HEZARFEN OTONOM

Yazılım mimarisinin genel şeması bu şekilde. İlk olarak **algılama**dan başlayalım:

OpenCV kütüphanesindeki görüntü işleme fonksiyonlarını kullanarak Python ile oluşturuyorlar.

Şerit tespiti yazılımında kameradan görüntü alınıyor. İlgili alan seçiliyor (RoI), renk ayarları yapılıyor ve özel olarak geliştirdikleri kayan pencereler algoritması kullanılıyor. Ardından şeritlerin matematiksel modelleri çıkarılıyor ve şerit konumları koordinat sistemine geçiriliyor. Sonunda şerit bilgisi lokal planlayıcıya aktarılıyor.

RoI (region of interest) ise görüntünün tamamında çalışmak yerine belli bir kısmı seçip o alandaki pikseller üzerinde çalışmaya yarıyor.

Kayan pencereler algoritması ise RoI kullanarak görüntünün tamamını taramaya yarıyor.

Renk ayarı olarak da normalde BGR formatında olan görüntüyü önce blurlayıp sonra HSV uzayına taşıyorlar. Bu sayede şerit görüntüsü daha iyi alınabiliyor.

Kenar tespiti işleminin nasıl yapıldığı ayrıntılı olarak açıklanmış. Aldığımız görüntüyü HSV uzayına aktarmıştık. Şimdi piksel piksel rengin istediğimiz aralıkta olup olmadığına bakıyoruz. Eşik değerini geçtiyse 1, geçmemişse 0 olarak işaretleniyor ve bir maskeleme matrisi oluşturuluyor. Kenarların net tespiti için de benzer bir işlem yapılıyor. Bu işlem için Canny, Sobel, Roberts, Prewitt gibi operatörler kullanılıyormuş. Onlar Sobel kullanmışlar çünkü dikey ve yatayda ayrı ayrı tespit yapmasının yanı sıra daha az işlem gücü gerektiriyormuş.

Görüntü işlemlerden sonra bu hale geliyor. Ama sonra bu görüntüde perspektifi değiştirerek kuş bakışı bir görüntüye çeviriyorlar.

Sırada kayan pencereler algoritmasına girdi olarak vereceğimiz veriyi oluşturmak var. 0 ve 1lerle oluşturduğumuz matriste aslında 1 ile beyazı, 0 ile siyahı gösteriyorduk. Şimdi histogram hesabı ile matristeki her bir sütuna ait değerler toplanıyor. Şeritlerin koordinat sistemindeki yerini bulmamızı sağlayacak.

Elde edilen veri kayan pencereler algoritmasında kullanılıyor. Daha önceden bahsettiğim gibi bu algoritmada RoI ile belli bir alan sınırlandırılıyor ve bu kutu görüntü üzerinde kayarak piksellerin konumlarını net şekilde belirliyor. Bu aşamayla birlikte şeritlerin net koordinatı belirlenmiş oldu.

Daha kesin sonuçlar için matematiksel model oluşturuluyor ve modelle birlikte şeritlerin eğrilik yarıçapları tespit ediliyor. Tespit edilen şerit modelleri ve eğrilik yarıçapları lokal planlama düğümüne aktarılıyor ve böylece şerit tespit işlemi tamamlanıyor.

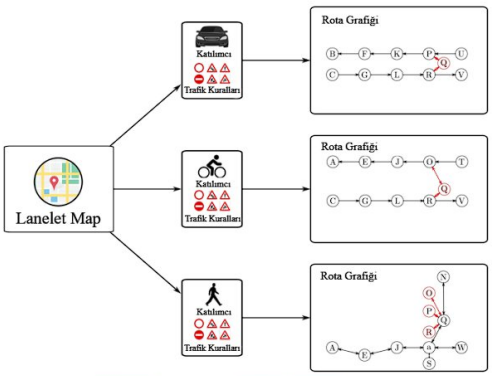
Engel tespiti için LiDAR sensöründen gelen veri kullanılıyor ve bir maliyet haritası çıkarılıyor. ROStaki paketleri kullanarak bu haritayı sürekli güncelliyorlar. Burada elde edilenler de lokal planlayıcıya aktarılıyor.

Lokalizasyon

Sensör füzyonu kullanılmış. Kullanılan sensörler: ZED2 model stereo kamera, IMU ve teker odometrisi. Bu sensörlerden elde edilen verilerin füzyonu için görsel odometri, teker odometrisi ve IMU verilerini EKF (Extended Kalman Filter) yardımıyla birleştiren ROS bünyesi altında sunulmuş robot_pose_ekf adlı bir paket kullanılmışlar. Raporda oldukça teknik açıklamalar yapılmış, açıkçası çok anlamadım Ama bu aşamanın kısaca sensörlerden gelen bilgileri ROS yardımıyla birleştirerek arabanın konumu, hangi yöne baktığı gibi bilgileri oluşturmaya yaradığını söyleyebilirim.

Planlama

Amaç, aracın harita verilerini kullanarak güvenli rotalar planlaması. Harita teknolojisi için HD Vector Map ve Lanelet2 adlı açık kaynaklı bir C++ kütüphanesi kullanılmış. Global planlayıcı A ve B noktaları arasında optimum yolu bulmaya yarıyor, bu planlayıcı için de yine Lanelet2 kütüphanesinin bir modülünü kullanmışlar.



Modül, Lanelet2-routing. Rota oluşturmak için ortamda bulunan trafik işaretlerini, kuralları yorumluyor ve yanda görüldüğü gibi rota planlaması yapıyor.

Global planlayıcı, ortamdaki durgun engelleri hesaba katarken lokal planlayıcı ortamdaki hareketli engelleri de görüyor. Bu sayede global planlayıcı ile oluşturulan rotanın en iyi biçimde takip edilmesini sağlıyor. Dynamic Window Approach tabanlı bir yazılımla çalışmışlar.

Dynamic Window Approach(DWA), aracın hareketinin belirli bir zaman aralığında incelenerek gelecek için daha iyi kontrol komutları oluşturmaya yarıyor. Hız ve ivme özelliklerini düşünerek bunları değiştirebildiğimiz seçenekler yaratıyoruz ve bu seçenekler arasında en iyi olan sonraki denemede kullanılıyor. Bu süreç tekrarlanarak en iyiye ulaşılıyor.

Kontrol

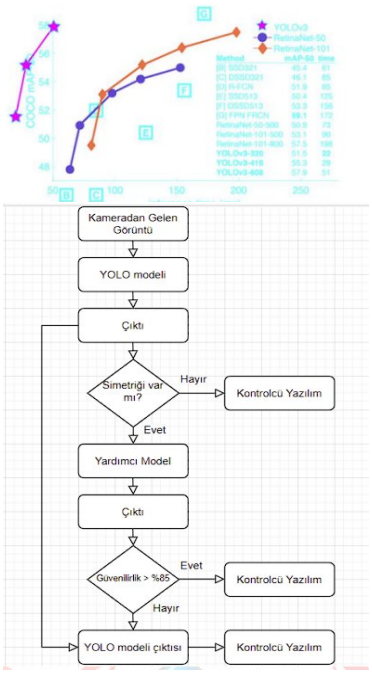
Lokal planlayıcı tarafından oluşturulan hareket kontrol komutlarını aracın takip etmesini sağlar. Bu işlemi gerçekleştirmek için PID kontrol tabanlı bir yazılım geliştirmişler. PID, sistemi istenen değere yaklaştırmak ve kararlı hale getirmek için kullanılan geri beslemeli bir kontrol algoritması. Sensörlerden gelen veriyi okuyarak algoritmayı çalıştırıyor ve motor, direksiyon gibi parçalara komut gönderiyor.

Arabada boylamsal ve enlemsel hareketi sağlamak için iki ayrı PID kontrolcüsü kullanılmış. Boylamsal olan aracın öteleme hızını düzenleyip sinyali teker motorlarına gönderiyor. Enlemsel olan ise direksiyon açısını düzenleyip Steer-by-Wire sistemine veri aktarımından sorumlu.

Steer-by-Wire direksiyonun tekerlekle arasındaki fiziksel bağlantıyı tamamen kaldırarak elektrik motorlarıyla çalışan bir sistem. Havacılıkta uzun süredir kullanılıyor, yeni yeni araçlarda da kullanılmaya başlanmış.

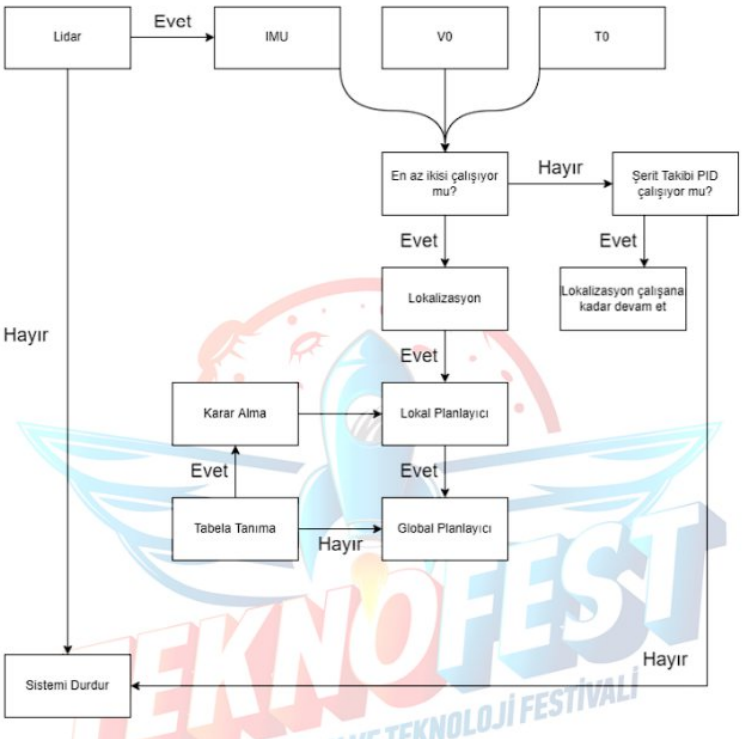
Tabela tanıma sisteminde YOLO algoritması kullanılıyor. Artı olarak CNN yani Geleneksel Konvolüsyonel Sinir Ağını kullanarak sistemin güvenliğini artırmayı amaçlamışlar. YOLO algoritması kullanılıyor, sebebi diğerlerine göre daha hızlı olması ve görüntüyü tek seferde işlemesi. Görüntüyü nxn parçalara bölüyor ve her parça kendi içinde nesne olup olmadığına ve varsa da orta noktası o parçada mı diye

bakıyor. Kesin bulunduğu yerleri 1, değilse 0la işaretleyerek bir puan hesaplıyor. Aradığımız nesnenin oluşan bu dikdörtgenlerin içinden hangisinde olduğunu da bu puana bakıp en yükseğini seçen Non-Maximum Suppression algoritması sayesinde anlıyoruz.



YOLO'nun en hızlı olduğu burada görünüyor. (COCO veri setinde eğitilmiş) Ama hızlı olmasına rağmen doğruluğunda sıkıntılar olabiliyor. Verileri karıştırıyor. O yüzden yardımcı model kullanmaya daha karar vermişler. Kullanılanlar: Geleneksel Konvolisyonel Sinir Ağı (CNN) ve Destek Vektör Makinesi (SVM).

Bu algoritmaları kullanmaya karar vermelerinin sebebi MNIST veri setiyle eğitilmeleri. Hali hazırda doğru ve güvenilir sonuçlar verdiğini kanıtlayan model örnekleri bulunuyormuş.



Aracın çalışma mantığını özetleyen bir şema. Acil durum planlama için yapılmış.

Yazılım mimarisinin genel tablosunda kontrol kısmı için PID ve Pure Pursuit yazılmış ama KTR raporunda Pure Pursuit için herhangi bir açıklama bulunmuyordu. Kısaca yolu takip etmek için kullanılan geometrik tabanlı bir kontrol algoritması olduğunu söyleyebilirim. Önceden bildiği rotaya göre bir hedef noktası seçiyor ve bu noktaya doğru dairesel yörünge oluşturabilecek şekilde bir direksiyon açısı belirliyor. Hedefe varana kadar bunu tekrarlıyor.

PID hata oranı hesaplayıp hedeflediği değerle gerçek değer arasındaki farkı sıfıra düşürmeyi amaçlıyordu ve daha karmaşık bir yapısı var. Pure Pursuit ise hata hesaplamıyor. Hedef noktaya ulaşmak için direksiyon açısının kaç olması gerektiğini belirliyor.

MİLAT

Sensörler

LiDAR: Lazer ışınlarını kullanarak etraftaki nesnelerin uzaklıklarını belirlemeye yarıyor. Işık hızıyla çalıştığı için çok hızlı şekilde etrafı tarayabiliyor. Ayrıca LiDAR 360 derece görüş sağlayabiliyor. Araçta OUSTER OS1-16 lidar tercih edilmiş. Kaportanın üst kısmında yer alıyor.

Kamera: DFK 33GX236 GigE color industrial kamera kullanılmış.

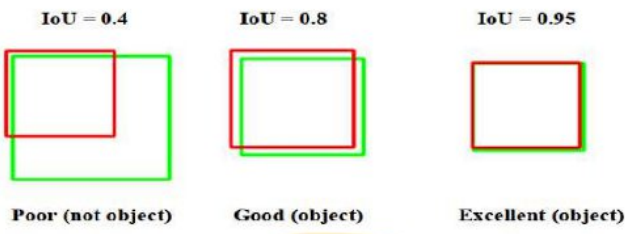
Yazılım Mimarisi

Genel bir hat çizelim: Tabela tespit algoritmaları, şerit takibi, lidar algoritması, karar mekanizmaları üzerine inceleme yapacağım.

İlk olarak **tabela tespit algoritmalarından** başlayalım. Kameradan alınan görüntüyü düzenleyerek verinin otonom sürüş algoritmalarında kullanılması sağlanıyor. Derin öğrenme ve makine öğreniminden yardım almışlar.

Bu araçta da tabelaları tanıyabilmek için YOLO algoritması kullanılmış. Model yaklaşık bir hafta boyunca 10 bin görüntüden oluşan bir veri setinde eğitilmiş. Nasıl çalıştığına gelirsek her görüntü eğitim için sinir ağından geçiriliyor. Ağ, trafik işaretinin nerede olabileceğini RoI ile belirliyor. Ardından işaretlerin hangi sınıfa ait olduğunu tahmin ediyor.

$$IoU = \frac{A \cap Gt}{A \cup Gt} \begin{cases} > 0.7 = \text{object} \\ < 0.3 = \text{not object} \end{cases}$$



Hangi sınıfta olduğuna karar verince de nesnenin koordinatları belirleniyor ve elde edilen veriler karar mekanizmasına aktarılıyor.

Şerit takibi için Python dili içinde bulunan OpenCV ve Numpy kütüphaneleri kullanılmış. Yapılan işlemler Hezarfen Otonom'un yaptığıyla tamamen aynı. Kameradan gelen görüntünün RoI yardımıyla bir parçasını alıyorlar. Siyah beyaza çevirip 0 1lerden maskeleme matrisini oluşturuyorlar ve perspektifi düzelttikten sonra şeritlerin koordinatlarını belirlemek için kayan pencereler algoritmasını kullanıyorlar.

LiDAR algoritmasında, lidar tarama verilerini alıp işleyerek engellere olan uzaklığı belirliyorlar ve karar mekanizmasına iletiyorlar. Bu algoritmada ROSpy kütüphanesi kullanılmış. ROSpy, Python dilinde aracın sensörleri, kameraları ve diğer parçaları arasında veri alışverişi yapmaya yarıyor.

Diyelim ki araç yola çıktı, bu algoritmaları nasıl kullanacak?

Tabela tespit algoritmasıyla trafik işaretlerine uyması ve engelleri tespit etmesi bekleniyor. Kurallara uydu ama önüne bir engel çıktıysa da lidar ile uzaklık kontrolü yapılıyor. Eğer uzaklık eşik değerin altında ise aracın yeni rota oluşturmasını bekliyorlar. Bu rotayı oluştururken şerit takibi yapılıyor ve güzergah üzerindeki uygun şerit belirleniyor. Araç tüm bu algoritmaları beraber kullanarak parkura devam ediyor.

Karar mekanizması için dört farklı durum belirlenmiş: düz git, sağa/sola dön, dur ve park. Bu işlevleri yerine getirmek için yine az önce bahsettiğimiz algoritma ve sensörleri kullanıyorlar. ROS yardımıyla gelen veriler birleştiriliyor ve master düğüm dedikleri karar mekanizması ortaya çıkıyor. Mekanizmayla 0-90 derece aralığındaki uygun direksiyon açıları belirlenip hareket sağlanıyor.

Yazılım mimarisi kısmı bitti, genel olarak özetleyip özgünlük durumuna bakalım:

Şerit takibi için Numpy ve OpenCv kütüphaneleri kullanılıyor.

Tabela tespiti için YOLOv4 kullanılıyor, başta açık kaynak veri setleri kullanılsa da yeterli olmadığını görüp ekiptekilerin derlediği özgün bir veri seti kullanıyorlar. Derin öğrenme modeli ile eğittikleri YOLOv4 testlerin sonunda yüzde yüz doğruluğa ulaşıyor.

Çevredeki engeller için LiDAR kullanıyorlar, daha net görüntü elde edebilmek için aracın ön kısmına yerleştirilmiş. Gelen veriler master düğümde birleştiriliyor ve dört ayrı durumda karar vermek için kullanılıyor.

Simülasyon ortamı

Simülasyon ortamı olarak Unreal Engine 4.27 kullanılıyor. Simülasyonda kullanılan algoritma Python programlama dili ile geliştirilmiş. Yarışmanın yapılacağı parkura uygun olarak yol üzerinde şeritler, başlangıç ve bitiş çizgisi, trafik işaret ve ışıkları ve park alanı bulunuyor.

Algoritma, simülasyondan aldığı kameradan gelen görüntüleri işleyerek direksiyon açısını ve hızını belirleyebilmiş ve testler başarıyla tamamlanmış.

Simülasyon ortamı olarak kullandıkları Unreal Engine, gerçekçi görselleştirmesiyle öne çıkıyor. Araçta kullanılan sensörlerin sanal modellerini oluşturabiliyoruz. Ayrıca ROS bağlantısıyla veri alışverişini de mümkün kılıyor. Dezavantajları, donanım için yüksek işlem gücü istiyor ve arayüzü bazıları tarafından karışık bulunabiliyor.

İTÜ GÜNEŞ ARABASI EKİBİ

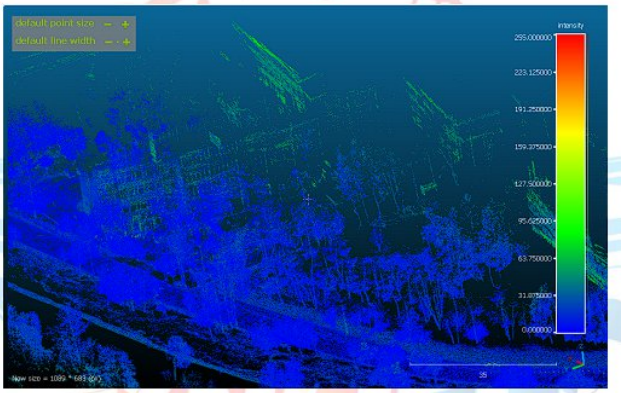
ROS 2 ve Arduino'nun ROS kütüphanesi kullanılmış. Genel olarak levha tespiti, levha uzaklıklarının ölçülmesi, haritalama çalışmaları, planlayıcı, kontrolcü, lokalizasyon, karar verme algoritması üzerinde duracağım.

İlk olarak **levha tespitiyle** başlayalım: YOLOv5 modeli kullanılmış. Eğitimi için Python dilindeki Torch kütüphanesinden faydalanmışlar. YOLO hızlı çalışması sebebiyle tercih ediliyordu, bu ekipte hızlı çalışması kadar doğruluk da önemsenmiş. Bu yüzdden YOLOv5 modelinin XL versiyonuyla 15 bine yakın veri ile eğitim yapılmış. Önceki versiyonlarına göre bounding box doğruluğunun arttığını söylüyorlar.

Uzaklık ölçerken ZED 2 Stereo Kameranın Python API'si kullanılmış. Üç boyutlu olarak kameradan verileri alıyorlar ve noktaya olan uzaklık formülüyle uzaklık tespiti yapılabiliyor. Veriler başta dediğim gibi ROS2 üzerinden ilerliyor.

Şöyle bir sorundan bahsedilmiş. Aracın tabela tespiti yaparken yoldaki hangi tabelayı kullanacağına karar vermesi gerekiyor. Planlayıcıyla uzaklık 5 ila 10 metre arasındaysa levhanın tespiti yapılıyor ve araç kurala uygun hareketine devam ediyor.

Velodyne Puck Lite 16 kanallı 3D lidar ile NDT (Normal distribution transform) tabanlı **haritalama işlemi** gerçekleştirilmiş. Bu haritalama işleminde odometri verisine ihtiyaç duyulmamasından dolayı IMU ve GPS gibi sensörleri de kullanılmamış.

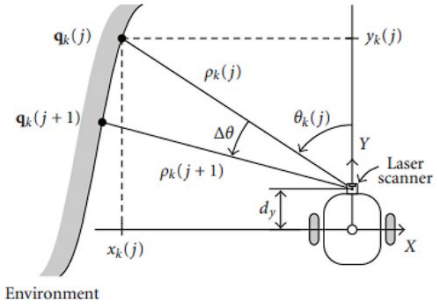


Bu sistemin nasıl çalıştığına gelecek olursak LiDARdan gelen veri noktalar halinde. Her ne kadar gelişigüzel noktalar gibi görünse de belli bir ortalamaları var ve bu üzerinde normal dağılım uygulanarak sistem modellenebiliyor. Aracın pozisyonunu bu dağılımın eşleştiği konuma göre anlayabiliyorsun.

Bir HD Map oluşturulmuş ve Lanelet2 kütüphanesiyle şeritler belirlenmiş. Bu kısımda ilk incelediğim araç olan Hezarfen Otonom'a benziyor. Bu verilerle birlikte araç, yollar hakkında fazlasıyla bilgiye sahip oluyor.

Planlayıcı ve kontrolcü

Bu aşamada hedef noktaya uygun rotayı oluşturmaya çalışıyoruz**.** Başlangıç ve bitiş noktaları laneletler yani şeritlerle belirleniyor. En kısa rota hesaplandıktan sonra noktalarla yol belirleniyor. İlerledikçe eski noktalar kayboluyor ve rota güncelleniyor. Aracın noktalara ulaşması için Pure Pursuit algoritması kullanılmış. Direksiyon açısını belirlemeye yaradığını öğrenmiştim.



Şekil 53, Pure Pursuit Geometrisi

NDT ile lokalizasyon

Aracın konumunu bulabilmesi ve pozisyonunu anlayabilmesi için için normal distribution transform kullanacağından bahsetmiştik. Lokalizasyon ile bunu bir adım öteye taşıyoruz. LiDARdan gelen veri ile haritadaki verinin eşleşmesi için probability density function(pdf) kullanılıyor. Anlık olarak bu haritaların karşılaştırılması sayesinde başka bir sensöre ihtiyaç duymadan haritadaki konumu tespit edebiliyoruz.

Karar verme algoritması

YOLOv5'ten gelen veriler kullanılarak şemada görüldüğü gibi bir karar mekanizması oluşturmuşlar. Artı olarak modelden gelen verinin yanlış olma ihtimaline karşı hafızada en çok olan levhaya göre hareket edecekmiş.