *eleman);
dread(void);
wtread(void);
```

```
/* global degiskenler */

struct ELEMAN pop_old[MAXPOP];
struct ELEMAN pop_new[MAXPOP];

double sum_fitness;

float net, eta, alpha, err_curr, maxe, maxep,nw,g1[100];
float far *wtptr[NMXHLR + 1];
float far *outptr[NMXHLR + 2];
float far *errptr[NMXHLR + 2];
float far *delw[NMXHLR + 1];
float far input[NMXINP][NMXIATTR];
float far ep[NMXINP];
float far outpt[NMXINP][NMXOATTR];
int nunit[NMXHLR+2];
unsigned int nhlayer, ninput, ninattr, noutattr;
long int cnt_num;
char task_name[20];
char var_file_name[20];
FILE *fp1, *fp2;


void init(void)
{
      int len1, len2, i;
      float *p1, *p2, *p3, *p4;

      len1 = len2 = 0;
      nunit[nhlayer + 2]=0;
      for(i = 0; i < (nhlayer+2); i++){
            len1 += (nunit[i] + 1) * nunit[i + 1];
            len2 += nunit[i] + 1;
      }
      p1 = (float *) calloc(len1 + 1, sizeof(float));
      p2 = (float *) calloc(len2 + 1, sizeof(float));
      p3 = (float *) calloc(len2 + 1, sizeof(float));
      p4 = (float *) calloc(len1 + 1, sizeof(float));
      wtptr[0] = p1;
      outptr[0] = p2;
      errptr[0] = p3;
      delw[0] = p4;
      for (i = 1; i < (nhlayer + 1); i++){
            wtptr[i] = wtptr[i - 1] + nunit[i] * (nunit[i - 1] + 1);
            delw[i] = delw[i - 1] + nunit[i] * (nunit[i - 1] + 1);
      }
      for (i = 1; i < (nhlayer + 2); i++){
            outptr[i] = outptr[i - 1] + nunit[i - 1] + 1;
            errptr[i] = errptr[i - 1] + nunit[i - 1] + 1;
      }
      for (i = 0; i < nhlayer + 1; i++)
            *(outptr[i] + nunit[i]) = 1.0;
}

dread(void)
{
      int i, c;

      //strcpy(var_file_name, taskname);

      strcpy(var_file_name, "_v.dat");
```

```
      if (( fp1 = fopen(var_file_name, "r")) == NULL) {
            printf("\n1 Veri dosyasi alinamiyor...");
            exit(0);
      }
      fscanf(fp1, "%u%u%u%f%f%u%lu\n", &ninput, &noutattr,
      &ninattr, &eta, &alpha, &nhlayer, &cnt_num);
      for (i = 0; i < nhlayer + 2; i++)
            fscanf(fp1,"%d", &nunit[i]);

      if ((c = fclose(fp1)) != 0)
            printf("\nFile cannot be closed %d ", c);
      return(0);
}

wtread(void)
{
      int i, j, c ;
      char wt_file_name[20];

      //strcpy(wt_file_name, taskname);
      strcpy(wt_file_name, "_w.dat");
      if ((fp2 = fopen(wt_file_name, "r")) == NULL){
            printf("\n Veri dosyasi a‡ilamiyor");
      exit(0);
      }
      for (i = 0; i < nhlayer + 1; i++){
            for (j = 0; j < (nunit[i] + 1) * nunit[i + 1]; j++)
                  fscanf (fp2, "%f", (wtptr[i] + j));
      }
      if ((c = fclose(fp2)) != 0)
            printf("\n%d dosya kapatilamiyor", c);
      return(0);
}

void forward(struct ELEMAN *eleman)
{
      int i = 0, m, n, p, offset;

      input[i][0] = (float)((0.15 - 0.05) / 100.F) * (eleman -> Ke.string)
+ 0.05;
      input[i][1] = (float)((0.001 - 0.0001) / 100.F) * (eleman ->
Kde.string) + 0.0001;
      for (m = 0; m < ninattr; m++)
            *(outptr[0] + m) = input[i][m];
      for (m = 1; m < nhlayer + 2; m++){
            for (n = 0; n < nunit[m]; n++){
                  net = 0.0;
                  for (p = 0; p < nunit[m - 1] + 1; p++){
                        offset = (nunit[m - 1] + 1) * n + p;
                        net += *(wtptr[m - 1] + offset) * (*(outptr[m - 1]
+ p));
                  }
                  *(outptr[m] + n) = 1 / (1 + exp(-net));
            }
      }
      for (n = 0; n < nunit[nhlayer + 1]; n++)
            outpt[i][n] = *(outptr[nhlayer + 1] + n);
      // fitness ifadesi
      // n = 0 degeri Maxovershhot olsun
      // n = 1 degeri setling time olsun
      eleman -> fitness = (double) ( 100. / (.001 + (outpt[i][0] * 10) +
                              (20. * outpt[i][1]) + (10. * outpt[i][2] )));
```

```
      if ((eleman -> Ke. string > 100) || (eleman -> Kde.string > 100))
            eleman -> fitness = 0.;
}

void select(struct ELEMAN *pop_old, struct ELEMAN *pop_new, int popsize)
{
      int generation = 0, j;
      double _random;
      double partialsum = 0.0, sumfitness = 0.0;

      //srand((unsigned) time(NULL));
      for (j = 0; j < popsize; ++j)
            sumfitness += pop_old[j].fitness;

/* rulet tekerleginin dondurulmesi ile 0 .. toplam uyumluluk
arasi sayi uretilmesi */
      for (generation = 0; generation < popsize; ++generation) {
            partialsum = 0.0; j = -1;
            _random = ((rand() % 100 ) / 100. ) * sumfitness;
            do {
                  j = j + 1;
                  partialsum += pop_old[j].fitness;

            } while ((partialsum < _random) && ( j != (popsize - 1)));
            pop_new[generation] = pop_old[j];
      }
}

void crossover(int popsize, struct ELEMAN *pop_new, double pcross)
{
      int parent1_indis, parent2_indis, _random = 0, i, sayac;
      struct ELEMAN parent1;
      struct ELEMAN parent2;

      sayac = (int)(pcross * popsize);
      for (i = 0; i < sayac; ++i) {
//eslesecek elemanlardan birincisi random bir sayi olarak bulunur
            _random = rand() % popsize;
            parent1_indis = _random;
//eslesecek elemanlardan ikincisi random bir sayi olarak bulunur
            _random = rand() % popsize;
            parent2_indis = _random;
/* havuzdan secilen rastgele secilen iki eleman parent1 ve
parent2'ya icerikleri yuklenir. */
            parent1 = pop_new[parent1_indis];
            parent2 = pop_new[parent2_indis];
/* elemanlarin crossover islemi 8. elemandan itibaren yapilmaktadir.
(sabit); bu yuzden string degeri ust ve alt byte olarak iki kisima
paylastirilir. */
            parent1.Ke.ustbyte = parent1.Ke.string & 0xff00;
            parent1.Ke.altbyte = parent1.Ke.string & 0x00ff;

            parent1.Kde.ustbyte = parent1.Kde.string & 0xff00;
            parent1.Kde.altbyte = parent1.Kde.string & 0x00ff;

            parent2.Ke.ustbyte = parent2.Ke.string & 0xff00;
            parent2.Ke.altbyte = parent2.Ke.string & 0x00ff;

            parent2.Kde.ustbyte = parent2.Kde.string & 0xff00;
            parent2.Kde.altbyte = parent2.Kde.string & 0x00ff;
/* paylastirilan kisimlar birlestirilerek yeni string eski stringin uzerine
kopyalanir. */
```

```
            parent1.Ke.string = parent1.Ke.ustbyte | parent2.Ke.altbyte;
            parent2.Ke.string = parent2.Ke.ustbyte | parent1.Ke.altbyte;

            parent1.Kde.string = parent1.Kde.ustbyte | parent2.Kde.altbyte;
            parent2.Kde.string = parent2.Kde.ustbyte | parent1.Kde.altbyte;

            pop_new[parent1_indis] = parent1;
            pop_new[parent2_indis] = parent2;
      }
}

void mutation(double pmutation, struct ELEMAN *pop_new, int popsize, int
n_generation)
{
      int i, sayac, uye, _bit;
/* mutasyon icin rastgele elemanin secilmesi */

      sayac = (int) (pmutation * popsize * 16);
      for (i = 0; i < sayac; ++i) {
            uye = rand() % popsize;
            if (n_generation < 50)
                  _bit = rand() % 15;
            else
                  _bit = rand() % 8;
            pop_new[uye].Ke.string ^= (unsigned) pow(2.0, (double)_bit);
            pop_new[uye].Kde.string ^= (unsigned) pow(2.0, (double)_bit);
      }
}

void initialize (struct ELEMAN *pop_old, int popsize)
{
      int i;

      dread();
      init();
      dread();
      wtread();
      for (i = 0; i < popsize; ++i) {
            pop_old[i].Ke.string = (unsigned) (rand() % 100);
            pop_old[i].Kde.string = (unsigned) (rand() % 100);
      }
}
/* fitness degerleri su anda deneme amaci ile rastgele bir fonksiyon
yardimi ile uretilmektedir. daha sonra sistem'den elde edilecek gercek
veriler ile gerceklenecektir. bu bolume YSA programi gelecek
*/

void Fitness_new(struct ELEMAN *pop_new, int popsize)
{
      int i;

      for (i = 0; i < popsize; ++i)
            forward(&(pop_new[i]));
}

void Fitness(struct ELEMAN *pop_old, int popsize)
{
      struct ELEMAN den;
      int i;

      for (i = 0; i < popsize; ++i)
            forward(&(pop_old[i]));
```

```
}

void statistics(struct ELEMAN *pop_old,int popsize)
{
      int i;

      sum_fitness = 0.0;
      for (i = 0; i < popsize; ++i)
            sum_fitness += pop_old[i].fitness;
}


// populasyon da en yuksek uyumluluga sahip elani buluyoruz.
// bu elemani bir sonraki jenerasyonun en kotu elemani uzerine kopyalicaz.
void sorting_popold(struct ELEMAN *pop_old, int popsize)
{
      int i, k;
      struct ELEMAN temp;

      for (i = 0; i < popsize - 1; ++i)
            for (k = 0; k < popsize - 1; ++k)
                  if (pop_old[k].fitness > pop_old[k+1].fitness) {
                        temp = pop_old[k];
                        pop_old[k] = pop_old[k + 1];
                        pop_old[k + 1] = temp;
                  }
}

void sorting_popnew(struct ELEMAN *pop_new, int popsize)
{
      int i, k;
      struct ELEMAN temp;

      for (i = 0; i < popsize - 1; ++i)
            for (k = 0; k < popsize - 1; ++k)
                  if (pop_new[k].fitness > pop_new[k+ 1].fitness) {
                        temp = pop_new[k];
                        pop_new[k] = pop_new[k + 1];
                        pop_new[k + 1] = temp;
                  }
}

void main(void)
{
      FILE *f1, *f2;
      int n_generation, k, popsize, i;
      double pcross, pmutation;

      srand((unsigned) time(NULL));
      clrscr();
      printf("\n crossover oranini giriniz =");
      scanf("%lf", &pcross);
      printf("\n mutasyon oranini giriniz =");
      scanf("%lf", &pmutation);
      printf("\n populasyon buyuklugunu giriniz =");
      scanf("%d", &popsize);
      printf("\n Jenerasyon sayisini giriniz =");
      scanf("%d", &n_generation);

      clrscr();
      if ((f1 = fopen("optim.dat", "w")) == NULL) {
            printf("Can not open file!..\n");
```

```
        exit(1);
    }
    if ((f2 = fopen("uyum.dat", "w")) == NULL) {
        printf("Can not open file!..\n");
        exit(1);
    }
    fprintf(f1, "\n\nCROSSOVER ORANI    : %lf\n", pcross);
    fprintf(f1, "MUTASYON ORANI   :%lf\n", pmutation);
    fprintf(f1, "POPULASYON NUFUSU      :%d\n", popsize);
    fprintf(f1, "JENERASYON SAYISI      :%d\n\n", n_generation);
    initialize(pop_old, popsize);
    for (k = 0; k < n_generation; ++k) {
        Fitness(pop_old, popsize);
        sorting_popold(pop_old, popsize);
        select(pop_old, pop_new, popsize);
        sorting_popnew(pop_new, popsize);
        pop_new[0] = pop_old[popsize-1];
        crossover(popsize, pop_new, pcross);
        mutation(pmutation, pop_new, popsize, n_generation);
        Fitness_new(pop_new, popsize);
        sorting_popnew(pop_new, popsize);
        //pop_new[0] = pop_old[popsize-1];
        for (i = 0; i < popsize; ++i)
            pop_old[i] = pop_new[i];

        statistics(pop_old, popsize);
        for (i = 0; i < popsize; ++i) {
            fprintf(f1, " --------%d. jenerasyon -------\n", k);
            fprintf(f1, "Ke degeri: %u\n", pop_old[i].Ke.string);
            fprintf(f1, "Kde degeri: %u\n", pop_old[i].Kde.string);
            fprintf(f1, "\n");
            fprintf(f1, "%d . chromosome icin uyumluluk: %lf\n\n",
                    i, pop_old[i].fitness);
        }
        fprintf(f2, "%d\n", (int)sum_fitness);
    }
    fclose(f1);
    fclose(f2);
}
```

Ek 2 Yapay Sinir Ağları (YSA) ile Sistem Tanıma Programı

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>
#include <ctype.h>

#ifndef VAX
#include <alloc.h>
#endif

#define NMXUNIT   10
#define NMXHLR    10
#define NMXOATTR  8
#define NMXINP    440
#define NMXIATTR  120
#define SEXIT     3
#define RESTRT    2
#define FEXIT     1
#define CONTNE    0

/* fonksiyon prototipleri */

void git(void);
int randoM(void);
init(void);
initwt(void);
set_up(void);
dwrite(char *taskname);
wtwrite( char *taskname);
void forward(int i);
int introspective (int nfrom, int nto);
int rumelhart(int from_snum, int to_snum);
user_session(void);
learning(void);

float eta, alpha, err_curr, maxe, maxep;
float huge *wtptr[NMXHLR+1];
float huge *outptr[NMXHLR+2];
float huge *errptr[NMXHLR+2];
float huge *delw[NMXHLR+1];
float huge target[NMXINP][NMXOATTR];
float huge input[NMXINP][NMXIATTR], se5, se6;

float huge ep[NMXINP];
float huge outpt[NMXINP][NMXOATTR];
int nunit[NMXHLR + 2], nhlayer, ninput, ninattr, noutattr, sd;
int result, i, m, errorcode;
long int cnt_num, cnt;
int nsnew, nsold;
char task_name[20], dr;
FILE *fp1, *fp2, *fp3, *fopen();
int fplot10;
char *err_file = "veri3.dat";
```

```
long randseed = 568731L;
char def[30], def1[20];
int mx, my;
void *p;

void git(void)
{
      clrscr();
      printf("YAPILAN ITERASYON= %ld", cnt);
      printf("\n ŞU ANKİ HATA = %f\n", err_curr);
      printf("ÇIKMAK İSTİYORMUSUNUZ ?'d(dur)-HERHANGi BİR TUŞA BASINIZ");
      if(getch() == 'd')
            cnt = cnt_num - 1;
}
//*********Rastgele sayi uretilmesi**********************************
int randoM(void)
{
      randseed = 15625L * randseed + 22221L;
      return((randseed >> 16) & 0x7FFF);
}
//*******Gizli katmanlar icin dinamik hafizada yer ayrilmasi**********
init(void)
{
      int len1, len2, i, k;
      float *p1, *p2, *p3, *p4;

      len1 = len2 = 0;
      nunit[nhlayer + 2] = 0;
      for(i = 0; i < (nhlayer+2); i++){
            len1 += (nunit[i] + 1) * nunit[i + 1];
            len2 += nunit[i] + 1;
      }
      p1 = (float *) calloc(len1 + 1, sizeof(float));
      p2 = (float *) calloc(len2 + 1, sizeof(float));
      p3 = (float *) calloc(len2 + 1, sizeof(float));
      p4 = (float *) calloc(len1 + 1, sizeof(float));
      wtptr[0] = p1;
      outptr[0] = p2;
      errptr[0] = p3;
      delw[0] = p4;
      for (i = 1; i < (nhlayer + 1); i++){
            wtptr[i] = wtptr[i - 1] + nunit[i] * (nunit[i - 1] + 1);
            delw[i] = delw[i - 1] + nunit[i] * (nunit[i - 1] + 1);
      }
      for (i = 1; i < (nhlayer + 2); i++){
      outptr[i] = outptr[i - 1] + nunit[i - 1] + 1;
      errptr[i] = errptr[i - 1] + nunit[i - 1] + 1;
      }
      for (i = 0; i < nhlayer + 1; i++){
            *(outptr[i] + nunit[i]) = 1.0;
      }
      return(0);
}
//******* Agirlik degerleri baslangicta rasgele atanir.****************
initwt(void)
{
      int i, j;
      for (j = 0; j < nhlayer + 1;j++)
            for (i = 0; i < (nunit[j] + 1) * nunit[j + 1]; i++){
                  *(wtptr[j] + i) = randoM() / pow(2.0, 15.0) - 0.5;
                  *(delw[j] + i) = 0.0;
      }
```

```
	return(0);
}

//********YSA mimarisinin belirlenmesi**********************
set_up(void)
{
	int i;

	eta = 0.9;
	// printf("\n Alfa momentum katsaysi (default=0.9)?: ");
	// scanf("%f",&eta);
	alpha = 0.7;
	// printf("\n Epsilon ogrenme orani (default=0.7)?: ");
	//scanf("%f",&alpha);
	maxe = 0.000001; maxep = 0.000001;
	printf("\nMaksimum toplam hata (default=0.00001)?: ");
	printf("\nMaksimum baslangic hatasi(default=0.00001)?:");
	cnt_num = 1000;
	printf("\nMaksimum iterasyon sayisi (default=1000)?:");
	//scanf("%ld",&cnt_num);
	cnt_num=100000;
	printf("\nGizli katman sayisi?:");
	scanf("%d", &nhlayer);
	for(i = 0; i < nhlayer; i++){
		printf("\n\t %d. gizli katmandaki dugum sayisi?:", i + 1);
		scanf("%d", &nunit[i + 1]);
	}
	fplot10 = 1;
	clrscr();
	printf("\nBilgisayar hesap yapiyor.");
	nunit[nhlayer + 1] = noutattr;
	nunit[0] = ninattr;
	return(0);
}

//********Egitim sonunda cikislarin dosyaya atanmasi *****************

dwrite(char *taskname)
{
	int i, j, c;
	char var_file_name[20];

	strcpy(var_file_name, taskname);
	strcat(var_file_name, "_v.dat");
	if ((fp1 = fopen(var_file_name, "w+")) == NULL)
	{
		perror("\n Veri dosyasi acilamiyor");
		exit(0);
	}
	fprintf(fp1, "%u %u %u %f %f %u %lu\n",
	ninput, noutattr, ninattr, eta, alpha, nhlayer, cnt_num);
	for (i = 0; i < nhlayer + 2; i++){
		fprintf(fp1, "%d ", nunit[i]);
	}
	fprintf(fp1, "\n");
	for (i = 0; i < ninput; i++){
		for (j=0; j<noutattr; j++)
			fprintf (fp1, "%f      ", outpt[i][j]);
		fprintf (fp1, "\n");
	}
	if ((c = fclose(fp2)) != 0)
		printf("\nFile cannot be closed %d", c);
```

```
      return(0);
}

//********Egitilmis agirliklarin dosyaya yuklenmesi****************
wtwrite( char *taskname)
{
      int i, j, c, k;
      char wt_file_name[20];

      strcpy(wt_file_name, taskname);
      strcat(wt_file_name, "_w.dat");
      if ((fp2 = fopen(wt_file_name, "w+")) == NULL){
            perror("\n Veri dosyasi acilamiyor");
            exit(0);
      }
      k = 0;
      for (i = 0; i < nhlayer + 1; i++)
      for (j = 0; j < (nunit[i] + 1) * nunit[i + 1]; j++){
            if(k == 8){
                  k = 0;
                  fprintf (fp2, "\n");
            }
            fprintf (fp2, "%f ", *(wtptr[i] + j));
            k++;
      }
      if ((c = fclose(fp2)) != 0)
            printf("\n%d.Dosya kapat□lam□yor ", c);
      return(0);
}

//************ agirliklarin uretilmesi ***************************
void forward(int i)
{
      int m, n, p, offset;
      float net;

      for (m = 0; m < ninattr; m++)
            *(outptr[0] + m) = input[i][m];
      for (m = 1; m < nhlayer + 2; m++){
            for (n = 0; n < nunit[m]; n++){
                  net = 0.0;
                  for (p = 0; p < nunit[m - 1] + 1; p++){
                        offset = (nunit[m - 1] + 1) * n + p;
                        net += *(wtptr[m - 1] + offset)
                        *(*(outptr[m - 1] + p));
                  }
            *(outptr[m] + n) = 1 / (1 + exp(-net));
            }
      }
      for(n = 0; n < nunit[nhlayer + 1]; n++)
            outpt[i][n] = *(outptr[nhlayer + 1] + n);
}

//********* Hatanin islemin sona erdirilmesi icin kontrolu *************
int introspective (int nfrom, int nto)
{
      int i, flag;

      if (cnt >= cnt_num)
             return(FEXIT);
      nsnew = 0;
      flag = 1;
```

```
      for (i = nfrom;( i < nto) && (flag == 1); i++){
            if (ep[i] <= maxep)
                  nsnew++;
            else flag = 0;
      }
      if (flag == 1)
            return(SEXIT);
      if (err_curr <= maxe)
            return(SEXIT);
      return(CONTNE);
}

//*********** Agirliklarin ayarlanmasi ***********************
int rumelhart(int from_snum, int to_snum)
{
      int i, j, k, m, n, p, offset, index;
      float out;
      char *err_file = "veri3.dat";

      nsold = 0;
      cnt = 0;
      result = CONTNE;
      if ((fp3 = fopen(err_file, "w")) == NULL){
            perror("Veri dosyasi acilamiyor.");
            exit(0);
      }
      do {
            err_curr = 0.0;
            for (i = from_snum; i < to_snum; i++){
                  forward(i);
                  for(m = 0; m < nunit[nhlayer + 1]; m++){
                        out = *(outptr[nhlayer + 1] + m);
                        *(errptr[nhlayer + 1] + m) =
                        (target[i][m] - out) * (1 - out) * out;
                  }
                  for (m = nhlayer + 1; m >= 1; m--){
                        for (n = 0; n < nunit[m - 1] + 1; n++){
                              *(errptr[m - 1] + n) = 0.0;
                              for (p = 0; p < nunit[m]; p++){
                                    offset = (nunit[m - 1] + 1) * p + n;
                                    *(delw[m - 1] + offset) = eta *
(*(errptr[m] + p))
                                    * (*(outptr[m - 1] + n)) +
                                    alpha * (*(delw[m - 1] + offset));
                                    *(errptr[m - 1] + n) += *(errptr[m] + p)
                                    * (*(wtptr[m - 1] + offset));
                              }
                              *(errptr[m - 1] + n) = *(errptr[m - 1] + n)*
                              (1 - *(outptr[m - 1] + n)) * (*(outptr[m -1] +
n));
                        }
                  }
                  for (m = 1; m < nhlayer + 2; m++){
                        for(n = 0; n < nunit[m]; n++){
                              for(p = 0; p < nunit[m - 1] + 1; p++){
                                    offset = (nunit[m - 1] + 1) * n + p;
                                    *(wtptr[m-1]+offset)+=*(delw[m-
1]+offset);
                              }
                        }
                  }
                  ep[i] = 0.0;
```

```
                for(m = 0; m < nunit[nhlayer + 1]; m++){
                        ep[i] += fabs((target[i][m] - *(outptr[nhlayer+1] +
m)));
                }
                err_curr += ep[i] * ep[i];
        }
        err_curr = 0.5 * err_curr / ninput;
        if (kbhit())
                git();
        fprintf(fp3, "%ld\t%f\n", cnt, err_curr);
        cnt++;
        result = introspective(from_snum, to_snum);
    } while (result == CONTNE);
    for (i = from_snum; i < to_snum; i++)
        forward(i);
    for(i = 0; i < nhlayer + 1; i++){
        index = 0;
        for (j = 0; j < nunit[i + 1]; j++){
            printf("\n\n Weights between unit %d of layer %d", j, i +
1);
            printf("and units of layer %d\n", i);
            for (k = 0; k < nunit[i]; k++)
                printf ("%f", *(wtptr[i] + index++));
            printf ("\n Threshold of unit %d of layer%dis%f",
            j, i + 1, *(wtptr[i] + index++));
        }
    }
    for (i = 0; i < ninput; i++)
        for (j = 0; j < noutattr; j++){
            printf("\n\n sample %d output %d=%f target %d=%f",
            i, j, outpt[i][j], j, target[i][j]);
            getch();
        }
    printf("\n\n Toplam iterasyon sayisi %ld", cnt);
    printf("\n Normalize edilmis sistem hatasi %f\n\n\n", err_curr);
    return(result);
}

//************** Egitim verilerinin YSA'ya tanitilmasi ******************
user_session(void)
{
    int i, j, showdata;
    char fnam[20], task_name[20], dtype[20];
    FILE *fp;

    printf("\n Egitme safhasinin basi ");
    printf("\n\t Veri dosyasinin adini giriniz (taskname) : ");
    scanf("%s",task_name);
    //strcpy(task_name, "sin");
    printf("\n Kac tane giris dugumu (veri sayisi)?:");
    scanf("%d",&ninattr);
    //ninattr = 2;
    printf("\nKac tane cikis dugumu?: ");
    scanf("%d",&noutattr);
    //noutattr = 2;
    printf("\n Toplam kac tane grup?: ");
    scanf("%d", &ninput);
    strcpy(fnam, task_name);
    strcat(fnam, ".dat");
    printf("\n Giris dosyasinin adi %s", fnam);
    if ((fp = fopen(fnam, "r")) == NULL){
```

```
            printf("\n %s dosyasi yok", fnam);
            exit(0);
      }
      //for (m=0;m<12;m++) fscanf(fp," %f",&se5);
      printf("\n Verilere bakmak istermisiniz.?");
      printf("\n Answer yes or no :");
      scanf("%s", dtype);
      showdata = ((dtype[0] == 'y') || (dtype[0] == 'Y'));
      for (i = 0; i < ninput; i++) {
            for (j = 0; j < ninattr; j++){
                  fscanf(fp," %f", &se5);
                  input[i][j] = se5;
                  if (showdata)
                        printf("\n%f ", input[i][j]);
            }
            for (j = 0; j < noutattr;j ++){
                  fscanf(fp," %f", &se6);
                  target[i][j] = se6;
                  if (showdata)
                        printf("\n %f", target[i][j]);
            }
      }
      if ((i = fclose(fp)) != 0)
      {
            printf("\n %d dosya kapatilamiyor.", i);
            exit(0);
      }
      return(0);
}

//*****************************************************
learning(void)
{
      int result;

      user_session();
      set_up();
      init();
      do {
            initwt();
            result = rumelhart(0, ninput);
      } while (result == RESTRT);
      dwrite(task_name);
      wtwrite(task_name);
      return(0);
}

//*****************************************************
main()
{
      learning();
      printf("\nEgitim islemi bitti. ");
      return(0);
}
```

Ek 3 Nöral-Genetik Tabanlı Optimal PI Kontrolörü

```
//************ N O T L A R --- O N E M L I
//integral katsayısı KI max = 2.5 olacak
//oran KATSAYISI max =0.003 min =0.0006


#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

#include <conio.h>

#define     DOSYA "pi028.dat"
#define     KP    0.096
#define     KI    15.
#define     REF   600     //2390 devir/dk
#define     BASE  0x280
#define     TRAIN 0
int adcoku (void);

int adcoku (void)
{
      int tempLSB, tempMSB, x;

      outportb(BASE + 16, 15);
      while (!((inportb(BASE + 20) & 0x01)))
            ;
      tempMSB = inportb(BASE + 18);
      tempLSB = inportb(BASE + 19) & 0x0C;
      x = tempMSB << 4;
      x += tempLSB;
      return x - 1888;
}

void main(void)
{
      FILE *f1, *data;

      register float sinyal;
      register int      hiz;
      register int hata = 0, hata1= 0, hata2 = 0;
      register long top_hata = 0L;
      register int hata_gec = 0;
      register int k = 0;
      register int i;
      int m;
      register float kont = 0.F, kont_int = 0.F;

      int error[3000];
      long tophiz = 0L,t;
      long ara_hiz = 0L;
      float top_error = 0.F, itae = 0.F;

      int speed[3000], ref[3000], kontrolor[3000];
      int read_hiz[100] = {0};
      float max_ovrsht;
      int max_hiz = 0, t_rise, flag = 1;
```

```
KONTROL:

      while (k < 6000) {
                  disable();
                  tophiz = 0;
                  ara_hiz = 0L;

                  for (i = 0; i < 3; ++i)
                        tophiz += adcoku();

                  read_hiz[0] = tophiz / 3;
                  for (i = 0; i < 35; ++i)
                        ara_hiz += read_hiz[i];
                  hiz = ara_hiz / 35;

                  if ((k < 3000) && (hiz > max_hiz))
                        max_hiz = hiz;


                  if (flag && ( hiz == REF)) {
                        t_rise = k;
                        flag = 0;
                  }

                  if ((k < 1000) || (k > 2000))
                        hata = REF - hiz;
                  else
                        hata = REF/2. - hiz;

                  if (k < 3000) {
                        speed[k] = hiz;
                        error[k] = hata;

                        top_hata += k * abs(hata); // IATE hesabı
                        ref[k] = hata + hiz;
                  }
                  kont += (float)(KP * (hata - hata1)) + (KI * 0.0005 *
hata);
                  if (k < 3000)
                        kontrolor[k] = (int)kont;
                  hata1 = hata;

                  for (i = 34; 0 < i ; --i)
                        read_hiz[i] = read_hiz[i-1];

                  hata_gec = hata;
                  k++;
                  for (i = 0; i < kont; i += 4) {
                        outp(0x378, 1);
                        if (!(inp(0x379) & 0x20)) {
                              outp(0x378, 0);
                              goto KONTROL;
                        }
                  }
                  outp(0x378, 0);
                  while ((inp(0x379) & 0x20))
                        ;
                  goto KONTROL;
      }
      enable();
      max_ovrsht = (float) (max_hiz - REF) / REF;
      outp(0x378, 0);
```

```
      outp(0x378, 0);
      clrscr();
// real time control islemi bitince sonuclarin ilgili dizilerden
// asagidaki gibi tanimli .dat uzantili bir dosyaya kaydedilmesi

      if ((f1 = fopen(DOSYA, "w+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#if TRAIN
      if ((data = fopen("data_2.dat", "a+")) == NULL) {
            printf("CAn not open file!..\n");
            exit(1);
      }
#endif
      for (k=0; k < 3000; ++k) {
            top_error += (float) abs(error[k]);
            itae += (float) k * 0.0005 *abs(error[k]);
            fprintf(f1, "%d\t%d\t%d\n", speed[k], error[k], kontrolor[k]);
      }
      for (k=2950; k < 3000; ++k)
            printf("hiz: %d\n", speed[k]);
#if TRAIN
      fprintf(data, "%f\n", KP);
      fprintf(data, "%f\n", KI / 100.F);
      fprintf(data, "%f\n", (top_error - 50000) / (240000 - 50000));
      fprintf(data, "%f\n", max_ovrsht);
      fprintf(data, "%f\n", (itae - 5000) / (40000 - 5000));
      fclose(data);
#endif
      fclose(f1);
      getch();
}
```

Ek 4 Nöral-Genetik Tabanlı Optimal Bulanık PI Kontrolörü

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>

#define     TRAIN       1
#define     DOSYA       "pi04.dat"
#define     REF         600    //2390 devir/dk
#define     BASE        0x280
#define     K1          400.
#define     K2          0.1

#define     Ts          0.0005
#define     EM          20.F      //80
#define     EMd         20.F      //80
#define     OM          20.F    //1800
#define     OM_alfa     10.F       //1
#define     Ge          .01      // .05
#define     Gce         .0001F      // .07

// global degiskenler, uyelik fonksiyonlari

float w[49]; // inference agirliklari
float w_a[49]; // inference alfa agirliklari
float y_a[49];
float y[49];
float PB, PM, PS, Z0, NS, NM, NB;
float PB1, PM1, PS1, Z01, NS1, NM1, NB1;
float PBd, PMd, PSd, Z0d, NSd, NMd, NBd;
float PBd1, PMd1, PSd1, Z0d1, NSd1, NMd1, NBd1;
float yPB, yPM, yPS, yZ0, yNS, yNM, yNB;
float yPBd, yPMd, yPSd, yZ0d, yNSd, yNMd, yNBd;
float u_bulanık, alfa;

void init(void);
void init_alfa(void);
void inference (void);
void inference_alfa (void);
void fuzzification (float error);
void fuzzification1 (float error);
void fuzzification_errate1 (float error_rate);
void fuzzification_errate (float error_rate);
int adcoku (void);

int adcoku (void)
{
      int tempLSB, tempMSB, x;

      outportb(BASE + 16, 15);
      while (!((inportb(BASE + 20) & 0x01)))
            ;
      tempMSB = inportb(BASE + 18);
      tempLSB = inportb(BASE + 19) & 0x0C;
      x = tempMSB << 4;
      x += tempLSB;
      return x - 1888;
}
```

```
void fuzzification_errate (float error_rate)
{
      PBd = 0.f; PMd = 0.f; PSd = 0.f; Z0d = 0.f; NSd = 0.f; NMd = 0.f;
      NBd = 0.f;


      if (error_rate >= EMd)
            PBd = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd = (3.f / EMd * error_rate) - 2;
            PMd = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd = (3.f / EMd * error_rate) - 1;
            PSd = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd = 3.f / EMd * error_rate;
            Z0d = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd = (-3.f / EMd * error_rate);
            Z0d = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd = (3.f / EMd * error_rate) + 2;
            NMd = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd = (3.f / EMd * error_rate) + 3;
            NBd = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd = 1;
}

void fuzzification_errate1 (float error_rate)
{
      PBd1 = 0.f; PMd1 = 0.f; PSd1 = 0.f; Z0d1 = 0.f; NSd1 = 0.f; NMd1 =
0.f;
      NBd1 = 0.f;


      if (error_rate >= EMd)
            PBd1 = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd1 = (3.f / EMd * error_rate) - 2;
            PMd1 = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd1 = (3.f / EMd * error_rate) - 1;
            PSd1 = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd1 = 3.f / EMd * error_rate;
            Z0d1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
```

```
            NSd1 = (-3.f / EMd * error_rate);
            Z0d1 = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd1 = (3.f / EMd * error_rate) + 2;
            NMd1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd1 = (3.f / EMd * error_rate) + 3;
            NBd1 = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd1 = 1;
}

void init(void)
//kural matrisi main kontroler icin
{

      yNB = - OM; yNM = -(2 * OM / 3.F); yNS = -( OM / 3.F); yZ0 = 0.F;
      yPB = OM; yPM = (2 * OM / 3.F); yPS = OM / 3.F;

      y[0]  = yNB;  y[1] = yNB; y[2]  = yNB; y[3]  = yNB; y[4]  = yNM; y[5]
= yNS; y[6]  = yZ0;
      y[7]  = yNB;  y[8] = yNB; y[9]  = yNB; y[10] = yNM; y[11] = yNS;
y[12] = yZ0; y[13] = yPS;
      y[14] = yNB; y[15] = yNB; y[16] = yNM; y[17] = yNS; y[18] = yZ0;
y[19] = yPS; y[20] = yPM;
      y[21] = yNB; y[22] = yNM; y[23] = yNS; y[24] = yZ0; y[25] = yPS;
y[26] = yPM; y[27] = yPB;
      y[28] = yNM; y[29] = yNS; y[30] = yZ0; y[31] = yPS; y[32] = yPM;
y[33] = yPB; y[33] = yPB;
      y[35] = yNS; y[36] = yZ0; y[37] = yPS; y[38] = yPM; y[39] = yPB;
y[40] = yPB; y[41] = yPB;
      y[42] = yZ0; y[43] = yPS; y[44] = yPM; y[45] = yPB; y[46] = yPB;
y[47] = yPB; y[48] = yPB;
}

void init_alfa(void)
//kural matrisi alfa kontroler icin
{

      yNBd = - OM_alfa; yNMd = -(2 * OM_alfa / 3.F); yNSd = -( OM_alfa /
3.F); yZ0d = 0.F;
      yPBd = OM_alfa; yPMd = (2 * OM_alfa / 3.F); yPSd = OM_alfa / 3.F;

      y_a[0]  = yNBd; y_a[1]  = yNBd; y_a[2]  = yNBd; y_a[3]  = yNSd;
y_a[4]  = yPSd; y_a[5]  = yPSd; y_a[6]  = yPSd;
      y_a[7]  = yNBd; y_a[8]  = yNBd; y_a[9]  = yNBd; y_a[10] = yNSd;
y_a[11] = yPSd; y_a[12] = yPSd; y_a[13] = yPSd;
      y_a[14] = yNBd; y_a[15] = yNBd; y_a[16] = yNMd; y_a[17] = yNSd;
y_a[18] = yPSd; y_a[19] = yPSd; y_a[20] = yPMd;
      y_a[21] = yNBd; y_a[22] = yNMd; y_a[23] = yNSd; y_a[24] = yZ0d;
y_a[25] = yPSd; y_a[26] = yPMd; y_a[27] = yPBd;
      y_a[28] = yNMd; y_a[29] = yNSd; y_a[30] = yNSd; y_a[31] = yPSd;
y_a[32] = yPMd; y_a[33] = yPBd; y_a[33] = yPBd;
      y_a[35] = yNSd; y_a[36] = yNSd; y_a[37] = yNSd; y_a[38] = yPSd;
y_a[39] = yPBd; y_a[40] = yPBd; y_a[41] = yPBd;
      y_a[42] = yNSd; y_a[43] = yNSd; y_a[44] = yNSd; y_a[45] = yPSd;
y_a[46] = yPBd; y_a[47] = yPBd; y_a[48] = yPBd;
}
```

```
void inference(void)
{
      int i;
      register float ara_toplam = 0, w_toplam = 0;


      w[0] = NB < NBd ? NB : NBd;
      w[1] = NM < NBd ? NM : NBd;
      w[2] = NS < NBd ? NS : NBd;
      w[3] = Z0 < NBd ? Z0 : NBd;
      w[4] = PS < NBd ? PS : NBd;
      w[5] = PM < NBd ? PM : NBd;
      w[6] = PB < NBd ? PB : NBd;

      w[7] = NB < NMd ? NB : NMd;
      w[8] = NM < NMd ? NM : NMd;
      w[9] = NS < NMd ? NS : NMd;
      w[10] = Z0 < NMd ? Z0 : NMd;
      w[11] = PS < NMd ? PS : NMd;
      w[12] = PM < NMd ? PM : NMd;
      w[13] = PB < NMd ? PB : NMd;

      w[14] = NB < NSd ? NB : NSd;
      w[15] = NM < NSd ? NM : NSd;
      w[16] = NS < NSd ? NS : NSd;
      w[17] = Z0 < NSd ? Z0 : NSd;
      w[18] = PS < NSd ? PS : NSd;
      w[19] = PM < NSd ? PM : NSd;
      w[20] = PB < NSd ? PB : NSd;

      w[21] = NB < Z0d ? NB : Z0d;
      w[22] = NM < Z0d ? NM : Z0d;
      w[23] = NS < Z0d ? NS : Z0d;
      w[24] = Z0 < Z0d ? Z0 : Z0d;
      w[25] = PS < Z0d ? PS : Z0d;
      w[26] = PM < Z0d ? PM : Z0d;
      w[27] = PB < Z0d ? PB : Z0d;

      w[28] = NB < PSd ? NB : PSd;
      w[29] = NM < PSd ? NM : PSd;
      w[30] = NS < PSd ? NS : PSd;
      w[31] = Z0 < PSd ? Z0 : PSd;
      w[32] = PS < PSd ? PS : PSd;
      w[33] = PM < PSd ? PM : PSd;
      w[34] = PB < PSd ? PB : PSd;

      w[35] = NB < PMd ? NB : PMd;
      w[36] = NM < PMd ? NM : PMd;
      w[37] = NS < PMd ? NS : PMd;
      w[38] = Z0 < PMd ? Z0 : PMd;
      w[39] = PS < PMd ? PS : PMd;
      w[40] = PM < PMd ? PM : PMd;
      w[41] = PB < PMd ? PB : PMd;

      w[42] = NB < PBd ? NB : PBd;
      w[43] = NM < PBd ? NM : PBd;
      w[44] = NS < PBd ? NS : PBd;
      w[45] = Z0 < PBd ? Z0 : PBd;
      w[46] = PS < PBd ? PS : PBd;
      w[47] = PM < PBd ? PM : PBd;
```

```
	w[48] = PB < PBd ? PB : PBd;

	for (i = 0; i < 49; ++i)
		ara_toplam += w[i] * y[i];
	for (i = 0; i < 49; ++i)
		w_toplam += w[i];
	//printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
	u_bulanık = ara_toplam / w_toplam;
}

void inference_alfa(void)
{
	int i;
	register float ara_toplam = 0.F, w_toplam = 0.F;

	w_a[0] = NB1 < NBd1 ? NB1 : NBd1;
	w_a[1] = NM1 < NBd1 ? NM1 : NBd1;
	w_a[2] = NS1 < NBd1 ? NS1 : NBd1;
	w_a[3] = Z01 < NBd1 ? Z01 : NBd1;
	w_a[4] = PS1 < NBd1 ? PS1 : NBd1;
	w_a[5] = PM1 < NBd1 ? PM1 : NBd1;
	w_a[6] = PB1 < NBd1 ? PB1 : NBd1;

	w_a[7] = NB1 < NMd1 ? NB1 : NMd1;
	w_a[8] = NM1 < NMd1 ? NM1 : NMd1;
	w_a[9] = NS1 < NMd1 ? NS1 : NMd1;
	w_a[10] = Z01 < NMd1 ? Z01 : NMd1;
	w_a[11] = PS1 < NMd1 ? PS1 : NMd1;
	w_a[12] = PM1 < NMd1 ? PM1 : NMd1;
	w_a[13] = PB1 < NMd1 ? PB1 : NMd1;

	w_a[14] = NB1 < NSd1 ? NB1 : NSd1;
	w_a[15] = NM1 < NSd1 ? NM1 : NSd1;
	w_a[16] = NS1 < NSd1 ? NS1 : NSd1;
	w_a[17] = Z01 < NSd1 ? Z01 : NSd1;
	w_a[18] = PS1 < NSd1 ? PS1 : NSd1;
	w_a[19] = PM1 < NSd1 ? PM1 : NSd1;
	w_a[20] = PB1 < NSd1 ? PB1 : NSd1;

	w_a[21] = NB1 < Z0d1 ? NB1 : Z0d1;
	w_a[22] = NM1 < Z0d1 ? NM1 : Z0d1;
	w_a[23] = NS1 < Z0d1 ? NS1 : Z0d1;
	w_a[24] = Z01 < Z0d1 ? Z01 : Z0d1;
	w_a[25] = PS1 < Z0d1 ? PS1 : Z0d1;
	w_a[26] = PM1 < Z0d1 ? PM1 : Z0d1;
	w_a[27] = PB1 < Z0d1 ? PB1 : Z0d1;

	w_a[28] = NB1 < PSd1 ? NB1 : PSd1;
	w_a[29] = NM1 < PSd1 ? NM1 : PSd1;
	w_a[30] = NS1 < PSd1 ? NS1 : PSd1;
	w_a[31] = Z01 < PSd1 ? Z01 : PSd1;
	w_a[32] = PS1 < PSd1 ? PS1 : PSd1;
	w_a[33] = PM1 < PSd1 ? PM1 : PSd1;
	w_a[34] = PB1 < PSd1 ? PB1 : PSd1;

	w_a[35] = NB1 < PMd1 ? NB1 : PMd1;
	w_a[36] = NM1 < PMd1 ? NM1 : PMd1;
	w_a[37] = NS1 < PMd1 ? NS1 : PMd1;
	w_a[38] = Z01 < PMd1 ? Z01 : PMd1;
	w_a[39] = PS1 < PMd1 ? PS1 : PMd1;
	w_a[40] = PM1 < PMd1 ? PM1 : PMd1;
	w_a[41] = PB1 < PMd1 ? PB1 : PMd1;
```

```
      w_a[42] = NB1 < PBd1 ? NB1 : PBd1;
      w_a[43] = NM1 < PBd1 ? NM1 : PBd1;
      w_a[44] = NS1 < PBd1 ? NS1 : PBd1;
      w_a[45] = Z01 < PBd1 ? Z01 : PBd1;
      w_a[46] = PS1 < PBd1 ? PS1 : PBd1;
      w_a[47] = PM1 < PBd1 ? PM1 : PBd1;
      w_a[48] = PB1 < PBd1 ? PB1 : PBd1;

      for (i = 0; i < 49; ++i)
            ara_toplam += w_a[i] * y_a[i];
      for (i = 0; i < 49; ++i)
            w_toplam += w_a[i];
      //printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
      alfa = ara_toplam / w_toplam;
}

void fuzzification1 (float error)
{
      PB1 = 0.f; PM1 = 0.f; PS1 = 0.f; Z01 = 0.f; NS1 = 0.f; NM1 = 0.f; NB1
= 0.f;

      if (error >= EM)
            PB1 = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB1 = (3.f / EM * error) - 2;
            PM1 = (-3.f / EM * error) + 3;
      }
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM1 = (3.f / EM * error) - 1;
            PS1 = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS1 = 3.f / EM * error;
            Z01 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS1 = (-3.f / EM * error);
            Z01 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS1 = (3.f / EM * error) + 2;
            NM1 = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM1 = (3.f / EM * error) + 3;
            NB1 = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB1 = 1;
}

void fuzzification(float error)
{
      PB = 0.f; PM = 0.f; PS = 0.f; Z0 = 0.f; NS = 0.f; NM = 0.f; NB = 0.f;

      if (error >= EM)
            PB = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB = (3.f / EM * error) - 2;
            PM = (-3.f / EM * error) + 3;
      }
```

```
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM = (3.f / EM * error) - 1;
            PS = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS = 3.f / EM * error;
            Z0 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS = (-3.f / EM * error);
            Z0 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS = (3.f / EM * error) + 2;
            NM = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM = (3.f / EM * error) + 3;
            NB = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB = 1;
}

void main (void)

{
      FILE *f1, *data;

      register float itae = 0.F, max_ovrsht, error_rate, sinyal, kont,
kont_pi = 0.F;
      register int      m, hiz, k = 0, i;
      register long top_error = 0L, top_hata = 0L, t, hata, hata1= 0L,
ara_hiz = 0L;
      int max_hiz = 0, read_hiz[30] = {0}, error[3000], ref[3000],
speed[3000];

      init();
      init_alfa();

KONTROL:

      while (k < 6000) {
                  disable();
                  ara_hiz = 0L;
                  read_hiz[0] = adcoku();
                  for (i = 0; i < 30; ++i)
                        ara_hiz += read_hiz[i];
                  hiz = ara_hiz / 30;
                  if ((k < 3000) && (hiz > max_hiz))
                        max_hiz = hiz;
                  hata = REF - hiz;
                  if (k < 3000) {
                        speed[k] = hiz;
                        error[k] = hata;
                        ref[k] = hata + hiz;
                        top_hata += k * abs(hata);

                  }
                  error_rate = (float)(hata - hata1) * Gce * (1.F / Ts);
                  fuzzification(hata * Ge);
                  fuzzification1(hata * Ge);
```

```
                fuzzification_errate(error_rate);

        fuzzification_errate1(error_rate);
                inference();
                inference_alfa();

                kont_pi += u_bulanık * Ts * K1;
                kont = kont_pi;
                for (i = 29; 0 < i ; --i)
                      read_hiz[i] = read_hiz[i-1];
                hata1 = hata;
                k++;
                for (i = 0; i < kont; i += 4) {
                      outp(0x378, 1);
                      if (!(inp(0x379) & 0x20))
                            goto KONTROL;
                }
                outp(0x378, 0);
                while ((inp(0x379) & 0x20))
                      ;
                goto KONTROL;
        }
        enable();
        max_ovrsht = ((float) max_hiz - REF) / REF;
        outp(0x378,0);
        clrscr();
// real time control islemi bitince sonuclarin ilgili dizilerden
// asagidaki gibi tanimli .dat uzantili bir dosyaya kaydedilmesi

        if ((f1 = fopen(DOSYA, "w+")) == NULL) {
              printf("Can not open file!..\n");
              exit(1);
        }
#if TRAIN
        if ((data = fopen("datapi.dat", "a+")) == NULL) {
              printf("Can not open file!..\n");
              exit(1);
        }
#endif
        for (k=0; k < 3000; ++k) {
              top_error += abs(error[k]);
              itae += (float) k * 0.0005 * abs(error[k]);
              fprintf(f1, "%d\t%d\t%d\n", speed[k], error[k], ref[k]);
        }
        for (k=2950; k < 3000; ++k)
              printf("hiz: %d\n", speed[k]);
        printf("IAE =%ld\n", top_error);
     // printf("yukselme zamani = %f\n", (t_rise / 3000.f));
        printf("max =%f\n", max_ovrsht);
        printf("ITAE =%f\n", itae);
#if TRAIN
        fprintf(data, "%f\n", Ge);
        fprintf(data, "%f\n", Gce);
        fprintf(data, "%f\n", (float) (top_error - 70000) / (400000 -
70000));
        fprintf(data, "%f\n", max_ovrsht);
        fprintf(data, "%f\n", (itae - 5000) / (50000 - 5000));
        fclose(data);
#endif
        fclose(f1);
        getch();
}
```

Ek 5 Nöral-Genetik Tabanlı Optimal Bulanık PID Kontrolörü

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#define     TRAIN       1
#define     DOSYA       "fpid05.dat"
#define     REF         600    //2390 devir/dk
#define     BASE        0x280
#define     K1          400.
#define     K2          0.1
#define     Ts          0.0005
#define     EM          20.F
#define     EMd         20.F
#define     OM          20.F
#define     OM_alfa     10.F1
#define     Ge          .1
#define     Gce         .001

// global degiskenler, uyelik fonksiyonlari

float w[49]; // inference agirliklari
float w_a[49]; // inference alfa agirliklari
float y_a[49];
float y[49];
float PB, PM, PS, Z0, NS, NM, NB;
float PB1, PM1, PS1, Z01, NS1, NM1, NB1;
float PBd, PMd, PSd, Z0d, NSd, NMd, NBd;
float PBd1, PMd1, PSd1, Z0d1, NSd1, NMd1, NBd1;
float yPB, yPM, yPS, yZ0, yNS, yNM, yNB;
float yPBd, yPMd, yPSd, yZ0d, yNSd, yNMd, yNBd;
float u_bulanık, alfa;

void init(void);
void init_alfa(void);
void inference (void);
void inference_alfa (void);
void fuzzification (float error);
void fuzzification1 (float error);
void fuzzification_errate1 (float error_rate);
void fuzzification_errate (float error_rate);
int adcoku (void);

int adcoku (void)
{
      int tempLSB, tempMSB, x;

      outportb(BASE + 16, 15);
      while (!((inportb(BASE + 20) & 0x01)))
            ;
      tempMSB = inportb(BASE + 18);
      tempLSB = inportb(BASE + 19) & 0x0C;
      x = tempMSB << 4;
      x += tempLSB;
      return x - 1888;
}


void fuzzification_errate (float error_rate)
```

```
{
      PBd = 0.f; PMd = 0.f; PSd = 0.f; Z0d = 0.f; NSd = 0.f; NMd = 0.f;
      NBd = 0.f;


      if (error_rate >= EMd)
            PBd = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd = (3.f / EMd * error_rate) - 2;
            PMd = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd = (3.f / EMd * error_rate) - 1;
            PSd = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd = 3.f / EMd * error_rate;
            Z0d = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd = (-3.f / EMd * error_rate);
            Z0d = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd = (3.f / EMd * error_rate) + 2;
            NMd = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd = (3.f / EMd * error_rate) + 3;
            NBd = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd = 1;
}

void fuzzification_errate1 (float error_rate)
{
      PBd1 = 0.f; PMd1 = 0.f; PSd1 = 0.f; Z0d1 = 0.f; NSd1 = 0.f; NMd1 =
0.f;
      NBd1 = 0.f;


      if (error_rate >= EMd)
            PBd1 = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd1 = (3.f / EMd * error_rate) - 2;
            PMd1 = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd1 = (3.f / EMd * error_rate) - 1;
            PSd1 = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd1 = 3.f / EMd * error_rate;
            Z0d1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd1 = (-3.f / EMd * error_rate);
            Z0d1 = (3. / EMd * error_rate) + 1;
```

```
	}
	else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
		NSd1 = (3.f / EMd * error_rate) + 2;
		NMd1 = (-3.f / EMd * error_rate) + 1;
	}
	else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
		NMd1 = (3.f / EMd * error_rate) + 3;
		NBd1 = (-3.f / EMd * error_rate) - 2;
	}
	else if (error_rate < - EMd)
		NBd1 = 1;
}

void init(void)
//kural matrisi main kontroler icin
{

	yNB = - OM; yNM = -(2 * OM / 3.F); yNS = -( OM / 3.F); yZ0 = 0.F;
	yPB = OM; yPM = (2 * OM / 3.F); yPS = OM / 3.F;

	y[0]  = yNB;  y[1] = yNB; y[2]  = yNB; y[3]  = yNB; y[4]  = yNM; y[5]
= yNS; y[6]  = yZ0;
	y[7]  = yNB;  y[8] = yNB; y[9]  = yNB; y[10] = yNM; y[11] = yNS;
y[12] = yZ0; y[13] = yPS;
	y[14] = yNB; y[15] = yNB; y[16] = yNM; y[17] = yNS; y[18] = yZ0;
y[19] = yPS; y[20] = yPM;
	y[21] = yNB; y[22] = yNM; y[23] = yNS; y[24] = yZ0; y[25] = yPS;
y[26] = yPM; y[27] = yPB;
	y[28] = yNM; y[29] = yNS; y[30] = yZ0; y[31] = yPS; y[32] = yPM;
y[33] = yPB; y[33] = yPB;
	y[35] = yNS; y[36] = yZ0; y[37] = yPS; y[38] = yPM; y[39] = yPB;
y[40] = yPB; y[41] = yPB;
	y[42] = yZ0; y[43] = yPS; y[44] = yPM; y[45] = yPB; y[46] = yPB;
y[47] = yPB; y[48] = yPB;
}

void init_alfa(void)
//kural matrisi alfa kontroler icin
{

	yNBd = - OM_alfa; yNMd = -(2 * OM_alfa / 3.F); yNSd = -( OM_alfa /
3.F); yZ0d = 0.F;
	yPBd = OM_alfa; yPMd = (2 * OM_alfa / 3.F); yPSd = OM_alfa / 3.F;

	y_a[0]  = yNBd; y_a[1]  = yNBd; y_a[2]  = yNBd; y_a[3]  = yNSd;
y_a[4]  = yPSd; y_a[5]  = yPSd; y_a[6]  = yPSd;
	y_a[7]  = yNBd; y_a[8]  = yNBd; y_a[9]  = yNBd; y_a[10] = yNSd;
y_a[11] = yPSd; y_a[12] = yPSd; y_a[13] = yPSd;
	y_a[14] = yNBd; y_a[15] = yNBd; y_a[16] = yNMd; y_a[17] = yNSd;
y_a[18] = yPSd; y_a[19] = yPSd; y_a[20] = yPMd;
	y_a[21] = yNBd; y_a[22] = yNMd; y_a[23] = yNSd; y_a[24] = yZ0d;
y_a[25] = yPSd; y_a[26] = yPMd; y_a[27] = yPBd;
	y_a[28] = yNMd; y_a[29] = yNSd; y_a[30] = yNSd; y_a[31] = yPSd;
y_a[32] = yPMd; y_a[33] = yPBd; y_a[33] = yPBd;
	y_a[35] = yNSd; y_a[36] = yNSd; y_a[37] = yNSd; y_a[38] = yPSd;
y_a[39] = yPBd; y_a[40] = yPBd; y_a[41] = yPBd;
	y_a[42] = yNSd; y_a[43] = yNSd; y_a[44] = yNSd; y_a[45] = yPSd;
y_a[46] = yPBd; y_a[47] = yPBd; y_a[48] = yPBd;
}
```

```
void inference(void)
{
      int i;
      register float ara_toplam = 0, w_toplam = 0;


      w[0] = NB < NBd ? NB : NBd;
      w[1] = NM < NBd ? NM : NBd;
      w[2] = NS < NBd ? NS : NBd;
      w[3] = Z0 < NBd ? Z0 : NBd;
      w[4] = PS < NBd ? PS : NBd;
      w[5] = PM < NBd ? PM : NBd;
      w[6] = PB < NBd ? PB : NBd;

      w[7] = NB < NMd ? NB : NMd;
      w[8] = NM < NMd ? NM : NMd;
      w[9] = NS < NMd ? NS : NMd;
      w[10] = Z0 < NMd ? Z0 : NMd;
      w[11] = PS < NMd ? PS : NMd;
      w[12] = PM < NMd ? PM : NMd;
      w[13] = PB < NMd ? PB : NMd;

      w[14] = NB < NSd ? NB : NSd;
      w[15] = NM < NSd ? NM : NSd;
      w[16] = NS < NSd ? NS : NSd;
      w[17] = Z0 < NSd ? Z0 : NSd;
      w[18] = PS < NSd ? PS : NSd;
      w[19] = PM < NSd ? PM : NSd;
      w[20] = PB < NSd ? PB : NSd;

      w[21] = NB < Z0d ? NB : Z0d;
      w[22] = NM < Z0d ? NM : Z0d;
      w[23] = NS < Z0d ? NS : Z0d;
      w[24] = Z0 < Z0d ? Z0 : Z0d;
      w[25] = PS < Z0d ? PS : Z0d;
      w[26] = PM < Z0d ? PM : Z0d;
      w[27] = PB < Z0d ? PB : Z0d;

      w[28] = NB < PSd ? NB : PSd;
      w[29] = NM < PSd ? NM : PSd;
      w[30] = NS < PSd ? NS : PSd;
      w[31] = Z0 < PSd ? Z0 : PSd;
      w[32] = PS < PSd ? PS : PSd;
      w[33] = PM < PSd ? PM : PSd;
      w[34] = PB < PSd ? PB : PSd;

      w[35] = NB < PMd ? NB : PMd;
      w[36] = NM < PMd ? NM : PMd;
      w[37] = NS < PMd ? NS : PMd;
      w[38] = Z0 < PMd ? Z0 : PMd;
      w[39] = PS < PMd ? PS : PMd;
      w[40] = PM < PMd ? PM : PMd;
      w[41] = PB < PMd ? PB : PMd;

      w[42] = NB < PBd ? NB : PBd;
      w[43] = NM < PBd ? NM : PBd;
      w[44] = NS < PBd ? NS : PBd;
      w[45] = Z0 < PBd ? Z0 : PBd;
      w[46] = PS < PBd ? PS : PBd;
      w[47] = PM < PBd ? PM : PBd;
      w[48] = PB < PBd ? PB : PBd;
```

```
	for (i = 0; i < 49; ++i)
		ara_toplam += w[i] * y[i];
	for (i = 0; i < 49; ++i)
		w_toplam += w[i];
	//printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
	u_bulanık = ara_toplam / w_toplam;
}

void inference_alfa(void)
{
	int i;
	register float ara_toplam = 0.F, w_toplam = 0.F;


	w_a[0] = NB1 < NBd1 ? NB1 : NBd1;
	w_a[1] = NM1 < NBd1 ? NM1 : NBd1;
	w_a[2] = NS1 < NBd1 ? NS1 : NBd1;
	w_a[3] = Z01 < NBd1 ? Z01 : NBd1;
	w_a[4] = PS1 < NBd1 ? PS1 : NBd1;
	w_a[5] = PM1 < NBd1 ? PM1 : NBd1;
	w_a[6] = PB1 < NBd1 ? PB1 : NBd1;

	w_a[7] = NB1 < NMd1 ? NB1 : NMd1;
	w_a[8] = NM1 < NMd1 ? NM1 : NMd1;
	w_a[9] = NS1 < NMd1 ? NS1 : NMd1;
	w_a[10] = Z01 < NMd1 ? Z01 : NMd1;
	w_a[11] = PS1 < NMd1 ? PS1 : NMd1;
	w_a[12] = PM1 < NMd1 ? PM1 : NMd1;
	w_a[13] = PB1 < NMd1 ? PB1 : NMd1;

	w_a[14] = NB1 < NSd1 ? NB1 : NSd1;
	w_a[15] = NM1 < NSd1 ? NM1 : NSd1;
	w_a[16] = NS1 < NSd1 ? NS1 : NSd1;
	w_a[17] = Z01 < NSd1 ? Z01 : NSd1;
	w_a[18] = PS1 < NSd1 ? PS1 : NSd1;
	w_a[19] = PM1 < NSd1 ? PM1 : NSd1;
	w_a[20] = PB1 < NSd1 ? PB1 : NSd1;

	w_a[21] = NB1 < Z0d1 ? NB1 : Z0d1;
	w_a[22] = NM1 < Z0d1 ? NM1 : Z0d1;
	w_a[23] = NS1 < Z0d1 ? NS1 : Z0d1;
	w_a[24] = Z01 < Z0d1 ? Z01 : Z0d1;
	w_a[25] = PS1 < Z0d1 ? PS1 : Z0d1;
	w_a[26] = PM1 < Z0d1 ? PM1 : Z0d1;
	w_a[27] = PB1 < Z0d1 ? PB1 : Z0d1;

	w_a[28] = NB1 < PSd1 ? NB1 : PSd1;
	w_a[29] = NM1 < PSd1 ? NM1 : PSd1;
	w_a[30] = NS1 < PSd1 ? NS1 : PSd1;
	w_a[31] = Z01 < PSd1 ? Z01 : PSd1;
	w_a[32] = PS1 < PSd1 ? PS1 : PSd1;
	w_a[33] = PM1 < PSd1 ? PM1 : PSd1;
	w_a[34] = PB1 < PSd1 ? PB1 : PSd1;

	w_a[35] = NB1 < PMd1 ? NB1 : PMd1;
	w_a[36] = NM1 < PMd1 ? NM1 : PMd1;
	w_a[37] = NS1 < PMd1 ? NS1 : PMd1;
	w_a[38] = Z01 < PMd1 ? Z01 : PMd1;
	w_a[39] = PS1 < PMd1 ? PS1 : PMd1;
	w_a[40] = PM1 < PMd1 ? PM1 : PMd1;
	w_a[41] = PB1 < PMd1 ? PB1 : PMd1;
```

```
      w_a[42] = NB1 < PBd1 ? NB1 : PBd1;
      w_a[43] = NM1 < PBd1 ? NM1 : PBd1;
      w_a[44] = NS1 < PBd1 ? NS1 : PBd1;
      w_a[45] = Z01 < PBd1 ? Z01 : PBd1;
      w_a[46] = PS1 < PBd1 ? PS1 : PBd1;
      w_a[47] = PM1 < PBd1 ? PM1 : PBd1;
      w_a[48] = PB1 < PBd1 ? PB1 : PBd1;

      for (i = 0; i < 49; ++i)
            ara_toplam += w_a[i] * y_a[i];
      for (i = 0; i < 49; ++i)
            w_toplam += w_a[i];
      //printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
      alfa = ara_toplam / w_toplam;
}


void fuzzification1 (float error)
{
      PB1 = 0.f; PM1 = 0.f; PS1 = 0.f; Z01 = 0.f; NS1 = 0.f; NM1 = 0.f; NB1
= 0.f;


      if (error >= EM)
            PB1 = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB1 = (3.f / EM * error) - 2;
            PM1 = (-3.f / EM * error) + 3;
      }
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM1 = (3.f / EM * error) - 1;
            PS1 = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS1 = 3.f / EM * error;
            Z01 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS1 = (-3.f / EM * error);
            Z01 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS1 = (3.f / EM * error) + 2;
            NM1 = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM1 = (3.f / EM * error) + 3;
            NB1 = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB1 = 1;
}

void fuzzification(float error)
{
      PB = 0.f; PM = 0.f; PS = 0.f; Z0 = 0.f; NS = 0.f; NM = 0.f; NB = 0.f;


      if (error >= EM)
            PB = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB = (3.f / EM * error) - 2;
```

```
            PM = (-3.f / EM * error) + 3;
      }
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM = (3.f / EM * error) - 1;
            PS = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS = 3.f / EM * error;
            Z0 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS = (-3.f / EM * error);
            Z0 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS = (3.f / EM * error) + 2;
            NM = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM = (3.f / EM * error) + 3;
            NB = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB = 1;
}

void main (void)

{
      FILE *f1, *data;

      register float itae = 0.F, max_ovrsht, error_rate, sinyal, kont,
kont_pi = 0.F;
      register int      m, hiz, k = 0, i;
      register long top_error = 0L, top_hata = 0L, t, hata, hata1= 0L,
ara_hiz = 0L;
      int max_hiz = 0, read_hiz[30] = {0}, error[3000], ref[3000],
speed[3000];


      init();
      init_alfa();

KONTROL:

      while (k < 6000) {
                  disable();
                  ara_hiz = 0L;
                  read_hiz[0] = adcoku();
                  for (i = 0; i < 30; ++i)
                        ara_hiz += read_hiz[i];
                  hiz = ara_hiz / 30;
                  if ((k < 3000) && (hiz > max_hiz))
                        max_hiz = hiz;
                  if ((k < 1000) || (k > 2000))
                        hata = REF - hiz;
                  else
                        hata = REF - hiz;
                  if (k < 3000) {
                        speed[k] = hiz;
                        error[k] = hata;
                        ref[k] = hata + hiz;
```

```
                        top_hata += k * abs(hata);

                    }

                    error_rate = (float)(hata - hata1) * Gce * (1.F / Ts);
                    fuzzification(hata * Ge);
                    fuzzification1(hata * Ge);
                    fuzzification_errate(error_rate);
                    fuzzification_errate1(error_rate);
                    inference();
                    inference_alfa();

                    kont_pi += u_bulanık * Ts * K1;
                    kont = (K2 * alfa) + kont_pi;
                    for (i = 29; 0 < i ; --i)
                          read_hiz[i] = read_hiz[i-1];
                    hata1 = hata;
                    k++;
                    for (i = 0; i < kont; i += 4) {
                          outp(0x378, 1);
                          if (!(inp(0x379) & 0x20))
                                goto KONTROL;
                    }
                    outp(0x378, 0);
                    while ((inp(0x379) & 0x20))
                          ;
                    goto KONTROL;
      }
      enable();
      max_ovrsht = ((float) max_hiz - REF) / REF;
      outp(0x378,0);

      clrscr();
// real time control islemi bitince sonuclarin ilgili dizilerden
// asagidaki gibi tanimli .dat uzantili bir dosyaya kaydedilmesi

      if ((f1 = fopen(DOSYA, "w+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#if TRAIN
      if ((data = fopen("datapid.dat", "a+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#endif
      for (k=0; k < 3000; ++k) {
            top_error += abs(error[k]);
            itae += (float) k * 0.0005 * abs(error[k]);
            fprintf(f1, "%d\t%d\t%d\n", speed[k], error[k], ref[k]);
      }
      for (k=2950; k < 3000; ++k)
            printf("hiz: %d\n", speed[k]);
      printf("IAE =%ld\n", top_error);
   // printf("yukselme zamani = %f\n", (t_rise / 3000.f));
      printf("max =%f\n", max_ovrsht);
      printf("ITAE =%f\n", itae);
#if TRAIN
      fprintf(data, "%f\n", Ge);
      fprintf(data, "%f\n", Gce);
      fprintf(data, "%f\n", (float) (top_error - 70000) / (400000 -
70000));
```

```
      fprintf(data, "%f\n", max_ovrsht);
      fprintf(data, "%f\n", (itae - 5000) / (70000 - 5000));
      fclose(data);
#endif
      fclose(f1);
      getch();
}
```

Ek 6 Nöral-Genetik Tabanlı Optimal Öz-Uyarlamalı Bulanık PI Kontrolörü

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>

#define     TRAIN       1
#define     DOSYA       "stflc43.dat"
#define     REF         600         //2390 devir/dk
#define     BASE        0x280
#define     EM          20.F
#define     EMd         20.F
#define     OM          20.F
#define     OM_alfa     10.F
#define     Ge          .15
#define     Gce         001F
#define     Gu          .038


// global degiskenler, uyelik fonksiyonlari

float w[49]; // inference agirliklari
float w_a[49]; // inference alfa agirliklari
float y_a[49];
float y[49];
float PB, PM, PS, Z0, NS, NM, NB;
float PB1, PM1, PS1, Z01, NS1, NM1, NB1;
float PBd, PMd, PSd, Z0d, NSd, NMd, NBd;
float PBd1, PMd1, PSd1, Z0d1, NSd1, NMd1, NBd1;
float yPB, yPM, yPS, yZ0, yNS, yNM, yNB;
float VBa, Ba, MBa, SBa, Sa, VSa, Z0a;
float u_bulanık, alfa;

void init(void);
void init_alfa(void);
void inference (void);
void inference_alfa (void);
void fuzzification (float error);
void fuzzification1 (float error);
void fuzzification_errate1 (float error_rate);
void fuzzification_errate (float error_rate);
int adcoku (void);

int adcoku (void)
{
      int tempLSB, tempMSB, x;

      outportb(BASE + 16, 15);
      while (!((inportb(BASE + 20) & 0x01)))
            ;
      tempMSB = inportb(BASE + 18);
      tempLSB = inportb(BASE + 19) & 0x0C;
      x = tempMSB << 4;
      x += tempLSB;
      return x - 1888;
}
```

```
void fuzzification_errate (float error_rate)
{
      PBd = 0.f; PMd = 0.f; PSd = 0.f; Z0d = 0.f; NSd = 0.f; NMd = 0.f;
      NBd = 0.f;

      if (error_rate >= EMd)
            PBd = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd = (3.f / EMd * error_rate) - 2;
            PMd = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd = (3.f / EMd * error_rate) - 1;
            PSd = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd = 3.f / EMd * error_rate;
            Z0d = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd = (-3.f / EMd * error_rate);
            Z0d = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd = (3.f / EMd * error_rate) + 2;
            NMd = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd = (3.f / EMd * error_rate) + 3;
            NBd = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd = 1;
}
void fuzzification_errate1 (float error_rate)
{
      PBd1 = 0.f; PMd1 = 0.f; PSd1 = 0.f; Z0d1 = 0.f; NSd1 = 0.f; NMd1 =
0.f;
      NBd1 = 0.f;


      if (error_rate >= EMd)
            PBd1 = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd1 = (3.f / EMd * error_rate) - 2;
            PMd1 = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd1 = (3.f / EMd * error_rate) - 1;
            PSd1 = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd1 = 3.f / EMd * error_rate;
            Z0d1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd1 = (-3.f / EMd * error_rate);
            Z0d1 = (3. / EMd * error_rate) + 1;
      }
```

```
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd1 = (3.f / EMd * error_rate) + 2;
            NMd1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd1 = (3.f / EMd * error_rate) + 3;
            NBd1 = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd1 = 1;
}

void init(void)
//kural matrisi main kontroler icin
{

      yNB = - OM; yNM = -(2 * OM / 3.F); yNS = -( OM / 3.F); yZ0 = 0.F;
      yPB = OM; yPM = (2 * OM / 3.F); yPS = OM / 3.F;

      y[0]  = yNB;  y[1] = yNB; y[2]  = yNB; y[3]  = yNB; y[4]  = yNM; y[5]
= yNS; y[6]  = yZ0;
      y[7]  = yNB;  y[8] = yNB; y[9]  = yNB; y[10] = yNM; y[11] = yNS;
y[12] = yZ0; y[13] = yPS;
      y[14] = yNB; y[15] = yNB; y[16] = yNM; y[17] = yNS; y[18] = yZ0;
y[19] = yPS; y[20] = yPM;
      y[21] = yNB; y[22] = yNM; y[23] = yNS; y[24] = yZ0; y[25] = yPS;
y[26] = yPM; y[27] = yPB;
      y[28] = yNM; y[29] = yNS; y[30] = yZ0; y[31] = yPS; y[32] = yPM;
y[33] = yPM; y[34] = yPB;
      y[35] = yNS; y[36] = yZ0; y[37] = yPS; y[38] = yPM; y[39] = yPB;
y[40] = yPB; y[41] = yPB;
      y[42] = yZ0; y[43] = yPS; y[44] = yPM; y[45] = yPB; y[46] = yPB;
y[47] = yPB; y[48] = yPB;
}

void init_alfa(void)
//kural matrisi alfa kontroler icin
{

      VBa = OM_alfa; Ba = (5 * OM_alfa / 6.F); MBa = (4 * OM_alfa / 6.F);
      SBa = (3 * OM_alfa / 6.F); Sa = (2 * OM_alfa / 6.F);
      VSa = OM_alfa / 6.F; Z0a = 0.F;

      y_a[0]  = VBa; y_a[1]  =  Ba;  y_a[2]  = MBa; y_a[3]  = SBa;  y_a[4]
= Sa;  y_a[5] = VSa;  y_a[6]  = Z0a;
      y_a[7]  = Ba;  y_a[8]  =  MBa; y_a[9]  = SBa; y_a[10] = Sa;   y_a[11]
= VSa; y_a[12] = Z0a; y_a[13] = VSa;
      y_a[14] = MBa; y_a[15] =  SBa; y_a[16] = Sa;  y_a[17] = VSa;  y_a[18]
= Z0a; y_a[19] = VSa;  y_a[20] = Sa;
      y_a[21] = SBa;  y_a[22] =  Sa; y_a[23] = VSa; y_a[24] = Z0a;  y_a[25]
= VSa; y_a[26] = Sa; y_a[27] = SBa;
      y_a[28] = SBa; y_a[29] =  VSa;  y_a[30] = Z0a; y_a[31] = VSa;
y_a[32] = Sa;  y_a[33] = SBa; y_a[34] = MBa;
      y_a[35] = VSa; y_a[36] =  Z0a;  y_a[37] = VSa; y_a[38] = Sa;
y_a[39] = SBa; y_a[40] =  MBa; y_a[41] = Ba;
      y_a[42] = Z0a; y_a[43] =  VSa;  y_a[44] = Sa; y_a[45] = SBa;
y_a[46] = MBa; y_a[47] = Ba; y_a[48] = VBa;
}


void inference(void)
```

```
{
        int i;
        register float ara_toplam = 0, w_toplam = 0;


        w[0] = NB < NBd ? NB : NBd;
        w[1] = NM < NBd ? NM : NBd;
        w[2] = NS < NBd ? NS : NBd;
        w[3] = Z0 < NBd ? Z0 : NBd;
        w[4] = PS < NBd ? PS : NBd;
        w[5] = PM < NBd ? PM : NBd;
        w[6] = PB < NBd ? PB : NBd;

        w[7] = NB < NMd ? NB : NMd;
        w[8] = NM < NMd ? NM : NMd;
        w[9] = NS < NMd ? NS : NMd;
        w[10] = Z0 < NMd ? Z0 : NMd;
        w[11] = PS < NMd ? PS : NMd;
        w[12] = PM < NMd ? PM : NMd;
        w[13] = PB < NMd ? PB : NMd;

        w[14] = NB < NSd ? NB : NSd;
        w[15] = NM < NSd ? NM : NSd;
        w[16] = NS < NSd ? NS : NSd;
        w[17] = Z0 < NSd ? Z0 : NSd;
        w[18] = PS < NSd ? PS : NSd;
        w[19] = PM < NSd ? PM : NSd;
        w[20] = PB < NSd ? PB : NSd;

        w[21] = NB < Z0d ? NB : Z0d;
        w[22] = NM < Z0d ? NM : Z0d;
        w[23] = NS < Z0d ? NS : Z0d;
        w[24] = Z0 < Z0d ? Z0 : Z0d;
        w[25] = PS < Z0d ? PS : Z0d;
        w[26] = PM < Z0d ? PM : Z0d;
        w[27] = PB < Z0d ? PB : Z0d;

        w[28] = NB < PSd ? NB : PSd;
        w[29] = NM < PSd ? NM : PSd;
        w[30] = NS < PSd ? NS : PSd;
        w[31] = Z0 < PSd ? Z0 : PSd;
        w[32] = PS < PSd ? PS : PSd;
        w[33] = PM < PSd ? PM : PSd;
        w[34] = PB < PSd ? PB : PSd;

        w[35] = NB < PMd ? NB : PMd;
        w[36] = NM < PMd ? NM : PMd;
        w[37] = NS < PMd ? NS : PMd;
        w[38] = Z0 < PMd ? Z0 : PMd;
        w[39] = PS < PMd ? PS : PMd;
        w[40] = PM < PMd ? PM : PMd;
        w[41] = PB < PMd ? PB : PMd;

        w[42] = NB < PBd ? NB : PBd;
        w[43] = NM < PBd ? NM : PBd;
        w[44] = NS < PBd ? NS : PBd;
        w[45] = Z0 < PBd ? Z0 : PBd;
        w[46] = PS < PBd ? PS : PBd;
        w[47] = PM < PBd ? PM : PBd;
        w[48] = PB < PBd ? PB : PBd;

        for (i = 0; i < 49; ++i)
```

```
            ara_toplam += w[i] * y[i];
      for (i = 0; i < 49; ++i)
            w_toplam += w[i];
      //printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
      u_bulanık = ara_toplam / w_toplam;
}

void inference_alfa(void)
{
      int i;
      register float ara_toplam = 0.F, w_toplam = 0.F;


      w_a[0] = NB1 < NBd1 ? NB1 : NBd1;
      w_a[1] = NM1 < NBd1 ? NM1 : NBd1;
      w_a[2] = NS1 < NBd1 ? NS1 : NBd1;
      w_a[3] = Z01 < NBd1 ? Z01 : NBd1;
      w_a[4] = PS1 < NBd1 ? PS1 : NBd1;
      w_a[5] = PM1 < NBd1 ? PM1 : NBd1;
      w_a[6] = PB1 < NBd1 ? PB1 : NBd1;

      w_a[7] = NB1 < NMd1 ? NB1 : NMd1;
      w_a[8] = NM1 < NMd1 ? NM1 : NMd1;
      w_a[9] = NS1 < NMd1 ? NS1 : NMd1;
      w_a[10] = Z01 < NMd1 ? Z01 : NMd1;
      w_a[11] = PS1 < NMd1 ? PS1 : NMd1;
      w_a[12] = PM1 < NMd1 ? PM1 : NMd1;
      w_a[13] = PB1 < NMd1 ? PB1 : NMd1;

      w_a[14] = NB1 < NSd1 ? NB1 : NSd1;
      w_a[15] = NM1 < NSd1 ? NM1 : NSd1;
      w_a[16] = NS1 < NSd1 ? NS1 : NSd1;
      w_a[17] = Z01 < NSd1 ? Z01 : NSd1;
      w_a[18] = PS1 < NSd1 ? PS1 : NSd1;
      w_a[19] = PM1 < NSd1 ? PM1 : NSd1;
      w_a[20] = PB1 < NSd1 ? PB1 : NSd1;

      w_a[21] = NB1 < Z0d1 ? NB1 : Z0d1;
      w_a[22] = NM1 < Z0d1 ? NM1 : Z0d1;
      w_a[23] = NS1 < Z0d1 ? NS1 : Z0d1;
      w_a[24] = Z01 < Z0d1 ? Z01 : Z0d1;
      w_a[25] = PS1 < Z0d1 ? PS1 : Z0d1;
      w_a[26] = PM1 < Z0d1 ? PM1 : Z0d1;
      w_a[27] = PB1 < Z0d1 ? PB1 : Z0d1;

      w_a[28] = NB1 < PSd1 ? NB1 : PSd1;
      w_a[29] = NM1 < PSd1 ? NM1 : PSd1;
      w_a[30] = NS1 < PSd1 ? NS1 : PSd1;
      w_a[31] = Z01 < PSd1 ? Z01 : PSd1;
      w_a[32] = PS1 < PSd1 ? PS1 : PSd1;
      w_a[33] = PM1 < PSd1 ? PM1 : PSd1;
      w_a[34] = PB1 < PSd1 ? PB1 : PSd1;

      w_a[35] = NB1 < PMd1 ? NB1 : PMd1;
      w_a[36] = NM1 < PMd1 ? NM1 : PMd1;
      w_a[37] = NS1 < PMd1 ? NS1 : PMd1;
      w_a[38] = Z01 < PMd1 ? Z01 : PMd1;
      w_a[39] = PS1 < PMd1 ? PS1 : PMd1;
      w_a[40] = PM1 < PMd1 ? PM1 : PMd1;
      w_a[41] = PB1 < PMd1 ? PB1 : PMd1;

      w_a[42] = NB1 < PBd1 ? NB1 : PBd1;
```

```
      w_a[43] = NM1 < PBd1 ? NM1 : PBd1;
      w_a[44] = NS1 < PBd1 ? NS1 : PBd1;
      w_a[45] = Z01 < PBd1 ? Z01 : PBd1;
      w_a[46] = PS1 < PBd1 ? PS1 : PBd1;
      w_a[47] = PM1 < PBd1 ? PM1 : PBd1;
      w_a[48] = PB1 < PBd1 ? PB1 : PBd1;

      for (i = 0; i < 49; ++i)
            ara_toplam += w_a[i] * y_a[i];
      for (i = 0; i < 49; ++i)
            w_toplam += w_a[i];
      //printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
      alfa = ara_toplam / w_toplam;
}


void fuzzification1 (float error)
{
      PB1 = 0.f; PM1 = 0.f; PS1 = 0.f; Z01 = 0.f; NS1 = 0.f; NM1 = 0.f; NB1
= 0.f;


      if (error >= EM)
            PB1 = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB1 = (3.f / EM * error) - 2;
            PM1 = (-3.f / EM * error) + 3;
      }
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM1 = (3.f / EM * error) - 1;
            PS1 = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS1 = 3.f / EM * error;
            Z01 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS1 = (-3.f / EM * error);
            Z01 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS1 = (3.f / EM * error) + 2;
            NM1 = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM1 = (3.f / EM * error) + 3;
            NB1 = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB1 = 1;
}

void fuzzification(float error)
{
      PB = 0.f; PM = 0.f; PS = 0.f; Z0 = 0.f; NS = 0.f; NM = 0.f; NB = 0.f;


      if (error >= EM)
            PB = 1.F;
      else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
            PB = (3.f / EM * error) - 2;
            PM = (-3.f / EM * error) + 3;
```

```
        }
        else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
                PM = (3.f / EM * error) - 1;
                PS = (-3.f / EM * error) + 2;
        }
        else if ((error >= 0) && (error < (EM / 3.F))) {
                PS = 3.f / EM * error;
                Z0 = (-3.f / EM * error) + 1;
        }
        else if ((error >= (-EM / 3.f)) && (error < 0)) {
                NS = (-3.f / EM * error);
                Z0 = (3. / EM * error) + 1;
        }
        else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
                NS = (3.f / EM * error) + 2;
                NM = (-3.f / EM * error) + 1;
        }
        else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
                NM = (3.f / EM * error) + 3;
                NB = (-3.f / EM * error) - 2;
        }
        else if (error < - EM)
                NB = 1.F;
}

void main (void)
{
        FILE *f1, *data;
        register float sinyal, itae = 0.F, kont = 0.F;
        register int max_hiz = 0, i, hiz, k = 0;
        long hata, hata1 = 0L, tophiz = 0L, ara_hiz = 0L, t;
        int read_hiz[30] = {0}, error[3000], speed[3000], ref[3000];
        float iae = 0.F, max_ovrsht, error_rate;

        init();
        init_alfa();
KONTROL:

        while (k < 6000) {
                        disable();
                        ara_hiz = 0L;
                        read_hiz[0] = adcoku();
                        for (i = 0; i < 30; ++i)
                                ara_hiz += read_hiz[i];
                        hiz = ara_hiz / 30;
                        if ((k < 3000) && (hiz > max_hiz))
                                max_hiz = hiz;
                        if ((k < 1000) || (k > 2000))
                                hata = REF - hiz;
                        else
                                hata = REF - hiz;
                        if (k < 3000) {
                                speed[k] = hiz;
                                ref[k] = hata + hiz;
                                error[k] = hata;
                        }
                        error_rate = (float)(hata - hata1) * Gce * 2000;
                        fuzzification(hata * Ge);
                        fuzzification1(hata * Ge);
                        fuzzification_errate(error_rate);
                        fuzzification_errate1(error_rate);
                        inference();
```

```
                inference_alfa();

                kont += (u_bulanık * Gu * alfa );
                //kont += (u_bulanık * Gu);
                for (i = 29; 0 < i ; --i)
                      read_hiz[i] = read_hiz[i-1];
                hata1 = hata;
                k++;
                for (i = 0; i < kont; i += 4) {
                      outp(0x378, 1);
                      if (!(inp(0x379) & 0x20))
                            goto KONTROL;
                }
                outp(0x378, 0);
                while ((inp(0x379) & 0x20))
                      ;
                goto KONTROL;
      }
      enable();
      max_ovrsht = (float) (max_hiz - REF) / REF;
      outp(0x378, 0);
      outp(0x378, 0);
      clrscr();
// real time control islemi bitince sonuclarin ilgili dizilerden
// asagidaki gibi tanimli .dat uzantili bir dosyaya kaydedilmesi

      if ((f1 = fopen(DOSYA, "w+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#if TRAIN
      if ((data = fopen("datastfz.dat", "a+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#endif
      for (k=0; k < 3000; ++k) {
            iae += (float) abs(error[k]);
            itae += (float) k * 0.0005 * abs(error[k]);
            fprintf(f1, "%d\n", speed[k]);
      }
      for (k=2950; k < 3000; ++k)
            printf("hiz: %d\n", speed[k]);
      printf("IAE=%f\n", iae );
      printf("max =%f\n", max_ovrsht);
      printf("ITAE =%f\n", itae);
#if TRAIN
      fprintf(data, "%f\n", Ge);
      fprintf(data, "%f\n", Gce);
      fprintf(data, "%f\n", (float) (iae - 80000) / (180000 - 80000));
      fprintf(data, "%f\n", max_ovrsht);
      fprintf(data, "%f\n", (itae - 20000) / (60000 - 20000));
      fclose(data);
#endif
      fclose(f1);
      getch();
}
```

Ek 7 Nöral-Genetik Tabanlı Optimal Öz - Uyarlamalı Bulanık P-ID Kontrolörü

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <math.h>
#define     TRAIN 0
#define     DOSYA       "fp-id12.dat"
#define     REF         600    //2390 devir/dk
#define     BASE        0x280
#define     Kp          .21
#define     Kd          0.0006
#define     Ki          5.7
#define     Ts          0.0005
#define     EM          20.F
#define     EMd         20.F
#define     OM          20.F
#define     OM_alfa     10.F
#define     Ge          .0433
#define     Gce         .000865

// global degiskenler, uyelik fonksiyonlari

float w[49]; // inference agirliklari
float w_a[49]; // inference alfa agirliklari
float y_a[49];
float y[49];
float PB, PM, PS, Z0, NS, NM, NB;
float PB1, PM1, PS1, Z01, NS1, NM1, NB1;
float PBd, PMd, PSd, Z0d, NSd, NMd, NBd;
float PBd1, PMd1, PSd1, Z0d1, NSd1, NMd1, NBd1;
float yPB, yPM, yPS, yZ0, yNS, yNM, yNB;
float yPBd, yPMd, yPSd, yZ0d, yNSd, yNMd, yNBd;
float u_bulanık, alfa;

void init(void);
void init_alfa(void);
void inference (void);
void inference_alfa (void);
void fuzzification (float error);
void fuzzification1 (float error);
void fuzzification_errate1 (float error_rate);
void fuzzification_errate (float error_rate);
int adcoku (void);

int adcoku (void)
{
      int tempLSB, tempMSB, x;

      outportb(BASE + 16, 15);
      while (!((inportb(BASE + 20) & 0x01)))
            ;
      tempMSB = inportb(BASE + 18);
      tempLSB = inportb(BASE + 19) & 0x0C;
      x = tempMSB << 4;
      x += tempLSB;
      return x - 1888;
}
```

```
void fuzzification_errate (float error_rate)
{
      PBd = 0.f; PMd = 0.f; PSd = 0.f; Z0d = 0.f; NSd = 0.f; NMd = 0.f;
      NBd = 0.f;


      if (error_rate >= EMd)
            PBd = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd = (3.f / EMd * error_rate) - 2;
            PMd = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd = (3.f / EMd * error_rate) - 1;
            PSd = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd = 3.f / EMd * error_rate;
            Z0d = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd = (-3.f / EMd * error_rate);
            Z0d = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd = (3.f / EMd * error_rate) + 2;
            NMd = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd = (3.f / EMd * error_rate) + 3;
            NBd = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd = 1;
}

void fuzzification_errate1 (float error_rate)
{
      PBd1 = 0.f; PMd1 = 0.f; PSd1 = 0.f; Z0d1 = 0.f; NSd1 = 0.f; NMd1 =
0.f;
      NBd1 = 0.f;


      if (error_rate >= EMd)
            PBd1 = 1.F;
      else if ((error_rate >= (2 * EMd / 3.f)) && (error_rate < EMd)) {
            PBd1 = (3.f / EMd * error_rate) - 2;
            PMd1 = (-3.f / EMd * error_rate) + 3;
      }
      else if ((error_rate >= (EMd / 3.F)) && (error_rate < (2 * EMd /
3.F))) {
            PMd1 = (3.f / EMd * error_rate) - 1;
            PSd1 = (-3.f / EMd * error_rate) + 2;
      }
      else if ((error_rate >= 0) && (error_rate < (EMd / 3.F))) {
            PSd1 = 3.f / EMd * error_rate;
            Z0d1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-EMd / 3.f)) && (error_rate < 0)) {
            NSd1 = (-3.f / EMd * error_rate);
```

```
            Z0d1 = (3. / EMd * error_rate) + 1;
      }
      else if ((error_rate >= (-2.F * EMd / 3.f)) && (error_rate < (-EMd /
3.F))) {
            NSd1 = (3.f / EMd * error_rate) + 2;
            NMd1 = (-3.f / EMd * error_rate) + 1;
      }
      else if ((error_rate < (-2 * EMd / 3.f)) && (error_rate >= - EMd)) {
            NMd1 = (3.f / EMd * error_rate) + 3;
            NBd1 = (-3.f / EMd * error_rate) - 2;
      }
      else if (error_rate < - EMd)
            NBd1 = 1;
}

void init(void)
//kural matrisi main kontroler icin
{

      yNB = - OM; yNM = -(2 * OM / 3.F); yNS = -( OM / 3.F); yZ0 = 0.F;
      yPB = OM; yPM = (2 * OM / 3.F); yPS = OM / 3.F;

      y[0]  = yNB;  y[1] = yNB; y[2]  = yNB; y[3]  = yNB; y[4]  = yNM; y[5]
= yNS; y[6]  = yZ0;
      y[7]  = yNB;  y[8] = yNB; y[9]  = yNB; y[10] = yNM; y[11] = yNS;
y[12] = yZ0; y[13] = yPS;
      y[14] = yNB; y[15] = yNB; y[16] = yNM; y[17] = yNS; y[18] = yZ0;
y[19] = yPS; y[20] = yPM;
      y[21] = yNB; y[22] = yNM; y[23] = yNS; y[24] = yZ0; y[25] = yPS;
y[26] = yPM; y[27] = yPB;
      y[28] = yNM; y[29] = yNS; y[30] = yZ0; y[31] = yPS; y[32] = yPM;
y[33] = yPB; y[34] = yPB;
      y[35] = yNS; y[36] = yZ0; y[37] = yPS; y[38] = yPM; y[39] = yPB;
y[40] = yPB; y[41] = yPB;
      y[42] = yZ0; y[43] = yPS; y[44] = yPM; y[45] = yPB; y[46] = yPB;
y[47] = yPB; y[48] = yPB;
}

void init_alfa(void)
//kural matrisi alfa kontroler icin
{

      yNBd = - OM_alfa; yNMd = -(2 * OM_alfa / 3.F); yNSd = -( OM_alfa /
3.F); yZ0d = 0.F;
      yPBd = OM_alfa; yPMd = (2 * OM_alfa / 3.F); yPSd = OM_alfa / 3.F;

      y_a[0]  = yNBd; y_a[1]  = yNBd; y_a[2]  = yNBd; y_a[3]  = yNSd;
y_a[4]  = yPSd; y_a[5]  = yPSd; y_a[6]  = yPSd;
      y_a[7]  = yNBd; y_a[8]  = yNBd; y_a[9]  = yNBd; y_a[10] = yNSd;
y_a[11] = yPSd; y_a[12] = yPSd; y_a[13] = yPSd;
      y_a[14] = yNBd; y_a[15] = yNBd; y_a[16] = yNMd; y_a[17] = yNSd;
y_a[18] = yPSd; y_a[19] = yPSd; y_a[20] = yPMd;
      y_a[21] = yNBd; y_a[22] = yNMd; y_a[23] = yNSd; y_a[24] = yZ0d;
y_a[25] = yPSd; y_a[26] = yPMd; y_a[27] = yPBd;
      y_a[28] = yNMd; y_a[29] = yNSd; y_a[30] = yNSd; y_a[31] = yPSd;
y_a[32] = yPMd; y_a[33] = yPBd; y_a[34] = yPBd;
      y_a[35] = yNSd; y_a[36] = yNSd; y_a[37] = yNSd; y_a[38] = yPSd;
y_a[39] = yPBd; y_a[40] = yPBd; y_a[41] = yPBd;
      y_a[42] = yNSd; y_a[43] = yNSd; y_a[44] = yNSd; y_a[45] = yPSd;
y_a[46] = yPBd; y_a[47] = yPBd; y_a[48] = yPBd;
}
```

```
void inference(void)
{
      int i;
      register float ara_toplam = 0, w_toplam = 0;


      w[0] = NB < NBd ? NB : NBd;
      w[1] = NM < NBd ? NM : NBd;
      w[2] = NS < NBd ? NS : NBd;
      w[3] = Z0 < NBd ? Z0 : NBd;
      w[4] = PS < NBd ? PS : NBd;
      w[5] = PM < NBd ? PM : NBd;
      w[6] = PB < NBd ? PB : NBd;

      w[7] = NB < NMd ? NB : NMd;
      w[8] = NM < NMd ? NM : NMd;
      w[9] = NS < NMd ? NS : NMd;
      w[10] = Z0 < NMd ? Z0 : NMd;
      w[11] = PS < NMd ? PS : NMd;
      w[12] = PM < NMd ? PM : NMd;
      w[13] = PB < NMd ? PB : NMd;

      w[14] = NB < NSd ? NB : NSd;
      w[15] = NM < NSd ? NM : NSd;
      w[16] = NS < NSd ? NS : NSd;
      w[17] = Z0 < NSd ? Z0 : NSd;
      w[18] = PS < NSd ? PS : NSd;
      w[19] = PM < NSd ? PM : NSd;
      w[20] = PB < NSd ? PB : NSd;

      w[21] = NB < Z0d ? NB : Z0d;
      w[22] = NM < Z0d ? NM : Z0d;
      w[23] = NS < Z0d ? NS : Z0d;
      w[24] = Z0 < Z0d ? Z0 : Z0d;
      w[25] = PS < Z0d ? PS : Z0d;
      w[26] = PM < Z0d ? PM : Z0d;
      w[27] = PB < Z0d ? PB : Z0d;

      w[28] = NB < PSd ? NB : PSd;
      w[29] = NM < PSd ? NM : PSd;
      w[30] = NS < PSd ? NS : PSd;
      w[31] = Z0 < PSd ? Z0 : PSd;
      w[32] = PS < PSd ? PS : PSd;
      w[33] = PM < PSd ? PM : PSd;
      w[34] = PB < PSd ? PB : PSd;

      w[35] = NB < PMd ? NB : PMd;
      w[36] = NM < PMd ? NM : PMd;
      w[37] = NS < PMd ? NS : PMd;
      w[38] = Z0 < PMd ? Z0 : PMd;
      w[39] = PS < PMd ? PS : PMd;
      w[40] = PM < PMd ? PM : PMd;
      w[41] = PB < PMd ? PB : PMd;

      w[42] = NB < PBd ? NB : PBd;
      w[43] = NM < PBd ? NM : PBd;
      w[44] = NS < PBd ? NS : PBd;
      w[45] = Z0 < PBd ? Z0 : PBd;
      w[46] = PS < PBd ? PS : PBd;
      w[47] = PM < PBd ? PM : PBd;
      w[48] = PB < PBd ? PB : PBd;
```

```
	for (i = 0; i < 49; ++i)
		ara_toplam += w[i] * y[i];
	for (i = 0; i < 49; ++i)
		w_toplam += w[i];
	//printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
	u_bulanık = ara_toplam / w_toplam;
}

void inference_alfa(void)
{
	int i;
	register float ara_toplam = 0.F, w_toplam = 0.F;


	w_a[0] = NB1 < NBd1 ? NB1 : NBd1;
	w_a[1] = NM1 < NBd1 ? NM1 : NBd1;
	w_a[2] = NS1 < NBd1 ? NS1 : NBd1;
	w_a[3] = Z01 < NBd1 ? Z01 : NBd1;
	w_a[4] = PS1 < NBd1 ? PS1 : NBd1;
	w_a[5] = PM1 < NBd1 ? PM1 : NBd1;
	w_a[6] = PB1 < NBd1 ? PB1 : NBd1;

	w_a[7] = NB1 < NMd1 ? NB1 : NMd1;
	w_a[8] = NM1 < NMd1 ? NM1 : NMd1;
	w_a[9] = NS1 < NMd1 ? NS1 : NMd1;
	w_a[10] = Z01 < NMd1 ? Z01 : NMd1;
	w_a[11] = PS1 < NMd1 ? PS1 : NMd1;
	w_a[12] = PM1 < NMd1 ? PM1 : NMd1;
	w_a[13] = PB1 < NMd1 ? PB1 : NMd1;

	w_a[14] = NB1 < NSd1 ? NB1 : NSd1;
	w_a[15] = NM1 < NSd1 ? NM1 : NSd1;
	w_a[16] = NS1 < NSd1 ? NS1 : NSd1;
	w_a[17] = Z01 < NSd1 ? Z01 : NSd1;
	w_a[18] = PS1 < NSd1 ? PS1 : NSd1;
	w_a[19] = PM1 < NSd1 ? PM1 : NSd1;
	w_a[20] = PB1 < NSd1 ? PB1 : NSd1;

	w_a[21] = NB1 < Z0d1 ? NB1 : Z0d1;
	w_a[22] = NM1 < Z0d1 ? NM1 : Z0d1;
	w_a[23] = NS1 < Z0d1 ? NS1 : Z0d1;
	w_a[24] = Z01 < Z0d1 ? Z01 : Z0d1;
	w_a[25] = PS1 < Z0d1 ? PS1 : Z0d1;
	w_a[26] = PM1 < Z0d1 ? PM1 : Z0d1;
	w_a[27] = PB1 < Z0d1 ? PB1 : Z0d1;

	w_a[28] = NB1 < PSd1 ? NB1 : PSd1;
	w_a[29] = NM1 < PSd1 ? NM1 : PSd1;
	w_a[30] = NS1 < PSd1 ? NS1 : PSd1;
	w_a[31] = Z01 < PSd1 ? Z01 : PSd1;
	w_a[32] = PS1 < PSd1 ? PS1 : PSd1;
	w_a[33] = PM1 < PSd1 ? PM1 : PSd1;
	w_a[34] = PB1 < PSd1 ? PB1 : PSd1;

	w_a[35] = NB1 < PMd1 ? NB1 : PMd1;
	w_a[36] = NM1 < PMd1 ? NM1 : PMd1;
	w_a[37] = NS1 < PMd1 ? NS1 : PMd1;
	w_a[38] = Z01 < PMd1 ? Z01 : PMd1;
	w_a[39] = PS1 < PMd1 ? PS1 : PMd1;
	w_a[40] = PM1 < PMd1 ? PM1 : PMd1;
	w_a[41] = PB1 < PMd1 ? PB1 : PMd1;
```

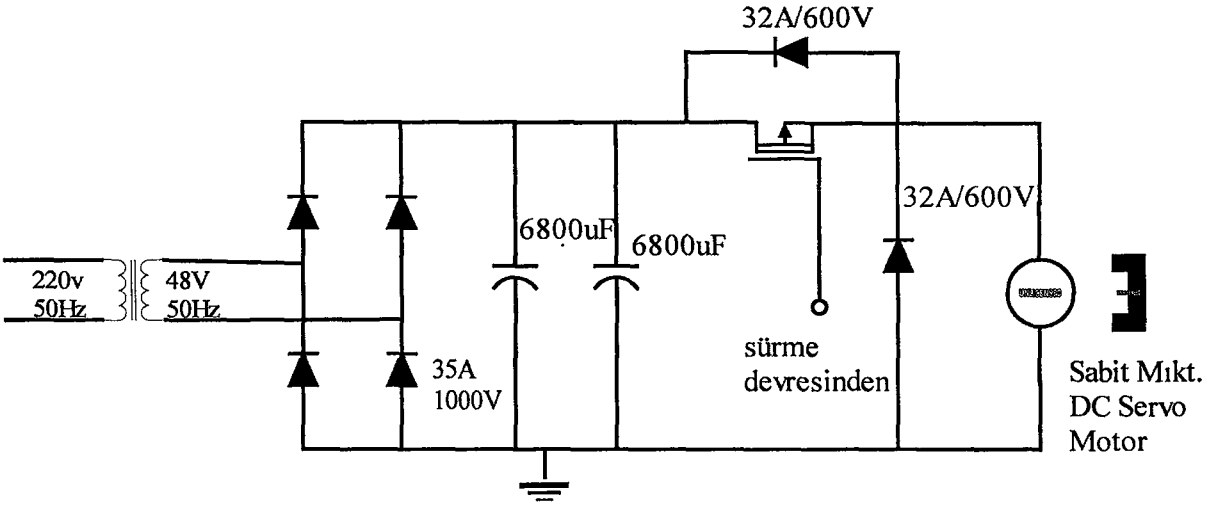
```
	w_a[42] = NB1 < PBd1 ? NB1 : PBd1;
	w_a[43] = NM1 < PBd1 ? NM1 : PBd1;
	w_a[44] = NS1 < PBd1 ? NS1 : PBd1;
	w_a[45] = Z01 < PBd1 ? Z01 : PBd1;
	w_a[46] = PS1 < PBd1 ? PS1 : PBd1;
	w_a[47] = PM1 < PBd1 ? PM1 : PBd1;
	w_a[48] = PB1 < PBd1 ? PB1 : PBd1;

	for (i = 0; i < 49; ++i)
		ara_toplam += w_a[i] * y_a[i];
	for (i = 0; i < 49; ++i)
		w_toplam += w_a[i];
	//printf("aratop: %f\tw_toplam: %f\n", ara_toplam, w_toplam);
	alfa = ara_toplam / w_toplam;
}


void fuzzification1 (float error)
{
	PB1 = 0.f; PM1 = 0.f; PS1 = 0.f; Z01 = 0.f; NS1 = 0.f; NM1 = 0.f; NB1
= 0.f;


	if (error >= EM)
		PB1 = 1.F;
	else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
		PB1 = (3.f / EM * error) - 2;
		PM1 = (-3.f / EM * error) + 3;
	}
	else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
		PM1 = (3.f / EM * error) - 1;
		PS1 = (-3.f / EM * error) + 2;
	}
	else if ((error >= 0) && (error < (EM / 3.F))) {
		PS1 = 3.f / EM * error;
		Z01 = (-3.f / EM * error) + 1;
	}
	else if ((error >= (-EM / 3.f)) && (error < 0)) {
		NS1 = (-3.f / EM * error);
		Z01 = (3. / EM * error) + 1;
	}
	else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
		NS1 = (3.f / EM * error) + 2;
		NM1 = (-3.f / EM * error) + 1;
	}
	else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
		NM1 = (3.f / EM * error) + 3;
		NB1 = (-3.f / EM * error) - 2;
	}
	else if (error < - EM)
		NB1 = 1;
}

void fuzzification(float error)
{
	PB = 0.f; PM = 0.f; PS = 0.f; Z0 = 0.f; NS = 0.f; NM = 0.f; NB = 0.f;


	if (error >= EM)
		PB = 1.F;
	else if ((error >= (2 * EM / 3.f)) && (error < EM)) {
```

```
            PB = (3.f / EM * error) - 2;
            PM = (-3.f / EM * error) + 3;
      }
      else if ((error >= (EM / 3.F)) && (error < (2 * EM / 3.F))) {
            PM = (3.f / EM * error) - 1;
            PS = (-3.f / EM * error) + 2;
      }
      else if ((error >= 0) && (error < (EM / 3.F))) {
            PS = 3.f / EM * error;
            Z0 = (-3.f / EM * error) + 1;
      }
      else if ((error >= (-EM / 3.f)) && (error < 0)) {
            NS = (-3.f / EM * error);
            Z0 = (3. / EM * error) + 1;
      }
      else if ((error >= (-2.F * EM / 3.f)) && (error < (-EM / 3.F))) {
            NS = (3.f / EM * error) + 2;
            NM = (-3.f / EM * error) + 1;
      }
      else if ((error < (-2 * EM / 3.f)) && (error >= - EM)) {
            NM = (3.f / EM * error) + 3;
            NB = (-3.f / EM * error) - 2;
      }
      else if (error < - EM)
            NB = 1;
}

void main (void)
{
      FILE *f1, *data;

      register float itae = 0.F, max_ovrsht, error_rate, sinyal, kont,
kont_pi = 0.F;
      register int      m, hiz, hiz1 = 0, hiz2 = 0, k = 0, i;
      register long top_error = 0L, top_hata = 0L, t, hata, hata1= 0L,
ara_hiz = 0L;
      int max_hiz = 0, read_hiz[30] = {0}, error[3000], ref[3000],
speed[3000];

      init();
      init_alfa();

KONTROL:

      while (k < 6000) {
                  disable();
                  ara_hiz = 0L;
                  read_hiz[0] = adcoku();
                  for (i = 0; i < 30; ++i)
                        ara_hiz += read_hiz[i];
                  hiz = ara_hiz / 30;
                  if ((k < 3000) && (hiz > max_hiz))
                        max_hiz = hiz;
                  if ((k < 1000) || (k > 2000))
                        hata = REF - hiz;
                  else
                        hata = REF - hiz;
                  if (k < 3000) {
                        speed[k] = hiz;
                        error[k] = hata;
                        ref[k] = hata + hiz;
                        top_hata += k * abs(hata);
```

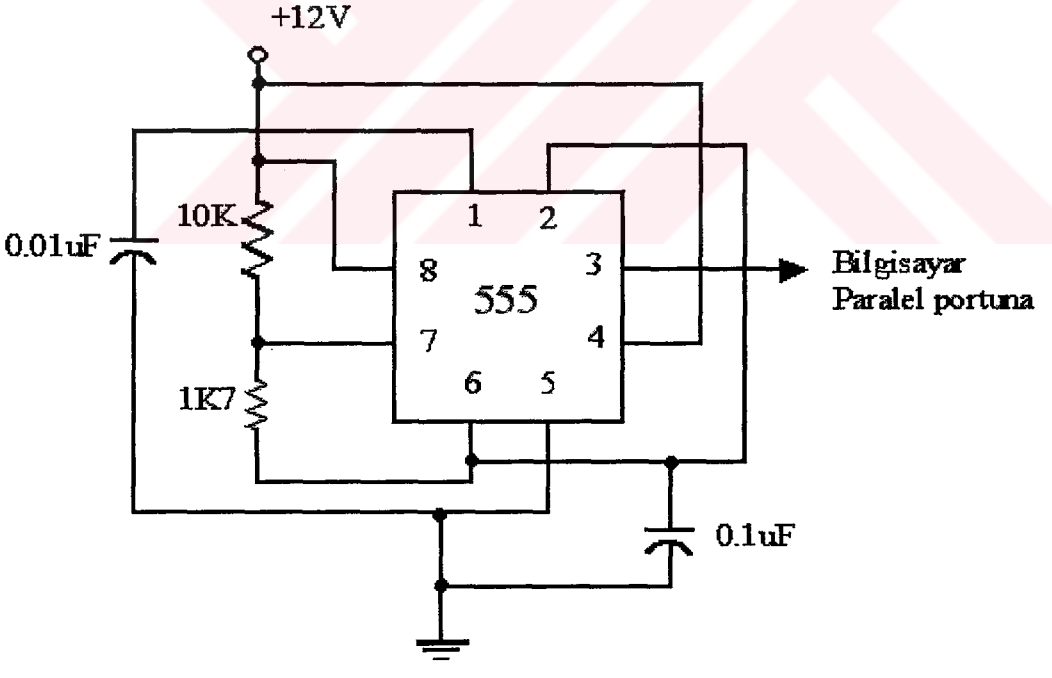
```
                }

                error_rate = (float)(hata - hata1) * Gce * (1.F / Ts);
                fuzzification(hata * Ge);
                //fuzzification1(hata * Ge);
                fuzzification_errate(error_rate);
                //fuzzification_errate1(error_rate);
                inference();
                //inference_alfa();

                kont += (Kp * u_bulanık) + (Ki * Ts * hata) -
                        (Kd * (hiz - (2 * hiz1) + hiz2) / Ts);

                for (i = 29; 0 < i ; --i)
                      read_hiz[i] = read_hiz[i-1];
                hata1 = hata;
                hiz2 = hiz1;
                hiz1 = hiz;
                k++;
                for (i = 0; i < kont; i += 4) {
                      outp(0x378, 1);
                      if (!(inp(0x379) & 0x20))
                            goto KONTROL;
                }
                outp(0x378, 0);
                while ((inp(0x379) & 0x20))
                      ;
                goto KONTROL;
      }
      enable();
      max_ovrsht = ((float) max_hiz - REF) / REF;
      outp(0x378,0);

      clrscr();
// real time control islemi bitince sonuclarin ilgili dizilerden
// asagidaki gibi tanimli .dat uzantili bir dosyaya kaydedilmesi

      if ((f1 = fopen(DOSYA, "w+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#if TRAIN
      if ((data = fopen("datapi.dat", "a+")) == NULL) {
            printf("Can not open file!..\n");
            exit(1);
      }
#endif
      for (k=0; k < 3000; ++k) {
            top_error += abs(error[k]);
            itae += (float) k * 0.0005 * abs(error[k]);
            fprintf(f1, "%d\t%d\t%d\n", speed[k], error[k], ref[k]);
      }
      for (k=2950; k < 3000; ++k)
            printf("hiz: %d\n", speed[k]);
      printf("IAE =%ld\n", top_error);
   // printf("yukselme zamani = %f\n", (t_rise / 3000.f));
      printf("max =%f\n", max_ovrsht);
      printf("ITAE =%f\n", itae);
#if TRAIN
      fprintf(data, "%f\n", Ge);
      fprintf(data, "%f\n", Gce);
```

```
      fprintf(data, "%f\n", (float) (top_error - 70000) / (400000 -
70000));
      fprintf(data, "%f\n", max_ovrsht);
      fprintf(data, "%f\n", (itae - 5000) / (70000 - 5000));
      fclose(data);
#endif
      fclose(f1);
      getch();
}
```

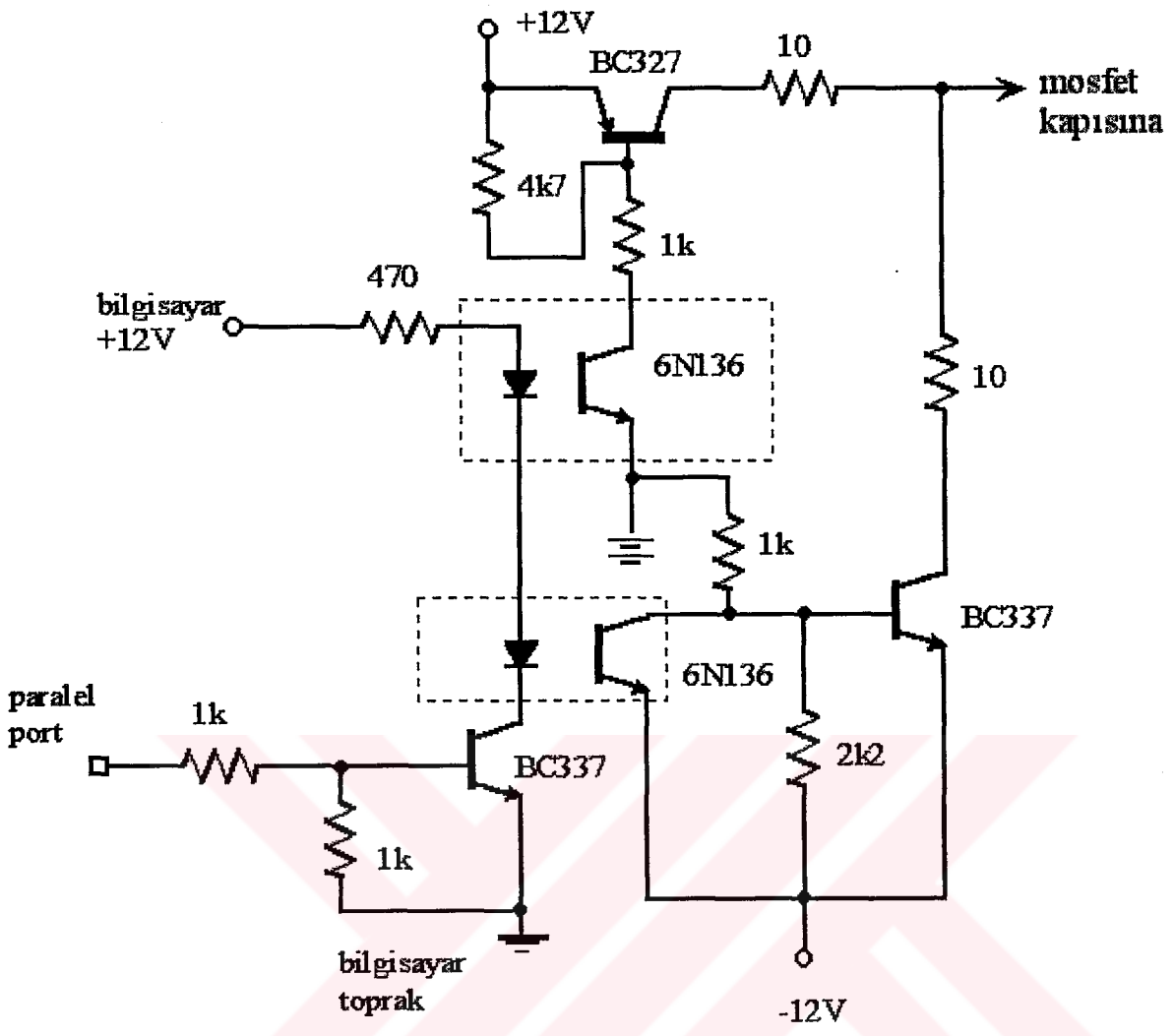
Ek 8 Motor Sürme ve Arayüz Devreleri



Şekil Ek 8.1 DC motor sürme devresi (güç devresi)



Şekil Ek 8.2 Örnekleme zamanını ayarlayan timer devresi



Şekil Ek 8.3 DC motor sürme devresi (zayıf akım devresi)

ÖZGEÇMİŞ

Doğum tarihi	18.08.1975	
Doğum yeri	İstanbul	
Doktora	1999-2002	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektrik Müh. Anabilim Dalı.
Yüksek Lisans	1997-1999	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektrik Müh. Anabilim Dalı.
Lisans	1993-1997	Yıldız Teknik Üniversitesi, Elektrik - Elektronik Fakültesi, Elektrik Müh. Bölümü.
Lise	1986-1993	Özel Eyüboğlu Lisesi.

Çalıştığı kurum

1998-Devam ediyor	Yıldız Teknik Üniversitesi, Elektrik – Elektronik Fakültesi, Araştırma Görevlisi .