T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ PROGRAMI

DERİN ÖĞRENME KULLANARAK İHA İLE HAREKETLİ BİR HEDEFİN OTONOM OLARAK YAKALANMASI

YÜKSEK LİSANS TEZİ

Metin KOÇ

İstanbul
Ocak, 2020

T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ
YÜKSEK LİSANS PROGRAMI

DERİN ÖĞRENME KULLANARAK İHA İLE HAREKETLİ BİR HEDEFİN OTONOM OLARAK YAKALANMASI

YÜKSEK LİSANS TEZİ

Metin KOÇ

Tez Danışmanı
Dr. Öğr. Üyesi Aydın Tarık ZENGİN

İstanbul
Ocak, 2020

ÖNSÖZ

Araştırmamdaki her aşamada bana yardımlarını ve desteklerini esirgemeyen değerli tez danışmanım Dr. A. Tarık ZENGİN'e, eğitim alanında dersleriyle bize vizyon katan çok değerli hocalarıma, Teknofest Nu.D 38 İHA Ekibine, Değerli Ö. Faruk Demirbaş ve Mert Yavuz'a, yüksek lisans eğitimim boyunca her aşamada beni destekleyen ve yüksek fedakârlık gösteren sevgili eşim Melike'ye, küçük meleğim Sümeyye'ye ve aileme teşekkürlerimi sunarım.

Metin KOÇ

İstanbul - 2020

ÖZET

DERİN ÖĞRENME KULLANARAK İHA İLE HAREKETLİ BİR HEDEFİN OTONOM OLARAK YAKALANMASI

Metin KOÇ

Yüksek Lisans, Bilgisayar Bilimleri ve Mühendisliği

Tez danışmanı: Dr. Öğr. Üyesi Aydın Tarık ZENGİN

Ocak-2020, 79 Sayfa + XI

İnsansız hava araçları yüksek manevra kabiliyeti ve havada asılı kalma yeteneği ile yaygın olarak kullanılır hale gelmiştir. Günümüzde askeri alanda sıklıkla kullanılır hale gelmiş olan İHA'lar yardımıyla kamikaze dalışları ve mühimmat konuşlandırması gerçekleştirilmektedir. Hareketli bir nesneyi İHA ile yakalamayı hedefleyen bu projede literatürde daha önce gerçekleştirilmiş çalışmalardan farklı olarak İHA uçuş modundayken hedef tespitini görüntü işleme teknikleri yerine derin öğrenmek teknikleriyle gerçekleştirmek, bu iki tekniğin karşılaştırması ve sonuçlarını ortaya koymak amaçlanmıştır. Araç belirli bir irtifada hedefi tanımlar ve akabinde 3 eksende takibini gerçekleştirerek hedefe doğru alçalır. Simülasyon ortamında gerçekleştirilen testlere göre derin öğrenme teknikleri hedefi tanımlama konusunda geçmiş çalışmalarda kullanılan yöntemlere göre daha isabetli olmaktadır. Simülasyon ortamı dışında henüz test gerçekleştirilememişse de simülasyon ortamında tatmin edici sonuçlar ortaya koyulmuştur.

Anahtar Kelimeler: İnsansız Hava Aracı, Görüntü İşleme, Derin Öğrenme, Otonom İniş

ABSTRACT

AUTONOMOUS PICKING UP MOVING TARGET BY UAV USING DEEP LEARNING

Metin KOÇ

Master of Science, Computer Sciences and Engineering

Supervisor: Dr. Aydin Tarik Zengin

January-2020, 79 Pages + XI

Unmanned aerial vehicles (UAVs) have found increasingly wide application due to their high maneuverability and remarkable ability to stay motionless while airborne. Munition deployment and kamikaze diving missions are two of the many modern use cases of UAVs in military operations. One aspect that sets this study apart from those in the past is that target identification during flight mode was done using only deep learning techniques, as opposed to image processing techniques, where the aim was to compare and assess the performance of the two different approaches. The target is identified from a certain altitude and tracked down in all three axes as the UAV performs controlled descent. According to tests performed in a computer simulation environment, the employed deep learning technique has achieved higher accuracy in target identification than techniques employed in earlier studies. Although no physical testing has been done as part of this study, satisfactory results have been attained within the simulation environment.

Keywords: Unmanned Air Vehicles, Image Processing, Deep Learning, Autonomous Landing

İÇİNDEKİLER

ÜÇÜNCÜ BÖLÜM

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

3D	:	3 Dimension : 3 Boyutlu
BSD	:	Berkeley Software Distribution
İHA	:	İnsansız Hava Aracı

BİRİNCİ BÖLÜM

1. GİRİŞ

1.1. İnsansız Hava Araçları

İnsansız hava aracı (İHA), bir pilot tarafından kontrol edilmeden uçuş yapabilen hava araçlarına verilen ortak isimdir. İHA'lar, ilk yıllarında, insanlar için tehlikeli görülen basit görevler için, çeşitli orduların hava kuvvetleri tarafından kullanıldı. Daha sonradan ticaret, bilim, ziraat, güvenlik, fotoğrafçılık ve nakliyat gibi diğer alanlarda kullanım alanı buldu.

1849 yılı Venedik kuşatması sırasında Avusturya ordusunun bir taktik hamlesi olarak şehre havadan bırakılmak üzere 200 yangın balonunun insansız hava aracı tarafından taşınması, kayıtlara geçen ilk İHA kullanımıdır (Kaplan, 2013; Layman, 1996; McKenna, 2016; P. & Hallion, 2003; Renner, 2016). 20. yüzyılın ilk yarısı boyunca İHA'ların askeri personel eğitiminde atış hedefi olarak kullanıldığı ve bu alanda gelişmeler yaşandığı bilinmektedir. İnsansız hava araçlarına dair gelişmeler Birinci Dünya Savaşı'nda hız kazandı. Bu dönemde Dayton-Wright Airplane Company tarafından, önceden belirlenen bir zamanda infilak edebilen pilotsuz hava torpidosu geliştirildi (Kanyike, 2012). "Hewitt-Sperry Otomatik Uçak" ve Charlsten Kettering'in belirlenmiş bir hedefe patlayıcı bırakabilen "Kettering Bug" aracı, Birinci Dünya Savaşı'nda tasarlanmış İHA'lara önemli birer örnektir. 1916 yılında A. M. Low tarafından ilk motorlu İHA denemesi olan "Aerial Target" uçuruldu (William & Taylor, 1977). İlk büyük ölçekli İHA, Reginald Denny tarafından 1935 yılında geliştirildi (William & Taylor, 1977). İkinci Dünya Savaşı'nda gelişmeler artarak devam etti. Almanya'da uçaksavar personeli eğitiminde hedef olarak kullanılmak ve hava saldırıları gerçekleştirmek için gelişmeler yaşandı.

1959 yılında Amerikan ordusu, düşman toprakları üzerinde pilotların güvenliği endişesiyle İHA teknolojisine ağırlık vermeye başladı (Wagner, 1982). "Red Wagon" isimli proje hayata geçirildi (Wagner, 1982). 1964 yılında Ryan Model 147, Ryan AQM-91 Firefly ve Lockheed D-21 İHA'ları ilk kez, Vietnam Savaşı'nda kullanıldı (Wagner, 1982). Yıpratma Harbi'nde (1967-1970), keşif kameralı ilk İHA'lar İsrail istihbaratı tarafından başarıyla kullanıldı ve Süveyş Kanalı ve çevresinden görüntüler

elde edildi. Böylece ilk defa, jet bazlı ağır İHA'ların aksine, kısa iniş pistlerini kullanabilen İHA'lar savaşta kullanılmış oldu (Dunstan, 2008).

1973 Yom Kippur Savaşı'nda İsrail, sahte hedef olarak düşman uçaksavarlarını teşvik etmek için insansız hava araçları kullanmıştır (Vijay Kumar Saxena, 2013). 1973 yılında ABD ordusu Güney Asya'da İHA kullandığını resmi olarak teyit etmiştir (Wagner, 1982). ABD Hava Kuvvetleri 100. Stratejik Keşif Filosu, Vietnam Savaşı süresince yaklaşık 3435 İHA harekâtı gerçekleştirmiştir (Wagner, 1982). 1973 Yom Kippur Savaşı'nda Mısır ve Suriye'de konuşlanan Sovyet uçaksavarlarının İsrail savaş uçaklarına büyük kayıplar verdirmesi üzerine İsrail ilk defa gerçek zamanlı gözetleme yapan İHA'yı geliştirdi (Azoulai, 2011; *"Russia Buys A Bunch of Israeli UAVs"*., 2015; SCHEVE, 2015). İHA'larca elde edilen görüntüler ve sahte hedef çalışmaları yardımıyla İsrail, Suriye'deki hava savunmasını 1982 Lübnan Savaşı arifesinde tamamen etkisiz hale getirmiş ve hiç pilot kaybı yaşamamıştır (Levinson, 2010). 1980'ler ve 1990'larda ilgili teknolojilerin olgunlaşması ve fiziken küçülmesi ile ABD ordusundaki İHA kullanımı yaygınlaştı. 1991 Körfez Savaşı'nda İHA'lar yaygın olarak kullanılmıştır.

2012 yılı itibariyle ABD Hava Kuvvetleri bünyesinde 7494 İHA bulunmaktadır ve bu sayı ABD Hava Kuvvetlerindeki uçak sayısının yaklaşık üçte birine tekabül etmektedir (Spencer Ackerman & Shachtman, 2012; Singer, 2009). İHA'lar CIA (Central Intelligence Agency) tarafından da kullanılmaktadır. 2013 itibariyle İHA'lar en az 50 farklı ülkenin hava kuvvetleri tarafından kullanılmaktadır (Radsan & Murphy, 2011). Son yıllarda sivillere ait İHA'ların, askeri İHA'lardan sayıca fazla olduğu bilinmektedir. Günümüzde İHA'lar, düşük maliyeti ve kolay bakımı sayesinde fotoğrafçılık, muhabirlik, nakliye, gözetleme, arama kurtarma gibi birçok alanda kullanım görmektedir. İHA'lar; navigasyon, gerçek-zaman sistemler, robotik ve uçuş kontrolü gibi yükseköğretim araştırma konuları için önemli bir unsur olmakla birlikte, yaygınlığı, pratikliği ve erişilebilirliği günden güne artmaktadır.

2013 Aralık ayında Deutsche Post, tıbbi malzeme naklini İHA'larla gerçekleştirmeyi test ettiği "Parcelcopter" projesiyle uluslararası basının ilgisini kazandı (Ginsburg, n.d.). Son yıllarda basın kuruluşlarının ünlülerin görüntülerini almak için İHA'lardan yararlandığı da bilinmektedir (Ginsburg, n.d.). 2014 yılında Hindistan'ın Tamil Nadu bölgesinde bir granit dolandırıcılığının tespiti için insanların sığamadığı mağaralarda,

üzerinde kamera ve sensörler bulunan İHA'larla arama yapılarak granit tespiti yapılmıştır (Selvaraj, n.d.).

1.1.1. Günümüzde İnsansız Hava Araçlarının Askeri Alanda Kullanılması

İHA'lar askeriyede geniş faaliyet alanına sahiptir ve günümüzde bombalama, hedef tespiti, tarama, suikast, haritalama, keşif, gözetleme, hedef saptırma gibi askeri görevler için kullanılmaya çok müsait birçok İHA modeli geliştirilmeye ve kullanılmaya devam etmektedir.

Şekil 1-1 Kale-Baykar tarafından üretilen Bayraktar TB2 [24]

Gözetleme faaliyetlerine bir örnek olarak, Kanadalı Aeryon Labs tarafından geliştirilen ve sessizce havada durup yerdeki obje ve insanların çekimini yapan küçük insansız hava aracı "Aeryon Scout" gösterilebilir (Gibiansky & Gopalakrishnan, 2017). İHA'lar ABD'de tayfun ve benzeri doğal afetler esnasında ve sonrasındaki arama kurtarma görevlerinde de yaygın olarak kullanılmaktadır.

1995 yılında Predator serisi İHA'ların üretilmesiyle beraber kamuoyunun İHA algısının merkezine İHA saldırıları yerleşti. Çünkü Predator serisi İHA'lar yüksek isabetli atış ile stratejik terör hedeflerinin ve yüksek rütbeli düşman öğelerin ortadan kaldırılması için kullanıldı. Buradaki "yüksek isabetli" kavramını tartışmaya açan bir konu ise, bazı araştırma raporlarında karşımıza çıkacağı üzere, öldürülen her bir yüksek rütbeli düşmanın yanında ilave olarak 28 kişinin de ölüyor olmasıdır (S. Ackerman, 2014).

ABD istihbarat örgütü CIA tarafından kontrol edilen Predator İHA, 3 Kasım 2002 tarihinde Yemen'de 6 El-Kaide şüphelisini etkisiz hale getirmiş, ve bu olayla ilk kez

silahlı bir Predator IHA savaş alanı dışında saldırı aracı kularak kullanılmıştır (Pike & Aftergood, 2012).

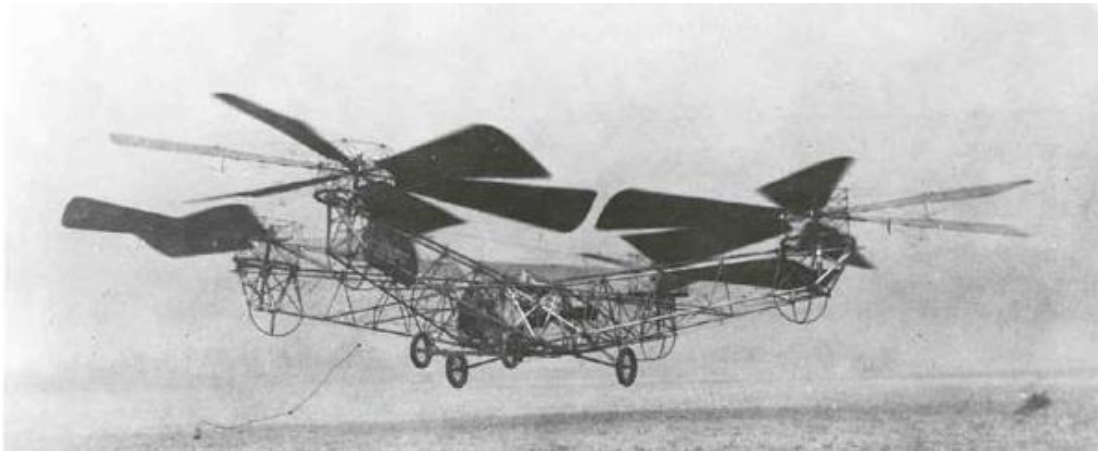
Yine, ABD istihbarat örgütü CIA üssünden kontrol edilen bir MQ-1 Predator model İHA'nın, 2005 yılı Mayıs ayında Pakistan'daki El-Kaide elebaşlarından Haitham el Yemeni ve Saimullah Khan'ın içerisinde bulunduğu kişisel aracı Hellfire füzesi ile vurarak bu iki terör şüphelisini etkisiz hale getirdiği rapor edilmiştir (Moreau & Yousafzai, 2017).

Şekil 1-2 : Hellfire füzeli General Atomics MQ-1 Predator – Kaynak: Wikipedia

1.1.2. Dört Döner Kanatlı Sistemler

Dört döner kanatlı sistemlerde (DDKS), ikisi saat yönü, ikisi saat yönünün tersine olmak üzere dört adet sabit pervane bulunur. Tork ve lift kuvvetleri pervane dönüş hızının değişimiyle kontrol edilir ve araçta istenen hareket elde edilir.

1920'li yıllarda yapılan insanlı tasarımlar dikey kalkış ve iniş yapan ilk başarılı araçlar arasındadır (Leishman, 2000). Ancak ilk prototipler düşük performans göstermiş ve sonrasında gelenlerse başarısız denge takviye sistemi ve kontrol yetisinin azlığı nedeniyle pilot için fazla iş yükü çıkarmıştır (Leishman, 2000).



Şekil 1-3 : De Bothezat Helicopter, 1923. Kaynak: Wikipedia

Son yıllarda DDKS tasarımlar insansız hava araçlarında yaygın konumdadır. Bu araçlar denge için elektronik kontrol sistemleri ve sensörlerden faydalanmaktadır. Bu İHA'lar, atik yapıları ve fiziksel olarak küçük olmaları sayesinde iç ve dış alanlarda uçurulmaya müsaittir.

Günümüzde tipik bir DDKS'de konum tespiti için ivmeölçer, jiroskop, manyetometre, barometre, sesüstü yakınlık sensörleri ve tercihen kamera ve GPS alıcısı bulunur.

İç mekân DDKS'lerde mutlak konum için GPS yerine manyetometre kullanılır. Ancak dış mekân DDKS'lere göre avantajları, görev süresinin kısa ve sahanın küçük olmasıdır. Dış mekân DDKS'ler ise daha dayanıklı, yük taşımaya müsait ve uzun süreli görevler için yetkin olmalıdır. Dış mekân DDKS'lerde GPS alıcısı aracılığıyla mutlak konum tespiti mümkündür.

Dört döner kanatlı sistemlerin benzer boyuttaki helikopterlere göre birçok fiziki avantajı vardır. Öncelikle, uçuş sırasında pervane pitch açısını değiştirmek için mekanik bir sisteme ihtiyaç yoktur. Böylece tasarım ve onarım yükü azalır (Pounds et al., 2006). Dört ayrı pervane kullanılması, her bir pervanenin çapının küçük olmasına olanak sağlar ve böylece her bir pervanenin ihtiva ettiği kinetik enerji düşük seviyede kalır. Böylece, olası bir kazada pervane ve çarpılan yüzeyde oluşan hasar minimize edilmiş olur. Küçük İHA'ların iç mekân uçuşlarında bu konu güvenlik bakımından yüksek önem arz etmektedir. Bazı küçük ölçekli İHA'larda pervanelerinin çevresine koruyucu bir yapı tesis edilmiştir (Hoffmann et al., 2007). DDKS'ler, üretim ve kullanım kolaylığı sayesinde sivil hobi kullanımında yaygındır.

Şekil 1-4: Songar Silahlı İHA

1.1.3. Benzer Çalışmalar

(Hu et al., 2017) çalışmasında IMU ve hareket yakalama sensöründen aldığı verileri değerlendirerek dikey hareket eden bir platform üzerine iniş gerçekleştirmiştir. (Falanga et al., 2017; Fu et al., 2016; Line, 2018; Mendes, 2012) çalışmalarında üzerinde Qr barkod bulunan hareketli bir araç üzerine otonom iniş gerçekleştirmiştir . (Battiato et al., 2017) yaptığı çalışmada üzerinde X işaretli bir görsel bulunan ve sekiz şeklinde bir yol izleyen hareketli bir araç üzerine otonom iniş gerçekleştirmiştir. (Lange et al., 2009) çalışmasında GPS'in karıştırıldığı bir ortamda optik flow sensörden aldığı hareket tahmini ve kamera görüntüsünden aldığı yükseklik tahmini ile iniş pedi üzerine iniş gerçekleştirmiştir. (Borowczyk et al., 2017) çalışmasında 50 [km/h] gibi yüksek hızla ilerleyen bir araç üzerine otonom iniş gerçekleştirmiştir. (Feng et al., 2018) rüzgâr altında 12 [m/s] hıza kadar hareket eden ve üzerinde Qr barkod bulunan araç üzerine model öngürlü kontrol algoritması kullanarak otonom iniş gerçekleştirmiştir. (Benavidez et al., 2014) çalışmasında Ar.Drone2.0 İHA'nın bulanık mantık algoritması kullanarak hareketli bir platform üzerine inişini sağlamıştır. (Gilberto Mendoza Chavez, 2016) çalışmasında hareket yakalama sensöründen aldığı İHA ve hareketli aracın datalarını değerlendirerek araç üzerine iniş gerçekleştirmiştir.

(Lins et al., 2015; Polvara, 2018) çalışmasında hareketli bir gemi üzerine iniş gerçekleştirmiştir. (Ling et al., 2014) düşük bütçeli İHA'nın otonom deniz aracı üzerine otonom iniş sağlamasını sağlamıştır. (Özkan, 2017) rotası belirli bir aracın üzerine inişin en kısa rotasını hesaplayarak iniş gerçekleştiren algoritma gerçekleştirmiştir. (Vlantis et al., 2015) üzerinde eğik bir yüzey bulunan hareketli bir araç üzerine iniş gerçekleştirmiştir.

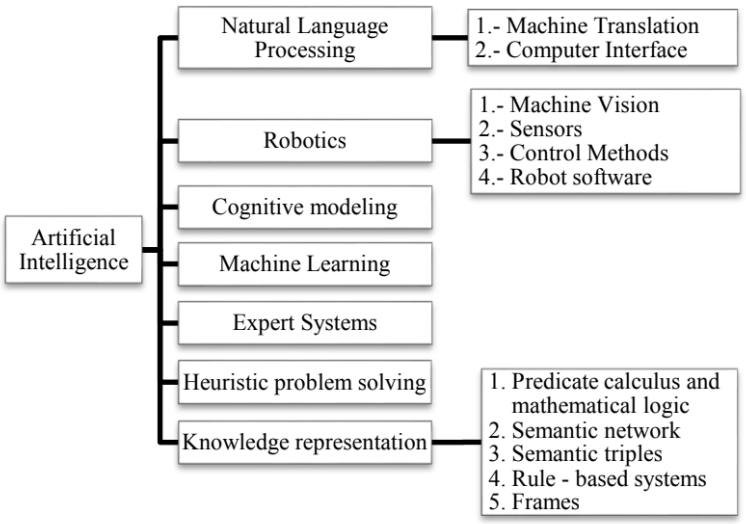
1.2. Yapay Zekâ

Yapay zekâ son yıllarda üzerinde çokça çalışmalar gerçekleştirilen alanlardan birisidir. Matematik, bilgisayar bilimleri, sanat, mimarlık gibi birçok alanda aktif olarak kullanılan yapay zekanın tanımı şu şekilde yapılmaktadır.

> *"İnsanlara özgü öngörme, öğrenme ve karar verme gibi yetenekleri bilgisayar sistemlerinin yerine getirmesini sağlayan teknolojiye yapay zeka adı verilmektedir."*(Millington & Funge, 2009)

1950'li yıllarda "Turing Testi" adlı makalede bilgisayarın insanın aklını nasıl kullandığına dair tatmin edici bir tez öne sürüldü (Mosavi & Varkonyi-Koczy, 2017).

Birçok alt alana sahip olan yapay zekâ sistemleri doğal dil işleme(NLP)'de yazma deneyimini artırmayı, Robotikte karar verme kabiliyeti yüksek robotlar geliştirmeyi, ulaştırmada otonom araçların kusursuza yakın hareket etmesini sağlamayı vb. birçok konuda insan zekasına yakın hareket etmeyi hedeflemiştir (Abdel-Hamid et al., 2012; Bannister, 2016; Bengio, 2009; Liu et al., 2017).



Şekil 1-5: Yapay zekâ araştırma alanları [28]

1.2.1. Derin Öğrenme

Derin öğrenme; obje tanıma, ses tanıma, makine tercümesi gibi birçok yapay zekâ alanında büyük gelişmelerin önünü açmıştır (Lecun et al., 2015). Derin mimari yapısı sayesinde derin öğrenme birçok yapay zekâ problemini çözebilmektedir. Bunun sonucu olarak da araştırmacılar derin öğrenmeyi geleneksel kullanım alanları olan yüz ve konuşma tanımanın dışına çıkarmaktadır (Osako et al., 2015). Örneğin, Tekrarlı Sinir Ağı (Recurrent Neural Network) kullanarak konuşma sinyalinde gürültü azaltılabildiğini göstermiştir (Gupta et al., 2015). Yığılı otomatik şifreleyiciler kullanarak genomdaki gruplaşmaları tespit etmiştir (Wang & Yang, 2017). Derin öğrenme kullanarak duygu analizi çalışması yapmıştır.

Günümüz, derin öğrenme araştırmalarının kendi tarihi boyunca en sık yaşandığı dönemdir.

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks

Şekil 1-6: Yapay zekâ ve derin öğrenme araştırma tarihçesinden önemli mil taşları (Lecun et al., 2015).

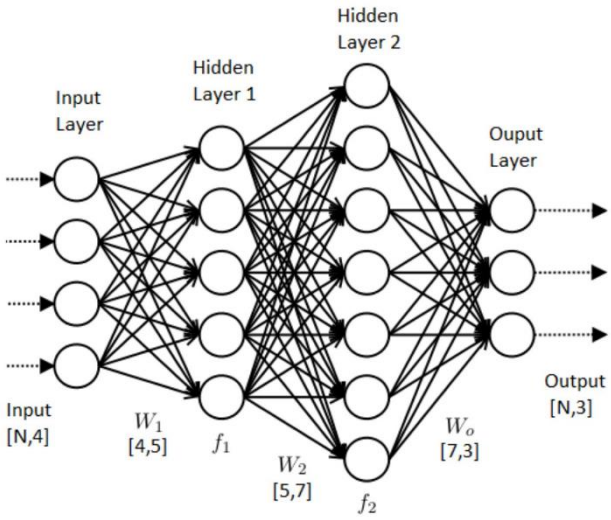
Ancak bu araştırmaların niteliğini anlamak için derin öğrenmenin tarihini de anlamakta fayda vardır. Derin öğrenme alanında araştırma yapıp bu disiplini ilerletebilmek için, derin öğrenme tarihi boyunca ne tür yaklaşımların denendiğini ve başarılı olduğunu, ne tür yaklaşımların ise denenip başarısız olduğunu görmek, bizden öncekilerin deneme yanılma yoluyla elde ettiği kazanımların farkında olmak gerekmektedir. Günümüzde derin öğrenmenin yapısının neden bu şekilde olduğunu anlamanın yolu, önce tarihini anlamaktan geçmektedir.

Derin öğrenme ve yapay zekâ araştırmalarının temelinde insanoğlunun kendi beyninin nasıl çalıştığını anlamak istemesi ve bunu modellemeye çalışması yatmaktadır. Bu bağlamda, insan beyninin nasıl çalıştığının ilk akademik irdelemelerini yapmış olan Aristo'nun M.Ö. 300'lü yıllardaki çalışmaları, ilk derin öğrenme araştırmaları sayılabilir (Wang & Raj, 2017). Aristo, insan bilincinin birbirine bağlantılı kavramlar

zincirinden oluştuğunu öne süren çağrışımcılık akımını başlatmıştır (Burnham, 1888). Zihne getirilen bu bakış açısı daha sonra 17. yüzyıldan itibaren David Hume, Dugald Stewart, Thomas Reid ve James Mill gibi araştırmacılar tarafından geliştirildi ve yeri geldikçe revize edildi (Wang & Raj, 2017).

1958'de Rosenblatt, sinir ağlarının birim ünitesi olan nöronların atası olarak kabul edilen "perceptron" kavramını ileri sürdü. Perceptron, özünde, girdileri ve çıktıları olan temel bir işlem birimidir ve sinir ağlarının bir türü olarak düşünebileceğimiz MLP'leri (Multi-Layer Perceptron) oluşturur. MLP'lerin girdi katmanı, gizli katmanları ve çıktı katmanı vardır. Gizli katman sayısı ve her katmandaki perceptron sayısı uygulamaya ve probleme göre değişkenlik gösterir.

MLP'leri evrensel olarak hesaplamada kullanışlı kılan bazı önemli matematiksel özellikleri vardır: tek gizli katmanlı bir MLP, tüm bool ifadeleri kesinlik ihtiva ederek çözebilir; tek gizli katmanlı bir MLP tüm kapalı ve sürekli fonksiyonlara makul bir isabetlilik oranı içerisinde yaklaşımda bulunabilir; iki gizli katmanlı bir MLP tüm fonksiyonlara makul bir isabetlilik oranı içerisinde yaklaşımda bulunabilir (Wang & Raj, 2017). Bu özellikler sayesinde de MLP'ler nümerik analiz ve yapay zekada çok güçlü bir konumda bulunmaktadır ve ileri beslemeli yapay sinir ağları olmaları hasebiyle, yapay sinir ağlarının öncüsüdür.



Şekil 1-7: İki gizli katmanlı bir MLP şeması. (Kaynak: datasciencecentral.com)

Yapay sinir ağlarının jenerik yapısının çok fazla hesaplamaya sebebiyet vermesi ve yüz milyonlarca nöron bağlantısı gerektirecek bazı problemler için zamanla yetersiz

kalmaya başlaması ile bazı yeni mimari yapılar ortaya çıkmıştır. Bunlardan günümüzde en yaygın olanları, Recursive Neural Network (RNN) ve Convolutional Neural Network (CNN) mimarileridir. RNN'ler çoğunlukla "hafıza" veya zaman-bağımlı verilerin birbiriyle ilişkisini ihtiva eden problemlerin çözümünde kullanılırken, CNN'ler ise zamandan bağımsız ancak fiziksel yakınlığın anlamsal yakınlık teşkil ettiği veriler (örneğin fotoğraftaki pikseller) için uygun ve başarılıdır. Fotoğraf veya videodan obje tanıma gibi işlemler çoğunlukla CNN bazlı derin öğrenme mimarilerinin kullanım alanını oluşturmaktadır.

CNN mimarisinde en az bir katmanda matematiksel konvolüsyon yapılır. Konvolüsyonun amacı, özünde, verinin içindeki anlamı kaybetmeden verinin salt büyüklüğünü azaltmaktır. Yani elde bulunan büyük miktardaki veriyi küçültmek, ancak içindeki anlamı kaybetmemektir. CNN'lerde birden fazla ve çeşitli çekirdek yapıları ve konvolüsyon metotları kullanılarak konvolüsyon yapılabilir.

Derin öğrenmenin temel unsuru olan ve günümüzde yaygın kullanım alanı bularak popülerleşmiş RNN ve CNN bazlı mimarilerin de temel taşını oluşturan yapay sinir ağlarının, analitik olarak optimize edilemeyen birçok parametresi vardır. Gizli katman sayısı, aktivasyon fonksiyonları, tam bağlı veya konvolüsyon katmanlarının konumları, öğrenme katsayısı, bunlara bazı örnekler olarak gösterilebilir. Bu parametrelere hiper parametre denmektedir ve problemin doğasına göre farklı değerlerde verimlilik göstermektedirler. Mimarinin tasarlayıcısı tarafından değiştirilerek en verimli halin deneysel olarak keşfedilmesi gerekmektedir. Ancak bazı hiper parametreleri otomatik olarak optimize eden algoritmalar da mevcuttur.

Günümüzde derin öğrenme bahsedildiği üzere son derece gelişmiş bir konumdadır ve en aktif araştırma alanlarından biridir. Tanınmış kurumların es tanıma, nesne tanıma, maskeleme gibi zorluklar üzerine her yıl düzenlediği yarışmalar ve yine aynı kurumların bu zorluklar için gerekli olacak veri tabanlarını sağlıyor olması ve bu veri tabanlarının büyütülmesinde sade vatandaşın katkılarından da istifade eden yaklaşımlar, bu alandaki akademik gelişmelere katalizör görevi görmektedir. Günümüzde en yaygın olarak bilinen ve en başarılı modellerin pek çoğu büyük yarışmaların kazananları olarak ortaya çıkmıştır.

1.2.2. Yapay Zekanın İnsansız Hava Araçlarında Kullanımı

İnsanlı hava araçları, yapıları gereği bazı dezavantaj ve kısıtlamalara sahiptir. İnsanlı bir aracın sahip olacağı minimum hacim ve kütle, buna bağlı olarak aracın sahip olması gereken teçhizatın gücü ve yakıt miktarı fiziksel kısıtlamalara örnek olarak gösterilebilir.

Bu kısıtlamaları aşmanın bir yolu uzaktan komuta edilen pilotlu İHA'lardır. Ancak bu gibi pilotlu araçlarla yapılacak bazı operasyonlar, komuta merkeziyle olan mesafeden kaynaklı kaçınılmaz gecikmenin tolere edilemeyeceği hassaslıkta olabilir. Özellikle askeri anlamda günümüzün sessiz, isabetli ve gizlilik ihtiva etmesi gereken operasyonları için bu gibi fiziksel kısıtlamalar; İHA'ların mümkün olduğunca otonom bir şekilde karar ve eylemleri gerçekleştirerek operasyonel tanımlarını ellerinden geldiğince kendi kendilerine ifa edebilmelerinin gerekliliğini ortaya çıkarmıştır. Tüm bunlara pilotun olası hata faktörü de eklendiğinde yapay zekâ İHA'larda neden geniş bir kullanım alanına sahip olduğu ortaya çıkmaktadır.

Yapay zekâ bazı İHA'larda sadece belirli modüllerin otomasyonu üzerinden pilota destek olmak için kullanılırken, bazı İHA'larda ise tamamen otonom bir yapı oluşturmak için kullanılmaktadır.

İlk zamanlarda uzman sistemler üzerinden pilot davranışını ve karar mekanizmasını taklit etme yaklaşımıyla geliştirilmeye çalışılan sistemlerde zamanla bu problemin daha farklı yaklaşımları zorunlu kılacak kadar kompleks olduğu anlaşılmıştır. Bunun sebebi ise uçmakta olan bir aracı sürekli etkilemekte olan dinamik faktörlerdi.

Şekil 1-8 : STM – Kargu otonom İHA. (Kaynak: millisavunma.com)

Uzman sistemler aracılığıyla bütüncül bir çözüm getirme arayışları, zamanla yerini eldeki problemi daha küçük parçalara ayırarak bulanık mantık ve yapay sinir ağı gibi yapay zekâ algoritmalarının işlendiği modüler yaklaşımlara bıraktı. Bu dönemde Anthony J Calise gibi araştırmacılar, İHA'ların uçuşuyla ilgili problemlerin birçoğunun çözümünün yapay sinir ağlarına bırakılabileceğini öne sürmüştür (Wyeth et al., 2000). Norman Fitz-Coy da zaman ilerledikçe küçük İHA'ların (MAV) kontrolünden sorumlu otonom sistemlerde örüntü tanıma, genetik algoritma, sinir ağları, bulanık mantık ve bilgi tabanlı sistemler gibi "yumuşak" hesaplama metotlarının başarılı olacağını öngörmüştür (Magazine, 2008).

Günümüzde yapay zekanın erişmiş olduğu en yeni kuram ve pratik kazanımlar İHA'lara rahatlıkla uyarlanabilir olup, bunun sonucu olarak hem askeri hem kamusal hem de sivil alanda gerek modüler otomasyon gerekse bütüncül otomasyon için yapay zekadan yaygın olarak istifade edilmektedir.

Yapay zekanın İHA'larda kullanılmasıyla ilgili günümüzde aşılması gereken önemli problemlerden biri, gerçek zamanlı yapay zekâ algoritmalarının performansının henüz istenen düzeye erişememiş olmasıdır. Derin öğrenme algoritmalarının yapıları gereği yüksek maliyetli olmaları, İHA'lara yapay zekâ sistemi kurulması konusunda bazı kısıtlar getirmektedir. Hata toleransı olmayan hassas görevlere çıkacak olan bir İHA'nın güçlü bir derin öğrenme algoritması çalıştırması, yani güçlü bir donanıma da

sahip olması gerekmektedir; bu da ekonomik maliyetin artması anlamına gelmektedir. Hata toleransı olan ve hassas olmayan görevlerde bile yapay zekâ modülünün "false-positive" verme senaryosu İHA'nın kendisi için risk arz edebileceğinden, gelecekte gerçek zamanlı yapay zeka algoritmalarının performansının artması, ve/veya bu algoritmaların paralel olarak çalışabildiği donanımların yaygınlaşması/ucuzlaması için ihtiyaç mevcuttur.

1.3. Görüntü İşleme

Bilgisayar bilimlerinde görüntü işleme, dijital ortamdaki görüntülerin, bir bilgisayar kullanılarak, belirli algoritmalar aracılığıyla işlenmesidir.

Dijital görüntü işleme, bilgisayarlı görünün önemli bir yapıtaşıdır. Bilgisayarlı görü disiplinler arası bir bilim dalı olmakla beraber, dijital görsellerden elde edilen veriden yüksek seviye çıkarımlar yapmayı sağlamakla birlikte, mühendislik bakış açısıyla, insanın görme sisteminin yaptığı görevleri otonomlaştırmayı hedefler (Sonka et al., 1993). Yine bilgisayarlı görü görevlerinin arasında, dijital görselleri elde etme, işleme, analiz etme, anlama, çıkarım yapma ve karar verme sayılabilir. Görüntü işleme ve bilgisayarlı görü kavramları zaman zaman iç içe geçebildiği ve birinin diğeri anlamında kullanılabildiği unutulmamalıdır.

Dijital görüntü işlemede kullanılan tekniklerin birçoğu 1960'lı yıllarda ABD'deki ileri gelen üniversiteler ve Bell Laboratories tarafından uydu görüntülerini işleme ve tıbbi fotoğraf geliştirme gibi amaçlarla ortaya koyulmuştur (Rosenfeld, 1969).

Şekil 1-9 : 31 Temmuz 1964, Ranger 7 adlı insansız aracın ayın yüzeyine çarpmadan 17 dakika önce gönderdiği fotoğraf. (Kaynak: NASA)

İlk zamanlardaki dijital görüntü işlemedeki başlıca amaç görüntü kalitesini artırmaktı. Bunun ilk başarılı örneklerinden biri ABD'deki Jet Propulsion Laboratory (JPL) tarafından gerçekleştirilmiştir. 1964'te bir görevden gönderilen binlerce Ay fotoğrafı üzerinde, Güneş ve Ay'ın o anki konumu da dikkate alarak geometrik düzeltme, gradyan transformasyonu, gürültü kaldırma gibi teknikler kullanıp Ay'ın yüzeyini haritalamakta büyük bir başarı elde etmişlerdir. Daha sonra 100.000 fotoğraf üzerinde daha kapsamlı görüntü işleme yapılarak Ay'ın topografik haritası, renk haritası ve panoramik mozaiği elde edilmiş ve ilk insanın Ay'a gitmesi için çok önemli bir unsur olmuştur (Gonzalez et al., 2009).

Dönemin donanımsal şartlarından ötürü dijital görüntü işlemenin maliyeti yüksekti, ancak 1970'ten itibaren daha ucuz ve erişilebilir bilgisayarların üretilmeye başlanmasıyla bu maliyet de azalmaya başlamıştır. Bunun sonucu olarak görüntüler gerçek zamanlı olarak işlenmeye başladı. Genel amaçlı bilgisayarların güçlenmesi ve

erişilebilir hale gelmesi ile günümüzde görüntü işleme hiç olmadığı kadar yaygın konumdadır.

Günümüzde görüntü işleme tıp, sanat, mühendislik ve askeri alanlarda geniş kullanım alanına sahiptir. İnternetin yaygın kullanılmasının doğurduğu ihtiyaçlardan biri olarak video ve fotoğrafların sıkıştırılmasında da görüntü işleme tekniklerinden sıkça faydalanılmaktadır.

Görüntü işleme aynı zamanda derin öğrenme modellerinin eğitiminden önce görsel verilerin önişlemesinde yoğun bir şekilde kullanılmakta ve modellerin hem eğitim süresini hem de başarımını artırmaktadır.

1.3.1. Görüntü İşlemenin İnsansız Hava Araçlarında Kullanımı

Görüntü bazlı kompleks kararlar veren sistemler görüntü önişleme için dijital görüntü işleme algoritmalarından yararlanmak durumundadır. Gerçek zamanlı görüntü bazlı karar verme mekanizmasına sahip olan otonom İHA'lar için, yapay zekâ modülünden bağımsız olarak, görüntüye önden yapılan görüntü işleme teknikleri hayati önem taşımaktadır. Kamera tarafından elde edilen salt görüntüden gürültü ve istenmeyen etkiler silinmeli, en yalın ve bilgi içeren haliyle sisteme girdi olarak verilmelidir, bu görevi üstlenen de aracın görüntü işleme alt prosedürüdür. Otonom olmayan İHA'larda da görüntü işleme algoritmalarına benzeri bir görev düşebilmektedir, zira komuta eden pilota ulaşan arayüzdeki görüntünün gürültü veya yanıltıcı kalıntılar taşıması görev için risk teşkil edecektir. Bahsedilen bağlamda İHA'larda görüntü işleme algoritmaları sıkça kullanılmaktadır.

Girdi olarak gelen görüntünün önişlenmesi haricinde, görüntü işleme metotları ile bir objenin konumunun maliyetsiz, ancak çok isabetli olmayarak, takip edilmesi mümkündür ki bu da İHA'lar için önemli bir kullanım alanı yaratmaktadır. Özellik çıkarma algoritmaları kullanarak nesne takibi yapılan birçok araştırma mevcuttur.

Otonom İHA'ların engel tespit ve kaçış mekanizmaları için görüntü işleme teknikleri kullanılmaktadır. Haritacılıkta İHA'larla elde edilen görüntülerin görüntü işleme teknikleri ile istenen forma getirildiği uygulamalar mevcuttur.

İKİNCİ BÖLÜM

2. MATERYAL-YÖNTEM

2.1. Simülasyon Ortamı

İnsansız hava araçları hem araştırma alanında hem ticari hem askeri hem de tarım alanı gibi sivil alanlarda çokça kullanılmış çok farklı senaryolarda başarılı uçuşlar gerçekleştirmiştir. Bu uçuşlar sırasında değişik hava koşulları, dış etkenler pilotunun kontrolü dışında İHA'nın kabiliyetlerini kısıtlamış, yer yer dengesini bozarak kaza kırımlar yaşamasına sebebiyet vermiştir. Bu kazalar sırasında İHA'nın ağır ve yüksek kuvvet uygulayabilen hareketli parçaları pilot veya çevrede bulunan canlı/cansız varlıklara da zarar vermiştir. Ayrıca yüksek fiyatlara sahip olan ticari İHA'ların kaza sonrası ciddi maddi kayıplar söz konusu olmaktadır. Bu tip durumların önüne geçmek adına birtakım çalışmalar yapılmakta, birçok ülkede ivedilikli önlemler alınmaktadır (Moskowitz et al., 2018).

Şekil 2-1 İHA kazası sonucu oluşan bir hasar [54]

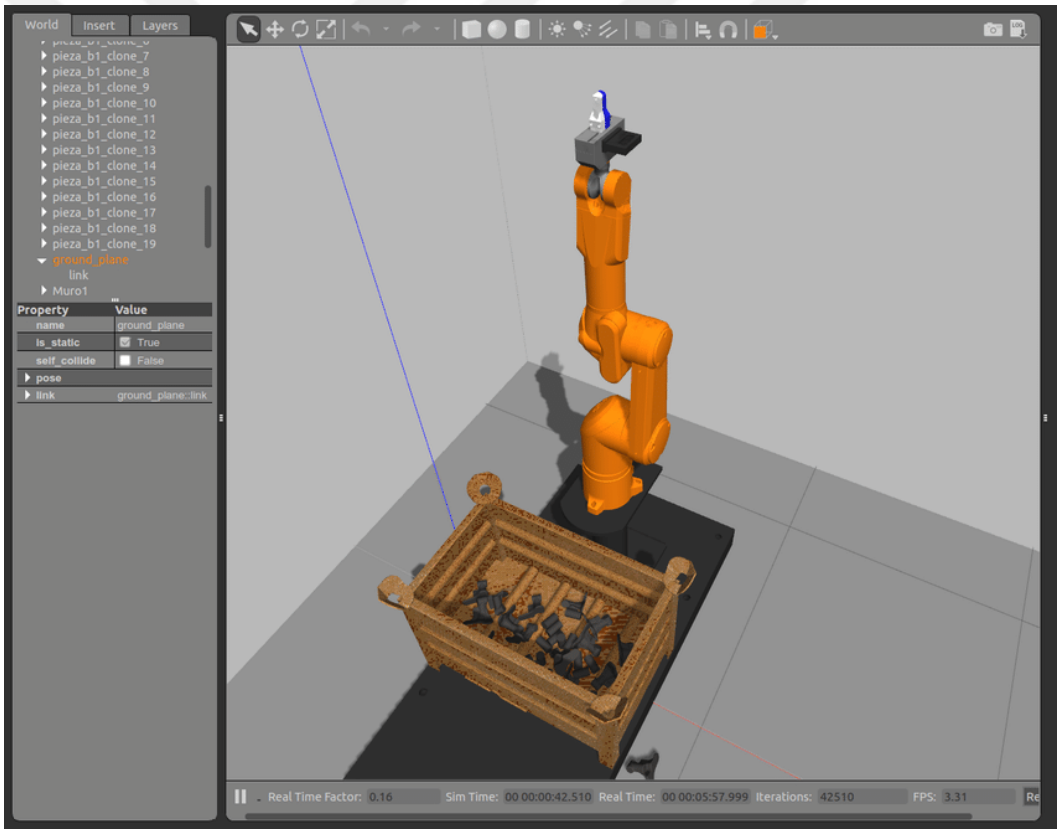
Bir İHA'nın kaza ve elverişsiz hava koşullarında uçuş zorluluğunu test etmek ve bu duruma karşı önlem almak amacıyla simülasyon programları oluşturulmuş, simülasyon ortamında rüzgâr, yer çekimi, engeller gibi dış etkenleri değiştilerek bu etkenlere karşı stabilitesini korumak için gerekli algoritmalar geliştirilebilir.

Son dönemde gerçekçi uçuş dinamiklerini sağlayan gerçekçi hava koşullarını gerçekleyebilen, birçok uçuş simülatörü ortaya çıkmıştır. Bu simülatörler sayesinde bir hava aracı ilk uçuşunu gerçekleştirmeden önce saatlerce simülasyon üzerinde

gerçekleştirerek hem zamandan hem maddi tasarruflar elde ederek kusursuz uçma deneyimi elde edebilmektedir.

Bu simülatörlerden birisi olan Gazebo simülatör robot bilimi ile uğraşan her kişi için gerekli tüm araçlara sahiptir ve yapay zekâ gibi alanlarda çalışmalar yapan bilimciler algoritmalarını hızlıca test etme, regrasyon testlerini gerçekleştirme ve gerçekçi robotlar tasarlama imkanına sahip olurlar (*Gazebo Simulation*, n.d.).

Gazebo iç ve dış ortamlarda robotları doğru ve verimli bir şekilde simüle etme imkânı sunmaktadır. Araştırmacıya sağlam bir fizik motoru, yüksek kaliteli grafikler ve kullanışlı bir arayüz sunarak verimlilik artışı sağlamaktadır ve en önemlisi açık kaynak kodlu bir ortam olan Gazebo simülatör herkes tarafından ücretsiz olarak kullanılabilmektedir.

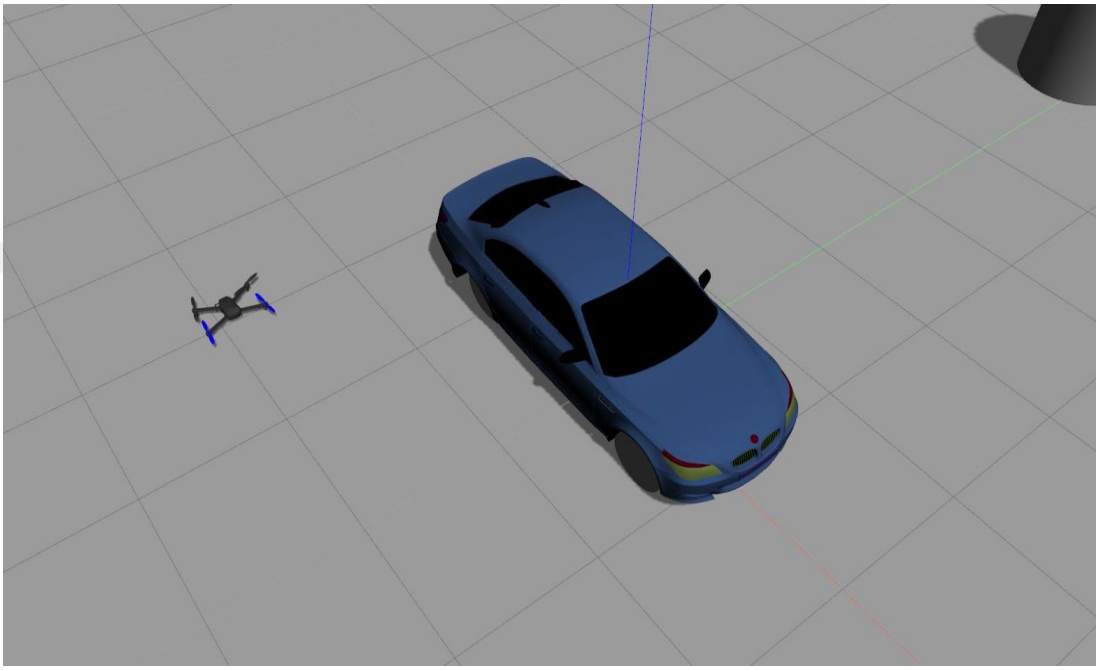


Şekil 2-2 : Çöp toplama görevi üstlenen bir robotik kolun Gazebo ortamındaki modeli [49]

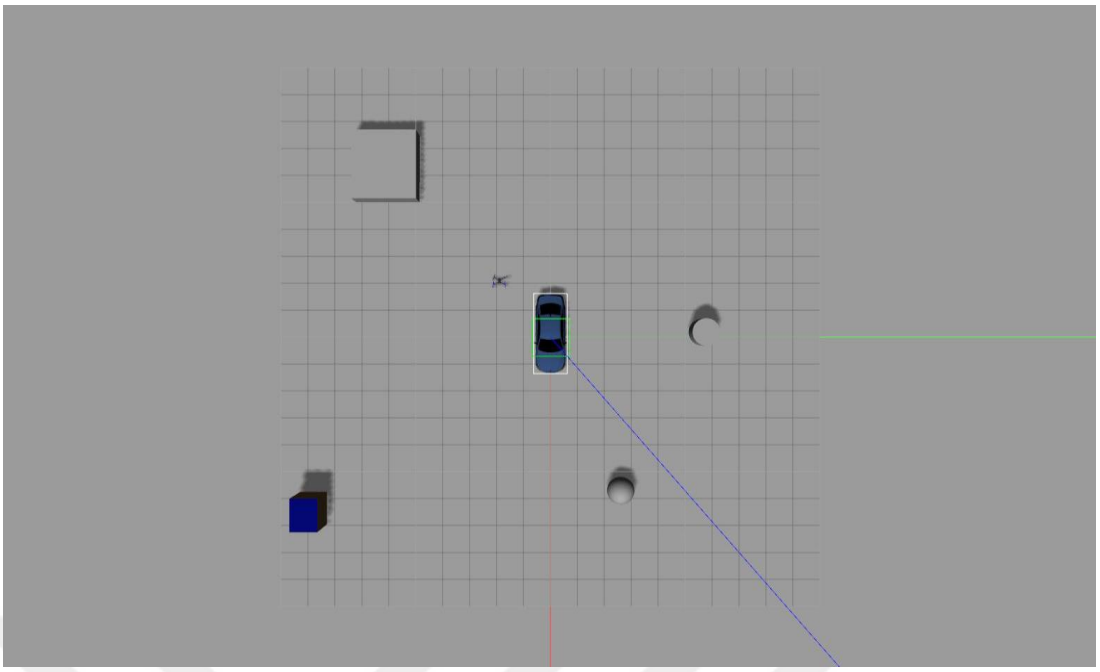
Bu çalışmanın konusu olan İHA'nın Gazebo simülasyon ortamında modellenmesi amacıyla akademik çalışmalarda sıklıkla kullanılan iris model İHA ve BMW 3.20 model bir aracın 3 boyutlu modelin kullanılmıştır. Hedef araç olarak kullanılacak olan

araba modeli 3 boyutlu tasarım programında modellenmiş ve akabinde urdf formatında kinetik modeli oluşturularak simülasyon ortamına aktarılmıştır.

Bu iki modele simülasyon ortamında hız, yükseklik, dönüş açısı gibi değişken parametreleri vermek için ROS(Robot Operating System) ara katman(Middleware) yazılımı kullanılmaktadır. Gazebo gibi açık kaynaklı olan ROS akademik çalışmalarda sıklıkla kullanılmakta ve güncel olarak geliştirilmektedir.



Şekil 2-3 : Simülasyon ortamında araba ve iHA modeli



Şekil 2-4 : Simülasyon ortamının üstten çekilmiş görüntüsü

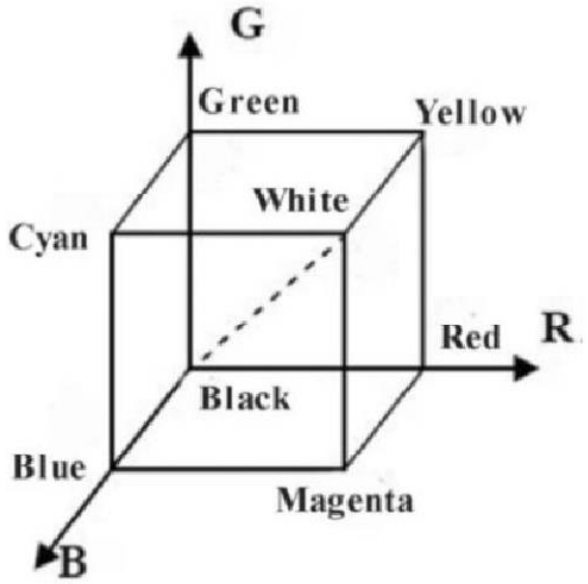
2.2. Görüntü İşleme İle Nesne Tanıma

Bilgisayar dünyasında donanımların hızla gelişmesi akabindeişlem gücünün yükselmesi insanlara bilgisayar kullanarak daha hızlı ve doğru hesap yapma kabiliyeti kazandrmıştır. Yüksek hesaplama gücü gerektiren görüntü işleme teknikleri günümüzde oldukça gelişmiş bu teknikleri otomatik olarak gerçekleyen birçok kütüphane ortaya çıkmıştır. Bunlardan bir tanesi olan OpenCV Berkeley Software Distribution(BSD) lisansı ile intel tarafından geliştirilmiştir. Bu kütüphane mükemmel çalışan birçok algoritma içermektedir. [50] C/C++ dilleri ile geliştirilen OpenCV kütüphanesi python, java gibi diller ile kullanılabilmektedir (Bradski & Kaehler, 2008; Corporation & Garage, 2013).

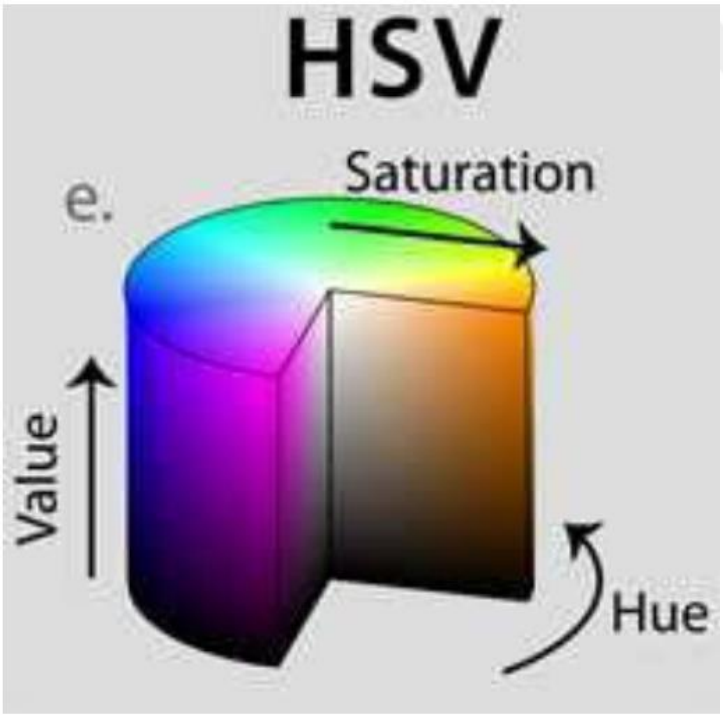
Yapılan bu çalışmada materihedef aracı görüntü üzerinde bulmak amacıyla OpenCV kütüphanesi kullanılacaktır. Öncelikle kullanılacak teknikten bahsetmek gerekirse, hedef renk tanıma yoluyla algılanacak ve nesne takibi sağlanacaktır. Nesne takibi ilerleyen bölümlerde açıklanacaktır.

Kameradan alınan görüntüler işlem gerçekleştirilen bilgisayara RGB(Kırmızı-Mavi-Yeşil) renk uzayında sağlanmaktadır. Her bir Pixel'i 0-255 aralığında 3 adet değere sahip olan resim yapılan bu çalışmada HSV(Hue Saturation Value) renk uzayına

çevirilecektir. Çünkü bu HSV renk uzayı insan gözünün renk uzayına daha yakın olduğu için algılama kolaylaşacaktır [50].



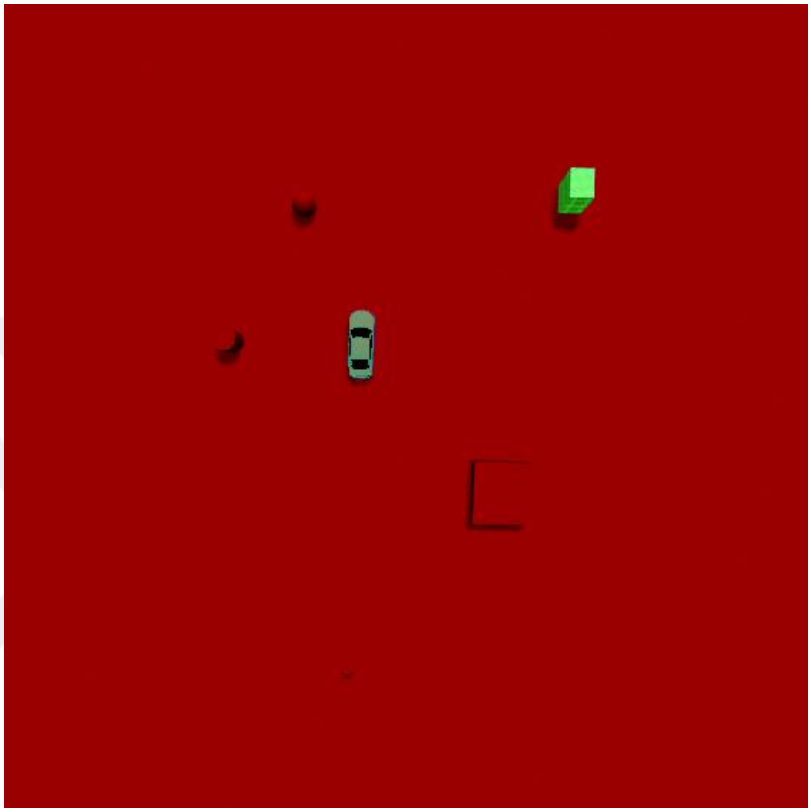
Şekil 2-5 : RGB Renk Uzayı (Ghinmine & Sapkal, 2017)



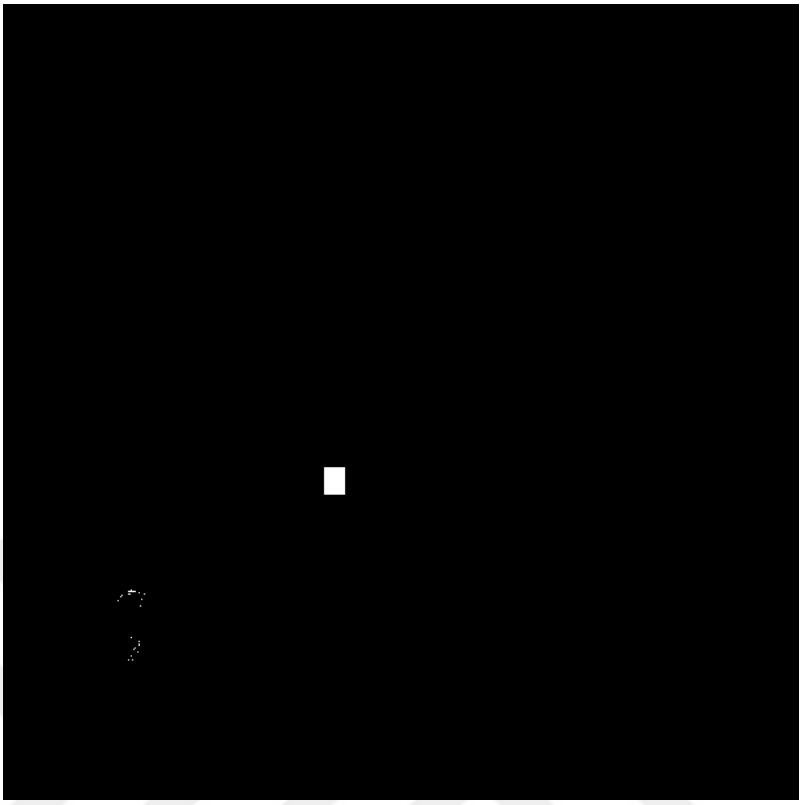
Şekil 2-6 : HSV Renk Uzayı Renk Skalası (Ghinmine & Sapkal, 2017)

OpenCv'nin verdiği imkanlarla kameradan alınan görüntü RGB renk uzayından HSV renk uzayına çevrilerek elde edilen çıktı cv2.inRange fonksiyonu yardımıyla işlenir ve

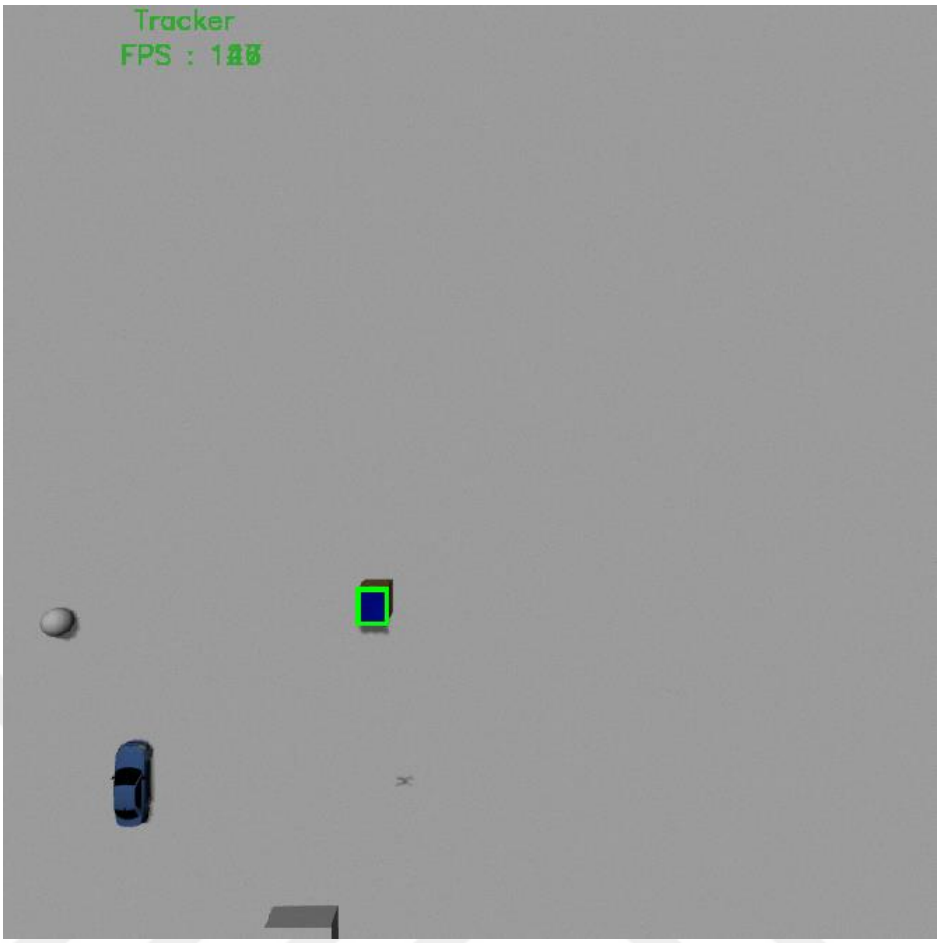
resim üzerinden istenilen renk aralığına sahip değerdeki alanlar ile bir maske elde edilir.Bu maske siyah-beyaz tek renkli bir resimdir. Akabinde cv2.findContours fonksiyonu yardımı ile köşeli olan beyaz alanlar tespit edilir ve resim üzerinde istediğimiz renkteki nesnelerin lokasyonları tespit edilir. Biz bu çalışmada hedef aracımız mavi renkli olduğu için mavi rengi ayrıştıracağız.



Şekil 2-7: HSV Renk uzayında sahnenin görüntüsü



Şekil 2-8 : Mavi renk filtresinin uygulama görüntüsü



Şekil 2-9: Tespit edilmiş kutu nesnesi

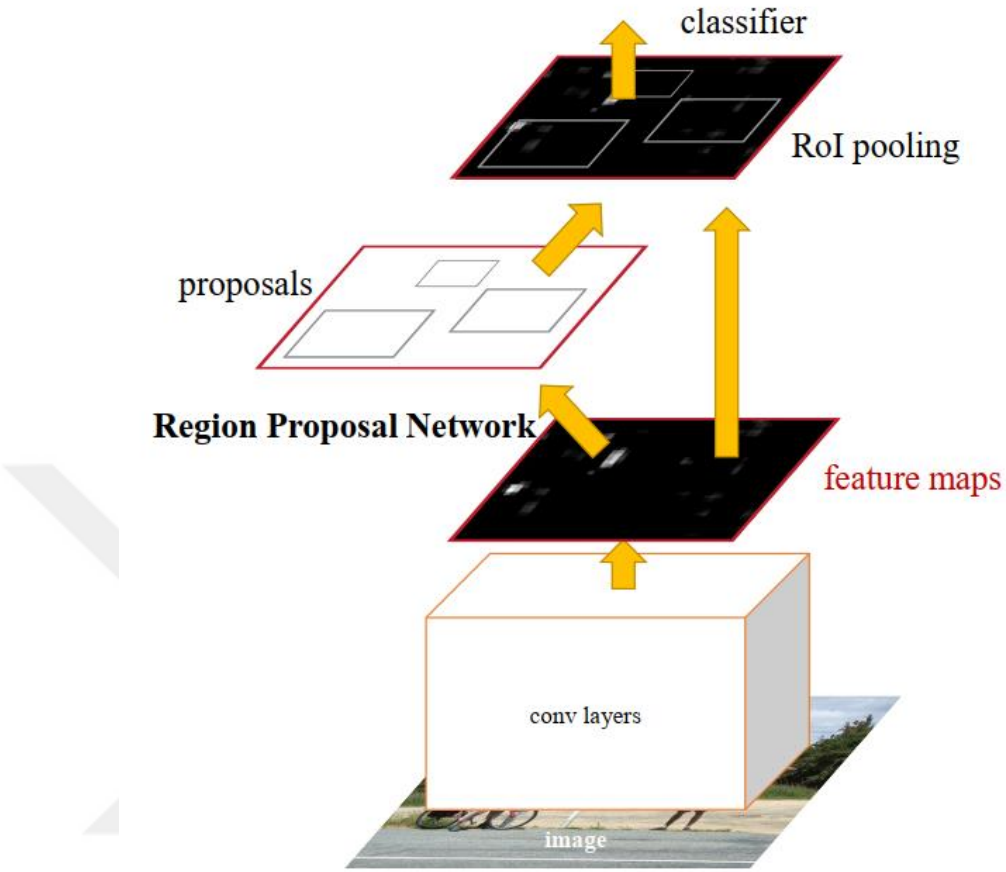
Eğer ekranda aynı renkte farklı cisimler bulunursa şayet, en büyük beyaz alan seçileceği için hedef araç yerine yanlış bir nesnenin seçilmesi muhtemeldir. Yapılan bu çalışmada bu senaryonun testleri gerçekleştirilecek ve sonuçları sonraki bir bölümde değerlendirilecektir.

2.3. Derin Öğrenme İle Nesne Tanıma

Görüntü işleme yöntemi ile yapalım nesne tanıma işleminin handikapları Bölüm2.2'de anlatılmıştır. Bu bölümde bu handikapların derin öğrenme algoritmalarıyla nasıl minimize edildiği hakkında bilgi verilecektir.

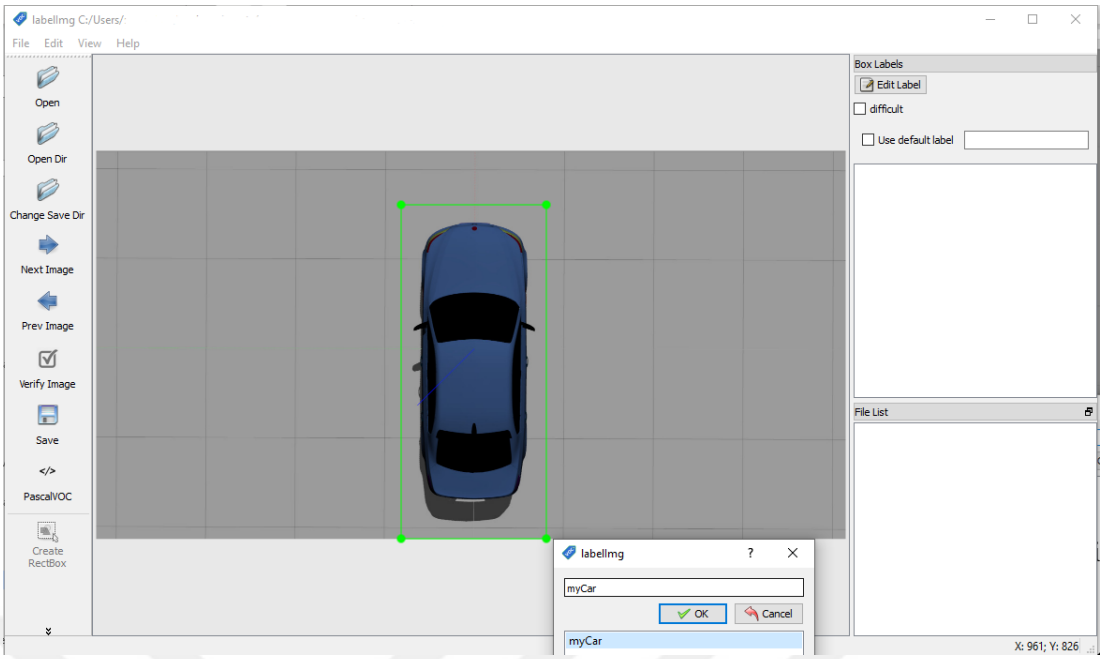
Derin öğrenmenin nesne algılama algoritmaları R-CNN ve sonrasında gelişmiş veriyonları olan Fast R-CNN, Faster R-CNN ile devam etmiştir. Bu çalışma algılama

doğruluğu daha yüksek ve aynı zamanda algılama süresi de yüksek olan Faster R-CNN ile yapılmıştır (Ren et al., 2017).



Şekil 2-10 : Faster R-CNN modeli (Ren et al., 2017)

Mevcutta Coco dataset ile eğitilmiş olan Faster R-CNN with Inception v2 modeli dataset içerisinde bulunan 90 adet nesne sınıfından olan nesneleri algılamakta ve resim üzerindeki lokasyon bilgisini çıktı olarak sağlamaktadır. Bu çalışmada hedef araç olan BMW 3.20 modeli transfer learning methodu ile yeniden eğitime tabi tutulmuştur. Bu işlemi gerçekleştirmek için Şekil 2.11'de görüldüğü gibi modelin üstten görüntülenen resimleriyle dataset oluşturulmuş, elde edilmiş resimler etiketleme işlemine tabi tutulmuştur. Akabinde tensorflow kütüphanesi yardımıyla 474 adet resim train için, 81 resim test için kullanılarak eğitim gerçekleştirilmiştir. Eğitim sonrası model %98 ve üzeri doğrulukla çalışmış, başarılı bir şekilde hedef aracı algılamış ve lokasyon bilgisini sisteme sağlamıştır.



Şekil 2-11 : Hedef aracın üstten görüntüsü ve etiketleme işlemi

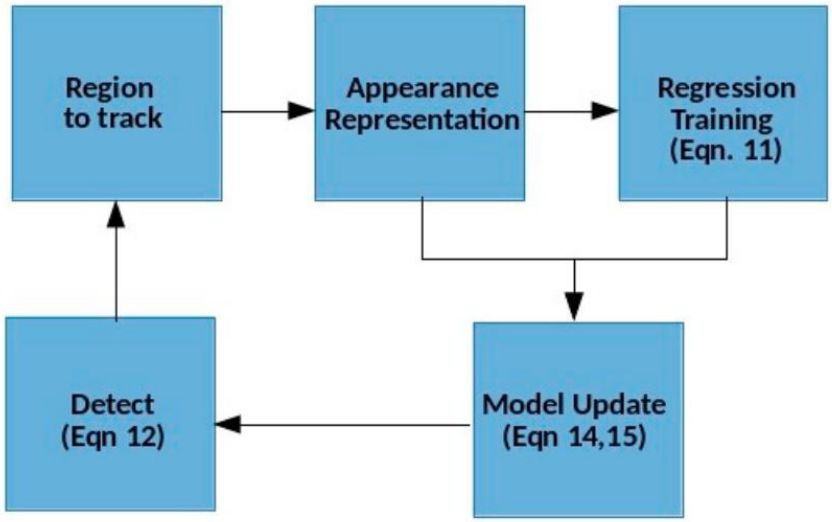
2.4. Nesne Takibi

Hareketli bir video üzerinde herhangi bir nesneyi hareket ettikçe izlemeyi amaçlayan disipline nesne takibi denir (Patel et al., 2018). Nesneler insanlar, hayvanlar, futbol maçında bir top gibi hareketli herhangi bir nesne olabilir. Nesne izleme tıbbi görüntüleme, trafik akışı analizi, otonom araçlar, insansız hava araçları gibi birçok alanda sıklıkla kullanılmaktadır.

Daha önceki bölümlerde bahsettiğimiz gibi nesneler video üzerinde tespit edilebilir ve bu videonun herhangi bir karesinde bu tespit işlemi tekrarlanarak nesne takibi gerçekleştirilebilir. Fakat bunun için her bir karenin baştan sonra taranması ve nesnenn tekrar tekrar tespit işleminin yapılması gerekmektedir ve bu işlem gücü olarak büyük maliyet demektir.

Nesne takibi teknolojisi sayesinde tespit edilen nesnenin bulunduğu dar alan(bounding box) OpenCV algoritmaları ile çok daha az işlem gücü gerektirerek bulunan nesnenin pixel hareketlerini hesaplayarak nesnenin bir sonraki konumunu hesaplar.

OpenCV üzerinde gerçekleştirilmiş olan KCF(Kernelised Correlation Filter) Tracker başarılı örneklerden bir tanesidir, hızlı ve isabetli sonuçlar vermektedir (George et al., 2018). KFC haricinde MIL, CSRT gibi OpenCV içerisinde gerçekleştirilmiş birkaç algoritma daha bulunmaktadır, farklı bir çalışmada detaylıca incelenebilir.

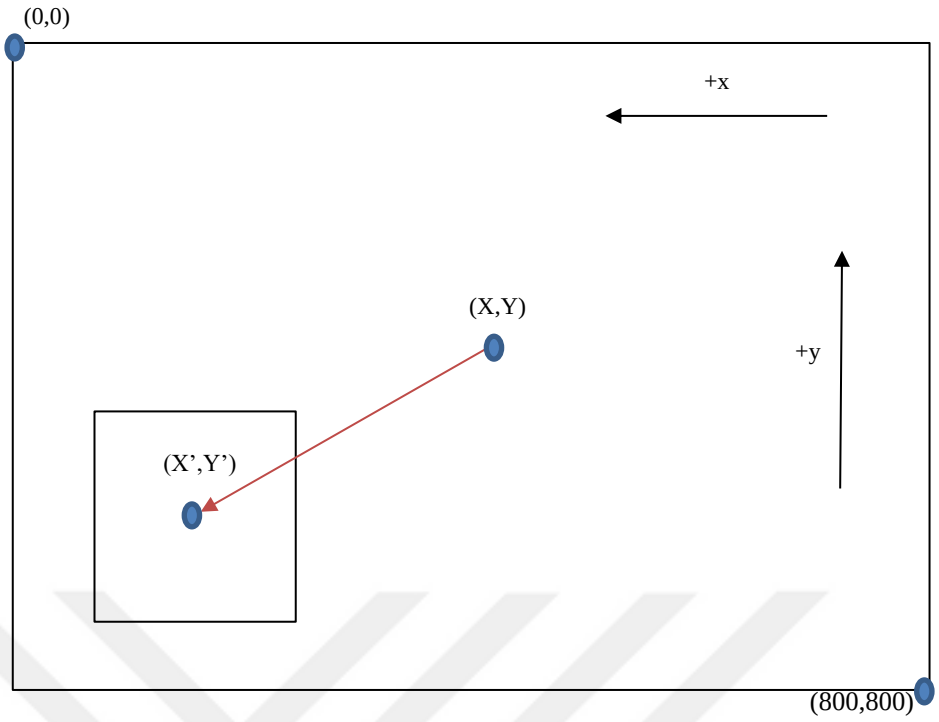


Şekil 2-12 : KCF Tracker Algoritması Blok Şeması (George et al., 2018)

Yapılan bu çalışmada İHA üzerinde görüntü işleme veya derin öğrenme yöntemleriyle algılanan aracın konumu ve çerçevesi KCF tracker fonksiyonuna verilmekte ve her bir karede tekrar tekrar nesnenin konumu güncellenerek, İHA'ya geri dönüş olarak sağlanmaktadır. Bu veriler büyük önem arzetmektedir, yanlış izleme veya nesnenin izini kaybetme durumunda tekrar nesne algılama yapılmalı veya İHA acil durum pozisyonuna alınmalıdır. Bu kimi zaman olduğu yere iniş olur ki zemin problemli olursa kırım yaşanabilir, diğer bir durum olarak İHA uçuşa başladığı yere geri döndürülmelidir. Bir sonraki bölümde İHA'nın bu edinilen veriler ışığında yaptığı hesaplamalar incelenecektir.

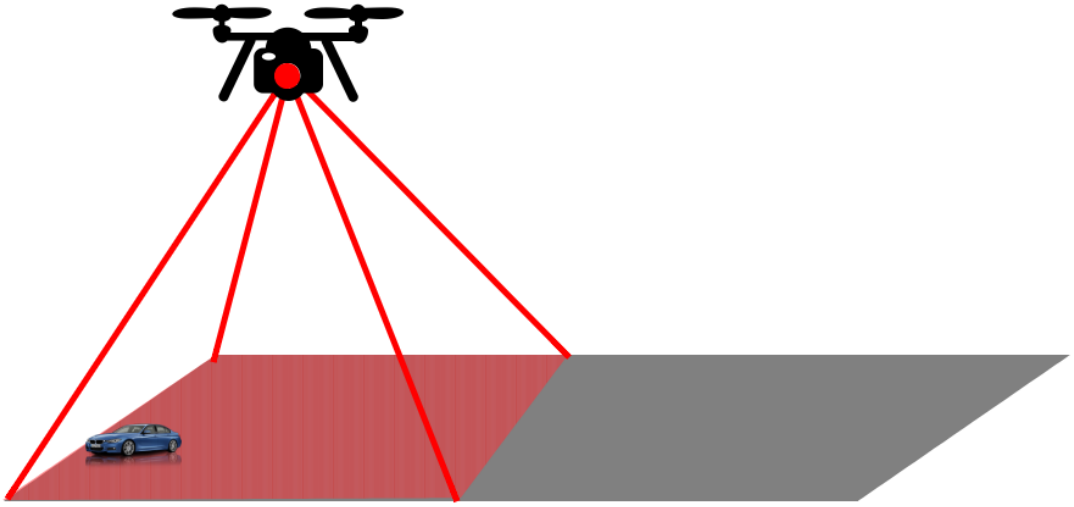
2.5. Rota Planlama

Bölüm 2.2 ve Bölüm 2.3'te bahsedildiği gibi hedef araç tespit edildikten sonra araç kamera görüntüsü üzerinde lokasyon elde edilmektedir. Simülasyon ortamına göre lokasyon bilgisi aşağıdaki gibidir.

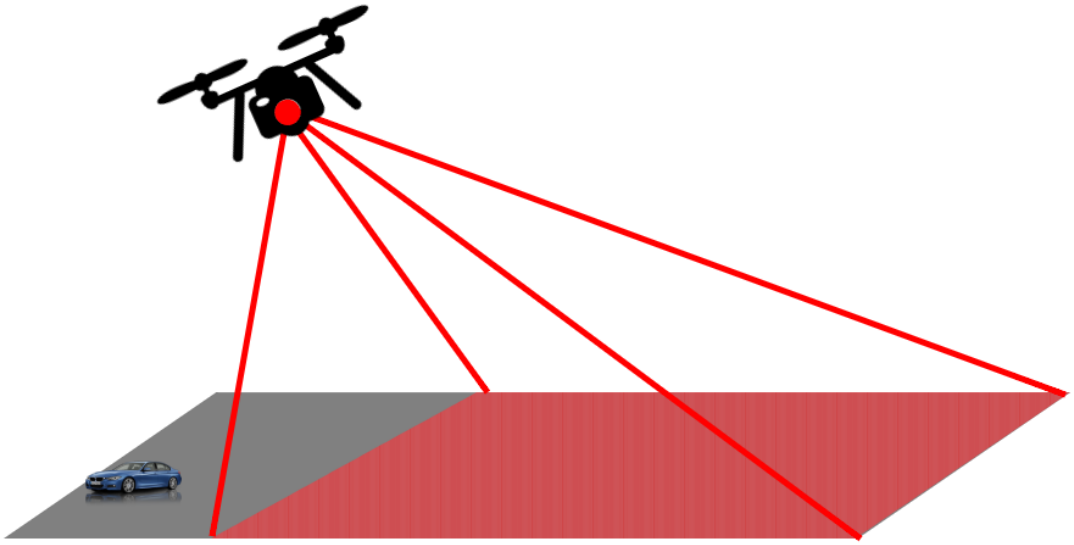


Şekil 2-13 : "Kamera koordinatları"

Şekil 2.14'te görüldüğü gibi İHA'nın kamerası düşey eksenli (z ekseni) 80 derecelik bir görüş açısına sahiptir. Yatay yönde (x ve y ekseninde) yüksek hızlarla (2 [m/s] ve üstü) hareket edilmek istendiğinde eğilme açısı $+5^{o}$ ve üzerine çıkmakta ve Şekil 2.15'te görüldüğü gibi hedef araç görüş açısından çıkmaktadır. Bu durum oluşması neticesinde nesne takip algoritması hedefi olan nesneyi kaybetmektedir. Bu sebeple yatayda çıkalabilecek olan max hız olan 2 [m/s] hız için PI denklemlerindeki Kp değerini kullanarak hızı limitlememiz gerekmektedir. Şekil 2.11'de görüldüğü gibi kameranın orta noktası (X:400,Y:400) değerine sahiptir. Nesnenin tespit edilebileceği en uzak nokta ise (X':0,Y':0), (X':0,Y':800), (X':800,Y':0), (X':800,Y':800) noktalarıdır. Hesaplanan ErrorX değeri maksimum 400 olmaktadır. Bu sebeple Kp değeri 1/200, Ki değeri ise 1/10000 olarak belirlenmiştir. Araç hızı max 2 [m/s] olarak belirlenmiştir. Ayrıca Z eksenindeki alçalma hareketi için sabit olarak 0.05 [m/s] hız kullanılmıştır.



Şekil 2-14: İHA görüntü açısı içerisinde bulunan nesne



Şekil 2-15: İHA görüntü açısı dışında kalan nesne

Bir sonraki bölüm itibari ile farklı test senaryolarında İHA ve hedef aracın hareketleri incelenmiş ve test sonuçları paylaşılmıştır.

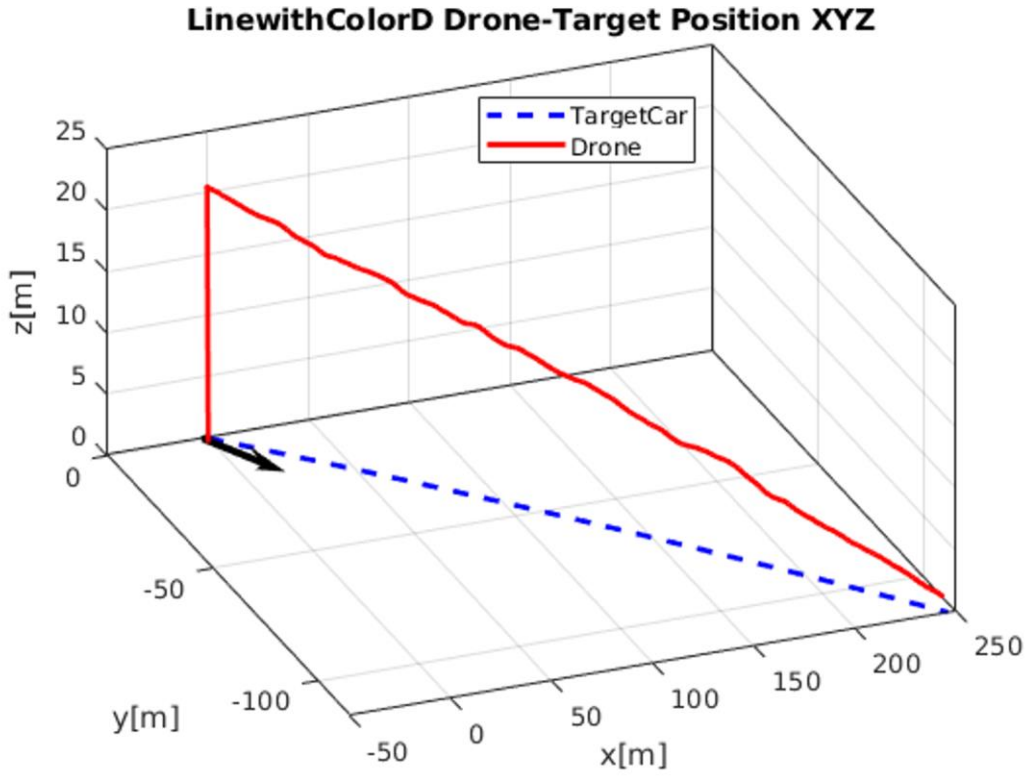
2.5.1. Görüntü İşleme İle Başarılı Nesne Yakalama

Gerçekleştirilmiş olan test düzeneğinde başlangıç anında İHA (-2,-2) koordinatlarında hedef araç ise sahnenin (0,0) koordinatlarında Şekil 2-16'da siyah ok şeklinde

gösterildiği gibi oryantasyonu θ_x=0.4 [rad] olacak şekilde yerleştirilmiştir. İHA 20 [m] yükseldikten sonra hedef araç hareketine başlamış, tanıma gerçekleşmiş ve İHA düşey eksende (Z) 0.05 [m/s] hızla alçalmaya ve aynı zamanda hedef aracı X, Y ekseninde takip etmeye başlamıştır.

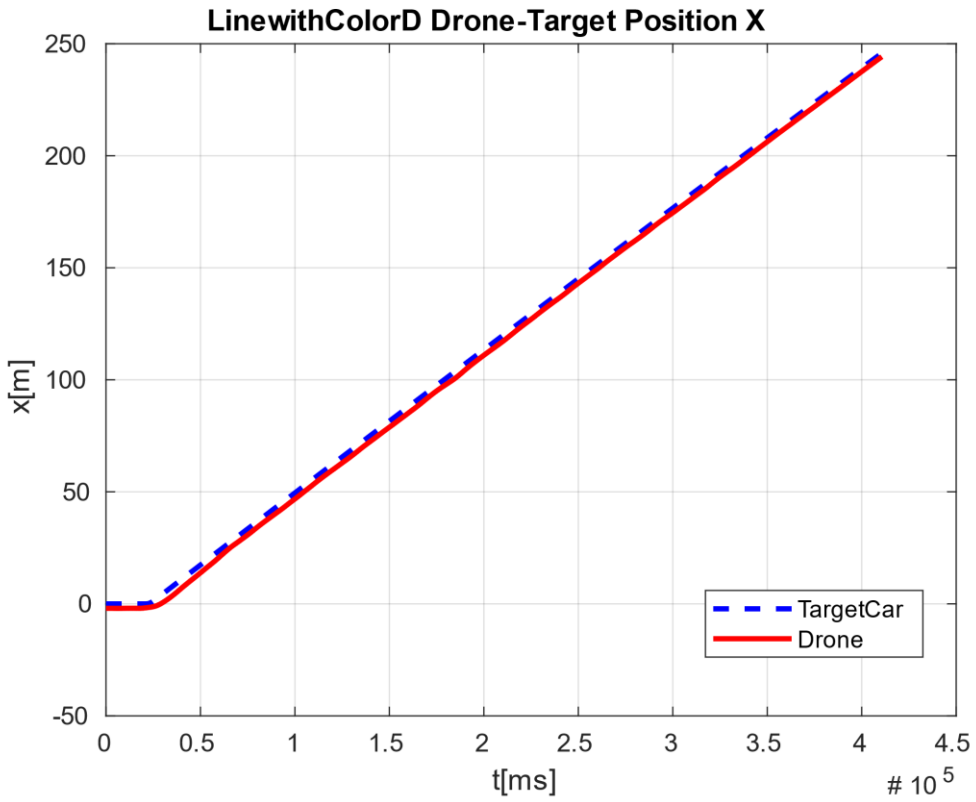
a. Düz Şekilli Rota

Bu test düzeneğinde hedef araç düz bir çizgi boyunca ilerlemiş İHA (250,-150) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-16'da 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-17'de XY eksenlerinde hareketleri, Şekil 2-18'de X ekseninde hareketleri, Şekil 2-19'da Y ekseninde hareketleri gösterilmiştir.

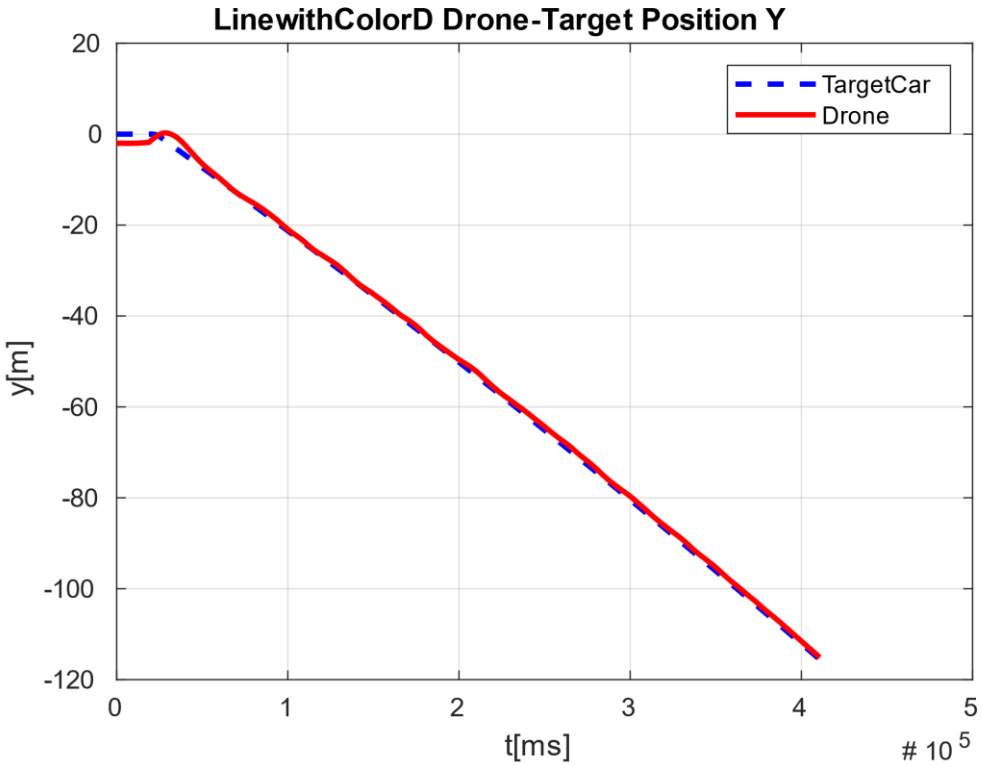


Şekil 2-16 : Düz giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi

Şekil 2-17: Düz giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



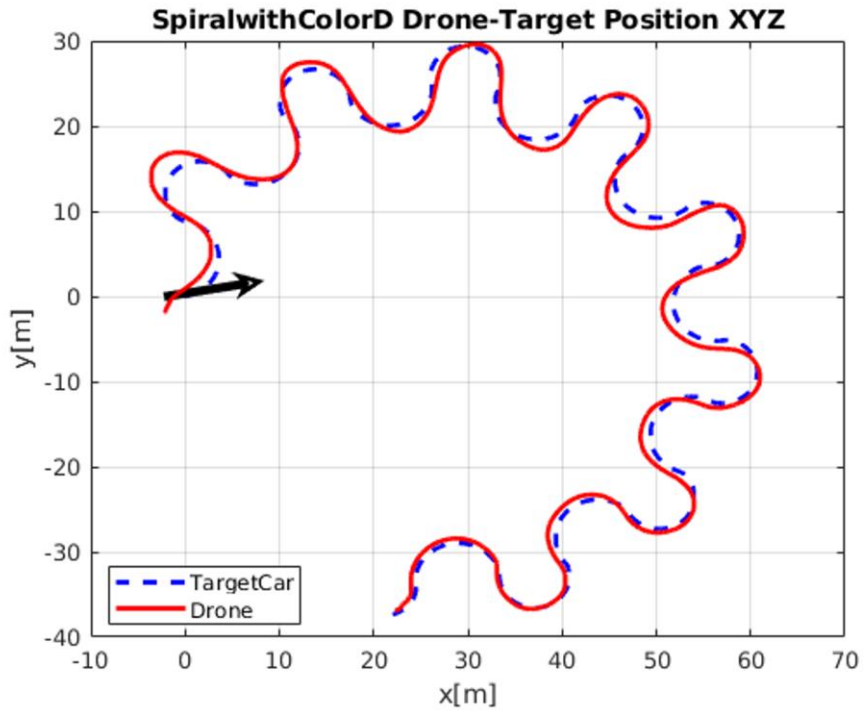
Şekil 2-18 : Düz giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



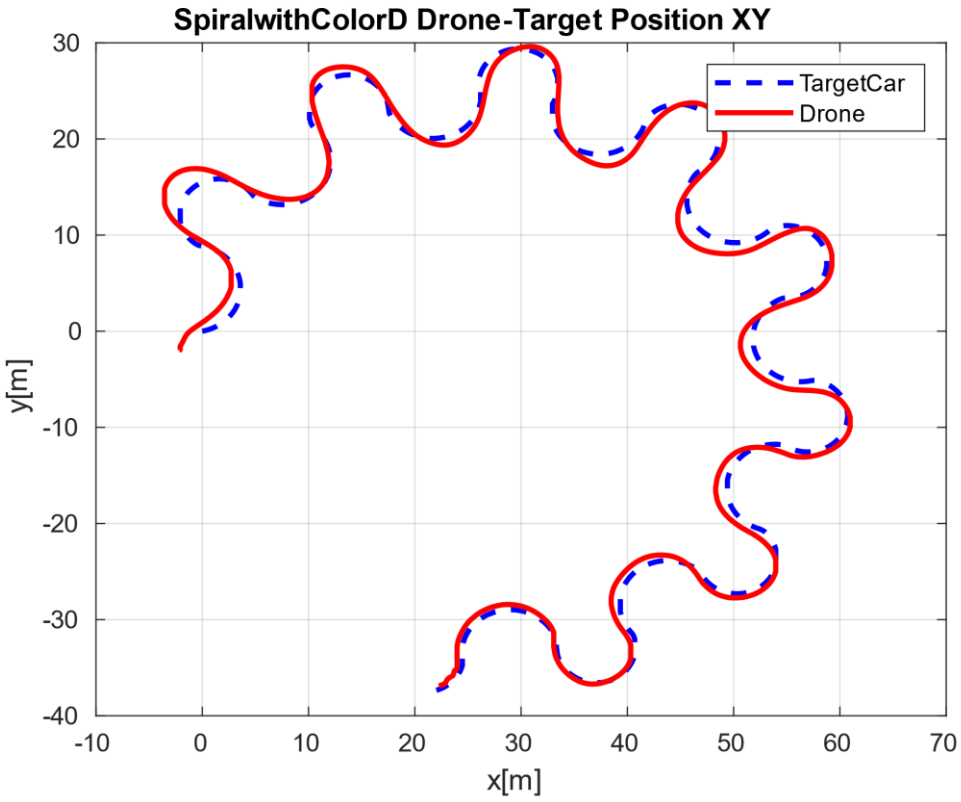
Şekil 2-19: Düz giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

b. Spiral Şekilli Rota

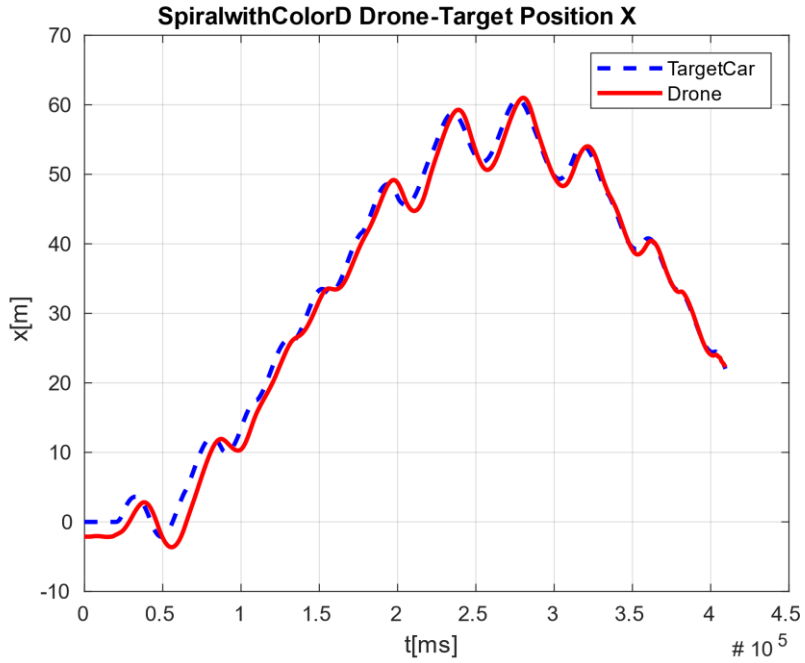
Bu test düzeneğinde hedef araç spiral bir rota boyunca ilerlemiş İHA (20,-40) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-20'de 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-21'de XY eksenlerinde hareketleri, Şekil 2-22'de X ekseninde hareketleri, Şekil 2-23'te Y ekseninde hareketleri gösterilmiştir.



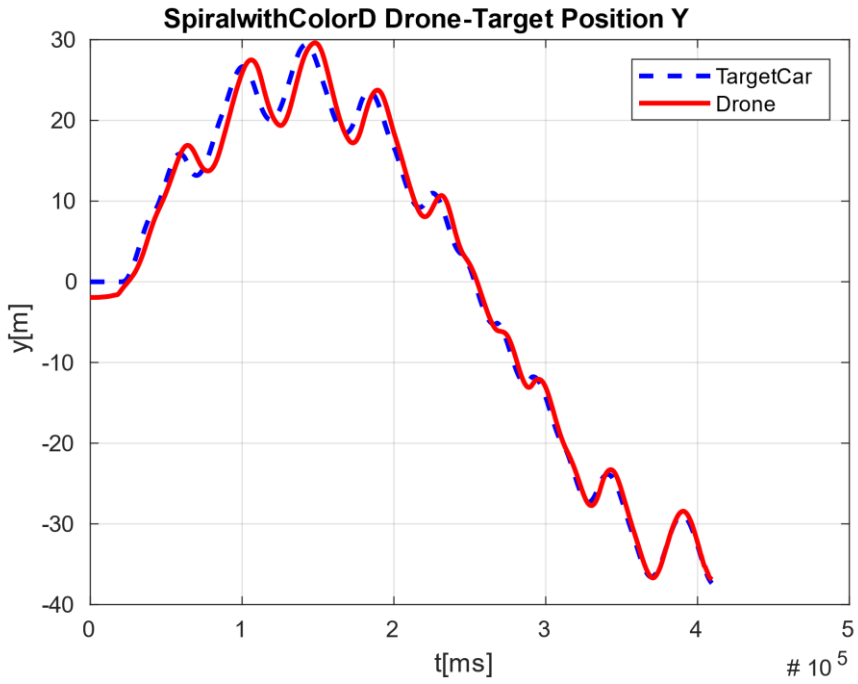
Şekil 2-20 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-21 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



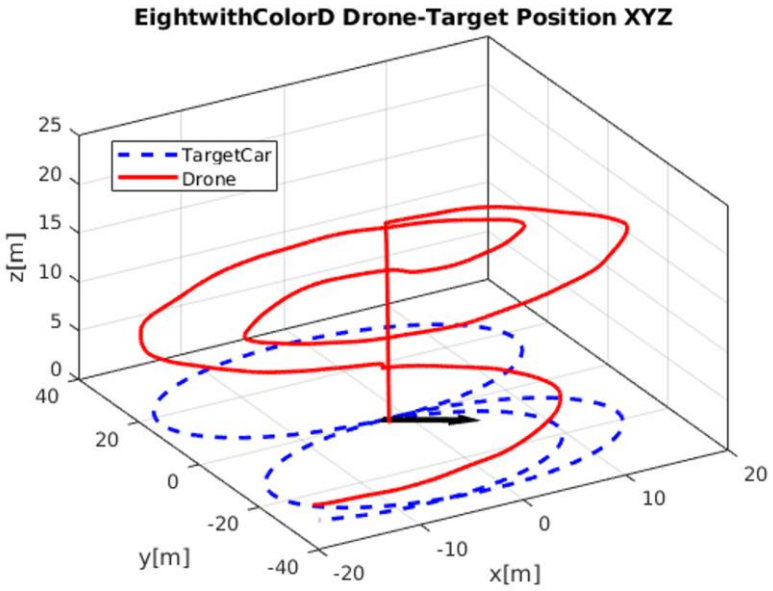
Şekil 2-22 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



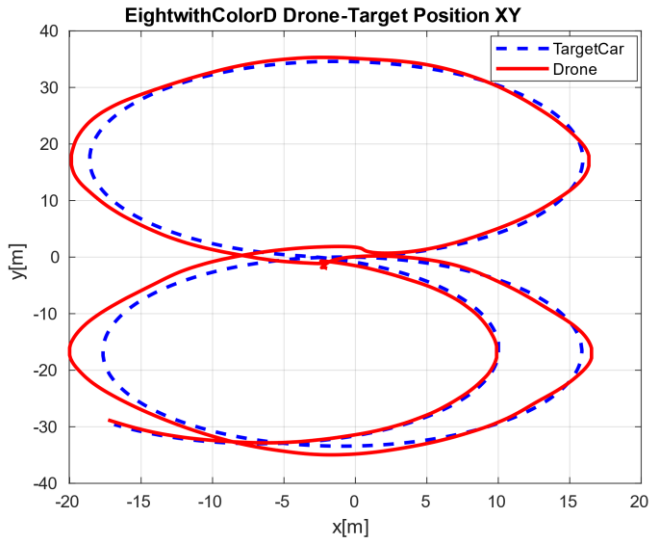
Şekil 2-23 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

c. Sekiz Şekilli Rota

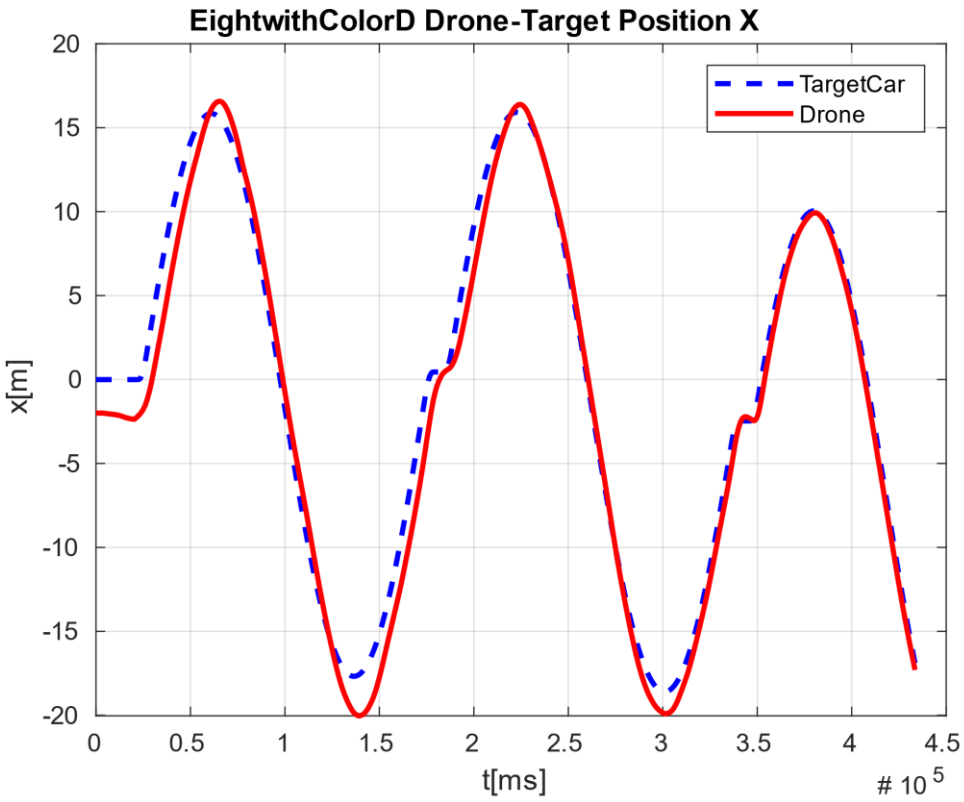
Bu test düzeneğinde hedef araç spiral bir rota boyunca ilerlemiş İHA (-15,-15) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-24'te 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-25'de XY eksenlerinde hareketleri, Şekil 2-26'da X ekseninde hareketleri, Şekil 2-27'de Y ekseninde hareketleri gösterilmiştir.



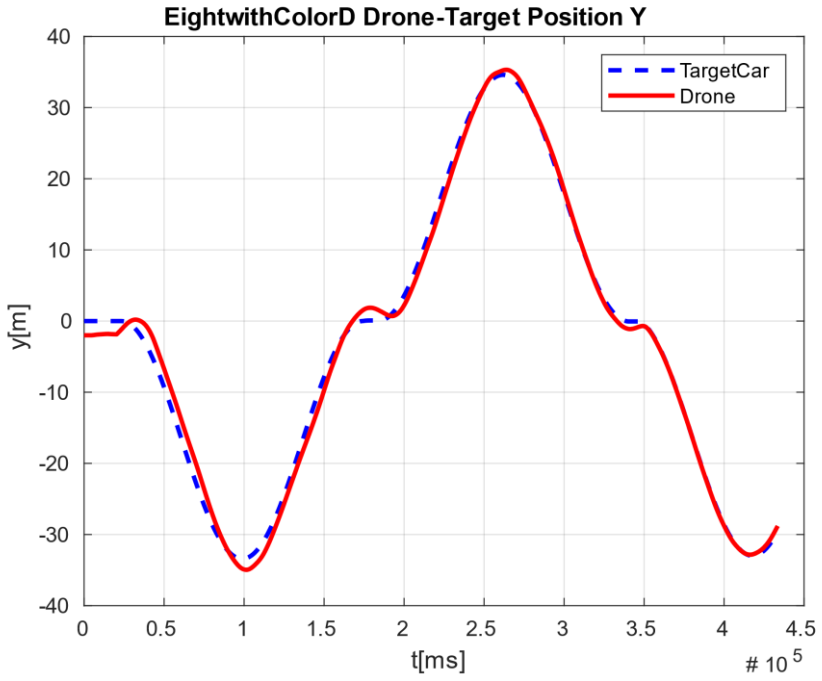
Şekil 2-24 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-25 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



Şekil 2-26 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



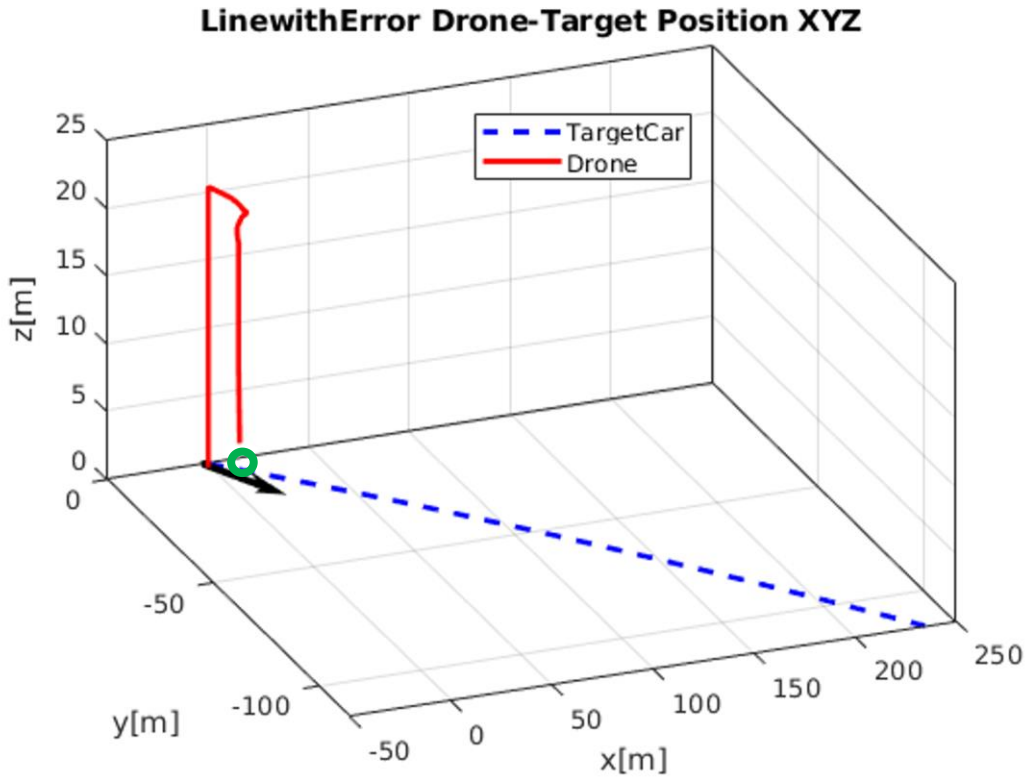
Şekil 2-27 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

2.5.2 Görüntü İşleme İle Başarısız Nesne Yakalama

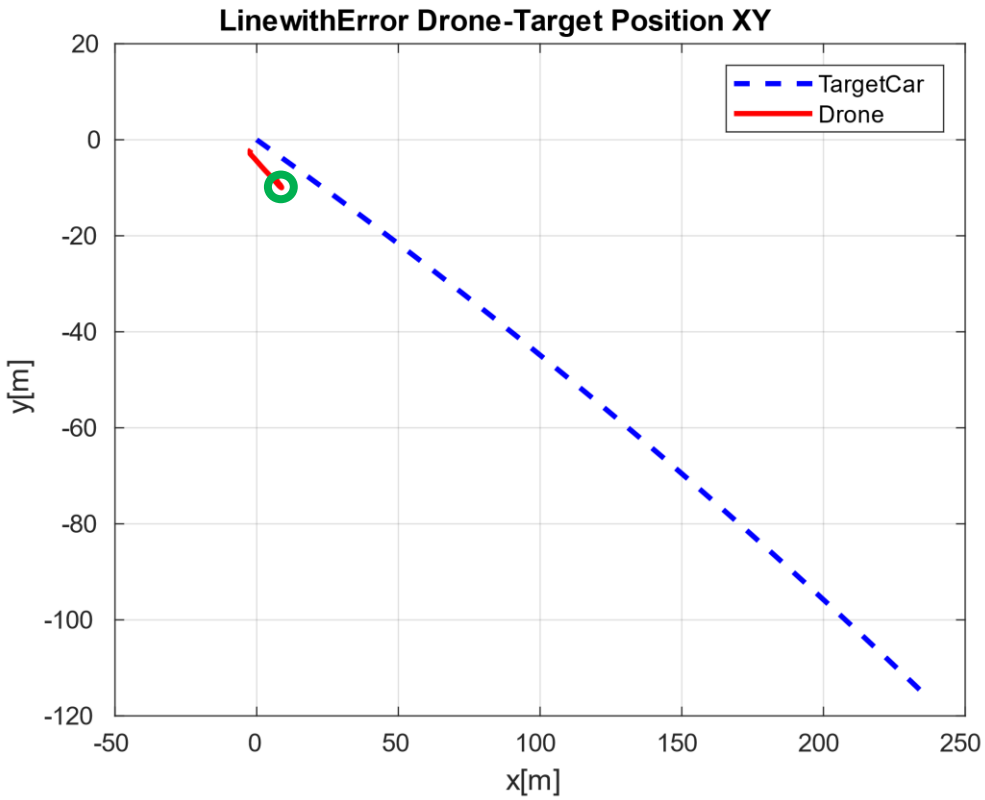
Gerçekleştirilmiş olan test düzeneğinde başlangıç anında İHA (-2,-2) koordinatlarında hedef araç ise sahnenin (0,0) koordinatlarında yerini almaktadır. Aynı zamanda hedef araç ile aynı renkte bulunan bir nesne Şekil 2-28'de ve Şekil 2-29'da yeşil daire içine alınmış şeklinde gösterilen (6,-6) koordinatlarına yerleştirilmiştir. İHA 20 [m] yükseldikten sonra hedef araç hareketine başlamış, hedef araç yerine yanlış bir işlem olarak yanlış nesne tanıma gerçekleşmiş ve İHA düşey eksende (Z) 0.05 [m/s] hızla diğer nesneye doğru alçalmaya başlamıştır. Hedef araç ise Şekil 2-28'de ki gibi rotasına devam etmiştir.

a. Düz Şekilli Rota

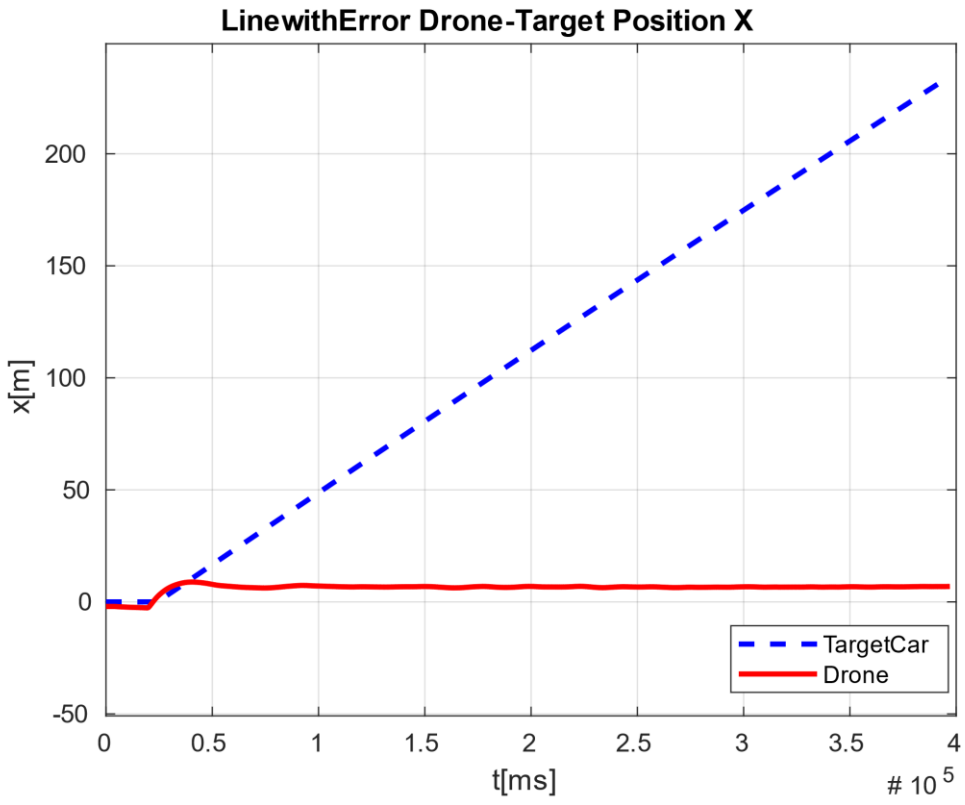
Bu test düzeneğinde hedef araç düz bir çizgi boyunca ilerlemiş İHA (6,-6) noktasında bulunan nesnenin üzerine doğru alçalmıştır. Şekil 2-28'de 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-29'da XY eksenlerinde hareketleri, Şekil 2-30'da X ekseninde hareketleri, Şekil 2-31'de Y ekseninde hareketleri gösterilmiştir.



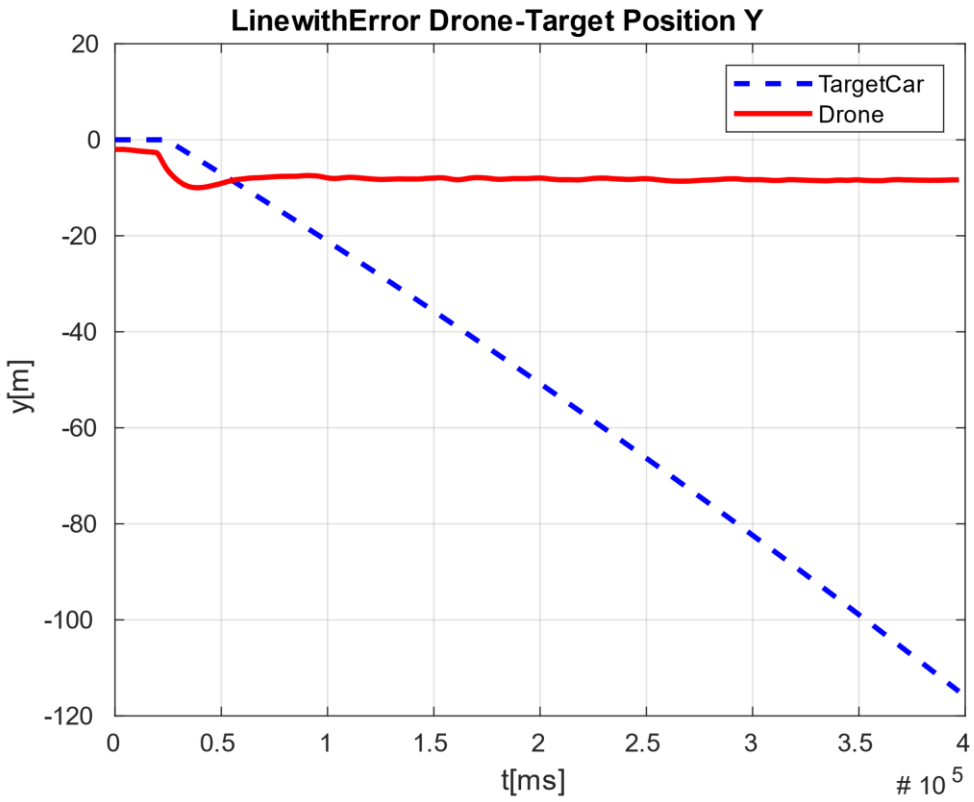
Şekil 2-28 : Düz giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-29 : Düz giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



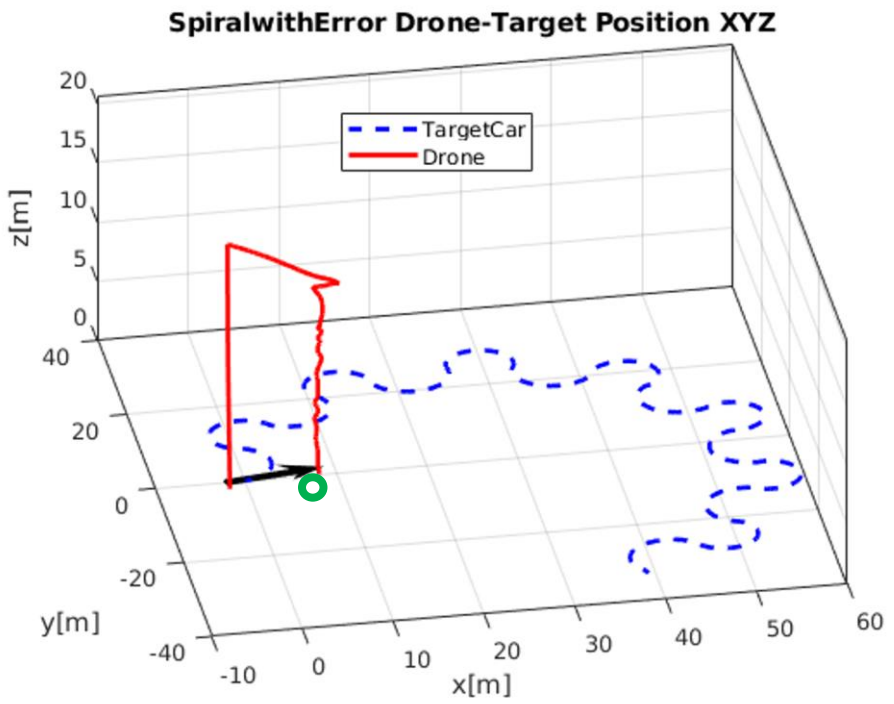
Şekil 2-30 : Düz giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



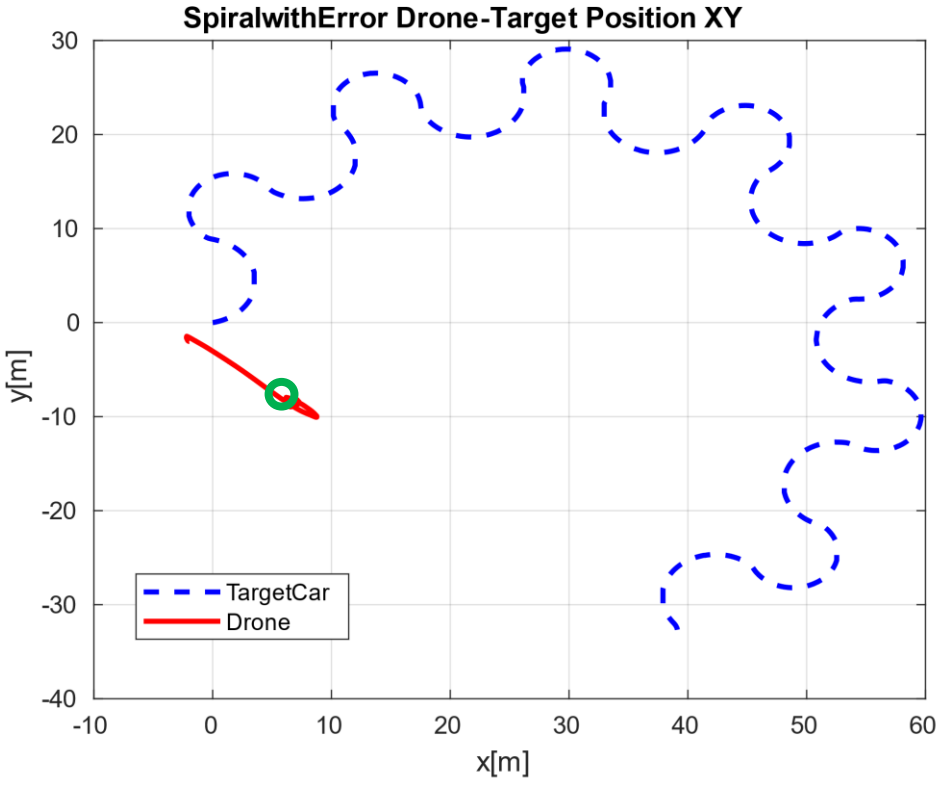
Şekil 2-31 : Düz giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

b. Spiral Şekilli Rota

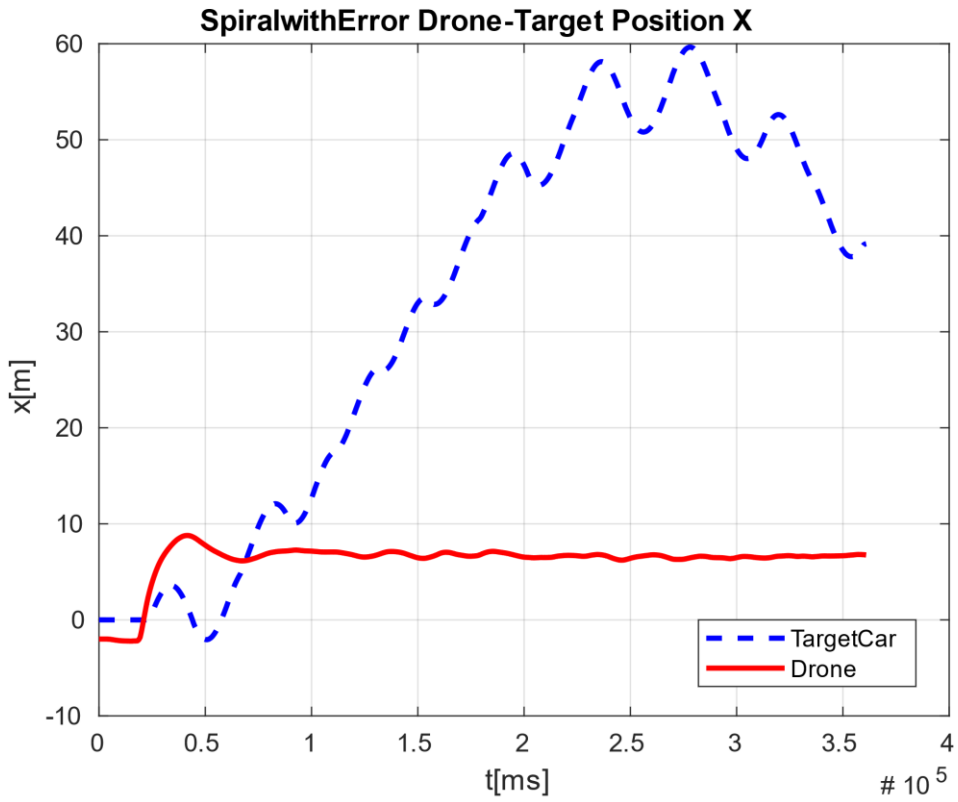
Bu test düzeneğinde hedef araç spiral şeklinde çizgi boyunca ilerlemiş İHA (6,-6) noktasında bulunan nesnenin üzerine doğru alçalmıştır. Şekil 2-28'de 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-29'da XY eksenlerinde hareketleri, Şekil 2-30'da X ekseninde hareketleri, Şekil 2-31'de Y ekseninde hareketleri gösterilmiştir.



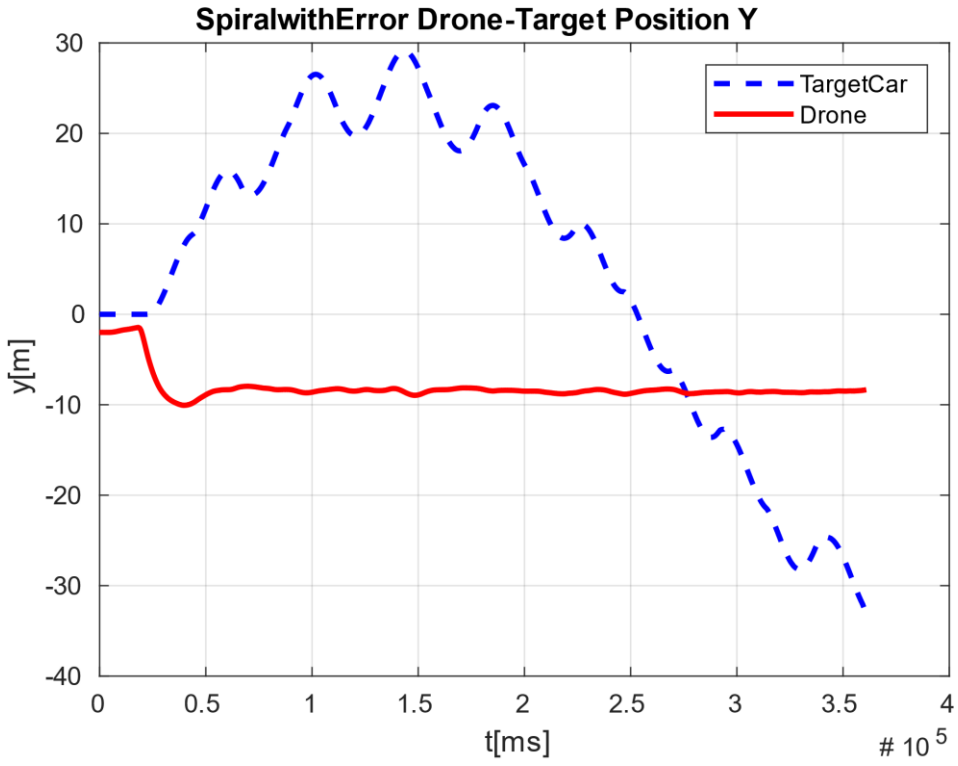
Şekil 2-32 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-33: Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



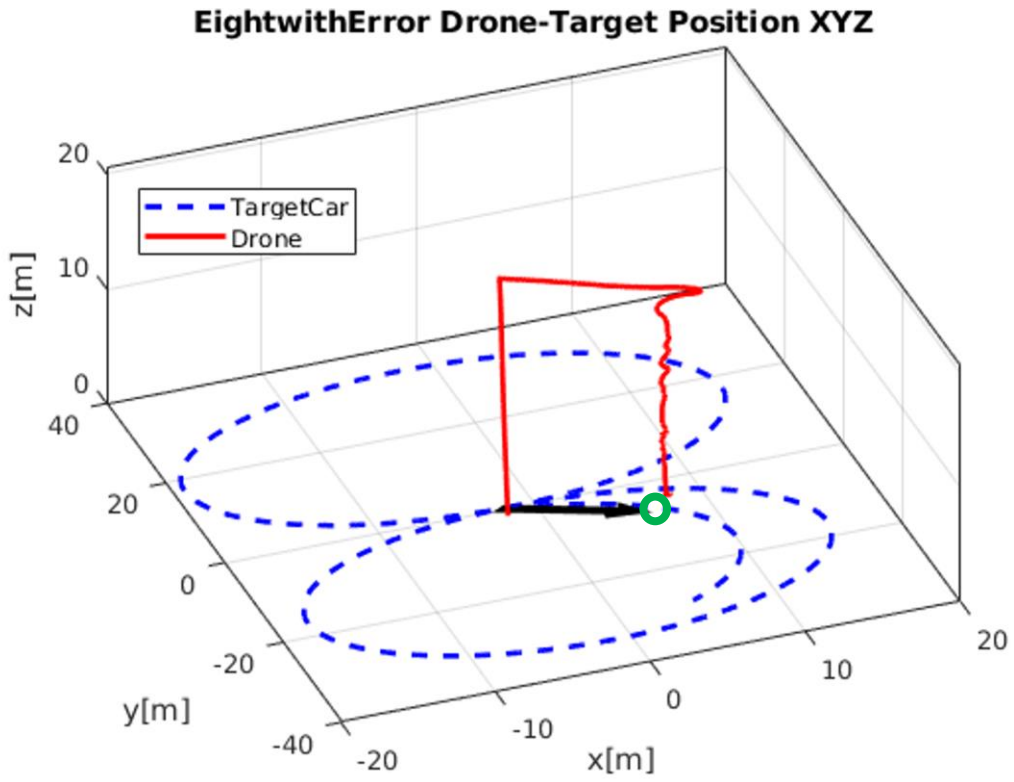
Şekil 2-34 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



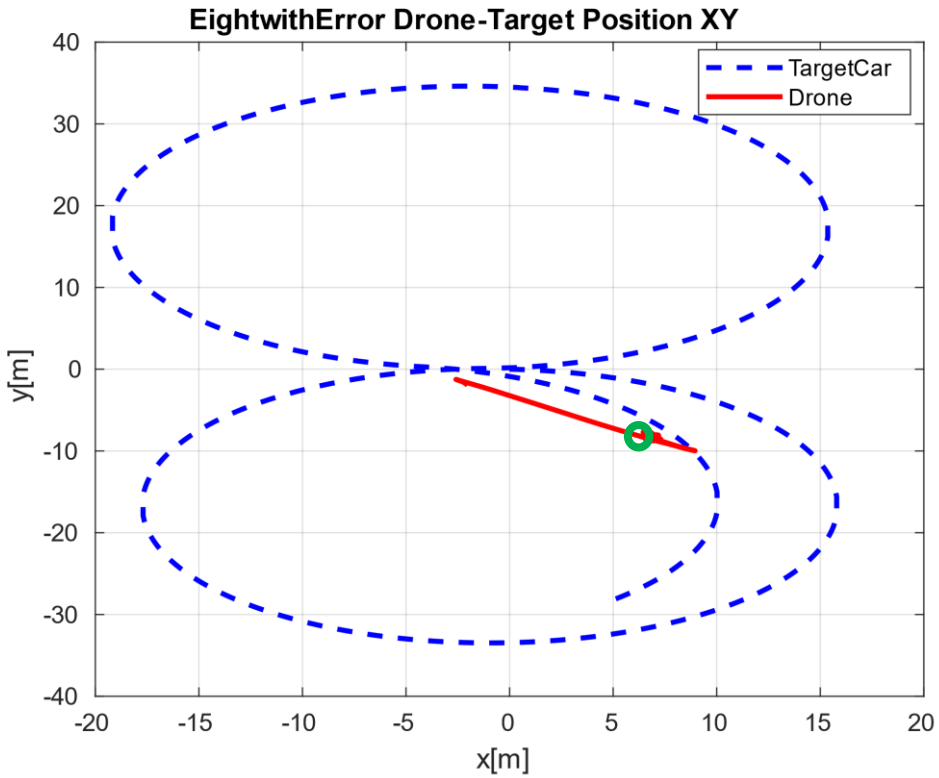
Şekil 2-35 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

c. Sekiz Şekilli Rota

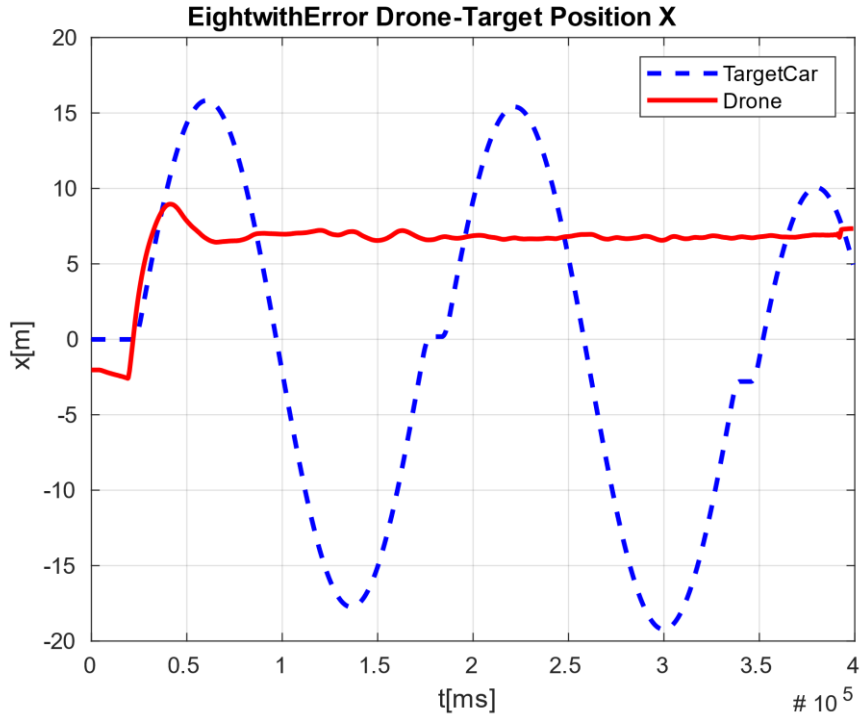
Bu test düzeneğinde hedef araç sekiz şeklinde çizgi boyunca ilerlemiş İHA (6,-6) noktasında bulunan kutunun üzerine doğru alçalmıştır. Şekil 2-28'de 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-29'da XY eksenlerinde hareketleri, Şekil 2-30'da X ekseninde hareketleri, Şekil 2-31'de Y ekseninde hareketleri gösterilmiştir.



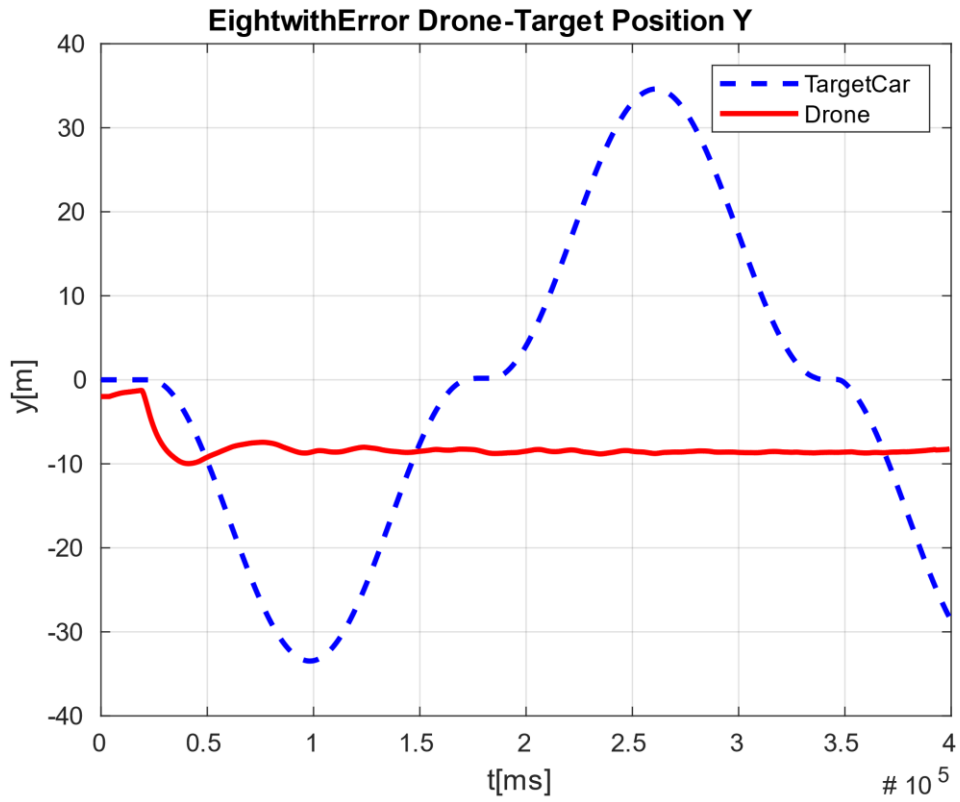
Şekil 2-36 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-37 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



Şekil 2-38 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



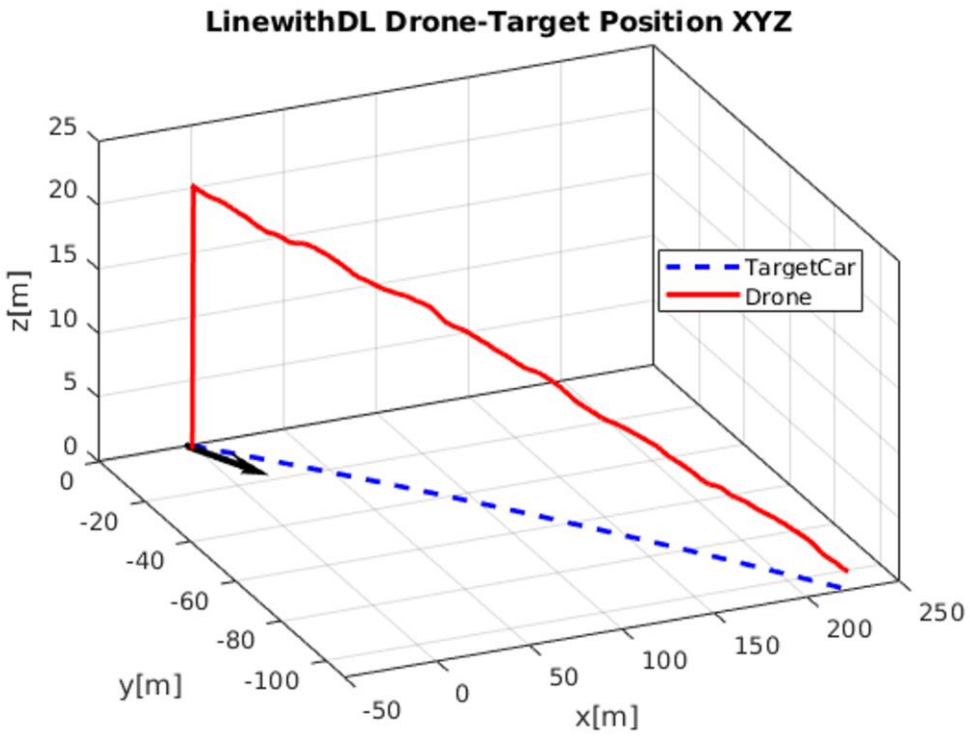
Şekil 2-39 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

2.5.3. Derin Öğrenme İle Başarılı Nesne Yakalama

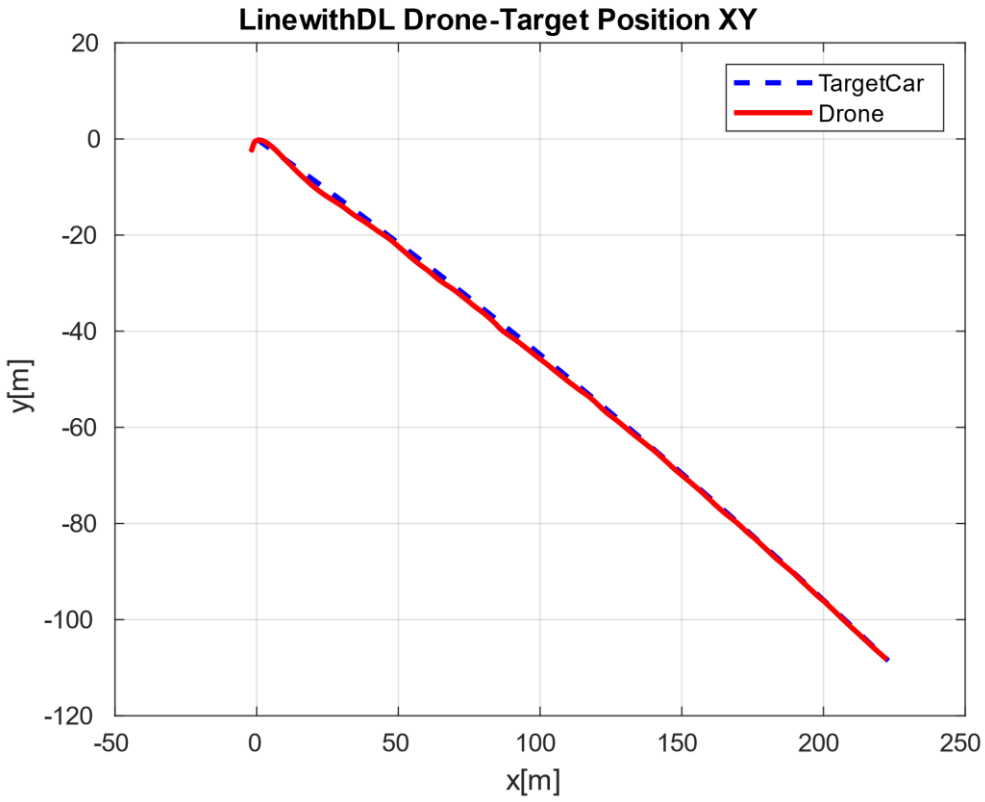
Gerçekleştirilmiş olan test düzeneğinde başlangıç anında İHA (-2,-2) koordinatlarında hedef araç ise sahnenin (0,0) koordinatlarında Şekil 2-16'da siyah ok şeklinde gösterildiği gibi oryantasyonu θ_x=0.4 [rad] olacak şekilde yerleştirilmiştir. İHA 20 [m] yükseldikten sonra hedef araç hareketine başlamış, tanıma gerçekleşmiş ve İHA düşey eksende (z) 0.05 [m/s] hızla alçalmaya ve aynı zamanda hedef aracı X ve Y ekseninde takip etmeye başlamıştır.

a. Düz Şekilli Rota

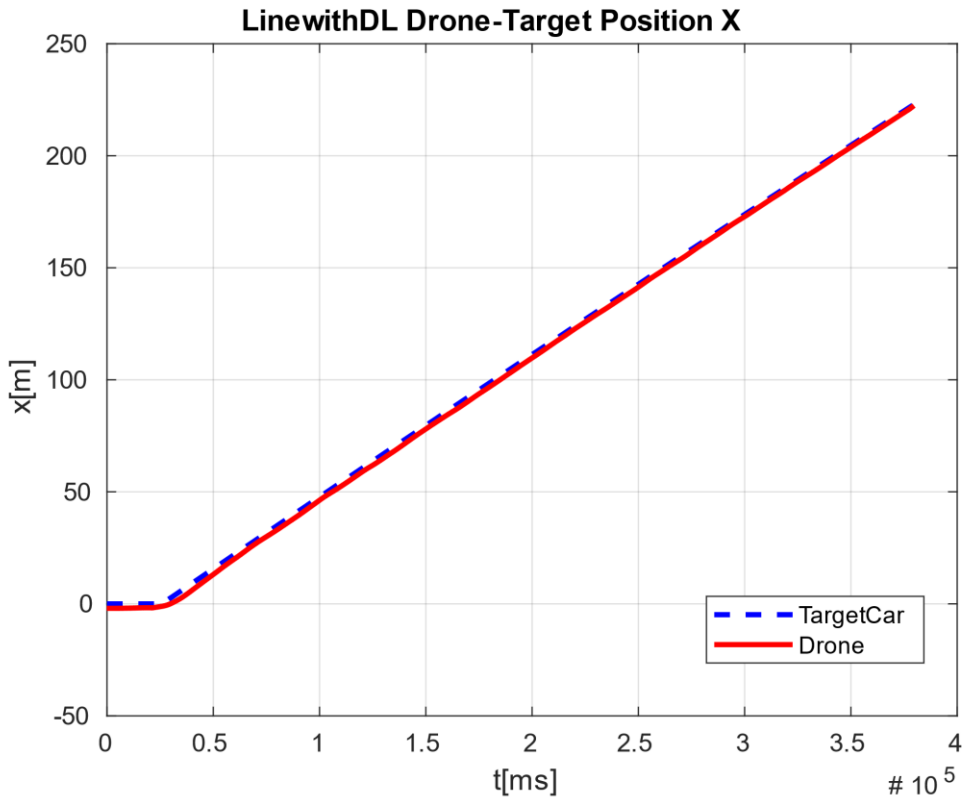
Bu test düzeneğinde hedef araç düz bir çizgi boyunca ilerlemiş İHA (230,-110) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-16'da 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-17'de XY eksenlerinde hareketleri, Şekil 2-18'de X ekseninde hareketleri, Şekil 2-19'da Y ekseninde hareketleri gösterilmiştir.



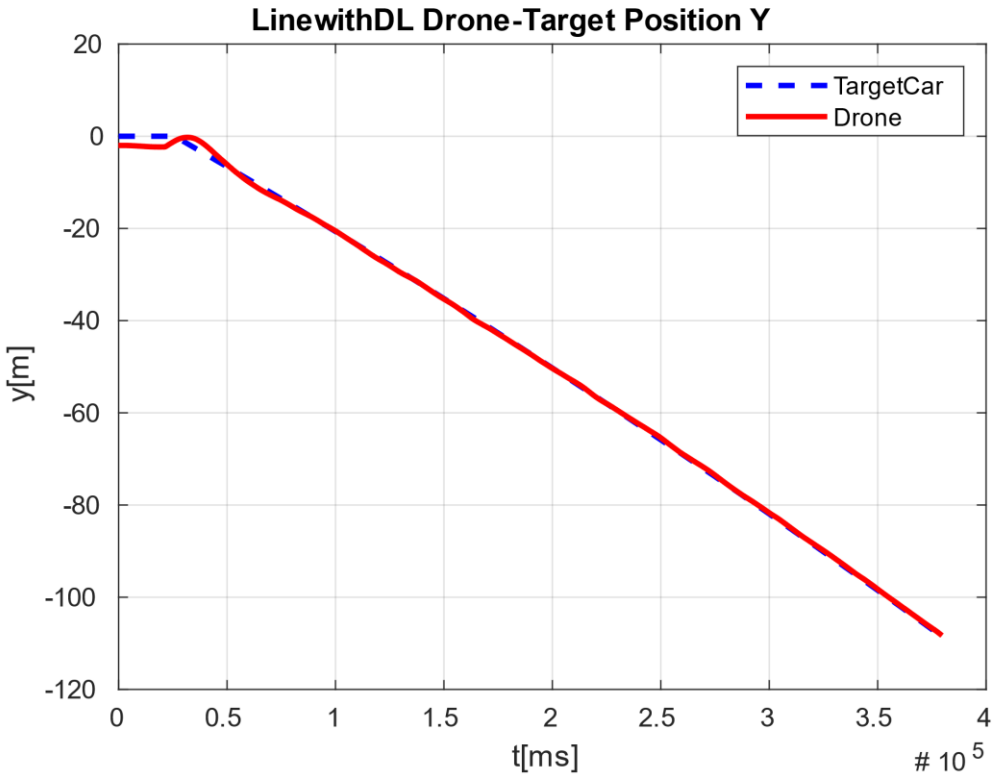
Şekil 2-40 : Düz giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-41 : Düz giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



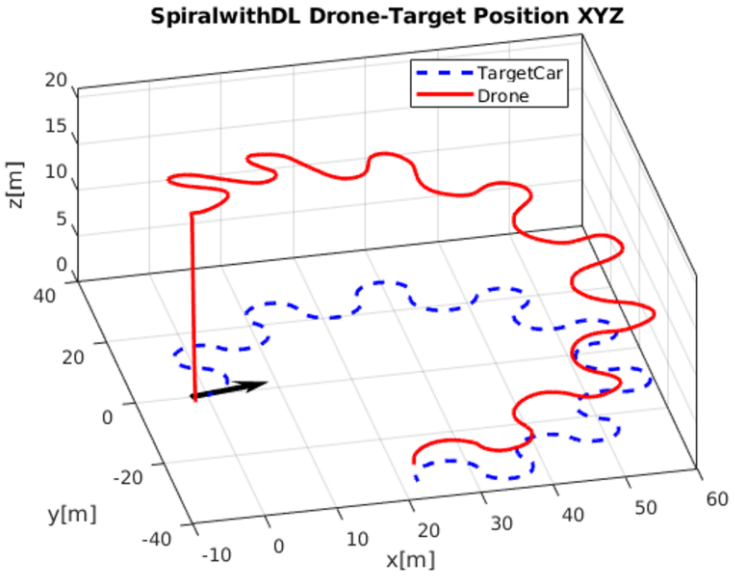
Şekil 2-42 : Düz giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



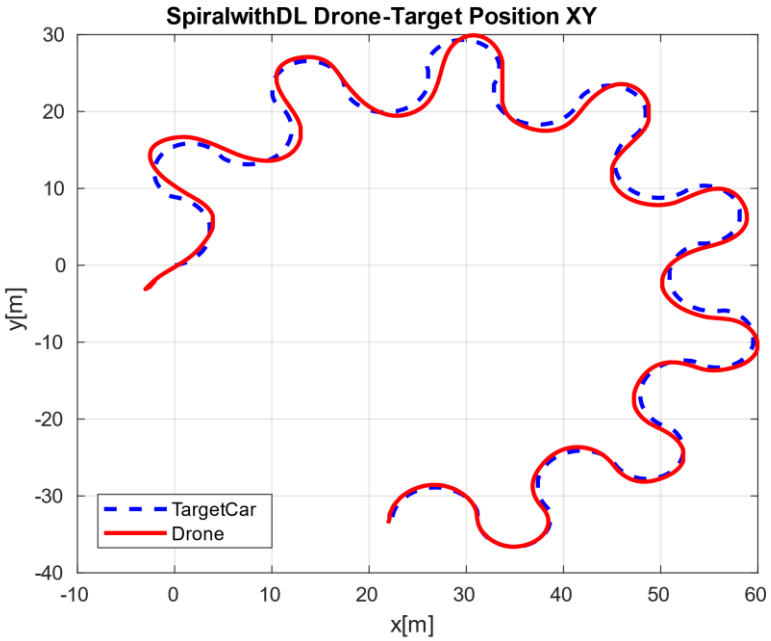
Şekil 2-43 : Düz giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

b. Spiral Şekilli Rota

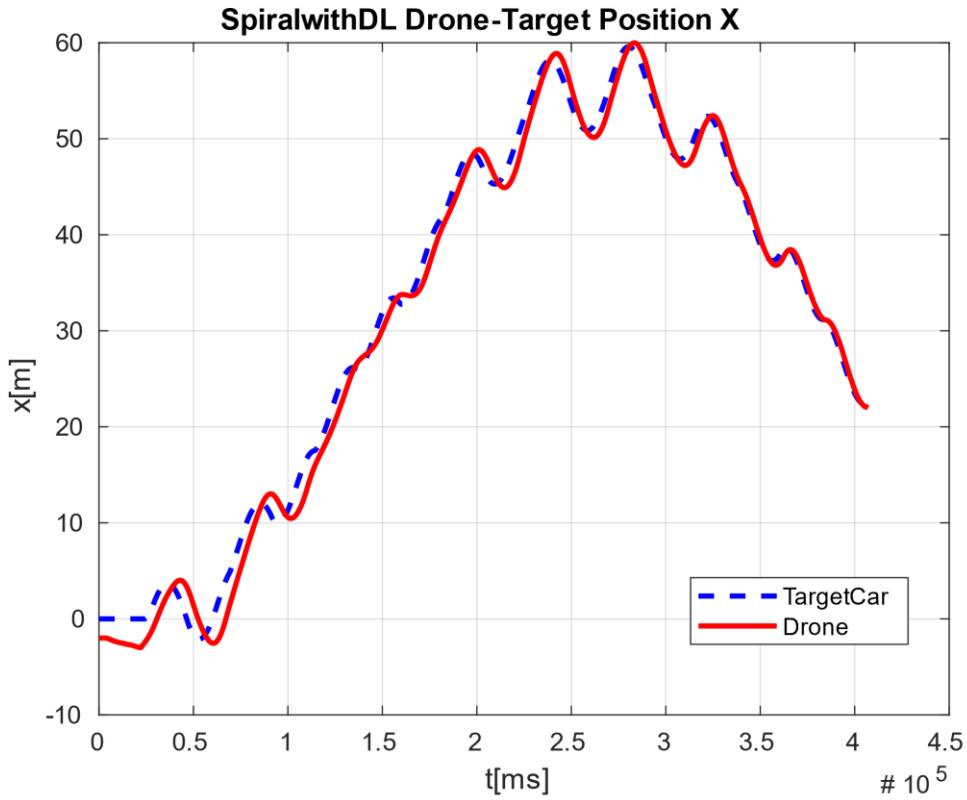
Bu test düzeneğinde hedef araç spiral bir rota boyunca ilerlemiş İHA (22,-35) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-20'de 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-21'de XY eksenlerinde hareketleri, Şekil 2-22'de X ekseninde hareketleri, Şekil 2-23'te Y ekseninde hareketleri gösterilmiştir.



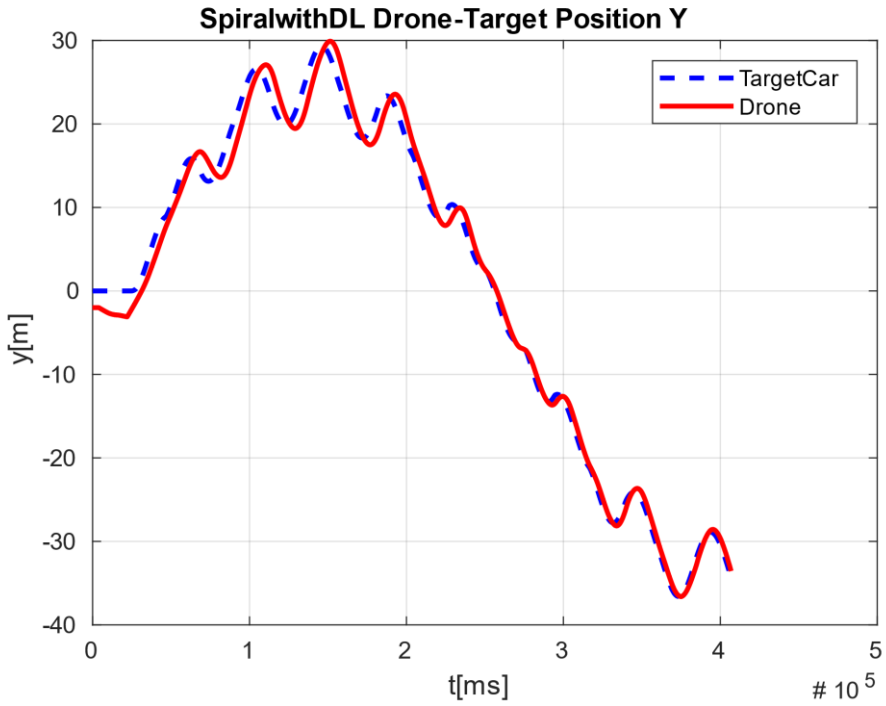
Şekil 2-44 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-45 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



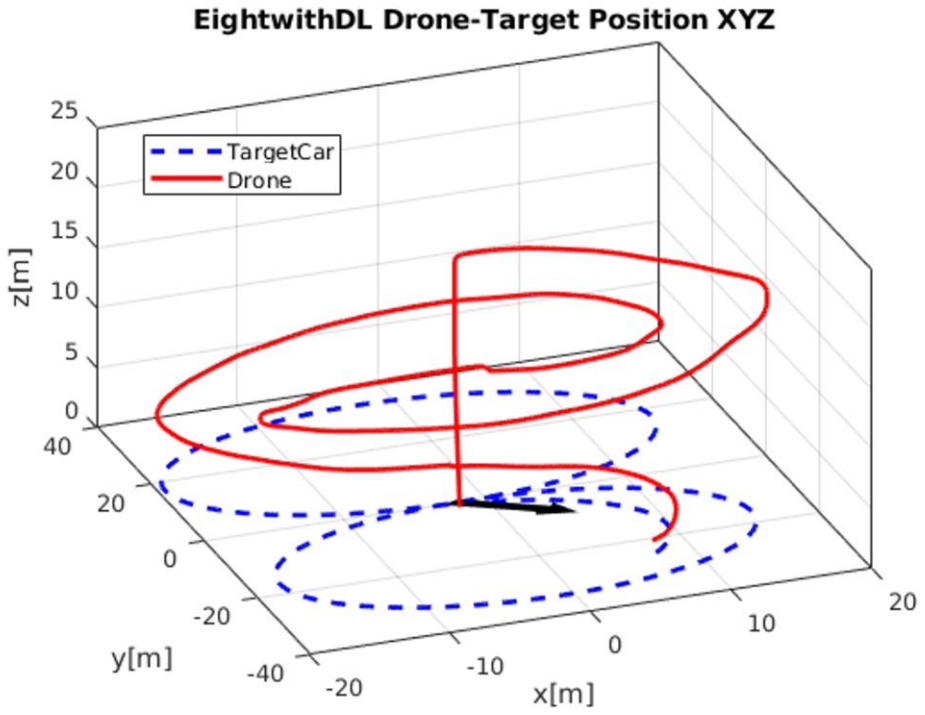
Şekil 2-46 : Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin X eksenindeki hareketi



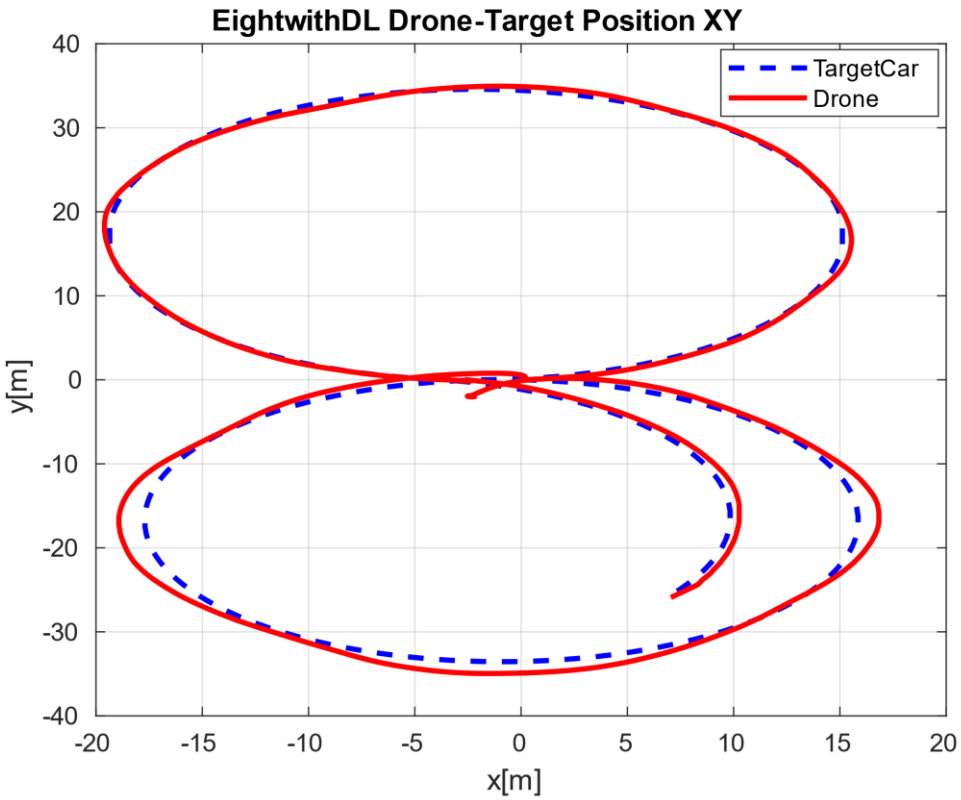
Şekil 2-47 :Spiral şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

c. Sekiz Şekilli Rota

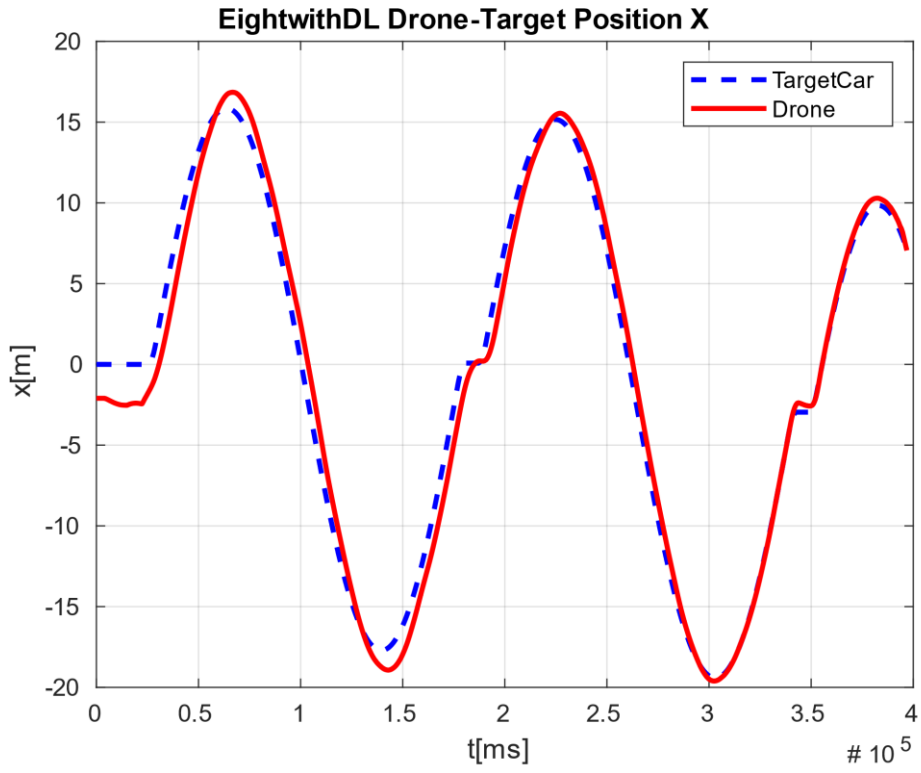
Bu test düzeneğinde hedef araç spiral bir rota boyunca ilerlemiş İHA (-15,-15) noktasına kadar alçalarak takibini başarıyla gerçekleştirmiştir. Şekil 2-24'te 3 eksenli grafikte araçların konumları ve hareket rotaları, Şekil 2-25'de XY eksenlerinde hareketleri, Şekil 2-26'da X ekseninde hareketleri, Şekil 2-27'de Y ekseninde hareketleri gösterilmiştir.



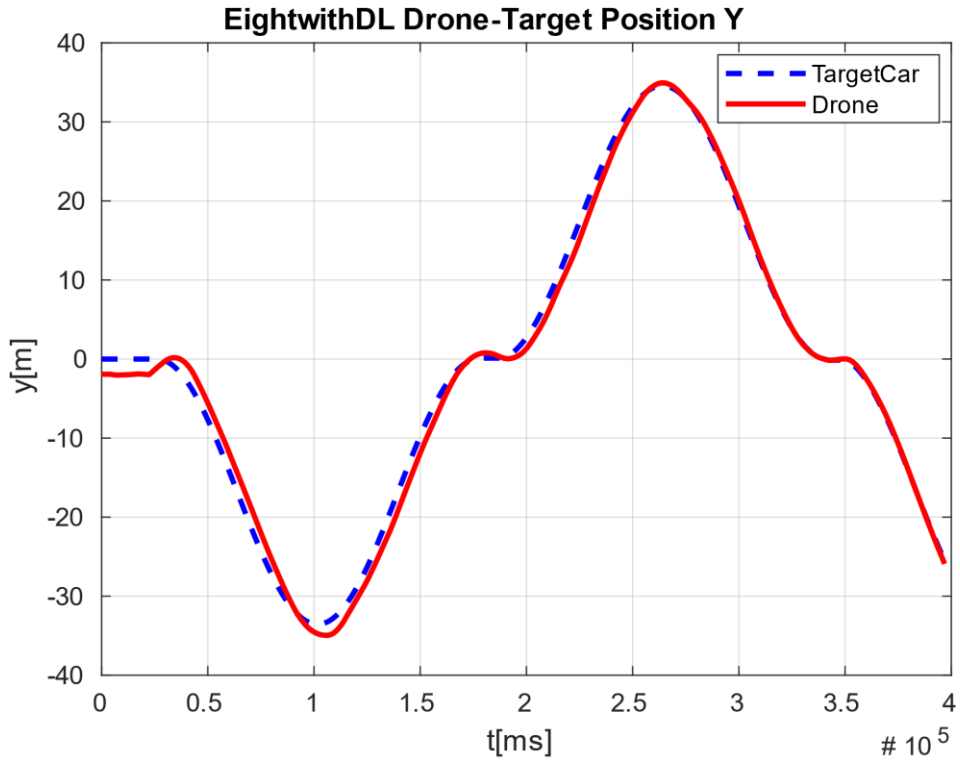
Şekil 2-48 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XYZ eksenindeki hareketi



Şekil 2-49 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin XY eksenindeki hareketi



Şekil 2-50 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi



Şekil 2-51 : Sekiz şeklinde giden nesneyi takip eden İHA ve nesnenin Y eksenindeki hareketi

ÜÇÜNCÜ BÖLÜM

3. SONUÇLAR VE ÖNERİLER

Seyir halindeyken tespit ettiği hareketli bir nesneyi İHA ile X,Y,Z eksenlerindeki hareketiyle alçalarak yakalamayı hedefleyen bu projede literatürde daha önce gerçekleştirilmiş çalışmalardan farklı olarak hedef tespitinin görüntü işleme teknikleri yerine son yıllarda tespit işlemlerinde sıkça kullanılan derin öğrenme teknikleriyle gerçekleştirmek ve bu iki tekniğin karşılaştırması ve sonuçlarını ortaya koymak amaçlanmıştır. Bu amaç doğrultusunda Coco dataset üzerine yeniden eğitilmiş Faster R-CNN derin öğrenme modeli kullanımı hedeflenmiştir.

Gazebo simülasyon ortamında otonom olarak hareket eden İHA alt tarafında bulunan 80 derecelik bakış açısı bulunan kamerada tespit ettiği hareketli nesneyi görüntü işleme ve derin öğrenme teknikleri kullanarak tespit etmiş ve OpenCV KCF Tracker takip algoritması aracılığıyla takip ederek aracın üzerine iniş sağlamıştır.

Renk algılama görüntü işleme tekniğiyle yapılan tespit etme işleminin ortamda aynı renkte farklı bir nesne bulunması halinde yanlış seçim yaptığı gözlenmiş, Faster R-CNN derin öğrenme modeli ile yapılan tespit işleminin daha isabetli olduğu gözlemlenmiştir. Nesnenin doğru tespit edilme oranı Gazebo simülasyon ortamında %98 ve üzeri olmuştur. Coco dataset'in içerisinde bulunan nesne sınıflarına ait nesnelerin çokça bulunduğu bir sahnede daha düşük isabet oranları elde edilebilir çünkü coco dataset içerisindeki resimler tamamen insan bakış açısıyla yandan çekilmiş resimlerden oluşmaktadır. Bunun aksine hem coco dataseti hem de DOTA veya COWC gibi üstten araç görüntülerinden oluşan datasetler kullanılarak eğitilen bir model daha isabetli sonuçlar verebilir.

Farklı rota şekillerinde yapılan her bir takip işlemi İHA tarafından başarıyla gerçekleştirilmiş, keskin dönüşler dahi olsa sorunsuzca takip işleminin devam ettiği görülmüştür.

3 eksende gerçekleştirilen takip işleminde daha yüksek hızlara çıkabilmek için İHA altında bulunan kameranın Gimbal ile desteklenmesi gerekmektedir, bu sayede İHA 5^{o} ve üzerinde eğimle hareket etmek istediğinde kameranın açısı değişmeyecek ve hedef nesneyi kaybetmeyecektir.

Gelecek çalışmalarda simülasyon ortamında gerçekleştirilen bu işlemlerin gerçek İHA ve araç üzerinde gimbal ile desteklenerek daha yüksek hızlarda gerçekleştirilmesi planlanmaktadır.

Coco dataset ile önceden eğitilmiş Faster R-CNN modeli üzerine DOTA ve COWC datasetleri ile eğitim gerçekleştirilecek simülasyon ortamı dışında da yüksek isabet oranına sahip bir model oluşturulacaktır.Bunun yanı sıra daha hızlı bir derin öğrenme modeli olan YoloV3 ile testler gerçekleştirilecek ve uygun model kullanılacaktır.

Son adım olarak İHA altına takılacak bir robotik kol ile yakalanan cismin kaldırılması ve taşıma işlemi gerçekleştirilmesi planlanmaktadır.

4. KAYNAKÇA

Abdel-Hamid, O., Mohamed, A. R., Jiang, H., & Penn, G. (2012). Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. https://doi.org/10.1109/ICASSP.2012.6288864

Ackerman, S. (2014). *41 men targeted but 1,147 people killed: US drone strikes – the facts on the ground*. https://www.theguardian.com/us-news/2014/nov/24/-sp-us-drone-strikes-kill-1147

Ackerman, Spencer, & Shachtman, N. (2012). Almost 1 In 3 U.S. Warplanes Is a Robot. *WIRED*.

Azoulai, Y. (2011). *Unmanned combat vehicles shaping future warfare*. https://en.globes.co.il/en/article-1000691790

Bannister, A. (2016). *Biometrics and AI: how FaceSentinel evolves 13 times faster thanks to deep learning*. https://www.ifsecglobal.com/global/biometrics-ai-facesentinel-deep-learning/

Battiato, S., Cantelli, L., D'Urso, F., Farinella, G. M., Guarnera, L., Guastella, D., Melita, C. D., Muscato, G., Ortis, A., Ragusa, F., & Santoro, C. (2017). A system for autonomous landing of a UAV on a moving vehicle. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-68560-1_12

Benavidez, P., Lambert, J., Jaimes, A., & Jamshidi, M. (2014). Landing of an Ardrone 2.0 quadcopter on a mobile base using fuzzy logic. *World Automation Congress Proceedings*. https://doi.org/10.1109/WAC.2014.6936156

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*. https://doi.org/10.1561/2200000006

Borowczyk, A., Nguyen, D. T., Phu-Van Nguyen, A., Nguyen, D. Q., Saussié, D., & Ny, J. Le. (2017). Autonomous Landing of a Multirotor Micro Air Vehicle on a High Velocity Ground Vehicle. *IFAC-PapersOnLine*, *50*(1), 10488–10494.

https://doi.org/10.1016/j.ifacol.2017.08.1980

Bradski, G., & Kaehler, A. (2008). Learning OpenCV, 1st Edition. In *Learning*.

Burnham, W. H. (1888). Memory, Historically and Experimentally Considered. I. An Historical Sketch of the Older Conceptions of Memory. *The American Journal of Psychology*. https://doi.org/10.2307/1411406

Corporation, I., & Garage, W. (2013). *OpenCV*. Wikipedia. http://en.wikipedia.org/wiki/OpenCV

Dunstan, S. (2008). *Israeli Fortifications of the October War 1973*.

Falanga, D., Zanchettin, A., Simovic, A., Delmerico, J., & Scaramuzza, D. (2017). Vision-based autonomous quadrotor landing on a moving platform. *SSRR 2017 - 15th IEEE International Symposium on Safety, Security and Rescue Robotics, Conference*. https://doi.org/10.1109/SSRR.2017.8088164

Feng, Y., Zhang, C., Baek, S., Rawashdeh, S., & Mohammadi, A. (2018). *Autonomous landing of an UAV on a moving platform using model predictive control*. *2018-Janua*, 2298–2303. https://doi.org/10.1109/ASCC.2017.8287533

Fu, M., Zhang, K., Yi, Y., & Shi, C. (2016). Autonomous landing of a quadrotor on an UGV. *2016 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2016*, 988–993. https://doi.org/10.1109/ICMA.2016.7558697

Gazebo Simulation. (n.d.). http://gazebosim.org

George, M., Jose, B. R., & Mathew, J. (2018). Performance Evaluation of KCF based Trackers using VOT Dataset. *Procedia Computer Science*. https://doi.org/10.1016/j.procs.2017.12.072

Ghinmine, S. V., & Sapkal, P. A. (2017). Comparative Study of RGB, HSV & YCbCr Color Model Saliency Map. *International Journal of Innovative Research in Computer and Communication Engineering*. https://doi.org/10.15680/IJIRCCE.2017

Gibiansky, A., & Gopalakrishnan, E. (2017). Quadcopter Flight Mechanics Model and Control Algorithms. **学位**论文.

Gilberto Mendoza Chavez. (2016). *Autonomous Landing on Moving Platforms Thesis by Gilberto Mendoza Ch ´ avez In Partial Fulfillment of the Requirements For the Degree of Masters of Science*.

Ginsburg. (n.d.). *Paparazzi Agency - We've Used Drones For A Long Time*. https://www.tmz.com/2014/08/04/paparazzi-using-drones-video-celebrity-homes/

Gonzalez, R. C., Woods, R. E., & Masters, B. R. (2009). Digital Image Processing, Third Edition. *Journal of Biomedical Optics*. https://doi.org/10.1117/1.3115362

Gupta, A., Wang, H., & Ganapathiraju, M. (2015). Learning structure in gene expression data using deep architectures, with an application to gene clustering. *Proceedings - 2015 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2015*. https://doi.org/10.1109/BIBM.2015.7359871

Hoffmann, G. M., Huang, H., Waslander, S. L., & Tomlin, C. J. (2007). Quadrotor helicopter flight dynamics and control: Theory and experiment. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2007, 2*(August), 1670–1689. https://doi.org/10.2514/6.2007-6461

Hu, B., Lu, L., & Mishra, S. (2017). A Control Architecture for Time-Optimal Landing of a Quadrotor Onto a Moving Platform. *Asian Journal of Control*. https://doi.org/10.1002/asjc.1693

Kanyike, R. (2012). *History of U.S. Drones*. https://understandingempire.wordpress.com/2-0-a-brief-history-of-u-s-drones/

Kaplan, P. (2013). *Naval Aviation in the Second World War (Images of War)*.

Lange, S., Sünderhauf, N., & Protzel, P. (2009). A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments. *2009 International Conference on Advanced Robotics, ICAR 2009*.

Layman, R. D. (1996). *Naval Aviation in the First World War: Its Impact and Influence*.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. In *Nature*. https://doi.org/10.1038/nature14539

Leishman, J. G. (2000). *Principles of Helicopter Aerodynamics*.

Levinson, C. (2010). *Israeli Robots Remake Battlefield". The Wall Street Journal*. A10.

Line, V. (2018). *Autonomous Tracking and Landing of a Multirotor UAV on an Outdoor Ground Moving Platform*. *June*.

Ling, K., Chow, D., Das, A., & Waslander, S. L. (2014). Autonomous maritime landings for low-cost VTOL aerial vehicles. *Proceedings - Conference on Computer and Robot Vision, CRV 2014*. https://doi.org/10.1109/CRV.2014.13

Lins, S., Garratt, M., Lambert, A., & Li, P. (2015). 6DoF Motion estimation for UAV landing on a moving shipdeck using real-Time on-board vision. *Australasian Conference on Robotics and Automation, ACRA*.

Liu, W., Ma, H., Qi, H., Zhao, D., & Chen, Z. (2017). Deep learning hashing for mobile visual search. *Eurasip Journal on Image and Video Processing*, *2017*(1). https://doi.org/10.1186/s13640-017-0167-4

Magazine, H. T. C. (2008). *MAV Flight Control: Realities and Challenges*. http://www.hiphtechcareers.com/doc198e/flightcontrol198e.html

McKenna, A. (2016). The Future of Drone Use Opportunities and Threats from Ethical and Legal Perspectives. In *The Future of Drone Use*. https://doi.org/10.1007/978-94-6265-132-6

Mendes, S. (2012). *Vision-based automatic landing of a quadrotor UAV on a floating platform*. 224.

Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games*.

Moreau, R., & Yousafzai, S. (2017). *Al Qaeda Commander Ilyas Kashmiri Killed in U.S. Predator Strike*. The Daily Beast.

Mosavi, A., & Varkonyi-Koczy, A. R. (2017). Integration of machine learning and optimization for robot learning. *Advances in Intelligent Systems and Computing*, *519*(September), 349–355. https://doi.org/10.1007/978-3-319-46490-9_47

Moskowitz, E. E., Siegel-Richman, Y. M., Hertner, G., & Schroeppel, T. (2018). Aerial drone misadventure: A novel case of trauma resulting in ocular globe

rupture. *American Journal of Ophthalmology Case Reports*. https://doi.org/10.1016/j.ajoc.2018.01.039

Osako, K., Singh, R., & Raj, B. (2015). Complex recurrent neural networks for denoising speech signals. *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, WASPAA 2015*. https://doi.org/10.1109/WASPAA.2015.7336896

Özkan, B. (2017). Guidance and Control of a Quadrotor towards a Moving Land Platform. *Gazi Journal of Engineering Sciences*.

P., R., & Hallion. (2003). *Taking Flight: Inventing the Aerial Age, from Antiquity through the First World War*.

Patel, B., Ray, N., & Patel, P. (2018). Motion based Object Tracking. *International Journal of Electronics, Electrical and Computational System*, *7*(4), 581–588.

Pike, J., & Aftergood, S. (2012). *RQ-1 Predator MAE UAV*. FAS. https://fas.org/irp/program/collect/predator.htm

Polvara, R. (2018). Vision-Based Autonomous Landing of a Quadrotor on the Perturbed Deck of an Unmanned Surface Vehicle. *Drones*, *2*(2), 15. https://doi.org/10.3390/drones2020015

Pounds, P., Mahony, R., & Corke, P. (2006). In the Proceedings of the Australasian Conference on Robotics and Automation. *Modelling and Control of a Quad-Rotor Robot*. https://doi.org/10.1.1.127.1200

Radsan, A. J., & Murphy, R. (2011). Measure twice, shoot once: Higher care for cia-targeted killing. *University of Illinois Law Review*, *2011*(4), 1201–1242.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2016.2577031

Renner, S. L. (2016). *Broken Wings: The Hungarian Air Force, 1918-45*.

Rosenfeld, A. (1969). Picture Processing by Computer. *ACM Computing Surveys (CSUR)*. https://doi.org/10.1145/356551.356554

"Russia Buys A Bunch of Israeli UAVs". (2015). https://www.strategypage.com/htmw/htairfo/articles/20090409.aspx

SCHEVE, T. (2015). *"A Brief History of UAVs."* https://science.howstuffworks.com/reaper.htm

Selvaraj, A. (n.d.). In a first, Tamil Nadu police use UAV in murder probe. *The Times Of India*. https://timesofindia.indiatimes.com/city/chennai/In-a-first-Tamil-Nadu-police-use-UAV-in-murder-probe/articleshow/30967123.cms

Singer, P. W. (2009). *A Revolution Once More: Unmanned Systems and the Middle East*. Brookings. https://www.brookings.edu/articles/a-revolution-once-more-unmanned-systems-and-the-middle-east/

Sonka, M., Hlavac, V., & Boyle, R. (1993). Image Processing, Analysis and Machine Vision. In *Image Processing, Analysis and Machine Vision*. https://doi.org/10.1007/978-1-4899-3216-7

Vijay Kumar Saxena. (2013). *The Amazing Growth and Journey of UAVs & Ballistic Missile Defence Capabilities*.

Vlantis, P., Marantos, P., Bechlioulis, C. P., & Kyriakopoulos, K. J. (2015). Quadrotor landing on an inclined platform of a moving ground vehicle. *Proceedings - IEEE International Conference on Robotics and Automation*, *2015-June*(June), 2202–2207. https://doi.org/10.1109/ICRA.2015.7139490

Wagner, W. (1982). *Lightning Bugs and other Reconnaissance Drones*. Armed Forces Journal International.

Wang, H., & Raj, B. (2017). *On the Origin of Deep Learning*. 1–72. http://arxiv.org/abs/1702.07800

Wang, H., & Yang, J. (2017). Multiple confounders correction with regularized linear mixed effect models, with application in biological processes. *Proceedings - 2016 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016*. https://doi.org/10.1109/BIBM.2016.7822753

William, J., & Taylor, R. (1977). *Jane's pocket book of remotely piloted vehicles: Robot aircraft today*.

Wyeth, G., Buskey, G., & Roberts, J. (2000). Flight control using an artificial neural network. *Proceedings of the Australian …*.

5. EKLER

EK 1: **Araç Kontrol Kod Çıktısı (Sekiz şeklinde rota)**

```
#!/usr/bin/env python

import rospy
import numpy as np
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64
from getkey import getkey, keys


def states_callback(states):

  print(str(states.pose[2].position.x)+"-
"+str(states.pose[2].position.y))

if __name__ == '__main__':

  rospy.init_node('master_car', anonymous=True)
  rate = rospy.Rate(20)

  Speed = Twist()
  left_Wheel = Float64()
  right_Wheel = Float64()

  speedFeeder = rospy.Publisher(
      "/r2d2_speed_controller/cmd_vel", Twist, queue_size=10)
  left_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_left_Wheel_controller/command", Float64, queue_size=10)
  right_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_right_Wheel_controller/command", Float64,
queue_size=10)

  while 1:

    count = 0

    while 1:

      Speed.linear.x = 4
      speedFeeder.publish(Speed)

      left_Wheel.data = -0.1
      left_Wheel_angle_pub.publish(left_Wheel)

      right_Wheel.data = -0.1
      right_Wheel_angle_pub.publish(right_Wheel)

      rate.sleep()

      count = count+1

      print count

      if count >= 3040:
```

```
    break

count = 0

while 1:
  Speed.linear.x = 0
  speedFeeder.publish(Speed)

  left_Wheel.data = 0
  left_Wheel_angle_pub.publish(left_Wheel)

  right_Wheel.data = 0
  right_Wheel_ angle _pub.publish(right_Wheel)

  rate.sleep()

  count = count+1

  print count

  if count >= 200:
    break

count = 0

while 1:
  Speed.linear.x = 4
  speedFeeder.publish(Speed)

  left_Wheel.data = 0.1
  left_Wheel_angle _pub.publish(left_Wheel)

  right_Wheel.data = 0.1
  right_Wheel_angle _pub.publish(right_Wheel)

  rate.sleep()

  count = count+1

  print count

  if count >= 3040:
    break

count = 0

while 1:
  Speed.linear.x = 0
  speedFeeder.publish(Speed)

  left_Wheel.data = 0
  left_Wheel_aci_pub.publish(left_Wheel)

  right_Wheel.data = 0
  right_Wheel_aci_pub.publish(right_Wheel)

  rate.sleep()
```

```
      count = count+1

      print count

      if count >= 200:
        break
```

EK 2: **Araç Kontrol Kod Çıktısı (Spiral şeklinde rota)**

```
#!/usr/bin/env python

import rospy
import numpy as np
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64


def states_callback(states):

  print(str(states.pose[2].position.x)+"-
"+str(states.pose[2].position.y))


if __name__ == '__main__':

  rospy.init_node('master_car', anonymous=True)
  rate = rospy.Rate(20)


  Speed = Twist()
  left_Wheel = Float64()
  right_Wheel = Float64()


  speedFeeder = rospy.Publisher(
      "/r2d2_Speed_controller/cmd_vel", Twist, queue_size=10)
  left_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_left_Wheel_controller/command", Float64, queue_size=10)
  right_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_right_Wheel_controller/command", Float64,
queue_size=10)

  while 1:
    count = 0

    while 1:

      Speed.linear.x = 4
      speedFeeder.publish(Speed)

      left_Wheel.data = 0.4
      left_Wheel_angle_pub.publish(left_Wheel)

      right_Wheel.data = 0.4
      right_Wheel_angle_pub.publish(right_Wheel)
```

```
      rate.sleep()

      count = count+1

      print count

      if count >= 450:
        break

    count = 0



    count = 0

    while 1:
      Speed.linear.x = 4
      Speed_deger_yolla.publish(Speed)

      left_Wheel.data = -0.4
      left_Wheel_angle_pub.publish(left_Wheel)

      right_Wheel.data = -0.4
      right_Wheel_angle_pub.publish(right_Wheel)

      rate.sleep()

      count = count+1

      print count

      if count >= 450:
        break

    count = 0
```

EK 3: **Araç Kontrol Kod Çıktısı (Düz rota)**

```
#!/usr/bin/env python

import rospy
import numpy as np
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64


def states_callback(states):

  print(str(states.pose[2].position.x)+"-
"+str(states.pose[2].position.y))


if __name__ == '__main__':
```

```
  rospy.init_node('master_car', anonymous=True)
  rate = rospy.Rate(20)


  Speed = Twist()
  left_Wheel = Float64()
  right_Wheel = Float64()


  speedFeeder = rospy.Publisher(
      "/r2d2_speed_controller/cmd_vel", Twist, queue_size=10)
  left_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_left_Wheel_controller/command", Float64, queue_size=10)
  right_Wheel_angle_pub = rospy.Publisher(
      "/r2d2_right_Wheel_controller/command", Float64,
queue_size=10)

  while 1:
    count = 0

    while 1:

      Speed.linear.x = 4
      speedFeeder.publish(Speed)

      left_Wheel.data = 0
      left_Wheel_angle_pub.publish(left_Wheel)

      right_Wheel.data = 0
      right_Wheel_angle_pub.publish(right_Wheel)

      rate.sleep()

      count = count+1

      print count
```

EK 4: **Görüntü işleme ile tespit Kod Çıktısı**

```
#!/usr/bin/env python

from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time

import rospy
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import sys

import signal
```

```
def handler(signum, frame):
  print("destroy ctrl z")
  cv2.destroyAllWindows()
  exit(0)


signal.signal(signal.SIGTSTP, handler)


blueLower = (110, 50, 50)
blueUpper = (138, 255, 255)


bbox = (287, 23, 86, 320)

bridge = CvBridge()

vs = None


def image_callback(ros_image):

  global bridge
  global vs
# convert ros_image into an opencv-compatible image
  try:
    cv_image = bridge.imgmsg_to_cv2(ros_image, "bgr8")
    # print "sekilsin ha ", cv_image.shape

  except CvBridgeError as e:
    print(e)
  vs = cv_image


if __name__ == '__main__':
  rospy.init_node('image_converter', anonymous=True)
  image_sub = rospy.Subscriber(
      "/iris/camera_red_iris/image_raw", Image, image_callback)
  pub = rospy.Publisher('konum', String, queue_size=10)

  rate = rospy.Rate(20)

  tracker = cv2.TrackerKCF_create()
  time.sleep(1)

  while not rospy.is_shutdown():

    frame = vs

    if frame is None:
      continue

    timer = cv2.getTickCount()
    ok, bbox = tracker.update(frame)
```

```
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)

    if not ok:

      print("tracker false")

      pub.publish("0,0,0")
      rate.sleep()

      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

      mask = cv2.inRange(hsv, blueLower, blueUpper)
      #res = cv2.bitwise_and(frame, frame, mask=mask)

      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                              cv2.CHAIN_APPROX_SIMPLE)

      cnts = cnts[0] if imutils.is_cv2() else cnts[1]
      center = None

      if len(cnts) > 0:

        filteredContours = []
        for i in cnts:
          filteredContours.append(i)

        c = max(cnts, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(c)

        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        ok = tracker.init(frame, (x-100, y-100, 200, 200))

        print("tracker init"+str(ok))

        #cv2.imshow('original', frame)
        #cv2.imshow('mask', mask)

    else:

      p1 = (int(bbox[0]), int(bbox[1]))
      p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))

      center = int(bbox[0] + bbox[2]/2), int(bbox[1] + bbox[3]/2)
      cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)

      pub.publish(str(int(bbox[0] + bbox[2]/2)) +
                  ","+str(int(bbox[1] + bbox[3]/2))+"," +
str(bbox[2]))

      print(str(int(bbox[0] + bbox[2]/2)) +
            ","+str(int(bbox[1] + bbox[3]/2))+"," + str(bbox[2]))

      rate.sleep()


    for i in range(1, len(pts)):
```

```
        if pts[i - 1] is None or pts[i] is None:
          continue


        thickness = int(np.sqrt(64 / float(i + 1)) * 2.5)
        cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)


      cv2.putText(frame, " Tracker", (100, 20),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)


      cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)

      cv2.namedWindow('Frame', cv2.WINDOW_NORMAL)
      cv2.imshow("Frame", frame)

      cv2.namedWindow('Hsv', cv2.WINDOW_NORMAL)
      cv2.imshow("Hsv", hsv)

      cv2.namedWindow('Mask', cv2.WINDOW_NORMAL)
      cv2.imshow("Mask", mask)
      key = cv2.waitKey(1) & 0xFF

      # if the 'q' key is pressed, stop the loop
      if key == ord("q"):
        break

  # close all windows
  log_kamera.close()
  cv2.destroyAllWindows()
```

EK 5: **Derin öğrenme ile tespit Kod Çıktısı**

```
import tensorflow as tf
import numpy as np
import time
import cv2

from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time

import rospy
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import sys

WIDTH = 800.
```

```
HEIGHT = 800.


bridge = CvBridge()

vs = None

confidence = 0


def image_callback(ros_image):

  global bridge
  global vs

  try:
    cv_image = bridge.imgmsg_to_cv2(ros_image, "bgr8")

  except CvBridgeError as e:
    print(e)
  vs = cv_image


def main():

  rospy.init_node('image_converter', anonymous=True)
  image_sub = rospy.Subscriber(
      "/iris/camera_red_iris/image_raw", Image, image_callback)
  pub = rospy.Publisher('konum', String, queue_size=10)

  tracker = cv2.TrackerKCF_create()

  NUM_CLASSES = 1
  # PATH_TO_CKPT = "/home/faruk/Desktop/uav/fin/faster-rcnn/ckpt-
22143/frozen_inference_graph.pb"
  # PATH_TO_LABELS = "/home/faruk/Desktop/uav/fin/labelmap.pbtxt"
  PATH_TO_CKPT = "/home/metinkoc/Desktop/frozen_inference_graph.pb"
  PATH_TO_LABELS = "/home/metinkoc/Desktop/labelmap.pbtxt"

  print("** Loading model into memory...")
  # ## Load a frozen Tensorflow model into memory.
  detection_graph = tf.Graph()
  with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
      serialized_graph = fid.read()
      od_graph_def.ParseFromString(serialized_graph)
      tf.import_graph_def(od_graph_def, name='')

  # TF returns [ymin xmin ymax xmax] in normalized form (between 0
and 1)
  # tracker expects (xmin, ymin, width, height)
  # cv2.rectangle expects (topleft), (bottomright)
  # ROI returns (xmin, ymin, width, height)

  with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
```

```
      while True:

        # Handle quitting
        if cv2.waitKey(25) & 0xFF == ord('q'):
          cv2.destroyAllWindows()
          break

        frame = vs

        timer = cv2.getTickCount()
        ok, bbox = tracker.update(frame)

        fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)

        if not ok:

          ########### Start of Detection ###########

          # Expand dimensions since the model expects images to have
shape: [1, None, None, 3]
          image_np_expanded = np.expand_dims(frame, axis=0)
          image_tensor =
detection_graph.get_tensor_by_name('image_tensor:0')
          # Each box represents a part of the image where a
particular object was detected.
          boxes =
detection_graph.get_tensor_by_name('detection_boxes:0')
          # Each score represent how level of confidence for each of
the objects.
          # Score is shown on the result image, together with the
class label.
          scores =
detection_graph.get_tensor_by_name('detection_scores:0')
          classes =
detection_graph.get_tensor_by_name('detection_classes:0')
          num_detections = detection_graph.get_tensor_by_name(
              'num_detections:0')

          # Actual detection.
          (boxes, scores, classes, num_detections) = sess.run(
              [boxes, scores, classes, num_detections],
              feed_dict={image_tensor: image_np_expanded})

          ########### End of Detection ###########

          # if we don't have a confident guess, ignore it and move
on.
          confidence = np.squeeze(scores)[0]
          if confidence < .92:
            print("** No confident detection, trying again...")
            cv2.imshow('uav', frame)
            continue

          print("** Object detected")
          bbox_normalized = np.squeeze(boxes)[0].tolist()
          # bbox_normalized is now [ymin, xmin, ymax, xmax] all
between 0-1
```

```
            # enlarge it for tracking safety (reduce the chance to get
evaded)
            bbox_normalized = enlarge_bbox(bbox_normalized)

            # denormalize our enlarged bounding box (get its real
pixes values)
            bbox = denormalize_bbox(bbox_normalized, WIDTH, HEIGHT)
            # bbox is now [ymin, xmin, ymax, xmax], with actual pixel
values.
            # reformat into (xmin, ymin, width, height) for the
tracker:
            bbox_for_tracker = (
                bbox[1], bbox[0], bbox[3]-bbox[1], bbox[2]-bbox[0])

            topleft = (int(bbox[0]-100), int(bbox[1]+100))
            bottomright = (int(bbox[2]+100), int(bbox[3]-100))

            print bbox[0]
            print bbox[1]
            print bbox[2]
            print bbox[3]
            print confidence

            cv2.rectangle(frame, topleft, bottomright, (0, 200, 0), 2,
1)

            tracker = cv2.TrackerKCF_create()
            ok = tracker.init(frame, bbox_for_tracker)

            print("tracker init"+str(ok))
          else:
            p1 = (int(bbox[0]), int(bbox[1]))
            p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))

            center = int(bbox[0] + bbox[2]/2), int(bbox[1] +
bbox[3]/2)
            cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)

            if int(bbox[0] + bbox[2]/2) > 150 and int(bbox[0] +
bbox[2]/2) < 650:
              if int(bbox[1] + bbox[3]/2) > 150 and int(bbox[1] +
bbox[3]/2) < 650:
                pub.publish(str(int(bbox[0] + bbox[2]/2)) + "," +
                            str(int(bbox[1] + bbox[3]/2))+"," +
str(bbox[2]))

          cv2.putText(frame, str(confidence*100), (int(bbox[0]), int(
              bbox[1])), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255),
2, cv2.LINE_AA)
          cv2.imshow('uav', frame)


def denormalize_bbox(bbox_normal, maxwidth, maxheight):
  """
  Takes normalized coordinates in [ymin, xmin, ymax, xmax]
  and returns actual pixel coordinates.
```

```
    """
    ymin = bbox_normal[0] * maxheight
    xmin = bbox_normal[1] * maxwidth
    ymax = bbox_normal[2] * maxheight
    xmax = bbox_normal[3] * maxwidth

    return [int(i) for i in [ymin, xmin, ymax, xmax]]


def enlarge_bbox(bbox_normal, padding=.055):
    """
    Takes normalized coordinates in [ymin, xmin, ymax, xmax].
    Returns a slightly enlarged bounding box in the same form.
    Example: [.3, .3, .8, .8] may become [.25, .25, .91, .91]~.
    """
    # shrink ymin and xmin by a percentage, but check for the
boundary.
    # because they cannot be less than 0 (a bounding box larger than
the screen)
    enlarged_ymin = max(0, bbox_normal[0] - padding)
    enlarged_xmin = max(0, bbox_normal[1] - padding)

    # grow ymax and xmax by a percentage, but check for the boundary.
    enlarged_ymax = min(1, bbox_normal[2] + padding)
    enlarged_xmax = min(1, bbox_normal[3] + padding)

    return [enlarged_ymin, enlarged_xmin, enlarged_ymax,
enlarged_xmax]


if __name__ == "__main__":
    main()
```

EK 6: İHA Kontrol Kod Çıktısı

```
#!/usr/bin/env python

import rospy
import mavros
import math
import time
from mavros_msgs.msg import PositionTarget
from mavros_msgs.srv import *
from mavros_msgs.msg import Altitude, State
from std_msgs.msg import String
from geometry_msgs.msg import Pose, Point
from nav_msgs.msg import Odometry
from sensor_msgs.msg import NavSatFix
from tf.transformations import euler_from_quaternion,
quaternion_from_euler
from gazebo_msgs.msg import ModelStates


msg = PositionTarget()

totalErrorX = 0
```

```
totalErrorY = 0
totalErrorZ = 0
lastErrorR = 0

fx = open("/home/metinkoc/x.txt", "w+")
fx.write("Drone position.x")
fx.write(",")
fx.write("Drone position.y")
fx.write(",")
fx.write("Drone position.z")
fx.write(",")
fx.write("Drone linear.x")
fx.write(",")
fx.write("Drone linear.y")
fx.write(",")
fx.write("Drone linear.z")
fx.write(",")
fx.write("Car position.x")
fx.write(",")
fx.write("Car position.y")
fx.write(",")
fx.write("Car linear.x")
fx.write(",")
fx.write("Car linear.y")
fx.write("\r\n")

fx.flush()
r = 3
theta = 0
count = 0.0
wn = 1


rtlLatitude = 0
rtlLongitude = 0

latitude = 0
longitude = 0
altitude = Altitude()
isModeChanged = False


def call_back(data):

  global msg

  global totalErrorX
  global totalErrorY
  global totalErrorZ

  print(data.data)

  XY = data.data
  XY = XY.split(",")

  x = float(XY[0])
  y = float(XY[1])
```

```
z = float(XY[1])

if(x == 0 and y == 0 and z == 0):
  if altitude < 3:
    msg.velocity.x = 0
    msg.velocity.y = 0
    msg.velocity.z = 1
  else:
    msg.velocity.x = 0
    msg.velocity.y = 0
    msg.velocity.z = 0
  return

# ----------------------

kp = 1.0/200.0
ki = 1.0/50000.0

kpZ = 1.0/1600.0
kiZ = 1.0/50000.0

msg.velocity.z = 0

errorX = 400-y
totalErrorX = totalErrorX + errorX

if(totalErrorX > 10000):
  totalErrorX = 10000

if(totalErrorX < -10000):
  totalErrorX = -10000

# turevY = errorY - lastErrorY
# yaklasmaY = errorY*kp + turevY*kd
uX = errorX*kp + ki*totalErrorX

msg.velocity.y = uX

print("----------------")
print("x:", x)
print("errorX:", errorX)
print("uX:", uX)
print("----------------")

errorY = x-400
totalErrorY = totalErrorY + errorY

if(totalErrorY > 10000):
  totalErrorY = 10000

if(totalErrorY < -10000):
  totalErrorY = -10000


uY = errorY*kp + ki*totalErrorY

msg.velocity.x = yaklasmaY
```

```
  print("----------------")
  print("y:", y)
  print("errorY:", errorY)
  print("uY:", uY)
  print("----------------")
  print("----------------")

  fy.write(str(errorY))
  fy.write(",")

  msg.velocity.z = -0.05


def states_callback(states):

  global fx
  # print(str(states.pose[2].position.x)+"-
"+str(states.pose[2].position.y))

  fx.write(str(states.pose[1].position.x))
  fx.write(",")
  fx.write(str(states.pose[1].position.y))
  fx.write(",")
  fx.write(str(states.pose[1].position.z))
  fx.write(",")
  fx.write(str(states.twist[1].linear.x))
  fx.write(",")
  fx.write(str(states.twist[1].linear.y))
  fx.write(",")
  fx.write(str(states.twist[1].linear.z))
  fx.write(",")
  fx.write(str(states.pose[2].position.x))
  fx.write(",")
  fx.write(str(states.pose[2].position.y))
  fx.write(",")
  fx.write(str(states.twist[2].linear.x))
  fx.write(",")
  fx.write(str(states.twist[2].linear.y))
  fx.write("\r\n")

  fx.flush()


def globalPositionCallback(globalPositionCallback):
  global latitude
  global longitude

  global rtlLatitude
  global rtlLongitude

  latitude = globalPositionCallback.latitude
  longitude = globalPositionCallback.longitude

  if(rtlLatitude == 0):
    rtlLatitude = latitude
    rtlLongitude = longitude
```

```
def altitude_callback(data):
  global altitude

  altitude = data


def setOffboardMode():
  rospy.wait_for_service('/mavros/set_mode')
  try:
    flightModeService = rospy.ServiceProxy(
        '/mavros/set_mode', mavros_msgs.srv.SetMode)
    # http://wiki.ros.org/mavros/CustomModes for custom modes
    flightModeService(
        custom_mode='OFFBOARD')  # return true or false
  except rospy.ServiceException, e:
    print "service set_mode call failed: %s. GUIDED Mode could not
be set. Check that GPS is enabled" % e


def setDisarm():
  rospy.wait_for_service('/mavros/cmd/arming')
  try:
    armService = rospy.ServiceProxy(
        '/mavros/cmd/arming', mavros_msgs.srv.CommandBool)
    armService(False)
  except rospy.ServiceException, e:
    print "Service arm call failed: %s" % e


def setArm():
  rospy.wait_for_service('/mavros/cmd/arming')
  try:
    armService = rospy.ServiceProxy(
        '/mavros/cmd/arming', mavros_msgs.srv.CommandBool)
    armService(True)
  except rospy.ServiceException, e:
    print "Service arm call failed: %s" % e


def setTakeoffMode():
  rospy.wait_for_service('/mavros/cmd/takeoff')
  try:
    takeoffService = rospy.ServiceProxy(
        '/mavros/cmd/takeoff', mavros_msgs.srv.CommandTOL)
    takeoffService(altitude=7, latitude=latitude,
                   longitude=longitude, min_pitch=0, yaw=0)
  except rospy.ServiceException, e:
    print "Service takeoff call failed: %s" % e


def setLandMode():
  rospy.wait_for_service('/mavros/cmd/land')
  try:
    landService = rospy.ServiceProxy(
        '/mavros/cmd/land', mavros_msgs.srv.CommandTOL)
```

```
    # http://wiki.ros.org/mavros/CustomModes for custom modes
    landService(altitude=0, latitude=rtlLatitude,
                longitude=rtlLongitude, min_pitch=0, yaw=0)
  except rospy.ServiceException, e:
    print "service land call failed: %s. The vehicle cannot land " %
e


if __name__ == '__main__':

  rospy.init_node('master_drone', anonymous=True)
  rate = rospy.Rate(20)

  rospy.Subscriber("/mavros/global_position/raw/fix",
                   NavSatFix, globalPositionCallback)

  rospy.Subscriber('/mavros/altitude', Altitude,
                   altitude_callback)

  rospy.Subscriber('/gazebo/model_states', ModelStates,
                   states_callback)

  pub = rospy.Publisher('/mavros/setpoint_raw/local',
                        PositionTarget, queue_size=10)

  fy = open("y.txt", "w+")

  while(latitude == 0):
    print "Waiting for gps"

  setArm()
  time.sleep(3)
  # setTakeoffMode()

  msg.header.stamp = rospy.Time.now()
  msg.header.frame_id = "world"
  msg.coordinate_frame = PositionTarget.FRAME_BODY_NED
  msg.type_mask = PositionTarget.IGNORE_PX |
PositionTarget.IGNORE_PY | PositionTarget.IGNORE_PZ |
PositionTarget.IGNORE_AFX | PositionTarget.IGNORE_AFY |
PositionTarget.IGNORE_AFZ | PositionTarget.IGNORE_YAW |
PositionTarget.IGNORE_YAW_RATE

  msg.velocity.x = 0.0
  msg.velocity.y = 0.0
  msg.velocity.z = 2.0

  count = 0

  while count < 10:
    pub.publish(msg)
    count = count+1

  setOffboardMode()

  counter = 0
```

```
while 1:

  pub.publish(msg)
  rate.sleep()

  if altitude.local > 20:
    rospy.Subscriber('/konum', String,
                     call_back)

    break

  print("Altitude:" + str(altitude.local))

msg.velocity.z = 0

while(1):

  pub.publish(msg)
  rate.sleep()

setLandMode()
```

ÖZGEÇMİŞ

1986 yılı Giresun doğumlu ve 2011 yılında Bilgisayar Mühendisliği eğitimini tamamlayan Metin KOÇ, mezuniyetinden bu yana kamu ve özel sektörde Elektronik Ar-Ge alanında çalışmaya devam etmektedir. Gömülü sistem teknolojileri konularında uzman ve Akıllı şehir teknolojileri alanında birçok proje geliştirmiş olup aynı zamanda İnsansız hava araçları alanında çeşitli çalışmaları bulunmaktadır.

İHA-0, İHA-1, Amatör telsiz sertifikalarına sahip olan Metin KOÇ, havacılık alanında meraklı ve aktif geliştiricidir.

Evli ve 1 çocuk babasıdır.