İSTANBUL OKAN ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

VERİMLİ SEÇMELİ GÜVENLİ BİR VERİ SETİ KÜMELEMESİ ÖNERİSİ VE OTONOM SÜRÜŞE YÖNELİK BİR UYGULAMASI

YÜKSEK LİSANS TEZİ

FAHRETTİN ORKUN KIZILIRMAK

tarafından

YÜKSEK LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Mayıs 2019

Program: Bilgisayar Mühendisliği

VERİMLİ SEÇMELİ GÜVENLİ BİR VERİ SETİ KÜMELEMESİ ÖNERİSİ VE OTONOM SÜRÜŞE YÖNELİK BİR UYGULAMASI

YÜKSEK LİSANS TEZİ

FAHRETTİN ORKUN KIZILIRMAK

tarafından

İSTANBUL OKAN ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Onaylayan:

Danışman
Prof. Dr. Bekir Tevfik Akgün

Üye
Dr. Öğr. Üyesi Pınar Yıldırım

Üye
Doç. Dr. Erchan APTOULA
(Gebze Teknik Üniversitesi)

Mayıs 2019

Program: Bilgisayar Mühendisliği

ABSTRACT

A SAMPLE EFFICIENT SELECTIVE SAFE

DATA AGGREGERATION PROPOSAL

AND AN APPLICATION FOR AUTONOMOUS DRIVE

The objective of this work is to develop a sample efficient end-to-end deep learning method for self-driving cars, where it is attempted to minimize number of times the expert driver is called to improve the driving policy, through selective analysis of the obtained samples. End-to-end imitation learning is a popular method for computing self-driving car policies. The standard approach relies on collecting pairs of inputs (camera images) and outputs (steering angle etc.) from an expert policy and fitting a deep neural network to this data to learn the driving policy. Although this approach had some successful demonstrations in the past, learning a good policy might require a lot of samples from the expert driver, which might be resource-consuming. In this work, a novel framework developed based on the Safe Dataset Aggregation (safe DAgger) approach, where the current learned policy is automatically segmented into different trajectory classes, and the algorithm identifies trajectory segments/classes with weak performance at each step. Once the weak trajectory segments are identified, sampling algorithm focuses on calling the expert policy only on these segments, which significantly lowers both the number of times the expert is called and the convergence rate. The presented simulation results show that the proposed approach can yield significantly better performance compared to the standard Safe DAgger algorithm, while using the same number of samples from the expert.

Keywords: End-to-End Learning, Self-Driving Cars, Data Aggregation

KISA ÖZET

VERİMLİ SEÇMELİ GÜVENLİ BİR
VERİ SETİ KÜMELEMESİ ÖNERİSİ VE
OTONOM SÜRÜŞE YÖNELİK BİR UYGULAMASI

Bu çalışmanın amacı sürücüsüz araçlara yönelik örnek toplama açısından verimli bir uçtan uca derin öğrenme algoritması geliştirmektir. Bunu da elde edilen örnekleri seçici olarak sınıflandırma ile, sürüş politikasını geliştirmek adına, uzman sürücüyü en az sayıda çağırarak yapmaya çalışmaktır. Uçtan uca taklit öğrenmesi sürücüsüz sürüş politikalarında kullanılan popüler bir yöntemdir. Standart yaklaşım giriş (kamera görüntüleri) ve çıkış (direksiyon açısı vb.) ikililerini uzman sürücüden toplayıp bir derin sinir ağı içerisine yerleştirmek üzerinedir. Bu yaklaşım geçmişte bazı başarılı örnekler sergilemiş olsa da bir sürüş politikası öğrenmek uzman sürücüden alınan çok fazla örneğe ihtiyaç duymaktadır ki bu da kaynak açısından çok maliyetlidir. Bu çalışmada güvenli veri seti kümelemesi (SafeDAgger) yaklaşımı üzerine temellendirilmiş, öğrenilmiş politikanın farklı güzergâh sınıflarına ayrıldığı ve her bir sınıfın her bir tekrarda zayıflıklarına göre değerlendirildiği, özgün bir algoritma çerçevesi geliştirilmiştir. Her bir zayıf güzergâh parçası belirlendikten sonra, örnek toplayan algoritma yalnızca bu zayıf bölgelerde uzman politikayı çağırmak üzere tanımlanmıştır, bu da uzmana yapılan çağrıların sayısını ve politikanın yakınsama oranını önemli bir ölçüde azaltmaktadır. Yapılan simülasyon sonuçları göstermektedir ki yaklaşım uzmandan aynı sayıda örnek toplanırken standart SafeDAgger algoritmasına göre önemli ölçüde daha başarılı sonuçlar sunmuştur.

Anahtar Kelimeler: Uçtan Uca Öğrenme, Sürücüsüz Arabalar, Veri Seti Kümelemesi

Aileme ve Hande Sargın'a

TEŞEKKÜR

Bu çalışmanın gerçekleşmesi için önayak olan, gerekli altyapı ve kaynak desteğini sağlayan çalışmakta olduğum AVL Araştırma ve Mühendislik Türkiye şirketine ve tüm yardımları için çalışma arkadaşım Dr. Ahmetcan Erdoğan'a teşekkür ederim.

Bu çalışma Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) tarafından TEYDEB 1515 / 5169901 kapsamında desteklenmiştir.

İÇİNDEKİLER

TABLO LİSTESİ

ALGORİTMA LİSTESİ

DENKLEM LİSTESİ

ŞEKİL LİSTESİ

SEMBOLLER

t: Zaman

s: $Durum$

l: Ceza puanı

π: Sürüş politikası

Ω: Güzergâh

a: Aksiyon

x: Gözlem

T: Bir güzergahtaki gözlem/aksiyon ikililerinin sayısı

c: Sınıflandırıcı

ϵ: Sınıflandırıcının hata oranı

D: Eğitim veri seti

KISALTMALAR

NN Sinir ağları (Neural Networks)

CNN Evrişimsel sinir ağları (Convolutional Neural Networks)

SUV Spor arazi aracı (Sports Utility Vehicle)

LSTM Kısa Uzun Dönem Hafıza (Long Short Term Memory)

FC Tam Bağlı (Fully Connected)

FPS Saniye Başına Kare Sayısı (Frames Per Second)

FCN Tamamen Evrişimsel Sinir Ağı (Fully Convolutional Neural Network)

DAgger Veri Seti Kümelemesi (Data Aggregation)

SafeDAgger Güvenli Veri Seti Kümelemesi (Safe Data Aggregation)

SAE Otomotiv Mühendisleri Topluluğu (Society of Automotive Engineers)

I. GİRİŞ

Son yıllarda, özellikle derin öğrenme yönteminde elde edilen gelişmeler sayesinde sürücüsüz araba konusunda da büyük ilerlemeler sağlanmıştır. Özellikle sürücüsüz sürüş politikaları oluşturma amaçlı görüş tabanlı yöntemleri kullanma fikri, birçok araştırmacının ilgisini çekmiş ve çeşitli öğrenme ve kontrol mimarilerinin oluşmasında rol almıştır. Görüş tabanlı bu yöntemler kabaca klasik yöntemler ve uçtan uca yöntemler olarak sınıflandırılabilir. Bu yöntemlerin kısaca tanıtılmasından sonra önerilen algoritmanın uçtan uca derin öğrenme yöntemine olan katkısı bu bölümde tanıtılacak olup bunlardan önemli olanlar ikinci bölümde ayrıntılı olarak açıklanacaktır.

Klasik yöntemler sürücüsüz araç problemini üç aşamada değerlendirmektedir; algılama, rota planlama ve kontrol [1]. Algılama safhasında, renk zenginleştirme, kenar bulma vb. gibi görüntü işleme ve özellik belirleme teknikleri görsel veriler üzerinde kullanılarak şerit işaretleri gibi elemanların tespit edilmesi amaçlanmaktadır. Rota planlama safhasında, referans alınan ve aracın şu an izlediği güzergahlar, algılama aşamasında elde edilen özellikler ile tespit edilmeye çalışılır. Kontrol safhası ise direksiyon açısı, hızlanma, yavaşlama gibi kontrol aksiyonlarının, referans güzergâh ve şu anki durum bilgileri ile uygun bir kontrol algoritması sonucunda belirlenmesini içerir. Bu tarz klasik yöntemlerin performansı büyük oranda algılama aşamasında elde edilen bilgilere bağımlıdır. Bu da elle tanımlanan özellik ve kurallar sebebi ile standart altı bir sonuç doğurabilir [2]. Ayrıca klasik yöntemin sıra tabanlı yapısı neticesinde hatalara karşı dayanıklılık azalabilmektedir. Öyle ki algılama safhasında bir özelliğin tespiti sırasında karşılaşılan hata, başarısız bir kontrol kararını doğurabilir.

Diğer taraftan uçtan uca öğrenme yöntemleri, sürüş fonksiyonlarını uzman bir sürücü politikasından alınmış örnekler yardımı ile öğrenir. Öğrenilmiş fonksiyon, kontrol girdilerini yalnızca görsel veri üzerinden üreterek, klasik kontrol mimarisinin üç safhada yaptığı işi tek bir adımda birleştirmektedir. Görüntüleri kontrol girdileri ile uçtan uca yöntemler ile ilişkilendirme konusunda günümüzde en popüler olan yaklaşım sinir ağlarıdır (NN). Pomerleau tarafından yapılan ALVINN isimli çalışma [3] bu konudaki öncü çalışmalardan biridir. Çalışma ileri beslemeli bir sinir ağı kullanarak araç üzerinde ileriye bakan kameradan alınan görüntüleri, direksiyon girdileri ile haritalandırmak üzerinedir. Nvidia firmasının araştırmacıları ise özellik çıkarımının otomatikleştirilmesi ve direksiyon girdisinin tahmin edilmesi için evrişimsel sinir ağları (CNN) yöntemini [4] kullanmışlardır. Sahne ayırma işlerinin öğrenilmesi konusundaki performansı arttırmak için ise bir tam evrişimsel sinir ağı ve uzun kısa dönem hafızalı ağ (FCN-LSTM) mimarisi [5] öne sürülmüştür. Büyük ölçekli video veri setleri üzerinde yapılan bir çalışmada [6] ise, tahmin algoritmasında kullanılmak üzere görüntü üzerindeki bölgeleri vurgulamak için bir görsel dikkat modeli kullanılmıştır. [7] numaralı çalışmada ise bir evrişimsel sinir ağı ve uzun kısa dönem hafızalı ağ (CNN-LSTM) kullanılarak hızlanma ve direksiyon açısı girdilerinin senkronize olarak tespit edilmesi amaçlanmıştır.

Salt uçtan uca öğrenme politikaları öğretilen uzmanın performansı ile sınırlıdır. Uzmandan alınan eğitim ve doğrulama verilerindeki kayıpların az olacağı düşünülse bile, öğrenilmiş sürüş politikasının sürüşü sırasında biriken hatalı davranışlar uzun

vadede düşük performansa neden olabilir. Bu düşük performansın en büyük sebebinin ise öğrenilmiş politikanın, politikayı öğrendiği uzmanın karşılaşmadığı çeşitli durumları gözlemlemesi ve bilmediği bu durumlara karşı kontrol girdileri üretmeye çalışmasından ileri gelmektedir. Veri seti kümelemesi (DAgger) algoritması [8] bu soruna uzmandan ve öğrenilmiş politikadan tekrar tekrar veri toplayarak eğitimi pekiştirmek şeklinde bir çözüm bulmaya çalışmaktadır. Buradaki fikir uzmanı öğrenilmiş politikanın karşılaştığı örneklerle de gözlemleyerek eğitim miktarını arttırmaktır. Ancak DAgger daha iyi bir sürüş performansı sağlasa da öğrenilmiş politikayı iyileştirmek adına uzmana çok defa başvurması gerektiğinden gerçek hayat durumları için çok da verimli olmayabilmektedir. Güvenli veri seti kümelemesi (SafeDAgger) algoritması [9] ise DAgger algoritmasının bir uzantısı olarak uzmana yapılan çağrıların sayısını, güvenli olmadığını tahmin ettiği güzergahlarla sınırlayarak, azaltmaya çalışmaktadır. DAgger algoritmasının bir diğer uzantısı ise takım veri seti kümelemesi (EnsembleDAgger) algoritmasıdır [10]. Bu algoritma ise karar mekanizmasındaki dağılımları birden çok model ve dağılım haritası kullanarak bir güvenlik kriterinden geçirerek tahmin etmeye çalışmaktadır.

Bu çalışmada DAgger ve SafeDAgger algoritmalarının örnekleme açısından daha verimli bir uzantısı olarak Verimli Seçmeli Güvenli Veri Kümelemesi algoritması önerilmektedir. SafeDAgger algoritmasına benzer şekilde, önerilen algoritma da öğrenilmiş politikanın sunduğu güzergahın güvenliliğini tahmin etmektedir. Buna ek olarak, önerilen algoritma öngörülen güzergahın farklı sürüş karakteristiği bölümlerine (güvenli sola dönüş ya da güvensiz düz gidiş gibi) göre sınıflandırılması ve buna göre

en çok dikkat isteyen bölümlerin belirlenmesi ile sürüş stratejisinin geliştirilmesini amaçlamaktadır. Her bir sonraki eğitim döngüsünde algoritma, daha önceki döngülerde belirlenmiş problemli bölümlere odaklanarak bu bölümler hakkında daha fazla uzman politika örneği toplamaya çalışmaktadır. Bu da eğitim kalitesini arttırarak, uzman politikaya yakınsamayı hızlandırmaktadır. Çalışma sırasında yapılan simülasyon deneyleri sonucunda Verimli Seçmeli Güvenli Veri Kümelemesi algoritması ile standart SafeDAgger algoritmasından, aynı maksimum uzman sorgusu ile sınırlandırıldığında, daha iyi performans elde dildiği görülmektedir.

Dahası, önceki sürücüsüz araca yönelik uçtan uca derin öğrenme çalışmalarında temel odak noktası bir veya bir seri görüntü kullanarak direksiyon açısını tahmin etmeye çalışmak iken bu çalışmada bunun yanında hızlanma ve yavaşlama kontrolünün girdileri de tahmin edilmeye çalışılmaktadır.

II. KULLANILAN YÖNTEMLER

Bir makine öğrenmesi algoritması, verilerden öğrenme yeteneğine sahip algoritmadır. Mitchell'in 1997 yılında çıkardığı kitabında [11] bu öğrenme durumu şu şekilde açıklanmaktadır. "E deneyiminden T görevini P performans kriterine göre öğrenmek üzere tasarlanmış bir bilgisayar programının öğrendiğinin söylenebilmesi için bu programın T görevini yaparken ki performansının E deneyimi ile P performans kriterine göre artması gerekmektedir." Bu çalışma kapsamında düşünülecek olursa, sunulan algoritmanın bir öğrenme algoritması olabilmesi için, öğrenme boyunca sürüş yaparken verilen kriterlere göre daha güvenli sürüş yapar hale gelmesi anlamına gelmektedir. Bir makine öğrenmesi yöntemi olarak önerilen yöntem içerisinde yapay sinir ağları, taklit öğrenmesi, veri seti kümelemesi ve sınıflandırıcı gibi yöntemleri barındırmaktadır. Bu yöntemler kullanılarak oluşturulan algoritmanın test edilmesi için ise bir test ortamına ihtiyaç duyulmaktadır ve test ortamı bilgisayar simülasyon ortamında yaratılmıştır. Bu yöntemlerin ve simülatör ortamının çalışma için önemli ayrıntıları bu bölümde anlatılacaktır.

2.1. Evrişimsel Sinir Ağları

Evrişimsel sinir ağları ızgara benzeri bir topolojiye sahip olan verilerin işlenmesi için uzmanlaşmış bir tür sinir ağıdır [12]. Örnek olarak düzenli zaman aralıklarında numuneleri alarak 1 boyutlu ızgara olarak düşünülebilecek zaman serisi verileri ve 2 boyutlu piksel ızgarası olarak düşünülebilecek resim verileri düşünülebilir. Evrişimli

ağlar pratik uygulamalarda çok başarılı olmuştur. “Evrişimli sinir ağı” adı, ağın evrişim denilen matematiksel bir işlem kullandığını gösterir. Evrişim özel bir tür doğrusal işlemdir. Evrişimli ağlar, genel matris çoğaltması yerine evrişimi, katmanlarından en az bir tanesinde kullanan basit sinir ağlarıdır.

2.2. Taklit Öğrenmesi

Gerçek dünya problemlerinde makine öğrenmesinden beklenti genel olarak tek bir kararın tahmin edilmesi değil, daha ziyade bir kararlar serisinin oluşturulmasıdır. Bu tarz problemler genellikle ardışık karar verme problemleri olarak tanımlanır. Otonom araçlar için örnek vermek gerekirse, bir aracın sürülmesi tek bir duruma karar vermek değil bir durumlar serisinin zaman içerisinde oluşturulmasıdır. Makine aracı sürdükçe, aracın çevresindeki dünya da değişmektedir. Bu da karmaşık kontrol mekanizmalarına ve geri besleme döngülerine ihtiyaç duyulmasına neden olur.

Örnek bir senaryo olarak çok basit bir araç tasarlandığı varsayılsın. Bu araç için verilmesi gereken tek karar dönüşler olsun. Aracın herhangi bir anda sadece sağa dön, sola dön ya da durumunu koru şeklinde üç seçeneği olsun. Ayrıca halihazırda zaten araç sürmeyi bilen bir uzmanın kararlarına erişim bulunduğunu varsayalım. Kendi kendine sürüş yapmasını istediğimiz aracın bu uzmanı taklit ederek, araç sürmeyi öğrenmesi istenmekte. Bu duruma taklit ederek öğrenme denilmektedir.

Araç her $\mathrm{t} = 1 \ldots T$ anında sensörlerinden karar vermesine yardım edecek bir s_t bilgisi almaktadır. Bu bilgi öndeki araca ait bir kamera görüntüsü veya bir radar verisi

olabilir. Bu bilgi ışığında araç $a_t$aksiyonunu, bu tanımladığımız üç dönüş seçeneğinden biri olmak üzere, gerçekleştirerek bir l_t ceza puanı alır. Bu ceza puanı doğru sürüşler için sıfır ancak bir kaza durumunda çok büyük olarak seçilebilir. Bu durum sonraki her t anı için bu şekilde devam eder.

Amaç bu sensör bilgilerini (s_t) aksiyonlarla (a_t) birbirine bağlayabilecek bir π fonksiyonunun öğrenilmesi olacaktır. Destekli öğrenme metodu kapsamında bu fonksiyona politika denmektedir. Politikanın başarısı, gözlem, aksiyon ve ödül üçlüsünden oluşan $\Omega = s_1, a_1, l_1, s_2, a_2, l_2, \dots s_t, a_t, l_t$ güzergâhı boyunca rastgele dünya durumlarına karşın beklenen cezaları ($\mathbb{E}_{\Omega\sim\pi}[\sum_{t=1}^{T} l_t\ ]$) minimize etmesi ile ölçülmektedir.

2.2.1. Sınıflandırma ile Taklit Öğrenmesi

Bir uzmandan alınmış bir Ω güzergâhı bir seri gözlem ve bir seri aksiyondan oluşmaktadır. Taklit öğrenmesinin ana fikri bu gözlemlerle aksiyonlar arasındaki ilişkiyi taklit etmesine yarayacak bir sınıflandırma bulmaktır.

Basitçe s bilgilerini a aksiyonlarına bağlayacak büyük, çok sınıflı bir sınıflandırma algoritması kullanılarak, N tane güzergâh için eğitilmiş gözlem ve aksiyon ikilileri için toplamda NT tane (T bir güzergahtaki gözlem ve aksiyon ikilisi sayısı) örnekten oluşan bir eğitim verisi elde edilebilir.

Algoritma II-1 Gözetimli Taklit Öğrenmesi Eğitimi

1: $D \leftarrow \langle (x, a) : \forall n, \forall (x, a, l) \in \tau_n \rangle$ **//Tüm gözlem/aksiyon ikililerini topla**

2: **return** $\mathcal{A}(D)$ **//Çok sınıflı sınıflandırma algoritmasını eğit**

Algoritma II-2 Gözetimli Taklit Öğrenmesi Testi

1: **for** $t = 1 \ldots T$ do

2: $x_t \leftarrow$ şu anki gözlem

3: $a_t \leftarrow f(x_t)$ // politikanın aksiyonu

4: a_t aksiyonunu uygula

5: $l_t \leftarrow$ ceza puanını gör

6: **endfor**

7: **return** $\sum_{t=1}^{T} l_t$ //toplam ceza puanı

Ancak bu eğitimin başarısı, uzmanın ne kadar iyi sürüş yaptığına ve seçilen sınıflandırma algoritmasının başarısına bağlı olacaktır. Bu algoritmanın ceza puanlarını hesaplamaya çalışan bir teorem, olabilecek maksimum ceza puanının şu şekilde hesaplanabileceğini göstermiştir.

ϵ sınıflandırıcının hata oranı olmak üzere:

Denklem II.1 Ceza Puanı Üst Sınırı

$$\underbrace{\mathbb{E}_{\Omega \sim \pi}\left[\sum_t l_t\right]}_{Politikanın\ \ ceza\ puanı} \leq \underbrace{\mathbb{E}_{\Omega \sim \text{uzman}}\left[\sum_t l_t\right]}_{uzmanın\ \ ceza\ puanı} + T^2\epsilon$$

En kötü koşulda tek bir adımdaki ceza $l^{\text{maks}}\epsilon$ ve tüm cezaların toplamı $l^{\text{maks}}\,T\epsilon$ olacaktır. Ceza puanlarının toplamının T değişkenine bağlı olması demek çok

küçük hata oranlarında dahi politikanın cezalarının şişmesine neden olacaktır, $\epsilon \geq 1/\mathrm{T}$ olduğu koşulda politika başarısız olacaktır.

2.2.2. Başarısızlık Analizi

Taklit öğrenmesi, gözetimli bir öğrenme olduğundan en büyük problem algoritmanın hatalarını telafi edebilmesidir. Algoritma sadece uzmandan gelen güzergahlarla eğitildiğinden eğitim, veri seti sadece olması gerektiği gibi düzgün kullanımları içerir. Bu durumların dışına çıkıldığında ne yapacağını çok da hayal edemez. Örneğin uzman güzergâh içerisinde hiçbir zaman aracın bariyerlere doğru ilerlediği bir veri bulunmaz. Dahası uzman güzergâh genellikle ileri yönlü sürüş yaptığından algoritmanın geri gitmekle alakalı bir bilgisi olmayabilir. Bu durumda henüz mükemmel bir sürüş yapamayan algoritma birkaç hata yapıp bir bariyere doğru sıkıştığı zaman geri giderek duvardan kurtulabileceğini bilemez. İleri doğru gitmeye çalışarak sonsuza kadar o bariyerde takılı kalacaktır. Bu duruma bileşik hata denir.

Bileşik hata durumunu şu şekilde örnekleyebiliriz. Algoritmaya $t = 0$ anında içinde sıfır ya da bir bulunduran bir resim gösterilsin. Algoritmanın yapması gereken tek şey bu resmin sıfır ya da bir olduğunu bildirmek. Doğru bildirim yapmak $l_t = 0$; yanlış bildirim ise $l_t = 1$ şeklinde sonuçlanacaktır. Sonrasında $t = 1$ anında yeni bir resimle yine sıfır ya da bir gösterilsin. Algoritmanın bu adımda yapması gereken xor kullanarak eğer gördüğü resim önceki ile aynı ise önceki cevabı vermeli, değil ise diğer cevabı vermelidir. Ve bu algoritma $t > 1$ için bu şekilde devam etsin.

Bu algoritmada uzman kolaylıkla hiç hatasız testi tamamlayabilir. Ancak eğitilen algoritma en ufak bir hata yaptığında bundan sonra yapacağı tüm kararlar yanlış olacaktır ("şans eseri" tekrardan hata yapıp doğru yola dönene kadar).

Denklem II.2 Ceza Puanı ve Hata Oranı Arasındaki İlişki

ϵ sınıflandırıcının hata oranı olmak üzere, optimal bir uzman eşliğinde çalışan algoritmanın ϵ hata oranıyla alabileceği ceza puanı en az şu şekilde hesaplanmaktadır:

$$\underbrace{E_{\Omega\sim\pi}\left[\sum_t l_t\right]}_{Politikanın\ \ ceza\ puanı} \geq \frac{T+1}{2} - \frac{1}{4\,\epsilon}[1-(1-2\epsilon)^{T+1}]$$

Bu da alt sınırın T^2 şeklinde büyüdüğüne işaret etmektedir.

2.3. Veri Seti Kümelemesi

Destekli taklit öğrenmesi "uzman güzergâhından" çıktığı zaman tek başına kaldığı için başarısız olmaya mahkûm bir öğrenme şeklidir. İdeal olarak, eğer algoritma mümkün olan tüm durumlar için eğitilebilseydi, sorunlarla karşılaşıldığında da ne yapacağını bilirdi. Ancak bunu gerçekleştirmek mümkün değildir ve mümkün olsaydı bile bunun adı öğrenme değil ezberleme olurdu. Algoritma ezberden tüm durumlarla başa çıkabilirdi. Bu mümkün olmadığından algoritma politikası π tüm durumların bir alt kümesiyle eğitilmektedir, ancak sadece uzman tarafından sağlanmış durumların öğrenilmesi durumunda algoritma kendi hatalarını telafi etmeyi öğrenemeyecektir. Bir şekilde algoritmanın kendi başına gelen hatalardan da öğrenmesini devam ettirmesi gerekmektedir.

Politikanın karşılaştığı durumlara karşı eğitilmesi süreci şu şekilde işlemelidir. Bir π politikası ile başlanır ve bu politikanın koşturulması sırasında nelerle karşılaştığı kaydedilir. Karşılaşılan durumlara karşı politika eğitilir ve başarılı olması sağlanır ve politika yeterince eğitilene kadar bu süreç tekrarlanır.

Bu tam olarak veri seti kümelemesi algoritmasının ("DAgger") yaptığı şeydir. Otonom araç için örneklendirmek gerekirse. Başlangıçta bir süre bir uzman sürücünün başlangıç politikası π_0'ın oluşturulması için aracı sürmesi sağlanır ve politika eğitilir. Sonrasında uzman sürücünün araç üzerindeki kontrolü etkisiz hale getirilir ve $\pi_0$politikasının aracı sürmesi sağlanır. π_0 politikasının hata yaptığı durumlarda dahi uzman aracı kontrol etmeye çalışır ancak aracın kontrolü $\pi_0$politikasında olduğundan bu çaba bir sonuç getirmez ve araç ideal sürüş rotasından çıkar. Tüm bu durum kaydedilir.

π_0 politikasının yaratmış olduğu rota ile bu rotaya girildiğinde uzmanın yapmaya çalıştığı aksiyonların kayıtlarından oluşan yeni bir eğitim verisi oluşturulmuş olur. Bu veri ile π_1 politikası hem başlangıç politikasının eğitildiği veri seti hem de π_0 politikasının hatasının yer aldığı yeni veri seti ile eğitilir ve artık π_0 politikasın tespit etmiş olduğu bir hatadan nasıl kurtulunacağını öğrenmiş bir politika oluşturulmuş olur. Bu süreç aşağıda verilmiş olan algoritmada belirtilen maksimum tekrarlama sayısınca tekrarlanır.

Algoritma II-3 DAgger Eğitimi $(A, MaksTekrar, N, uzman)$

1: $\left\langle \Omega_n^{(0)} \right\rangle_{n=1}^{N}$ ← uzmanı N kere koştur

2: $D_0 \leftarrow \left\langle (s,a) : \forall n, \forall (s,a,l) \in \Omega_n^{(0)} \right\rangle$ //Tüm ikilileri topla

3: $\pi_0 \leftarrow A(D_0)$ //Başlangıç politikasını eğit

4: **for** $i = 1 \ldots MaksTekrar$ do

5: $\left\langle \Omega_n^{(i)} \right\rangle_{n=1}^{N}$ ← Politika π_{i-1} 'i N kere koştur

6: $D_i \leftarrow \left\langle \left(s, uzman(s)\right) : \forall n, \forall (s,a,l) \in \Omega_n^{(i)} \right\rangle$ //veri seti topla: π_{i-1} tarafından gözlenmiş durumlar ve uzman tarafından alınmış aksiyonlar

7: $\pi_i \leftarrow A\left(\bigcup_{j=0}^{i} D_j\right)$ //π_i politikasını şimdiye kadarki tüm veri setleri ile eğit

8: **endfor**

9: **return** $\langle \pi_0, \pi_1, \ldots, \pi_{MaksTekrar} \rangle$

Verilmiş olan bu algoritma sonucunda öğrenme süreci boyunca üretilmiş tüm politikalar ($\langle \pi_0, \pi_1, \ldots, \pi_{\{MaksTekrar\}} \rangle$) çıktı olarak sunulmaktadır. Ancak bunlardan hangisinin en iyi sonucu verdiğini söylemek doğrudan mümkün değildir. Mantıken son üretilen sonucun ($\pi_{\{MaksTekrar\}}$) diğerlerinden daha iyi sonuç verdiği düşünülebilir ancak DAgger doğası gereği dengesiz davranabilir ve monoton olarak başarısını arttıran örnekler üretemeyebilir. Bu durumda son üretilen politika diğerlerinden daha iyi olmayabilir. Sonuncuyu seçmek dışındaki yöntem ise, tüm politika çıktılarını apayrı bir deney senaryosunda test ederek orada en iyi sonucu vereni seçmek olacaktır. Bu yöntem daha fazla işlem yükü getirse de genel olarak daha doğru sonucu vermektedir.

DAgger algoritması ve gözetimli öğrenme arasındaki temel fark uzman ile olan ilişkidir. Gözetimli taklit öğrenmesinde, uzmanın daha önceden gerçekleştirmiş olduğu aksiyonlara pasif olarak göz atılmakta iken, DAgger algoritmasında bunun yanında uzmana "Sen bu '*s*' durumunu görsen ne yapardın?" şeklinde sürekli sorular sorulmaktadır. Bu da uzmana çok daha fazla talep anlamına gelmektedir.

Bir diğer soru ise DAgger algoritması içinde yer alan N sayısının kaç olacağıdır. Pratikte en başta başlangıç politikasını oluştururken bu sayıyı yüksek tutmak iyi bir π_0 politikasıyla başlamayı sağlayarak, sonraki her iterasyondaki tekrar sayısının bire kadar düşmesini sağlayabilir.

DAgger algoritması bileşik hatalara karşı daha güçlüdür denilebilir, çünkü test sırasında karşılaştığı durumlara karşı eğitilir. Bu durum şu teoremle gösterilebilir:

Denklem II.3 DAgger Algoritmasının Ceza Puanı

ϵ sınıflandırıcının hata oranı, π öğrenilmiş politika olsun ve anlık kayıpların $[0, l^{(maks)}]$ arasında olduğunu varsayalım

$$\underbrace{E_{\Omega\sim\pi}\left[\sum_t l_t\right]}_{\text{Politikanın ceza puanı}} \leq \underbrace{E_{\Omega\sim uzman}\left[\sum_t l_t\right]}_{\text{Uzmanın ceza puanı}} + l^{(maks)}T\epsilon + O\left(\frac{l^{(maks)}T\log T}{MaksTekrar}\right)$$

Eğer ceza fonksiyonu yakınsak ise ve maksimum tekrar sayısı da $\tilde{O}\left(\frac{T}{\epsilon}\right)$ ise;

$$\underbrace{E_{\Omega\sim\pi}\left[\sum_t l_t\right]}_{\text{Politikanın ceza puanı}} \leq \underbrace{E_{\Omega\sim uzman}\left[\sum_t l_t\right]}_{\text{Uzmanın ceza puanı}} + l^{(maks)}T\epsilon + O(\epsilon)$$

Bu sonuçlar, eğer $MaksTekrar$ sayısı yeterince yüksek ise, öğrenilmiş π politikasının ceza puanının Tϵ şeklinde artacağını göstermektedir, ki bu da istediğimiz bir sonuçtur.

2.3.1. Masraflı Algoritmaların Uzman Olarak Kullanılması

DAgger algoritması içerisinde uzmana olan bağımlılık yüksek olduğundan, DAgger algoritmasının en önemli kullanım senaryolarından biri normalde yavaş olan algoritmaları taklit edip öğrenmesidir. Buna şu iki durum örnek gösterilebilir:

i. Oyun Oynama. Satranç ya da bir bilgisayar oyunu gibi bir oyun sırasında, bir durumda kaba kuvvet yaklaşımı ("Brute Force") ile arama yapmamızı gerektiren yarı optimum bir uzman algoritmanın çalıştırılması gerekebilir. Ancak gerçek zamanlı bu oyun sırasında bu algoritma çok masraflı olacağından, bu işlemi taklit edecek bir politikanın eğitimi sırasında kullanılması ve gerçek zamanlı oyun sırasında, taklit eden bu politikanın kullanılması daha az masraflı olabilir.

ii. Ayrık En İyileştiriciler ("Discrete Optimizers"). Ayrık en iyileştirme problemleri gerçek zamanlı işlem yapmak için çok masraflı olabilmektedir. Örneğin büyük bir tablo içerisinde en kısa yolu bulmak bile gerçek zamanlı arama için çok yavaş kalmaktadır. En kısa yolları eğitim verisi olarak bir politika için kullanıp bu politikanın gerçek zamanlı olarak gerçek algoritmayı taklit etmesi daha verimli şekilde çalışabilir.

Satranç oynamayı öğrenmeye çalışan bir politika yaratmaya çalışırken, DAgger eğitimi algoritması (Algoritma II-3) her adımda uzman algoritmadan, oynandığı takdirde oyunun sonunda oyuncunun kazanmasını sağlayacak bir a aksiyonu bulmasını talep edecektir. Bunu kesin bir doğrulukla hesaplamak, çok basit oyunlar haricinde, çok zor olacaktır. Bu durumda yeni bir yaklaşımda bulunmak gerekebilir.

Algoritma II-4 DerinliğiSınırlıDerinlikÖncelikliArama $(x, h, MaksDerinlik)$

1: **if** $x\ bir\ uç\ noktası$ ya da $MaksDerinlik \leq 0$ **then** //eğer daha derine gidilemiyorsa
2: **return** $(\perp, h(x))$ // "aksiyon yok" kararını ve puanı döndür
3: **else**
4: $EnİyiAksiyon, EnİyiPuan \leftarrow \perp, -\infty$ //en iyi aksiyon ve puanını takip et
5: **for all** $a\ aksiyonu$ **from** x **do**
6: $(\_, puan) \leftarrow$ **DerinliğiSınırlıDerinlikÖncelikliArama(**$\boldsymbol{x \circ a, h, MaksDerinlik - 1}$**)**
//a aksiyonunun puanını al ve derinliği bir
//bir azaltarak algoritmayı yeniden çağır
7: **if** $puan \geq EnİyiPuan$ **then**
8: $EnİyiAksiyon, EnİyiPuan \leftarrow a, puan$ //En iyi aksiyon ve puanı güncelle
9: **endif**
10: **endfor**
11: **endif**
12: **return** $(EnİyiAksiyon, EnİyiPuan)$ //En iyi aksiyon ve puanı döndür

Bu durumda genel yaklaşım, derinliği sınırlandırılmış bir derinlik öncelikli arama algoritması kullanmak olabilir. Bir s durumundan başlayarak ve belirli bir derinliğe kadar inilir. Bu sayede bir arama ağacı oluşturulur. Ancak oyunun sonları haricinde bu

ağacın dalları bir uçbirim olmayacağından, bu uçbirim olmayan dalların durumunu değerlendirmek için biraz daha buluşsal olmak gerekebilir. Bu buluşsal puanları ağacın kök birimine kadar toplayarak ilerlendiğinde nispeten daha iyi sonuç veren bir aksiyon elde edilir. Bu aksiyon olabilecek en iyi aksiyon olmayabilir ama burada hız ve tahmin kalitesi arasında bir ödünleşim vardır.

2.4. Güvenli Veri Seti Kümelemesi

DAgger algoritmasının destekli öğrenme yaklaşımını daha isabetli bir hale getirmek için başarılı bir algoritma olduğu önceki bölümlerde açıklanmıştır. Ancak özellikle eğitim aşamasında DAgger uzmana her durumda bağlı olduğu için maliyetli bir algoritma olduğu da görülmektedir. Bu durumun önüne geçilmek adına Zhang J. Ve Cho K. [9] çalışmalarında DAgger algoritmasına bir güvenlik sınıflandırıcısı ekleyerek, eğitim ve test aşamalarında uzman sorgulamasını daha verimli bir hale getirmeyi amaçlayan SafeDAgger algoritmasını öne sürmüşlerdir.

2.4.1. Güvenlik Sınıflandırıcısı

Bu çalışmada öne sürülen güvenlik sınıflandırıcısı $c_{\text{güvenlik}}$, $s(t)$ durumunu ve birincil π politikasını alarak, π politikasının referans alınan uzman π^* politikasından sapma ihtimalini, uzman π^* politikasını sorgulamadan döndürmeyi hedefler.

Birincil π politikasının, uzman π^* politikasından sapma ihtimali şu denklem yardımı ile gösterilebilir:

Denklem II.4 Politikanın Hata Oranı

$$\epsilon(\pi, \pi^*, s(t)) = \left|\left|\pi(s(t)) - \pi^*(s(t))\right|\right|^2$$

Hata metriği bu çalışmada iki politika arasındaki çıktı farkının karesi olarak alınmış olsa da bu metrik yapılan çalışmaya göre değiştirilebilir, genişletilebilir. Bu denkleme bakılarak optimum güvenlik sınıflandırıcısı $c^*_{\text{güvenlik}}$ şu şekilde tanımlanabilmektedir:

Denklem II.5 Güvenlik sınıflandırıcısı

$$c^*_{\text{güvenlik}}(\pi, s(t)) = \begin{cases} 0, eğer\ \epsilon(\pi, \pi^*, s(t)) > \tau \\ 1, diğer \end{cases}$$

Denklemdeki τ daha önceden belirlenmiş bir eşik değeri olup, bu değere bağlı olarak güvenlik sınıflandırıcısı, π politikasına içinde bulunulan $s(t)$ durumunda güvenilip güvenilemeyeceğini gösterir. Bu gösterim uzman sorgulaması yapılmadan yapılır.

Öğrenme Güvenlik sınıflandırıcısı kendiliğinden ortaya çıkan bir algoritma değildir, eğitim sırasında başka bir eğitim seti ya da basitçe orijinal eğitim setinin bir alt kümesini kullanılarak oluşturulur. Bu set şu şekilde gösterilebilir: $D' = \{s'(1), s'(2), \dots, s'(N)\}$.

Sürüş: Güvenli Strateji Destekli öğrenmenin ötesinde, tanımlanan güvenlik sınıflandırıcısı $c_{\text{güvenlik}}$ sayesinde daha güvenli bir sürüş stratejisi oluşturulabilir. Bu stratejide sınıflandırıcı, her bir zaman aralığı için birincil politikanın sürüşünün güvenli olup olmadığını belirler. Eğer güvenli ise (örneğin: $c_{\text{güvenlik}}(\pi, s(t)) = 1$) birincil politikanın belirlediği aksiyon uygulanır. Eğer güvenli değil ise (örneğin: $c_{\text{güvenlik}}(\pi, s(t)) = 0$), referans alınan uzman π^* politikasının belirlediği aksiyon kullanılır.

Burada kullanılan güvenlik sınıflandırıcısının başarılı olduğu varsayılırsa, yüksek ceza puanlarına yol açacak kaza ve benzeri tehlikeli durumların önüne geçilmiş olacaktır.

Değer Fonksiyonu ile Bağlantı Destekli öğrenme içerisinde bir değer fonksiyonu $V^{\pi}(s)$, verilen politikanın şu anki s durumundan itibaren gelecekte toplayacağı değer puanlarını hesaplamakta kullanılır. Bu tanım, güvenlik sınıflandırıcısı ve değer fonksiyonu arasında bir bağlantı olduğunu göstermektedir. Güvenlik sınıflandırıcısı $c_{\text{güvenlik}}(\pi, s(t))$, birincil π politikasının şu anki durumdan bir ödül noktasına kadar referans uzman politikası ile aynı sonucu verip vermeyeceğine göre başarılı 1 ödülünü ya da başarısız 0 ödülünü verir.

Bu tanımlara göre güvenlik sınıflandırıcısının da aracı kendisi sürecek kadar bilgi sahibi olduğu gibi bir anlam akıllara gelmektedir. Güvenlik sınıflandırıcısını değer fonksiyonu olarak kullanma düşüncesi sonucunda, verilen bir s durumunda,

seçilebilecek en iyi $\hat{a}$ aksiyonu $\arg max_{a \in A(s)} c_{\text{güvenlik}}(\pi, \delta(s,a))$ şeklinde tanımlanabilirdi. Ancak bu formülasyon içerisindeki δ geçiş fonksiyonu belirsiz olduğundan mümkün değildir. Çalışma içerisinde tanımlanan önerilen güvenlik sınıflandırıcısının yalnızca durumu değil durum ve aksiyon ikililerini (s, a) göz önünde bulundurmak üzere genişletilmesi fikri ortaya atılmıştır. Bu sayede güvenlik sınıflandırıcısı bir Q değer fonksiyonuna daha yakın bir denklem haline gelebilir.

2.4.2. Döngüde Güvenlik Sınıflandırıcısı

Bu bölümde çalışma içerisinde sunulan SafeDAgger algoritmasının ayrıntılarına yer verilecektir. Orijinal DAgger algoritması üzerine tanımlanan iki temel değişiklikten söz edilmektedir.

Bunlardan ilki, orijinal DAgger algoritmasında yer alan saf strateji yerine güvenli strateji ile eğitim örneklerinin toplanmasıdır (Algoritma II-5 Güvenli Veri Seti Kümelemesi satır 6). Politika güvenli olmadığında kullanılmaz ve kontrol referans uzman politikaya geçer, bu sayede kaza yapmadan daha uzun eğitim örnekleri toplanabilir.

İkinci olarak, alt küme seçimi (Algoritma II-5 Güvenli Veri Seti Kümelemesi satır 7) referans uzman sorgularını önemli ölçüde azaltmaktadır. Yalnızca güvenlik sınıflandırıcısının 0 olarak tespit ettiği küçük bir alt küme, referans uzman aksiyonları ile etiketlenmektedir. Orijinal DAgger algoritmasında ise referans uzmanın aldığı tüm aksiyonlar etiketlenmektedir.

Algoritma II-5 Güvenli Veri Seti Kümelemesi

1: Referans uzman π^* politikasından D_0 topla

2: Referans uzman π^* politikasından $D_{\text{güvenlik}}$ topla

3: $\pi_0 = \arg\min_{\pi} l_{destekli}\ (\pi, \pi^*, D_0)$

4: $c_{güvenlik\ ,0} = \arg\min c_{güvenlik}\ l_{güvenlik}\ (c_{güvenlik}\ , \pi_0, \pi^*, D_{güvenlik} \cup D_0)$

5: for i=1 do M

6: Güvenli strateji π_{i-1} ve $c_{güvenlik\ ,i-1}$ kullanarak $D^{'}$ topla

7: Alt küme Seçimi

$D^{'} \leftarrow s(t) \in D^{'} | c_{güvenlik\ ,i-1}(\pi_{i-1}, s(t)) = 0$

8: $D_i = D_{i-1} \cup D^{'}$

9: $\pi_i = arg\min_{\pi} l_{destekli}\ (\pi, \pi^*, D_i)$

10: $c_{güvenlik\ ,i} = \arg\min c_{güvenlik}\ l_{güvenlik}\ (c_{güvenlik}\ , \pi_i, \pi^*, D_{güvenlik} \cup D_i)$

11: endfor

12: return π_M ve $c_{\text{güvenlik},M}$

(Mavi yazı tipi SafeDAgger algoritmasının DAgger algoritmasına eklediği bölümleri göstermektedir.)

Bunların yanında, bu alt küme seçimi sayesinde, seçimden sonra gelecek olan destekli öğrenme sırasında daha zorlu senaryolara odaklanıldığından, toplam eğitim örnekleri azalmasına rağmen asıl problemli bölümlere ait örnekler kaybedilmemiş oluyor.

Birincil politika hem D_0 eğitim seti hem de karşılaşılan zorlu senaryoları içeren D_i eğitim seti ile güncellendiğinde, güvenlik sınıflandırıcısı da güncellenir. Bu sayede güvenlik sınıflandırıcısının, güncellenmiş politikadaki zorlu ve tehlikeli bölümlere odaklandığından emin olunur.

Orijinal DAgger algoritması ve SafeDAgger algoritması arasındaki farklara rağmen, çalışma içerisinde bulunan τ eşiği yavaşça her adımda arttırılarak, SafeDAgger algoritmasının da DAgger algoritmasının sağladığı tüm teorik gereklilikleri yerine getirmesi sağlanmıştır. Çalışma içerisinde sonradan, bunu sağlamak için sabit bir τ değerinin bile yeterli olduğunun deneylerle kanıtlandığından bahsedilmektedir.

2.5. Kullanılan Oyun Motoru

Unreal Engine 4, iki ve üç boyutlu oyun geliştiricileri için kütüphaneler ve tasarım ortamı sunan, güçlü ve gelişmiş bir oyun ve grafik motorudur. Unreal Engine 1998 yılında Epic Games tarafından sunulmuş ve o zamandan günümüze kadar geliştirilerek dördüncü jenerasyonunu 2014 yılında yayınlanmıştır, yayınlandığı günden beri sayısız oyun için altyapı oluşturmuştur.

Motor C++ dilinin üzerine kurulduğu için, birçok açık kaynaklı veya lisanslı kütüphaneyi bulundurmakta ve kullanımını mümkün kılmaktadır. Motorun kendisi ‘Kullanılabilir kaynaklı’ (‘source-available’) olarak lisanslandırılmıştır. Bu lisans tipinin tamamen açık kaynaklı lisanstan farkı bazı durumlarda lisanslandırma ücretinin uygulanabilir olmasıdır. Ancak bu lisanslandırma karlılık, satış miktarları gibi finansal konularca belirlenmektedir. Kullanılabilir kaynaklı lisans sayesinde kullanıcılar sadece kendi kodlarını değil, motora ait kodları ve kütüphaneleri de değiştirip düzenleyebilmektedir. Bu da kullanıcılara yüksek esneklik sağlamaktadır.

2.5.1. Motorun Özellikleri

Motorun kendiliğinden sağladığı modüller ile birçok özellik kullanıcılara sunulmaktadır. Aşağıda bu özelliklerden bazıları listelenmiştir.

2.5.1.1. Fotoğraf Kalitesinde Görselleştirme

Epic Games gerçek zamanlı ve fotoğraf kalitesinde grafiklerin bu motor ile görselleştirilebildiğini iddia etmektedir. İçerisinde dinamik gölgeler, ışıklandırmalar ve yansımalarla alakalı modüller bulundurduğundan, insan gözü için gerçek hayata en yakın deneyimi sunduklarını belirmektedirler. Bu durum, ADAS işlerinde kullanılacak simülatör tasarımlarında gerçek hayattaki ‘Mobileye’ gibi kamera sensör sistemlerinin taklit edilmesi için bir fırsat sunmaktadır.

2.5.1.2. Açık Kaynaklı C++ Kodları

Bütün kütüphaneler, değişikliklere ve hata ayıklama işlerine açık şekilde sunulmuştur. Bu sayede kullanıcı sadece kendi eklediği kodları değil motora ait olan kodları da kendine göre değiştirebilir ya da hata ayıklaması için kullanabilir.

2.5.1.3. Mavi Kopya (Blueprint)

Motor kullanıcılarına Blueprint adında, tek bir satır kod yazmadan modelleme ve programlama yapabilecekleri ve bir görsel kodlama altyapısı da sunmaktadır. Kullanımı tercihe bağlı olan bu altyapı sayesinde kullanıcılar, daha ziyade sürükle bırak

elemanlarından oluşan bir arayüz ile geliştirmelerini yapabilmekte ve akış şeması benzeri bir yapıya sahip olan görsel arayüz yardımı ile hata ayıklama ve kod takibini daha rahat bir şekilde gerçekleştirebilmektedir

2.5.1.4. Arazi ve Bitki Örtüsü

Motor içerisinde arazi yapısı, bitki örtüsü, yeryüzü elemanlarını ve yeryüzü şekillerini verimli bir şekilde oluşturmaya yarayan bir süsleme aracı bulunmaktadır. Bu araç sayesinde ağaçlar, çalılar ve kayalar gibi dekorasyon öğeleri kolayca yerleştirilebilmekte ve gerçek dünya benzeri bir simülasyon ortamı yaratılabilmektedir.

2.5.1.5. Gelişmiş Yapay Zekâ

İçinde barındırdığı yapay zekâ paketleri sayesinde motor, aktör elemanlara çevresel farkındalık ve akıllı hareket düzenleri sağlayabilmektedir. Daha ziyade aksiyon oyunları için geliştirilmiş bu paketler yaya elemanların hareketleri için tasarlanmış olsa da bazı paketler yardımı ile simülatör içerisinde tekerlekli araçlar için de bir yapay zekâ modülü oluşturmak için kullanılabilmektedir.

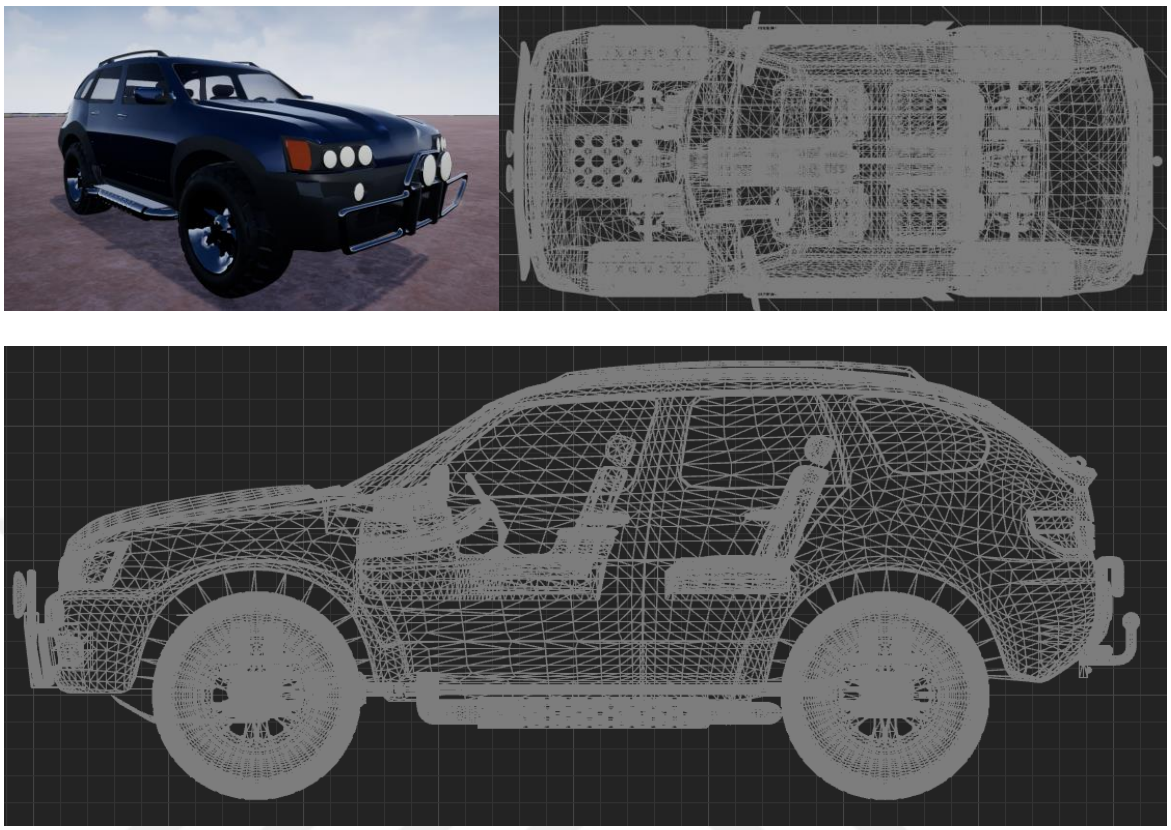
2.5.1.6. Limitsiz Genişletilebilirlik

Motorun sonradan eklenebilir modüler (plug-in) yapısı sayesinde sonradan geliştirilebilir tüm özellikler için bir altyapı sağlanmaktadır. Epic Games bu sayede hayal edilebilecek tüm teknolojilerin bu motora entegre edilmesinin mümkün olduğunu iddia etmektedir.

2.5.2. Araç Modeli

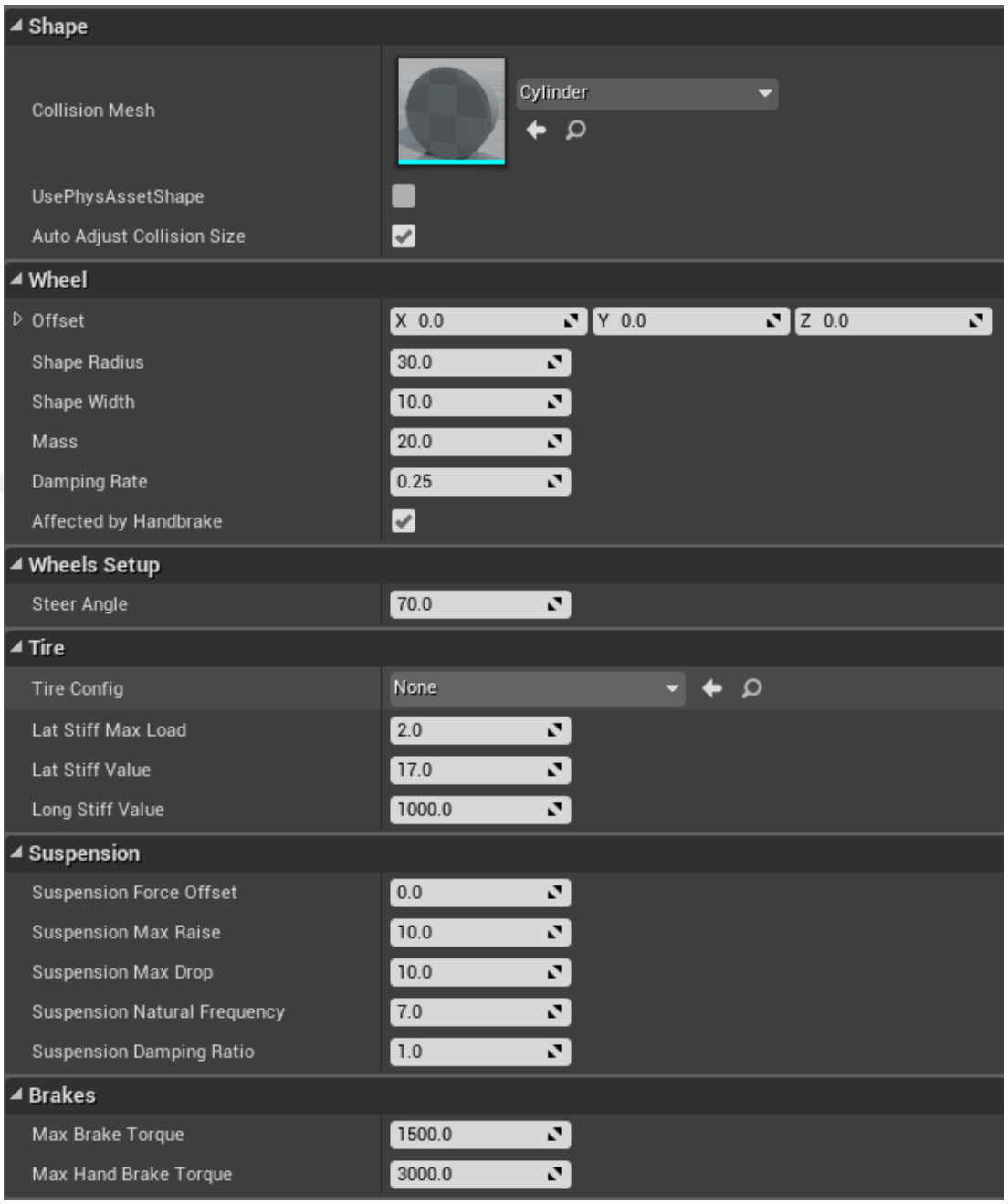
Unreal Engine 4 oyun motoru sayesinde gerçeğe yakın seviyelerde tekerlekli araç modellemeleri mümkündür. Araç modellemesi için Unreal Engine 4 kütüphaneler sunsa da bunların ihtiyaçlar doğrultusunda birleştirilmesi ve ayarlanması gerekmektedir. Bu çalışma içerisinde sürücüsüz arabalar üzerine geliştirme yapılacağından 4 tekerlekli ve önden direksiyon sistemli bir araç tasarlanması gerekmektedir. Kontrollerin kolaylığı ve araç dinamiklerinin daha kararlı olması adına 4 tekerden çekiş sistemi seçilmiştir. Bahsedilen bu aracın tasarımı içerisinde birkaç temel parça bulunmaktadır.

Bu parçaların ilki tekerleklerdir. Unreal Engine 4 içerisinde bir tekerlek tasarımı ile gelmektedir. Bu tekerlek tasarımı dahilinde tekerleğin boyutları ve ağırlıklarının yanı sıra tekerleklere bağlı süspansiyon sisteminin kalibrasyonu, lastik yapısı ve malzemesi ve çekiş özellikleri Şekil II.2'de görüldüğü gibi istenildiği gibi ayarlanabilmektedir. Bu özelliklerle birlikte araç üzerine yerleştirilecek bir tekerleğin çekiş gücü, fren gücü ve direksiyon komutlarına tepki verip vermeyeceği ile birlikte bu komutların her biri ile alakalı kalibrasyon değerleri de girilebilmektedir. Bu değerler istenildiği gibi önceden ya da dışarıdan alınabilecek bir araç dinamiği modeli ile tanımlanabilmektedir. Bu özelliklerin yanında görsel amaçlı tekerleklere silindir şeklinde istenilen giydirme yapılarak gerçekçi bir tekerlek görünümü verilir.



Şekil II.1 3 Boyutlu SUV araç modeli

Diğer bir temel parça ise araç gövdesidir. Araç gövdesi de tekerleklere benzer olarak son derece kalibre edilebilir bir bölümdür. Gövdenin boyutları, ağırlığı, ağırlık merkezinin tam yeri ve istenirse tüm oynar parçaları bu model sayesinde ayarlanabilmektedir.



Şekil II.2 Unreal Engine 4 tekerlek kalibrasyon arayüzü

III. OTONOM SÜRÜŞ

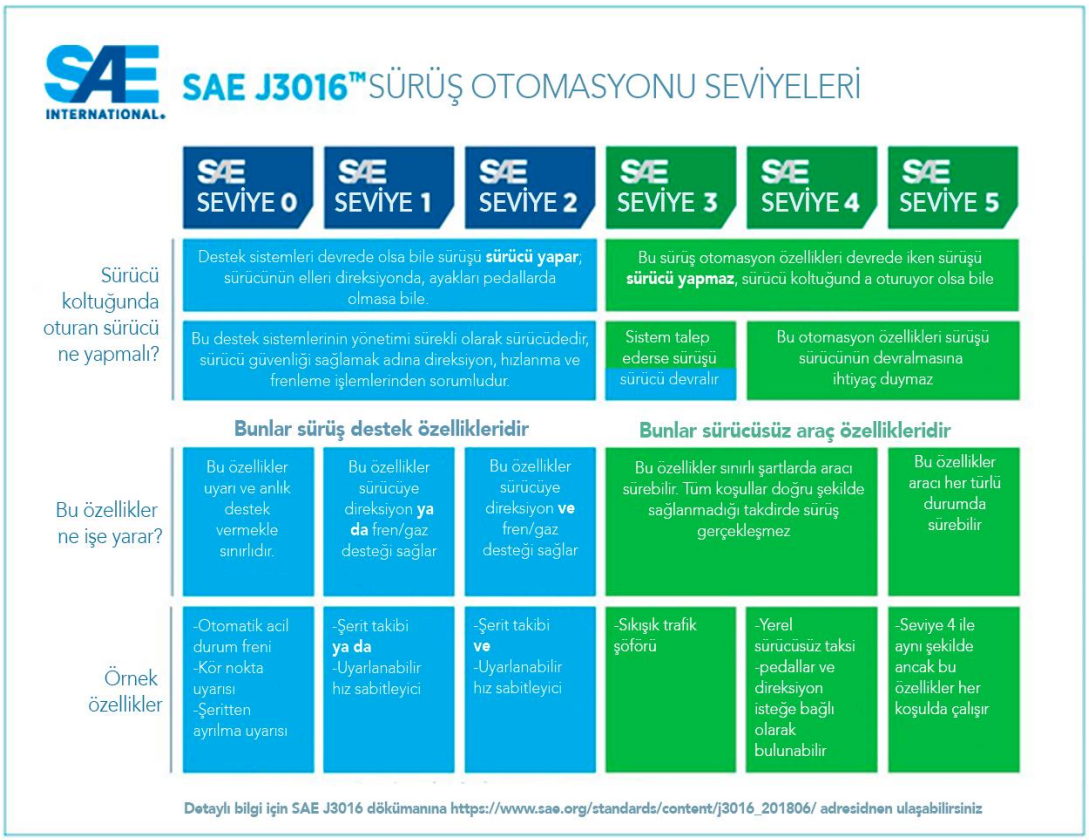
Günümüzde satılmakta olan birçok araç halihazırda bir takım otonom sürüş özellikleri ile donatılmış durumdadır, hatta birçok sürücüsüz araç prototipi Avrupa, Japonya ve Amerika gibi konumlarda yollarda test edilmektedir. Bu teknolojiler markete çok hızlı şekilde giriş yaptığı gibi, bu teknolojilerin daha da hızlı çoğalması ve yaygınlaşması beklenmektedir. Sürücüsüz aracın birçok yararından bahsetmek mümkündür; ileri teknoloji güvenlik, trafik tıkanıklarının azalması, yakıt ekonomisi ve doğanın korunması ile birlikte araç sahiplerinin bunlara bağlı stres seviyesinin azalması akla ilk gelen yararlardandır. Otoriteler de halihazırda var olan kuralları değiştirerek ve yenilerini ekleyerek bu gelişimin güvenlik, yasal sorumluluk ve gizlilik açısından insanların iyiliği adına sonuçlandığından emin olmalıdırlar. Sürücüsüz otomobil teknolojisi yüksek oranda yetişkinlik seviyesine gelmiştir. Tamamen sürücüsüz otomobiller için gerekli çekirdek teknolojilerin neredeyse tamamı günümüzde mevcuttur ve bu teknolojilerin de çoğu birçok araca sürücü destek sistemi altında eklenmiştir.

Sürücüsüz araç kavramı içerisinde birçok geniş çağlı teknoloji, altyapı, beceri, içerik, iş alanı, ürün ve servisleri barındırmaktadır. Dahası otomotiv otomasyonu çok daha büyük bir otomasyon ve ağın bir parçasıdır. Bu ağdaki en güncel parçalar; kişisel bilgisayarlar, cep telefonları ve internet, birbirlerine yakınsamış ve iç içe geçmişlerdir, hatta diğer makinelerle de birleşip fiziksel ortamda algı ve etki yeteneği de kazanmışlardır. Bu makineler sadece motorlu araçlar değil uçangözler, kişisel bakım robotları, 3 boyutlu yazıcılar, gözetleme cihazları ve birçoğundan oluşmaktadır.

Otomobil otomasyonundan söz ederken de aynı bu şekilde teknoloji ve toplumun tamamına etki edecek bir kavramdan söz edilmektedir.

3.1. Otonom Sürüş Seviyeleri

Otomotiv Mühendisleri Topluluğu (SAE), Amerika merkezli, otomotiv, havacılık ve ticari araçlar konusunda uzman ve küresel çapta standartlar belirleyen bir kuruluştur. Alanında araçların en küçük motor parçalarından, en genel konulara kadar birçok konuda standartları belirleyen SAE otonom sürüş hakkında da 2016 yılında SAE J3016 adı ile bir standart belirlemiştir. Bu standart en son 15 Haziran 2018'de güncellenmiştir ve otonom sürüşü 6 seviyede şu şekilde belirlemektedir:

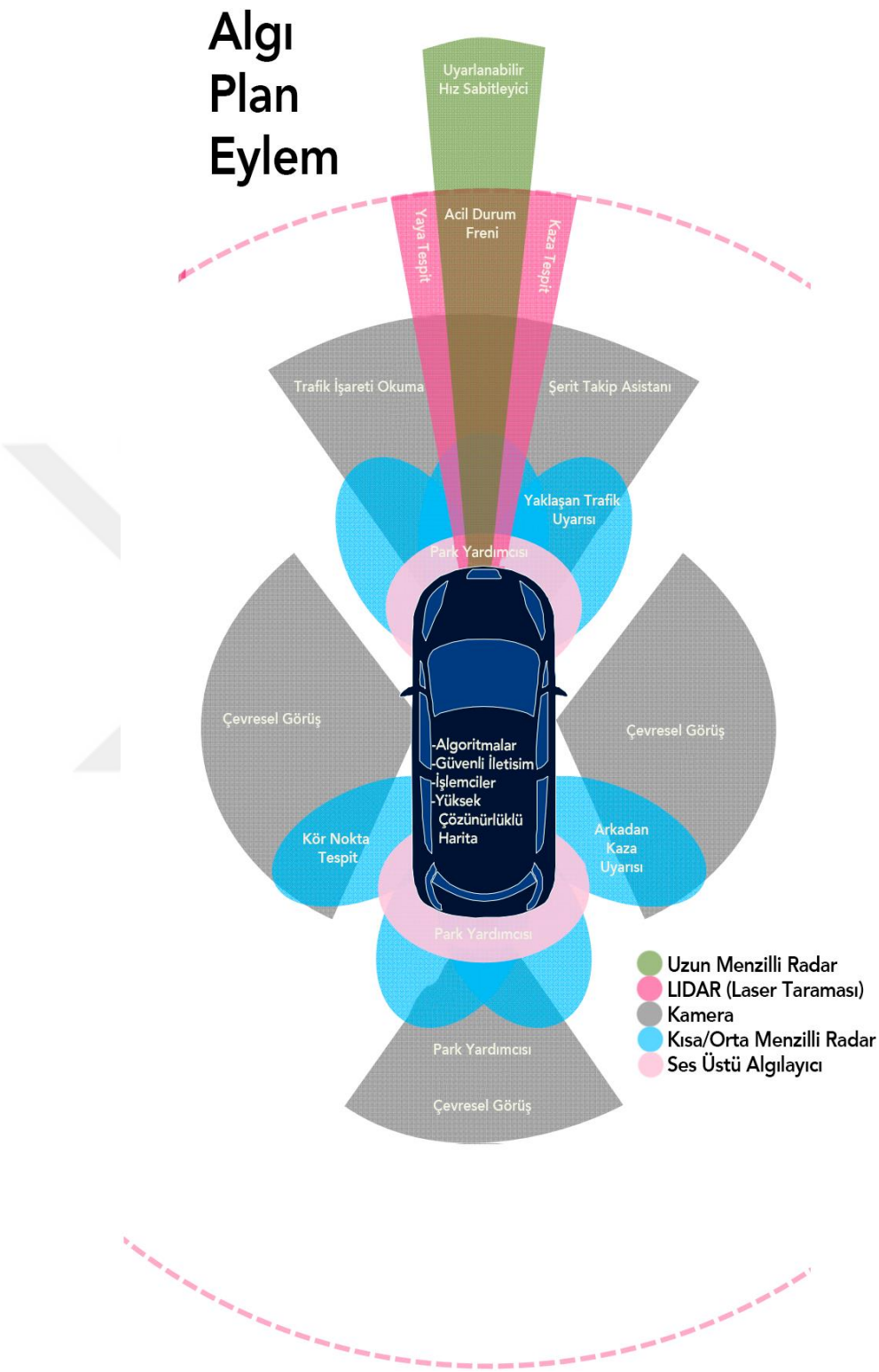


Şekil III.1 Kaynak: SAE Standardı J3016 (Çeviri: F. Orkun KIZILIRMAK)

Şekil III.1'de de görüldüğü gibi temelde bu seviyeler iki bölümde incelenmektedir; sürücü destek sistemleri ve sürücüsüz araç sistemleri. Sürücü destek sistemleri her daim sürücünün gözetimine ve sorumluluğuna ihtiyaç duymaktadır. Sürücüsüz araç ise belirli koşullar altında ya da her koşulda aracı tamamen kendi kendine sürebilen sistemdir. Bu seviyeler içerisinde kullanılan donanımlar ise büyük oranda benzerlikler gösterebilmektedir. Seviyeleri birbirinden ayıran şey ayrı ayrı donanımlar değil, bu donanımların birbiri ile uyum içinde çalışabilmesidir.

3.2. Sürücüsüz Araç Donanımları

Hangi seviye olursa olsun bir aracın otonom gidebilmesi için algılama, karar alma ve uygulama için çeşitli donanımlara ve yazılımlara ihtiyaç vardır. Bu donanımların ve sağladığı çıktıların bazıları Şekil III.2 üzerinde gösterilmiştir. Bu donanımların tamamı bir araya gelmeden tamamen güvenli bir otonom sürüşten bahsetmek günümüz şartlarında mümkün değildir. Otonom sürüş sırasında sadece aracın gideceği güzergâh ve yolun şekli değil, diğer araçlar, yayalar, engeller gibi dış etkenlerin de göz önünde bulundurulması gerekmektedir. Bu nedenle ayrı ayrı bu donanımlar ne kadar gelişmiş olsa da otonom sürüşü tek başına gerçekleştiremezler.



Şekil III.2 Araçların algılaması, plan yapması ve dinamik sürüş ortamına tepki vermesini sağlayan teknolojiler

3.3. Çalışmanın Otonom Sürüş İçerisindeki Yeri

Önceki bölümde bahsedildiği üzere hali hazırda mevcut hiçbir sistem tek başına tamamen otonom bir sürüş yapma yeteneğine sahip değildir. Bu durum önerilen algoritma için de geçerlidir. Ancak önerilen sistem aracı yayalar, diğer araçlar vb. diğer engellerin olmadığı özel bir ortamda insansız bir şekilde idare edebilmektedir. Kendinden önce gelen uçtan uca derin öğrenme yöntemlerinin verimliliğini artırmak için öne sürülen algoritma, aracın ileri yönlü bakan kamerasından aldığı görsel verileri kullanarak aracın direksiyon açısı ve hız çıktılarını üretmektedir. Bunu yaparken de eğitimler sırasında kendine öğretilen güzergâh tiplerinde alınan aksiyonları taklit etmektedir.

Bu bağlamda algoritma şu haliyle bir sürücü destek sistemi olarak çalışabilmektedir. Sürücüye şehir içi veya trafikli ortamlarda sürüşü devrederken, açık otoban veya pist gibi ortamlarda sürücüden kontrolü devralabilir ya da sürücü için direksiyon açısı ve hız gibi çeşitli uyarılar veya öneriler üretebilir.

Bu çalışma her ne kadar ileride sürücüsüz araçlarda kullanılabilmek için tasarlanmış olsa da temelde bir makine öğrenmesi yöntemidir. Bu çalışma içerisinde günümüzün popüler konularından olan sürücüsüz araçlar üzerine bir uygulama ile gerçeklenmiştir. Ancak bu çalışma içerisindeki sözde algoritmaların (Pseudo Algorithm) başka alanlara uygulanarak bu alanlarda da verimli şekilde çalışabilmesi mümkündür.

IV. VERİMLİ SEÇMELİ GÜVENLİ BİR VERİ SETİ KÜMELEMESİ ÖNERİSİ

Sürücüsüz araçlara yönelik önceki çalışmalar bir veya bir seri görüntü verisini kullanarak yalnızca gerekli direksiyon açısını tahmin etmeye odaklanmaktadır. Daha üst seviye bir sürücüsüz sistem ileri yönlü hareketlerin kontrolünden sorumludur. Bu çalışmada [7] içerisinde önerilen çoklu görevlendirmeli model kullanılarak yatay ve ileri yönlü kontrol girdilerinin üretilmesi hedeflenmektedir. Bunun yanında klasik gaz ve fren komutları yerine doğrudan bir hız kontrolcüsünden yararlanılmaktadır. Şekil IV.1 Genel Sistem

Bu çalışmada kullanılan verimli, seçmeli ve güvenli veri seti kümelemesi yöntemi Şekil IV.1'de görülmektedir. Görüldüğü üzere direksiyon komutları, aracın önünde bulunan kameralardan alınan ham görüntü girdilerinin evrişimsel katmanlardan geçmesi ile üretilmektedir. Hız ise hız profillerinin zaman serisinin LSTM katmanlarından geçmesi ile tahmin edilmektedir.

Daha önceki çalışma [7] içerisinde yapılana benzer olarak burada ileri yönlü (aracın hızı) ve yatay hareket arasında tek yönlü bir eşleşme yer almaktadır. Öyle ki aracın hızı, tahmin modelini büyük ölçüde etkilemektedir. Düşük hızlı bir virajda aracın sahip olduğu yatay dinamikler, yüksek hızda girilen bir viraj için farklılık göstermektedir. Dolayısıyla hızın yatay kontrole de etkisi bulunmaktadır.

Dahası planlanmış güzergahlar içerisinde düz yönlü güzergahların diğer tüm güzergahları bastırdığı gözlemlenmiştir, dolayısıyla öğrenilmiş sürüş politikası düz güzergahları seçme eğilimli olmaktadır.

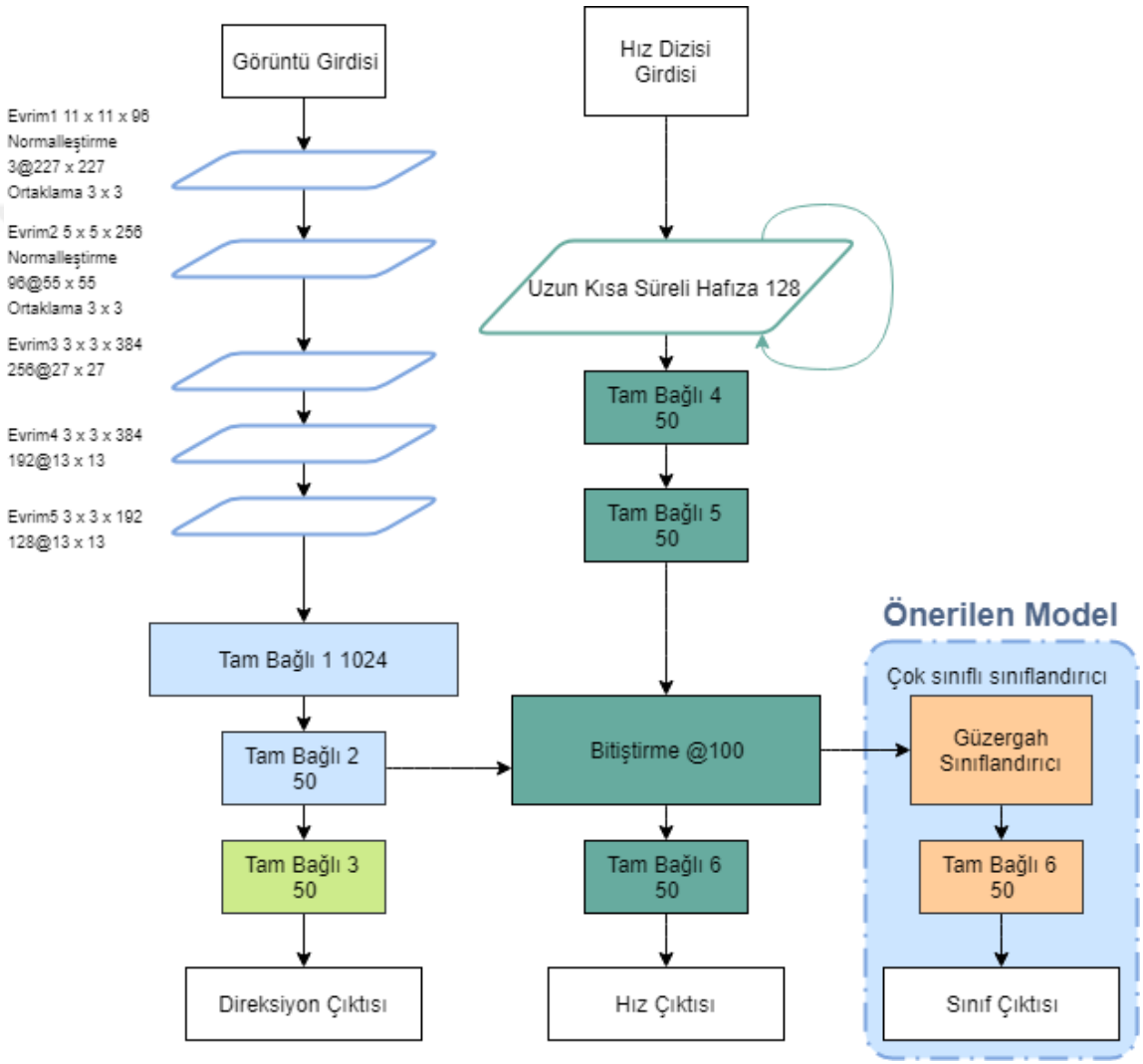
Bu nedenlerden ötürü sağa, sola dönüş ve düz gitmeyi içeren tüm ana manevraları ve farklı hız profillerini içeren 12 sınıf bu çalışma içerisinde tanımlanmıştır. Şekil IV.1 üzerinde vurgulanmış olan çok sınıflı sınıflandırıcı, güvenli veri seti kümelemesi (SafeDAgger) yöntemini genişleten özgün bir algoritma olarak bu çalışmanın ana katkısıdır.

Şekil IV.2'de gösterildiği üzere, tahmin modelinin güvenilirliği değerlendirildikten ikili bir sınıflandırma ile, güvenli ve güvensiz olarak, belirlendikten sonra güzergâh, Şekil V.12 gösterilen eğitim pistindeki tüm manevraları kapsayabilecek 6 ana sınıftan birine sınıflandırılmaktadır.

Çok sınıflı sınıflandırıcı hem s durumun kısmı bir gözlemini hem de π tahmin politikasını alarak güzergahın hangi bölümünün π^* referans politikasından sapabileceği bilgisini içeren bir etiket üretir. Eğitim modeli için etiketler "One Hot Encoding" adında değişkenlerin ikili olarak temsil edildiği bir yöntem ile üretilmektedir. Etiketler bir sıra işlem sonucunda üretilir. Bunlardan ilki tahmin politikasının ürettiği sonucun referans politikanın ürettiği sonuçtan L2-norm metriğine göre ne kadar uzak olduğuna bakarak güvenli olup olmadığının tespit edilmesidir. Bu L2-norm metriği şu şekildedir:

Denklem IV.1 L2-norm Metriği

$$c_{safe}(\pi, s) = \begin{cases} 0, ||\pi(s) - \pi^*(s)||^2 > \tau \\ 1, diğer \end{cases}$$



Şekil IV.1 Genel Sistem Modeli (farklı renkler farklı modülleri temsil etmektedir)

Burada bulunan τ değişkeni önceden tanımlanan ve gelişigüzel bir değişkendir. Bu çalışmada $\tau = 0.5$ olarak seçilmiştir, bu da $0.25°$ direksiyon açısına ve $1\ m/s$ araç hızına denk gelmektedir.

Güzergâh tiplerini etiketlemek için, düz ve virajlı yolları birbirinden ayırmak adına, bir eşik değeri tanımlanmıştır. Bu eşik değeri $\tau_{dönüş}: 0.25°$ olarak belirlenmiştir. Bunun yanında düz veya viraj güzergahında araç dinamikleri farklı olduğundan, $\tau_{hız,düz}: 13.75 m/s$ ve $\tau_{hız,dönüş}: 10 m/s$ olmak üzere iki farklı hız eşik değeri belirlenmiştir.

Algoritma IV-1 Verimli Seçmeli Güvenli Veri Seti Kümelemesi Algoritması, önerilen yöntemi detayları ile tarif etmektedir. Bu algoritmanın SafeDAgger algoritmasının örnekleme açısından verimli bir uyarlaması olduğu gösterilmektedir. Algoritma referans politikayı, algılanmış ham görüntüyü ve aracın kaydedilmiş hızını alır ve tahmin modelini başlangıç veri setine ve referans politikaya göre eğitir. Sonra referans politikayı kullanarak algoritma en yüksek hata ihtimaline sahip olabilecek güzergahları sınıflandırabilecek hale gelir.

Bu yaklaşım sayesinde, SafeDAgger algoritmasından farklı olarak, model hatalarında çevrimiçi olarak eğitim veri setine dönüş yapılmayacaktır, çünkü en yüksek hataya dair karar kümelenmiş veriye bağlı olarak alınacaktır. Baskın güvensiz sınıf seçildiği zaman algoritma bu verinin sınıfına bakacak ve önceki veri seti ile birleştirerek modeli daha dengeli bir veri ile eğitecektir. Bu süreç model bir denge noktasına

yakınsayıncaya kadar ya da maksimum tekrar sayısına ulaşılıncaya kadar devam edecektir.

Algoritma IV-1 Verimli Seçmeli Güvenli Veri Seti Kümelemesi Algoritması

Giriş: π^*, s

Çıkış: $\pi_N, c_{güvenlik\ ,N}$

1: Referans uzman π^* politikasından D_0 topla

2: Referans uzman π^* politikasından $D_{\text{güvenlik}}$ topla

3: $\pi_0 = \arg\min_{\pi} l_{destekli}\ (\pi, \pi^*, D_0)$

4: $c_{güvenlik\ ,0} = \arg\min c_{güvenlik}\ l_{güvenlik}\ (c_{güvenlik}\ , \pi_0, \pi^*, D_{güvenlik} \cup D_0)$

5: **for** i=1 do N

6: $c_s \leftarrow s$ için sınıf tipini belirle

7: $c_{tümGüvenliler} = sınıflandırıcı(\pi^*, \pi_i, s)$

8: $c_{güvenlik\ ,i} = \max(c_{tümGüvenliler})$

9: **if** $c_s == c_{güvenlik\ ,i}$ **then**

10: $s_{güvenli} \leftarrow [\,]$

11: **else**

12: $s_{güvenli} \leftarrow s$

13: **end**

14: D' topla

15: $D' \leftarrow \{s_{güvenli} \in D' | c_{güvenlik\ ,i-1}(\pi_{i-1}, s_{güvenli} = 0)\}$

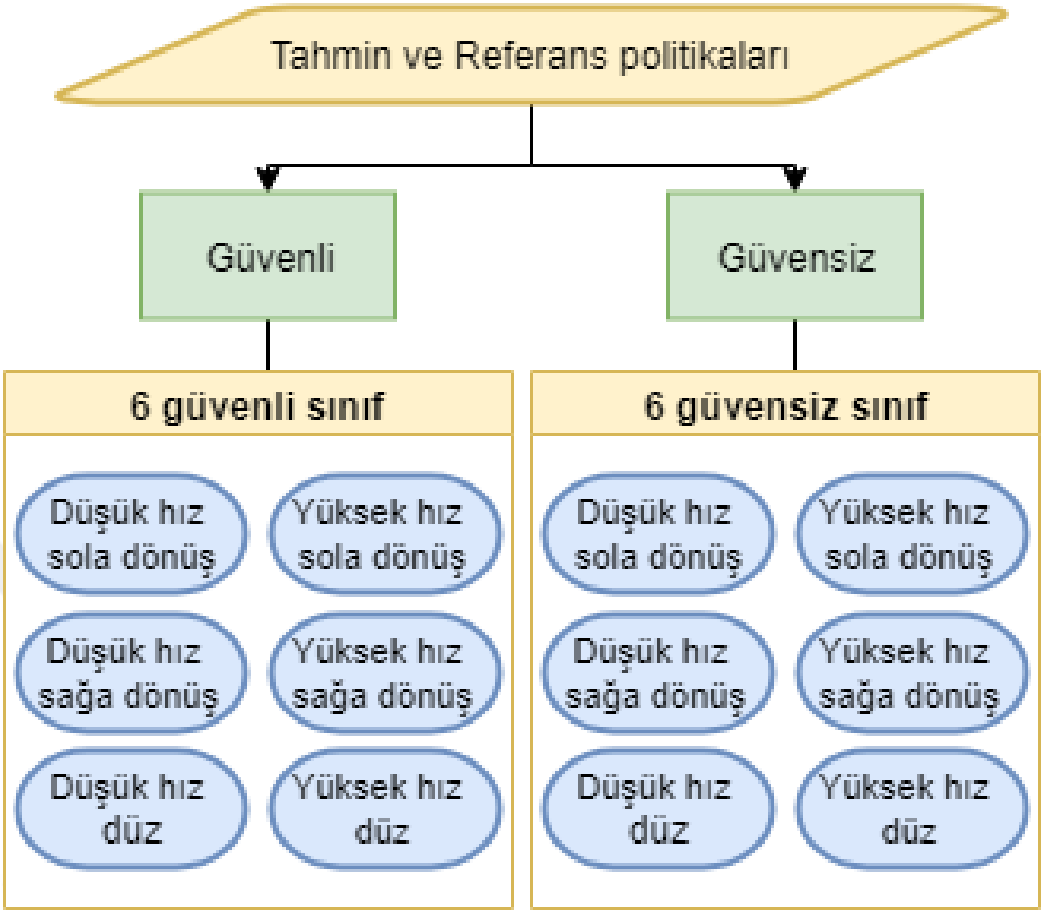
16: $D_i = D_i \cup D'$

17: $\pi_i = arg\min_{\pi} l_{destekli}\ (\pi, \pi^*, D_i)$

18: $c_{\text{güvenlik ,i}} = arg\min_{c_{güvenlik}} l_{güvenli}\ (c_{güvenlik}\ , \pi_i, \pi^*, D_{güvenli} \cup D_i)$

19: **end**

(Mavi yazı tipi önerilen algoritmanın, SafeDAgger algoritması ile farklılıklarını göstermektedir.)



Şekil IV.2 12-Sınıflı sınıflandırma süreci

V. GELİŞTİRME VE ÇIKTILAR

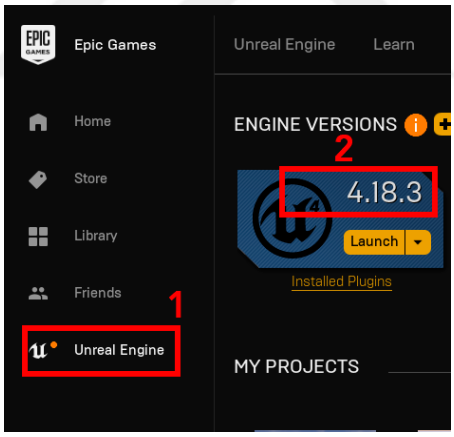
Bu bölümde önerilen yöntemin otonom sürüşe yönelik şekilde bir uygulamasında dair detaylara yer verilmektedir. Algoritmanın istenilen şekilde çalışıp sonuç verdiğini test edebilmek adına bir test ortamı oluşturulmuştur. Unreal Engine 4 oyun motoru kullanılarak oluşturulan bu test ortamının nasıl yaratıldığı ve kütüphanelerinin nasıl kurulduğu ve kullanıldığına dair ayrıntılar açıklanmaktadır. Derin öğrenme modülünün kurulumu, önerilen yöntem ile iletişim kurması için yapılanlar, önerilen yöntemin Python dili ile uygulamasına dair ayrıntılar ve simülasyon sonuçları yine bu bölümde yer almaktadır.

5.1. Simülasyon Sisteminin ve Derin Öğrenme Modülünün Kurulması

Airsim insansız pervaneli hava araçları ve arabalar için yapay zekâ çalışmalarında kullanılmak üzere, Microsoft tarafından sunulan bir Unreal Engine eklentisidir. Yeni derin öğrenme, bilgisayarla görme ve destekli öğrenme algoritmalarında geliştirme ve test için fotoğraf kalitesinde görselleştirme yapabilen bir altyapı sağlar. İçinde hali hazırda derlenmiş ve Python, C++, C# ve Java kodlama dillerinde yazılan derin öğrenme yöntemleri ile iletişim kuran API'ler bulundurmaktadır. Zaten derin öğrenme işleri için son yıllarda en baskın kodlama dili de Python kodlama dilidir.

5.1.1. Unreal Engine 4 Oyun Motorunun Kurulması

Unreal Engine 4, Epic Games tarafından sağlanan bir oyun motoru olduğu için Epic Games kullanıcıların motoru kurmasını ev kullanmasını kolaylaştırmak adına bir başlatıcı sağlamaktadır. https://www.unrealengine.com/en-US/download sitesinden otomatik olarak indirilebilen bu başlatıcıyı yönergeleri izleyerek indirme ve kurma işlemleri bittikten sonra geliştirme yapmak için bir üyelik oluşturulması ve üye girişi yapılması gerekmektedir. Yönergeleri izleyerek bu adımları da tamamladıktan sonra başlatıcı içerisinden Unreal Engine 4.18 versiyonunun kurulumu başlatılabilmektedir. Bu versiyonun seçilme sebebi Airsim eklentisinin desteklediği en yüksek versiyonun bu çalışma yapıldığı esnada bu sürüm olmasıdır.



Şekil V.1 Unreal Engine 4 kurulum adımları

5.1.2. Airsim Eklentisinin Kurulması

Airsim eklentisinin kurulması için bilgisayarda Visual Studio 2017 ile birlikte VC++ ve Windows SDK 8.1 eklentilerinin de bulunması gerekmektedir. Bu

yazılımların kurulmasının ardından Windows başlat menüsünden "`x64 Native Tools Command Prompt for VS 2017`" uygulaması başlatılır ve "git" servisinin depolarından gerekli dosyaları çekmek üzere "`git clone https://github.com/Microsoft/AirSim.git`" ve "`cd AirSim`" komutları çağırılır. Bu işlem sayesinden depolardan otomatik olarak gerekli kütüphane dosyaları indirilir. İndirme işlemi bittikten sonra "`build.cmd`" komutu ile kurulum otomatik olarak başlatılır ve eklenti "`Unreal\Plugins`" klasörü altına oluşturulur.

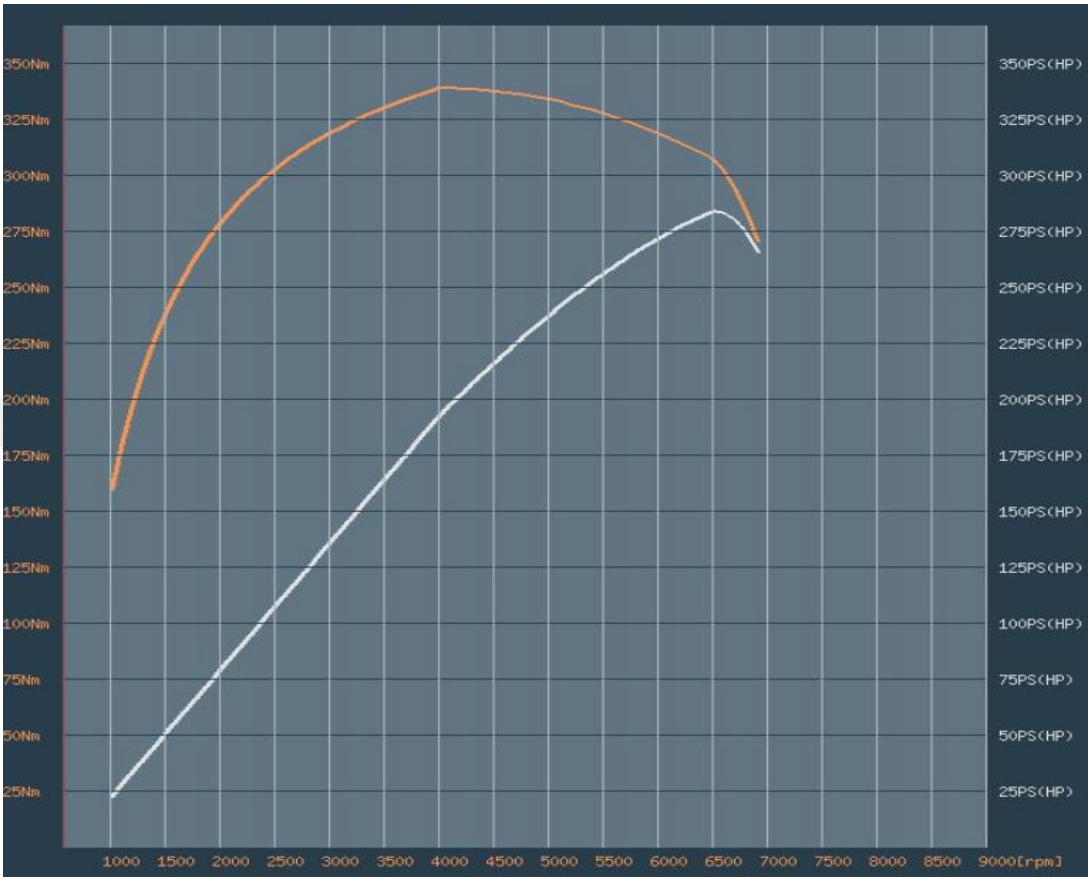
Kurulumu daha önceden yapılmış olan Unreal Engine 4 içerisinden "`Unreal/Environments/Blocks`" klasörüne otomatik olarak oluşturulmuş olan örnek projeyi başlatmak için "`AirSim\Unreal\Environments\Blocks`" konumuna gidilir ve "`update_from_git.bat`" dosyası çalıştırılır. Bu işlem otomatik olarak projeyi en son sürümüne günceller. Oluşturulmuş olan *.sln uzantılı dosya VS2017 yardımı ile açılır. "`Blocks`" isimli projenin başlangıç projesi olarak seçildiğine dikkat edilerek, derleme ayarı "`DebugGame_Editor`" ve "`Win64`" olarak seçilir ve F5 butonuna basılarak derleme işlemi başlar. Derleme işleminin bitmesi ile birlikte oynat tuşuna basılarak örnek proje başlatılır. Buradan sonra örnek proje içerisinde bulunan elemanlar istenildiği gibi yeni projelere taşınabilir.

Zaman Damgası	Hız(km/sa)	Gaz Pedal Pozisyonu	Direksiyon pozisyonu	Fren Pedal Pozisyonu	Vites	Görüntü
1389492	52	1.000000	0.027214	0.000000	2	img_0.png
1389521	52	1.000000	0.027383	0.000000	2	img_1.png
1389568	52	1.000000	0.026734	0.000000	2	img_2.png
1389614	53	1.000000	0.027639	0.000000	2	img_3.png
1389676	53	1.000000	0.027859	0.000000	2	img_4.png
1389717	54	0.790118	0.028068	0.000000	2	img_5.png
1389763	54	0.470468	0.028494	0.000000	2	img_6.png
1389825	54	0.107794	0.028545	0.000000	2	img_7.png
1389867	55	0.000000	0.028810	0.000000	2	img_8.png
1389927	55	0.000000	0.028652	0.000000	N	img_9.png
1389968	55	0.000000	0.027862	0.000000	N	img_10.png
1390014	55	0.000000	0.027686	0.000000	N	img_11.png
1390071	54	0.460986	0.027524	0.000000	1	img_12.png
1390115	53	1.000000	0.027892	0.000000	1	img_13.png
1390171	52	1.000000	0.029903	0.000000	1	img_14.png
1390216	51	1.000000	0.031911	0.000000	1	img_15.png
1390277	50	1.000000	0.033744	0.000000	1	img_16.png
1390316	49	1.000000	0.034231	0.000000	1	img_17.png
1390375	48	1.000000	0.034767	0.000000	1	img_18.png
1390415	47	1.000000	0.034749	0.000000	1	img_19.png
1390475	46	1.000000	0.034039	0.000000	1	img_20.png
1390516	45	1.000000	0.032823	0.000000	1	img_21.png
1390578	44	1.000000	0.031169	0.000000	1	img_22.png
1390620	44	1.000000	0.029960	0.000000	1	img_23.png
1390663	43	1.000000	0.028719	0.000000	1	img_24.png
1390718	43	1.000000	0.027199	0.000000	1	img_25.png
1390777	42	1.000000	0.025971	0.000000	1	img_26.png
1390817	42	1.000000	0.025460	0.000000	1	img_27.png
1390875	41	1.000000	0.024673	0.000000	1	img_28.png

Şekil V.2 Airsim örnek kayıt çıktısı

Airsim içerisinde sağlanan API'ler yardımı ile görüntüler, araca dair araç durumu, hız ve kontrol öğeleri gibi bilgiler doğrudan ya da kayıt yöntemi ile istenilen derin öğrenme ya da kontrol algoritmasına iletilebilir. Airsim kurulumun yapılması sonucunda simülatör başlatıldığında bir kayıt butonu ekranda yer almaktadır. Bu kayıt butonu sayesinde daha önceden tanımlanmış gerekli veriler eğitim verilerinin toplandığı oturum boyunca kaydedilir. Bu kayıtlar örnek olarak Şekil V.2 şeklinde olmaktadır. Burada bahsi geçen görüntü kolonu ise her bir zaman damgasına ait ileri yönlü bakan kameraya ait resimlerin isimlerini göstermektedir.

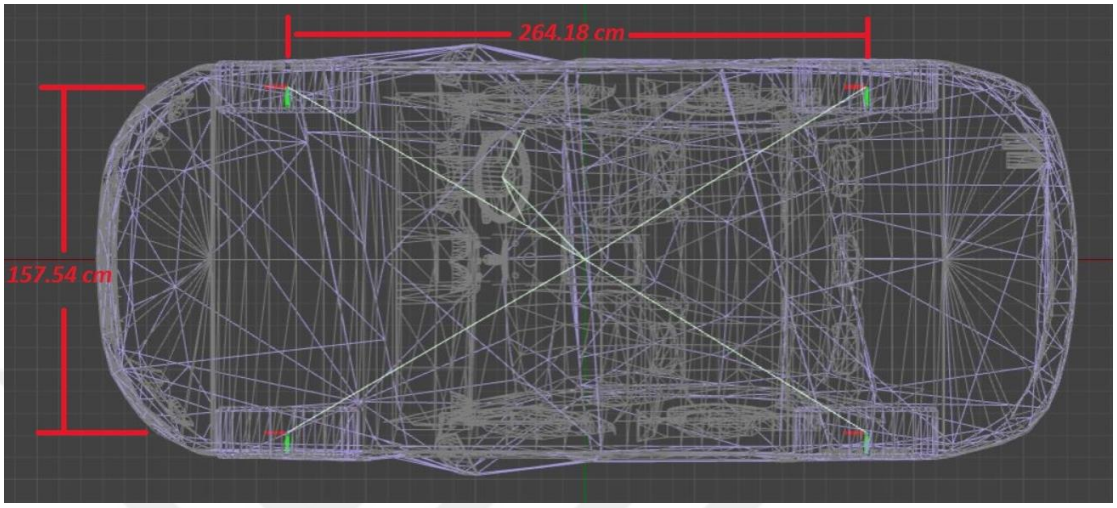
5.1.3. Araç Modeli



Şekil V.3 Araç modeli için tanımlanan tork ve beygir gücü eğrileri

Bölüm başında da bahsedildiği üzere simülatör içerisinde kullanılacak olan araç modeli 4 tekerlekli, önden direksiyonlu ve 4 tekerden çekiş sistemine sahip olarak seçilmiştir. Gerçekçi bir SUV araç modeli oluşturmak üzere araç boyutları Şekil V.4'de görüldüğü şekilde ve ağırlığı 1800kg seçilmiştir. Çekiş dinamikleri ise 4 tekerleğe eşit şekilde 6500 devirde 280 beygir güç ve 4000 devirde 340 Nm tork değeri dağıtılmıştır (Şekil V.3). Diferansiyel tipi olarak kontrol kolaylığından ötürü sınırlı kaymalı 4 teker

diferansiyel seçilmiştir. Tekerleklerin direksiyon açısı kalibrasyonu Şekil V.5'de görüldüğü şekilde bir eğride tanımlanmıştır.



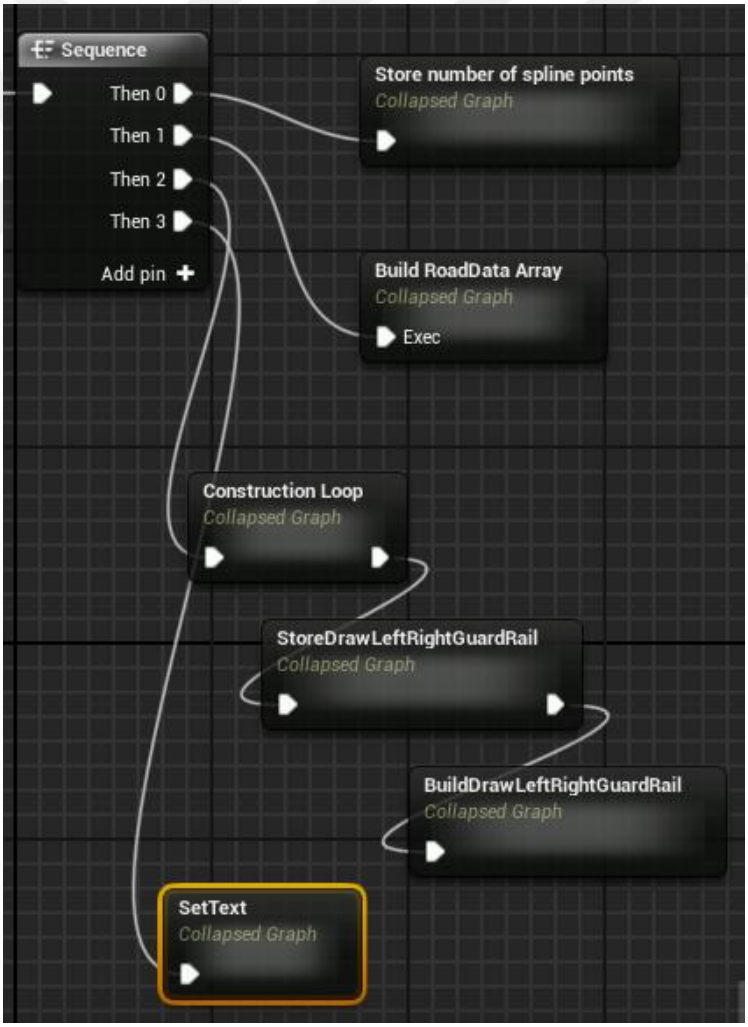
Şekil V.4 Araç boyutları

Şekil V.5 Direksiyon açısı kalibrasyonu

5.1.4. Pist tasarım aracı

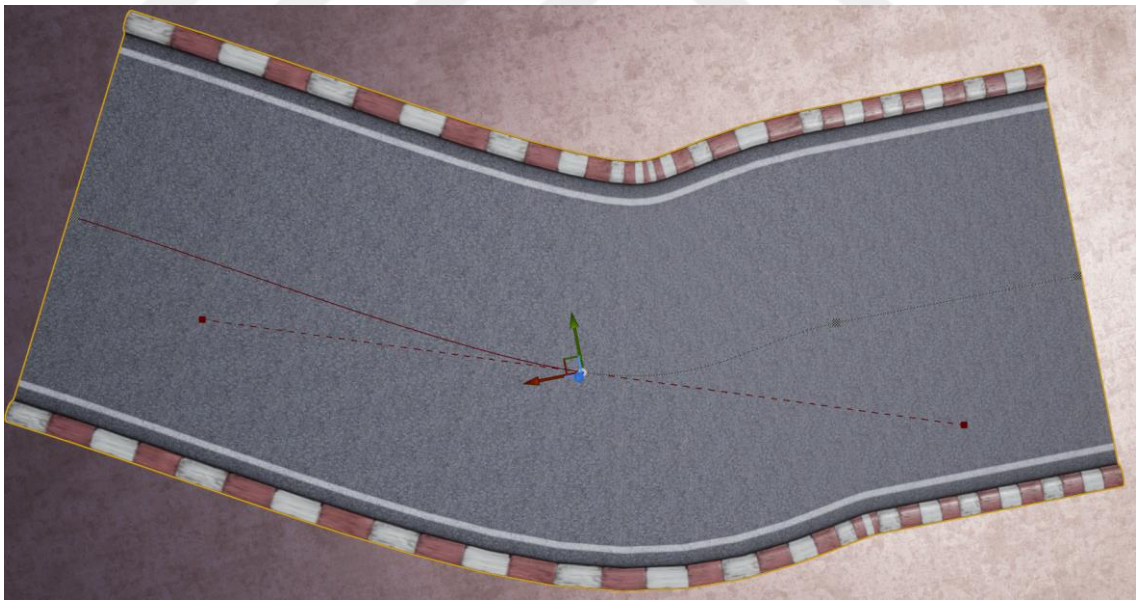
Derin öğrenme algoritmalarının denenebilmesi için simülatör ortamı içerisinde araç kameralarına gerçekçi görüntüler sunabilecek bir ortamın hazırlanması gerekmektedir. Unreal Engine 4 kaplamalar ve ışıklandırmalar gibi birçok kütüphane ve ayara sahip olsa da pist tasarımı ve ortam görünümünün elle yapılması gerekmektedir.

Oyun motoru içerisine 3 boyutlu tasarım programlarınca üretilmiş durağan kalıp dosyaları ve kaplamaları eklemek mümkündür. Pist tasarımlarını kolaylaştırmak ve elle birleştirme yükünden kurtulmak üzere otomatik olarak dikdörtgen bölümleri art arda bir eğri üzerine yerleştirebilen bir mavi kopya modeli oluşturulmuştur. Mavi kopya temelde sürükle bırak şeklinde kodlanan C++ tabanlı, Unreal Engine 4 motoruna özel bir kodlama şekli olduğundan grafikler üzerinde satır sayısı olarak kodlamayı görmek mümkün değildir. Ancak bu modelden oluşturulan dosya içeriği incelendiğinde yaklaşık 450 satır kodlama içerdiği görülmektedir.



Şekil V.6 Pist tasarım aracı programlaması

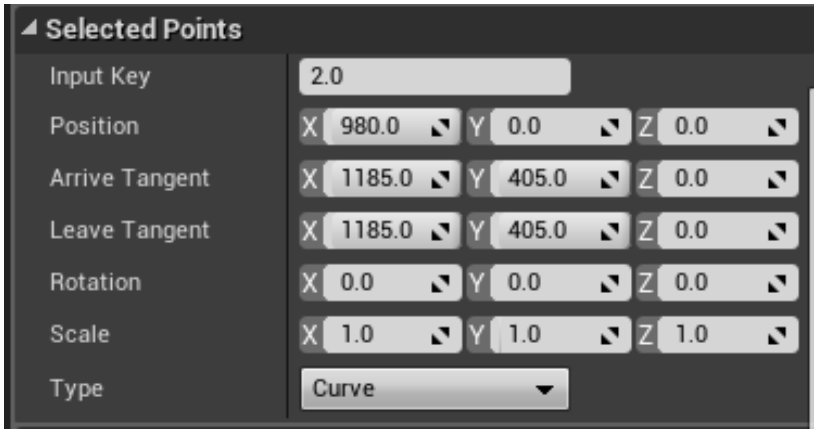
Bu model bir obje şeklinde geliştirme ortamına eklendiği zaman içerisinde kullanıcı tarafından sürüklenebilen 2 adet tutma noktası oluşturur. Bu noktalardan biri tutulup sürüklendiği zaman iki nokta arasında kalan eğri üzerine pist parçaları otomatik olarak dizilir. Model içerisinde bulunan bu tutma noktaları klavye üzerindeki “Alt” tuşuna basılarak sürüklendiği zaman ise tutulan nokta yerinde kalır iken sürüklenilen yerde yeni bir tutma noktası oluşur. Bu sayede bu noktalardan geçen sürekli eğriler oluşur. Bu eğriler de yine aynı şekilde pist parçaları ile doldurularak pist otomatik olarak oluşturulmuş olur. Kullanıcının yapması gereken tek şey istenilen şekilde noktaları dizerek daha önceden belirlenmiş pistin hatlarını tanımlamak olacaktır.



Şekil V.7 Pist tasarım aracı kullanım ekranı

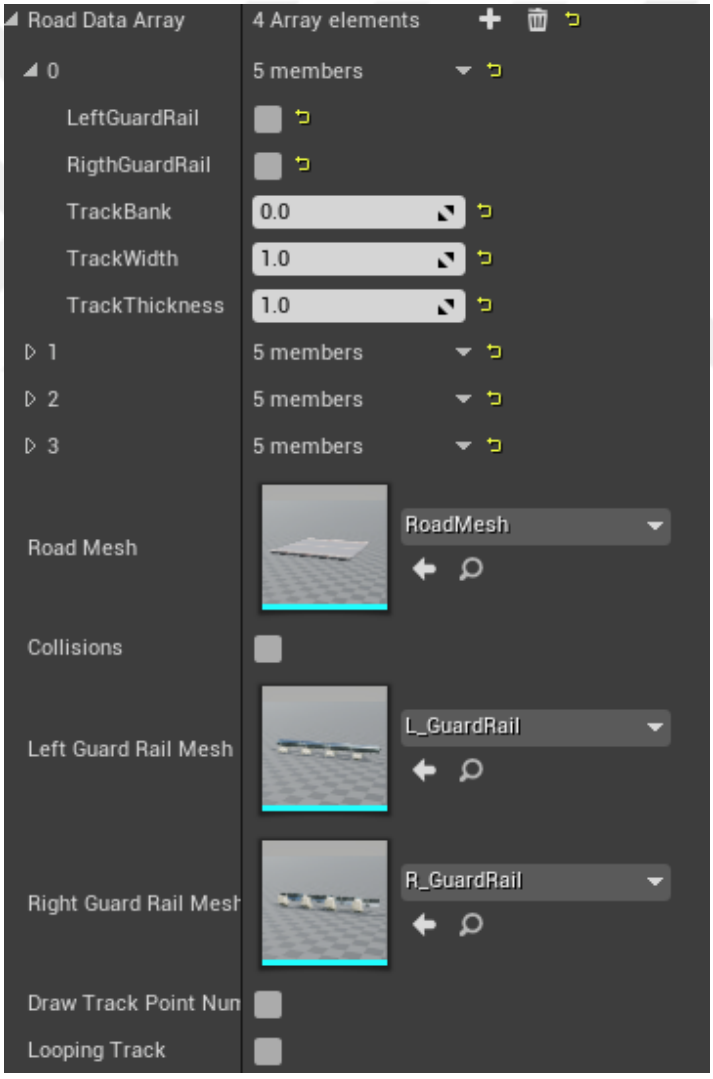
Bu tutma noktalarından geçen eğriyi kullanıcı istediği her şekilde biçimlendirebilmektedir. Şekil V.8 üzerinde görüldüğü gibi, yaratılan her bir noktaya ait bir ayar menüsü bulunmaktadır. Bu menü sayesinde her bir noktanın uzaydaki, yükseklik dahil olmak üzere, tam konumunu, yönünü ve noktaya yaklaşan ya da noktadan çıkan tanjant çizgilerinin yaklaşma uzaklaşma açılarını tek tek ayarlamak mümkündür. Bu ayarlamalar bu çalışma içerisinde elle yapılmış olsa da her bir noktanın bilgisini bir harita üzerinden alıp otomatik olarak tanımlamak da zor değildir. Ancak bu çalışma içerisinde 4 adet harita tanımlanacağından otomasyon algoritmasının yazılması için kaynak harcanmamıştır.

Pist için eğri tasarımının yapılmasının ardından Şekil V.9 üzerinde görüldüğü gibi yolun kendisine ve sağ ve sol kenarlarına ait kalıp tasarımları bir menü yardımı ile seçilebilmektedir. Bu menü üzerinde görülen “Road data array” bölümünden yolun noktalar arasında kalan her bir bölümü için ayar yapılabilmektedir. Her bir bölüm için sağ ve sol bariyerler açılabilir ya da kapatılabilir, bölüm genişlikleri, kalınlıkları ve burulma miktarı belirlenebilir.



Şekil V.8 Pist noktaları ayarlama arayüzü

Son olarak pist tasarım aracı içerisine bir boole değişkeni eklenmiştir. Bu değişken pistin ucu açık mı yoksa kendini tamamlayan bir yapısı mı olduğunu tanımlamaktadır. Bu değişkenin doğru olarak atanması durumunda pist eğirişi üzerinde yaratılan en son nokta yaratılan ilk noktaya verilen yaklaşma ve uzaklaşma açıları dahilinde bir eğri ile bağlanır. Bu sayede eğrinin iki ucunu elle bağlamak için emek harcamaya gerek kalmaz, ki yaklaşma ve uzaklaşma açılarını elle denk getirmeye çalışmak zahmetli bir iştir.

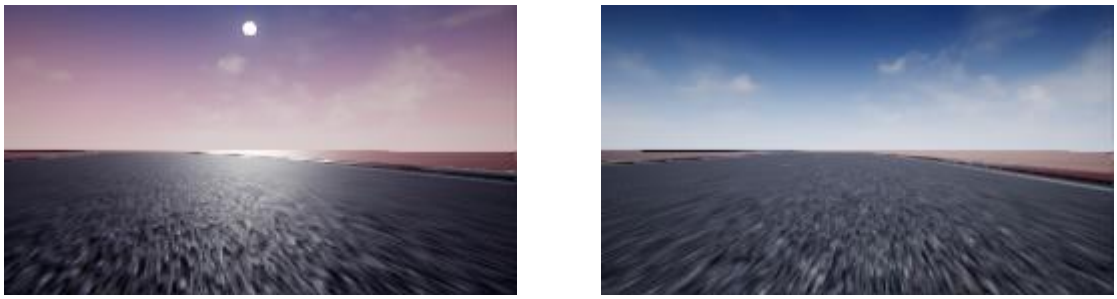


Şekil V.9 Pist parçalarının özelliklerini ayarlamak için arayüz

5.1.5. Çevre Tasarımı

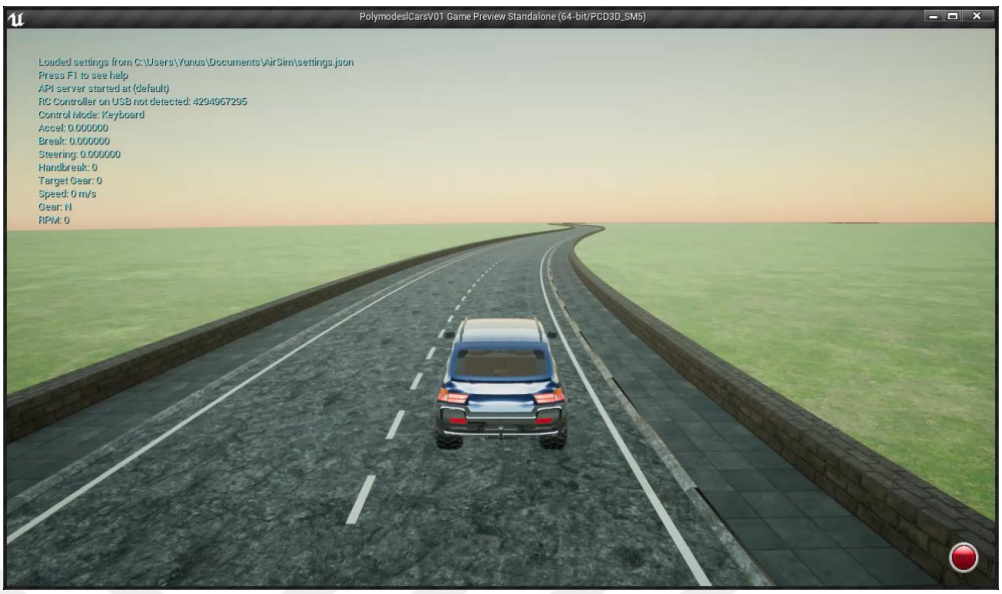
Unreal Engine oyun motoru içerisinde sunulan tasarım altyapısı sayesinde isteğe özel olarak çevre tasarımı ve senaryo tanımlamaları yapılabilmektedir. Simülatör kendi içinde fotoğraf kalitesinde çevre tasarımları yapmaya izin verdiğinden, gerçek hayata da uyarlanması planlanan derin öğrenme işleri için uygun bir platformdur.

Önerilen derin öğrenme algoritmasının öğrenme işlerinde kullanılmak üzere Şekil V.12'de haritası görünen pist tasarımı, tasarım bölümünde anlatılan pist tasarım aracı kullanılarak hazırlanmıştır. Önerilen derin öğrenme algoritması içerisinde bulunan 6 farklı güzergâh tipi de bu harita içerisinde yer almaktadır. Bunların birer örneği yine harita üzerinde belirtilmiştir. Öğrenme süreci içerisinde ilerledikçe bu 6 güzergâh önce güvensiz, eğitimde ilerledikçe de güvenli olarak etiketleneceğinden önceki bölümlerde anlatılmış olan 12 sınıfın tamamına bu harita içerisinde yer verilmiştir.



(a) Eğik açılı güneş ışığı (b) Dik açılı güneş ışığı

Şekil V.10 Güneş ışığı açıları

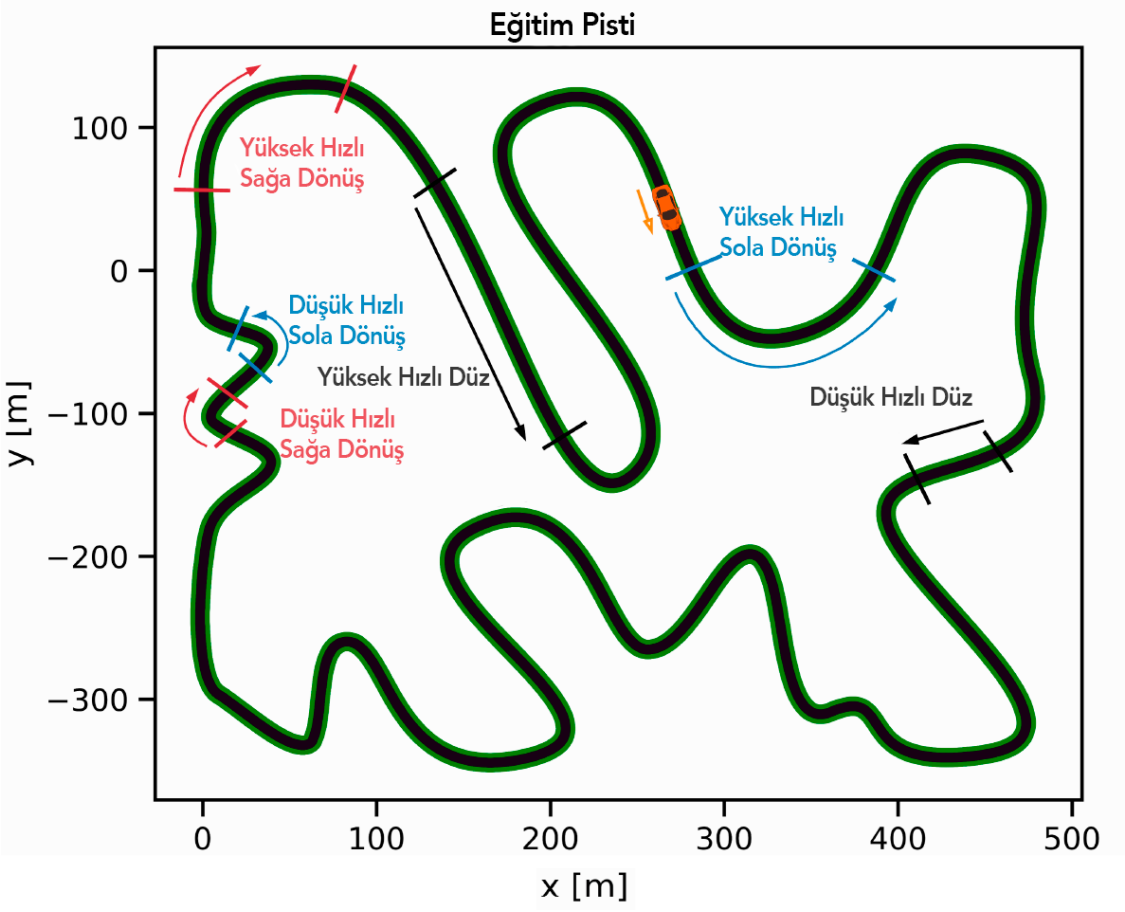


Şekil V.11 Yol tasarımı

Şekil V.11'de görüldüğü üzere yol tasarımı gerçek hayata benzer şekilde özelliklere sahiptir. Ortada yolu iki ayrı şeride ayıran kesikli şerit çizgileri yer almaktadır. Yol kenarlarında içten dışa doğru sırasıyla düz şerit çizgileri, kaldırım yükseltisi, kaldırım alanı ve alçak bir duvar bulunmaktadır. Ayrıca kaplama detaylarının da gerçek asfalt, kaldırım taşı ve tuğla görüntüleri ile benzemesine dikkat edilmiştir.

Gerçek hayat senaryoları ile benzerlik sağlamak adına öğleden sonra güneşini taklit edecek şekilde bir güneş ışığı modeli de tasarım içerisinde yer almaktadır. Bu ışıklandırma sayesine güneş tam tepeden değil de yatay bir açıyla gelerek, araç üzerinde bulunan kameraları da tek düze olmayan bir ışık ile zorlamaktadır. Bu da yine simülatör ortamını gerçek hayatta karşılaşılacak senaryolara yaklaştırmaktadır (Şekil V.10)

Ayrıca Airsim içerisinde yer alan bir hava durumu kontrol mekanizması sayesinde simülatör ortamı içinde güneşli, karlı, yağmurlu, bulutlu bir hava yaratılabilmekte. Dahası bu hava durumunu araç kamerası için daha gerçekçi bir hale getirmek adına kameranın gördüğü yoldaki kaplamalar da değiştirilerek yolların ıslak, karlı tozlu ya da sonbahar yaprakları ile bozulmuş şekilde gerçekçi olarak oluşturulması da sağlanabilmektedir. Çalışmanın gelecekte devam etmesi halinde bu durumların da senaryolar içerisinde eklenmesi planlanmaktadır.

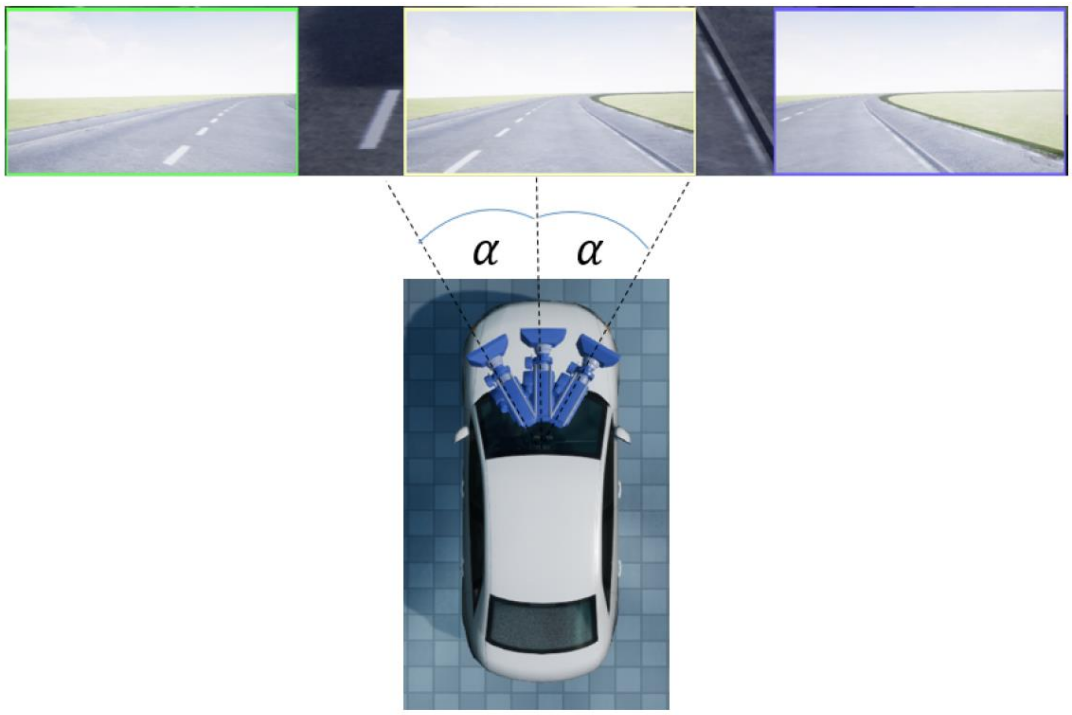


Şekil V.12 Eğitim Pisti

Daha önce gözlemlenmemiş açılardan da veri toplamak, eğitim setinin gücünü arttırmakta kullanılabilir düşüncesi ile araç üzerinde ileri yönlü bakan kameranın yanına, Şekil V.13'de görüldüğü gibi, α açısı ile döndürülmüş 2 adet daha kamera eklenmiştir. Airsim API'si öne bakan kamera için kesin referans etiketi üretmektedir, ancak diğer 2 kamera için kesin referans etiketindin α açısı ve eşik değerleriyle ayarlanması gerekmektedir. Bu ayarlama şu şekilde yapılmıştır:

Denklem V.1 Kamera Açılarının Sapmaları

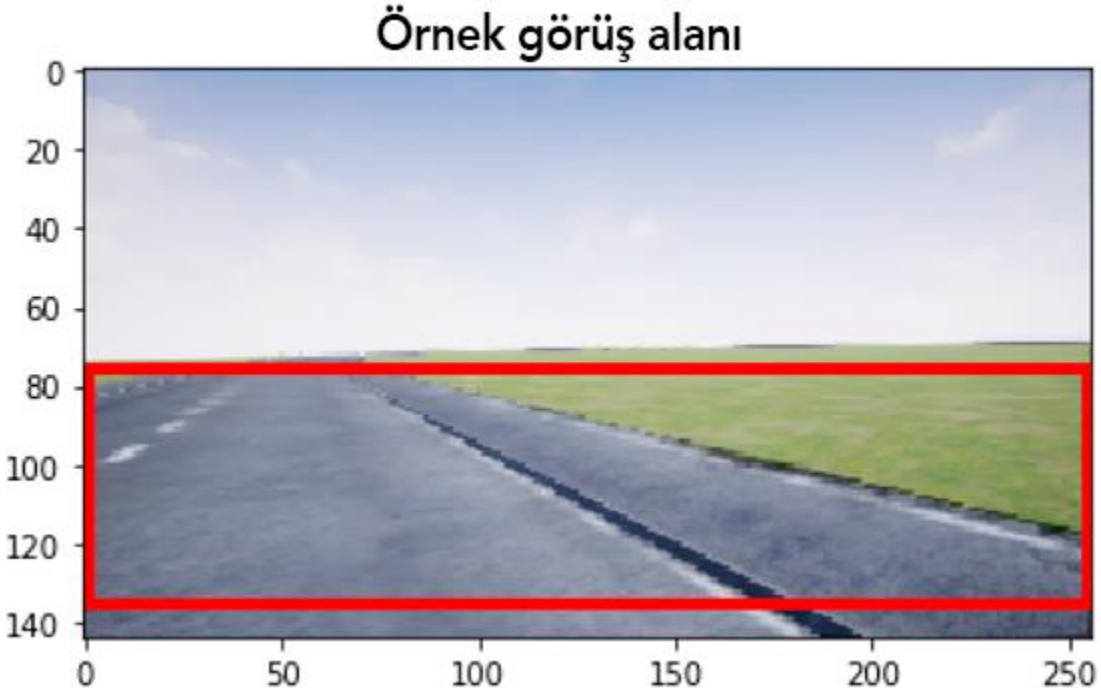
$$\begin{bmatrix} L_{sol} \\ L_{sağ} \end{bmatrix} = \begin{bmatrix} L_{merke\ z_{direksiyon}} & + \alpha L_{merke\ z_{hız}} - k_{hız} \\ L_{merke\ z_{direksiyon}} & - \alpha L_{merke\ z_{hız}} - k_{hız} \end{bmatrix}$$



Şekil V.13 α açısı ile yerleştirilmiş 3 kamera görüşü

5.1.6. Veri Ön İşlemesi

Araç üzerindeki kameralardan toplanan eğitim verilerinin her biri karşılık gelen hız ve viraj aksiyonuna göre etiketlenmelidir. Alınan her bir görüntü, algoritmanın işlem yükünü azaltmak adına, 144x256x3(RGB) çözünürlüğüne düşürülmüştür. Yapay sinir ağının gereksiz detaylara takılmasını engellemek adına görüntü üzerinde bir ilgi bölgesi tanımlanmıştır. Bu ilgi bölgesi Şekil V.14'de görüldüğü gibi, görüntünün alt tarafına yakın 59x255 piksel büyüklüğünde seçilmiştir. Her bir görüntünün parlaklığı rastgele bir oranda RGB uzayından HSV uzayına oradan da geri RGB uzayına dönüştürülerek değiştirilmiştir. Ayrıca görsellere rastgele şekilde ufak döndürmeler de uygulanmıştır, bu da gerçek hayatta kamerada oluşacak titreşimleri taklit etmektedir.



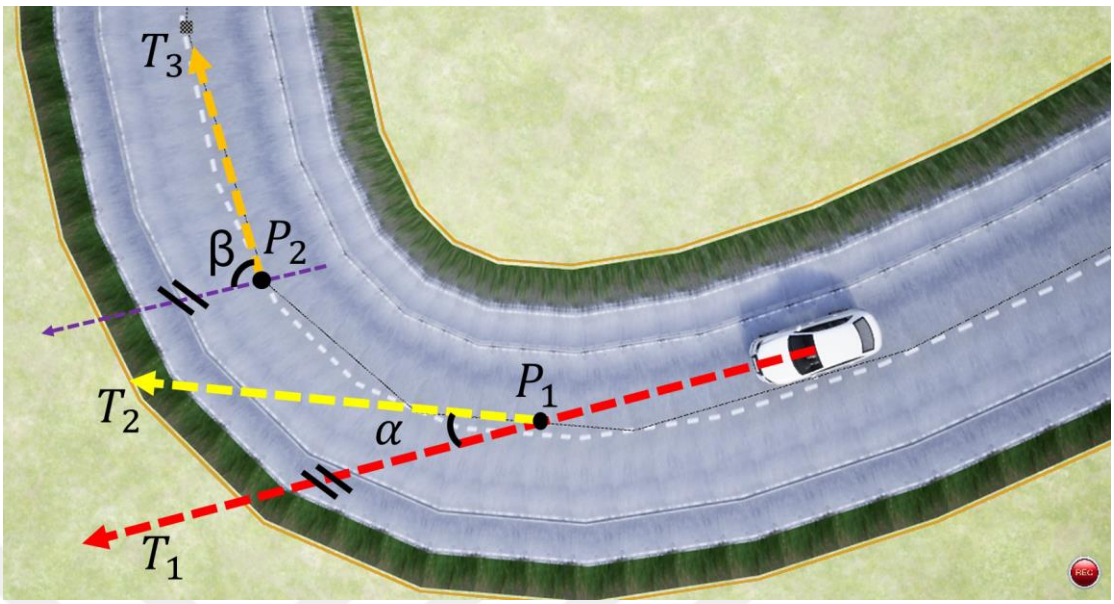
Şekil V.14 Görüş alanı

5.1.7. Uzman Araç Modeli

Veri toplama işini otomatikleştirmek adına kural tabanlı bir uzman politika tanımlanmıştır (bkz. Şekil V.15). Bu uzman politikanın davranış mantığı şu şekildedir:

Aracın şekil üzerinde şu an olduğu noktadan çıkan araç güzergahına tanjant çizgi T_1 ve aracın olduğu noktadan T_1 tanjantı boyunca l_{ref} kadar uzaklıktaki nokta P_1 olarak tanımlansın. P_1 noktasından yine aynı şekilde uzanan güzergaha tanjant çizgi T_2 olarak tanımlandığında T_1 ve T_2 arasında kalan α açısı uzman politikanın direksiyon aksiyonudur.

Hız aksiyonu için ise benzer bir mantıkla, aracın şekil üzerindeki konumundan güzergâhı boyunca, $V_{anlık}$ anlık hız ve $k_{direksiyon}$ ince ayarla seçilmiş direksiyon katsayısı olmak üzere, $l_{ref}\ x\ V_{anlık}\ x\ k_{direksiyon}$ kadar uzaklıktaki nokta P_2 olsun. Güzergaha göre P_2 noktasından geçen tanjant T_3 olarak seçildiğinde, T_2 ve T_3 arasında kalan açı da β olsun. Uzmanın hız aksiyonu bu durumda, V_{seyir} araç için önceden atanmış sabit seyir hızı ve $k_{hız}$ ince ayarla seçilmiş hız katsayısı olmak üzere, $V_{seyir} - k_{hız}x\beta$ olacaktır. Bu çalışma içerisinde $l_{ref} = 1m, V_{seyir} = 13.8m/s, k_{direksiyon} = 5$ ve $k_{hız} = 10$ olarak deneysel olarak belirlenmiştir.



Şekil V.15 Uzman politika

5.2. Öncül Algoritmaların Uygulanması

Önerilen algoritmanın kendinden önce gelen ve temel aldığı SafeDAgger algoritmasına karşı performansının sınanması için öncelikle SafeDAgger algoritmasının öne sürüldüğü şekilde uygulanması ve bu uygulamanın öne sürüldüğü çalışmadaki [9] ile aynı sonuçları ürettiğinden emin olunması gerekmektedir. Bu amaçla SafeDAgger çalışmasında yapıldığı gibi önce DAgger çalışmasının [8] uygulanması gerekmiştir.

Bölüm 2.3 içerisinde ayrıntıları ile anlatılan DAgger algoritması Python dili ile kodlanmış ve Airsim kütüphanesi yardımı ile simülatör üzerinde önce eğitim sonrasında da test pistleri içerisinde çalıştırılarak L2-normları toplanmıştır. Bu işlemin ayrıntıları Bölüm 5.3 içerisinde önerilen algoritmanın uygulanması ile aynı şekilde

gerçekleştirilmiştir ve DAgger algoritmalarının uygulanması adına algoritmaya özel yaklaşık 400 satırlık bir kodlama yapılmıştır. Bu kodlamalardan DAgger algoritmasının ana döngüsünü oluşturan fonksiyon Kod Parçası V.1 içerisinde örnek olarak paylaşılmıştır.

```
def T_step_iteration(T_step,state_buf,iter_number):
  for i in range(0,T_step):
    client.simPause(False)
    car_state = client.getCarState()
    image_buf[3*i:3*i+3] = get_image_triple()
    state_buf = np.append(state_buf, car_state.speed)
    state_buf = state_buf[-20:]
    state[0,:,0]= state_buf
    prev_states[3*i]   = state_buf
    prev_states[3*i+1] = state_buf
    prev_states[3*i+2] = state_buf
    GT_Labels[3*i,:]= [car_state.GTSteering,car_state.GTSpeed]
    GT_Labels[3*i+1,:]= [car_state.GTSteering-0.6,car_state.GTSpeed]
    GT_Labels[3*i+2,:]= [car_state.GTSteering+0.6,car_state.GTSpeed]
    inputs = [image_buf[3*i:3*i+1,76:135,0:255]/255.0,state[0:1]]
    model_output = model.predict(inputs)
    SpeedSetPoint = (1-iter_number/10)
      *round(float(car_state.GTSpeed), 10)+iter_number/10
      *round(float(model_output[0][1]), 10)
    car_controls.steering = (1-iter_number/10)
      *round(float(car_state.GTSteering),10)+iter_number/10
      *round(float(model_output[0][0]), 10)
    SpeedControllerOutput = PID(SetPoint = SpeedSetPoint,
      feedback_value = car_state.speed)
    car_controls.throttle = SpeedControllerOutput
    client.setCarControls(car_controls)
```

```
    client.simPause(True)
  return prev_states,state_buf,image_buf, GT_Labels
```

Kod Parçası V.1 DAgger algoritması her örnek için ana döngüsü Python Kodu

Ardından Bölüm 2.4 içerisinde ayrıntılarıyla açıklanmış olan SafeDAgger algoritması da ayrı bir Python dosyasında uygulanmış ve önce eğitim sonra da test pistleri içerisinde çalıştırılarak L2-normları toplanmıştır. Toplanan bu L2-normlarının birbiri ile karşılaştırılması sonucunda orijinal SafeDAgger çalışmasında [9] olduğu gibi, SafeDAgger algoritması DAgger algoritmasına göre yaklaşık %7-10 daha başarılı olmuştur. Bu durum orijinal çalışma ile birebir benzerlik gösterdiğinden kurulan simülatör ortamının önerilen algoritma ile SafeDAgger algoritmasını L2-normları açısından karşılaştırmak için uygun olduğu sonucu ortaya çıkmaktadır. SafeDAgger algoritması için de yine yaklaşık 400 satırlık bir kodlama yapılmıştır. Bu kodlamalardan SafeDAgger algoritmasının ana döngüsünü oluşturan fonksiyon Kod Parçası V.2 içerisinde örnek olarak paylaşılmıştır.

```
def T_step_iteration(T_step,state_buf,iter_number):
  for i in range(0,T_step):
    Safety = True
    while Safety:
      client.simPause(False)
      car_state = client.getCarState()
      image_buf[3*i:3*i+3] = get_image_triple()
      state_buf = np.append(state_buf, car_state.speed)
      state_buf = state_buf[-20:]
      state[0,:,0]= state_buf
      inputs = [image_buf[3*i:3*i+1,76:135,0:255]/255.0, state[0:1]]
```

```
model_output = model.predict(inputs)
GTSTeer = round(float(car_state.GTSteering), 10)
PrSTeer = round(float(model_output[0][0]), 10)
if (GTSTeer - PrSTeer) > TauSteer:
  Safety = False
  SpeedSetPoint = round(float(car_state.GTSpeed), 10)
  SteerSetPoint = round(float(car_state.GTSteering), 10)
  SpeedControllerOutput = PID(SetPoint = SpeedSetPoint,
    feedback_value = car_state.speed)
  car_controls.steering = SteerSetPoint
  car_controls.throttle = SpeedControllerOutput
  client.setCarControls(car_controls)
  client.simPause(True)
else:
  Safety = True
  SpeedSetPoint = (1-
    iter_number/10)*round(float(car_state.GTSpeed),10)+
    iter_number/10*round(float(model_output[0][1]),10)
  SteerSetPoint = (1-iter_number/10)
    *round(float(car_state.GTSteering),10)
    +iter_number/10*round(float(model_output[0][0]), 10)
  SpeedControllerOutput = PID(SetPoint = SpeedSetPoint,
    feedback_value = car_state.speed)
  car_controls.steering = SteerSetPoint
  car_controls.throttle = SpeedControllerOutput
  client.setCarControls(car_controls)
  GT_Steering = {2}, GT_Speed =
    {3}'.format(round(float(model_output[0][0]), 6),
    round(float(model_output[0][1]), 6),
    round(float(car_state.GTSteering), 6),
    round(float(car_state.GTSpeed), 6)))
  client.simPause(True)
prev_states[3*i]   = state_buf
```

```
      prev_states[3*i+1] = state_buf
      prev_states[3*i+2] = state_buf
      GT_Labels[3*i,:]= [car_state.GTSteering
        ,car_state.GTSpeed]
      GT_Labels[3*i+1,:]= [car_state.GTSteering-0.6
        ,car_state.GTSpeed]
      GT_Labels[3*i+2,:]= [car_state.GTSteering+0.6
        ,car_state.GTSpeed]
    return prev_states,state_buf,image_buf, GT_Labels
```

Kod Parçası V.2 SafeDAgger algoritması her örnek için ana döngüsü Python Kodu

5.3. Derin Öğrenme Modülünün Eğitilmesi

Birincil politika π_0 ve sınıflandırıcı $c_{güvenlik,0}$ için uzman π^* politikasından 2800 görüntüden oluşan bir veri seti eğitimde kullanılmak üzere toplanmıştır. Verinin iyileştirilmesi sürecinde Nesterov Adam iyileştiricisi (Nadam), ilk öğrenme oranı 10^{-5} ve 0.99 moment değerleri ile kullanılmıştır. Eğitim 10 epoch boyunca 32'li kümeler halinde devam etmiştir.

Eğitilmiş birincil politika π_0 ve $c_{safe,0}$ tüm önceden toplanmış örnekleri güzergahlarını sınıflandırmak ve her bir örneğin L2-normunu hesaplamak için de kullanılmıştır. Tüm örneklerin L2-normu hesaplandıktan sonra ortalama L2-normu $ortalama(\mu)$ standart sapma σ hesaplanmıştır. $\mu + \sigma$ L2-normundan yüksek değerli örnekler güzergâh sınıflarına göre filtrelenmiştir. Filtrelenmiş örneklere ait birer zayıflık katsayısı şu şekilde hesaplanmıştır:

Denklem V.2 Sınıfın Zayıflık Katsayısı

$$sınıf_{zayıf\ ,i} = \frac{N_{L2>(\mu+\sigma)}}{N_{tümü}}\ x\ ortalama_{L2,i}$$

Bu denklemde bulunan $N_{L2>(\mu+\sigma)}$ ortalama L2-normundan 1σ kadar uzaklaşan örneklerin sayısını, $N_{tümü}$ bu sınıftaki örneklerin tümünün sayısını, $ortalama_{L2,i}$ sayısı ise bu sınıftaki tüm L2-normlarının ortalamasını ifade etmektedir. Her bir güzergâh zayıflık katsayılarına göre dizilir ve en baskın 2 güvensiz sınıf veri seti kümelemesi sınıfı olarak belirlenir. Bunların dışında zayıflık katsayısı birden küçük olan tüm örnekler izin verilebilir olarak seçilir.

Veri seti kümelemesi safhasında π_0 politikası aracı baskın sınıflarda 10'lu kümeler halinde veri toplamak üzere sürer. Eğer politika sürüş sırasında daha önceden güzergâh sınıflandırıcısı tarafından işaretlenmiş baskın bir sınıfa denk gelirse, uzman politika aracın sürüşünü devralır ve kümeleme için ayrılmış süre içerisinde bu güzergaha ait örnek toplar. Eğer politika sürüş sırasında aslında güvensiz olan ancak zayıflık katsayısı sayesinde izin verilebilir bir sınıfa denk gelirse aracı sürmeye devam eder. Diğer bütün güvensiz sınıflar için uzman politika sürüşü, 10'lu küme sorgulaması ile sınırlandırılmış şekilde devralır. Eğer sorgulama limitine ulaşılmışsa veya 10 küme veri baskın sınıflar ise kümeleme işlemi durur ve eğitim süreci başlar. Bu eğitim ve eğitim verisi kullanılarak yapılan sürüş algoritmalarının uygulanması adına ayrı bir Python dosyası içerisinde yaklaşık 600 satırlık bir kodlama yapılmıştır. Bu algoritmanın öncül algoritmalardan farklı bazı yönlerini göstermek adına her bir örnek

için her tekrarda kullanılan ana döngü fonksiyonu Kod Parçası V.3 içerisinde, sınıflandırıcı fonksiyon ise Kod Parçası V.4 içerisinde paylaşılmıştır.

```
def T_step_iteration(T_step,state_buf,iter_number,class_type,Allow):
        Counter0 = 0
        Counter1 = 0
        Counter2 = 0
        idx_Unsafe_in_choosen_class = []
        idx_Unsafe_in_other_class = []
        image_buf = np.zeros((T_step*3, 144, 256, 3))
        GT_Labels = np.zeros((T_step*3,2))
        prev_states= np.zeros((T_step*3,20))
        Total_class = np.zeros(12)

        for i in range(0,T_step):
            Safety = True

            while Safety:
                client.simPause(False)
                car_state = client.getCarState()
                image_buf[3*i:3*i+3] = get_image_triple()
                state_buf = np.append(state_buf, (car_state.speed-
7)/8)
                state_buf = state_buf[-20:]
                state[0,:,0]= state_buf
                inputs = [image_buf[3*i:3*i+1,76:135,0:255]/255.0,
state[0:1]]
                model_output = model.predict(inputs)

                GTSTeer_and_GTSpeed =
np.array([round(float(car_state.GTSteering), 10),
round(float(car_state.GTSpeed), 10)])
                PrSTeer_and_PrSpeed =
np.array([round(float(model_output[0][0]),
10),round(float(model_output[0][1]*8+7), 10)])
                binaryLabel = Classifier(PrSTeer_and_PrSpeed,
GTSTeer_and_GTSpeed)
                Total_class[binaryLabel] += 1

                if (binaryLabel == class_type[0]):
                    Take_Data_from_that_Class = True
                    Counter0 = Counter0+1
                elif binaryLabel == class_type[1]:
                    Take_Data_from_that_Class = True
                    Counter1 = Counter1+1
                else:
                    Take_Data_from_that_Class = False

                if Take_Data_from_that_Class:

                    Safety = False
                    idx_Unsafe_in_choosen_class.append(3*i)
```

```
                    idx_Unsafe_in_choosen_class.append(3*i+1)
                    idx_Unsafe_in_choosen_class.append(3*i+2)
                            SpeedSetPoint                                =
round(float(car_state.GTSpeed), 10)
                    SteerSetPoint                                        =
round(float(car_state.GTSteering), 10)
                    SpeedControllerOutput     =     PID(SetPoint     =
SpeedSetPoint, feedback_value = car_state.speed)
                    car_controls.steering = SteerSetPoint
                    car_controls.throttle = SpeedControllerOutput
                    client.setCarControls(car_controls)
                    client.simPause(True)
                elif       np.linalg.norm(GTSTeer_and_GTSpeed       -
PrSTeer_and_PrSpeed) > Tau:
                    if binaryLabel in Allow:
                        Safety = True
                                SpeedSetPoint                            =
round(float(model_output[0][1]*8+7), 10)
                        SteerSetPoint                                    =
round(float(model_output[0][0]), 10)
                        SpeedControllerOutput    =    PID(SetPoint    =
SpeedSetPoint, feedback_value = car_state.speed)
                        car_controls.steering = SteerSetPoint
                        car_controls.throttle = SpeedControllerOutput
                        client.setCarControls(car_controls)
                        client.simPause(True)
                    else:
                        Safety = False
                        idx_Unsafe_in_other_class.append(3*i)
                        idx_Unsafe_in_other_class.append(3*i+1)
                        idx_Unsafe_in_other_class.append(3*i+2)
                        SpeedSetPoint                                    =
round(float(car_state.GTSpeed), 10)
                        SteerSetPoint                                    =
round(float(car_state.GTSteering), 10)
                        SpeedControllerOutput    =    PID(SetPoint    =
SpeedSetPoint, feedback_value = car_state.speed)
                        car_controls.steering = SteerSetPoint
                        car_controls.throttle = SpeedControllerOutput
                        client.setCarControls(car_controls)
                        client.simPause(True)
                else:
                    Safety = True
                    SpeedSetPoint                                        =
round(float(model_output[0][1]*8+7), 10)
                    SteerSetPoint  =  round(float(model_output[0][0]),
10)
                    SpeedControllerOutput     =     PID(SetPoint     =
SpeedSetPoint, feedback_value = car_state.speed)
                    car_controls.steering = SteerSetPoint
                    car_controls.throttle = SpeedControllerOutput
                    client.setCarControls(car_controls)
                    client.simPause(True)

            prev_states[3*i]   = state_buf
            prev_states[3*i+1] = state_buf
```

```
            prev_states[3*i+2] = state_buf
            GT_Labels[3*i,:]=                    [car_state.GTSteering
,(car_state.GTSpeed-7)/8]
            GT_Labels[3*i+1,:]=             [car_state.GTSteering-0.6
,(car_state.GTSpeed-7)/8]
            GT_Labels[3*i+2,:]=             [car_state.GTSteering+0.6
,(car_state.GTSpeed-7)/8]
                  min_batch_number=
int(len(idx_Unsafe_in_choosen_class)/              32)              +
(len(idx_Unsafe_in_choosen_class) % 32 > 0)

        req_sample_number = min_batch_number*32
        missing_sample_number=                  int(req_sample_number-
len(idx_Unsafe_in_choosen_class))
        idx_Unsafe_in_other_class_random_sampled                      =
random.sample(idx_Unsafe_in_other_class,missing_sample_number)
        full_idx_list          =idx_Unsafe_in_choosen_class          +
idx_Unsafe_in_other_class_random_sampled
        full_idx_list_shuffled = full_idx_list[:] # Copy words
        shuffle(full_idx_list_shuffled) # Shuffle newwords
        return
prev_states[full_idx_list_shuffled],state_buf,image_buf[full_idx_list
_shuffled], GT_Labels[full_idx_list_shuffled]
```

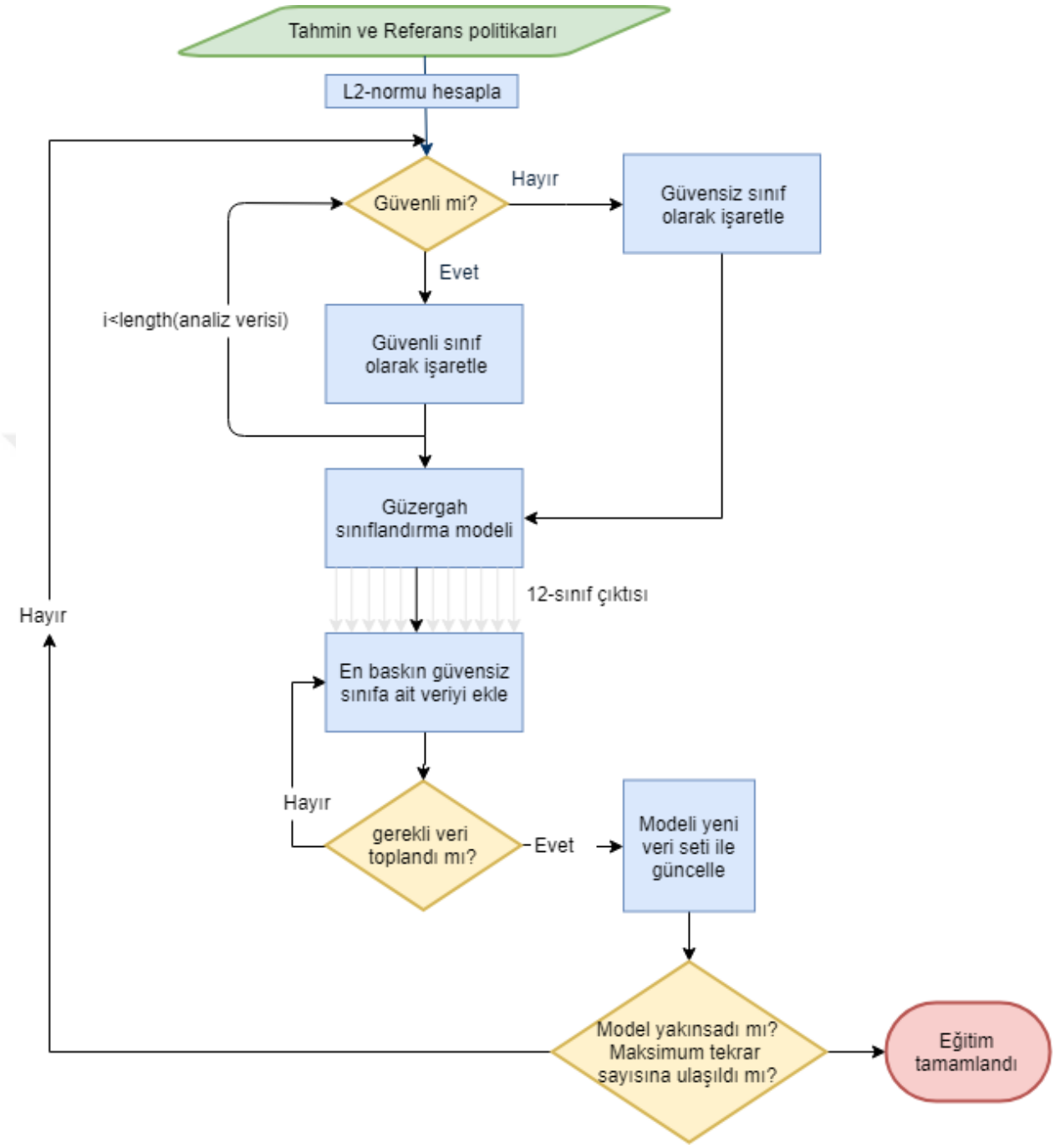
Kod Parçası V.3 Önerilen algoritma her örnek için ana döngüsü Python Kodu

```
def Classifier(PrSTeer_and_PrSpeed, GTSTeer_and_GTSpeed):

        if np.linalg.norm(GTSTeer_and_GTSpeed - PrSTeer_and_PrSpeed)
< Tau:

            if GTSTeer_and_GTSpeed[0] < turn_class_trigger_left and
GTSTeer_and_GTSpeed[1] < speed_turn_trigger:
                binaryLabel = np.array(0)
            elif GTSTeer_and_GTSpeed[0] < turn_class_trigger_left and
GTSTeer_and_GTSpeed[1] > speed_turn_trigger:
                binaryLabel = np.array(1)
            elif GTSTeer_and_GTSpeed[0] > turn_class_trigger_right
and GTSTeer_and_GTSpeed[1] < speed_turn_trigger:
                binaryLabel = np.array(2)
            elif GTSTeer_and_GTSpeed[0] > turn_class_trigger_right
and GTSTeer_and_GTSpeed[1] > speed_turn_trigger:
                binaryLabel = np.array(3)
            elif GTSTeer_and_GTSpeed[0] < turn_class_trigger_right
and GTSTeer_and_GTSpeed[0] > turn_class_trigger_left and
GTSTeer_and_GTSpeed[1] < speed_straight_trigger:
                binaryLabel = np.array(4)
            elif GTSTeer_and_GTSpeed[0] < turn_class_trigger_right
and GTSTeer_and_GTSpeed[0] > turn_class_trigger_left and
GTSTeer_and_GTSpeed[1] > speed_straight_trigger:
                binaryLabel = np.array(5)
            else:
                print('Error in Labeling')

        else:
```

```
            if (GTSTeer_and_GTSpeed[0] < turn_class_trigger_left and
GTSTeer_and_GTSpeed[1] < speed_turn_trigger):
                binaryLabel = np.array(6)
            elif (GTSTeer_and_GTSpeed[0] < turn_class_trigger_left
and GTSTeer_and_GTSpeed[1] > speed_turn_trigger):
                binaryLabel = np.array(7)
            elif GTSTeer_and_GTSpeed[0] > turn_class_trigger_right
and GTSTeer_and_GTSpeed[1] < speed_turn_trigger:
                binaryLabel = np.array(8)
            elif GTSTeer_and_GTSpeed[0] > turn_class_trigger_right
and GTSTeer_and_GTSpeed[1] > speed_turn_trigger:
                binaryLabel = np.array(9)
            elif (GTSTeer_and_GTSpeed[0] < turn_class_trigger_right
and GTSTeer_and_GTSpeed[0] > turn_class_trigger_left)
            and GTSTeer_and_GTSpeed[1] < speed_straight_trigger:
                binaryLabel = np.array(10)
            elif GTSTeer_and_GTSpeed[0] < turn_class_trigger_right
and GTSTeer_and_GTSpeed[0] > turn_class_trigger_left
            and GTSTeer_and_GTSpeed[1] > speed_straight_trigger:
                binaryLabel = np.array(11)
            else:
                print('Error in Labeling')
        return binaryLabel
```

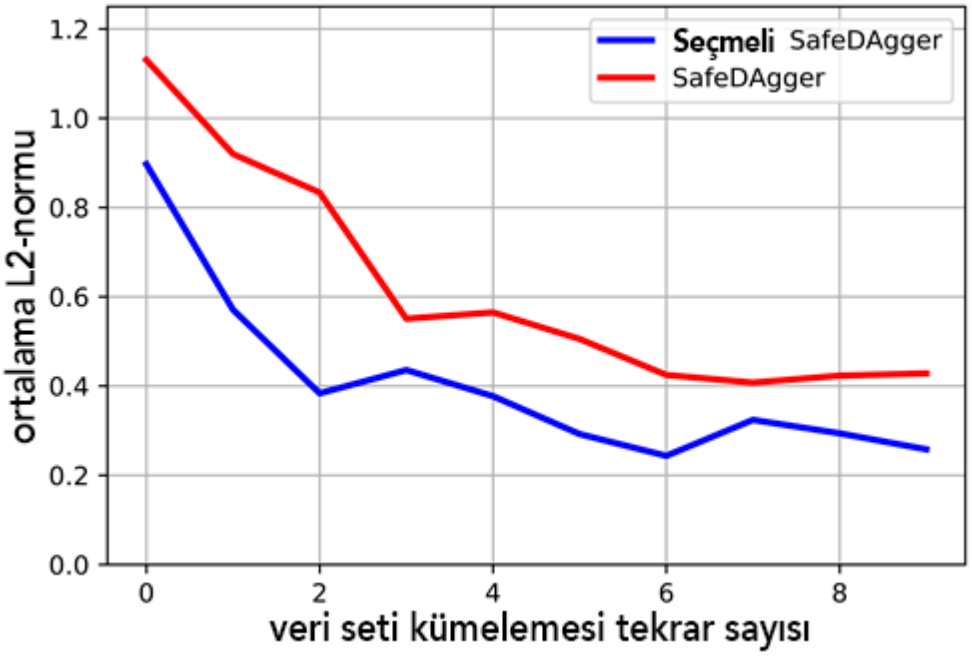
Kod Parçası V.4 Önerilen algoritma güvenlik sınıflandırıcısı kodu

Eğitimden sonra π_0 yerini π_1 politikasına bırakır ve önceden toplanmış örnekler yeni politika ile tekrardan işlenir ve baskın zayıf sınıflar ihtiyaca göre tekrardan belirlenir. Bu süreç Şekil V.16 üzerinde görüldüğü gibi 10 kere daha tekrarlanır.



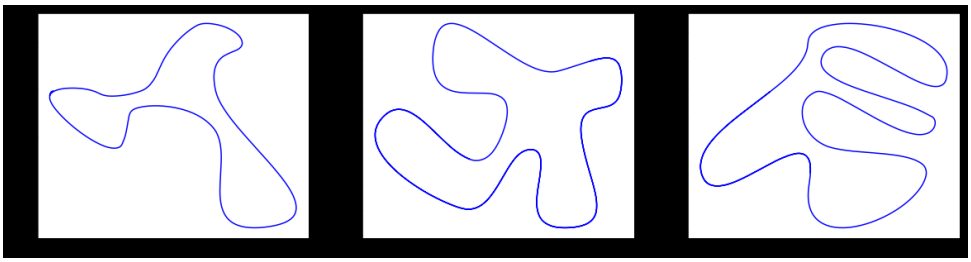
Şekil V.16 Eğitim süreci akış şeması

5.4. Simülasyon Sonuçları



Şekil V.17 Her tekrarda 10000 örnekten oluşan L2 normu ortalaması

Eğitilmiş model her bir tekrarlamada çevreden 10000 örnek alarak ve L2-normları hesaplanarak test edilmiştir. Şekil V.17'da görüldüğü üzere önerilen yöntem, SafeDagger yöntemine göre, her tekrarda daha yüksek performans göstermiştir. Algoritmanın genel performansı görülen daha önceden test edilmemiş 3 farklı test pistinde (Şekil V.18) test edilmiştir.



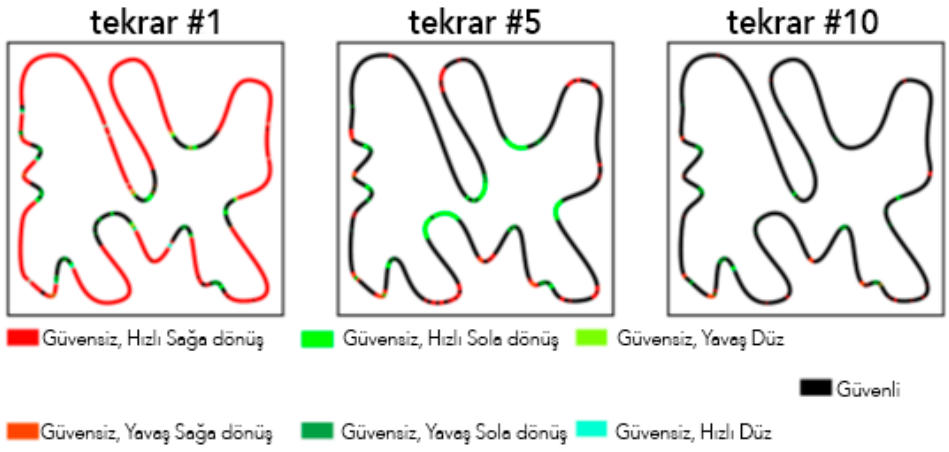
Şekil V.18 Test pistlerinin geometrileri

Tablo V-1 Eğitilmemiş test pistinde ortalama L2-normlarının karşılaştırması

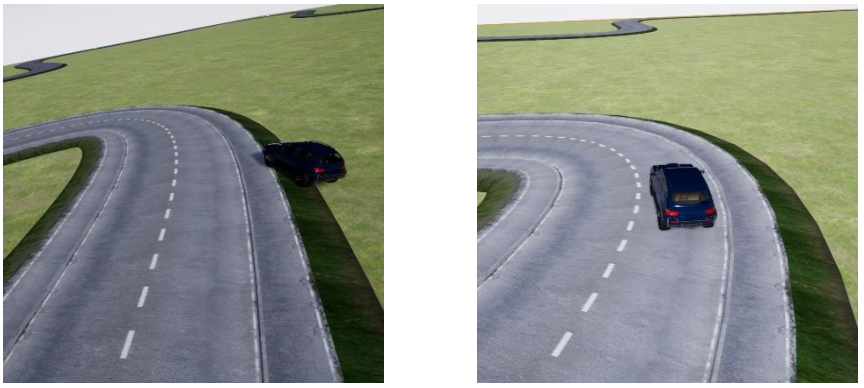
	Önerilen Yöntem	SafeDAgger
1.Test Pisti	0.4794	0.5518
2.Test Pisti	0.3295	0.4986
3.Test Pisti	0.3254	0.3632

Tablo V-1'de görüleceği üzere önerilen örnek açısından verimli, seçmeli ve güvenli veri kümelemesi yöntemi, güvenli veri kümelemesi yöntemine bütün pistlerde üstünlük sağlamıştır. Önerilen algoritmanın seçmeli yapısı diğer tüm sınıflara baskın gelen güvensiz sınıfları belirleyerek model hatalarının, diğer veri seti kümelemesi yöntemlerine göre, daha hızlı bir şekilde yakınsamasını sağlamıştır.

Şekil V.19'de görüldüğü üzere, veri seti kümelemesi tekrar #1'de, veri setinin büyük bir bölümü güvensiz olarak işaretlemiştir. Tekrarlar devam ettikçe tekrar #5 ve tekrar #10'da görüldüğü üzere problemli bölümler teker tek çözülmüş ve model hataları en aza indirgenmiştir.



Şekil V.19 Önerilen modelin yakınsama oranı. Veri seti kümelemesi işlemi tekrarlandıkça sonuçların daha güvenli hale geldiğini göstermektedir.

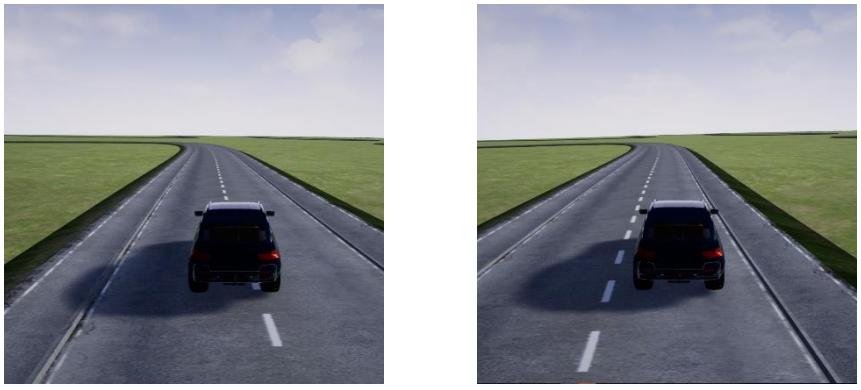


(a) Başlangıç eğitim seti (güvensiz) (b) Eğitim tekrar #10 (güvenli)

Şekil V.20 Eğitim öncesi ve sonrası düşük hızla sola dönüş

(a) Başlangıç eğitim seti (güvensiz) (b) Eğitim tekrar #10 (güvenli)

Şekil V.21 Eğitim öncesi ve sonrası yüksek hızla sağa dönüş



(a) Başlangıç eğitim seti (güvensiz) (b) Eğitim tekrar #10 (güvenli)

Şekil V.22 Eğitim öncesi ve sonrası yüksek hızlı düz

VI. SONUÇ

Bu çalışmada, Güvenli Veri Seti Kümelemesi (SafeDAgger) algoritmasının, veri seti kümelemesi konusunda örnek toplamada seçici ve verimli, özgün bir uyarlaması sürücüsüz araçlar için gerçeklenmiştir. Her bir tekrarda önerilen algoritma, tahmin modelinin performansını değerlendirmiş ve farklı güzergâh sınıflarında olan zayıflıkları tespit ederek modeli bu zayıflıklara karşı eğitmiştir. Testler sonucunda algoritmaya ait elde edilen performans sonuçları da uyarlamanın temel aldığı Veri Seti Kümelemesi (DAgger) algoritması ve Güvenli Veri Seti Kümelemesi (SafeDAgger) algoritmasına karşı üstün gelmiştir. Önerilen algoritma sürücüsüz araçlar dışında da oyunlarda yapay zekâ elemanları yetiştirmek, el yazısı tanıma, arabalar dışında diğer sürücüsüz araçlar ve benzeri görüntü işleme temelli birçok başka alanda uygulanabilir bir yapıdadır. Gelecekte sınıflandırma modelinin iyileştirilmesi ve sürüş politikası çıktısının hata oranının sıfıra yakınsaması konularında çalışma devam edecektir. Bunun yanında algoritmanın simülasyon ortamından çıkarılıp gerçek hayatta, üzerinde kameralar ve bir sürüş modülü bulunan kumandalı araç üzerinde test edilmesi planlanmaktadır. Bu aşamada başarılı olunduğu takdirde ise pist ortamında gerçek araçlarla test sürecinin başlatılması planlanmaktadır.

VII. KAYNAKÇA

[1] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, Fletcher, E. Frazzoli, A. Huang, S. Karaman ve O. Koch, «A Perception-Driven Autonomous Urban Vehicle,» *Journal of Field Robotics,* no. 10, pp. 727-748.

[2] Z. Chen ve X. Huang, «End-to-end learning for lane keeping of selfdriving cars,» *IEEE Intelligent Vehicles Symposium,* cilt IV, pp. 1856-1860, 2017.

[3] D. Pomerleau, «ALVINN: An Autonomous Land Vehicle in a Neural Network,» *NIPS,* 1988.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao ve K. Zieba, «End to end learning for self-driving cars,» *CoRR,* cilt 07316, p. 1604, 2016.

[5] J. Kim ve J. Canny, «Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention,» *IEEE International Conference on Computer Vision (ICCV), Venice,* pp. 2961-2969 , 2017.

[6] H. Xu, Y. Gao, F. Yu ve T. Darrell, «End-to-End Learning of Driving Models from Large-Scale Video Datasets,» *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu,* pp. 3530-3538, 2017.

[7] Z. Yang, Y. Zhang, J. Yu, J. Cai ve J. Luo, «End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception,» *arXiv preprint,* no. 1801.06734, 2018.

[8] S. Ross, G. Gordon ve A. Bagnell, «A reduction of imitation learning and structured prediction to no-regret online learning,» *Journal of Machine Learning Research*, no. 15, pp. 627-635, 2011.

[9] J. Zhang ve K. Cho, «Query-efficient imitation learning for end-to-end simulated driving,» *AAAI*, 2017.

[10] K. Menda ve e. al, «EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning,» *CoRR*, cilt abs/1807.08364, 2018.

[11] T. . M. Mitchell , Machine Learning, New York, NY, USA: McGraw-Hill, Inc., 1997.

[12] I. Goodfellow, Y. Bengio ve A. Courville, Deep Learning, MIT Press, 2016.

[13] S. Shah, D. Dey, C. Lovett ve A. Kapoor, «Airsim: High-fidelity visual and physical simulation for autonomous vehicles,» *Field and Service Robotics* , 2017.

[14] M. Teti, E. Barenholtz, S. Martin ve W. Hahn, «A Systematic Comparison of Deep Learning Architectures in an Autonomous Vehicle,» *arXiv preprint*, cilt 1803.09386.

[15] S. Du, H. Guo ve A. Simpson, «Self-Driving Car Steering Angle Prediction Based on Image Recognition,» 2017.

[16] F. Codevilla, M. Müller, A. Lopez, V. Koltun ve A. Dosovitskiy, «End-to-end driving via conditional imitation learning,» *Available: https://arxiv.org/abs/1710.02410*, 2017.

ÖZGEÇMİŞ

EĞİTİM

Bilgisayar Mühendisliği Yüksek Lisans Öğrencisi, Okan Üniversitesi, Haziran 2016 – Günümüz.

Mühendislik ve Doğa Bilimleri Fakültesi Mekatronik Lisans Mezunu, Sabancı Üniversitesi, Eylül 2009 – Haziran 2014

YAYINLAR

Kızılırmak, F. O. ve Erçakır, O. 2017. *"Derin Öğrenme Tabanlı Otonom Sürüş Kontrol Algoritması"*, Bahçeşehir Üniversitesi XXII. Türkiye'de Internet Konferansı, İstanbul, TÜRKİYE

Kızılırmak, F. O. ve Erçakır, O. 2017. *"Taşıtlar Arası Haberleşme Sistemlerinde Karşılaşılan Güvenlik Sorunları ve Çözüm Mimarileri"*, Bahçeşehir Üniversitesi XXII. Türkiye'de Internet Konferansı, İstanbul, TÜRKİYE