NEÜ MEKATRONİK MÜHENDİSLİĞİ

****

Mekatronik mühendisliği uygulamaları

Havin Gürkan

20010101041

İçindekiler

1. Projenin amacı
2. Projenin önemi
3. 1.Adım: Şerit Takip Sistemi
4. 2.Adım: Araç, Trafik ışığı ve Plaka Tespiti Uygulaması
5. 3.Adım : Mesafe Ölçümü ve Araç Takip Mesafesi Hesabı
6. 4.Adım: Hız Tespiti
7. 5.Adım: Projenin Tamamlanması
8. Karşılaşılan Hatalar ve Sorunlar
9. Colab IDE linkleri
10. Projenin amacı

Bu çalışmanın amacı, OpenCV ve Derin Öğrenme kullanarak bir video akışından araçları tespit etmek, takip etmek ve hızlarını tahmin etmektir. Bu amaç doğrultusunda şerit takibi, nesne tanıma, hız takibi, mesafe hesabı ve araç takip sistemi gibi konularda gelişmiş algoritmalar geliştirilmiştir. Bu çalışma, araç tespiti, takibi ve hız tahmini gibi önemli bilgisayar görüşü problemlerini ele almaktadır. Bu problemlere düşük güç tüketen çözümler sunulmaktadır. Bu çözümler, şu uygulama alanlarında kullanılabilir: trafik güvenliği, akıllı şehirler, otonom araçlar, hız kontrolü. Bu çalışma, bu alanda daha fazla araştırma yapılmasını teşvik edebilir ve yeni teknolojilerin geliştirilmesine katkı sağlayabilir. Elde edilen sonuçlar, geliştirilen sistemlerin gerçek zamanlı performans gösterdiğini ve yüksek doğruluk oranlarına sahip olduğunu göstermektedir.

1. Projenin önemi

Bu projenin bilimsel önemi, bilgisayar görüşü alanında yeni ve etkili yöntemler geliştirmesi ve mevcut yöntemlerin performansını artırmasıdır. Bu yöntemler, video akışından araç tespiti, takibi ve hız tahmini gibi zorlu problemleri çözmek için OpenCV ve Derin Öğrenme gibi güçlü araçları bir arada kullanmaktadır. Bu projenin sonuçları, bu alanda daha fazla araştırma yapılmasına ve yeni problemlere çözüm bulunmasına katkı sağlayabilir.

Bu projenin teknolojik önemi, düşük maliyetli ve düşük güç tüketen çözümler sunmasıdır. Bu çözümler, pahalı ve yüksek güç tüketen donanımlara ihtiyaç duymadan, video akışından araç tespiti, takibi ve hız tahmini yapabilen sistemler geliştirmeyi mümkün kılmaktadır. Bu çözümler, akıllı telefonlar, tabletler, dizüstü bilgisayarlar gibi taşınabilir cihazlarda da çalışabilir.

Bu projenin toplumsal önemi, trafik güvenliği, akıllı şehirler, otonom araçlar, hız kontrolü gibi birçok uygulama alanında kullanılabilir olmasıdır. Bu uygulamalar, insan hayatını kolaylaştırmak, trafik kazalarını azaltmak, çevreyi korumak, enerji tasarrufu sağlamak gibi amaçlarla geliştirilebilir. Bu projenin sonuçları, bu uygulamaların gerçekleştirilmesine ve yaygınlaştırılmasına yardımcı olabilir.

1. 1.Adım: Şerit Takip Sistemi

Bu adımda; bir video akışından araç şeritlerini tespit etmek, gelecek eğrileri tahmin etmek ve şerit eğriliğini ölçmek için bir işlem akışı içermektedir. Bu işlem akışı, kamera kalibrasyonu, bozulma düzeltmesi, perspektif dönüşümü, ikili görüntü oluşturma, şerit piksellerini bulma, şerit sınırlarını uydurma, şerit eğriliği ve araç konumu belirleme, şerit sınırlarını orijinal görüntüye geri dönüştürme ve şerit sınırları, şerit eğriliği ve araç konumu hakkında görsel ve sayısal bilgi sunma gibi adımlardan oluşmaktadır. Bu kod dosyası, bilgisayar görüşü alanında önemli problemlere çözüm sunan ve trafik güvenliği, akıllı şehirler, otonom araçlar, hız kontrolü gibi uygulamalarda kullanılabilmektedir.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 1: Proje klasörü içerisinden önemli bazı klasör dosyaları**

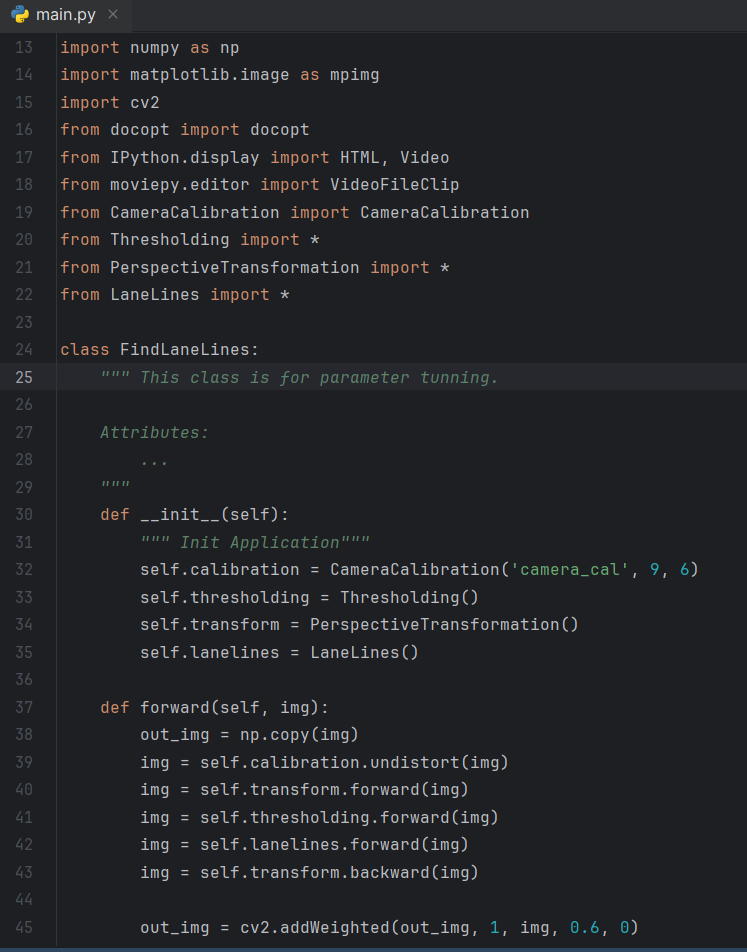
Klasör içerisindeki resimlerin amacı, kamera kalibrasyonu, perspektif dönüşümü ve algoritma performansı testi gibi işlemler için kullanılmaktadır.

“camera\_cal” dosyası içerisinde satranç tahtasından bazı kareler bulunmaktadır. Ayrıca “test\_images” dosyası içerisinde ise kullanılan videolarda bazı kareler bulunmaktadır. Klasörlerin önemi aşağıda açıklanmaktadır:

* **Satranç tahtası resimleri**: Kamera kalibrasyonu için kullanılır. Kameranın görüş alanını, odak uzaklığını ve merkez noktasını belirlemek için bilinen bir desen olan satranç tahtası resimleri kullanılır. Bu resimler, kameranın bozulmalarını düzeltmek ve perspektif dönüşümü yapmak için gerekli olan iç ve dış parametreleri hesaplamak için kullanılır.
* **Eğimli resimler**: Perspektif dönüşümü ve algoritma performansı testi için kullanılır. Eğimli resimler, algoritmanın farklı açılardan çekilmiş görüntülerde şerit tespiti yapabilmesini sağlar. Perspektif dönüşümü, eğimli resimleri düz bir görünüme dönüştürerek şerit sınırlarını daha kolay uydurmayı mümkün kılar. Algoritma performansı testi, eğimli resimlerde şerit tespiti yaparken algoritmanın doğruluğunu ve hızını ölçer.
* **Videodan kareler**: Algoritma performansı testi için kullanılır. Videodan kareler, algoritmanın hareketli görüntülerde şerit tespiti yapabilmesini sağlar. Algoritma performansı testi, videodan karelerde şerit tespiti yaparken algoritmanın doğruluğunu, hızını ve kararlılığını ölçer.

Kareleri olmayan bir video kullanırsanız, uygulama verimli çalışmaz. Çünkü kareler, videonun temel bileşenleridir. Kareler olmadan, video akışını işlemek mümkün değildir. Kareler, videonun zaman içindeki değişimini gösterir. Kareler, algoritmanın şerit tespiti yaparken şeritlerin konumunu, eğriliğini ve araç konumunu belirlemesine yardımcı olur.

Algoritmanın ana dosyası aşağıda açıklanmıştır.

 metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Açıklama otomatik olarak oluşturuldu

CameraCalibration: kamera kalibrasyonu için kullanılır.

Thresholding: görüntü işleme aşamasında eşikleme işlemlerini gerçekleştirir.

PerspectiveTransformation: görüntüyü perspektif açaısından düzeltmek için kullanılır.

LaneLines: şerit tespiti ve görselleştirmesi için kullanılır.

FindLaneLines: tüm bu işlemleri bir araya getirerek şerit tespiti uygulamasını yürütür. İçeriği kısaca aşağıda belirtilmiştir.

* forward metodu, bir görüntüyü şerit tespiti işleminden geçirir ve sonuç görüntüyü döndürür.
* process\_image metodu, bir giriş görüntü dosyasını alır, şerit tespiti uygular ve çıktıyı belirtilen dosyaya kaydeder.
* Process\_video metodu, bir giriş video dosyasını alır, her kareyi şerit tespiti işleminden geçirir ve çıktıyı belirtilenn dosyaya kaydeder.

main fonksiyonu:

* Docopt kütüphanesi kullanılarak komut satırı argümanları işlenir
* --video bayrağına göre, bir görüntü veya video dosyasının işlenmesi için FinndLaneLines metodunu çağırır.

Bu kod, kamera kalibrasyonu, perspektif dönüşümü, eşikleme ve şerit tespiti gibi adımları içeren bir şerit tespiti pippeline’ını uygular. Ayrıca girişin bir görüntü veya video dosyası olmasına bağlı olarak işlemi yönetir.

metin, araba, taşıt, araç, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 2:Projenin çıktısı olan videodan bir kare**

1. 2.Adım: Araç, Trafik ışığı ve Plaka Tespiti Uygulaması

Araç, Trafik ışığı ve Plaka Tespiti için aynı işlemler uygulanmıştır. Bu nedenle sadece plaka tanıma işlemi baz alınarak proje adımlarının açıklanması aşağıda yapılmıştır.

1. Roboflow ile veri seti oluşturuldu ve gerekli eğitim yapıldı.
2. Roboflow’dan alınan Apı adreslerinin olduğu kod model dosyasını oluşturma koduna entegre edildi.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. Colab model dosyasını oluştururken aşağıdaki çıktılar elde edildi.



1. Colab aracılığıyla oluşturulan “best.py” uzantılı model doyası pycharm ıde üzerinde videolar aracılığıyla test edilmiştir.

yol, karayolu, dış mekan, yol, yöntem, otoyol içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 3: Araç tespiti yapan kodun çıktısı olan videodan bir kare**

dış mekan, trafik ışığı, sinyal verme cihazı, hafif içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 4: Trafik ışığı tespitinden yapılan test sonuçları**

Bu işlemde olasılıkların düşük olmasının nedeni; az veri kullanımıdır. Proje kapsamında veri artışına ihtiyaç duyulmamıştır.

Bu projede genel olarak veri setinin %80’i eğitim için %20’si test için kullanılmıştır.

1. 3.Adım : Mesafe Ölçümü ve Araç Takip Mesafesi Hesabı

Bu adımda asıl amaç araç takip mesafesini ölçmektir. Fakat buu işlem video üzerinden yapılacağı için test amacıyla ilk olarak mesafe ölçümü ele alınmıştır.

* Mesafe ölçümü: Bu aşamada ilk olarak el tespit edililmelidir. Bu işlem için belirli fonksiyonlar kullanılmaktadır. Daha sonra elin üzerindeki pixellerin kendilerine ait numaraları bulunmaktadır. 2.(işaret parmağı) ve 5.(küçük parmak) pixelleri yardımıyla iki parmak arasındaki mesafe bulunur. Kameraya yaklaştıkça mesafe artar, uzaklaştıkça azalır. Bu mantık ile çalışan algoritmanın çıktıları mezura ile test edilmiş ve doğruluğu kanıtlanmıştır. Test aşamasından bazı görüntüler aşağıda verilmiştir.

metin, kişi, şahıs, bilgisayar, duvar içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Video bir araç içerisinde çekilmiştir. Bu nedenle kamera ile araç arasındaki mesafe göze alınarak işlem yapılmıştır. Video süresince anlık olarak mesafeler güncellenmektedir. Bu işlemi yaparken araç tanıma değil plaka tanıma kodundan yararlanılmıştır. Algoritmanın çalışma mantığı kısaca aşağıda açıklanmıştır.
  + - “calculate\_distance”, kameradan plakaya olan mesafeyi hesaplamak için kullanılan bir fonksiyondur.
    - Kamera kalibrasyonu için gerekli olan fokal uzaklık ve referans mesafe değerlerini tanımlar. Fokal uzaklık, kameranın odak noktasının merkezine olan uzaklığıdır. Referans mesafe, plakanın olduğu konumdan kameraya olan mesafedir. Bu değerler, gerçek bir senaryoda ölçülmesi gereken değerlerdir. Bu kodda, örnek olarak 1000 ve 50 değerleri kullanılmıştır.
    - Bir while döngüsü ile video akışını sürekli olarak işler. Bu döngüde, cap.read fonksiyonu ile video akışından bir kare alır. results = model(img, stream=True) satırı ile de bu kareyi YOLO modeline verir. Model, karedeki nesneleri tespit eder ve sonuçları results değişkenine atar.
    - plate\_center = ((x1 + x2) // 2, (y1 + y2) // 2) satırı ile sınırlayıcı kutunun merkez noktasını hesaplar. Bu nokta, plakanın merkezi olarak kabul edilir. plate\_width = x2 - x1 satırı ile de sınırlayıcı kutunun genişliğini hesaplar. Bu değer, plakanın genişliği olarak kabul edilir.
    - distance = calculate\_distance(focal\_length, reference\_width, plate\_width) satırı ile kameradan plakaya olan mesafeyi hesaplar. Bu hesaplamada, calculate\_distance fonksiyonu, fokal uzaklık, referans mesafe ve plaka genişliği değerlerini kullanır. Bu fonksiyon, basit bir orantı kurarak mesafeyi bulur. Mesafe, santimetre cinsinden hesaplanır.
    - cv2.circle(img, plate\_center, 5, (0, 255, 0), -1) satırı ile plakanın merkezine yeşil bir nokta çizer. cv2.putText(img, f’Mesafe: {int(distance)} cm’, (max(0, x1), max(50, y1)), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2) satırı ile de plakanın yanına mesafeyi beyaz bir yazı ile yazar. Bu şekilde, karedeki her bir plaka için mesafe hesaplanmış ve gösterilmiş olur.

 metin, dış mekan, kara taşıtı, araba içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 5: Araç Mesafe takibi**

1. 4.Adım: Hız Tespiti

Bu adımdaki amaç; YOLOv8 ve DeepSORT adlı iki derin öğrenme modelini birleştirerek gerçek zamanlı nesne takibi yapmaktır. Bu modeller, bir görüntü veya video akışından araç, trafik ışığı, yaya, hayvan gibi nesneleri tespit edebilir, kimliklerini ve izlerini koruyabilir, hızlarını ve sayılarını tahmin edebilir. Bu kod dosyası, bilgisayar görüşü alanında önemli problemlere çözüm sunan ve trafik güvenliği, akıllı şehirler, otonom araçlar, hız kontrolü gibi uygulamalarda kullanılabilen bir projedir. Kodun çalışma prensibi aşağıda kısaca açıklanmıştır.

* Öncelikle, kamera kalibrasyonu yaparak kameranın görüş alanını, odak uzaklığını ve merkez noktasını belirler. Bu parametreler, görüntüdeki nesnelerin gerçek dünya koordinatlarını hesaplamak için kullanılır.
* Sonra, YOLOv8 modeli ile görüntüdeki nesneleri tespit eder. Bu model, görüntüyü küçük bölgelere ayırır ve her bölgede nesne olup olmadığını, nesnenin sınıfını, konumunu ve boyutunu belirler. Bu model, nesneleri sınıflandırmak için 80 farklı kategori kullanır.
* Ardından, DeepSORT modeli ile görüntüdeki nesneleri takip eder. Bu model, nesnelerin kimliklerini ve izlerini korumak için bir Kalman filtresi ve bir derin öğrenme tabanlı özellik çıkarıcı kullanır. Bu model, nesnelerin hareketlerini tahmin eder ve görüntüdeki nesneler arasında bir eşleştirme yapar.
* Son olarak, nesnelerin hızlarını tahmin etmek için, nesnelerin gerçek dünya koordinatlarını, zaman damgalarını ve kare hızını kullanır. Nesnelerin gerçek dünya koordinatlarını, kamera kalibrasyonu parametrelerini ve perspektif dönüşümü matrisini kullanarak hesaplar. Nesnelerin zaman damgalarını, görüntü veya video akışının başlangıcından itibaren geçen süreyi ölçerek belirler. Kare hızını, görüntü veya video akışının saniyedeki kare sayısını ölçerek bulur. Nesnelerin hızlarını, nesnelerin gerçek dünya koordinatlarındaki değişimi, zaman damgalarındaki farkı ve kare hızını kullanarak hesaplar. Nesnelerin hızlarını, km/saat veya mil/saat gibi birimlerde gösterir.

**metin, yol, karayolu, dış mekan, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Şekil 6: Projenin video çıktısından bir görüntü**

1. 5.Adım: Projenin Tamamlanması

İlk 4 adım birleştirilerek proje tamamlanacaktır. Yapılan önemli işlemler aşağıda belirtilmiştir.

* output\_path adlı bir değişken tanımlar. Bu değişken, işlenmiş video dosyasının kaydedileceği yolu belirtir. Bu video dosyası, nesnelerin tespit edildiği, takip edildiği ve hızlarının tahmin edildiği bir video olacaktır.
* main fonksiyonunu çağırır. Bu fonksiyon, video dosyasını kare kare okur, her karede YOLOv3 ve DeepSORT modellerini kullanarak nesne tespiti ve takibi yapar, nesnelerin hızlarını tahmin eder, nesnelerin sınırlayıcı kutularını, kimliklerini ve hızlarını gösterir ve işlenmiş kareleri birleştirerek yeni bir video dosyası oluşturur.

metin, yol, karayolu, dış mekan, iki yeri birbirine doğrudan bağlayan yol içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yol, karayolu, sahne, yol, yöntem içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Şekil 7: Kodun çıktıları verilmiştir.**

1. Karşılaşılan Hatalar ve Sorunlar

* Şerit takip mesafesi olmak üzere ciddi kaynak sıkıntı sorunu
* Proje alt maddesi olan çoğu konu hakkında sadece demo olması
* Tensorflow başta olmak üzere çok fazla kütüphane kullanıldı ve hepsinin kabul ettiği python sürümünün farklı olması
* Pycharm IDE sürekli çözülemeyen hatalar vermesi

Bu hataların sonucu olarak çoğunlukla colab üzerinden çalışmalara devam edildi. Daha fazla araştırma yapılarak bilgi birikimimin artması ve kodlara yorum yapabilme, değiştirebilme, baştan yazabilme ile kaynak sıkıntısı sorunu çözülmüştür.

Colab kodlarının olduğu link:

<https://colab.research.google.com/drive/1LwfH64HgwYW4OkEu3R5TJNw95_kyfLtT#scrollTo=EP73rGlBIexM>

<https://colab.research.google.com/drive/1jpnofK1kXLIJAPuw9YIRS2Yf25gBWHRF#scrollTo=ITgb043YszrF>

<https://colab.research.google.com/drive/1nPGcKGB2ZGO1s8qlflXoWVpIkMlT2J5L#scrollTo=WkR9WDjCEn18>

<https://colab.research.google.com/drive/1TJMnmY5FBGWFGHX8RoOqV-9OOG3j4q2D>