



SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ



SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

T.C.
**SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ**

OTONOM QUADCOPTER İLE EN HIZLI TURU TAMAMLAMA

Fatih Ersagun TOSUN

MÜHENDİSLİK TASARIMI

**Mayıs-2025
KONYA
Her Hakkı Saklıdır**

PROJE KABUL VE ONAYI

Fatih Ersagun TOSUN tarafından hazırlanan “OTONOM QUADCOPTER İLE EN HIZLI TURU TAMAMLAMA” adlı proje çalışması .../.../... tarihinde aşağıdaki jüri üyeleri tarafından oy birliği/oy çokluğu ile Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği bölümünde Mühendislik Tasarımı Projesi olarak kabul edilmiştir.

Jüri Üyeleri

İmza

Danışman

Üye

Üye

Yukarıdaki sonucu onaylarım.

Bilgisayar Mühendisliği
Bölüm Başkanı

PROJE BİLDİRİMİ

Bu projedeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve proje yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yaptığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by project rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

İmza

Fatih Ersagun TOSUN

Tarih:/..../....

ÖZET

MÜHENDİSLİK TASARIMI

OTONOM QUADCOPTER İLE EN HIZLI TURU TAMAMLAMA

Fatih Ersagun TOSUN

SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2025, 49 Sayfa

Jüri

Bu proje, önceden tanımlanmış bir parkurda yer alan kontrol noktalarından geçerek en kısa sürede turu tamamlayabilen otonom bir quadcopter sistemi geliştirmeyi amaçlamaktadır. Sistem, kullanıcı tarafından tamamlanan bir turda Lidar ve GPS verilerini kullanarak kontrol noktalarını tespit edecek ve A* algoritmasıyla optimal bir rota planlayacaktır. Oluşturulan rota, PID kontrolcüsü ile takip edilecek ve her turun sonunda Lidar, GPS ve IMU sensörlerinden elde edilen veriler analiz için sunulacaktır. Proje, simülasyon ortamında başlayacak olup, bu sayede teorik düzeyde otonom hava araçlarının parkur bazlı hareket yetenekleri üzerine kapsamlı bir çalışma ortaya konması hedeflenmektedir.

Anahtar Kelimeler: AStar, Kontrol, İHA, Parkur, Optimizasyon, Otonom

ABSTRACT

ENGINEERING DESIGN

PROJE BAŞLIĞININ İNGİLİZCE'SİNİ BURAYA YAZINIZ

Fatih Ersagun TOSUN

**SELCUK UNIVERSITY
FACULTY OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING**

2025, 49 Pages

Jury

This project aims to develop an autonomous quadcopter system capable of completing a given course in the shortest possible time by passing through predefined checkpoints on the track. The system will detect checkpoints using Lidar and GPS data during a user-completed lap, and this data will be used by path planning algorithm (A*) to generate an optimal route. The generated route will be tracked with a PID controller, and data from the Lidar, GPS, and IMU sensors will be recorded at the end of each lap and presented on a screen for analysis. The project will commence in a simulation environment, thereby aiming to present a comprehensive theoretical study on the course-based movement capabilities of autonomous aerial vehicles.

Keywords: AStar, Autonomous, Control, Optimization, Racing Circuit, UAV

ÖNSÖZ

Fatih Ersagun TOSUN

Konya / 2025

İÇİNDEKİLER

ÖZET	iv
ABSTRACT.....	v
ÖNSÖZ	vi
İÇİNDEKİLER	vii
SİMGELER VE KISALTMALAR	iv
1. GİRİŞ	1
1.1. Projenin Konusu	1
1.2. Projenin Amacı	1
1.3. Projenin Önemi	1
2. KAYNAK ARAŞTIRMASI	2
3. MATERİYAL VE YÖNTEM	4
3.1. İnsansız Hava Araçları (UAV)	4
3.1.1. İnsansız Hava Araçlarının Tarihçesi.....	4
3.1.2. İnsansız Hava Araçlarının Kullanım Alanları	5
3.1.3. İnsansız Hava Araçlarının Sınıflandırılması.....	5
3.2. Quadcopter Uçuş Mekanığı	6
3.2.1. İnsansız Hava Aracı için Quadcopter Seçilme Sebebi.....	6
3.2.2. Quadcopter Uçuş Fiziği	6
3.3. İnsansız Hava Aracı Üzerinde Kullanılabilen Sensörler	9
3.3.1. Ataletsel Ölçüm Birimi (IMU)	9
3.3.2. Küresel Navigasyon Uydu Sistemi (GNSS).....	9
3.3.3. LIDAR	10
3.3.4. Kamera.....	10
3.3.5. Barometre	10
3.3.6. Ultrasonik Sensörler	11
3.3.7. Manyetometre	11
3.3.8. VOR.....	11
3.4. Oransal İntegral Türevsel (PID) Kontrol Algoritması.....	12
3.4.1. Oransal (P) Kontrol.....	13
3.4.2. İntegral (I) Kontrol.....	14
3.4.3. Türevsel (D) Kontrol	14
3.4.4. Kontrol Katsayılarının Ayarlanması (Tuning).....	15
4. Uygulama.....	15
4.1 Simülasyon Ortamı	15
4.2 Simülasyon Ortamında İnsansız Hava Aracı	16
4.3 Oransal İntegral Türevsel (PID) Kontrol Algoritmasının Gerçekleştirilmesi	19
4.4. Simülasyon Ortamında Parkurun Oluşturulması	22

4.5. Kontrol Noktalarının Algılanması	24
4.6. Yol Planlama Algoritmasının Gerçekleştirilmesi	27
4.6.1 A* Algoritması	27
4.6.2 Simülasyon Üzerinde A* Algoritması	28
4.7. Tur Verilerinin Toplanması	30
4.7.1 Tur Verilerinin Toplanma İşlemi	31
4.7.2 Verilerin Görselleştirilmesi.....	31
5. SONUÇLAR VE ÖNERİLER	32
5.1 Sonuçlar	32
5.2 Öneriler	35
5.2.1 Kontrol Noktalarının Algılanması	35
5.2.2 Yol planlama Algoritması.....	36
5.2.3 Tur Optimizasyonu	36
5.2.4 Verilerin Görselleştirilmesi.....	36
KAYNAKLAR	37
EKLER	Error! Bookmark not defined.
ÖZGEÇMİŞ	Error! Bookmark not defined.

SİMGELER VE KISALTMALAR

Simgeler

A*: A Star

Kısaltmalar

UAV	: Unmanned Aerial Vehicle
GPS	: Global Positioning System
IMU	: Inertial Measurement Unit
GNSS	: Global Navigation Satellite
LIDAR	: Light Detection and Ranging
RADAR	: Radio Detection and Ranging
VHF	: Very High Frequency
VOR	: VHF Omnidirectional Range
IR	: Infra Red
SLAM	: Simultaneous Localization and Mapping
FPV	: First Person View
PID	: Propotional Integral Derivative
PD	: Propotional Derivative
Carla	: CAR Learning to Act
ROS	: Robot Operating System
PHGA	: Parçalı Hücresel Genetik Algoritma

1. GİRİŞ

1.1. Projenin Konusu

Bu proje, otonom bir insansız hava aracının (UAV), belirlenmiş bir parkur üzerinde hareket ederek en kısa sürede bitiş noktasına ulaşmasını sağlamak için gereken algoritmaların ve kontrol mekanizmalarının geliştirmesini ele almaktadır.

Projenin temel odağı, UAV'ın üzerindeki sensörler aracılığıyla çevresini algılaması ve yol planlama algoritmalarını kullanarak en uygun rotayı belirlemesidir. Bu rotayı belirledikten sonra, UAV üzerinde bulunan kontrol sistemi, rotayı takip ederek hem stabil bir uçuş gerçekleştirecek performansını gözlemlemek için tur veriler toplayacaktır. Toplanan bu veriler, ileriki zamanlarda uçuşlar için gerekli optimizasyon süreçlerinde kullanılabilir mesine ve UAV'ın performansının gözlemlenmesine olanak sağlayacaktır.

1.2. Projenin Amacı

Bu projenin temel amacı, otonom bir UAV'ın belirli bir parkuru en hızlı ve verimli bir şekilde tamamlamasını sağlayacak algoritmalar ve kontrol sistemleri geliştirmektir. Çalışma kapsamında, UAV'ın çevresini algılayarak parkur üzerinde en uygun rotayı belirlemesi, rotayı takip etmesi ve engellerden kaçınması gibi beceriler geliştirilmesini hedeflenmektedir.

Proje, öncelikle simülasyon ortamında gerçekleştirilecektir. Simülasyon, algoritmaların geliştirilmesi ve performanslarının güvenli bir şekilde test edilmesi için uygun bir ortam sağlayacaktır. Simülasyondan elde edilmesinin ardından, geliştirilen sistem fiziksel bir UAV için uygun olup olmadığı sonucuna bakılacaktır.

1.3. Projenin Önemi

Otonom insansız hava araçları (UAV), son yıllarda hızla gelişen ve birçok farklı alanda uygulama alanı bulunan teknolojilerdir. Bu Proje, otonom UAV sistemlerinin geliştirilmesi ve performanslarının optimize edilmesi adına önemli bir adımdır. Geliştirilen algoritmalar ve kontrol sistemleri, yalnızca belirli parkurlarda değil, çok daha geniş bir uygulama yelpazesinde kullanılabilecek bir temel sağlayacaktır.

2. KAYNAK ARAŞTIRMASI

Bu bölümde, insansız hava araçları yönelik otonom yol takibi, yönüğe takibi ve stabilité kontrolü alanındaki akademik çalışmalar incelenmiştir. Klasik algoritmalar (A^* , Dijkstra vb.), yapay zekâ temelli yöntemler (ör. Derin pekiştirmeli öğrenme, genetik algoritmalar), çevresel etkenler (rüzgâr, engeller) ve SLAM yaklaşımları gibi farklı perspektiflerden yapılan araştırmalar derlenmiştir.

2024 yılında Soheila ve arkadaşları insansız hava araçlarında (UAV) yol planlama yöntemlerini kapsamlı bir şekilde inceleyen “UAV Path Planning Techniques: A Survey” adında bir derleme çalışması yapmıştır. Çalışmada, yol planlama teknikleri; klasik algoritmalar (A^* , Dijkstra gibi), yumuşak bilişim yöntemleri (genetik algoritmalar, yapay sinir ağları) ve hibrit yaklaşımlar olarak kategorize edilmiştir. Ayrıca, çevre modelleme, yol yapısı, enerji verimliliği, optimalite ve tamlık gibi kriterler değerlendirilmiştir. Her yöntemin avantajları ve kısıtlamaları tartışılırak farklı uygulamalar için öneriler sunulmuştur. Çalışma, özellikle otonom uçuş görevlerinde hangi yöntemlerin daha etkili olduğunu anlamak isteyen araştırmacılar için önemli bir referans kaynağıdır. (Soheila ve ark., 2024)

2024 yılında Guoqing ve arkadaşları hassas tarım uygulamalarında kullanılan UAV için kısa mesafe öncelikli bir yol planlama algoritmasını üzerine çalışma yapmıştır. Çalışmada, özellikle tarım alanlarının kapsamlı bir şekilde taranması için gereken enerji verimliliği ve zaman optimizasyonu üzerinde durulmuştur. Algoritma, tarım alanlarının düzenine ve boyutuna göre en kısa mesafeli rotaları belirleyerek UAV'nin uçuş süresini ve enerji tüketimini minimize etmeyi hedefler. Çalışma hem simülasyon hem de gerçek dünya uygulamalarında test edilmiş, sonuçlar algoritmanın etkinliğini ve uygulama potansiyelini ortaya koymuştur. Bu araştırma, hassas tarımda otonom sistemlerin optimizasyonu için yenilikçi bir çözüm sunmaktadır. (Guoqing ve ark., 2024)

2023 yılında Ahmad ve arkadaşları UAV için engel algılama ve önleme yöntemlerini kapsamlı bir şekilde incelemektedir. Çalışma, sensör tabanlı yöntemler (LIDAR, radar, kamera), algoritma odaklı yaklaşımlar (makine öğrenimi, derin öğrenme) ve hibrit sistemler dahil olmak üzere mevcut teknolojileri detaylandırmaktadır. Algoritmaların çevresel koşullara adaptasyonu, gerçek zamanlı işlem yetenekleri ve doğruluk oranları gibi performans ölçütleri karşılaştırılmıştır.

Ayrıca, farklı uygulama alanlarına uygun yöntemlerin avantajları ve kısıtlamaları değerlendirilmiştir. Çalışma, UAV'lerin karmaşık ve dinamik ortamlarda güvenli bir şekilde hareket edebilmesi için kritik bir bilgi kaynağı sunmaktadır. (Ahmad ve ark., 2023)

2024 yılında Ahmet ve arkadaşlarının Gazi Üniversitesi Mühendislik Mimarlık Fakültesi dergisinde yayınladığı makalede drone rotalarının optimizasyonu için yeni bir genetik algoritma tasarlanmıştır. UAV için performans odaklı yol planlama sürecinde parçalı hücresel genetik algoritmayı (PHGA) inceleyen bir çalışmıştır. Bu yöntem, klasik genetik algoritmaların esnekliğini, hücresel yapıların bölgesel etkileşim yetenekleriyle birleştirerek daha hızlı ve etkili çözüm sunmayı hedeflemektedir. Araştırmada, UAV'nin uçuş performansını etkileyen enerji tüketimi, menzil ve manevra kabiliyeti gibi faktörler dikkate alınmış, algoritma bu kriterlere göre optimize edilmiştir. Simülasyon sonuçları, PHGA'nın karmaşık ve dinamik çevrelerde etkili yol planlaması yapabildiğini ve geleneksel yöntemlere kıyasla daha iyi performans sağladığını göstermiştir. Bu çalışma, UAV uygulamalarında hem güvenilir hem de verimli bir yol planlama stratejisi geliştirilmesi için önemli bir katkı sunmaktadır. (Ahmet ve ark., 2024)

Hayri'nin 2022 yılında doktora tezinde farklı çevresel koşullarda quadrotor insansız hava araçları için meta-sezgisel yöntemler kullanılarak rota optimizasyonunu ele almaktadır. Çalışmada, genetik algoritma, parçacık sürü optimizasyonu ve yapay arı kolonisi gibi meta-sezgisel teknikler karşılaştırılmış ve dinamik çevresel faktörlere adaptasyon yetenekleri incelenmiştir. Bu yöntemler, enerji verimliliği, uçuş süresi ve engellerden kaçınma gibi kriterlere göre optimize edilmiştir. Ayrıca, seçilen en uygun algoritma gerçek dünya koşullarında uygulanarak etkinliği test edilmiştir. Çalışmanın sonuçları, farklı çevre şartlarında meta-sezgisel yöntemlerin quadrotor UAV'lar için etkili ve güvenilir bir rota planlama stratejisi sunduğunu göstermektedir. (Hayri, 2022)

3. MATERİYAL VE YÖNTEM

3.1. İnsansız Hava Araçları (UAV)

İnsansız hava araçları, İngilizce "Unmanned Aerial Vehicle" teriminin Türkçe kısaltması olup, temel olarak üzerinde herhangi bir pilotun bulunmadığı ve havada hareket edebilen bir hava aracı anlamına gelir. Bu hava araçları, yaygın olarak drone olarak da bilinir ve uzaktan bir operatör tarafından kontrol edilebildiği gibi, önceden programlanmış yazılımlar ve Küresel Konumlama Sistemi (GPS) aracılığıyla otonom olarak da uçabilirler. İHA sistemleri sadece hava aracını değil, aynı zamanda yer kontrol istasyonlarını ve veri bağlantılarını da içerir ve robotik ile havacılık teknolojilerinin birleşimiyle ortaya çıkarlar, çeşitli sensörler ve kameralarla donatılabilirler ve askeri, ticari, sivil ve rekreatif alanlar da dahil olmak üzere pek çok farklı amaç için kullanılabilirler. (UMILES Group, 2023)

3.1.1. İnsansız Hava Araçlarının Tarihçesi

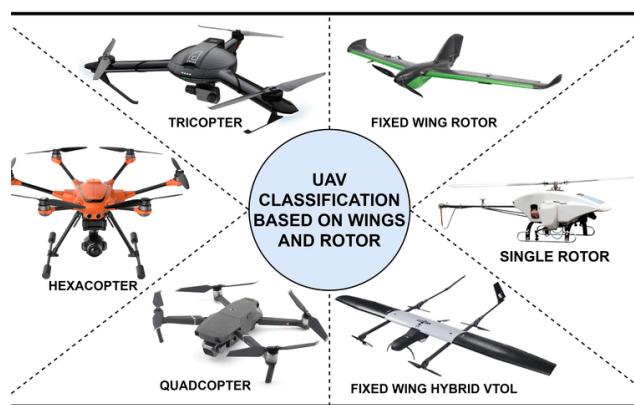
İnsansız hava araçlarının geçmişi Birinci Dünya Savaşı sırasında Britanya ve ABD'de geliştirilen ilk pilotsuz araçlara kadar uzanmaktadır; Britanya'nın 1917'de test edilen Aerial Target adlı radyo kontrollü uçağı ve ABD'nin 1918'de ilk uçuşunu yapan Sperry otomatik pilotu bu öncü örneklerdir. Bu ilk denemeler umut vadedici olsa da savaş sırasında operasyonel olarak kullanılmamışlardır. İki savaş arası dönemde UAV'ların geliştirilmesi ve test edilmesi devam etmiş, 1935'te Britanya'nın hedef eğitim amaçlı radyo kontrollü uçakları üretilmiş ve bu dönemde 'drone' teriminin, bu modellerden birinin adı olan 'Queen Bee'den ilham alınarak kullanılmaya başlandığı düşünülmektedir. Vietnam Savaşı'nda keşif UAV'ları büyük ölçüde ilk kez kullanılmış ve UAV'lar muharebede aldatıcı unsur, sabit hedeflere karşı füze fırlatma ve psikolojik operasyonlar için broşür atma gibi yeni roller üstlenmeye başlamıştır. Vietnam Savaşı'nı takiben, Britanya ve ABD dışındaki ülkeler de insansız hava teknolojilerini keşfetmeye başlamış ve zamanla daha gelişmiş, daha uzun süre havada kalabilen ve daha yüksek irtifalara çıkabilen yeni modeller ortaya çıkmıştır. Başlangıçta askeri amaçlarla geliştirilen UAV'lar, günümüzde iklim değişikliği izleme, doğal afetlerde arama kurtarma, fotoğrafçılık, film yapımı ve mal teslimatı gibi birçok farklı alanda kullanılmaktadır. (Imperial War Museums)

3.1.2. İnsansız Hava Araçlarının Kullanım Alanları

İnsansız hava araçları, tarımda ve kargo taşımacılığında kullanılırken; sivil alanda arama ve kurtarma çalışmalarında, trafik izlemede, hava durumu ölçümlerinde ve yangınla mücadelede önemli rol oynarlar. Askeri uygulamalarda hava sahası savunması, istihbarat toplama, keşif ve hatta silah platformu olarak görev alabilirler. Rekreasyonel olarak hobi amaçlı uçuşlar diğer alanlarda sinematografik çekimler, eğlence amaçlı ışık gösterileri, bilimsel araştırmalar, acil durum müdahalesi, güvenlik uygulamaları ve 3D modelleme gibi çeşitli amaçlarla da kullanılmaktadır. Bu çok yönlü kullanım alanları, UAV teknolojisinin sürekli gelişimine paralel olarak artmaya devam etmektedir. (Ferrovial, 2021)

3.1.3. İnsansız Hava Araçlarının Sınıflandırılması

İnsansız hava araçları, kanat şekillerine göre temel olarak üç ana kategoriye ayrılabilir: rotorlu UAV'lar, sabit kanatlı UAV'lar ve hibrit dikey kalkış ve iniş (VTOL) sabit kanatlı UAV'lar. Rotorlu UAV'lar, dikey olarak kalkış ve iniş yetenekleri ile öne çıkarlar ve bu manevraları gerçekleştirmek için bir veya daha fazla rotora (pervane) sahip olurlar. Kendi içlerinde de farklılaşan rotorlu UAV'lar arasında tek rotorlu UAV'lar, helikopterlere benzer yapılarıyla daha ağır yükleri taşıma kapasitesine sahipken, çok rotorlu UAV'lar (quadcopter, hexacopter, octocopter gibi), kolay kontrol edilebilirlikleri ve manevra kabiliyetleri sayesinde hem başlangıç seviyesi kullanıcılar hem de çeşitli uygulamalar için popüler bir seçeneklerdir.



Şekil 3.1. İnsansız Hava Araçlarının Sınıflandırılmış Hali

Diğer yandan, sabit kanatlı UAV'lar, geleneksel uçak tasarımlarını andırır ve havada kalmak için kanatlarının sağladığı kaldırma kuvvetine ihtiyaç duyarlar; bu nedenle, genellikle kalkış ve iniş için pistlere ihtiyaç duyarlar. Daha uzun menzilli

görevler, askeri operasyonlar ve büyük ekipman taşımacılığı gibi alanlarda tercih edilirler. Son olarak, hibrit VTOL sabit kanatlı UAV'lar, hem dikey kalkış ve iniş yapabilme özelliğini taşırlar hem de sabit kanatlı UAV'lar daha verimli yatay uçuş yeteneklerinden faydalananılar. Bu sayede, operasyonel esneklik sağlayarak hem dar alanlardan kalkış imkanı sunarlar hem de uzun mesafelerde etkin uçuş gerçekleştirebilirler. Bu kanat şekli temelli sınıflandırmanın yanı sıra, UAV'lar kullanım amaçlarına (ticari, askeri, rekreatif vb.), ağırlıklarına, uçuş menzillerine, güç kaynaklarına (pil, benzin vb.) ve kontrol mekanizmalarına (uzaktan kontrollü, otonom) göre de farklı kategorilere ayrılabilirler. (TechTarget, 2021)

3.2. Quadcopter Uçuş Mekanığı

3.2.1. İnsansız Hava Aracı için Quadcopter Seçilme Sebebi

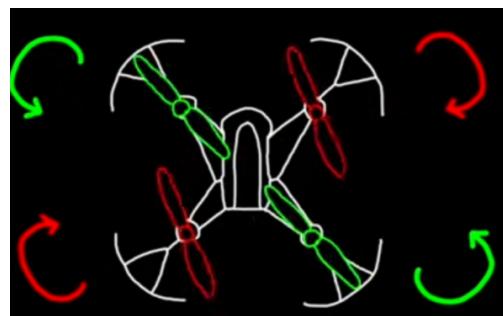
Kaynak araştırması bölümünü yazmaya buradan başlayınız. Bu projesi için quadcopter seçimi çeşitli avantajlar sunmaktadır. İlk olarak, quadcopterler yapısal basitlikleri ve uçuş esnekliği sayesinde tercih edilmektedir. Helikopterler gibi, quadcopterler de yatay olarak dönen dar kanat profilleri ile kaldırma kuvveti üretirler, bu da onları gerçek bir helikopter türü yapar. Geleneksel helikopterlerde bulunan ve karmaşık bir yapıya sahip olan swashplate mekanizmasına ihtiyaç duymazlar çünkü dört bağımsız rotora sahiptirler. Bu dört rotor sayesinde, swashplate mekanizmasının sağladığı kontrol seviyesine ulaşılabilir.

Ayrıca Modern mikroişlemcilerin maliyetlerinin düşmesiyle birlikte, quadcopterlerin elektronik ve hatta tamamen otonom kontrolü ticari, askeri ve hobi amaçlı uygulamalar için uygun hale gelmiştir. Altı serbestlik derecesine (üç doğrusal ve üç dönme) ve yalnızca dört bağımsız girişe (rotor hızları) sahip olmalarına rağmen, quadcopterler rotasyonel ve doğrusal hareketin birleşimi sayesinde bu altı serbestlik derecesini elde edebilirler. Bunlar sayesinde küçük boyutları ve hızlı manevra kabiliyetleri, karmaşık hava manevraları içeren uçuş rutinlerini gerçekleştirmeyi mümkün kılar. Bu özellikler, belirli bir parkuru otonom olarak tamamlamayı hedefleyen bir proje için quadcopter uygun bir seçim olmuştur.

3.2.2. Quadcopter Uçuş Fiziği

Bir quadcopter'in havada dengeli bir şekilde uçabilmesi ve manevra yapabilmesi için dört motorunun dönüş yönleri kritik bir öneme sahiptir. Temel olarak, karşılıklı

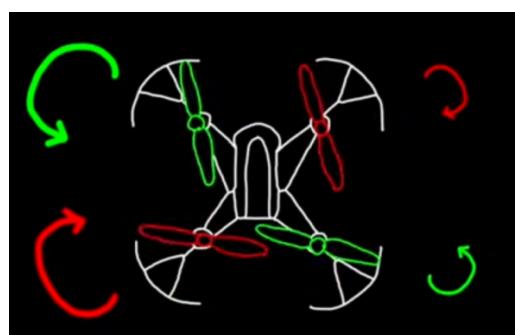
çapraz konumdaki motorlar aynı yönde dönerken, diğer çapraz konumdaki motorlar zıt yönde döner. Örneğin, ön sol ve arka sağ motorlar saat yönünde dönerken, ön sağ ve arka sol motorlar saat yönünün tersine dönebilir. Bu farklı dönüş yönlerinin en önemli sebebi tork dengelemesidir. Her motor döndüğünde, Newton'un üçüncü yasası gereği eşit ve zıt bir reaksiyon torku oluşturur. Eğer tüm motorlar aynı yönde dönseydi, bu torklar birleşerek quadcopter'in kendi merkezi etrafında zıt yönde dönmesine neden olurdu. Çapraz motorların zıt yönlerde dönmesi sayesinde, bu tork etkileri birbirini nötrler ve quadcopter'in dengeli bir şekilde havada asılı kalmasını sağlar.



Şekil 3.2. UAV üzerinde motorların dönüş yönlerinin gösterimi

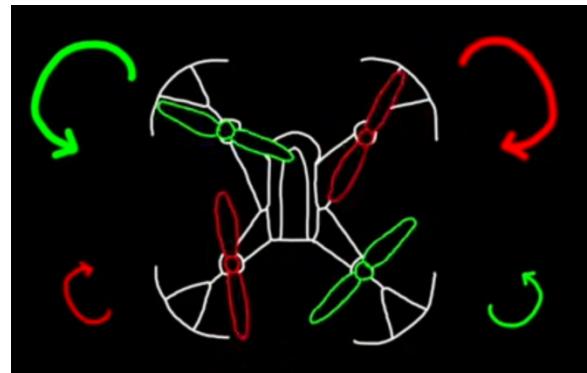
Havada asılı kalma (hovering) durumu, tüm dört motorun da eşit miktarda yukarı yönlü itme kuvveti üremesiyle gerçekleşir. Bu itme kuvvetinin toplamı, quadcopter'in ağırlığına eşit olduğunda, dikey yönde bir denge oluşur ve drone sabit bir irtifada kalır

Quadcopter'in sağa veya sola doğru hareket etmesi (roll), motorların hızlarının dengesiz bir şekilde ayarlanmasıyla mümkündür. Örneğin, sola doğru roll yapmak için, quadcopter'in sağ tarafındaki motorların gücü artırılırken, sol tarafındaki motorların gücü azaltılır. Bu güç farkı, quadcopter'in sol tarafa doğru eğilmesine ve bu yönde hareket etmesine neden olan bir net kuvvet oluşturur. Sağda doğru roll için ise bu işlemin tam tersi uygulanır.



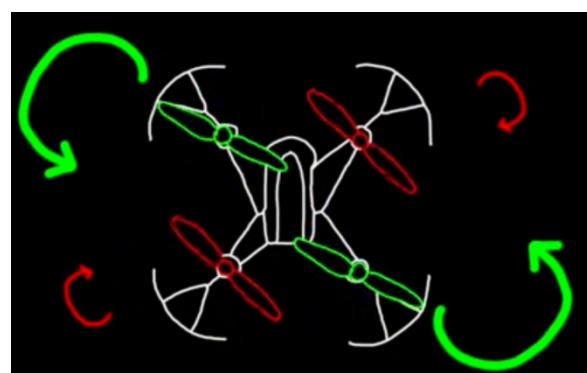
Şekil 3.3. UAV üzerinde sağa doğru roll hareketinin motorlar üzerinde gösterimi

İleri veya geri doğru hareket etme (pitch) mekanizması da benzer bir prensibe dayanır. İleriye doğru pitch yapmak için, arka taraftaki iki motorun gücü artırılırken, ön taraftaki iki motorun gücü azaltılır. Bu, quadcopter'in ön tarafının aşağıya doğru eğilmesine ve ileriye doğru hareket etmesine yol açar. Geriye doğru pitch için ise ön motorların gücü artırılır ve arka motorların gücü azaltılır.



Şekil 3.4. UAV üzerinde ileri yönlü pitch hareketinin motorlar üzerinde gösterimi

Quadcopter'in sağa veya sola dönmesi (yaw) ise, saat yönünde dönen motorlar ile saat yönünün tersine dönen motorlar arasındaki itme kuvveti dengesinin bozulmasıyla sağlanır. Saat yönünde yaw yapmak için, saat yönünün tersine dönen motorların (genellikle ön sol ve arka sağ) gücü artırılırken, saat yönünde dönen motorların (genellikle ön sağ ve arka sol) gücü azaltılır. Bu durum, quadcopter üzerinde saat yönünün tersine bir açısal momentum oluşturur. Açısal momentumun korunumu yasası gereği, quadcopter da saat yönünde dönmeye başlar. Saat yönünün tersine dönüş için ise bu işlemin tam tersi uygulanır.



Şekil 3.5. UAV üzerinde sola doğru yaw hareketinin motorlar üzerinde gösterimi

Özetle, bir quadcopter'in karmaşık hareketleri, dört motorunun hızlarının ve dolayısıyla ürettikleri itme kuvvetlerinin hassas bir şekilde kontrol edilmesiyle mümkün olur. Motorların zıt dönüş yönleri temel dengeyi sağlarken, her bir motorun hızının bağımsız olarak ayarlanabilmesi, UAV'un üç boyutlu uzayda istenen şekilde hareket etmesine olanak tanır.

3.3. İnsansız Hava Aracı Üzerinde Kullanılabilen Sensörler

3.3.1. Ataletsel Ölçüm Birimi (IMU)

Bir Ataletsel Ölçüm Birimi (IMU) genellikle akselerometre ve jiroskop sensörlerini bir arada bulundurur. Jiroskop, bir cihazın açısal hızını ölçerken, akselerometre ise doğrusal ivmeyi ölçer. Bu iki sensörün birleşimiyle oluşan IMU, bir quadcopter'in dönüş hızları ve ivmelenmeleri hakkında kapsamlı bilgi sağlar. Bu veriler, UAV'un dengesini koruması, istenilen yönde hareket etmesi ve havada sabit kalabilmesi (hovering) için kullanılır. IMU'lar quadcopter'lar için durum tahmini ve stabilizasyonunda kritik bir rol oynadığını söyleyebiliriz. (Baykar, 2025)

3.3.2. Küresel Navigasyon Uydu Sistemi (GNSS)

GNSS (Küresel Navigasyon Uydu Sistemi), típkı IMU (Ataletsel Ölçüm Birimi) gibi bir quadcopter'in uçuşu için kritik öneme sahip bir sensör sistemidir. GNSS, dünya yörüngeindeki bir uydu ağından gelen sinyalleri kullanarak aracın konumunu, hızını ve zamanını yüksek doğrulukla belirler. GPS, GLONASS, Galileo ve BeiDou gibi sistemleri kapsayan GNSS, enlemi, boylamı ve yüksekliği üçgenleme yöntemiyle hesaplar. Ayrıca, uydu sinyallerindeki Doppler kayması sayesinde hız ölçümü yapılabilir ve hassas zaman bilgisi elde edilebilir. GNSS'in temel işlevleri arasında otonom navigasyon, konum sabitleme, eve dönüş ve gerçek zamanlı konum izleme gibi görevler yer alır. IMU'nun ivme ve açısal hızı ölçerek denge ve yönelim sağlaması gibi, GNSS de quadcopter'in dünya üzerindeki konumunu tespit ederek karmaşık görevleri gerçekleştirmesini sağlar. Bu iki sistem birlikte çalışarak otonom uçuş kabiliyetlerinin temelini oluşturur. (Baykar, 2025)

3.3.3. LIDAR

Kaynak araştırması bölümünü yazmaya buradan başlayınız. LIDAR (Light Detection and Ranging), temel olarak nesneleri algılamak ve haritalamak için kızılötesi (IR) lazer ışını kullanan bir sistemdir. LIDAR, çok kısa süreli ve yüksek güçlü lazer pulsları yayar; bu pulsalar nesnelere çarpıp geri yansır ve yansıyan sinyaller hassas optik alıcılar tarafından algılanarak mesafe bilgisi hesaplanır. LIDAR sistemleri, engel tespiti, uzun menzilli haritalama, kapalı alan haritalaması ve otonom robotik uygulamalar gibi birçok alanda kullanılır. Özellikle yapay zeka ile entegre edildiğinde, quadcopter'in çevresel farkındalığını artırarak engellerden kaçınmasını ve kapalı ortamlarda yön bulmasını sağlar. (Baykar, 2025)

3.3.4. Kamera

Kamera sistemleri, bir quadcopter'in çevresini algılamasında ve çeşitli görevleri gerçekleştirmesinde kritik rol oynayan sensör bileşenleridir. Quadcopter'larda kamera kullanımının yaygınlaştığı başlıca alanlar arasında görüntü işleme ve nesne tanıma yer alır. Bilgisayarla görme algoritmaları sayesinde nesne takibi, hedef tanıma veya yüz algılama gibi görevler gerçekleştirilebilir. Haritalama ve yüzey analizi, özellikle tarım, inşaat ve arama-kurtarma gibi sektörlerde, quadcopter'ların havadan görüntü toplayarak ortamındaki yapıları analiz etmesini sağlar. Görsel SLAM yöntemi ile kamera verileri kullanılarak aynı anda hem harita çıkarılabilir hem de konum bu harita üzerinde belirlenebilir. FPV uçuşta ise pilot, quadcopter'a monte edilmiş kameradan gelen canlı görüntülerini bir ekran veya gözlük aracılığıyla izleyerek aracı uzaktan kontrol edebilir. Son olarak, kamera sistemleri yapay zekâ algoritmalarıyla entegre edildiğinde, quadcopter'in engelleri algılayarak yön tayini yapması ve görevleri bağımsız şekilde tamamlaması mümkün hale gelir. (Baykar, 2025)

3.3.5. Barometre

Barometreler, atmosfer basıncını ölçerek aracın deniz seviyesine göre yüksekliğini hesaplar. Bu sayede, quadcopter sabit bir irtifada uçabilir ve ani yükseklik değişimlerinden kaçınabilir. Özellikle otomatik kalkış, iniş ve belirli bir yükseklikte sabit durma gibi görevlerde barometrik veriler kritik rol oynar. Ayrıca, GPS verileriyle birleştirildiğinde daha hassas bir yükseklik ölçümleri sağlanabilir. Barometre, genellikle

IMU gibi sensörlerle birlikte çalışarak uçuş kontrol algoritmalarının daha kararlı ve güvenli şekilde işlemesine katkıda bulunur. (Baykar, 2025)

3.3.6. Ultrasonik Sensörler

Quadcopter'larda ultrasonik sensörler, özellikle düşük irtifalarda hassas mesafe ölçümü ve engel algılama amacıyla kullanılır. Bu sensörler, ses dalgaları gönderip bu dalgaların nesnelere çarpıp geri dönme süresini ölçerek mesafe hesaplar. Özellikle iç mekân uçuşlarında, GPS sinyallerinin yetersiz olduğu durumlarda, ultrasonik sensörler sayesinde quadcopter zemine olan uzaklığını doğru şekilde belirleyebilir. Bu da otomatik kalkış, iniş ve sabit yükseklikte uçuş gibi görevlerde büyük avantaj sağlar. Ayrıca, engellerin tespit edilerek çarpışmaların önlenmesinde de etkili bir çözüm sunar. Genellikle diğer sensörlerle birlikte çalışarak uçuş güvenliğini ve kontrol hassasiyetini artırır. (Baykar, 2025)

3.3.7. Manyetometre

Kaynak araştırması bölümünü yazmaya buradan başlayınız. Manyetometre, yarı iletken bir malzemenin dışsal manyetik akı alanına bağlı olarak elektriksel direncinin değişmesi prensibine dayanan bir cihazdır. Bu modül, X, Y, Z eksenlerindeki manyetik alan değerlerini ölçerek azimut ve yön açısını derece cinsinden belirleyebilir. Quadcopter'larda yön tespiti ve navigasyon görevlerinde kullanılarak cihazın hangi yöne baktığını veya hareket ettiğini anlamasını sağlar. Özellikle otonom navigasyon ve eve dönüş fonksiyonları gibi gelişmiş uygulamalarda doğru yön tayini için kritik rol oynar. Ayrıca, çevresel manyetik alanları analiz etmek amacıyla ham manyetik alan verileri de toplanabilir. (Baykar, 2025)

3.3.8. VOR

VOR (VHF Omnidirectional Range), havacılıkta sıkılıkla kullanılan bir radyo navigasyon sistemidir ve quadcopter gibi insansız hava araçlarında da konum ve yön belirleme amacıyla yardımcı bir sensör olarak entegre edilebilir. VOR sistemi, yer istasyonlarından yayılan çok yüksek frekanslı radyo sinyallerini kullanarak hava aracının istasyona göre açısal konumunu belirler. Bu sistem sayesinde, quadcopter hangi yönde ilerlediğini ve VOR istasyonuna göre konumunu hassas bir şekilde tayin edebilir. Özellikle uzun menzilli uçuşlar veya geleneksel havacılık altyapısıyla entegre sistemlerde VOR verileri, GNSS gibi diğer navigasyon sistemlerini destekleyici olarak kullanılabilir.

Böylece quadcopter hem otonom yön bulma hem de harici sinyal kaynaklarına dayanarak güvenilir uçuş planlaması gerçekleştirebilir. (Baykar, 2025)

3.4. Oransal İntegral Türevsel (PID) Kontrol Algoritması

Oransal İntegral Türevsel (PID) kontrol algoritması, endüstriyel otomasyon ve kontrol sistemlerinde yaygın olarak kullanılan kontrol yöntemini açıklamaktadır. PID kontrolü, bir sistemdeki belirli bir değişkeni (örneğin sıcaklık, hız, seviye vb.) istenen bir referans değerinde (setpoint) tutmak amacıyla kullanılan geri beslemeli bir kontrol döngüsüdür. Bu kısımda, PID kontrolörünün temel prensipleri, bileşenleri ve çalışma mantığı detaylı bir şekilde ele alınacaktır.

Bir PID kontrol algoritması oransal, integral ve türevsel kontrol terimlerinin toplanması ile Formül 3.1.'de ki şeklini alır. PID kontrolcüsü, sürekli olarak sistemin çıktı değerini ölçer ve bunu istenen setpoint ile karşılaştırır. Elde edilen hata sinyali, PID algoritması tarafından işlenerek sisteme uygulanacak kontrol sinyali hesaplanır. Amaç, bu kontrol sinyali aracılığıyla sistemin çıktısını referans değerine (setpoint) en kısa sürede ve en az salınımla ulaştırmaktır.

$$u(t) = u_p(t) + u_I(t) + u_D(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (3.1)$$

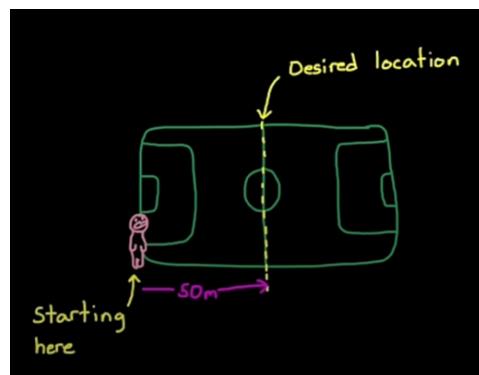
PID kontrol döngüsü tipik olarak şu şekilde işler: Sistemdeki bir değişkenin ölçülen değeri, önceden belirlenmiş bir referans değer (setpoint) ile karşılaştırılır. Eğer bu iki değer arasında bir farklılık, yani bir hata mevcutsa, PID kontrolörü bu hatayı temel olarak oransal, integral ve türevsel terimleri hesaplar. Bu hesaplama sonuçunda elde edilen kontrol sinyali, sistem üzerindeki bir aktuatörü (örneğin bir vana, motor veya ısıtıcı) harekete geçirerek sistemin çıktısını setpoint'e doğru yönlendirir. Bu ölçme, karşılaştırma ve kontrol sinyali üretme süreci kesintisiz bir döngü halinde devam eder, böylece sistemin çıktısının her zaman istenen hedef değerde tutulması amaçlanır.

3.4.1. Oransal (P) Kontrol

Oransal kontrol, kontrol sinyalini doğrudan hatayla orantılı olarak üretir. Hata ($e(t)$), referans değeri (setpoint) ile anlık değer (feedback) arasındaki farktır. Formül 3.2'de K_p değeri, hataya karşı daha hızlı bir tepki verilmesini sağlar ancak aşırı salınımlara ve kararsızlığa yol açabilir.

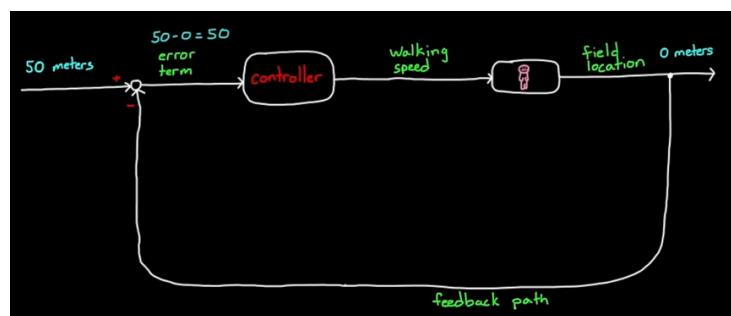
$$u_p(t) = K_p e(t) \quad (3.2)$$

Örnek: Oransal kontrol ile bir insanın hedef noktaya gitmek istemesi üzerinden gösterelim.



Şekil 3.6. Oransal kontrol için hazırlanmış örnek problem

Şekil 3.6'da problemiz olduğunu düşünelim. Gitmek istediğimi bir nokta var ve o noktayı uzaklığımız 50 metre bu bizim referans değerimiz (setpoint) oluyor. Şu anki durumuz 0 yani hiçbir yol kat etmemiş haldeyiz bu da bizim anlık değerimiz (feedback) oluyor. Bunun ikisinin arasındaki farkı ise bizim hatamız oluyor. Kontrolcüye verdığımız bu hata sayesinde bize bir çıktı üretecek ve oradaki insan gitmek istediği noktaya biraz daha yaklaşacak feedback ile tekrar sistem beslenecek.



Şekil 3.7. Oransal kontrol ile oluşturulmuş sistem

Şekil 3.7'de gördüğümüz kontrolcü (Controller) hataya bağlı olarak oransal olarak bize yürüme hızı veren bir sistem olarak düşünebiliriz. Mesela 0,1 olarak belirleyebiliriz. Hata değerimiz 50 iken 0,1 oransal katsayımız ile çarpılarak bunu hız olarak insana verilecek daha sonra feedback olarak geri donecek ve tekrar 45 metre olan hatamız 0,1 ile çarpılarak tekrar insana yürüme hızı olarak verilecek bu istenilen referans noktasına gidene kadar devam edecek. İşte bu bizim PID sistemin ilk aşaması olan P yani oransal olarak belirlediğimiz sistemdir.

3.4.2. Integral (I) Kontrol

Integral kontrol, sistemdeki geçmiş hataların birikimli etkisini hesaba katar. Oransal kontrol kalıcı bir durum hatası bırakabilirken, integral terim bu hatayı zaman içinde 'toplayarak' kontrol sinyalini artırır veya azaltır. Temel amacı, sistemin setpoint'e ulaştıktan sonra kararlı durumda kalan hatayı tamamen ortadan kaldırmaktır. Integral kontrol sinyali, hatanın zaman içindeki integralini (alanını) alır ve bu sayede küçük ama sürekli hataların zamanla büyümeyi ve kontrol sinyalini düzelterek hatayı sıfıra indirmesini sağlar. Integral kazancı (K_i) ayarlanarak bu düzeltme hızının ne kadar etkili olacağı belirlenir

$$u_I(t) = K_i \int_0^t e(\tau) d\tau \quad (3.3)$$

3.4.3. Türevsel (D) Kontrol

Türevsel kontrol ise hatanın değişim hızını dikkate alır. Amacı, gelecekteki olası hataları tahmin ederek kontrol sinyalini buna göre ayarlamaktır. Özellikle ani değişikliklerde veya hızlı hata artışlarında devreye girerek kontrol sinyalinde bir 'frenleme' etkisi yaratır. Bu sayede sistemin aşırı salınım yapmasını engellemeye ve daha kararlı bir yanıt vermesine yardımcı olur. Türevsel kazanç (K_d) ayarlanarak bu 'frenleme' etkisinin gücünü belirlenir.

$$u_D(t) = K_d \frac{de(t)}{dt} \quad (3.4)$$

3.4.4. Kontrol Katsayılarının Ayarlanması (Tuning)

PID kontrolör ayarı, sistemin hedef performansa ulaşmasını amaçlar. Genellikle önce oransal kazanç tepkiyi hızlandırır, sonra türevsel kazanç salınımları azaltır ve son olarak integral kazanç kalıcı hatayı giderir. Deneme yanılma veya Ziegler-Nichols gibi metodlarla yapılan bu ayarlama, doğru yapıldığında hızlı ve kararlı bir kontrol sağlar; aksi takdirde performans düşebilir.

4. Uygulama

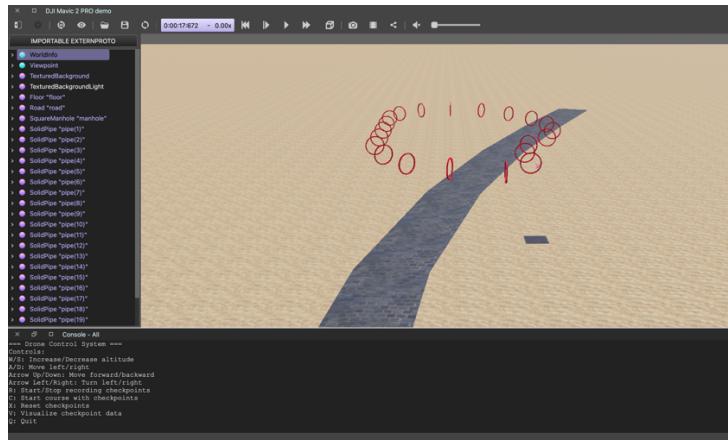
4.1 Simülasyon Ortamı

Robotik ve otonom sistemlerin geliştirilmesi sürecinde, gerçek dünya testleri maliyetli, zaman alıcı ve hatta tehlikeli olabilir. Bu nedenle, simülasyon ortamları, algoritmaların tasarılanması, test edilmesi ve doğrulanması için vazgeçilmez araçlar haline gelmiştir. Bu bölümde, robotik simülasyonunda öne çıkan üç platforma CARLA, Gazebo ve Webots degeinilecek ve ardından Webots simülatörünün temel kullanımı detaylı bir şekilde anlatılacaktır.

CARLA (CAR Learning to Act), özellikle otonom sürüsür araştırmaları için geliştirilmiş açık kaynaklı bir simülatördür. Gerçekçi şehir ortamları, çeşitli sensör modelleri ve trafik senaryoları sunar. Sensör verilerinin (kamera, lidar, radar vb.) gerçekçi bir şekilde simüle edilmesine olanak tanır.

Gazebo, geniş bir robot ve sensör yelpazesini destekleyen, açık kaynaklı ve genel amaçlı bir 3D robotik simülatördür. Karmaşık ortamların modellenmesine ve çoklu robot senaryolarının simüle edilmesine imkan tanır. ROS (Robot Operating System) ile güçlü bir entegrasyona sahiptir.

Webots, Cyberbotics tarafından geliştirilen, ticari bir robotik simülatördür. Kullanıcı dostu arayüzü, geniş robot kütüphanesi ve çeşitli sensör modelleri ile öne çıkar. Özellikle eğitim ve hızlı prototipleme için popülerdir. Python, C++, Java gibi dillerle programlanabilir ve ROS ile entegrasyonu da bulunmaktadır.



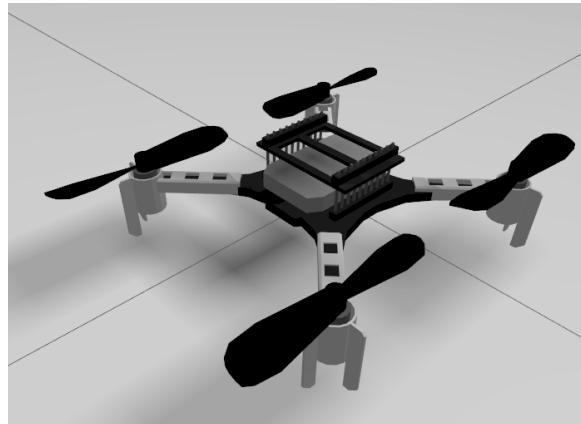
Şekil 4.1. Webots simülasyon arayüzü

Webots, robotik simülasyonları için sıkılıkla tercih edilen bir araçtır çünkü kullanıcı dostu arayüzü sayesinde yeni başlayanlar bile kolayca simülasyonlar oluşturabilir, geniş robot ve sensör kütüphanesi hızlı prototiplemeye olanak tanır ve Python, C++, Java gibi çeşitli programlama dilleriyle kontrol imkanı sunar.

Özellikle eğitim ve hızlı prototipleme için ideal olan Webots, aynı zamanda ROS entegrasyonu ve gerçekçi fizik simülasyonu sayesinde de güçlü bir platform sunarak, robotik araştırmaları ve geliştirmeleri için cazip bir seçenek haline gelir. Bunun dışında Mac cihazlarda da diğerlerine kıyasla daha kolay bir kurulum ile çalışmaya uygun hale getirilebilir.

4.2 Simülasyon Ortamında İnsansız Hava Aracı

Bu proje kapsamında Webots simülasyonu üzerinde bulunan “crazyfile” modeli kullanılmıştır. Dışarıdan bir drone modellenmesi ve onu sonradan Webots üzerinden ağırlığının, motor noktalarının, ağırlık merkezi gibi birçok durumdan oluşabilecek zorluklardan dolayı simülasyon üzerinde başka projelerde stabil bir şekilde çalışabilen bu model tercih edilmiştir.



Şekil 4.2. Webots crazyfile Modeli

Bu modelin üzerinde “X” şeklinde yerleştirilmiş 4 motor, sağ-sol ve ön-arka olmak üzere 4 tane lidar sensörü bunların yanında IMU ve GPS sensörleri de bulunmaktadır. Ekstra herhangi bir durum olduğu zaman bu model üzerine Bölüm 3.3’te gördüğümüz sensörleri ekleyip kod üzerinde kolaylıkla erişim sağlanabilmektedir.

Bu modeli C, Python ve Java dillerinden model üzerindeki sensörlerle ve motorları erişim imkanımız bulunmaktadır. Bu projede Python diliyle modeli kontrol etme tercih edilmiştir. Python ile model üzerine erişim sağlanırken “controller” kütüphanesinin “Robot” sınıfından yeni bir nesne üretmesi ile mümkün kılmaktadır.

```

1  """main controller."""
2
3  from controller import Robot
4
5  robot = Robot()
6
7
8  timestep = int(robot.getBasicTimeStep())
9
10
11 while robot.step(timestep) != -1:
12     pass
13
14

```

Şekil 4.3. Temel drone Python dosyası

Şekil 4.3’te gördüğünüz gibi “Robot” sınıfında yeni bir nesne üretimi ve bu nesne üzerinden zaman koşuluyla çalışan bir döngü görmekteyiz. Gerçekleştirmek istediğimiz tüm işlemleri bu döngü içerisinde olacaktır.

Projenin tamamında kendi oluşturduğumuz sınıf olan “Octopus” ile devam edecektir. Model üzerindeki tüm kontrolleri bu sınıfın içerisindeki fonksiyonlar

sağlayacaktır. Model üzerinde bulunan tüm sensörleri kullanabilmek için onları aktifleştirmek gerekmektedir. Octopus sınıf bu işlemi Şekil 4.4'teki kodlar ile yapmaktadır.

```
# Initialize sensors
self.imu = robot.getDevice("inertial_unit")
self.imu.enable(timestep)
self.gps = robot.getDevice("gps")
self.gps.enable(timestep)
self.gyro = robot.getDevice("gyro")
self.gyro.enable(timestep)

# Initialize lidar sensors
self.front_lidar = robot.getDevice("range_front")
self.front_lidar.enable(timestep)
self.back_lidar = robot.getDevice("range_back")
self.back_lidar.enable(timestep)
self.left_lidar = robot.getDevice("range_left")
self.left_lidar.enable(timestep)
self.right_lidar = robot.getDevice("range_right")
self.right_lidar.enable(timestep)

# Initialize camera
self.camera = robot.getDevice("camera")
self.camera.enable(timestep)
self.camera_width = self.camera.getWidth()
self.camera_height = self.camera.getHeight()
```

Şekil 4.4. Model üzerindeki sensörlerin aktif edilmesi

Aynı zamanda model üzerinde motorları aktif edilmesi ve Bölüm 3.2.2'de bahsedilen motorların dönüş yönlerinin ayarlanması işlemini Şekil 4.5'te göstermektedir.

```
# Initialize motors
self.m1_motor = robot.getDevice("m1_motor")
self.m1_motor.setPosition(float('inf'))
self.m1_motor.setVelocity(-1)
self.m2_motor = robot.getDevice("m2_motor")
self.m2_motor.setPosition(float('inf'))
self.m2_motor.setVelocity[1]
self.m3_motor = robot.getDevice("m3_motor")
self.m3_motor.setPosition(float('inf'))
self.m3_motor.setVelocity(-1)
self.m4_motor = robot.getDevice("m4_motor")
self.m4_motor.setPosition(float('inf'))
self.m4_motor.setVelocity(1)
self.motors = [self.m1_motor, self.m2_motor, self.m3_motor, self.m4_motor]
```

Şekil 4.5. Model üzerindeki motorların aktif edilmesi ve ayarlanması

Yapılan bu işlemlerden sonra model üzerindeki Lidar, GPS, IMU ve kamera sensörlerinde veriler okunabilmektedir. Aynı motorlara verilen güç değerleri ile çalışabilecek durumdadırlar.

4.3 Oransal İntegral Türevsel (PID) Kontrol Algoritmasının Gerçekleştirilmesi

Bölüm 3.4'te bahsedilen PID kontrol algoritmasını simülasyonda üzerindeki modele gerçekleştirilmesine deðinilmektedir. PID kontrol algoritmasını yükselme, belirli bir yükseklikte sabit kalma, belirli bir noktaya ilerleme gibi hareket sistemlerinde kullanılmaktadır. PID kontrol yapısı birçok noktada kullanılacağı için ayrı bir fonksiyon olarak gerçekleştirilmiştir.

```
def pid(self, dt, desired_vx, desired_vy, desired_yaw_rate, desired_altitude, actual_roll, actual_pitch, actual_yaw_rate,
       actual_altitude, actual_vx, actual_vy):

    gains = {"kp_att_y": 1, "kd_att_y": 0.5, "kp_att_rp": 0.5, "kd_att_rp": 0.1,
             "kp_vel_xy": 2, "kd_vel_xy": 0.5, "kp_z": 10, "ki_z": 5, "kd_z": 5}
```

Şekil 4.6. PID kontrol algoritma fonksiyonu

Bu fonksiyonda toplam 11 tane parametre alıyordur. Bunlar: dt (delta time-kontrol döngüleri arası zaman farkı), “desired_vx” ve “desired_vy” (X ve Y eksenlerinde istenen hızlar), “desired_yaw_rate” (istenen sapma hızı), “desired_altitude” (hedef yükseklik), “actual_roll” ve “actual_pitch” (gerçek yuvarlanma ve eğilme açıları), “actual_yaw_rate” (gerçek sapma hızı), “actual_altitude” (mevcut yükseklik), “actual_vx” ve “actual_vy” (gerçek X ve Y hızları) şeklindedir.

Fonksiyon içerisinde önce kazanç parametreleri tanımlanmaktadır (kp_att_y, kd_att_y, kp_att_rp, kd_att_rp, kp_vel_xy, kd_vel_xy, kp_z, ki_z, kd_z). Bu parametrelerden yükseklik, konum ve hız için oluşturulacak PID için hazırlanmış kazanç değerleridir. Bunlar nasıl hesaplandığını Bölüm 3.4.4'te gösterilmektedir. Buradaki kazanç değerleri deneysel olarak bulunmuştur.

```
vx_error = desired_vx - actual_vx
vx_deriv = (vx_error - self.past_vx_error) / dt
vy_error = desired_vy - actual_vy
vy_deriv = (vy_error - self.past_vy_error) / dt
desired_pitch = gains["kp_vel_xy"] * np.clip(vx_error, -1, 1) + gains["kd_vel_xy"] * vx_deriv
desired_roll = -gains["kp_vel_xy"] * np.clip(vy_error, -1, 1) - gains["kd_vel_xy"] * vy_deriv
self.past_vx_error = vx_error
self.past_vy_error = vy_error
```

Şekil 4.7. PID kontrol algoritması hızla bağlı roll ve pitch açısı hesaplama

Şekil 4.7'deki kod bölümünde, model'in hız kontrolü kısmını sağlamaktadır. X ve Y eksenlerindeki hız hataları hesaplanmaktadır. “vx_error” istenen X hızı ile gerçek X hızı arasındaki fark, “vy_error” ise istenen Y hızı ile gerçek Y hızı arasındaki farklıdır. Ardından bu hataların türev değerleri (vx_deriv ve vy_deriv) hesaplanmaktadır.

Bu işlem mevcut hata ile bir önceki döngüdeki hata arasındaki farkın zaman farkına (dt) bölünmesiyle yapılır ve hatanın değişim hızını vermektedir. Daha sonra bu hata ve türev değerleri kullanılarak istenen pitch ve roll açıları hesaplanır. X eksenin hız hatası (vx_error) istenen pitch açısını belirlenmektedir. Bu hata -1 ile +1 arasında sınırlandırılarak için “clip” metodu kullanılmaktadır. kp_vel_xy kazancı ile çarpılarak ve türev terimi ($kd_vel_xy * vx_deriv$) eklenerek PD kontrolü oluşturulmaktadır. Benzer şekilde Y eksenin hız hatası (vy_error) istenen roll açısını belirler, ancak burada negatif işaret kullanılır çünkü model'in koordinat sisteminde Y eksenindeki pozitif hız için negatif roll açısı gereklidir. Son olarak, bir sonraki döngüde türev hesaplaması yapabilmek için mevcut hata değerleri “ $past_vx_error$ ” ve “ $past_vy_error$ ” değişkenlerinde saklanır.

```

alt_error = desired_altitude - actual_altitude
alt_deriv = (alt_error - self.past_alt_error) / dt
self.altitude_integrator += alt_error * dt
alt_command = gains["kp_z"] * alt_error + gains["kd_z"] * alt_deriv + \
|   gains["ki_z"] * np.clip(self.altitude_integrator, -2, 2) + 48
self.past_alt_error = alt_error

```

Şekil 4.8. PID kontrol algoritması yüksekliği hesaplama

Şekil 4.8'deki kod bölümü, model'in yükseklik kontrolü (altitude control) kısmını gerçekleştirmekte ve Şekil 4.7'deki sistem kıyasla tam bir PID kontrolü uygulamaktadır. İlk olarak yükseklik hatası (alt_error) hesaplanır; bu, istenen yükseklik ile gerçek yükseklik arasındaki farktır. Ardından bu hatanın türev değeri (alt_deriv) hesaplanır; mevcut yükseklik hatası ile bir önceki döngüdeki yükseklik hatası arasındaki farkın zaman farkına (dt) bölünmesiyle elde edilir ve yükseklik hatasının değişim hızını gösterir. Integral terimi için $altitude_integrator$ değişkenine mevcut hata ile zaman farkının çarpımı ($alt_error * dt$) eklenir; bu işlem zamanla biriken hataları kompanse etmek için yapılır ve kalıcı durum hatalarını ortadan kaldırır.

```

pitch_error = desired_pitch - actual_pitch
pitch_deriv = (pitch_error - self.past_pitch_error) / dt
roll_error = desired_roll - actual_roll
roll_deriv = (roll_error - self.past_roll_error) / dt
yaw_rate_error = desired_yaw_rate - actual_yaw_rate
roll_command = gains["kp_att_rp"] * np.clip(roll_error, -1, 1) + gains["kd_att_rp"] * roll_deriv
pitch_command = -gains["kp_att_rp"] * np.clip(pitch_error, -1, 1) - gains["kd_att_rp"] * pitch_deriv
yaw_command = gains["kp_att_y"] * np.clip(yaw_rate_error, -1, 1)
self.past_pitch_error = pitch_error
self.past_roll_error = roll_error

```

Şekil 4.9. PID kontrol algoritması duruş hesaplaması

Şekil 4.9'deki kod bölümü, model'in duruş kontrolü (attitude control) kısmını gerçekleştirmekte ve model'in açısal pozisyonunu kontrol etmektedir. İlk olarak pitch hatası (pitch_error) hesaplanır; bu, hız kontrolü bölümünden gelen istenen pitch açısı ile sensörlerden okunan gerçek pitch açısı arasındaki farktır. Pitch hatasının türev değeri (pitch_deriv) mevcut hata ile bir önceki döngüdeki hata arasındaki farkın zaman farkına bölünmesiyle hesaplanır ve pitch hatasının değişim hızını verir. Benzer şekilde roll hatası (roll_error) istenen roll açısı ile gerçek roll açısı arasındaki fark olarak hesaplanır ve roll_deriv ile türev değeri elde edilir. Yaw kontrolü için yaw_rate_error hesaplanır; bu, istenen sapma hızı ile gerçek sapma hızı arasındaki farktır. Roll komutu (roll_command) PD kontrolü kullanılarak hesaplanır; roll hatası -1 ile +1 arasında tutmak için "clip" fonksiyonu kullanmaktadır. "kp_att_rp" kazancı ile çarpılır ve türev terimi (kd_att_rp * roll_deriv) eklenir. Pitch komutu (pitch_command) benzer şekilde PD kontrolü ile hesaplanır ancak negatif işaret kullanılır; bu, model'in koordinat sistemi ve motor konfigürasyonu nedeniyle gerekli olan bir dönüşümdür. Yaw komutu (yaw_command) sadece oransal kontrol kullanır; yaw hız hatası sınırlandırılarak kp_att_y kazancı ile çarpılır ve integral veya türev terimi kullanılmaz çünkü yaw kontrolü genellikle daha basit tutulur. Son olarak, bir sonraki döngüde türev hesaplaması yapabilmek için mevcut pitch ve roll hataları past_pitch_error ve past_roll_error değişkenlerinde saklanır.

```
m1 = alt_command - roll_command + pitch_command + yaw_command
m2 = alt_command - roll_command - pitch_command - yaw_command
m3 = alt_command + roll_command - pitch_command + yaw_command
m4 = alt_command + roll_command + pitch_command - yaw_command
```

Şekil 4.10. Motorların güçlerinin ayarlanması

Quadcopter'da motorlar genellikle X konfigürasyonunda veya + konfigürasyonunda yerleştirilir ve her motor farklı hareket bileşenlerine farklı şekilde katkıda bulunur. Yükseklik (alt_command) değeri tüm motorlara eşit olarak eklenir çünkü yükseklik kontrolü için tüm motorların aynı miktarda güç üretmesi gereklidir.

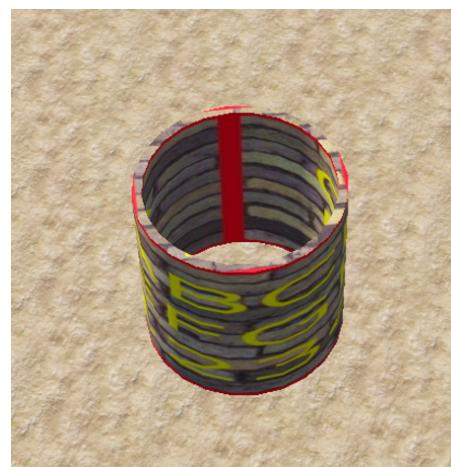
"roll_command" için M1 ve M2 motorlarından çıkarıldığında M3 ve M4 motorlarına eklenir. Bu quadcopter'in sol-sağ ekseniinde yuvarlanması için sol taraftaki motorların güçlerinin azaltılıp sağ taraftaki motorların güçlerinin artırılması anlamına gelir. Bölüm 3.2.2'de bu olay açıklanmış Şekil 3.3'te gösterilmiştir.

“pitch_command” için M1 ve M3 motorlarına eklenirken M2 ve M4 motorlarından çıkarılır; bu quadcopter’ın ileri-geri ekseninde eğilmesi için arka motorların güçlerinin artırılıp ön motorların güçlerinin azaltılması demektir. Şekilde 3.4’tে bu olay gösterilmiştir.

“yaw_command” için ise M1 ve M3 motorlarına eklenirken M2 ve M4 motorlarından çıkarılır. Bu quadcopter’ın dikey ekseninde dönmesi için çapraz motorların zıt yönlerde güç üretmesi gerektiği içindir çünkü quadcopter motorları çiftler halinde zıt yönlerde döner ve bu sayede tork dengeleme sağlanır. Şekilde 3.5’de gösterilmiştir. Bu matematik formülasyon, her motorun quadcopter’ın altı serbestlik dereceli hareketine (yükarı-aşağı, ileri-geri, sol-sağ, roll, pitch, yaw) nasıl katkıda bulunacağını belirleyen fiziksel dinamiklerin matematiksel ifadesidir.

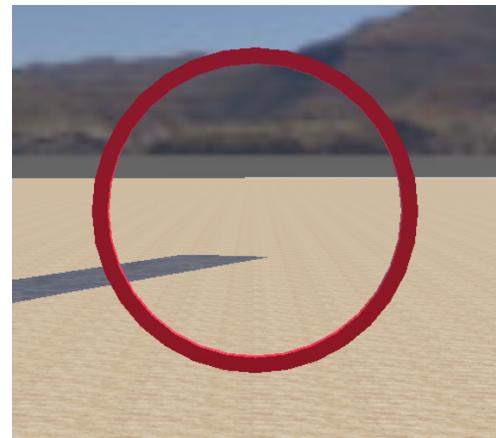
4.4. Simülasyon Ortamında Parkurun Oluşturulması

Parkurun kontrol noktaları (checkpoint) için Webots üzerinde bulunan “SolidPide” nesnesi kullanılmıştır. Şekil 4.11’da gözükmektedir. Nesnenin bu hali kontrol noktası olabilecek halde olmadığından dolayı bu nesne üzerinde bazı değişiklikler yapılmıştır.



Şekil 4.11. Webots üzerindeki SolidPide nesnesi

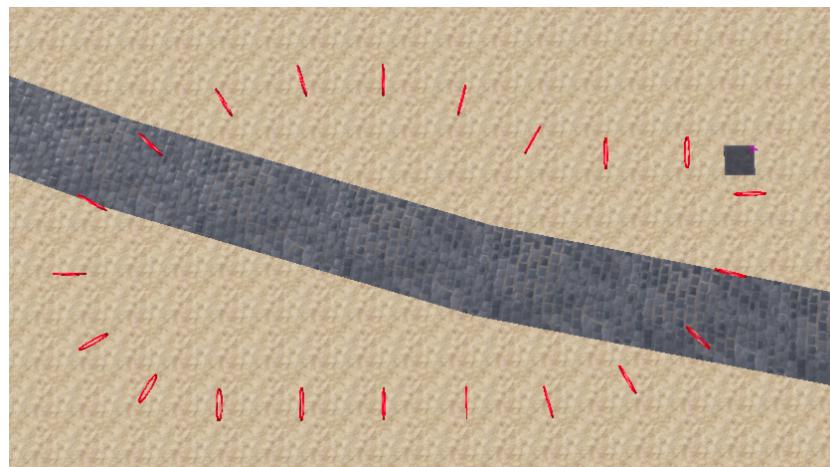
Bu nesne üzerinde ilk olarak yüksekliğini 0,05 olarak, yarı çapını 0,5 olarak ayarlanmıştır. Bu işlemlerden sonra bir kontrol noktası olarak kullanılabilcektir son olarak daha belirgin durması için renginin kırmızı olarak ayarlanmasıından sonra Şekil 4.12’deki halini almıştır.



Şekil 4.12. Webots üzerinde bir kontrol noktası

Kontrol noktasının nasıl olacağını belirlenmiştir. Parkur hazırlamak için bir başlangıç ve bitiş noktası belirleyip bu iki nokta arasına belirli bir düzen ile kontrol noktalarını yerleştirilip parkur tamamlanmıştır.

Bu proje kullanılmak için hazırlanılacak parkurun başlangıç ve bitiş noktaları aynı yer olarak belirlenmiştir. Bir döngü şeklinde olacak şekilde kontrol noktalarını yerleştirilmiştir.



Şekil 4.13. Webots üzerinde hazırlanmış parkur

Hazırlanan parkur üzerinde hem sağa hem sola dönüşler, düz yollar bulunmaktadır. Bunlar sayesinde modelin kabiliyetini test etmek için yeterli bir parkur oluşturmuştur.

4.5. Kontrol Noktalarının Algılanması

Yol planlama algoritmasında kullanmak üzerine kontrol noktalarının konumlarının algılanması gerekmektedir. Bu işlemi yapabilmek için iki farklı yol vardır. Birincisi görüntü işleme yardımıyla kontrol noktalarını algılayarak turu tamamlamak, ikincisi kullanıcı tarafından ilk turun manuel olarak tamamlanması sonucunda sensörler sayesinde kontrol noktalarının yerine algılamasıdır. Bu projede ikinci yöntem olan ilk turun kullanıcı manuel olarak tamamlanması ile sensörleri kontrol noktasını algılaması yapılmaktadır.

Kullanıcı manuel bir şekilde kontrol işlemini gerçekleştirmesi için bir sürüş mekanığı kodlanması gerekmektedir. Manuel kontrol için “W” ve “S” tuşları yükselme ve açılma, “A” ve “D” tuşları sağa sola roll işlemleri, “İleri Ok” ve “Geri Ok” ileri geri hareketi son olarak “Sağa Ok” ve “Sola Ok” sağa sola yaw işlemleri için seçilmiştir.

```

if key == ord('W') and (current_time - last_altitude_change_time) >= ALTITUDE_CHANGE_INTERVAL:
    drone.FLYING_ATTITUDE += ALTITUDE_CHANGE_STEP
    last_altitude_change_time = current_time
elif key == ord('S') and (current_time - last_altitude_change_time) >= ALTITUDE_CHANGE_INTERVAL:
    drone.FLYING_ATTITUDE -= ALTITUDE_CHANGE_STEP
    last_altitude_change_time = current_time
elif key == ord('A'):
    sideways_desired = drone.MAX_VELOCITY
elif key == ord('D'):
    sideways_desired = -drone.MAX_VELOCITY
elif key == Keyboard.LEFT:
    yaw_desired = 0.5
elif key == Keyboard.RIGHT:
    yaw_desired = -0.5
elif key == Keyboard.UP:
    forward_desired = drone.MAX_VELOCITY
elif key == Keyboard.DOWN:
    forward_desired = -drone.MAX_VELOCITY

```

Şekil 4.14. Manuel kontrol kodu

Şekil 4.14'te manuel kullanıcı için belirlediğimiz tuş kombinasyonun kod üzerinde nasıl bir hal aldığı gösterilmiştir. Bu kodda, her tuşun yapması gerektiği işlem için mesela yaw hareketi için istenilen değere ekleme ve çıkarma işlemi gibi tüm tuşların işlevleri tanımlanmıştır.

Bölüm 4.3'te bahsedilen model üzerindeki PID kontrol algoritmasını kullanarak bir tuşa basıldığı zaman yapılacak işlemin model üzerinde uygulamasını sağlayan yapıyı Şekil 4.15'te görülmektedir.

```

roll, pitch, yaw = drone.imu.getRollPitchYaw()
yaw_rate = drone.gyro.getValues()[2]
altitude = drone.gps.getValues()[2]

motor_power = drone.pid_controller.pid(timestep/1000.0, forward_desired, sideways_desired,
                                         yaw_desired, target_altitude,
                                         roll, pitch, yaw_rate,
                                         altitude, 0, 0)

drone.m1_motor.setVelocity(-motor_power[0])
drone.m2_motor.setVelocity(motor_power[1])
drone.m3_motor.setVelocity(-motor_power[2])
drone.m4_motor.setVelocity(motor_power[3])

```

Şekil 4.15. Manuel kontrol yapısı için PID ve motor güçlerinin ayarlanması

Bu işlemler bize simülasyon üzerinde modelin kontrol yetkisi tanımlıdır. Bu kontrol ile bir tur tamamladıktan sonra kontrol noktalarını algılaması gerekmektedir. Kontrol noktasını algılayabilmek için model üzerinde Lidar ve GPS sensörlerini kullanacağız.

Model normal bir uçuş yaparken eğer bir kontrol noktasının içinden geçmiyorsa sağ ve sol Lidar'dan belirli bir değer üzerinde değer okumaktadır. Bu değer bizim eşik (threshold) değerimiz oluyor. Biz dronu kontrol ederken drone sürekli olarak bu iki Lidar'dan gelen değerleri kontrol ediyor ve bizim belirlediğimiz eşik değeri (Kontrol noktasının yarı çapı 0,5 olduğu için bu değer 0,5 olarak belirlenmiştir.) üzerinde altında bir veri geldiği zaman bir kontrol noktası içinden geçtiğini anlamakta ve tam olarak kontrol noktasını nerede geçtiğini hesaplamak için kontrol noktasının geçmeden ve geçtiğinden sonra kada veri toplamaya devam etmekte. Elinde olan bu verilerin en iyi ve anlaşılabilir olanını seçerek o konum bilgileri “checkpoints.json” dosyasının içine kaydettmektedir.

```

if left_distance < drone.LIDAR_THRESHOLD or right_distance < drone.LIDAR_THRESHOLD:
    x, y, z = drone.gps.getValues()
    roll, pitch, yaw = drone.imu.getRollPitchYaw()

    v_x = (x - drone.last_position[0]) / (drone.robot.getTime() - drone.last_time) if hasattr(drone, 'last_position') else 0
    v_y = (y - drone.last_position[1]) / (drone.robot.getTime() - drone.last_time) if hasattr(drone, 'last_position') else 0
    v_z = (z - drone.last_position[2]) / (drone.robot.getTime() - drone.last_time) if hasattr(drone, 'last_position') else 0

    side = "left" if left_distance < right_distance else "right"

    passage_data = {
        'position': {
            'x': x,
            'y': y,
            'z': z
        },
        'orientation': {
            'roll': roll,
            'pitch': pitch,
            'yaw': yaw
        },
        'velocity': {
            'v_x': v_x,
            'v_y': v_y,
            'v_z': v_z
        },
        'passage_info': {
            'side': side,
            'left_lidar': left_distance,
            'right_lidar': right_distance,
            'timestamp': drone.robot.getTime()
        }
    }

```

Şekil 4.16. Kontrol noktası algılamasın için temel yapısı

Şekil 4.16'de bahsettiğimiz sağ ve sol Lidar'dan gelen veriler eşik değerler altında olduğu sürece drone üzerinden konum hız ve roll, pitch, yaw değerlerini kaydetmektedir. Bununla en iyi ve anlaşılabilen kontrol noktasını bulmak için kontrol edecktir.

```
best_point = passage_filter.add_point(passage_data)
if best_point:

    if hasattr(drone, 'last_checkpoint_position'):
        last_x, last_y, last_z = drone.last_checkpoint_position
        distance = sqrt((x - last_x)**2 + (y - last_y)**2 + (z - last_z)**2)
        if distance < 1.0:
            return None

    checkpoint_id = drone.checkpoint_manager.add_checkpoint(
        best_point['position'],
        best_point,
        orientation=(roll, pitch, yaw)
    )
```

Şekil 4.17. Kontrol noktasını algılama ve kaydetme

Şekil 4.17'de görüldüğü gibi en iyi kontrol noktası seçildikten sonra “checkpoint_manager” üzerinden “checkpoints.json” dosyasına kaydetme işlemi yapılmaktadır.

```
"1": {
    "id": 1,
    "position": {
        "x": -2.153587485760401,
        "y": 5.714737369617939e-06,
        "z": 3.034899865286885
    },
    "connections": [...],
    "passage_history": [
        {
            "timestamp": "2025-05-22T02:25:30.471983",
            "approach_side": "right",
            "lidar_readings": {...},
            "orientation": {...},
            "velocity": {...}
        }
    ]
},
```

Şekil 4.18. Kaydedilen kontrol nokta verisi

4.6. Yol Planlama Algoritmasının Gerçekleştirilmesi

4.6.1 A* Algoritması

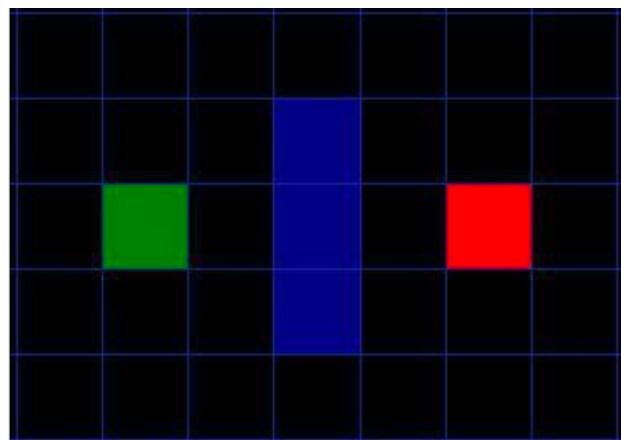
A* algoritması, bilgisayar bilimlerinde yaygın olarak kullanılan bilgilendirilmiş bir arama algoritmasıdır. Özellikle yol bulma ve grafik geçişi gibi problemler için etkilidir. Algoritma, başlangıç noktasından hedef noktasına giden en düşük maliyetli yolu bulmayı amaçlar. Bunu yaparken, iki temel değeri birleştiren bir sezgisel (heuristic) fonksiyon kullanır: başlangıç noktasından mevcut düğüme olan gerçek maliyet ve mevcut düğümden hedefe olan tahmini maliyet. A* algoritması, her adımda değeri en düşük olan düğümü seçerek ilerler.

$$h(n) = \text{Tahmini Maliyet}$$

$$g(n) = \text{Gerçek Maliyet}$$

$$f(n) = h(n) + g(n)$$

Formül 4.1. A* algoritması formülsel gösterimi

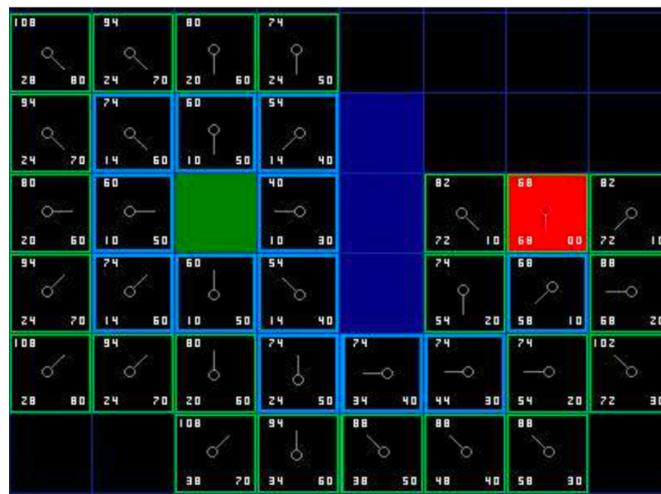


Şekil 4.19. A* algoritması için hazırlanan problem

Şekil 4.19'deki problem üzerinde A* algoritması, bu ızgara üzerindeki yol bulma problemini çözmek için öncelikle başlangıç noktası olan yeşil kareyi ele alır. Bu noktadan itibaren, ulaşabileceğiniz komşu kareleri değerlendirmeye başlar. Değerlendirme sürecinde, her bir komşu için başlangıçtan o noktaya olan gerçek maliyeti (genellikle her adım için 1 birim) ve o noktadan hedefe olan tahmini maliyeti (örneğin, Manhattan mesafesi gibi bir sezgisel ile hesaplanan) birleştirerek bir toplam maliyet (f değeri) hesaplar. Amaç, bu f değeri en düşük olan komşuyu bir sonraki adım olarak seçmektir. (Tahir, 2025)

Algoritma ilerledikçe, açık olan (henüz ziyaret edilmemiş veya en iyi yol olmadığı belirlenmemiş) komşuları bir kenarda tutar ve her adımda bu açık kümeden en düşük f

değerine sahip olanı seçer. Resimde görünen mavi engel, algoritmanın doğrudan yeşilden kırmızıya gitmesini engeller. Bu durumda A*, engelin etrafındaki boş kareleri de değerlendirecek ve bunlara ulaşmanın maliyetini hesaplayacaktır. Sezgisel fonksiyon sayesinde, hedef noktaya daha yakın olan karelere öncelik verilir, böylece algoritma gereksiz yere uzak noktalara gitmekten kaçınır. (Tahir, 2025)



Şekil 4.20. A* algoritması ile çözülmüş problem

Sonuç olarak Şekil 4.20'de, A* algoritması, başlangıç noktasından başlayarak, engeli dikkate alarak ve her adımda en umut vadeden (düşük f değerli) komşuyu seçerek hedef noktasına ulaşmaya çalışır. Hedefe ulaştığında, takip ettiği yol, başlangıçtan hedefe olan en düşük maliyetli yol olarak kabul edilir. Bu süreçte, algoritma hem başlangıçtan gelinen maliyeti hem de hedefe kalan tahmini maliyeti dengeli bir şekilde kullanarak etkili bir arama gerçekleştirir. (Tahir, 2025)

4.6.2 Simülasyon Üzerinde A* Algoritması

Simülasyonda üzerinde A* algoritması ile yol bulma işlemi yapmamız için kullanıcının manuel olarak tamamladığı turda kontrol noktalarını tespit edip “checkpoints.json” içine kaydettiğimiz dosyadan tekrar okuyup A* algoritması için hazır hale getirmemiz gerekmektedir. Şekilde 4.21'da bu işlem gösterilmektedir. Dosya içerisindeki okunulan kontrol noktaları A* algoritması için kullanılabilen bir yapı olan “CheckpointNode” sınıfının bir nesnesi haline getiril ve bu düğümler arasındaki bağlar ayarlanır.

```

checkpoints = []
for cp_id, cp in drone.checkpoint_manager.get_all_checkpoints().items():
    checkpoint = CheckpointNode(
        int(cp_id),
        (cp['position']['x'], cp['position']['y'], cp['position']['z']),
        (
            cp['passage_history'][-1]['roll'] if cp['passage_history'] else 0.0,
            cp['passage_history'][-1]['pitch'] if cp['passage_history'] else 0.0,
            cp['passage_history'][-1]['yaw'] if cp['passage_history'] else 0.0
        )
    )
    if cp['passage_history']:
        last_passage = cp['passage_history'][-1]
        checkpoint.potential_field['lidar_left'] = last_passage['lidar_readings']['left']
        checkpoint.potential_field['lidar_right'] = last_passage['lidar_readings']['right']
    checkpoints.append(checkpoint)

if not checkpoints:
    print("Checkpoint bulunamadı!")
    return

create_checkpoint_connections(checkpoints)

```

Şekil 4.21. Kaydedilen kontrol noktaların okunması ve dönüşümü

Bu işlemlerin gerçekleştirilmesinden sonra A* algoritmasını gerçekleştirmek için gerekli fonksiyonun çalışması gerekmektedir. Şekil 4.22'da bu fonksiyon nasıl çalıştığı gösterilmektedir.

A* algoritması aslında gerçek A* algoritmasından çok basitleştirilmiş bir sıralı yol izleme sistemidir. Algoritma başlangıç ve hedef kontrol noktalarını numaralına bulur, sonra kontrol noktalarını numara sırasına göre dizilmiş listede başlangıçtan hedefe kadar tek tek ilerler. Her adımda mevcut kontrol noktasını yola ekler, sonra bir sonraki checkpoint ile (varsayımsız) iki sonraki kontrol noktasını da dikkate alarak optimal ara nokta hesaplar. Eğer keskin bir dönüş varsa model daha yumuşak geçiş yapması için orijinal checkpoint pozisyonundan hafifçe kaydırılmış bir ara nokta oluşturur. Algoritma basit bir şekilde numara sırasını takip ederek kontrol noktasından kontrol noktasına geçer ve yolu tamamlar.

```

drone.route_recorder.start_recording(drone.current_lap)
lap_start_time = drone.robot.getTime()

path = find_path_between_checkpoints(first_checkpoint.id, last_checkpoint.id, checkpoints)
if path is None:
    print("Hata: Checkpoint'ler arasında yol bulunamadı!")
    return
print("Normal rota kullanılıyor...")

for i, checkpoint in enumerate(path):
    current_pos = drone.gps.getValues()

    if is_in_potential_field(current_pos, checkpoint):
        continue

    target_pos = (checkpoint.position[0], checkpoint.position[1], avg_height)

    if not drone.goto(target_pos):
        print(f"Checkpoint {checkpoint.id}'e ulaşamadı!")
        return

    drone.current_checkpoint = checkpoint.id
    print(f"\nTır: {(drone.current_lap)/(drone.max_laps)}")
    print(f"Checkpoint: {(drone.current_checkpoint)/(drone.total_checkpoints)}")

    checkpoint_start_time = drone.robot.getTime()
    stabilization_time = float(checkpoint.stabilization_time)

    while drone.robot.getTime() - checkpoint_start_time < stabilization_time:
        drone.robot.step(drone.timestep)
        drone.route_recorder.record_point(drone)

    current_time = drone.robot.getTime()
    lap_time = current_time - lap_start_time
    drone.lap_times.append(lap_time)

```

Şekil 4.22. A* algoritması kullanılması

A* algoritmasından çıkan sonuç tekrar kontrol edilir. Bu kontrol gidilecek rotanın kontrol noktası içinden geçip geçmediğini kontrol etmektedir. Bu Fonksiyon model'in belirli bir kontrol noktasına etkili olduğu alan içinde bulunup bulunmadığını kontrol eder. Fonksiyon model'in mevcut pozisyonunu (x , y , z) ile kontrol noktasına pozisyonunu karşılaştırır ve her eksen için ayrı ayrı tolerans kontrolü yapar. X eksen, Y ekseni ve Z ekseni için değerlerini kullanarak model'in checkpoint merkezinden ne kadar uzakta olabileceğini belirler. Eğer drone bu üç eksende de belirlenen tolerans aralıkları içindeyse (yani kontrol noktasına yeterince yakınsa) fonksiyon True döndürür, aksi halde False döndürür. Bu mekanizma model'in hangi kontrol noktasına kontrolü altında olduğunu anlamak ve o kontrol noktasına özgü navigasyon parametrelerini (hız, stabilizasyon süresi gibi) uygulamak için kullanılır.

Bu fonksiyonlardan çıkan konum değeri Bölüm 4.2'da bahsedilen "Octopus" sınıfının içinde bulunan "goto" fonksiyonu ile o konuma gidecektir. "goto" fonksiyonu içerisinde PID kontrolcüsü barındıran bir fonksiyondur. Bunun sayesinde Bölüm 3.4'te gösterilen işlemler ile ilerleme sağlamaktadır.

4.7. Tur Verilerinin Toplanması

Bu proje kapsamında geliştirilen otonom quadcopter sisteminin performansını değerlendirmek ve uçuş optimizasyonları için gerekli verileri elde etmek amacıyla, her bir tur tamamlandığında çeşitli sensörlerden veri toplanması planlanmaktadır. Toplanacak temel veriler ve bu verilerin önemi aşağıda detaylandırılmıştır.

LIDAR verileri, quadcopter'in çevresindeki engelleri algılama ve haritalama yeteneğini değerlendirmek için LIDAR sensöründen elde edilen mesafe bilgilerini içerir. Bu veriler, otonom navigasyon algoritmalarının doğruluğunu ve engellerden kaçınma performansını analiz etmek için kullanılacaktır.

GPS verileri, quadcopter'in parkur üzerindeki konumunu yüksek doğrulukla belirlemek için GPS sensöründen elde edilen enlem, boylam ve yükseklik bilgilerini içerir. Bu veriler, yol planlama algoritmalarının etkinliğini ve quadcopter'in belirlenen rotayı ne kadar hassas bir şekilde takip edebildiğini değerlendirmek için kullanılacaktır.

IMU verileri, quadcopter'in dönüş hızları ve ivmelenmeleri hakkında bilgi edinmek için IMU sensöründen elde edilen açısal hız ve ivme değerlerini içerir. Bu veriler, quadcopter'in dengesini koruma ve istenen manevraları gerçekleştirmeye yeteneğini analiz etmek için kullanılacaktır.

Bu verilerin toplanması ve analizi, otonom quadcopter sisteminin geliştirilmesi ve iyileştirilmesi sürecinde önemli bir rol oynayacaktır. Elde edilen sonuçlar, sistemin güçlü ve zayıf yönlerini belirlememize yardımcı olacak ve gelecekteki çalışmalara yön verecektir.

4.7.1 Tur Verilerinin Toplanma İşlemi

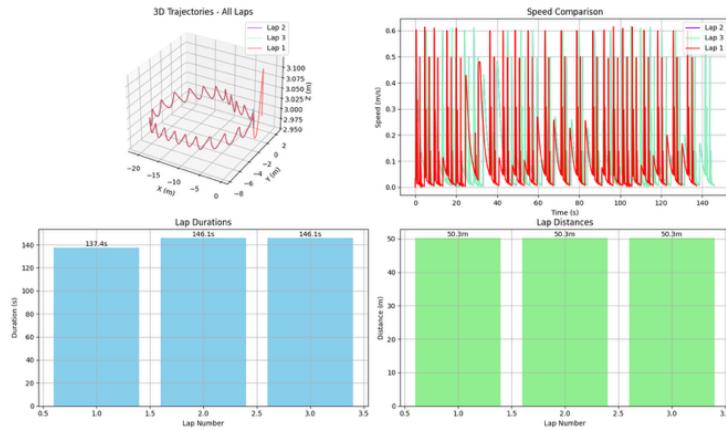
Simülasyon üzerinde verileri toplama işlemi “RouteRecorder” sınıfı üzerinden yapılmaktadır. Bu sınıfı Bölüm 4.2’da anlatılan “Octopus” sınıfın içerisinde bulunan “goto” fonksiyonu ve otonom olarak hareket etmesini sağlayan fonksiyon içerisinde bulunan komutla birlikte kayıt işlemek yapmaktadır. Şekil 4.23’te bir noktada kaydedilen veriyi göstermektedir.

```
{
  "timestamp": 0.02400000000000091,
  "position": [
    -6.116379348439192e-06,
    -8.19442727434181e-06,
    3.0140209887703513
  ],
  "orientation": [...],
  "velocity": [...],
  "lidar_readings": {...},
  "checkpoint_id": 0,
  "is_checkpoint": true
},
```

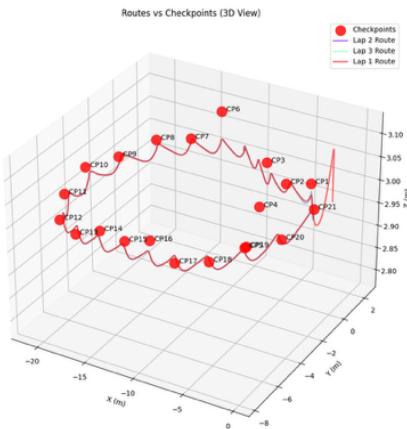
Şekil 4.23. Bir turun herhangi bir zamanında kaydedilen veri

4.7.2 Verilerin Görselleştirilmesi

Turlardan toplanan verileri ve kontrol noktalarını tespit edilirken toplanan verileri simülasyondan bağımsız bir yapı ile görselleştirilmiştir. Bu yapı sayesinde Tur verisi ile kontrol noktaları arasında farkları, sadece tur verilerini görselleştirmek için yapılmıştır. Şekilde 4.24’te tur verilerinin görselleştirildiği görülmektedir. Bu şekilde tur içinde izlenen rota, hız değişimleri, tamamlanma süresi ve toplam alınan mesafe gösterilmiştir. Şekilde 4.25’te ise turlarda izlenilen rota ile kontrol noktalarının karşılaştırılması görselleştirilmesidir.



Şekil 4.24. Tur verilerinin görselleştirilmiş hali

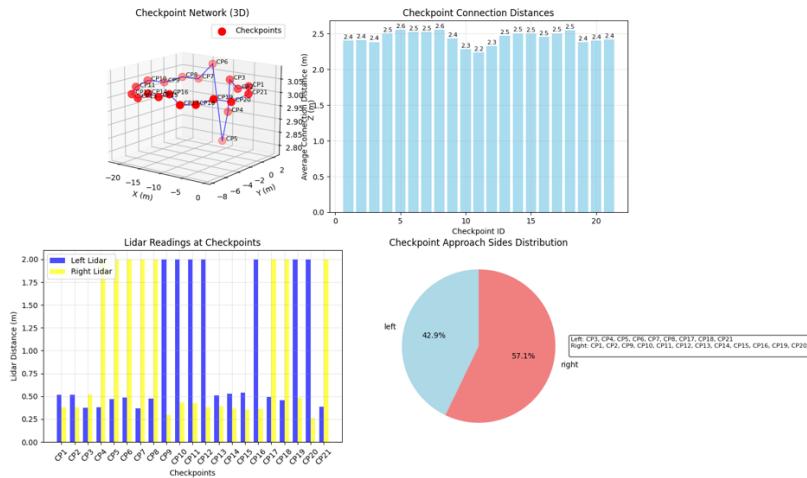


Şekil 4.25. Tur verisi ile kontrol noktaların karşılaştırılması

5. SONUÇLAR VE ÖNERİLER

5.1 Sonuçlar

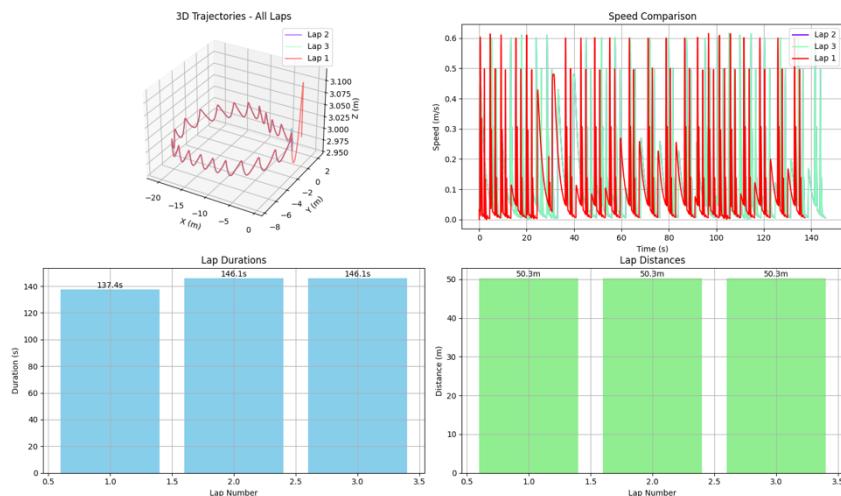
Oluşturulan parkur üzerinde model'in testleri gerçekleştirilmiş ve verileri toplanmıştır. Bölüm 4.7.2'de bahsedilen veri görselleştirme işlemleri ile görselleştirilmiş ve bunun üzerine aşağıda yargılaraya varılmıştır.



Şekil 4.26. Kontrol noktası algılamak için yapılan tur verisi

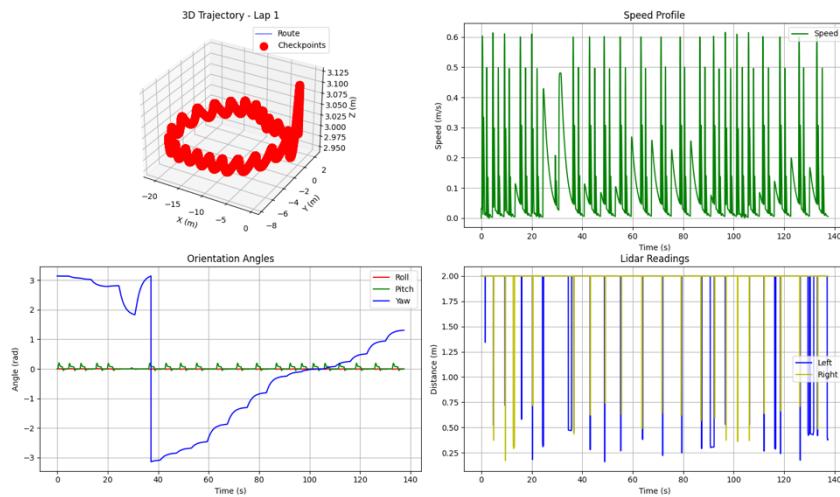
Kullanıcı tarafından kontrol noktalarını algılamak için yapılan turda bazı kontrol noktalarından geçerken modelin yükseklik verilerinde turun başlangıç kısmında dalgalanma olduğu gözlenmektedir. Bunun yanında her kontrol noktası arasında 2,2 ile 2,6 arasında bir değer bulduğu görülmektedir. Bu test için hazırlanan parkurda her kontrol noktası arasında 2,4 ile 2,5 arası mesafe bırakılmış model bu noktaları 2,2 ile 2,6 arasında değerler olarak bulunmuştur bu da sistemin %75 oranında bir doğruluk oranı olduğunu göstermektedir.

Modelin otomatik olarak yaptığı 3 turdan sonra alınan veriler kendi arasında analiz edilmesi üzere Şekil 4.27'de gösterilmiştir. Bu verilerden yola çıkararak tüm turlarda alınan yolun aynı olmasına rağmen ilk turun tamamlanma süresinin ikinci ve üçüncü tura kıyasla daha az olduğu gözlemlenmiştir. Hız zaman grafiğinde ise tüm turlarda stabil bir hız yakalanamamış ve sürekli olarak dalgalı bir yapıda olduğu gözlemlenmiştir.

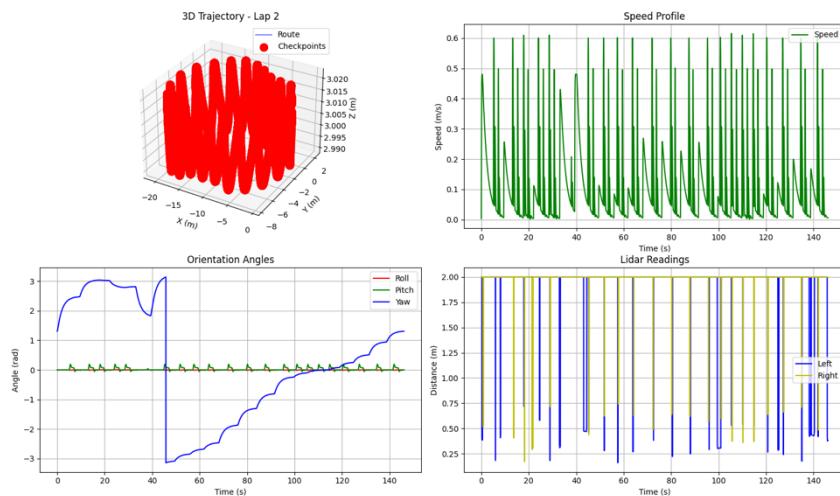


Şekil 4.27. Tüm tur verilerinin karşılaştırılması

İlk turun ve sonraki turların parkuru bitirme süreleri arasındaki farkın sebebinin öğrenmek için Şekil 4.28'de sadece ilk turun verileri görselleştirilmiş, Şekil 4.29'da sadece ikinci turun verileri görselleştirilmiştir. Buna ilk sebep olabilecek olayın hız zaman grafiğindeki 0 – 40 saniyeler arasındaki farklılıklardır. İlk turun başlangıcında ikinci tura kıyasla daha hızlı bir şekilde başladığı görülmektedir. Bu duruma ikinci sebep olabilecek olayda yaw açısı ve zaman grafiğinin 0 – 40 saniyeleri arasındaki farklılıklardır. İlk turda yaw açısı daha büyük bir değerden başlaması sonra stabil bir değere gelmesi, ikinci turda ise bunun tam tersi olarak yaw açısının düşük bir değerden gelerek stabil olduğu göstermektedir. Şekil 4.27'da görülen zaman farkının bu iki koşul olarak bulunmuştur.

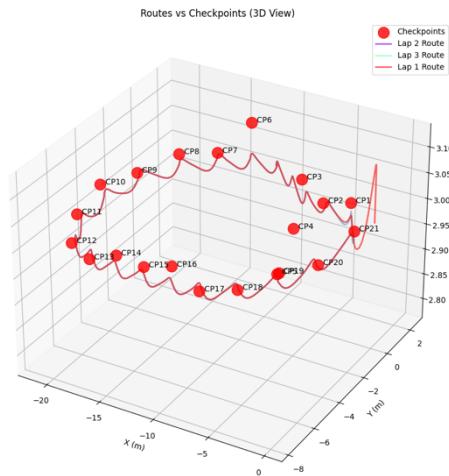


Şekil 4.28. İlk turun verilerinin görselleştirilmesi



Şekil 4.29. İkinci turun verilinin görselleştirilmesi

Son olarak kontrol noktalarının tespit edildiği konumlar ile yapılan turların Şekil 4.30'da karşılaştırılmıştır. Bu karşılaştırma sonunda yol planlama algoritması turun başında dalgalanan yüksekliği stabil bir hale getirmiş olduğu gözlemlenmektedir. Aynı zamanda bazı kontrol noktalarında (3, 10, 16) x ve y konumlarında yaptıkları iyileştirmeler gözlemlenmiştir.



Şekil 4.30. Kontrol noktaları ile turların karşılaştırılması

Tüm bunlar sonucunda model parkuru bazı iyileştirme işlemlerle tamamladığını gözlemlenmiştir lakin hızda yaşanan dalgalanmalar dikkate alındığında bu iyileştirmelerin yeterli olmadığı sonucuna varılmıştır.

5.2 Öneriler

5.2.1 Kontrol Noktalarının Algılanması

Bu projede kontrol noktaların algılanması için kullanıcının manuel olarak bir tur tamamlaması gerekmektedir. Bunun yerine bir görüntü işleme sistemi eklenmesiyle kontrol noktaları görüntü işleme sonucu tespit edilmesi ve o noktalardan geçirilerek konumlarının algılanması sağlanabilir.

5.2.2 Yol planlama Algoritması

Bu projede yol planlama algoritması olarak A* algoritması kullanılmıştır. Turları daha hızlı bir şekilde tamamlanması sağlamak için genetik algoritma, bulanık mantık gibi algoritmalar ile bu testlerin tekrar yapılması ve gerekli veriler analizi sonucunda en iyi yol planlama algoritması seçilerek daha hızlı bir sistem yapılabilir.

5.2.3 Tur Optimizasyonu

Projede parkur birden fazla kere tamamlanmaktadır ve her turda gerekli veriler toplanmaktadır. Bu verileri kullanarak turları daha iyi optimize hale getirebilecek bir sistem tasarlanabilir ve bu sayede daha hızlı bir sistem ortaya konulmuş olacaktır.

5.2.4 Verilerin Görselleştirilmesi

Simülasyon ortamında model üzerinde yapılan testlerden toplanan veriler sadece testler bittikten sonra simülasyondan bağımsız bir sistem ile görselleştirilmektedir. Simülasyon anında anlık olarak verileri görerek analiz yapmanın daha sağlıklı olacağından dolayı simülasyona entegre olan bir veri görselleştirme sistemi model üzerinde analizlerin daha sağlam olacaktır.

KAYNAKLAR

Anonim, [Yıl Bilinmiyor], Quadcopter Dynamics, Simulation, and Control.

Baykar, S. Y., 2025, Otonom Sürüste Kullanılan Sensörler, Ders Notu.

Built In, 2021, What Is a Drone?, [online], [Ziyaret Tarihi: 2 Nisan 2025],
<https://builtin.com/drones>.

CARLA Takımı, [Yıl Bilinmiyor], CARLA introduction [online], [Ziyaret Tarihi: 23 Mayıs 2025], https://carla.readthedocs.io/en/latest/start_introduction/.

Chamola, 2021, Classification of UAV based on wings and rotors, [online], [Ziyaret Tarihi: 23 Mayıs 2025], https://www.researchgate.net/figure/Classification-of-UAV-based-on-wings-and-rotors_fig2_344592031.

Debricked, [Yıl Bilinmiyor], github-carla-simulator/carla-vs-github-osrf/gazebo-vs-github-cyberbotics/webots [online], [Ziyaret Tarihi: 23 Mayıs 2025],
<https://debricked.com/select/compare/github-carla-simulator/carla-vs-github-osrf/gazebo-vs-github-cyberbotics/webots>.

EngineersEscape, 2019, How a Quadcopter Works - Flight Mechanics, Components, & Sensors (2), [online], <https://www.youtube.com/watch?v=u-LfLW9Kgis>, [Ziyaret Tarihi: 20 Mart 2025].

Ferrovial, 2021, Drones, [online], <https://www.ferrovial.com/en/resources/drones/>, [Ziyaret Tarihi: 2 Nisan 2025].

Ford, M. S. ve Çakmakçı, M., 2025, Hareket ve Yol Planlama, Ders Notu.

Gezer, A., Turan, Ö., ve Baklacioğlu, T., 2024, "Parçalı Hücresel Genetik Algoritma ile İnsansız Hava Aracı Performansına Dayalı Yol Planlama", Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi.

Ghambari, S., Zarei, A., ve Rad, M. A., 2024, "UAV Path Planning Techniques: A Survey", Journal of Advanced Intelligent Systems.

Imperial War Museums, A Brief History of Drones, [online],
<https://www.iwm.org.uk/history/a-brief-history-of-drones>, [Ziyaret Tarihi: 2 Nisan 2025].

İncekara, H., 2022, "Farklı Çevre Şartlarında Meta-Sezgisel Yöntemlerle Quadrotor İHA Rota Optimizasyonu ve Gerçeklenmesi", Doktora Tezi.

Islam, M., Okasha, M. ve Idres, M. M., 2017a, Dynamics and control of quadcopter using linear model predictive control approach.

Islam, M., Okasha, M. ve Idres, M. M., 2017b, "Dynamics and Control of Quadcopter Using Linear Model Predictive Control Approach", Journal of Intelligent Control Systems.

- Johnson-Laird, N., 2016, Basic Physics of Drones, [online],
<https://www.youtube.com/watch?v=PkbkO3e0ev0>, [Ziyaret Tarihi: 20 Mart 2025].
- Khan, M., 2014, Quadcopter Flight Dynamics, International Journal of Scientific & Technology Research, 3 (8).
- MATLAB, 2018, Drone Simulation and Control, Part 1: Setting Up the Control Problem, [online], <https://www.youtube.com/watch?v=hGcGPUqB67Q>, [Ziyaret Tarihi: 20 Mart 2025].
- Morgan, R., 2016, Drone Theory 101: Part 1. The basics, and how an fpv quadcopter functions!, [online], https://www.youtube.com/watch?v=K05UwsiqZ_E, [Ziyaret Tarihi: 20 Mart 2025].
- Norris, D., 2014, Build Your Own Quadcopter, TAB.
- Sabin Civil Engineering, 2021, Drones | How do they work?, [online],
https://www.youtube.com/watch?v=N_XneaFmOmU, [Ziyaret Tarihi: 2 Nisan 2025].
- Sağ, T., 2025, A* Algoritması, Ders Notu.
- TechTarget, 2021, What is a drone (UAV)?, [online],
<https://www.techtarget.com/iotagenda/definition/drone>, [Ziyaret Tarihi: 2 Nisan 2025].
- UMILES Group, 2023, What is a drone and what is it for? Characteristics and functions, [online], <https://umilesgroup.com/en/what-is-a-drone-and-what-is-it-for-characteristics-and-functions/>, [Ziyaret Tarihi: 2 Nisan 2025].
- Wang, Y., Zhang, T., ve Chen, X., 2023, "A Shortest Distance Priority UAV Path Planning Algorithm for Precision Agriculture", International Journal of Agricultural Robotics.
- Zhao, L., Huang, J., ve Li, K., 2023, "A Survey on Obstacle Detection and Avoidance Methods for UAVs", Journal of Aerospace Engineering.