GÖRÜNTÜLERDEN ŞERİT ALGILAMA PROBLEMİNİN ÇÖZÜMÜ

Mürvet Nur ŞEN, Rabia GÜNEŞ, Şevki KARAGÖL, Ümmühan TEPEBAŞ
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
Kocaeli,Türkiye

ÖZET- Bu projede, projeyi yapan kişilerden görüntü işleme algoritmalarının çalışma mantığının anlaşılması ardından bu algoritmaların kullanımı ile yol görüntüleri üzerinden şerit algılama probleminin çözümüne yönelik bir sistem tasarlanması istenmektedir. Bu tür sistemler genellikle yapay zeka ile otomobil teknolojisini buluşturan otonom araçlarda kullanılmaktadır. Kullanım amacı ise otonom sürüş seviyelerine göre farklılık göstermektedir. İnsan kontrolüne gerek kalmayan tam otonom seviyede aracın sürücüsüz şekilde ilerlemesini sağlamak için, insan faktörünün ön planda olduğu seviyelerde ise şerit takip asistanı gibi yardımcı sistemlerler ile insan kontrolüne yardımcı olmak, güvenli ve sorunsuz sürüşü sağlamaktır.

Anahtar kelimeler- şerit, otonom sürüş, algılama, yapay zeka, yol, görüntü işleme.

I.GİRİŞ

Geçmişten günümüze trafik kazaları incelendiğinde sürücü hatası kaynaklı kazaların yaşanma sıklığının oldukça fazla olduğu görülmektedir. Teknolojinin zaman içindeki gelişimi göz önüne alındığında, yeni nesil araçların sürücülere sağladığı olanaklar artmaktadır. Yeni nesil araçlar, özellikle otonom araçlar, araç park ve fren sistemleri ile birlikte kaza oranlarını düşürmektedir. Günümüzde de otonom araçlara entegre edilen ve edilmeye devam edilecek olan şerit tanıma sistemi, sürücülerin aracı geleneksel şekilde kullanması alışkanlıklarını değiştirmekte ve araç çevresindeki şeritlerin gerçek zamanlı tespitini sağlamaktadır. Genel olarak şerit takip sistemi, seyir halindeyken aracın bulunduğu şeridi görsel olarak tanıyan ve araç şeritten belirli bir yanal hızda ayrıldığında sürücüyü uyaran bir sistemdir.

Makineler için atfedilen yapay zeka kavramı, insan zekası temel alınarak oluşturulmuştur. Ancak gelişmiş bilgisayarların ve uzman sistemlerin yazılımsal olarak kullanımının artması ve yaygınlaşmasıyla birlikte yapay zeka hem kavramsal olarak hem de uygulama olarak karşımıza çıkmaktadır. [1].

Bu sistemin gerçeklenmesi için de genellikle yapay zeka tabanlı, açık kaynak kodlu bir görüntü işleme kütüphanesi olan "OpenCV" kullanılmaktadır. Görüntü işleme, temel olarak, görüntülerden elde edilmek istenen özelliklere doğrudan ulaşılabilmesini veya bu özelliklerin öne çıkarılabilmesini sağlamak için algoritmaların kullanılarak görüntülerin işlenmesi sürecidir. Bu sürece günümüzde resim veya videoların veri olarak kullanıldığı derin öğrenme yapılarında eğitimin daha başarılı olması için ihtiyaç duyulmaktadır.[2]. Bu çalışmada da, veri seti için hazır olan yol görüntüsü üze-

rinden şerit tespitini sağlamak amacıyla görüntü işleme kütüphanesi olan "OpenCV" kullanılmaktadır.

II.HAZIRLIKLAR VE BİLGİLER

Kodlama aşamasına geçmeden önce grup üyeleri arasında görev dağılımı yapılmıştır. Yapılan görev dağılımına göre konu hakkında kaynak taraması aşamasına geçilmiştir. Kaynak taraması sonrasında elde edilen bilgiler ışığında projenin gerçeklenmesi aşamasında uygulanması gereken aşamalar ve kullanılacak yapılar belirlenmiştir. Ardından kodlama aşamasına geçilmiştir. Kodlama aşamasında Visual Studio Code kodlama platformu kullanılmıştır. Uygulamanın çalıştırılabilmesi için gerekli kütüphaneler command prompter aracılığıyla bilgisayara kurulmuştur ve Visual Studio Code üzerinden projeye import edilmiştir. Import işlemlerinin tamamlanmasından sonra proje aşamalarının gerçeklenmesi çalışmalarıyla devam edilmiştir.

III.YÖNTEM

Projede uygulanacak adımlara geçmeden önce test videosu üzerinden örnek bir frame alınmış ve adımlar bu frame üzerinden tamamlanmıştır. Son olarak ise adımlar videodaki her bir frame için uygulanabilir bir hale getirilmiştir bu sayede algoritma video üzerinde çalışmaya uygun bir yapıya kavuşturulmuştur.

A. Görüntünün okunması ve filtrelenmesi

Görüntü işlemenin en temel aşaması verilerin bilgisayarlar tarafından tanınmasıdır. Bunun için görüntü formatında bulunan veri sayısal bir yapı haline getirilmelidir. Verinin sayısal bir yapı haline getirilmesi için bir matris tanımlanır ve görüntüdeki piksellerin yoğunluk değerleri bu matrise işlenir. Yoğunluk değerlerinin matrise işlenmesinin ardından görüntü işlenmeye hazır hale gelmiş olur. Görüntü renksiz ise her bir piksel matris üzerinde bir yoğunluk değerine sahip olur. Eğer görüntü renkli ise üç kanala (RGB) sahiptir ve her bir piksel bu üç kanalın yoğunluklarının kombinasyonu şeklindedir. Görüntü işleme için kullanılan en popüler kütüphanelerden birisi "OpenCV" kütüphanesidir. Bu çalışmada da "OpenCV" kütüphanesi ve matrisler üzerinde işlemleri kolaylaştıran "Numpy "kütüphanesi kullanılmaktadır.

```
import numpy as np
import cv2
```

Şekil 1 - Import işlemleri

Öncelikle kütüphaneler import edilmiştir sonrasında ise görüntünün işlenebilmesi için resmin okunması işlemi gerçekleştirilmiştir.

```
image = cv2.imread('testImg.png')
```

Şekil 2 - Resmin okunması

Şekil 3 - Üç kanallı yapıdaki görsel

Resmin okunmasın ardından resim üzerinde gri tonlamalı dönüştürme işlemi yapılmıştır. Bu sayede üç kanallı yapıda olan pikseller tek kanallı yapıya dönüştürülmüştür çünkü tek kanallı yapıda görüntü işleme daha hızlıdır ve ilerleyen aşamalarda yoğunluk geçişlerinden faydalanarak kenarların tespit edilmesi planlanmıştır. Görüntüyü "grayscale" yapıda elde etmek için ise "cv2.COLOR_RGB2GRAY" flagi kullanılmıştır.

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

Şekil 4 – Grayscale görüntünün oluşturulması

Şekil 5 – Grayscale görüntü

Grayscale görüntü elde edildikten sonra görüntünün gürültüsünün azaltılması gerekmektedir. Burada bahsedilen gürültü kenar tespitini zorlaştıran yanlış kenarlardır. Bu işlemin gerçekleştirilmesi için "cv2.GaussianBlur" fonksiyonu kullanılmaktadır. Gaussian Blurlama, görüntüleri 'bulanıklaştırmak', ayrıntı ve gürültüyü ortadan kaldırmak için kullanılan iki boyutlu konvolüsyon (çekirdek matris ile resim üzerindeki piksellerin çarpımı işlemi) operatörüdür [3]. 5x5 çekirdek matris çoğu durum için iyi bir seçimdir.

```
blur = cv2.GaussianBlur(gray, (5,5), 0)
```

Şekil 6 – GaussianBlır fonksiyonun kullanılması

B. Canny Algoritmasının Uygulanması

Görüntü blurlama işlemi ardından projenin temel yapıtaşı olan kenar algılama işlemine geçilmektedir. Kenar algılama işleminin yapılması için ise "Canny Algoritması" kullanılmaktadır. Canny Kenar Belirleme Algoritması; görüntüde keskin olarak belirlenmiş kenarları bulmak için geliştirilmiş ve aşamaları olan bir algoritmadır. Görüntünün gürültülerini azaltmak amacıyla Gaussian çekirdekle konvolüsyon alınarak azaltılır [4]. Yaptığımız çalışmada da bir önceki aşamada Gaussian blurlama işlemi gerçekleştirilmiştir. Algoritma, piksellerin parlaklık değişimlerinden faydalanarak yüksek parlaklık geçişlerinin olduğu bölgeleri (gradient) beyaz piksellerle kaplamaktadır, böylelikle "Canny Gradient Image" oluşturulmaktadır ve görsel üzerindeki kenarların tespit işlemi tamamlanmaktadır.

```
canny_image = cv2.Canny(blur,50,120)
```

Şekil 7 – Canny fonksiyonu kullanımı

```
def canny_function(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    blur = cv2.GaussianBlur(gray, (5,5), 0)
    return cv2.Canny(blur,50,120)
```

Şekil 8 – Filtreleme işlemleri ve Canny fonksiyonunun kullanımı

"canny_function" metodunun kullanılmasının ardından renkli görüntü, aşağıda görüleceği gibi kenarların tespit edilmiş halde olduğu bir görsel haline getirilmektedir.

Şekil 9 – Filtreleme işlemleri ve Canny fonksiyonunun kullanımının sonucu

C. İlgilenilecek Bölgenin Tespiti

Görüntünün kenar algılamalarının tespiti gerçekleştikten sonra ilgilenilen bölgenin tespiti işlemine geçilmektedir. Bu işlemin amacı araç önündeki bölgenin izole edilerek şerit tespit işleminin daha iyi yapılmasını sağlamaktır. Bölge tespiti yapılırken koordinatların bulunmasına yardımcı olması için "matplotlib" kütüphanesi kullanılmaktadır.

İlgilenilen bölgenin tespiti için x ve y değerlerinin izdüşümleri gözlenmiş olup görsellerde görülen bölge tespit edilmiştir.

Şekil 10 – lgilenilecek Bölgenin Tespiti

Bölgenin tespitinin ardından tespit edilen bölgenin beyaz, kalan bölgenin siyah olduğu bir maske oluşturulmuştur.

Şekil 11 – Oluşturulan maske

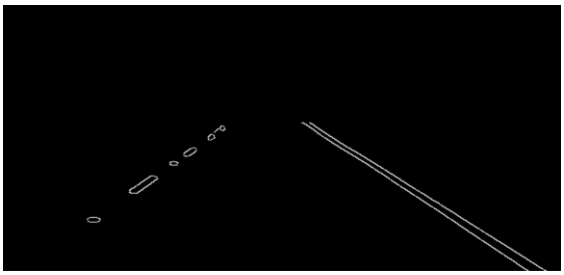
Oluşturulan bu maske "bitwise_and" işlemiyle kenar tespitlerinin yapıldığı görsel üzerine uygulanmıştır. Bu işlemler sonucunda bir sonraki adım olan "Hough Transform" aşamasına uygun görsel oluşturulmuştur.

```
def roi(image):
    x = int(image.shape[1]) #x = 1920
    y = int(image.shape[0]) #y = 1073

    shape = np.array([[250, int(y)],
     [1750, int(y)], [int(0.55*x), int(0.6*y)],
      [int(0.45*x), int(0.6*y)]])
    # kose koordinatları = 250,1073 1750,1073 1056,643 864,643
    mask = np.zeros_like(image)
    cv2.fillPoly(mask, np.int32([shape]), 255)
    masked_image = cv2.bitwise_and(image, mask) #normal resim
    #uzerine secili alan uygulanıyor.

    return masked_image #maskelenmis hali donuyor
```

Şekil 12 – İlgilenilecek bölgenin tespit edildiği ve maskeleme işleminin yapıldığı fonksiyon



Şekil 13 – "roi" fonksiyonunun kullanılması sonucu elde edilen alan

D. Hough Transform

Hough dönüşümü temelde kenarların olası geometrik şekilleri oylaması mantığı ile çalışmaktadır. Hough dönüşümü kullanılarak şekil tespiti genel olarak aşağıdaki adımlar ile özetlenebilir:

- Kaynak görüntü üzerinde kenarlar belirlenir.
- Bir eşikleme yöntemi kullanılarak görüntü ikili (siyah,beyaz) hale getirilir.
- Her kenar pikseli için noktanın üzerinde olabileceği olası geometrik şekillerin polar koordinattaki değerleri kullanılan bir matris üzerinde birer birer artırılarak her kenar pikselin olası şekilleri oylaması sağlanmış olur.
- Matris değeri en yüksek olan şekiller en çok oy alan şekiller olduklarından görüntü üzerinde bulunma veya belirgin olma olasılıkları en yüksek olmaktadır.

Hough transform ile birlikte görüntü üzerindeki çizgiler ortaya çıkarılmaktadır. Fakat her kenara göre bir kenar çizgisi ortaya çıktığı için düz bir şerit görüntüsü elde edilememektedir bu nedenle görüntü optimize edilmeye muhtaç bir yapıdadır.

```
lines = cv2.HoughLinesP(cropped_image, 1, np.pi
 10, np.array([]), minLineLength=20, maxLineGap
```

Şekil 14 – Hough transform işleminin uygulunması

Şekil 15 – Tespit edilen çizgiler

E.Optimizasyon

Bu aşamada koordinatlara göre ortalama bir kenar çizgisi elde edilmek istenmiştir. Her kenar çizgisinin eğimi tespit edilerek ait olduğu tarafa (sağ veya sol) göre sınıflandırılıp ortalama bir kenar çizgisi çıkarılmıştır.

```
for line in lines:
    for x1,y1,x2,y2 in line:

        slope = (y1-y2)/(x1-x2)
        if slope > 0.27:
            if x1 > 500:
                yintercept = y2 - (slope*x2)
                rightSlope.append(slope)
                rightIntercept.append(yintercept)


        elif slope < -0.27:
            if x1 < 600:
                yintercept = y2 - (slope*x2)
                leftSlope.append(slope)
                leftIntercept.append(yintercept)
```

Şekil 16 – Optimizasyon işleminin yapılması

```
leftavgSlope = np.mean(leftSlope[-30:])
leftavgIntercept = np.mean(leftIntercept[-30:])

rightavgSlope = np.mean(rightSlope[-30:])
rightavgIntercept = np.mean(rightIntercept[-30:])
```

Şekil 17- Ortalama alma işlemleri

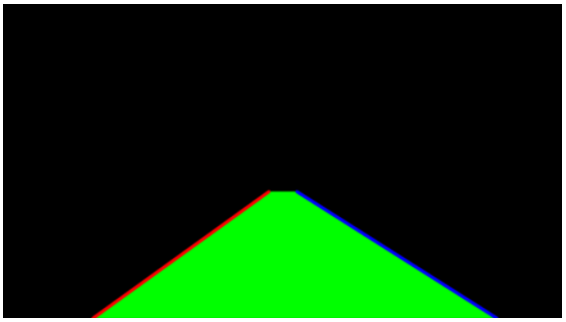
F.Şerit Tespiti ve Görsel Üzerine Uygulanması

Bu aşama projenin son aşamasıdır. Bu aşamada öncelikle ortalaması bulunan değerlere göre şeritlerin uç noktaları tespit edilmektedir. Tespit edilen noktalar üzerinden içi dolu bir şerit görüntüsü çizdirilmektedir. Çizdirilen bu görüntü ise renkli görsel ile birleştirilmektedir.

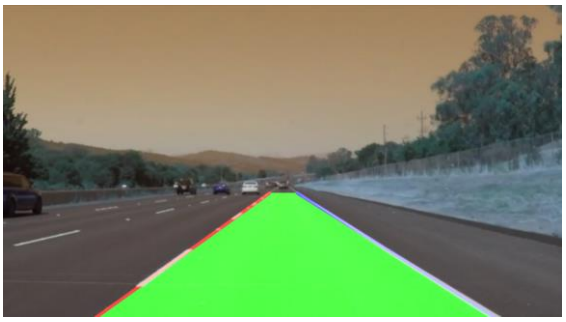
```
def find_coordinates(line_image, avg_intercept, avg_slope):
    x1 = int((0.60*line_image.shape[0] - avg_intercept)/avg_
    x2 = int((line_image.shape[0] - avg_intercept)/avg_slope

    return x1,x2
```

Şekil 18- Uç noktaların tespit edilmesini sağlayan fonksiyon



Şekil 19- Tespit Edilen Şerit



Şekil 20- Tespit Edilen Şeridin Renkli Görsel ile Birleştirilmesi

```
detected_lane = draw_lines(line_image,lines)
combo_image = cv2.addWeighted(lane_image , 0.8, detected_la
```

Şekil 21- Şeridin çizdirilmesi ve renkli görsel ile birleştirilmesi işlemi

IV.SONUÇ

Çeşitli algoritmalar ve kütüphanelerin kullanılması ile araç içi kameralardan elde edilen görüntüler üzerinden şerit algılama işlemi gerçekleştiren bir sistem geliştirilmiştir.

V.REFERANSLAR

[1]https://dergipark.org.tr/en/download/article-file/2134675

[2]https://medium.com/@adem.akdogan/opencv-k%C3%BCt%C3%BCphanesi-ile-g%C3%B6r%C3%BCnt%C3%BC-i%CC%87%C5%9Fleme-uygulamal%C4%B1-af50033f7d8

[3]http://www.ibrahimcayiroglu.com/Dokumanlar/GoruntuIsleme/Goruntu_Isleme_Ders_Notlari-5.Hafta.pdf

[4]https://medium.com/caglargul-blog/emgucv-canny-kenar-bulma-algoritmas%C4%B1-416890399107

[5]https://www.muzafferkadir.com/hough-transform-ile-goruntudeki-daireleri-bulmak/

[5]https://www.lexus.com.tr/discover-lexus/lexus-otomobilleri-hakkinda-hersey/otonom-araclar-nasil-calisir-otonom-arac-teknolojisi?lexReferrer=https%3A%2F%2Fwww.google.com%2F#section-1

[6]https://medium.com/@hamzaerguder/s%C3%BCr%C3%BCc%C3%BCs%C3%BCz-ara%C3%A7lar-i%C3%A7in-hough-transform-ile-%C5%9Ferit-bulma-%C3%A7al%C4%B1%C5%9Fmas%C4%B1-9c5ab8d1bcfb

[7]https://www.youtube.com/watch?app=desktop&v=eLTLtUVuuy4

[8]https://www.hackster.io/kemfic/simple-lane-detection-c3db2f