

1. Detaylı Tasarım Özeti:

Uçuş Kontrol Kartı: İHA'nın pilot, sensör ve yer istasyonunda aldığı bilgilerle otonom bir şekilde uçuşmasını sağlayan donanım. Kullanımının kolay olması, düzenli olarak güncelleme alması, aynı fiyatattaki diğer oto-pilotlara göre daha iyi performans göstermesi sebebiyle Pixhawk Cube Orange kullanmayı tercih etmişler. Uçuş kontrol kartı içinde bulunan ARM STM32H ana işlemci ve STM32F103 yardımcı işlemci ile sağlam performans veriyormuş. Ayrıca kartın içindeki IMU, barometre, manyetometre ve ivmeölçer gibi sensörler ile uçuş performansı artırılmış. Kartın içine yüklenecek yazılımın açık kaynak kodlu PX4 yazılımınınmasına karar vermişler.

GPS: GPS modülü olarak yüksek hassasiyetli here3 modelinin kullanılmasına karar verilmiş. GPS modülleri tek başına kullanıldıklarında İHA'nın hızına bağlı olarak 2 metreye yakın hatalı sonuç veriyormuş. Bu durumun önlenmesi için RTK base istasyonu kullanılmış. Base uyumlu bir rover olan Here 3 GPS anteni ile hata payı 2 cm seviyelerine kadar düşürmeyi amaçlamışlar.

Görev Bilgisayarı: Yarışma görevlerinin yerine getirilmesi için gerekli olan görüntü işleme algoritmaları, takip algoritmaları ve QR kod okuma işlemini gerçekleştirerek uçuş kontrol kartına gerekli komutları gönderecek bilgisayardır. Bu işlemler için birden fazla sınır ağını paralel çalıştırabilen görüntü işleme ve yapay zekâ uygulamaları için üretilmiş 128-core Maxwell GPU'ya sahip NVIDIA Jetson Nano bilgisayarı kullanılmış. Üzerindeki HDMI, USB, Micro-USB gibi veri portları, MIPI-CSI kamera konnektörleri ve gigabit seviyesinde veri传递i sağlayan ethernet portu gibi lazım olan tüm I/O portlarına sahipmiş bu canavar. Ayrıca küçük ve hafif olması önemli avantajlarından biri.

Lidar: Kamikaze İHA görevi sırasında İHA yere çakılma riskini azaltmak için ve otonom iniş gerçekleştirirken güvenliğinin sağlanması için 12 m'ye kadar ölçüm yapabilen TF Mini LiDAR kullanılmış. Eğer bütçe yeterse 40 m'ye kadar mesafe ölçebilen LidarLite v3 modeline geçeceklermiş ama uygun bütçeyi bulup geçmişler yoksa bütçeyi bulamayıp geçmemişler mi yazmıyor.

1.1 Alt Sistem Özeti: Aracın uçuş kontrolünün sağlanması için Pixhawk Cube otopilot kartı ve bu kartın içinde PX4 yazılımı kullanılmıştır. PX4 yazılımı Ardupilot gibi açık kaynaklı yazılım. PX4'un seçilmesinin sebebi sabit kanat desteğinin Ardupilot'a göre daha iyi olması ve takımın tüm çalışmaları bu yazılım üzerinden yapmış olması. Otopilotun uçuşu sorunsuz gerçeklestirebilmesi için içinde IMU sensörü de bulunan yüksek hassasiyetli HERE3 GPS/GNSS modülü ve hava hızının ölçülmesi için pitot tüpü kullanılmış. Pixhawk'ın içinde ayrıca IMU manyetometre, ivmeölçer sensörleri bulunmaktadır. Takım otopilot seçimi için piyasadaki çeşitli kartları araştırmış sonra diğer takımlara sormuş hangisi daha iyi diye en son Pixhawk Cubde'de karar kılımişlar.

	Pixhawk Black Cube	Pixhawk Orange Cube	Pixhawk 4
İvmeölçer Sayısı	3	3	2
Manyetometre Sayısı	3	1	1
Barometre Sayısı	2	2	1
RAM	256 KB	1 MB	512 KB
İşlemci	STM32F427 V	STM32H753	STM32F100
Yardımcı İşlemci	32 bit STM32F103	STM32F103 32bit ARM Cortex-M3 24 MHz 8KB SRAM	STM32F100 32 Bit ARM Cortex M3, 24MHz, 8KB SRAM

Yarışmadaki görevlerin gerçekleştirilmesi için nesne tanıma işlemlerinde yüksek performans veren ve küçük boyutlu, düşük güç tüketimine sahip Jetson Nano görev bilgisayarı tercih edilmiş. Raspberry Pi istenilen performansı sağlamamış, Jetson NX ve AGX gibi daha gelişmiş modeller ise hem pahalı hem de zaten Jetson Nano istenilen ihtiyaçları karşılamaya yetiyormuş o yüzden uygun fiyatlı ve yüksek performanslı Jetson Nano tercih edilmiş. Bu bilgisayara bağlı bir şekilde görüntünün alınacağı e-CAM24_CUNX kamera modülü ve görüntü iletimini gerçekleştirecek Rocket M5 erişim noktası kullanılmış. Haberleşmeyi tamamlamak için yer istasyonunda Ubiquiti PowerBeam anten ve RFD868+ telemetri kullanılmış.

2.Otonom Sistemler:

Otonom uçuş için ESC'yi yönlendirmek lazımmış. Bu ESC kontrol alanlarını yönetiyor ve servo ile motorların hızını ayarlıyor. Servoları ve ESC'yi ticari PID kontrol ile yöneten ve daha önceki yarışmalarda başarısını kanıtlamış Pixhawk uçuş kontrol kartı tercih edilmiş. Pixhawk, hem kendi üzerindeki hem de harici (GPS ve pitot tüpü) sensörleri kullanarak otonom uçuş için gerekli olan uçağın konum, yükseklik, hız ve açısal verilerini (*roll*, *pitch*, *yaw*) yüksek doğrulukla tespit edebiliyor. Bunlarla beraber yer istasyonu ile telemetri üzerinden haberleşebilir, otonom iniş ve kalkış emri yer istasyonundan verilebilir. PID katsayıları ve sensörleri kalibre edildikten sonra uçak stabil bir kalkış ve iniş yapabiliyormuş. Sonrasında yeni bir emre kadar loiter yapıyor yani bir bölge üzerinde havada dairler çizerek turluyor.

2.1 Otonom Kilitlenme Görevi: Hedef İHA'nın kitlenme görevi ve bunun için gerekli olan uygulamalar için Pixhawk'ı yöneten bir yapay zekâ kartı kullanılmış. Uçağın görev döngüsünün (hedef tespiti, takibi ve uçuş kontrol girdisi verme) performansı için minimum hız gereksinimi, saniyede beş kareden (5 FPS) daha hızlıymış yani, sistemin her 0,2 saniyede bir yeni verileri işleyip uçuş kontrol kartına iletiyor. Bu işlem süresinin 0,2 saniyeyi aşması, uçağın hedefi başarıyla yakalama ve kilitleme yeteneğini önemli ölçüde sıkıntiya sokuyor. İlk olarak NVIDIA Jetson Nano üstünde ilk uçuş testleri gerçekleştirilmiş.

Algoritma, her döngünün başında uçağın konumu, hızı, ivmelenmesi, batarya durumu ve pitot hızı gibi verileri Pixhawk'tan Mavlink protokolüyle alıyor. Ardından, yer istasyonundan hedef İHA'ların konum bilgilerini alıyor. Bu verilerle birlikte, uçağın anlık durumu yarışma sınırlarıyla karşılaştırılmış. Eğer sınır ihlali söz konusuya ya da batarya seviyesi düşükse, yer istasyonuna uyarı gönderiliyor. Durum kritikse, uçak ya belirlenen loiter noktasına yönlendirilir ya da iniş emri veriliyormuş. Bu işlemlerin ardından algoritma döngüye giriyormuş.

Sınır ve güvenlik kontrolünde sorun yoksa, sistem önceki döngüden aktif bir hedef takibi olup olmadığını kontrol ediyor. Aktif kilitlenme yoksa, kameraldan alınan görüntü işleniyor. Görüntü işleme sonucu hedef tespiti yapılamazsa ve son 3 saniyede herhangi bir tespit gerçekleşmemişse, rakip İHA'ların konum verilerine bakılarak İHA'nın uçuş konisi içinde kalan en yakın hedefe yönelmesi sağlanıyor ancak son 3 saniyede bir tespit varsa, önceki uçuş komutu tekrarlanıyor.

Eğer görüntü işleme ile bir hedef tespit edilirse, bu hedefin konumu, görüntüsü ve boyutu takip algoritmasına aktarılır. Görüntü takibinde, hedefe bir ID atanır ve yarışma telemetrisinden alınan bilgilerle eşleştirilerek aynı hedefe tekrar kilitlenmesinin önüne geçilmiştir. Devamında takip sürecine geçirilir ve her döngüde hedefin konumu ile boyutu takip edilmiştir. Görüntü tanımlaması devam ederse,

bu bilgiler Kalman filtresine aktarılır. Ancak hedef görüntüsü bir sonraki döngüde kaybolursa en son verilen komut tekrarlanırmış.

Sadece hedef İHA'nın görüntüdeki yerini tahmin edebilmek için Katman_filtresi kullanacaklarımış çünkü hedef İHA ve takımın İHA'sı stabil değilmiş birde uçağın ağırlığı ve dinamik özellikleri kesin olarak kestirilememiştir. Tahmin edilen pozisyonu doğru Kalman filtresi takımın İHA'sına'ye yönelik atayıp bu bilgiyi Pixhawk'a iletecek. Yine bu aşamada herhangi bir başarısızlık olursa son emir tekrarlanacaktır. Pixhawk'a ayrıca bu aşama hız verisi de gönderilerek tespit edilen uçağın alanı gelecekte uçuş testlerinden sonra belirlenecek alandan küçük olursa uçağın hızını artırmasını, geniş olursa da hızını düşürmesini istenecemdir. Böylelikle hedefin konumunu ve gerçek boyutunu bilmeden yönelik ve hız ayarlaması yapabilecek. Bu algoritmanın başarılı olabilmesi için ana döngünün hızlı çalışması ve hedef İHA'nın görüntüsü üzerinde büyük zıplamalar olmadan kısa aralıklarla oynaması gerekmektedir ki kontrol algoritması aradaki farkı sönümlerebilse.

2.1.1.Yapay Zekâ Modeli Takip Algoritması: Kalman filtresine alternatif olarak takviyeli öğrenme kullanılması düşünülmüş. Hedef İHA'nın görüntüdeki yeri, boyutu ve uçağın son 5 adımdaki uçuş verileri girdi olarak verilip çıktı olarak uçağa yönelik ve hız verisi vermesi beklenmiştir.

Environment, Agent ve Value ağıları Pytorch kullanılarak iki katmanlı MLP (çok katmanlı algılayıcı) yapılarıla oluşturulmuş ama alternatif olarak Tianshou isimli açık kaynaklı RL kütüphanesini de kullanmayı düşünülmüşler. Environment ağı, uçağın son 5 saniyelik verileriyle ve olası yönelik komutlarına göre gelecekteki konumunu tahmin ediyor; bu model gerçek uçuşlardan elde edilen Pixhawk log verileriyle eğitilmiş. Value ağı, olası hareket komutlarını Monte Carlo ağaç taraması ile değerlendirip her biri için bir başarı değeri çıkarmış; bu sırada Environment ağı da kullanılmış. Bu sayede, kilitlenme sırasında hangi durumların başarıya, hangilerinin başarısızlığa yol açacağı tahmin edilmiş.

Agent algoritması ise Value ağının çıktılarıyla, kendi değerine göre en uygun kararı verecek. Modelin girdi ve çıktıları ayrı olup modelin eğitimi yaklaşık 10 bin simülasyon oyunu sürecektir. Ancak bu yöntemin dezavantajı, gerçek uçak dinamигine çok yakın bir simülasyon gerektirmesi ve eğitim süreçlerinin yüksek işlem gücü ile uzun zaman olması.

Döngünün sonunda, üretilen yönelik ve hız komutlarının uçağın uçuş zarfına uygunluğu kontrol edilerek Mavlink protokolü üzerinden Pixhawk'a iletildiğinde istasyonuna hedef kilitlenme durumu bildirecek ve yarışma ağına güncel telemetri verileri aktarılacaktır.

2.1.2 Görüntü İşleme Algoritması:

Jetson Nano'ya ilk olarak NVIDIA JetPack SDK kurularak jetson-inference açık kaynak yazılım CMAKE araçları ile derlenerek Jetson-inference içinde bulunan C++/CUDA ile yazılmış detectnet görüntüyü yakalama ve takip algoritmaları eklenerek özelleştirilmiştir. AARCH64 mimarisi için özel olarak derlenmiştir.

Görüntü işleme algoritması için NVIDIA TensorRT kullanılmıştır. Yüksek performanslı derin öğrenme modeli için bir SDK olan NVIDIA TensorRT, tahmin uygulamaları için düşük gecikme süresi ve yüksek verim sağlayan bir derin öğrenme tahmin iyileştiricisi içeriyor.

TensorRT, PyTorch(GPU'ları ve CPU'ları kullanarak derin öğrenme için optimizasyonu sağlayan bir kütüphane) ve TensorFlow ile entegre bir şekilde çalışabiliyor böylece 1 satır kodla 6 kat daha hızlı çıkarım elde edebiliyor. Open Neural Network Exchange olarak bilinen ONNX Microsoft, Aws ve Facebook ortaklığıyla, deep learning modellerini temsil etmek ve farklı araçlar/frameworkler arasında geçiş yapmak ya da kombinasyonlar oluşturmak için geliştirilmiş açık kaynak kodu yazılım.

Temeli NVIDIA CUDA programlama üzerine olan Jetson-inference, NVIDIA firmasının jetson nano, TX1 and TX2 gibi geliştirme kartları için temel yapay zeka işlemlerini yapacak araçlar sunduğu açık kaynak kodlu bir yazılım.

Burada UFF, ONNX, ENGINE model dosyalarında bulunan eğitimli modeller ile:

- IMAGENET
- DETECNET
- SEGNET
- POSENET

gibi işlemler yapılabiliyor. Projede object detection işlemi için DETECTNET aracını kendilerine göre özelleştirmişler bu sayede kamerada görülen hava aracının görüntü alanına göre koordinatının ve genişliğinin tespiti yapmışlar.

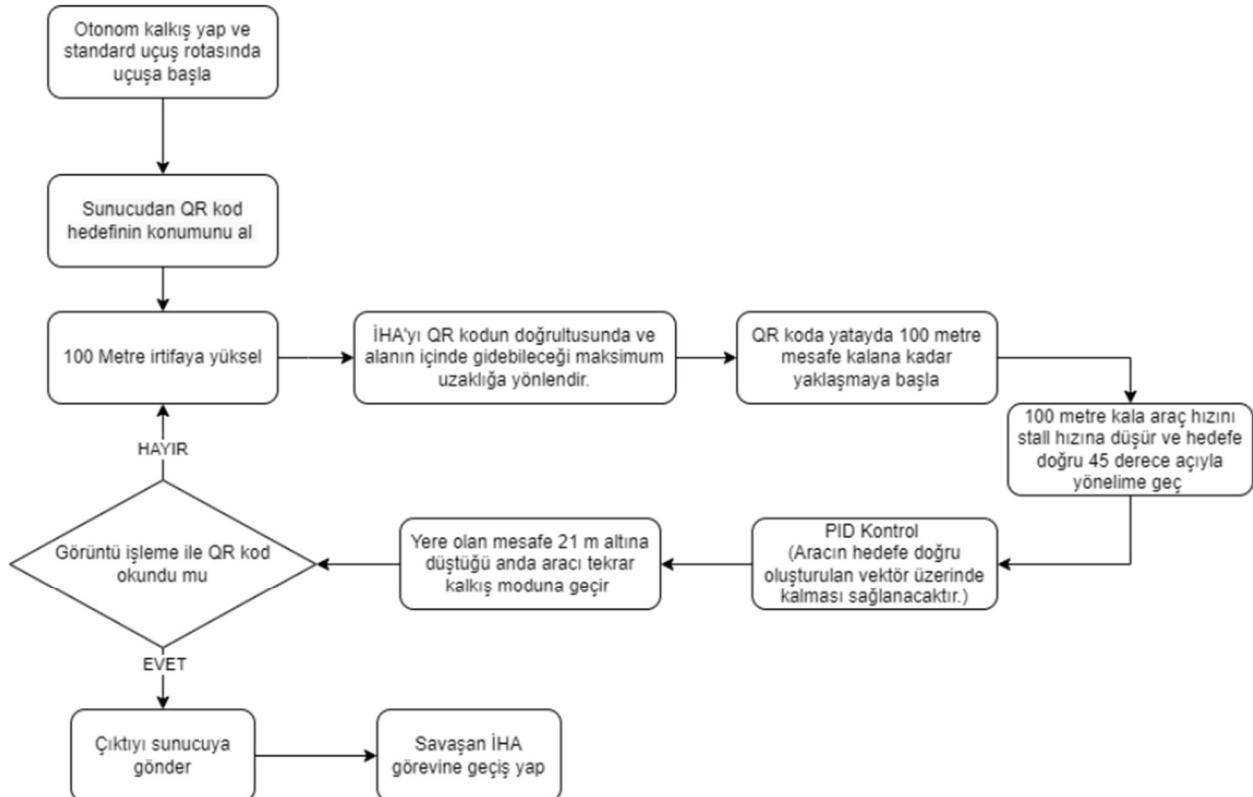
CUDA (Compute Unified Device Architecture): GPU (Graphics Processing Unit) için NVIDIA'nın sunduğu C programlama dili üzerinde eklenti olarak kullanıma sunulan bir mimari ve teknoloji. CUDA, NVIDIA tarafından geliştirilmiş ve çalışması için NVIDIA GPU ve son sürüm sürücülere ihtiyaç var. CUDA G8X üzeri, GeForce, Quadro ve Tesla'yı içeren her GPU üzerinde çalışır. NVIDIA, ekran kartı mimarilerinin ileriye doğru kod uyumluluğu sayesinde, Geforce 8 için geliştirilen programların herhangi bir düzeltme yapılmadan gelecek nesil ekran kartlarında hızlanmalardan otomatik olarak faydalananacak şekilde kullanılabileceğini belirtiyor. CUDA kütüphanesi, geliştiricilerin CUDA özellikli GPU'lar üzerindeki hafızalara ve Stream Processorları kontrol edebilmesini sağlıyor.

PathScale: C ile yazılmış algoritmaların GPU üzerinde çalışmasını sağlayan geliştirme araçları kümesi.

Jetson-inference: PyTorch kullanımıyla ile ilgili araçlar sunuyor. Bu araçlar ve train_ssd.py aracını kullanarak 14391 hava aracı fotoğrafıyla 100 dönem eğitilmiş. Sonrasında bu modeli TensorRT engine formatına dönüştürmek için önce ONNX formatına dönüştürmüştür. Bu işlemi yaparken yine jetson-inference'in onlara sunduğu onnx export aracını kullanmışlardır. Daha sonra kendilerine göre özelleştirtilmiş DetectNet aracı ile modelin eğitimi yapılmış.

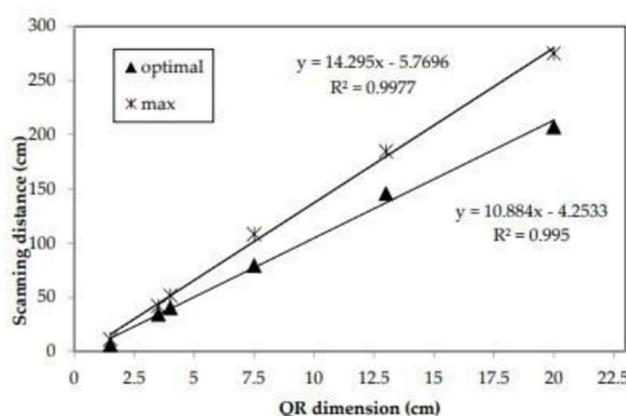
YOLOv5s nesne tanıma modeli ve Python kullanılarak eğitilen modelden 5 FPS alınırken, C++ ve TensorRT kullanılarak yeni eğitilen modelde 40 FPS tanıma hızına kadar çıktıığını görmüşler.

2.2 Kamikaze Görevi: Kamikaze İHA görevinde İHA'nın kaza kırıma uğraması riskinin yüksek olması sebebiyle son iki uçuş sırasında denenmesine karar vermişler. Göreve ilişkin algoritma akış şemasını aşağıya koydum.



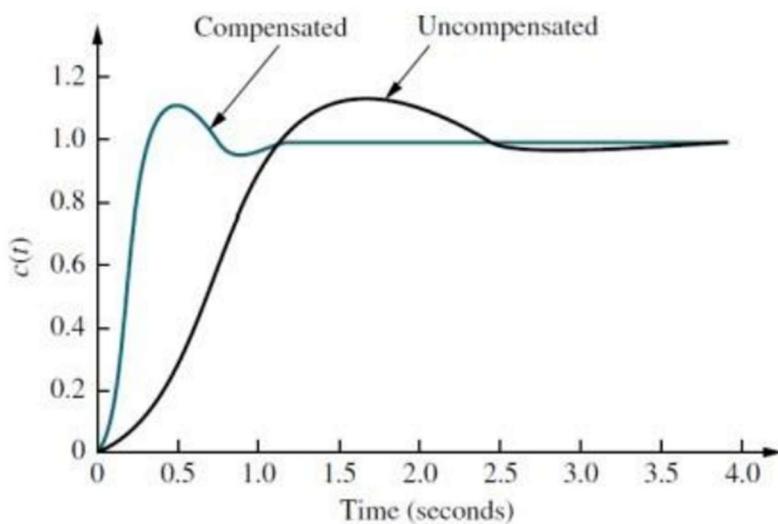
Şekil 22: Kamikaze görevi algoritma akış şeması

Araç sunucudan gelen QR kodun konum bilgisini aldıktan sonra düzgün bir dalış gerçekleştirebilmek için hedefin doğrultusunda alanda gidebileceği en uzak noktaya ilerleyecek ve 100 metre irtifaya yükselecek. Daha sonra 100 metre mesafe kalana kadar hedefe yaklaşacak ve 100 metre kaldığında dalış işlemini gerçekleştirebilmek için pitch açısını 45 derece olarak ayarlayarak hedefe yaklaşmaya başlayacakmış. 100 metreden itibaren dalış yapılacak rota düz bir vektör şeklinde belirlenmiş ve aracın bu vektör üzerinde ilerleyebilmesi için PID kontrol kullanılmış. Pixhawk Küp'e gönderilen komut ile elevator açısı değiştirerek aracın pitch açısını kontrol ediyormuş. İstenen sonuç vektörünün elde edebilmek için gerekli rudder açısının belirlenmesi PID kontrol ile yapılmış. "Exploring Smart Glasses for Augmented Reality" adlı makalede yapılan QR kod tanıma ile ilgili yapılan ölçümelerde aşağıdaki sonuçlar elde edilmiş.



Grafikte görüldüğü üzere QR kod kenar boyutuyla tarama mesafesi orantılı olarak artıyor. Bu oran ile yaklaşık 20-25 metreden tanıma yapılması bekleniyormuş. Görüntü tanıma mesafesinin optimum değerini bulmak için çevresel koşullara ve kamera çözünürlüğünə bağlı testler yapılmış. Okuma işlemini gerçekleştirmek amacıyla İHA, 20 metre mesafeye kadar alçalacak ve okuma sonucu ne olursa olsun hemen tekrar yükselecektmiş. Eğer okuma görevi başarılıysa Savaşan İHA görevine geçilecek; başarısız olursa görev tekrarlanıormuş. Bu hassas iniş görevi sırasında GPS irtifa verisinin yetersiz kalma ihtimaline karşı, tam ve hızlı yükseklik ölçümü için uçağın altına bir Lidar sensörü yerleştirmiştir. Bu Lidar, çarpışmayı önleyecek ve kontrol algoritmasının doğru çalışmasını garanti altına alacakmiş. Ayrıca kameranın yarışma öncesinde zoom oranının değiştirilmesine izin verilmemiş. Yapılacak testlerde iyi performans vermesi durumunda zoom oranını 2x olarak ayarlamayı düşünmüşler.

P kontrolöre integral kompenzatörü eklemişler ve sistemin sürekli hal hatası minimuma indirgenmiş birde türev kompenzatörü ile geçici hal davranışları iyileştirilmiş. Bu sayede ayarlanan parametreler ile sistemin en hızlı ve doğru şekilde istenilen vektör üzerine konumlanması sağlanmış. PID kontrolün sisteme etkisini aşağıdaki grafikte göstermiştir.



3. Simülasyon:

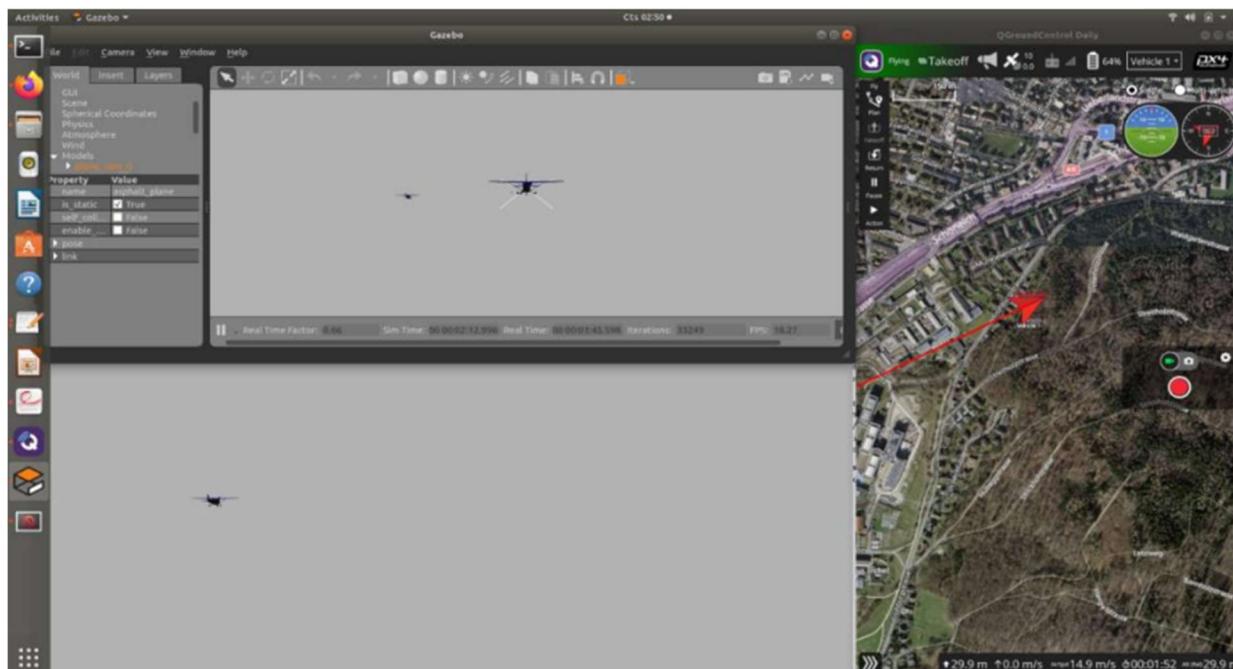
Simülasyon testlerinin gerçekte yaşanabilecek durumları önceden test etmek, kodları ona göre geliştirip iyileştirmek ve sistemin optimizasyonu için gerçek ortamda test edilmeden önce yapılması gerekmekte. Fizik tabanlı simülasyon ortamları arasından Gazebo, ROS (Robot Operating System) yazılımı ve PX4 uçuş kontrol yazılımının birlikte kullanımını İHA'lar üzerinde yapabilmesinden dolayı tercih etmişler. Gazebo ayrıca aerodinamik etkileri simüle etmesi, İHA'ya özel barometri, GPS ve Lidar gibi sensörleri barındırması bakımından tercih edilmiş. Donanımın döngüde olduğu (Hardware in the Loop) simülasyonlarda Gazebo ile görselleştirilebilmiş bu sayede kontrol kartları sanki gerçekte uçuyormuşçasına test etme şansları olmuş.

Hem Gazebo simülasyonlarında hem de gerçek uçuş sisteminde (Jetson Nano üzerinde), uçuş kontrol kartına komut göndermek için MAVROS yazılımı kullanılmış. MAVROS, robotik dünyasında yaygın kullanılan ve farklı yazılım dillerinin iletişimini sağlayan ROS (aracı yazılım - middleware) ile MAVLink protokolünü birleştirmiştir. Bu seçimi ROS'un veri aktarımı ve alt sistemler arası bağlantı özelliklerinden yararlanmak ve MAVLink üzerinden uçuş kontrol kartını doğrudan kontrol etmek içi yapmışlar. MAVROS ayrıca yer istasyonları için (QGroundControl gibi) bir UDP MAVLink köprüsü oluşturarak, hem simülasyonda hem de gerçek uçuşa bağlantı kurulmasını sağlamıştır. Sonuç olarak, MAVROS, kullanıcının veya görüntü işlemeden gelen takip komutlarına göre İHA'nın hareket etmesi için uçuş kontrol kartına komut ileten temel bağlantı aracı olarak işlev görecektir.

Literatürde uçuş kontrol komutları göndermek için oluşturulmuş daha birçok kütüphane mevcutmuş ama bu kütüphanelerin bir kısmı simülasyon üzerinde test edilmiş ve istedikleri şeyleri karşılamadığı için tercih edilmemiş.

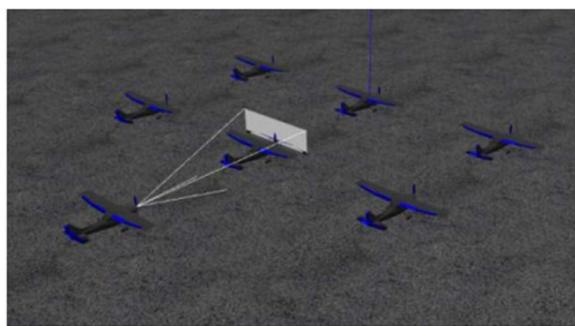
3.1. Otonom uçuş ve takip denemeleri

Takım İHA ile uçuş yapmadan önce riski azaltmak için ilk otonom uçuş, yer istasyonu ile bağlantı, mod değişimi komutlarının denenmesi için uçuş simülasyon üzerinden yapmış ve aşağıdaki görüntüler elde etmiş



3.1.2. Savaşan İHA Görevi Testleri

Takım savaşan İHA görevi için simülasyonda aynı anda çok sayıda uçağın uçuşu gerçekleştirmiştir ve kilitlenme testi ve takip algoritmaları denemiştir. Simülasyon testlerine ait otonom uçuş ve araç kamerasından alınan görüntü ile yapılan kilitlenme testi görselleri aşağıya koymuştum.



Yarışma alanının tam manasıyla simüle etmek amacıyla çok sayıda model ile testlerin yapılması



Kilitlenme testi