T.C.
SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ROBOTLARIN TERS KİNEMATİK ÇÖZÜMÜNDE SEZGİSEL OPTİMİZASYON ALGORİTMALARININ KULLANILMASI VE FPGA İLE GERÇEKLEŞTİRİLMESİ

DOKTORA TEZİ

Serkan DERELİ

Enstitü Anabilim Dalı : MEKATRONİK MÜHENDİSLİĞİ

Tez Danışmanı : Prof. Dr. Raşit KÖKER

Haziran 2019

T.C.
SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ROBOTLARIN TERS KİNEMATİK ÇÖZÜMÜNDE SEZGİSEL OPTİMİZASYON ALGORİTMALARININ KULLANILMASI VE FPGA İLE GERÇEKLEŞTİRİLMESİ

DOKTORA TEZİ

Serkan DERELİ

Enstitü Anabilim Dalı : MEKATRONİK MÜHENDİSLİĞİ

Bu tez 13/06/2019 tarihinde aşağıdaki jüri tarafından oybirliği/oyçokluğu ile kabul edilmiştir.

Prof. Dr. **Recep KOZAN** **Jüri Başkanı**	**Prof. Dr.** **Ali Fuat BOZ** **Üye**	**Prof. Dr.** **Raşit KÖKER** **Üye**
Prof. Dr. **Serdar KÜÇÜK** **Üye**	**Doç. Dr.** **Faruk YALÇIN** **Üye**	

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Serkan DERELİ
13/06/2019

TEŞEKKÜR

Doktora eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Prof. Dr. Raşit KÖKER'e; pek çok kişinin tecrübe ve bilgi eksikliği nedeniyle cevaplayamadığı FPGA ile ilgili sorunlarımda kendisini her daim yanımda hissettiğim sevgili Bertan Taşkın'a teşekkürlerimi sunarım.

Evde oturma düzenini benim çalışabileceğim şekilde yeniden ayarlayarak bana ve robotuma rahat bir çalışma ortamı sunan eşime, motivasyonum düştüğünde yanıma gelerek beni ateşleyen biricik kızıma ve evdeki çalışma ortamında yaptığı şirinliklerle enerjimi hep yüksek tutan aslan parçası sevgili oğluma gönülden şükranlarımı sunmak isterim.

Ayrıca bu çalışmanın maddi açıdan desteklenmesine olanak sağlayan Sakarya Uygulamalı Bilimler Üniversitesi Bilimsel Araştırma Projeleri (BAP) Komisyon Başkanlığına (Proje No: 2016-50-02-015) teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

ABA	: Ateş böceği algoritması
DOF	: Serbestlik derecesi (Degree of freedom)
FPGA	: Alanda programlanabilir kapı dizisi
GA	: Genetik algoritma
GSA	: Yerçekimi arama algoritması
HSA	: Uyum arama algoritması
KPSO	: Kuantum parçacık sürü optimizasyonu
PAL	: Programlanabilir dizi mantığı
PLA	: Programlanabilir mantık dizisi
PLD	: Programlanabilir mantık aygıtı
PSO	: Parçacık sürü optimizasyonu
RDV	: Rassal azalan hızlı
SD	: Saat darbesi
VHDL	: Yüksek hızlı donanım tanımlama dili
YAK	: Yapay arı kolonisi
YSA	: Yapay sinir ağları

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ROBOTLARI TERS KİNEMATİK ÇÖZÜMÜNDE SEZGİSEL OPTİMİZASYON ALGORİTMALARININ KULLANILMASI VE FPGA İLE GERÇEKLENMESİ

ÖZET

Bu tez çalışmasında, öncelikle dokuz adet Dynamixel servo motor kullanılarak 7-eklemli gereğinden fazla serbestlik derecesine (redundant) sahip seri bir robot manipülatörü tasarlanmış, sonrasında ise bu manipülatörün kinematik analizleri yapılmıştır. Bu kinematik analizlerden yola çıkarak robot manipülatörün ters kinematik hesabı doğadan ilham alınarak geliştirilen ve sürü zekâsına sahip sezgisel optimizasyon algoritmalarıyla gerçekleştirilmiş ve aynı zamanda tasarlanan bu robot manipülatöründe MATLAB arabirimi ile oluşturulan bir GUI vasıtasıyla uygulanmıştır. Bu işlem için daha önce literatürde kullanılan dört adet sürü zekâsı esasına göre çalışan algoritmayla birlikte ilk defa bu çalışmada kullanılan ve golf topunun hareketlerinden esinlenilerek oluşturulan Rassal Azalan Hızlı (RDV) isimli yeni bir IW tekniği ile güçlendirilen PSO tabanlı sezgisel optimizasyon algoritması kullanılarak ters kinematik problem çözülmüştür. Bu manada öncelikle literatüre yeni kzandırılan RDV IW tekniği ile güçlendirilen bu algoritma ile diğer dört adet sezgisel optimizasyon algoritmaları pozisyon hatası ve çalışma süresi bakımından karşılaştırılmıştır.

Sezgisel optimizasyon algoritmalardan elde edilen işlem süresinin gerçek bir robotta uygulanabilirliğinin düşük olması nedeniyle işlem süresini çok daha minimum bir değere indirmek maksadıyla ters kinematik çözüm daha önce literatürde kullanılmayan ve FPGA tabanlı oluşturulan sentezlenebilir PSO ile donanımsal olarak gerçekleştirilmiştir ve sonuçlar yazılımsal değerlerle karşılaştırılmıştır.

Sonuçta robotik temelindeki karmaşık bir problemin çözümünde sürü zekâsına sahip algoritmaların yazılım ve donanım temelli başarımları karşılaştırılmıştır. Sentezlenebilir özelliğe sahip FPGA tasarım VHDL olarak bilinen donanım tanımlama dili ile gerçekleştirilmiş, sonuçlar simülasyon olarak ortaya konmakla birlikte Nexys 4 DDR kartı ile de gerçeklenmiştir. Bu çalışmada kullanılan bütün yazılımsal ve donanımsal sezgisel optimizasyon algoritmaları MATLAB arabirimi vasıtasıyla tasarlanan redundant robot manipülatöründe uygulanmış ve servo motorların çözünürlük kısıtı nedeniyle bu aşamada ortaya çıkan gerçek hata değerleride özellikle gözlemlenerek analiz edilmiştir.

Anahtar kelimeler: Redundant robot manipülatörü, ters kinematik çözüm, parçacık sürü optimizasyonu, yapay arı kolonisi, ateş böceği algoritması, FPGA, VHDL

THE USE OF METAHEURISTIC OPTIMIZATION ALGORITHMS FOR INVERSE KINEMATICS SOLUTION OF ROBOTS AND IMPLEMENTATION WITH FPGA

SUMMARY

In this study, firstly, serial robot manipulator which has 7-degree degree of freedom has designed by using Dynamixel servos and then kinematic analysis of this manipulator has performed. The inverse kinematics problem of the designed robot manipulator has been solved by heuristic algorithms developed with inspiration from nature and with a swarm intelligence and also these algoithms have been implemented in this designed robot manipulator by means of a GUI created with the MATLAB interface. In order to realize this process, the inverse kinematic problem was solved with a new IW technique called Random Decreasing Speed (RDV) which was used for the first time in this study together with the algorithm used on the basis of four swarm intelligence used in the literature. The RDW IW technique is a technique developed with inspiration from the movements of the golf ball and is used in conjunction with the PSO algorithm. In this context, this algorithm, firstly reinforced by the RDV IW technique, has been compared with the other four heuristic optimization algorithms in terms of position error and working time.

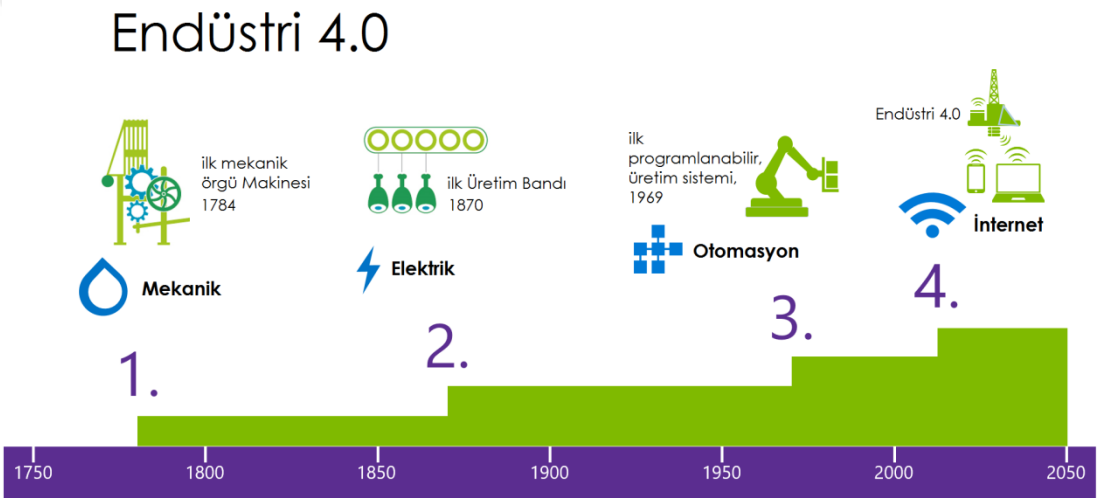
Since the calculation time obtained from heuristic optimization algorithms is far from feasible in a real robot, researchers have turned to FPGA technologies. Therefore, in this study, the inverse kinematics solution is realized with hardware based synthesizable PSO digital circuit design which has not been used in literature before.

As a result, software and hardware based performances of herd intelligence algorithms are compared in solving a complex problem based on robotics. The FPGA design, which can be synthesized, has been realized with the hardware description language known as VHDL, the results have been presented as simulation but also realized with Nexys 4 DDR card. All the software and hardware heuristic optimization algorithms used in this study were applied in redundant robot manipulator designed by MATLAB interface and the actual error values that occur at this stage due to the limitation of servo motors were analyzed and analyzed.

Keywords: Redundant robot manipulator, inverse kinematic solution, particle swarm optimization, artificial bee colony, firefly algorithm, FPGA, VHDL

BÖLÜM 1. GİRİŞ

Bugün en temel ve basit işlerden tutun, en zor ve karmaşık işlere kadar hemen hemen pek çok görevde rahatlıkla kullanılabilen robotlar üçüncü sanayi devriminin merkezinde bulundukları gibi yine son dönemde adından sıkça bahsedilen Endüstri 4.0 adı ile anılan yeni bir devriminde önemli bir parçası olmaya devam edecektir (Bulut ve Akçacı, 2017, pp. 55-57).



Şekil 1.1. Sanayi devrimleri (Bayrak, 2018)

Tüm sistemlerle uyumlu çalışması, büyük iş parçalarından küçük iş parçalarına kadar pek çok sistemde sınırsız döngüde çalışabilmesi, işlemlerdeki doğruluk ve tekrarlanabilirlik gibi önemli parametreleri yüksek oranda yakalamaları robotları cazip kılan özellikler denebilir (Staicu, 2019, pp. 121-146). Ancak son zamanlarda iş süreçlerindeki yoğunluğun artması robotik sistemlerin doğruluk oranlarını koruyarak hızlanma gereksinimlerini ortaya çıkarmıştır. Araştırmacılar ilk başlarda bu problemin çözümü için yapay zeka tekniklerini önermişler ve tüm dünyada hem akıllı hemde öğrenen robotlar piyasadeki yerini almıştır (Hyder, Siau ve Nah, 2018; Siau ve Wang, 2018, pp. 1-11; Deepak ve Parhi, 2016, pp. 417-439). İnsan beyninin çalışmasından esinlenilerek yapay sinir ağları (Song et al, 2015, pp. 124-131), kuş ve balık

sürülerinden esinlenilerek parçacık sürü optimizasyonu algrotiması (Wang et al, 2016,pp. 299-316; Dereli ve Köker, 2018, pp. 77-85; Küçük, 2013, pp. 129-149), arıların en kaliteli balı yapmak amacıyla izledikleri yoldan esinlenilerek yapay arı kolonisi algoritması (Xi et al, 2019), tropikal iklim bölgelerinde yaşayan ateş böceklerinin davranışlarından esinlenilerek ateş böceği algoritması geliştirilen bu tekniklerden bazılarıdır (Dereli ve Köker, 2019b, pp. 1-14; Sadhu et al, 2018, pp. 50-68). Tüm bu teknikler ile sistemler eğitilerek arzu edilen işlerin yüksek doğruluk oranlarında yapılması sağlanırken aynı zamanda sistemler içerinde gerçekleştirilen zorluk derecesi yüksek hesaplamalar daha kısa sürelerde sonuca ulaştırılmıştır (Caceres, Rosario ve Amaya, 2017). Ancak son zamanlarda teknolojik gelişmelere paralel olarak tasarımı yeniden değiştirilebilir entegre yapılar gömülü sistem mimarilerinin yerini almaya başlamıştır (Bao et al, 2017, pp. 1503-1512). Bu mimariler donanımsal olarak tasarlandığından hızlı işlem yapabilme kabiliyetinin yanı sıra, tasarım zamanlarının kısalması ve donanım yapılarının son kullanıcı tarafından yeniden oluşturulmasına imkan tanımasıyla göze çarpmaktadır (Rodríguez et al, 2018). FPGA yani "Alanda Programlanabilir Kapı Dizileri" olarak adlandırılan mimariler aynı zamanda paralel işlem yaparak sistemlerin hızlarını kat be kat artırmaktadır (Makryniotis ve Dasygenis, 2016; Li et al, 2017). Gömülü sistem teknolojisi olarak kullanımı giderek yaygınlaşan FPGA, robotik araştırmalarda da günümüzde özellikle robot kontrolü (Alkhafaji et al, 2018, pp. 1-25), hareket planlaması, ters kinematik hesaplaması ve eğitimi konularında yaygınca kullanılmaktadır.

Robotlarda ters kinematik konusu ziyadesiyle araştırmacılar tarafından ilgi çekici olmuş ve fazlasıyla çalışılan bir konu olarak gündemde yerini almıştır (Küçük ve Bingül, 2014, p. 1983). Bu konu ile ilgili 2000'li yıllarda yapay sinir ağı temelli sistemler çok daha fazla tercih edilmekteydi. Çünkü bu yıllarda endüstride 6-eklemli robot manipülatörleri yaygınca kullanılmaya başlanmış ve geleneksel yöntemlerin bu tarz robotlarda ki ters kinematik çözümde yetersiz kalması, tam aksine yapay sinir ağlarının özellikle hızlı çalışabilme ve öğrenme yeteneklerinin olması da ayrıca bunda büyük pay sahibi olmuştur (Bingül, Ertunç ve Oysu, 2005, p. 1). Ancak çok hassas işlemlerde yani hatanın son derece düşük olması gereken sistemlerde elde edilen hata değerleri kabul edilebilir sınırların dışına çıktığından dolayı zamanla yerini yapay sinir ağları ile birlikte tavlama benzetimi ve genetik algoritma gibi başka optimizasyon tekniklerinin birlikte kullanıldığı hibrit

sistemler almaya başlamıştır. Bu durumu aşmak adına (Köker, 2013, pp. 528-543) yapay sinir ağları ile genetik algoritmayı birleştirerek hibrit bir zeki teknik geliştirmiş ve pozisyon hatası bakımından yapay sinir ağlarına göre tatmin edici sonuçlar elde etmiştir. Benzer şekilde (Son, Anh ve Chau, 2014), evrimsel algoritmayı yapay sinir ağları ile birleştirerek hibrit bir sistem geliştirmiş ve pozisyon hatası bakımından robot manipülatörünün kullanabileceği değerler elde etmişlerdir. Son aşamada ise hassas işlemler söz konusu olduğunda dahi etkili çözümler ortaya koyabilen sürü tabanlı zeki optimizasyon teknikleri (parçacık sürü optimizasyonu, yapay arı kolonisi, ateş böceği, karınca kolonisi, vb…), ters kinematik problemlerin çözümünde odak noktasına oturmuştur. Özellikle uygunluk fonksiyonu şeklinde ifade edilebilen bütün problemlere uyarlanabilme kabiliyetine sahip olan bu teknikler gerçekten de pozisyon hatası bakımından etkili çözümler ortaya koymasına rağmen çözüme ulaşma süresi açısından tatminden uzak kalmışlardır. Araştırmacılar bu sorunun üstesinden gelebilmek adına bir algoritmanın zayıf yönlerini güçlendirerek sonuca pozitif katkı yapmaya çalıştıysalarda bu durum arzu edilen noktaya ulaşamamıştır. Örneğin parçacık sürü optimizasyonunun zayıf özellikli parçacıklarının güçlendirilmesiyle elde edilen kuantum parçacık sürüsü optimizasyonu bu tekniklerden sadece biridir ve pozisyon hatası bakımından yeterince tatmin edici olsada süre açısından her ne kadar üç kat iyileşme yaptıysada aslından bu durum istenilen düzeyde olamamıştır (Dereli ve Köker, 2019a, pp. 1-19). Benzer şekilde (Çavdar, Mohammed ve Milani, 2012, pp. 329-333)'de yapay arı kolonisini güçlendirerek pozisyon hatası bakımından istenilen seviyede iyileştirme elde etmelerine rağmen hesaplama süresi açısında tatmin edici sonuçlara ulaşamamışlardır.

Günümüzde araştırma dünyası, sistemleri çok daha hızlı çalışabilir hale getirmek, işlemleri çok daha kısa sürede tamamlayabilmek için donanım temelli olması, tasarımın sonradan değiştirilebilir olması, paralel çalışabilme yeteneği ve çok hızlı çalışması gibi nedenlerle FPGA'lı gömülü sistem mimarilerine odaklanmış bulunmaktadır (Sulaiman et al, 2009, pp. 3575-3576; Çavuşlu, Karakuzu ve Şahin, 2010, pp. 83-84). Özellikle (Palangpour, Venayagamoorthy ve Smith, 2009, p. 138), parçacık sürü optimizasyonu algoritmasını donanımsal olarak tasarlayarak Sphere ve Rosenbrock fonksiyonlarında test etmişler, 6000 ile 28000 kat aralığında çalışma süresini azaltabilmişlerdir. Bu motivasyondan yola çıkarak bu tez çalışmasında sürü tabanlı zeki optimizasyon

teknikleriyle yazılımsal olarak saniyeler düzeyinde elde edilen karmaşık bir problemin çözüm süresini FPGA ile zeki algoritmalardan birinin donanımsal tasarımının yapılarak çok daha küçük değerlere ulaştırmaktır.

Bu tez çalışmasında dikkat çekici iki durum söz konusudur: Bunlardan birincisi daha önce herhangi bir bilimsel araştırmada kullanılmayan RDV (Rassal Azalan Hızlı) olarak isimlendirilen IW tekniği ile klasik PSO algoritması 1000 kat ile 1000000 kat arasında hızlandırılmıştır. Böylece literatüre bundan sonraki çalışmalarda kullanılabilecek yeni bir teknik kazandırılmıştır. Diğer önemli husus ise FPGA entegrelerinde çalışabilen sentezlenebilir PSO sayısal devresi ile ters kinematik çözümün hesaplanma süresi saniyeler seviyesinden milisaniyeler seviyelerine çekilebilmiştir.

Tez çalışmasının bundan sonraki kısmı şu şekilde organize edilmiştir: İkinci bölümde kapsamlı bir literatür çalışması yapılmıştır. Literatürde hem zeki optimizasyon teknikleri hem bu tekniklerle ters kinematik çözümün gerçekleştirilme şekli ve hemde FPGA'ya dair araştırmalar irdelenmiştir. Üçüncü bölümde genel olarak robotlar üzerine temel bilgiler verildikten sonra çalışmada kullanılan 7-eklemli seri robot manipülatörü tanıtılarak kinematik analizleri yapılmış, yine çalışmada kullanılan problem ve bu problemi çözmek için kullanılan zeki tekniklerin tanıtımı yapılmıştır. Zeki optimizasyon tekniklerinden bu çalışmada kullanılanlar kapsamlı şekilde algoritma ve akış şeması üzerinden anlatılmıştır. Ayrıca bu bölümde çalışmanın şekillenmesinde ana yapı olarak kabul edilebilecek FPGA konusu hem teorik açıdan hemde bu çalışmada kullanılan uygulama kısmı detaylıca sunulmuştur. Dördüncü bölüm ise çalışmada kullanılan her bir teknik ile elde edilen araştırma bulgularının sunulup analizlerinin yapılarak detaylıca irdelendiği bölümdür. Karşılaştırmalar özellikle algoritmaların pozisyon hatası ve çözüme ulaşma süreleri açısından üstünlükleri, avantajları ve dezavantajları şeklinde sunulmuştur. Tez çalışması, bu çıkarılan kapsamlı sonuçların ortaya konarak sonlandırıldığı beşinci bölüm ile noktalanmıştır.

BÖLÜM 2. KAYNAK ARAŞTIRMASI

Seri robot manipülatörlerine ait ters kinematik denklemleri lineer olmayan denklemlerdir ve robot manipülatörünün eklem sayısına göre de karmaşıklığı üssel olarak artmaktadır. Karmaşıklığı artan bu robot manipülatörlerinin ters kinematik çözümünde geleneksel teknikler yetersiz kaldığından dolayı zeki optimizasyon teknikleri bu problem için etkili çözümler ortaya koyabilmektedir.

Bingül ve arkadaşları çalışmalarında geri yayılım algoritması temelli tasarladıkları üç farkı yapay sinir ağı modeliyle 6-eklemli robot manipülatörün ters kinematik çözümünü gerçekleştirmişlerdir. Rasgele seçtikleri 8000 verinin %70'ini ağı eğitmek için geri kalan %30'unu ise test işlemi için kullanmışlardır. Sonuçta tüm ağ tasarımlarında her bir eklem açısının çıkışta sahip olduğu hata ile her bir sistemin hata değeri RMS (root mean square) cinsinden verilmiştir. Sistemlerin hata değerleri 8.78, 10.16 ve 10.64 olarak verilmiştir (Bingül, Ertunç ve Oysu, 2005, pp. 1-5).

Zhang ve arkadaşları yaptıkları çalışmalarında altı giriş ve bir çıkışlı RBF tabanlı yapay sinir ağı modeliyle 6-eklemli MOTOMAN manipülatörün ters kinematik çözümünü gerçekleştirmişlerdir. Sonuçta elde ettikleri değerle geri yayılım (BP) algoritmasını karşılaştırmışlar ve bu konuda RBF algoritmasının yapay sinir ağlarında hem hesaplama yeteneği hemde çözüme yakınsama özelliği bakımından daha başarılı olduğunu ortaya koymuşlardır (Zhang, Lü ve Song, 2005, pp. 144-147).

Momani ve arkadaşları üç eklemli bir robot manipülatörünün ters kinematik hesabını hem geleneksel genetik algoritma ile hemde sürekli genetik algoritma ile gerçekleştirmişlerdir. Ters kinematik problem eklemlerin başlangıç açılarından final açısına ulaşmaları şeklinde formülleştirilmiş ve yol sapmasının minimum

olması hedef olarak belirlenmiştir. Sonuçlar ele alındığında sürekli GA her yönüyle geleneksel GA'ya göre çok daha iyi sonuçlar ortaya koymuştur (Momani, Abo-Hammour ve Alsmadi, 2016, pp. 1-9).

Ayyıldız ve Çetinkaya yaptıkları çalışmada tasarladıkları 4-dof tut ve yerleştir türünden seri robot manipülatörüne (pick-and-place) ait ters kinematik çözümü dört farklı (GA, PSO, KPSO, GSA – Yerçekimi Arama Algoritması) heuristik optimizasyon algoritması ile hesaplayarak sonuçları karşılaştırmışlardır. Çalışmalarında robot manipülatörünün çalışma uzayındaki yüz farklı noktaya en küçük hata ile ulaşması ve yine en küçük hata ile spiral bir yörünge izlemesini içerek iki farklı senaryo gerçekleştirmişlerdir. Sonuçlar hem hesaplama zamanı hemde minimum pozisyon hatası bakımından karşılaştırılmıştır. Her iki senaryoda da çalışma süresi açısından bakıldığında en kötü sonuçların GA ve GSA ile alındığı en iyi değerin ise KPSO ile alındığı görünmektedir. Ortalama pozisyon hatası karşılaştırmasında ise yine GA en kötü değeri döndürürken diğer üç algoritma birbirlerine yakın değerler ortaya koymuşlardır (Ayyıldız ve Çetinkaya, 2016, pp. 825-836).

Almusawi ve arkadaşları, yapmış oldukları çalışmalarında 6-DOF Denso robot manipülatörü kullanmışlar ve bu manipülatörün ters kinematik çözümünü YSA temelli yeni bir yaklaşımla gerçekleştirmişlerdir. Geleneksel YSA ile oluşturulan sinir ağı yapısında giriş katmanına robot manipülatörünün arzu edilen pozisyon ve oryantasyon bilgileri verilerek çıkışta eklem açıları elde edilmekteydi. Bu şekilde yapmış oldukları testler neticesinde 121 iterasyonda MSE değerini 1.1892e-05 elde etmişlerdir. Hâlbuki kendi önermiş oldukları yeni yaklaşıma göre ise sinir ağı giriş katmanına robot manipülatörünün arzu edilen pozisyon ve oryantasyon bilgilerinin yanında hali hazırda bulunmuş olduğu konumdaki eklem açılarıda geri bildirim olarak verilmiştir. Bu sinir ağı yapısında yaptıkları testlerde ise 68 iterasyonda 3.3029e-08 gibi bir MSE değeri elde etmişlerdir. Bu da göstermektedir ki önermiş oldukları yeni sinir ağı yaklaşımı son derece başarılıdır (Almusawi, Dülger ve Kapucu, 2016).

Köker ve Çakar, 5-DOF robot koluna ait ters kinematik hesaplaması için YSA temelli genetik algoritma ve tavlama benzetimi içeren yeni bir model önermişlerdir. Önerilen bu

modelde YSA için giriş değerleri robot kolunun arzu edilen pozisyon ve oryantasyon bilgileridir. YSA çıkış parametreleri olan eklem açıları GA için başlangıç değerleri olarak kabul edilmiştir. GA içerisinde çaprazlama ve mustasyon ile oluşturulan yeni popülasyona birde tavlama benzetimi ilave edilmiştir. Böylece tavlama benzetimi ile GA'nın işlem süresinin kısaltılması amaçlanmıştır. Yapılan test işlemleri neticesinde GA ile 61. Jenerasyonda minimum pozisyon hatasına ulaşırken önerilen hibrit sistemle 29. Jenerasyonda arzu edilen değer elde edilmiştir (Köker ve Çakar, 2016, pp. 553-565).

Çavdar ve arkadaşları yapay arı kolonisi temelli yeni bir heuristik yöntem ile Puma 6-DOF robot manipülatörünün ters kinematik problemini çözmüşlerdir. Yapay arı kolonisi algoritmasının en iyi çözümlere yakınsama özelliğini güçlendirme üzerine geliştirdikleri yeni yöntemde; arılar yeni çözümler bulmak için YAK'ta tek bir "j" parametresini değiştirmektedir. Hâlbuki önerilen bu yöntemde birden fazla "j" parametresi değiştirilebilmektedir. Yapılan testlerde önerilen yöntemle PSO ve HSA (Harmony Search) pozisyon hatası ve çalışma süresi bakımından karşılaştırılmıştır. HSA ile 0,376 saniyede 10E-04'lerde çıkan sonuç PSO'da 0,0361 saniyede 10E-08'lerde çıkarken önerilen yöntemle 0,0286 saniyede 10E-13'lerde çıkmıştır (Çavdar, Mohammed ve Milani, 2012, pp. 329-333).

Rokbani ve arkadaşları, yapmış oldukları çalışmada üç eklemli planar bir sistemin ateş böceği algoritması ile çalışma uzayındaki konumunu elde etmişlerdir. Bunu yaparken tıpkı bu tez çalışmasında olduğu gibi daha karmaşık olan ters kinematik denklemleri yerine çok daha basit olan ileri kinematik denklemlerini kullanmışlardır. Eklemli sistemi hedef noktasına ulaştırırken ateş böceklerinin parlaklık düzeylerini kullanmışlar ve bu parametre çözüm için maksimum düzeyde belirleyici olmuştur. Yani ateş böcekleri eklem açılarını onların parlaklık seviyesi ise çözüme ne kadar yakın olduklarını göstermektedir. İteratif bir yapı içerisinde ateş böcekleri ileri kinematik denklemleri vasıtasıyla çözüme yaklaştırılmışlardır. Sonuçta milisaniye düzeyinde hesaplama süresi ve 10E-08 düzeyinde ise pozisyon hatası değerleri elde etmişlerdir (Rokbani, Casals ve Alimi, 2015, pp. 369-395).

Erkol ve Demirel, kinematik hesaplamaların karmaşık ve çözümünün uzun zaman aldığı tezinden yola çıkarak yaptıkları çalışmada her biri üç ekleme sahip altı bacaklı örümcek benzeri bir mobil robot tasarlamışlar ve bu robotun ters kinematik çözümünü FPGA ile elde etmişlerdir. Robotla iletişim kurmak için DSP, FPGA, XBee, IR içerien bir kontrol kartı tasarımı da gerçekleştirmişler ve bu kartın bilgisayarla haberleşmesinide seri port üzerinden yapmışlardır. Test sonuçlarına göre simülasyonla elde ettikleri saniye seviyelerindeki süreyi mikrosaniye seviyelerine çekmeyi başarmışlardır (Erkol ve Demirel, 2014, pp. 1164-1173).

El-Sherbiny ve arkadaşları, yaptıkları çalışmada özellik YAK algoritmasının arama uzayındaki yeni çözümleri keşfetme ve optimum çözüme yakınsama özelliklerinin çok iyi olmadığı tezinden yola çıkarak algoritmanın bu özelliklerini geliştirmeyi amaçlamışlar ve K-ABC ismini verdikleri YAK algoritması temelli yeni bir yaklaşım önermişlerdir. Yaptıkları değişiklikle arıların yeni yiyecek kaynaklarına yönlenirken bir önceki yiyecek kaynağına değil de o ana kadar ki en iyi çözüm etrafında bulunan yiyecek kaynaklarına yönelmelerini sağlamışlardır. Algoritmanın test işlemini hem sekiz adet test fonksiyonunda gerçekleştirmişler hemde 5-DOF robot manipülatörünün ters kinematik çözümünü gerçekleştirmişlerdir. Elde ettikleri ters kinematik çözümü PSO, YAK, YSA ve GA ile de karşılaştırmışlar ve en iyi değeri 10E-07 seviyelerinde olarak bu algoritma ile bulmuşlardır (El-Sherbiny, Elhosseini ve Haikal, 2018, pp. 24-38).

Soylak ve arkadaşları, çalışmalarında üç eklemli plasma kesme robotunun ters kinematik problemini YSA ile çözmüşlerdir. Tasaladıkları YSA'da girişler eklem açıları ve çıkışlar ise uç elemanın x, y, z koordinatlarıdır. Levenberg–Marquardt eğitme algoritması ile ağı eğitmişler ve eğitme değerlerinde bulunmayan eklem açıları ile de ağı doğrulamışlardır. Doğrulama işleminde aynı zamanda robota ait bir yörüngede kullanılmıştır (Soylak, Oktay ve Türkmen, 2017, pp. 470-479).

Bahrin ve arkadaşları, endüstri 4.0 devriminde robotların yerinin ne olacağı ile ilgili bir araştırma makalesi yayınlamışlardır. Makalede özellikle robotların bu endüstri aşamasında tamamıyla akıllı olacakları ve tek bir amaca yönelik değil çok amaçlı olarak üretileceklerini vurgulamışlardır. Yazarlar ayrıca bu sanayi devriminde robot

üreticilerinin asıl odaklanacakları konuların; güvenlik, esneklik, çok yönlülük ve işbirliği eksenlerinde olacağını belirtmişlerdir. Bu çalışmada göze çarpan diğer bir hususta; robotların sadece kendileri veya diğer makinelerle değil aynı zamanda insanların çalıştığı alanlardada insanlarla işbirliği içinde olacaklarının ortaya konmasıdır (Bahrin, Othman ve Azli, 2016, pp. 6-13).

Rokbani ve Alimi, parçacık sürü optimizasyonu ve onun hız kontrol parametrelerinden bazılarını kullanarak 2-eklemli bir robot koluna ait ters kinematik çözümü simülasyon tabanlı olarak gerçekleştirmişlerdir. Çalışmalarında random, doğrusal azalan, daralma faktörü, basitleştirilmiş olmak üzere beş farklı variant kullanmışlar ve sonuçları karşılaştırmalı olarak analiz etmişlerdir. Analizleri neticesinde basitleştirilmiş varyantlı PSO'nun çok daha etkili bir sonuç ortaya koyduğunu belirtmişlerdir (Rokbani ve Alimi, 2013, pp. 1602-1611).

Çavuşlu ve arkadaşları (Çavuşlu, Karakuzu ve Şahin, 2010, pp. 83-84) yaptıkları çalışmada önceden tasarlamış oldukları bir yapay sinir ağının eğitimini donanımsal olarak oluşturdukları parçacık sürü optimizasyonu ile gerçekleştirmişlerdir. Tasarladıkları donanımsal sistemde kullanılan bütün parametreleri IEEE 754 kayan noktalı sayı formatında tanımladıklarından dolayı Altera FPGA kartında gerçekleştirme anında kart üzerindeki lojik elemanların %70'ine yakınını kullanmak durumunda kalmışlardır. Ancak sistemin arzu edilen şekilde çalışarak ağın eğitiminde başarılı olduklarını vurgulamışlardır.

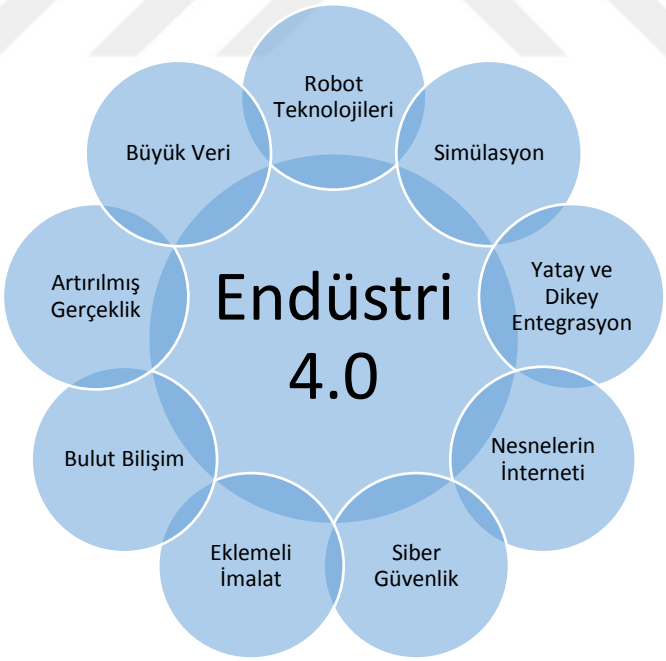
Allaire ve arkadaşları yaptıkları çalışmalarında insansız hava aracı için gerçek zamanlı yol planlaması yapan sistemi genetic algoritma ile tasarlamışlardır. Ancak hava aracını yönlendirmede süre konusunda sıkıntı yaşadıkları için genetik algoritmayı VHDL dili ile donanımsal olarak tasarlamışlar ve sonuçta yazılımsal olarak süreyi 10000 kata kadar azaltabilmişlerdir. Aslında donanımsal tasarımla etkili sonuçlar elde etme konusunda en çarpıcı ve bu tez çalışmasına da ilham kaynağı olan bu makaledir. Bu tez çalışmasında da neredeyse saniyeler düzeylerinde elde edilen pozisyon hatası değerleri milisaniye düzeylerinde elde edilmiştir (Allaire et al, 2009, pp. 495-510).

Bu tez çalışmasında 7-eklemli redundant (gereğinden fazla serbestlik dereceli) seri robot manipülatör tasarımı gerçekleştirildikten sonra DH parametreleri vasıtasıyla kinematik analizleri yapılmış ve denklemleri çıkarılmıştır. Bu denklemler aracılığıyla seri robot manipülatörünün ters kinematik problemi hem yazılımsal hemde donanımsal olarak çözülmüştür. Yazılımsal çözüm için sezgisel optimizasyon algoritmalarından parçacık sürü optimizasyonu, yapay arı kolonisi, ateş böceği ve quantum davranışlı parçacık sürü optimizasyonu algoritmalarından faydalanılmıştır. Ayrıca parçacık sürü optimizasyonu tekniğini iyileştirmek için hız kontrol parametrelerinden olan IW (inertia weight) teknikleri arasına RDV-IW (Random Descending Velocity – Rasgele Azalan Hızlı IW) ismiyle yeni bir teknik daha kazandırılmış ve yazılımsal çözümle gerçekleme işlemine bu teknikte ilave edilmiştir. Donanımsal ters kinematik çözümün gerçeklenmesi için Nexys 4 DDR FPGA donanım kartı kullanılmıştır. Bu kart ekseninde VHDL dili kullanılarak PSO algoritması donanımsal olarak tasarlanmış ve kart üzerinde çalıştırılmıştır. Çalışmada kullanılan tüm bu teknikler Matlab programında tasarlanıp kodlanan GUI arabirimi vasıtasıyla karşılaştırmalı bir şekilde kullanılmıştır. Matlab GUI penceresinden, bilgisayara seri port üzerinden bağlı olan 7-dof robot manipülatörü manuel olarak bir konuma yönlendirildikten sonra aynı GUI üzerinden seçilen yazılımsal veya donanımsal teknik ile bu konum en küçük hata ile hesaplanıp robot manipülatörü bu konuma yönlendirilmiştir.

BÖLÜM 3. MATERYAL VE YÖNTEM

3.1. Robot Manipülatörleri

Endüstri 4.0 isminin ziyadesiyle teleffuz edildiği günlerden geçmekteyiz. Şüphesiz bu telaffuzlar nihayetinde yeni bir standartın başlangıcı durumundadır. Endüstri 4.0, bir süreç içerisinde bulunan sistemleri ve cihazları birbirine bağlayarak sürecin herhangi bir insan müdahalesine maruz kalmadan tamamlanmasıdır. Yani bir fabrikada bir ürünün üretiminden önce, üretimi esnasında ve üretiminden sonraki aşamaların sanal-fiziksel bir ortam sayesinde cihazların birbirleriyle haberleşerek tamamlamalarıdır. Kısaca fabrikaların akıllı hale gelmeleridir (Lee, Bagheri ve Kao, 2015, pp. 18-23).

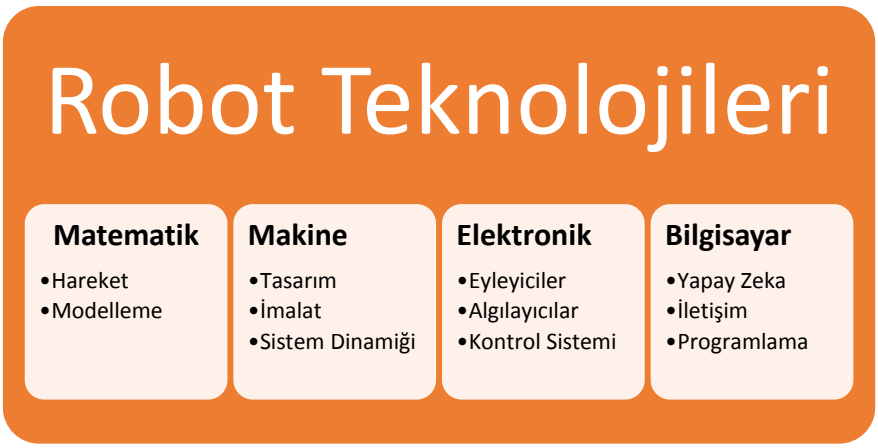


Şekil 3.1. Endüstri 4.0 standartı işlem döngüsü (Bahrin et al., 2016)

Her yeni bir endüstri devriminde olduğu gibi bu devrimde de robotlar yerlerini daha sağlamlaştırıp yine endüstrinin merkezinde olmaya devam edeceklerdir. Çünkü

endüstriyel üretimin vazgeçilmez unsurlarının başında büyük işleri kolaylıkla yapabilme, işleri en az hata ile tekrarlama gibi özelliklere sahip olan robotlar gelmektedir. Bu özelliklerine ilave olarak bu yeni devrimde robotlardan esnek ve çok yönlü çalışma, diğer makine veya nesnelerle işbirliğ yapma, insan gibi düşünerek akıllıca heraket etmek gibi özelliklere de sahip olması beklenmektedir (Bahrin et al, 2016, pp. 6-13; Lu, 2017, pp. 1-10).

İlk olarak Çek yazar Karel Çapek'in 1922'de bir oyununda kullanmasıyla tanınan robotlar, teknolojinin gelişimine paralel olarak pek çok evreden geçmiş ve farklı tasarımlarla karşımıza çıkmıştır. Bugün literatürde bir robot, materyal veya parçaları bir yörünge kapsamında bir yerden başka bir yere taşıma, yerleştirme, döndürme, montajını yapma gibi görevleri yerine getiren ve yeniden programlanabilen bir araç olarak tanımlanırken Oxford İngilizce sözlüğünde ise, "özellikle bilgisayar tarafından programlanabilen karmaşık bir dizi eylemi otomatik olarak gerçekleştirebilen bir makine" olarak tanımlanmaktadır (Ben-Ari ve Mondado, 2018, pp. 1-20) ve Şekil 3.2'de göründüğü gibi gelişimi için disiplinler arası işbirliği gerektirmektedir (Khan & Khan, 2017, pp. 38-45). Örneğin matematik bilimciler robotun hareketlerini tanımlayıp modellemesini yaparken, makine mühendisleri robotun tasarım, imalat ve sistem dinamiğiyle; elektrik ve elektronik mühendisleri aktuatörler, algılayıcılar ve kontrol sistemiyle; bilgisayar mühendisleri ise yapay zekâ, iletişim ve robotun programlanmasıyla ilgilenirler (Akı, 2010).



Şekil 3.2. Robot teknolojisini oluşturan disiplinler

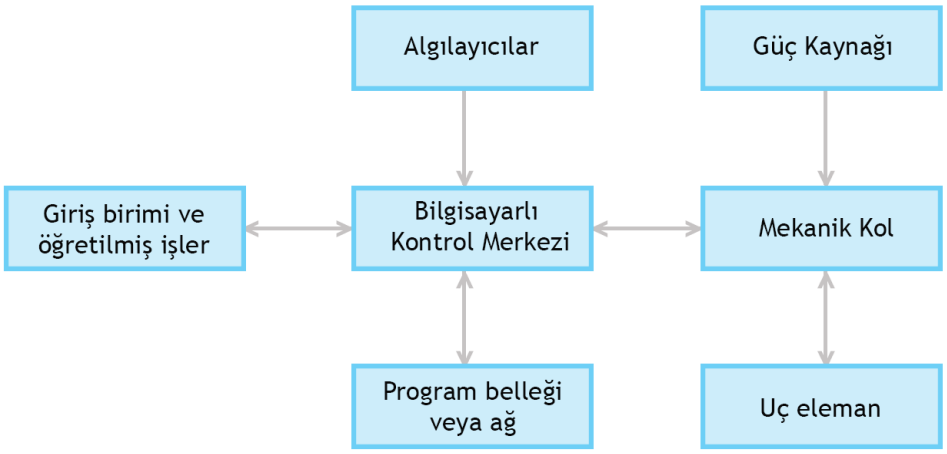
Robotlarla beraber hayatımıza da pek çok kavram girmiştir. Böylece robotları ayırırken ve karşılaştırırken bu kavramlar ayırt edici bir etkiye sahip olmuşlardır. Bu ifadeler şu şekilde sıralanabilir (Jazar, 2010):

- Tekrarlanabilirlik (repaetability): Bir robota verilen bir işin defalarca aynı şekilde yapılmasıdır.
- Çözünürlük (resolution): Robot uç elemanının en küçük yer değiştirme değeri olarak tanımlanır. Bu da elbette eklemleri hareket ettiren aktuatörlerin hareket kabiliyeti ile doğrudan ilgilidir.
- Serbeslik Derecesi (Degree of freedom): Eklemlerle yani aktuatörlerle oluşturulan bağımsız uzuvların sayısıdır.
- Manipülatör: Eklemlerle birbirine bağlanmış ve kinematik bir zincir oluşturan sistemlerdir.
- Eyleyici (Aktuatör): İki eklemi birbirine bağlamak için kullanılan tahrik sistemidir. Hidrolik ve servo motor sistemleri günümüzde yaygınca kullanılmaktadır.
- Çalışma Alanı (work-space): Robotun uç elemanının uluşabildiği maksimum konumlarının yatay düzleme düşen izdüşümü olarak ifade edilir. Bir diğer ifadeyle robot kolunun ulaşabileceği toplam çalışma alanıdır.
- Yük Kapasitesi (Payload): Uç elemanın taşıyabileceği parça veya malzemenin ağırlığıdır.
- Doğruluk (Accuracy): Robota ait uç elmanın belirlenen bir noktaya her dafasında ulaşabilme yeteneğidir.

Pek çok nesne gibi robotlarında işlevini tam manasıyla yerine getirebilmek için özellikle mekanik aksam (gövde) itibariyle dört temel unsura sahip olması gerekmektedir ve aşağıda görünmektedir (Alp, 2012).

- Mekanik Kısım: Robotun iskelet kısmıdır.
- Uç Eleman: Bir robotun gerçek manada işi yapan kısmıdır.
- Aktuatörler: Eklemleri hareket ettirmek suretiyle uç noktayı istenen bir konuma taşımak için kullanılan kısımdır.
- Kontrol Birimi: Robotun yapacağı işi tanımlayan, heraketini programlayan kısmıdır.

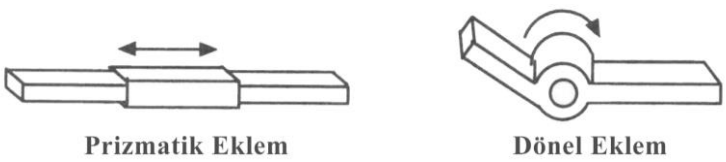
Ancak bir robotik sistem bundan çok daha fazlasıdır, çünkü bütün bu elemanlardan oluşan sistemin istediğimiz gibi çalışabilmesi için bir sisteme dönüşmesi gerekmektedir. Bu sistem Şekil 3.3'te görünmektedir (Spong, Hutchinson ve Vidyasagar, 2006, pp. 7). Göründüğü üzere mekanik yapı bu sistem içerisinde tek bir parça olarak görünmektedir.



Şekil 3.3. Robotik sistemi oluşturan üniteler

3.1.1. Robotların Sınıflandırılması

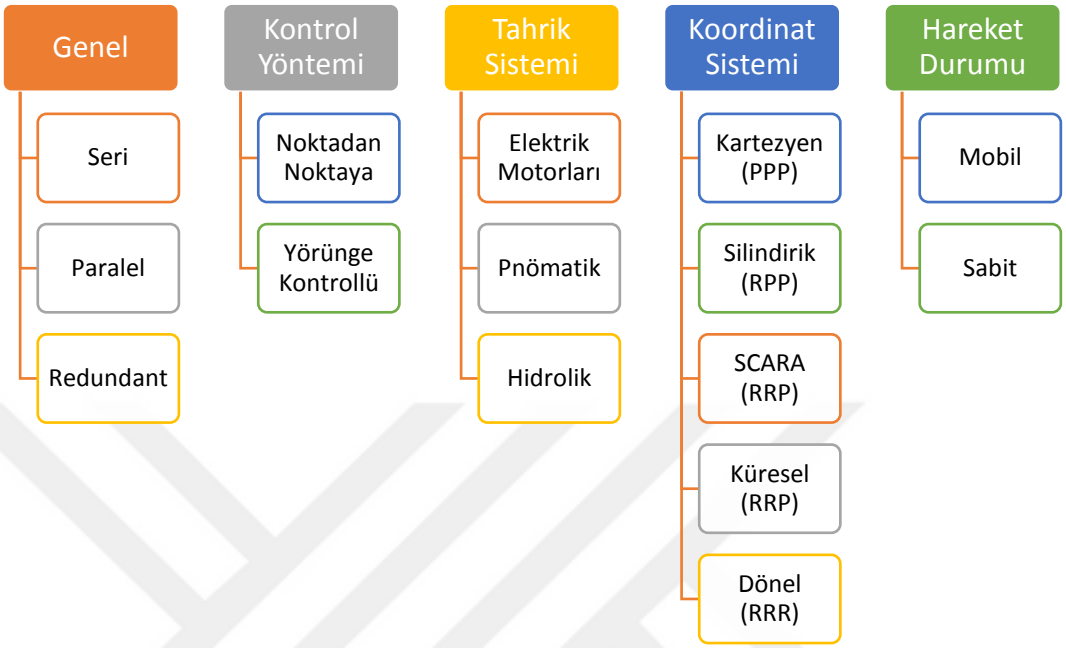
Bir robot birbirine gore bağımsız hareket edebilen linkler ve bu linkleri birbirine bağlamak için kullanılan aktuatörlerin oluşturduğu eklem yapılarından oluşur. Bu eklem yapıları, öteleme (Prismatic) ve dönme (Revolute) hareketi yaparak uç elemanı çalışma uzayında belli bir konuma yönlendirirler. Öteleme hareketi ile oluşan yer değiştirmeye eklem kayması (joint offset), dönme hareketi ile oluşan yer değiştirmeye ise eklem açısı (joint angle) adı verilir ve bu hareketler Şekil 3.4 de görünmektedir.



Şekil 3.4. Dönme ve öteleme hareketleri (Bingül ve Küçük, 2015, pp. 4)

Robotlar serbestlik derecelerine, kontrol yöntemlerine, tahrik sistemine, eklem yapılarına, hareket durumuna göre sınıflandırılmaktadır. Ancak bunlar arasında en yaygın

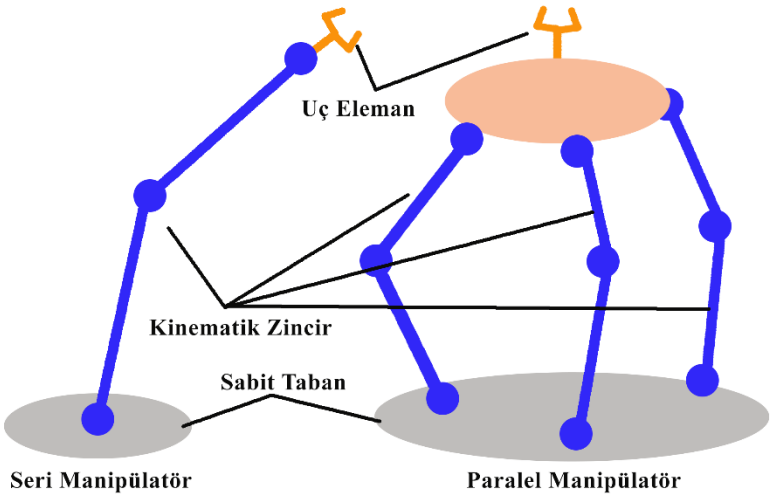
kullanılanılanı ise eklem yapılarına göre yapılan sınıflandırmadır (Bingül ve Küçük, 2015, pp. 7).



Şekil 3.5. Robotların sınıflandırılması

3.1.2. Genel duruma göre robotlar

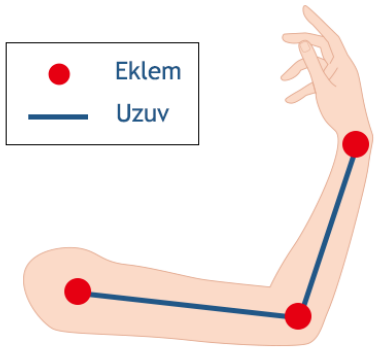
Bu çalışmada, robotlar genel olarak; seri, paralel ve gereğinden fazla (redundant) olmak üzere üçe ayrılmaktadır.



Şekil 3.6. Seri ve paralel manipülatör (Francis, 2018)

3.1.2.1. Seri manipülatörler

Seri manipülatörler, genellikle prizmatik ve dönel olmak üzere çeşitli bağlantılar ile birbirine bağlanarak oluşturulan mekanizmalardır. Sabit bir tabanla birlikte belli sayıda uzuv ve bir adet uç elemandan oluşur. Paralel manipülatörün aksine açık döngülü bir kinematik yapıya sahiptirler (Raj, Iyer ve Dash, 2016). Şekil 3.7'te üç eklemli yapıya sahip olan bir insan kolu görünmektedir. Bu yapıdan esinlenilerek oluşturulduğundan ve bu yapıya benzerliğinden dolayı seri manipülatörler aynı zamanda robot kolu olarakta anılmaktadır (XYZ, 2018).



Şekil 3.7. İnsan kolu ile seri robot kolu benzerliği

3.1.2.2. Paralel manipülatörler

Literatürde sıklıkla platform manipulatör olarakta adını duyduğumuz bu manipulatörler, sabit bir konuma bacaklarla bağlantılı, hareketli bir mekanizma olarak tanımlanmaktadır (Can, 2008). Başka bir tanımda ise, kapalı döngü kinematik zincir sistemine sahip ve bir tabana (base) sabitlenmiş şekilde hareketli bacaklarla (veya kollarla) uç elemanı bu tabana bağlı olan mekanizmadır (Merlet, 1996, pp. 17-24).

Paralel manipülatörlerin seri olanlara gore en önemli avantajı ağır yük kaldırabilme imkanlarıdır. Şekil 3.8'de göründüğü gibi, bir kaç bacak veya kol bu yükün ağırlığını paylaşmaktadır. Hâlbuki seri manipülatörde ise tek bir kol bu yükün ağırlığını çekmek zorundadır. Bu manipülatörlerin diğer önemli bir avantajı ise hassas çalışabilme

yetenekleridir. Ancak en önemli dezanatajı ise küçük çalışma uzayına sahip olmasıdır (Ruiz et al, 2018, pp. 809-821).

Şekil 3.8. Paralel Delta manipülatör

Paralel manipülatörler piyasada küçük parçaları paketleme işinden, ağır parçaların montajına, hassas lehimleme işinden parçaları bir banttan yakındaki diğer banta aktarmaya kadar pek çok işte kullanılmaktadır. Platform olarak adlandırılmaları ise uçuş ve deprem simülatörleri olarak kullanılma durumundandır (Schreiber ve Gosselin, 2018, pp. 91-105).

3.1.2.3. Redundant manipülatörler

Tipik olarak robot manipülatörlerin ana görevi, çalışma uzayında en az hata ile istenen yörüngede ilerlemesidir. Fakat pratikte robot eklemleri fiziksel olarak bazı sınırlamalara sahiptir. Ayrıca pek çok uygulamada robotların çalışma uzayında engeller bulunmaktadır. Bu problemlerin üstesinden gelmek için gereğinden çok serbestlik dereceli robotlar (redundant manipulators) tercih edilir (Toshani ve Farrokhi, 2014, pp. 766-781). Gereğinden çok serbestlik dereceli robot kolları (redundant manipulators); değişkenlerine (Ɵ ve d) sonsuz sayıda çözüm üretebilen robot kolları olarak tanımlanır (Çonkur, 2003). Bu robot kolları iç ve dış engeller ortaya çıktığında değişik konfigürasyonlar seçerek bu engeller arasından geçebilirler (Eren, 2006). Çünkü üç boyutlu çalışma uzayındaki

herhangi bir noktaya ulaşmak için altı serbestlik derecesi yeterli olduğu için standart sanayi robotlarının en fazla sahip olabileceği serbestlik derecesi altıdır.

Sınırlı sayıdaki serbestlik derecesinin sebep olduğu çok sayıda problem vardır. Bunlardan biri, robotun çalışma alanının bir kısmının tekillikler yüzünden (singularity) kullanılamamasıdır. Bir diğeri ise ters kinematiği için sınırlı sayıda çözüm olmasından dolayı, robotun her zaman çalışma alanında engellerden kaçınacak şekilde kendini ayarlayamamasıdır. Gereğinden çok serbestlik dereceli robot kolları ise engellerden kaçınacak tarzda kollarını istediği gibi ayarlayabilir. Böylece her türlü karışık ortama uyum sağlayabilmesi ve girilmesi zor bölgelere rahatlıkla girmesi mümkündür (Shi et al, 2018, pp. 1-11). Bununla birlikte serbestlik derecesi altıdan fazla olan bu robot manipülatörlerinde artıklık (redundancy) durumu söz konusu olur. Bu durumda, manipülatörün çalışma uzayında eklemlerden herhangi biri tarafından kullanılan alan başka bir eklem tarafından da kullanılabileceği için çakışma durumu oluşur. Bu nedenlede bu manipülatörler ancak özel durumlar için kullanılmak üzere tasarlanırlar (Bingül & Küçük, 2015, pp. 5).

"Redundant Robot", sınırsız sayıda kinematik çözümü mevcut olan, buna karşın engelleri kolaylıkla aşabilen, boşluklardan geçebilen, hünerli bir şekilde nesnelere ulaşabilen robottur. Diğer bir ifadeyle; bir robot sistemi görevini yerine getirirken sonsuz sayıda eklem konfigürasyonu alıyorsa "Kinematik artıklık" meydana geliyor demektir (Neythalath, Brandstötter ve Hofbaur, 2016, pp. 31-38).

Redundant robotların başlıca kullanım alanları şu şekilde sıralanabilir (Eren, 2006):

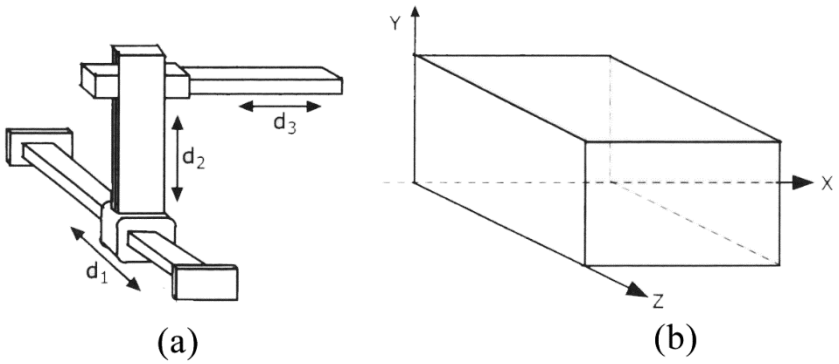
- Büyük makinelerin içlerine tamir ve bakım için girebilmektedirler.
- Serbestlik dereceleri sınırlı olan ve çalışma ortamları çok itinayla hazırlanması gereken günümüzdeki endüstriyel robotların hareket kabiliyetlerini çok fazla arttırabilmektedirler.
- Uzay istasyonu inşası gibi el becerisi, mikro-elektronik imalatı gibi vakum ortamı gerektiren çok çeşitli işleri yapabilmektedirler.
- Bazı beyin ameliyatlarında, cerrahın elle ulaşmasının çok zor olduğu beyin kısımlarına ulaşmada kullanılabilmektedirler.

3.1.3. Koordinat sistemine göre robotlar

Robotlar, bu sınıflandırmada ilk üç eklemin yapısına göre değerlendirilmektedir. Bunun için robotlar; ilk üç eklem yapısı tamamen prizmatik olan Kartezyen, birinci eklem dönel, ikinci ve üçüncü eklemi prizmatik olan silindirik, ilk iki eklem dönel ve üçüncüsü prizmatik olan küresel ve SCARA, tüm eklemleri dönel olan için RRR olmak üzere beş sınıfa ayrılmaktadır (Spong, Hutchinson and Vidyasagar, 2006, pp. 5-7; Bingül ve Küçük, 2015, pp. 7-14; Robotpark, 2015).

3.1.3.1. Kartezyen (PPP) robotlar

Kinematik olarak en basit olan robot grubundan olup tüm eklemleri öteleme hareketi yaptığından dolayı bu ismi almış olan robotlardır. Literatürde kızak robot olarakta isimlendirilmektedir.



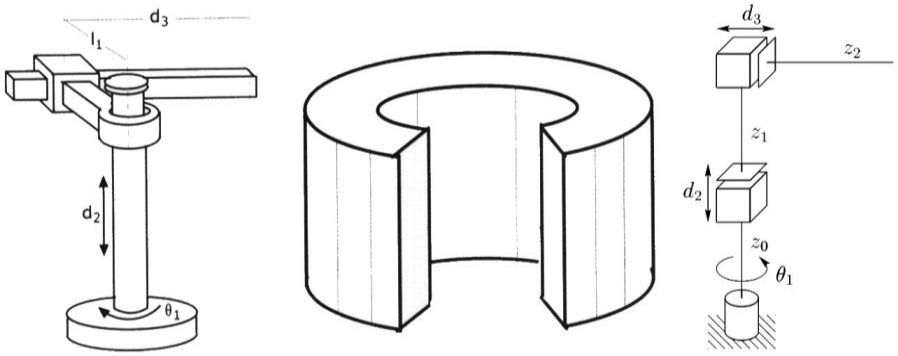
Şekil 3.9. Kartezyen robot (a) ve çalışma uzayı (b) (Robotpark, 2015)

Yapısının basit olması nedeniyle kontrolünün de basit olması, yük kaldırma kapasitesinin fazla olması ve çalışma uzayının her alanına ulaşma kabiliyetinin aynı olması üstünlüklerindendir. Buna rağmen çalışma uzayının son derece sınırlı olması bu robotun en önemli dezavantajıdır. En basit uygulaması, freze makinesinde frezeleme işlemi esnasında uç kısmın x-y ekseninde hareket ederken aynı zamanda yukarı-aşağı hareketi gösterilebilir (Ayyıldız, 2018, pp. 575-580).

Şekil 3.10. Örnek bir kartezyen robot (Robotpark, 2015)

3.1.3.2. Silindirik (RPP) robotlar

Eksenleri silindirik bir koordinat sistemi oluşturan, bir adet dönme hareketi iki adet öteleme hareketi yapan ekleme sahip olan robot türüdür. Dönel bir koordinat sistemine sahip olması uç elemanın hızlı hareket etmesine yardım eder, Kartezyen robotlara göre daha geniş çalışma alanına sahiptir (Bingül & Küçük, 2015, pp. 10).

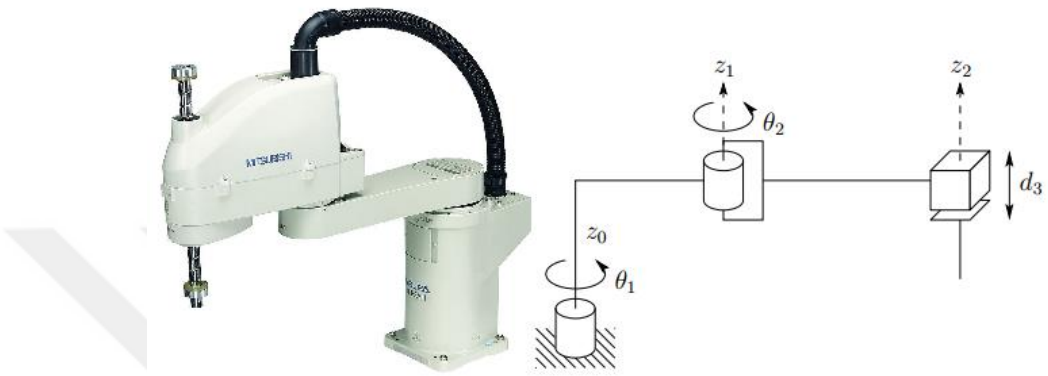


Şekil 3.11. Silindirik robot ve çalışma uzayı (Robotpark, 2015; Spong, Hutchinson and Vidyasagar, 2006, pp. 15)

Bu robotlar genellikle, nokta kaynaklama, makine araçlarını kontrol etme ve montaj operasyonları gibi işlevler için kullanılmaktadırlar (Spong, Hutchinson and Vidyasagar, 2006, pp. 14).

3.1.3.3. SCARA (RRP) robotlar

SCARA (Selective Compliant Assembly Robot Arm - Seçici Uyumlu Montaj Robot Kolu) robotlar, koordinat çerçevesi bakımından küresel bir robot gibi görünmesine rağmen geometrik açıdan bakıldığında ise küresel robottan tamamen farklıdır.

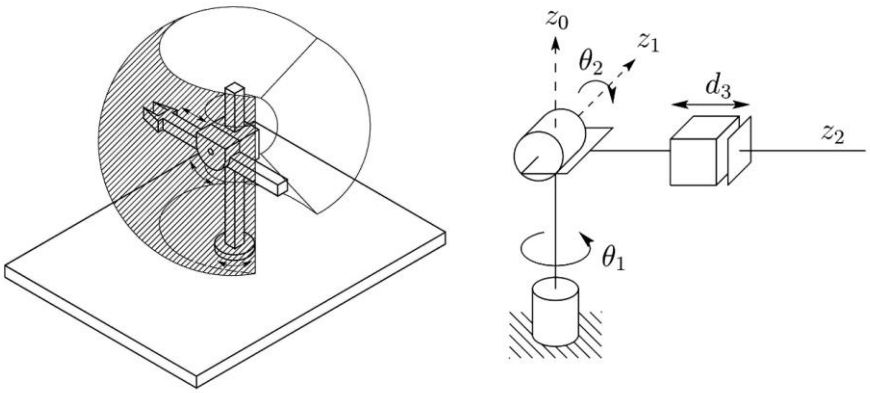


Şekil 3.12. SCARA (RRP) robot ve çalışma uzayı (Spong, Hutchinson and Vidyasagar, 2006, pp. 13)

Şekil 3.12'de göründüğü gibi paralel eklem dizilimi sayesinde SCARA robot, x ve y eksenlerine oldukça esnek hareket kabiliyeti olmasına rağmen z eksenine ise bu konuda son derece katıdır. Bu durum robotun seçici uyumlu olabilmesine olanak tanımakla birlikte ve montaj işleri için son derece uyumlu olmasını sağlamıştır. Ayrıca iki eklemli katlanabilir kol yapısı da SCARA robota önemli bir katkı yapmıştır. Çünkü bu kol sayesinde robot sınırlı alanlara ulaşabilmekte ve birbirine yakın istasyonlarda yükleme ya da boşaltma yapabilmektedir.

3.1.3.4. Küresel (RRP) robotlar

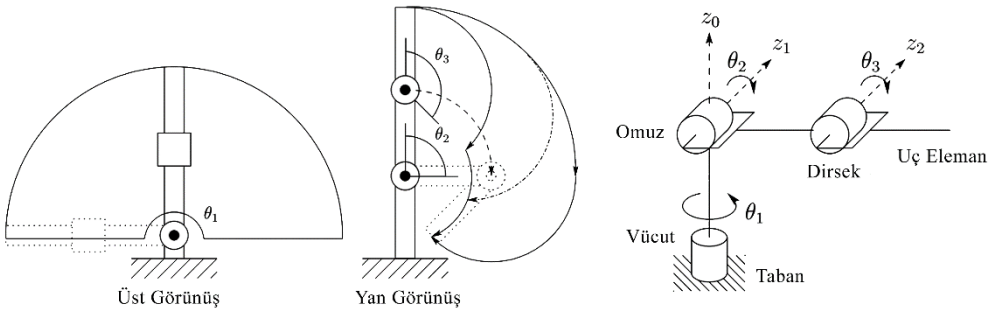
Bu robotların iki adet dönel ve bir adet prizmatik eklemleri bulunmakta olup SCARA robotlar gibi endüstride ziyadesiyle kullanılmaktadır. En fazla çalışma uzayına sahip robot tipi olmasına karşın çok karmaşık kinematik denklemlere sahiptir. Bu nedenle de kontrolleri oldukça zordur.



Şekil 3.13. Küresel robot ve çalışma uzayı (Robotpark, 2015; Spong, Hutchinson and Vidyasagar, 2006, p. 12)

3.1.3.5. Dönel (RRR) robotlar

Literatürde eklemli (articulated) olarak anılan dönel (revolute) robotların bütün eklemleri dönel cinstendir. Piyasada en çok bilinenleri elbow manipülatördür. Bu robotlar büyük bir çalışma uzayına sahiptirler, eklemlerin tamamı dönel olduğu için hareketi nispeten kolaydır. Ayrıca hız ve esnek yapılarıyla da ön planda olan robotlardır. Buna karşın çalışma uzayındaki her noktaya ulaşamaz ve kinematik denklemleride oldukça karmaşıktır. Her eklemin oluşturduğu küçük hatalar uç noktaya doğru zincirleme olarak artabileceği için doğruluğu daha düşük olabilmektedir (Bingül & Küçük, 2015, pp. 14).



Şekil 3.14. Dönel robot yapısı ve çalışma uzayı (Spong, Hutchinson and Vidyasagar, 2006, p. 14)

3.1.4. Çalışmada kullanılan robot manipülatörü

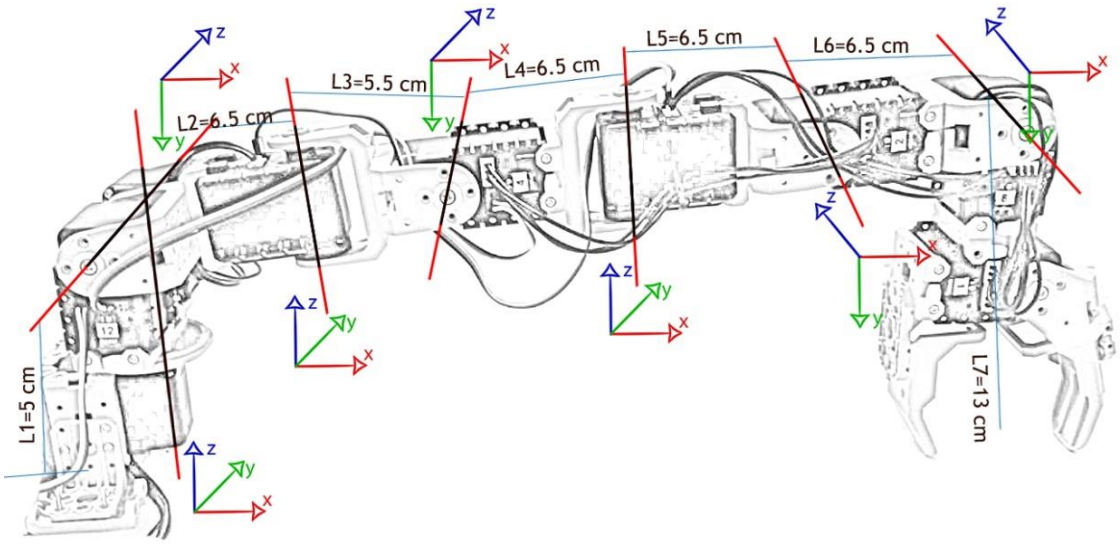
Bu çalışmanın amacı, zeki optimizasyon tekniklerinin çalışma biçimlerini incelemenin yanında bu tekniklerin çok karmaşık ve çözümü uzun zaman alan problemlerdeki davranışlarını ortaya koyarak sonuçları analiz etmektir. Robotik alanda ileri kinematik

probleminin çözümü son derece basitken ters kinematik probleminin çözümü ise bir o kadar zordur. Özellikle, eklem sayısı arttıkça robotik sistemin karmaşıklığı artacağından problemlerin çözümü de o doğrultuda artmaktadır. Ayrıca, dönel eklemlerin çözümü prizmatik eklemlerin çözümünden çok daha zordur.

Literatürde, ters kinematik problem analitik yöntemlerle, onun yetersiz kaldığı durumlarda ise sayısal yöntemlerle çözülmüştür (Özkarakoç, 2009). Ancak sayısal yöntemler gereğinden fazla serbeslik derecesine (redundant) sahip robot manipülatörlerinin ters kinematik çözümünde yetersiz kaldıklarından yapay zekâ teknikleri bu aşamada devreye girmiştir. Bu çalışmada da bunun bir örneği ortaya konmuştur.

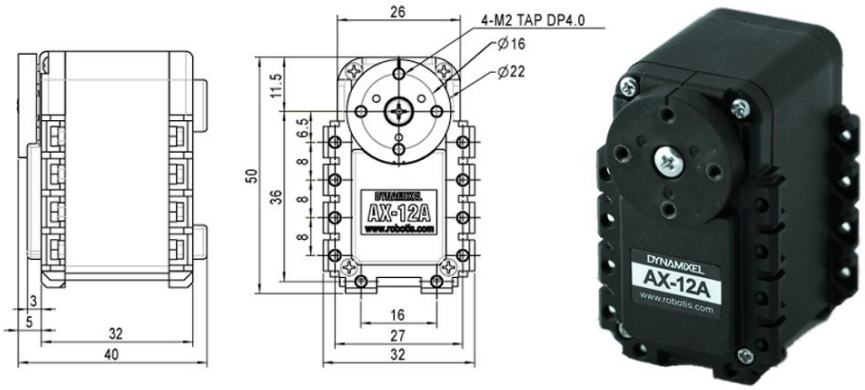
3.1.4.1. Robot manipülatörünün özellikleri

Çalışmanın amacına uygun olması itibariyle Şekil 3.15'te görünen 7-DOF robot manipulatorü tasarlanmıştır. Seçilen robot manipülatörü, 7-dönel (revolute) eklemden oluşan seri bir manipulator olup gereğinden fazla eklem sayısına sahip olduğundan dolayı redundant özelliklere sahiptir.



Şekil 3.15. Bu çalışmada kullanılan 7-DOF robot manipülatörü

Robot manipülatöründe bağlantı noktaları, eyleyiciler, uzuvlar, uç eleman ve kontrol ünitesi olmak üzere tamamı Robotis "Bioloid" serisinden "Premium Kit" kullanılarak gerçekleştirilmiştir (Robotis Premium, 2012). Manipülatörün en önemli unsuru olan eyleyiciler ise yine aynı firma tarafından üretilen ve sette bulunan Dynamixel AX-12A servo motorlar olup Şekil 3.16'da görünmektedir.

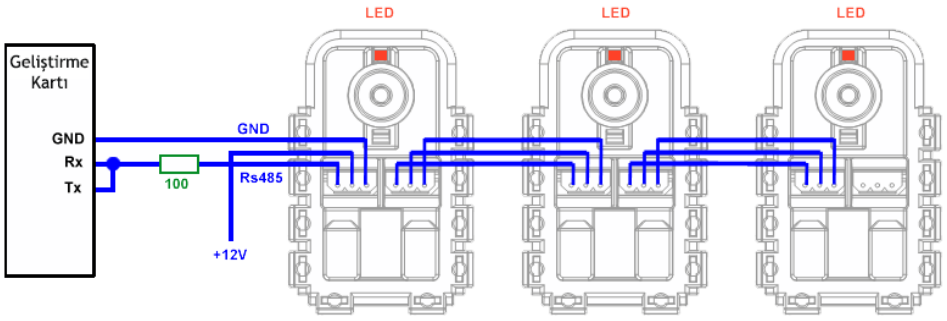


Şekil 3.16. Dynaixel Ax-12A servo motor

Pek çok test amaçlı projede yaygın bir kullanım alanı bulunan Dynamixel servo motorlarının en önemli avantajı zincir bağlantı şeklinde olan yapısıdır. Bu çalışmada kullanılan AX-12A serisinin özellikleri ise şu şekildedir:

- Ağırlık: 53.5g (AX-12/AX-12+), 54.6g (AX-12A)
- Boyutlar: 32mm * 50mm * 40mm
- Çözünürlük: 0.29°
- Sabit Tork: 1.5N.m (12.0Volt'ta, 1.5A)
- Yüksüz Hız: 59 rpm (12Volt'ta)
- Çalışma Derecesi: 0° ~ 300°
- Çalışma Sıcaklığı: -5℃ ~ +70℃
- Voltaj: 9 ~ 12V (Tavsiye edilen voltaj: 11.1V)
- Komut Sinyali: Dijital paket
- Haberleşme Hızı: 7343bps ~ 1 Mbps
- Geri besleme: Pozisyon, Sıcaklık, Yük, Giriş Voltajı, vb.

Dynamixel servolar zincir bağlantı şekline sahiptir. Yani iki adet servo motor bağlantı kablosu yardımı ile iki adet servo motor bir geliştirme kartına bağlanabilir ve aynı anda kontrol edilebilir.



Şekil 3.17. Dynamixel servoların bağlantı şekli

Bu çalışmada kullanılacak 7-DOF robot manipülatörü tasarlanırken toplamda dokuz adet Dynamixel Ax-12a servo motor kullanılmıştır. Manipülatörün ikinci eklemi olan ana eksende tek servo bağlantısı yetersiz kaldığından bu eksene ikinci bir servo montajı yaparak tork kuvvetlendirilmiştir. İlave bir servoda uç eleman için kullanılmıştır. Yani haricen bir uç eleman kullanmak yerine bu servolardan bir tanesi kullanılarak bir uç eleman yapılmıştır.

3.1.4.2. Robot manipülatörünün kinematik analizi

Hareket bilimi olarak anılan kinematik aslında, hareketin nedenlerini sorgulamaksızın özelliklerini ve sonuçlarını inceleyen bilim dalıdır (Gümüş, 2017, pp. 75-151). Robot manipülatörlerinde kinematik performans doğrudan robotun yapısıyla ilişkili olup manipülatörün çalışma uzayında esnek ve hünerlice çalışmasına yardımcı olur (Küçük ve Bingül, 2006, p. 567). Robot manipülatörlerinde kinematik, eklem açıları ve bu eklemler arasındaki mesafe ile uç elemanın çalışma uzayındaki konumu arasındaki ilişkiyi ortaya koyar. Bunun için bir manipülatörün ileri ve ters kinematik olmak üzere iki tür kinematik analizi yapılır. Daha kolay olan ve eklem açıları belli iken uç elemanın konumunu bulmak ileri kinematiktir; daha zor ve karmaşık olan ve uç elemanın konumu yani x, y ve z koordinatları belli iken eklem açılarını bulmak içinse ters kinematik kullanılır (Ben-Ari ve Mondada, 2018, pp. 267-291). Bunun için öncelikle manipülatörün kinematik

analizinin yapılması gerektiğinden çalışmada bu analizler için DH (Denavit-Hartenberg) yöntemi kullanılmaktadır. Tablo 3.1'de 7-DOF robot manipülatörünün DH parametreleri görünmektedir.

Tablo 3.1. DH parameters for robot manipulator

i	a_i(cm)	α_i (°)	d_i(cm)	Θ_i (°)(Range)
1	0	-90	L_1=5	$-90 < \Theta_1 < 90$
2	L_2=6,5	-90	0	$-180 < \Theta_2 < 0$
3	L_3=5,5	-90	0	$-90 < \Theta_3 < 90$
4	L_4=6,5	-90	0	$-90 < \Theta_4 < 90$
5	L_5=6,5	90	0	$-90 < \Theta_5 < 90$
6	L_6=6,5	0	0	$-90 < \Theta_6 < 90$
7	L_7=13	0	0	$-90 < Q_7 < 90$

Tablo 3.1'de çıkarılan parametrelerden, (Denklem 3.1)'de kullanılan genel dönüşüm matrisi yardımıyla her bir eklemin dönüşüm matrisi çıkarılmıştır. En sonunda (Deneklem 3.2) ile bütün matrislerin çarpımıyla da ileri yön kinematik matrisi oluşturulmuştur. Bu denklemlerde *sθ sinθ*'yı, *cθ* ise *cosθ*'yı ifade etmektedir.

$$ {}^{i}_{i-1}T = \begin{bmatrix} cos\theta\mathrm{i} & -cos\alpha i.sin\theta\mathrm{i} & sin\alpha i.sin\theta\mathrm{i} & ai.cos\theta\mathrm{i} \\ sin\theta\mathrm{i} & cos\alpha i.cos\theta\mathrm{i} & -cos\theta\mathrm{i}.\sin\alpha\mathrm{i} & ai.sin\theta\mathrm{i} \\ 0 & sin\alpha i & cos\alpha i & di \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

$$ A_{\mathrm{End-Effector}} = {}^{7}_{0}T = {}^{1}_{0}T.{}^{2}_{1}T.{}^{3}_{2}T.{}^{4}_{3}T.{}^{5}_{4}T.{}^{6}_{5}T.{}^{7}_{6}T = \begin{bmatrix} n_x & s_x & a_x & P_x \\ n_y & s_y & a_y & P_y \\ n_z & s_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

$$ {}^{1}_{0}T = \begin{bmatrix} c\theta_1 & 0 & -s\theta_1 & 0 \\ s\theta_1 & 0 & c\theta_1 & 0 \\ 0 & -1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{2}_{1}T = \begin{bmatrix} c\theta_2 & 0 & -s\theta_2 & L_2c\theta_2 \\ s\theta_2 & 0 & c\theta_2 & L_2s\theta_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

$$
{}^{3}_{2}T = \begin{bmatrix} c\theta_3 & 0 & -s\theta_3 & L_3c\theta_3 \\ s\theta_3 & 0 & c\theta_3 & L_3s\theta_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{4}_{3}T = \begin{bmatrix} c\theta_4 & 0 & -s\theta_4 & L_4c\theta_4 \\ s\theta_4 & 0 & c\theta_4 & L_4s\theta_4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{5}_{4}T = \begin{bmatrix} c\theta_5 & 0 & s\theta_5 & L_5c\theta_5 \\ s\theta_5 & 0 & -c\theta_5 & L_5s\theta_5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{6}_{5}T = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & L_6c\theta_6 \\ s\theta_6 & c\theta_6 & 0 & L_6s\theta_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{7}_{6}T = \begin{bmatrix} c\theta_7 & -s\theta_7 & 0 & L_7c\theta_7 \\ s\theta_7 & c\theta_7 & 0 & L_7s\theta_7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

(Denklem 3.2)'de görünen işlem neticesinde ortaya çıkan matriste n_x, n_y, n_z, s_x, s_y, s_z, a_x, a_y ve a_z yönelim elemanlarını ifade ederken p_x, p_y ve p_z ise konum elemanlarıdır.

$$
\begin{aligned}
n_x = -c\theta_7(&s\theta_6(s\theta_4(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3) - c\theta_1c\theta_4s\theta_2) \\
&- c\theta_6(c\theta_5(c\theta_4(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3) + c\theta_1s\theta_2s\theta_4) \\
&- s\theta_5(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3))) \\
&- s\theta_7(c\theta_6(s\theta_4(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3) - c\theta_1c\theta_4s\theta_2) \\
&+ s\theta_6(c\theta_5(c\theta_4(s\theta_1s\theta_3 + c\theta_1c\theta_2c\theta_3) + c\theta_1s\theta_2s\theta_4) \\
&- s\theta_5(c\theta_3s\theta_1 - c\theta_1c\theta_2s\theta_3)))
\end{aligned} \tag{3.4}
$$

$$
\begin{aligned}
n_y = c\theta_7\Big(&s\theta_6(s\theta_4(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1) + c\theta_4s\theta_1s\theta_2) \\
&- c\theta_6\big(c\theta_5(c\theta_4(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1) - s\theta_1s\theta_2s\theta_4) \\
&- s\theta_5(c\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3)\big)\Big) \\
&+ s\theta_7(c\theta_6(s\theta_4(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1) + c\theta_4s\theta_1s\theta_2) \\
&+ s\theta_6\big(c\theta_5(c\theta_4(c\theta_1s\theta_3 - c\theta_2c\theta_3s\theta_1) - s\theta_1s\theta_2s\theta_4) \\
&- s\theta_5(c\theta_1c\theta_3 + c\theta_2s\theta_1s\theta_3)\big))
\end{aligned} \tag{3.5}
$$

$$
\begin{aligned}
n_z = c\theta_7\big(&c\theta_6(c\theta_5(c\theta_2s\theta_4 - c\theta_3c\theta_4s\theta_2) - s\theta_2s\theta_3s\theta_5) \\
&+ s\theta_6(c\theta_2c\theta_4 + c\theta_3s\theta_2s\theta_4)\big) \\
&- s\theta_7(s\theta_6(c\theta_5(c\theta_2s\theta_4 - c\theta_3c\theta_4s\theta_2) - s\theta_2s\theta_3s\theta_5) \\
&- c\theta_6(c\theta_2c\theta_4 + c\theta_3s\theta_2s\theta_4))
\end{aligned} \tag{3.6}
$$

$$
\begin{aligned}
s_x = s\theta_7(s\theta_6\Big(&s\theta_4(s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3 - c\theta_1 c\theta_4 s\theta_2)\\
&- c\theta_6\big(c\theta_5(c\theta_4(s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4)\\
&- s\theta_5(c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3)\big)\Big)\\
&- c\theta_7(c\theta_6(s\theta_4(s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) - c\theta_1 c\theta_4 s\theta_2)\\
&+ s\theta_6\big(c\theta_5(c\theta_4(s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4)\\
&- s\theta_5(c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3)\big))
\end{aligned}
\tag{3.7}
$$

$$
\begin{aligned}
s_y = c\theta_7\Big(&c\theta_6(s\theta_4(c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) + c\theta_4 s\theta_1 s\theta_2)\\
&+ s\theta_6\big(c\theta_5(c\theta_4(c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4)\\
&- s\theta_5(c\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3)\big)\Big)\\
&- s\theta_7(s\theta_6(s\theta_4(c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) + c\theta_4 s\theta_1 s\theta_2)\\
&- c\theta_6\big(c\theta_5(c\theta_4(c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4)\\
&- s\theta_5(c\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3)\big))
\end{aligned}
\tag{3.8}
$$

$$
\begin{aligned}
s_z = -c\theta_7\big(&s\theta_6(c\theta_5(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5)\\
&- c\theta_6(c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4)\big)\\
&- s\theta_7(c\theta_6(c\theta_5(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5)\\
&+ s\theta_6(c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4))
\end{aligned}
\tag{3.9}
$$

$$
\begin{aligned}
a_x = s\theta_5(c\theta_4(s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4) + c\theta_5(&c\theta_3 s\theta_1\\
&- c\theta_1 c\theta_2 s\theta_3)
\end{aligned}
\tag{3.10}
$$

$$
\begin{aligned}
a_y = -s\theta_5(c\theta_4(c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4) - c\theta_5(&c\theta_1 c\theta_3\\
&+ c\theta_2 s\theta_1 s\theta_3)
\end{aligned}
\tag{3.11}
$$

$$
a_z = s\theta_5(c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) + c\theta_5 s\theta_2 s\theta_3 \tag{3.12}
$$

$$
\begin{aligned}
p_x = {} & L_2 c\theta_1 c\theta_2 - L_5 s\theta_5 (c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3) + L_3 s\theta_1 s\theta_3 \\
& + L_5 c\theta_5 (c\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4) \\
& - L_6 s\theta_6 (s\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) - c\theta_1 c\theta_4 s\theta_2) \\
& - L_7 c\theta_7 \Big(s\theta_6 (s\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) - c\theta_1 c\theta_4 s\theta_2) \\
& - c\theta_6 \big(c\theta_5 (c\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3) \big) \Big) \\
& + L_6 c\theta_6 \big(c\theta_5 (c\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3) \big) \\
& - L_7 s\theta_7 \Big(c\theta_6 (s\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) - c\theta_1 c\theta_4 s\theta_2) \\
& + s\theta_6 \big(c\theta_5 (c\theta_4 (s\theta_1 s\theta_3 + c\theta_1 c\theta_2 c\theta_3) + c\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (c\theta_3 s\theta_1 - c\theta_1 c\theta_2 s\theta_3) \big) \Big) + L_4 c\theta_4 (s\theta_1 s\theta_3 \\
& + c\theta_1 c\theta_2 c\theta_3) + L_3 c\theta_1 c\theta_2 c\theta_3 + L_4 c\theta_1 s\theta_2 s\theta_4
\end{aligned}
\tag{3.13}
$$

$$
\begin{aligned}
p_y = {} & L_5 s\theta_5 (c\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3) \\
& + L_7 s\theta_7 \Big(c\theta_6 (s\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) + c\theta_4 s\theta_1 s\theta_2) \\
& + s\theta_6 \big(c\theta_5 (c\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (c\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3) \big) \Big) + L_2 s\theta_2 s\theta_1 - L_3 c\theta_1 s\theta_3 \\
& - L_5 c\theta_5 (c\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4) \\
& + L_6 s\theta_6 (s\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) + c\theta_4 s\theta_1 s\theta_2) \\
& - L_4 c\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) \\
& - L_6 c\theta_6 \big(c\theta_5 (c\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (s\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3) \big) \\
& + L_7 c\theta_7 \Big(s\theta_6 (s\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) + c\theta_4 s\theta_1 s\theta_2) \\
& - c\theta_6 \big(c\theta_5 (c\theta_4 (c\theta_1 s\theta_3 - c\theta_2 c\theta_3 s\theta_1) - s\theta_1 s\theta_2 s\theta_4) \\
& - s\theta_5 (c\theta_1 c\theta_3 + c\theta_2 s\theta_1 s\theta_3) \big) \Big) + L_3 c\theta_2 c\theta_3 s\theta_1 + L_4 s\theta_1 s\theta_2 s\theta_4
\end{aligned}
\tag{3.14}
$$

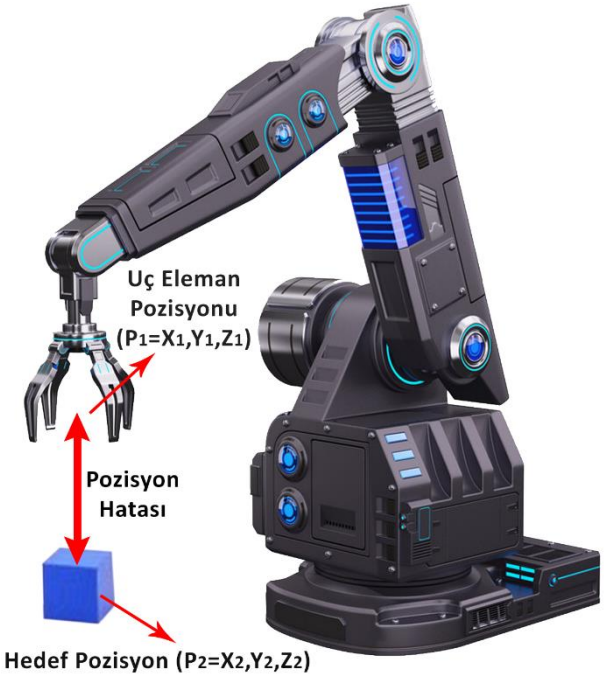
$$
\begin{aligned}
p_z = {} & L_1 - L_2 s\theta_2 + L_6 s\theta_6 (c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4) \\
& - L_7 s\theta_7 \big(s\theta_6 (c\theta_5 (c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5) \\
& - c\theta_6 (c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4 + c\theta_3 s\theta_2 s\theta_4) \big) - L_3 c\theta_3 s\theta_2 \\
& + L_4 c\theta_2 s\theta_4 \\
& + L_6 c\theta_6 (c\theta_5 (c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5) \\
& + L_5 c\theta_5 (c\theta_2 s\theta_4 - c\theta_3 c\theta_4 s\theta_2) + L_7 c\theta_7 (c\theta_6 (c\theta_5 (c\theta_2 s\theta_4 \\
& - c\theta_3 c\theta_4 s\theta_2) - s\theta_2 s\theta_3 s\theta_5) + s\theta_6 (c\theta_2 c\theta_4 + c\theta_3 s\theta_2 s\theta_4)) \\
& - L_4 c\theta_3 c\theta_4 s\theta_2 - L_5 s\theta_2 s\theta_3 s\theta_5
\end{aligned}
\tag{3.15}
$$

Bu çalışmayı gerçeklemek için kullanılan Robotis setiyle beraber gelen ve servoları yönlendirmek için kullanılan kontrol kartı kapalı devre sistemiyle çalıştığından dolayı çalışmada sadece p_x, p_y ve p_z konum denklemleri kullanılmıştır. Bu durum çalışmayı oldukça kısıtlamış gibi görünsede çalışmanın asıl konusu olan zeki optimizasyon yöntemlerinin doğru bir şekilde analizine engel teşkil etmemiştir. Örneğin, uç eleman bir konuma doğru bir şekilde ulaşmasına rağmen oryantasyon parametreleri kullanılmadığından dolayı o noktada bulunan bir nesneyi tutamayabilmektedir. Bu çalışmada asıl amaç zeki optimizasyon yöntemleri kullanılarak uç elemanın istenilen bir konuma en az hata ile ulaşmasını sağlamak olduğundan uç elemanın bir nesneyi tutması göz ardı edilmiştir.

3.2. Pozisyon Hatası ve Uygunluk Fonksiyonu

Bu çalışmanın temelindeki amaçlardan biri, robot manipülatörü uç elemanını, ileri kinematik denklemlerini kullanarak önceden belirlenen noktaya ulaştıracak en uygun eklem açılarını hesaplamaktır.

$$Pozisyon\ Hatası = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{3.16}$$



Şekil 3.18. Pozisyon hatasının resmedilmesi

Pozisyon hatası uç elemanın gerçek konumu ile arzu edilen konumu arasındaki mesafenin ölçülmesi anlamına gelmektedir. Optimizasyon teknikleri ile pozisyon hatasını minimum seviyeye indirecek eklem açılarını bulmak için problem uygunluk fonksiyonu şekline dönüştürülmelidir. Bu tez çalışmasında kullanılan uygunluk fonksiyonu öklit tarafından ifade edilen (Denklem 3.16)'dir.

Şekil 3.18'de P_1 noktası robot manipülatörüne ait uç elamanının gerçek konumudur. Hâlbuki asıl amaç uç elemanı P_2 noktasına yönlendirmekti, dolayısıyla aradaki mesafe pozisyon hatası olarak karşımıza çıkmaktadır.

3.3. Sezgisel Optimizasyon Algoritmaları

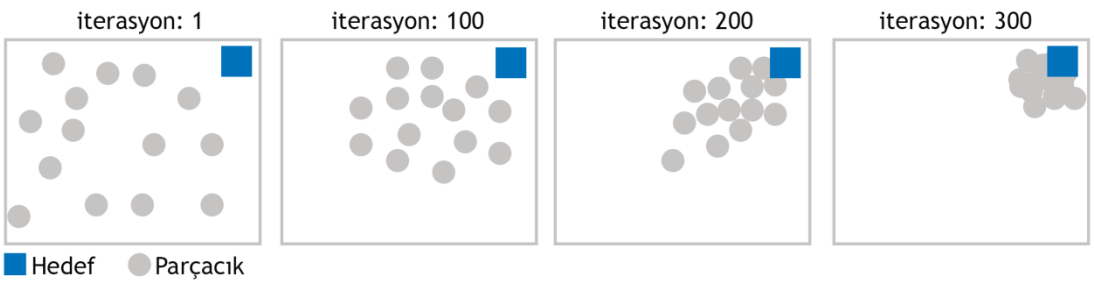
Zamanla problemlerin karmaşıklaşmasına paralel olarak bu problemlerin çözümünde sayısal, geometrik ve iteratif yöntemlerin yetersiz kalmasına sebep olmuştur (Köker, 2013, pp. 528-543). Buna binaen araştırmacılar gerek doğadan gereksede insan vücudunun çalışma sisteminden esinlenerek yapay zekâ optimizasyon tekniklerini veya diğer bir ifadeyle meta sezgisel araştırma tekniklerini kullanagelmişlerdir. Bu teknikler herhangi bir çözümü gerçekleştirmek veya hedefe ulaşmak için çeşitli alternatiflerden en etkili olanlara karar vermek amacı ile kullanılan yöntemlerdir (Blum ve Li, 2008, pp. 43-86). Geleneksel yöntemlerin çözüm uzayı büyük olan problemlerin çözümünde kesin sonuç üretmelerine karşın hesaplama sürelerinin arzu edilenden çok daha uzun sürmesi nedeniyle araştırmacılar yeni tekniklere yönelmişler ve sezgisel teknikleri bilim dünyasına kazandırmışlardır. Her ne kadar bu teknikler kesin çözümü garanti etmesede amaca uygun kabul edilebilir sınırlar dâhilinde çözüme ulaşmayı garanti etmektedirler (Dereli ve Köker, 2019, pp. 3). Başka bir ifadeyle, büyük boyutlu ve çözümü geleneksel yöntemlerle çok uzun zaman alan problemlerin çözümü için, kabul edilebilir sürelerde en iyiye yakın çözümler verebilen algoritmalardır (Çavdar, Mohammed ve Milani, 2012, pp. 329-333). Sürü algoritmalarıda sezgisel tabanlı ve yaygın kullanıma sahip optimizasyon tekniklerindendir. Bu teknikler kullanılarak zor ve karmaşık yapıya sahip problemlerin çözümünde optimum değerler elde edildiğinden araştırmacılar tarafından ilgi odağı haline gelmişlerdir (Karaboğa & Akay, 2009, pp. 61-85; Yang ve He, 2013, pp. 36-50). Bu algoritmalar doğadaki hayvanların sosyal davranışlarından ve hareketlerinden ilham

alarak geliştirilmişlerdir. Örneğin parçacık sürü optimizasyonu balık ve kuş sürülerinin hareketlerini örnek olarak alırken (Kennedy ve Eberhart, 1995, pp. 1942-1948), yapay arı kolonisi algoritması, bal arılarının doğada yiyecek arama davranışlarından (Karaboğa, 2005), karınca kolonisi algoritması ise benzer şekilde karıncalardan (Dorigo, Birattari ve Stutzle, 2007, pp. 28-39) ilham almıştır.

Sürü bireyleri, basit bir yapıya sahip olmalarına karşın bir araya geldiklerinde mükemmel bir işbirliği yeteneğine sahiptirler (Blum ve Li, 2008, pp. 43-86). Zaten sürününde en önemli özelliği bireylerin birbirleriyle üst düzey bir uyum içinde hareket etmeleridir (Karaboğa, 2011, p. 182).

3.3.1. Parçacık sürü optimizasyonu (PSO)

İlk olarak Kennedy ve Eberhart tarafından kullanılan bu teknik kuş ve balık sürülerinden esinlenilerek geliştirilmiş güçlü bir arama algoritmasıdır (Kennedy & Eberhart, 1995, pp. 1942-1948).



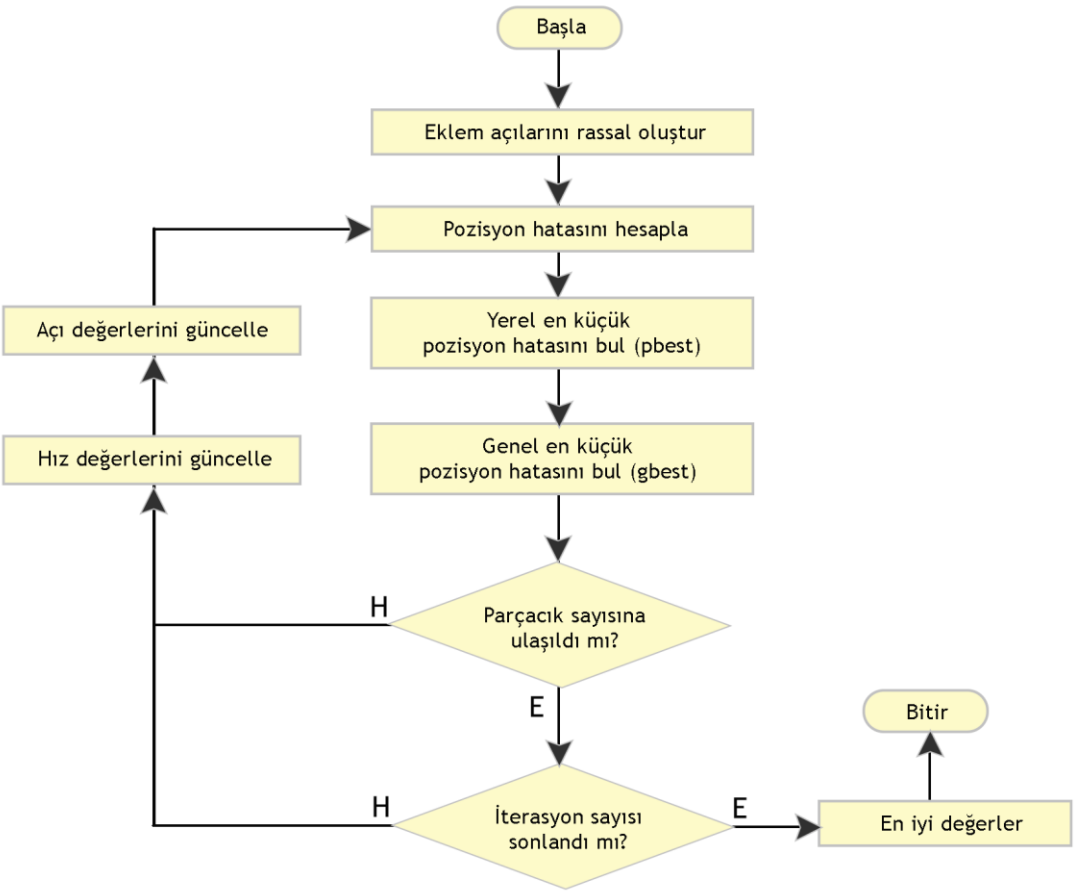
Şekil 3.19. Parçacık sürü algoritması temsili

Diğer sezgisel algoritmalarla karşılaştırıldığında kolay uygulanabilir ve güçlü kontrol parametrelerinin olması bu algoritmanın avantajları arasındadır (Sun et al, 2011, pp. 3763-3775). Günümüzde, popülasyon tabanlı olan bu teknik çok parametreli ve çok değişkenli optimizasyon problemlerine çözümler üretmek için kullanılmaktadır (Alataş, 2007). Parçacıklar optimal çözüme ulaşmak için deneyimlerini ve komşuluklarını birlikte kullanmaktadırlar (Poli, Kennedy ve Blackwell, 2007, pp. 33-57).

Parçacık sürü optimizasyonu sadece iki denkleme sahiptir. Bunlardan biri parçacıkları hedefe taşırken kullanılan (Denklem 3.18) ve diğeri parçacıkların yeni posizyon bilgilerini üreten (Denklem 3.17) 'dır. Bu denklemler her iterasyonda yeniden hesaplanır ve parçacıkların konumu hıza göre iyileştirilerek güncellenir (Bansal et al, 2011, p. 643).

$$v_{id} = v_{id}.\mathrm{w} + c_1.r_1.(p_{best} - x_{id}) + c_2.r_2.(g_{best} - x_{id}) \quad (3.17)$$

$$x_{id} = x_{id} + v_{id} \quad (3.18)$$



Şekil 3.20. Parçacık sürüsü optimizasyon algoritması

(Denklem 3.17) ve (Denklem 3.18)'de kullanılan; d=1, 2, …, 7 eklem açılarını göstermek üzere boyutu; i=1, 2, …, S parçacık sayısını; c1 ve c2, yerel en iyi değerle genel en iyi değerin ağırlıklarını; r1 ve r2 ise [0, 1] arasında rasgele sayıları ifade etmektedir.

Bu tez çalışmasında uygulanan PSO akış diyagramı Şekil 3.20'de verilmiştir. İlk olarak parçacık sayısı kadar 7-adet eklem açısı belli aralıklarda rasgele olarak belirlenmektedir.

Devamında rassal olarak belirlenen eklem açılarından her bir parçacığın pozisyon hatası hesaplanmaktadır. Hesaplanan pozisyon hatalarından en küçüğü bulunduktan sonra diğer iterasyona geçilirken Denklem 3.17 ve 3.18'e göre parçacık hızı ve konumu güncellenmektedir. Böylece parçacıklar hesaplanan hız değerleriyle hedefe ilerlemekte ve en iyi çözümü bulabilmektedir. O nedenle hız değerlerinin etkisini artırmak amacıyla (Denklem 3.17)'da "w" ile gösterilen atalet ağırlığı (Inertia Weight) teknikleri kullanılmaktadır (Özdemir, 2017, pp. 47-52). Parçacık sürü optimizasyonu algoritması ise şu şekildedir:

- Başla
- PSO başlangıç
    - Parçacıklara uygun aralıklarda başlangıç eklem açılarının atanması
- While İterasyon
    - For (her parçacık)
        - Denklem (3.13, 3.14 ve 3.15) ile x, y ve z pozisyon bilgilerini elde et
        - Denklem (3.16) ile önceden belirlenen pozisyon ile bir önceki adımda hesaplanan Pozisyon Hatasını hesapla
        - Posizyon Hatası < Pbest ise Pbest değerini güncelle
    - End For (Parçacık)
    - Pbest < Gbest ise Gbest değerini güncelle
    - For Döngüsü (Her parçacık)
        - Denklem (3.17) ile parçacıkların hızlarını güncelle
        - Denklem (3.18) ile her bir parçacığa ait eklem açılarını güncelle
    - End For (Parçacık)
- End while (İterasyon)
- Gbest değerinin geri döndürülmesi
- Bitir

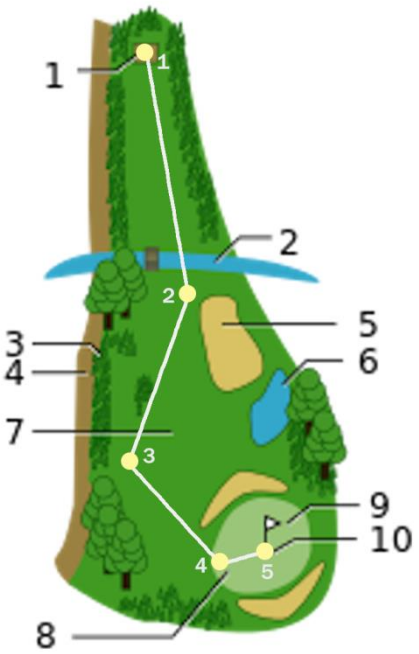
3.3.1.1. RDV (Random Descending Velocity - Rassal Azalan Hızlı) PSO

Parçacık sürü optimizasyonu algoritmasının en önemli avantajı parametrelerinin azlığı olmasına karşın bu durum aslında en iyi çözüme yakınsama konusunda algoritmanın en büyük eksikliği olarak özellikle vurgulanmaktadır. (Marini ve Walczak, 2015, pp. 153-165; Bansal et al, 2011, p. 640). Araştırmacılar bunun için atalet ağırlığı olarak isimlendirilen (Inertia Weight) teknikleri geliştirmişlerdir. Bu tekniklerin asıl amacı parçacık sürü optimizasyonunda parçacıkların ilerlemesini sağlayan hız parametresini kontrol ederek parçacıkların daha iyi bir değere ulaşabilmesi için onlara momentum kazandırmaktır (Adewumi ve Arasomwan, 2016, p. 17). Bu çalışmada ilk defa kullanılan RDV (Rassal Azalan Hız) IW tekniğide bu amaçla kullanılan bir tekniktir ve ilerleyen iterasyonlarda parçacıkların en iyi değeri elde eden parçacığa dönüşmesini esas almaktadır. Bu teknik, tıpkı golf oyununda oyuncunun hedefe yaklaştıkça topun hızını azaltması gibi parçacıkların hızını azaltmaktadır.

Bir golf oyunu oyuncuların golf topunu belirlenen deliklere sokabilmek için başlangıç vuruşunu en kuvvetli, sonraki vuruşları ise bir önceki vuruşa oranla daha yavaş yaptıkları bir oyundur. Golf sahası ne kadar büyük olursa olsun bu durum değişmemekte olup bu oyundaki en önemli esas ise daha az vuruşla topu belirlenen deliğe sokmaktır. Şekil 3.21'de ortalama bir golf parkuru görünmektedir. Bu sahada siyah rakamlar sahanın tanımlamasının yapıldığı, beyaz rakamlar ise örnek vuruşları gösteren rakamlardır. Sahanın tanımlamasında; 1 – başlama vuruşu noktası (teeing ground), 2 – su engeli, 3 – çim olmayan zor yüzey (rough), 4 – sınır dışı, 5 – kum tepesi (bunker), 6 – su engeli, 7 – fairway, 8 – putting green, 9 – pin ve 10 – delik şeklinde ifade edilir. Golf sahasındaki çimler vuruşları zorlaştırmak adına farklı uzunlukta bırıkılmakta olup asıl oyun alanı olan "fairway" de ise çimler daha kısadır ve oyuncular bir an önce toplarını bu bölgeye atmaya çalışmaktadır.

Şekil 3.21'de örnek bir oyun tamamlama işlemi de görünmektedir. Oyuncu önce "1" numaralı vuruşu ki bu vuruş golfte "driving" olarak isimlendirilir. Sonra daha yavaş bir şiddetle "2" numaralı vuruşu gerçekleştirir. "3" numaralı vuruş ile artık top "fairway" alanındadır ve vuruşlar daha kolay olmaktadır. Oyuncu "4" numaralı vuruş ile topu en kaliteli bölge olan ve "8" numara ile gösterilen "pata alanına" göndermiştir. Bu bölgedeki çimler hem çok daha kısa hem kaliteli hemde topların daha iyi yuvarlanması içindir.

Böylece oyuncular bu bölgede çok yumuşak bir vuruşla oyunu kolaylıkla bitirebilmektedirler. İşte golf oyununda ki bu esaslara rassal bazı parametrelerin de katılması ile RDV IW yöntemi ortaya çıkmıştır.



Şekil 3.21. Ortalama bir golf parkuru ve örnek vuruşlar

RDV (Rassal Azalan Hız) tekniğinde parçacıkların yeni bir konuma yönlenmesini sağlayan hız parametresinin değerini rassal olarak belli bir oranda azaltarak nihayetinde parçacıkları en iyi çözüme ulaşan asıl parçacığa benzetmek esasına göre çalışır. Algoritmada bu nedenle kullanılan iki önemli unsur barındırmaktadır: Biri tavlama benzetiminde yerel en iyi değere sıkışıp kalan algoritmayı bu sıkışıklıktan kurtarmak için kullanılan "delta" (Δ) parametresi ve ateş böceği algoritmasında (Denklem 3.27)'de kullanılan ve katsayı parametresi olarak adlandırılan α'dır. RDV tekniği PSO algoritması içerisinde şu şekilde çalışmaktadır:

- Başla (her iterasyon)
    - Başla (her parçacık)
        - Δ değerini hesapla
        - Eğer Δ > rand ise alfa = alfa * alfa_dump
        - w = alfa
        - Parçacık hızını hesapla

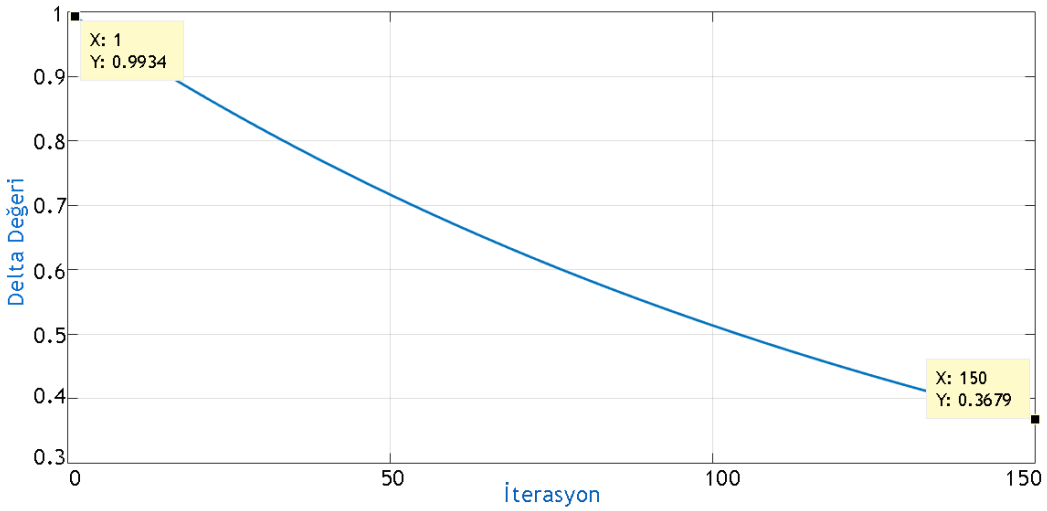
        - Parçacık konumunu güncelle
    - Bitir (parçacık)
- Bitir (iterasyon)

Teknikte “delta” parametresi bulunulan iterasyon ile maksimum iterasyonun bölümünün üssel değeri olup (Denklem 3.19)’de eşitliği verilmiştir.

$$\Delta = e^{\frac{-iterasyon}{Mak_{iterasyon}}} \tag{3.19}$$

Delta parametresi iterasyonun başlangıcından “0.99” değeri ile başlayarak iterasyonun sonuna doğru “0,1” değeri arasında lineer olarak azalma eğilimi göstermektedir.

Algoritmada kullanılan “alfa” [0, 1] arasında rasgele bir sayı olup bu çalışmada “0,3” olarak alınmıştır. “alfa_dump” ise [0,5 - 1] arasında olup bu çalışmada “0,95” olarak alınmıştır. Bu parametre “alfa” nın belli oranda azaltılması için kullanılmıştır.



Şekil 3.22. Delta parametresi değerleri

Tıpkı RDV gibi PSO’da kullanılan çok sayıda IW tekniği mevcuttur. Bazı IW (atalet ağırlığı) teknikleri ve formülleri şu şekildedir:

Rassal IW (Random IW):

$$w = 0{,}5 + \frac{rand}{2} \tag{3.19}$$

Doğrusal Azalan IW (Lineer Decreasing IW):

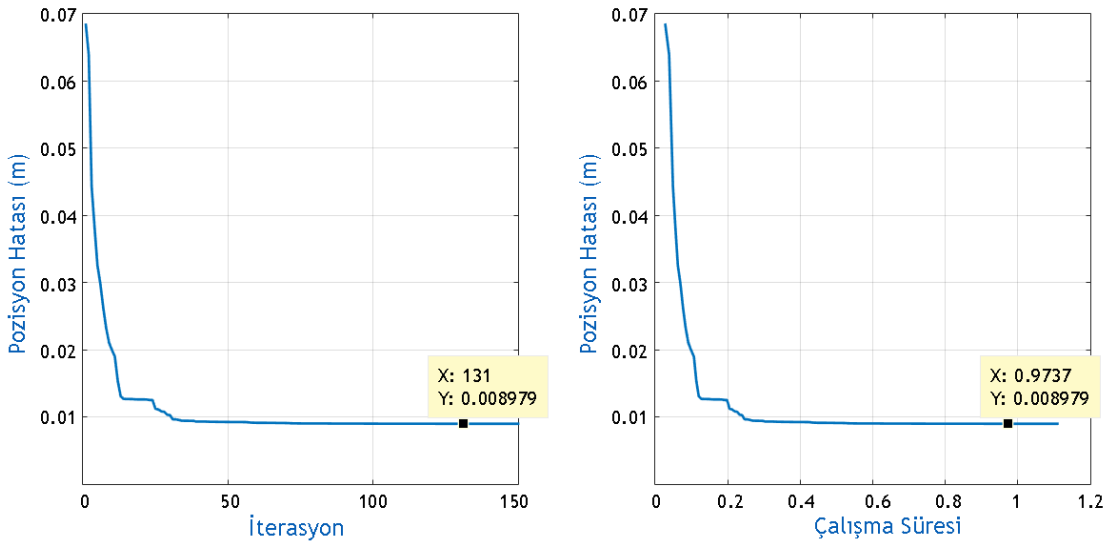
$$w = (0{,}9 - 0{,}4) * (\frac{Mak_{iterasyon} - iterasyon}{Mak_{iterasyon}}) \quad (3.20)$$

Rassal Kaotik IW (Random Kaotik IW):

$$z = 4 * z * (1 - z) \quad (3.21)$$

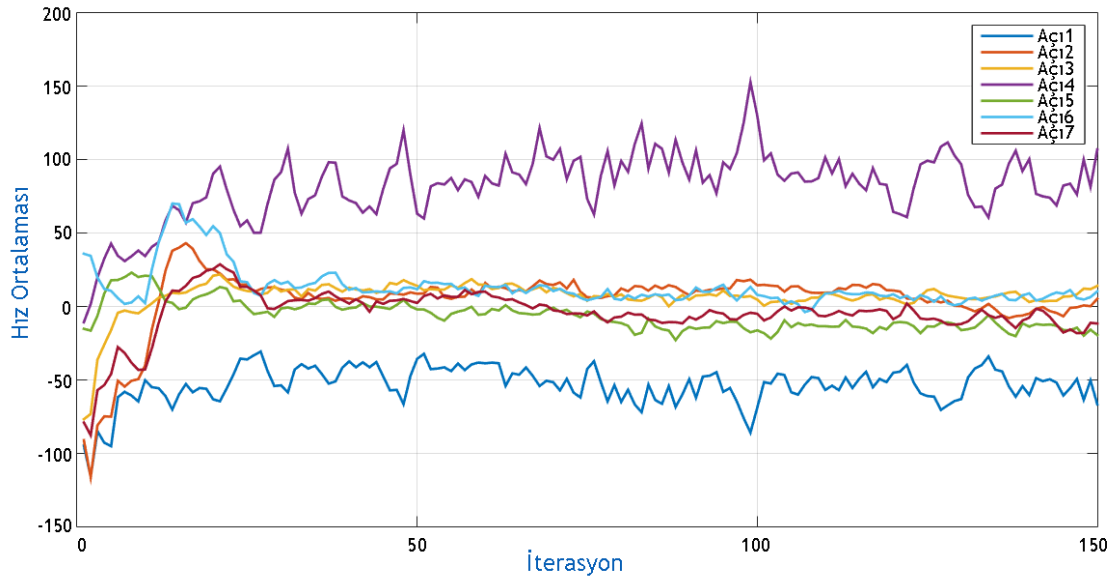
$$w = 0{,}5 * rand + 0{,}5 * z \quad (3.22)$$

Bu tez çalışmasında RDV IW tekniği; yukarıda formülleri verilen Rassal IW, Doğrusal Azalan IW ve Rassal Kaotik IW teknikleriyle karşılaştırılmıştır. Burada önemli olam durum elde edilen pozisyon hatası ile hız ortalamalarının arasındaki ilişkidir. Çünkü hız parametrelerinin değeri senkronize bir şekilde azaldıkça elde edilecek pozisyon hatasında da iyileşme söz konusu olmaktadır.



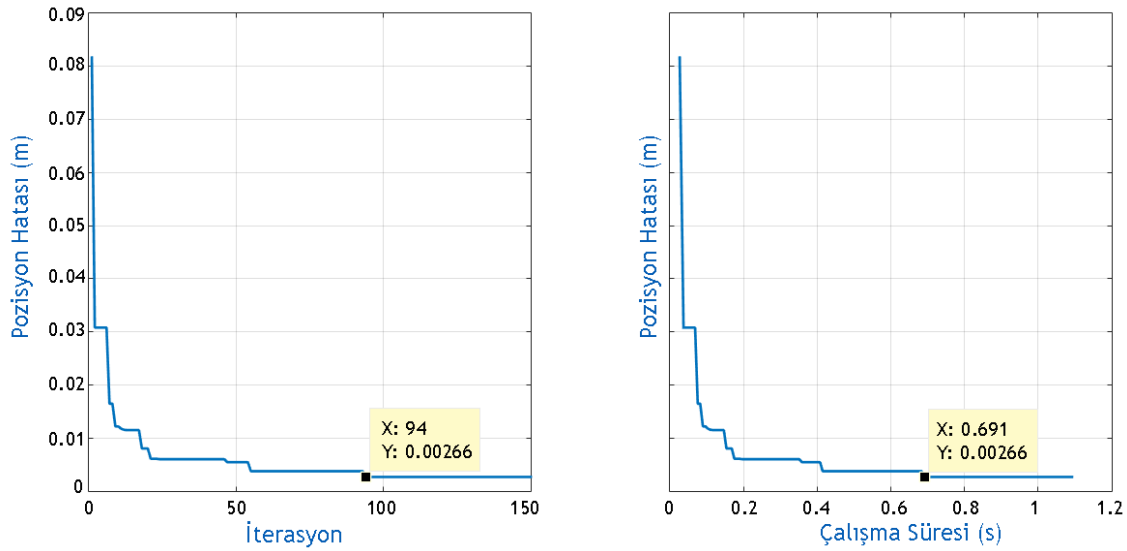
Şekil 3.23. Random IW pozisyon hatası ve hesaplama süresi

Şekil 3.23'de Rassal IW ile elde edilen pozisyon hatası ile hesaplama süresi, Şekil 3.24'de ise açıların hız ortalamaları görünmektedir. "Teta 4" ve "Teta 6" dışındaki açıların hız ortalaması sıfıra yaklaşmış ama sonrasında yatay bir seyir izlemiştir. Bu da pozisyon hatasında daha fazla iyileşme olmasının önüne geçmiştir.

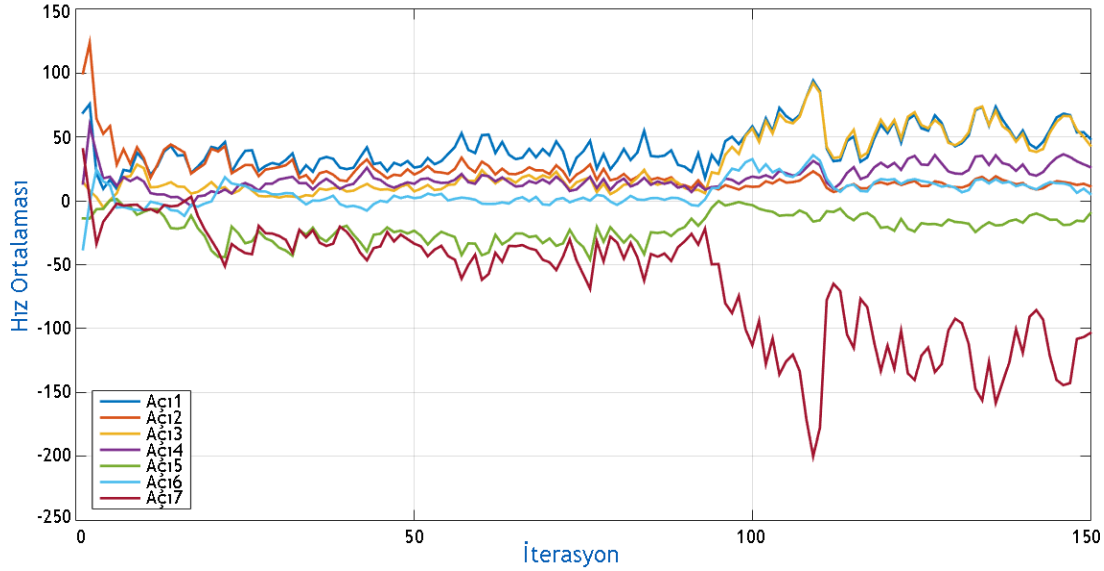


Şekil 3.24. Random IW her bir açının hız ortalaması

Şekil 3.25'de Rassal Kaotik IW ile elde edilen pozisyon hatası ve hesaplama süresi ile Şekil 3.26'de her bir açının ilerlemesini sağlayan hız parametrelerine dair ortalamaları görünmektedir. Grafiklerde 94. İterasyona kadar aslında bir iyileşme söz konusu olmasına rağmen hız parametrelerindeki değişime paralel olarak iyileşmede durmuştur.

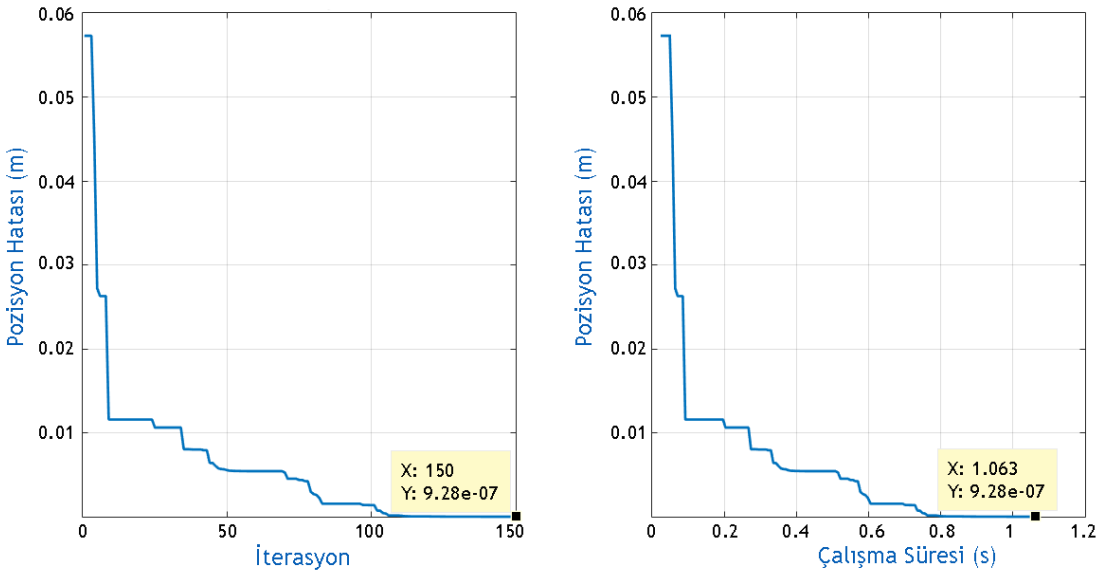


Şekil 3.25. Random Kaotik IW ile elde edilen pozisyon hatası ve hesaplama süresi

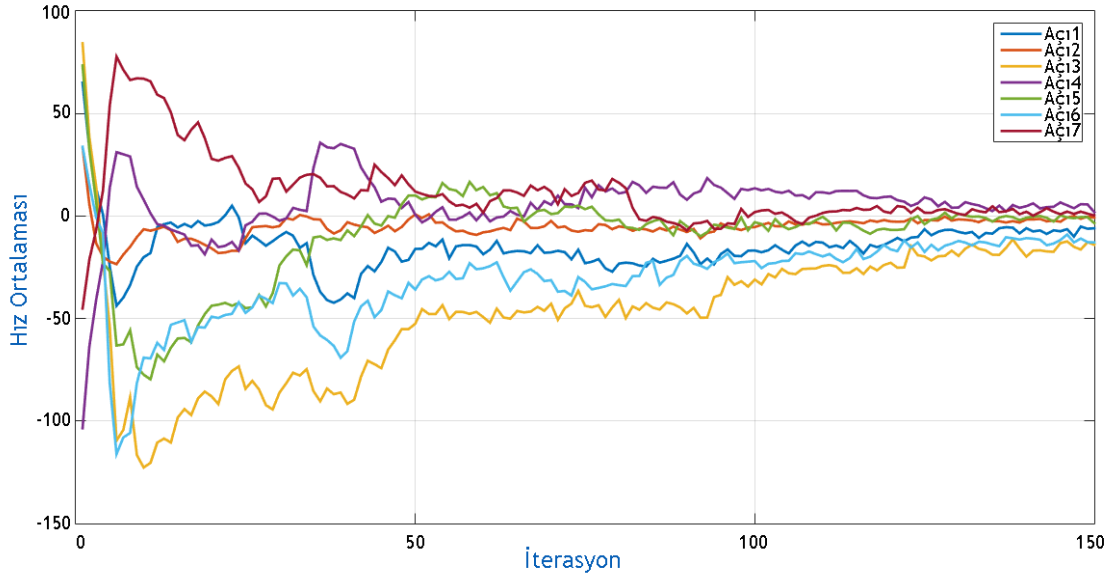


Şekil 3.26. Random Kaotik IW de kullanılan hız ortalamaları

Şekil 3.27'de Doğrusal Azalan IW teknikği ile elde edilen pozisyon hatası ve hsaplama süresi ile Şekil 3.28'da ise açı parçacıklarının çözüme ilerlemesini sağlayan hız ortalamaları görünmektedir. Hız ortalamalarına bakıldığında önceki teknikler arasında arzu edilen seyir burada gerçekleşmiştir. Bu duruma paralel olarakta pozisyon hatası 10^{-7}'li değerlere ulaşmıştır.

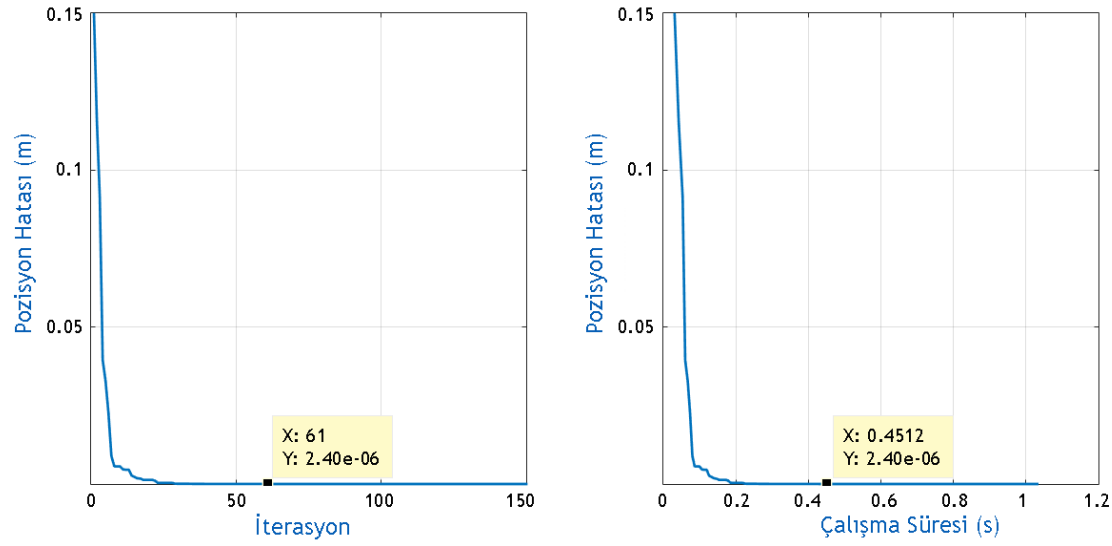


Şekil 3.27. Doğrusal Azalan IW ile elde edilen pozisyon hatası ve hesaplama süresi

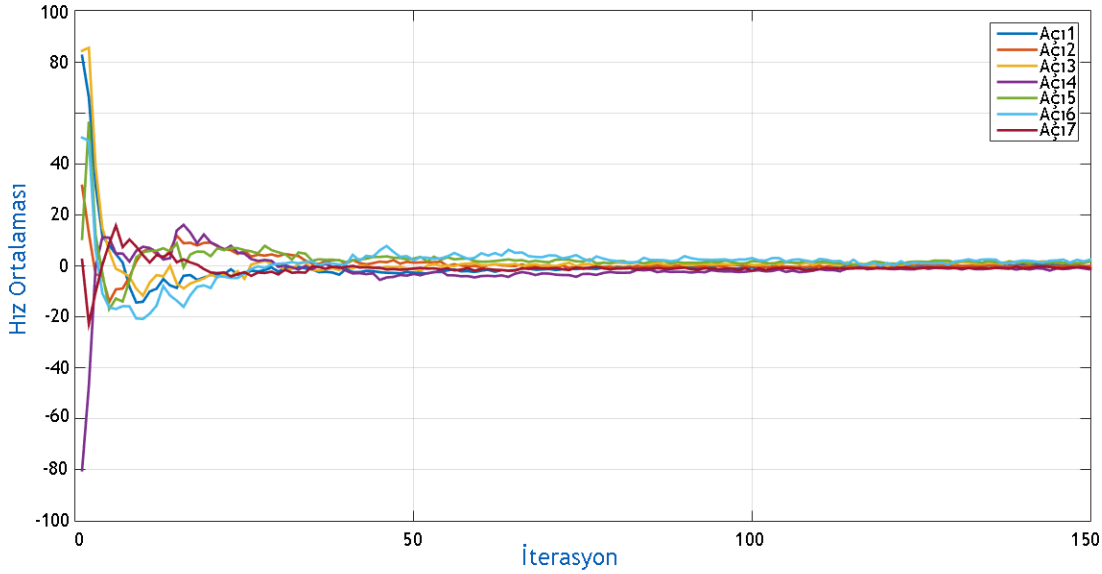


Şekil 3.28. Doğrusal Azalan IW de kullanılan hız ortalamaları

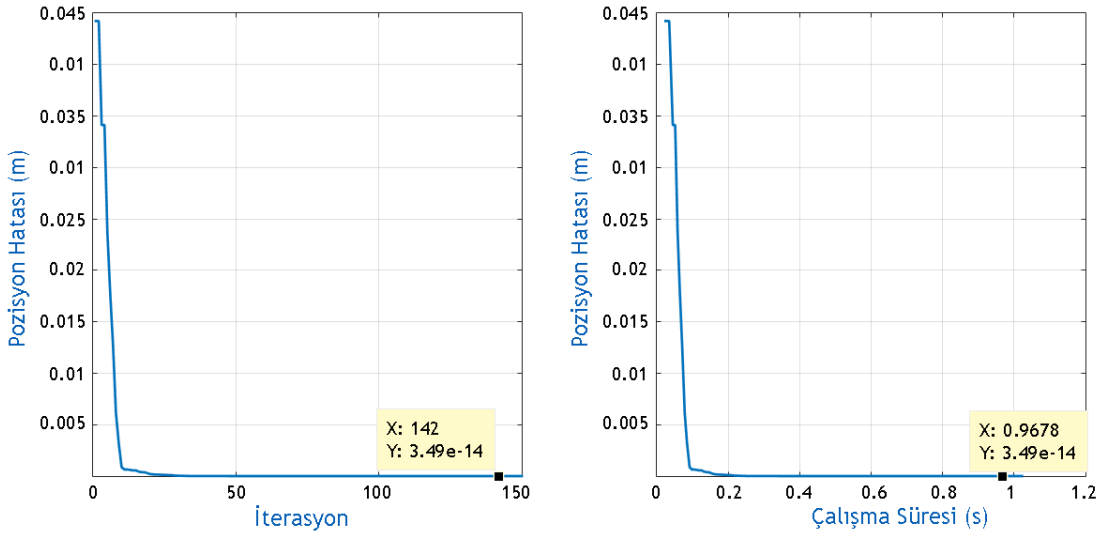
Şekil 3.29 ve 3.31 de RDV tekniği ile elde edilen pozisyon hatası ve hesaplama süreleri ile Şekil 3.30 ve 3.32'de ise açıların en iyi çözüme doğru ilerlemesini sağlayan hız parametrelerindeki değişim görünmektedir. Hız parametrelerindeki değişimler senkronize olarak sıfıra yaklaştıkça çözümde iyileşmektedir. Şekil 3.29 ve 3.30 ilk RDV tekniği denemesini göstermektedir. Pozisyon hatası değeri 10^{-6}'larda hız ortalamaları ise sıfıra yakın olmakla birlikte az da olsa dağınık görünmektedirler.



Şekil 3.29. RDV IW ile elde edilen pozisyon hatası ve hesaplama süresi (Test-I)

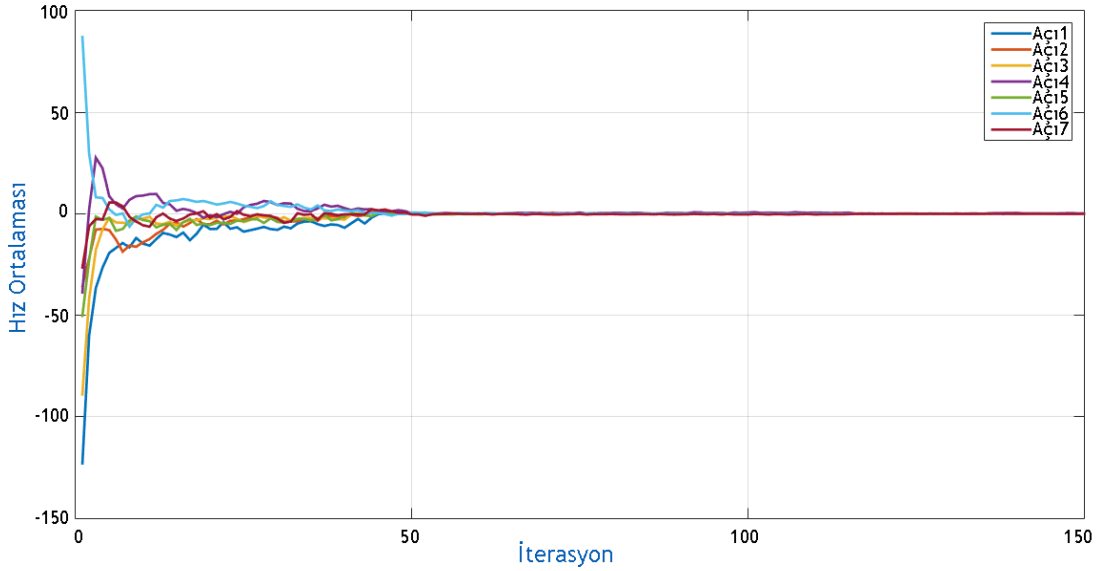


Şekil 3.30. RDV IW de açıları ilerleten hız ortalamaları (Test-I)



Şekil 3.31. RDV IW ile elde edilen pozisyon hatası ve hesaplama süresi (Test II)

Şekil 3.31 ve 3.32'da ise ikinci RDV tekniği örneğidir. Şekil 3.31'de ki pozisyon hatası değeri birinci denemeye göre çok daha iyidir. Benzer şekilde Şekil 3.32'de ki hız ortalamalarıda sıfıra evrilmişler ve sıfır noktasında bir doğru oluşturmuşlardır. Böylece sonuçlarda göstermektedir ki; PSO algoritmasında hız parametresi en iyi çözümün elde edilmesinde önemli bir faktördür. Ortaya çıkarılan IW teknikleride bu sebeple kullanılan önemli tekniklerdir. Bu tez çalışmasında ilk defa kullanılan Rassal Azalan Hızlı (RDV PSO) tekniği ortaya koyduğu sonuçlar açısından kullanıma uygun olan bir tekniktir.



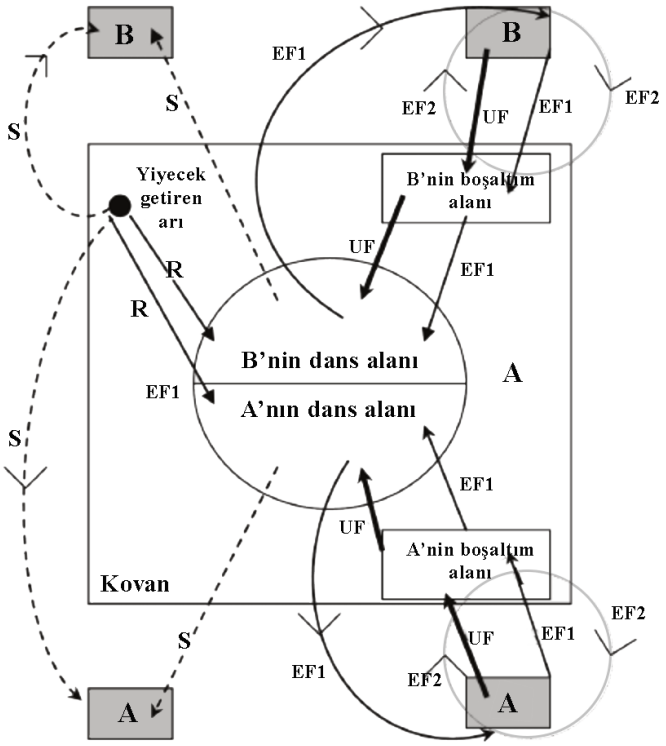
Şekil 3.32. RDV IW de açıları ilerleten hız ortalamaları (Test II)

3.3.2. Yapay arı kolonisi algoritması (YAK)

Bal arılarının doğada kendilerine özgü zekice yiyecek arama davranışlarından esinlenilerek Karaboğa tarafından 2005 yılında geliştirilmiştir (Karaboğa, 2005, pp. 1-10). Sosyal yaşamda var olan her sürü gibi bal arılarınında kendine özgü bir takım davranışları bulunmaktadır Her birey içgüdüsel olarak bu davranışları bilir ve hayatını bu davranışlara göre devam ettirir. Yiyecek kaynaklarının bulunması, bulunan nektarın kovana getirilmesi; bireyler arasındaki dans temelli iletişim ve balın oluşumu gibi sosyal düzenleri arasında her bireyin kendine has görevi vardır (Mohamed, Elarini ve Othman, 2013, p. 402). Bu inanılmaz işleyişten yola çıkan araştırmacılar problem çözümlerinde etkili sonuçlar üreten bir algoritma geliştirebilmişlerdir. Elbette diğer sürülerde olduğu gibi bal arıları sürüsü içinde sosyal düzen, hayatta kalmak ve besin arama ekseninde gerçekleşmektedir (Seker ve Hocaoglu, 2013, p. 128).

Arıların yiyecek arama ve getirme davranışları Şekil 3.33'de gösterilmektedir. Bu şekilde A ve B bulunmuş yiyecek kaynağı olarak farzedilmiştir. Görevi belli olmayan bal arısı kaynak bilgisinden haberi olmaksızın aramaya başladığında bu arı için iki ihtimal söz konusu olacaktır. Bunlardan biri, S ile gösterilen kâşif arı olup yiyecek aramaya başlayabilir. Diğeri ise dans alanı içinde dans etmek suretiyle kaynaklar hakkında bilgi

aktaran arıları izleyerek tarif edilen kaynaklara yönlendirilen ve şekilde R ile gösterilen bir gözcü arı olabilir. Bir şekilde herhangi bir kaynağa ulaşan arı artık görevli arı olmuştur ve kovana nektar getirmeye başlar.



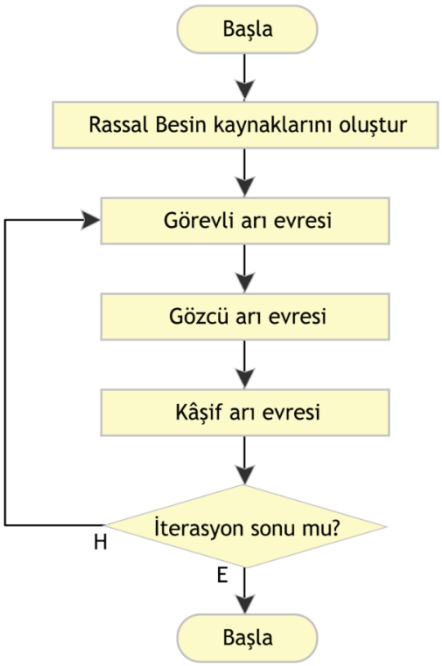
Şekil 3.33. Bal arılarının doğada yiyecek arama davranışları (Karaboğa, 2011, p. 205)

Kovana nektar taşıyan arı için üç ihtimal mevcuttur (Karaboğa ve Akay, 2009, pp. 63-65):

- EF2: Arı bilgi paylaşımında bulunmadan kovana nektar taşımaya devam eder.
- EF1: Kaynağa dönmeden dans alanına gider ve diğer arılara besin kaynağı ve nektar mikarı hakkında bilgilendirir.
- UF: Kaynağı terk ederek dans alanında gözcü arı olup diğer arıların bilgilerini takip eder.

Yapay arı kolonisi tekniğine göre kovanda yaşayan arılar işçi, gözcü ve kâşif olmak üzere üç sınıftır. Koloninin yarısının işçi arı diğer yarısınında gözcü arı olduğu kabul edilmekle birlikte kâşif arı yiyecek kaynağı tükenen görevli arının rasgele yiyecek kaynağı araması durumunda geçici bir dönem olarak geçiş yaptığı arı sınıfıdır (Karaboğa ve Akay, 2009, p. 63). Ayrıca bu teknikte bazı varsayımlar ön plana çıkmaktadır. Bunlardan ilki her bir

nektar kaynağı tek bir görevli arı tarafından ortaya çıkarılmaktadır. Dolayısıyla algoritmada kullanılacak besin sayısı ile görevli arı sayısının birbirine eşit olması gerekmektedir. Diğer bir varsayım ise görevli arı sayısı ile gözcü arı sayısının birbirine eşit olmasıdır.



Şekil 3.34. Arılar arası sınıf değişim evresi

Yapay arı kolonisi algoritması şu şekildedir (Dereli et al, 2019, p. 77):

- Başlangıç besin kaynaklarını oluştur
- Nektar miktarını hesapla (Denklem 3.23)
- While iterasyon
    - GÖREVLİ ARI EVRESİ
    - Görevli arılar besin kaynaklarına komşu besin kaynakları oluşturur (Denklem 3.24)
    - Komşu besin kaynaklarının nektar miktarını hesapla
    - GÖZCÜ ARI EVRESİ (Seleksiyon)
    - While gözcü arıların dağıtılması
        - Sıradaki gözcü arıya besin kaynağı seçtir
        - Gözcü arının seçtiği besin kaynağına komşu besin kaynağı oluştur
        - Komşu besin kaynağının nektar miktarını hesapla

    - End while
    - En iyi besin kaynağının pozisyon bilgisini kaydet
    - Nektarı tükenen besin kaynaklarından ayrıl (Denklem 3.27)
    - KÂŞİF ARI EVRESİ
    - Bırakılan besin kaynakları yerine rasgele besin kaynakları üret
- End while iterasyon
- En iyi pozisyon kaynağı bilgilerini döndür

Bir besinin kalitesi yapay arı kolonisi algoritmasında ne kadar yüksekse o kaynağın uygunluk değeride o derece yüksektir. Bu noktada algoritmayı kullanan kişinin amacı ister maksimum veya isterse minimum değeri bulmak olsun nektarın kalitesi çözümün uygunluk değerine denk gelmektedir (Karaboğa ve Baştürk, 2007, p. 462).

Besin kaynakları arama yapılan çözüm uzayında yer alacaktır. Dolayısıyla algoritma başlarken işçi arılar tıpkı sabahları kovandan ayrılır gibi rasgele olarak yiyecek kaynağı aramaya başlarlar. Bu çalışmada gerçekleştirilecek çözüm için düşünülecek olursa; her bir eklem açısına kendi değer aralıklarında rasgele bir değer atanarak algoritma başlamaktadır. Bu işlem (Denklem 3.23) ile gerçekleştirilmektedir.

$$x_{i,j} = x_j^{min} + rand(0,1)(x_j^{max} - x_j^{min}) \quad (3.23)$$

Denklemlerde kullanılan; j=1, 2…7 eklem açısı olmak üzere boyutu; i=1, 2…SN nektar sayısını; x_j^{min} ve x_j^{max} ise önceden belirlenmiş olan alt değer ile üst değer arasındaki değerlerden oluşan besin kaynaklarının üretilmesi sağlanmış olmaktadır.

Arama uzayında çözüm değerleri araştırılırken işçi arılar besin kaynaklarından rasgele olarak bir tanesini belirlerler ve bu besin kaynağının kalitesini yani çözüm değerini hesaplarlar. Elde edilen çözüm değeri hafızaya alınır. Daha sonra görevli arılar (Denklem 3.24)'ü kullanarak rasgele gittikleri besin kaynakları yakınlarında daha fazla nektarın bulunduğu yeni besin kaynakları ararlar. (Önder, Özdemir ve Yıldırım, 2013, pp. 65-66).

$$v_{i,j} = x_{i,j} + \varphi_{i,j}(x_{i,j} - x_{k,j}) \quad (3.24)$$

$x_{i,j}$, mevcut besin kaynağını, $v_{i,j}$ komşu besin kaynağını, $\varphi_{i,j}$ [-1,1] arasında rasgele bir değeri göstermektedir ve son olarak SN nektar sayısı olmak üzere k ∈ [1, SN] 'dir. Elde edilen yeni komşu besin kaynağı, önceki besin kaynağından daha iyi bir değere sahipse bu durumda önceki besin kaynağı ile ilgili bilgiler yerine yeni kaynak ile ilgili bilgiler hafızada tutulmaya başlar. Elbette görevli arının bulduğu yeni komşuluk değeri, problemin amacına uygun olan sınır değerlerini aşmamalıdır. Bunu sağlamak için (Denklem 3.25) kullanılmaktadır.

$$x_{i,j} = \begin{cases} x_{min}^{i}, & x_i < x_{min}^{i} \\ x_{max}^{i}, & x_i > x_{max}^{i} \end{cases} \tag{3.25}$$

Görevli arılar komşu kaynaktan topladığı besin miktarı ile ilgili bilgileri kovana dönerek gözcü arılara aktarmakta ve gözcü arılarda nektar miktarı fazla olan besin kaynaklarına yönelmektedirler. Bir besin kaynağının diğer besin kaynaklarından daha iyi olmasının göstergesi uygunluk fonksiyonu (Denklem 3.26) neticesinde elde ettiği değerdir. Algoritmada gözcü arıların nektar miktarı en fazla olan besin kaynaklarına gönderilmesi için (Denklem 3.26) ile her bir besin kaynağının seçilme olasılığı değeri oluşturulmaktadır. Bu denklemde P_i olasılık değeridir ve bu değerin yüksek olması en fazla besin miktarına sahip olduğunu göstermekte olduğundan gözcü arılar için seçilme ihtimali daha fazladır (Mohamed, Elarini ve Othman, 2013, pp. 402-403).

$$P_i = fitnes_i / \sum_{j=1}^{SN} fitnes_i \tag{3.26}$$

Bu denklemde P_i, gözcü arıların i.yiyecek kaynağını seçme ihtimali; $fitnes_i$ ise besin kaynağının uygunluk değeri; SN ise toplam yiyecek kaynağıdır.

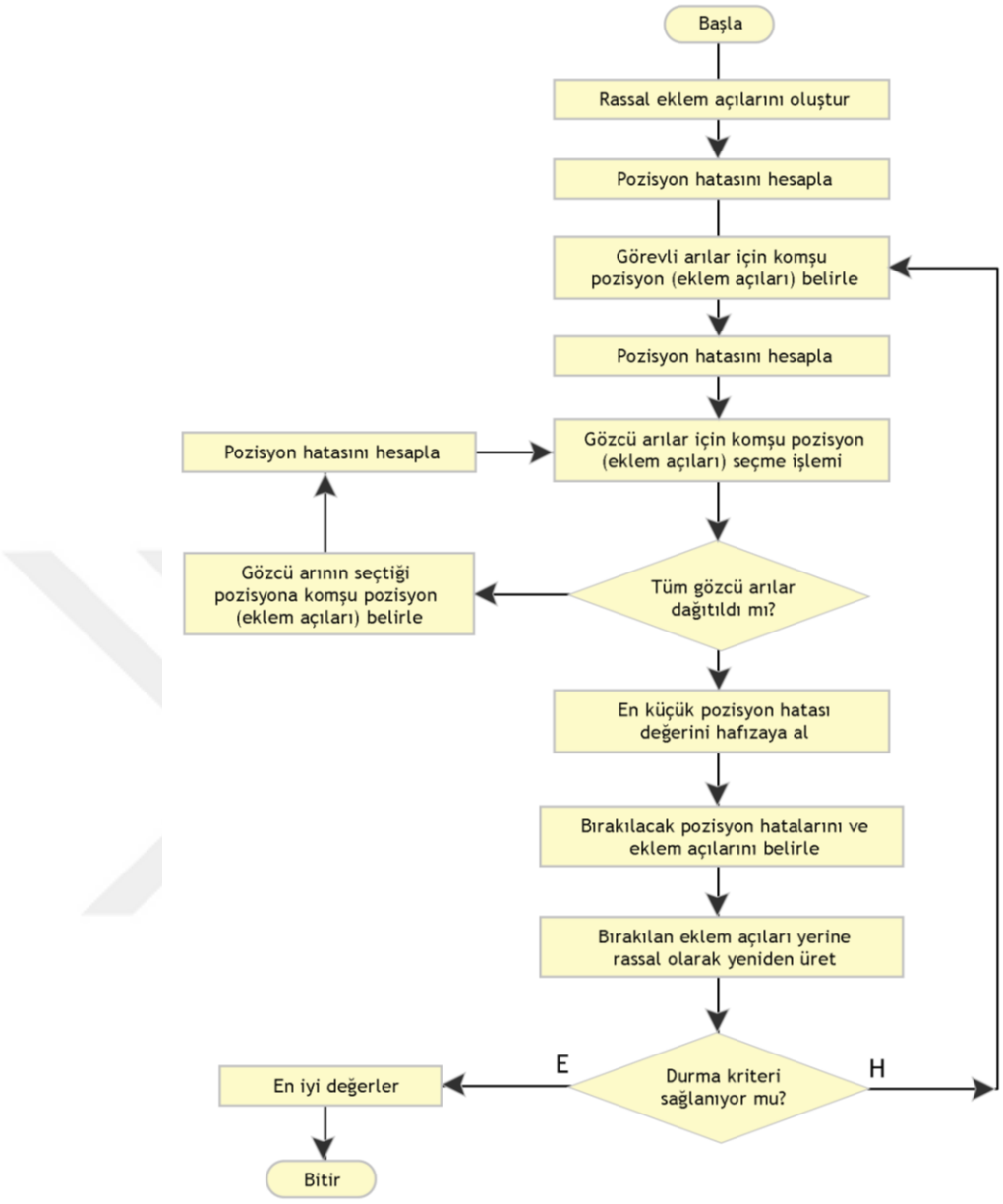
Algoritmanın buraya kadar olan evresi görevli arı evresiydi, bundan sonra ise ikinci evre olan gözcü arı evresidir. Bu evrede gözcü arılar P_i olasılık değerlerini dikkate alarak besin kaynaklarını seçerler. Olasılık değerlerine bakıldığında ilgili kaynağın uygunluk değeri ile o kaynağın gözcü arılar tarafından seçilmesi arasında doğru orantı mevcuttur (Önder, Özdemir ve Yıldırım, 2013, pp. 67). Bu algoritmada gözcü arıların seçimi rulet tekerleğine göre yapılmaktadır ve algoritması şu şekildedir:

- Rasgele sayı (0,1)
- Toplam = 0
- i = 0
- repeat
    - i = i + 1
    - Toplam = Toplam + Pi
- Until (Toplam >= Rasgele sayı)
- Return i

Gözcü arı rulet tekerleği yöntemine göre seçimini yaptıktan sonra seçtiği yiyecek kaynağına (Denklem 3.24)'ü kullanarak yeni bir komşu besin kaynağı oluşturacaktır. Üretilen besin kaynağının uygunluk değeri seçilen besin kaynağının uygunluk değerinden daha iyiyse yeni besin kaynağı üretilen komşu besin kaynağı olacaktır. Aksi halde gözcü arı mevcut seçmiş olduğu besin kaynağı bilgisini koruyacaktır. Bu işlem gözcü arıların besin miktarı çok olan kaynaklara yönelmesine dolayısıylada uygunluk fonksiyonunun daha iyi değerler bulmasına yardımcı olmaktadır (Makas, 2015, p. 15).

Eğer komşuluk üretimi ile çözümler belli bir aşamaya gelmiş ve daha fazla iyileşmiyorsa bu durumda besin kaynağı için tanımlanan limit değeri devreye girer ve gözcü arı devresi bitmiş kâşif arı devresi başlamış demektir. Yani gözcü arı varsayılan değeri terkederek kâşif arı olmuş ve (Denklem 3.23)'ü kullanarak rasgele yeni bir besin kaynağına yönelmiş demektir. Böylece algoritma bu davranışıyla aslında ümit vaat etmeyen problem çözümlerinin terkedilmesini sağlamış olmaktadır (Karaboğa, 2011, p. 210). Bunun için bu çalışmada kullanılan limit değeri formülü şu şekilde olup bu denklemde SN besin kaynağı sayısını, $Dimension$ ise problem boyutunu temsil etmektedir:

$$Limit = (SN * Dimension)/2 \tag{3.27}$$



Şekil 3.35. Yapay arı kolonisi algoritması akış şeması

Algoritmanın bu çalışmada kullanılan akış şeması Şekil 3.35'te verilmiştir. Bu çalışmada kullanılan diğer algoritmalarda olduğu gibi başlangıç değerleri rassal olarak verilmiştir. Yani ters kinematik çözümü elde edecek 7-adet eklem açısı belirlenen popülasyon sayısı kadar rasgele verilmiştir. Aslında bu algoritmada başlangıçta rassal olarak elde edilen değerler kaşif arıların bal arısı besini olan nektarı elde edecekleri çiçekleri rasgele arama işlemidir. Hemen akabinde bu eklem açıları neticesinde oluşan pozisyon hatası elde edilmiştir yani kaşif arılar artık yerini nektarı kovana taşıyacak olan görevli arılara bırakmıştır. Görevli arılar elde ettikleri besin miktarlarını kovana getirerek elde ettikleri

bu pozisyon hatası bilgilerini (nektar) boşaltım alanlarında dans ederek gözcü arılarla bilgi paylaşımında bulunurlar. Gözcü arılarda paylaşılan nektar (pozisyon hatası) miktarlarının bulunduğu konumlardan istediklerini seçerek bu konumlara dağılırlar. Devamında tekrar pozisyon hatası hesaplaması yaparak önceki pozisyon hatası değerini minimuma indirmeye çalışırlar. Eğer gözcü arının seçtiği konumdaki pozisyon hatası değeri iyileşmiyorsa gözcü arı artık o konumu terkederek tekrar kaşif arı olur ve rasgele yeni bir konuma yönlenir. Algoritma bu şekilde tekrar ederek belli bir iterastonda veya belli bir pozisyon hatası değerinde sonlanır.

3.3.3. Ateş böceği algoritması (ABA)

Tropikal iklim bölgelerinde yaşayan ateş böceklerinin avlarını çekmek, eşleriyle çiftleşmek için onları etkilemek ve diğer yırtıcı hayvanlardan korunmak amacıyla vücut yüzeylerinde bulunan fotojenik hücreler sayesinde (Babu & Kannan, 2002, p. 51) yaydıkları ritmik ateşlerden esinlenilerek 2008 yılında Xin-She Yang tarafından geliştirilmiştir (Yang, 2010, p. 81). Ayrıca ürettiği kimyasal esansla diğer yırtıcı hayvanlara karşı tatlarının acı olduğu hissini verir. Günümüzde dünya üzerinde 2000'den fazla ateş böceği türü mevcuttur ve genellikle yaz gecelerinde çok aktiftirler (Lewis ve Cratsley, 2008, p. 294). Sezgisel arama gücüne sahip algoritmalardan biri olduğundan dolayı günümüzde zor, karmaşık ve hesaplaması uzun zaman alan problemlerin çözümünde etkin sonuçlar üretmesi nedeniyle pek çok alanda yaygın olarak kullanılmaktadır (Fister et al, 2013, p. 37).

İncelemeler neticesinde ateş böceklerinin ışık yayma şiddeti düşük olanların daha parlak olanlara doğru hareket ettiği ortaya konmuştur. Bu nedenle bu en iyileme tekniğinin temelinde parlaklık ve ışık şiddeti kavramları yatmaktadır (Boz ve Çimen, 2017, p. 125). Bu nedenle algoritmanın uygulanabilmesi için şu kurallar kabul görmektedir (Aydilek, 2017, p. 1099):

- Ateş böceklerinin cinsiyeti yoktur, böylece cinsiyeti ne olursa olsun kendine diğer ateş böceklerini çekebilir.

- Çekicilik ateş böceğinin parlaklığı ile ilgilidir. Ancak bir ateş böceğini, ona yakın olan ateş böceği daha parlak bulurken uzaktaki diğer bir ateş böceği ise daha az parlak görmektedir. Dolayısıyla parlaklık aradaki uzaklığa bağlı olarak değişir.
- Parlaklık uygunluk (amaç) fonksiyonu ile belirlenir. En parlak olan ateş böceği aslında problemin çözümünde en iyi değere sahip ateş böceğidir. Bu durum hem minimum hemde maksimum değerleri elde etme husunda da geçerlidir.

Ateş böceği sürüsünde en uygun çözümler elde etmek için verilen problemin uygunluk (amaç) fonksiyonuna dönüştürülmesi ve bu fonksiyonunda ateş böceği sürüsü içinde daha parlak ve çekici olan yerlere gitmede yardımcı olan yanıp sönen ışık veya ışık şiddeti ile ilişkili olmalıdır. Bu algoritmada sürü içerisinde yer alan ateş böcekleri tek cins olarak kabul edilmektedir. Dolayısıyla bir ateş böceği ne kadar parlak olursa diğer ateş böceklerini o oranda kendine çekecektir (Yang, 2009).

$$x_{i,j} = x_j^{min} + rand(0,1)(x_j^{max} - x_j^{min}) \tag{3.28}$$

Ateş böceği algoritmasının bu tez çalışmasında kullanılan algoritması Şekil 3.36'da görünmektedir. Diğer sezgisel tekniklerde olduğu gibi algoritmada kullanılan parametreler başlangıç konumuna alınırken eklem açılarıda belli sınır değerlerinde rasgele olarak başlatılır. Bunun için (Denklem 3.28) kullanılır.

Amaç fonksiyonuna göre "m" tane ateş böceği kullanılmakta ve her bir "i" ateş böceğinin "x_i" çözümünün uygunluk değeri hesaplanmaktadır. Elde edilen uygunluk değerinin gerçek çözüme olan uzaklığı o çözümün kalitesini ortaya koymaktadır. Amaç fonksiyonuna uygulanan her bir ateş böceğinin çözümüm o ateş böceğinin parlaklığını ve ışık yoğunluğunu ifade etmektedir (Pamuk, 2016, p. 45). Işık yoğunluğu formülü şu şekildedir:

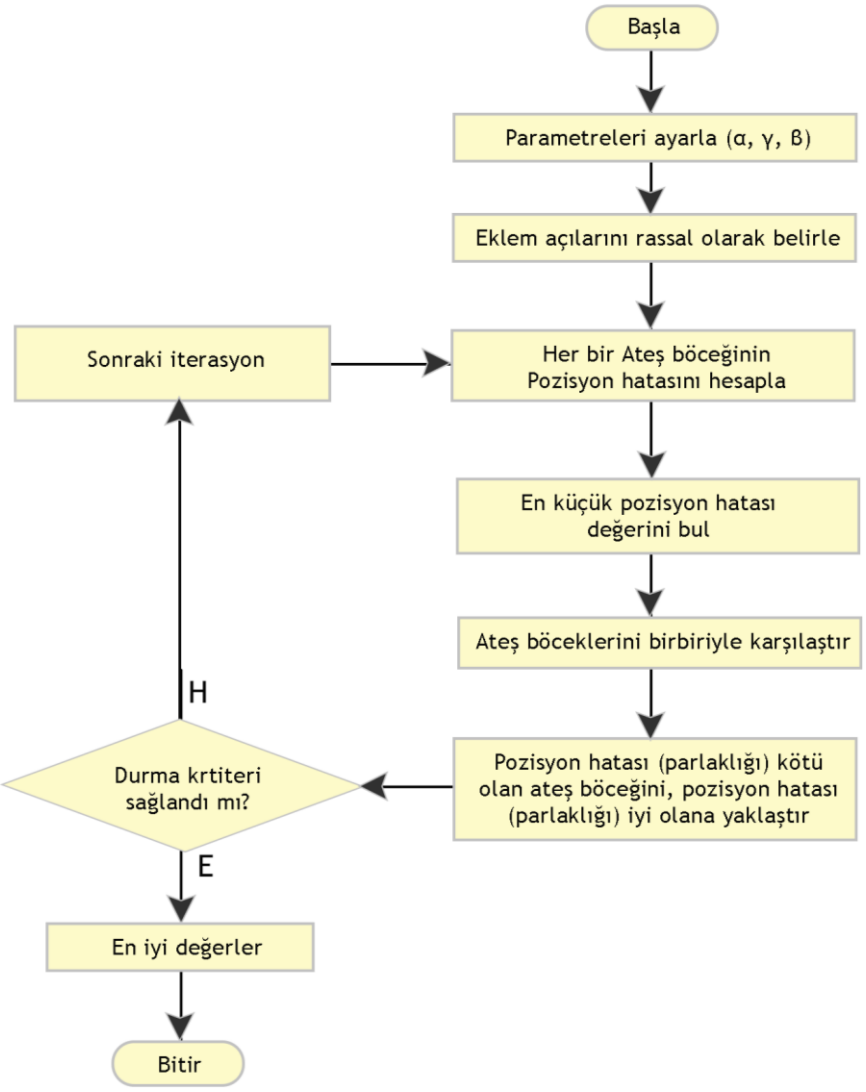
$$I = I_0 e^{-\gamma r^2} \tag{3.29}$$

I: Işık yoğunluğu,

I_0: Başlangıç ışık yoğunluğu,

γ: Işık soğurma katsayısı (sabit emilim katsayısı),

r: İki ateş böceği arasındaki uzaklık.



Şekil 3.36. Ateş böceği algoritması akış şeması

Bir ateş böceğinin çekiciliği aradaki mesafeye bağlı olarak değişmektedir ve eşitlik (Denklem 3.30) 'da gösterilmektedir.

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_i - x_j)^2} \tag{3.30}$$

x_i: i. ateş böceğinin uygunluk değeri,

x_j: j. ateş böceğinin uygunluk değeri,

d: problem boyutu.

$$\beta = \beta_0 e^{-\gamma r^2} \tag{3.31}$$

β: Ateş böceğinin çekiciliği,
β_0: İki ateş böceği arasındaki uzaklık (r) "0" ise çekicilik katsayısı ($\beta \in [0,1]$).

β ifadesindeki değere bağlı olarak az çekici olan i. ateş böceği kendisinden çok daha çekici olan j. ateş böceğine doğru hareket eder ve denklem 3.32 ile gösterilmektedir.

$$x_i = x_i + \beta_0 e^{-\gamma r^2}\left(x_j - x_i\right) + \alpha\varepsilon_i \tag{3.32}$$

α: Rastlantı değişkeni ($\alpha \in [0,1]$) ,
ε_i: Gauss dağılımı.
x_j: Daha parlak olan ateş böceği,
x_i: Daha mat olan ateş böceği.

Bu tez çalışmasında ters kinematik problemin çözümünde kullanılan algoritma Şekil 3.36'da görünmektedir. Burada problemin çözüm değeri en parlak ateş böceğini simgelemekte ve çözüm için ateş böcekleri bu çözüme doğru ilerlemektedir. Çözüm değerine en yakın noktada duracak olan ateş böceği aslında bizim gerçek çözümümüz olmaktadır. Her sürü algoritmasında olduğu gibi bu algoritmada da ateş böceklerinin başlangıç konumu rasgele olarak belirlenmektedir. Devamında ise sürüdeki her bir ateş böceğinin parlaklığı yani bu çalışmadaki pozisyon hatası hesaplandıktan sonar her bir etaş böceğinin pozisyon hatası değeri (parlaklığı) birbiriyle karşaılaştırılmıştır. Parlaklığı az olan yani pozisyon hatası değeri kötü olan ateş böcekleri sonraki aşamada ise parlaklığı daha iyi olan çözüme doğru ilerlemektedir. Algoritmanın durması belli bir iterasyon değerinden sonra gerçekleşmektedir. Ateş böceği algoritması şu şekildedir:

- Uygunluk fonksiyonu f(x), $x=[x_1,x_2, \dots , x_n]$
- Başlangıç popülasyonunu oluştur (Denklem 3.28)
- Uygunluk fonksiyonunu hesapla
- Çekicilik, emilim ve rastlantı parametreleri belirlenir (β, γ, α)
- While (iterasyon)

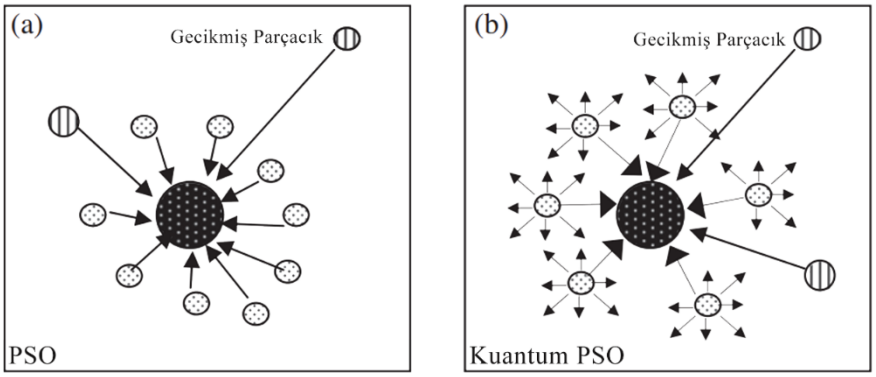
- For i=1:m (m: popülasyon sayısı)
- For j=1:m
  - if $f(x_i)<f(x_j)$
  - i. Ateş böceği j. Ateş böceğine hareket eder (Denklem 3.32)
  - end if
  - Uzaklık belirlenir (Denklem 3.30)
  - Çekicilik çarpanı belirlenir (Denklem 3.31)
  - Ateş böceği konumu güncellenir
- End for j
- End for I
- Ateş böcekleri sıralanır ve en parlak olan bulunur

- End While (iterasyon)
- En iyi değeri döndür

3.3.4. Kuantum Parçacık Sürü Optimizasyonu (KPSO)

Kuş ve balık sürülerinin hareketlerinden yola çıkılarak geliştirilen parçacık sürü optimizasyon tekniği pek çok lineer olmayan çok zor problemin çözümünde etkili sonuçlar üretmesine (Zhang, Wang ve Ji, 2015, p. 2) rağmen, çok az parametreye sahip olması nedeniyle bu teknik global en iyi değerlerden ziyade yerel en iyi değerlere takılarak daha iyi sonuçlar çıkmasını engelleyebilmektedir (Bai, 2010, p. 181). Kuantum yönteminde ise global yakınsama yapabilmek adına sürü içerisindeki zayıf bireylere de fırsat verilmektedir. Bunun için bu yöntemde hız ve konum vektörleri yerine dalga teorisi fonksiyonu kullanılarak parçacıklar dinamik hale getirilir (Pant, Thangaraj, & Abraham, 2008, p. 2). Diğer bir ifadeyle; PSO algoritmasında gecikmiş olan parçacıklar (Şekil 3.37) çözüme minimum düzeyde katkı yaparken KPSO algoritmasında çok daha fazla katkıda bulunmaktadırlar (Sun et al, 2011, p. 3766).

Kuantum PSO algoritması olasılık yoğunluğu fonksiyonununa ait önceden belirlenen bir kuantum uzayında parçacık tanımlaması yapar. Artık parçacıkların durumu hız ve pozisyon ile ifade edilmek yerine normal duruma dönüşmüş kuantum durumu olarak ifade edilir (Tan et al., 2015).



Şekil 3.37. PSO ve QPSO algoritmasında parçacıkların hareketi (Sun et al, 2011, p. 3766)

$$x_{id}(t+1) = \{g_{id} \pm \alpha | mbest_d - x_{id}(t)| \log\left(\frac{1}{u}\right)\} \quad (3.33)$$

$$g_{id} = \varphi . pbest_{id} + (1-\varphi) gbest_d \quad (3.34)$$

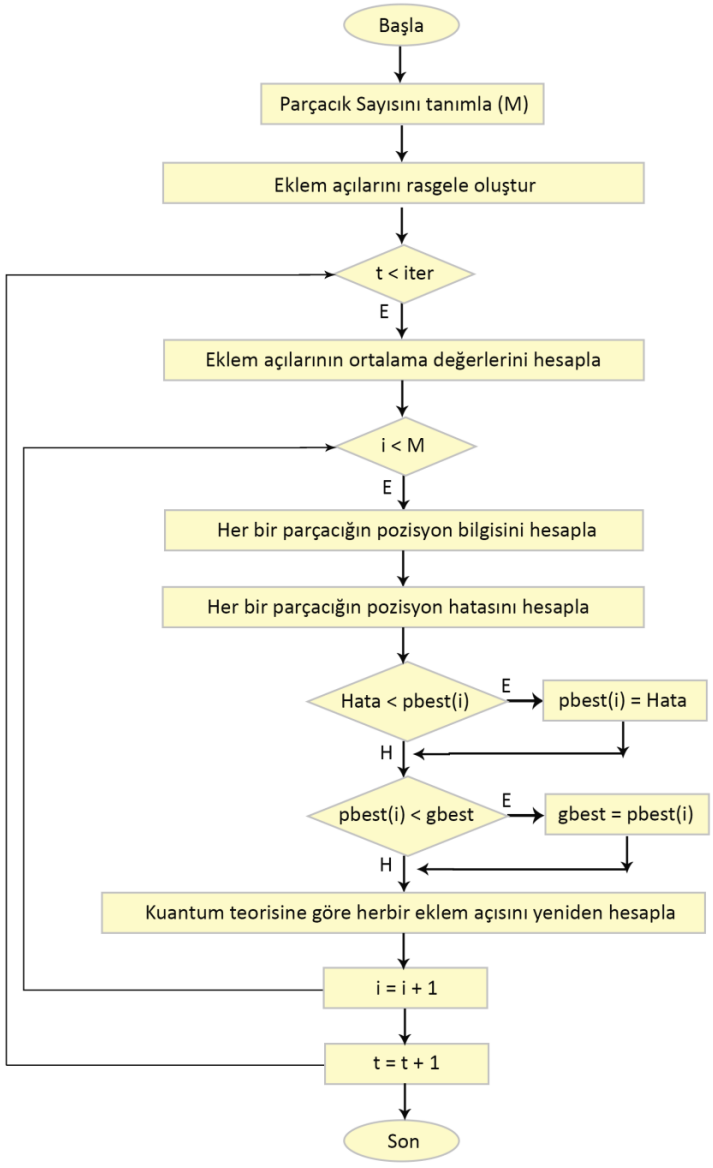
$$mbest_d = \sum_{i=1}^{M} pbest_{id} / M \quad (3.35)$$

$$\alpha = (\alpha_1 - \alpha_0)(iter_{max} - iter)/(iter + \alpha_0) \quad (3.36)$$

QPSO'da parçacıkların hareketi (Denklem 3.33, 3.34, 3.35 ve 3.36) ile gerçekleşmektedir. "M" parçacık sayısını, "mbest" tüm parçacıkların en pozisyonlarının ortalamasını, "gbest" genel en iyi değeri ve "pbest" ise yerel en iyi değeri ifade etmektedir. [α_0=0,5] ve [α_1=1,0] ve [u=rand(0,1)] olarak tanımlanmıştır. (Denklem 3.33)'te random sayı [0,5]'ten büyükse (+) değilse (-) işlem yapılmaktadır.

Bu tez çalışmasında kullanılan kuantum parçacık sürü optimizasyonu algoritması Şekil 3.38'de görünmektedir. Algoritmanın hemen başında parçacık konumları rasgele belirlenmektedir. Devamında tüm parçacıkların her bir eklem açısının ortalama değeri hesaplanmaktadır. İşte kuantum parçacıklarının hareketi bu ortalama değere göre yapılmaktadır. Ancak PSO algoritmasından asıl fark şu ki: Parçacıkların konumu önce konumuna göre iyileşebileceği gib yani hedef konuma yaklaşabileceği gibi hedef noktadan uzaklaşarak daha kötü bir konuma yerleşebilir. Bu sayede gecikmiş olan yani

geride kalan parçacıklarda çözüm ihtimali olan parçacık durumuna terfi etmiş olmaktadır. Halbuki PSO'da konumu kötüleşen parçacık ihmal edilmekte ve çözüme katılmamaktadır. Ancak simülasyon sonuçalrı göstermektedir ki ihmal edilen parçacıklar çözüme daha hızlı ve daha hassas bir şekilde yaklaşabilmektedir.



Şekil 3.38. Kuantum parçacık sürü optimizasyonu akış şeması

Kuantum PSO sezgisel tekniği için bu çalışmaya uyarlanan akış şeması Şekil 3.38'de görümektedir. Kuantum PSO için algoritma şu şekilde ifade edilebilir:

Parçacık boyutu başlangıcı (M)

For t = 1: Maksimum iterasyon (T)

mbest değerini hesapla

$\beta = (\beta_1 - \beta_0)$. (T - t) / T + β_0 ($\beta_0 = 0.3$ and $\beta_1 = 0.9$)

For i = 1: parçacık sayısı (M)

If Pozisyon Hatası (x_i) < pbesti ise pbesti = Pozisyon Hatası (x_i)

gbest = minimum(pbesti)

For d = 1: Parametre Boyutu

ϕ = rand (0, 1)

u = rand (0, 1)

$g_{id} = \phi.pbest_{id} + (1 - \phi).gbest_d$

If rand (0, 1) > 0,5

$Xid = g_{id} + \beta$. abs(mbestd – X_{id}) . log(1/u)

Else

$X_{id} = g_{id} - \beta$. abs($mbest_d$ – X_{id}) . log(1/u)

End if

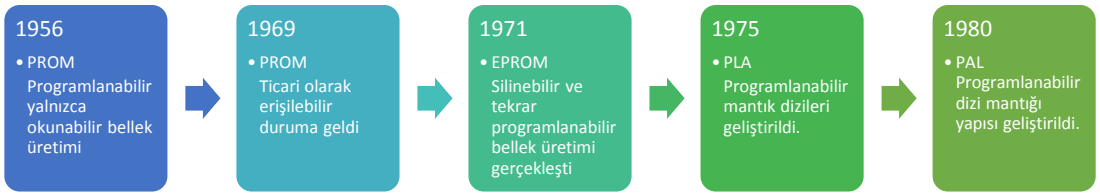
End for (d)

End for (i)

End for (t)

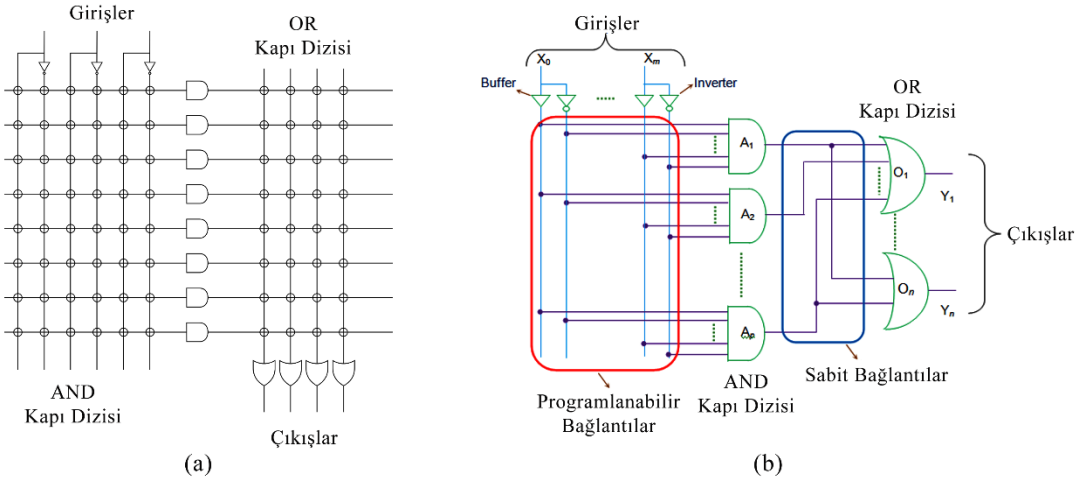
3.4. FPGA (Field Programming Gate Array)

FPGA endüstrisinin temelleri 1980'li yıllara kadar programlanabilir mantık cihazları (PLD – Programmable Logic Devices) ve programlanabilir salt okunur hafıza (PROM – Programmable Read Only Memory) teknolojileriyle atılmıştır. Elbette tahmin edilebilirki bu teknolojiler, tasarımı sonradan değiştirilemez ve tasarım hatası mevcutsa düzeltmesi yapılamaz teknolojilerdir. Bu nedenle mühendislik araştırmaları donanımların sonradan değiştirilmesine odaklanmaya başlamıştı. Böylece 1971 yılına gelindiğinde, günümüzde en büyük FPGA üreticisisi olan Altera isimli firma sonraki yıllarda ilk FPGA olarak anılanacak olan EP300'ü piyasaya sürdü. Bu tümleşik devre ile EPROM hücrelerine ulta-viole yani mor ışık vererek onların silinmesine ve tekrar yazılabilmesine yardımcı olmuştur (Sulaiman et al, 2009, pp. 3575-3576).



Şekil 3.39. Sayısal devre teknolojilerinin gelişim evresi

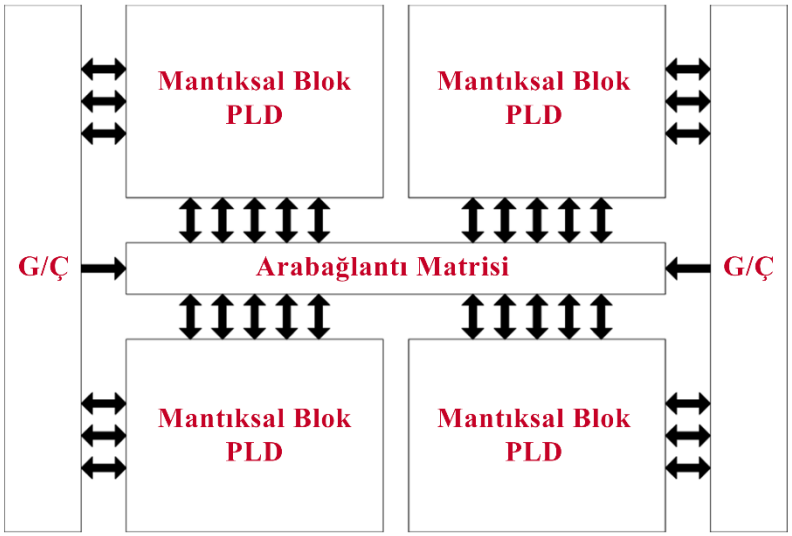
Programlanabilir kapı dizisi (PLA) fikri, sayısal tasarım ve programlama fikrini geliştiren bu yolda önemli mesafeler aldıran bir fikirdi. PLA (Programmable Logic Array), NAND kapıları ile oluşturulmuş ve kullanıcının istediği tasarımı yapabilmesi için araya girerek istediği kapıları bağlamasının yeterli olduğu bir çipten başka bir şey değildi. Ancak gerçek programalama çözümüne 1980'lerde PAL (Programlanabilir Dizi Mantığı – Programmable Array Logic) ile ulaşılmıştır (Krishna ve Roy, 2017, pp. 14-15).



Şekil 3.40. PLA (a) ve PAL (b) yapıları (Anon., 2018)

PAL, kullanıcı tarafından programlanabilir bağlantılara sahip iki seviyeli AND-OR yapısında olup hem maliyet hemde performans açısından PLA'lara üstünlük sağlamaktaydı. O ana kadar oluşturulan bu yapıların tamamına PLD (Programming Logic Device) yani programlanabilir mantık aygıtları ismi verilişken sonra aşamada bu aygıtlar ara bağlantı noktalarıyla bir araya getirilerek CPLD (Complex PLD) entegresi oluşturulmuştur (Şekil 3.41). Hâlbuki FPGA'lar arabağlantı noktalarındaki işlevselliğin artmasıyla beraber çok daha basit bloklardan oluşmaktadır (Serrano, 2008, p. 231).

1980'lerin sonuna gelindiğinde ise asıl FPGA fikrinin oluşmasına büyük katkı sunan bir gelişme yaşandı. Steve Casselman 600.000'den fazla yeniden programlanabilir cihazla bir bilgisayar üretme fikrini ortaya koydu ve 1992 yılında geliştirmiş olduğu bu bilgisayarın patentini aldı (Casselman, Thornburg, ve Schewel, 1995, p. 120). 1990'ların ortalarında ise FPGA fikri pek çok teknolojik alana girerek geniş bir yayılım gerçekleştirmiştir. O dönemde ağ iletişimi ve haberleşme en büyük faydalanıcı teknolojiler olmasına karşın zamanla bu işlem otomotiv sektörüne, endüstriyel ve tüketici uygulamalarına kadar genişledi (Monmasson et al, 2011, p. 226).



Şekil 3.41. CPLD içyapısı

Günümüzde FPGA aslında ASIC'in alternatifi olarak görünmesine rağman gerçek tamda öyle değildir. ASIC (Application Specific Integrated Curcuits), özel uygulamalar için geliştirilen tümleşik bir devredir ve FPGA gibi sayısal devrelerin yanında istenirse analog devreler ve alıcı-verici devrelerde içerebilmekte; güç tüketimi açısından daha az enerji harcayıp daha az yer kaplamakta ve daha hızlı işlem yapabilmektedir. Ancak tasarım maliyetleri açısından daha avantajlı hale gelebilmeleri için aynı ASIC devreden milyonlarca basılmalıdır (Kuon & Rose, 2010, pp. 39-58).

ASIC'lerin belkide en önemli dezavantajı, literatürde NRE (yenilenmeyen mühendislik - non-recurring engineering) olarak geçen ve yapılan tasarımların değişme kabiliyetinin sınırlı olup yeniden üretim aşamasına dönülmesidir (Munden, 2004, pp. 10-11). Yani ASIC tasarımında yapılan bir yanlışlığı düzeltmek özellikle üretimden sonra sistemi

yeniden üretmekten geçmektedir. İşte bu nedenlerden dolayı FPGA günümüzde artık daha fazla tercih edilebilir hale gelmiştir. Ancak, yapılan FPGA tasarımı artık son noktaya ulaştıysa ve bundan sonra değişmesi pek mümkün görünmüyorsa, yer ve güç tüketimini azaltmak adına bu tasarım ASIC'e dönüştürülerek kullanılmaktadır. Özellikle FPGA içerisindeki tasarım VHDL gibi sayısal tasarım dilleri ile gerçekleştirildiyse bu iş çok daha kolay olmaktadır (Kelleci, 2017, pp. 7-8). Bu iki teknolojinin detaylı karşılaştırması Şekil 3.43'da gösterilmiştir

Şekil 3.42.Sayısal mantık devrelerinin hiyerarşisi

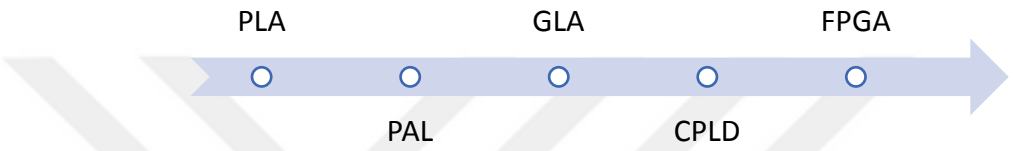
Bilindiği üzere ASIC'ler mikroişlemci ve mikrodenetleyicilerin aksine özel bir takım işlevleri veya görevleri yerine getirmek üzere tasarlanmış tümleşik devrelerdir. Şekil 3.43'te FPGA ile ASIC'lerin genel başlıklar halinde karşılaştırması verilmiştir. Bu şekle göre FPGA'nın etkinliği ilk üç başlıkta ortaya çıkmaktadır.

Özellik	FPGA	ASIC
Pazara sürme süresi	Hızlı	Yavaş
NRE (gelişme maliyeti)	Düşük	Yüksek
Tasarım şeması	Basit	Karmaşık
Parça maliyeti	Yüksek	Düşük
Performans	Orta	Yüksek
Güç tüketimi	Yüksek	Düşük
Parça büyüklüğü	Orta	Düşük

Şekil 3.43.FPGA ve ASIC karşılaştırması

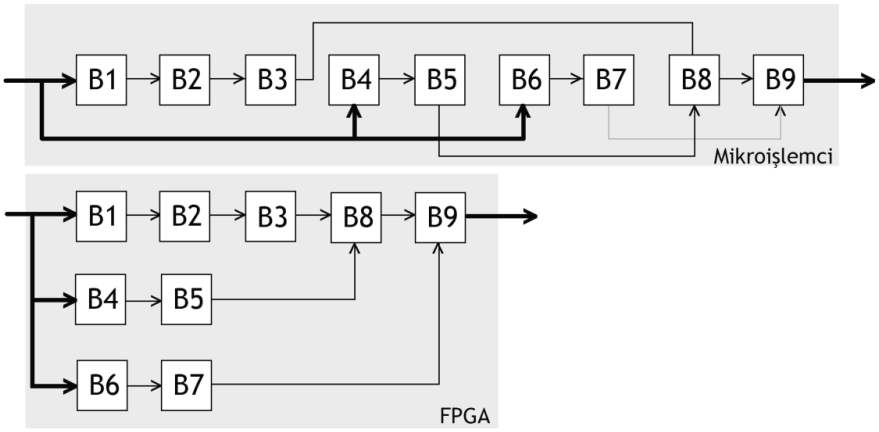
3.4.1. FPGA'ların Yapısı ve Özellikleri

FPGA (Alanda Programlanabilir Kapı Dizisi – Field Programmable Gate Array), herhangi bir alanda ihtiyaç duyulan sayısal devrelerin tasarlanabildiği ve gerektiğinde bu devrelerin donanım yapılarının sonradan değiştirilebildiği entegre devredir. Programlanabilir mantık aygıtlarının gelişim sıralamasında en sonda bulunan FPGA'ların en önemli özelliği tasarımlarının sonradan değiştirilebilmesidir (Farooq, Marrakchi, ve Mehrez, 2012, p. 8).



Şekil 3.44.PLD'lerin gelişim süreçleri

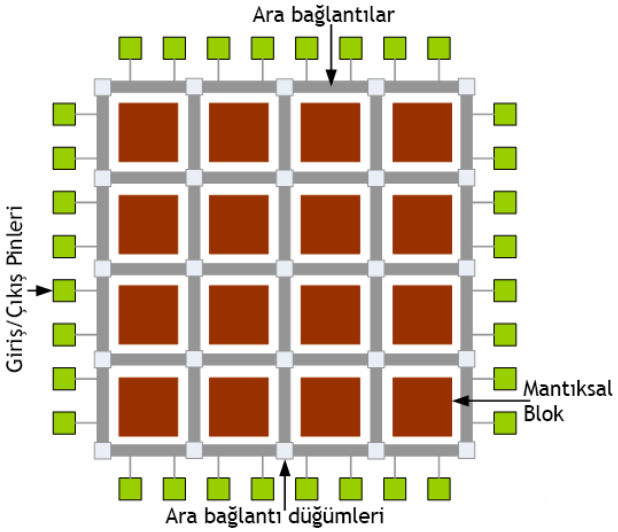
FPGA, paralel işlem yapabilme kabiliyetine sahiptir. Aynı amaca ilişkin oluşturulann bir devreyi çoklu çalıştırarak veya birbirinden bağımsız ve birbirinin oluşturdukları sonuçları kullanmayan devreleri saat darbesi ile aynı anda yani senkronize çalıştırarak paralel işlem yapılmış olmaktadır. Örneğin aynı anda yüzlerce resmi filtrelemeniz durumunda veya aynı anda binlerce siyali dönüştürmeniz gerektiğinde elbette paralel işlem yeteneği önemli derecede sistemi hızlandırmış olacaktır (Kastensmidt ve Rech, 2016, p. 3). Şekil 3.45'te mikroişlemcinin ve FPGA'nın belli görevleri seri ve parallel olarak işleme mimarileri görünmektedir.



Şekil 3.45.FPGA ve mikroişlemcinin seri ve paralel ve seri çalışma şekli

FPGA’ların da günümüzde simülasyon yazılımları yaygınlaşmıştır. Böylece yapılan tasarımın test edilerek hatalarından arındırılması ve en son çalışır durumdaki hali FPGA kartına yüklenebilmektedir. Oysa bu özellik FPGA’dan önceki PLD’lerde olmayan önemli bir özelliktir (Parvez ve Mehrez, 2010, pp. 77-80).

FPGA’nın en önmeli özelliği ise sahada programlanabilmektir. Yani bu entegre çalıştığı ortamda programlanabilmekte gerekli durumlarda bu tasarım değiştirilebilmektedir. Dolayısıyla bu durum hem zaman hemde maliyet açısında tasarımcıların elini kuvvetlendirmektedir (Sarıtaş ve Karataş, 2015, pp. 4-5).

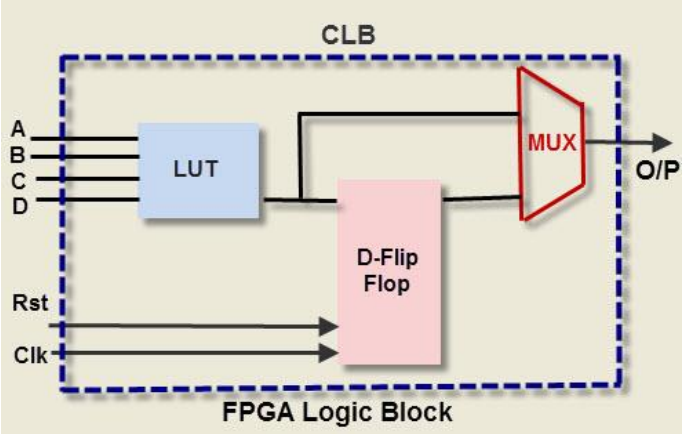


Şekil 3.46. FPGA'nın yapısı ve bileşenleri

Genel olarak FPGA üç bölümden oluşmaktadır:

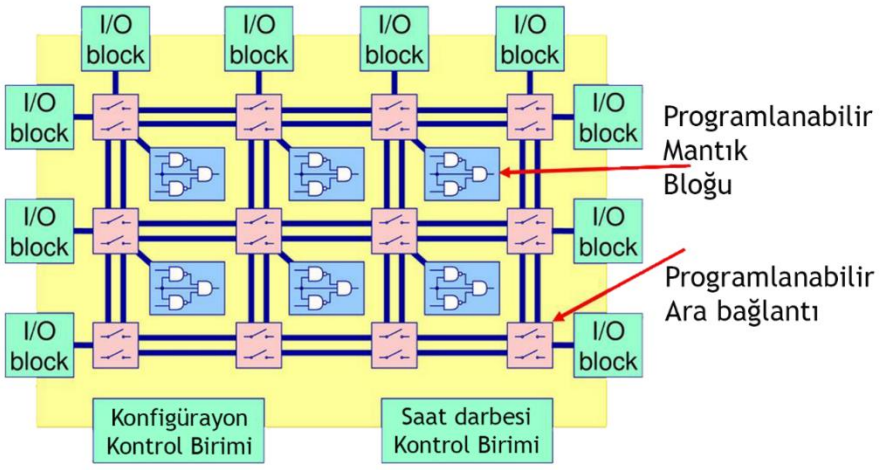
- CLB (Configurable Logic Block); yani programlanabilir mantık blokları
- Mantık bloklarını birbirine bağlamak için kullanılan programlanabilir ara bağlantılar (Interconnect Resources)
- FPGA’nın dış dünya ile bağlantısını sağlayan ve programlanabilen giriş çıkış birimleri

Programlanabilir mantık blokları (CLB), FPGA’ların temel yapısını oluşturan bloklardır. Şekil 3.31’de görüdüğü gibi yapılarında bir veya birkaç adet doğruluk tablosu (LUT – Look up table) ile birlikte çoklayıcı ve saklayıcı (Flip Flop) bulunmaktadır. Mantıksal işlemler bu bloklarda gerçekleştirilir.



Şekil 3.47.FPGA'lara ait programlanabilir mantık bloğu

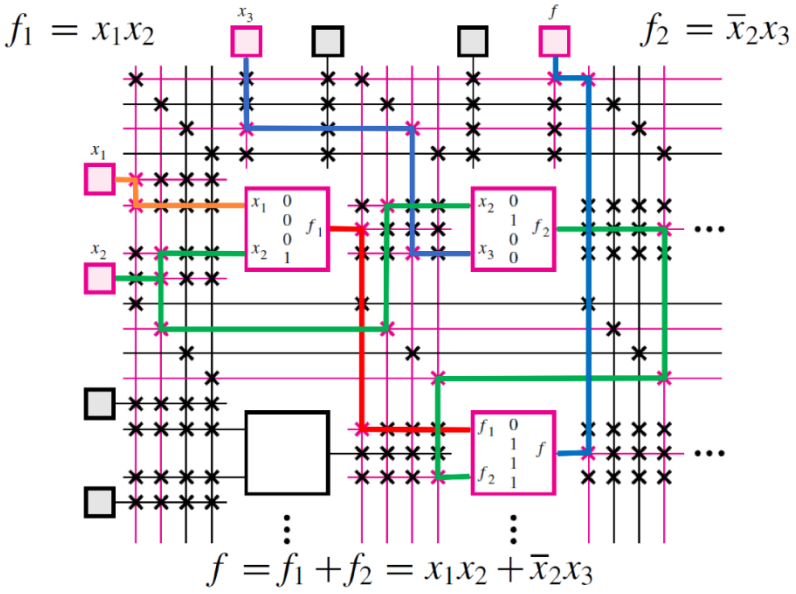
Her üreticinin geliştirmiş olduğu FPGA'da ki mantık bloklarının hem yapıları hemde adlandırılmaladı farklılık arz edebilmektedir. Xilinx firması ürettiği FPGA'larda kullandığı bu bloklara mantık hücresi (Logic Cell veya Slice) ismini verirken Altera firması ise mantık elemanı (LE) ismini vermiştir (Krishna & Roy, 2017, pp. 20-21).



Şekil 3.48.Temel FPGA yapısı ve programlanabilir arabağlantılar

Programlanabilir ara bağlantılar (Interconnect Resources), yapılan sayısal devre tasarımı için elbette tek bir mantık hücresi yeterli gelmeyecek birden fazla mantık hücresi kullanılması gerekecektir. Sayısal tasarımda mantık blokları arasında bağlantı kurularak bir bloktan diğerine sinyal aktarımı yapılması gerekecektir. İşte bunun için kullanılan yapılardır (Sulaiman et al, 2009, p. 3576).

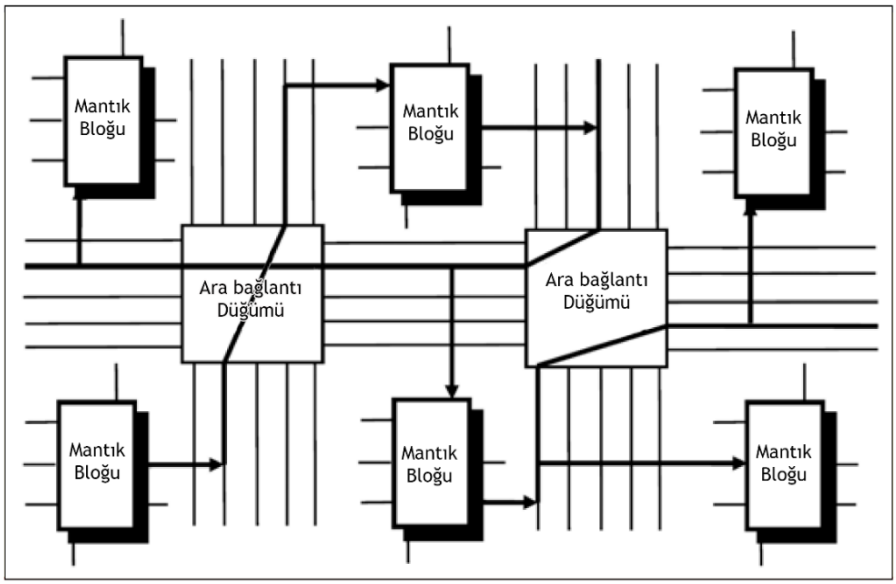
Şekil 3.48'de göründüğü üzere arabağlantılar anahtarlara benzetilerek ifade edilmeye çalışılmıştır. Aslında tam da böyledir ki, ara bağlantılar bir anahtar misali iletişim halinde olması gereken mantık blokların arasındaki bağlantıyı sağlayan anahtarın kapatılıp diğerlerinin açılması şeklinde programlanmaktadır.



Şekil 3.49.FPGA içerisindeki ara bağlantıların sinyal taşımaları (Christi, 2015)

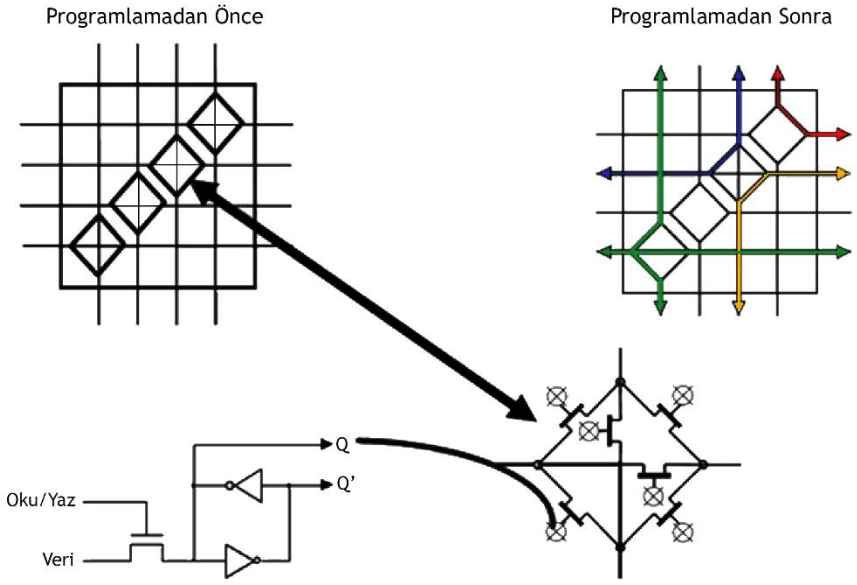
Şekil 3.49'de ara bağlantılar vasıtasıyla sinyallerin bir mantık bloğundan diğerine nasıl taşındığı açıkça görünmektedir. f1 ve f2 fonksiyonları mantık bloklarında işlemden geçerler. Hem mantık bloklarına gelen giriş verileri hemde mantık bloklarından çıkan çıkış verileri sinyal olarak arabağlantılar vasıtasıyla diğer mantık bloğuna ve oradan da giriş/çıkış birimlerine aktarılır.

Ara bağlantıların doğru bir şekilde yönlendirilmesi ise Şekil 3.50'te göründüğü gibi arabağlantı düğüm noktasında (Switch Matrix) gerçekleştirilmektedir. Yani, bir mantık bloğundan çıkan sinyalin diğer mantık bloğuna veya çıkış birimlerine yönlendirilmesi ise anahtar kutusu (Switch Box) tarafından gerçekleştirilmektedir. Bu anahtar kutularının gerçekte bağlantılarının nasıl oldukları ise Şekil 3.51'de görünmektedir (Monmasson et al, 2011, p. 227).



Şekil 3.50.Arabağlantılar ve anahtar kutuları arasındaki ilişki (Christi, 2015)

Şekil 3.50'de arabağlantıların mantık bloklarla bağlantısı açıkça görünmektedir. Şekil 3.51'de göründüğü üzere arabağlantı sistemini oluşturan mantıksal devrenin oluşumu açıkça gösterilmektedir. Özellikle programlama sonrasında yapılan şaseleme işleminin hangi devre ile gerçekleştiğine dikkat çekmek çok önemlidir.



Şekil 3.51.Programlanabilir ara bağlantılarının yapısı (Christi, 2015)

Programlanabilen giriş çıkış birimleri, FPGA'ların içerisindeki mantık blokları ile dış dünyada bulunan çevresel arabirimler ve farklı uygulamalarla arasındaki bağlantıyı

sağlayan yapılardır. Standart bir FPGA kartında azımsanmayacak derecede geniş bir alana sahip olan bu birimlerin tasarımı besleme gerilimi ile referans gerilimi arasında büyük farklar olduğundan oldukça karmaşıktır (Teubner & Woods, 2013, p. 22).

RAM blokları, verileri geçici olarak kaydeden yapılardır ve pek çok uygulamada verilerin geçici veya kalıcı olarak saklanması durumu sözkonusudur. FPGA'da da mantık blokları arasında işlem yaparken verileri geçici olarak kaydetmesi gerektiğinde yine bu mantık blokların arasında dağınık halde bulunan saklayıcılar yoğun olarak kullanılmaktaydı. Ancak günümüzde pek çok FPGA kartında ayrılmış bellek bölümleri bulunmaktadır ve bu sayede mantık blokları diğer işlemlerde kullanılmak üzere ayrılmış olmaktadır (Sarıtaş ve Karataş, 2015, p. 15).

3.4.2. FPGA'ların programlanması

Aslında FPGA'da programlamanın tam karşılığı tasarımdır. Yani yapılmak istenen bir işlem için programlamadan ziyade tasarım yapılır. Ancak bu tasarım kodlama yöntemiyle yapıldığından dolayı bu ifade kullanılmaktadır. FPGA'ların programlanması için günümüzde yaygın olarak kullanılan ve isimlerini donanım tanımlama dili olarak alan VHDL (Hardware Description Language) ve Verilog kullanılır (Cavanagh, 2015). Bu tanımlama dillerinin yapısı C ya da Pascal dillerine çok benzediği için donanım tasarlama işine yeni başlayanlar aslında büyük bir yanılgıya düşerek işe başlamaktadırlar. Çünkü onlar kodlama işini yazılımsal olarak düşünerek işe başlamaktadırlar. Hâlbuki bu iş tam olarak elektroniksel bir devre tasarlamaktır. O nedenle FPGA programlama işine başlamadan evvel donanım tasarlama ya da sayısal devre tasarlama mantığının iyi bir şekilde kavranması gerekmektedir (Sarıtaş & Karataş, 2015; Sulaiman et al, 2009, pp. 3576-3578).

Gerek Verilog gerekse de VHDL 1980'li yıllarda hayatımıza girmiş ve IEEE standartı almış güçlü donanım tanımlama dilleri arasında olup günümüzde en yaygın şekilde kullanılan sayısal devre tasarımı dilleridir. Tasarımcılar bu diller sayesinde yapmış oldukları tasarımları RTL (Register-Transfer-Level) seviyesinde test imkânı

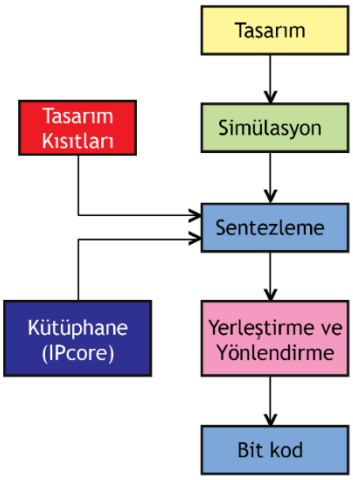
bulmuşlardır. Öncesinde ise tasarımcılar test işlemi için mantık kapılarını kullanmak zorundaydılar (Dekker, 2014).

VHDL, Verilog'a göre daha eski olup daha ziyade Pascal programlama dilini esas almıştır. Bu nedenle bu dilin karakteristik özelliklerini kendinde barındırmaktadır. Oysa Verilog, C programlama dilinin kodlama tekniğini esas almıştır. Bu nedenle VHDL'nin tersine C'de olduğu gibi Verilog'da büyük/küçük harf ayrımına itina göstermektedir. Ayrıca C dilinin kullanımı programcılar arasında daha yaygın olduğundan dolayı Verilog, VHDL diline nazaran daha kolay öğrenilmektedir (Wilson, 2015, pp. 20-21).

VHDL, güçlü tipte yazılmış bir dildir ve zayıf tipleri ya da değişkenlerin farklı sınıflarla karıştırılmasını asla kabul etmez. Yani Verilog ile karşılaştırıldığında aşırı düzeyde kuralcı olan bir dildir. O nedenle bu dili kullanacak tasarımcılar VHDL'nin pek çok ayrıntısına hâkim olmak durumundadırlar. Hâlbuki Verilog, zayıf tipte yazılan bir dildir ve katı kuralcılıktan çok daha uzaktır (Joan, 2010).

3.4.3. FPGA tasarım teknikleri ve akış şeması

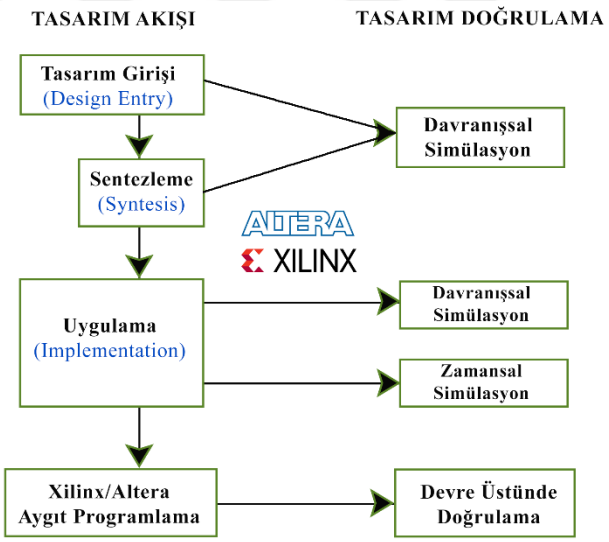
Aslında pekçok yazılım geliştirme süreci gibi sayısal devre tasarım sürecide belli aşamalardan oluşmaktadır. Özellikle bu tasarımların gömülü sistem programlama olarak anılan sistemlerle benzerliği son derece fazladır. Nihayetinde sayısal tasarımda da fiziksel bir aygıta yazılan kodun yüklenmesi gerekmektedir.



Şekil 3.52.FPGA tasarım akışı

Elbette her projenin başlangıcı bir fikirden ibarettir. Burada ise fikrin projeye dönüşmesi için sayısal devre tasarımı yapmakla süreç başlamakta, bir dizi evrelerden geçmekte (Şekil 3.52) ve nihayetinde yeni bir donanım tasarımı ile son bulmaktadır (Botros, 2005).

Günümüzde FPGA konusunda öncü olan iki önemli üretici firma bulunmaktadır. Bunlar Altera ve Xilinx'tir. Her iki firmanında pek çok türde üretmiş olduğu farklı özellikte FPGA kartı bulunmaktadır. Elbette tasarımlarda önemli ölçüde olmasa da yapısal farklılıklar mevcuttur. Ancak tasarım akışı (design flow) açısından kullanılan araçlar ya da arabirimler dışında herhangi bir farklılık sözkonusu olmamaktadır. Şekil 3.53'da görüleceği üzere Sentezleme aşamasında kullanılan arabirim Altera'da Quartus II iken Xilinx firması ISE veya Vivado arabirimini kullanmaktadır (Sklyarov, Skliarova ve Sudnitson, 2012, p. 1144).

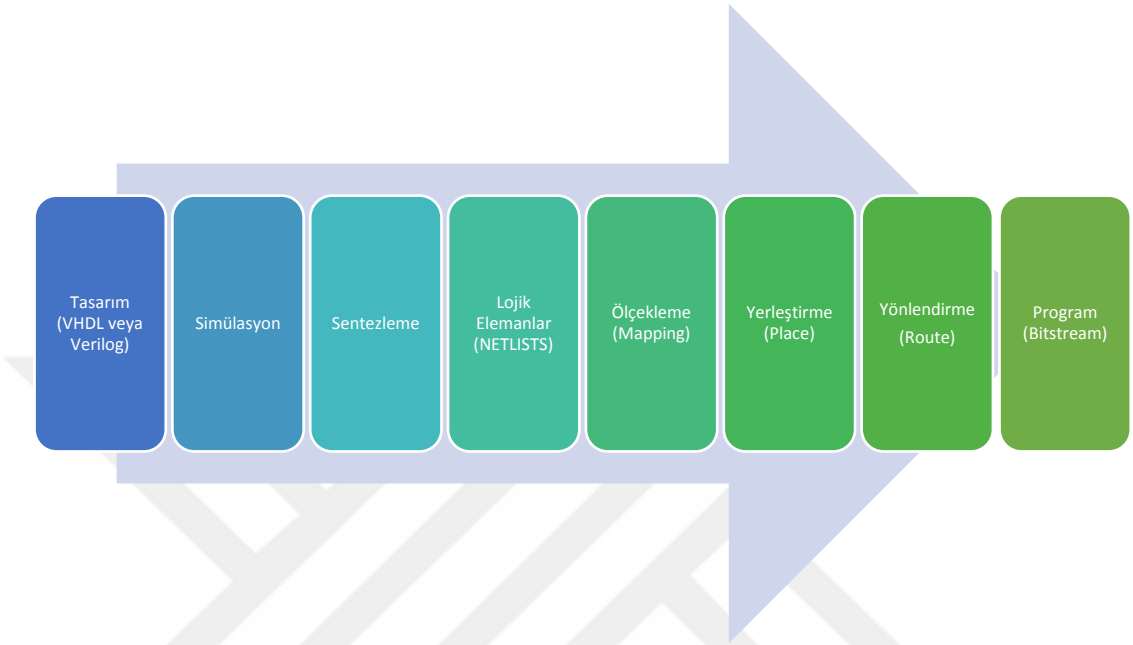


Şekil 3.53.Altera ve Xilinx üreticileri için tasarım akışı

3.4.3.1. Tasarım (Design Entry veya RTL Design)

Proje fikrinin ilk uygulamaya geçtiği aşama olup bu fikrin bilgisayarla temsilini ifade etmektedir. Bunun için günümüzde yaygın şekilde kullanılan donanım tanımlama dilleri (HDL – Hardware Description Language) olarak bilinen VHDL ve Verilog kullanılmaktadır. Bu iki dilden herhangi biri ile sayısal devre tasarımı bilgisayar

ortamında gerçekleştirilir. Burada bilinmesi gereken en önemli husus, yazım kuralları bakımından geleneksel programlama dilleri olan Pascal veya C'ye benzemesine karşın bu dillerden farklıdır. Özellikle kodlamadaki ifadeleri sıralı bir şekilde çalıştırmaması en önemli farktır (Stavinov, 2011).



Şekil 3.54.FPGA tasarım akışı

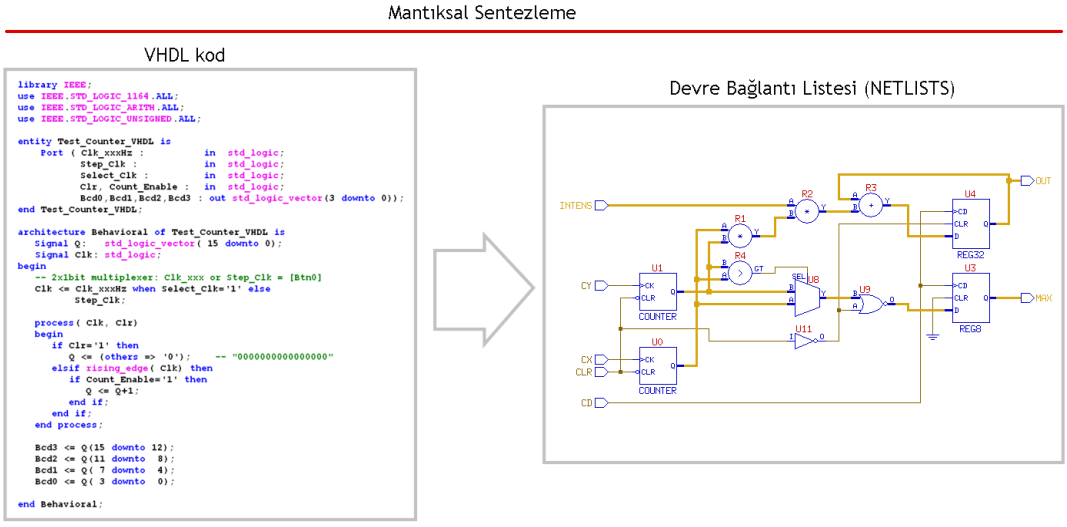
Şekil 3.54'te VHDL veya Verilog ile yapılan tasarımdan sonra karta yükleyinceye kadar yapılan işlemler görünmektedir. VHDL kodlayarak yapılan tasarım aslında sentezleme ile beraber mantıksal aşamalara sokulmaktadır.

3.4.3.2. Benzetim (Simulation)

Yapılan tasarımın FPGA kartına yüklenmeden önceki en son halini fonksiyonel açıdan görmek adına yapılan işlemdir. Ancak burada dikkat edilmesi gereken en önemli etken tasarım ekranında yapılan bütün işlemlerin benzetim ekranından öteye gidememesidir. Yani kodlama ekranında kullanılan bazı komutların sentezlenememesidir. Örneğin VHDL ile gerçel sayılar üzerinde işlem yapmak için kullanılan "math_real" kütüphanesindeki komutlar sentezlenememektedir. O nedenle tasarımcılar kodlama işlemini yaparken, yaptıkları tasarımın FPGA kartına yüklenmesi söz konusu ise bu tasarımdaki komutların sentezlenebilir olmasına dikkat etmeleri gerekmektedir (Sklyarov, Skliarova, & Sudnitson, 2012, p. 1144).

3.4.3.3. Sentezleme (Sytesis)

Bu bölümde; oluşturulan HDL dosyasına ve kullanılan FPGA modeline göre bir liste (Netlist) oluşturulur. Kısacası RTL ve HDL kodlarının boolean cebrine dönüştürülerek lojik elemanların oluşturulduğu aşamadır. Aslında bu liste; mantıksal devrenin optimizasyonu, kaydedicilerin gereksiz yüklerden arındırılarak dengeli şekilde kullanımı ve zamanlama performansını artırmak için son derece önemlidir ve bazı aşamalardan sonra elde edilir. Böylece bu liste (Netlist) HDL tasarımının çok verimli bir şekilde kullanılmasını sağlar (Serrano, 2008, p. 234). Şekil 3.55'de bir HDL tasarımın sentezlenerek ağ listesinin oluşturulması görünmektedir. Bu şekilden de net bir şekilde göründüğü üzere VHDL kodlama ile oluşturulan tasarım sayısal lojik elemanlara dönüştürülerek birbirine bağlanmıştır.

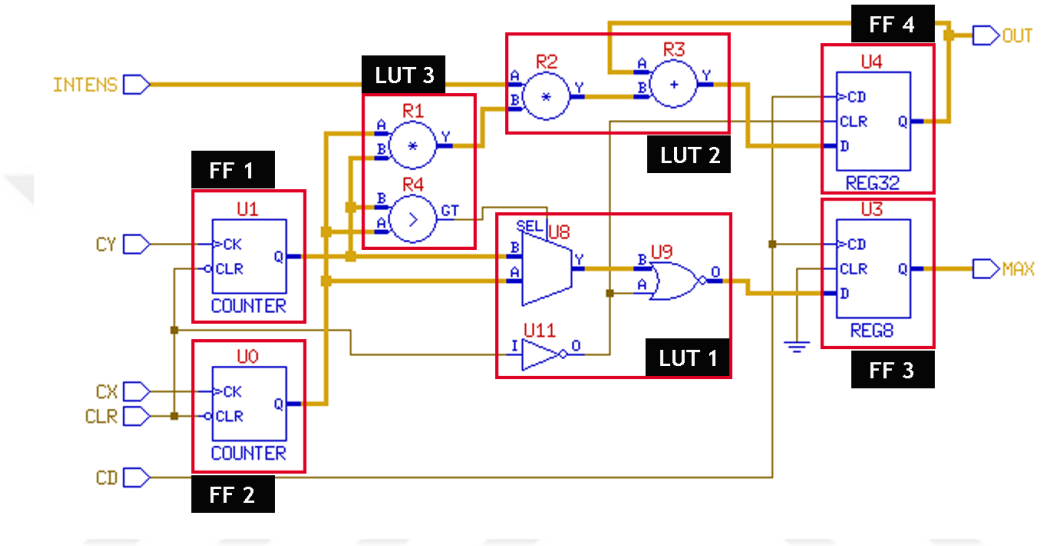


Şekil 3.55. HDL tasarımın mantıksal sentezlenme aşaması (Lojic Synthesis)

3.4.3.4. Yerleşim Planlaması veya Ölçekleme (Mapping)

Boolean cebrine dönüştürülerek elde edilen lojik elemanlar bu aşamada, kullanılan FPGA modeli esas alınarak kullanılacak LUT, kaydedici ve FF elemanları ile devre giriş ve çıkış pinleri sayısı belirlenir. Eğer belirlenen bu eleman sayısı kullanılacak FPGA modelinde ki sayıdan fazla ise bu işlem burada ortaya çıkar ve kullanılan arabirim (Quartus, ISE,

Vivado, vb) gerekli hata uyarısını vererek işlemi keser (Wilson, 2015, p. 53). Şekil 3.38'de örnek bir planlama (Mapping) işlemi görünmektedir. Lojik elemanların kullanacağı LUT ve FF sayısı ile giriş ve çıkış pinleri sayıları belirlenmiştir. Elbette bu sayılar belirlendikten sonra gerekli LUT ve FF sayısı kullanılacak FPGA modeliyle uyumlu olmak zorundadır. Aksi halde işlem burada kesilmek zorunda kalacaktır (Hassan & Anis, 2010, p. 13).



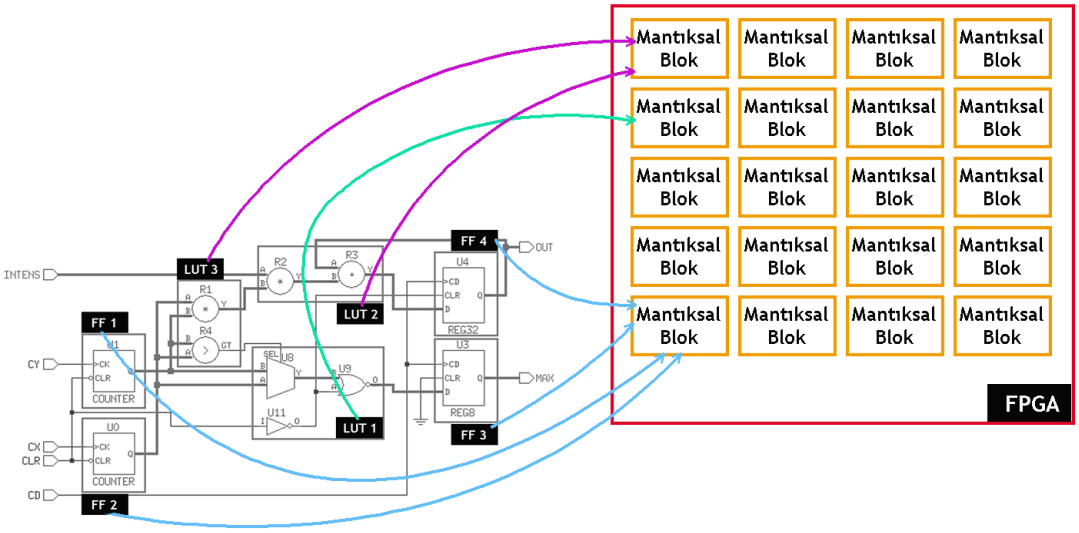
Şekil 3.56.Ölçekleme (Mapping) aşaması

3.4.3.5. Yerleştirme (Placing)

Planlama (Mapping) aşamasında elde edilen Netlist elemanlarının (LUT, FF, RAM, giriş-çıkış pinleri, gibi) FPGA kartında bulunan mantık bloklara yerleştirildiği aşamadır. Yerleştirme işleminde elemanlar farklı mantık bloklarına yerleştirilebilir (Cavanagh, 2008).

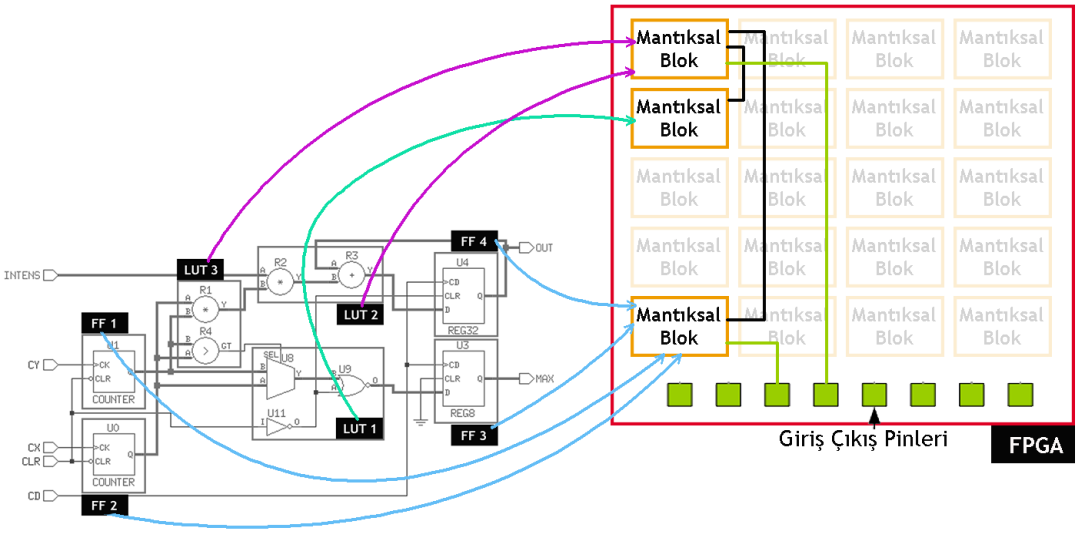
Şekil 3.57'de haritalama (mapping) yöntemiyle elde edilen elemanların gerçek FPGA kartındaki mantıksal bloklara yerleşimi görünmektedir. Burada bütün FF'lar tek bir mantıksal bloğa, LUT2 ve LUT3 aynı mantıksal bloğa, LUT1 ise yine farklı bir mantıksal bloğa yerleştirilmiştir. Örneğin bu tez çalışmasında kullanılan donanımda 15,850 adet mantıksal blok, her bir mantıksal blokta ise 8 adet FF ve 4 adet 6 girişli LUT

bulunmaktadır. Yani yukarda ki resimde gösterilen devre elemanları aslında tek bir mantıksal blokta işlenebilmektedir.



Şekil 3.57.Tasarım akışı içerisinde bulunan yerleştirme aşaması

3.4.3.6. Yönlendirme (Routing)



Şekil 3.58.Yönlendirme aşaması

Sentezleme neticesinde elde edilen eleman listesinde (Netlist) bulunan öğelerin (LUT, FF, RAM,…vb), yerleştirildiği mantık bloklarının programlanabilir ara bağlantılarla (programmable interconnection) birbirine bağlandığı aşamadır. Böylece hangi mantık

bloğunda tasarıma ait bir öğe varsa o mantık bloğu tasarıma bağlanmış olmaktadır (Hassan & Anis, 2010, p. 16; Wilson, 2015, p. 53).

Yerleşim planlaması (Mapping) aşamasından mantıksal öğeler farklı LUT veya FF'lere yerleştirilmesine ragmen kullanılan FPGA kartının kapasitesine bağlı olarak aynı mantık bloğunda barıdırılabilmektedir. Şekil 3.58'de bunun bir örneği görünmektedir. LUT2 ve LUT3 aynı mantık bloğuna yerleştirilmiştir.

3.4.3.7. Program (Bitstream) Dosyası

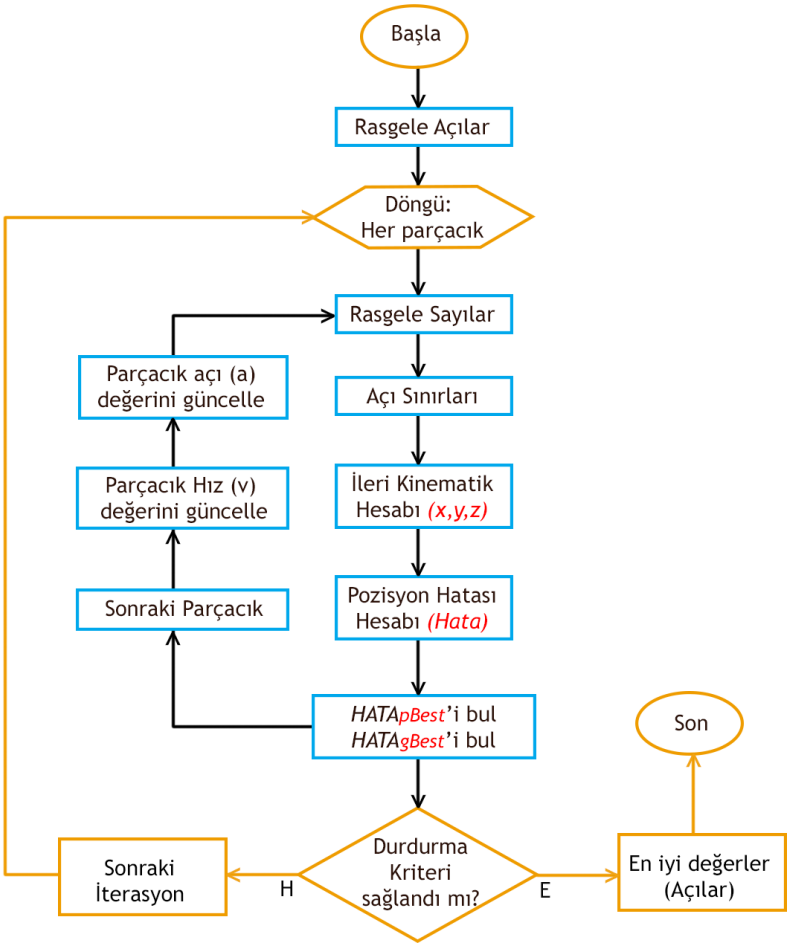
Tüm aşamalar başarıyla tamamlandığında oluşan ve FPGA programı hakkında her türlü bilginin bulunduğu dosyadır. Bitstream ifadesi, FPGA programlamanın aslında oluşturulan bit akımı ile gerçekleştiği fikrinden dolayı ortaya çıkmıştır. Oluşan "bitstream" dosyası FPGA projesinde yeniden açılarak pin atamaları yapılmalıdır. Örneğin şekil 3.58'ta örnek bir yönlendirme görünmektedir. Bu şekilde "clk" pini ile beraber onbir adet "input" pini bulunmaktadır. Her bir pini FPGA üzerindeki girişlerle ilişkilendirmek gerekmektedir. Örneğin "clk" girişini FPGA üzerinde bulunan "E3" nolu pine atama işlemini "bitstream" dosyasında gerçekleştirdikten sonra bu dosya FPGA'ya yüklenir (Çavuşlu & Kösten, 2015, pp. 274-282).

3.4.4. Donanımsal Parçacık Sürü Optimizasyonu ile Ters Kinematik Hesabı

Ters kinematik konusu, robotik biliminin en temel konuları arasında yer almaktadır ve araştırmacılar tarafından ziyadesiyle ilgi odağı olmuştur. Çünkü bu konu, zor, karmaşık yapılı ve çözümü uzun süren ama aynı zamanda öncelikle çözümlenmesi gereken bir konudur. Ancak günümüzde konu pek çok yöntemle çözüme kavuşmasına rağmen çözüm süresi açısından henüz istenilen seviyelere ulaşamamıştır. İşte bu nedenle bu durum konunun bu tez çalışmasında ele alınmasına öncülük etmiştir.

Parçacık sürü optimizasyonu, literatürde pek çok alanda yaygın şekilde kullanılan sürü zekâsına odaklanarak en iyi çözümleri bulmak için kullanılan bir sezgisel en iyileme tekniğidir. Algoritmada kullanılan parametre sayısının azlığı nedeniyle diğer sezgisel tekniklere göre nispeten daha kolay olduğu bilinmektedir. Ancak arama ve en iyi çözüme

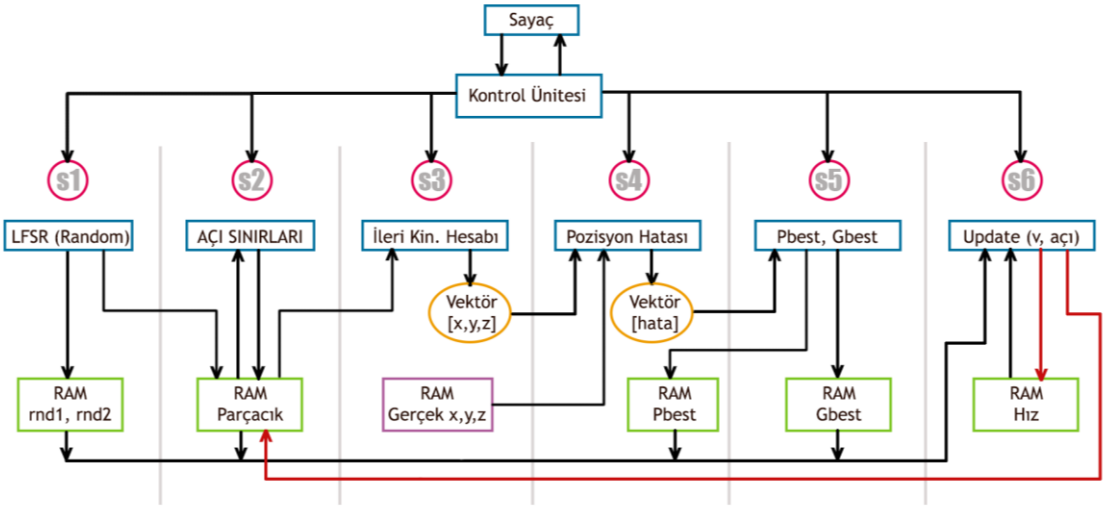
yönelme sezgisi özellikle IW (inertia weight) olarak bilinen atalet ağırlığı özelliği ile güçlendirilebilmektedir (Harrison et al, 2016). Böylece klasik PSO ya göre çok daha iyi sonuçlar elde edilebilmiştir (Dereli & Köker, 2018, p. 83).



Şekil 3.59. PSO algoritması donanımsal çalışma akış şeması

Şekil 3.20'de klasik PSO akış şeması görünmektedir. Bu tip rassal arama işlemi ile en iyi çözüme ulaşmayı amaçlayan algoritmalar genellikle rassal değerlerle işleme başlarlar. Bu algoritmanın yazılımsal olarak kodlanması ile donanımsal olarak tasarlanması açısından bazı ufak faklılıklar olsada genel itibariyle çok büyük değişiklikler olmamaktadır. Şekil 3.59'de PSO algoritmasının donanımsal çalışma akış şeması görünmektedir. Elbette bu akış şeması yazılımsal PSO ile karşılaştırıldığında en büyük fark rasgele sayılar kısmında göze çarpmaktadır. Çünkü donanımsal işlemlerde kullanılacak rasgele sayılar önceden belirlenip saklama aygıtlarında saklanmak durumundadır. Burada da her parçacığın hız

güncellenmesinde kullanılacak iki adet rasgele sayı öncesinde üretilerek hazırda bekletilmektedir.



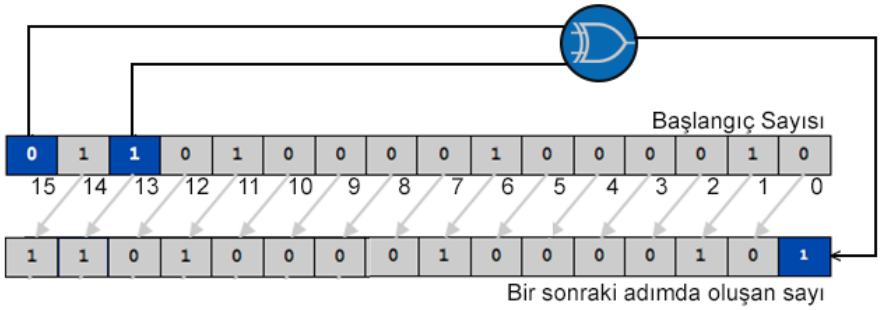
Şekil 3.60. PSO algoritmasının blok şeması

PSO algoritmasının FPGA kartı için tasarlanan blok şeması şekil 3.60'te görünmektedir. Genel itibariyle altı aşamadan (s1, s2, s3, s4, s5, s6) oluşan mantıksal devrede "s5" aşaması davranışsal olarak tasarlanmasına karşın diğer bütün aşamalar yapısal olarak tasarlanmıştır. Şekilden de açıkça görüneceği üzere tasarım "LFSR (rasgele değerler), açı sınırları, ileri kinematik hesabı, pozisyon hatası, pbest - gbest ve güncelleme" olmak üzere altı adet alt tasarımdan (component) oluşmaktadır. Tasarlanan devre ardışıl niteliğe sahip olduğundan dolayı kontrol ünitesi olmaksızın bu devreyi doğru bir şekilde çalıştırmak imkânsızdır. O nedenle "s1"'den "s6"'ya kadar olan ifadeler her bir aşamayı ifade etmektedir. Örneğin "s1" aşaması tamamlanmadan "s2" aşamasına geçiş mümkün olmamaktadır. Bunun içinde her bir alt devreye "aktif" ucu eklenerek devrenin sadece kendi aşamasında çalışması sağlanmıştır.

3.4.4.1. Rasgele sayılar ve açılar (LFSR)

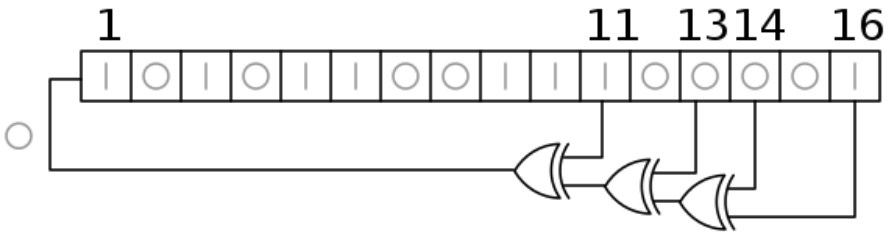
LFSR (Doğrusal Geribeslemeli Kaydırma Yazmacı - Left Feedback Shift Register), dizi şifreleme sınıflandırmaları arasında olup rasgele doğrusal ahenk dizisi üretmek için kullanılan bir yöntemdir. Çünkü dizi şifreleyicilerin ürettikleri anahtarın düzgün bir dağılıma sahip olması ve dışardan tahmin edilemez olması gerekmektedir. İşte ihtiyaç

duyulan özelliklere sahip anahtarlar üretmek için doğrusal geri beslemeli kaydırma yazmacı (LFSR) çok sık kullanılmaktadır (Doğan, 2006, p. 8). İkilik tabandaki sayılar üzerinden çalışan bu sistem iki adımdan oluşmaktadır. İlki; tanımlanan doğrusal bir fonksiyon ile mevcut bitlerden yeni bir bit elde edilir, diğeri ise mevcut sayıların bir bit kaydırılarak açılan boşluğa oluşturulan yeni bitin yerleştirilmesidir (Din et al, 2016, p. 436).



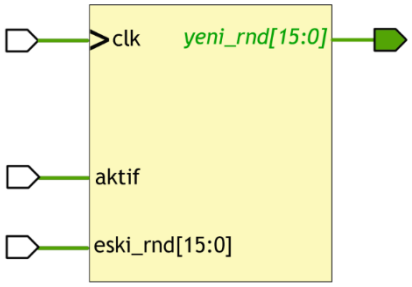
Şekil 3.61. LFSR algoritmasının uygulanması

Şekil 3.61'te LFSR algoritmasının 16-bitlik bir sayı üzerindeki örnek bir uygulaması görünmektedir. Doğrusal fonksiyon 13. ve 15. bitleri mantıksal özel veya (EXOR) işlemine tabi tutarak yeni bir sayı oluşturuyor ve bu sayı 0. bite gelirken 0. bitten itibaren diğer sayılar sola doğru bir bit kaydırılarak yeni sayı oluşturuluyor. İlk sayı onaltılık tabanda "6872H" iken oluşturulan yeni sayı onaltılık tabanda "d085H" 'tir.



Şekil 3.62. LFSR'de doğrusal fonksiyon oluşturma

Şekil 3.62'de doğrusal geribeslemeli kaydırma yazmacı için örnek bir fonksiyon eşliğinde işlemin yapılışı görünmektedir. LFSR için kullanılan fonksiyon EXOR (özel veya) ve EXNOR (özel veya değil) işlemlerinden oluşmaktadır. Yani bitler kendi aralarında mantıksal EXOR ve EXNOR işlemine tabi tutulmaktadır (Kim et al, 2016).

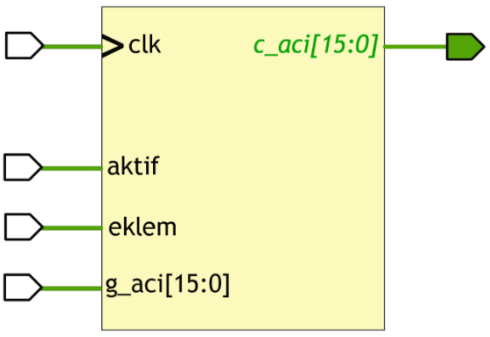


Şekil 3.63. Bu çalışmada kullanılan rasgele sayı üreteci

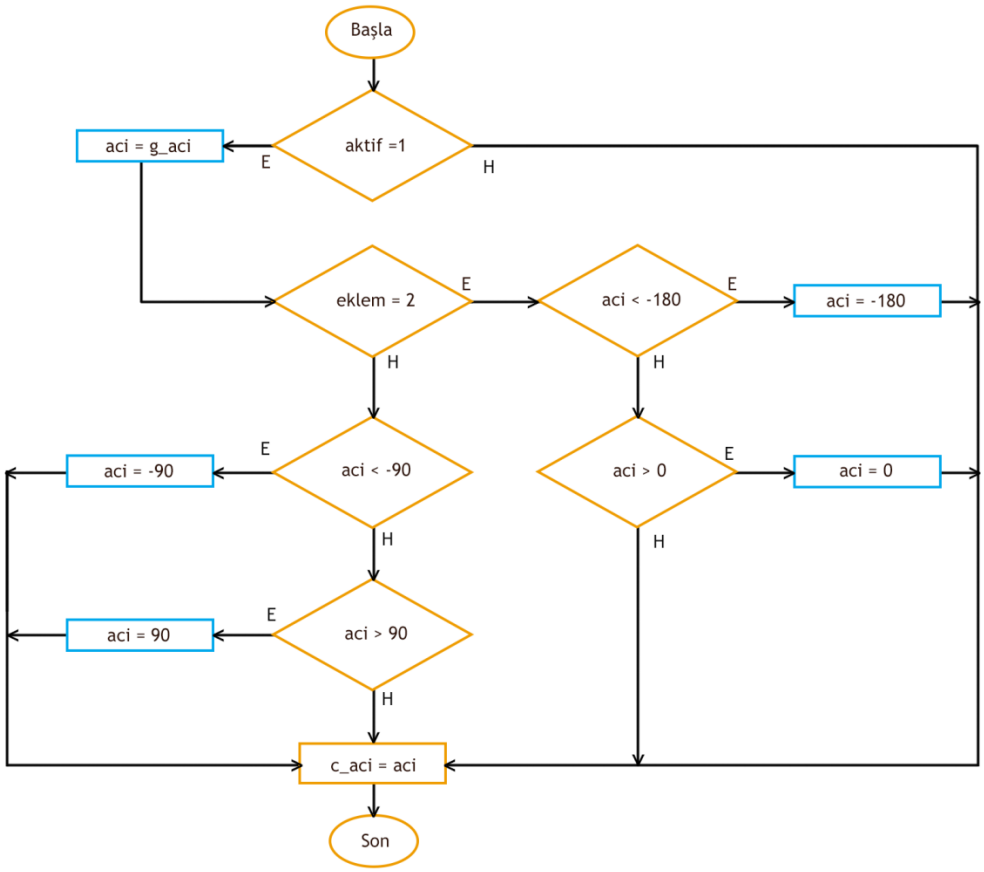
Bu çalışmada kullanılan rasgele sayı üretici devre tasarımı Şekil 3.61'de görünmektedir. PSO algoritmasında hem açıların başlangıç değerleri hemde pozisyon güncelleme esnasında kullanılan iki adet rasgele sayılar bu devre tasarımı ile oluşturulmuştur. Bu LFSR tasarımında 13'üncü, 5'inci ve 3'cü bitler özel veya değil (EXNOR) işlemine tabi tutulmuş ve oluşan bu yeni ilk sayının sola bir bit kaydırılmasından sonra 0. bite yerleştirilmiştir.

3.4.4.2. Açı sınırları

Robot manipülatöründeki eklemler belli aralıklarda açı değerlerini alarak konumlama gerçekleştirmektedirler. Bu sınırın dışına çıkılması manipülatörün yanlış konumlanmasına veya arzu edilen konumdan farklı bir konuma yönlenmesine sebep olacaktır. O nedenle sezgisel optimizasyon algoritmalarının tamamında bu işlem yapılmaktadır. Bu çalışmada açıların alabileceği değerler Tablo 3.1'de gösterilmektedir. Buradan da görüneceği üzere 2. eklem 0 derece ile -180 derece arasında değerler alabilirken dışındaki diğer tüm eklemler -90 derece ile 90 derece arasında değerler alabilmektedir. Bu işlem için kullanılan tasarımın akış şeması şekil 3.64'de görünmektedir.



Şekil 3.64. Açı sınırları alt devresi giriş-çıkış pinleri



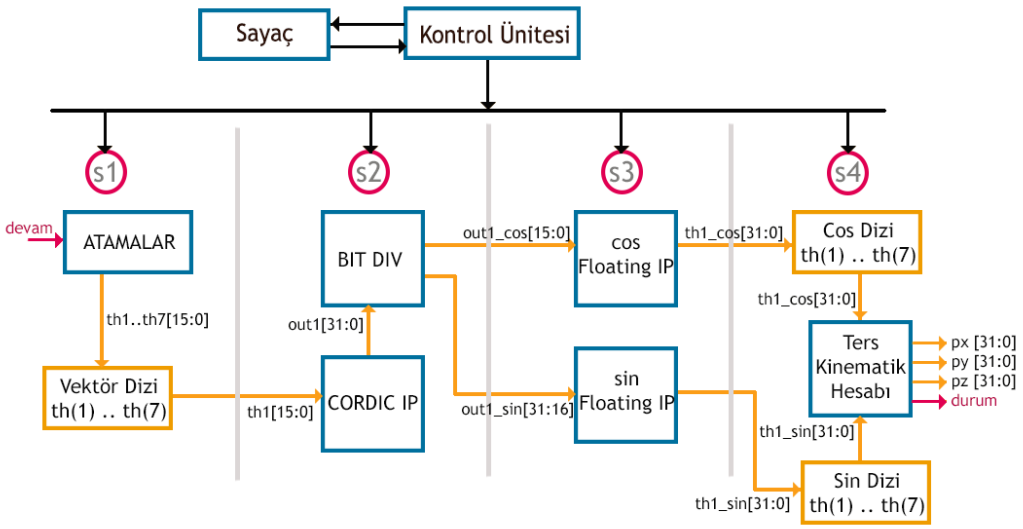
Şekil 3.65. Açı sınırları için izlenen akış şeması

Şekil 3.64'te görünen akış şemasında "aktif" ucu, alt tasarımın çalışması gerektiği durumlarda "1" olur. Diğer durumlarda "0" yapılarak hem gereksiz çalışması ve hemde yanlış değerler üretmesi engellenmiş olur. Şekil 3.65'te ki alt devre giriş çıkış pinlerinde göründüğü üzere "g_aci" giriş pinidir ve 16-bitten oluşan bir sayıdır. Tasarım bir sayının sınır aralıklarında olup olmadığını, sınır aralıklarında değilse sınır aralık değerlerine çekildiği bir dizi işlem aşamalarından oluşur. En son aşamada ki 16-bitlik çıkış pini olan "c_aci" sayısı bu aralıklarda oluşan sayıdır.

3.4.4.3. İleri kinematik hesabı

Alt mantıksal devre (component) olarak kullanılan bu tasarımda yedi adet eklem açısı alınır ve ileri kinematik denklemleri vasıtasıyla manipülatöre ait uç elemanın x, y ve z eksenlerindeki konumu elde edilir. Şekil 3.66'da göründüğü üzere mantıksal devre de "Cordic" ve "Floating Point" olmak üzere iki adet "IP Core" kullanılmıştır. "Cordic IP", 16-bitlik sayıların sinüs ve kosinüs değerlerini 16-bit + 16-bit yani toplamda 32-bit olarak

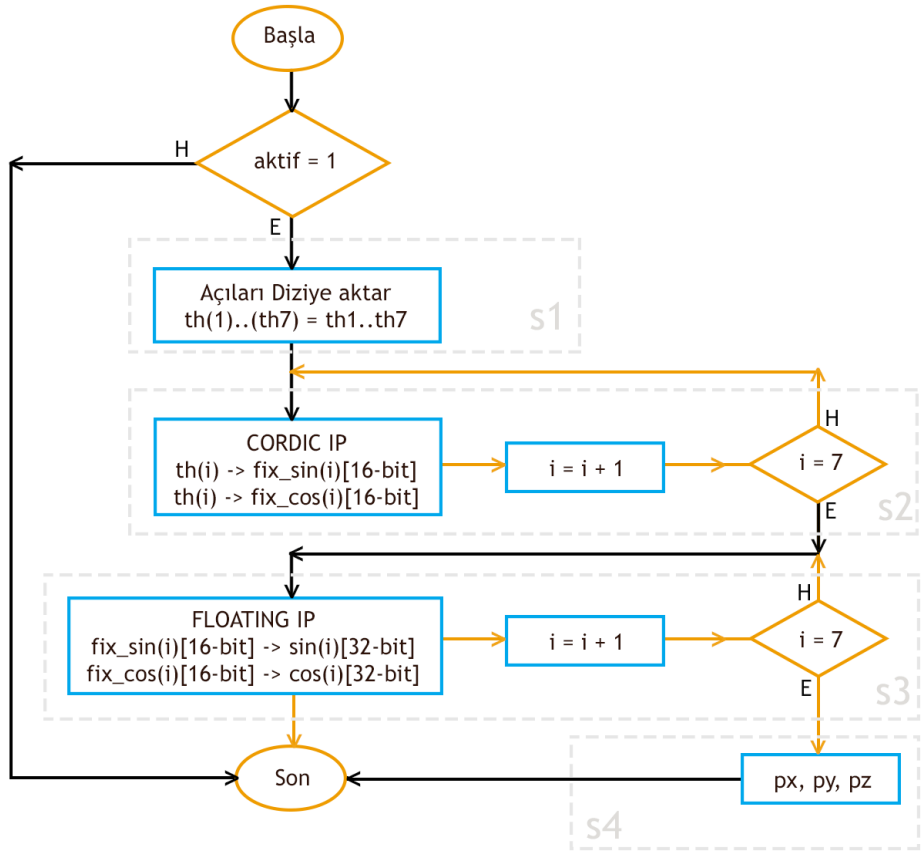
üreten bir "IP Core"'dur. "Floating Point IP" ise 16-bitlik sayıları 32-bitlik ondalık sayıya dönüştürmek için kullanılır.



Şekil 3.66. İleri kinematik blok devre şeması

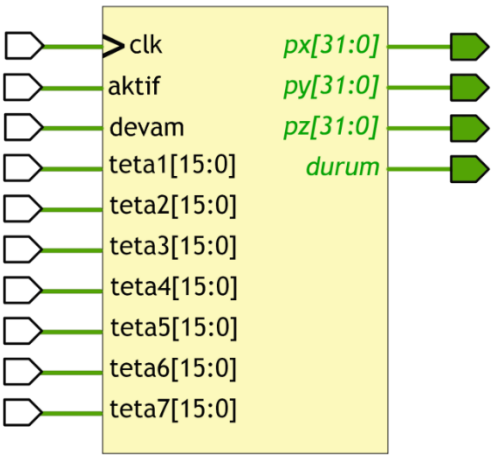
Şekil 3.66'da mantıksal devrenin yapısal blok şeması görünmektedir. Bu devrede "s1" ve "s4" aşamaları davranışsal olarak "s2" ve "s3" aşamaları ise yapısal olarak tasarlanmıştır. Mantıksal devre genel olarak dört adımdan (s1, s2, s3, s4) oluşmaktadır: Birinci adımda her bir eklem açısı öncelikle bir diziye atılır. İkinci adımda, bu dizideki her bir eklem açısının "Cordic IP" ile 16-bitlik sinüs ve kosinüs değerleri üretilir. Üçüncü adımda 16-bitlik kosinüs ve sinüs değerleri 32-bitlik ondalık sayıya dönüştürülür ve son adımda ise ileri kinematik hesabı gerçekleştirilir.

Şekil 3.67'de ileri kinematik hesaplaması yapan mantıksal devrenin akış şeması görünmektedir. Algoritmada "CORDIC" algoritmasının çalıştığı "s2" aşamasında ve "FLOATING" algoritmasının çalıştığı "s3" aşamasında birinci eklem açısından yedinci eklem açısına kadar döngü kullanılmış ve "s2" aşamasında bütün eklem açıları için sabit (fixed) tipinde 16-bitlik sinüs ve kosinüs değerleri elde edilmiştir. Benzer şekilde "s3" aşamasında da yine tekrarlı yapı kullanılmış ve her defasında bir eklem açısının sinüs ve kosinüs değerleri 32-bitlik ondalıklı sayıya dönüştürülmüştür. "s4" aşaması ise 32-bitlik x, y ve z konumlarının hesaplandığı aşamadır.



Şekil 3.67. İleri kinematik mantıksal devresinin akış şeması

Tasarlanan mantıksal devrenin yapısal şeması şekil 3.68'de göründüğü üzere; "Cordic IP" ile sinüs ve kosinüs değerlerinin elde edilmesi ve "Floating IP" ile sinüs ve kosinüs değerlerinin 32-bitlik ondalıklı sayıya dönüştürülmesi her bir eklem açısı için tekrar etmektedir. Eğer herhangi bir eklem açısı süreci tamamlamazsa yapılan ileri kinematik hesabı yanlış olacaktır. O nedenle "durum" pini bu amaç için kullanılmıştır. Bu devrede "aktif" pini bir önceki açı sınırları alt devresinde olduğu gibi mantıksal devrenin kontrol ünitesi tarafından uygun zamanlarda devreye girmesini sağlamak için kullanılmıştır. Ana mantıksal devrede her bir parçacığın konum değerleri (x, y, z) bir RAM bölgesine yazılmaktadır. Yazma işleminin zamanlaması için bu pin kullanılmıştır. "devam" pini ise parçacıkların tamamının ileri kinematik hesaplamasının yapılabilmesi için bir döngü oluşturmak için kullanılmıştır.

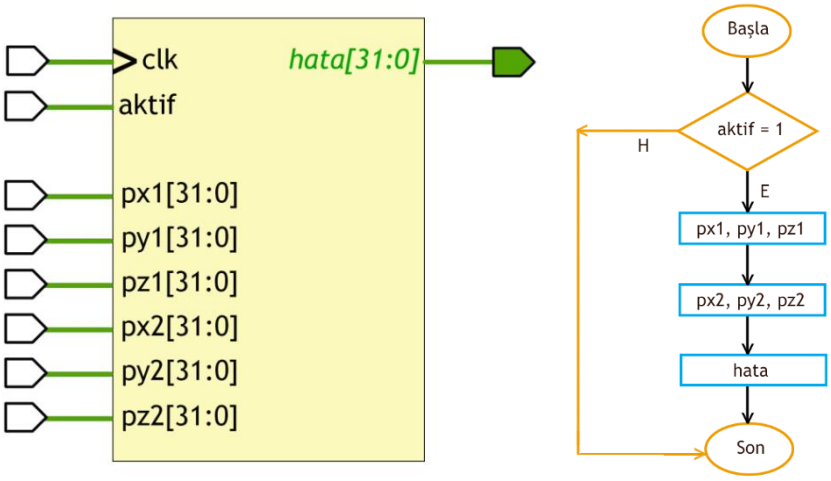


Şekil 3.68.İleri kinematik alt devresi giriş-çıkış pinleri

3.4.4.4. Pozisyon hatası

Bu çalışmada robot manipülatörüne ait uç elemanın önceden belirlenen bir noktaya en yakın şekilde ulaşması için sürü algoritmaları temelinde sezgisel teknikler kullanılmıştır. Sezgisel teknikler rassal olarak hedefe yaklaşan teknikler olduğundan dolayı elde edilen çözümün arzu edilen noktaya ne kadar yakın olduğu pozisyon hatası değeri ile belirlenmektedir ve bunun için (Denklem 3.15) kullanılmaktadır. Bu işlem FPGA devresinde bu mantıksal devre ile elde edilmektedir.

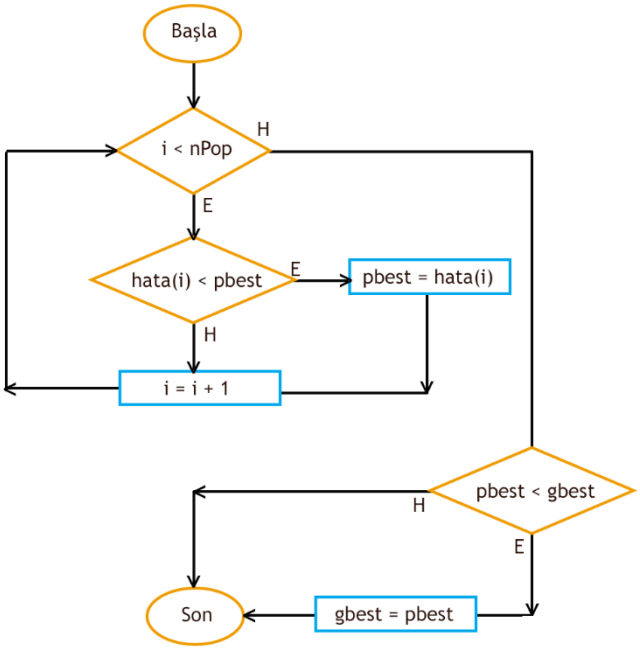
Pozisyon hatası alt devresi giriş-çıkış pinleri Şekil 3.69'da görünmektedir. "aktif" pini önceki devrelerde olduğu gibi bu mantıksal devrenin uygun zamanlarda çalışması için kullanılmaktadır. "px1, py1, pz1" arzu edilen ve robot manipülatörünün ulaşması istenen konum bilgilerini, "px2, py2, pz2" ise çözüm olarak bulunan eklem açıları sonucu ortaya çıkan konum bilgileridir. Bu işlem neticesinde çıkış olarak 32-bitlik "hata" pini kullanılmıştır.



Şekil 3.69. Pozisyon hatası alt devresi giriş-çıkış pinleri (sol şekil) ve akış şeması (sağ şekil)

3.4.4.5. En iyi değerler

Bir önceki aşamada elde edilen pozisyon hatasına göre çalışan bu bölümde en küçük pozisyon hatasına sahip eklem açıları yani parçacık bulunur. Tüm PSO devresinde sadece bu aşama davranışsal olarak tasarlanmıştır ve akış şeması Şekil 3.70'te görünmektedir.



Şekil 3.70. En iyi değerlerin elde edildiği mantıksal devre akış şeması

Akış şemasında da göründüğü üzere başlangıçta büyük bir değer atanan “pbest” değeri her parçacığın pozisyon hatası değeri ile karşılaştırılır. Eğer parçacığa ait pozisyon hatası değeri “pbest” değerinden daha küçükse bu durumda yeni “pbest” değeri değişir. Tüm parçacıkların pozisyon hatası değerleri karşılaştırılmışsa son aşamada da “pbest” değeri “gbest” ile karşılaştırılır ve daha küçük olan değer “gbest” olarak devam eder.

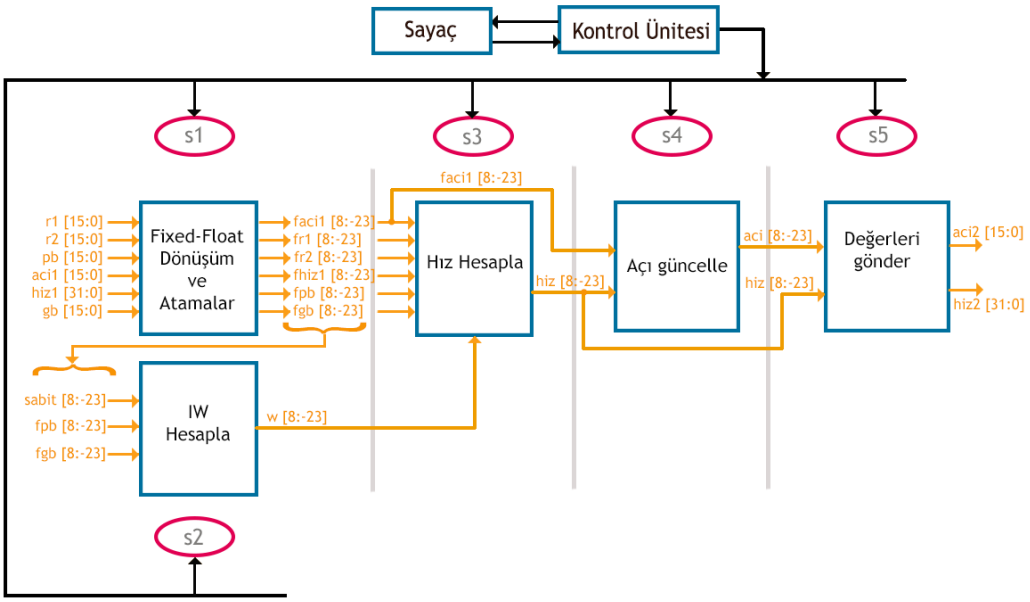
Bu mantıksal devre bölümü davranışsal olarak çalıştığından ötürü “aktif” pininin kullanılmasına gerek yoktur. Çünkü bu devre sadece, “kontrol ünitesi”’nin çalışma sırasını bu bölüme yönlendirdiğinde çalışmasını gerçekleştirecektir.

3.4.4.6. Açı ve hız değerlerini güncelleme

PSO algoritmasına göre parçacıkların çözüm noktasına ulaşmak için kullandıkları önemli aşamalardan bir tanesidir. Bu mantıksal alt devre de Şekil 3.71’de göründüğü üzere beş aşamadan (s1, s2, s3, s4, s5) oluşur ve her aşamaya geçiş diğer mantıksal devrelerde olduğu gibi “kontrol ünitesi” vasıtasıyla gerçekleşir. Bu aşamanın en belirgin özelliği tüm işlemler 32-bitlik ondalıklı sayılar şeklinde yapılmasıdır. Bu nedenle devreye gelen bütün sayılar öncelikle 32-bite genişletilmiş sonrasında ise ondalıklı sayıya (float[8,-23]) dönüştürülmüştür. Çıkışta ise güncel açı değeri 16-bit, hız değeri ise 32-bit olarak işlenmektedir.

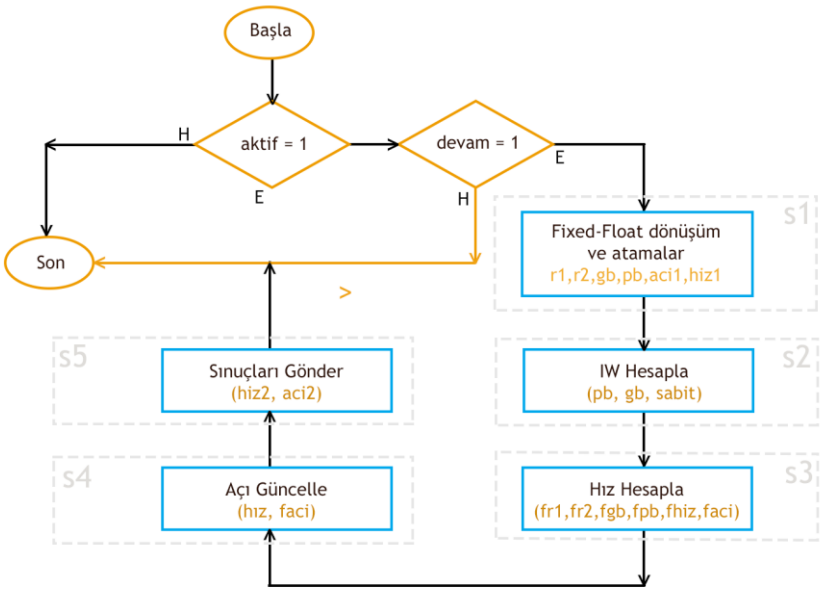
Devrenin blok şeması şekil 3.71’de görünmektedir. Devreye 16-bitlik veriler olarak giriş yapan rasgele sayılar (r1 ve r2), güncellenecek açı değri (aci1), yerel ve genel en iyi parçacığa ait açı değerleri (pb ve gb), 32-bite genişletildikten sonra yine 32-bitlik ondalıklı sayılara dönüştürülmektedirler. Bu işlemler devrenin “s1” yani ilk aşamasında yapılmaktadır. Bu işlem bittikten sonra “kontrol ünitesi” devreye girerek çalışmayı bir sonraki yani “s2” aşamasına atlatır. Bu aşamada ise güncelleme anında kullanılacak hız değerini belli bir ağırlıkta değiştirmek için atalet ağırlığı işlemi (IW) yapılmaktadır. Burada “Sabit IW” işlemi yapılmıştır. Daha sonra ise bir sonraki “s3” aşaması devreye girecek ve parçacığın bir sonraki konuma geçmesini yani çözüme yaklaşmasını sağlayan hız değeri hesaplanmıştır. Bu işlemde bittikten sonra açı değerinin yani parçacık

konumunun güncellendiği “s4” aşaması devreye girmektedir. En son aşamada ise hesaplanan hız ve açı değerleri üst devreye iletilmektedir.



Şekil 3.71. Pozisyon güncelleme mantıksal devresi blok şeması

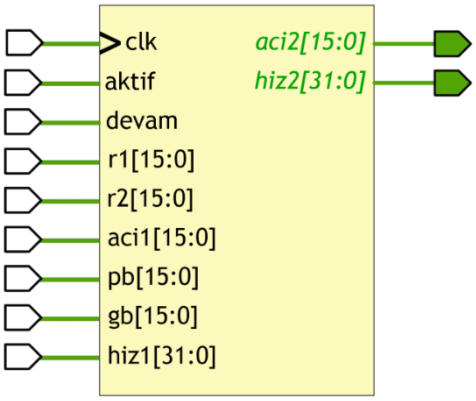
Mantıksal devrenin akış şeması Şekil 3.72’de görünmekte olup çalışmasının ilk koşulu “aktif” girişinin, sonrasında ise “devam” bitinin “1” olmasıdır. Sonrasında işlemler ardışıl bir şekilde sırayla devam etmektedir.



Şekil 3.72. Pozisyon güncelleme devresi akış şeması

Bu mantıksal devreye ait giriş-çıkış pinleri Şekil 3.73'te görünmektedir. Bu pinlerin açıklamaları şu şekildedir:

- "clk"; saat darbesi girişi.
- "aktif"; devrenin kontrol ünitesi tarafından aktif hale gitirildiği giriş pini. "1" ise devre aktiftir, diğer durumda ise pasiftir.
- "devam"; parçacık sayısınca işlemin devam etmesi için kullanılan giriş pinidir. "1" ise işlem yapılacak parçacık hala var demektir. "0" ise işlem yapılacak parçacığın kalmadığını belirtir.
- "r1 ve r2"; rasgele üretilen sayılar.
- "aci1"; konum güncellemesi yapılacak parçacık açı değeri girişi.
- "pb"; yerel en iyi parçacığa ait açı değeri (pbest).
- "gb"; genel en iyi parçacığa ait açı değeri (gbest).
- "hiz1"; parçacığın varsayılan hızı.
- "aci2"; güncellenen açı değeri.
- "hiz2"; güncellenen hız değeri.



Şekil 3.73. Pozisyon ve hız güncelleme giriş-çıkış pinleri

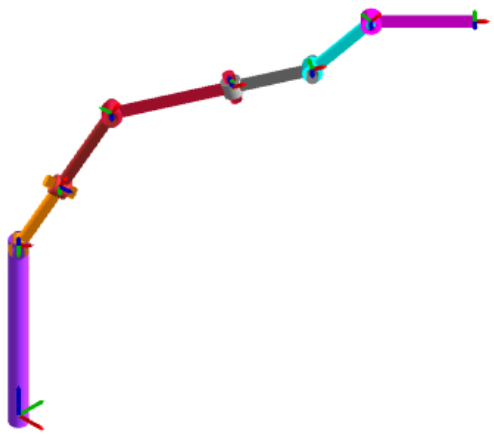
BÖLÜM 4. ARAŞTIRMA BULGULARI

Bu bölümde meta-sezgisel tekniklerle yapılan testlerin sonucu detaylı bir şekilde sunulurak analiz edilmiştir. Yapılan analizde izlenen yol şu şekildedir:

- Adım-1: Robot manipülatörü manuel olarak belirlenen eklem açılarıyla belli bir noktaya yönlendirilmiştir.
- Adım-2: Robot manipülatörünün adım-1'de yönlendirildiği noktanın çalışma uzayndaki konum bilgileri düz kinematik denklemleriyle (x, y ve z) elde edilmiştir.
- Adım-3: Sürü zekâsı ile çalışan algoritmalarla (Denklem 3.15) kullanılarak adım-2'de elde edilen konuma en yakın nokta hesaplanmaya çalışılmıştır.

Tablo 4.1.Manuel olarak belirlenen eklem açıları ve uç eleman konumu

Eklem	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
Açılar	45	-60	0	-45	0	30	-45
UE Konum Bilgileri		X		Y		Z	
Değerler (cm)		19,75		24,84		23,36	



Şekil 4.1. Robot manipülatörüne ait uç elemanın konumu

4.1. Simülasyon Sonuçları ve Analizleri

Bu tez çalışmasında ki amaçlardan biri robot kolu uç elemanının çalışma uzayında önceden belirlenen bir konuma minimum hata yaklaşmasını sağlayacak eklem açılarının sezgisel optimizasyon algoritmaları ile bulunmasıdır. Diğeri ise bu işlem için ihtiyaç duyulan hesaplama süresini gerçek bir robotta uygulanabilir duruma FPGA ile getirmektir. Bunun için bu algoritmalardan bir tanesinin (PSO) FPGA tasarımının yapılarak aynı işlemin bu tasarım vasıtasıyla da gerçeklenmesi konusuna odaklanılmıştır. Sonuçlar ve analizleri bu bölümde detaylıca ortaya konmuş ve analiz edilmiştir. Her bir sezgisel algoritma 10 defa çalıştırılmış ve elde edilen sonuçlar pozisyon hatası ve çalışma süreleri açısından karşılaştırılmıştır.

4.1.1. Parçacık sürü optimizasyonu temelli yapılan çalışma sonuçları

Parçacık sürü optimizayonu gerek IW teknikleriyle gereksede farklı yöntemlerin PSo'ya entegre edilmesiyle çözümü iyileştirme hususunda güçlendirilebilmektedir. Bu çalışmada PSO ile birlikte Quntum PSO ve RDV PSO yöntemleride kullanılmıştır.

4.1.1.1. Parçacık sürü optimizasyonu çalışma sonuçları

Yapılan benzetimde kullanılan sabit parametrelere verilen değerleri şu şekildedir:

- İterasyon sayısı: 100
- Popülasyon sayısı: 100
- c1 = 1.4, c2 = 1.4 ve w = 0,7

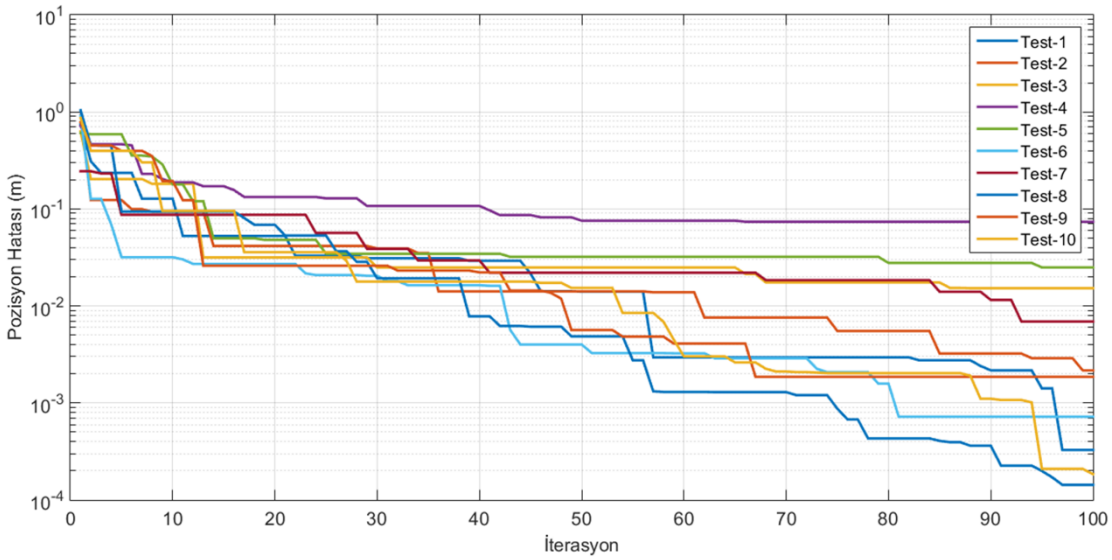
Tablo 4.2. PSO algoritması ile yapılan çalışma sonuçları

Test No	Çözüm Süresi (s)	Pozisyon Hatası (m)	İterasyon Sayısı	İterasyon Süresi (s)
1	0.6734	3.28e-04	97	0.6544
2	0.7252	2.16e-03	98	0.7313
3	0.5779	1.52e-02	88	0.6502
4	0.6323	8.24e-03	94	0.6802

Tablo 4.2. PSO algoritması ile yapılan çalışma sonuçları (devamı)

5	0.7056	2.49e-02	95	0.7365
6	0.6705	7.22e-04	83	0.8552
7	0.7949	6.90e-03	93	0.8552
8	0.7803	1.43e-04	97	0.7992
9	0.4947	1.85e-03	71	0.6838
10	0.6318	2.09e-04	95	0.6630
Ortalama	0.6686	6.06e-03	91	0.7309

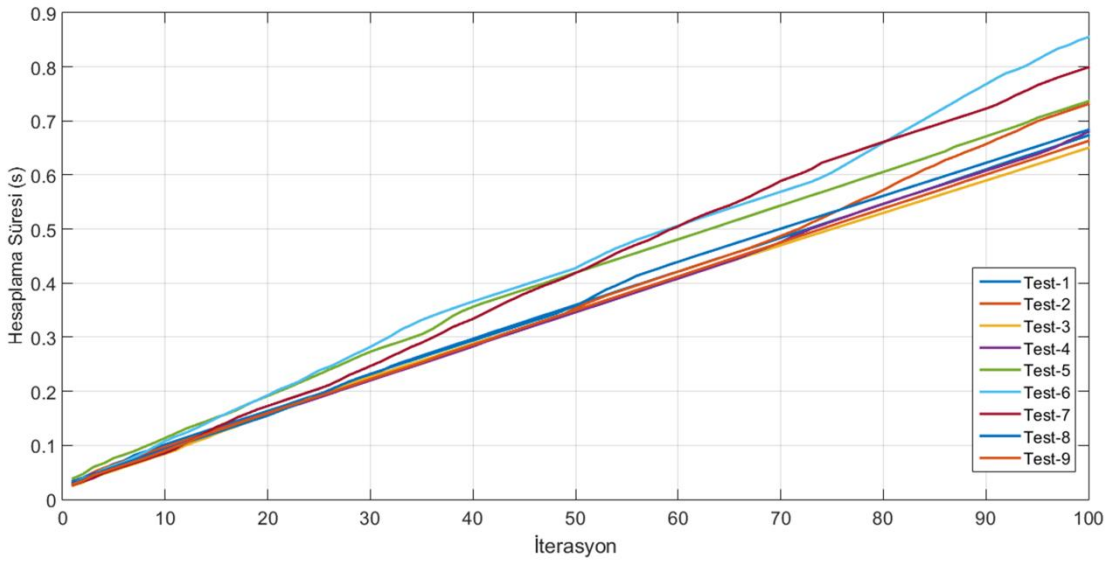
Tablo 4.2, arka arkaya yapılan simülasyon sonuçlarını göstermektedir. 100 popülasyonla yapılan çalışmalarda algoritmanın tamamlanma süresi her çalışmada ortalama 0.7 saniye olarak göze çarpmaktadır. En iyi çözüm olarak 1.43e-04 cm en kötü çözüm ise 2.49e-02 cm olarak ortaya çıkmıştır.



Şekil 4.2.PSO algoritması pozisyon hatası (m)

Şekil 4.2 de pozisyon hatası grafikleri, Şekil 4.3'te ise algoritmanın tamamlanma yani 100 parçacık ile 100 iterasyonun bitiş süreleri gösterilmektedir. Bitiş süreleri arasında büyük bir fark olmamakla birlikte pozisyon hatası sonuçlarında 100 katlık bir fark dikkati çekmektedir. Bu da sezgisel tekniklerin en iyi çözümü garanti edememeleri sonucunun getirmiş olduğu bir durumdur. Şekil 4.3'te algoritmanın 10 denemede ki herbir testi 100

iterasyona tamamlama süresi görünmekte olup aslında sürelerin birbirine çok yakın olmakla birlikte 0.65 ile 0.85 s arasındadır.



Şekil 4.3.PSO algoritmasının tamamlanma süresi (s)

4.1.1.2. Kuantum PSO ile yapılan çalışma sonuçları

PSO algoritmasındaki parçacıkların çözümü arama yeteneklerinin geliştirilmesi amacıyla ortaya çıkan bir tekniktir. Yapılan testlerde parçacık sayısının aynı olduğu durumlarda çalışma sürelerinin aynı olmasına karşın çözüm olarak çok daha iyi neticelerin elde edildiği bir algoritmadır. Bu tez çalışmasında elde edilen değerler 100 parçacık ve 100 iterasyon ile elde edilmiştir.

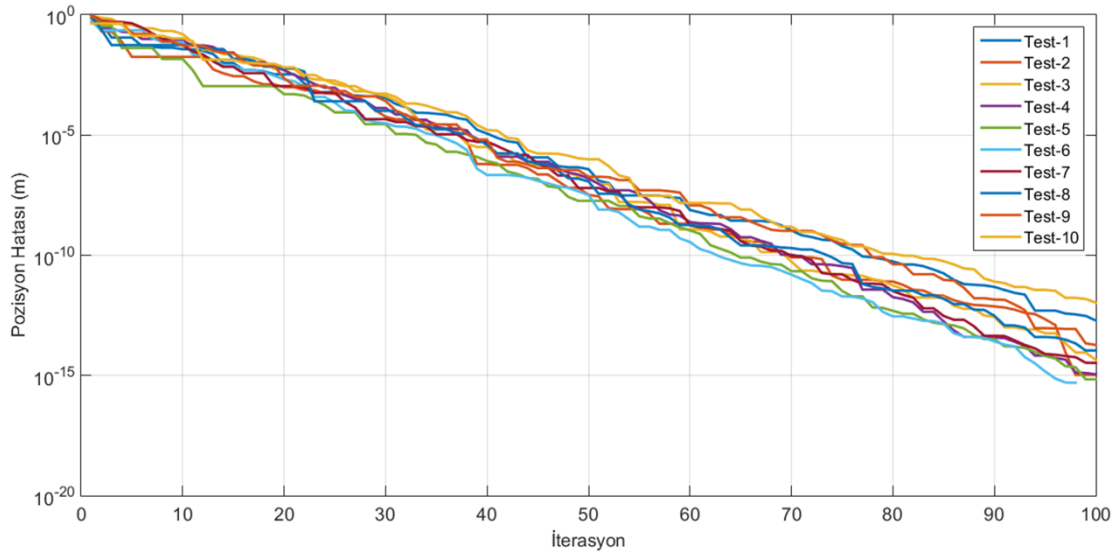
Tablo 4.3. Kuantum PSO çalışma sonuçları

Test No	Çözüm Süresi (s)	Pozisyon Hatası (m)	İterasyon Sayısı	İterasyon Süresi (s)
1	0.2880	4.98e-13	94	0.3071
2	0.2176	8.57e-13	90	0.2561
3	0.2921	8.99e-15	98	0.2976
4	0.2677	1.11e-15	100	0.2677
5	0.2274	6.87e-16	99	0.22294
6	0.2613	4.97e-16	98	0.2666

Tablo 4.3. Kuantum PSO çalışma sonuçları (devamı)

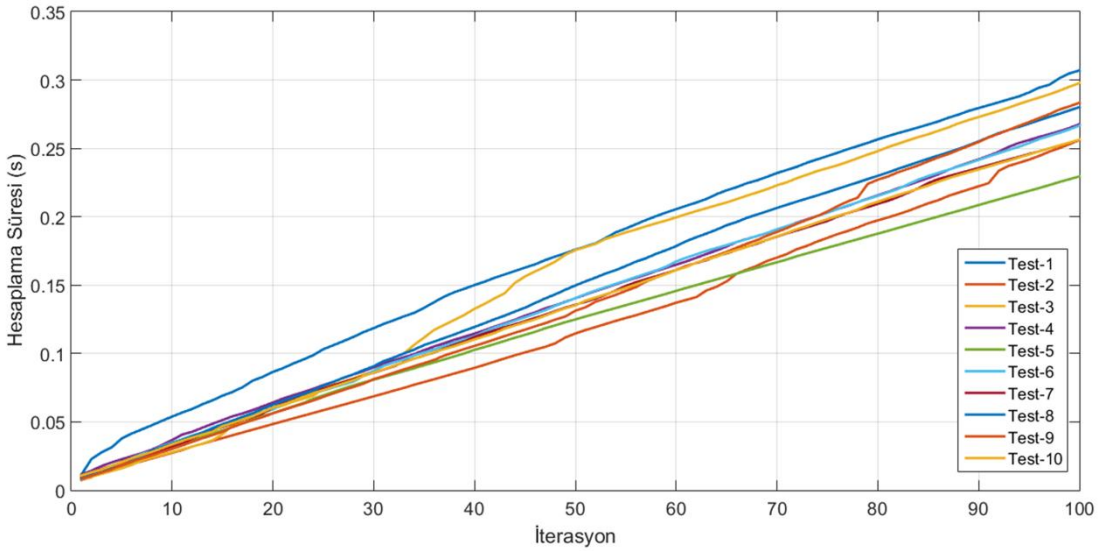
7	0.2562	3.31e-15	100	0.2562
8	0.2777	1.09e-14	99	0.2802
9	0.2833	1.85e-14	100	0.2833
10	0.2563	1.07e-12	100	0.2563
Ortalama	0.2627	2.47e-13	97	0.2694

Tablo 4.3'te Kuantum PSO ile yapılan 10 farklı test sonucu hem pozisyon hatası hemde çözüme ulaşma süresi açısından ortaya konmuştur. Pozisyon hatası değerinin ortalama olarak 10^{-13}'lerde olması elbette önemlidir. Ancak bu değerlerin elde edildiği iterasyon sayılarına bakarak başka bir sonuç daha çıkarılabilir ki oda şudur: İterasyon sayısı arttırıldığında algoritma pozisyon hatası değerlerini daha iyileştirecektir. Bu durum Şekil 4.4'te çok daha net bir şekilde görünmektedir. Çünkü diğer algoritmalara bakıldığında belli bir iterasyon sayısından sonra grafiğin sabit bir değer aldığı açıkça görünmektedir. Burada ise grafik eğimli bir şekilde sonlanmıştır.



Şekil 4.4.Quantum PSO pozisyon hatası sonuçları (m)

Şekil 4.5'te 100 parçacık ile 100 iterasyonun tamamlanma süreleri görünmektedir. PSO algoritmasında olduğu gibi KPSO tekniğinde de sürenin ortalama 0.26 saniyelerde olduğu görünmektedir. Genel itibariyle herbir denemenin sonuçlanma süresi 0.2 ile 0.3 s aralığındadır.



Şekil 4.5. Kuantum PSO çalışma süreleri (s)

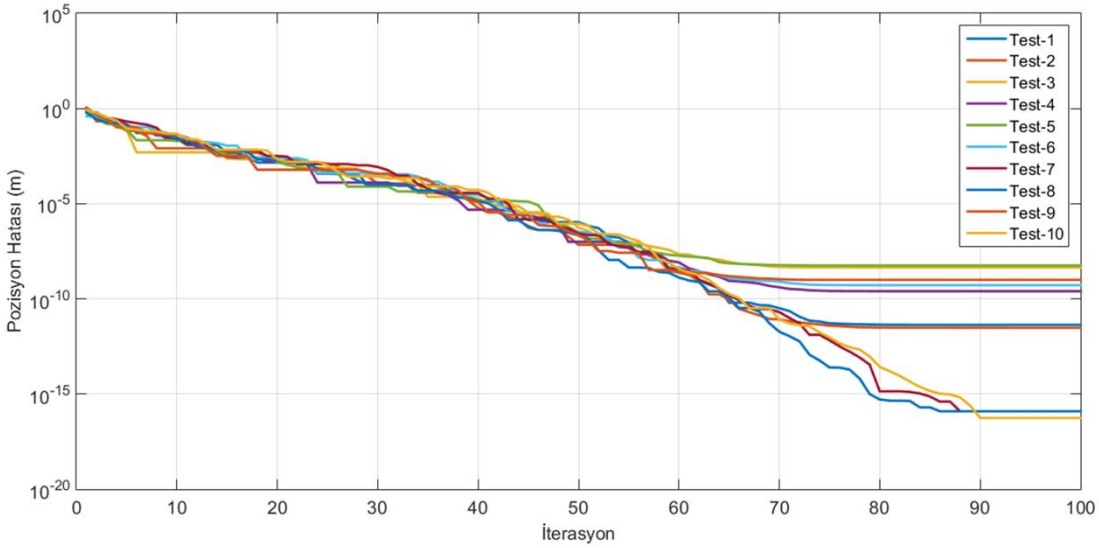
4.1.1.3. RDV PSO tekniği çalışma sonuçları

Bölüm 3'te detaylıca açıklanan bu IW tekniği parçacıkların arama yeteneklerini geliştirmek için her iterasyonda parçacıkların hızlarını rassal olarak belli oranda azaltmaktadır. Böylece büyük adımlarla en iyi çözümü kaçırma ihtimali ortadan kalkmaktadır.

Tablo 4.4.RDV PSO çalışma sonuçları

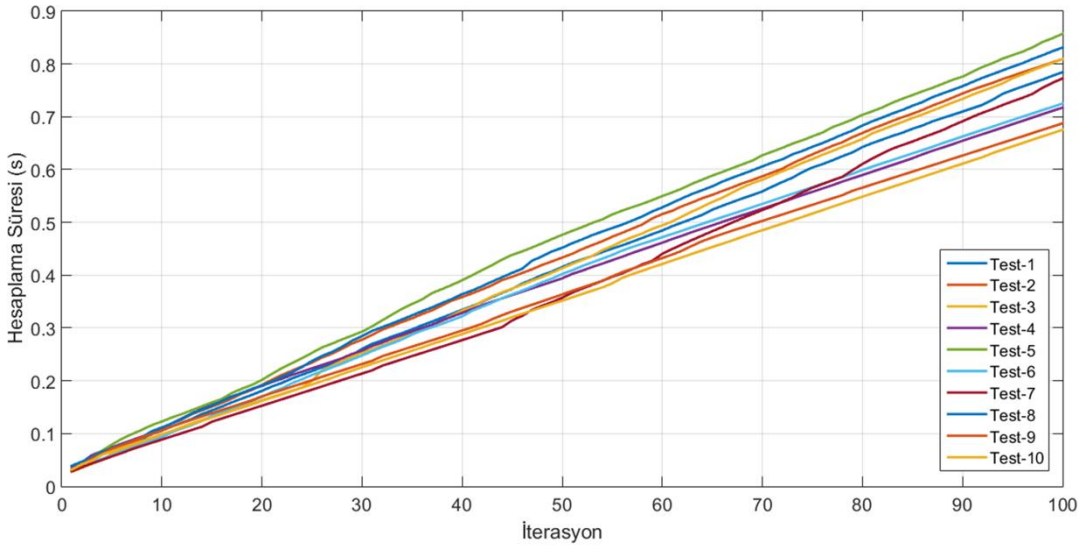
Test No	Çözüm Süresi (s)	Pozisyon Hatası (m)	İterasyon Sayısı	İterasyon Süresi (s)
1	0.7507	4.31e-12	95	0.7845
2	0.7904	3.02e-12	97	0.8094
3	0.6431	4.49e-09	80	0.8093
4	0.5835	2.54e-10	81	0.7174
5	0.6701	5.63-10	78	0.8573
6	0.5858	5.27e-10	80	0.7251
7	0.6747	1.24e-16	90	0.7728
8	0.7130	2.01e-16	86	0.8311
9	0.5394	9.98e-10	78	0.6877
10	0.6112	5.55e-17	90	0.6754
Ortalama	0.6561	6.97e-10	85	0.767

Tablo 4.4'ten de görüneceği üzere parçacıkların arama yeteneklerinin klasik PSO'ya göre iyileştiği ama Quantum PSO'ya göre yetersiz kaldığı görünmektedir. Ayrıca hesaplama sürelerinin biraz daha dağınık olduğu açıktır.



Şekil 4.6.RDV PSO pozisyon hatası (cm)

Şekil 4.6 pozisyon hatası sonuçlarını göstermektedir. RDV tekniği belli oranda PSO algoritmasını açıkça geliştirmiştir. Ortalama 10^{-10}'lara ulaşan en iyi değerle aslında, parçacıkları konumlandıran ve en iyi değere taşıyan hız denkleminde yapılan küçük bir değişiklik ile iyi neticeler alınabileceğini göstermiştir. O nedenle hız parametresi sezgisel algoritmalar içinde kullanılan çok etkili parametrelerden bir tanesidir. Ancak Şekil 4.7'de ise algoritmanın tamamlanma süresi görünmekte sürelerin 0.65 – 0.85 s aralığında olmak üzere burada biraz dağınık olduğu görünmektedir.



Şekil 4.7. RDV PSO hesaplama süresi (s)

4.1.2. Yapay arı kolonisi algoritması ile yapılan çalışma sonuçları

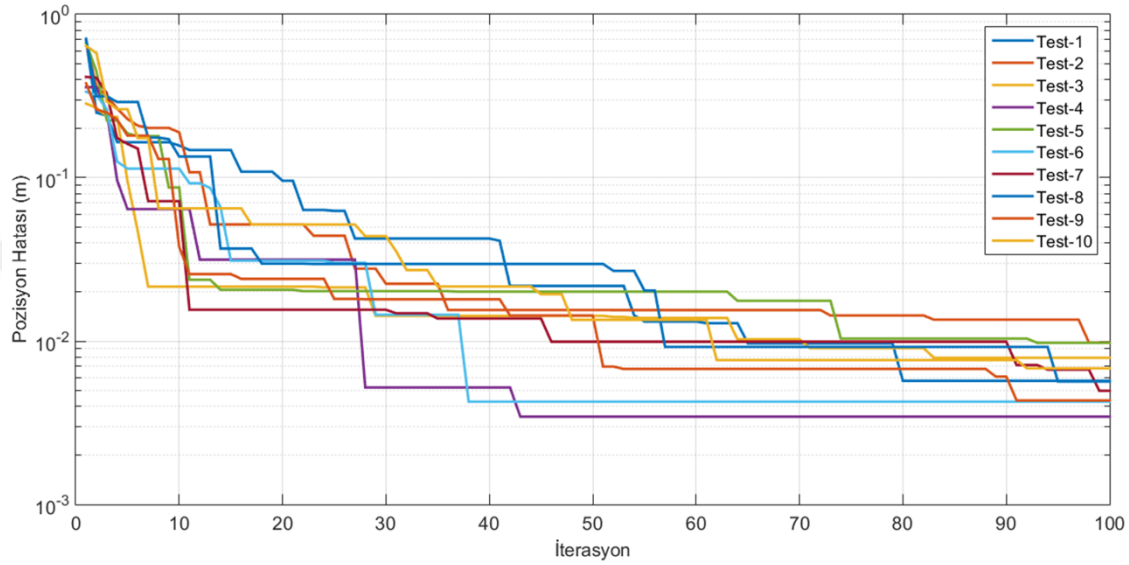
Yapılan benzetim çalışmasında kullanılan arı kolonisi özellikleri şu şekildedir:

- İterasyon sayısı: 100
- Popülasyon sayısı: 100
- Limit parametresi: (Popülasyon*Boyut)/2 = 350

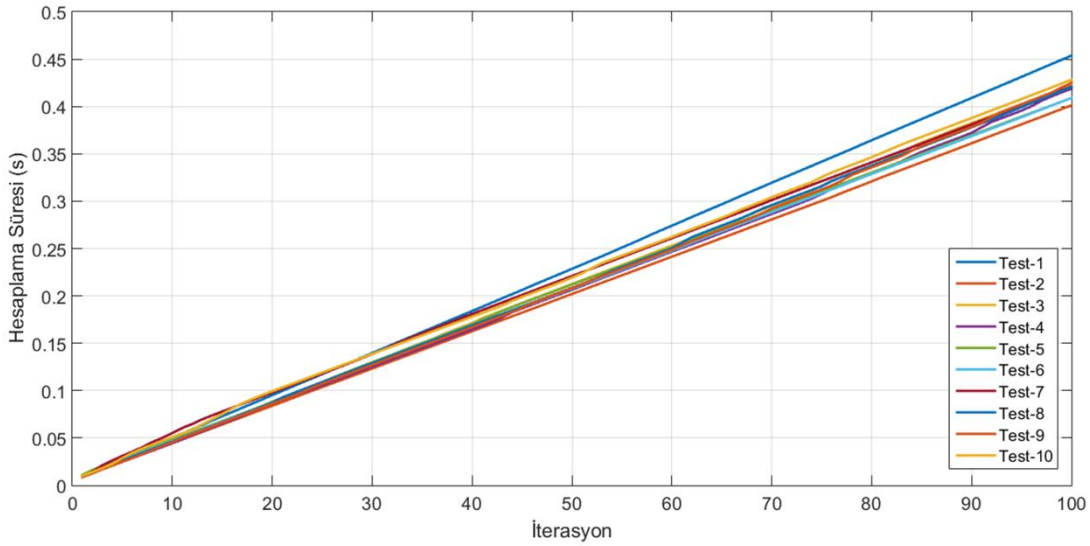
Tablo 4.5. YAK algoritması ile yapılan çalışma sonuçları

Test No	Çözüm Süresi (s)	Pozisyon Hatası (m)	İterasyon Sayısı	İterasyon Süresi (s)
1	0.3643	5.71e-03	80	0.4536
2	0.3931	9.78e-03	98	0.4011
3	0.3511	7.90e-03	83	0.4226
4	0.3778	1.95e-03	91	0.4187
5	0.3814	9.75e-03	93	0.4088
6	0.3526	1.96e-03	86	0.4059
7	0.4170	4.96e-03	98	0.4089
8	0.3993	5.66e-03	95	0.4205
9	0.3827	4.33e-03	91	0.4248
10	0.3958	6.81e-03	92	0.4281
Ortalama	0.3815	5.88e-03	90	0.4193

YAK algoritmasıyla yapılan 10 adet çalıştırma neticesinde elde edilen sonuçlar Tablo 4.5'de görünektedir. Burada "Çözüm Süresi" minimum pozisyon hatasının elde edilmesi için geçen süreyi, "İterasyon Sayısı" ise bulunan minimum pozisyon hatasının elde edildiği iterasyon değerini göstermektedir. En son sütunda yer alan "İterasyon Süresi" ise 100 iterasyon için algoritma tarafından harcanan süreyi göstermektedir.



Şekil 4.8. YAK algoritması ile testlerin elde ettiği pozisyon hatası değerleri (m)



Şekil 4.9.YAK algoritması ile yapılan testlerin hesaplama süresi (s)

Sonuçlara 100 iterasyonun tamamlanma süresi açısından bakıldığında bütün sonuçların birbirine çok yakın olduğu görünmektedir. Pozisyon hatası olarak sonuçlar yorumlanacak

olursa 10^{-3} 'lerde değerlerin algoritma tarafından bulunduğu görünmektedir. Çözüme ulaşma süresi açısından sonuçları yorumlamak gerektiğinde ise dokuzuncu test sonucunun en iyi değeri 0.3511 s ve 7.90e-03 cm olarak ortaya çıktığı görünmektedir.

Yapılan teslerin grafikleri pozisyon hatası ve hesaplama süresi olarak elde edilmiştir. Şekil 4.8'de pozisyon hatası değerleri görünmektedir. Çok büyük bir dağılım olmadığı grafiklerde de göze çarpmaktadır. Şekil 4.9'te her bir test işleminin tamamlanması için geçen süre görünmektedir. Her bir test işleminin çok yakın zaman aralıklarında işlemini tamamladığı açıkça görünmektedir. Tek bir deneme dışında kalan diğer tüm denemelerin 0.4 – 0.45 s aralığında kümelendiği açıkça görünmektedir.

4.1.3. Ateşböceği sürü algoritması ile yapılan çalışma sonuçları

Algoritmada kullanılan sabit parametrelere atanan değerler şu şekildedir:

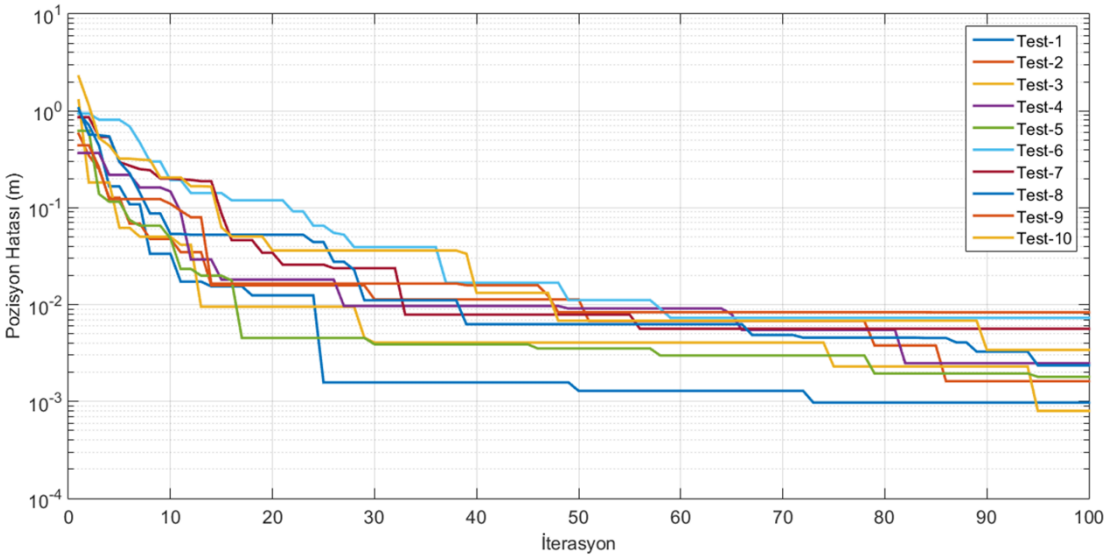
- Gama: 1
- Beta: 2
- Alfa: 0,3
- Popülasyon: 50
- İterasyon: 100

Tablo 4.6, ateşböceği sürü algoritması kullanılarak elde edilen sonuçları göstermektedir. Pozisyon hatası sonuçlarının 10^{-3} ve 10^{-4} arasında dağıldığını söylemek yanlış olmayacaktır. Zaten ortalama pozisyon hatası değerine bakıldığında 10^{-3} olarak karşımıza çıkmaktadır. Her ne kadar Tablo 4.6'da elde edilen test sonuçları 10^{-3} değeri daha fazla olsada 10^{-4} değeri testlerde olduğu sürece bu iki değer arasında dağılım olduğu kabul edilebilir. Şekil 4.10 algoritmanın pozisyon hatası çözümlerini göstermektedir. Değerlerin birbirine yakın olması bu konuda algoritmanın kararlılığını açıkça göstermektedir.

Tablo 4.6. Ateşböceği sürü algoritması çalışma sonuçları

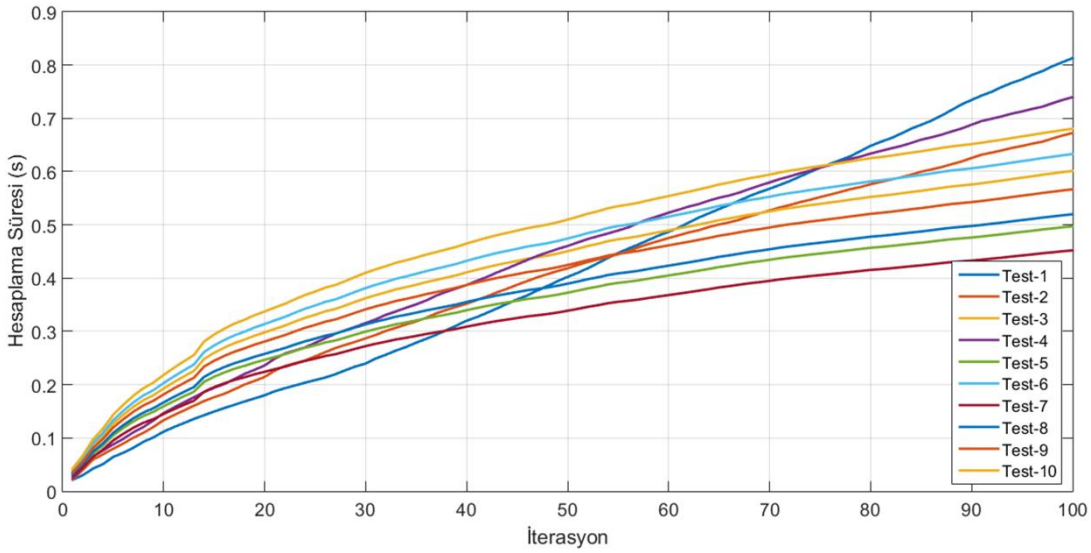
Test No	Çözüm Süresi (s)	Pozisyon Hatası (m)	İterasyon Sayısı	İterasyon Süresi (s)

1	0.5908	9.74e-04	75	0.8132
2	0.6039	1.62e-03	86	0.6725
3	0.5887	8.01e-04	95	0.6013
4	0.6435	2.48e-03	82	0.7397
5	0.4869	1.79e-03	95	0.4973
6	0.6041	3.17e-03	89	0.6330
7	0.4260	3.64e-03	86	0.4521
8	0.5090	2.36e-03	95	0.5199
9	0.5315	8.33e-03	85	0.5667
10	0.6516	3.41e-03	90	0.6805
Ortalama	0.5636	2.85e-03	87	0.6176



Şekil 4.10. Ateş böceği sürü algoritması pozisyon hatası (m)

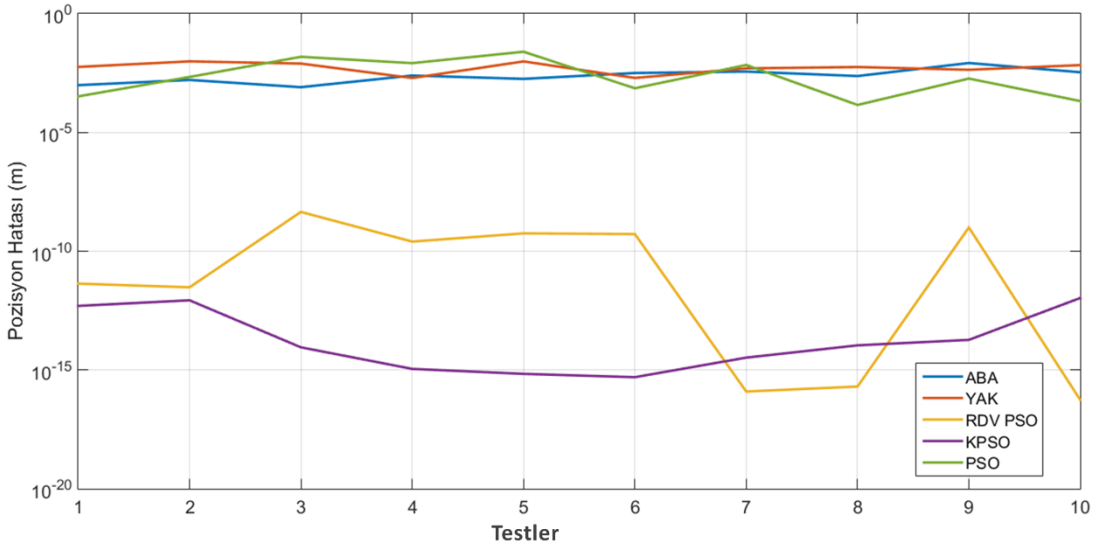
Şekil 4.11'de hesaplama süresi grafiğine bakıldığında ise algoritmanın tamamlanma süresi açısından durumun daha dağınık olması göze çarpmaktadır. Algoritmanın tamamlanması için geçen sürenin 0.4 s ile 0.85 s arasında olduğu açıkça görünmektedir.



Şekil 4.11. Ateş böceği sürü algoritması hesaplama süresi (s)

4.2. Pozisyon Hatası Karşılaştırması

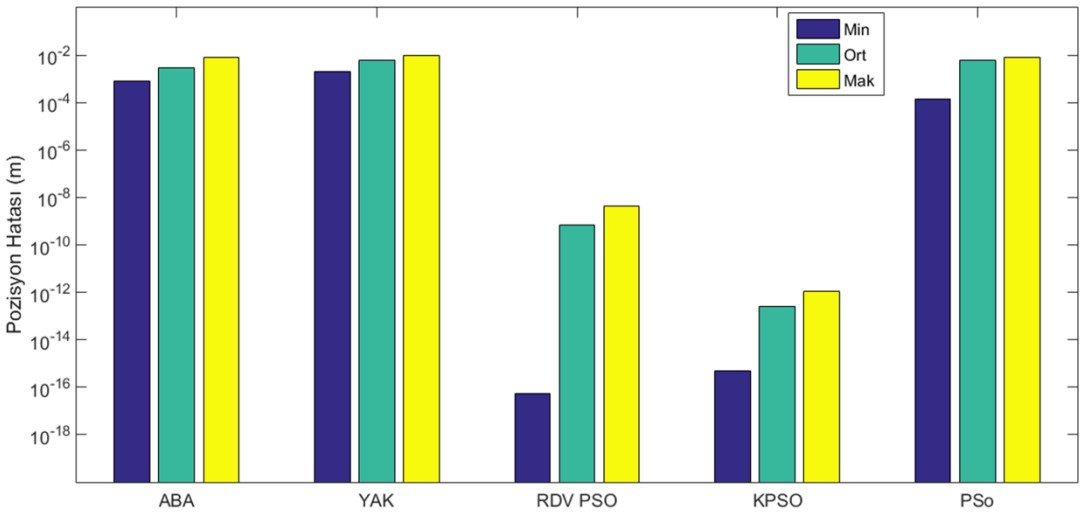
Şekil 4.12, geleneksel PSO, Kuantum PSO, RDV PSO, ateş böceği ve yapay arı kolonisi için 10 adet testten elde edilen sonuçları göstermektedir. Pozisyon hatası açısından en iyi değerlerin RDV PSO ile Kuantum PSO tarafından elde edildiğini göstermektedir.



Şekil 4.12. Tüm algoritmalar için pozisyon hatası karşılaştırması (m)

Her ne kadar PSO, YAK ve FA algoritmaları için elde edilen değerlerin RDV PSO ve Quantum PSO'ya göre daha kötü değerler ortaya koymasına karşın bu algoritmaların

(PSO, YAK ve FA) elde ettiği değerlerin birbirine yakın olduğu açıkça görünmektedir. Hâlbuki RDV PSO ile Kuantum PSO algoritmaları ile elde edilen değerlerin arasındaki farklarda göze çarpmaktadır.

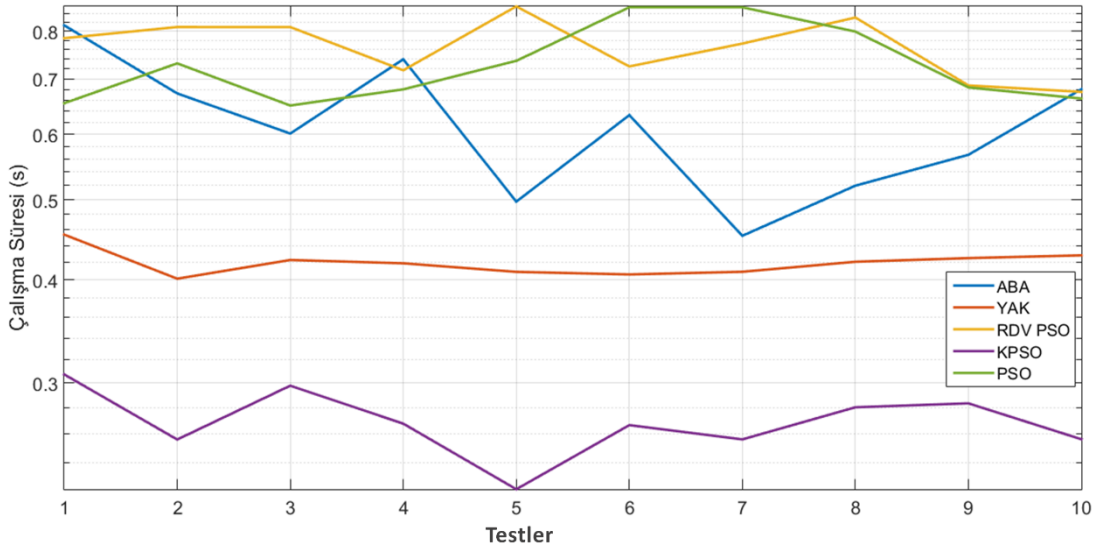


Şekil 4.13. Algoritmalara ait ortalama, maksimum ve minimum pozisyon hatası (m) değerleri

Şekil 4.13'te tez çalışmasında kullanılan beş algoritmayla elde edilen on adet teste ait ortalama, minimum ve maksimum değerler grafiği görünmektedir. Minimum değerler grafikte en soldaki bar yani "lacivert" renkte gösterilmekte olup algoritmayla elde edilen 10 testin en iyi değerini ortaya koymaktadır. Her bir grafiğin en solundaki bar (sarı renkli) ise 10 testin en kötü değerlerini, ortadaki grafik bar (yeşil renkli) ise ortalama değeri göstermektedir. Her bir algoritma için ortalama değerin en kötü yani maksimum değere yakın olduğu açıkça görünmektedir.

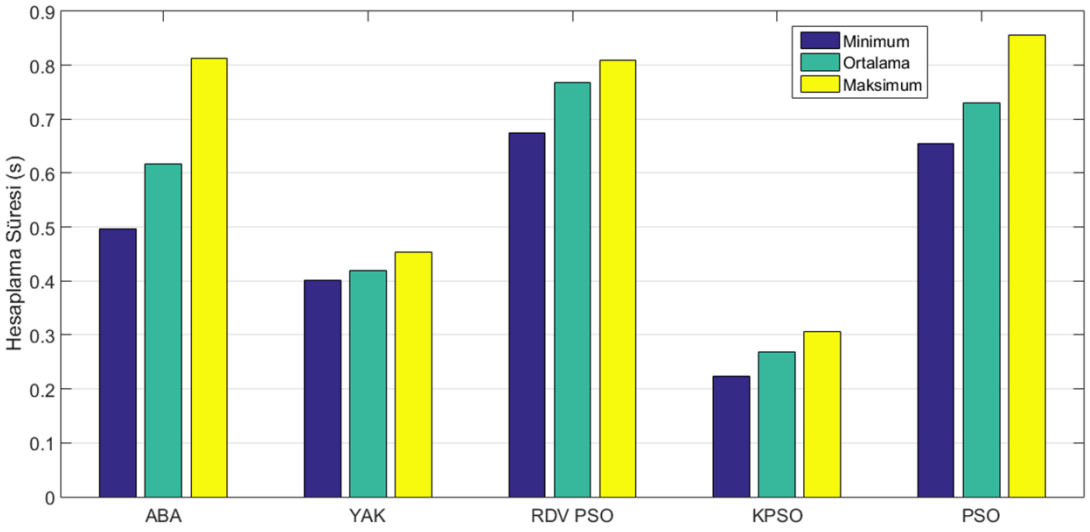
4.3. Çalışma Zamanı Karşılaştırması

Herbir algoritmaya ait 10 testin hesaplanma sürelerine dair karşılaştırmalı grafik Şekil 4.14'te görünmektedir. Grafiğe göre PSO, RDV PSO ve YAK algoritmaları, testlerin tamamında birbirine yakın sürelerde işlemi tamamlamışlardır. Ancak FA (Ateş böceği sürüsü) ile elde edilen hesaplama süresine bakıldığında ise en iyi süre ile en kötü süre arasında hayli süre farkı olduğu açıkça görünmektedir. Çalışma ve iterasyon süreleri açısından en iyi değerlerin Kuantum PSO tarafınan elde edildiği açıktır.



Şekil 4.14.Tüm algoritmalar için hesaplama süresi (s) karşılaştırması

En iyi hesaplama süresi ile en kötü hesaplama sürelerinin karşılaştırmalı grafiği ise Şekil 4.15'te verilmiştir. PSO ve KPSO algoritmalarında ortalama değer minimum ve maksimum değere son derece yakındır.



Şekil 4.15.Algoritmalara ait ortalama, maksimum ve minimum hesaplama süresi (s) değerleri

YAK algoritmasında ise ortalama değer minimum değere çok daha yakındır. Bu da göstermektedir ki değerler genelde minimum değere daha yakındır. Hâlbuki FA ve RDV PSO algoritmalarında ortalama değerin ne minimum değere ne de maksimum değere yakın değildir. Bu da göstermektedir ki, hesaplama süreleri minimum ve maksimum değer arasında dağılıma sahiptir. Zaten bu Şekil 4.14'te açıkça görünmektedir.

4.4. Popülasyon ve iterasyon sayısı karşılaştırması

Tablo 4.7'de bu çalışmada kullanılan yazılımsal sezgisel tekniklerde kullanılan bazı parametreler ve algoritmaların çıktı değerleri görünmektedir.

Tablo 4.7. Çalışmada kullanılan algoritmaların süre ve pozisyon hatası açısından karşılaştırması

	FA	YAK	RDV PSO	KPSO	PSO
Popülasyon Sayısı	50	100	100	100	100
İterasyon Sayısı	100	100	100	100	100
Ortalama Hesaplama Süresi (s)	0.6176	0.4193	0.7671	0.2694	0.7309
Ortalama Pozisyon Hatası (m)	2.85e-03	5.88e-03	6.94e-10	2.47e-13	6.06e-03

Sürü zekâsına dayalı olarak çalışan algoritmaların tamamında sürü sayısı ve iterasyon sayısı en iyi çözüme yaklaşmada önemli bir parametredir. Ancak bu değerlerin fazla olması daha iyi bir çözüm elde etmesine karşın algoritmanın hesaplama süresine negatif oranda etki etmekte ve süreyi uzatmaktadır. Bu tez çalışmasında çalışılan algoritmaların ilgili parametreleri ve elde edilen ortalama hesaplama süresi ile ortalama pozisyon hatası Tablo 4.7'de gösterilmektedir.

Literatürde ters kinematik probleminin çözümü hususunda sezgisel optimizasyon algoritmalarının birbirleriyle karşılaştırılması sıklıkla karşılaşılan bir durumdur. Tablo 4.8'de seçilen bazı çalışmalarda yapılan sezgisel algoritma karşılaştırmaları görünmektedir. Bu tez çalışmasında da bu konu derinlemesine çalışılmıştır. Ancak literatürdeki çalışmalardan en önemli farkı RDV PSO isminde yeni bir teknikle güçlendirilen PSO algoritmasının kulllanılması ve sezgisel algoritmaların donanımsal olarak gerçekleştirilmesidir. Çünkü bu çalışmada elde edilen hesaplama süresinin gerçek

bir robotta uygulanabilirliği sorgulanmış yazılımsal olarak elde edilen hesaplama sürelerinin bu durumdan uzak kaldığı ortaya konmuştur. Bu duruma çözüm üretmek adına PSO algoritması donanımsal olarak gerçekleştirilerek elde edilen sürenin bu duruma çok daha uygun olduğu açıkça görünmektedir. Ayrıca test amacıyla kullanılan robot manipülatörünün 7-eklemli olmasıda önceki çalışmalardan bu tez çalışmasını ayıran önemli bir noktadır.

Tablo 4.8. Literatürde çalışılan sezgisel algoritma karşılaştırmaları

Araştırma	DOF	Kullanılan Algoritma	Karşılaştırılan Algoritma	Karşılaştırma Parametreleri
Çavdar ve Alavi, 2012, p. 332	6-DOF	Bee	PSO	
		6.31e-13	3.32e-08	Hata (m)
		0.0286	0.0361	Süre (s)
Rokbani, Casals ve Alimi, 2015, p. 562	3-DOF	10 Ateş böceği	60 Ateş böceği	
		1.27e-17	1.78e-18	Hata (m)
		1.21e-03	7.15e-03	Süre (s)
Durmuş ve Timurtaş, 2011	6-DOF	HSA	PSO	
		2.31e-14	3.32e-08	Hata (m)
		0376	0.0361	Süre (s)
Ayyıldız ve Çetinkaya, 2015, p. 833	4-DOF	PSO	GA	
		7.70e-06	3.96e-04	Hata (m)
		0.0196	0.1753	Süre (s)
Rokbani ve Alimi, 2013, p. 1608	2-DOF	IW-PSO	PSO	
		6.65e-05	4.11e-03	Hata (m)
		4024	4887	İterasyon

Tablo 4.8. Literatürde çalışılan sezgisel algoritma karşılaştırmaları (devamı)

Sherbiny, Elhosseini ve Haikal, 2017, p. 2545	6-DOF	Adaptive Neuro Fuzzy Inference System	GA	
		5.426e-03		

		0.0380	7.64e-04	Hata (m)
			83.1239	Süre (s)
Küçük ve Bingül, 2014, p. 1996	6-DOF	NIKA (New Inverse Kinematics Algorithm)	Newton-Raphson Algorithm	
		1.02e-04	0.2266	Hata (m)
		0.327456	0.331493	Süre (s)
Dereli ve Köker, 2017, pp. 82-83	7-DOF	IW-PSO	PSO	
		3.64e-03	9.13e-03	Hata (m)
		2846	4773	İterasyon
Shi ve Xie, 2017	6-DOF	Adaboost Yapay Sinir Ağı		
		0.00267		Hata (m)
		0.3		Süre (s)
Sherbiny, Elhoseini ve Haikal, 2018, pp 32-34	5-DOF	Yeni Değişkenli ABC	Geleneksel ABC	
		1.42e-08	5.68e-03	Hata (m)
		0.428	0.446	Süre (s)
Mahanta et al., 2019, p. 528	6-DOF	ABA	IWO	
		1.65e-15	5.98e-05	Hata (m)
		2.9	3.9	Süre (s)

4.5. FPGA Simülasyon Sonuçları

Bu bölümde FPGA ile gerçekleştirilen donanım temelli parçacık sürü optimizasyonu algoritmasının simülasyon sonuçları irdelenmiştir. Sentezlenebilir özelliği olan bu sayısal devre Vivado arabirimi 2017.4 sürümünde VHDL donanım tasarım dili kullanılarak gerçekleştirilmiştir. Ayrıca bu sayısal tasarım 20, 30, 50 ve 100 popülasyon ile ayrı ayrı test edilerek sonuçları ortaya konmuştur. Nihayetinde elde edilen eklem açıları ile ortaya çıkan 7-eklemli robot manipülatörü oryantasyonu RoboAnalyzer yazılımı vasıtasıyla

gösterilmiştir (Gupta et al, 2017). Tasarımlarda kullanılan parametreler ve değerleri şu şekildedir:

- İterasyon sayısı= 50 (20 parçacıkta 100 iterasyon)
- c_1=1.4, c_2= 1.4 ve w= 0,7

Tasarımlarda rassal sayılar için LFSR kullanılmıştır ve her iterasyonda kullanılacak rassal değerler başlangıçta üretilmektedir. Benzer şekilde PSO için başlangıç eklem açılarıda rassal olarak üretilmiştir. Açı değerlerinin atanmasında 16-bit kullanılmıştır. Bu 16-bitin radyana dönüşümünde en değerli bit işaret biti, sonraki 2-bit tamsayı ve geri kalan 13-bit ise ondalık kısmı temsil etmektedir. Hem rasgele sayılarda hem de eklem açılarında rassallığın başlangıç değerleri şu şekildedir:

Tablo 4.9. Rassal sayıların başlangıç değerleri

Rnd_1	Rnd_2	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
c526	8207	1921	f7a0	02ca	fd36	0b2b	f4d5	0860

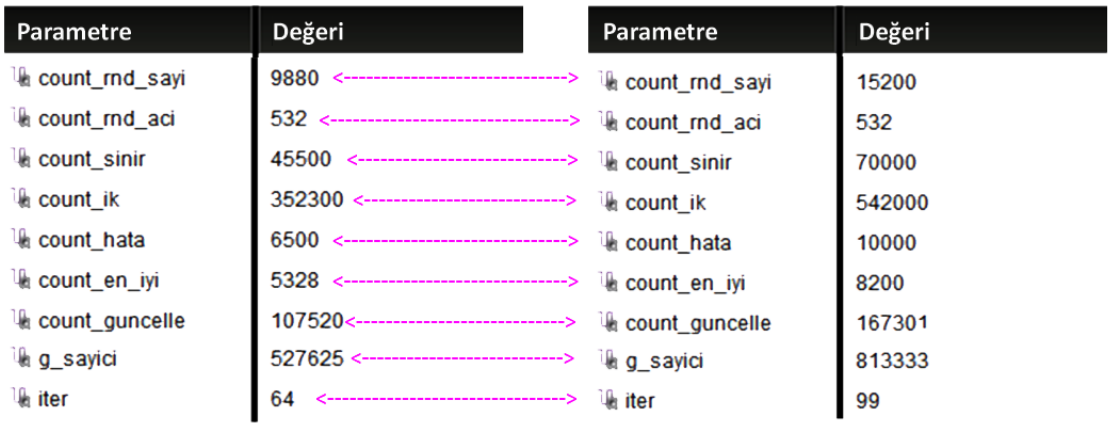
Sonuçlar pozisyon hatası ve çözüme ulaşma süresi açısından karşılaştırılmıştır. FPGA'da süre doğrudan kart üzerindeki çipin frekans değeri ile ilgili olduğundan dolayı işlemlerdeki saat darbesi sayısıda ayrıca sürenin yanında ifade edilmiştir. Çünkü bu çalışmada kullanılan Nexys 4 DDR kartı 450 Mhz saat frekansına sahip olduğundan dolayı tasarımlar 3ns'lik saat darbeleri ile çalıştırılmıştır. Eğer kullanılan kart daha fazla frekans değerine sahipse elbette çalışma süreside o oranda düşecektir.

Gerçekleştirilen tasarım "Rnd Sayılar", "Rnd Açılar", "Açı Sınırları", "İleri Kinematik", "Pozisyon Hatası", "En İyi Değerler" ve "Açıları Güncelle" olmak üzere yedi bölümden oluşmaktadır. Sonuçlar analiz edilirken her bir bölüm için kullanılan süreler ile birlikte saat darbesi sayısıda özellikle belirtilmiştir. Çünkü FPGA'larda aslında süreyi oluşturan unsur devrenin çalıştığı saat darbesidir.

4.5.1. Yirmi parçacık için simülasyon sonuçları

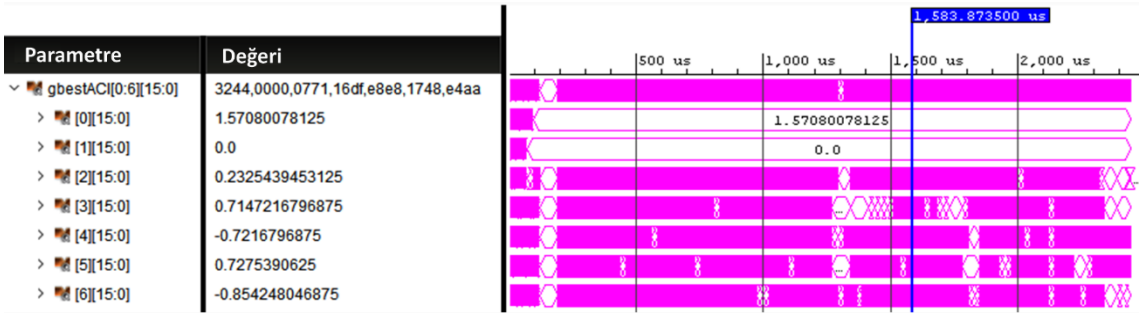
Bu test aşamasında 100 iterasyon kullanılmış olup her iterasyonda elde edilen pozisyon hatası değerleri Şekil 4.16'da görünmektedir. Şekildende göründüğü üzere hata değeri en son iterasyonda olmak üzere sürekli düşme eğilimindedir. Ancak bu düşüşler 10^{-3}'lerde kaldığından dolayı Şekil 4.16'da 64.iterasyon işaretlenmiştir. Bu iterasyon değerinde elde edilen pozisyon hatası değeri 9,76e-03 cm iken son iterasyonda elde edilen oran 2,58e-03 cm olarak görünmektedir.

Şekil 4.16. 20 parçacık için pozisyon hatası değerleri



Şekil 4.17.Alt devrelerin kullandıkları saat darbeleri sayısı

Çözüm ve algoritmanın tamamlanma süresini belirleyen saat darbesi sayıları ise Şekil 4.17'de görünmektedir. Hem tüm devrenin hemde alt devrelerin kullandıkları saat darbeleri burada görünmektedir. Şekilde sol sütun çözüme ulaşma sağ sütun ise algoritmanın tamamlanma saat darbesi sayılarını göstermektedir. En iyi değer olarak seçilen 64. İterasyonda elde edilen eklem açıları Şekil 4.18'de görünmektedir. Bu şekilde aynı zamanda 64.iterasyonda kullanılan sürede 1,583 ms olarak görünmektedir.



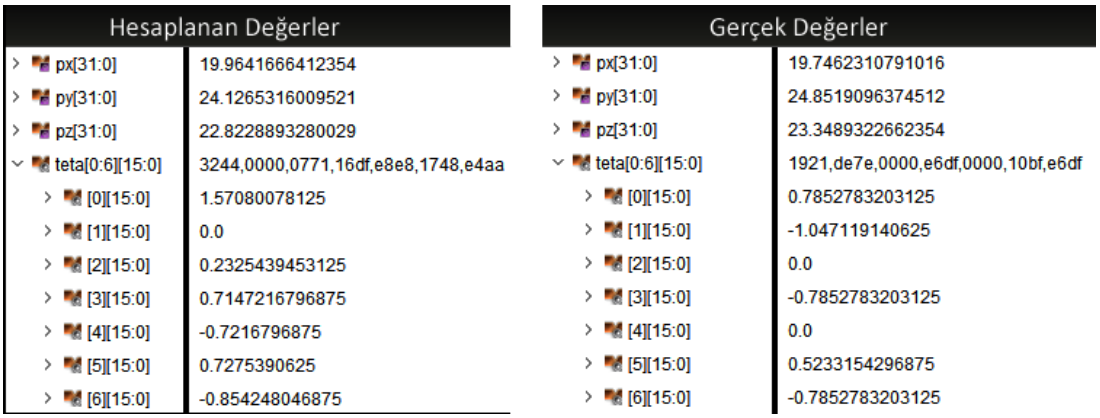
Şekil 4.18.64. iterasyonda elde edilen eklem açıları

20 parçacıkla yapılan test işleminde elde edilen eklem açıları Tablo 4.10'da görünmektedir.

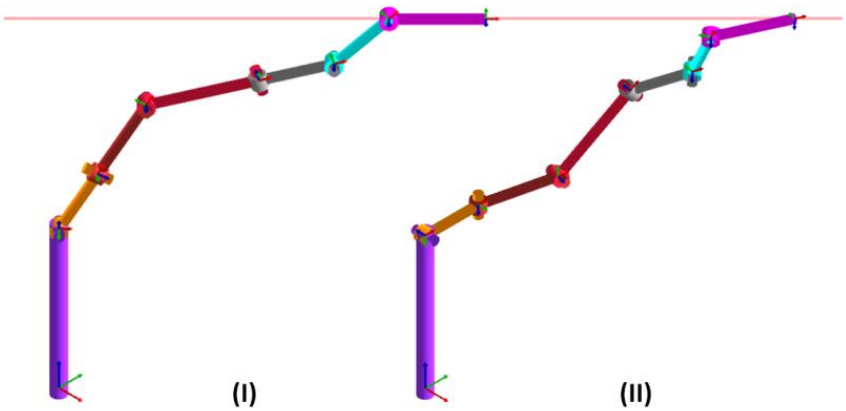
Tablo 4.10. Yirmi parçacık testinde elde edilen eklem açıları

Eklem Açıları	Onaltılık Taban	Onluk Radyan	Onluk Derece (°)	Gerçek Değerler (°)
θ_1	3244	1.5708	90	45
θ_2	0000	0.0	0	-60
θ_3	0771	0.2325	13,32	0
θ_4	16df	0.7147	40,95	-45
θ_5	e8e8	-0.7216	-41.34	0
θ_6	1748	0.7275	41.68	30
θ_7	e4aa	-0.8542	-48.94	-45

Tablo 4.10'daki eklem açıları ile robot koluna ait uç elemanın konum bilgileri hesaplandığında ise Şekil 4.19'daki bilgiler elde edilecektir.



Şekil 4.19.Robot kolu uç elemanına ait konum bilgileri



Şekil 4.20.Elde edilen robot kolu oryantasyonu

4.5.2. Otuz parçacık için simülasyon sonuçları

İterasyon	Değer
gbestErr[0:49][31:0]	3f7eb852,3e2b0c80,3dcfd
[0][31:0]	0.995000004768372
[1][31:0]	0.16703987121582
[2][31:0]	0.101491667330265
[3][31:0]	0.0853753536939621
[4][31:0]	0.0409483797848225
[5][31:0]	0.0153231024742126
[6][31:0]	0.0153231024742126
[7][31:0]	0.0153231024742126
[8][31:0]	0.0153231024742126
[9][31:0]	0.0153231024742126
[10][31:0]	0.0153231024742126
[11][31:0]	0.0153231024742126
[12][31:0]	0.0153231024742126
[13][31:0]	0.0153231024742126
[14][31:0]	0.013250358402729
[15][31:0]	0.0105502521619201

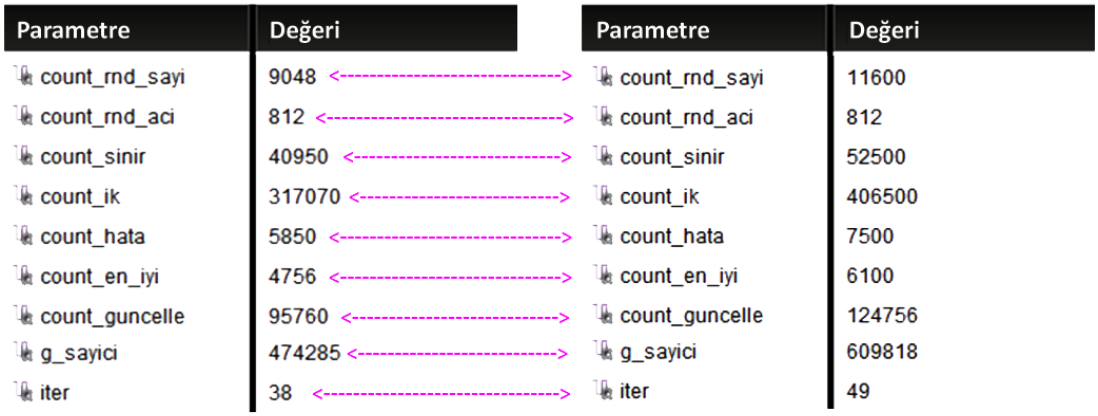
İterasyon	Değer
[16][31:0]	0.00965658109635115
[17][31:0]	0.00965658109635115
[18][31:0]	0.00965658109635115
[19][31:0]	0.00838207267224789
[20][31:0]	0.00663380417972803
[21][31:0]	0.00490683456882834
[22][31:0]	0.00393201364204288
[23][31:0]	0.0024317535571754
[24][31:0]	0.00140412990003824
[25][31:0]	0.000551203615032136
[26][31:0]	0.000348789355484769
[27][31:0]	0.000348789355484769
[28][31:0]	0.000287785631371662
[29][31:0]	0.000258223182754591
[30][31:0]	0.000255980470683426
[31][31:0]	0.000109667191281915
[32][31:0]	0.000102688958577346

İterasyon	Değer
[33][31:0]	6.78888827678747e-05
[34][31:0]	2.38942375290208e-05
[35][31:0]	2.38942375290208e-05
[36][31:0]	2.38942375290208e-05
[37][31:0]	1.90935224964051e-05
[38][31:0]	5.37909227205091e-06
[39][31:0]	5.37909227205091e-06
[40][31:0]	5.37909227205091e-06
[41][31:0]	5.37909227205091e-06
[42][31:0]	5.37909227205091e-06
[43][31:0]	5.37909227205091e-06
[44][31:0]	5.37909227205091e-06
[45][31:0]	5.37909227205091e-06
[46][31:0]	5.37909227205091e-06
[47][31:0]	5.37909227205091e-06
[48][31:0]	5.37909227205091e-06
[49][31:0]	5.37909227205091e-06

Şekil 4.21.Her iterasyon sonucu oluşan pozisyon hatası (cm)

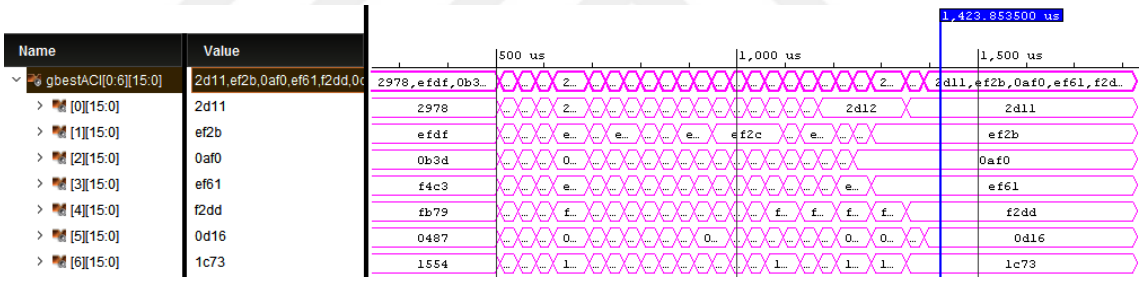
Her bir iterasyonda elde edilen pozisyon hatası değerleri Şekil 4.21'de görünmektedir. 38. iterasyonda elde edilen 5,37e-06 değeri en küçük pozisyon hatası olarak hesaplanmıştır.

Şekil 4.22'de sayısal devredeki her bir alt devrenin tamamlanması için sayılan saat darbesi sayıları görünmektedir. "g_sayici" parametresi genel sayıcı olup her bir alt devrenin aslında toplamı mahiyetinde işlev görmektedir. 38. iterasyonda en iyi değer elde edilmiş olduğundan "g_sayici" parametresi 3 ns ile çarpıldığında geçen süre hesaplanmaktadır. Dolayısıyla 1,42 ms'de en küçük pozisyon hatası değeri elde edilmiştir. Algoritmanın tamamı için geçen süre ise 1,83 ms'dir.



Şekil 4.22.En küçük pozisyon hatasının ve tüm algoritmanın saat darbesi sayısı

Şekil 4.23'de en küçük pozisyon hatasını sağlayan eklem açıları 16-bitlik olarak radyan cinsinden görünmektedir. Ayrıca açı değerlerinin güncellemeler neticesinde hangi değerleri alarak oluştuğuda kısmen görünebilmektedir. Şekil üzerinde sarı ile işaretli çizgi en küçük pozisyon hatasının elde edildiği iterasyonu ve geçen süreyi 1,423 ms olarak göstermektedir.



Şekil 4.23.Minimum pozisyon hatasını sağlayan eklem açıları

Elde edilen eklem açılarının hem onaltılık tabandaki değerleri hemde onluk tabandaki karşılıkları radyan ve derece cinsinden dönüşümleri Tablo 4.11'da görünmektedir.

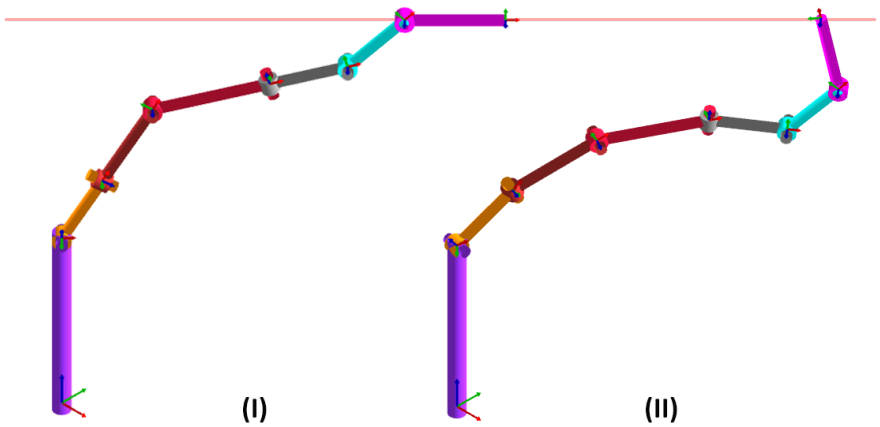
Tablo 4.11. 30 parçacık için elde edilen eklem açıları ve dönüşümleri

Eklem Açıları	Onaltılık Taban	Onluk Radyan	Onluk Derece (°)	Gerçek Değerler (°)
θ_1	2d11	1.4083	80.73	45
θ_2	ef2b	-0.5260	-30.15	-60
θ_3	0af0	0.3417	19.58	0
θ_4	ef61	-0.5194	-29.77	-45
θ_5	F2dd	-0.4105	-23.53	0
θ_6	0d16	0.4089	23.44	30
θ_7	1c73	0.8890	50.96	-45

Hesaplanan Değerler		Gerçek Değerler	
px[31:0]	19.7517967224121	px[31:0]	19.7462310791016
py[31:0]	24.8445873260498	py[31:0]	24.8519096374512
pz[31:0]	23.3571357727051	pz[31:0]	23.3489322662354
teta[0:6][15:0]	2d11,ef2b,0af0,ef61,f2dd,0d16,1c73	teta[0:6][15:0]	1921,de7e,0000,e6df,0000,10bf,e6df
[0][15:0]	1.4083251953125	[0][15:0]	0.7852783203125
[1][15:0]	-0.5260009765625	[1][15:0]	-1.047119140625
[2][15:0]	0.341796875	[2][15:0]	0.0
[3][15:0]	-0.5194091796875	[3][15:0]	-0.7852783203125
[4][15:0]	-0.4105224609375	[4][15:0]	0.0
[5][15:0]	0.408935546875	[5][15:0]	0.5233154296875
[6][15:0]	0.8890380859375	[6][15:0]	-0.7852783203125

Şekil 4.24. 30 parçacık için kullanılan gerçek ve hesaplanan değerler

Benzer şekilde en iyi pozisyon hatası değerini veren eklem açıları ve bu eklem açıları ile manipülatöre ait uç elemanın konumu Şekil 4.24'de simülasyon ekranında görünmektedir. Bu koşullarda her bir ekleme Tablo 4.11'da görünen açılar verildiğinde robot manipülatörü Şekil 4.25'daki oryantasyona sahip olmaktadır.



Şekil 4.25. Robot manipülatörü oryantasyon

Şekil 4.23'de elde edilen eklem açıları sonucu ortaya çıkan uç elemanın pozisyon değerlerinin gerçek değerlere ne kadar yakın olduğu açıkça görünmektedir. Şekil 25'de bu durum simülasyonda daha açık görünmektedir. Şekilde (I), önceden belirlenen eklem açılarıyla oluşan oryantasyonu, (II) ise hesaplanan eklem açılarıyla oluşan oryantasyonu göstermektedir.

4.5.3. Elli parçacık için simülasyon sonuçları

50 parçacık için 50 döngülü test işleminde elde edilen sonuçlar Şekil 4.26'de görünmektedir. 43. İterasyonda en küçük değer 5,25e-06 olarak elde edilmiş ve bu değer 50. İterasyona kadar tekrar edilmiştir.

İterasyon	Değer
gbestErr[0:49][31:0]	3f7eb852,3e2b0c80,3d9c878c
[0][31:0]	0.995000004768372
[1][31:0]	0.16703987121582
[2][31:0]	0.0764304101467133
[3][31:0]	0.0681813433766365
[4][31:0]	0.0457709133625031
[5][31:0]	0.0191821567714214
[6][31:0]	0.01419922336936
[7][31:0]	0.0093600545078516
[8][31:0]	0.0093600545078516
[9][31:0]	0.0093600545078516
[10][31:0]	0.0093600545078516
[11][31:0]	0.0093600545078516
[12][31:0]	0.00787456706166267
[13][31:0]	0.00584944803267717
[14][31:0]	0.00484284851700068
[15][31:0]	0.00454561458900571

İterasyon	Değer
[16][31:0]	0.00408201105892658
[17][31:0]	0.0038016629405319 7
[18][31:0]	0.00346412020735443
[19][31:0]	0.00320704560726881
[20][31:0]	0.00251416629180312
[21][31:0]	0.0020534242503345
[22][31:0]	0.00171287555713207
[23][31:0]	0.00137103057932109
[24][31:0]	0.0011662719771266
[25][31:0]	0.00105869944673032
[26][31:0]	0.00105869944673032
[27][31:0]	0.000898259459063411
[28][31:0]	0.000863112974911928
[29][31:0]	0.00073845376027748
[30][31:0]	0.000645368476398289
[31][31:0]	0.000514566956553608
[32][31:0]	0.000430148851592094

İterasyon	Değer
[33][31:0]	0.000367634173016995
[34][31:0]	0.00028699851827696
[35][31:0]	0.000231549158343114
[36][31:0]	7.96744789113291e-05
[37][31:0]	2.87027651211247e-05
[38][31:0]	2.04334155569086e-05
[39][31:0]	9.88445844996022e-06
[40][31:0]	9.88445844996022e-06
[41][31:0]	9.88445844996022e-06
[42][31:0]	9.88445844996022e-06
[43][31:0]	5.25856876265607e-06
[44][31:0]	5.25856876265607e-06
[45][31:0]	5.25856876265607e-06
[46][31:0]	5.25856876265607e-06
[47][31:0]	5.25856876265607e-06
[48][31:0]	5.25856876265607e-06
[49][31:0]	5.25856876265607e-06

Şekil 4.26. 50 parçacık için 50 iterasyona ait pozisyon hataları (cm)

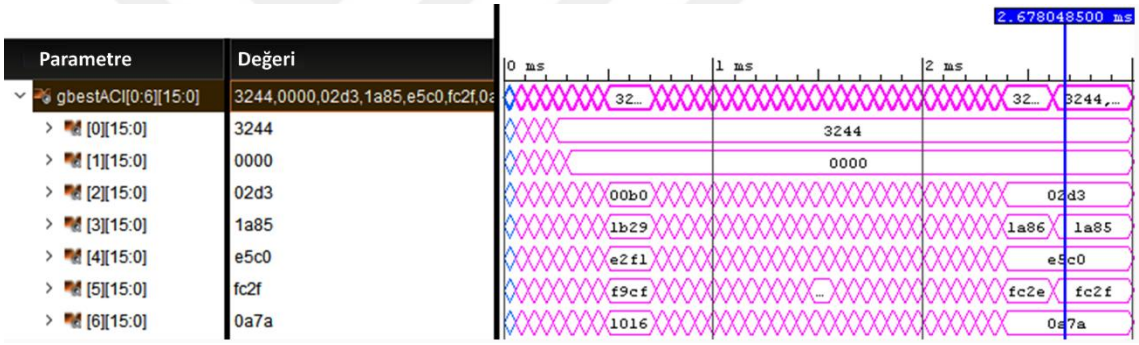
Şekil 4.27'de ise, gerçekleştirilen tasarımdaki her bir alt komponentin saat darbesi sayıları hem 50 iterasyon hemde en küçük pozisyon hatasının elde edildiği 43 iterasyon için ortaya çıkan değerler iki ayrı bölüm olarak gösterilmiştir. "g_sayici" parametresi tüm komponentlerin toplamda kullandıkları saat darbesi sayısını göstermektedir.

Parametre	Değeri	Parametre	Değeri
count_rnd_sayi	16856	count_rnd_sayi	19600
count_rnd_aci	1372	count_rnd_aci	1372
count_sinir	75250	count_sinir	87500
count_ik	582650	count_ik	677500
count_hata	10750	count_hata	12500
count_en_iyi	8686	count_en_iyi	10100
count_guncelle	180600	count_guncelle	210000
g_sayici	876207	g_sayici	1019627
iter	43	iter	50

Şekil 4.27. En küçük pozisyon hatasının elde edilme ve algoritmanın tamamlanma saat darbesi sayısı

Kullanılan FPGA kartı hızına bağlı olarak devre tasarımı 0,87 us ile 3 ms arasında bir çalışma süresi elde edecektir. Bu çalışmada kullanılan kart 3 ns saat frekansına sahip olduğundan dolayı sayısal devre çalışmasını 3,058 ms sonunda sonlandırmaktadır. Şekil 4.27'de "count_rnd_aci" parametrelerinin aynı olduğu görünmektedir. Çünkü bu parametreler, algoritma başlangıcındaki eklem açılarının rassal olarak tutulması için kullanılan sayısal tasarıma aittir ve algoritmada tek bir kez için uygulanır.

Şekil 4.28'de en küçük pozisyon hatasını sağlayan eklem açıları görünmekte olup çözümün bu çalışmada kullanılan FPGA kartında 2,678 ms sonunda elde edildiği görünmektedir. Tablo 4.12, simülasyonda elde edilen ve Şekil 4.28'de görünen eklem açılarının onaltılık tabanda, onluk tabanda radyan olarak ve onluk tabanda derece cinsiden dönüşümlerini göstermektedir.



Şekil 4.28. Minimum pozisyon hatasını sağlayan eklem açıları

Tablo 4.12. 50 parçacık için elde edilen eklem açıları ve dönüşümleri

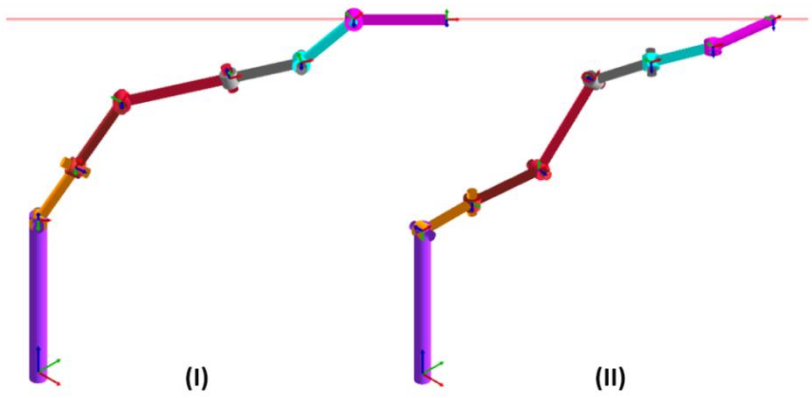
Eklem Açıları	Onaltılık Taban	Onluk Radyan	Onluk Derece (°)	Gerçek Değerler (°)
θ_1	3244	1.5708	90	45
θ_2	0000	0.0	0	-60
θ_3	02d3	0.0882	5.05	0
θ_4	1a85	0.8287	47.50	-45
θ_5	e5c0	-0.8203	-47.02	0
θ_6	fc2f	-0.1192	-6.83	30
θ_7	0a7a	0.3273	18.76	-45

Tablo 4.12'de görünen eklem açıları bu çalışmada kullanılan 7-Dof robot manipülatörüne uygulandığında Şekil 4.30'da görünen robot kolu oryantasyon elde edilmektedir. Şekilde (I), önceden belirlenen eklem açılarıyla oluşan oryantasyonu, (II) ise hesaplanan eklem açılarıyla oluşan oryantasyonu göstermektedir.Tablo 4.12'deki değerler ile oluşan uç elemanın "x", "y" ve "z" koordinatlarına denk gelen konum bilgileri "cm" cinsinden Şekil 4.29'da görünmektedir.

Hesaplanan Değerler	
px[31:0]	19.7526836395264
py[31:0]	24.8450870513916
pz[31:0]	23.3571262359619
teta[0:6][15:0]	3244,0000,02d3,1a85,e5c0,fc2f,0a7a
[0][15:0]	1.57080078125
[1][15:0]	0.0
[2][15:0]	0.0882568359375
[3][15:0]	0.8287353515625
[4][15:0]	-0.8203125
[5][15:0]	-0.1192626953125
[6][15:0]	0.327392578125

Gerçek Değerler	
px[31:0]	19.7462310791016
py[31:0]	24.8519096374512
pz[31:0]	23.3489322662354
teta[0:6][15:0]	1921,de7e,0000,e6df,0000,10bf,e6df
[0][15:0]	0.7852783203125
[1][15:0]	-1.047119140625
[2][15:0]	0.0
[3][15:0]	-0.7852783203125
[4][15:0]	0.0
[5][15:0]	0.5233154296875
[6][15:0]	-0.7852783203125

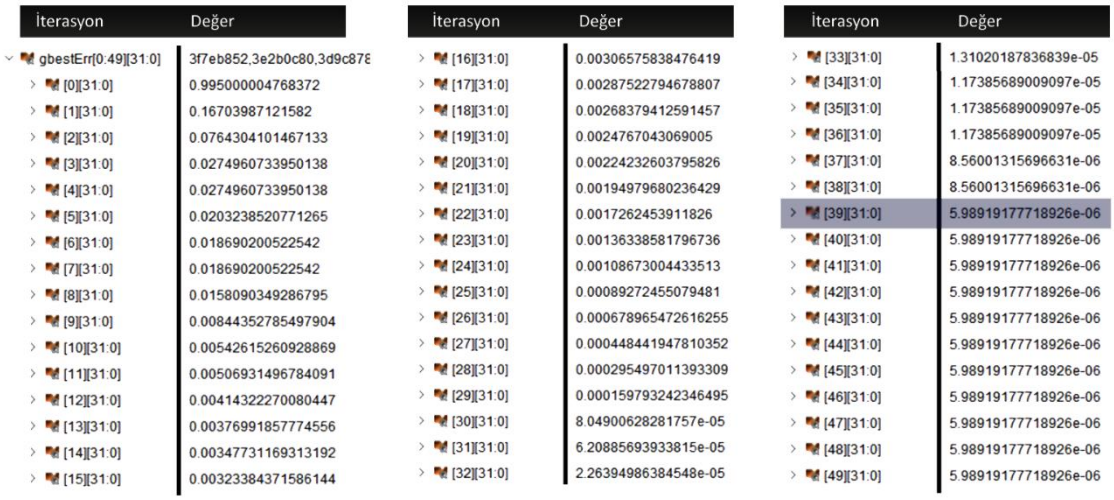
Şekil 4.29. 50 parçacık için kullanılan gerçek ve hesaplanan değerler



Şekil 4.30. Robot manipülatörü oryantasyonu

4.5.4. Yüz parçacık için simülasyon sonuçları

Gerçekleştirilen sayısal PSO tasarımı 100 parçacık için test edildiğinde Şekil 4.31'da gösterilen 50 farklı değer elde edilmiştir. En küçük pozisyon hatası değeri 5,98e-06 ile 39. iterasyonda hesaplanan değerdir.



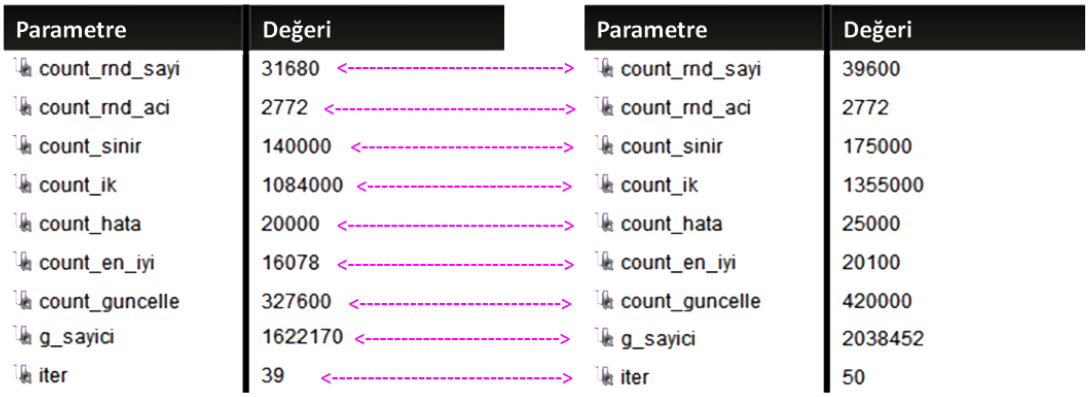
İterasyon	Değer
gbestErr[0:49][31:0]	3f7eb852,3e2b0c80,3d9c87
[0][31:0]	0.995000004768372
[1][31:0]	0.16703987121582
[2][31:0]	0.0764304101467133
[3][31:0]	0.0274960733950138
[4][31:0]	0.0274960733950138
[5][31:0]	0.0203238520771265
[6][31:0]	0.018690200522542
[7][31:0]	0.018690200522542
[8][31:0]	0.0158090349286795
[9][31:0]	0.00844352785497904
[10][31:0]	0.00542615260928869
[11][31:0]	0.00506931496784091
[12][31:0]	0.00414322270080447
[13][31:0]	0.00376991857774556
[14][31:0]	0.00347731169313192
[15][31:0]	0.00323384371586144

İterasyon	Değer
[16][31:0]	0.00306575838476419
[17][31:0]	0.0028752279467880 7
[18][31:0]	0.00268379412591457
[19][31:0]	0.002476704306900 5
[20][31:0]	0.00224232603795826
[21][31:0]	0.00194979680236429
[22][31:0]	0.0017262453911826
[23][31:0]	0.00136338581796736
[24][31:0]	0.00108673004433513
[25][31:0]	0.00089272455079481
[26][31:0]	0.000678965472616255
[27][31:0]	0.000448441947810352
[28][31:0]	0.000295497011393309
[29][31:0]	0.000159793242346495
[30][31:0]	8.04900628281757e-05
[31][31:0]	6.20885693933815e-05
[32][31:0]	2.26394986384548e-05

İterasyon	Değer
[33][31:0]	1.31020187836839e-05
[34][31:0]	1.17385689009097e-05
[35][31:0]	1.17385689009097e-05
[36][31:0]	1.17385689009097e-05
[37][31:0]	8.56001315696631e-06
[38][31:0]	8.56001315696631e-06
[39][31:0]	5.98919177718926e-06
[40][31:0]	5.98919177718926e-06
[41][31:0]	5.98919177718926e-06
[42][31:0]	5.98919177718926e-06
[43][31:0]	5.98919177718926e-06
[44][31:0]	5.98919177718926e-06
[45][31:0]	5.98919177718926e-06
[46][31:0]	5.98919177718926e-06
[47][31:0]	5.98919177718926e-06
[48][31:0]	5.98919177718926e-06
[49][31:0]	5.98919177718926e-06

Şekil 4.31. 100 parçacık için 50 iterasyona ait pozisyon hataları

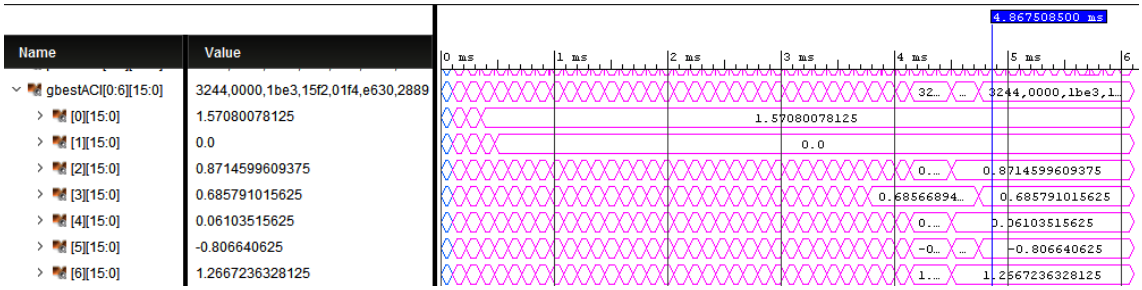
Şekil 4.32 ise 100 parçacıkla test edilme anında alt devrelerin işletilmesi için kullandıkları saat darbelerinin sayısını göstermektedir. Bu devrelerden "count_rnd_aci" parametresinin saydığı alt devre tek bir defa 100 parçacık için çalışır. Onun haricindeki bütün alt devreler 100 parçacık ve 50 iterasyon için çalışmaktadırlar.



Parametre	Değeri	Parametre	Değeri
count_rnd_sayi	31680	count_rnd_sayi	39600
count_rnd_aci	2772	count_rnd_aci	2772
count_sinir	140000	count_sinir	175000
count_ik	1084000	count_ik	1355000
count_hata	20000	count_hata	25000
count_en_iyi	16078	count_en_iyi	20100
count_guncelle	327600	count_guncelle	420000
g_sayici	1622170	g_sayici	2038452
iter	39	iter	50

Şekil 4.32. Alt devrelere ait saat darbelesi sayısı

Şekil 4.33, en küçük pozisyon hatasını sağlayan eklem açılarını ve bu çalışmada kullanılan FPGA kartındaki eklem açılarının elde edilme süresini göstermektedir. Bu sürenin 4,867 ms olduğu açıkça görünmektedir.



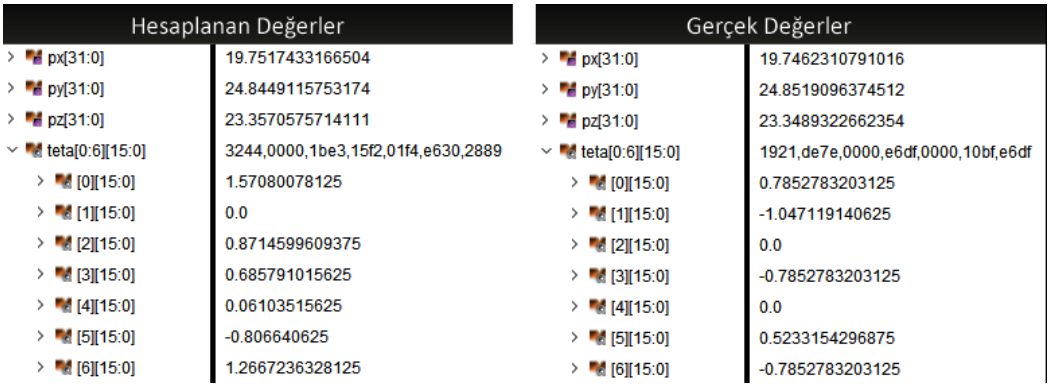
Şekil 4.33. Minimum pozisyon hatasını sağlayan eklem açıları

En küçük pozisyon hatasını sağlayan eklem açılarının onaltılı tabandaki, onluk tabandaki radyan ve derece cinsinden değerleri Tablo 4.13'de görünmektedir.

Tablo 4.13. 100 parçacık testinde elde edilen en iyi eklem açıları ve dönüşümleri

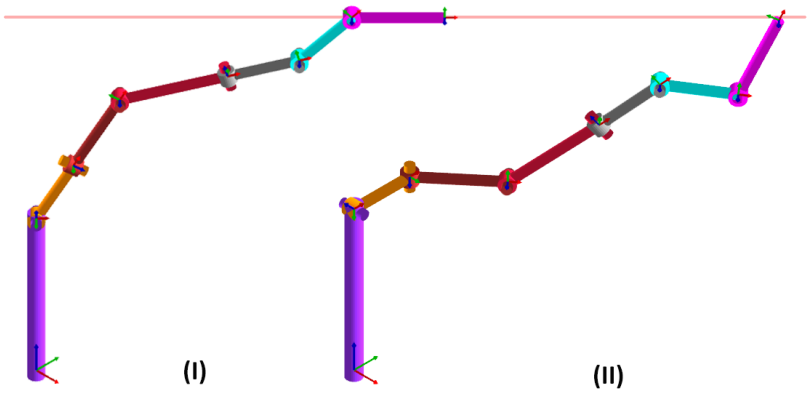
Eklem Açıları	Onaltılık Taban	Onluk Radyan	Onluk Derece (°)	Gerçek Değerler (°)
θ_1	3244	1.5708	90	45
θ_2	0000	0	0	-60
θ_3	1be3	0.8714	49.92	0
θ_4	15f2	0.6857	39.28	-45
θ_5	01f4	0.0610	3.49	0
θ_6	e630	-0.8066	-46.21	30
θ_7	2889	1.2667	72.57	-45

Tablo 4.13'deki değerler ile oluşan uç elemanın "x", "y" ve "z" koordinatlarına denk gelen konum bilgileri Şekil 4.34'te görünmektedir.



Şekil 4.34. 100 parçacık için kullanılan gerçek ve hesaplanan değerler

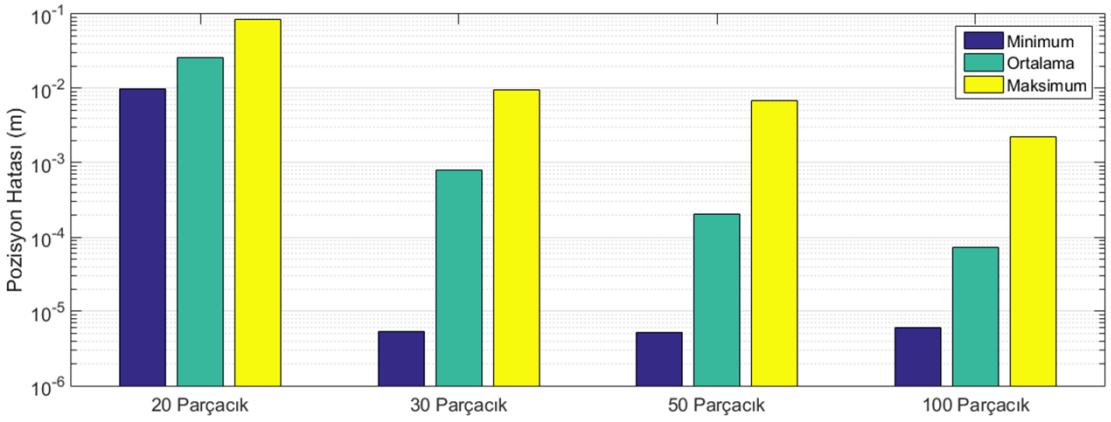
Şekil 4.33'te elde edilen eklem açıları robot kolu simüle edildiğinde ortaya çıkan robot kolu oryantasyonu Şekil 4.35'te görünmektedir. Bu şekilde (I), önceden belirlenen eklem açılarıyla oluşan oryantasyonu, (II) ise hesaplanan eklem açılarıyla oluşan oryantasyonu göstermektedir.



Şekil 4.35. Robot kolu oryantasyonu

4.5.5. Sonuçların analizi

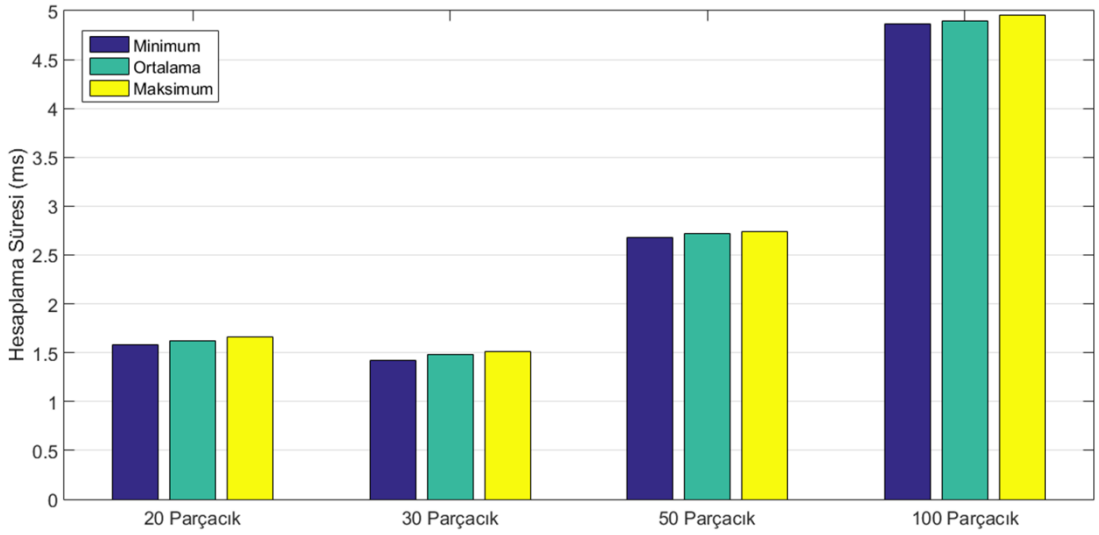
Çalışmanın bu bölümünde parçacık sürü optimizasyonu algoritmasının VHDL ile sayısal tasarımı gerçekleştirilmiştir. Sürü tabanlı algoritmalarda parçacık sayısı problemin hem sonucunu hemde çözüm süresini doğrudan etkilemektedir. Bu nedenle yapılan tasarım 20, 30, 50 ve 100 farklı parçacıkla 50 farklı test ile edilmiş olup; 20 parçacıkla yapılan test 100 iterasyon, diğerleri ise 50 iterasyonda çalıştırılarak sonuçlar elde edilmiştir.



Şekil 4.36. Her bir parçacık için elde edilen minimum, maksimum ve ortalama pozisyon hatası değerleri

FPGA sisteminde 50 farklı test sonucunda 50 farklı değer elde edebilmek adına başlangıç rassal sayılar ve açılar belirlenmelidir. Onun yerine algoritmalar 50 iterasyonu tamamladıklarında sistemdeki "reset" ucu algoritmaları başlangıç konumuna yönlendirirken rassal sayı üreteçlerini ise resetlemeden devam etmelerini sağlamıştır. Bu sayede algoritma her resetlendiğinde rassal sayılar değişmektedir.

Şekil 4.36'da gerçekleştirilen 50 test sonunda en küçük yani en iyi pozisyon hatasını, en büyük yani en kötü pozisyon hatasını ve ortalama değerleri görünmektedir. Bu değerlerden sadece 20 parçacıkla elde edilen değerlerin her üçüde en kötü değerler olarak görünmektedir.



Şekil 4.37. Her bir parçacık için elde edilen minimum, maksimum ve ortalama süreleri

Şekil 4.37'de her bir parçacıkla gerçekleştirilen 50 testin çözüme ulaşırken elde ettikleri minimum, maksimum ve ortalama süreleri görünmektedir.

Tablo 4.14. Parçacık sayısına göre elde edilen en iyi değerlerin karşılaştırmalı sonuçları

Parçacık Sayısı	Pozisyon Hatası (m) Ort / Min	Çözüm Süresi (ms) (SD*3ns) Ort / Min	Çözüm İterasyon Sayısı Ort / Min	Çözüm Saat Darbesi Sayısı (SD)
20	2.56e-02 / 9.76e-03	1.622 / 1.583	69 / 64	536842 / 527625
30	7.88e-04 / 5.37e-06	1.483 / 1.423	43 / 38	490638 / 474285
50	2.05e-04 / 5.25e-06	2.717 / 2.678	45 / 43	886146 / 876207
100	7.26e-05 / 5.98e-06	4.894 / 4.867	41 / 39	1627891 / 1622170

Tablo 4.14, yapılan dört farklı parçacık sayısına göre 50 farklı testin elde edilen ortalama ve en iyi sonuçlarını göstermektedir. Yapılan testlerde kritik eşiğin 30 parçacık olduğu görünmektedir. Çünkü birinci testte hem pozisyon hatası hemde çözüm süresi ikinci teste

göre oldukça geride kalmıştır. Birinci test dışındaki tüm testlerde pozisyon hatası 10^{-6}'larda çıkmış olmasına karşın 30 parçacıkta en iyi sürenin 1,423 ms ve ortalama sürenin 1.483 ile çok daha minimumda olduğu açıktır. Çözüme ulaşılan iterasyon sayısı açısından bakıldığında yine 30 parçacıkla yapılan test bu kıyaslamada da en iyi seçenek olarak görünmektedir. Bu durum elbette saat darbesi sayısında açıkça görünmekte olup bu tez çalışmasında kullanılan FPGA kartının saat frekansı 3 ns olması nedeniyle süre bu şekilde görünmektedir. Eğer ki saat frekansı 1 ns olan bir kart tercih edilseydi bu durumda elbette sürenin 3'te 1'e düşeceği açıkça görünecektir.

Tablo 4.15. Yazılımsal değerlerle FPGA ile elde edilen donanımsal değerlerin karşılaştırılması

Test No	Algoritma	Popülasyon Sayısı	Pozisyon Hatası (cm)	Çözüm Süresi (ms)	Kaç Katı?	İterasyon Çözüm/Toplam
1	PSO	100	1.52e-02	559.3	393	78/100
2	KPSO	100	9.82e-10	220.1	155	97/100
3	RDV PSO	100	7.25e-07	770.3	542	97/100
4	YAK	100	1.08e-03	465.6	327	207/300
5	ABA	50	4.30e-05	657.6	463	181/200
6	FPGA (PSO)	30	5.37e-06	1.42	-	38/50

Çalışma neticesinde yazılım ile elde edilen sonuçlarla FPGA ile elde edilen sonuçlar karşılaştırıldığında ise karşımıza Tablo 4.14'de görünen değerler çıkacaktır. Çözüm süresi ve pozisyon hatası olarak her bir algoritma ile elde edilen 10 adet testin ortalama değerleri verilmiştir. Yazılımsal olarak elde edilen değerlerin 220 ms ile 800 ms arasında olduğu açıktır. Bu değerler bir robot manipülatörünü hareket ettirmek veya arzu edilen noktalara konumlandırmak için ziyadesiyle fazladır. Halbuki, FPGA ile yani donanımsal olarak elde edilen değer ise 1.42 ms'dir. Dolayısıyla arada 150 ile 550 kat fark mevcuttur. Bu nedenle burada FPGA'nın robotların veya buna benzer karmaşık yapılı sistemlerin çalışmasında çok önemli bir katkı sunacağı aşikardır.

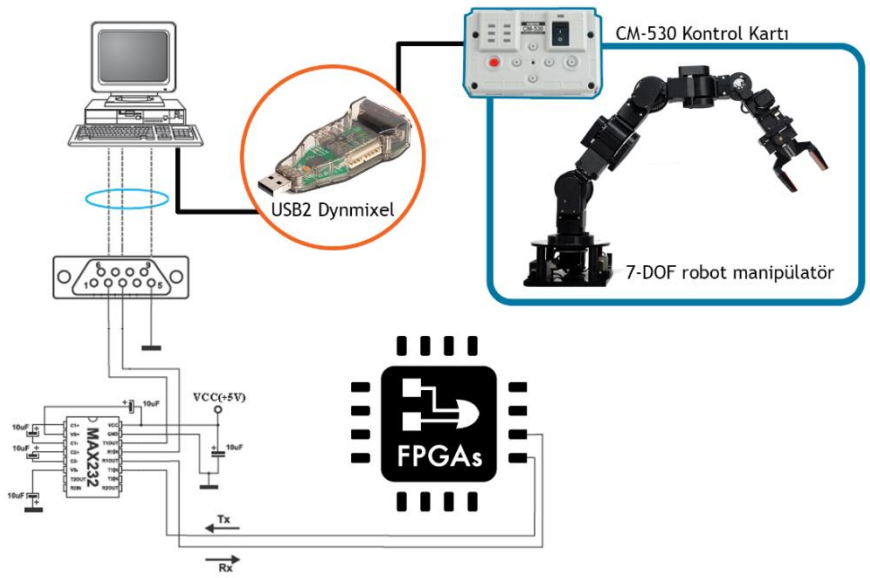
Tablo 4.16, tüm testlerde çözüme ulaşılan saat darbeleri sayısını tasarımın alt bölümlerine bölünmüş şekilde göstermektedir. Elbette bir tasarımın çalışa ma süresi tasarımın büyüklüğüne bağlı olmakla birlikte tasarımın tekrar çalışma ve iterasyon sayısına

doğrudan bağlıdır. 30 parçacıkla yapılan ikinci testteki alt tasarımlara ait saat saat darbeleri sayısı "Rasgele Açılar" alt tasarımı hariç diğerlerinde daha azdır. "Rasgele Açılar" alt devresinde parçacık sayısı kadar rasgele açı değeri üretildiğinden daha uzun sürmüştür. O nedenle diğer karşılaştırmalarda olduğu gibi burada da 30 parçacıkla yapılan test en iyi değerleri ortaya koymaktadır.

Tablo 4.16. PSO ile çözüme ulaşılan saat darbesi sayılarının karşılaştırması (en iyi değerlere ait)

Bölümler	Değişken	Çözüm Saat Darbesi Sayısı			
		20 Par	30 Par.	50 Par.	100 Par
Rasgele Sayılar	count_rnd_sayi	9880	9048	16856	31680
Rasgele Açılar	count_rnd_aci	532	812	1372	2772
Açı Sınırları	count_sinir	45500	40950	75250	140000
İleri Kinematik	count_ik	352300	317070	582650	1084000
Pozisyon Hatası	count_hata	6500	5850	10750	20000
En İyi Değerler	count_en_iyi	5328	4756	8686	16078
Açı ve Hız Güncelle	count_guncelle	10752l0	95760	180600	327600
Genel Sayıcı	g_sayici	527625	474285	876207	1622170
İterasyon	iter	64/100	38/50	43/50	39/50

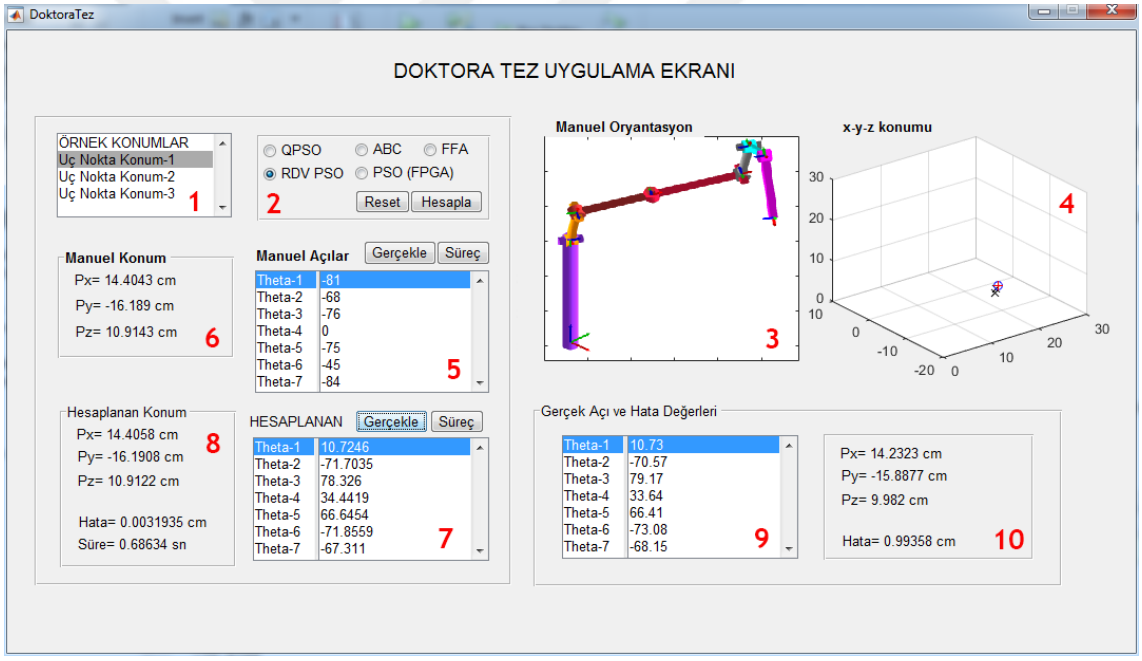
4.6. 7-DOF Robot Kolu ile Algoritmaların Gerçeklenmesi



Şekil 4.38. Tez çalışmasında kullanılan sistemin blok şeması

Çalışılan optimizasyon teknikleri Dynamixel servo motorlarla tasarımı yapılan ve Şekil 3.15'de gösterilen robot manipülatörü ile gerçekleştirilmiştir. Bu tarz oluşturulan robot manipülatörlerinin kontrolü kendi control kartlarıyla gerçekleşmekte ve bilgisayardaki Matlab benzeri üçüncü parti yazılımlar ile haberleşmek için "Usb2Dynamixel" donanım arabirimini kullanmaktadır. Benzer şekilde FPGA kartı ile bilgisayara very aktarma veya bilgisayardan verileri alma işlemi UART protokolü ile gerçekleştirilmiştir.

Bu tez çalışmasında FPGA kartının Dynamixel motorlarla doğrudan bağlantısının yapılamaması nedeniyle seri port üzerinden bilgisayarla haberleştikten sonra robot manipülatörüne ulaşmaktadır (Şekil 4.38). Ancak FPGA kartı 5V çıkışa sahiptir, halbuki bilgisayar seri port girişi 12V ile işlem yapmaktadır. Bu nedenle Max232 entegresi bu tarz farkları denkleştirmek için kullanılmıştır.



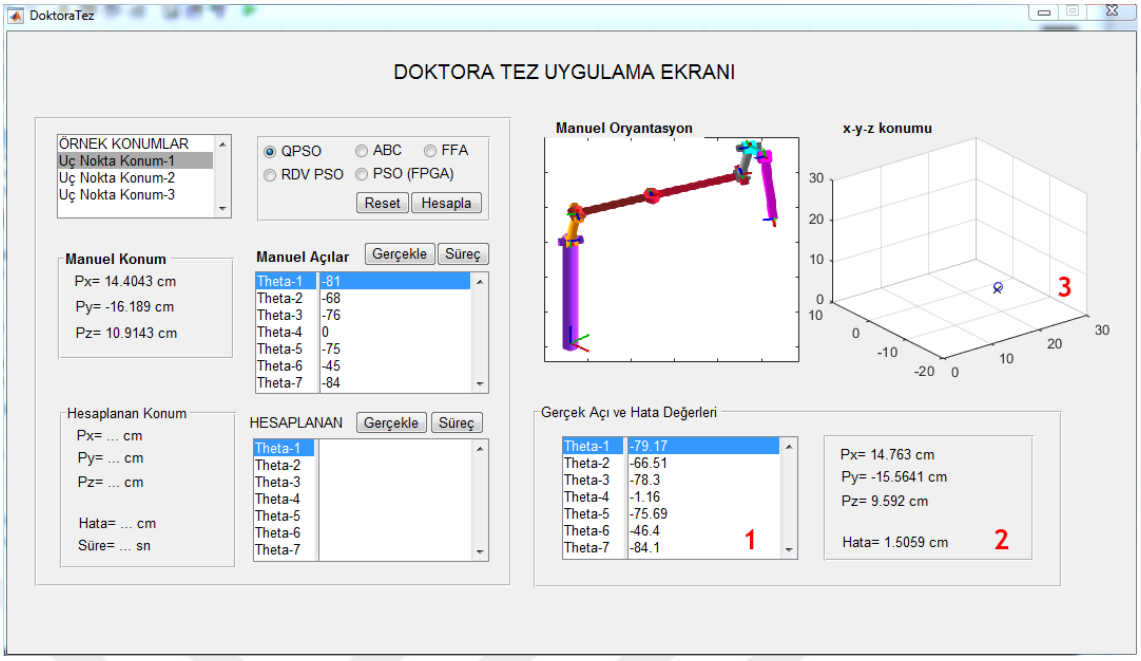
Şekil 4.39. Doktora tezi için geliştirilen Matlab GUI arayüzü

Gerek FPGA'dan gelen gerekse de optimizasyon tekniklerinden gelen yazılımsal işlem sonucu elde edilen açı değerlerini robot manipülatörüne aktarmak için Matlab arabirimi kullanılmıştır. Bunun için Şekil 4.39'da görünen Matlab GUI arayüzü geliştirilmiştir. Bu arayüzde önceden belirlenmiş "x, y ve z" konumlarına manipulator uç elemanı konumlandırmak için bu çalışmada kullanılan zeki optimizasyon teknikleri kullanılmıştır.

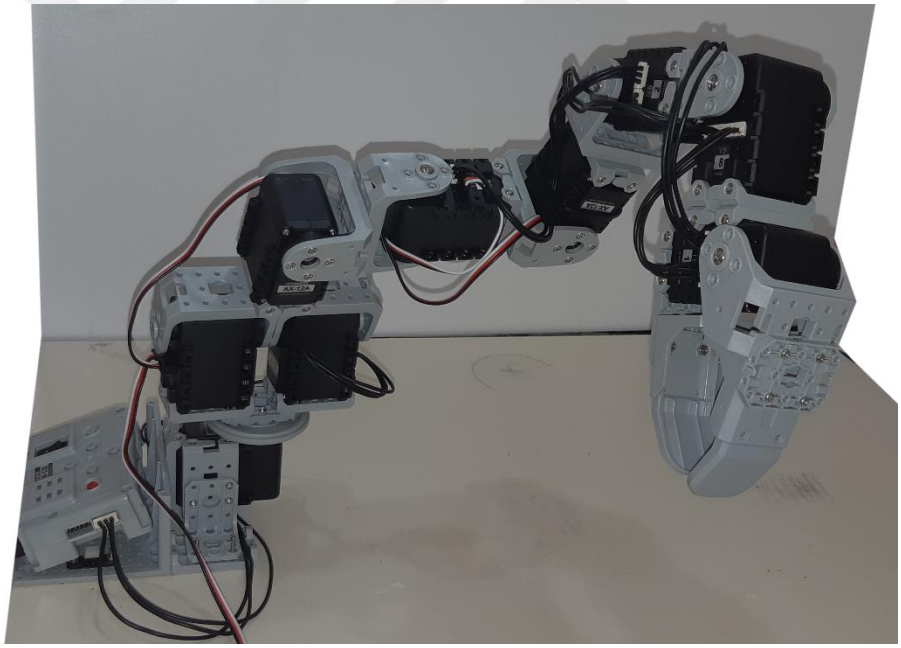
Şekil 4.39'de ki ekran göründüğü üzere 10 farklı göstergeye sahiptir ve her bir gösterge numaralarla gösterilmiştir. Bu göstergeler şu amaçla kullanılmaktadır:

- (1); robot manipülatörü için önceden manuel olarak belirlenen çalışma uzayı konumlarını göstermektedir.
- (2); (1) numaralı alandan seçilen konumu hesaplamak için kullanılacak zeki optimizasyon tekniğinin seçildiği göstergedir.
- (3); seçilen konuma dair örnek manipulator oryantasyonunu göstermektedir.
- (4); seçilen konuma ait x, y ve z noktasını gösteren grafiğin bulunduğu alandır.
- (5); seçilen konuma yönlendirilen manipülatöre manuel olarak atanan eklem açılarını gösterir.
- (6); manuel olarak konumlandırılan manipulator uç elemanının x, y ve z noktalarını gösterir.
- (7); (6) numaralı alanda verilen x, y ve z koordinatlarına uç elemanı konumlandıran ve (2) numaralı göstergeden seçilen optimizasyon tekniği ile hesaplanan eklem açılarını göstermektedir.
- (8); hesaplanan eklem açıları neticesinde elde edilen x, y ve z noktalarını gösterir.
- (9); "Gerçekle" butonuna basıldıktan sonra hesaplanan eklem açılarını servo motorların hangi hata oranında gerçekleştirdiklerini gösteren alandır.
- (10); hesaplamayla elde edilen x, y ve z konumuna uç elemanı konumlandırırken servo motorların hataları nedeniyle ortaya çıkan gerçek x, y ve z konumu ile hata değerini göstermektedir.

Şekil 4.39'da ki ilk manuel olarak belirlenen konum seçilip 7-eklemli manipülatörümüzü bu konuma yönlendirdiğimizde Matlab GUI ekranı Şekil 4.40'deki gibi olacaktır. Bu şekilde göründüğü üzere servo motorların çalışması neticesinde elde edilen eklem açıları (1) numaralı göstergede, bu eklem açıları neticesinde x, y ve z konumları ve hata değeri (2) numaralı göstergede, manuel olarak belirlenen açılarla elde edilen nokta ile servoların gerçek zamanlı hareketi neticesinde elde edilen noktanın gösterildiği alan ise (3) numaralı göstergedir. Robot manipülatörünün manuel olarak belirlenen noktaya gerçek zamanlı olarak konumlanmasına dair görüntü Şekil 4.41'da verilmiştir. Eklem açıları ile x, y, z konumlarına ait elde edilen verilerin karşılaştırması ise Tablo 4.17'te görünmektedir.



Şekil 4.40. İlk konum seçilerek manipülatörde gerçekleştirilmesi



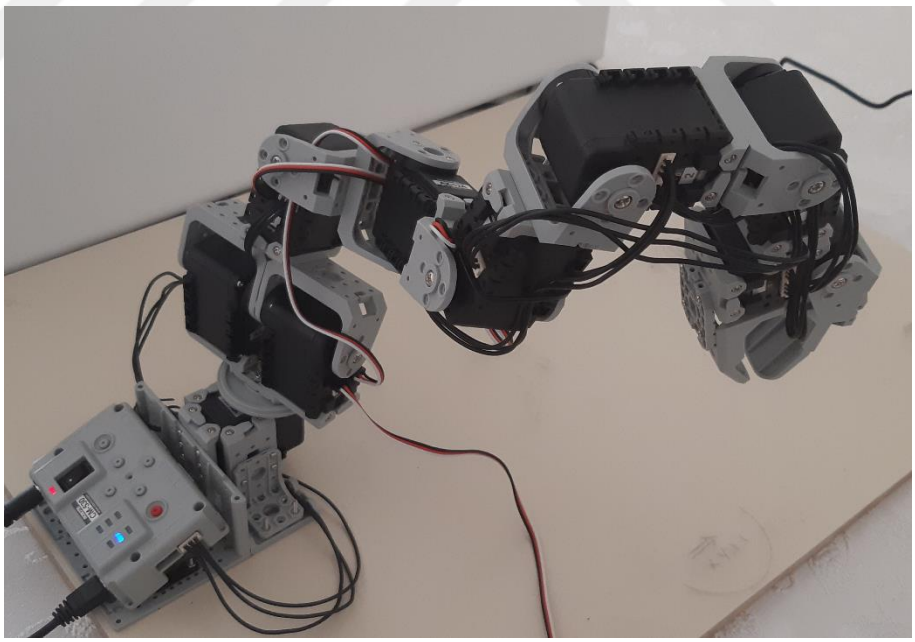
Şekil 4.41. Manuel değerler neticesinde robot manipülatörünün konumlanması

Tablo 4.17. Manuel olarak belirlenen eklem açıları ile gerçeklenen eklem açılarının karşılaştırılması

Eklem Açıları	Manuel (Derece)	Gerçek (Derece)	Fark (cm)		x (cm)	y (cm)	z (cm)
1	-81	-79.17	1.83	Manuel	14.40	-16.18	10.91
2	-68	-66.51	1.49	Gerçek	14.76	-15.56	9.59

3	-76	-78.3	2.3	Hata	0.36	0.62	1.32
4	0	-1.16	1.16				
5	-75	-75.69	0.69				
6	-45	-46.4	1.4				
7	-84	-84.1	0.1	Genel Hata (Öklit - cm)			1.5059

Şekil 4.39'de ise Şekil 4.40'de manuel olarak belirlenen noktaya RDV PSO tekniği ile hesaplama neticesinde (7) numaralı alanda elde edilen eklem açıları ve (8) numaralı alanda x, y ve z noktaları ile hata değeri görünmektedir. Bu hesaplamayla elde edilen eklem açıları ile servo motorlar yönlendirildiğinde ise eklem açıları (9) numaralı göstergede bu eklem açılarının oluşturduğu hata oranı ise (10) numaralı göstergede görünmektedir. (4) Numaralı göstergede ise üç boyutlu koordinat sistemindeki x, y ve z konumları görünmektedir. "o" simgesi, manuel olarak belirlenen ve hesaplama yöntemiyle elde edilmesi gereken konumu göstermektedir. "+" simgesi, seçilen zeki optimizasyon yöntemiyle hesaplanarak elde edilen konumu, "x" simgesi ise servoların kendi hassasiyetleri ile hareketi sonucunda elde ettikleti konumu göstermektedir.



Şekil 4.42. Hesaplanan değerler neticesinde manipülatörün konumlandırılması

Hesaplama ile elde edilen değer aslında hedefe istenilen oranda yaklaşmışken kullanılan servo motorların hata oranları sebebiyle hedeften daha fazla uzaklaşılmıştır. Hesaplama neticesinde elde edilen eklem açılarına göre robot manipülatörünün gerçek zamanlı konumlandırılması Şekil 4.42'ta görünmektedir. Robot manipülatörünün gerçek zamanlı

olarak konumlandırılması neticesinde servo motorlardan kaynaklı ortaya çıkan gerçek eklem açıları ile karşılaştırmalı sonuçlar Tablo 4.18'de verilmektedir.

Tablo 4.18. Hesaplanan eklem açıları ile gerçeklenen eklem açılarının karşılaştırılması

Eklem Açıları	Hesaplama (Derece)	Gerçek (Derece)	Fark (cm)		x (cm)	y (cm)	z (cm)
1	10.72	10.73	0.01	Hesaplama	14.40	-16.19	10.91
2	-71.70	-70.57	1.13	Gerçek	14.23	-15.88	9.98
3	78.32	79.17	0.85	Hata	0.17	0.31	0.93
4	34.44	33.64	0.80				
5	66.64	66.41	0.23				
6	-71.85	-73.08	1.23	Hesaplanan Hata (cm)			0.0031
7	-67.31	-68.15	0.84	Gerçek Hata (cm)			0.9935

Tablo 4.18'de açıkça görünmektedir ki; zeki optimizasyon teknikleri ile elde edilen hata "0.0031 cm" iken servo motorların hareketi esnasında ortaya çıkan hatalar sebebiyle gerçekte "0.99 cm" olarak ortaya çıkmaktadır. Bu durum elbette Dynamixel Ax12A servo motorların adım hassasiyeti değeri olan "0.29" ile doğrudan ilgilidir. Tablo 4.19'de bu servoların 60°'den 100°'ye hareketlerini gerçekleştirirken almış oldukları adım değerleri ve bu adım değerlerine karşılık gelen açı değerleri görünmektedir.

Tablo 4.19. Dynamixel Ax12a servoların 60°'den 100°'ye hareketi

Adım No	Değer	Açı	Adım No	Değer	Açı	Adım No	Değer	Açı
1	207	60.03	9	258	74.82	17	311	90.19
2	213	61.77	10	265	76.85	18	318	92.22
3	219	63.51	11	271	78.59	19	324	93.96
4	226	65.54	12	277	80.33	20	331	95.99
5	232	67.28	13	284	82.36	21	337	97.73
6	238	69.02	14	291	84.39	22	343	99.47
7	245	71.05	15	297	86.13			
8	251	72.79	16	304	88.16			

Tablo 4.19'a bakıldığında servo motor 207 adım değerine karşılık gelen 60.03°'lik açıyla başlamıştır. Göründüğü gibi servo motor daha başlangıçta 0.3°'lik bir hata ile haretine başlamış, toplamda 22. adımda yani 343 adım değerinde hareketini tamamlamış ve almış olduğu açı değeride 99.47° olarak görünmektedir. Burada servo motorun 0.53°'lik bir hata değeri ile hareketini tamamladığı açıkça görünmektedir.

BÖLÜM 5. TARTIŞMA VE SONUÇ

Bu tez çalışmasında seri robotların kontrolündeki en temel problemlerden biri olan ve zorluk derecesi NP grubuna giren ters kinematik çözüm konusuna araştrmacıların getirmiş oldukları öneriler kronolojik olarak irdelenerek çözümün gelişimsel süreci incelenmiş ve buradan yola çıkılarak tez çalışması şekillenmiştir. Kapsamlı literatür çalışması neticesinde 2000'li yıllara kadar ters kinematik problemine çözüm için araştırmacıların çeşitli cebirsel, geometrik ve sayısal yöntemler önerdikleri görülmüştür.

Cebirsel yöntemde kapalı form çözümler kullanılır. Bu yöntemde robot manipülatörünün ters kinematik denklemleri yüksek dereceli polinomlara dönüştürülür ve devamında bu polinom kökleri sayısal olarak belirlenir (Pieper, 1968, p. 36). Ancak bu yöntemin en büyük sorunu robot parametreleri açısından sınırlı sayıda çözümü garanti etmesidir (Wang, Xu ve Hao, 2001, p. 74). Ayrıca her robot manipülatörünün benzersiz geometrik bir yönü mevcuttur. Cebirsel yöntemler bu yaklaşımı dikkate almadıklarından dolayı genellikle geometrik yöntemlerle birleştirilerek ters kinematik çözümler üretmiştir (Wang ve Chen, 1991, pp. 489-499). Sayısal yöntemlerde ise başlangıç değerinin seçimi sonuca doğrudan etki eder. Bunun yanında çözüme belli bir değere kadar yakınsama göstererek çözümün iyileşmesine katkıda bulunamaz. Ayrıca her defasından benzer çözümler üreterek yeni çözümlerin ortaya çıkmasına engel olmaktadır. Burada bahsedilen her üç yöntem, ters kinematik problemin çözümü için robot serbestlik derecesinden ve geometrisinden bağımsız olarak genel ifadeler ortaya koymakta zorlanmaktadır. Tüm bunlar ışığında cebirsel, geometrik ve sayısal yöntemlerin ters kinematik çözüm için etkili sonuçlar üretemekte zorluklar çıkardığı açıkça görünmektedir (Cidderwar ve Babu, 2010, p. 1084). Özellikle bu tez çalışmasında test işlemleri için kullanılan 7-DOF gereğinden fazla serbestlik derecesine sahip robot manipülatörü için etkili bir analitik çözüm ortaya konamamaktadır (Chen ve Lau, 2016, p. 267).

Ters kinematik problemin çözümünde cebirsel, geometrik ve sayısal yötemlerin yetersiz kaldıkları durumundan yola çıkan araştırmacılar doğadan esinlenilerek geliştirilen (Dereli, Köker, Öylek ve Ay, 2019, pp. 75-77) sezgisel tekniklere yoğunlaşmışlardır. Çünkü bu teknikler pek çok karmaşık ve lineer olmayan problemde etkili sonuçlar ortaya koyduklarından dolayı hemen her alanda yaygın şekilde kullanılmaktadır. Sezgisel tekniklerden ilki olan ve yüksek tahmin gücü ve hızlı çalışması ile ön plana çıkan yapay sinir ağları hem planar hemde eklemli robotlarda yoğun olarak kullanılmaya başlanmıştır (Hasan, Hamouda, Ismail ve Al-Assadi, 2006, pp. 432-438; Mayorga ve Sanongboon, 2005, pp. 1-23). Araştırmalarda bu algoritmanın çok hızlı çalışmasının yanında hassas işlemlerde etkili çözümler üretememesi ve eğitme sürelerinin uzun sürmesi gibi nedenlerle daha fazla geliştirilmesi gündeme gelmiş ve araştırmacılar bulanık mantık, tavlama benzetimi (Köker ve Çakar, 2016, pp. 553-565) ve evrimsel algoritmalarla birlikte oluşturdukları hibrit tekniklerle yapay sinir ağlarının oluşturduğu hata değerlerini çok daha minimum değerlere çekebilmişlerdir (Bahashti et al, 2003, pp. 924-929). Özellikle yapay sinir ağları ve genetik algoritma tarzı evrimsel algoritmalarla bu problemin çözümüne literatürde sık şekilde rastlanmaktadır (Kalra ve Prakash, 2003, pp; Oyama et al, 2001, pp. 787-805; Caceres, Rosairo ve Amaya, 2019).

Yapay sinir ağlarının arzu edilen şekilde minimum değerler üretememesi ve çözüme ulaşma süresi bakımından çok uzun süreler vermesi neticesinde sürü algoritmaları bu konuya çözüm olabilecek farklı bir teknik olarak gündeme geldiklerinde araştırmacılar ters kinematik problemi bir uygunluk fonksiyonu şeklinde ifade ederek bu tekniklerle çözüme ulaşmışlardır (Ayyıldız ve Çetinkaya, 2016, pp. 825-836). Yine bu algoritmaların zayıf yönlerini güçlendirerek, çözüme olumsuz etki eden parametrelerini değiştirerek veya iki ve daha fazla tekniği birleştirerek hibrit bir teknik ile çözümün geliştirilebildiği görünmektedir (Çavdar, Mohammed ve Milani, 2012, pp. 329-333). Sürü algoritmaları dışında pek çok zeki algoritmanın ters kinematik problemin çözümünde yoğun olarak kullanılmış olması bu tez çalışmasında sürü algoritmalarıyla çalışılmasının başlıca nedenidir (Huang, Chen ve Wang, 2012, pp. 3105-3110; Rokbani, Casals ve Alimi, 2015, pp. 369-395). Tüm bu bilgiler ve araştırmalar neticesinde bu teknikler pozisyon hatası bakımından etkili çözümler üretmelerine rağmen çözüm süresi açısından gerçek bir robot

manipülatöründe uygulanacak şekilde değerler ortaya koyamadıkları için çözüm süresinin düşürülmesi konusu gündeme alınmıştır.

Bu tez çalışmasında dikkat çekici iki durum söz konusudur: Bunlardan birincisi daha önce literatürde herhangi bir bilimsel araştırmada kullanılmayan RDV (Rassal Azalan Hızlı) olarak isimlendirilen IW tekniği ile klasik PSO algoritması 1000 kat ile 1000000 kat arasından hızlandırılmıştır. Böylece literatüre bundan sonraki çalışmalarda kullanılabilecek yeni bir teknik kazandırılmıştır. Diğer önemli husus ise FPGA entegrelerinde çalışabilen sentezlenebilir PSO sayısal devresi ile ters kinematik çözümün hesaplanma süresinin saniyeler seviyesinden milisaniyeler seviyelerine çekilebilmiş olmasıdır.

Bu çalışmada doğadan ilham alınarak geliştirilen karmaşık ve lineer olmayan problemler karşısında etkili performanslar ortaya koyan sezgisel algoritmaların donanımsal olarak tasarlanarak işlem süresini kısaltmalarına odaklanılmıştır. Bu durumu çok daha çarpıcı hale getirmek adına yapısı oldukça karmaşık olan ve bu nedenle çözümü çok uzun zaman alan 7-eklemli seri robot manipülatörünün ters kinematik problemi örnek olarak alınmıştır. Bu problem, sezgisel tekniklerden parçacık sürü optimizasyonu (PSO), yapay arı kolonisi (YAK), ateş böceği sürüsü (FA), kuantum parçacık sürü optimizasyonu (KPSO) ve PSO algoritmasının güçlendirilmesini sağlayan IW tekniklerinden olan ve ilk defa bu tez çalışmasında kullanılan rassal azalan hızlı parçacık sürü optimizasyonu (RDV PSO) kullanılarak çözülmüştür. Bu tekniklerden parçacık sürü optimizasyonu (PSO) algoritması VHDL dili ile donanımsal olarak tasarlanarak Nexys 4 DDR FPGA kartında ters kinematik probleminin çözümünün gerçeklenmesi için kullanılmıştır. Geliştirilen teknikler Matlab GUI arayüzü üzerinden robot manipülatörü ile haberleştirilerek elde edilen çözümler gerçeklenmiştir. Ancak gerçekleme esnasında kullanılan servo motorların hassasiyetlerinin düşük olması nedeniyle özellikle pozisyon hatası değerinin artarak hesaplanan değerden farklı çıktığı gözlemlenmiştir.

Bu tez çalışması kapsamında kullanılan teknikler çalışma süresi ve pozisyon hatası bakımından karşılaştırılmış olup yazılımsal olarak en iyi çözüm süresi ve en iyi pozisyon hatası değeri kuantum parçacık sürü optimizasyonu algoritması tarafından elde edilmiştir.

Ancak donanımsal (FPGA) olarak tasarlanan parçacık sürü optimizasyonu algoitması kuantum PSO algoritmasından yaklaşık 300 kat daha kısa sürede çözüme ulaşmıştır. Sonuç olarak çalışmamızdan elde edilen önemli bulgular şu şekilde özetlenebilir:

a. Sezgisel optimizasyon tekniklerleri problemin uygunluk fonksiyonuna dönüştürülmesi neticesinde kullanılabildiğinden dolayı bu çalışmada Öklit Bağıntısı uygunluk fonksiyonu olarak kullanılmıştır.
b. Bu teknikler, çözümü NP derecesinde zor, karmaşık ve uzun zaman alan problemler için ideal olmasına karşın her algoritmanın çözüme yakınsama oranı farklılık arz etmektedir. Bu çalışmada kullanılan yapay arı kolonisi, parçacık sürüsü, ateş böceği sürüsü ve kuantum parçacık sürüsü algoritmaların tamamı pozisyon hatası açısından tatmin edici fakat birbirinden farklı değerler ortaya koymuştur.
c. Bu çalışmada kullanılan yazılımsal algoritmaların çalışma süresi bakımından birbirlerine üstünlükleri çok küçük oranda olsada vardır. Fakat bu süreler gerçek bir robota uygulanacak nitelikte değildir.
d. Kullanılan malzeme kısıtı nedeniyle simülasyonda elde edilen değerlerle gerçek zamanlı çalışma esnasında elde edilen değerler birbirinden farklılık gösterebilmektedir. Bu çalışmada kullanılan servo motorların hassasiyetinin düşük olması sebebiyle 100 katlık pozisyon hatası farkı ortaya çıkmıştır.
e. Tüm algoritmaların gelişmeye açık olduğu dolayısıyla bazı parametrelerinin güçlendirilerek daha iyi sonuçların elde edileceği aşikârdır. Bu çalışmada parçacık sürüsü algoritması için yeni bir varyant (IW) tekniği tanıtılmıştır. Bu teknikle elde edilen değerler klasik parçacık sürü optimizasyonundan çok daha kaliteli pozisyon hatası değerleri üretmiş ancak süre bakımından benzer değerleri ortaya koymuştur.
f. Yazılımsal olarak bakıldığında bu çalışmada kullanılan sürü algoritmaları pozisyon hatası açısından birbirlerine üstünlük gösterselerde hesaplama süresi açısından yaklaşık benzer değerleri (0.3 s – 0.9 s) ortaya koymuşlardır.

KAYNAKLAR

Adewumi, A., & Arasomwan, M., (2016), 'On the performance of particle swarm optimisation with (out) some control parameters for global optimization', International Journal of Bio-Inspired Computation, 8, p14-32.

Akı, O., (2010), Robot Bilimi ve Robotların Sınıflandırılması. Erişim adresi: http://www.ozanaki.com/home/uploads/teaching/secbst426-robotik-bilim/robot-bilimi-02-robotlarin-siniflandirilmasi.pdf (Erişim Tarihi: 29/01/2019).

Alataş, B., (2007), 'Kaotik Haritalı Parçacık Sürü Optimizasyon Algoritmaları Geliştirme', Doktora Tezi, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Elazığ.

Allaire, F. C., Tarbouchi, M., Labonté, G., & Fusina, G., (2009), 'FPGA implementation of genetic algorithm for UAV real-time path planning', J Intell Robot Syst, 54, pp. 495-510.

Alkhafaji, F.S., Hasan, W.Z., Isa, M.M., & Sulaiman, N., (2018), 'Robotic Controller: ASIC versus FPGA - A Review', Journal of Computational and Theoretical Nanoscience, 15, p1-25.

Almusawi, A.R., Dülger, L.C., & Kapucu, S., (2016), 'A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242)', Computational intelligence and neuroscience [online], 2016, DOI: 10.1155/2016/5720163.

Alp, O.E., (2012), 'Genel Amaçlı Robot Kolu Tasarımı', Yüksek Lisans Tezi, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, İzmir.

Aydilek, İ.B., (2017), 'Değiştirilmiş ateşböceği optimizasyon algoritması ile kural tabanlı çoklu sınıflama yapılması' Journal of the Faculty of Engineering and Architecture of Gazi University, 32, p1097-1107.

Ayyıldız, M., (2018), 'Al ve yerleştir robot otomasyonu için bir çerçeve çalışması', Karaelmas Fen ve Mühendislik Dergisi, 8, p575-580.

Ayyıldız, M., & Çetinkaya, K., (2016), 'Comparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-DOF serial robot manipulator', Neural Computing and Applications, 27, p825-836.

Babu, B.G., & Kannan, M., (2002), 'Lightning bugs', Resonance, 7, p49-55.

Baheshti, M.T.H., Tehrani, A.K., & Ghanbari, B., (2003), 'An optimized adaptive fuzzy inverse kinematics solution for redundant robots', International Symposium on Intelligent Control, Houston, USA, p924–929.

Bahrin, M.A., M.F. Othman, N.N., & Talib, M.F., (2016), 'Industry 4.0: A review on industrial automation and robotic', Journal Teknologi (Sciences & Engineering), 78, p137-143.

Bai, Q., (2010), 'Analysis of Particle Swarm Optimization Algorithm', Computer and Information Science, 3, p180-184.

Bansal, J., Singh, P., Saraswat, M., Verna, A., Jadon, S., & Abraham, A., (2011), 'Inertia Weight Strategies in Particle Swarm Optimization', Third World Congress on Nature and Biologically Inspired Computing (NABIC). Salamanca, Spain.

Bao, S., Yan, H., Chi, Q., Pang, Z., & Sun, Y., (2017), 'FPGA-Based Reconfigurable Data Acquisition System for Industrial Sensors', EEE Transactions on Industrial Informatics, 13, p1503-1512.

Bayrak, H., (2018), Endüstri 4.0 Nedir [online]. Erişim Adresi: https://www.kolaybpm.com/2015/10/07/endustri-4-0-nedir/ (Erişim Tarihi: 28/11/2018).

Ben-Ari, M., & Mondada, F., (2018), 'Kinematics of a Robotic Manipulator', In: Elements of Robotics, Cham: Springer.

Bingül, Z., & Küçük, S., (2015), Robot Kinematiği, Kocaeli: Umuttepe Yayınları.

Bingül, Z., Ertunç, H.M. & Oysu C., (2005), 'Comparison of inverse kinematics solutions using neural network for 6R robot manipulator with offset', ICSC congress on computational intelligence methods and applications, IEEE.

Blum, C., & Li, X., (2008), 'Swarm Intelligence in Optimization'. In: Swarm Intelligence: Introduction and Applications. Heidelberg: Springer.

Botros, N.M., (2005), HDL Programming Fundamentals: VHDL and Verilog. Da Vinci Engineering Press.

Boz, A.F., & Çimen, M.E., (2017), 'Common Emitter BJT Amplifier Design using PSO, CS and FA', Cumhuriyet University Faculty of Science Journal, 38, p119-130.

Bulut, E., & Akçacı, T., (2017), 'Endüstri 4.0 ve İnovasyon Göstergeleri Kapsamında Türkiye Analizi', ASSAM Uluslararası Hakemli Dergi, 4, p55-77.

Caceres, C., Rosario, J. M., & Amaya, D., (2017), 'Approach of kinematic control for a nonholonomic wheeled robot using artificial neural networks and genetic algorithms', Proceeding of the International Bioinspired Intelligence Conference and Workshop (IWOBI). Funchal, Portugal.

Can, F. C., (2008), 'Analysis and Syntesis of Parallel Manipulator', Doktora Tezi, Teknoloji Enstitüsü, İzmir.

Casselman, S., Thornburg, M., & Schewel, J., (1995), 'Creation of hardware objects in a reconfigurable computer, International Workshop on Field Programmable Logic and Applications (s. 119-128). Springer, Berlin, Heidelberg.

Cavanagh, J., (2008), Digital Design and Verilog HDL Fundamentals, New York: CRC Press.

Cavanagh, J., (2015), Sequential Logic and Verilog HDL Fundamentals, New York: CRC Press.

Chen, J., & Lau, H. Y., (2016). 'Inverse kinematics learning for redundant robot manipulators with blending of support vector regression machines', In IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO) (pp. 267-272). IEEE.

Christi, R. F., (2015), FPGA Based System Design, CPLD and FPGA. Erişim Adresi: https://www.slideshare.net/maliktauqir/fpga-03cpldandfpga (Erişim Tarihi: 24/03/2019).

Cidderwar, S.S., & Babu, R., (2010), 'Comparison of RBF and MLP neural networks to solve inverse kinematic problem for 6R serial robot by a fusion approach', Engineering Applications of Artificial Intelligence, 23, p1083-1092.

Çavdar, T., Mohammed, M., & Milani, R. A., (2012), 'A New Heuristic Approach for Inverse Kinematics of Robot Arms', American Scientific Publishers, 19, p329-333.

Çavuşlu, M.A., & Kösten, M.M., (2015), VHDL ile Sayısal Tasarım ve FPGA Uygulamaları. İstanbul: Kodlab Yayınları.

Çavuşlu, M.A., Karakuzu, C., & Şahin, S., (2010), 'Parçacık Sürü Optimizasyonu Algoritması ile Yapay Sinir Ağı Eğitiminin FPGA Üzerinde Donanımsal Gerçeklenmesi', Journal of Polytechnic, 13, p83-92.

Çonkur, E. Ş., (2003), Gereğinden Çok Serbestlik Dereceli Robot Kollarının Yörünge Planlaması İçin Geliştirilmiş Bir Yazılım, 11. Ulusal Makina Teorisi Sempozyumunda sunulan bildiri. Gazi Üniversitesi, Ankara, 4-6 Eylül.

Deepak, B.B., & Parhi, D.R., (2016), 'Control of an automated mobile manipulator using artificial immune system', Journal of Experimental & Theoretical Artificial Intelligence, 28, p417-439.

Dekker, R., (2014), Electronic Design [online]. Erişim Adresi: https://www.electronicdesign.com/what-s-difference-between/what-s-difference-between-vhdl-verilog-and-systemverilog (Erişim Tarihi: 19/02/2019).

Dereli, S., & Köker, R., (2018), 'IW-PSO Aproach to the Inverse Kinematics Problem Solution of a 7-DOF Serial Robot Manipulator', Sigma J Eng & Nat Sci, 36, p77-85.

Dereli, S., & Köker, R., (2019a), 'A meta-heuristic proposal for inverse kinematics solution of 7-DOF serial robotic manipulator: quantum behaved particle swarm algorithm', Artificial Intelligence Review [online], p1-16, DOI: 10.1007/s10462-019-09683-x

Dereli, S., & Köker, R., (2019b), 'Calculation of the inverse kinematics solution of the 7-DOF redundant robot manipulator by the firefly algorithm and statistical analysis of the results in terms of speed and accuracy', Inverse Problems in Science and Engineering [online], p1-14, DOI: 10.1080/17415977.2019.1602124.

Dereli, S., Köker, R., Öylek, İ., & Ay, M., (2019), 'A Comprehensive Research on the Use of Swarm Algorithms in the Inverse Kinematics Solution', Journal of Politechnic, 22, p75-79.

Din, M., Pal, S.K., Muttoo, S.K., & Jain, A., (2016), 'Applying Cuckoo Search for analysis of LFSR based cryptosystem', Perspectives in Science, 8, p435-439.

Doğan, A. Y., (2006), 'Sfinks Dizi Şifreleme Algoritmasının VHDL ile Yazılımı ve FPGA Üzerinde Gerçeklenmesi', Yüksek Lisans Tezi, İTÜ Fen Bilimleri Enstitüsü, İstanbul.

Dorigo, M., Birattari, M., & Stutzle, T., (2007), 'Ant colony optimization', IEEE Computational Intelligence Society, 1(4), p28-39.

Programmable Array Logic [Online]. (2018), Erişim Adresi: https://www.electrical4u.com/programmable-array-logic/ (Erişim Tarihi: 04/02/2019).

El-Sherbiny, A., Elhosseini, M.A., & Haikal, A.Y., (2018), 'A new ABC variant for solving inverse kinematics problem in 5 DOF robot arm', Applied Soft Computing, 73, p24-38.

Eren, İ., (2006), 'Gereğinden Çok Serbestlik Dereceli Robot Kolu Kontrol Sistemi Tasarımı ve Uygulaması', Yüksek Lisans Tezi, Pamukkale Üniversitesi Fen Bilimleri Enstitüsü, Denizli.

Erkol, H. O., & Demirel, H., (2014), 'A VHDL application for kinematic equation solutions of multi-degree-of-freedom systems', Journal of Zhejiang University Science C, 15, p1164-1173.

Erkol, H. O., & Demirel, H., (2016), 'Design and implementation of an 18-dof six-legged robot: ABR1', International Journal of Advanced Research in Computer Engineering & Technology, 5, p1756-1762.

Farooq, U., Marrakchi, Z., & Mehrez, H., (2012), Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization. New York: Springer Science & Business Media.

Fister, I., Fister Jr. I., Yang, X.S., & Brest, J., (2013), 'A comprehensive review of firefly algorithms', Swarm and Evolutionary Computation, 13, p34-46.

Francis, S. (2018) Robotics and Automation [Online]. Erişim Adresi: https://roboticsandautomationnews.com/2018/07/16/kawasaki-offers-insight-into-how-industrial-robots-are-made/18312/ (Erişim Tarihi: 13/02/2019).

Gupta, V., Chittawadigi, R.G., & Saha, S.K., (2017), RoboAnalyzer: Robot Visualization Software for Robot Technicians. In Procedings Advances in Robotics. ACM.

Gümüş, S., (2017), 'Hareket Bilgisi (Kinematik)' in Genel Fizik 1. Ankara: Pegem Yayıncılık, pp. 75-151.

Hamid Toshani, M. F., (2014), 'Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: A Lyapunov-based approach', Robotics and Autonomous Systems, 62, p766-781.

Harrison, K. R., Engelbrecht, A. P., & Ombuki-Berman, B. M., (2016), 'Inertia weight control strategies for particle swarm optimization', Swarm Intelligence, 10, p267-305.

Hassan, H., & Anis, M., (2010), Low-Power Design of Nanometer FPGAs: Architecture and EDA, MA: Morgan Kaufman Publisher.

Huang, H.C., Chen, C.P., & Wang, P.R., (2012), 'Particle swarm optimization for solving the inverse kinematics of 7-DOF robotic manipulators', IEEE international conference on systems, man, and cybernetics, p3105-3110.

Hyder, Z., Siau, K., & Nah, F.F.H., (2018), Use of Artificial Intelligence, Machine Learning, and Autonomous Technologies in the Mining Industry. Thirteenth Midwest Association for Information Systems Conference. Saint Louis, Missouri.

Jazar, R.N., (2010), Theory of Applied Robotics: Kinematics, Dynamics, and Control, New York: Springer.

Joan, B., (2010), Difference between Verilog and VHDL [online]. Erişim Adresi: http://www.differencebetween.net/technology/difference-between-verilog-and-vhdl/ (Erişim Tarihi: 20/02/2019).

Kalra, P., & Prakash, N.R., (2003), 'A neuro genetic algorithm approach for solving inverse kinematics of robotic manipulators', IEEE International Conference on Systems, Man and Cybernetics, 2, p1979-1984.

Karaboğa, D., (2005), An idea based on honey bee swarm for numerical optimization. Technical Report, Erciyes Üniversitesi, Kayseri.

Karaboğa, D., (2011), Yapay Zekâ Optimizasyon Algoritmaları. Ankara: Nobel Yayıncılık.

Karaboğa, D., & Akay, B., (2009), 'A comparative study of Artificial Bee Colony algorithm', Applied Mathematics and Computation, 214, p108-132.

Karaboğa, D., & Akay, B., (2009), 'A servey: algorithms simulating bee swarm intelligence', Artif Intell Rev, 31, p61-85.

Karaboğa, D., & Baştürk, B., (2007), 'A powerful and efficient algorithm for numerical function optimization: Artificial Bee Colony (ABC) Algorithm', Journal of Global Optimization, 39, p459-471.

Kastensmidt, F., & Rech, P., (2016), FPGAs and Parallel Architectures for Aerospace Applications. Soft Errors and Fault-Tolerant Design [online]. DOI: 10.1007/978-3-319-14352-1.

Kelleci, B., (2017), VHDL ve Verilog ile Sayısal Tasarım. Ankara: Seçkin Yayıncılık.

Kennedy, J., & Eberhart, R., (1995), Particle swarm optimization. Perth, Australia: EEE international conference on neural networks, pp. 1942-1948.

Khan, M. S., & Khan, M. A., (2017), 'A Brief Survey on Robotics', International Journal of Computer Science and Mobile Computing, 6, p38-45.

Kim, K., Lee, J., & Choi, K., (2016), An energy-efficient random number generator for stochastic circuits. 21st Asia and South Pacific Design Automation Conference (ASP-DAC). Macau: IEEE, pp. 256-261.

Köker, R., (2013), 'A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization', Information Sciences, 222, p528-543.

Köker, R., & Çakar, T., (2016), 'A neuro-genetic-simulated annealing approach to the inverse kinematics solution of robots: a simulation based study', Engineering with Computers, 32, p553-565.

Krishna, G., & Roy, S., (2017), 'Fundamentals of FPGA Architecture', Advanced Engineering Technical and Scientific Publisher, 2, p12-30.

Kuon, I., & Rose, J., (2010), Quantifying and exploring the gap between FPGAs and ASICs. Springer Science & Business Media.

Küçük, S., Bingül, Z., (2014), 'Inverse kinematics solutions for industrial robot manipulators with offset wrists', Applied Mathematical Modelling, 38, p1983-1999.

Küçük, S., (2013), 'Energy minimization for 3-RRR fully planar parallel manipulator using particle swarm optimization', Mechanism and machine theory, 62, p129-149.

Küçük, S., Bingül, Z., (2006), 'Comparative study of performance indices for fundamental robot manipulators', Robotics and Autonomous Systems, 54, p567-573.

Lee, J., Bagheri, B., & Kao, H., (2015), 'A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems', Manufacturing Letters, 3, p18-23.

Lewis, S., & Cratsley, C., (2008), 'Flash signal evolution, mate choise and predation in fireflies', Annual Revision in Entomology, 53, p293-321.

Li, Y., Gai, K., Qiu, M., Dai, W., & Liu, M., (2017), 'Adaptive human detection approach using FPGA-based parallel architecture in reconfigurable hardware', Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.3923.

Lu, Y., (2017), 'Industry 4.0: A survey on technologies, applications and open research issues', Journal of Industrial Information Integration, 6, p1-10.

Mahanta, G. B., Deepak, B. B. V. L., Dileep, M., Biswal, B. B., & Pattanayak, S. K. (2019). Prediction of Inverse Kinematics for a 6-DOF Industrial Robot Arm Using Soft Computing Techniques. In Soft Computing for Problem Solving (pp. 519-530). Springer, Singapore.

Makas, H., (2015), 'Güncel En İyileme Algoritmalarının Paralel ve Birlikte Uygulamaları ve Performans Analizleri', Doktora Tezi, Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Sakarya.

Makryniotis, T., & Dasygenis, M., (2016), Rapid Implementation of Embedded Systems using Xilinx Zynq Platform, SouthEast European Design Automation, Computer Engineering, Computer Networks and Social Media Conference, ACM (pp. 6-10).

Marini, F., & Walczak, B., (2015), 'Particle swarm optimization (PSO). A tutorial', Chemometrics and Intelligent Laboratory Systems, 149, p153-165.

Mayorga, R., & Sanongboon, P., (2005), 'An Artificial neural network approach for inverse kinematics computation and singularities prevention of redundant robots', Journal of Intelligent and Robot Systems, 44, p1-23.

Merlet, J.P., (1996), 'Redundant Parallel Manipulators', Laboratory Robotics and Automation, 8, p17-24.

Mohamed, A. F., Elarini, M. M., & M.Othman, A., (2014), 'A new technique based on Artificial Bee Colony Algorithm for optimal sizing of stand-alone photovoltaic system', Journal of Advanced Research, 5, p397-408.

Momani, S., Abo-Hammour, Z. S., & Alsmadi, O. M., (2016), 'Solution of inverse kinematics problem using genetic algorithms', Applied Mathematics & Information Sciences, 10, p1-9.

Monmasson, E., Idkhajine, L., Cirstea, M.N., Bahri, I., Tisan, A., & Naouar, M.W., (2011), 'FPGAs in Industrial Control Applications', IEEE Transactions on Industrial Informatics, 7, p224-243.

Munden, R., (2004), ASIC and FPGA verification: a guide to component modeling. San Francisco: Elsevier.

Neythalath, N., Brandstötter, M., & Hofbaur, M., (2016), Redundancy resolution of a 9 DOF serial manipulator under hard task constraints. Symposium on Robot Design, Dynamics and Control, Springer, Cham, pp. 31-38.

Oyama, E., Agah, A., MacDorman, K.F., Maeda, T., & Tachi, S., (2001), 'A modular neural architecture for inverse kinematics model learning', Neurocomputing, 38, p797-805.

Önder, E., Ozdemir, M., & Yıldırım, B., (2013), 'Combinatorial Optimization Using Artificial Bee Colony Algorithm and Particle Swarm Optimization Supported Genetic Algorithm', Kafkas University Journal of Economics and Administrative Sciences Faculty, 4, p59-70.

Özdemir, M., (2017), 'Particle Swarm Optimization for Continuous Function Optimization Problems', International Journal of Applied Mathematics, Electronics and Computers, 5, p47-52.

Özkarakoç, M., (2009), 'Scara Robot Manipülatör ile Seçme ve Yerleştirme Uygulaması', Yüksek Lisans Tezi, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, İzmir.

Palangpour, P., Venayagamoorthy, G. K., & Smith, S. C., (2009), 'Particle Swarm Optimization: A Hardware Implementation', CDES, p134-139.

Pamuk, N., (2016), 'Kaotik ateşböceği optimizasyon algoritması kullanılarak termik güç santralleri etkisindeki ekonomik yük dağıtım problemlerinin çözümü', Kirklareli University Journal of Engineering and Science, 2, p38-59.

Pant, M., Thangaraj, R., & Abraham, A., (2008), A New Quantum Behaved Particle Swarm Optimization. The 10th Annual Conference on Genetic and Evolutionary Computation, ACM, pp. 87-94.

Parvez, H., & Mehrez, H., (2010), Application-specific mesh-based heterogeneous FPGA architectures. New York: Springer Science & Business Media.

Pieper, D.L., (1968), 'The kinematics of robots under computer control', Doktora Tezi, Stanford Üniversitesi.

Poli, R., Kennedy, J., & Blackwell, T., (2007), 'Particle Swarm Optimization', Swarm Intelligence, 1, p33-57.

Raj, N.J., Iyer, K., & Dash, A. K., (2016), Design, fabrication, kinematic analysis and control of a 3-DOF serial manipulator. International Conference on Next Generation Intelligent Systems (ICNGIS), IEEE, Kottayam, India.

Robotis Bioloid Series - Premium and GP (2012), Erişim Adresi: http://en.robotis.com/model/page.php?co_id=prd_premium (Erişim Tarihi: 02/01/2019).

Rodríguez, A., Valverde, J., Portilla, J., Otero, A., Riesgo, T., & Torre, E.D., (2018), 'Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework', Sensors, 18(6), p1-30.

Rokbani, N. & Alimi A.M., (2015), 'Inverse Kinematics using Particle Swarm Optimization, a Statistical Analysis', Procedia Engineering, 64, p1602-1611.

Rokbani, N., Casals A. & Alimi A.M., (2015), 'IK-FA, a New Heuristic Inverse Kinematics Solver Using Firefly Algorithm' in: Azar A., Vaidyanathan S. (eds) Computational Intelligence Applications in Modeling and Control. Studies in Computational Intelligence, vol 575. Springer, Cham, pp. 369-395.

Ruiz, A.G., Santos, J.C., Croes, J., Desmet, W., & Silva, M.M., (2018), 'On redundancy resolution and energy consumption of kinematically redundant planar parallel manipulators', Robotica, 36, p809-821.

Sadhu, A.K., Konar, A., Bhattacharjee, T., & Das, S., (2018), 'Synergism of Firefly Algorithm and Q-Learning for Robot Arm Path Planning', Swarm and Evolutionary Computation, 43, p50-68.

Sarıtaş, E., & Karataş, S., (2015), Her Yönüyle FPGA ve VHDL. Ankara: Palme Yayıncılık.

Schreiber, L.T., & Gosselin, C., (2018), 'Kinematically redundant planar parallel mechanisms: Kinematics, workspace and trajectory planning', Mechanism and Machine Theory, 119, p91-105.

Seker, A.A., & Hocaoglu, M.H., (2013), 'Artificial Bee Colony Algorithm for Optimal Placement and Sizing of Distributed Generation', International Conference on Electrical and Electronics, Bursa, pp. 127-131.

Serrano, J., (2008), 'Introduction to FPGA design', CAS - CERN Accelerator School: Course on Digital Signal Processing, Sigtuna, Sweden, pp. 231-247.

Sheldon, D., Kumar, R., Lysecky, R., Vahid, F., & Tullsen, D., (2006), 'Application-Specific Customization of Parameterized FPGA Soft-Core Processors', IEEE/ACM

International Conference on Computer Aided Design, San Jose, CA, USA, pp. 170-173.

Shi, H., Chen, J., Pan, W., Hwang, K., & Cho, Y.Y., (2018), 'Collision Avoidance for Redundant Robots in Position-Based Visual Servoing', IEEE Systems Journal, 99, p1-11.

Siau, K., & Wang, W., (2018), 'Building Trust in Artificial Intelligence, Machine Learning, and Robotics', Cutter Business Technology Journal, 31, p47-53.

Sklyarov, V., Skliarova, I., & Sudnitson, A., (2012), 'Methodology and international collaboration in teaching reconfigurable systems', IEEE Global Engineering Education Conference (EDUCON), Marrakech, Morocco, pp. 1143-1152.

Song, C., Xie, S., Zhou, Z., & Hu, Y., (2015), 'Modeling of pneumatic artificial muscle using a hybrid artificial neural network approach', Mechatronics, 31, p124-131.

Soylak, M., Oktay, T., & Turkmen, İ., (2017), 'A simulation-based method using artificial neural networks for solving the inverse kinematic problem of articulated robots', Journal of Process Mechanical Engineering, 231, p470-479.

Spong, M.W., Hutchinson, S., & Vidyasagar, M., (2006), Robot modeling and control (Cilt 3). New York: Wiley.

Staicu S., (2019), 'Dynamics of Constrained Robotic Systems' in Dynamics of Parallel Robots. Springer, Cham, pp. 121-146.

Stavinov, E., (2011), 100 Power tips for FPGA designers. Evgeni Stavinov.

Sulaiman, N., Obaid, Z.A., Marhaban, M.H., & Hamidon, M.N., (2009), 'Design and Implementation of FPGA-Based Systems - A Review', Australian Journal of Basic and Applied Sciences, 3, p3575-3596.

Sun, J., Fang, W., Palade, V., Wu, X., & Xu, W., (2011), 'Quantum-behaved particle swarm optimization with Gaussian distributed local attractor point', Applied Mathematics and Computation, 218, p3763-3775.

Tan, W., Dong, S., Liu, X., & Wang, B., (2015), 'An Improved PSO Algorithm Based on SA and Quantum Theory and Its Application', International Journal of Database Theory and Application, 8, p189-198.

Teubner, J., & Woods, L., (2013), Data Processing on FPGAs, Morgan & Claypool Publishers.

Toshani, H., & Farrokhi, M., (2014), 'Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: A Lyapunov-based approach', Robotics and Autonomous Systems, 62, p766-781.

Xi, W., Wang, Y., Chen, B., & Wu, H., (2019), 'Iterative learning control of robot based on artificial bee colony algorithm', Journal of Systems and Control Engineering, DOI: 10.1177/0959651818824202.

Wang, L.T., & Chen, C.C., (1991), 'A combined optimization method for solving the inverse kinematics problems of mechanical manipulators', IEEE Transactions on Robotics and Automation, 7, p489-499.

Wang, X.S., Xu, S.F., & Hao, J.F., (2001), 'New inferential method and efficient solutions for inverse kinematics of robot MOTOMAN', Journal of China University of Mining Technology, 30, p73-86.

Wang, Y., Shi, Y., Ding, D., & Gu, X., (2016), 'Double global optimum genetic algorithm–particle swarm optimization-based welding robot path planning', Engineering Optimization, 48, p299-316.

Wilson, P., (2015), Design Recipes for FPGAs: Using Verilog and VHDL, Elsevier Publishers.

XYZ, A., (2018), How Are Industrial Robots Built? A Guide on the Components and the Movement of Robot Arms [online]. Erişim Adresi: https://robotics.kawasaki.com/ja1/xyz/en/1804-03/ (Erişim Tarihi: 31/01/2019).

Yang, X.S., (2008), Nature-Inspired Metaheuristic Algorithms. Luniver Press.

Yang, X.S., (2009), 'Firefly Algorithms for Multimodal Optimization', International Conference on Stochastic Algorithms: Foundations and Applications. Sapporo, Japan, pp. 169-178.

Yang, X.S., & He, X., (2013), 'Firefly Algorithm: Recent Advances and Application', Int. J. Swarm Intelligence, 1, p36-50.

Zhang, P. Y., Lü, T. S., & Song, L. B., (2005), 'RBF networks-based inverse kinematics of 6R manipulator.' The International Journal of Advanced Manufacturing Technology, 26, p144-147.

Zhang, Y., Wang, S., & Ji, G., (2015), 'A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications', *Hindawi Mathematical Problems in Engineering*, DOI:10.1155/2015/931256.

ÖZGEÇMİŞ

Serkan Dereli, 25/01/1980'da Amasya'da doğdu. İlk, orta ve lise eğitimini Tokat'ta tamamladı. 1997 yılında Turhal Endüstri Meslek Lisesi'nden mezun olduktan iki sene sonra başladığı Gazi Üniversitesi Bilgisayar Sistemleri Öğretmenliği Bölümü'nü 2003 yılında bitirdi. 2013 yılında Sakarya Üniversitesi Sakarya Meslek Yüksekokulu'nda Bilgisayar Programcılığı Öğretim Görevlisi olarak göreve başlamadan önce MEB'de öğretmenlik ve özel sektörde yazılım ve veritabanı uzmanı olarak görev yaptı. 2013 yılında Sakarya Üniversitesi Mekatronik Mühendisliği Bölümü'nde yüksek lisans eğitimine, 2015 yılında ise doktora eğitimine başladı. Halen Sakarya Uygulamalı Bilimler Üniversitesi Adapazarı Meslek Yüksekokulu'nda öğretim görevlisi olarak görevine devam etmektedir.