TRAFİKTE YORGUNLUK TESPİT SİSTEMİ DOKÜMANI

Sema YEŞİLKAYA

Özet

Sürücünün yorgunluğunu algılama projesi, trafikteki kaza oranlarının belirli bir yoğunluğunun yorgun sürücülerden oluşması dolayısıyla önem kazanmaktadır.Bu projenin amacı yorgun sürücüleri algılayıp onları dinlenmeye yönlendirme veya araç kullanmayı bıraktırma gibi koşulların sıklığına bağlı olarak sonuçların da değişmesidir. Bu çalışamada Opencv kullanılmıştır. Opencv aracılığıyla yüzdeki bölgelerin seçilmesi sağlanıp frameler olarak işleme alınması sağlanmıştır.Çalışma yüksek doğrulukla çalışmaktadır. Gözlük,esnerken ağız kapatma, kişilerin göz farklılığı gibi durumlar göz önüne alınarak hazırlandığı için doğruluğa yakın olarak tasarlanmıştır.Sonuçlar da gösteriyor ki yorgun sürücülerde belirtilen koşullar ele alınırsa daha sağlıklı ve bilinçli bir trafikte bulunulacaktır.

Anahtar Kelimeler: Sürücü yorgunluğu, Opencv, Cascade.

İçindekiler Tablosu

1 Giriş

Günümüzde trafik güvenliği, toplumun en önemli meselelerinden biri haline gelmiştir. Trafik kazalarının önemli bir kısmının yorgun sürücülerden kaynaklandığı bilinmektedir. Bu çalışma, yorgun sürücüleri etkin bir şekilde tespit ederek trafikteki kaza oranlarını azaltmayı amaçlamaktadır. Yorgunluk, sürücünün tepki süresini yavaşlatır ve karar verme yeteneğini olumsuz etkiler, bu da potansiyel olarak tehlikeli durumlara yol açabilir. Bu nedenle, yorgun sürücülerin erken tespiti, trafik güvenliğini artırmada kritik bir rol oynamaktadır.

Bu çalışmanın motivasyonu, trafik kazalarını önleme ve yollarımızı daha güvenli hale getirme arzusundan kaynaklanmaktadır. Yorgunluk algılama sistemlerinin geliştirilmesi, sürücülerin dinlenmeye yönlendirilmesi veya araç kullanmayı bırakmaları konusunda uyarılmasını sağlayarak, trafikteki güvenliği artırabilir.

OpenCV kütüphanesi kullanılarak geliştirilen bu sistem, yüz tanıma ve görüntü işleme tekniklerini kullanarak sürücünün yorgunluk seviyesini belirlemektedir. Gözlük takma, esneme sırasında ağız kapatma ve kişisel göz farklılıkları gibi faktörler dikkate alınarak tasarlanmıştır. Bu sayede, sistem yüksek doğrulukla çalışmakta ve çeşitli koşullarda güvenilir sonuçlar üretmektedir.

Literatürdeki mevcut çalışmalar, yorgunluk algılama konusunda çeşitli yöntemler önermiş olsa da, bu çalışma, yüz tanıma teknolojisindeki son gelişmeleri ve görüntü işleme algoritmalarını birleştirerek, daha etkin ve doğru bir çözüm sunmayı hedeflemektedir. Bu çalışma, yorgun sürücüleri algılama konusunda literatüre önemli katkılarda bulunmayı ve trafik güvenliğini artırmayı amaçlamaktadır.

2 Yöntem

- Yüz tanıma: Cascade
  Cascade yüz tanıma yöntemi, nesne tanıma alanında oldukça yaygın olarak kullanılan bir yöntemdir. Temel olarak, bir yüzün farklı özelliklerini (gözler, burun, ağız vb.) tanımak için bir dizi önceden eğitilmiş sınıflandırıcıdan oluşur. Bu sınıflandırıcılar, yüzün farklı bölgelerini tarar ve yüzün varlığını veya yokluğunu belirlemeye çalışır. Cascade yöntemi, hızlı çalışma süresi ve yüksek doğruluk seviyeleri nedeniyle tercih edilir.

- Tanınan yüzden gözleri ayırt etme : Cascade
  Cascade yöntemi, yüz tanıma işlemi sırasında gözleri ayırt etmek için de kullanılır. Gözleri ayırt etmek için yine önceden eğitilmiş sınıflandırıcılar kullanılır. Bu sınıflandırıcılar, gözlerin varlığını veya yokluğunu tespit eder ve yüz tanıma sistemine daha fazla bilgi sağlar.

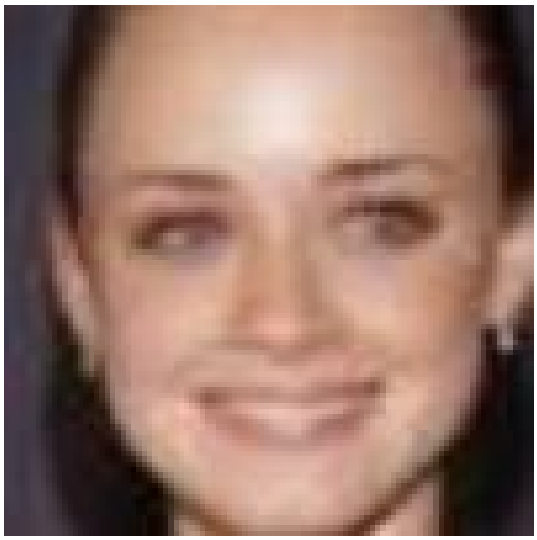
- Göz durumlarının sınıflandırılması : SVM+ HOG
  Destek vektör makinesi (Support Vector Machine,SVM), nesne tanıma ve sınıflandırma için kullanılır. Nesnelerin farklı bölümlerini (örneğin gözler, burun, ağız) tanımak için deformable part model kullanır. HOG, nesnelerin kenarlarını ve şekillerini tanımak için kullanılır. Göz durumlarını sınıflandırmak için kullanılabilir.

- Yorgunluk Tespiti : Perclos Sürücülerin yorgunluk seviyelerini tespit etmek için kullanılır. Sürücünün göz kapanma oranını (gözlerin ne kadar süreyle kapalı olduğunu)

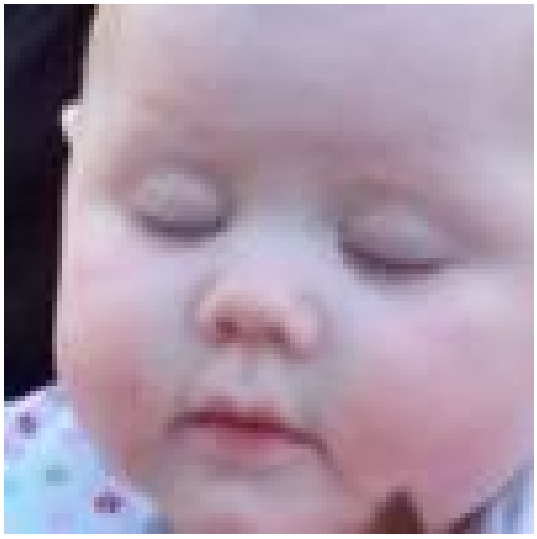
hesaplar. Yüksek Perclos değerleri, sürücünün yorgun olduğunu ve dikkatinin dağıldığını gösterebilir. Yöntemler genel olarak [1] baz alınarak belirlenmiştir.

$$P = \frac{\text{kapalı göz içeren kare sayısı}}{\text{toplam kare sayısı}} \times 100 \tag{1}$$

Yukarıdaki (1) numaralı eşitlik Perclos değerinin nasıl hesaplanacağını gösterir.



Şekil 1: Veri setinden açık gözlü yüz örneği [2].

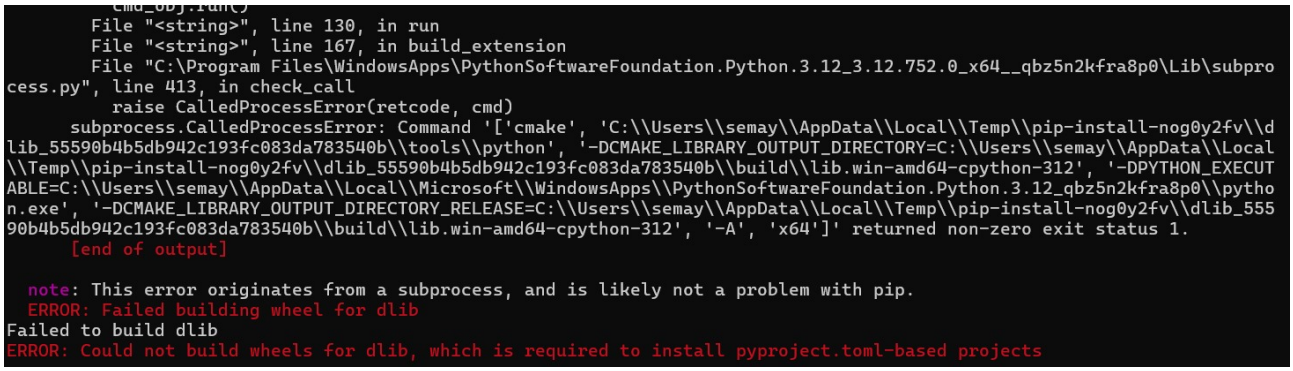


Şekil 2: Veri setinden kapalı gözlü yüz örneği [2].

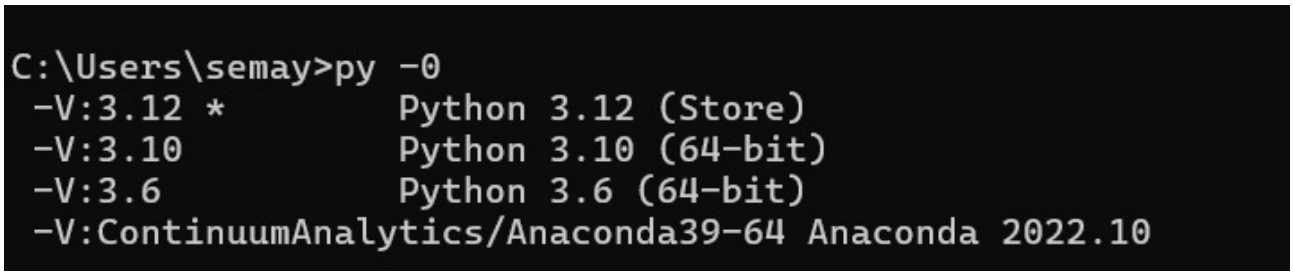
3 Bulgu ve Tartışma

Projede kullanabilmek için dlib kütüphanesi import edilmiştir . Dlib yüz tanımlama gibi pek çok yerde kullanılan geniş kapsamlı ve açık kaynaklı bir kütüphanedir.

Bilgisayarımda başka bir python sürümü(3.9) bulunması gerekmektedir ve bunu standart olarak kullanabilmek için path olarak eklenmiştir. Fakat yine de olumlu bir sonuç elde edilmemiştir.

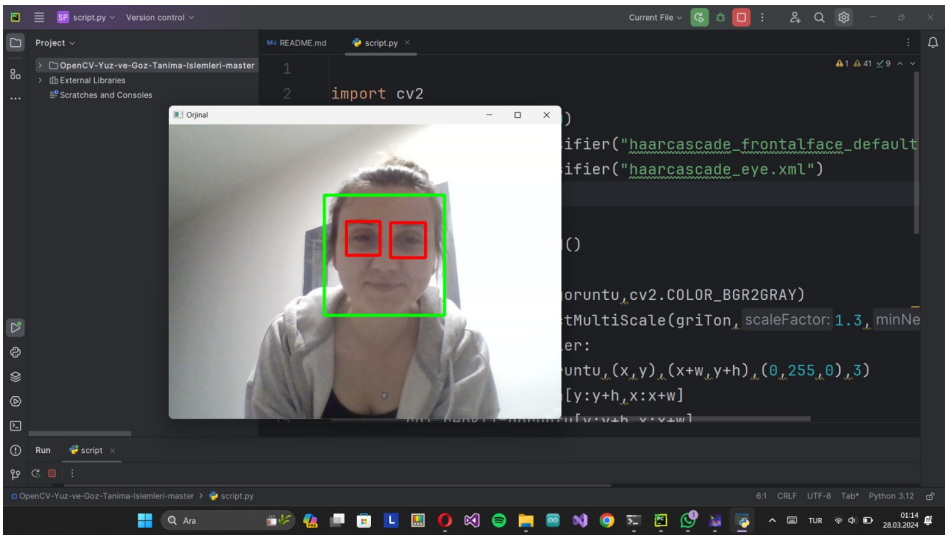


Şekil 3: Alınan hata.



Şekil 4: Python sürümleri.

Bilgisayarda yaşanan sürüm uyuşmazlığı çözülemediği için dlib olmadan bu projenin nasıl yapıldığı araştırıldı. Sadece Opencv import ederek yani import cv2 komutuyla yapılan bir kod örneği incelendi ve Haarcascade ile de çalışıldığını görüldü.
Haarcascade için hazır kodların yanında kendimiz de oluşturabiliriz. Pozitif ve negatif resimlere ihtiyacımız olur. Bu resimleri etiketleme işlemi gibi işlemden geçirdikten sonra bir tarafa nesnenin varlığını diğer tarafa olmadığını tanıtarak bu belgeler .bat formunda kaydedilir. Sistem tarafından çalıştırılıp tanınması için de .xml uzantısı olarak değiştirilir. Bu değişimi gerekli araçlarla yapabilirsiniz. [3] burada anlatılan sistemi yararlı bulunmuştur ve ilerleyen süreçte bu veri setini kullanılacaktır.



Şekil 5: Yüz ve göz tespiti.

Bu sisteme göz kapalılığı da eklenecektir.

1 Göz Kapalılığı Tespiti

Göz kapalılığını hesaplarken gözün etrafında bulunan 6 noktadan faydalanılır. Bu noktalar arası uzaklığını azalması ve belirli eşik değerin altında belirli bir süre kalması durumunda yorgunluk tespit edilmiş demektir. Bu duruma göre numpy ekledim ve hesaplama yaparak göz açıklık kapalılık kodunu yazmaya çalıştım. Kodun çalışması için dlib gerekli.

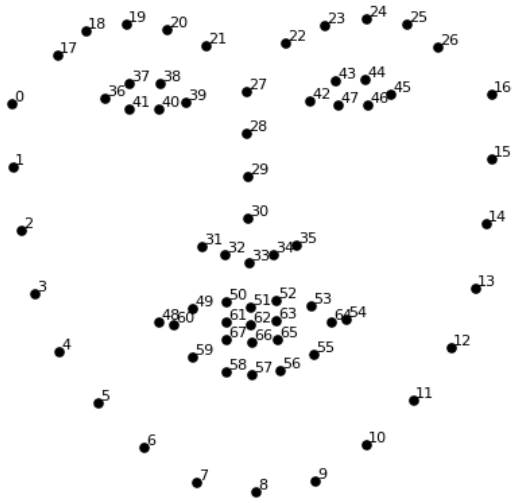
Dlib kütüphanesini yüklemekle ilgili sorun vardı ve bu sorun çözüldü. Sorunu çözerken işlemlere baştan başlandı.Öncelikle VsCode derleme araçlarını indirildi. Cmake kuruldu. Daha sonra pip install dlib komutuyla bu kütüphaneyi indirildi [4].İndirip örnek projelerde kullanmaya devam edebilebilirdi fakat dlib olmadan kullanılan projede denendi. Gözün open- close durumlarında hata alındı.

Dlib kütüphanesinde landmark denen bir özellik vardır.Dlib kütüphanesindeki "landmark" terimi, yüzdeki belirli noktaları veya bölgeleri tanımlamak için kullanılır.Dlib'in en yaygın kullanılan yüz landmark dedektörü, 68 noktalı bir yapıya sahiptir ve yüzün ana hatlarını hızlı ve güvenilir bir şekilde tespit edebilir.

Göz mesafesini hesaplamak için kullandığım standart olarak kullanılan eşik değer sabittir.İlerleyen süreçte bu eşik değer değiştirelecektir. Kullanıcıdan ilk önce gözünün açık ve kapalı hallerini isteyerek ona göre hesaplamaya geçilecektir.

```python
def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    dis = (A + B) / (2.0 * C)
    return dis
```

Şekil 6: Göz mesafesi hesaplama.

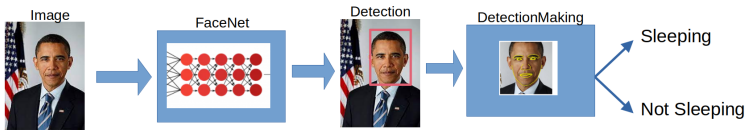


Şekil 7: Göz tespiti için Dlib kütüphanesinin kullandığı landmark değerleri[5].

Projede kullanılan yöntem geleneksel yöntemdir. Bu süreçte amaç, bu yöntem(Haarcascade) ile yeni teknolojilerden birini karşılaştırıp hangisinin daha sağlıklı sonuç verdiğini görmek. Yeni yöntem olarak CNN kullanılan bir örnek proje [6] bulundu. CNN, yani Evrişimli Sinir Ağları (Convolutional Neural Networks), görüntü işlemede kullanılan bir derin öğrenme algoritmasıdır. Görselleri girdi olarak alır ve farklı operasyonlarla bu görsellerdeki özellikleri (features) yakalar ve sınıflandırır [7]. CNN, özellikle görüntü tanıma, nesne tespiti ve benzeri görsel tabanlı görevlerde etkili bir şekilde kullanılır. CNN'nin temel bileşenleri şunlardır:

Evrişim Katmanı (Convolutional Layer): Görüntü üzerinde gezinen ve özellikleri yakalayan filtreler içerir.
Aktivasyon Fonksiyonu (ReLU gibi): Modelin doğrusal olmayan özellikleri öğrenmesini sağlar.
Havuzlama Katmanı (Pooling Layer): Görüntü boyutunu küçültür ve özellikleri yoğunlaştırır.
Tam Bağlantılı Katman (Fully Connected Layer): Öğrenilen özellikleri kullanarak sınıflandırma yapar [8].

Bu katmanlar, bir görüntünün temel özelliklerinden daha karmaşık özelliklere kadar çeşitli soyutlama seviyelerinde özellikleri öğrenmek için bir arada çalışır. CNN modelleri, bu özellikleri öğrenmek ve güncellemek için eğitim sürecinde sürekli olarak kendilerini iyileştirir.



Şekil 8: CNN ile sürücü yorgunluğu tespiti [9].

Tablo 1: Haarcascade ve CNN Karşılaştırması.

Özellik	**Haarcascade**	**CNN**
Hızlı tespit süresi	Evet	Hayır
Düşük kaynak gereksinimi	Evet	Hayır
Yüksek doğruluk	Hayır	Evet
Özellik mühendisliği gerekmez	Hayır	Evet
Yüksek hesaplama gücü gereksinimi	Hayır	Evet
Uzun eğitim süresi	Hayır	Evet

Yukarıda gördüğünüz üzere CNN daha iyi bir sonuç çıkarıyor. Eğer doğruluk önemsenmeseydi Haarcascade kullanılabilirdi.CNN ile yapılmış projede Tensorflow ve Theano yüklemesinde sorun yaşanıldığı için çalıştırılamadı. Cascade için de başın pozisyonunu algılayıp üç boyutlu sisteme dönüştürerek vücut postürünü algılayan fonksiyon eklendi.

Yeni CNN kodunda Tenforflow sürüm hatası alındı. Thaeno kütüphanesini Numpy ile benzer olması dolayısıyla kaldırıldı.Kod sahibi de Numpy kullanmıştı zaten.Ayrıca göz aralıklarını hesaplayarak gözün kapalı açık durumunu tespit ettiğimiz landmark olarak adlandırdığımız noktalar mantığında ve yine aynı yöntemle, ağız açıklığını tespit eden fonksiyonu da ekleyerek çoklu faktör kontrolü sağlandı. Bu değerlerden 3 veya daha fazlası gerçekleşiyorsa kişi yorgun olarak tanımlanıp konumuna göre en yakın otele yönlendirme yapılıyor.

```
if(map_counter>=3):
    map_flag=1
    map_counter=0
    vlc.MediaPlayer('take_a_break.mp3').play()
    webbrowser.open("https://www.google.com/maps/search/hotels+or+motels+near+me")
```

Şekil 9: Yorgunluk eşiğine göre otele yönlendirme[6].

```
def getFaceDirection(shape, size):
    image_points = np.array( object: [
                                shape[33],    # Nose tip
                                shape[8],     # Chin
                                shape[45],    # Left eye left corner
                                shape[36],    # Right eye right corner
                                shape[54],    # Left Mouth corner
                                shape[48]     # Right mouth corner
                            ], dtype="double")

    # 3D model points.
    model_points = np.array([
                                (0.0, 0.0, 0.0),             # Nose tip
                                (0.0, -330.0, -65.0),        # Chin
                                (-225.0, 170.0, -135.0),     # Left eye left corner
                                (225.0, 170.0, -135.0),      # Right eye right corne
                                (-150.0, -150.0, -125.0),    # Left Mouth corner
                                (150.0, -150.0, -125.0)      # Right mouth corner

                            ])
```

Şekil 10: Baş pozisyonu tahmini için tanımlamalar[6].

2 Eşik Değerin Kişileştirilmesi

Her insanın göz boyutu farklıdır. Bu durum göz tespit algoritmalarında sorun çıkarmaktadır. Örneğin gözün açıklık kapalılık durumuna göre yorgunluk seviyesi algılanması olgusunda verilen eşik değere göre normalde gözü açık bir bireyin gözü kapalı algılanması olası bir durumdur.

Belirli bir eşik değere göre sürücünün göz durumunu tespit etmek sağlıklı bir yöntem değildir. Bu sebeple belirli bir eşik değer yerine sürücünün gözünün açık ve kapalı fotoğrafını alıp oranlayıp yeni eşik değer oluşturulmasıyla hesaplanan bir durum daha net sonuçlar elde edilmesine sebep olacaktır.

Kapalı gözün tespit edilmesiyle kapalı olduğu süre de tespit edilebilir. Böylece sürücüye örneğin 3. saniyede odaklanın gibi bir uyarı verilir. Uyarı verilirken pygame kütüphanesini kullanıldı. Playsound kütüphanesi yaygın kullanılıyordu fakat entegre etmekte hata alındı ve hata ile uğraşmak yerine alternatiflerini araştırmak tercih edildi. Alternatif olarak da pygame kullanıldı.

```python
# Kullanıcıdan EAR için açık ve kapalı göz verilerini toplamak
def collect_eye_data():
    # Kullanıcıya gözlerini açık tutması söylenir
    input("Gözlerinizi açık tutun ve 'enter' tuşuna basın.")
    _, acik_goruntu = kamera.read()
    acik_ear = process_image(acik_goruntu)

    # Kullanıcıya gözlerini kapatması söylenir
    input("Gözlerinizi kapalı tutun ve 'enter' tuşuna basın.")
    _, kapali_goruntu = kamera.read()
    kapali_ear = process_image(kapali_goruntu)

    # Özel eşik değeri hesaplanır
    ozel_esik_degeri = (acik_ear + kapali_ear) / 2
    return ozel_esik_degeri
```

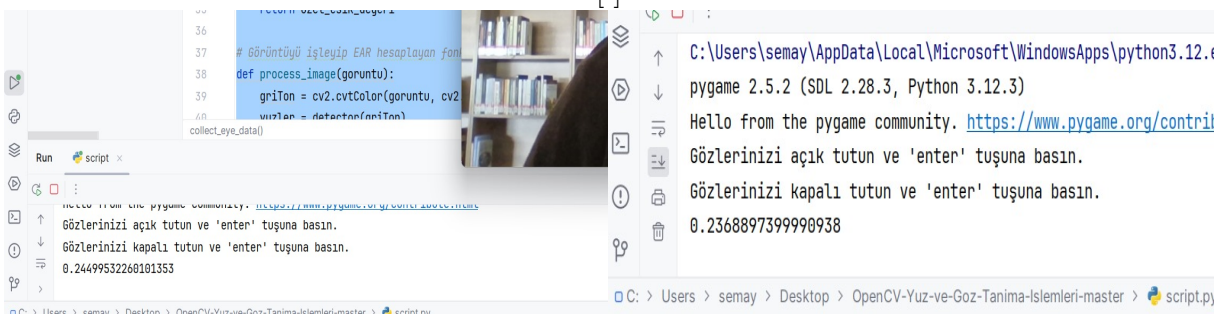
Şekil 11: Kişiye özel eşik değerin oluşturulması.

```python
if ear < ozel_esik_degeri:
    kapali_sayac += 1
    if kapali_sayac >= kapali_zaman * 30:  # FPS varsayılan olarak 30 kabul edilir
        pygame.mixer.music.play()  # Ses dosyasını çal
        kapali_sayac = 0  # Sayacı sıfırla
else:
    kapali_sayac = 0  # Göz açık ise sayacı sıfırla
```

Şekil 12: Gözlerin kapalı olduğu süreyi hesaplayıp sürücüyü uyarma.

3 Eşik Değere Göre Yorgunluk Durumunun Kategorileştirilmesi

[!]



(a) (b)

Şekil 13: Özel eşik değerlerin alınması örnekleri.

Tablo 2: Yorgunluk Durumları.

Açıklık Oranı	**Tanımlama**	**Yapılacak**
%60	Hafif yorgun	3 tekrarda sürüşe odaklanma uyarısı
%50	Tespit edilen eşik değer	Bu değere göre oranlama
%40	Çok yorgun	Otele yönlendirme

Kişiye özel olarak aldığımız eşik değer aslında tam açık (%100) açık gözün yarısı değerini almaktaydı. Bu değer kişiye özgü açıklıklar alınıp hesaplandığı için belirli bir değere göre sınıflandırmak yerine bu değerin yüzdeliklerini alarak sınıflandırmak daha sağlıklı sonuçlar doğuracaktır. Açıklık oranı %70 olan bir gözün hafif yorgun olarak belirlenmesi ve bu durumun 3 kez tekrarlanması üzerine "Sürüşe odaklan!" seslendirmesini yapan bir alarmın çalmasına karar verdim. Açıklık oranı %30 olan bir gözün çok yorgun olarak tanımlanması üzerine aracın hareketine son verilip uygun bir otele yönlendirme yapılmasına karar verdim.

4 Esneme Algılama

Göz dışında yorgunluğu tespit etmek için hangi durumların kontrol edileceğine bakıldı. Esneme durumu ve başın pozisyonunu zaten proje kapsamında ama bu hafta nabız alma durumu da araştırıldı. Koltuktan alınabilecek tek nabız diz kapağı arkasından geçen damarda. Bu projenin ilerlenmesi durumunda atılacak adımlardan biridir bu sebeple ileriy dönük bir araştırma olmuştur. Ayrıca ağız yani esneme durumunu ele alındı. Göz tespit etme ile aynı ilerlendi. Dlib kütüphanesinin sunduğu landmarkları tanımlayarak onları tespit etmek ve bir eşik değere göre esneme var yok hesabı yaptırmak gibi özet geçilebilecek bir aşama gerçekleştirildi.

```
# Ağız açıklık oranını hesaplayan fonksiyon
1 usage
def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[13], mouth[19]) # 52-56
    B = dist.euclidean(mouth[14], mouth[18]) # 50-58
    C = dist.euclidean(mouth[15], mouth[17]) # 51-57
    D = dist.euclidean(mouth[12], mouth[16]) # 48-54
    MAR = (A + B + C) / (2.0 * D)
    return MAR
```

Şekil 14: Ağzı tespit etmek için kullanılan fonksiyon.

Şekil 15: Nabız alınabilecek yerler [10].

5 Baş Pozisyonu Belirleme

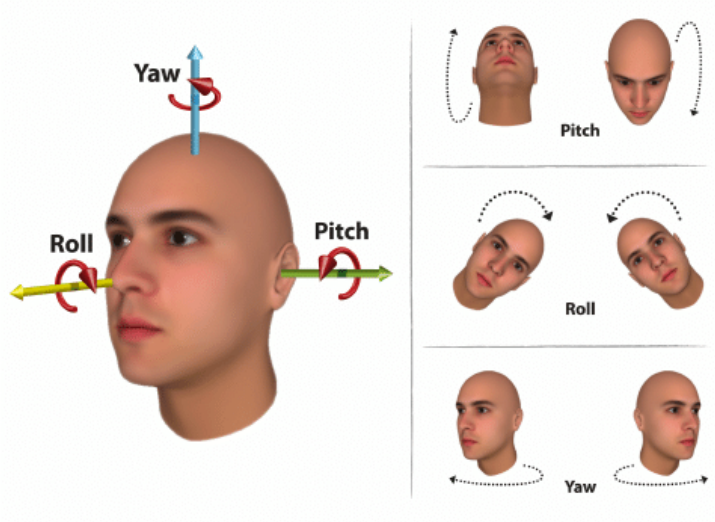
Bu hafta başın pozisyonunu da projeye ekleyerek belirlediğim parametreleri tamamlayıp bu parametreler bir arada değerlendirmeye alınmıştır. Başın pozisyonunu belirleyebilmek için öncelikle 2 boyutlu olarak yüzden aldığımız noktaların 3 boyutlu eksene çevrilmesi gerekir. Bu çevirme için bir model kullanıldı. Aynı zamanda bu dönüşümlerin yapılması için de bazı kavramlar bilinmelidir.

```
def getFaceDirection(shape, size):
    image_points = np.array( object: [
                                shape[33],    # Nose tip
                                shape[8],     # Chin
                                shape[45],    # Left eye left corner
                                shape[36],    # Right eye right corner
                                shape[54],    # Left Mouth corner
                                shape[48]     # Right mouth corner
                            ], dtype="double")

    # 3D model points.
    model_points = np.array([
                                (0.0, 0.0, 0.0),             # Nose tip
                                (0.0, -330.0, -65.0),        # Chin
                                (-225.0, 170.0, -135.0),     # Left eye left corner
                                (225.0, 170.0, -135.0),      # Right eye right corne
                                (-150.0, -150.0, -125.0),    # Left Mouth corner
                                (150.0, -150.0, -125.0)      # Right mouth corner

                            ])
```

Şekil 16: Baş pozisyonunu belirlemek için üç boyutlu model[11].

- Yaw (Y): Yaw, nesnenin yatay eksende (genellikle zemin düzlemi etrafında) dönme hareketini ifade eder. Bir uçağın yatay eksende sola veya sağa dönmesi gibi bir örnektir.
- Pitch (P): Pitch, nesnenin dikey eksende (genellikle burnun yukarı veya aşağı doğru hareketi) dönme hareketini ifade eder. Bir uçağın burnunun yukarı veya aşağı doğru eğilmesi gibi bir örnektir.
- Roll ®: Roll, nesnenin uzun ekseni etrafında dönme hareketini ifade eder. Bir uçağın kanatlarının bir tarafı yukarı doğru kalkarken diğer tarafı aşağı doğru inmesi gibi bir örnektir.

Bu kavramlara göre euler açıları hesaplanır. Euler açıları bu proje için başın pozisyonunu belirlemede kullanılmaktadır.



Şekil 17: Baş pozisyonunu anlamak için bilinmesi gerekenler.[12].

```
Traceback (most recent call last):
  File "C:\Users\semay\Desktop\OpenCV-Yuz-ve-Goz-Tanima-Islemleri-master\scr
    euler_acilari = bas_pozisyonu(landmarks)
                    ^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\semay\Desktop\OpenCV-Yuz-ve-Goz-Tanima-Islemleri-master\scr
    euler_acilari = cv2.decomposeProjectionMatrix(rotasyon_matris)[-1]
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
cv2.error: OpenCV(4.9.0) D:\a\opencv-python\opencv-python\opencv\modules\cal
```

Şekil 18: Euler ölçülerinde hata.

6 Baş Pozisyonu Belirleme

- SolvePnP Fonksiyonu Çağrısı: cv2.solvePnP(modelnoktalari, imagepoints, kameramatrisi, bozulmakatsayilari) satırı, 3D model noktalarını (örneğin, bir nesnenin köşe noktaları), görüntüdeki bu noktaların eşlenik 2D görüntü noktalarını, kamera matrisini ve bozulma katsayılarını kullanarak baş pozisyonunu tahmin eder.
- model noktalari: 3D modelin dünya koordinatlarındaki noktaları.
- image points: Görüntüdeki bu noktaların eşlenik 2D görüntü noktaları.
- kamera matrisi: Kamera kalibrasyonundan elde edilen iç parametre matrisi.
- bozulma katsayilari: Kamera bozulma katsayıları.
- Rotasyon Vektörünün Euler Açılarına Dönüştürülmesi: cv2.Rodrigues(rotasyonvektor) satırı, rotasyon vektörünü 3x3 rotasyon matrisine dönüştürür. Bu matris, nesnenin dünya koordinatlarındaki rotasyonunu temsil eder.
- Projeksiyon Matrisinin Oluşturulması: projeksiyonmatris = np.hstack((rotasyonmatris, translasyonvektor)) satırı, rotasyonmatrisi ve translasyon vektörünün yatay olarak birleştirilmesiyle projeksiyon matrisini oluşturur.
- Projeksiyon matrisi, nesnenin dünya koordinatlarından görüntü koordinatlarına dönüşümü için kullanılır.
- Projeksiyon Matrisinin Decompose Edilmesi: cv2.decomposeProjectionMatrix(...) satırı, projeksiyon matrisini iç parametre matrisi, rotasyon matrisi, translasyon vektörü, Euler açıları ve diğer bileşenlere ayırır.
- euleracilari.ravel() ile Euler açıları döndürülür.

```
 [309 387]]
Traceback (most recent call last):
  File "C:\Users\semay\Desktop\OpenCV-Yuz-ve-Goz-Tanima-Islemleri-master\script.py", line 133, in <modu
    mouth = get_mouth_shape(landmarks)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\semay\Desktop\OpenCV-Yuz-ve-Goz-Tanima-Islemleri-master\script.py", line 113, in get_m
    return np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in range(48, 68)])
                      ^^^^^^^^^^^^^^
AttributeError: 'numpy.ndarray' object has no attribute 'part'
```

Şekil 19: Landmarksların numpya dönüştürme hatası.

```
-       -
 [262 411]]
Ağız bölgesinde yeterli sayıda nokta bulunamadı.

Process finished with exit code 0
```

Şekil 20: Ağız bölgesi bulunamadı hatası.

7 Hataların Giderilip Baş Pozisyonunun Eklenmesi

Ağız durumunu hesaplarken alınan hata landmark sayısını kontrol etmek için fonksiyon içerisine yerleştirilen if kontrolü sebebiyle olduğu anlaşılmıştır. Bu kontrolü kaldırınca gerekli esneme algılaması sağlanmaktadır.

```
# Ağız açıklık oranını hesaplayan fonksiyon
1 usage
def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[13], mouth[19]) # 52-56
    B = dist.euclidean(mouth[14], mouth[18]) # 50-58
    C = dist.euclidean(mouth[15], mouth[17]) # 51-57
    D = dist.euclidean(mouth[12], mouth[16]) # 48-54
    MAR = (A + B + C) / (2.0 * D)
    return MAR
```

Şekil 21: Ağız hesabının yapılması.

Landmarkların farklı işlemlere tabii tutulması sebebiyle iki farklı landmark tanımlaması uygulandı. Biri baş pozisyonu için rotasyonlar uygulanacak biri de görüntü alınarak ağız ve göz bölgesi için gerekli oranlamalar yapılacak şekilde ayarlanmıştır.

```
_, goruntu = kamera.read()
griTon = cv2.cvtColor(goruntu, cv2.COLOR_BGR2GRAY)
yuzler = detector(griTon)
for yuz in yuzler:
    landmarkss = predictor(griTon, yuz)
    mouth = get_mouth_shape(landmarkss)
    mar = mouth_aspect_ratio(mouth)
    landmarks = np.array([(p.x, p.y) for p in predictor(griTon, yuz).parts()])
    euler_acilari = bas_pozisyonu(landmarks)
```

Şekil 22: Landmarkların ayarlanması.

Başın poziyonunu algılamak için eklenen fonksiyon ve matris işlemlerinde araştırmalara göre eksik bir bölüm olduğu fark edildi. Rcv2.Rodrigues fonksiyonu ile rotasyon vektöründen rotasyon matrisine dönüşüm yapılması gerekmektedir.

```
    # Euler açılarına dönüştür
    euler_acilari = cv2.Rodrigues(rotation_vector)[0]
    return euler_acilari
```

Şekil 23: Göz tespiti için Dlib kütüphanesinin kullandığı landmark değerleri.

Gerekli dönüşümler yapıldıktan sonra uyuklama kontrolü için başın pozisyonlarından pitch değeri kullanılmaktadır.Bu değer dışındaki değer ileriye dönük olarak kullanılabileceği gibi şu an pitch değeri proje için yeterli bir kontrol sağlamaktadır.

```
#uyuklama kontrolü
def check(euler_acilari):
    pitch = euler_acilari[0, 0] * (180 / np.pi)
    # Başın aşağı eğilme derecesi (pitch) ile uyuklama
    if pitch < 10:
        return "Uyuklama tespit edildi!"
    else:
        return "Uyuklama tespit edilmedi."
```

Şekil 24: Göz tespiti için Dlib kütüphanesinin kullandığı landmark değerleri.

4 Sonuç

Projede sürücülerin yorgunluk durumlarının algılanması sorununa çözüm üretilmiştir. Göz kapalılık durumu tespiti, ağza bakılarak uykululuk tespiti ve başın pozisyonuyla vücut duruşu tespiti sağlandı. Göz tespitinin kişiye özel olması sebebiyle yüksek bir başarım oranı sağlanmıştır. Diğer çalışmalarda geleneksel yöntemle ilerlenmeyip landmark konusu kullanılmamıştır. Bu nedenle Haarcascade yöntemi ile referans alınan projeler olsa da ilerlemelerin sağlanmadığı görülmüştür. Başın pozisyonu, esneme algılama ve kişiye özel göz tespiti uygulamaları farklılık göstermektedir.

Referanslar

[1] MonaOmidyeganehShervinShirmohammadiBehnooshHariri, "Yawning detection dataset." `http://ieee-dataport.org/open-access/yawdd-yawning-detection-dataset`, Sun, 12/13/2020. 21.03.2024.

[2] M. R. Lab, "Mrl eye dataset." `http://mrl.cs.vsb.cz/eyedataset`, 2021. 21.03.2024.

[3] Dasar, "driver dataset." https://www.mediafire.com/file/1aq02tpidk105fv/dasar_haartrain.rar/file, Sun, 12/13/2020. Erişim tarihi: 21.03.2024.

[4] O. Şahin, "Windows'a dlib kütüphanesini kurma." https://onursahin.net/windowsa-dlib-kutuphanesini-kurma/, 2024. Erişim tarihi: 03.04.2024.

[5] B. Amos, "The 68 landmarks detected by dlib library." https://www.researchgate.net/figure/The-68-landmarks-detected-by-dlib-library-This-image-was-created-by-Br fig2_329392737, 2018. Erişim tarihi: 03.04.2024.

[6] V. Dhawan, "Driver drowsiness detection." https://github.com/SuperThinking/driver-drowsiness-detection, 2024. Erişim tarihi: 13.04.2024.

[7] T. Ergin, "Cnn nedir." https://medium.com/@tunceregin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad, 2024. Erişim tarihi: 13.04.2024.

[8] Özgür Doğan, "Cnn nedir." https://teknoloji.org/cnn-convolutional-neural-networks-nedir, 2024. Erişim tarihi: 13.04.2024.

[9] Erkan, "Driver drowsiness detection system process diagram." https://github.com/eadali/pytorch-drowsiness-detection/blob/main/doc/process.png, 2024. Erişim tarihi: 03.04.2024.

[10] "Facebook'ta tuyo life tarafından paylaşılan görsel." https://www.facebook.com/tuyolife/photos/a.758789660963596/2140828869426328/?type=3. Erişim tarihi: 14.05.2024.

[11] neelanjan00, "Driverdrowsinessdetection." https://github.com/neelanjan00/Driver-Drowsiness-Detection.git, 2020/ 2022. 21.03.2024.

[12] E. A. R. M. D. R. M. B. G. D. C. Cavalcanti, "Enhanced real-time head pose estimation system for mobile device." https://www.researchgate.net/figure/Orientation-of-the-head-in-terms-of-pitch-roll-and-yaw-movements-describing-the-t fig1_279291928, 2014.

[13] aayushrai, "Driversafety." https://github.com/aayushrai/Driver_safety.git, 2019. 21.03.2024.

[14] bulentsezen, "tensorflowderinogrenme." https://github.com/bulentsezen/tensorflow_derin_ogrenme.git, 2022/2024. 21.03.2024.

[15] F. Cogen, "Driver fatigue detection with image processing." https://www.researchgate.net/publication/345253476_Driver_fatigue_detection_with_image_processing, 2020. 21.03.2024.

[16] M. Copilot, "Microsoft copilot: An ai-powered code completion tool." https://www.microsoft.com/en-us/ai/copilot, 2024. Erişim tarihi: 13.04.2024.