ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

SEZGİSEL YÖNTEMLERİN SÜRÜ ROBOTLARI GÖREV PAYLAŞIMINDA KULLANILMASI

Mehmet ÖZASLAN

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2021

ÖZET

Yüksek Lisans Tezi

SEZGİSEL YÖNTEMLERİN SÜRÜ ROBOTLARI GÖREV PAYLAŞIMINDA KULLANILMASI

Mehmet ÖZASLAN

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Mehmet Serdar GÜZEL

Sezgisel algoritmalar karmaşık, doğrusal olmayan problemlerin çözümündeki başarılarıyla son zamanda birçok alanda kullanılmaktadır. Sürü robotların hareket planlamalarında çok parametreli modellere ihtiyaç duyulmaktadır. Bu parametrelerin optimum değerlerinin bulunması problemi klasik algoritmalarla çözülemeyecek karmaşıklıktadır.

Bu tez çalışmasında RoboCode savaş simülatörü ile sürü robotların hareketlerinin sezgisel optimizasyon algoritmalarından genetik algoritma ve parçacık sürü optimizasyon algoritması kullanarak geliştirilmesi, geliştirilen iki farklı robot türünden oluşan sürülerin aynı rakip robotlarla savaştırılması sureti ile genetik algoritma ve parçacık sürü optimizasyon algoritmasının etkinliğinin değerlendirilmesi amaçlanmıştır.

RoboCode ortamında, aynı rakiplerle, aynı iterasyon (PSO için güncelleme) sayısında aynı popülasyon (PSO için sürü) genişliğindeki koşullarda genetik algoritma ve parçacık sürü optimizasyonu algoritması ile robotlar eğitilmiştir. Bu her iki robotun eğitimi esnasında farklı karakteristikteki aynı rakip robotlarla karşılaştırılmış ve sürülerin eğitim esnasındaki uygunluk(başarı) değerleri izlenmiştir.

Robotların eğitimi sonrasında genetik algoritma ile eğitimin daha uzun sürede gerçekleştiği görülmüştür. En iyi bireysel başarı değerine ve sürü ortalama başarı değerine genetik algoritma ile ulaşılmıştır. Buna karşın parçacık sürü optimizasyon algoritması ile sürü ortalama başarı değerinin en yüksek başarıyı gösteren bireyin başarı değerine yaklaşmaya devam ettiği görülmüştür.

Aralık 2021, 54 sayfa

Anahtar Kelimeler: Sürü robotlar, Sezgisel algoritmalar, genetik algoritma, parçacık sürü optimizasyon algoritması

ABSTRACT

Master Thesis

USAGE OF HEAURISTIC METHODS IN TASK SHARING OF SWARM ROBOTS

Mehmet ÖZASLAN

Ankara University
Graduate School of Natural and Applied Sciences
Depaetment of Computer Engineering

Supervisor: Doç.Dr. Mehmet Serdar GÜZEL

Heuristic algorithms have been used in many fields recently with their success in solving complex, nonlinear problems. Multi-parameter models are needed for motion planning of swarm robots. The problem of finding the optimum values of these parameters is so complex that it cannot be solved by classical algorithms.

In this thesis, it is aimed to develop the movements of the swarm robots using the heuristic optimization algorithms genetic algorithm and particle swarm optimization algorithm with the RoboCode battle simulator, and to evaluate the effectiveness of the genetic algorithm and particle swarm optimization algorithm by fighting the swarms consisting of two different robot types with the same rival robots.

In the RoboCode environment, robots are trained with genetic algorithm and particle swarm optimization algorithm in conditions of same population (swarm for PSO) width with same competitors, same number of iterations (updates for PSO). During the training of these two robots, they were compared with the same rival robots with different characteristics and the fitness (success) values of the swarms during the training were monitored.

After the training of the robots, it was observed that the training with the genetic algorithm took longer. The best individual success value and the average success rate of the herd were reached by genetic algorithm. On the other hand, with the particle swarm optimization algorithm, it has been observed that the average success value of the swarm continues to approach the success value of the individual with the highest success.

December 2021, 54 pages

Key Words: Swarm robots, Heuristic algorithms, genetic algorithm, particle swarm optimization algorithm

TEŞEKKÜR

Bu çalışmanın yürütülmesi ve yüksek lisans dersleri aşamasında; değerli bilgi ve önerileriyle bana destek olan danışman hocam Doç. Dr. Mehmet Serdar GÜZEL başta olmak üzere bölümümüzdeki tüm hocalarıma, desteğini hiçbir zaman esirgemeyen eşim Nurgül Ece ÖZASLAN'a, eğitim hayatım boyunca hep yanımda olan aileme sonsuz teşekkürlerimi ve şükranlarımı sunarım.

Mehmet ÖZASLAN
Ankara, Aralık 2021

İÇİNDEKİLER

KISALTMALAR DİZİNİ

GA	Genetik algoritma
GP	Genetik programlama
PSO	Parçacık sürü optimizasyonu
PÖ	Pekiştirmeli Öğrenme
YSA	Yapay Sinir Ağları

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Robot teknolojileri son yıllarda çok hızlı bir gelişmiştir. Bu gelişimin bir sonucu olarak birçok alanda insanların yaptığı işleri robotlar yapar hale gelmiştir. Robotların bu gelişiminde en önemli rolü kullanılan algoritmalar almaktadır. Algoritmalar, robotların verimliliğini en üst seviyeye çıkaracak şekilde tasarlanır(Obaidy ve Ayesh 2008). Gelişen yapay zekâ yaklaşımlarına paralel olarak konvansiyonel olarak programlanmış, sadece belirli işleri yapmak üzere tasarlanmış robotların yerini yapay zekâ algoritmaları kullanan robotlar almaktadır.

Robotların programlanmasında kullanılan başlıca yapay zeka algoritmaları optimizasyon algoritmalarıdır. Optimizasyon, var olan bir problemin mevcut çözümleri arasından en iyisini seçme işidir(Alba, 2005). Optimizasyonda probleminin çözülebilmesi için problemin doğasına uygun kurallar ve bağlantılar kullanılarak matematiksel modeller geliştirilir. Bu modellerde karar değişkenleri, amaç fonksiyonu ve kısıtlayıcılar önemlidir(Murty 2003). Burada amaç fonksiyonu optimize edilecek fonksiyon, karar değişkenleri amaç fonksiyonundaki değişkenler ve kısıtlayıcılar da amaç fonksiyonun sağlanmasında uyulacak değer ya da fonksiyonlardır.

Bir optimizasyon modelinin çözümlenmesinde çeşitli algoritmalar kullanılır. Bu algoritmalar belirleyici(deterministtik) ve rastsal(skolastik) olarak ikiye ayrılır. Deterministtik algoritmalarda parametreler sabit sayılardır. Bu algoritmalarda rastsal üretilmiş değerler parametre olarak kullanılmaz. Stokastik algoritmalarda ise parametreler rastgele üretilmiş değerlerdir(Yang, 2013). Stokastik algoritmalar sezgisel veya meta-sezgisel olarak da isimlendirilir.

Karmaşık problemlerin modellenmesinde sezgisel optimizasyon algoritmaları geleneksel algoritmalara göre daha kısa sürede kabul edilebilir çözümler sunmaktadır. Bundan dolayı karmaşık optimizasyon problemlerini çözerken sezgisel optimizasyon algoritmalarını sıklıkla kullanılmaktadır. Bu algoritmalar her zaman kesin çözümü vaat etmezler ancak çözüm için kısıtlı zaman ve kaynağın olduğu durumlarda tercih edilmektedir.

Sezgisel algoritmaların son yıllarda kullanımın artması bir başka deyişle bu tip algoritmaların başarılı kılan en önemli faktör yerel aramalar sonucu kendi elde ettiği değerleri ve komşuluğundaki ajanlar tarafından elde edilen değerleri hafızasında tutmasıdır. Hafızasında tuttuğu bu değerler doğrultusunda bir sonraki iterasyonda seçeceği değerlerin rastlantısal değil de bilinçli olmasına dolayısı ile algoritma başarısının artmasına neden olur.

Sezgisel optimizasyon algoritmaları biyolojik tabanlı, sosyal tabanlı ve fiziksel tabanlı olarak gruplara ayrılabilir(Simha ve Carnahan 2001). Sezgisel optimizasyon algoritmalarından genetik algoritmalar (GA) biyolojik tabanlı, parçacık sürü optimizasyon (PSO) algoritmaları da sosyal tabanlı sezgisel algoritmalar grubunda değerlendirilir. Görüldüğü gibi gerek biyolojik tabanlı optimizasyon algoritmaları dolayısı ile GA gerek ise sosyolojik tabanlı optimizasyon algoritmaları dolayısı ile PSO canlılığı taklit eder.

Optimizasyon algoritmalarından önemli bir kısmı evrimsel algoritma türlerinin uygulanması ile çözülmüştür. Genetik algoritma en sık kullanılan sezgisel algoritmadır. GA John Holland tarafından önerilmiştir (Holland,1973). GA canlıların evriminden esinlenerek geliştirilmiştir. GA, istenen çözümleri elde etmek için iteratif süreci kullanan global arama tabanlı sezgisel bir algoritmadır. GA, kalıtım, seleksiyon, çaprazlama veya rekombinasyon, mutasyon ve üreme gibi çeşitli biyolojik teknikleri kullanır.

Parçacık sürü algoritması (PSO), Kennedy ve Ebernet tarafından 1995 yılında tanıtılmış bir algoritmadır. Sezgisel türdeki bu algoritmanın esin kaynağı doğadaki kuşlar, balıklar gibi sürü halinde yaşayan hayvanlardır(Kennedy ve Eberhart, 1995). Kuşların yem ararken ki davranışları incelenmiş, sürünün en başındaki kuşun yiyeceğe en yakın olabileceği varsayımına göre diğer kuşların pozisyon aldığı saptanmıştır. Kuşların bu davranışları modellenmiş ve PSO algoritması ortaya çıkmıştır.

GA ve PSO bir çok optimizasyon probleminin çözümünde kullanılmaktadır. Bu iki sezgisel optimizasyon algoritması karşılaştırıldığında PSO'nun GA'ya göre daha kolay

uygulanabildiği görülmektedir. PSO, GA'ya göre daha az parametre buna bağlı olarak daha az sayıda yineleme kullanmaktadır (Navarro ve Matia 2012).

GA ayrık problemin çözümünde oldukça başarılıdır. Bu GA'nın ayrık yapıda olmasından kaynaklanır. Buna karşın PSO süreklilik arz eder. Bir ayrık problem PSO ile çözümlenmek isteniyorsa öncelikle bazı yapısal değişikliklerin yapılması gerekebilir. PSO değişkenleri, mevcut konumlarına ve hız vektörüne bağlı olarak çeşitli değerler alabilirler. PSO global optimumu bulmaya çalışırken, GA genellikle problemin global optimumu yerine yerel bir optimum veya hatta keyfi noktalara yakınsar(Shabir ve Singla,2016). GA ve PSO'ya ait performans ve işlem açısından karşılaştırma sonuçları çeşitli testler sonucunda elde edilmiştir. Bunlardan bazıları Glonski'nin hız düşürücü testi, uydu tasarımı ve teleskop dizisidir(Panduro vd.2009).

Bu tez kapsamında sezgisel optimizasyon algoritmalarının sürü robotların görev paylaşımları üzerine uygulanabilirliği üzerine çalışmalar yapılmıştır. Bu doğrultuda RoboCode isimli simülasyon savaş oyunu üzerinde GA ve PSO optimizasyon algoritması uygulanarak algoritmaların performansı test edilmiştir.

RoboCode IBM tarafından 2001 yılında Java programlama dili tabanında geliştirilen, açık kaynak kodlu bir savaş simülatörü programıdır. Bu simülatörde amaç, iki boyutlu ortamda platform tarafından kullanıcılara sunulan sınıfları kullanarak bir savaş robotu programlamak ve bu robotları, yine bu simülatör tarafından sağlanan ve arena ismi verilen ortamda birbirleriyle mücadele ettirerek kullanıcının programladığı robotun performansını ölçmektir. RoboCode ortamın esas amacı olan programlama eğitiminin etkili bir şekilde yapılmasıdır. Bu konuda çok sayıda yayınlanmış çalışma bulunmaktadır. RoboCode ortamı basit olarak robotların mücadelesini sağlamaktadır. Her bir robot *gövde, namlu* ve *radar* isimleri verilen parçalardan oluşmakta olup simülasyonda belirli ara yüzlerle kullanıcıların bu parçaları kontrol etmesi sağlanmıştır.

RoboCode için bir ajan ilk olarak Kobayashi ve Uchida tarafından tasarlanmıştır. Bu yaklaşım 2002 roboCode japon kupasında yarışan robota uygulanmıştır(Kobayashi ve Uchida,2003). Öte yandan Alaiba ve Rotaru'nunda prolog dilinde tank ajanları geliştirdikleri çalışmalar bulunmaktadır (Alaiba ve Rotaru, 2008).

Bu tez, temel olarak iki kısımdan oluşmaktadır. İlk kısımda genetik optimizasyon algoritması, parçacık sürü optimizasyon algoritması ve RoboCode hakkında birtakım araştırmalar ve bilgilendirmeler mevcuttur. İkinci kısım ise GA ve PSO ile eğitilmiş robotun RoboCode ortamına uygulanması ve robotların savaş sonuçlarına ait test sonuçlarını içermektedir.

1.1 Robocode Robotları Geliştirmek İçin Sezgisel Yöntemler Kullanılan Önceki Çalışmalar

Enistein genetik algoritmalar üzerine yaptığı RoboCode çalışmalarında bazı istenilen değerle ulaşmıştır. Ancak robotların eğitiminde YSA'nın daha başarılı sonuçlar verebileceğini iddia etmektedir(Einstein,2003).

Gade ve ark. Robocode plattformunda pekiştirmeli öğrenme, genetik programlama ve YSA'yı kullanmışlardır. Sinir ağı güdümlü hedefleme sistemlerinin başarılı olduğunu belirtirken, bunu Robocode'daki çarpık uygunluk ortamı nedeniyle oldukça etkisiz bir strateji olan rastgele hedefleme ile karşılaştırıyorlardı(Gade,2003). Robot eğitiminde YSA'yı kullanan bir diğer çalışma Czajkowskia ve Patana tarafından yapılmıştır. Çalışmalarında robot eğitiminin YSA ile yapılması yanı sıra Back-Propagation, Conjugate Gradient ve LevenbergMarquardt eğitim algortimaları uygulamalarını da karşılaştırmışlardır (A. Czajkowskia, K Patana, 2009) .

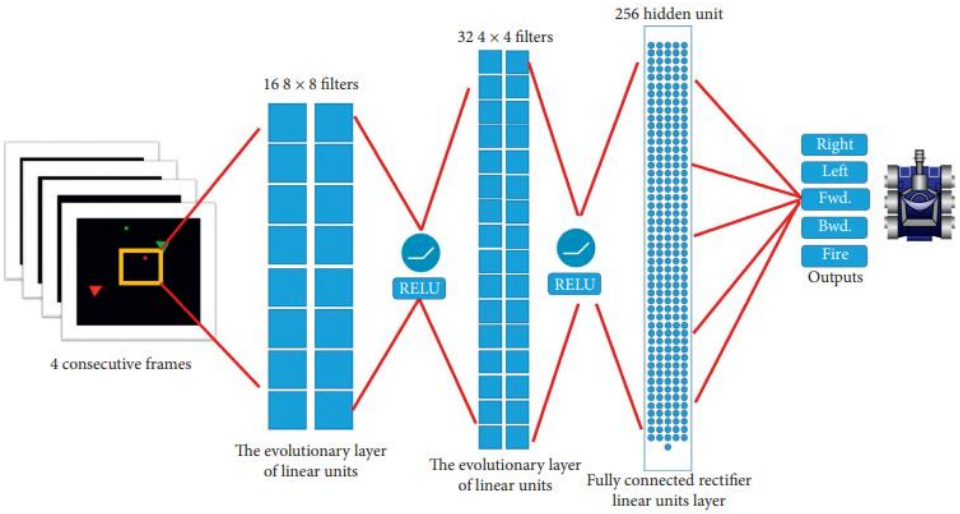
Wyatt ve Klein 2003 yılında yaptıkları çalışmada RoboCodeortamında GA'yı ilk defa uyguladılar (D. Wyatt ve D. Klein,2003). Hong ve Cho, BugBear adlı rekabetçi bir robota karşı ara sıra iyi performans gösterebildiğinden, Robocode'a biraz daha başarılı bir genetik algoritma uygulamaya çalıştı BugBear 2002'de şampiyon robotlardan biriyken, uluslararası rekabete açık robotların karmaşıklığı ve yetkinliği o zamandan beri önemli ölçüde arttı(Hong ve Cho,2004).

Shichel, Ziserman & Sipper, Genetik algoritmayı RoboCode robotlarında kullandı. Bu yöntemler geliştirdiği robot ile Haikubut liginde elle kodlanmış rakipleri arasında üçüncü olmuştur(Shichel,2005). Inja, GA ile eğitilmiş robotların başarılarını ispatlamış ancak global en iyiye ulaşamamıştır(M. Inja,2012). Harper RoboCode için GA

çalışmaları yapmış, robotun başarısını arttırmak için pekiştirmeli öğrenme ve yapay sinir ağı kullanmıştır. Bu sayede eğitilen robot rakibi karşısında %74.8 başarı elde etmiştir(Harper,2014).

Nielsen ve Jensen RoboCode robotlarının eğitilmesi için nöroenrim, YSA ve PÖ kullandılar. Çalışmalarında bu tür makine öğrenmesi yöntemleri ile robotların eğitilebileceğini ortaya koydular(Nielsen ve Jensen, 2011). Degryse, RoboCode için PÖ ve YSA ile robotlar eğitmiştir. Çalışmalarında RoboCode platformu için çok katmanlı yapay sinir ağlarının kullanımını incelemiştir (Degryse,2017). Son zamanlarda yapılan çalışmada Kayakokü ve Güzel PÖ tabanlı algoritmalar kullanılarak eğitilen robotların başarısını ortaya koymuşlardır(Kayakoku ve Güzel,2021).

RoboCode ortamında Parçacık Sürü Optimizasyon Algoritmasının direk olarak değil de yapay sinir ağlarının bağlantılarının ağırlıklarını optimize edilmesi suretiyle kullanıldığı görülmektedir. Bu konuda (Guesgen ve Shi,2006) ve (Alexiev, 2013) çalışmalarını görmekteyiz.



Şekil 1.1 Evrişimsel Q ağı modeli, Kayakoku ve Guzel,2006

2. SEZGİSEL ALGORİTMALAR

2.1 Parçacık Sürü Optimizasyonu

PSO, sürüyü oluşturan parçacıkların tek başlarına yapamadıkları bir görev için sürü zekâsı ile hareket etmek suretiyle en iyi sonucu ulaşma sürecidir. Bu süreçte parçacık kendi bilgisini kendi deneyimleri ve grubun ulaştığı bilgilerden istifade ederek geliştirir. Bütün parçacıkların bu gelişimi göz önüne alındığında sürünün en iyi çözüme doğru hareketi söz konusu olur(Campo vd. 2006).

PSO'nun ilk kullanım alanı film endüstrisi alanında olmuştur. Bilgisayar animasyonlarında bulut veya patlama gibi nesneleri içeren görüntüleri birleştirme problemi bu sektördeki karmaşık problemlerdendir. Bu problemin çözümünde çok sayıdaki hareketli parçacığın karmaşık davranışları meydana getirmesi fikri üzerinde durulmuştur(Reeves,1983). PSO sistemi rastsal olarak konumlanmış noktalar (parçacıklar) üretir. Noktaların başlangıç değerleri rastsaldır. Bu parçacıklar Stokastik olarak hareket ederler. Her parçacığın başlangıç hızı da yine sistem tarafından belirlenir. Bu amaçla parçacıklara hız vektörü de tanımlanır.

Hız vektörleri iteratif olarak belli aralıktaki rastsal değerleri alabilirler. Her bir parçacık sahip olduğu hız vektörüne göre belirli kurallar içerisinde bulunduğu konumdan yeni bir konuma hareket ettirilir.

Reynolds(1987), sürü algoritmaları çalışmalarında Reevs'in parçacık sistemi yaklaşımını temel almıştır. Bunun yanında Reynolds, Reevs'e oryantasyon ve iç cisim kavramlarının ilave etmiştir. Bu ilaveler sayesinde bireysel cisimlerden sürü kurallarının takip eden cisimler meydana gelmiştir. Bu sayede cisimler yakınlarındaki cisimlerle çarpışmaktan kaçınma; birbirlerinin hız vektörlerini uygunlaştırmak için çaba sarf etme ve birbirlerine yakın kalmaya çalışma gibi sürüye ait özellikler uygulanabilir olmuştur. Bireylerin zekasını arttıran bu modelin geliştirilmesi, bireysel yörüngelerin oluşturulma ihtiyacını ortadan kaldırmıştır(Karaboğa,2017).

Reynolds'tan sonra PSO'ya Kennedy ve Eberhant'ın (1995) katkıda bulunduğunu görmekteyiz. Bu katkı sosyal davranışlar eksenindedir. Sosyal davranışlar açısından Heppner ve Grenander(1990) tarafından basit hedefler önerilmişti. Kennedy ve Eberhant bu basit hedeflere alternatif olarak yiyecek aramayı teklif etmiştir. Bu parçacık sürü optimizasyonu belirli standartlara taşımıştır.

PSO algoritmasına göre problemin çözümü için parçacıklardan oluşan bir sürü kullanılır. Burada sürünün her bir parçacığı probleme ait bir çözümü içeren değerleri taşır. Evrimsel optimizasyon algoritmalarıyla karşılaştırıldığında, sürü popülasyona, parçacıklar ise popülasyondaki bireylere karşılık gelmektedir.

PSO sisteminde parçacıklar çok boyutlu bir uzayda hareket halindedirler. Bu hareketin kaynağı kendisinin ve komşularının tecrübelerinin ortaya çıkardığı dinamiktir. Örneğin $x_i(t)$, t. Zaman adımında uzaydaki i. Parçacığın pozisyonunu göstersin tersi ifade edilmedikçe t ayrık zaman adlarını ifade edecektir. Parçacığın pozisyonu, mevcut pozisyonuna $v_i(t)$ hızı ilave edilmek suretiyle değiştirilir(Engelbrecht,2006).

$$x_i(t+1)= x_i(t)+ v_i(t+1) \qquad (1)$$

Hız vektörünün görevi parçacığın optimizasyonunu yönlendirerek deneysel açıdan bilgisini yansıtmak ve sosyal açıdan komşularından almış olduğu bilgileri barındırmaktır. Burada bahsi geçen deneysel bilgiden kasıt bilişsel bileşendir. Bilişsel bileşen, parçacığın süreç boyunca bulduğu en iyi konum ile hali hazırdaki konumu düşünüldüğünde bu konumlar arasındaki uzaklık ile doğru orantılıdır. Denklemdeki sosyal bileşen ise parçacığın süreç boyunca sürüdeki parçacıklar ile etkileşimi sonucu elde ettiği bilgilerden oluşur.

En iyileme süreci parçacıkların alacağı rastgele değerler ile başlatılır. Yinelemeli olarak güncellemeler vasıtasıyla optimum çözüm bulunmaya çalışılır. Optimum çözüme ulaşmak için her yinelemede parçacık konumları iki değere (pozisyona) göre güncellenir. Bunlar bilişsel bileşen ve sosyal bileşendir. Bilişsel bileşen parçacığın mevcut iterasyona kadar parçacığın ulaştığı en iyi uygunluk derecesidir. Bu değere

pbest(bireysel eniyi) olarak da adlandırılmaktadır. İkinci pozisyon, sosyal bileşen, ise sürü tarafından mevcut iterasyona kadar olan ulaşılan en iyi uygunluk değeridir. Bu değer gbest (küresel eniyi) olarak adlandırılır.

PSO'da her parçacık sürüdeki bütün parçalar ile komşudur. Bu komşuluk durumunda bir parçacığın hızındaki değişme sürüdeki diğer tüm parçacıkların etkisi ile oluşmuş bir bilgiyi temel alır. Bir sürüdeki parçacıkların sahip olduğu uygunluk değerlerinden en yükseğine "gbest" (sosyal bileşen) denir. Formül 2'de gbest $\hat{y}(t)$ olarak gösterilmiştir. PSO için i. Parçacığın hızı aşağıdaki gibi hesaplanır(Engelbrecht,2006).

$$v_{ij}(t+1)= v_{ij}(t)+c_1 r_{1j}(t)[y_{ij}(t)-x_{ij}(t)]+ c_2 r_{2j}(t)[\hat{y}_j(t)-x_{ij}(t)] \quad (2)$$

Burada;

$v_{ij}(t)$: t zaman adımında j=1,…,n boyutunda i. parçacığın hızı;

$x_{ij}(t)$:t zaman adımında j boyutunda i. parçacığın pozisyonu;

c_1 ve c_2 :bilişsel ve sosyal bileşenlerin etkisini belirleyen pozitif ivme sabitleri

$r_{1j}(t)$ ile $r_{2j}(t)$: [0,1] aralığında üniform dağılımından üretilmiş rastsal sayılardır.

$r_{1j}(t)$ ve **$r_{2j}(t)$** rastsal sayıları algoritmaya stokastik bir özellik sağlamaktadır.

c_1 parametresi parçacığın kendi tecrübesine göre hareketini etkiler. Bilişsel katsayı olarak ta adlandırılır. c_2 ise sürüdeki diğer parçacıkların tecrübesinin parçacık üzerine etkisini etkileyen katsayıdır. Sosyal katsayı olarak ta adlandırılır. Gerek c_1'e gerekse c_2'ye verilecek değer parçacığın hareketini doğrusal olarak etkileyecektir. Her parçacığı pbest ve gbest konumlarına doğru hareketini ivmelendirmektedir. Yüksek değer verildiğinde parçacığın hedefe doğru hareketini hızlandırırken beklenmedik sorunlara sebep olabilmektedir. Değerin düşük seçilmesi ile de parçacık hedefe yönlenmeden önce uzak konumlarda dolaşarak çözümü geciktirir.

Parçacık i ile ilgili en iyi kişisel pozisyon y_i, parçacık tarafından başlangıçtan itibaren erişilmiş en iyi pozisyondur. Minimizasyon problemleri dikkate alındığında bir sonraki zaman adımında kişisel en iyi pozisyon aşağıdaki gibi hesaplanır

$$y_i(t+1) = \begin{cases} y_i(t) & , f(\mathrm{x_i}(\mathrm{t}+1)) \geq f(y_i(t)) \\ \mathrm{x_i}(\mathrm{t}+1) & , f(\mathrm{x_i}(\mathrm{t}+1)) < f(y_i(t)) \end{cases}$$

Burada $f: R_x^n \rightarrow R$ uygunluk fonksiyonudur. Uygunluk fonksiyonu, karşılık gelen çözümün optimuma ne kadar yakın olduğunu ölçmektedir. t. zaman adımında küresel en iyi pozisyon ŷ(t) ise aşağıdaki formülle tanımlanır.

$$\hat{y}(t) \in \{y_0(t), \dots, y_{ns}(t)\} \mid f\big(\hat{y}(t)\big) = \min\{f\big(\, y_0(t)\big), \dots, f\big(\, y_{ns}(t)\big)\}$$

Burada n_s , sürüdeki toplam parçacık sayısını temsil etmekte ve denklemde yazılan tanım, ŷ'nin şu ana kadar herhangi bir parçacık tarafından keşfedilen en iyi pozisyon olduğunu ifade etmektedir. Aynı zamanda en iyi pozisyon mevcut sürünün parçacıklarından seçilebilir. Bu durumda ifade aşağıdaki gibi yazılabilir.

Adım 1. Nx boyutlu S sürüsünün oluştur.

Adım 2. Kontrol parametre değerlerini ata

REPEAT

Adım 3. Sürüdeki her parçacık için (i=1, … ,n_s);

pBest belirle:

Eğer pBest < bulunan değer **ise** bulunan değeri PBest yap

gBest belirle:

Eğer gBest < mevcut değer **ise** bulunan değeri gBest yap

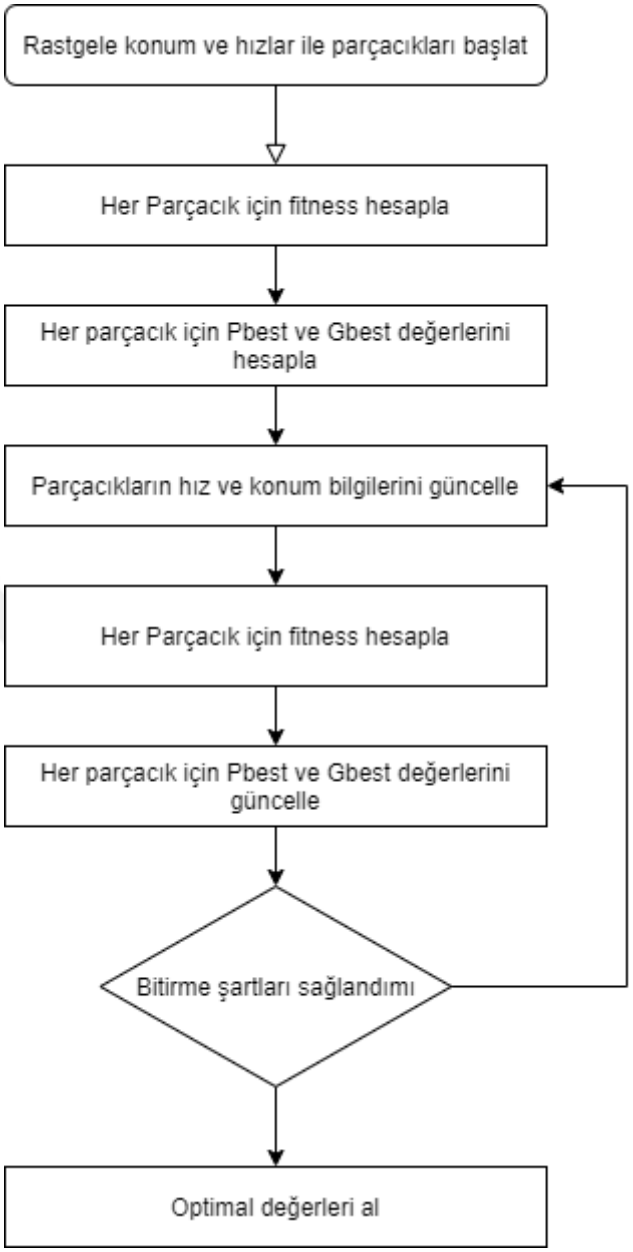
Adım 4. Her bir parçacık için (i=1, … n_s)

Denklem 2'yi kullanarak hızı güncelle

Denklem 1'i kullanarak pozisyonu güncelle

UNTIL (durdurma kriterleri sağlanıncaya kadar)

$\hat{y}(t) = \min\{f(x_0(t)), \dots, f(y_{ns}(t))\}$PSO algoritmasına ait sözde kod aşağıda verilmiştir.



Şekil 2.1 PSO akış diyagramı

2.2 Genetik Algoritma

Genetik algoritma(GA) J. Holland tarafından geliştirilmiş sezgisel bir algoritmadır. Evrim teorisinden esinlenen evrimsel algoritmalar sınıfına girmektedir. Evrimsel algoritmalar, evrim teorisinin temel dinamikleri olan en iyinin hayatta kalmasını ve doğal seçilimi modelleyen sezgisel arama algoritmalardır. GA, kalıtım, seleksiyon, çaprazlama veya rekombinasyon, mutasyon ve üreme gibi çeşitli biyolojik teknikleri kullanır.

Genetik algoritmaların kolay uygulanabilir olması ve uygulanmasının büyük miktarlarda depolama gerektirmemesi, optimizasyon problemlerinde çözüm uzayının genişliği ve kabul edilebilir bir hesaplama süresinde tarayarak optimale yakın çözümler bulması bu algoritmayı sıklıkla tercih edilir hale getirmiştir.

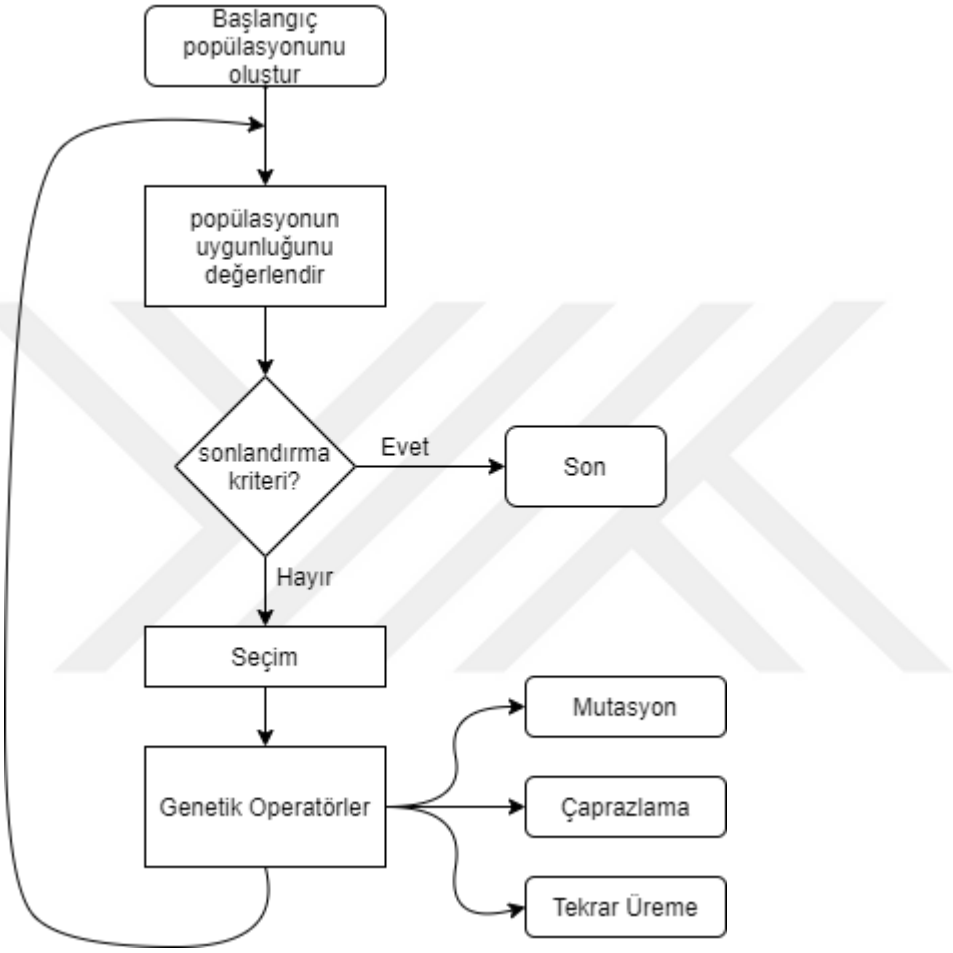
Genetik algoritmalar değerleri rastgele oluşturulmuş başlangıç popülasyonu ile başlarlar. Algoritmaya ait uygunluk fonksiyonu belirlenir ve popülasyondaki bütün bireyler bu uygunluk fonksiyonundan geçirilir. Yetenekleri tespit edilen bireylere üreme, çaprazlama ve mutasyon gibi genetik operatörleri uygulanarak yeni bir popülasyon oluşturulur ve önceki nesil değiştirilir. Zaman geçtikçe, en yeterli bireyler hayatta kalırken, soruna en kötü çözümleri sunanlar ortadan kalkar. Bu döngü, kullanıcı tarafından belirlenen döngü sayısınca devam eder.

GA, çözümün hem yapısının hem de parametrelerinin aynı anda keşfedildiği bir model olan tümevarım yöntemidir. GA'nın yaygın kullanılmasının sebebi bir problemin çözümü için operatör tabanlı davranışları üretebilmesidir. Ek olarak, GA, bu çalışmada önerilen, tankın çevre hakkında çok az bilgiye sahip olduğu gibi, çok az bilginin mevcut olduğu karmaşık alanlara iyi uyum sağlar. Bir GA sistemi, yazılımcıların programlama yeteneklerinin ötesinde davranışlara bile yol açabilir ve yaratıcı sonuçlar üretebilir.

Genetik algoritmanın yapısı şu aşamalardan oluşmaktadır.

İlk olarak başlangıç popülasyonu oluşturulmaktadır. Popülasyondaki bireyler uygunluk fonksiyonundan geçirilerek en yüksek puanlı bireyler tespit edilirler. Sonra bireylere

mutasyon, çaprazlama ve doğal seleksiyon gibi genetik operatörler uygulanır. Bu işlemler sonrasında zayıf bireyler elenir, güçlü bireyler ve onlardan türetilen yeni bireyler ile yeni bir nesil oluşturulur. Bu işlem optimal bir çözüm bulununcaya kadar devam eder. Genetik algoritmaya ait akış diyagramı şekil 2.2'de görülmektedir.



Şekil 2.2 Genetik algoritmaya ait akış şeması

Basit bir genetik algoritma her biri algoritmanın başarısını önemli derecede etkileyen beş bölümden meydana gelmektedir. Bunlar; çözümlerin temsil şekli, başlangıç popülasyonun oluşturulması, uygunluk ve kalitenin değerlendirilmesi fonksiyonu, genetik operatörler ve kontrol parametreleridir. Bu temel adımlar genetik algoritmanın sözde kodunda gösterilmiştir.

Adım 1: Rastgele veya sezgisel olarak bir başlangıç popülasyonu tanımla.

REPEAT

Adım 2: Popülasyondaki her birey için uygunluk değerini hesapla

Adım 3: Eğer durdurma kriterleri sağlanıyor ise araştırmayı durdur. Yoksa aşağıdakileri yap

Adım 4: Doğal seçilim işlemini uygula.

Adım 5: Çaprazlama işlemi uygula

Adım 6: Mutasyon işlemi uygula

UNTIL (durdurma kriterleri sağlanıncaya kadar)

Genetik algoritma ile bazı tanımlar aşağıda verilmiştir.

Seçilim: Bu işlem uygunluk kriterine göre seçilmiş yeniden üretilecek kromozomların seçimini temsil eder.

Yeniden Üretim: Bu işlem mevcut bir nesilden yeni bir neslin oluşturulmasıdır.

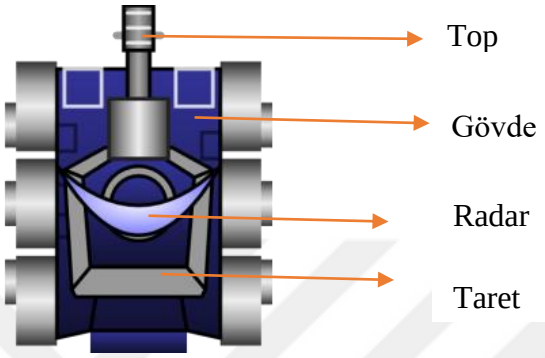
Çaprazlama: Bu işlem ile kromozomlar arasında genetik materyaller değiştirilir. Bu değişim tek bir noktadan olabildiği gibi birden fazla noktadan da olabilir.

Mutasyon: Bu işlem, bir bireyin kromozomunun değiştirilmesidir. Böylelikle algoritma makul bir sonuca ulaşsa da evrimine devam eder.

Durdurma Kriteri: Hedeflenen sonuç elde edilince veyahut belirlenen döngü sayısına ulaşılmışsa algoritma durdurulur.

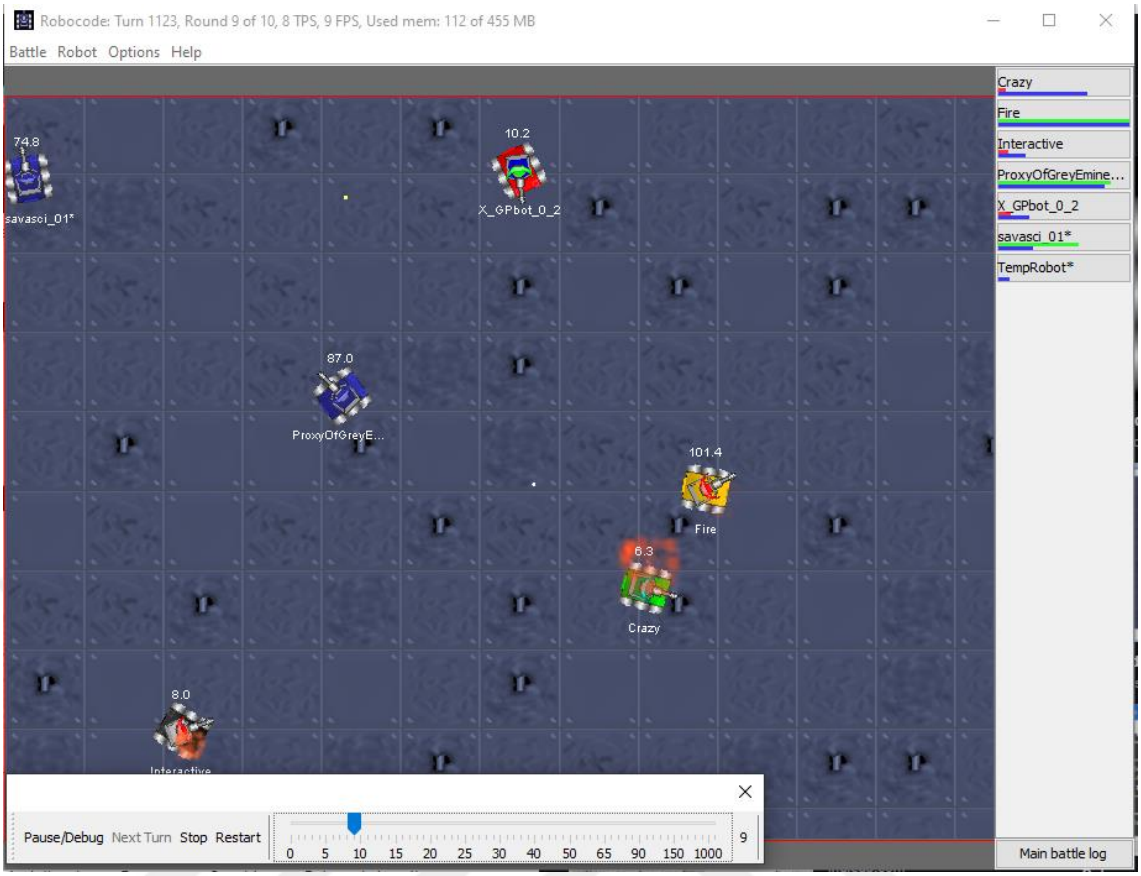
3. ROBOCODE

RoboCode Robotları ana program altında iş parçacıkları olarak çalışan, Java veya .NET uygulaması olarak yazılmış olay güdümlü sınıflardır. Ana döngünün görevi robot etkinliklerini kontrol etmektir.



Şekil 3.1 RoboCode tankın yapısı

Birkaç araştırmacı, RoboCode için simüle edilmiş dövüş robotlarının oluşturulmasını otomatikleştirmeye çalıştı. Örneğin, Eisenstein (Eisenstein,2003) Geliştirilen her robotun uygunluğunu belirlemek için, rakibine ne kadar hasar verdiğine bağlı olarak bir puan ve ayrıca ayakta kalan son robot olmak için önemli bir bonus verildi. Eisenstein, otomatikleştirmeye çalıştığı robotları insanlar tarafından kodlanmış robotlarla eğitmeye çalışmıştır. Harici testler için RoboCode Ligi'nde ilk 4'te olduğu için uzman olarak etiketlenen "SquigBot" adlı bir robot kullanıldı. 13 nesil sonra eğitim robotlarına karşı %100, 60 nesil sonra birden fazla düşmanla mücadelede %50 başarı elde edildi. Eisenstein'ın robotları, bire bir savaşta harici test robotunu zamanın %50'sini yenebildi.



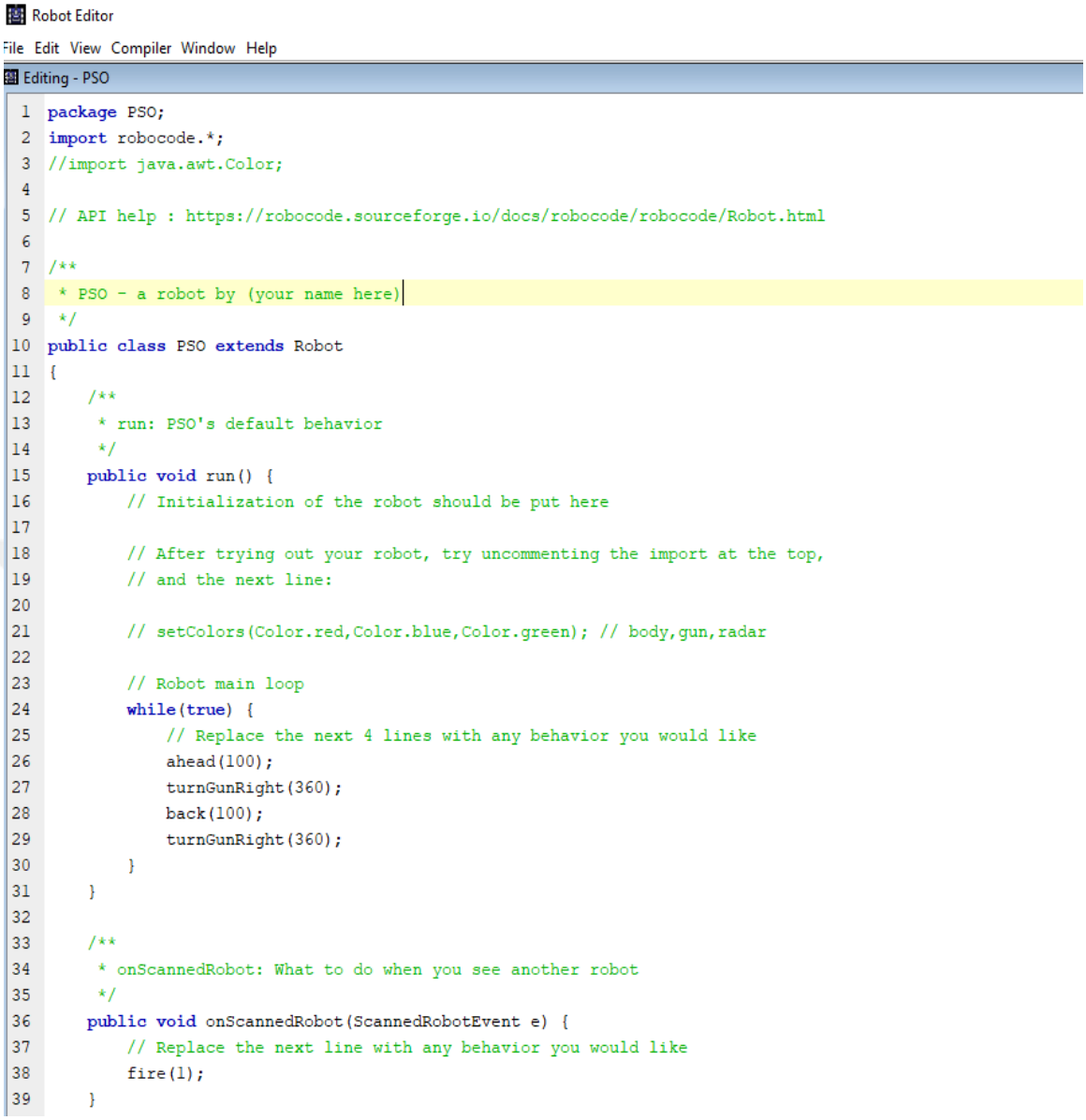
Şekil 3.2 RoboCode simülatörünün görünümü

3.1 Robocode Temelleri

Robocode, IBM AlphaWorks tarafından Java'da programlanabilir, rakip robotlar oluşturmak ve çalıştırmak için bir frameworktür. İlke defa 2001 yılında dağıtılan sürümünden bu yana RoboCode platformunda çok köklü değişiklikler meydana gelmiştir. Günümüzde RoboCode, turnuvaların ve liglerin olduğu, robot sıralamalarının yapıldığı, aktif web sitesine sahip büyük bir topluluk haline gelmiştir.

3.2 Oynanış

Robocode'da bir dizi robot bir arenada savaşır. Savaşın kaç tur süreceği kullanıcı tarafından belirlenir. Savaşın kazananı bütün turlarda toplam en fazla puanı alan robottur. Biri hariç tüm robotlar öldüğünde bir tur sona erer.



Şekil 3.3 RoboCode kod geliştirme ortamı

Her robot savaşa 100 enerji puanı ile başlar, enerji puanı tükenince ölür. Robotlar tank üzerindeki topları kullanarak rakip tankı vurmak suretiyle hasar vermeye çalışır. Her tank 1200 piksel çapındaki bir daireyi tarayan radara sahiptir. Radar, taranan robot ile ilgili oyun mesafesi, yön, hız, isim ve enerji bilgilerini verir.

Robotun üç esas parçası olan top, radar ve gövde bölümleri bağımsız olarak 360 derece dönme yeteneğine sahiptir. Bunun yanında eğer istenirse top tankın hareket yönüne kilitlenebilir. Top değişik güçlerde ateş etme yeteneğindedir. Ancak top ateş ettiğinde

ısınır ve tekrar kullanılabilmesi soğuması gerekir. Topun ateş gücü ile mermi hız ters orantılıdır.

Robotların çarpışması her iki robotta da hasara neden olur ve çarpma sonucu her robot 0.6 enerji puanı kaybeder. Ancak bunun yanında çarpışma anında robotların yönlerine bakılır. Eğer robot rakibinin üzerine doğru gidiyorken çarpışma gerçekleşmişse 1.2 puanla ödüllendirilir.

Savaş arenası 800x600 varsayılan boyutlarındadır. Ancak bu değerler değiştirilebilir. Arenanın sınırlarında duvar vardır. Robotların boyutu 36 × 45 pikseldir. Robot sabit veya ivmeli hızla ileri, geri hareket edebilir. Hız değerinin pozitif olması ileri yönü, negatif olması geri yönü temsil eder. Robotun ulaşabileceği azami sürat 8 px/pik'tir

Robocode, robotların olası eylemlerini sınırlayan bir fizik modeline sahiptir; robot dönerken tam hızda hareket edemez. Ateş etmek robota enerji kaybettirir. Mermi bir hedefe çarptığında kaybedilen enerji 3 faktörle geri kazanılır. Bazı örnekler için Çizge 1'de gösterilmiştir.

RoboCode robotları sıra ile hareketlerini belirlerler ve her robotun hareketini seçmek için belirli bir süresi vardır. Robotların hareket seçmeleri tamamlandığında sırası ile seçtikleri hareketler icra edilir. Bir robot kendine tanınan sürede hareketini seçmezse o tur boş geçer, toplam otuz turda boş geçerse robot elenir. Bundan dolayı hareket seçiminde hareketin karmaşıklığı ile gerekli süre arasında bir denge sağlanmalıdır. Karmaşık hesaplamalar nedeni ile tur kaçırmaktansa basit hareketler daha iyi bir stratejidir.

Çizelge 3.1 Bazı ateş gücü değerlerinde verilen hasar, geri alınan enerji ve mermi hızını gösteren tablo

Ateş Gücü	**Verilen Hasar**	**Geri alınan enerji**	**Mermi hızı(piksel/an)**
0.1	*0.4*	*0.3*	*19.7*
1.0	*4.0*	*3.0*	*17*
2.0	*10.0*	*6.0*	*14*
3.0	*16.0*	*9.0*	*11*

RoboCode, bir maçın galibini belirlemek için bir puanlama sistemi kullanır. Kazanan robot en fazla rauntta hayatta kalan robot olmayabilir. Galip, mücadele boyunca en fazla puanı toplayan robottur. Bir robot aşağıda belirtilmiş eylem veya durumlarda puan alır.

- Her robot öldüğünde, her canlı robota 50 puan verilir.
- Hayatta kalan son robota her ölü robot için 10 puan ek ödül verilir.
- Verilen her hasar puanı 1 puan ile ödüllendirilir.
- Bir düşmanı öldürmek, o robota verilen toplam hasarın% 20'sini kazanır.
- Rakibin çarpışması, çarpışma sonucunda verilen hasar başına 2 puan ile ödüllendirilir.
- Rakibini çarparak öldürmek, o robota verilen toplam hasarın% 30'unu kazanır.

RoboCode motoru tarafından her tiki yürütmek için kullanılan işlem döngüsü aşağıda verilmiştir:

- Tüm robotlar harekete geçene kadar kodlarını yürütür
- Zaman günceller
- Tüm mermiler hareket eder ve çarpışma olup olmadığını kontrol eder
- Tüm robotlar hareket eder (bu sırayla rota, hızlanma, hız, mesafe)
- Tüm robotlar tarama yapar (ve ekip mesajlarını toplar)
- Savaş alanı çizer

RoboCode, her robotun kendine ait bir olay kuyruğuna sahip olduğu, olay odaklı bir sisteme dayanmaktadır. Örnek olaylar "robot bir kurşunla vuruldu", "robot tarama yapıyor". Temelde iki robot sınıfı vardır bunlar, *Robot* veya *AdvancedRobo*t sınıflarıdır. AdvancedRobot Java sınıfını temel alan robotlar zaman uyumsuzdur, yani engelleme yapmayan yöntemleri kullanarak eylemleri aynı anda gerçekleştirmek mümkündür. AdvancedRobot aşağıdaki olayları alır:

- **BulletHitEvent:** Robotun kurşunu rakip robota çarptığında alınır. Merminin başlığını, sahibini, gücünü, hızını, kurbanını ve merminin hala aktif olup olmadığını söyleyen bir durum işaretini tutan bir Bullet nesnesi ile birlikte rakibin adı ve kalan enerji hakkında bilgi verir.
- **BulletHitBulletEvent:** Robotun mermilerinden biri başka bir mermiye çarptığında ve imha edildiğinde alınır. İki mermi nesnesini döndürebilir.
- **BulletMissedEvent:** Robotun mermilerinden biri duvara çarptığında ve mermi nesnesini verdiğinde alınır.
- **HitByBulletEvent:** Bir mermi tarafından vurulduktan sonra alınır ve merminin başlığı, gücü ve hızı ile birlikte atıcı adını verir.
- **HitRobotEvent:** Başka bir robotla çarpıştıktan sonra alınır ve diğer robotun adını, enerjisini, yönünü ve robotun diğer robota doğru ilerleyip ilerlemediğini verir.
- **HitWallEvent:** Robot bir duvarla çarpıştığında alınır. Robotun duvara açısını verir.
- **ScannedRobotEvent:** Radar bir robotu taradığında alınır. Diğer robotun adını, enerjisini, yönünü, mesafesini, istikametini ve hızını verir.
- **WinEvent**: Robot bir tur kazandığında ve hiçbir bilgi içermediğinde alınır.
- **DeathEvent:** Robot öldüğünde ve hiçbir bilgi içermediğinde alınır.
- **SkippedTurnEvent** Robot, dönüşün belirli bir zaman dilimi içinde bir eyleme karar vermediğinde alınır. Hiçbir bilgi içermiyor.

- **RobotDeathEvent**: Başka bir robot öldüğünde alınır. Öldürülen robotun adını tutar.

3.3 ROBOCODE Stratejileri

Savaşlarda stratejiler önemlidir. RoboCode'da da çeşitli stratejiler uygulanır. Bu stratejiler dört sınıfta değerlendirilir.

• **Algılama:** Robotun rakiplerini tespit etmesi amacıyla radarı ne şekilde kullanacağına karar vermek.

• **Planlama:** Hedeflere ve kısa vadede yapılacak eylemlere karar vermek.

• **Hareket:** Robotun hareketlerine karar vermek.

• **Hedefleme ve atış:** Ne zaman, hangi güçte ve ne yöne ateş edileceğine karar vermek.

Bir robot bire bir savaşlarda başarılı olabilir, ancak yakın dövüş savaşlarında başka stratejilere ihtiyaç vardır. Yakın dövüşte iyi bir stratejinin ilk etapta çatışmalardan uzak durmak olduğunu düşünebiliriz. Daha sonra, diğer robotlar birbirlerini öldürdükten sonra, son robotu daha kolay öldürebilir, çünkü muhtemelen çok fazla hasar görmüş olacaktır. Ancak bu strateji tekli savaşlar için uygun değildir.

Aşağıda, bir robotu kontrol etmek için çeşitli stratejiler ve herhangi bir belirgin artı ve eksileri sunulmaktadır.

3.3.1 Algılama

Rakiplerin konumlarının tespit etmek için en etkili yöntem radarın sürekli bütün arenayı taramasıdır. Bunun yanında tespit edilmiş rakibin yerinin hatırlanması ayrıca önemlidir. Ancak, robotlar dövüş nedeniyle nadiren arenaya eşit olarak yayıldığından, bu optimal bir strateji değildir. Alanın boş bölümlerini tarayarak zaman harcamaktır.

Diğer bir strateji radarın bir yöne tararken rakiplerin tamamını tespit ettiği anda radara tersi yönde hareket ettirmektir. Bu yaklaşım rakip robotun konumunun ardışık olarak değişeceğidir. Yani t anında x noktasında olan bir robot t+1 anında x-1 ile x+1 noktaları arasında olacaktır. Bu yöntem tekli mücadelelerde çok etkilidir. Bu, hedefin izini kaybetmemek için radarı her turda biraz ileri geri hareket ettirerek yapılabilir.

RoboCode'da rakibin ateş ettiğini döndüren bir metot yoktur. Radar tespit ettiği rakibin enerji bilgisini alabildiğine göre, eğer bir rakip radarın üst üste yaptığı iki taramada da tespit edilmiş ve enerjisi düşmüşse buradan muhtemelen rakibin ateş ettiğini gösteriyordur sonucuna varılabilir.

3.3.2 Planlama

RoboCode'daki asıl hedef, rakiplere saldırarak ve öldürerek kazanmak olduğundan, bir robot saldırmak istiyorsa hangi rakibe saldıracağına karar vermelidir. Bu, yakın dövüş savaşlarında en alakalı olanıdır, çünkü bire bir savaşlarda bu sorun, rakibe saldıracağınızı veya kaçınacağınızı belirlemeye indirgenir.

Bu noktada şu stratejiler sergilenebilir. En düşük enerjideki robota saldır. Bütün rakiplerden daha zayıf olunduğu durumlarda saklan. En yakın hedefe saldır. Rakibin mesafesi ile enerjisini oranlayarak saldır.

RoboCode, taramanın aksine, robot sınıflarının hiçbirinde planlamaya yardımcı olacak hiçbir yöntem sunmaz.

3.3.3 Hareket

Belirli bir noktada sabit kalmak kötü bir stratejidir. Düz çizgilerde arenada hareket etmek de kötü bir stratejidir, çünkü bir robot bir rakibin rotasını tahmin etmek için çok az sorun yaşayacaktır, bu nedenle gelecekteki pozisyonları hedeflemek kolay olacaktır. Bu şekilde hareket eden robotlar muhtemelen rakiplerinden çok sayıda isabet alacaktır.

Dairesel hareket: Yönün sabit olmadığı bir hareket tercih edilebilir ancak çapı bilinen bir çemberdeki hedefin pozisyonunu hesaplamak kolay bir iştir.

Rastgele hareket: rakip robotların bir sonraki konumunuzu tahmin edememesi iyi bir yöntemdir. Ancak sürekli yön değiştirince rakibi tespit etmek de zorlaşacaktır. Ayrıca duvarlara çarparak enerji kaybetme olasılığı artar.

Anti-Yerçekimi: Robotu tehlikeli bölgelerden kaçırarak güvenli bölgeye hareketini sağlayacak algoritmalara göre hareket etmek akılcı bir yöntemdir.

Mermiden kaçınmak: Radar, mermilerin ateşlendiğini doğrudan tespit edememesinin yanı sıra, rakibin topunun hangi yöne dönük olduğunu belirleyemez. Dolayısı ile merminin hangi doğrultuda gelmekte olduğu bulunamaz. Buna karşın rakipteki enerji kaybından yola çıkılarak merminin hızı bulunabilir. Fikir ateş sırasında rakibin pozisyonunda merkeze sahip bir dalga cephesini simgeleyen bir daire çizmek ve daha sonra merminin hızı ile yarıçapı arttırmaktır. Bu şekilde, robot, mermiyi vurmaya yakın olana kadar tahmin edilen mermiyi görmezden gelebilir ve bu noktada ani, rastgele bir

Robocode AdvancedRobot, hareket aktüatörlerini kontrol etmek için çizge-2’ belirtilen metotlarına sahiptir.

Çizelge 3.2 AdvancedRobot metodları

setAhead	and ahead. Belirli bir mesafeyi ileriye taşı
setBack and back	Belirli bir mesafeyi geri taşıma.
setMaxTurnRate	Robotun dönüş hızını sınırlayın
setMaxVelocity	Robotun hareket hızını sınırlayın
setStop and stop	Hareketleri durdurun ve robotun ne yaptığını hatırlayın
set Resume and resume	SetStop ile durdurulan hareketi sürdürme veya dur
setTurnLeft ve turnLeft	Robotu sola çevirin
setTurnRight ve turnRight'ı	Robotu sağa çevirin

3.3.4 Hedefleme ve Ateş Etme

Hedefleme görevi, silah taretini döndürme açısını hesaplamaktır, böylece bir düşman robotuna vurmak için doğru yönü gösterir. Ancak bu hedefi vurmayı garantilemez. Çünkü merminin rakibe ulaşması için gerekli sürede rakip yer değiştirebilir. Merminin farklı hızlara sahip olması da hedefi vurmada diğer bir faktördür.

Basit bir strateji, rakipleri taramak ve bir tane bulunduğunda anında tam güçle rakibin yönüne ateş etmektir. Bu yöntem iyi sonuç vermesine rağmen hareketli robotlarda enerjinin azalasına neden olur.

Fire metodu, bir robotun her yöne ancak düşük bir güçle ateş ettiği basit bir stratejidir. Rastgele yapılan ateşlerle rakibin vurulacağı öngörüsüne dayanır.

Move-in Move-out metodu, Vur kaç taktiği olarak ta düşünülebilir. Rakip tespit edilince o yöne karşı hızlı bir saldırı ve ardından o bölgeden uzaklaşma taktiğidir. Bu, büyük olasılıkla bire bir savaşlarda oldukça işe yarayacaktır, çünkü arenanın büyük bölümleri boş olacaktır. Ancak rakipten kaçmak önemsiz olmayabilir.

Daha gelişmiş bir strateji, hedefin gelecekteki konumlarını radar taramasından döndürülen başlık ve hızdan tahmin etmeye çalışmak olabilir.

4. METODOLOJİ

Bu tez çalışmasında Sezgisel optimizasyon algoritmalarından en sık kullanılan genetik algoritma optimizasyonu ve parçacık sürü optimizasyon algoritmaları kullanılmıştır. Sürü robotların hareketinde sezgisel yaklaşımlar bu iki optimizasyon algoritması ile elde edilen robotlar özelinde incelenmiştir. Sürü robotların temsili için RoboCode simülatör platformu kullanılmıştır. Bu platformun kullanılma sebebi, algoritmaların robotlara uygulanışının kolaylığı ve uygulama sonuçlarının hızlı şekilde tespit edilebilmesine olanak sağlayan Robocode API'sinin olmasıdır.

Robotlar ve algoritmalar Java dilinde yazılmıştır. Bunun temel sebebi RoboCode programının Java dilinde geliştirilmiş olmasıdır.

Bu çalışma GA ve PSO algoritmaları kullanılarak beş ayrı robot geliştirme senaryosu önermektedir. Her senaryo farklı özelliklere sahip RoboCode platformundaki hali hazırdaki robotlarla savaştırılarak, GA ve PSO ile robot eğitmeye, bu eğitime ile sürüdeki ortalama öğrenme başarısının incelenmesine dayanmaktadır. Bu kapsamda her senaryoda GA ve PSO ile birer robot eğitilecek, eğitilen robotun ve sürünün ortalama başarı değerleri karşılaştırılacak, GA ve PSO ile eğitilen robotlar birbiri ile çarpıştırılarak eğitilen robot dışındaki robotlara karşı olan başarı puanlarının karşılaştırılması suretiyle uygulanan optimizasyon algoritmalarının başarısı tespit edilmiş olacaktır.

Robotların eğitilmesi için her bir senaryoda savaştırılacak robotlar şu şekildedir.

1. Senaryo: Crazy robot kullanılarak GA ve PSO ile eğitim.

2. Senaryo: Fire robot kullanılarak GA ve PSO ile eğitim.

3. Senaryo: Corners robot kullanılarak GA ve PSO ile eğitim.

4. Senaryo: VelociRobot robot kullanılarak GA ve PSO ile eğitim.

5. Senaryo: PaintingRobot robot kullanılarak GA ve PSO ile eğitim.

4.1 Genetik Algoritma Optimizasyon Çalışmaları

4.1.1 Genetik algoritma temel yöntem ve yaklaşımlar

Genetik programlama ve evrimsel programlama, tüm programları oluşturan özel evrimsel hesaplama biçimleridir. Evrimsel programlama makine talimatları düzeyinde gerçekleşirken, genetik programlama genellikle daha yüksek seviyeli programlama dilleriyle (genellikle LISP) sınırlıdır. Genetik programlamanın benzersiz bir özelliği, bit dizisi gibi doğrusal bir yapının aksine bir ağaç yapısını almasıdır. Evrimsel hesaplamanın temel kavramları doğru kalır (Brownlee, 2011).

Bu tezde GA ağacının yapısında iki tip düğüm kullanılmıştır. Bunlar fonksiyon düğümleri ve uç düğümleridir. Fonksiyon düğümleri, bir veya daha fazla ariteye sahiptir. Terminal düğümlerinin ariteleri sıfırdır. Bir düğümün niteliği, sahip olması gereken alt düğümlerin sayısını belirler. Bu ayrım aynı zamanda GP ağacının boyutunun üst ve alt sınırlarının belirlenmesine de hizmet eder. Alt sınır, hiçbir düğümün terminal olamayacağı minimum derinliği belirler. Üst sınır, tüm düğümlerin terminal olması gereken maksimum derinliği belirler.

Bir GP'deki genetik operatörler, bir ağaç yapısına uyacak şekilde geleneksel genetik algoritmalardan değiştirilir. Çaprazlama, bir ebeveynden bir veya daha fazla alt ağaç seçilerek ve bunları başka bir ebeveynin rastgele seçilmiş alt ağaçlarına eklenerek yapılır. Mutasyon benzer şekilde yapılır, sadece ikinci bir ebeveynden birini seçmek yerine eklenecek rastgele bir alt ağaç oluşturur. Diğer çoğaltma operatörleri, çoğaltma ve mimari değişiklikleri içerir. Çoğaltma, bir robotun bir nesilden diğerine geçmesi anlamına gelir. Mimari değişiklikler, GP'nin uyması gereken ağacın minimum/maksimum derinliklerini değiştirmek gibi kuralları değiştirir.

Robocode Robotlar yapı olarak oldukça sınırlıdır. Başka hiçbir olay meydana gelmezken Robotun davranışını tanımlayan bir run() yöntemi olmalıdır. Robot sınıfı, Robot soyut yöntemlerinin herhangi bir alt kümesini de içerebilir. Bu yöntemler, savaşla ilgili olaylar meydana geldiğinde Robocode'un olay işleyicisi tarafından çağrılır. Bu tezde yazılan genetik programın odaklandığı ana olaylar şunlardı:

onScannedRobot(ScannedRobotEvent e)
onHitByBullet(HitByBulletEvent e)
onHitWall(HitWallEvent e)

Bu olaylar beklendiği zaman çağrılır ve değişkenleri ayarlamak için kullanılabilecek 'e' olayı içinde paketlenmiş verileri içerir.

onScannedRobot( ) içinde, GP nereye hareket edeceğine, döneceğine, Robotun silahını ve radarını çevireceğine ve ateş edeceğine dair bir dizi talimat geliştirir. Ek olarak, GP bu olayı, programın diğer alanları tarafından çağrılabilecek statik değişkenlerin değerlerini ayarlamak için kullanır (özellikle run() yöntemi içinde). Her eylem yöntemi, kendi kökü olan kendi GP ağacıdır, ancak evrim süreci, genlerin farklı ağaçlar arasında geçiş yapmasına veya tüm ağaçların farklı eylem yöntemlerine taşınmasına izin verir.

Bu tezde oluşturulan genetik program dört java sınıfından oluşmaktadır. Bunlar;

- RunGP ,
- Metabot ,
- ExpressionNode,
- BattleRunner

dir. ExpressionNode sınıfı, ağaçlarda kullanılan düğümlerin özelliklerinin tanımladığı ve bunun yanı sıra mevcut ifadeleri ve düşük seviyeli yardımcı yöntemlerini (koşul ifadeleri) tanımlar. Her düğüm, ifadenin kendisini temsil eden bir string dizisi ve alt düğümlerinin her birini tutan bir ExpressionNode dizisi içerir. Diğer sınıf alanları, düğümün niteliğini, derinliğini ve bir uçbirim olup olmadığını içerir.

Bu genetik programda kullanılan yöntemlerin çoğu özyinelemelidir. ExpressionNode sınıfında düğümler özyinelemeli olarak türetilmiştir. Sınıfın en önemli yöntemi, her düğümün alt düğüm argümanlarını bir diziye derlediği ve cevap döndürmeden önce kendi ifadesini eklediği, derinlik öncelikli bir ağaç geçişi kullanan compose()

yöntemidir. Diğer yöntemler, derinliğe dayalı olarak uygun ifadelerin atanmasını ve üreme yöntemleri için belirli bir düğümün bulunmasını ve döndürülmesini içerir.

MetaBot sınıfı, derlenmiş Java sınıfı dosyaları olan programın asıl çıktısını temsil eder. Her MetaBot nesnesi, ifade ağaçlarının kökleri olarak hareket eden bir ExrpessionNodes dizisine sahiptir. Robot kodunun ana gövdesi bir dize olarak temsil edilir. Robotları derlerken, ifade ağaçlarının dize temsillerini Robot'un koduna eklemek için MetaBot yöntemi setCode() kullanılır. Bu sınıf aynı zamanda dosya yönetiminin çoğunu ve tüm derlemeleri de yönetir.

BattleRunner sınıfı, Robocode programının kendisi için arayüz görevi görür. Bir RoboCode motorunu başlatır. Motor, savaşları yürütmek için kullanılır ve puanlar ve diğer ilgili bilgiler şeklinde geri bildirim sağlar. BattleRunner sınıfı daha sonra bu puanları alır ve her robotun uygunluğunu hesaplar.

Son olarak, RunGP sınıfı gerçek genetik algoritmayı içerir. Popülasyonu temsil etmek için bir dizi MetaBot kullanır ve hangi genetik operatörün kullanılacağının seçimini ve belirlenmesini yönetir. Genel program mimarisi, ExpressionNode üzerinde MetaBot oluşturmayı ve MetaBot ve BattleRunner üzerinde RunGP oluşturmayı içerir.

Programın en zorlu tasarım yönü, ifade ağaçlarının büyümesini/büzülmesini nasıl düzenleyeceğinizi bulmaktı. Bu sonuçta, çaprazlamalarda hangi boyutlardaki alt ağaçların kullanılabileceğini belirleyen bir algoritma geliştirerek, ikinci ebeveynde böyle bir alt ağaç bularak ve birinci ebeveynde bunun için geçerli bir ekleme noktası bularak çözüldü. Bu, rastgele seçim derinliklerini koruma ihtiyacı nedeniyle zorlaştı. Ortaya çıkan algoritma, insertAt() ve getSubTree() yöntemlerine taşınsa da, ExpressionNode sınıfındaki insert() yönteminde görülebilir.

Toplam Skor=Mermi Hasarı + Ramming Hasarı+ Hayatta Kalma Skoru

Mermi Hasarı : Düşman mermisinin her çarpmasından alınan hasar için bir puan
Çarpma Hasarı : Düşman çarpmasının her çarpmasından alınan hasar için bir puan
Hayatta Kalma Skoru : zafer için 60 puan + mermi hasarının % 20'si + Çarpma hasarının % 30'u

Şekil 4.1 Skor Hesaplama

4.1.2 Uygunluk Değerinin Hesaplanması

Popülasyon içindeki her program için, GP gerçek uygulama ile uygunluklarını değerlendirilmiştir. Bu değerlendirme, yerleşik motor kontrol yöntemlerine sahip olan Robocode Kontrol API'si aracılığıyla sağlanmıştır. Her nesil için, popülasyonun her bir üyesi, belirli bir tur sayısı için bir rakibe karşı çalıştırılmıştır. Her turdan sonra, hem GP robotu hem de rakip için varsayılan puan toplanır.

Göreceli bir puan, GP robotunun puanının, puanının toplamına ve rakibin puanına bölünmesiyle hesaplanır. Bazı erken Robotlar ateş etmediği ve sıfır almadığı için, zindeliğin savunma yeteneklerini de yansıtmasını sağlamak için puanlarına önemsiz bir sabit eklenir. Robotun genel uygunluğunu üretmek için her turun elde edilen göreceli puanının ortalaması alınır. 0,50 veya daha yüksek bir uygunluk, Robotun rakibiyle eşit veya ondan daha iyi eşleştiğini gösterir.

Robocode deterministik olmayan bir oyun olduğundan, uygunluğu belirlemeye yönelik herhangi bir yöntemin şansı hesaba katması gerekir. Şansı en aza indirmenin tek yolu, her test için çok sayıda tur(nesil) kullanmaktır. Bunun için her GP robotu 25-30 nesilde başarı grafiği durağana yaklaşsa da 100 nesilde eğitilmiştir. Bu nesil sayısının seçilmesinin bir sebebi de PSO ile eğitilecek robotların çok daha fazla tura ihtiyaç duymaları ve adil bir karşılaştırma yapabilmek için her iki algoritmada eşit tur sayısı kullanma gereğidir.

4.1.3 Üreme

Popülasyon içindeki her bir üyenin uygunluğu belirledikten sonra, yeni neslin üretilmesi aşamasına geçilmiştir. Yapılan denemelerde rekabete dayalı turnuva seçiminin rulet seçiminden daha etkili olduğu görülmüştür. Rulet seçiminin, popülasyon içinde biraz daha yavaş bir yakınsamaya izin verdiği bununla beraber nihai sonuçta kayda değer bir gelişme meydana getirmediği görülmüştür. Turnuva seçiminde, ilk büyümenin çok daha hızlı olduğu ve erken yakınsamanın programın çok sayıda çalışması için gerekli zamana sahip olunmasına bağlı olarak telafi edildiği görüldü.

Üreme işlemini belirlemek için güvercin deliği seçimi kullanılmıştır. Bu çalışmada çaprazlama için 0.85, replikasyon için 0.05 ve mutasyon için 0.01 oranları kullanılmıştır. Geleneksel genetik algoritmalara kıyasla %1'lik bir mutasyon olasılığı Koza ve Poli'nin çalışmasında önerildiği için seçilmiştir.(Koza ve Poli, 2005).

Çaprazlama yöntemi ile iki ebeveynden tek bir birey elde edilir ancak bu birey her iki ebeveynin özelliğini gösterebilir. Çaprazlamanın bir GP ağacının kökünde meydana gelme olasılığı %30'dur, bu da çocuk bireyin tam olarak ikinci ebeveyn gibi belirli bir eylemi gerçekleştirmesine neden olur. Ek olarak, belirli bir kök geçişinin ikinci ebeveynin GP ağacını farklı bir eylem yöntemine yerleştirme olasılığı %5'tir. Çapraz geçişin bir terminal düğümünde meydana gelme olasılığı %10'dur. Program, her bir ebeveynden rastgele seçilmiş iki farklı çaprazlama noktası kullanır.

Mutasyon, vakaların %15'inde bir terminalde, vakaların %1'inde kökte bir alt ağaç içinde rastgele bir düğüm seçilmesini içeriyordu. Geri kalan %84'lük durumda düğüm, ağaçtaki minimum derinliğe ve en derin düğüme dayalı bir olasılık dağılımı oluşturularak seçilmiştir. Her derinliğin olasılığı tekdüzedir.

Replikasyon için birey seçiminde fazla çeşitliğe ihtiyaç yoktur. Hem şimdiye kadarki en iyi Robot hem de önceki nesildeki en iyiler, nadir genleri korumak amacıyla bir sonraki nesle kopyalanır.

4.2 Parçacık Sürü Algoritması Optimizasyon Çalışmaları

PSO'nun RoboCode robotlarına uygulanması Genetik algoritmaya nispeten daha kolaydır. PSO'da ifade kodlarını GA'da olduğu gibi karmaşık ağaç yapısından elde edilmemesi, Genetik operatörler ve bu operatörlerin ağaç yapısına uygulanabilirliğinin karmaşık olmasından kaynaklanan uygulama zorluğu PSO'da görülmemesi ve kısa sürede sonuçlanan programlar gibi özellikler göz önünde bulundurulduğunda PSO'yu GA'ya karşı uygulama yönünden güçlü kılmaktadır.

Çalışmanın bu bölümünde PSO'nun RoboCode'a uygulanma adımları ve işlemleri ele alınmaktadır. Bu çalışmada PSO türlerinden lBest türü PSO kullanılmıştır. lBest türü PSO algoritmaları yıldız topolojisine sahiptir. Yani bütün parçacıklar komşu olarak kabul edilir. lBest türünün seçilmesinin sebebi parçacık sayısının çok fazla olmamasından kaynaklanmaktadır.

Robocode sınıfı bir robot çok sayıda alt sınıf içermektedir. Bu çalışmada robotun hareket ve savaşma becerilerini oluşturan bu alt sınıflardan dört tanesinin eğitilmesine odaklanılmıştır. Bu alt sınıflar aşağıdaki gibidir.

Metod	Görevi
run	Bütün robotlar main metottur
scannedRobot	Bu metot robot başka bir robot gördüğünde çağrılır.
onHitByBullet	Robot bir mermi ile vurulduğunda bu metot çağrılır.
onHitwall	Robot bir duvara çarptığında bu metot çağrılır.

Çizelge 4.1 Robot sınıfının alt sınıfları

PSO uygulaması Java dilinde yazılmıştır ve beş sınıftan oluşmaktadır. Bunlar;

1. Pso_v1 sınıfı: sürü ve parçacıklara ait değerlerin yer aldığı, sürünün oluşturulduğu, sürüye ait iterasyonların ve güncellemelerin yapıldığı sınıftır.

2. Parcacik sınıfı: Parçacıkların oluşturulduğu, parçacıklara ait değerlerin belirlendiği, parçacıklara ait ifadelerin toplanarak java programlarına dönüştürüldüğü sınıftır.

3. Parcacikgenleri sınıfı: parçacıklara ait genlerin tutulduğu, gen havuzu gören sınıftır.

4. Fitness: RoboCode API'sinin çalıştırıldığı, PSO robotu (savasci_01) ile rakip robotun savaştığı, PSO robotuna ait uygunluk değerlerinin elde edildiği sınıftır.

5. Sabitler sınıfı: Programa ait sabitlerin tutulduğu sınıftır.

Sürü oluşturma işleminde, sürü 100 adet parçacıktan oluşturulmuştur. Sürüdeki parçacık sayısı benzer çalışmalar incelenerek karar verilmiştir. Sürü büyüklüğü optimum değerlere yakınsamayı çabuklaştırmaktadır. Ancak fiziksel kaynaklar göz önünde bulundurulduğunda 100 sayısı ideal görülmüştür.

Her parçacık 12 boyutlu olacak şekilde oluşturulur. Bir başka deyişle her parçacığın başarısı 12 faktöre bağlıdır. Boyutlar 0 ile 50 başlangıçta rastgele belirlenmiş değerler ile ifade edilmektedir. Boyut değerlerinin 0 ile 50 arasının sebebi ileriki aşamalarda yakından inceleyeceğimiz gen havuzunda 50 adet genin bulunmasıdır.

Her parçacığa başlangıçta 0 ile 1 arasında rastsal hız değeri atanır. İlk hız güncellemesinde bu hızlar güncellenecek olsalar bile yerel en iyi ve global en iyi parçacıkların hareketlerinin güncellenmeme ihtimaline karşı başlangıçta küçük bir hız değeri verilmiştir.

RoboCode optimizasyon işlemi elli adet gen ile belirlenmiş. Robotun manuel olarak programlanmasında sıklıkla kullanılan aksiyona yönelik çeşitli yöntemlere ardışık parametreler verilerek bir komutlar dizisi oluşturulmuştur. Bu yöntemlere ardışık değerlere sahip parametreler verilmesi PSO'nun uygulanışının kilit noktasını oluşturmaktadır. Parçacığın aksiyon karakteristiğini belirleyen genler bir dizi içinde konumu sabit olacak şekilde tutulurlar. Komşularıyla sadece yakın parametre farkı vardır. Bir parçacığın geni (ifadesi) parçacığın hız güncellemesine bağlı olarak komşu gen olacak şekilde değiştirilir.

İfadelerin oluşturulması: java dilinde yazılan robot sınıfı kodları sabit bölümler ve değişken bölümlerden oluşur. Bu çalışma yukarıda belirtilen beş adet alt sınıfa ait kodların oluşturulmasına odaklanmıştır. Bu beş alt sınıf için gen havuzundan üçer tane gen, her parçacığın sahip olduğu 12 özellikten rastgele seçilen üç değerle, belirlenir. Belirlenen bu üç gen ifadesi birleştirilerek tek bir koda dönüştürülür. Böylelikle beş alt sınıf için dört kod parçası elde edilmiş olunur. Sonrasında bütün kod parçaları birleştirilerek robot sınıfını oluşturacak tek bir kod elde edilir.

Uygunluk değerlerinin bulunması: RoboCode platformu sahip olduğu RoboCode Kontrol API ile robotların savaşlarının istatiksel bilgilerini kullanıcılara vermektedir. Bir robotun uygunluk değeri RoboCode Kontrol API sayesinde referans alınan belirli bir robot ile savaştırılması ile tespit edilir. Bu çalışmada sürüyü oluşturan bütün parçacıklar RoboCode kütüphanesindeki beş farklı robotla savaştırılmış elde edilen 0 ile 1 arasındaki uygunluk değerine göre gBest ve pBest elde edilmiştir.

Güncelleme: Her bir iterasyonun sonucunda parçacıkların sahip oldukları, o iterasyonda elde ettikleri uygunluk değeri, şimdiye kadar elde ettikleri en iyi uygunluk değeri olan pBest ve sürüde o iterasyona kadar elde edilmiş en iyi uygunluk değeri olan gBest'e göre PSO hız güncelleme formülüne göre her parçacığın yeni hızı belirlenir.

Bu formülde yer alan C_1 ve C_2 katsayıları için 0.5 değeri alınmıştır. r_1 ve r_2 değerleri 0 ile 1 arasındaki sayılardan rastgele seçilmiştir. c_1r_1 çarpımı ile her bir parçacığın uygunluk değeri ile pBest değerinin farkının çarpımı ile bilişsel bileşen bulunmuştur. Yine c_2r_2 çarpımı ile her bir parçacığın uygunluk değeri ile gBest değerinin farkının çarpımı ile sosyal bileşen bulunmuştur. Bilişsel bileşen, sosyal bileşen ve mevcut hızın toplanmasıyla yeni hız elde edilmiştir. Her parçacığın o boyuttaki konumunu temsil eden veriye yeni hız eklenerek yeni konum elde edilmiş olur. Böylelikle parçacığın temsil ettiği ifade bu yeni konuma karşılık gelen ifade ile güncellenmiş olur.

5. ARAŞTIRMA BULGULARI VE DENEYSEL SONUÇLAR

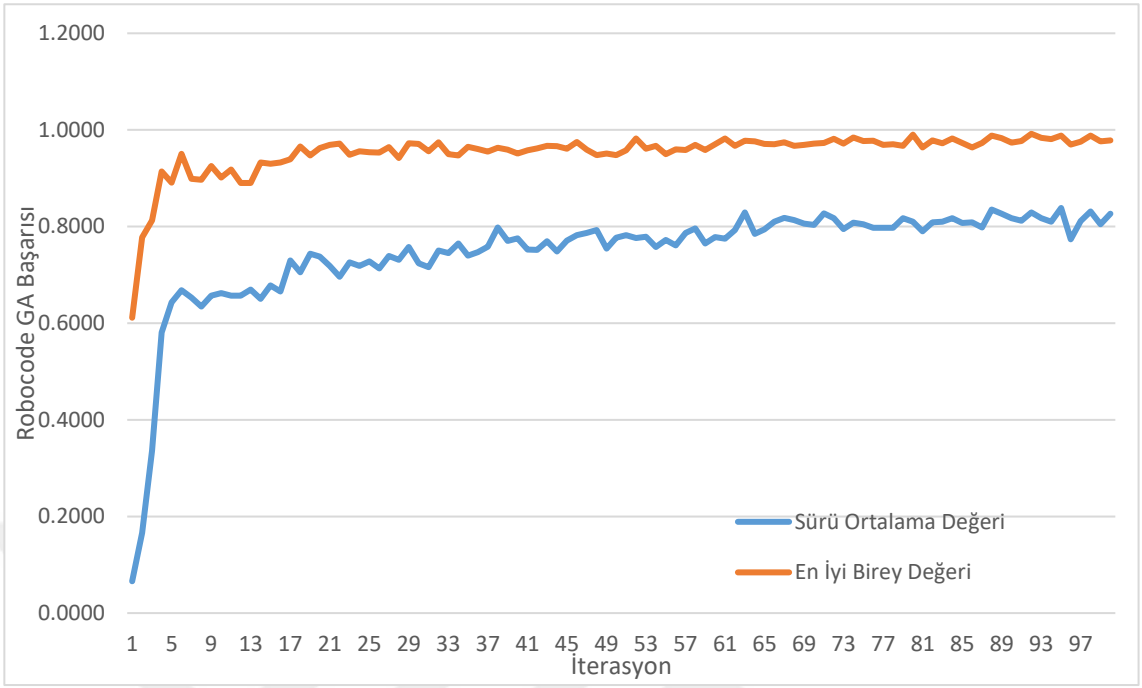
Bu çalışmada RoboCode platformunda sezgisel yöntemler ile beş farklı senaryoda robotlar eğitilmiştir. Sezgisel yöntem olarak genetik algoritma optimizasyonu ve parçacık sürü optimizasyon algoritması kullanılmıştır. Robotların hem GA hem de PSO ile eğitimlerinde en iyi başarıyı gösteren robot ve sürünün ortalama başarısı incelenmiştir. Robotların başarısı RoboCode API tarafından sağlanan karşılaşma başarı puanı ile ölçülmüştür. Karşılaşma başarı puanı 0 ile 1 arasında değer alacak şekilde aşağıdaki gibi hesaplanmıştır.

$$Robot\ başarısı = \frac{\text{robot puanı}}{\text{karşılaşmada alınan toplam puan}}$$

Çalışmanın bu aşamasında beş senaryoda gerek GA ile gerekse de PSO ile eğitilen robot başarısı ve sürü başarı ortalaması ayrı ayrı incelenmiştir.

5.1 1. Senaryo: "Crazy" İsimli Robotla Yapılan Eğitim

Bu senaryoda robotlar crazy isimli robotla mücadele ettirilerek eğitilmiştir. Eğitim GA ve PSO için eşit olacak şekilde 10000 karşılaşma ile gerçekleştirilmiştir. Şekil 5.1' de GA kullanılarak eğitilmiş robot ve sürüye ait başarı grafiği görülmektedir. Her nesilde en iyi başarıyı gösteren farklı robotların değerleri "en iyi birey" eğrisi ile görülmektedir. İlk nesilde bile yaklaşık %60 başarı gösteren bireyler gözlenmektedir. Bireysel başarı hızla yükselerek %95 başarı seviyesinin üzerine çıktığı gözlemlenmektedir. Sürü başarı ortalamasının %5 seviyesinden başlayıp sürüdeki yüksek başarılı bireylerin genlerinin yeni nesillere aktarılması sonucu hızla yükseldiği, bireysel başarının en yüksek değere yaklaşması dolayısıyla durağanlaşmasına paralel olarak sürü başarısındaki artışın da durağanlaştığı % 80 civarında yatay seyrettiği görülmektedir.

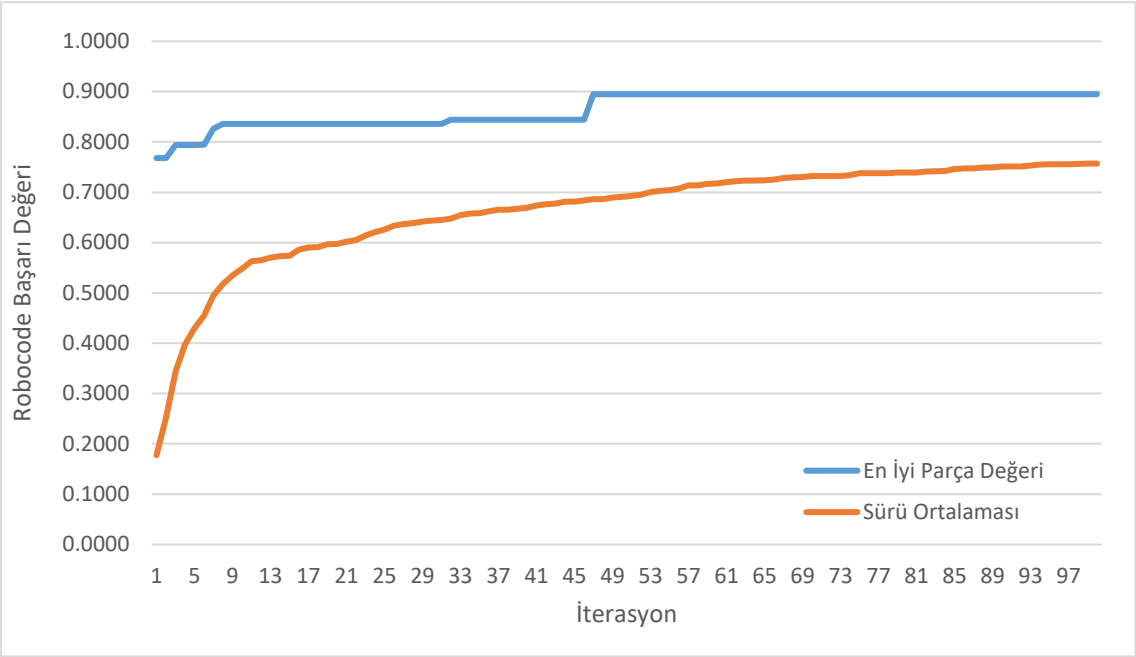


Şekil 5.1 “Crazy" robotuna karşı GA ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Şekil 5.2’de “Crazy” simli robot ile karşılaştırılarak PSO ile eğitilen robotlara ait bireysel en iyi sonuçları ve sürü başarı ortalaması sonuçları görülmektedir. İlk iterasyonda en yüksek başarı gösteren parçacıkların başarı değeri % 78 olarak görülmektedir. Bu yüksek sayılabilecek bir değerdir. Bu yüksek değer sezgisel algoritmalardaki şans faktörünün etkisini göstermek bağlamında değerlidir. Diğer bir taraftan en yüksek değere sahip parçacık altı kez değişmiştir.

100 parçacıktan oluşan sürüye ait ortalama başarı değeri ilk iterasyonlarda %20 seviyelerinde iken yüksek bireysel başarı değeri neticesinde yüksek bir ivmeyle artmıştır. %55 başarı seviyelerinden itibaren sürü ortalaması başarı değerinin daha düşük bir eğimle artamaya devam ettiği görülmektedir. 100 iterasyonda dahi eğrinin artış eğiliminde olduğu görülmektedir. PSO’nun doğası gereği bu artış, hızı azalmak kaydıyla en iyi bireysel değere yakınsamaya devam edecektir. Eğitim verilerine bakılarak aynı robotla aynı birey/parçacık sayısına sahip ve aynı iterasyon sayısınca GA

ile eğitilen robotların PSO ile eğitilen robotlara nazaran daha başarılı oldukları görülmektedir.

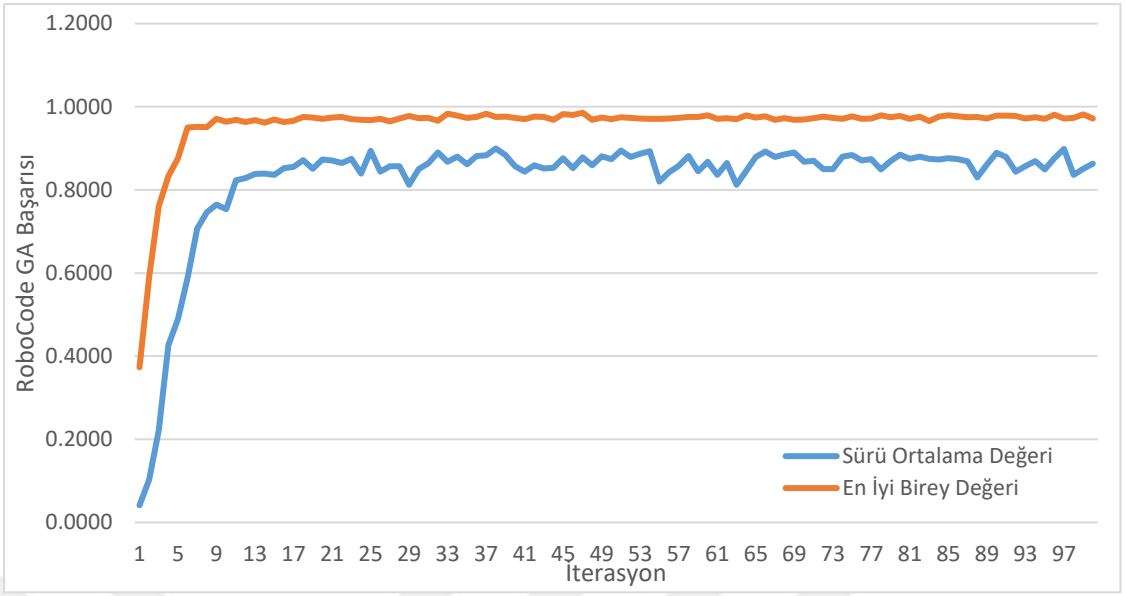


Şekil 5.2"Crazy" robotuna karşı PSO ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

5.2 2. Senaryo: "Paintingrobot" İsimli Robotla Yapılan Eğitim

Bu senaryoda robotlar "paintingRobot" ile eğitilmiştir. Senaryoda robotlar GA ve PSO ile eğitilmişlerdir. PaintingRobot'un seçilme nedeni diğer robotlara karşı başarısının nispeten daha düşük olmasıdır. Algoritmaların az hareketli bir robota karşı eğitilmeleri sonucu gözlemlenmiştir.

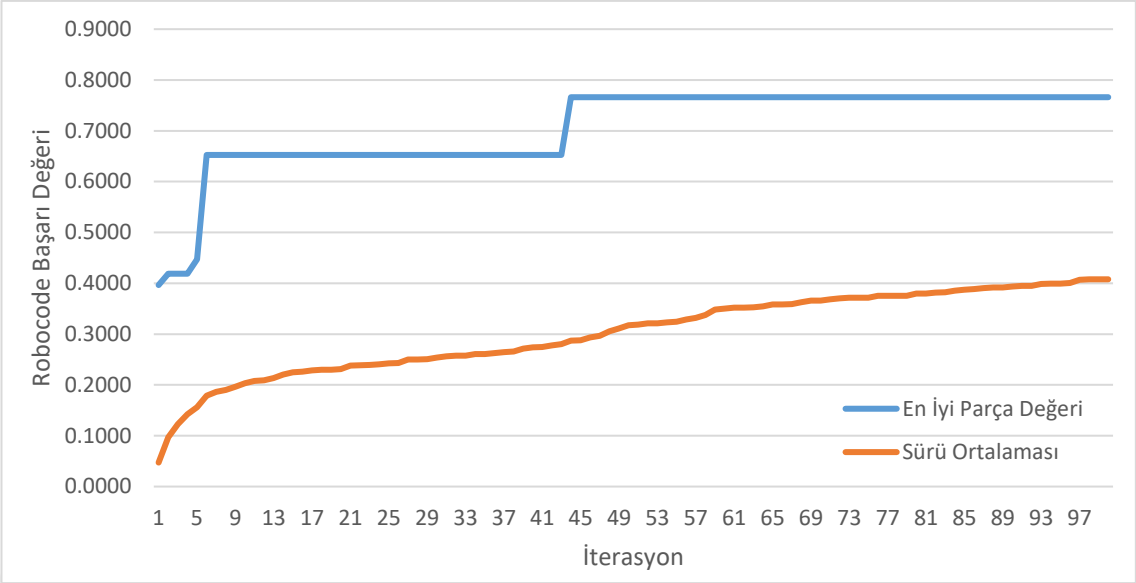
Şekil 5.3'de paintingRobota karşı GA ile eğitilmiş sürü görülmektedir. En iyi performansı gösteren bireylerin başarısının hızla %95 üzerine çıktığı bununla birlikte sürü başarı değerinin bireysel başarıya paralel olarak %85'ler seviyesinde olduğu gözlemlenmektedir.



Şekil 5.3 “PaintingRobot" robotuna karşı GA ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Şekil 4’de Painting Robot ile savaştırılarak PSO ile eğitilmiş sürüdeki en iyi skora sahip robotların oluşturduğu “En İyi Parça Değeri” grafiği ve sürü ortalaması başarı değerinin gösterdiği “sürü ortalaması” değerleri görülmektedir. “En İyi Parça Değeri” nin %40 gibi değerden başladığı beş farklı parçacık tarafından en iyi skorun elde edildiği, buna rağmen başarının en fazla % 78 değerine çıktığı görülmektedir.

“Sürü Ortalama Başarı Değerinin” %5 gibi çok düşük bir değerden başlayıp “En İyi Parça Değerine” bağlı yükselme eğilimde olduğu ancak başarının %40 sevisinde kaldığı görülmektedir. 100. iterasyon itibari ile “Sürü Ortalama Başarı Değerinin” yükselişinin devam ettiği görülmektedir.

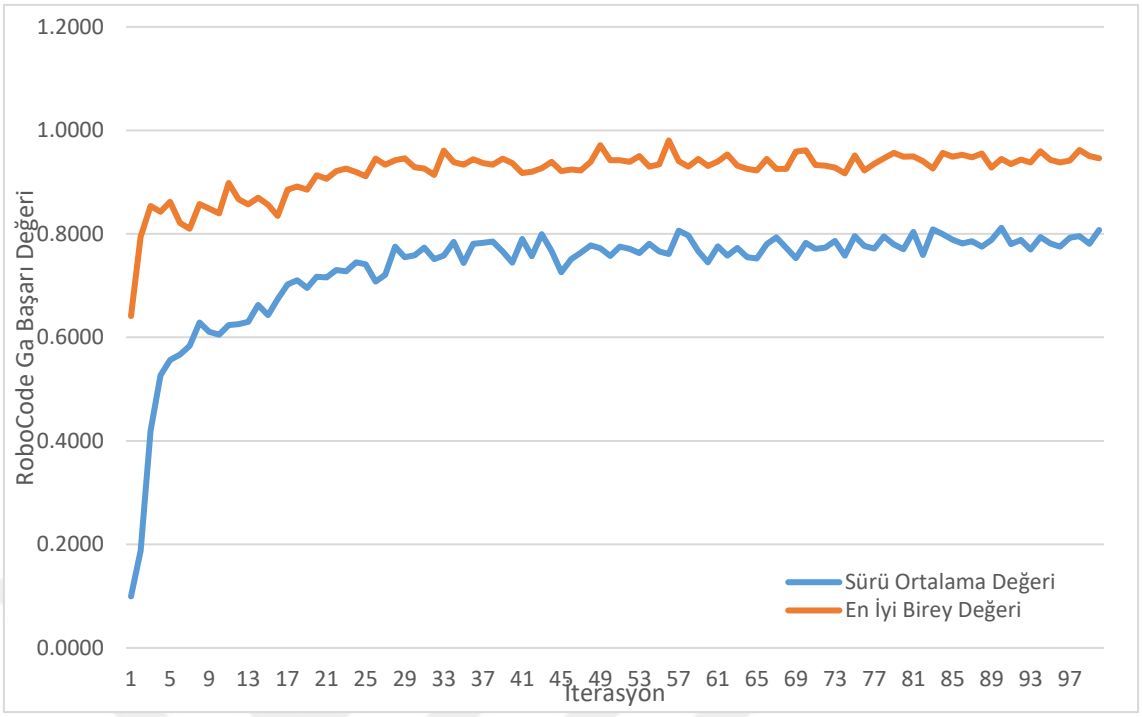


Şekil 5.4 “PaintingRobot" robotuna karşı PSO ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

5.3 3. Senaryo: “Velocirobot” İsimli Robotla Yapılan Eğitim

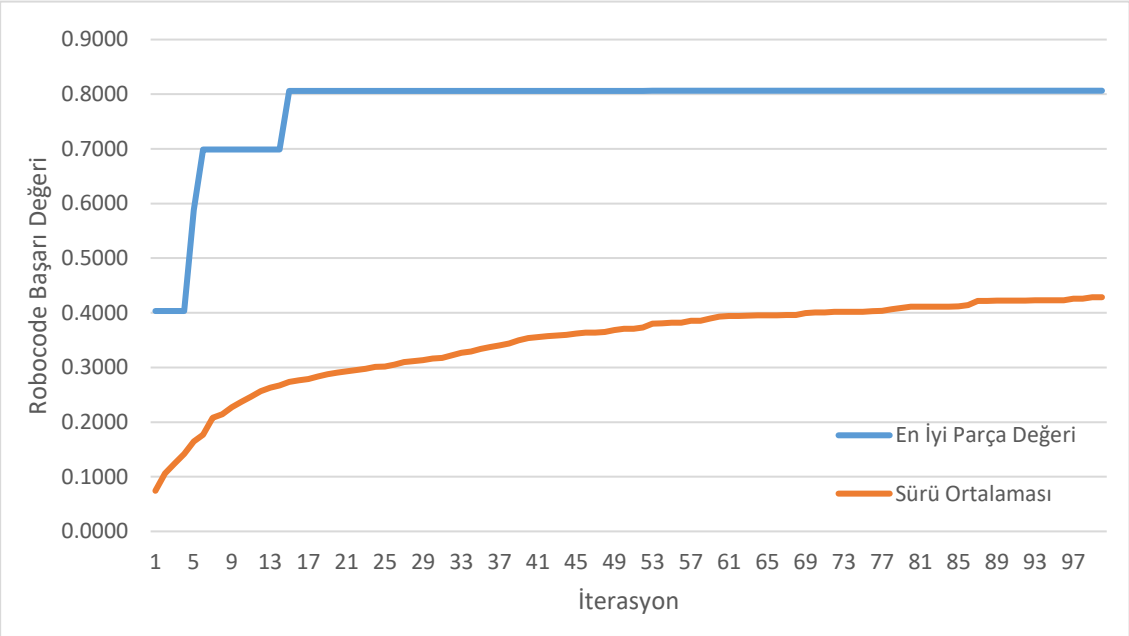
Bu senaryoda robotlar VelociRobot isimli robotla mücadele ettirilmek suretiyle eğitilmişlerdir. İlk olarak robotlar GA ile eğitilmiş, bu eğitim 10000 müsabakadan oluşmuştur. Diğer senaryolarda olduğu gibi 100 nesil boyunca eğitim devam ettirilmiştir. Şekil 5.5’de En İyi Birey Değeri %40’lık bir başarı ile başlayıp ilk beş nesil sonucunda %85 üzerine çıktığı 100 nesil sonunda %95 civarında seyrettiği görülmektedir.

Sürü ortalama değerinin %5 seviyesinden başlayıp hızlı bir artışla 10. nesilde %60, 30. nesilde % 75 seviyesine çıktığı görülmektedir. Diğer senaryolardaki sonuçlara benzer olarak sürü ortalama değerinin, en iyi birey değerinin %10-%15 altında devam ettiği görülmektedir.



Şekil 5.5 “VelociRobot" robotuna karşı GA ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Şekil 5.6’da VelociRobot ile savaştırılarak PSO ile eğitilmiş sürü robota ait sürü ortalama başarı değeri ve en yüksek skora sahip robotların oluşturduğu “En İyi Parça Değeri” grafiği görülmektedir. En iyi parça skorunun %40 değerinde başladığı, En iyi skoru yapan parçacığın üç defa değiştiği 16. İterasyonda %80’e çıktığı ve sabit kaldığı görülmektedir. Sürü ortalamasının ilk iterasyonda %7 seviyesinde olduğu düzenli bir yükselişle %45 seviyesine çıktığı ve yükselme eğiliminde olduğu görülmektedir. Elde edilen sonuçlarda dikkat çeken bir diğer sonuçta En İyi Parça Değeri ile sürü ortalaması değerlerinin birbirine diğer senaryolara göre daha uzak olmasıdır.

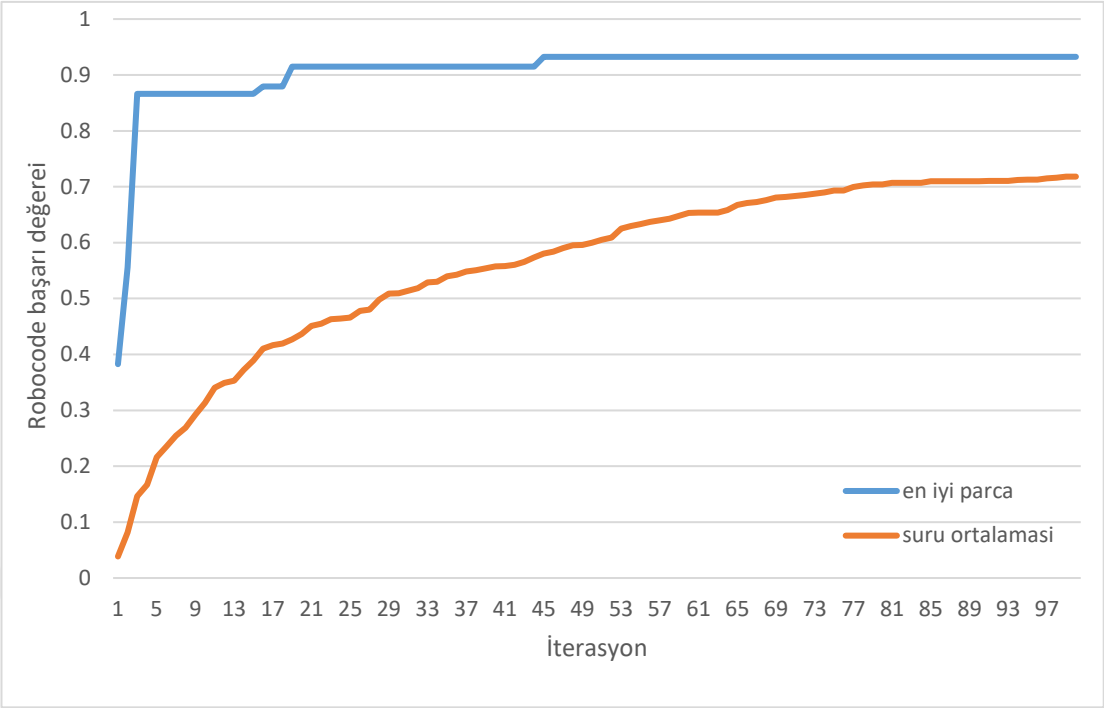


Şekil 5.6 "VelociRobot" robotuna karşı PSO ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

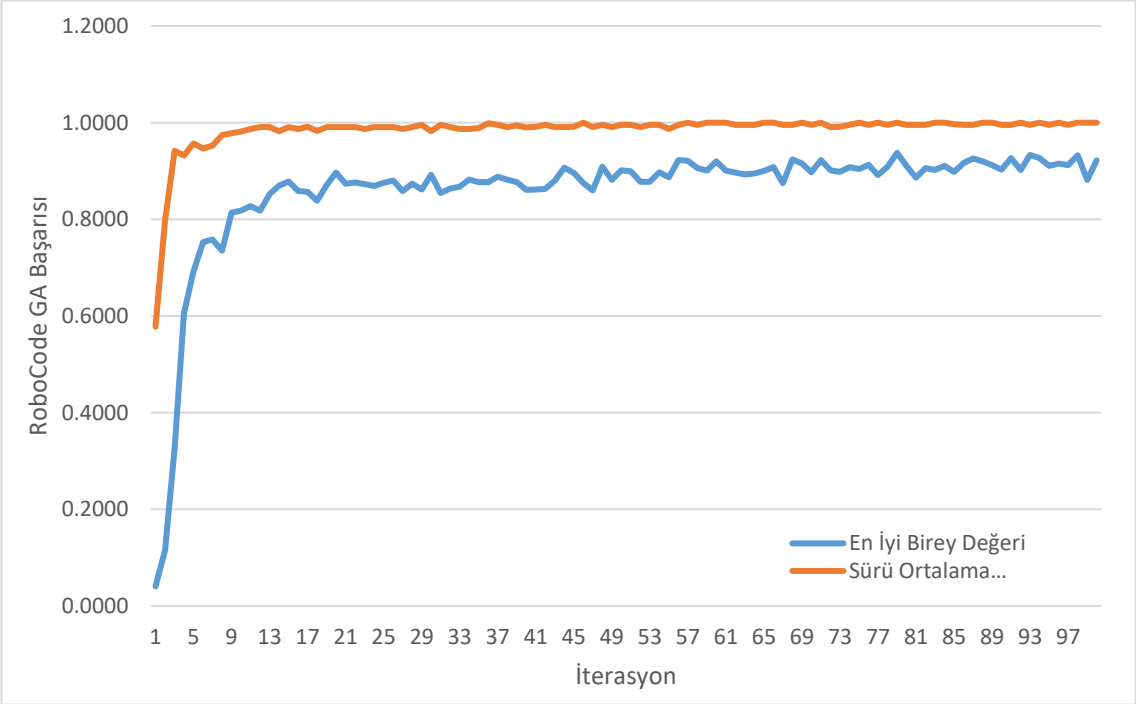
5.4 4. Senaryo: "Fire" İsimli Robotla Yapılan Eğitim.

Bu senaryoda robotlar "sample" sınıfından bir "Fire" isimli bir robotla eğitilmiştir. Fire robotunun genel özelliği hareket kabiliyetinin kısıtlı olması bunun yanında ateş gücünün yüksek olmasıdır.

İlk olarak robotlar GA ile eğitilmiş, bu eğitim 10000 müsabakadan oluşmuştur. Diğer senaryolarda olduğu gibi 100 nesil boyunca eğitim devam ettirilmiştir. Şekil 5.7'de görüldüğü gibi ilk nesilde dahi en iyi birey değeri %50'nin üzerinde görülmüştür. 10. nesilden sonra en iyi birey değeri %98 üzerine çıkmıştır. Sürü ortalama başarı değeri %5'ler seviyesinde başlamış, hızlı bir yükselişle 15. Nesilde %80'e ulaşmış ve 20. nesilden sonra %90'larda yatay bir seyir izlemiştir.



Şekil 5.7 “Fire" robotuna karşı GA ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

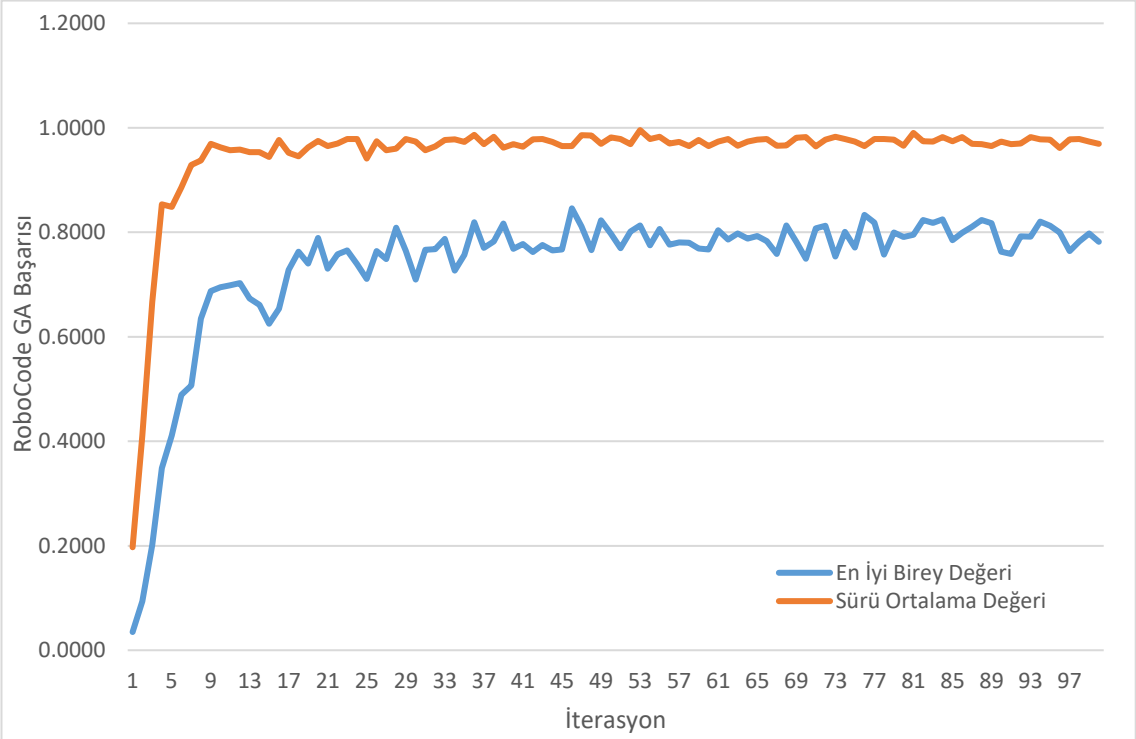


Şekil 5.8“Fire" robotuna karşı PSO ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Senaryonun ikinci aşamasında robotlar PSO algoritması ile eğitilmiştir. Bu eğitime ait başarı değeri en yüksek parçacık ve sürü ortalama başarı değeri şekil 5.8'de gösterilmiştir. Grafikte de görülebileceği gibi en yüksek başarılı parçacığa ait başarı değeri ilk iterasyonda %40 civarındadır. Birkaç iterasyon sonra %85 seviyesine ulaşıldığı görülmektedir. 100 iterasyonluk eğitim sürecinde sürüdeki 5 farklı parçacığın en yüksek değere ulaştığı görülmektedir.

5.5 5. Senaryo: "Corners" İsimli Robotla Yapılan Eğitimi

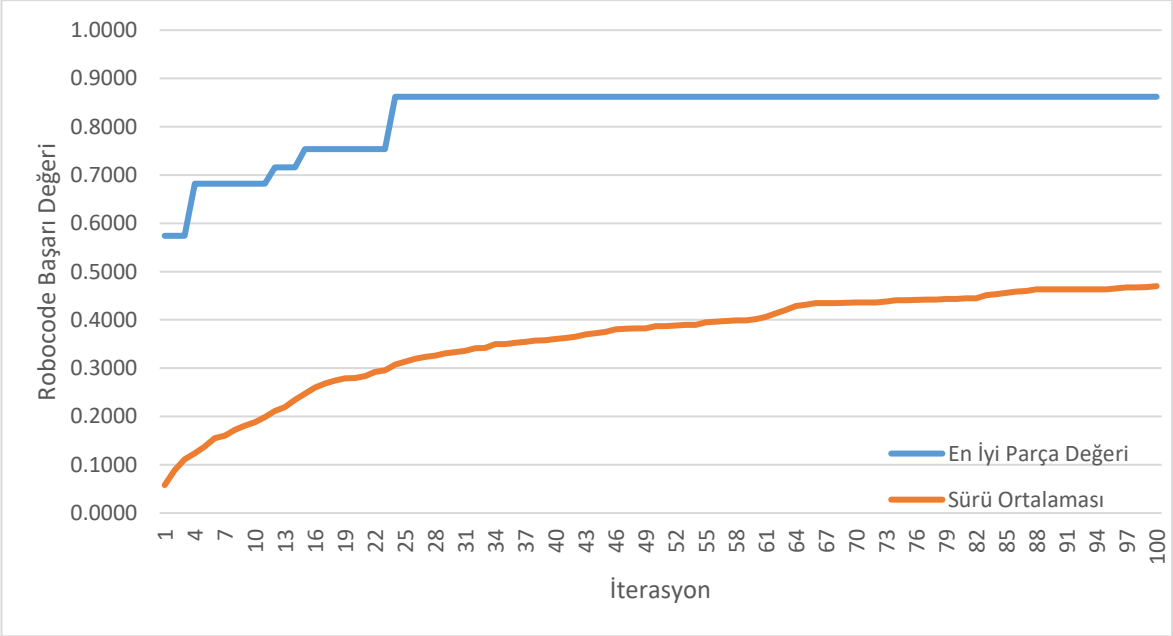
Bu senaryoda robotlar "sample" sınıfından "Corners" isimli robotla eğitilmiştir. Corners savaşın başında bir köşeye giderek burada savunma savaşı yapan bir strateji izler.



Şekil 5.9 "Corners" robotuna karşı GA ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Corners robot ile eğitime ilk olarak GA kullanılarak başlanılmıştır. GA ile eğitilen sürüye ait ortalama değer ve en iyi birey değeri şekil 5.9'da görülmektedir. İlk nesilde en yüksek başarıyı gösteren bireyin %20'lerde başarı gösterirken 10. Nesilde bu başarının %95 üzerine çıktığı görülmektedir. Buna paralel olarak sürü başarı ortalamasının da hızlı bir şekilde yükseldiği 20. nesle ulaşıldığında %80 oranında başarıya ulaştığı ve yatay bir seyir izlemeye başladığı görülmektedir. Diğer senaryolarda olduğu gibi En iyi birey ile sürü ortalaması arasında yaklaşık %20'lik bir farkın oluştuğu ve nesil sayısı arttıkça bu artıştan etkilenmeden bu farkın korunduğu görülmektedir.

Senaryonun ikinci kısmında Corners robotu ile PSO algoritmaları kullanılarak sürü eğitilmiştir. Şekil 5.10'te bu eğitime ait grafik görülmektedir. Birinci iterasyonda en iyi değere sahip parçacığın %56 olduğu görülmektedir. Başlangıca göre bu yüksek değer rastsal olarak değerlendirilmektedir. Bununla birlikte ilerleyen iterasyonlarda 5 farklı parçacığın en iyi değerlere sahip olduğu, en yüksek %86 düzeyinde başarı gösterildiği gözlemlenmektedir.



Şekil 5.10 "Corners" robotuna karşı PSO ile eğitilmiş en iyi robot değeri ve sürü ortalama başarı değeri

Çizelge 5.1 Çeşitli senaryolarda GA ve PSO ile eğitilen robot başarısı değeri ve sürü ortalama değeri

	Robot Adı	Rakip Adı	En Yüksek Bireysel Değer (%)	Sürü Ortalama Değeri (%)
Senaryo 1	GA_Crazy	Crazy	98,80	82,64
	PSO_Crazy	Crazy	89,49	75,71
Senaryo 2	GA_Painting	PaintingRobot	98,13	89,87
	PSO_Painting	PaintingRobot	76,61	40,78
Senaryo 3	GA_Veloci	VelociRobot	96,20	80,75
	PSO_Veloci	VelociRobot	80,65	42,85
Senaryo 4	GA_Fire	Fire	100	92,58
	PSO_Fire	Fire	93,24	71,81
Senaryo 5	GA_Corners	Corners	98,25	83,36
	PSO_Corners	Corners	86,21	46,99

Metodoloji kısmında yapılan çalışmalar ve elde edilen sonuçlar çizge 4'te gösterilmektedir. Çizgede her senaryo için ayrı bir satır açılmıştır. Bu satırlar da o senaryodaki hangi sezgisel algoritmanın kullanıldığına göre ayrılmıştır. Her satırda sezgisel algoritma ile eğitilmiş robotun adı, bu robotun hangi rakip robot ile karşılaştırılarak eğitildi yer almaktadır.

6. SONUÇLAR VE ÖNERİLER

6.1 Sonuçlar

Bu tez kapsamında meta-sezgisel optimizasyon algoritmalarının sürü robotların görev paylaşımında kullanılması amaçlanmıştır. Sürü robotların eğitilmesi için RoboCode savaş simülasyon platformu kullanılmış, yine bu platformda sezgisel algoritmalarla eğitilen robotların başarıları değerlendirilmiştir.

Yapılan literatür taramasında sürü robotların sezgisel optimizasyon algoritmalarından sadece GA ile eğitilme çalışmaları olduğu görülmüştür. PSO algoritmasının ise dolaylı olarak kullanıldığı görülmektedir. Bu çalışma YSA ile eğitilen robotlarda YSA ait bağlantıların PSO ile optimize edilmesi üzerinedir.

Bu çalışmada RoboCode platformunda PSO algoritması direkt olarak uygulanmıştır. Bunun yanı sıra GA algoritma ile yapılan çalışmalar incelenmiş; bu metodu kullanan çalışmalarda önerilen yaklaşımlar, sabitler göz önünde bulundurularak GA RoboCode oyununa uygulanmıştır.

Bu tez çalışmasında PSO ve GA RoboCode uygulamasına 5 senaryo düzenlenerek uygulanmıştır. Her senaryoda RoboCode platformu "sample" sınıfına ait farklı karakteristikteki robotlar kullanılmıştır. Her senaryoda bu sınıflarla GA ve PSO algoritmaları kullanan robotlar savaştırılarak PSO ve GA robotları eğitilmiştir.

Birinci senaryoda robotlar sezgisel algoritma kullanılarak "crazy" isimli robota karşı savaştırılmıştır. GA yaklaşımıyla sürü ortalama başarı değeri %82,64, en yüksek başarı puanına sahip birey değeri %98,80 seviyesine ulaşılmıştır. Gerek sürü bazında gerekse birey bazında elde edilen başarı yüzdeleri oldukça yüksek olup, GA'nın "crazy" robotuna karşı başarılı olduğu görülmektedir. Bu senaryoda ikinci olarak yine "crazy" robotuna karşı PSO kullanılarak sürü eğitilmiştir. Sürü başarısında %75,71, en yüksek başarı puanına sahip birey değeri ise %89,49 olarak ölçülmüştür. PSO algoritmasının elde ettiği bu yüzdeler başarılı olarak değerlendirilebilir. Algoritmaların bu senaryo

özelinde biri birleriyle mukayeselerinde ise GA'nın daha başarılı olduğu görülmektedir. Bunun yanı sıra GA'nın bireysel anlamda %100'lük başarıya yaklaştığı sürü başarısı açısından çok yüksek bir değere sahip olmasına rağmen uygulanan 100 iterasyon boyunca 20.iterasyondan sonra yatay bir seyir izlediği dolayısı ile gelişimi durmuştur. Ancak PSO ile eğitilen sürünün başarısının 100 iterasyon boyunca arttığı ve sürü ortalamasının artma eğiliminde olduğundan eğitimin devam ettirilmesi halinde sürü başarı ortalamasının en iyi bireysel değere yaklaşmaya devam edeceği görülmektedir.

İkinci senaryoda "sample" sınıfından PaintingRobota karşı, PSO ve GA'nın uygulanması suretiyle robotlar eğitilmiştir. PaintingRobot basit savunma yaparken gücünü saldırıdan almaktadır. GA ile eğitilen en iyi birey değeri %98,13 olarak görülmüştür. Sürü ortalaması ise %89,87 olarak ölçülmüştür. PSO ile eğitilen en iyi bireyin ise %76,61 değere ulaştığı, sürü ortalamasının ise %40,78 gibi düşük bir değerde kaldığı gözlemlenmektedir. PaintingRobota karşı GA'nın hen bireysel anlamda hem de sürü ortalaması anlamında daha iyi sonuçlara ulaştığı buna karşın PSO'nun bireysel anlamında iyi bir başarı elde etmesine rağmen sürü ortalaması açısından düşük bir başarı ortaya koyduğu görülmektedir. Her ne kadar PSO sürü ortalama başarısının yükselme eğiliminde olduğu görülse de GA ile karşılaştırıldığında GA'nın daha iyi sonuç ortaya koyduğu görülmektedir.

Üçüncü senaryoda "sample" sınıfından VelociRobota karşı, PSO ve GA uygulanarak robotlar eğitilmiştir. GA ile eğitilen en iyi birey değeri %96,20 olarak görülmüştür. Sürü ortalaması ise %80,75 olarak ölçülmüştür. PSO ile eğitilen en iyi bireyin ise %80,65 değere ulaştığı, sürü ortalamasının ise %42,85 değerinde olduğu gözlemlenmektedir. İkinci senaryoda görülen sonuçlara yakın sonuçlar gözlemlenmektedir. PSO sürü ortalamasının yine yükseliş eğiliminde olduğu. Ancak başarı ortalamasının düşük olması PSO ile eğitilen sürülerde agresif karakteristikli robotlara karşı sürü başarısının düşük olduğunu göstermektedir.

Dördüncü senaryoda "sample" sınıfından Fire robotuna karşı sürü robotlar PSO ve GA ile eğitilmiştir. Fire robot savunma özelliği hiç olmayıp saldırı gücü yüksek bir robottur. GA ile eğitilen robotlardan %100 başarı elde eden bireyler olmuştur. Sürü ortalama

başarısında ise %92,58 gibi çok yüksek bir sürü ortalaması görülmektedir. PSO ile eğitilen robotlarda en yüksek başarı %93,24, ortalama sürü başarı ortalaması ise %71,81 olmuştur. Savunma yapan robotlara karşı her iki algoritmanın da başarılı olduğu görülmektedir.

Beşinci senaryoda ise "Corners" isimli robota karşı GA ve PSO algoritmaları kullanılarak sürü robotlar eğitilmiştir. GA ile eğitilen sürüdeki en iyi başarı puanına sahip birey %98,25 değerinde bir başarı elde etmişken, sürü ortalama başarı değeri %83,36 olmuştur. PSO ile eğitilen sürüdeki en iyi başarı puanına sahip birey %86,21 değerinde başarı göstermişken, sürü ortalaması %46,99 gibi bir değerde kalmıştır. Ancak sürü ortalama değerlerinin yükseliş eğiliminde olduğu görülmektedir. Agresif bir karakteristiğe sahip "corners" robotuna karşı PSO sürü orta lama değeri diğer senaryolar ile örtüşmektedir.

Bütün senaryolar ortak değerlendirildiğinde şu sonuçlara ulaşılmaktadır. Bu çalışma özelinde GA'nın PSO ya karşı gerek bireysel başarıda gerekse sürü ortalama başarı değerinde daha yüksek bir performans göstermiştir.

GA ile eğitilen sürülere ait bireysel başarı değeri ve sürü ortalama başarı değeri ilk iterasyonlardan itibaren hızla yükselip sonrasında yatay bir seyir izlemektedirler. Bireysel olarak %100'e yaklaşan başarı değeri için bu durum normal olsa dahi, daha düşük başarı değerine sahip sürü ortalamasının yatay seyretmesi bu yöntemin kısıtlı yönünü göstermektedir.

PSO ile eğitilen sürülerin savunma yapan robotlara karşı başarısı nispeten yüksek olsa dahi, saldırgan, agresif robotlara karşı özellikle sürü ortalama başarı değerinin düşük olduğu sonucuna varılmıştır. Sürü ortalamasının 100. İterasyonda dahi yükseliş eğiliminde olması PSO'nun sürü başarısı üzerindeki etkisinin görmek için daha fazla iterasyona ihtiyaç duyulduğunun ortaya koymaktadır. PSO ile eğitilen sürülerde 100 iterasyon boyunca en iyi performansı gösteren bireyin 5-10 arasında farklı robotlar olduğu gözlenmiştir. PSO'nun yapısı gereği başarı değeri mütakip iterasyonlarda değişmeyen en iyi parçacık bir lokal maksimim olarak kabul edilir. Bu da PSO'nun

lokal maksimum veya minumumdan kurtulmada etkili olduğuna dair yapılan diğer çalışmalarla örtüşmektedir.

GA'nın PSO'ya göre daha başarılı olduğu gözlemlenmiştir. Bunu sebebi olarak GA'nın RoboCode'a uygulandığı bir çok çalışma sonucunda ağaç yapısı, sabitlerin değeri, derinlikler gibi algoritma için gerekli parametre bilgilerinde optimum değerlere ulaştığı ve bu çalışmada bu optimum değerlerden yararlanması söylenebilir. Buna karşın PSO algoritmasının RoboCode'a ilk defa bu çalışmada uygulanması, algoritmanın parametrelerinde olası optimum olmayan seçimler olduğu değerlendirilmektedir. Bir diğer neden ise GA ile PSO karşılaştırılmasının eşit şartlarda olması için eşit sürü büyüklüğü ve eşit iterasyon sayısıdır. PSO'nun GA'ya göre daha fazla iterasyona ihtiyaç duyduğu, eşit iterasyon sayısının PSO için bir kısıt olduğu sonucuna varılmıştır.

6.2 Öneriler

Bu çalışmada sezgisel yöntemlerden GA ve PSO ile RoboCode savaş simülasyon platformu üzerinde eğitilen robotların başarıları değerlenmiştir. GA ile PSO'nun aynı robotlarla eğitilmeleri ve sonrasındaki mücadeleleri neticesinde elde ettikleri başarılar değerlendirilmiştir. Bu algoritmaların özelikle ilk defa bu platforma uygulanan PSO algoritmasının diğer bir sezgisel algoritma ile karşılaştırılmadan kendi başarısını arttıracak parametre düzenlemelinin yapılacağı çalışmalara ihtiyaç vardır.

Çoklu robot savaşlarında GA ve PSO'nun eğitilmesi ve başarısı bu tezin dışında bırakılmıştır. GA'nın ve PSO'nun çoklu robot savaşlarında eğitilmeleri ve başarılarının gözlemlenmesi sonraki çalışmaların konusu olacaktır.

RoboCode savaş simülasyon platformu, sezgisel optimizasyon algoritmalarının uygulanması için elverişli bir platform olarak karşımıza çıkmaktadır. Bu tez çalışması kapsamı dışındaki çeşitli sezgisel algoritmaların bu platforma uygulanması sonraki optimizasyon çalışmalarının konusu olacaktır.

KAYNAKLAR

Alaiba, V., & Rotaru, A. (2008, October). Agent architecture for building Robocode players with SWI-Prolog. In *2008 International Multiconference on Computer Science and Information Technology* (pp. 3-7). IEEE.

Alba, E. (2005). Parallels and Metaheuristic. John Wiley, New Jersey

Alexiev, V. (2013). Machine Learning through Evolution: Training Algorithms through Competition.

Brownlee, Jason. *Clever Algorithms: Nature-Inspired Programming Recipes*. 1st Ed. LuLu Enterprises,2011. eBook.

Campo, A., Nouyan, S., Birattari, M., Groß, R. and Dorigo M. 2006. "Negotiation of goal direction for cooperative transport. Ant Colony Optimization and Swarm Intelligence" Volume 4150 of the series Lecture Notes in Computer Science, pp. 191-202.

Czajkowskia, A., & Patana, K. (2009). Real-Time Learning of Neural Networks and its Application to the Prediction of Opponent Movement in the Robocode Enviroment. In *INTERNATIONAL PHD WORKSHOP. CONFERENCE ARCHIVES* (Vol. 11).

Deep Q-Learning for Robocode, Baptiste DEGRYSE, MSc thesis, Ecole polytechnique de Louvain, 2017.

Einstein, J 2003, 'Evolving Robocode Tank Fighters', Technical Report 2003-026, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, 2003.

Gade, M, Knudsen, M, Kjær, RA, Christensen, T, Larsen, CP, Pedersen, MD & Andersen, JSK 2003, Applying Machine Learning to Robocode, Aalborg University, Aalborg, Denmark

Guesgen, H. W., & Shi, X. D. (2006). An Artificial Neural Network for a Tank Targeting System. In *FLAIRS Conference* (pp. 463-465).

Harper, R. (2014). Evolving robocode tanks for Evo robocode. *Genetic Programming and Evolvable Machines*, *15*(4), 403-431.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM journal on computing*, *2*(2), 88-105.

Hong, J, & Cho, SB 2004, 'Evolution of Emergent Behaviours for Shooting Game Characters in Robocode', Congress on Evolutionary Computation 2004, vol. 1, pp. 634-638.

Inja, M. T. (2012). Genetic Programming and Robocode.

Karaboğa, D. (2017). Yapay Zeka Optimizasyon Algoritmalari. Nobel Akademi Yayıncılık.

Kayakoku, H., Guzel, M. S., Bostanci, E., Medeni, I. T., & Mishra, D. (2021). A Novel Behavioral Strategy for RoboCode Platform Based on Deep Q-Learning. *Complexity*, *2021*.

Kennedy, J., Eberhart, R. C. (1995). Particle swarm optimization (pp: 1942-1948). *Proc. IEEE International Conference on Neural Network*, Perth, WA, Australia.

Kobayashi, K., Uchida, Y., & Watanabe, K. (2003, August). A study of battle strategy for the Robocode. In *SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734)* (Vol. 3, pp. 3373-3376). IEEE.

Koza, John R., and Riccardo Poli. "Ch. 5: Genetic Programming." Trans. Array *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Edmund K Burke and Graham Kendall. 2nd Ed.Springer, 2005. 127-164.

Murty, K. G., 2003, Optimization Models For Decision Making: Volume 1, Internet Edition, Chapter 1: Models for Decision Making, 1-18.

Navarro, I. and Matía, F. 2012. "An Introduction to Swarm Robotics". ETSI Industriales, Universidad Politécnica de Madrid, c/José Gutiérrez Abascal, 2, 28006 Madrid, Spain.

Nielsen, J. L., & Jensen, B. F. (2011). Modern AI for games: RoboCode.

Obaidy, M. Al. and Ayesh, A. 2008. "The Implementation of Optimization Algorithm for Energy Efficient Dynamic Ad hoc Wireless Sensor Networks", Proceedings of the Fourth International Workshop on Advance Computation for Engineering Applications, Japan, pp. 17-23

Panduro, M. A., Brizuela, C. A., Balderas, L. I. and Acosta, D. A., 2009. "A comparison between Genetic Algorithms, Particle Swarm Optimization and the Differential Evolution method for the design of scan able circular antenna arrays", Progress In Electromagnetic Research B, Vol. 13, 171-186.

Shabir, S., & Singla, R. (2016). A comparative study of genetic algorithm and the particle swarm optimization. *Int. J. Electr. Eng*, *9*(2), 215-223.

Shichel, Y, Ziserman, E & Sipper, M 2007, 'Designing an Evolutionary Strategizing Machine for Game Playing and Beyond', IEEE Transactions on Systems, Man, and Cybernetics: Part C, vol. 37, no. 4, pp. 583-593.

Simha, R., Carnahan, J., 2001, Nature's Algorithms: Natural and Social Metaphors in Algorithm Design, IEEE Potentials, Vol. 20, No. 2, 21-24.

Wyatt, D., & Klein, D. (2003). *Genetic Programming for Robocode Strategy*. Tech. Rep., University of Washington.

Yang, X. S. (2013). Optimization and Metaheuristic Algorithms in Engineering. Metaheursitics in Water, Geotechnical and Transport, Elsevier.

.