T.C.

BURSA TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

PEKİŞTİRMELİ ÖĞRENME İLE ROBOT KOL YÖRÜNGE KONTROLÜ

YÜKSEK LİSANS TEZİ

Abdurrahman Sefer DOĞRU

Mekatronik Mühendisliği Anabilim Dalı

EYLÜL 2022

T.C.

BURSA TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

PEKİŞTİRMELİ ÖĞRENME İLE ROBOT KOL YÖRÜNGE KONTROLÜ

YÜKSEK LİSANS TEZİ

Abdurrahman Sefer DOĞRU

(19241782025)

Mekatronik Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Ahmet MERT

Eylül 2022

BTÜ, Lisansüstü Eğitim Enstitüsü'nün 19241782025 numaralı Yüksek Lisans Öğrencisi Abdurrahman Sefer DOĞRU, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "PEKİŞTİRMELİ ÖĞRENME İLE ROBOT KOL YÖRÜNGE KONTROLÜ" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı: **Doç. Dr. Ahmet MERT** .....

Bursa Teknik Üniversitesi

Jüri Üyeleri: **Doç. Dr. Gökhan GELEN** .....

Bursa Teknik Üniversitesi

Dr. Öğr. Üyesi Ulvi BAŞPINAR .....

Marmara Üniversitesi

Teslim Tarihi **:**

Savunma Tarihi **: 26 Eylül 2022**

20.04.2016 tarihli Resmî Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, Bursa Teknik Üniversitesi'nin abonesi olduğu intihal yazılım programı kullanılarak Lisansüstü Eğitim Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

İNTİHAL BEYANI

Bu tezde görsel, işitsel ve yazılı biçimde sunulan tüm bilgi ve sonuçların akademik ve etik kurallara uyularak tarafımdan elde edildiğini, tez içinde yer alan ancak bu çalışmaya özgü olmayan tüm sonuç ve bilgileri tezde kaynak göstererek belgelediğimi, aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim.

Öğrencinin Adı Soyadı: Abdurrahman Sefer DOĞRU

İmzası:

Eşime ve çocuklarıma,

ÖNSÖZ

Yüksek lisans eğitimim boyunca değerli katkılarını esirgemeyen değerli danışmanım Doç. Dr. Ahmet MERT'e sonsuz teşekkürlerimi sunarım.

Bu süreçte bilgi, birikim ve değerli fikirleriyle zamanlarını ayırıp tezin gelişimine katkı sunan Abdullah TÜRKMEN'e, M. Ebubekir TORBALI'ya ve ofislerini açarak çalışma ortamı sunan ÜÇGEM MEKANİK A.Ş.'ye teşekkürlerimi borç bilirim.

Bu yola çıkmaya teşvik eden, her daim destek ve anlayışını eksik etmeyen değerli eşime, anneme ve babama en derin sevgi ve hürmetlerimle teşekkür ederim.

Eylül 2022

Abdurrahman Sefer DOĞRU
(Makine Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

DDPG **:** Derin Deterministik Politika Gradyanı

DÖ **:** Derin Öğrenme

DPÖ **:** Derin Pekiştirmeli Öğrenme

İHA **:** İnsansız Hava Aracı

MKS **:** Markov Karar Süreci

PÖ **:** Pekiştirmeli Öğrenme

SAC **:** Yumuşak-Aktör Kritik (İng. Soft-Actor Critic)

TD3 **:** İkiz Gecikmeli Politika Gradyanı (İng. Twin Delayed Policy Gradient)

YSA **:** Yapay Sinir Ağı

SEMBOLLER

$\boldsymbol{A}$	**:** Eylemler kümesi
$\boldsymbol{a}$	**:** Eylem
$\boldsymbol{a_t}$	**:** $\boldsymbol{t}$ anında ki eylem
$\mathbb{E}$	**:** Beklenen, tahmin edilen
$\boldsymbol{\hat{q}}$	**:** Tahmin edilen q değeri
$\boldsymbol{r}$	**:** Ödül
$\boldsymbol{S}$	**:** Durumlar kümesi
$\boldsymbol{s_t}$	**:** $\boldsymbol{t}$ anında ki durum (gözlemler)
$\boldsymbol{t}$	**:** Zaman, adım
$\boldsymbol{v}$	**:** Değer fonksiyonu
$\boldsymbol{v_\pi(s)}$	**:** Pi politikasını takip eden değer fonksiyonu
$\boldsymbol{v_\pi(a\|s)}$	**:** $\boldsymbol{s}$ durumunda $\boldsymbol{a}$ eyleminin değeri
$\boldsymbol{\alpha}$	**:** Öğrenme oranı
$\boldsymbol{\gamma}$	**:** Azaltma faktörü
$\boldsymbol{\pi}$	**:** Politika
$\boldsymbol{\pi_*}$	**:** Optimal politika

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

PEKİŞTİRMELİ ÖĞRENME İLE ROBOT KOL YÖRÜNGE KONTROLÜ

ÖZET

Robotların kullanımı son yıllarda teknolojinin gelişmesine bağlı olarak yaygınlaşmakta ve görevlerini otonom olarak gerçekleştirmeleri ile ilgili çalışmalar da gün geçtikçe artmaktadır. Verilen işleri insanlara nazaran daha güçlü ve hassas olarak yapabilme potansiyeli olan robot kolları, endüstriyel tesislerin yanı sıra ameliyathanelerde ve uzay görevleri gibi farklı alanlarda da kullanılarak geniş bir alanda hizmet vermektedir. Robot kol kontrolü problemine literatürde mevcut bulunan model tabanlı yaklaşımlar, fiziksel sistemi ifade eden matematiksel modelin oluşturulması ve kontrol algoritmalarının geliştirilmesini gerektirmektedir. Bu matematiksel modellerin eldesi her zaman kolay olmamakla birlikte çözümlerinin yapılamaması gibi durumlar bulunabilmekte ve bu yöntemler istenen performansı gösterememektedir.

Günümüz teknolojisinin son bahis konularından olan yapay zekâ bu durum için bir çözüm ve kolaylık sağlama potansiyeli barındırmaktadır. Bu alanda etkili bir çözüm makine öğrenmesinin bir alt dalı olan Pekiştirmeli Öğrenmedir (PÖ). PÖ, herhangi işlenmiş bir veriye ihtiyaç duymadan, keşif yaparak ve bu keşifte ortaya konulan davranışları belirli kriterler doğrultusunda değerlendirerek öğrenme süreciyle optimum davranışları öğrenir. Böylece matematiksel modellemelere ihtiyaç duymadan bir kontrol seçeneği sunar. Bu yönüyle PÖ endüstriyel robotlardan insansız hava araçlarına kadar birçok alanda araştırılmış ve geliştirilmeye devam etmektedir.

Bu çalışmada matematiksel modelden bağımsız olarak robot kol yörünge kontrol problemi için PÖ algoritmaları kullanılmıştır. Sürekli durum ve eylem uzayları için geliştirilmiş model-bağımsız, politika-dışı, aktör-kritik PÖ algoritmaları, derin deterministik politika gradyanı (DDPG), ikiz gecikmeli politika gradyanı (TD3) ve soft aktör-kritik (SAC) kullanılmıştır. Bu algoritmalar kendi aralarında eğitim süreçleri ve eğitilmiş ajanların simülasyonu aracılığıyla görevleri yerine getirmelerindeki doğru konumlandırma, istenilen sürede yerine getirme gibi çeşitli parametrelerce kıyaslanmıştır. Robot kolun bulunduğu çevreye sabit engeller konularak engelli çevre oluşturulmuş ve bu engelli çevre için eğitim süreçleri ve simülasyonlar tekrarlanmıştır. Çevrede bulunan engel ile çarpma gibi bozucu bir sinyalin eğitim süreçlerine ve ajanların davranışlarına etkileri gözlemlenmiştir.

Algoritmalar ile ajanların hazırlanması, eğitim ve test süreçleri Matlab programı kullanılarak gerçekleştirilmiştir. Robot kol modeli Matlab kütüphanesinden alınmış gerekli modifikasyonlar ve çevre modellemesi yine Matlab'ın Simulink/Simscape arayüzü kullanılarak yapılmıştır. Sonuç olarak algoritmaların verilen görevleri yüksek bir başarı oranı ile yerine getirmeyi öğrenebildiği görülmüştür.

Anahtar kelimeler: Pekiştirmeli Öğrenme, Engelden Kaçınma, Robot Kol, DDPG, TD3, SAC.

ROBOTIC ARM TRAJECTORY CONTROL WITH REINFORCEMENT LEARNING

SUMMARY

Robots have become widespread in recent years due to the development of technology, and studies on performing given tasks autonomously are increasing daily. Robot arms, which have the potential to achieve given tasks more powerfully and more precisely than humans, are used in a wide range of areas, such as operating rooms and space missions, as well as industrial facilities. Model-based approaches to the robot arm control problem in the literature require creating a mathematical model expressing the physical system and developing control algorithms. However, these mathematical models are not always easy to obtain, there may be situations such as being unable to solve them, and these methods do not show the desired performance.

Artificial intelligence, which is one of the last topics of today's technology, has the potential to provide a solution and convenience for this situation. A practical solution in this area is Reinforcement Learning (RL), a sub-branch of machine learning. Reinforcement learning learns the optimum behaviours through the learning process by making discoveries and evaluating the behaviours revealed in this discovery according to specific criteria, without needing any processed data. Thus, it provides a control option without the need for mathematical models. In this respect, RL has been researched and developed in many areas, from industrial robots to unmanned aerial vehicles.

This study used RL algorithms for the robot arm trajectory control problem, regardless of the mathematical model. Deep deterministic policy gradient (DDPG), twin-delayed policy gradient (TD3) and soft actor-critic (SAC) algorithms which are model-free, off-policy, actor-critic reinforcement learning algorithms that developed for continuous state and action spaces, were used. These algorithms were compared among themselves by various parameters such as correct positioning and fulfilment in the desired time in performing tasks through training processes and simulation of trained agents. A second environment with obstacles was created by placing fixed obstacles in the environment where the robot arm is located, and training processes and simulations were repeated for this environment. The effects of a disruptive signal, such as hitting an obstacle in the environment, on the training processes and the behaviour of the trained agents were observed.

The agents' preparation, training and testing processes with the algorithms were carried out using the Matlab program. The robot arm model was taken from the Matlab library, and the necessary modifications and environment modelling were made using Matlab's Simulink/Simscape interface. As a result, it has been seen that the algorithms can learn to perform the given tasks with a high success rate.

Keywords: Reinforcement Learning, Obstacle Avoidance, Robotic Arm, DDPG, TD3, SAC

1. GİRİŞ

Robot kolları insan vücudundan esinlenerek insanların yaptığı işleri onlardan daha güçlü ve hassas bir şekilde yapabilen makinelerdir. Endüstriyel tesislerin ana bileşenleri haline gelen robot kolları, kaynak uygulamaları, malzemelerin taşınması, boyama işlemleri, paketleme, montaj, kesme, cilalama, yapıştırma, ameliyat gibi çok geniş bir alanda kullanım imkanına sahiptir[1]. Robot kolları, azaltılmış işçilik, işletme maliyetleri ve artan işçi güvenliği gibi avantajları ile ön plana çıkmaktadır. Farklı serbestlik dereceleri, farklı tasarımlar ve farklı kontrol algoritmaları ile endüstriyel uygulamalarda kullanılan robot kolları için yeniden programlanabilirlik, tekrarlanabilirlik, taşıma kapasitesi ve hız gibi özellikler karakteristik olarak karşımıza çıkmaktadır. Farklı tasarım ve işlevsellikteki robot kollar Şekil 1.1'de verilmiştir [2].

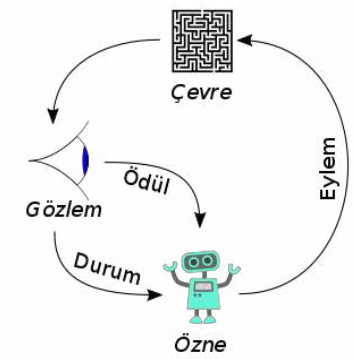
Şekil 1.1 : Farklı tasarım ve işlevsellikteki robot kollar.

Robotları programlamak özel bilgi ve yetenek gerektirir. Belirtilen görevleri yerine getirmek ve varsayılan konumlara ulaşmak için robotun hareketlerinin planlanması gerekmektedir. Bu planlama için birçok yöntem bulunmaktadır [3]. Mevcut yöntemlerde hareket şekilleri ve robotun ulaşması gereken konumlar ile ilgili birçok detayın belirtilmesi gerekir. Bu kadar detayı belirtmek yerine robotun nasıl davranacağını öğretmek ve robotun öğrenmesi fikri Pekiştirmeli Öğrenme [4] alanındaki gelişmelerle birlikte ilgi çekmektedir. Literatürde görsel sistemler ile insan eli yörüngesinin parametreleri robota aktarılarak kontrol etmek [5], beyin sinyallerini kullanmak [6] gibi farklı önerilerde mevcuttur. Yalnızca robotun ne yapması gerektiği

ve hangi konuma gitmesi gerektiğini belirledikten sonra bunu nasıl yapacağını öğrenmesine izin vermek daha kolaydır.

Literatürde makine öğrenimini kullanmaya yönelik çeşitli çalışmalar bulunmaktadır. Mahadevan [7] robot makine öğrenmesi için bir dizi farklı yaklaşım sunup karşılaştırmıştır. Bir robotun dinamikleri öğrenmesi için parametrik olmayan öğrenme tekniği üzerine Schaal ve Atkeson çalışma yapmış [8] yaklaşık 100 tekrarlı bir eğitim ile robotu eğitmeyi başarmışlardır. Aynı yazarlar bir diğer çalışmalarında sarkaç sallama görevini, görevi yerine getiren insanı izleterek öğretme metodunu kullanmışlar [9]. Bu yöntem ile aynı işlemin tekrar edildiği, nokta kaynak uygulaması gibi, uygulamalar için uygun olduğunu ve daha karmaşık görevlerde yeterli olmadığı sonucuna varmışlardır.

Pekiştirmeli Öğrenme (PÖ), robotik sistemlerin kontrol politikalarını verimli bir şekilde öğrenmesini sağlayan, gelecek vaat eden bir Makine Öğrenimi dalıdır. Bir robotun kontrolündeki farklı görevler, sıralı karar verme süreçleri olarak kabul edilir ve PÖ sürecinde her adımda bir karar alınması gerekir. Burada ortam (çevre) ile etkileşimde olan bir ajan (robot) bulunmaktadır. Kontrol sürecinin her adımında, ajan bir eylem almaya karar verir (motor hareketleri gibi) ve sensörleri aracılığıyla ortamın durumunu gözlemler. Robot ortam ile iletişime geçtiğinde ortam değişir ve karşılığında bir ödül verilir. Bu çevrim temsili olarak Şekil 1.2'de verilmiştir. Robotun amacı, optimum politikayı keşfederek elde edilen toplam ödülü maksimize etmektir. Robot politikasını çevresi ile deneme-hata etkileşimi ile günceller. Böylece PÖ robotun hedefe ulaşmak için en iyi davranışları otonom olarak keşfetmesini(öğrenmesini) sağlar [10].



Şekil 1.2 : Pekiştirmeli öğrenmede, özne(ajan)-çevre etkileşim çevrimi.

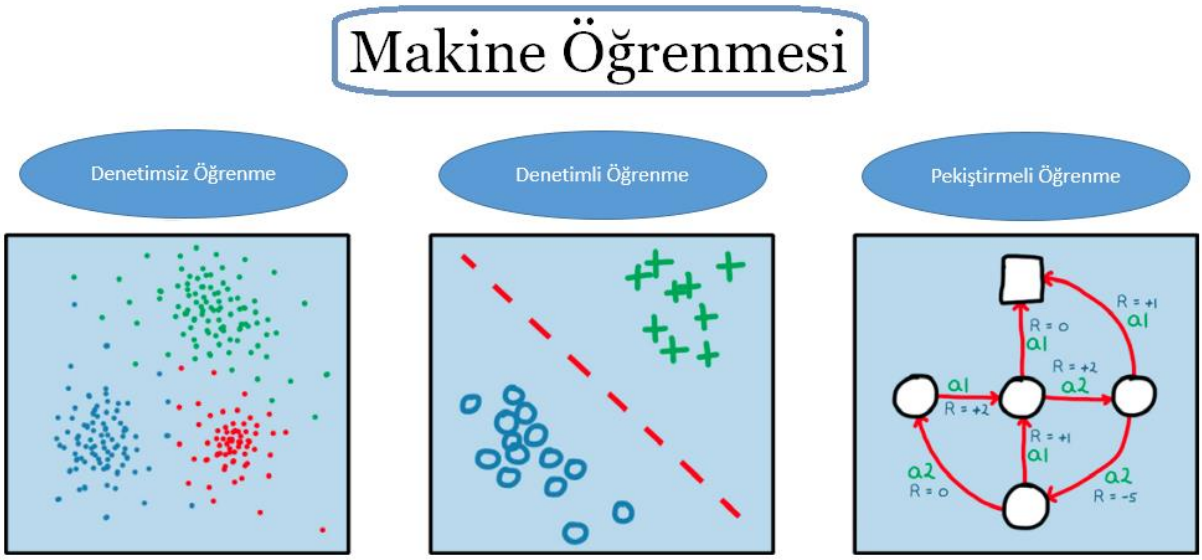
Özakar (2016), yüksek lisans tez çalışmasında ters sarkacın dengede tutulması, top düşürmeme ve top fırlatma problemlerinin çözümü için PÖ algoritmalarını uygulamış ve elde ettiği sonuçları detaylı analizler ile ortaya koymuştur [11]. Engin (2019), yüksek lisans tez çalışmasında bir mobil robot modelinin kontrolünü PÖ algoritmaları ile yapmış ve iki boyutlu labirent ortamında robotun hedefe ulaşana kadar geçen zaman, adım sayısına bağlı olarak kıyaslamalarını yapmıştır [12]. Yamaç (2020), yüksek lisans tez çalışmasında kablo ile sürülen paralel robotların denetiminde PÖ algoritmalarını uygulamış ve olumlu sonuçlar almıştır [13].

Pekiştirmeli Öğrenme ile robot kontrolü, birçok avantajı ile öne çıkıp kolaylıklar sunmasının yanında iyi politikalar oluşturmak, uygun ödül işlevlerini, durumu ve eylem alanlarını dikkatlice tanımlama gibi bazı zorlukları da beraberinde getirmektedir [14]. Bu seçimleri yapmak için benzersiz bir metodoloji yoktur ve parametre ayarları zaman alıcıdır [15]. Bu sebeple PÖ algoritmalarının performanslarının kıyaslanması ve sonuçların yayınlanması daha sonraki yapılacak çalışmalara ışık tutacaktır. Bu çalışmada robot kolunun yörünge kontrolü için birkaç farklı PÖ algoritması uygulanmasının ve sonuçlarının yolların şekli, konumlandırma doğruluğu gibi parametrelerce kıyaslanması hedeflenmektedir.

2. MAKİNE ÖĞRENMESİ

Bir problemin tam olarak çözüm yolu bilindiğinde, girdileri belirli işlemlerden geçirip kesin bir çıktı veren bir algoritma yazmak mümkündür. Fakat, bazı problemlerin çözüm yolu net olarak bilinemeyebilir veya matematiksel olarak çok karmaşık olabilir. Makine öğrenmesi metotları bu durumlara bir çözüm sunma kapasitesini barındırmaktadır. 1959 yılında Arthur Lee Samuel'in [16] çalışmasında bahsetmesiyle kavramlaşan makine öğrenmesini, bilgisayarlara eldeki veriler ve deneyimlerden anlamlı çıkarımlar yapmayı veya tahminlerde bulunmayı öğretmek olarak tanımlayabiliriz [17]. Günümüz bilgisayar teknolojilerinin gelişmesiyle artan hesaplama gücü, makine öğrenmesini tıptan, pazarlamaya, robot kontrolünden, uzay çalışmalarına kadar çok geniş bir araştırma sahasında uygulanabilir kılmıştır.

Temel olarak üç makine öğrenmesi kategorisi bulunmaktadır. Şekil 2.1'de gösterildiği gibi bunlar denetimli öğrenme, denetimsiz öğrenme ve pekiştirmeli öğrenmedir [18].



Şekil 2.1 : Makine öğrenmesi metotlarının sınıflandırılması.

Denetimsiz öğrenmede veri seti geliştirici tarafından etiketlenmemiş, bağımsız değişkenlere bağlı olarak bir çıktı ortaya konulmamıştır. Bu veri seti ile algoritma veriler arasında ilişkileri bulup, bu ilişkilere dayalı olarak verileri sınıflandırarak bir model oluşturur.

Denetimli öğrenmede ise algoritmalar geliştirici tarafından girdi ve çıktıları etiketlenerek hazırlanmış veri setleriyle eğitime tabi tutulur. Bu eğitim sürecinde algoritma etiketlenmiş verileri kullanarak girdiler ve çıktılar arasındaki ilişki ve

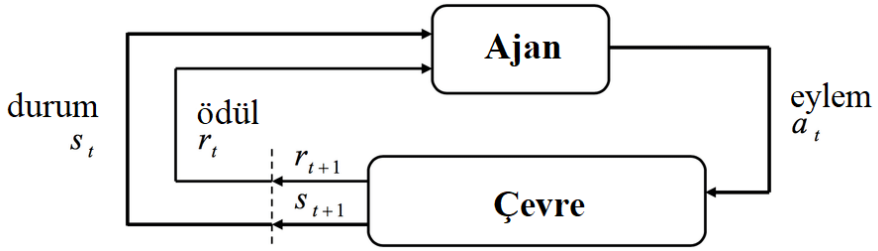
kuralları öğrenir. Böylece yeni girilecek verilere bu kuralları uygulayarak çıktıyı tahmin eder.

Son olarak Pekiştirmeli Öğrenmede ise algoritma bulunduğu durumda bir eylem gerçekleştirir ve buna karşılık bir ödül sinyali üretilir. Bu ödül sinyaline göre kendini geliştiren algoritmanın amacı ödülünü maksimize etmektir. Bu tezde makine öğrenmesi metotlarından pekiştirmeli öğrenme algoritmaları kullanılacaktır.

2.1 Pekiştirmeli Öğrenme

Pekiştirmeli Öğrenme (PÖ), numerik bir ödül sinyalini maksimize etmek için ne yapılması – durumlar ve eylemler arasında nasıl bağlantı kurulması- gerektiğini öğrenmedir.[4] PÖ'de belirgin bir hedefi olan ve çevresiyle etkileşime girip bunu değerlendirebilen bir ajan bulunmaktadır. Ajana tam olarak ne yapması gerektiği söylenmez, bunun yerine belirli kurallar çerçevesinde ajan deneme ve yanılma metoduyla bulunduğu çevreyi keşfederek, keşfederken elde ettiği tecrübeyi de kullanarak, ne yapması gerektiğini öğrenir. Ajanın dışında politika (π), ödül (r), değer fonksiyonu (v) ve zorunlu olmayan çevre modeli de alt unsurlardır. Politika, ajanın farklı çevresel durumlara nasıl tepki vermesi gerektiğini tanımlar. Ödüller ise durum (s) ve bu durumdaki eyleme (a) bağlı olarak çevrenin ajana vermiş olduğu sayısal değerlerdir. Değer fonksiyonu da bir politikaya bağlı olarak hareket eden ajanın bulunduğu durumdan itibaren uzun vadeli ödül getirilerini hesaplayan bir araçtır. Çevre modeli ise ortamı anlayarak algoritma performansını arttırmaya yardımcı olan fakat bulunması zorunlu olmayan diğer bir unsurdur.

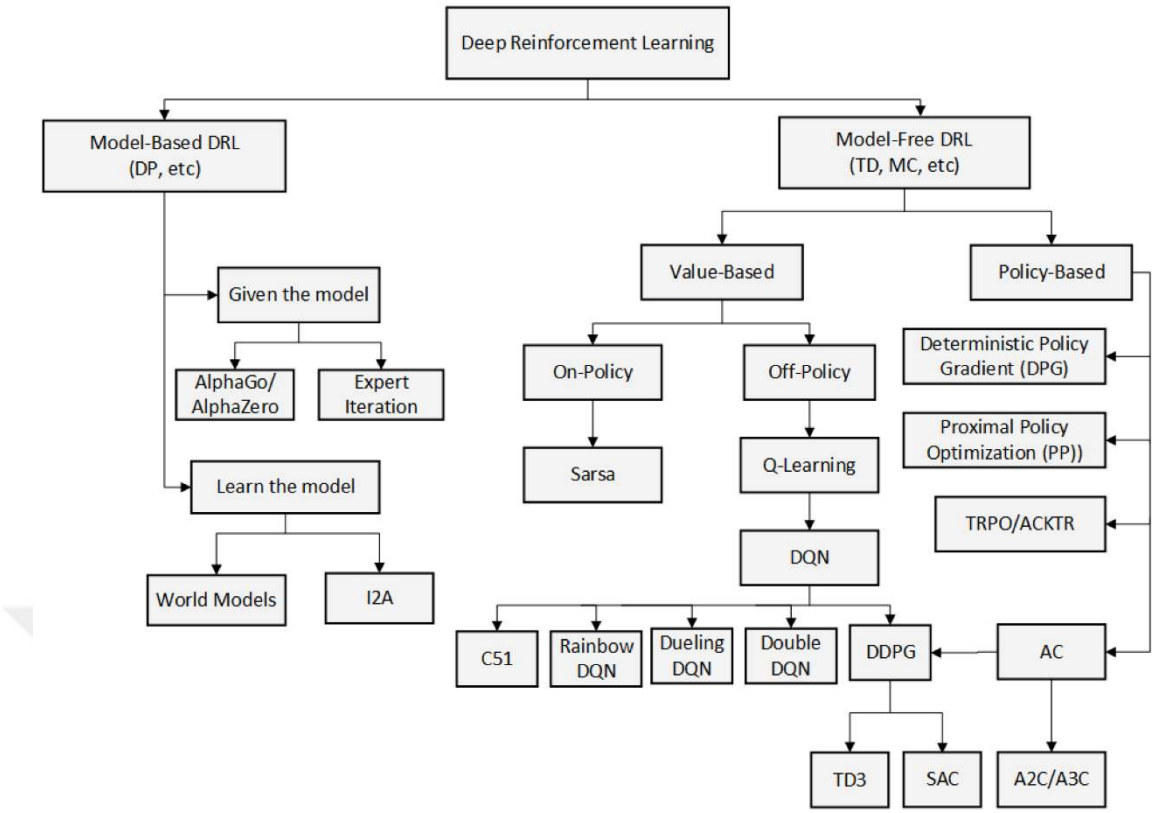
PÖ algoritmalarının temel amacı, ajanın hedeflenen görevi en yüksek ödül ile ve en doğru şekilde yerine getirmesi için gereken optimal politikayı (π^*) en hızlı şekilde öğrenmesini sağlamaktır. Ajan çevresi ile t ayrık zamanlarında etkileşime girer. Ajan t anında eylemler havuzundan (A) bir $a_t \in A$ eylemi gerçekleştirerek çevresiyle etkileşime girdiğindeki etkileşimi Şekil 2.2'de gösterilmiştir. Bu a_t eylemi sonucunda ajan s_t durumunda hedefe yaklaşması veya uzaklaşmasına bağlı olarak bir r_t ödülü alır. Bu ödül hedefe yaklaşma durumunda getirisi büyük aksi durumlarda küçük veya negatif bir ceza olabilir. Örneğin bir mobil robotun güzergâh planlama eğitimi sürecinde, t anında alınan a_t aksiyonu (örn., ileri git) ile ileri giden mobil robot hedefe yaklaşmış veya uzaklaşmış olabilir. Eğer ajan $s_t \in S$ durumundan hedefe yaklaşmışsa ödül $r_t \in R$ pozitif, uzaklaşmışsa daha küçük veya negatif olabilir.



Şekil 2.2 : Ajan ve çevresi ile etkileşimi.

Derin Pekiştirmeli Öğrenme (DPÖ), derin öğrenme (DÖ) yaklaşımlarını PÖ algoritmalarında kullanarak tecrübelere dayalı optimum çözümü sunmayı hedefleyen bir makine öğrenmesi alanıdır. Bu tecrübeler ajanın en iyi davranışını belirlemek için yapılan tekrarlara ve ödül işlevinin değerlendirmesine dayanır. Örneğin bir insansız hava aracı (İHA)'nın tüm sistemi, ajan ve çevre arasındaki etkileşime dayandığından, optimal stratejiye ulaşmak, yüksek boyutlu girdiler ve çevrenin doğru bir şekilde temsil edilmesini gerektirir. PÖ algoritmasını bir İHA sistemine uygulamak Markov karar süreçlerinin (MKS'ler) özelleştirilmiş bir temsili olarak düşünülebilir[19]. Geleneksel PÖ algoritmaları, İHA'daki geçmiş durumun doğru bir tanımını oluşturmak için yüksek kaliteli ve çok boyutlu girdiler gerektirir. Bu gereksinimler, geleneksel PÖ yaklaşımlarının kapasiteleri sebebiyle yetersiz kalmalarına neden olur. DPÖ bu duruma bir çözüm üretmeyi vaat etmektedir.

PÖ algoritmaları model tabanlı, modelden bağımsız, değer temelli veya politika temelli metotlar, Monte Carlo metotları ve zamansal fark metotlar, politikalı(on-policy) ve politika-dışı(off-policy) metotlar gibi farklı şekillerde kategorize edilmiştir. Bu kategorizelere ait bir şema Şekil 2.3'te verilmiştir. [20]



Şekil 2.3 : Pekiştirmeli öğrenme algoritmalarının tasnifi. DP (Dynamic Programming), TD (Temporal Difference), MC (Monte Carlo Methods), I2A (Imagination-Augmented Agent), DQN (Deep Q-Network), TRPO (Trusted Region Policy Optimization), ACKTR (Actor Critic using Kronecker-Factored Trust Region), AC (Actor-Critic), A2C (Advantage Actor Critic), A3C (Asynchronous Advantage Actor Critic), DDPG (Deep Deterministic Policy Gradient), TD3 (Twin Delayed DDPG), SAC (Soft-Actor Critic).

Bu tezde modelden bağımsız, değer temelli, politika-dışı (off-policy) algoritmalardan DDPG, TD3 ve SAC algoritmaları yörünge kontrolü için üç serbestlik dereceli bir robota simülasyon ortamında eğitime tabi tutulacak. Eğitim sonrası yörünge kontrol görevi simule edilerek ajanların performansları çeşitli parametrelerce kıyaslanacaktır.

2.1.1 Kontrol görevlerinde genel elemanlar

Kontrol görevleri, belirli bir görevi çözmek için her an uygun eylemlerin yapılması gereken problemlerdir. Bu bölümde PÖ'nün temellerine girmeden önce tüm kontrol görevlerinde bulduğumuz beş unsur incelenecektir. Bunlardan durum (S), görev ortamının anlık olarak hangi durumda olduğunu açıklayan tüm bilgileri içeren ilk temel unsurumuz. Robotik kol kontrol probleminde durum manipülatörün pozisyonu, açısal ve çizgisel hızları, bir satranç oyununda her bir taşın tahtada bulunduğu

konumlar örnek olarak verilebilir. Durum her zaman içinde bulunduğu an ile bağlantılıdır, yani duruma bir t zamanına atıfta bulunuruz (s_t).

Tüm kontrol görevlerinde bulduğumuz bir diğer unsur, görev sırasında gerçekleştirilen eylemlerdir (A). Eylemler, oyuncunun zaman içinde her an gerçekleştirebileceği hareketlerdir. Robotik kol için, eylemler, nesneyi almak ve hareket ettirmek için eklemlerin dönüşünü değiştirmekten ibarettir. Satranç örneğinde ise her bir taşın kurallar dahilinde yapabileceği hamlelerdir. Eylemler de zaman içindeki bir ana bağlıdır (a_t) ve o t anındaki duruma göre seçilirler.

Üçüncüsü, ajana eylemlerinin etkinliği hakkında bilgi veren bir mekanizmaya ihtiyacımız var. Bu mekanizma ödüllerdir (R). Ödül, aracının bir eylemi gerçekleştirdikten sonra aldığı sayısal bir değerdir, bu eylemi gerçekleştirmenin anında etkisini belirler. Robotik kol örneğinde, robot nesneyi hedefine doğru bir şekilde hareket ettirdikçe çevreden olumlu bir ödül almalıdır. Bu ödüller, geri bildirim şeklinde görev hedeflerinin bir temsilidir. Bu nedenle, elde ettiğimiz ödüllerin büyüklüğü hedefe yaklaşmanın bir ölçüsüdür.

Tüm kontrol görevlerinde ortak olan dördüncü unsur ajandır. Ajan, zamanın her anında durumunu gözlemleyerek ve eylemlerde bulunarak göreve katılacak olan varlıktır. Bunlar verilen örneklerde durumu gözleriyle gözlemleyen ve gerekli eylemleri yerine getiren bir insan olabilir. Pekiştirmeli öğrenmede ise eylemleri gerçekleştiren algoritmalar olacaktır.

Son olarak, ajanın yüzde yüz kontrol etmediği veya edemediği görevin tüm yönlerini içeren çevredir. Örneğin, satranç örneğinde kalan süre ajanın tasarrufunda olmadığı için çevrenin bir parçasıdır. Ayrıca, ajan rakibinin hangi hamleleri yapacağını da kontrol edemez ve bilemez, rakiplerin hamleleri de çevrenin bir parçasıdır. Robotik kol durumunda, tüm fiziksel etmenler çevrenin bir parçasıdır. Örneğin, ajan yerçekimi kuvvetini veya nesnelerin sürtünmesini değiştiremez. İlk etmen durum(S) hakkında tekrar düşünüldüğünde, karar verme için çevrenin ilgili yönlerinin bir temsili olarak görülebilir.

2.1.2 Markov karar süreçleri

Matematikte markov karar süreci (MKS) ayrık zamanlı ve stokastik kontrol süreci olarak tanımlanır. MKS, bir ajanın ödüle dayalı optimizasyon kriterlerini en üst düzeye çıkaran eylem dizisini seçtiği sıralı karar verme problemleri için temel algoritmadır [21]. MKS araştırmalarının temellerinin çoğunluğu 1960 yılında R. Howard tarafından atılmıştır [22]. PÖ algoritmaları MKS modelini temel olarak almıştır. MKS dört temel unsurla temsil edilir (S, A, R, P). Durum uzayı (S), eylem uzayı (A), durum ve eylem çiftlerine karşılık ödüller kümesi (R) ve son olarak bir durumdan diğer duruma bir eylem seçerek geçme olasılıkları kümesi (P).

Bir diğer önemli hususta bir durumu (s_t) takip eden durum (s_{t+1}), müteakip durum sadece mevcut duruma bağlıdır ve önceki tüm durumlardan bağımsızdır. 2.1 nolu denklem sürecin hafızası olmadığı, mevcut durumun sadece takip eden durumu etkilediği sonraki durumlardan ve önceki durumlara bir etkisinin olmadığını göstermektedir. Bu özellikleri sağlayan süreçler MKS olarak tanımlanır. MKS'ler Markov zincirlerinin eylemler ve ödüller ile genişletilmiş halidir.

$$P[S_{t+1}|S_t = s_t] = P[S_{t+1}|S_t = s_t, S_{t-1} = s_{t-1}, \dots, S_0 = s_0] \quad (2.1)$$

Mevcutta farklı MKS türleri bulunmaktadır. İlk olarak sonlu ve sonsuz olarak ayrılan MKS'lerde durumlar, eylemler ve ödüller sonlu ise sonlu MKS eğer bunlardan biri veya daha fazlası sonsuz ise sonsuz MKS olarak tanımlanır. Ayrıca, devamsız/epizodik ve devam eden/sürekli Markov karar süreçleri olarak ayrım yapılır. Devamsız süreç belirli koşullar altında sona erer. Satranç örneğinde, oyuncu veya rakibi şah mat ettiğinde görev sona erer. Tersine, devam eden bir Markov karar sürecinin bir sonlandırma koşulu yoktur. Örnek olarak herhangi bir bitirme koşulu olmayan sıcaklık kontrol süreçleri verilebilir.

Bu çalışma robot kolun ucunun istenilen noktaya gelmesiyle görev nihayete ulaşacağı için devamsız/epizodik, eylemler olan robot kolun aktivatörlerine uygulanan gerilimin ve durumları olan manipülatör ucunun konumunun sonsuz olmasından ötürü süreç sonsuz olacaktır.

2.1.3 Gezinge ve bölüm

Gezinge ajan bir durumdan diğerine geçtiğinde oluşan izdir. Yunan harfi τ ile gösterilen yörünge, bu süreç boyunca üretilen öğelerin dizisidir. Yörüngenin başlangıcında ajan s_0 durumundadır. Daha sonra bu duruma göre a_0 eylemini gerçekleştirir ve sonuç olarak ilk ödülü r_1 'i alır. Daha sonra s_1 durumunda a_1 eylemini yapar ve bu zincir gezingenin son durumuna ulaşana kadar devam eder. Öte yandan, bir bölüm sadece ilk durum 0 anında başlayan ve son T durumunda biten bir yörüngedir. Bir bölüm denklemi 2.2'te verilmiştir.

$$\tau = s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T,\ a_T, r_T \tag{2.2}$$

2.1.4 Getiri ve ödül

Kontrol görevinin amaçları, çevrenin ajanın gerçekleştirdiği eylemlere yanıt olarak aracıya verdiği ödüllerle (r_t) temsil edilir. Görevin en iyi şekilde çözülmesi için bu ödüllerin toplamının mümkün olan en üst düzeye çıkarılması gerekmektedir. Bununla birlikte, kısa vadede bir ödül elde etmek için harekete geçmenin uzun vadede elde ettiğimiz ödülleri daha da kötüleştirebileceği durumlar olabilir. Örneğin satrançta bir düşman taşını ele geçirmek önemli bir konumu kaybetmemize ve rakibin bizi yenmesini kolaylaştırabilir. Bu nedenle, elde etmek istenen şey uzun vadeli ödüllerin toplamının maksimize edilmesi olarak nitelenir. Ödül, eylemlerimizin ürettiği anlık sonuçtur. Öte yandan getiri, ajanın belirli bir zaman noktası t anından, görevin tamamlandığı T anına kadar elde ettiği ödüllerin toplamıdır. Yani getiri, bölüm boyunca yaptığımız eylemlerin uzun vadeli sonucudur. Uzun vadeli ödüllerin toplamının maksimize edilmesi istendiği için, beklenen getirinin maksimize edilmesinin istendiği de söylenilebilir. Denklem 2.3'te getirinin matematiksel formülü verilmiştir.

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \tag{2.3}$$

2.1.5 Azaltma faktörü γ

Pekiştirmeli öğrenmede kullanılan modelden bağımsız yaklaşım ile ajan ortamı ve ortama bağlı olarak alacağı ödülü tam olarak bilmediği için gelecek ve şimdi arasında bir bağlantı kurması gerekmektedir. γ parametresi gelecek ve şimdiki ödüller arasındaki dengeyi sağlamaktadır. Bu dengeyi sağlamak için getiri hesaplanırken

gelecekteki ödülleri γ kuvvetleri ile çarpılır. $\gamma \in [0,1]$, olduğundan gelecekteki ödüllerin değerleri kaçıncı adımda olduklarına bağlı olarak gittikçe küçülür. Bir diğer deyişle belirli bir ödülü elde etmek ne kadar uzun sürerse değeri o kadar çok düşer. Bu hamle ile ajan hedefe en verimli şekilde ulaşmaya teşvik edilir. γ azaltma faktörü ile düzenlenmiş getiri 2.4 numaralı denklemde verilmiştir. Getiri formülüne bu yapılan ekleme ile hedef, uzun vadeli azaltılmış getiriyi maksimize etmeye çalışmak olarak güncellenir.

$$G_0 = r_1 + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots + \gamma^{T-t-1} r_T \tag{2.4}$$

2.1.6 Politika $\pi(s)$

Pekiştirmeli öğrenmenin temel kavramlarından biri olan politika (π), girdi olarak bir durum (s) alan ve o durumda yapılacak eylemi (a) döndüren bir fonksiyon olarak tanımlanır. Politika deterministik veya stokastik olabilir. Deterministik politika $\pi(s)$ ile temsil edilip çıktı olarak bir eylem verirken, stokastik politika $\pi(a|s)$ şeklinde gösterilip, çıktı olarak s durumunda a eyleminin gerçekleşme olasılığını verir. Pekiştirmeli öğrenme algoritmaları temelde en uygun politikayı bulmaya çalışır. Deterministik politikanın matematiksel gösterimi denklem 2.5'te stokastik politikanın gösterimi ise denklem 2.6' verilmiştir.

$$\pi(s) = a \tag{2.5}$$

$$\pi(s) = [p(a_1), p(a_2), \dots, p(a_n),] \tag{2.6}$$

2.1.7 Durum değeri $v_\pi(s)$ ve aksiyon değeri $v_\pi(a|s)$

Bir durumun değeri, o durumdan başlayarak ve π politikasını izleyerek bölüm sonuna kadar çevre ile etkileşime girerek elde etmesi beklenen getiri olarak tanımlanır. Farklı politikalar aynı durum için farklı eylemler seçip farklı getiriler elde edeceği için, durumun değeri izlenen politikaya bağlıdır. Durum değer fonksiyonu matematiksel gösterimi ve getiri denklemi 2.5 ile genişletilmiş hali 2.7'de gösterilmiştir.

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

$$v_\pi(s) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots + \gamma^{T-t-1} r_T \,|S_t = s\right] \tag{2.7}$$

Öte yandan, belirli bir durumdaki bir eylem de q-değeri aracılığıyla değerlendirilir. Bir durumdaki bir eylemin q-değeri, s durumunda başlanarak ve a aksiyonunu

gerçekleştirdikten sonra bölümün sonuna kadar π politikasını izleyerek çevre ile etkileşime girmesiyle elde etmesi beklenen getiridir. Q-değerleri birçok algoritmada optimal politikayı bulmayı basitleştirdiklerinden kapsamlı bir şekilde kullanılmaktadır. Q-değeri fonksiyonu matematiksel gösterimi ve getiri denklemi 2.5 ile genişletilmiş hali 2.8 numaralı denklemde verilmiştir.

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a]$$

$$q_\pi(s,a) = \mathbb{E}\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots + \gamma^{T-t-1} r_T \,|S_t = s, A_t = a\big] \quad (2.8)$$

2.1.8 Bellman denklemleri ve MKS'lerin çözümü

Bellman eşitliği, Richard Bellman tarafından geliştirilen MKS problemlerinin çözümünde hesaplama yükünü azaltan optimizasyon yöntemidir[23]. Pekiştirmeli öğrenmede en iyi politikayı bulmak bir optimizasyon problemi olarak tanımlanabilir. Bu optimizasyon problemini bellman eşitliği ile çözüm getirilir.

Durum değeri denklemi 2.7'de getirinin ikinci teriminden itibaren γ parantezine alınarak ardıl adım için getiri formülü parantez içinde elde edilir. Bir durumun değeri beklenen getiridir. Bu matematiksel beklenti, o politikayı takip eden her bir eylemi gerçekleştirme olasılığı ile o eylemi yapmaktan elde edilmesi beklenen getirinin çarpımı olarak yazılır. Ve getiri, her olası halef duruma ulaşma olasılığı ile o duruma ulaşıldığında elde edilen ödül ve o durumun indirgenmiş değerinin çarpımı olarak ifade edilir. Böylece bir durumun değeri ile diğer durumların değerleri arasında tekrarlı bir ilişki denklem 2.9'daki gibi kurulur.

$$v_\pi(s) = \mathbb{E}\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots + \gamma^{T-t-1} r_T \,|S_t = s\big] \quad (2.7)$$

$$= \mathbb{E}\big[r_{t+1} + \gamma(r_{t+2} + \gamma^1 r_{t+3} + \gamma^2 r_{t+4} \dots + \gamma^{T-t-2} r_T)|S_t = s\big]$$

$$= \mathbb{E}[r_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (2.9)$$

Burada s' mevcut durum s'yi takip eden halef durumu ifade etmektedir. Q-değer fonksiyonu içinde aynı işlemler yapıldığında, bir q-değerini diğer q-değerleri cinsinden ifade eden 2.10 denklemi elde edilir.

$$q_\pi(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \dots + \gamma^{T-t-1} r_T \,|S_t = s, A_t = a\right] \quad (2.8)$$

$$= \mathbb{E}\left[R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} \dots + \gamma^{T-t-2} R_T)|S_t = s, A_t = a\right]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$$

$$= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right] \quad (2.10)$$

Burada a' halef durum s' da gerçekleştirilen eylemi temsil etmektedir. Özyinelemeli olarak elde edilen denklemler, bellman optimallik eşitliği olarak 2.11 ve 2.12 deki gibi yazılır. 2.11 numaralı denklemde durum değer fonksiyonu, 2.12 numaralı denklemde ise q-değer fonksiyonu verilmiştir. Burada * sembolü optimalliği anlatmaktadır.

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \quad (2.11)$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \quad (2.12)$$

Bir durumun değeri, bu durumdan itibaren beklenen getiridir. Dolayısıyla bir kontrol görevini çözmek, her durumun değerini maksimize etmeyi veya her q-değerini maksimize etmeyi içerir. Bu getirileri maksimize etmek için, tüm durumlarda optimal eylemleri gerçekleştirecek optimal politika π_*'ı bulmak gerekir. Optimal politika denklemleri q-değeri için 2.13 numaralı denklemde, durum değerleri için 2.14 numaralı denklem ile verilmiştir. Optimal politika tam olarak her durumda maksimum beklenen getiriye yol açan eylemi seçen politika olarak tanımlanır. Durum değerinin kullanılması, eylemin yol açtığı durumları ve bu ardıl durumda elde etmeyi beklediğimiz getiriyi hesaba katacaktır. Q-değerlerini kullanmak ise basitçe q-değeri en yüksek olan eylemi seçecektir.

$$\pi_*(s) = arg\max_a q_*(s,a) \quad (2.13)$$

$$\pi_*(s) = arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s)] \quad (2.14)$$

Optimum politikayı bulmak için optimal değerlere, optimum değerleri bulmak için optimal politikanın bilinmesi gerekir. Bu tezde MKS değer-tabanlı çözüm algoritmalarının iteratif olarak çözüldüğü algoritmalar kullanılmıştır.

2.2 Zamansal Fark Metodu ve Q-Öğrenme

Bu tezde kullanacağımız pekiştirmeli öğrenme algoritmalarının temelini oluşturduğu için Q-Öğrenme algoritmaları ve temel kavramları bu bölümde açıklanmıştır.

2.2.1 Zamansal fark metodu

Zamansal fark metodu birçok pekiştirmeli öğrenme algoritmanın temelini oluşturur. Bu metot deneyimlere dayalı olarak optimal değerleri öğrenen bir yöntemler ailesinin temelidir[4]. Bu yöntemde, ajan, ziyaret edilen durumlar, gerçekleştirilen eylemler ve elde edilen ödüller ile bir gezinge oluşturarak çevreyi deneyimler. Bu deneyimler ile 2.15'te denklemi verilen zamansal fark hatası hesaplanır, bu hata değer tahminlerini ve politikayı güncellemek için kullanılır.

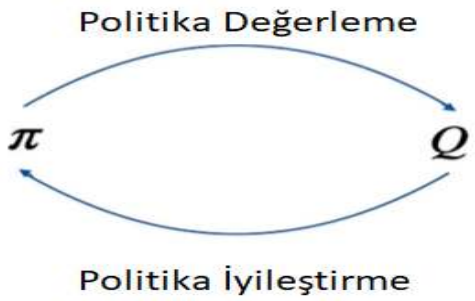
$$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{2.15}$$

Bu yöntem modelden bağımsız bir yöntemdir. Çevre dinamiğine ilişkin bir model bulunmaz, bu sebeple politikaya yön vermek için çevre dinamiklerine ihtiyaç duyan durum değerleri yerine politikayı yönlendirmek için q-değerleri kullanılır. Bu şekilde, politika basitçe en yüksek Q değerine sahip eylemi seçer.

Bu metotta q-değerlerini güncellemek için önyükleme yöntemi kullanılır. Önyüklemenin matematiksel ifadesi denklem 2.16'te verilmiştir. Burada α öğrenme katsayısı olarak adlandırılan 1 ve 0 arasında bulunan bir orandır. Zamansal farkın ne kadar güncelleme işlemine etki edeceğini belirler.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\,[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.16}$$

Zamansal fark metodu, politika değerleme ve politika iyileştirmenin birbirini takip ettiği, Şekil 2.4'te gösterimi verilen, genelleştirilmiş politika iterasyonu şablonunu kullanır. Bu şablon ile değerleme ve iyileştirme süreçleri birbirlerini optimal q-değerlerine ve optimal politikaya doğru yönlendirir. Bu şablon ajanın her adımından sonra tekrarlanır ve q-değerleri bölümün bitmesini beklemeden sürekli güncellenir. Bölüm başındaki öğrenme, bölümün geri kalanında politikayı etkileyerek karar verme sürecini iyileştirir. Öğrenme sürecinin daha stabil ve eş dağılımlı olmasını sağlanır. [4]



Şekil 2.4 : Genelleştirilmiş politika iterasyonu şablon gösterimi.

2.2.2 Q-Öğrenme

Chris Watkins tarafından 1989'da sunulan Q-Öğrenme [24], 1992 yılında Watkins ve Dayan [25] tarafından kanıtlanmış, modelden bağımsız bir pekiştirmeli öğrenme algoritmasıdır. Q-öğrenme her durum-eylem çiftine verilen q-değerlerini bir tabloda saklar ve bu tabloyu elde edilen deneyimlerle bellman denklemi aracılığıyla zamansal fark metodunu kullanarak güncelleyerek algoritmayı modelden bağımsız hale getirir. Q değerlerini bir tabloda sakladığı için literatürde tablo tabanlı metot olarak adlandırılan sınıfta yer alır. Öte yandan, politika-dışı stratejisini izleyen bir algoritmadır. Politika-dışı stratejisinde biri optimizasyon sürecine katılmak ve diğeri çevreyi keşfetmek için olmak üzere iki ayrı politika kullanır. Optimizasyon sürecine katılan hedef politika $\pi(s) = \arg\max_a Q(s,a)$, her zaman en yüksek q değerine sahip eylemi seçen açgözlü olarak nitelendirilen politikadır. Diğer yandan, keşif politikası $b(s)$ çevreyi keşfetmek ve uygun örnekleri toplamak için kullanılır. Zamansal fark hatası ve önyükleme yöntemi ile oluşturulan Q-öğrenme algoritması güncelleme kuralı denklem 2.17'da verilmiştir.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q\big(S_{t+1}, \pi(S_{t+1})\big) - Q(S_t, A_t)\right] \quad (2.17)$$

Burada α katsayısı öğrenme katsayısı olarak adlandırılır. Ardıl durum S_{t+1} da ardıl eylem A_{t+1}'i, takip edilen π politikası belirlediğinden $\pi(S_{t+1})$ olarak belirtilmiştir.

Bu bilgiler ışığında Q-Öğrenme algoritmasının sözde kodu çizelge 2.1'deki gibidir. Burada ilk olarak α ve γ katsayıları [0, 1] aralığından uygun olarak tanımlanır. Ardından durum ve eylem çiftlerine bağlı olarak q-değerlerini içeren tabloya sıfır değerleri atanır. Bu altyapı hazırlandıktan sonra eğitim sürecinin kaç bölümden oluşacağı N ve her bölümün kaç adımdan oluşacağı T değerlerine göre iç içe döngüler

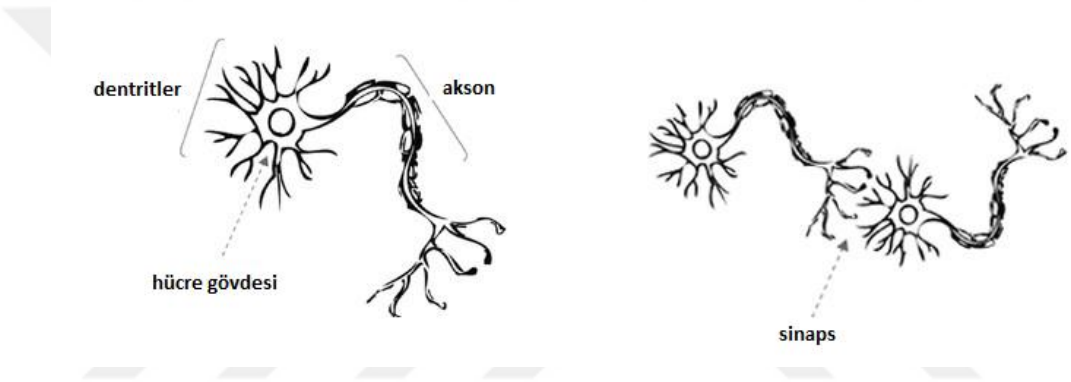
oluşturulur. Bölüm başında çevre sıfırlanır ve başlangıç pozisyonu s_0 gözlemlenir. Keşif algoritmasıyla bu pozisyonda bir eylem seçilir bu eyleme sonucunda yeni durum s_{t+1} ve elde edilen ödül r_{t+1} gözlemlenir. Bu değerler ile 2.16 denklemi aracılığıyla durumun yeni q-değeri belirlenir. Her adımda bu işlem bölüm sonuna kadar tekrar edilir. Belirtilen bölüm sayısı kadar eğitim gerçekleştiğinde tahmini en optimal politika π ve tahmini en doğru q-değerleri elde edilmiş olur.

Çizelge 2.1 : Q-Öğrenme algoritması sözde kodu.

	Q-Öğrenme Algoritması
1	α öğrenme katsayısı, γ azaltma faktörünü belirle
2	Q(s,a) = 0, s ∈ S, a ∈ A, Q değeri başlangıç değerlerine sıfır ata
3	n←1, **için döngüye gir**
4	Çevreyi sıfırla, başlangıç pozisyonunu gözlemle $s \leftarrow s_0$,
5	t←0, **için döngüye gir**
6	$a_t \sim b(s_t)$, keşif algoritmasıyla bir aksiyon seç
7	a_t *eylemini gerçekleştir ve* s_{t+1}, r_{t+1}'i gözlemle
8	$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q\left(s_{t+1}, \pi(s_{t+1})\right) - Q(s_t, a_t)\right]$ Denklemine göre q-değerlerini güncelle.
9	s ⟵ s_{t+1} , t ← t+1,
10	t = T **ise döngüyü bitir**. T bir bölümde atılacak adım sayısı.
11	n←n+1
12	n=N **ise döngüyü bitir**. N, algoritmanın kaç bölüm çalışacağı
13	Çıktı: Tahmini en optimal politika π ve Q değerlerini elde et.

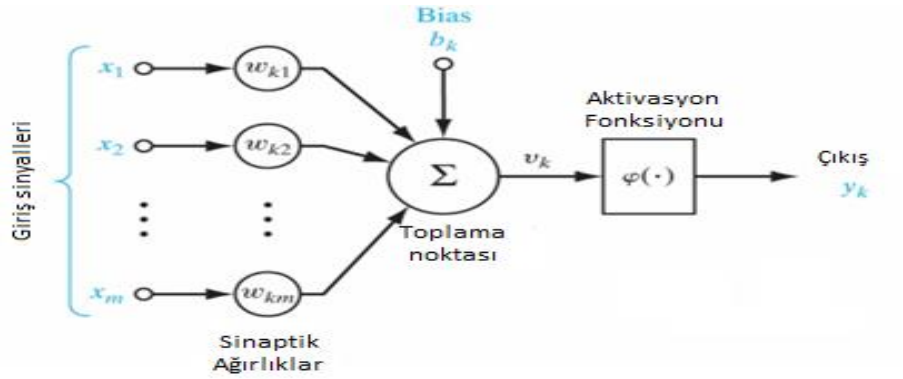
3. YAPAY SİNİR AĞLARI

İlk yapay sinir ağı modeli 1943 yılında Warren McCulloch ve Walter Pitts tarafından elektrik devreleriyle basit sinir ağı modellenmiştir[26]. Genel olarak insan merkezi sinir sistemini taklit eden bir sistem olup, insan beyninin özelliklerinden olan öğrenme, hatırlama gibi yollar ile elde ettiği verilerden, yeni bilgiler türetme, genelleme yapabilme, keşfedebilme gibi yetenekleri herhangi bir yardım almadan otomatik olarak gerçekleştirebilmek için geliştirilen bilgisayar yazılımlarıdır. İlk olarak biyolojik üniteler olan nöronların modellenmesi ve bilgisayar sistemlerinde uygulanması ile başlamış, bilgisayar teknolojisinin gelişmesi hesaplama güçlerinin yükselmesi ile kullanımı yaygınlaşmıştır.



Şekil 3.1 : Biyolojik nöronlar.

Basitleştirildiğinde biyolojik sinir hücreleri dentrit uçlarından elde edilen sinyal, merkezde ağırlandırılarak gövde boyunca iletilir ve sinapslarda çıkış sinyali üretilir. Yapay nöronlar modellenirken gerçek nöronların karmaşıklığı oldukça soyutlanır bir yapay nörona ait gösterim Şekil 3.2'de verilmiştir. Temel olarak girdi sinyali, sinyali çarpan bir ağırlık, aktivasyon fonksiyonu ve çıkış sinyalinden oluşur. Girdiler nörona gelen verilerdir. Gelen sinyal ağırlıklar(w) ile çarpılıp eşik değeri (b) eklendikten sonra iletir, bu sayede girdilerin çıktı üzerindeki etkisi ayarlanabilmektedir. Aktivasyon fonksiyonlarının (φ) ise çeşitleri vardır, temel amacı lineerliği bozmak olsa da bazı aktivasyon fonksiyonları eşik görevi de görürler, üretilen sinyal belli değerin altında kaldığı müddetçe sabit bir sinyal göndermek gibi. Bir nöron matematiksel olarak denklem 3.1 ve 3.2'deki gibi tanımlanabilir.

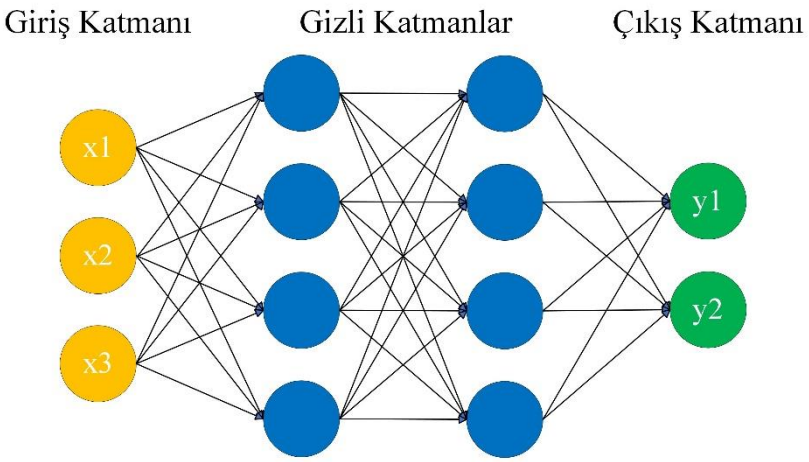


Şekil 3.2 : Yapay nöronun gösterimi.

$$u_k = \sum_{i=1}^{m} w_{ki} x_i \tag{3.1}$$

$$y_k = \varphi(u_k + b_k) \tag{3.2}$$

Bu yapay nöronlar bir araya gelerek katmanları, bu katmanlarda bir araya gelerek yapay sinir ağını oluşturur. Giriş katmanı, gizli katmanlar ve çıkış katmanı temel katmanlardır. Örnek bir YSA mimari Şekil 3.3'de gösterilmiştir. Giriş katmanı verilerin ilk durağıdır. Bu veriler pikseller, çevreden gelen sinyaller gibi makine öğrenmesi verisi olabilir. Giriş ve çıkış katmanları arasındaki tüm katmanlar gizli katman olarak adlandırılır ve nihai sinyal üretilir. Gizli katman sayısının birden fazla kullanılması durumuna ise derin yapay sinir ağı denir.



Şekil 3.3 : Örnek yapay sinir ağı mimarisinin gösterimi.

Yapay bir nöronun ağırlığı ne kadar yüksekse, onunla çarpılan girdi o kadar güçlü olur. Ağırlıklar negatif de olabilir, böyle bir durumda sinyalin negatif ağırlık tarafından engellendiği söylenilebilir. Yapay bir nöronun ağırlıkları ayarlanarak belirli girdiler

için istenilen çıktı elde edilebilir. Ancak yüzlerce veya binlerce nörondan oluşan bir YSA olduğunda, gerekli tüm ağırlıkları elle bulmak oldukça karmaşık olur. Bu sebeple ağdan istenilen çıktıyı elde etmek için YSA'nın ağırlıklarını ayarlayabilen algoritmalar mevcuttur [27]. Ağırlıkları ayarlama işlemine öğrenme veya eğitim denir. YSA türlerinin sayısı ve kullanımları oldukça fazladır. McCulloch ve Pitts'in (1943) ilk sinirsel modelinden bu yana YSA olarak kabul edilen yüzlerce farklı model geliştirilmiştir.

3.1 Stokastik Gradyan İnişi

Stokastik gradyan inişi, uygun özellikleriyle (örneğin, türevlenebilir veya alt türevlenebilir) bir amaç fonksiyonunu optimize etmek için yinelemeli bir yöntemdir. Özellikle yüksek boyutlu optimizasyon problemlerinde çok yüksek hesaplama yükünü azaltır ve daha düşük bir yakınsama oranı için daha hızlı yinelemeler sağlar.[28] Yapay sinir ağları da türevlenebilir veya alt türevlenebilir bir fonksiyon kabul edilerek parametre güncellemeleri stokastik gradyan iniş algoritmasıyla yapılır. Bu bölümde stokastik gradyan algoritması ve yapay sinir ağları parametre güncellemesinin kısa bir özeti verilmiştir.

Bu algoritmada başlangıç olarak, çevreden elde edilen ödüller ve sinir ağı tahminleri kullanılarak beklenen kayıp fonksiyonu $\hat{L}(w)$ bulunur. Sırasıyla takip edenler uygulanır. İlk olarak, çevreden elde edilen ödüller ve sinir ağı tahminleri kullanılarak kayıp fonksiyonunun değeri tahmin edilir. Bu fonksiyonun tahminleri ile gradyan vektörü bulunur. Gradyan vektörü, her elemanı sinir ağının her bir parametresine göre tahmini kayıp fonksiyonunun kısmi türevi olduğu bir vektördür. Denklem 3.3'te gradyan vektörünün matematiksel gösterimi verilmiştir.

$$\nabla \hat{L} = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n}\right] \tag{3.3}$$

Gradyan, sinir ağının her bir parametresinin değiştirilmesi gerektiği yönü gösteren bir vektördür. Bu yön kayıp fonksiyonunun mümkün olduğu kadar büyümesini sağlayan yöndür. Gradyan vektörü, geri yayılım adı verilen bir algoritma ile hesaplanır[29]. Gradyan vektörü hesaplandıktan sonra güncelleme kuralı denklem 3.4'teki gibidir. Bu güncelleme kuralında, yeni parametreler elde etmek için önceki değerlerden, kayıp fonksiyonunun gradyanı α ile çarpılarak çıkarılır. α, burada adım boyutu olarak

adlandırılır ve ilerlemenin boyutunu belirler. Böylece parametreler kayıp fonksiyonunun maksimum büyüme yönünün tersi yönünde belirli bir oranda hareket ettirilmiş olur. Yani, maksimum büyümenin tahmini yönünün tersi yönünde gidildiği için, kayıp fonksiyonu için en düşük değerleri üretecek parametreler bulunmuş olur.

$$w_{t+1} = w_t - \alpha \nabla \hat{L}(w) \tag{3.4}$$

3.2 Yapay Sinir Ağlarının Optimizasyonu

Bu başlık altında, optimal q-değerlerine yaklaşmak için sinir ağlarının optimizasyonunu incelenmiştir. Ulaşılmak istenen şey, yapay sinir ağının girdi katmanına bir durum girdi olarak iletildiğinde, sinir ağının o durumdaki her eylem için çıktı olarak doğru q-değerleri tahminleri üretmesidir. Bunu başarmak için nöronlar arasındaki bağlantıların gücünü belirleyen sinir ağının w^* parametreleri için en uygun değerlerin bulunması gerekir. w^* için optimal değerler, sinir ağının $\hat{q}(s, a|w)$ değerlerini tahmin ederken yaptığı hataları en aza indiren değerlerdir. Bu hataları bulmak için literatürde ortalama mutlak hata, ortalama karesel hata gibi birçok yöntem bulunmaktadır. Seçilen hata hesaplama yöntemine bağlı olarak w^* parametrelerinin hesaplanması için farklı çözümler elde edilir. Her çözüm farklı karakteristikler barındırır.[27]

Bu bölümde ortalama karesel hatanın en aza indirilmesi incelenmiştir. Bu hesaplama çevreden örnek olarak alınan deneyimlere dayanılarak yapılır. Ortalama karesel hata 3.5 numaralı denklem ile hesaplanır. Belirli bir durum ve eylem için hedef değer y, o eylemin o durumdaki gerçek değerini düşündüğümüz ifadedir. Eylemi gerçekleştirdikten sonra elde edilen ödülün yanı sıra ulaşılan müteakip durumda yapılan bir sonraki eylemin q-değerinin azaltılmış tahmininden oluşur ve $y = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}|w)$ ifadesiyle temsil edilir. Buradaki tahmin yapay sinir ağı tarafından üretilmiştir. Bir diğer terim ise sinir ağının o durum için ürettiği kestirilen değer $\hat{y}$ 'dir ve $\hat{y} = \hat{q}(S_t, A_t|w)$ ifadesi ile temsil edilir.

$$\hat{L}(w) = \frac{1}{N}\sum_{i=0}^{N}[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}|w) - \hat{q}(s_t, a_t|w)]^2 \tag{3.5}$$

4. DERİN PEKİŞTİRMELİ ÖĞRENME

Durum-eylem çiftlerinin sayısı arttığında politika parametrelerini tabloda ifade etmek zorlaşmaktadır. Bu nedenle karmaşık ve yüksek ölçekli problemlerin çözümünde Q-tabloların kullanılması uygun değildir. Derin pekiştirmeli öğrenme (DPÖ) derin sinir ağları ve pekiştirmeli öğrenmenin birlikte kullanımı ile oluşturulmuştur.[30]

PÖ algoritmalarına YSA'ları dahil etmek, tablo tabanlı algoritmalardan sinir ağı tabanlı yöntemlere geçişi sağlar. Burada YSA'lar değerlerin tahmin edilmesine izin veren birer araçtır. Ayrıca YSA kullanmak ayrık durum uzayından, sürekli durum uzayına geçilmesinde kolaylık sağlar. Her ne kadar sürekli durum ve eylem uzaylarında tablo tabanlı metotlarla çalışmak için yöntemler geliştirilmişse de bu yöntemler değer tahminlerinin sınırlı olması, artan boyutlarda hesaplama karmaşıklığının çok hızlı büyümesi ve elde edilen bu değerlerin saklanmasında yüksek hafıza kapasitesi istemesi gibi dezavantajları vardır[4]. Tablo tabanlı metotlarda her bir durum, her bir eylem ve her bir durum-eylem çiftine bağlı olarak durum veya q-değerleri hafızada tutulur. Yapay sinir ağları tabanlı metotlarda ise durum ve eylemi girdi olarak alan ve q-değerlerini çıktı olarak veren bir fonksiyonun parametrelerini hafızada tutmak yeterlidir. Öğrenme süreci sırasında, bu parametreler gerçek değer fonksiyonuna veya q-değer fonksiyonuna uyması için optimize edilir. YSA tabanlı metotlarda öğrenme sürecinde de en uygun q-değer fonksiyonu ve politikası bulunana kadar genelleştirilmiş politika iterasyonu izlenir. Fakat bu kez durum veya q-değerleri değil fonksiyon parametreleri güncellenir. Öğrenme sürecinin başlangıcında, optimize edilecek olan fonksiyon herhangi bir şekle sahip olabilir ve her durum için optimal değerleri yetersiz bir şekilde tahmin edebilir. Öğrenme süreci ilerledikçe bu fonksiyon optimal değer fonksiyonuna yaklaşır ve sürecin sonunda optimal değer fonksiyonuna mümkün olduğunca yakın olacaktır.

Bu bölümde bu çalışmada kullanılan DPÖ algoritmalarına geçmeden önce derin pekiştirmeli öğrenme algoritmalarında kullanacağımız temel kavramlar ve derin Q-öğrenme algoritması incelenmiştir.

4.1 Derin Q-Öğrenme

Q-öğrenmenin temel zayıflığı, genelleştirme eksikliğidir. Q-öğrenme, iki boyutlu bir eylem ve durum tablosundaki sayıları günceller. Q-öğrenme ajanlarının eğitim

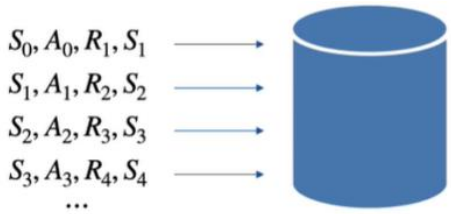
sürecinde tecrübe edilmemiş durumların değerini tahmin etme yeteneği yoktur. Bu sorunla başa çıkmak için derin YSA'ları durum ve eylem değerlerini belirleyen bir fonksiyon yaklaşımcısı olarak kullanan Deep Q-Öğrenme tanıtılmış ve 2013 yılında Mnih V., ve arkadaşları bir makale yayınlayarak DPÖ algoritmalarının kullanabilirliğini ispatlamıştır. [31]

Derin Q-öğrenme, YSA ile zamansal fark algoritması kullanan Q-öğrenme algoritmasının birleşimidir. Burada bir t anındaki durumu girdi olarak alarak beklenen eylemin q-değerini hesaplayacak bir sinir ağı bulunur. Q-öğrenme algoritmasının bir uzantısı olduğundan, politika-dışı bir öğrenme stratejisi izler. Tahmini q değerlerine göre epsilon açgözlü olacak bir keşif politikası kullanarak çevre keşfedilir ve sinir ağını güncellemek için, tahmini q değerlerine göre açgözlü olacak ayrı bir politika kullanılır. Bu nedenle, kayıp fonksiyonundaki hedef değeri, keşfedilen sonraki eyleme göre değil, optimize edilecek politika olan hedef politika tarafından seçilene göre hesaplanır. Algoritmayı biraz daha basit bir hale getirmek için hedef politika açık bir fonksiyon yerine maximum değerli eylemi seçen basit bir fonksiyon olarak yazılır. Böylece müteakip eylemin q-değeri gerektiğinde, hedef politikanın seçeceği maksimum q-değerine sahip eylem hesaplamalarda kullanılır.

Öte yandan, bu algoritmada, ajan tarafından gözlemlenen deneyimleri depolayacak bir tekrar belleği kullanılır. Bu bellek deneyimin rastgele seçilmiş yığınlarına dayanarak, sinir ağının parametrelerinin güncellenmesinde kullanılır. Son olarak hedef değerleri hesaplamak için sinir ağının bir kopyasını yaparız. Bu sinir ağı, stokastik gradyan inişiyle değişmez. Parametreleri θ aynı kalır ve parametre güncellemelerine kararlılık kazandırırken q-değerlerinin de doğru tahminlerini çok daha erken elde edilmesini sağlar.

4.1.1 Tekrar belleği

Tekrar belleği (B), ajanın eğitim esnasında gözlemlediği durum geçişlerinin depolandığı bir veri tabanıdır. Ajan her bir eylemi gerçekleştirdiğinde, ajanın içinde bulunduğu durumu s, ajanın gerçekleştirdiği eylemi a, bu eylemin sonucunda elde ettiği ödülü r ve ajanın ulaştığı müteakip durum s' bu belleğe kaydedilir.[32]



Şekil 4.1 : Tekrar belleği temsili gösterim.

Bellekte saklanabilecek maksimum sayıda giriş sayısı önceden belirlenir ve bu sayı aşıldığında, en eski girişler silinerek yerine ajanın gözlemlediği en son geçişler kaydedilir. Ardından, sinir ağını güncellemek için, bellekte depolanan tüm durum geçişlerinden belirlenen miktarda rastgele bir yığın $K = (S, A, R, S') \sim B$ seçilir. Bu yığın ile kayıp fonksiyonu denklem 4.1 hesaplanır ve kayıp fonksiyonu tahminine dayanarak sinir ağı güncellenir.

$$L(\theta) = \frac{1}{K}\sum_{i=1}^{|K|}\left[R_i + \gamma \hat{q}\left(S_i', A_i' | \theta_{hedef}\right) - \hat{q}(S_i, A_i|\theta)\right]^2 \qquad (4.1)$$

4.1.2 Hedef ağ

Derin Q-öğrenme algoritmasını kararsız bir hale getirecek iki teknik bir arada kullanılmaktadır. Bunlardan ilki bir tahmini değeri referans alarak bir başka tahmini değerin güncellendiği önyükleme işlemi, diğeriyse fonksiyon yaklaşımcısı olarak kullanılan bir yapay sinir ağı. Bir fonksiyon yaklaşımcısı olarak kullanılan yapay sinir ağı bir durumun tahminini referans alınarak gradyan inişine göre güncellendiğinde, sadece o durum için değil, ona yakın tüm durumlar için tahminde değişiklik yapılmış olur. Ajan bir eylem gerçekleştirdiğinde, sonraki durum genellikle bir öncekine çok benzer. Bunun anlamı, optimize edilmek istenen tahmini q-değeri $\hat{q}(S_i, A_i|\theta)$ güncellendiğinde, tahminin takip etmesi istenen değer olan hedef değer $\hat{q}\left(S_i', A_i'|\theta_{hedef}\right)$ de değişir. Q-değer tahminleri hareketli bir hedefe yaklaşacağından, bu süreç kararsız olacaktır. Öğrenme sürecinin istikrarlı olması için, tahminlerin hareket etmesi gereken doğru değerleri temsil ettikleri hedef değerlerin de kararlı olması gerekir. Bu kararlılığı sağlamak için, q-değerlerini tahmin eden sinir ağının tam bir kopyası oluşturulur ve bu kopya sadece hedef değerleri tahmin etmek için kullanılır. Bu sinir ağı, hedef ağ olarak bilinir. Tahminlerin hatasını en aza indirmek

için gradyan iniş uygulandığında, bu hedef ağ değiştirilmez, parametreleri aynı kalarak öğrenme süreci boyunca hedef tahminlerinin sabit kalacağı anlamına gelir.

Bu nedenle kayıp fonksiyon ifadesi 4.1'de sinir ağı parametreleri olarak θ_{hedef} olarak belirtilmiştir. Çünkü bu tahminler hedef ağ kullanılarak yapılmaktadır. Daha önceden belirlenmiş k sayıda bölümden sonra, ajanın temel sinir ağındaki parametrelerle hedef ağın parametreleri senkronize edilir. Bunun sonucunda hedef değer tahminleri de öğrenme süreci boyunca daha doğru hale gelir. Derin Q-Öğrenme sözde kodu çizelge 4.1'deki gibidir.

Çizelge 4.1 : Derin Q-öğrenme algoritması sözde kodu.

	Derin Q-Öğrenme Algoritması
1	α öğrenme katsayısı, γ azaltma faktörü ve ε rastgelelik katsayısını belirt
2	θ parametreleriyle ağı oluştur. Hedef ağı kopyala $\theta_{heedef} \leftarrow \theta$
3	$b \leftarrow \hat{q}(s, a\|\theta)$'lara ilişkin olarak epsilon obur politikayı tanımla
4	$B \leftarrow$ tekrar belleğini başlat
5	n←1, **için döngüye gir**
6	Çevreyi sıfırla, başlangıç pozisyonunu gözlemle $s \leftarrow s_0$,
7	t←0, **için döngüye gir**
8	$a_t \sim b(s_t)$, keşif algoritmasıyla bir aksiyon seç
9	a_t eylemini gerçekleştir ve s_{t+1}, r_{t+1}'i gözlemle
10	$(s_t, a_t, r_{t+1}, s_{t+1})$ geçişini tekrar belleğine kaydet
11	$K = (S, A, R, S')$ yığınını bellekten seç
12	$L(\theta) = \frac{1}{K}\sum_{i=1}^{\|K\|}\left[R_i + \gamma \max_a \hat{q}\left(S_i', a\|\theta_{hedef}\right) - \hat{q}(S_i, A_i\|\theta)\right]^2$ K yığını ile kayıp fonksiyonu değerini hesapla
13	Sinir ağı parametrelerini stokastik gradyan inişi ile güncelle $\theta = \theta - \alpha\nabla L(\theta)$
14	s ⟵ S_{t+1} , t ← t+1,
15	t = T **ise döngüyü bitir**. T bir bölümde atılacak adım sayısı.
16	Her k bölüm için hedef ağı parametrelerini eşitle, $\theta_{hedef} \leftarrow \theta$
17	n←n+1
18	n=N **ise döngüyü bitir**. N, algoritmanın kaç bölüm çalışacağı
19	Çıktı: Tahmini en optimal politika π ve $\hat{q}(s, a\|\theta)$ tahmin değerlerini elde et.

Bu algoritmada ilk olarak rastgele θ parametreleriyle bir YSA ve bunun kopyalanmasıyla hedef ağ oluşturulur. Hemen ardından tekrar belleği başlatılır ve ajanın keşif politikasıyla seçtiği aksiyonlar ile (S, A, R, S') geçişleri belleğe kaydedilir. Bu bellekten rastgele seçilen yığın ile kayıp fonksiyonu hesaplanır ve sinir ağı parametreleri stokastik gradyan inişi ile güncellenir. Belirli sayıda bölüm tekrarında

YSA parametreleri hedef YSA'ya kopyalanır. Eğitim süreci bittiğinde tahmini optimal politika π elde edilir.

4.2 Politika Gradyan Yöntemler

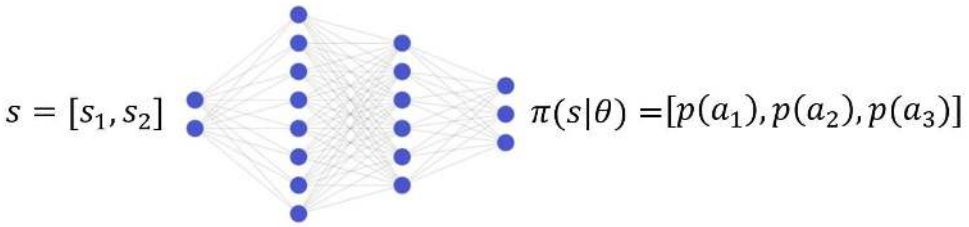
Bu bölümde, politika gradyan yöntemler incelenmiştir. Şimdiye kadar incelenen tüm algoritmalar q değerlerinin tahminlerine dayanır. Bilinen ilk algoritma ailesi, her bir durum-eylem çiftlerinin tahmini q-değerlerinin bağımsız olarak depolandığı tabloları kullanır. Ancak bu yöntemler, durum sayısı çok olduğunda karmaşık görevlerle başa çıkmakta güçlük çeker. Bu klasik algoritmaları daha esnek ve güçlü hale getirmek için fonksiyon yaklaşımcısı olarak YSA'lar kullanılmaktadır. Bu YSA'lar her yeni değerin bağımsız bir tahminini tutmak yerine, öğrenme süreci sırasında şekli değişen bir fonksiyon barındırarak q-değerlerini mümkün olduğu kadar doğru tahmin eder.

Bu iki ailenin ortak özelliği olarak politika, eylemleri gerçekleştirmek için tahmini q değerlerini dikkate alır. Örneğin, açgözlü bir politika $a = arg\max_a Q(s,a)\ veya\ a = arg\max_a \hat{q}(s,a|\theta)$, mevcut durumdaki eylemlerin değerine bakar ve en yüksek getiriyi sağlayacağını tahmin ettiği eylemi seçer. Bu iki yöntem ailesi arasındaki fark, ilk durumda bu değerlerin bir tablodan gözlemlenmesi ve diğerinde bir sinir ağı tarafından üretilmesidir. Basitçe, politika bu değerlere göre tanımlanır.

Politika gradyan yöntemleri olarak bilinen algoritma ailesi ise farklı olarak q-değerlerini tahmin etmek için değil, her bir eylemi gerçekleştiren politikanın olasılıklarını tahmin etmek için bir fonksiyon tahmincisi kullanılır.

$$\pi(a|s,\theta)\ \in [0,1] \tag{4.2}$$

Girdi olarak bir durum alan yapay sinir ağı, bu duruma dayalı olarak, her bir eylemi gerçekleştirme olasılığı olan bir olasılık vektörü üretir. Elbette bu değerlerin her biri sıfır ile bir arasında olacaktır (Denklem 4.2). Bir eylemin gerçekleştirilme olasılığının yüzde 100 olduğu ve tüm olasılıkların toplamının bir olacağı anlamına gelir.



Şekil 4.2 : Stokastik yapay sinir ağı temsili gösterimi.

Politika gradyan yöntemlerinde başka bir deyişle, sinir ağı politikadır. Bu politikalar her eylemin kendisiyle ilişkili bir olasılığa sahip olacağı stokastik politikalardır.

Bu algoritma ailesi değere dayalı yöntemlerin yetersiz kaldığı durumlarda bazı avantajlar sağlar. Bunlardan ilki değere dayalı yöntemlerde stokastik politikaları temsil etmek, yeterince basit ve efektif olmamaktadır. Ajanın, görevle ilgili tüm gerekli bilgilere sahip olmadığı bir görevde optimal politika, örneğin, bir eylemi yüzde 70 olasılıkla ve diğerini zamanın yalnızca yüzde 30'unda yapmak olabilir.

$$\pi(a'|s) = \begin{cases} 1, & eğer \quad a' = arg\max_a \hat{q}(s,a|\theta) \\ 0, & değilse \end{cases} \tag{4.3}$$

Örnek deterministik politika denklem 4.3'te verilmiştir. Bu politika her zaman en yüksek q-değerine sahip eylemi seçen açgözlü bir politikadır. Bu politika deterministtir. 4.4 numaralı denklemde ise epsilon açgözlü politikanın matematiksel ifadesi verilmiştir. Bu politikada ajan her seferinde ϵ bir olasılık ile rastgele bir eylemde bulunur. Bu politika stokastik olduğundan daha çok çevreyi keşfetmeye yöneliktir.

$$\pi(a'|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & eğer \quad a' = arg\max_a \hat{q}(s,a|\theta) \\ \frac{\epsilon}{|A|}, & değilse \end{cases} \tag{4.4}$$

Politika gradyan yöntemlerinin bir başka avantajı öğrenme sürecinde politikanın daha pürüzsüz değişmesidir. Değere dayalı yöntemlerde, bir durumdaki maksimum q değeri bir eylemden diğerine değiştiğinde, yeni eylemi gerçekleştirme olasılığı yüzde sıfırdan yüzde yüze çıkar ve artık optimal olmayan eylemin gerçekleşme olasılığı sıfır olur. Buna karşılık, politika gradyan yöntemlerinde, eylemin etkili olduğu anlaşıldığında, eylemin gerçekleşme olasılığı gitgide artar, eylemin çok iyi olmadığı durumda da yavaş yavaş azalır.

4.2.1 Politika performansı

Optimal politikayı bulmak için, sinir ağı kullanılarak uygulanan bir politikanın diğerleriyle karşılaştırılması gerekir. Bunun için, politikanın performansıyla ilgili bilgi veren bir tür ölçüye ihtiyaç duyulur. Sinir ağının parametrelerini değiştirmek, her durum için farklı olasılık vektörleri üretir. Bu parametrelerin değişmesi yeni bir politika anlamına gelir. Bu nedenle politikanın performansı $J(\theta)$, yapay sinir ağının parametrelerinin bir fonksiyonu olarak tanımlanır. Eğer $J_{\pi_1}(\theta)$ değeri, $J_{\pi_2}(\theta)$ den büyükse π_1 politikası π_2 politikasına tercih edilir. Bu durumda hedef, politika performansını maksimize eden θ parametrelerini bulmak olarak tanımlanır (Denklem 4.5).[4]

$$\pi_*(a|s,\theta) = \arg\max_{\theta} J(\theta) \tag{4.5}$$

Optimal θ parametrelerini bulmak için sinir ağının parametrelerine göre kısmi türevlerini içeren gradyan vektörü, 4.6 numaralı denklem, kullanılır. 4.7 numaralı denklemde matematiksel ifadesi verilen gradyan artışı yöntemiyle belirlenen politika performansı, ajanın çevreden bahse konu politikayı izleyerek edindiği deneyimler kullanılarak tayin edilir. Bu denklemde α öğrenme oranı, gradyan doğrultusundaki adımın büyüklüğünü belirten bir sabit sayıdır.

$$\nabla\hat{J}(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \ldots, \frac{\partial J(\theta)}{\partial \theta_n}\right] \tag{4.6}$$

$$\theta_{t+1} = \theta_t + \alpha\nabla\hat{J}(\theta) \tag{4.7}$$

4.2.2 Politika gradyan teoremi

Bu teorem, politikanın performansını tahmin edebilmek için çevreden hangi değerlerin toplanması gerektiğini söyleyen bir formüldür. Yapılması gereken ilk şey, performans ile ne denilmek istendiğini tam olarak tanımlamaktır. Başlangıç durumu s_0'ın değeri, bu durumdan itibaren beklenen ödüllerin indirgenmiş toplamını temsil eder. Bu nedenle, optimal politika elde etmek için bu ilk durumun değerini maksimize eden politikanın elde edilmesi gerekir (Denklem 4.8).

$$J(\theta) = v_\pi(s_0) \tag{4.8}$$

Performansın hem eylem seçimlerine hem de bu seçimlerin yapıldığı durumların dağılımına bağlı olması ve bunların her ikisinin de politika parametresinden etkilenmesi sebebiyle, politika parametrelerini iyileştirme sağlayacak şekilde değiştirmek zor görünmektedir. Bir durum verildiğinde, politika parametresinin eylemler üzerindeki etkisi ve dolayısıyla ödül üzerindeki etkisi, parametreleştirme bilgisinden nispeten basit bir şekilde hesaplanabilir. Ancak politikanın durum dağılımı üzerindeki etkisi, çevrenin bir işlevidir ve genellikle bilinmez. Gradyan, politika değişikliklerinin durum dağılımı üzerindeki bilinmeyen etkisine bağlı olduğunda, politika parametresine göre, performans gradyanı durum dağılımının türevini içermeyen, politika parametresine göre performans gradyanı için analitik bir ifade sağlayan, politika gradyan teoremi bu zorluğa bir teorik cevap olarak bulunmaktadır (Denklem 4.9).[4]

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi(a|s,\theta) \tag{4.9}$$

Burada $\nabla J(\theta)$, politikaların performansının gradyanı, $\mu(s)$, π politikasını izleyen durumların dağılımı, $q_{\pi}(s,a)$ π politikasını takip eden durum-eylem çiftinin q değeri, $\nabla \pi(a|s,\theta)$ ifadesi ise s durumunda iken a eyleminin gerçekleşme olayının gradyan değeridir. Politikanın performansının gradyanı, her bir durumdaki her eylemin dönütü ile o durumda o eylemi gerçekleştirme olasılığının gradyanı ile orantılıdır ve o politikayı izleyen her durumun gözlemlenme sıklığıyla ağırlıklandırılır. Herhangi bir durumdaki bir eylem, getiriyi artırmak için, pozitif bir getiri üretiyorsa, o eylemi seçme olasılığı artırılır. Eğer bir eylem negatif getiri sağlıyorsa, onu seçme olasılığı azaltılır. Bu teorem sayesinde, ajanın gözlemleyebileceği değerler kullanılarak politika iyileştirilir.

4.2.3 Entropi düzenleme

Politika gradyan algoritmalarını kullanırken, her eylem için optimal seçilme olasılığını bulmak için ajanın çevreyi keşfini sürdürmesi gerekir. Politika yapay sinir ağı olduğunda bunu sürdürmek için yapılan olasılık vektörlerinin entropisini yüksek tutmaya teşvik etmektir.

Entropi rastgele bir değişkenin belirsizlik seviyesini ölçen, fizikteki entropi kavramından farklı, bir bilgi teorisi kavramıdır. Rastgele değişkenin alacağı değerler

hakkında önceden iyi bir fikir edinilebildiği durumda, o rastgele değişkenin entropisi düşüktür. Bu rastgele değişkenlerden örneklenecek değer önceden tahmin edilemiyorsa, o olayın entropisi yüksek olacaktır. Örneğin bir zar atma olayında hilesiz bir zarda tüm yüzeylerin gelme ihtimali eşit olduğu için bu olayın entropisi yüksektir. Eğer bu olayda hileli bir zar kullanılırsa her durumda gelecek yüzey bilindiği için bu durumda entropi düşük olacaktır. Matematiksel olarak 4.10 formülü ile hesaplanır.

$$H(x) = -\sum_{x \in X} p(x).\ln p(x) \tag{4.10}$$

Pekiştirmeli öğrenme sürecinde rastgele değişken, politikanın seçeceği eylemdir (Denklem 4.11). Ajanın çevresini keşfe teşvik etmek için entropiyi tahmini politika performansı hesabına dahil edilir (Denklem 4.12). Gradyan artışı entropiyi de yüksek tutmaya çalışır ve ajanı keşif yapmaya teşvik eder.[33]

$$H_\pi(a_t) = -\sum_{a \in A} \pi(a|s_t).\ln \pi(a|s_t) \tag{4.11}$$

$$\theta_{t+1} = \theta_t + \alpha[\gamma^t G_t \ln \nabla\pi(a_t|s_t, \theta_t) + \beta\nabla H(\pi)] \tag{4.12}$$

Bu teknik, politika optimizasyonunda bazı faydalar sağlar. Birincisi, çevrenin keşfidir. Diğer bir faydası da algoritmaları daha kararlı hale getirmesidir. Yani, algoritma performansında dramatik bir düşüş olmadan, daha önce görmedikleri durumlarla başa çıkabilme yeteneğine sahip olur. Ve son olarak, politika en uygun değerleri bulduğunda, algoritmanın onu daha iyi yapan son yüzde birini iyileştirmesi zordur, entropi, farklı eylemler deneyerek bu küçük ayrıntıları düzeltmeye yardımcı olur.

5. MATERYAL VE YÖNTEM

Bu bölümde algoritmaları uyguladığımız sistem, çevre tasarımı, algoritmaların parametreleri ve ağ mimarileri verilmiştir.

Bu çalışma i7 11800H işlemci, RTX 3060 Laptop grafik kartına ve 16 GB RAM'a sahip Windows 11 işletim sisteminde laptopta Matlab 2022a versiyonuyla yapılmıştır.

5.1 Kullanılan Algoritmalar

Bu bölümde tezde robot kolun yörünge kontrolü görevi için kullanılan ve kıyaslanan algoritmalar hakkında bilgi verilmiştir. Bu algoritmalar modelden bağımsız, politika tabanlı, aktör-kritik ve politika-dışı algoritmalardan derin deterministik politika gradyanı (DDPG), ikiz gecikmeli politika gradyanı (TD3) ve yumuşak-aktör kritik (SAC) algoritmalarıdır.

5.1.1 Derin deterministik politika gradyan (DDPG)

Sürekli eylem uzaylarıyla başa çıkmak için, Q fonksiyonunu kritik olarak kullanan ve Deterministik Politika Gradyanını kullanarak politikayı güncelleyen bir aktör-kritik algoritması Lillicrap ve arkadaşları tarafından ortaya atılmıştır. Bu algoritma modelden bağımsız ve politika-dışıdır.[34]

Sürekli eylem uzayına sahip görevler, ayrık bir olası eylemler kümesi yerine, sürekli olan bir dizi olası eyleme sahiptir, bu da olası eylemlerin sayısının sonsuz olduğu anlamına gelir. Sürekli eylem uzaylarıyla ilgili sorun, çevrede eylemde bulunmak için en yüksek tahmini q değerine sahip eylem $a^*(s) = arg\max_a Q^*(s,a)\,'nın$ bulunması gerektiğidir. Bu da her olası eylem için q değerlerinin hesaplanması ve ardından en yüksek değere sahip olanın seçilmesi için tüm q değerlerinin karşılaştırılmasını gerektirir. Ancak sürekli eylem uzaylarında, sonsuz sayıda olası eylem vardır ve hepsinin q-değerlerini hesaplamak imkansızdır. Bu görevleri çözmek için DDPG algoritması, Q fonksiyonunun eyleme göre diferansiyel olduğunu varsayar. Bu, bir eylemin değeri olan Q fonksiyonu değerinin, eylemler değiştikçe çok düzgün değişeceği anlamına gelir. Bu varsayım, iki eylem hemen hemen aynıysa çok benzer değerlere sahip olmaları gerektiğidir. Örneğin bir teknenin yönetimi probleminde dümeni 15 dereceye ayarlamak, onu 15,1 dereceye ayarlamakla çok benzerdir. Bu varsayım, sinir ağlarını eğitmek için kullanılan teknik olan stokastik gradyan yükselişi

kullanılarak q-değer fonksiyonunu optimize edebilmeyi sağlar. Politika $\pi(s|\theta)$, durumları girdi olarak alan ve eylem üreten bir sinir ağı kullanılarak temsil edilir. Böylece sinir ağını optimize eden mekanizmalarla, en yüksek q-değerlerine sahip eylemleri üretmek için politikanın parametreleri değiştirilir. Belirli bir durumda bir eylem seçilir ve sonra o durumun ve o eylemin q-değerini hesaplamak için kritik olarak adlandıracağımız (Q) sinir ağı kullanılır. Elde edilen Q değeriyle politikanın parametrelerine bağlı gradyanı hesaplanır. Bu gradyan, politikanın seçtiği eylemi değiştirerek q-değerinin mümkün olduğu kadar yüksek olması için uygulanması gereken parametrelerdeki değişiklikleri verecektir. Stokastik gradyan yükselişinin güncelleme kuralı kullanılarak, φ olarak gösterilen politikanın parametreleri, öğrenme oranı α'yla orantılı bir miktarda q-değerinin maksimum büyümesi yönünde kaydırılır (Denklem 5.1).

$$\varphi_{n+1} = \varphi_n + a\nabla_{\varphi} Q\left(s, \pi_{\varphi}(s)\right) \tag{5.1}$$

Politika gradyan ailesinden olan aktör-kritik metotlarda biri eylemleri seçmek (politika) ve diğeri uygunluklarını değerlendirmek için(Q-ağı) iki ayrı sinir ağı bulunur. Politika seçimlerinin daha iyi eylemler olması ve Q-ağının bir durumun değerini daha iyi tahmin etmeyi öğrenmesi için her ikisinin de aynı anda güncellenmesi gerekir. Politika Q-ağı üzerinden güncellenir. Durumun q-değerleri ve politika tarafından seçilen eylemi, gradyan vektörü bulmak için, Q-ağını dahil etmeden, sadece politikanın parametrelerine göre hesaplanır ve ardından politikayı iyileştirmek için stokastik gradyan yükselişi uygulanır. Q-değerlerinin daha iyi tahminlerini üretmek için Q ağının optimizasyonu denklem 5.2 kullanılarak diğer algoritmalardakine benzer şekilde yapılır. Q-değeri tahminleri hedefe itilerek, Q-ağının tahminleri zaman içinde iyileştirilir.

$$L(\theta) = Q_{\theta}(s, a) - r + \gamma Q(s', a') \tag{5.2}$$

Bu algoritma, ayrık eylem uzayından sürekli eylem uzayına geçişin yanı sıra bir başka problemi ve çözümünü de sunar. Deterministik bir politika her zaman en iyi düşünülen eylemi seçen politika anlamına gelir. Bu durumda çevreyi keşfetmek için bir yöntem gerekir. Sonsuz bir eylem yelpazesinde rastgele eylemde bulunmak yerine politikanın seçtiğine benzer eylemleri keşfetmek çözüme daha uygun olacaktır. Bu keşfin gerçekleşip politikanın iyileştirilebilmesi için politika en iyi olduğunu düşündüğü eylemi seçer ve ardından eylem değerlerine biraz gürültü ϵ uygulanır. Bu gürültü,

ortalama sıfır ve belirli bir standart sapma σ ile normal bir dağılımdan örneklenebilir ve politika tarafından seçilen eylemi bozmak için kullanılır (Denklem 5.3). Bunu yapmak önemlidir, çünkü politika yalnızca aşina olduğu eylemler arasından en iyi eylemi bilebilir.

$$\mu'(s) = \mu(s) + \epsilon \quad ve\, \epsilon \sim \mathcal{N}(0, \sigma) \tag{5.3}$$

Algoritma eğitim sürecinde keşif için gürültü eklemenin yanında, eğitim sürecinin kararlılığı için hedeflenen ağları ve algoritmayı daha örnek verimli hale getirmek için bir tekrar arabelleği kullanılır. Örnek verimliliği, algoritmanın ortamdan alınan daha az örnekle daha yüksek miktarda öğrenme gerçekleştirebilmesi anlamına gelir. DDPG algoritmasının sözde kodu çizelge 5.1'deki gibidir.

Çizelge 5.1 : DDPG algoritması sözde kodu.

	DDPG Algoritması
1	Kritik ağı $Q(s, a\|\theta^Q)$ve aktör ağı $\mu(s\|\theta^\mu)$ rastgele ağırlıkları ile oluştur.
2	$\theta_h^Q \leftarrow \theta^Q, \quad \theta_h^\mu \leftarrow \theta^\mu$ Parametreleriyle, Hedef ağlar $Q_h\ ve\ \mu_h$'i oluştur.
3	$B \leftarrow$ Tekrar belleğini başlat
4	n←1, **için döngüye gir**
6	Çevreyi sıfırla, başlangıç pozisyonunu gözlemle $s \leftarrow s_0$,
7	t←0, **için döngüye gir**
8	$a_t = \mu(s_t\|\theta^\mu) + \mathcal{N}_t$, mevcut politika ve gürültü ile bir aksiyon seç
9	a_t eylemini gerçekleştir ve s_{t+1}, r_{t+1}'i gözlemle
10	(s_t, a_t, r_t, s_{t+1}) geçişini tekrar belleğine kaydet
11	$K = (s_i, a_i, r_i, s_{i+1})$ Yığınını bellekten seç
12	$y_i = r_i + \gamma Q_h(s_{i+1}, \mu_h(s_{i+1}\|\theta_h^\mu)\|\theta_h^Q)$
13	$L = \frac{1}{K}\sum_{i=1}[y_i - Q(s_i, a_i\|\theta^Q)]^2$ L'yi minimize edecek şekilde kritik ağını güncelle
14	$\nabla_{\theta^\mu} J \approx \frac{1}{K}\sum_i \nabla_a Q(s, a\|\theta^Q)\|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s\|\theta^\mu)\|_{s_i}$ Örneklenmiş politika gradyanı ile aktör ağını güncelle
15	Hedef ağları güncelle $\theta_h^Q \leftarrow \tau\theta^Q + (1-\tau)\theta_h^Q$ $\tau \in [0,1]$ $\theta_h^\mu \leftarrow \tau\theta^\mu + (1-\tau)\theta_h^\mu$
16	t = t+1, t = T **ise döngüyü bitir**. T bir bölümde atılacak adım sayısı.
17	n←n+1, n=N **ise döngüyü bitir**. N, algoritmanın kaç bölüm çalışacağı

5.1.2 İkiz gecikmeli politika gradyan (TD3)

Fujimoto ve arkadaşları, DDPG algoritmasında Q ağının değer tahminlerinin genellikle olduğundan fazla tahmin edildiğini ve bunun da zayıf politika güncellemelerine ve önemli önyargılara neden olduğunu iddia etmektedir. Bu sorunu çözmek ve DDPG'nin performansını artırmak için birtakım iyileştirmeler ile gelen yeni deterministik politika gradyanı TD3 algoritmasını önermişlerdir.[35]

DDPG algoritmasının çeşitli sorunları vardır. Bunlardan biri, hiper parametre seçimine çok duyarlı olmasıdır. Temel olarak algoritmadan doğru şekilde faydalanabilmek için doğru hiper parametrelerin seçimi gereklidir. Hiper parametre seçimine hassas bir algoritmada doğru değerlerden ufak sapmalar bile bu algoritmaların performansını düşürür. İdeal olarak, doğru hiper parametrelerle çok iyi performans gösterecek bir algoritmayla çalışmak istenilir. Fakat bu parametre ayarı genellikle deneme yanılma yöntemleriyle tespit edildiğinden hem çok zaman alır hem de her zaman kolay olmaz. Bu sebeple daha geniş bir parametre aralığında dahi iyi performans gösteren, hiper parametrelere daha az bağımlı, algoritmalarla çalışmak daha efektiftir. Hiper parametre seçimine hassas bir algoritma olması DDPG'nin kırılgan olmasına sebebiyet verir.

İkinci olarak, daha derin bir problem, q-değeri tahminlerinin olması gerekenden daha yüksek tahmin edilmesi problemi olan, maksimizasyon önyargısıdır. Burada Q-ağı değer tahmininde bulunurken bazı eylemlerin değerlerini olması gerekenden yüksek tahmin eder. Yapılacak eylemlerin seçimi de Q değerlerinin tahminlerine dayandığından, maksimizasyon önyargısı algoritmanın daha iyi eylemleri seçmesine mâni olacaktır. Bu da politikanın optimal politikanın altında olacağı, ajanın daha kötü bir performansa sahip olacağı anlamına gelir.

Bu sorunları çözmek için TD3 algoritması, DDPG algoritmasına üç iyileştirme sunar. Bunlardan ilki kırpılmış çift Q-öğrenme olarak adlandırılan, q-değerlerini hesaplamak için her biri kendi hedef ağına sahip iki ayrı Q ağı oluşturulması tekniğidir. Bu teknikte sinir ağları güncellenirken, q-değeri tahmini için her iki sinir ağı da birbirinden bağımsız olarak kullanılır. Ardından, iki ağdan gelen iki değer içerisinden en düşük olanı seçilir. Böylece, iki bağımsız sinir ağından biri hata yapar ve q değerini fazla tahmin ederse, diğeri daha düşük bir q değeri sunarak bunu düzeltir ve algoritma düşük değeri kullanır. İki bağımsız Q ağının bulunması, her birinin diğerinin fazla tahminini

düzeltmesine izin verir. Bunun etkisi, artık çok daha gerçekçi bir hedef $\hat{y} = r + \min_{i=1,2} Q_i(s', a')$ hesaplanır ve bu sinir ağlarının her birini bu hedefi kullanarak günceller (Denklem 5.4).

$$L(\theta_i) = (Q_i(s, a) - \hat{y}\,) \tag{5.4}$$

İkinci iyileştirme, gecikmeli politika güncellemeleridir. DDPG algoritmasında, öğrenme sürecinin başlangıcında, politikayı iyileştirmek için kullanılan q değerlerinin tahminleri kötüdür. Politika bu kötü tahminler ile eylemlerini gerçekleştirirse, kararsız hale gelerek ve aynı eylemler arasında gidip gelir. Politika kararsız olduğunda tecrübe edilen durumların dağılımı da düzensiz olur. Bu durumu önlemek için, Q değerlerinin her x güncellemesinde politika yalnızca bir kez güncellenir. Makalede bu değer 2 olarak kullanılmıştır [35].

Üçüncü ve son iyileştirme tekniği politika yumuşatmadır. Algoritma aynı durumda her zaman aynı eylemi yapması anlamına gelen deterministik bir politikadır. Sinir ağının belirli bir eyleme gerçekçi olmayan yüksek bir değer ataması durumunda, deterministik politika, bu yüksek değerden hızla yararlanarak her zaman o durumda o abartılmış eylemi seçer ve o eylemin etrafında sivri bir tepe noktası oluşmasına sebebiyet verir. Bu durum DDPG'nin ilk önermesi olan yakın eylemlerin yakın değerlere sahip olması gerektiği ve bunların pratikte geçerli olma eğiliminde olması önermesiyle ters düşer. Burada önerilen, Q-ağı güncellemesinin hedefi $\hat{y}$ 'i hesaplanırken, politika tarafından seçilen eyleme biraz gürültü $\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$ eklenerek yakındaki bir eyleme hafifçe itmektir (Denklem 5.5). Bu yeni eylem, politika tarafından seçilene çok yakın olduğundan, çok benzer bir değere sahip olmalıdır, bu nedenle bu hedef geçerli olacaktır.

$$\hat{y} = r + \gamma\, Q(s', \mu(s') + \epsilon) \tag{5.5}$$

Bu üç iyileştirme ile önerilen TD3 algoritmasının sözde kodu çizelge 5.2'deki gibidir. Burada DDPG sözde kodundan farklı olarak politika μ yerine π ile temsil edilmiştir.

Çizelge 5.2 : TD3 algoritması sanki kodu.

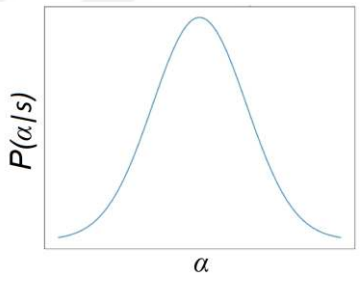
	TD3 Algoritması
1	Kritik ağlar $Q_{\theta_1} ve\ Q_{\theta_2}$ aktör ağı π_ϕ'yi rastgele ağırlıklar $\theta_1, \theta_2 ve\ \phi$ ile oluşturur.
2	$\theta_{1_h} \leftarrow \theta_1, \quad \theta_{2_h} \leftarrow \theta_2\ ve\ \phi_h \leftarrow \phi$ Parametreleriyle, Hedef ağlar $Q_{1_h}, Q_{2_h}\ ve\ \pi_h$'i oluştur.
3	$B \leftarrow$ Tekrar belleğini başlat.
4	n←1, **için döngüye gir**
6	Çevreyi sıfırla, başlangıç pozisyonunu gözlemle $s \leftarrow s_0$,
7	t←0, **için döngüye gir**
8	$a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ mevcut politika ve gürültü ile bir aksiyon seç
9	a Eylemini gerçekleştir ve s', r'i gözlemle
10	$B \leftarrow (s, a, r, s')$ Geçişi tekrar belleğine kaydet
11	$K = (s_i, a_i, r_i, s_{i+1})\ yığınını\ bellekten\ seç$
12	$\tilde{a} \leftarrow \pi_{\phi'} + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$
13	$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
14	$\theta_i \leftarrow arg\min_{\theta_i} K^{-1} \sum (y - Q_{\theta_i}(s,a))^2$, Kritik ağını güncelle
15	$\boldsymbol{eğer}\ t\ mod\ d\ \boldsymbol{ise}$, d politika güncelleme sıklığı
16	$\nabla_\phi J(\phi) = K^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$ Deterministik politika gradyanı ile aktör ağını güncelle
17	Hedef ağları güncelle $\theta_{i_h} \leftarrow \tau\theta_i + (1-\tau)\theta_{i_h}$ $\tau \in [0,1]$ $\phi_h \leftarrow \tau\phi + (1-\tau)\phi_h$
18	t = t+1, t = T **ise döngüyü bitir**. T bir bölümde atılacak adım sayısı.
19	n←n+1, n=N **ise döngüyü bitir**. N, algoritmanın kaç bölüm çalışacağı

5.1.3 Yumuşak-aktör kritik (SAC)

Haarnoja ve arkadaşları, 2018 yılında yumuşak-aktör kritik algoritmasını duyurdular. SAC'i hiper parametrelere hassas olmayan, politika-dışı, modelden bağımsız pekiştirmeli öğrenme algoritması olacak şekilde geliştirdiler[36].

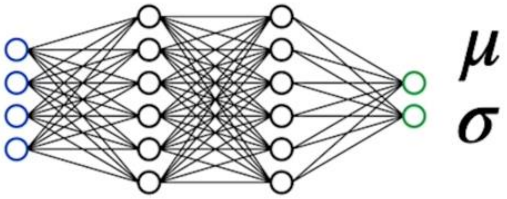
Hem DDPG hem de TD3 deterministik politikalar kullanan algoritmalardır. Bu, belirli bir durumda, bu politikaların her zaman aynı eylemi seçeceği anlamına gelir. Bu tür politikaları kullanmak, ajanın çevreyi keşfetmesini sağlamak için bazı ekstra hamleler yapmayı gerektirir, verilen epsilon oranında rastgele eylem seçen epsilon-açgözlü politikalar, derin öğrenme durumunda, hedef politika yumuşatma ve TD3 durumunda, eylemlere rastgele gürültü ekleme gibi.

Öte yandan, stokastik politikalar bu ekstra hamlelere ihtiyaç duymaz çünkü her zaman aynı durumda aynı eylemi seçmezler. Bunun yerine, olası eylemlerin her birine olasılıklar atar ve bu olasılıktan örnekler çıkarırlar. Örneğin, bir s durumunda stokastik bir politika, o durumdaki her bir eyleme seçme olasılığı atar. Farklı eylemlerin farklı olasılıkları olur ve en yüksek olasılığa sahip bir veya daha fazla eylem bulunabilir.



Şekil 5.1 : Bir s durumunda eylemlerin normalleştirilmiş olasılık dağılımı.

Politikayı temsil etmek için sinir ağları kullanıldığında; sürekli eylem uzayında sonsuz sayıda olası eylem olduğundan, politika sonsuz miktarda olasılık hesaplaması gerekir. Bu imkansızlıkla baş etmek için, bu olasılık dağılımları, normal bir dağılım olmaya zorlanır. Böylece imkânsız ve sonsuz bir olasılık hesabı yerine olasılık dağılımının ortalaması (μ) ve bu dağılımın standart sapması (σ)olmak üzere iki değer hesaplanır. Politika, μ ve σ çıktılarını veren bir sinir ağı olarak temsil edilir ve bu iki değerle olasılık fonksiyonunu oluşturarak eylemi bu olasılıktan örnekler. Çoğu zaman en yüksek olasılığa sahip değerleri alarak onları pekiştirirken, bazen ortalamadan sapan eylemler ile keşif yapar.



Şekil 5.2 : Stokastik politikanın sinir ağı ile temsili.

Sinir ağı başarıyı erken bulursa, olasılık dağılımını daha ince olmaya zorlar, böylece her zaman aynı eylemleri seçer ve olası değerlerin geri kalanı daha düşük bir olasılığa sahip olur. Yani, bu olasılık dağılımının standart sapması sıfıra doğru gider ve nispeten keşfetmeyi erkenden bırakır. Eğitim süreci boyunca politikanın keşfi sürdürmesi, farklı eylemleri de denemesi için eylemlerin olasılığının yayılması olan sigmanın değerini nispeten yüksek tutmak gerekir. Entropi-düzenli pekiştirmeli öğrenme tekniği bu soruna bir çözüm olarak sunulmuştur[37]. Bu teknikte, amaç politikanın entropisini mümkün olduğu kadar yüksek tutarken getiriyi en üst düzeye çıkarmaktır. Böylece, politikanın entropisini yüksek tutarak, ajan keşfi yüksek tuttuğu için de ödüllendirilir. Burada $\hat{y}$ hedef güncelleme denklemi entropinin de dahil edilmesiyle denklem 5.6'daki halini alır. Burada α entropinin ne kadar önemli olduğunu belirleyen bir katsayıdır. Bu değişiklikle birlikte politika güncelleme kuralı da denklem 5.7'deki gibi olur.

$$\hat{y} = r + \gamma \left(\min_{i=1,2} Q_i(s', a') - \alpha \ln \pi(a'|s') \right) \tag{5.6}$$

$$J(\phi) = \left(\min_{i=1,2} Q_i(s', a') - \alpha \ln \pi(a'|s') \right) \tag{5.7}$$

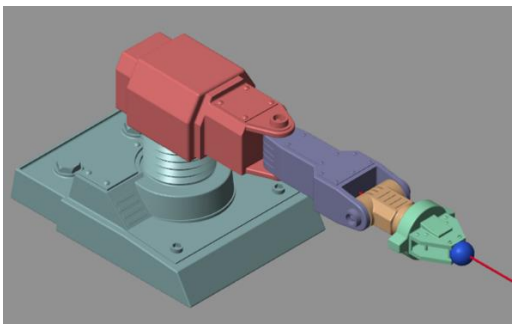
Yumuşak-aktör kritik algoritmasının sözde kodu yukarıdaki bilgiler doğrultusunda çizelge 5.3'teki gibidir.

Çizelge 5.3 : Yumuşak-aktör kritik algoritmasının sözde kodu.

	SAC Algoritması
1	Kritik ağlar $Q_{\theta_1} ve\ Q_{\theta_2}$ ve aktör ağı π_ϕ'yi rastgele ağırlıklar $\theta_1, \theta_2 ve\ \phi$ ile oluştur.
2	$\theta_{1_h} \leftarrow \theta_1, \quad \theta_{2_h} \leftarrow \theta_2$ Parametreleriyle, hedef ağlar Q_{1_h}, Q_{2_h}'i oluştur.
3	$B \leftarrow$ Tekrar belleğini başlat
4	n←1, **için döngüye gir**
6	Çevreyi sıfırla, başlangıç pozisyonunu gözlemle $s \leftarrow s_0$,
7	t←0, **için döngüye gir**
8	$\alpha \sim \pi_\phi(\cdot\,|s)$, mevcut politika ile bir aksiyon seç.
9	α, eylemini gerçekleştir ve s', r'i gözlemle
10	$B \leftarrow (s, a, r, s')$ geçişini tekrar belleğine kaydet
11	***eğer*** $t\ mod\ d$ ***ise***, d güncelleme sıklığı
12	Rastgele $K = (S, A, R, S')$ yığınını B'den örnekle
13	Q fonksiyonu hedef değerlerini güncelle. $y(r, s') = r + \gamma\left(\min_{i=1,2} Q_{i_h}(s', \bar{a}') - \alpha \log \pi_\theta(\bar{a}'|s')\right), \bar{a}' \sim \pi_\phi(\cdot\,|s')$
14	Q fonksiyonlarını bir adım gradyan düşüş uygulayarak güncelle $\nabla_{\theta_i} \frac{1}{\lvert K\rvert} \sum_{(s,a,r,s')\in K} \left(Q_{\theta_i}(s,a) - y(r,s')\right)^2, i = 1,2$
15	Politikayı bir adım gradyan yükselişi uygulayarak güncelle $\nabla_\phi \frac{1}{\lvert K\rvert} \sum_{s\in K} \left(\min_{i=1,2} Q_{\theta_i}\left(s, \tilde{a}_\phi(s)\right) - \alpha\, log \pi_\phi\left(\tilde{a}_\phi(s)|s\right)\right)$, $\tilde{a}_\phi(s), \phi$'e göre türevlenebilen $\pi_\phi(\cdot\,|s)$'den bir örnektir.
16	Hedef ağları güncelle $\theta_{i_h} \leftarrow \tau\theta_i + (1-\tau)\theta_{i_h}, \tau \in [0,1]$
17	t ← t+1, t = T **ise döngüyü bitir**. T bir bölümde atılacak adım sayısı.
18	n←n+1, n=N **ise döngüyü bitir**. N, algoritmanın kaç bölüm çalışacağı

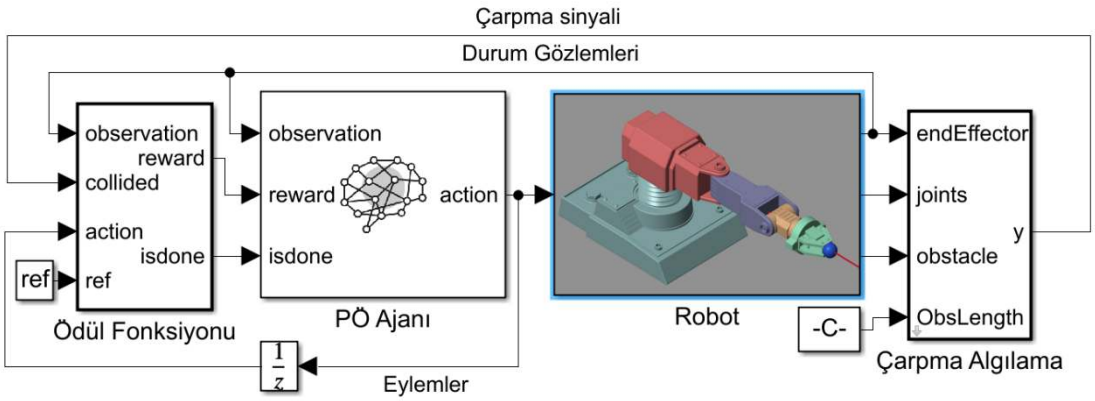
5.2 Robot Kol

Çalışmada kullanılan Şekil 5.3'te görseli verilen robot kol, Matlab'ın örnekler kütüphanesinde "Modeling A Robot Using STEP Files" dosyasında bulunan "Robot Arm" modelidir[38].



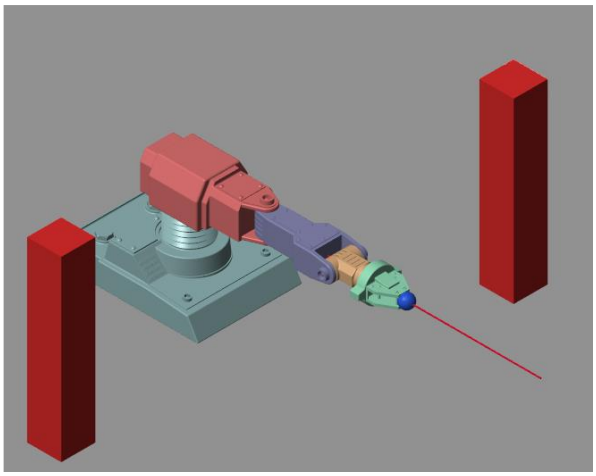
Şekil 5.3 : Tez çalışmasında kullanılan robot kol görseli.

Robot kol ucundaki tutucu çıkartılıp, yerine hedef takibini kolaylaştırmak için 3 cm çapında küresel katı madde tanımlandı. Yine hedef doğrultusu kırmızı bir çizgi yardımıyla görsel olarak algılanabilmesi için belirtildi. Simulink arayüzü aracılığıyla gerekli bloklar eklenerek pekiştirmeli öğrenme algoritmalarını uygulanabilir hale getirildi. Simulink'te hazırlanmış blok diyagramı Şekil 5.4'deki gibidir.



Şekil 5.4 : Robot kol simulink blok diyagramı.

Simscape/Simulink aracılığıyla çevreye iki adet görsel olarak sütun eklendi. Şekil 5.5'teki engelli çevre modeli görselleştirildi.



Şekil 5.5 : Engelli çevre görseli.

Çarpma algılama bloğuna işlevci ucun koordinatları girildi. Uygun fonksiyon yazılarak sütunların koordinatlarıyla çakışma durumunda çarpma sinyali üretmesi sağlandı. Engelsiz çevre modelinde burada herhangi bir sinyal üretilmedi.

Robot kolundaki eklemleri gövdeden uca doğru numaralandırılarak, robot kolun 3 ekleminden birincisinin, ana gövde ekleminin, hareketi $\mp 170°$ ile sınırlandırıldı. Diğer iki eklem $\mp 90°$ arasında hareket edebilmektedir. Bu çalışmada hedeflenen durum robot kolun rastgele bir pozisyondan, tüm eklem açılarının 0° olduğu hedef pozisyonuna gelmesi istenmiştir. Bunun için tüm eklemlerin sıfır derecede olduğu konumda mavi topun merkezi referans noktası olarak alınmıştır.

5.2.1 Durum uzayı

Bu görevde durum uzayımız süreklidir. Durum uzayımız gözlemlenebilir kabul edilmiş ve aşağıdaki 12 değeri içermektedir;

$$s_t = [x, y, z, v_x, v_y, v_z, w_1, w_2, w_3, \theta_1, \theta_2, \theta_3] \quad (5.8)$$

Burada x, y ve z işlevci ucun kartezyen koordinatlarını temsil etmektedir. $v_x, v_y \; ve \; v_z$ Terimleri ise işlevci ucun indisle belirtilen doğrultudaki çizgisel hızlarını temsil etmektedir. $w_1, w_2 \; ve \; w_3$ Terimleri de indislerinde belirtilen eklemin açısal hızını içermektedir. Son üç terim $\theta_1, \theta_2, ve \; \theta_3$ ise indislerinde belirtilen eklemin derecesini vermektedir. Kullanılan PÖ algoritmalarına bu terimler girdi olarak verilmiştir.

5.2.2 Eylem uzayı

Eylem uzayımız yine durum uzayı gibi süreklidir ve ajanımız tarafından durum girdilerine bir cevap mahiyetinde üretilmektedir. 3 ayrı eylem noktamız bulunmaktadır. Bu noktalarda eylem uzayımız 5.9'daki gibi tanımlanmıştır;

$$a_t = [\tau_1, \tau_2, \tau_3] \tag{5.9}$$

Burada τ eklemlere uygulanan torku temsil ederken, indislerde uygulandıkları eklemleri belirtmektedir. Eylemler $\tau \in [-1, 1]Nm$ olacak şekilde kısıtlanmıştır.

5.2.3 Ödül ve ceza

PÖ algoritmalarında ödül fonksiyonları hazırlanırken bayağı veya yoğun olarak verilebilmektedir. Bayağı ödül fonksiyonu sadece hedefi gerçekleştirme durumunda "tamamlandı" sinyaliyle algoritmayı ödüllendirir. Bu çalışmada hedefe yaklaştıkça da her adımda ödül/ceza sinyali üreterek hedefe doğru algoritmayı teşvik eden yoğun diye tanımlayabileceğimiz bir ödül fonksiyonu kullandık. Ödül fonksiyonu formülümüz 5.10'daki gibidir.

$$r_t = -k * \left(\sqrt{(x_h - x_t)^2} + \sqrt{(y_h - y_t)^2} + \sqrt{(z_h - z_t)^2}\right) - 100 * C_t + 10 * madeit \tag{5.10}$$

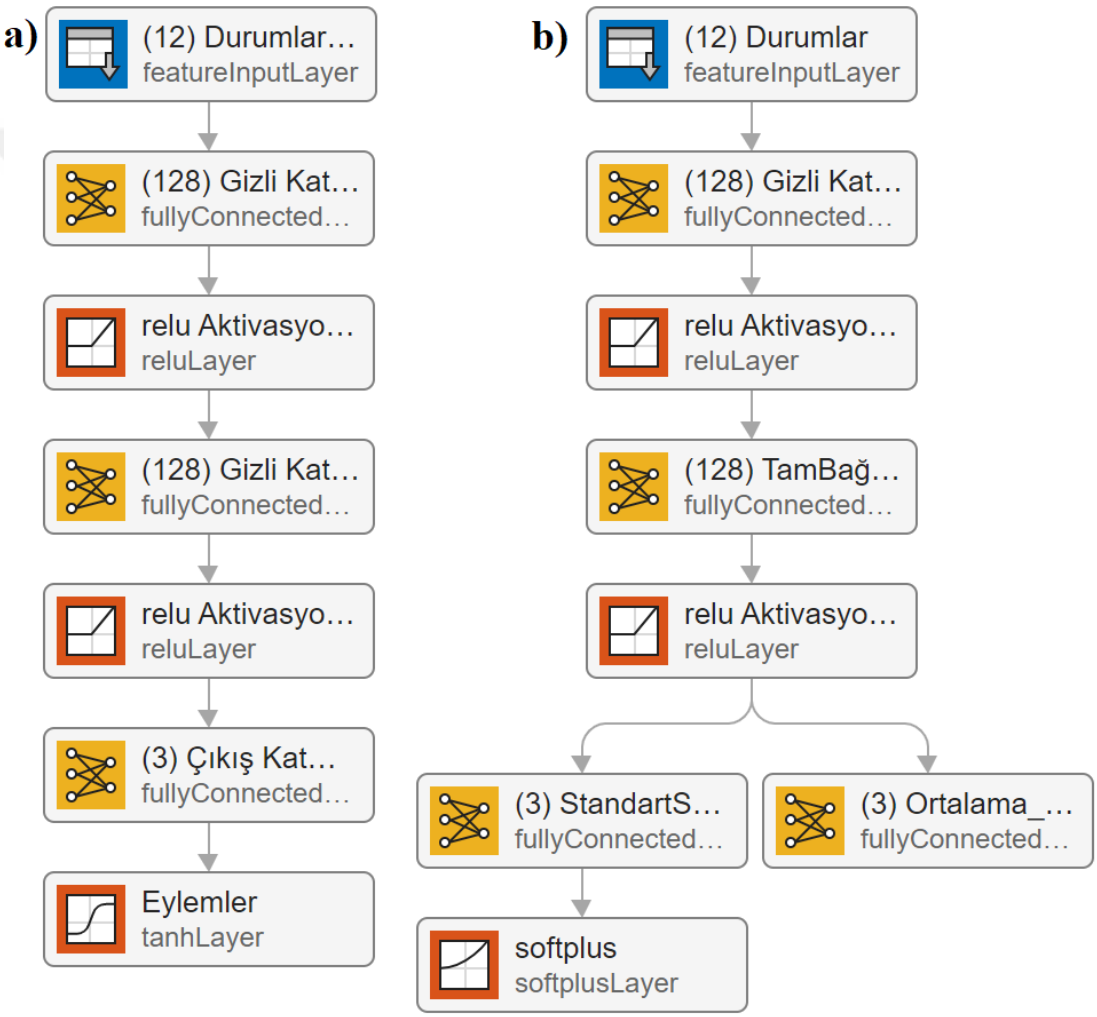
Burada her adımda hedef noktasından her bir koordinat elemanının öklid uzaklığını hesaplayarak bunun negatifini sinyal olarak vermektedir. Böylece işlevci uç hedefe yaklaştıkça ödül büyümekte, uzaklaştıkça küçülmektedir. Burada ortaya çıkan ödülü ölçeklemek için $k = 0.01$ katsayısı kullanılmıştır. C_t , terimi çarpışma algılama fonksiyonumuz tarafından üretilen sinyaldir. Çarpışma durumunda 1 sinyalini üretir. Ödül formülümüzde -100 ile çarpılarak eklenmiştir. $madeit$ Terimi ise, başarı olarak tanımlanan, işlevci ucun hedef noktasına belirli tolerans içerisinde ulaştığında 1 diğer durumlarda ise 0 üreten bir fonksiyon ile üretilmiştir.

Bu ödül fonksiyonu sayesinde çarpma olduğu zaman getiri -100 değerinden aşağıda kalmakta, başarı olarak tanımlanan kriter yerine geldiğinde ise 0'dan büyük bir ödül almaktadır.

5.2.4 YSA mimarileri

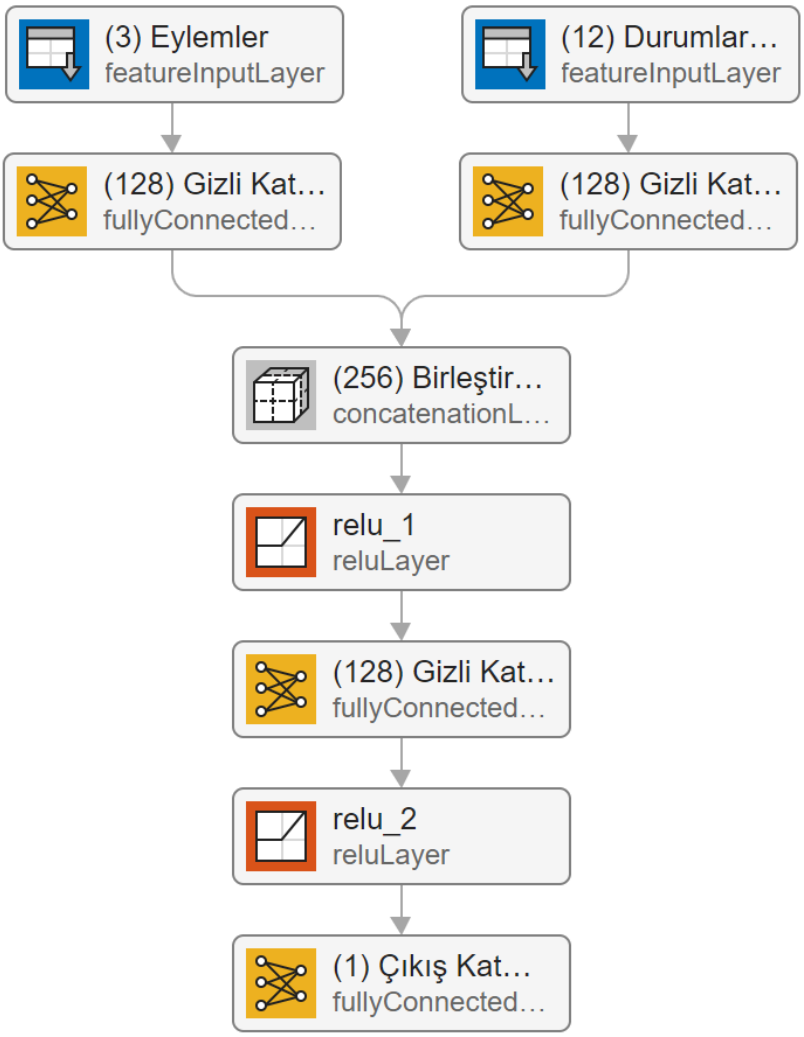
Bu bölümde her bir algoritma için kullandığımız YSA mimarileri verilmiştir. Aktör-Kritik algoritmalar kullanıldığından iki farklı ağ yapısı kullanıldı. Aktör YSA yapısı Şekil 5.6'te verilmiştir. Deterministik aktör barındıran DDPG ve TD3 algoritmalarında

Şekil 5.6-a'da gösterilen ağ yapısı kullanılmıştır. SAC algoritması ise stokastik aktör barındırdığından 5.6-b'deki ağ yapısı kullanılmıştır. Her iki ağ yapısında da giriş katmanları durum olarak 12 adet gözlem barındıran giriş katmanı içermektedir. Giriş katmanından sonra 128'er nöron içeren aktivasyon fonksiyonu olarak reLU fonksiyonunu kullanan iki adet tam bağlantılı gizli katman bulunmaktadır. Deterministik ağ yapısında çıkış katmanı doğrudan eylemi üretmekte, stokastik ağ yapısında ise çıkış katmanında eylemler dağılım ortalaması ve standart sapma olarak üretilmektedir.



Şekil 5.6 : Aktör YSA mimarisi, a) Deterministik aktör, b) Stokastik aktör.

Kritik için ise tezde kullanılan tüm algoritmalarda aynı mimari kullanılmıştır. Girdi olarak durumları ve bu durumlara karşı ajan tarafından gerçekleştirilen eylemleri alır. Bu katmanları yine 128 nöron barındıran gizli katman ve takip eden birleştirme katmanı bulunmaktadır. İkinci bir gizli katmandan sonra bir q-değeri üreten çıkış katmanı barındırmaktadır. Kritik YSA mimarisi Şekil 5.7'te verilmiştir.



Şekil 5.7 : Kritik YSA mimarisi.

5.2.5 Algoritma ve eğitim parametreleri

YSA mimarileri belirlendikten sonra ajanların karakteristik parametreleri çizelge 5.4'teki gibi belirlendi.

Çizelge 5.4 : PÖ ajanları için seçilen parametre değerleri.

Parametre Adı	Değeri
Örnekleme Süresi	0.01
Azaltma Faktörü γ	0.99
Yığın Sayısı	256
Tekrar Belleği Uzunluğu	10^6
Aktör-Kritik Öğrenme Oranı	1.10^{-4}

Aynı parametrelere sahip her bir ajan bir engelli çevrede ve bir engelsiz çevrede olmak üzere eğitildi. Eğitim sonucunda her algoritmadan iki farklı eğitilmiş ajan elde edildi. Eğitim süreci için belirlenen değerler çizelge 5.5'te verilmiştir.

Çizelge 5.5 : Eğitim süreci için belirlenen değerler.

Parametre Adı	Değeri
Bölüm Sayısı	10^4
Maksimum Adım Sayısı	150
Ortalamaya Katılan Bölüm Sayısı	50

Her ajan her bir çevrede 10.000 bölüm eğitilmiştir. Eğitim sonlandırma kriteri maksimum bölüm sayısıdır. Maksimum adım sayısı, her bir bölüm eğer ajan tarafından hedefe daha önce ulaşılmaz ise kaç adımda sonlandırılacağını belirleyen parametredir. Eğitim sürecini takip ettiğimiz bölüm yöneticisi arayüzünde anlık ortalama hesapları yapılırken son 50 bölümde elde edilen değerler hesaba katılmıştır.

6. BULGULAR VE TARTIŞMA

6.1 Eğitim Süreçleri ve Kıyaslanması

Eğitim sürecini takip ettiğimiz bölüm yöneticisindeki grafiğe baktığımızda ilk bin bölüm sonrasında ortalama getiri ve salınımlar belirli bir desene oturmuş olsa da çevrenin tam keşfi için eğitimler on bin bölüm yapılmıştır. Eğitim süreçlerinin getiri grafikleri tüm süreci kapsayan şekliyle Ek A'da verilmiştir. Bu bölümde eğitim karakteristiklerini görmek için ilk bin bölüm için engelsiz çevrede eğitim grafiği Şekil 6.1'de engelli çevrede eğitim grafiği ise Şekil 6.2'de verilmiştir. Her bir eğitimin toplam adım sayısı ve ortalama getirisi engelsiz çevre için çizelge 6.1 ve engelli çevre için 6.2'te verilmiştir. Her ne kadar çevre koşullarının etkisi kontrol edilememiş olsa da bir fikir olması açısından her bir eğitimin süreleri de genel bilgi olarak verilmiştir. Bu çizelgedeki ortalama getiri tüm ajanların eğitim sürecinin belirli bir desene oturmuş olduğu son bin bölümde elde edilen getiriler ile hesaplanmıştır.

Çizelge 6.1 : Engelsiz çevre ile eğitim sonuçları.

	DDPG	TD3	SAC
Ortalama Getiri	-25,35	-12,72	-4,65
Toplam Adım Sayısı	1.495.861	1.492.778	1.284.793
Eğitim Süresi	6:44:58	8:22:17	8:48:15

Çizelge 6.1'e baktığımızda en yüksek ortalama getiriye erişen SAC ajan olmuştur. DDPG ajanı nispeten diğer iki ajana göre eğitim süresi kısa olsa da hem eğitim süresi boyunca yüksek salınım göstermiş hem de daha düşük ortalamalara sahip olduğu görülmüştür. Bir diğer başarı göstergesi de bölümleri minimum adım ile bitirme başarısıdır. Burada da SAC ajan eğitim süresinde toplam adım sayısı düşüklüğü ile ön plana çıkmaktadır.

Çizelge 6.2 : Engelli çevre ile eğitim sonuçları.

	DDPG	TD3	SAC
Ortalama Getiri	-26,41	-19,56	-16,97
Toplam Adım Sayısı	1.472.386	1.462.832	1.450.095
Eğitim Süresi	10:15:31	14:22:29	9:34:41

Çizelge 6.2'te ise engelli çevreden gelen çarpma sinyalinin eğitim sürecine etkisi görülmektedir. Engelli çevreden gelen çarpma sinyali eğitim süreçlerinin uzamasına daha düşük ortalamalara ve daha yüksek adım sayısına sebebiyet vermiştir. Her ne kadar TD3 ajanın eğitim süresi artmışsa da performans olarak SAC ajan ile yakın sonuçlar üretmiştir. Fakat başarı kriteri yönünden bakacak olursak SAC ajan 0 değerinden yüksek değerlere daha fazla ulaşarak eğitim sürecinde başarı kriterini en çok başaran ajan olmuştur. Eğitim süresince her ajanın sıfırdan yüksek getiri elde ettiği bölüm sayısı çizelge 6.3'te verilmiştir.

Çizelge 6.3 : Ajanların sıfırdan yüksek getiri aldıkları bölüm sayıları.

Çevre Modeli	Başarı Sayıları		
	DDPG	TD3	SAC
Engelsiz Çevre	44	77	1914
Engelli Çevre	11	18	182

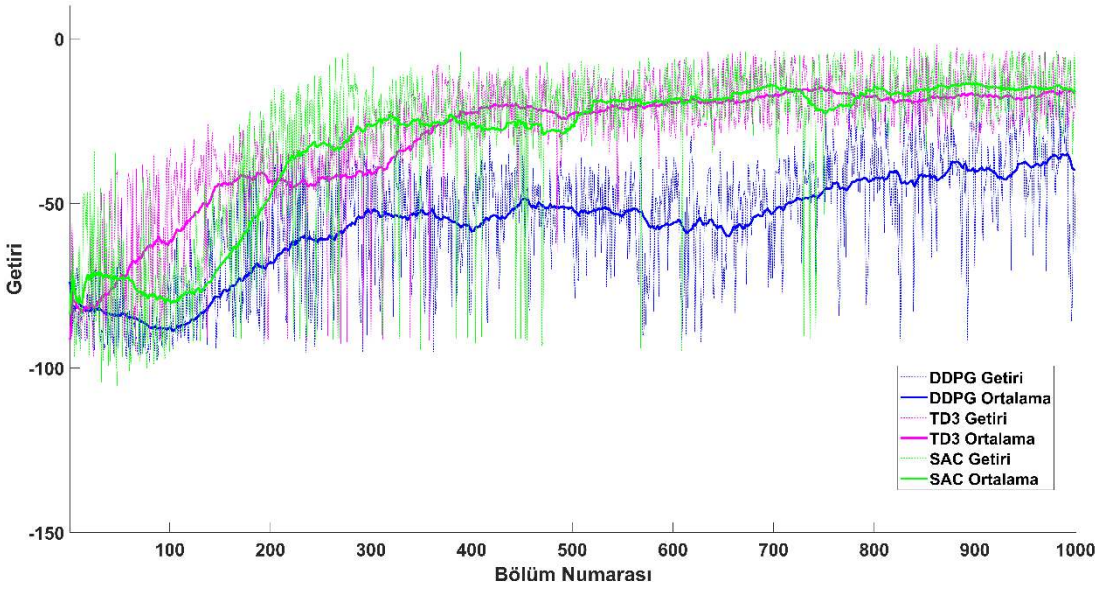
Şekil 6.3'te eğitim süreci boyunca elde edilen getirilerin histogram dağılımları eğitim süreç performansları hakkında bilgi vermesi açısından paylaşılmıştır. Bu dağılımlara baktığımızda TD3 ve SAC ajanları eğitim boyunca daha yüksek getirilere daha fazla bölümde ulaşmış ve dağılımları benzeşmektedir.

Aktör-kritik algoritmalarda önemli bir etken olan kritik ağının başlangıç pozisyonu için yapmış olduğu tahmini getiri olarak tanımlayabileceğimiz Q_0 değeri de algoritma eğitim süreçlerindeki performans kriterlerinden biridir. Q_0 Değeri ile bölüm getirisinin yakınlığı kritik ağının ne kadar doğru tahminler üretebildiğinin bir ölçüsüdür. Bu doğrultuda çizelge 6.4'te Q_0 ve bölüm getirisi arasındaki ortalama hata verilmiştir.

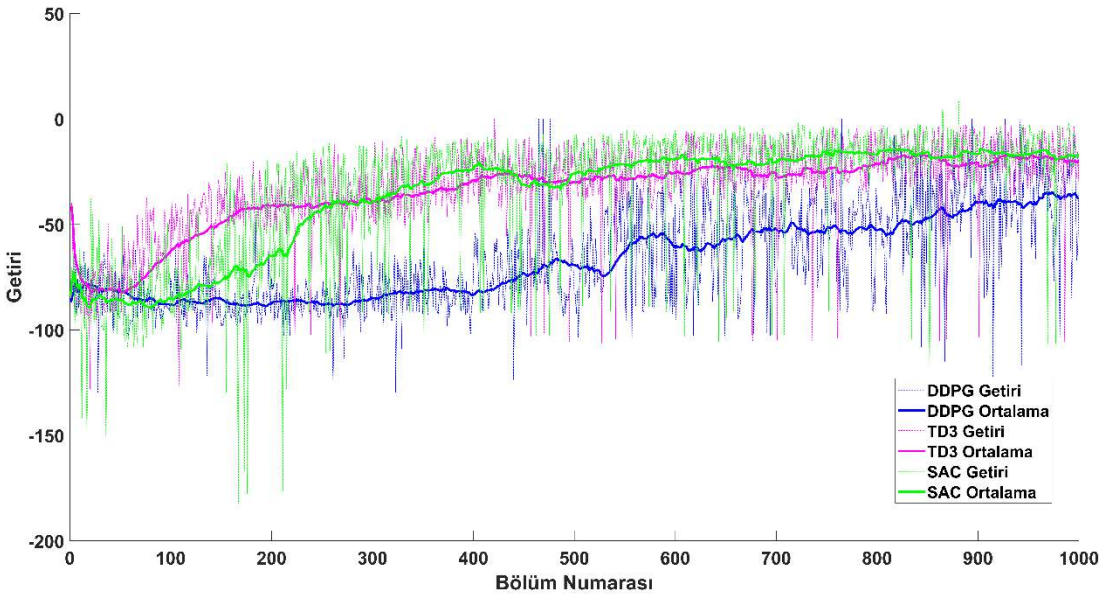
Çizelge 6.4 : Q_0 değeri ortalama hata tablosu.

Çevre Modeli	Q_0 Değeri ortalama hatası		
	DDPG	TD3	SAC
Engelsiz Çevre	±19.0337	±2.7388	±5.8220
Engelli Çevre	±23.7389	±5.4809	±6.5231

Çizelge 6.4'e baktığımızda TD3'ün engelsiz çevrede yüksek bir başarı elde ettiği görülmektedir. DDPG algoritmasının q-değerlerini abartması problemi de değerlere baktığımızda gözükmektedir. Ek A'da son 50 bölüm için Q_0 değeri detay grafiği verilmiştir.



Şekil 6.1 : İlk 1000 bölüm için engelsiz çevrede eğitim süreci grafiği.

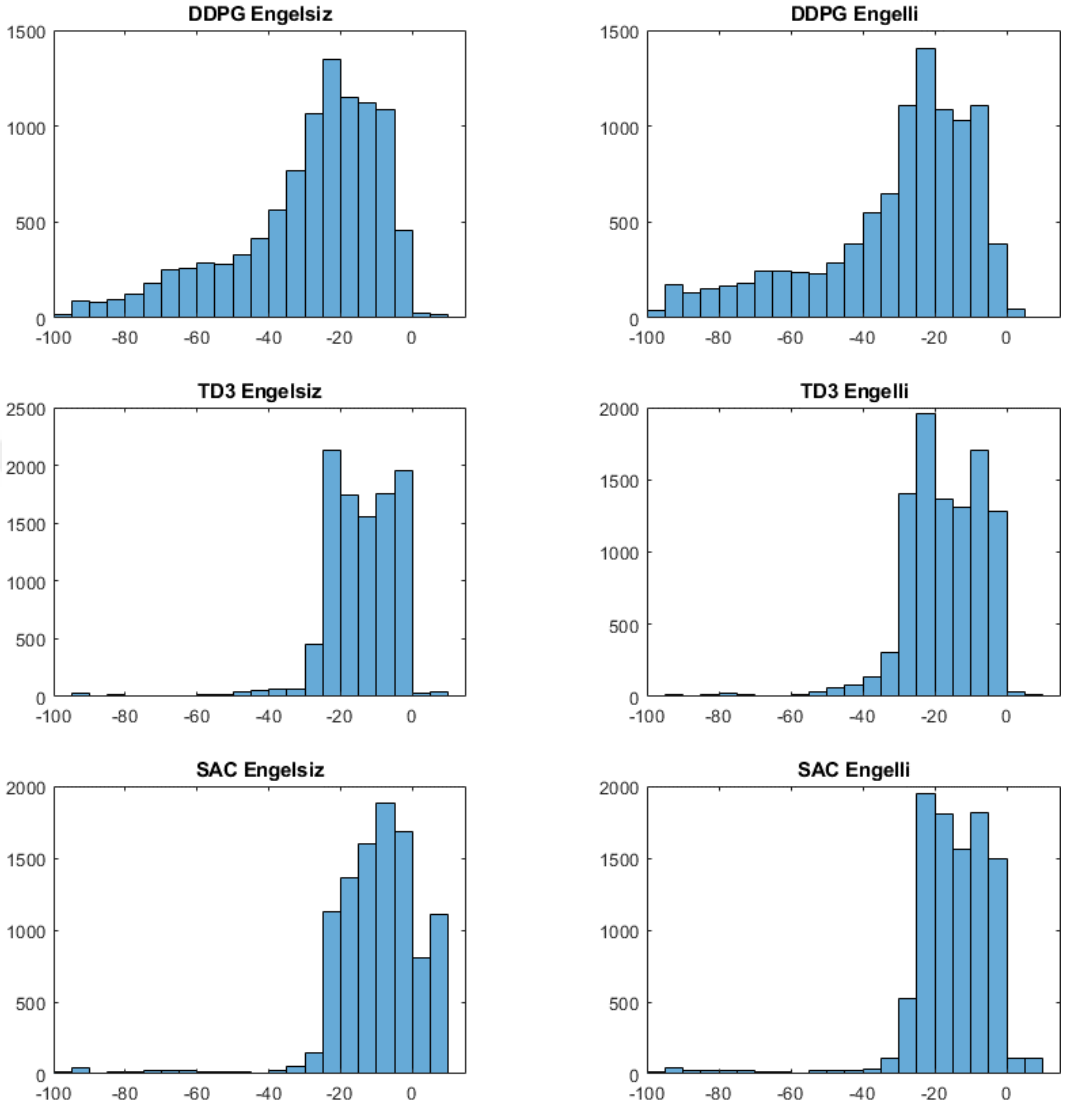


Şekil 6.2 : İlk 1000 bölüm için engelli çevrede eğitim süreci grafiği.

Tüm bu değer ve grafiklere baktığımızda, DDPG algoritması üzerine yapılan iyileştirmelerin isabetli olduğu gözükmektedir. Engelsiz çevrede yapılan eğitim sonuçları TD3 algoritmanın kritik tahminlerindeki iyileşmeyi çok net göstermektedir. Burada farklı olarak SAC ajanın tahmin hatası TD3 ajana göre fazla olmuştur. Bunun sebebi ise aktörün stokastik olması sebebiyle bölüm boyunca beklenmeyen bir eylemi de seçebilmesi olduğu düşünülmüştür.

Çalışmada engelli çevrede engeller ajana bir gözlem olarak sunulmamış yalnızca ödül fonksiyonunda tanımlandığından ötürü her ajanın eğitim süreçlerine olumsuz etkisi

büyüktür. Eğitim sürecinde ajanların eğitim sonuna kadar bir şekilde engellere çarptığı görülmüştür. Bu da bölüm ortalamalarında düşük değerlere sebebiyet vermiştir. Yine aynı sebepten kritik ağının bölüm başı tahmin hatalarında yükselme görülmüştür.

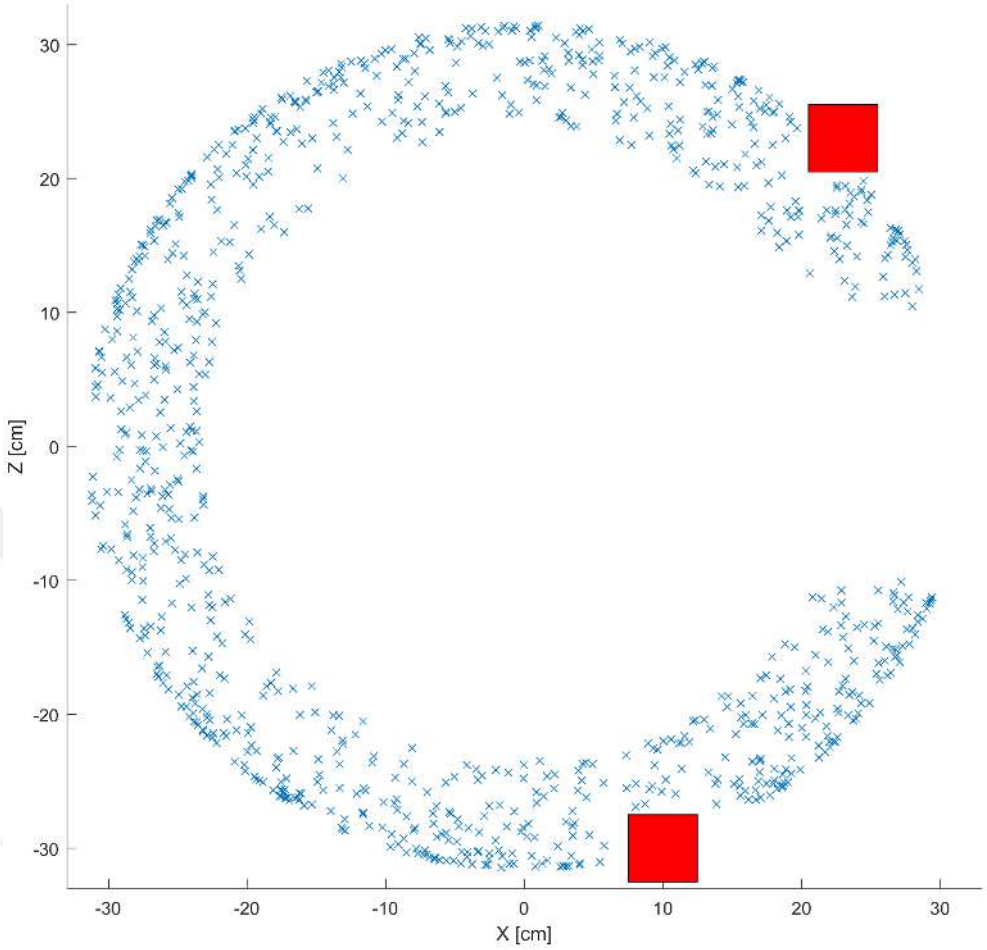


Şekil 6.3 : Her bir eğitim boyunca elde edilen getirilerin histogram dağılımları.

6.2 Eğitilmiş Ajanların Simülasyonu ve Kıyaslanması

Eğitilmiş ajanların simülasyonu için düzgün bir dağılım ile rastgele 1000 adet başlangıç noktası üretilmiştir. Her bir eğitilmiş ajan başlangıç noktaları bu 1000 nokta olacak şekilde simule edilmiş ve simülasyon sonuçları aşağıda verilmiştir. Şekil 6.4'te bu noktaların x-z düzleminde dağılımları gösterilmiştir. Bu simülasyonlarda ajanların eğitim sürecinde olduğu gibi görevi 150 adımda bitirmesi beklenmiş ve maksimum adım sayısı 150 ile sınırlanmıştır. İlk olarak başarı kriteri olarak belirlediğimiz

"madeit" sinyalini 1 üretme durumları incelenmiş, sonuçları çizelge 6.5'te verilmiştir. Ayrıca çizelge 6.6'da engelli çevrede engele çarparak biten bölüm sayıları verilmiştir.



Şekil 6.4 : Simülasyon için kullanılan 1000 noktanın x-z düzleminde dağılımları.

Çizelge 6.5 : Eğitilmiş ajanların simülasyon başarı sayıları.

Çevre Modeli	Başarı Sayıları		
	DDPG	TD3	SAC
Engelsiz Çevre	2	1000	841
Engelli Çevre	0	0	0

Çizelge 6.6 : Engelli çevrede engele çarpma sayıları.

Çevre Modeli	Çarpma Sayıları		
	DDPG	TD3	SAC
Engelli Çevre	8	22	26

Bu verilere baktığımızda DDPG ajanın başarı kriterlerini sağlayamadığı görülmüştür. Diğer iki ajan türünün engelsiz çevrede başarı kriterlerini yerine getirdiği ve fakat engelli çevrede başarı kriterini öğrenemedikleri gözükmektedir. Ek olarak maksimum adım sayısı üç katına çıkarılıp aynı noktalardan denemeler yapıldığında SAC ajan engelsiz çevrede tüm noktalar için başarı kriterini sağlamıştır. Çizelge 6.6 ya baktığımızda ise 1000 bölümdeki çarpma sayıları çok düşük bir oran olarak karşımıza çıkmaktadır. DDPG ajan başarı kriterini tam olarak öğrenmese de engelden kaçınmada diğer ajanlardan daha iyi bir performans göstermiştir. Çizelge 6.7'de ise simülasyonun son adımı olan 150'deki uç işlevcinin koordinatlarıyla hedef koordinatları arasındaki öklidyen uzaklığı hesaplanmış ve bu mesafelerin ortalamaları standart sapma verisiyle birlikte çizelgede verilmiştir.

Çizelge 6.7 : Eğitilmiş ajanların simülasyon hedefe yaklaşma hata ortalamaları.

Çevre	Hata ortalaması ± Standart Sapma [cm]		
Modeli	DDPG	TD3	SAC
Engelsiz Çevre	0.493±0.033	0.099±0.017	0.210±0.112
Engelli Çevre	0.440±0.001	0.864±0.218	1.965±0.240

Hedefe yaklaşma ortalamalarına baktığımızda engelsiz çevrede her ajanın verilen tolerans içerisinde sonuçlar verdiği görülmektedir. Başarı sinyali tablosuna bakıldığında kötü bir performans gösteren DDPG ajanın verilen tolerans 1 cm'den küçük sonuçlar üreterek müsaade edilen mesafe içerisinde kaldığı görülmektedir. Bunun sebebi incelendiğinde ise başarı sinyali için ikinci kriter olan ve uç noktanın hızının 0,5 cm/s den küçük olma kriterini yerine getiremediği görülmüştür. Ajan hedefe yaklaştığında ardışık olarak zıt sinyaller ile hedefin çevresinde bir titreşim hareketi yaptığı ve bu sebeple başarı sinyali üretemediği görülmüştür. Engelsiz çevrede hedefe yaklaşmada en başarılı ajan TD3 olmuştur. Diğer ajanlarda belirtilen toleranstan çok daha iyi bir başarı göstermişlerdir. Engelli çevrede ise DDPG ajan engelsiz çevredekine benzer bir başarı göstermiştir. TD3 ajanın da hata ortalaması büyüse de tolerans içerisinde kaldığı görülmektedir. SAC ajanın ise çok daha kötü bir ortalamaya sahip olduğu görülmektedir. SAC ajanın 150 adımdan daha fazla bir süre çalışmasına müsaade edildiğinde hedefe ulaşabildiğini söylemiştik engelli çevrede hata ortalamasının toleranstan büyük olmasının sebebi bazı bölümlerde 150 adımda

henüz hedefe varamamış olmasından kaynaklanmıştır. Bu hesabı yaparken çarpma sebebiyle biten bölümler hariç tutulmuştur.

Getiri ortalamaları çizelge 6.8'de verilmiştir. Çizelgede görüldüğü üzere DDPG ajan hem engelli hem engelsiz iki çevrede de yakın getiri ortalamaları getirmiş olduğundan benzer performans sergilemiştir. TD3 ve SAC ajanların ise ortalama getirisi engelli çevrede belirgin bir şekilde düşmüştür.

Çizelge 6.8 : Simülasyon getiri ortalamaları.

Çevre Modeli	Getiri ortalaması		
	DDPG	TD3	SAC
Engelsiz Çevre	-13.2625	-2.8497	-5.1810
Engelli Çevre	-14.9632	-20.5401	-19.7214

Çizelge 6.9'de simülasyon esnasında bölümlerdeki adımların ortalamaları verilmiştir. PÖ'de azaltma faktörü ile görevin en hızlı bir şekilde bitirilmesi de teşvik edilmektedir. DDPG ajanının hedefe toleranstan daha yakın yaklaşabilmesi fakat bitirmeyi öğrenememesinden ötürü maksimum adım sayısı yazılmıştır. TD3 ajanın da engelsiz çevrede SAC ajana göre daha hızlı bir şekilde bölümleri bitirdiği görülmüştür.

Çizelge 6.9 : Simülasyon bölüm bitirme adım sayıları ortalaması.

Çevre Modeli	Adım Sayıları Ortalaması		
	DDPG	TD3	SAC
Engelsiz Çevre	149.8±4.8	64.97±11.5	101.6±31.5
Engelli Çevre	148±0.13	146±0.21	145±0.22

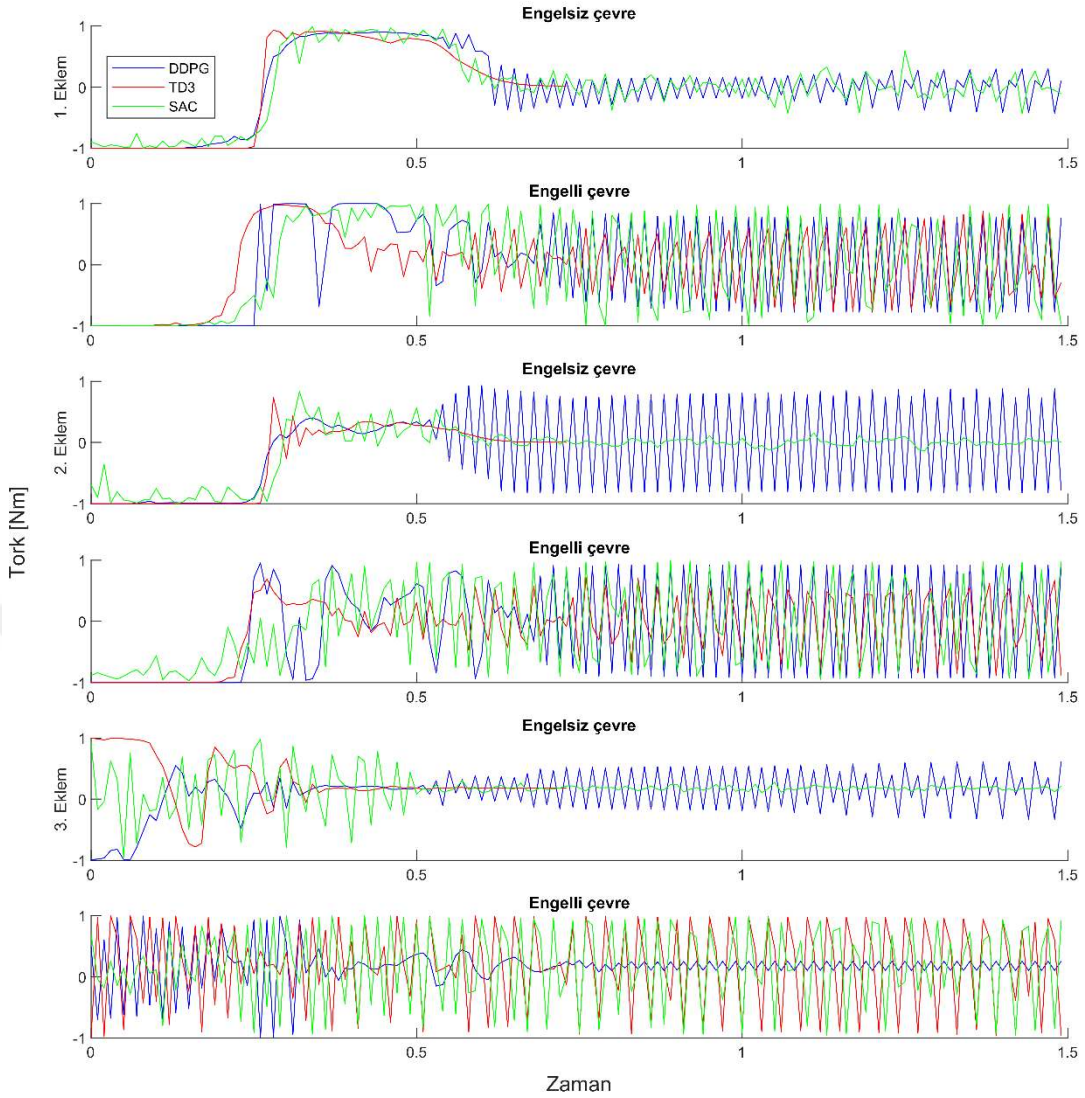
Tüm bu sonuçlara bakıldığında DDPG ajan üzerindeki iyileştirmeler ile geliştirilen TD3 ajanın iyi bir performans gösterdiği ve bu tarz kontrol problemlerinde daha iyi sonuçlar vereceği düşünülmektedir. Engelsiz çevrede TD3 ajanın hedefe en çok yaklaşan, en hızlı şekilde bölümleri bitiren ajan olduğu açıkça görülmüştür. SAC ajanın ise nispeten daha düşük performans göstermesi stokastik bir aktör olmasının bir sonucu olabileceği düşünülmektedir. Her ne kadar eğitim süreçlerinde SAC ajan ön plana çıkmış olsa da simülasyon sonuçları TD3 ve DDPG ajanın bu tarz kontrol problemlerine daha uygun olabileceği kanaati oluşturmuştur. Engelli çevrede ise tüm ajanların performansında düşüş olmuştur. Bunun sebeplerinden ilk ise çarpma ile

erken biten bölümler sebebiyle öğrenmenin zorlaşmasıdır. Engelden en az etkilenen her iki çevrede de benzer hata ve getiri sonuçlarıyla DDPG ajan olmuştur. SAC ajanın stokastik yapısı engelli çevrede dezavantaj oluşturmaktadır. Aynı başlangıç noktasından yapılan tekrarlı simülasyonlarda SAC ajanın rota değişiklikleri ve bunun sonucunda bazen engele çarpmazken bazen engele çarptığı görülmüştür.

6.2.1 Simülasyon eylem grafikleri

Bu bölümde eğitilmiş ajanların aynı noktalardan eylem tamamlanana kadar ürettiği sinyallerin grafikleri engelli ve engelsiz çevrede verilecek ve karakteristikleri anlaşılmaya çalışılacaktır. Bu bölümde Şekil 6.4'te aynı noktadan başlamak üzere her ajanın bölüm boyunca eklemlere ürettiği sinyaller engelsiz ve engelli çevreler için Şekil 6.5'te verilmiştir. Birkaç farklı nokta için kıyaslama grafikleri ek B'de verilmiştir.

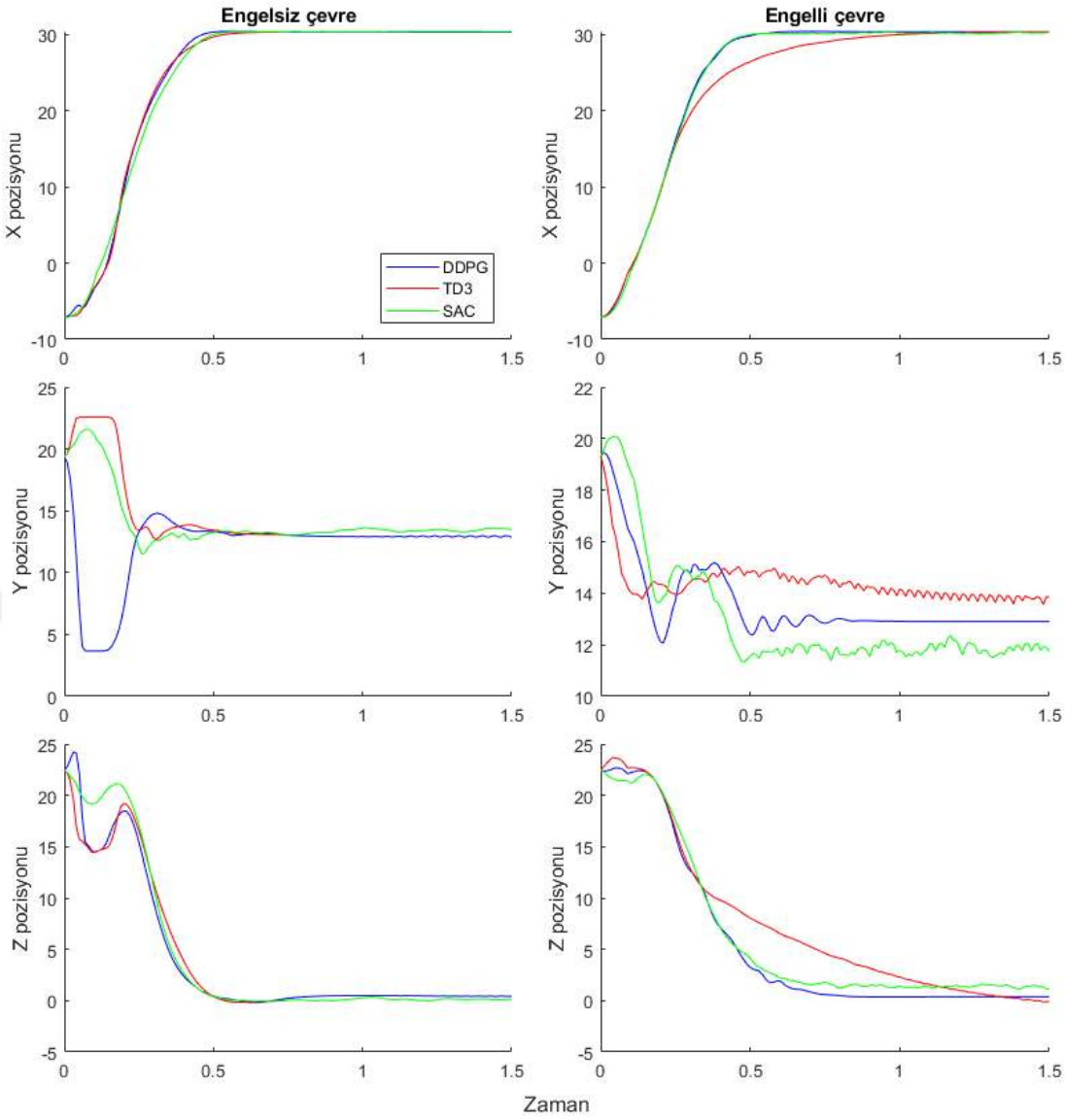
Birinci ve ikinci eklemin sinyalleri hedefe varıldığında sıfır olması istenirken üçüncü eklemde yer çekimine karşı koymak için sıfırdan büyük bir sinyal üretilmesi istenmektedir. Şekil 6.5'te TD3 ajanın engelsiz çevredeki sinyal çizgilerine bakıldığında bunu başardığı görülmektedir. DDPG ajan ise hem engelsiz hem de engelli çevrede salınımlı bir sinyal üretmiştir. SAC ajan ise engelsiz çevrede hedefe yaklaştıkça DDPG'ye nazaran daha düşük bir salınım göstermiştir. Engelli çevrede tüm ajanların hedefe yaklaştıktan sonra salınım yaptıkları ve sinyalleri istenilen değerlere indirgeyemedikleri görülmüştür. Bu salınımlar işlevci ucun titreşmesine sebebiyet vermiş ve bu titreşim bölüm bitirmek için gerekli olan başarı sinyali üretimini engellemiştir.



Şekil 6.5 : Bir bölümde eklemlere uygulanan torkların zamana bağlı grafikleri.

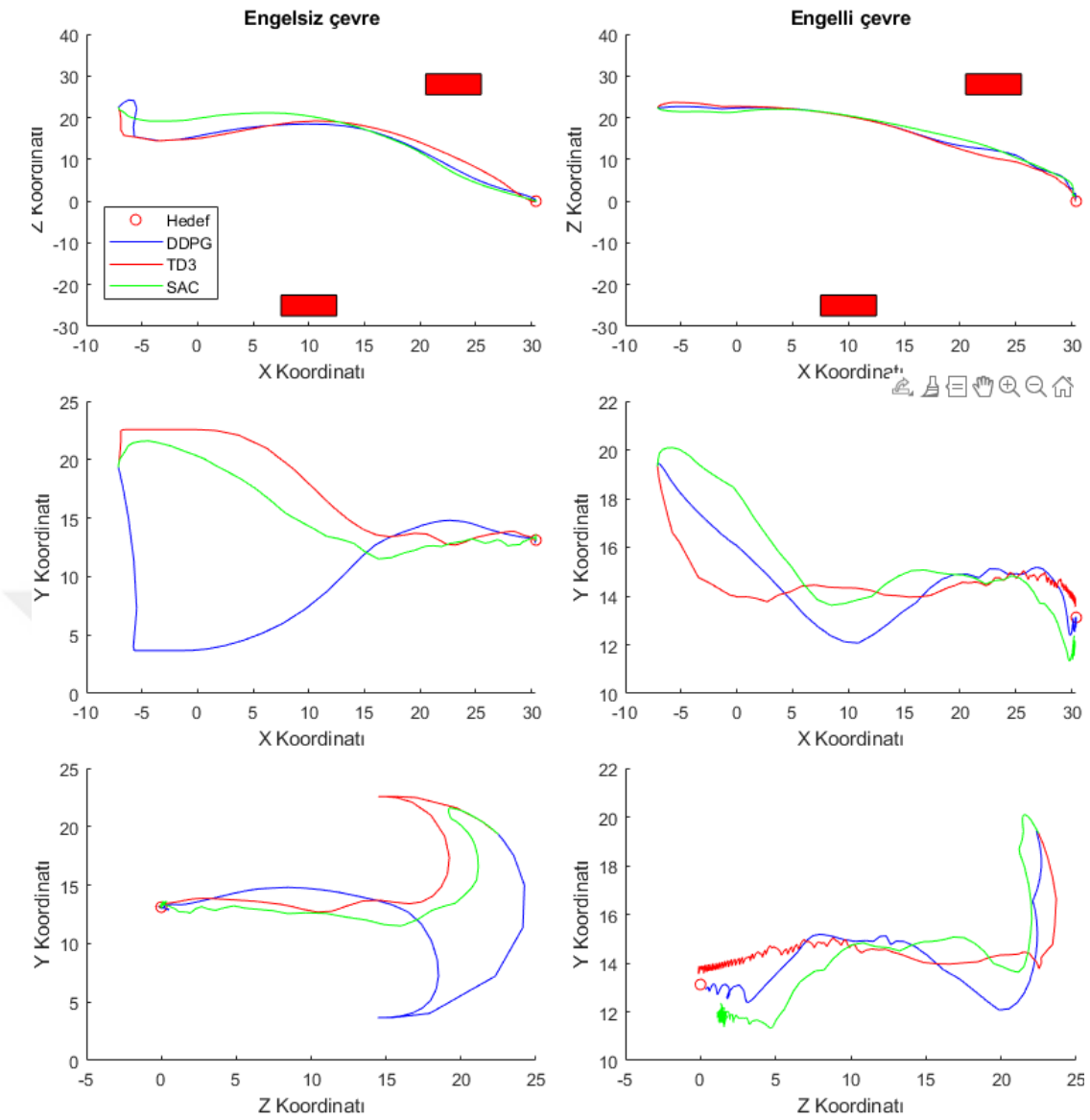
6.2.2 Simülasyon konum grafikleri

Bu bölümde bir noktadan hedefe giderken işlevci ucun zamana bağlı konum grafikleri ve iki boyutlu düzlemlerde izlediği yollar verilmiştir. Bu grafiklerde başlangıç noktası koordinatları x_0=-7.1, y_0=19.3, z_0=22.5 hedef noktasının koordinatları x_h=30.4, y_h=13.1 ve z_h=0'dır. Şekil 6.6'te her bir ajan için işlevci ucun koordinatlarının zamana bağlı grafikleri verilmiştir.



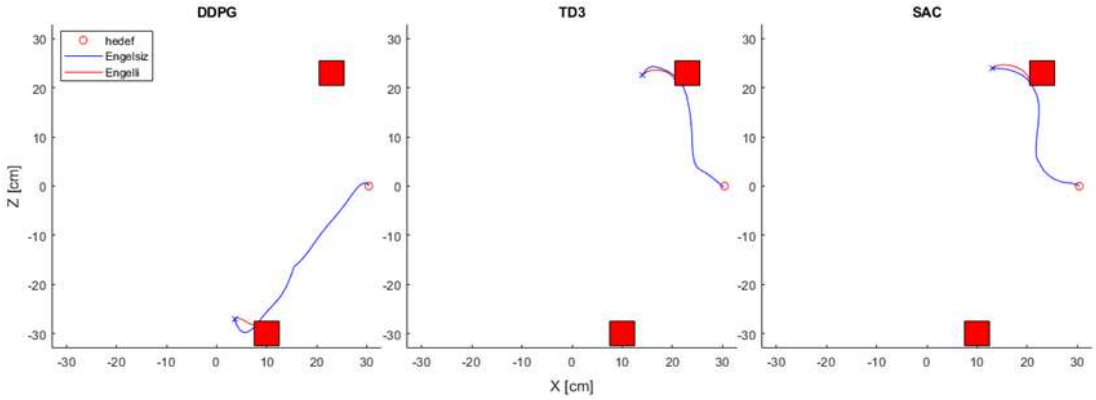
Şekil 6.6 : İşlevci ucun koordinatlarının zamana göre grafiği.

Bir önceki bölümde üretilen sinyallerin grafiklerini incelemiştik. Şekil 6.6'te bu üretilen sinyallerin işlevci ucun hareketine etkileri gözükmektedir. Engelsiz çevrede TD3 ajan konum hedeflerine iyi bir şekilde yakınsamış ve belirlenen toleransta durmuştur. Diğer ajanlar ise sinyal üretmeye devam ederek işlevci ucun titreşmesine sebebiyet vermiştir. Bu titreşim y ekseninde daha belirgin olarak görülmektedir. Engelli çevrede ise hedef koordinatlara en iyi yakınsamayı DDPG ajanın yaptığı görülmektedir. Tüm ajanlar hedef nokta etrafında titreşmişler ancak DDPG ajanın titreşimi daha küçük bir mesafede kalıp grafikte belli olmamıştır. Diğer ajanların ise özellikle y eksenindeki titreşimi belirgin olarak görülmektedir. Şekil 6.7'da ise hedefe giderken ajanların izledikleri yollar verilmiştir.

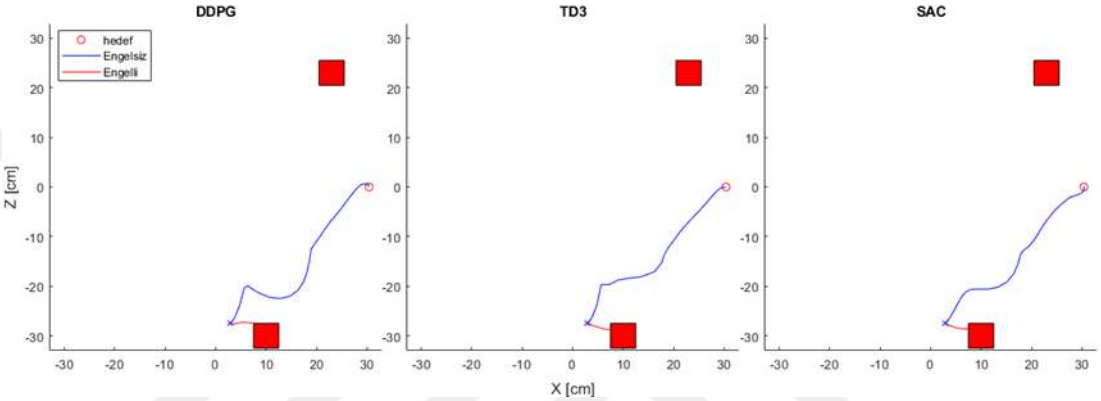


Şekil 6.7 : İşlevci ucun eksenlerde izlediği güzergâh sonuçları.

Şekil 6.7'da x-z düzleminde görsel olarak engeli anlamak için grafiklere engeller eklenmiştir. Şekil 6.8'de her bir ajanın engelli ve engelsiz olarak aynı noktalardan benzer güzergahlar izlemeye çalışması sonucu engele çarptığı noktalara örnekler verilmiştir. Şekilde görüleceği üzere engelli çevrede eğitilmiş ajan engelsiz ajan ile benzer güzergahı izlemek istemekte bunun sonucunda engele çarpıp bölümü erken bitirmektedir. Şekil 6.9'da aynı noktadan başlayan ve her ajanın da engele çarptığı örnek bölüm verilmiştir.

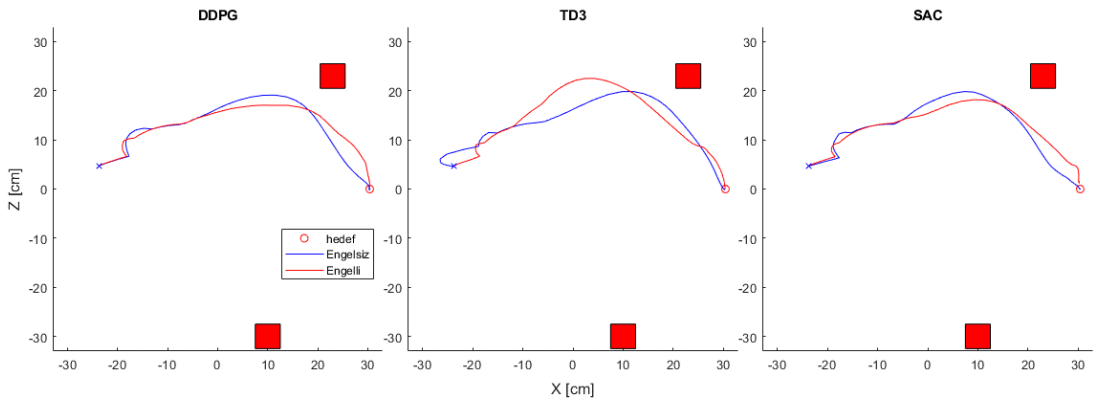


Şekil 6.8 : Engele çarpan güzergahların x-z düzlemi grafik örnekleri.

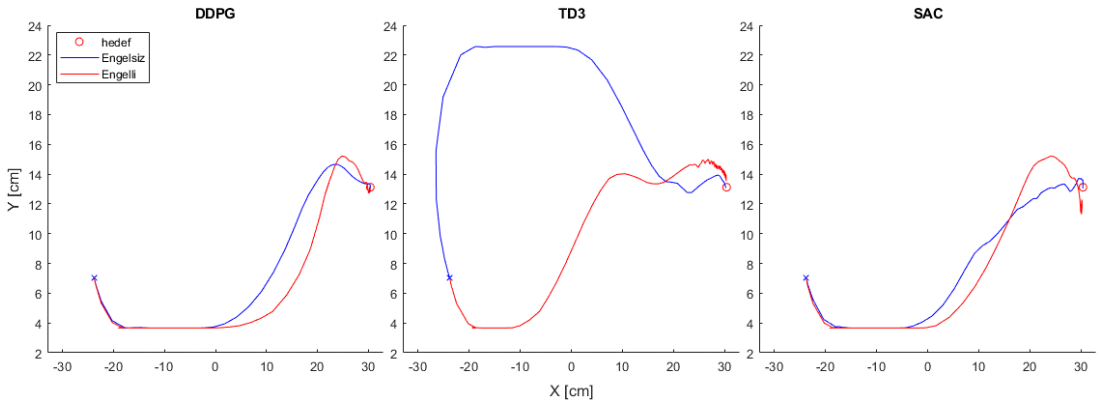


Şekil 6.9 : Tüm ajanların engele çarptığı örnek bir noktaya ait güzergahların x-z düzlem grafikleri.

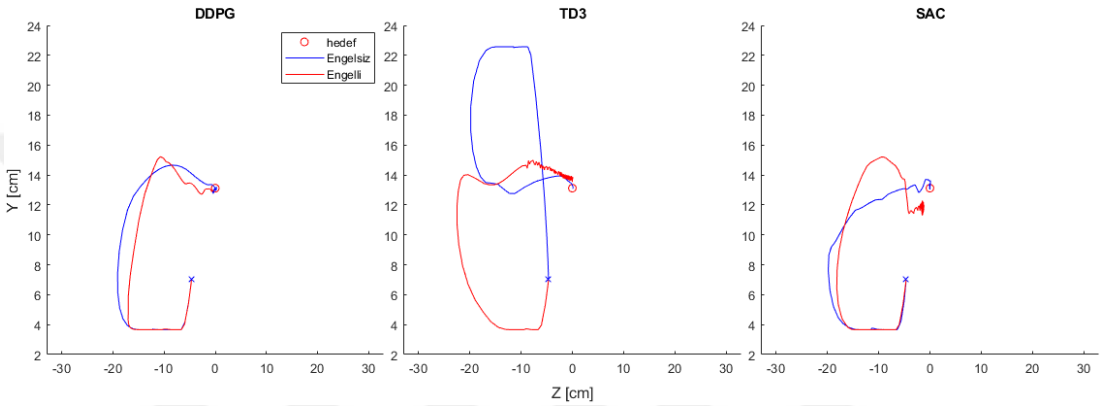
Şekil 6.9'a baktığımızda engele yakın başlayan bölümlerde engele hemen çarpma sonucu bölüm kısa sürmüş o çevreden yeterince örnek alınamamış ve öğrenme efektif olmamıştır. Bu sebeple engelli çevrede eğitimini tamamlamış ajanlar simülasyon yapılırken engele yakın çevrelerde başarısız olmuştur. Şekil 6.10'a bakıldığında ise tüm ajanların çevre ile yeterli bilgiyi toplayıp hedefe iyi bir şekilde yaklaştığı görülmüştür. Şekil 6.10'de x-z grafiği verilen noktanın x-y düzlemindeki grafiği Şekil 6.11'de verilmiştir. Şekil 6.12'de ise aynı noktanın y-z grafiği verilmiştir.



Şekil 6.10 : Aynı noktadan simülasyon sonucu güzergahlarının x-z grafiği.



Şekil 6.11 : Aynı noktadan simülasyon sonucu güzergahlarının x-y grafiği.



Şekil 6.12 : Aynı noktadan simülasyon sonucu güzergahlarının z-y grafiği.

Güzergâh grafikleri incelendiğinde DDPG ve SAC ajanların izledikleri güzergahların hem engelli ve engelsiz çevrede hem de birbirleriyle benzerlikleri dikkat çekmiştir. İzlenen yol ve engele çarpma ilişkisi x-z grafiklerinde daha iyi takip edilmiştir. X ve z eksenlerindeki hedeflere ajan daha kolay bir şekilde ulaşmış ve titreşim epey azdır fakat y koordinatında hedefe ulaşmakta güçlük çektiği görülmektedir. Bunun sebebi ise y ekseninde sürekli karşı koyması gereken bir dış kuvvet yer çekimi bulunmasıdır.

7. SONUÇ VE ÖNERİLER

Bu çalışmada, otonom olarak robot kol yörünge kontrolü ele alınmıştır. Akıllı kontrolörler tasarlamak için modelden bağımsız, aktör-kritik tabanlı ve politika-dışı pekiştirmeli öğrenme yöntemleri kullanılmıştır. Bu özelliklerde en gelişmiş üç algoritma olan DDPG, TD3 ve SAC algoritmaları simülasyon ortamında karşılaştırılmıştır. Engelsiz ve engelli olarak iki çevre modellenerek çarpma gibi bir bozucu girişin eğitim sürecine ve eğitilmiş ajanların performansına etkisi gözlemlenmiştir.

Kontrol için Matlab kütüphanesinde bulunan 3 serbestlik dereceli bir robot kol seçilmiştir. Kontrol problemi her bölüm başında rastgele bir noktadan başlayan robot kolun hedef olarak belirlenen noktaya ulaşması olarak belirlenmiştir. Algoritmalar Matlab "Pekiştirmeli Öğrenme Araçları (Reinforcement Learning Toolbox)" modülü araçlığıyla modellenmiş ve Simulink/Simscape ortamında simule edilmiştir.

Tüm aktör-kritik algoritmalar çevreden gelen ödül geri bildirimi ve durum bilgilerini kullanarak hedefe nasıl yaklaşacaklarını öğrenmişlerdir. Engelsiz çevre, algoritmaları eğitim süreçleri ve eğitilmiş ajanların gösterdikleri performansları kıyaslamak bakımından daha iyi sonuçlar vermiştir. Engelli çevre ise çarpma bozucu girişi ile eğitim sürecine olumsuz etki yapmış, ajanların çevre hakkında yeterli ve sağlıklı bilgi toplayamamalarına sebep olmuştur. Bu yetersiz eğitim sürecinde her ne kadar ajanlar hedefe yaklaşmayı öğrenmişlerse de onlardan beklenen hedefe varıp orada durmak yerine hedef etrafında salınım hareketi göstermişlerdir. Bu gibi engelden kaçınma problemleri için farklı geliştirmelere ihtiyaç olduğu açıkça görülmüştür.

Eğitim süreçlerinde engelsiz çevrede SAC ve TD3 algoritmaları benzer performanslar sergileyip daha az pürüzlü ve yüksek gelirli bir süreç sergilemişlerdir. DDPG ise getirilerde daha çok salınım göstererek ortalamada diğer algoritmalardan gözle görünür bir şekilde az getiriye ulaşmıştır. Eğitim süreci en hızlı biten algoritma ise DDPG olmuştur. SAC ve TD3 algoritmalar değişen YSA yapıları ile daha çok parametreyi hesaplamak durumunda kalmışlardır.

Eğitilmiş ajanların simülasyonunda ise engelsiz çevrede algoritmalar arasında deterministik olan DDPG ve TD3 algoritmaları -DDPG'nin hedefe çok yakın ve ufak bir salınım yapmasını başarı kabul edersek-en az adımla en hızlı şekilde hedefe ulaşan algoritmalar olmuştur. SAC ise hedefe varmayı öğrenmiş fakat stokastik yapısından

olsa gerek hedefe daha yavaş giden algoritma olmuştur. DDPG ajanın hedef etrafında yapmış olduğu titreşim hareketi algoritmanın hiper parametreleri optimize edilerek giderilebilir. Bu da iddia edildiği gibi DDPG'nin parametre bağımlılığını göstermektedir.

Engelli çevrede yapılan simülasyonlarda SAC algoritma stokastik yapısından ötürü engele yakın noktalarda farklı davranışlar sergileyebilmektedir. Gerçekleştirilen eylemlerin sonuçlarının çarpma gibi, hasar verici veya geri dönüşü olmayan durumların bulunduğu kontrol problemlerinde kullanmanın risk oluşturduğu görülmüştür. Bunun gibi kontrol problemlerinde deterministik algoritmalar daha güvenilir gözükmektedir.

Modelden bağımsızlığı ile ön plana çıkan bu algoritmalar ile dinamik haritalama yaparak robot kolun durumunu ve engelin yerini belirlemek için gerçekçi sensör girdileri kullanan daha gerçekçi simülasyon ortamlarında gelecekteki çalışmalar yapılabilir. Bu sinyaller ile çevre hakkında bilgi edinen algoritmalar için bu geliştirmelerin elzem olduğu görülmektedir. Engellerden kaçınma problemi için statik engeller gibi dinamik engeller de eklenebilir. Ve bu algoritmalar gerçek dünya uygulamalarıyla kıyaslanabilir.

KAYNAKLAR

[1] **Kayışlı, K., & Uğur, M.** (2017). 3 Serbestlik Dereceli Bir Robot Kolun Bulanık Mantık ve PID ile Kontrolü. Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji.

[2] **Yumurtacı, S., & Mert, T.** (2003). Robotik kaynak sistemleri ve gelişme istikametleri. Mühendis ve Makina, 44(526), 32-40.

[3] **Kormushev, P., Calinon, S., & Caldwell, D. G.** (2013). Reinforcement learning in robotics: Applications and real-world challenges. Robotics, 2(3), 122-148.

[4] **Sutton, R. S., & Barto, A. G.** (2018). Reinforcement learning: An introduction. MIT press.

[5] **Milecki, A & Owczarek, P.** (2019). The application of a vision system to detect trajectory points for soldering robot programming. Advances in Intelligent Systems and Computing, vol. 835, pp. 587–596.

[6] **Kubacki, A & Milecki, A.** (2019). Control of the 6-axis robot using a brain-computer interface based on steady state visually evoked potential (SSVEP). Lecture Notes in Mechanical Engineering, pp. 213–222.

[7] **Mahadevan, S**. (1996). Machine Learning for Robots: A Comparison of Different Paradigms. Machine Learning, pp. 1–14.

[8] **Schaal, S & Atkeson, C. G.** (2002). Robot learning by nonparametric regression. vol. 1994, no. Iros 1994, pp. 478–485.

[9] **Atkeson, C. G. & Schaal, S.** (1997). Learning tasks from a single demonstration. Proceedings - IEEE International Conference on Robotics and Automation, vol. 2, no. April, p. 1706.

[10] **Sutton, R. S. & Barto, A. G.** (2018). Reinforcement Learning: An Introduction. The MIT Press.

[11] **Özakar, R.** (2016). Pekiştirmeli Öğrenme Yöntemiyle Farklı Görevlerin Robotlara Öğretilmesi. Atatürk Üniversitesi.

[12] **Engin, C.D.** (2019). Control of Autonomous Mobile Robots Using Reinforcement Learning. Yıldız Teknik Üniversitesi.

[13] **Yamaç, F.** (2020). Kablo ile Sürülen Paralel Robotların Model Tabanlı ve Pekiştirmeli Öğrenme ile Konum Denetimi. Karadeniz Teknik Üniversitesi.

[14] **Kober, J., Bagnell, J. A. & J. Peters**. (2013). Reinforcement learning in robotics: A survey. International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274.

[15] **Franceschetti, A., Tosello, E., Castaman, N & Ghidoni, S**. (2020). Robotic arm control and task training through deep reinforcement learning. arXiv, no. January.

[16] **Samuel, A. L.** (1969). Some studies in machine learning using the game of checkers. II-Recent progress," Annual Review in Automatic Programming, vol. 6, no. PART 1, pp. 1–36.

[17] **Alpaydın, E.** (2010). Introduction to Machine Learning Second Edition, 2nd ed. The MIT Press.

[18] **Url-1** <https://uk.mathworks.com/discovery/reinforcement-learning.html>, erişim tarihi 01.08.2022

[19] **Huang, H., Yang, Y., Wang, H., Ding, Z., Sari, H. & Adachi, F**. (2020). Deep reinforcement learning for UAV navigation through massive MIMO technique. IEEE Transactions on Vehicular Technology, vol. 69, no. 1, pp. 1117–1121, Jan.

[20] **Azar, A. T., Koubaa, A., Ali Mohamed, N., Ibrahim, H. A., Ibrahim, Z. F., Kazim, M., ... & Casalino, G.** (2021). Drone deep reinforcement learning: A review. Electronics, 10(9), 999.

[21] **Puterman, M. L.** (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.

[22] **Howard, R. A.** (1960). Dynamic programming and markov processes. New York: The Technology Press of The Massachusetts of Technology.

[23] **Bellman, R.** (1954). The theory of dynamic programming. Bulletin of the American Mathematical Society, 60(6), 503-515.

[24] **Watkins, C. J. C. H.** (1989). Learning from delayed rewards.

[25] **Watkins, C. J., & Dayan, P.** (1992). Q-learning. Machine learning, 8(3), 279-292.

[26] **McCulloch, W. S., & Pitts, W.** (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[27] **Da Silva, I. N., Spatti, D. H., Flauzino, R. A., Liboni, L. H. B., & dos Reis Alves, S. F.** (2017). Artificial neural networks. Cham: Springer International Publishing, 39.

[28] **Bottou, L., & Bousquet, O.** (2007). The tradeoffs of large scale learning. Advances in neural information processing systems, 20.

[29] **Amari, S. I.** (1993). Backpropagation and stochastic gradient descent method. Neurocomputing, 5(4-5), 185-196.

[30] **François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J.** (2018). An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4), 219-354.

[31] **Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013).** Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[32] **Lin, L. J.** (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3), 293-321.

[33] **Williams, R. J., & Peng, J.** (1991). Function optimization using connectionist reinforcement learning algorithms. Connection Science, 3(3), 241-268.

[34] **Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D.** (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[35] **S. Fujimoto, H. van Hoof, and D. Meger**. (2018). Addressing function approximation error in actor-critic methods. In International conference on machine learning (pp. 1587-1596). PMLR.

[36] **Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., ... & Levine, S.** (2018). Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.

[37] **Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K.** (2008). Maximum entropy inverse reinforcement learning. In Aaai, Vol. 8, pp. 1433-1438.

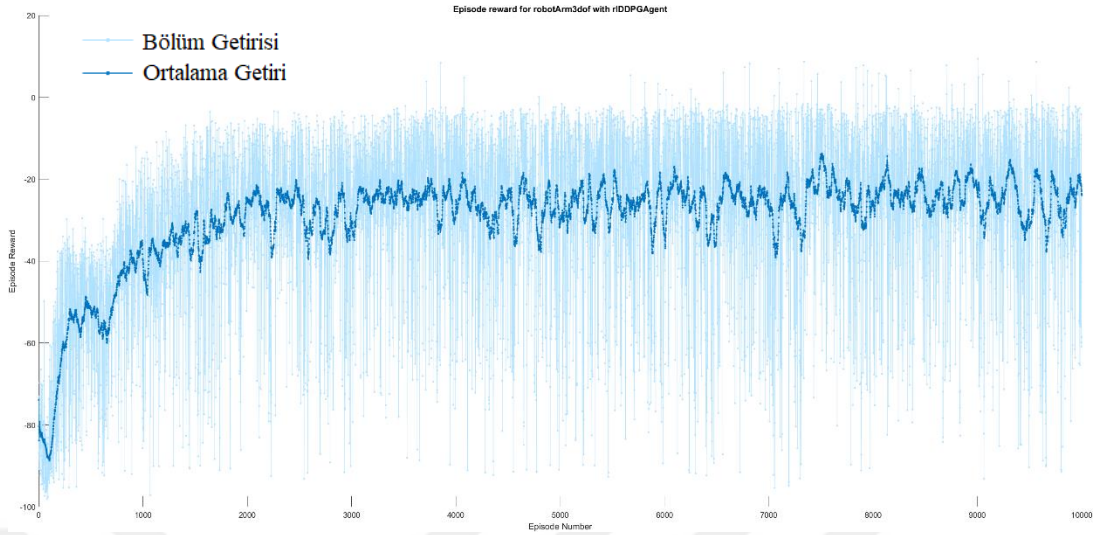
[38] **URL-2** <https://uk.mathworks.com/help/physmod/sm/ug/modeling-a-robot-using-step-files.html>, erişim tarihi 03.08.2022.

EKLER

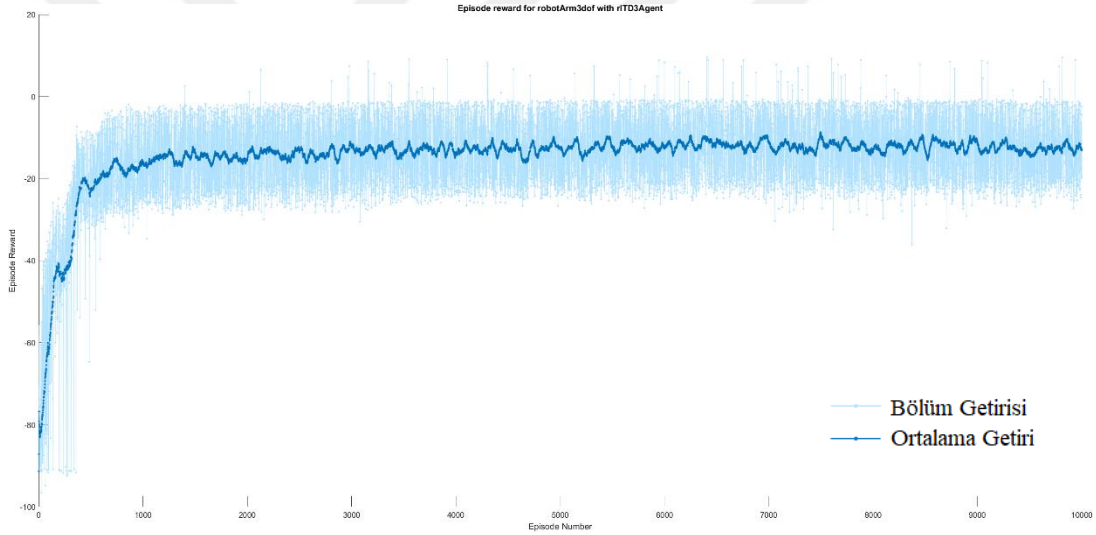
EK A : Eğitim Süreci Grafikleri

EK B : Eğitilmiş Ajanlarla Gerçekleştirilen 1000 Simülasyondan Örnek 3 Bölümün Simülasyon Sonucu Grafikleri

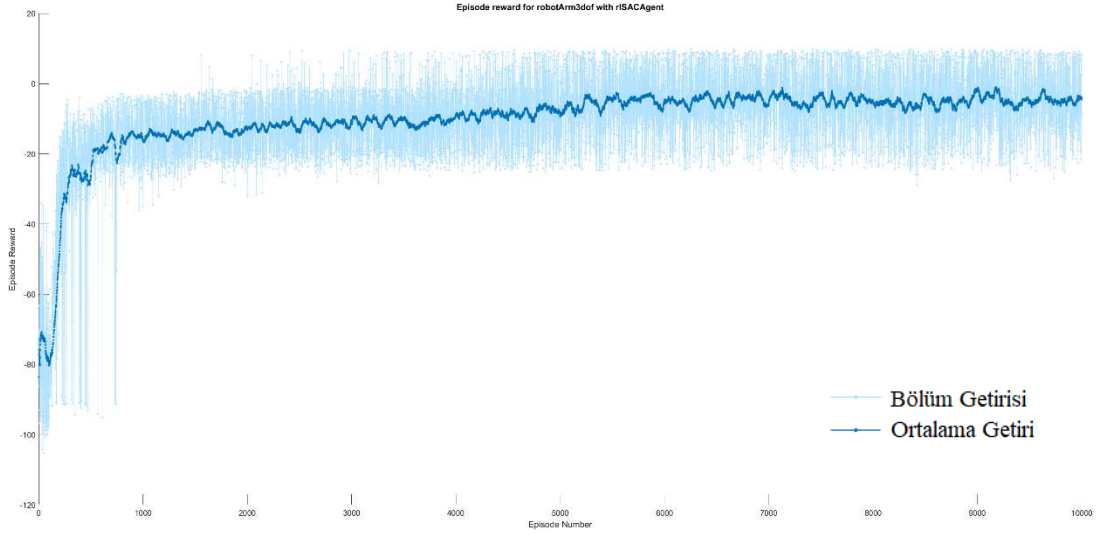
EK A



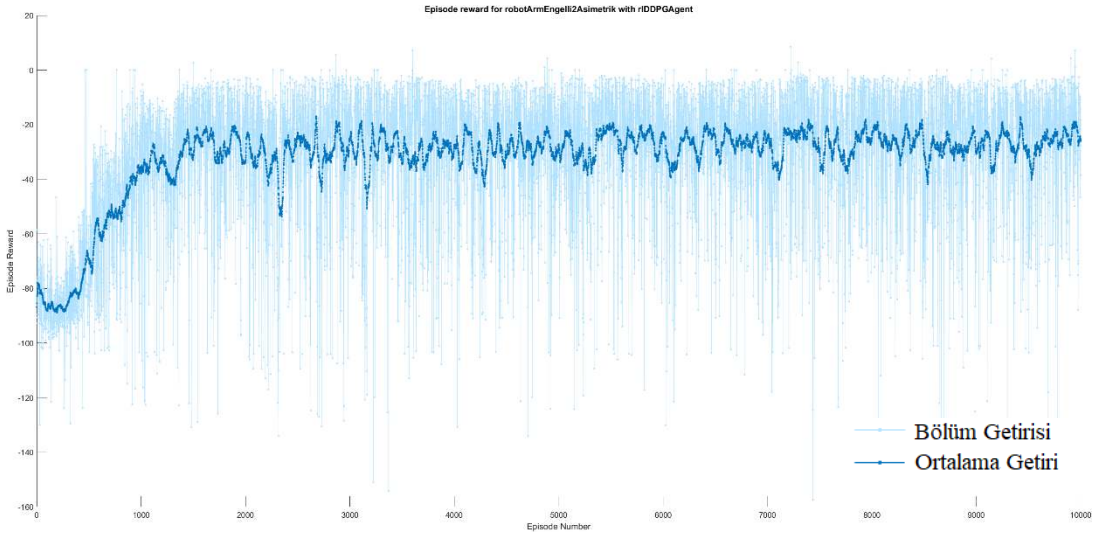
Şekil A.1 : DDPG ajanın engelsiz çevrede eğitim grafiği.



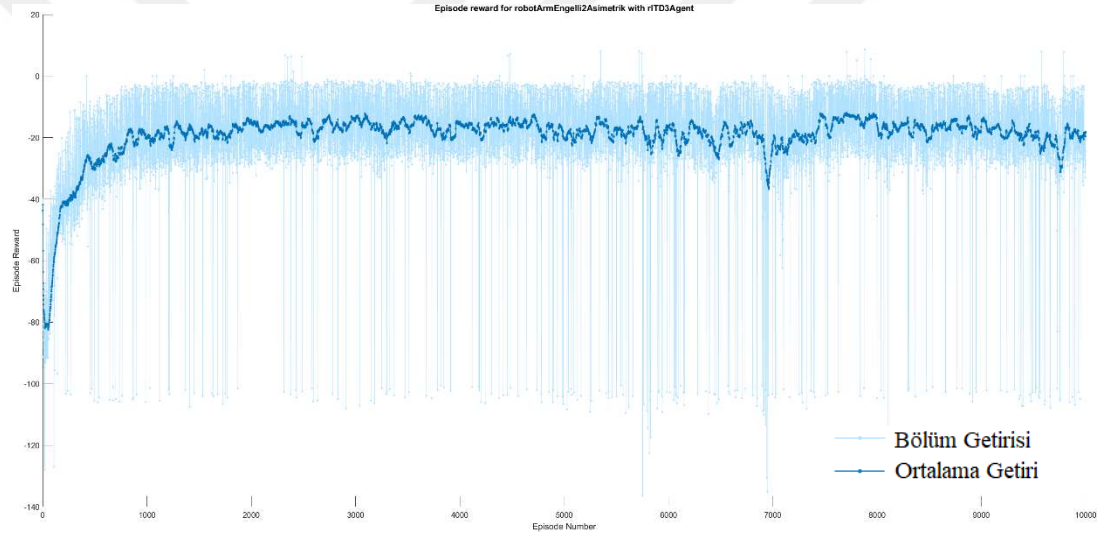
Şekil A.2 : TD3 ajanın engelsiz çevrede eğitim grafiği.



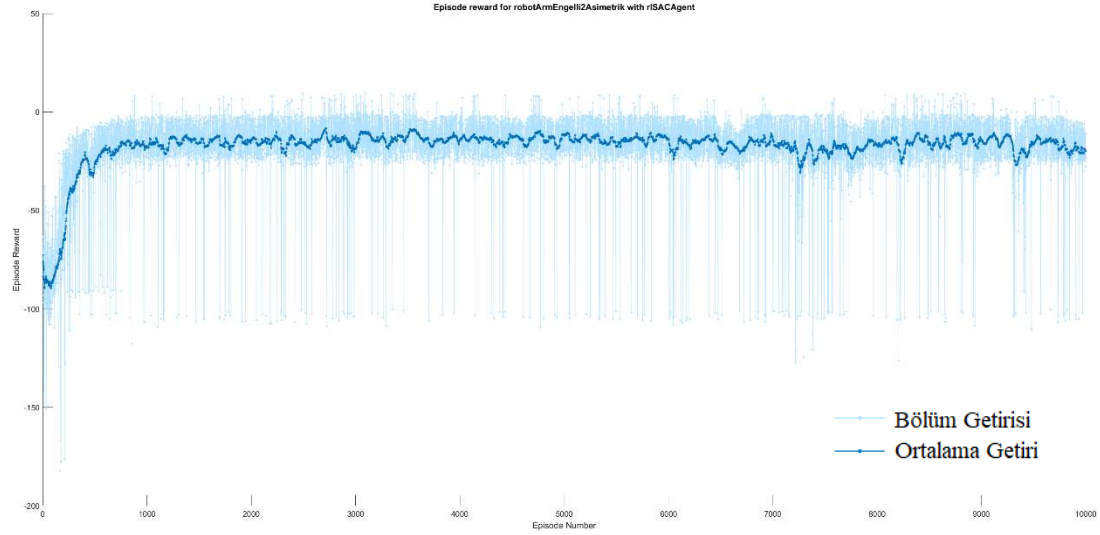
Şekil A.3 : SAC ajanın engelsiz çevrede eğitim grafiği.



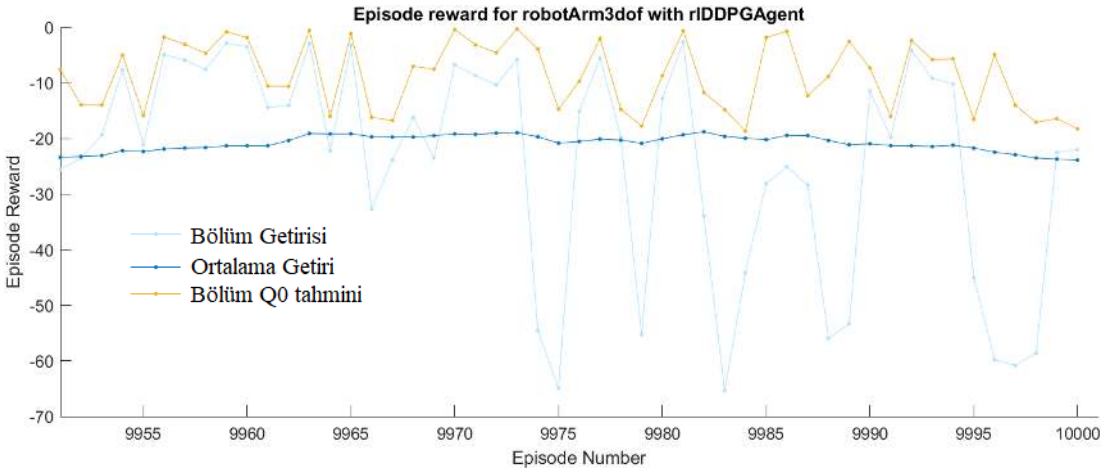
Şekil A.4 : DDPG ajanın engelli çevrede eğitim grafiği.



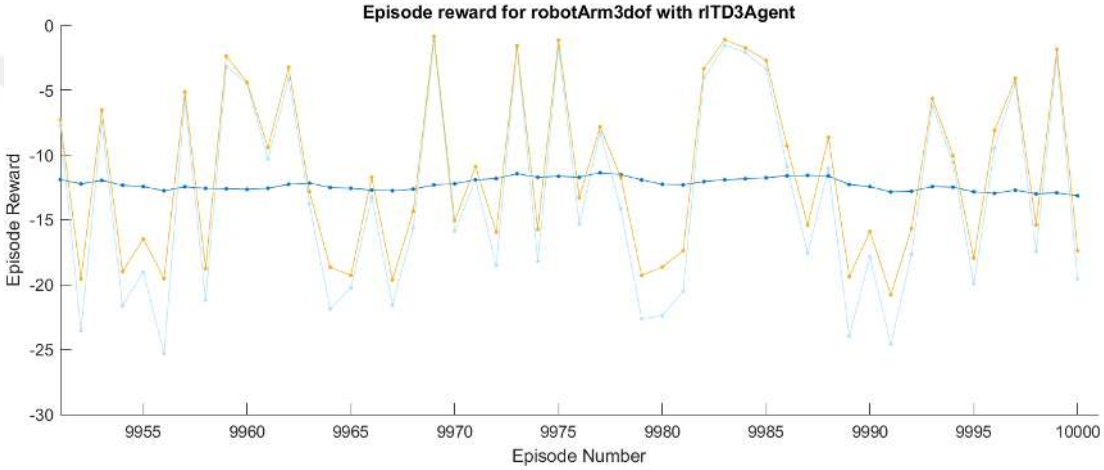
Şekil A.5 : TD3 ajanın engelli çevrede eğitim grafiği.



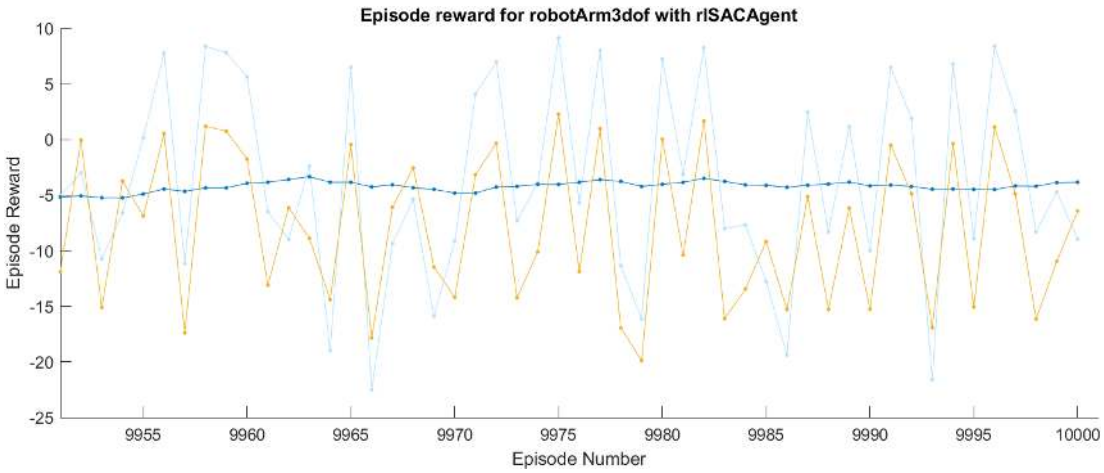
Şekil A.6 : SAC ajanın engelli çevrede eğitim grafiği.



Şekil A.7 : DDPG ajan eğitim sürecinde son 50 bölüm için bölüm Q0 grafiği.

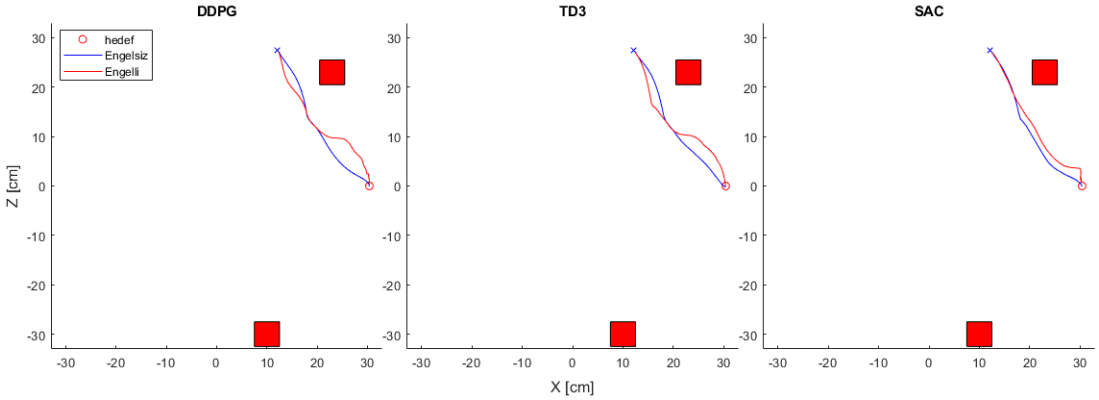


Şekil A.8 : TD3 ajan eğitim sürecinde son 50 bölüm için bölüm Q0 değeri grafiği.

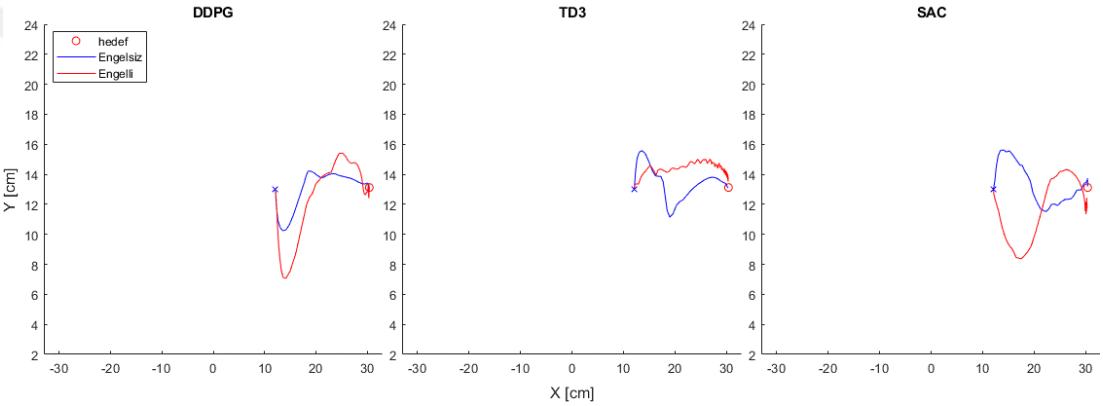


Şekil A.9 : SAC ajan eğitim sürecinde son 50 bölüm için bölüm Q0 değeri grafiği.

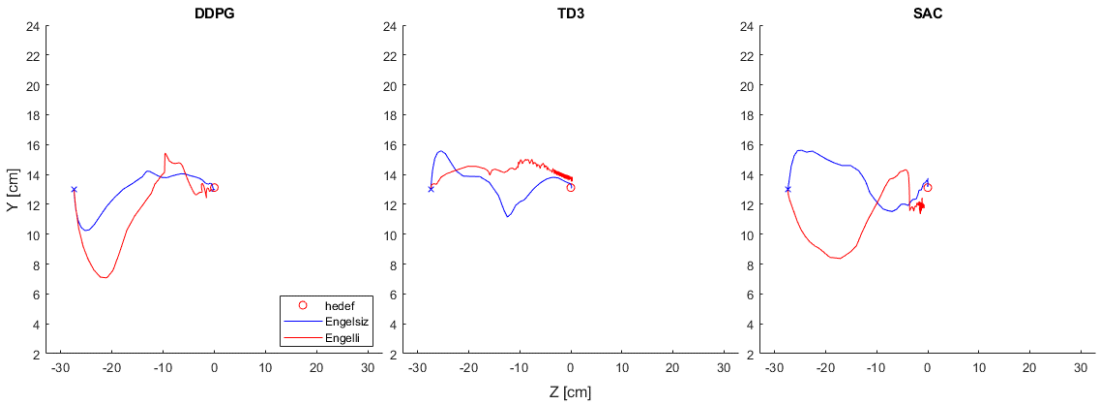
EK B



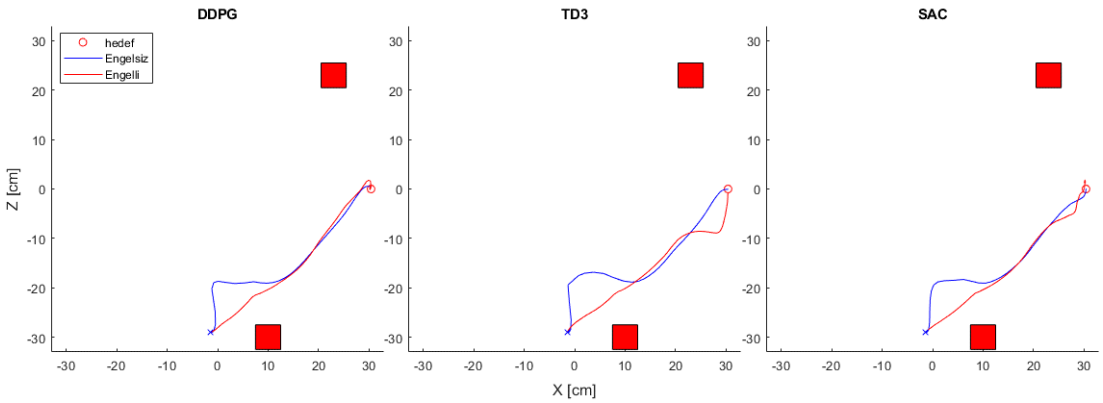
Şekil B.1 : 1 numaralı bölümde izlenen güzergahın x-z düzlem grafiği.



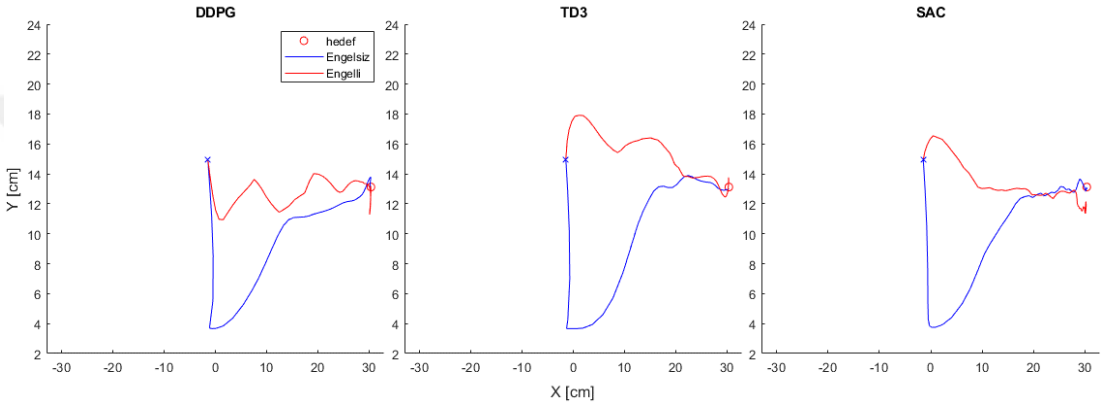
Şekil B.2 : 1 numaralı bölümde izlenen güzergahın x-y düzlem grafiği.



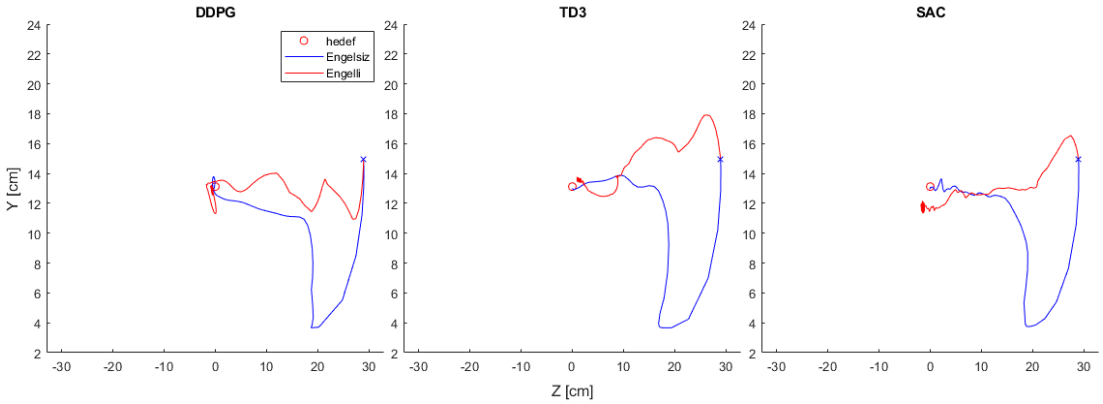
Şekil B.3 : 1 numaralı bölümde izlenen güzergahın z-y düzlem grafiği.



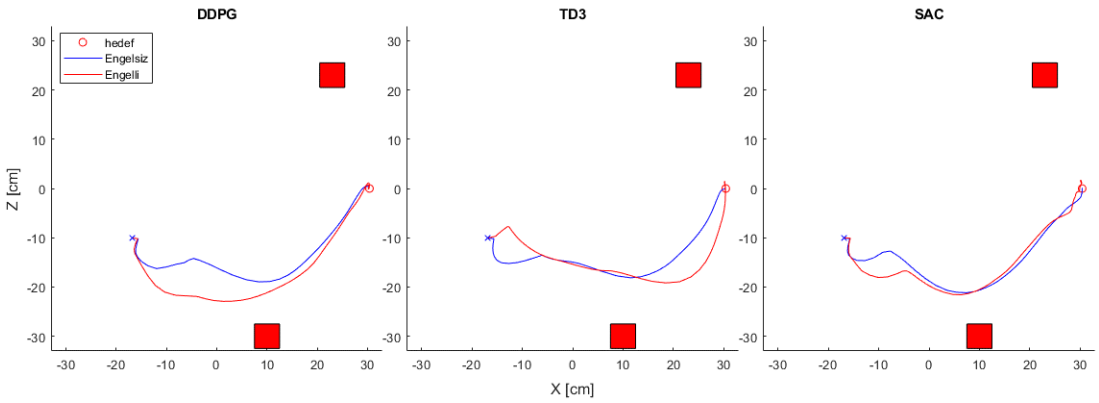
Şekil B.4 : 2 numaralı bölümde izlenen güzergahın x-z düzlem grafiği.



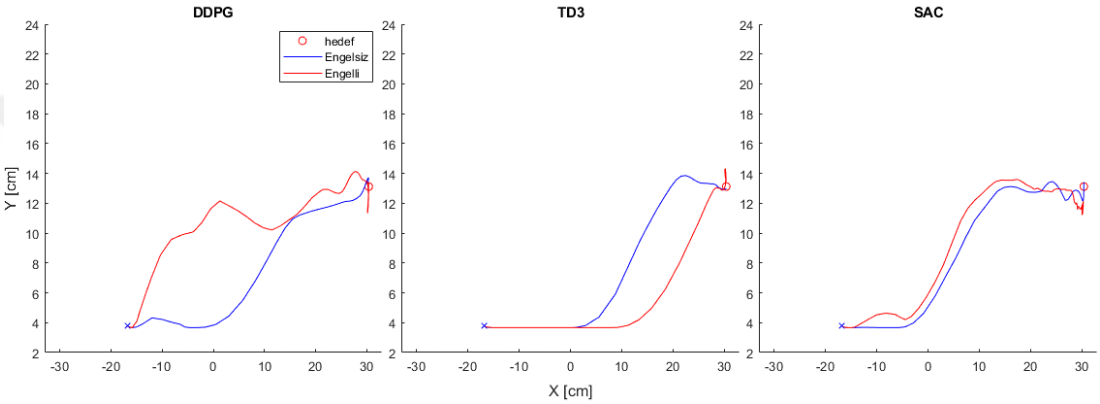
Şekil B.5 : 2 numaralı bölümde izlenen güzergahın x-y düzlem grafiği.



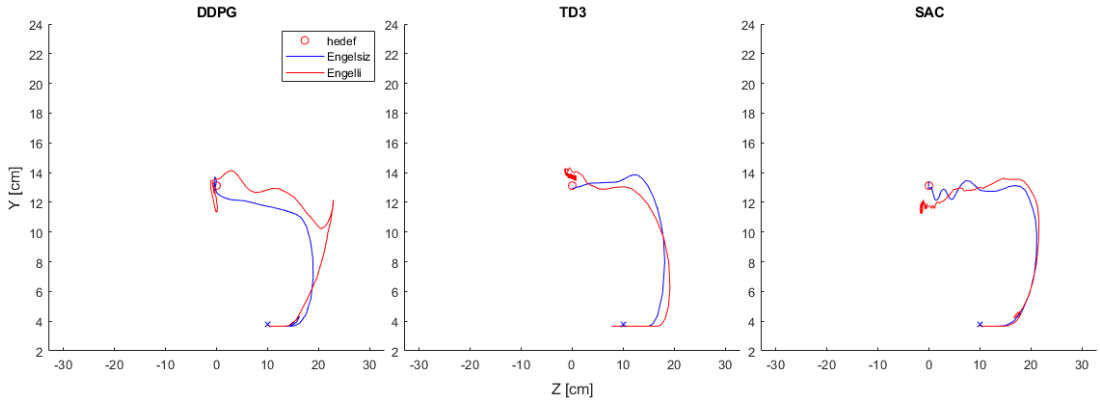
Şekil B.6 : 2 numaralı bölümde izlenen güzergahın z-y düzlem grafiği.



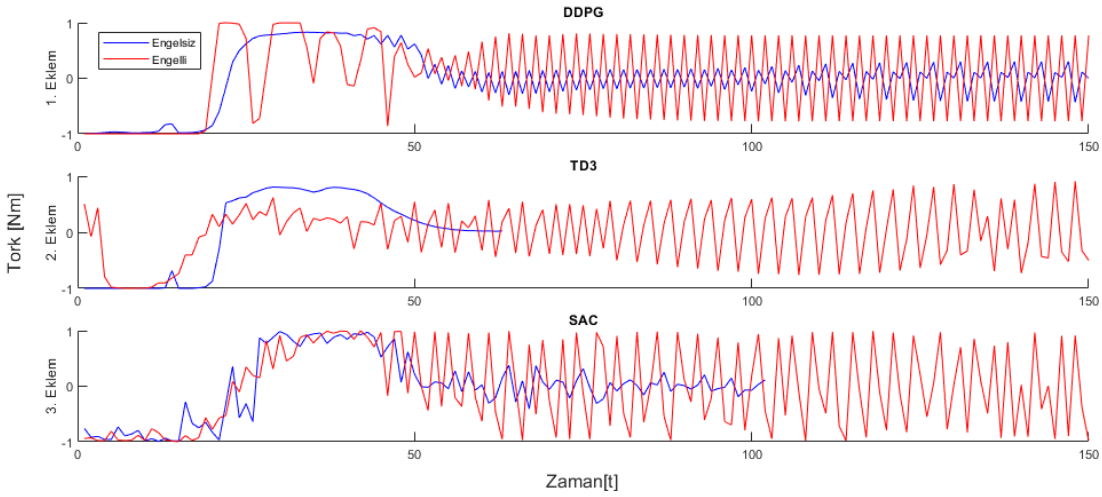
Şekil B.7 : 3 numaralı bölümde izlenen güzergahın x-z düzlem grafiği.



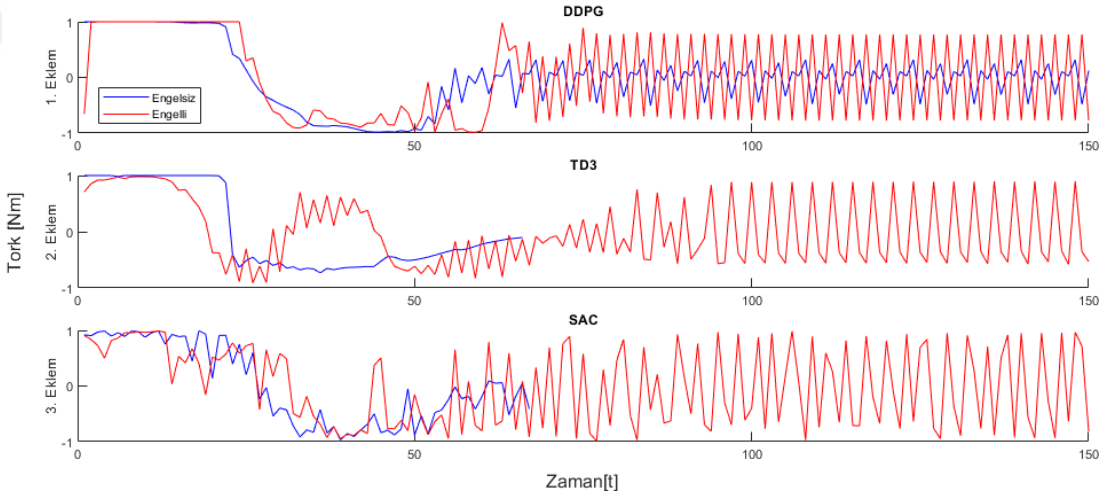
Şekil B.8 : 3 numaralı bölümde izlenen güzergahın x-y düzlem grafiği.



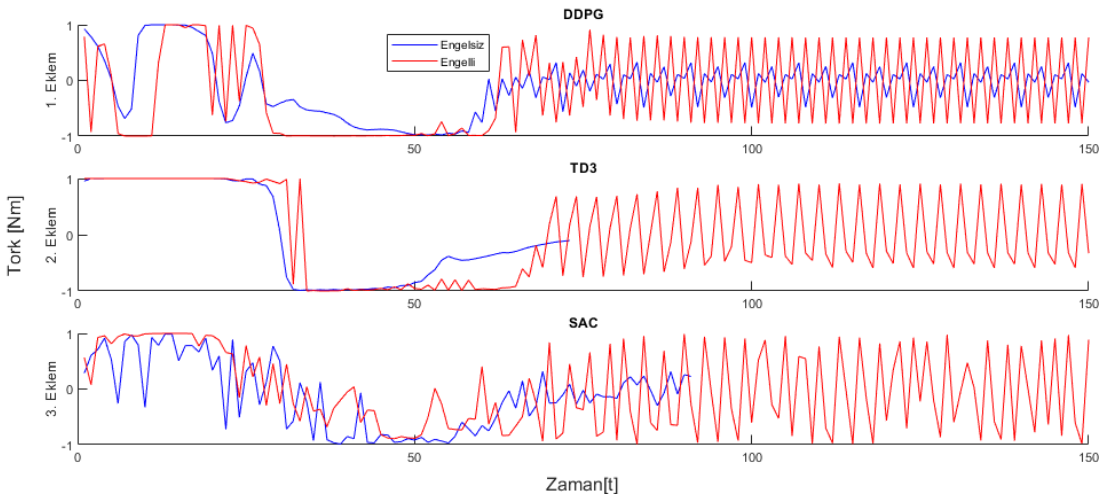
Şekil B.9 : 3 numaralı bölümde izlenen güzergahın z-y düzlem grafiği.



Şekil B.10 : 1 numaralı bölümde üretilen tork sinyalleri.



Şekil B.11 : 2 numaralı bölümde üretilen tork sinyalleri.



Şekil B.12 : 3 numaralı bölümde üretilen tork sinyalleri.

ÖZGEÇMİŞ

Adı-Soyadı **:** Abdurrahman Sefer DOĞRU

ÖĞRENİM DURUMU:

- **Lisans** **:** 2017, Bursa Uludağ Üniversitesi, Mühendislik ve Mimarlık Fakültesi, Makine Mühendisliği Bölümü
- **Yüksek Lisans** **:** 2022, Bursa Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Mekatronik Mühendisliği ABD

MESLEKİ DENEYİM VE ÖDÜLLER:

- AEB Mekanik Ltd. Şti. (2015-2019)