T.C

TRAKYA ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

DİJİTAL GÖRÜNTÜLER ÜZERİNDE UYGULANABİLEN ŞERİT TAKİP VE TESPİT SİSTEMLERİ

BİTİRME PROJESİ

EMRE DOĞAN

Danışmanı: Doç. Dr. Oğuzhan ERDEM

Ocak 2021

EDİRNE

ÖZET

Bu çalışmada öncelikle Bilgisayarlı Görü (Computer Vision) teriminin ne olduğu ve tekniklerinin nasıl kullanıldığından bahsedilmiştir. Daha sonra bunu gerçeklemek için kullanılan kütüphanelerden biri olan OpenCv kütüphanesi anlatılmıştır. Sonraki aşamalarda sırasıyla Temel bir Şerit Takip Sistem tasarımı yapılmış ve anlatılmıştır. Bunun üzerine aynı gerçek zamanlı olmayan görüntü üzerinde Gelişmiş bir Şerit Takip Sistemi tasarlanarak denenmiştir ve ondan da belirli sonuçlar elde edilerek son bölümde kullanılan algoritmalar karşılaştırılmıştır. İki Sisteminde avantajları ve dezavantajları ile neden daha gelişmiş bir sisteme ihtiyaç duyulduğu gösterilmiştir. En son bölümde ise sonuç ve öneriler belirtilmiştir.

Anahtar Kelimeler: Şerit Tespit Sistemi, Bilgisayarlı Görü, Şerit Takip Sistemi Algoritmaları

ABSTRACT

In this study, first, what the term Computer Vision is and how its techniques are used are mentioned. Then, the OpenCv library, which is one of the libraries used to implement this, is explained. In the next stages, a Basic Lane Departure System design was made and explained, respectively. Then, an Advanced Lane Tracking System was designed and tested on the same non-real-time image, and certain results were obtained from it, and the algorithms used in the last section were compared. It has been shown why a more advanced system is needed with its advantages and disadvantages in the Two System. In the last section, conclusions and recommendations are given.

Keywords: Lane Detection System, Computer Vision, Lane Departure System Algorithms

ÖNSÖZ

Bu çalışmada Şerit Takip ve Tespit Sistemlerinde kullanılan farklı Bilgisayarlı Görü Tekniklerinin ve kullanılan algoritmaların tanımlaması ve açıklaması yapılmıştır. Daha sonra eksik yanları ve avantajları bakımından karşılaştırılmıştır. Ayrıca OpenCv kütüphanesi kullanılarak kod haline getirilmiş ve gerçek zamanlı olmayan görüntüler üzerinde denenerek sonuçlar meydana getirilmiştir.

TEŞEKKÜR

Bu tez çalışmasına başlamamdan itibaren çalışmalarımda ve kaynak bulmamda bana her türlü destek veren ayrıca kendisinden çalışmalarım esnasında talep ettiğim her türlü görüşme için bana en yoğun olduğu günlerde dahi zaman ayıran değerli danışman hocam Doç. Dr. Oğuzhan ERDEM' e ve çalışmam esnasında çeşitli imkanlarından yararlanmamı sağlayan Trakya Üniversitesi' ne teşekkürü borç bilirim.

Aralık 2021

Emre DOĞAN

İÇİNDEKİLER

ÖZET I
ABSTRACT II
ÖNSÖZ III
TEŞEKKÜR IV
SİMGELER DİZİNİ VII
ŞEKİLLER DİZİNİ VIII
1. GİRİŞ 1
1.1 BİLGİSAYARLI GÖRÜ (CV) 1
1.2 OPENCV KÜTÜPHANESİ 2
1.3 KULLANILACAK ALGORİTMALARIN TANIMI 2
1.3.1 Temel Şerit Takip ve Tespit Sisteminde Kullanılan Algoritmalar 2
1.3.1.1Canny Köşe Bulma Algoritması 2
1.3.1.2 Hough Dönüşüm Algoritması 3
1.3.2 Gelişmiş Şerit Takip ve Tespit Sisteminde Kullanılan Algoritmalar 5
1.3.2.1 Görüntüdeki Açısal Bozulmanın Kaldırılması (Undistortion) 5
1.3.2.2 Sobel Filtre Algoritması 5
1.3.2.3 Kayan Pencere Algoritması 6
2. TEMEL ŞERİT TAKİP UYGULAMASI 7
2.1 ANA YOL GÖRÜNTÜNÜN EKLENMESİ 7
2.2 GÖRÜNTÜNÜN GRİ RENGE ÇEVİRİLMESİ 8
2.3 GÖRÜNTÜYE GAUSSİAN BULANIKLAŞTIRMA UYGULANMASI 8
2.4 CANNY KÖŞE BULMA ALGORİTMASININ UYGULANMASI 9
2.5 İLGİLİ ALANIN GÖRÜNTÜ ÜZERİNDEN ÇIKARILMASI 10
2.6 BU ALAN ÜZERİNDE HOUGH ÇİZGİ DÖNÜŞÜMÜ UYGULANMASI 11
2.7 ELDE EDİLEN NOKTALARIN BİRLEŞTİRİLMESİ 11
2.8 BİRLEŞTİRİLEN NOKTALARIN ŞERİT OLARAK ÇİZİLMESİ 12
2.9 ANA GÖRÜNTÜ İLE ŞERİT GÖRÜNTÜSÜNÜN BİRLEŞTİRİLMESİ 13
3. GELİŞMİŞ ŞERİT TAKİP UYGULAMASI 14

3.1 ANA ŞERİT GÖRÜNTÜSÜNÜ EKLEME 14
3.2 GÖRÜNTÜ ÜZERİNDEKİ BOZULMAYI GİDERME 14
3.3 GÖRÜNTÜYÜ HSL' YE ÇEVİRME VE SOBEL FİLTRE UYGULAMASI 15
3.4 GÖRÜNTÜ ÜZERİNDE PERSPEKTİF ÇARPITMA UYGULANMASI 16
3.5 HİSTORİK NOKTA TESPİTİ YAPILMASI 17
3.6 KAYAN PENCERE ALGORİTMASININ UYGULANMASI(SWA) 17
3.7 EĞRİ BULMA ALGORİTMASININ UYGULANMASI 18
3.8 ŞERİTLERİN ÇİZDİRİLMESİ VE ANA GÖRÜNTÜ İLE EKRANA GETİRİLMESİ 20
4. YAPILAN İKİ UYGULAMANIN KARŞILAŞTIRILMASI 21
4.1 GÖRÜNTÜDEKİ BOZUNMANIN GİDERİLMESİNİN SONUCU 21
4.2 CANNY VE SOBEL KÖŞE BULMA ALGORİTMALARININ 22
4.3 HOUGH TRANSFORM VE KAYAN PENCERE ALGORİTMALARININ KARŞILAŞTIRILMASI 23
4.4 YOLDAKİ KAVİS DURUMU 24
5. SONUÇ VE ÖNERİLER 25
5.1 SONUÇ 25
5.2 ÖNERİLER 25
5.3 GELECEK ÇALIŞMALAR 25
6. KAYNAKÇA 26
7. ÖZGEÇMİŞ 27

SİMGELER DİZİNİ

RGB: Red- Green- Blue (Kırmızı-Yeşil- Mavi)

CV: Computer Vision (Bilgisayarlı Görü)

HSL: Hue- Saturation- Lightness (Ton- Doygunluk- Parlaklık)

SWA: Sliding Window Algorithm (Kayan Pencere Algoritması)

CNN: Convolutional Neural Network (Konvolüsyonel Sinir Ağı)

ŞEKİLLER DİZİNİ

Şekil 1.1 Vektörlerin Gösterimi 3
Şekil 1.4 Görüntüdeki Bozulmaya Bir Örnek [7] 5
Şekil 1.5 Sobel Filtre Kernel Çarpımı 5
Şekil 1.6 Orijinal Resim Üzerinde Sobel Filtre Uygulanmış 6
Şekil 2.1 İçeriye Aktarılan Görüntü 7
Şekil 2.2 Orijinal Resim Üzerinde Gri Renk Algoritması Uygulanmış 8
Şekil 2.3 Gri Renge Çevrilmiş ve Üzerinde Gaussian Blur Uygulanmış Resim 9
Şekil 2.4 Canny Algoritması Uygulanmış Görüntü 10
Şekil 2.8 Şeritlerin Çizdirilmesi 12
Şekil 2.9 Temel Şerit Takip Sisteminin Son Hali 13
Şekil 3.1 Gelişmiş Şerit Takip Sistemi İçin Orijinal Resim 14
Şekil 3.2 Görüntü Düzeltme İşleminin Uygulanmış Hali 15
Şekil 3.3 Sobel Görüntü Çıkartılmış 16
Şekil 3.6 Kayan Pencere Algoritması Uygulanmış 18
Şekil 3.7.1 Çeşitli Şerit Görüntüleri 19
Şekil 3.7.2 Offset Metrikleri Çıkartılmış 19
Şekil 3.8 Gelişmiş Şerit Takip Sisteminin Son Hali 20
Şekil 4.1 Bozulma Oranları Karşılaştırılması 21
Şekil 4.2 Canny ve Sobel Filtre Uygulamalarının Karşılaştırılması 22
Şekil 4.4 Kavis Durumlarının Karşılaştırılması 24

1. GİRİŞ

Bilindiği üzere günümüzde kara araç trafiğindeki kaza oranlarını azaltma ve daha güvenli bir sürüş sağlanması amacıyla araçlar sürücüye yardımcı olan bazı sistemler barındırmaktadır. Bu sistemlerden birisi de Şerit Takip ve Tespit Sistemidir. Bu sistem gerçek zamanlı olarak şeritleri tanır ve kullanıcının şeridi ihlal etmesi halinde onu uyararak şeridin merkezinde yeniden konum almasını ister. Bunu ise aracın ön bölgesine veya üst konumuna yerleştirilmiş bir veya birkaç kameradan aldığı görüntüler üzerinde farklı Bilgisayarlı Görü Teknikleri ve bazı Makine Öğrenmesi Teknikleri kullanarak gerçekleştirir. Bu Bilgisayarlı Görü Teknikleri farklı algoritmalara dayanarak çeşitli şekillerde uygulanabilir. Bunun yanı sıra her bir algoritmanın kullanılmasının getirdiği avantaj ve dezavantajları vardır. Bunlar verimlilik açısından karşılaştırılabilir.

1.1 BİLGİSAYARLI GÖRÜ (CV)

Bilgisayarlı Görü, insan görsel sisteminin yapabildiği işlemleri bir bilgisayara yaptırabilme işlemidir. Bilgisayara verilen dijital bir görüntü girdisi üzerinden makine kendisi üzerinde programlanan yapı yardımı ile bu dijital görüntü üzerinde insan görsel sisteminin yapabildiği işlemleri otomatik olarak yapabilmektedir. Bu işlemlerden bazıları nesneleri tanımak, onlara anlam yüklemek ve onları gruplandırmak olabilir. Bu çalışmada yapılan Şerit Takip ve Tespit Sistemi de yine bu terim üzerinden ilerlemektedir. Bilgisayar bir kamera yardımı ile şerit görüntüsünü alır. Sonra içerisinde oluşturulan bir program sistemi ile bu görüntü üzerinde çeşitli çevirimler ve çeşitli algoritmalar uygulayarak şerit görüntüsünü tanır. Daha sonrasında yine içerisindeki kod sistemi yardımı ile bu görüntünün istenildiği şekilde kullanılmasını sağlar. [1][2]

1.2 OPENCV KÜTÜPHANESİ

OpenCv bir açık kaynak kodlu Bilgisayar Görü kütüphanesidir. İlk olarak İntel firması tarafından geliştirilmeye başlanmıştır. İçerisinde 2500' den fazla Görüntü İşleme ve Makine Öğrenmesi algoritması barındırmaktadır. Bu algoritmalar çeşitli yazılım dilleri ile kütüphanenin ilgili alana aktarılmasından sonra kullanılabilirler. Bu algoritmalar nesne sınıflandırma, yüz tanıma gibi işlemler için kullanılabilirler. Bu çalışmada bu algoritmalar Python yazılım dilinde ve Şeridin Takibi ve Tespiti amaçlı kullanılmıştır.[3][4]

1.3 KULLANILACAK ALGORİTMALARIN TANIMI

Bu tez çalışmasında Şeridin Tanınması ve Tespiti amaçlı bazı algoritmalar kullanılmıştır. Bu kısımda bu algoritmalar tanıtılacaktır. Sistem tasarımı esnasında detaylı şekilde uygulama biçimleri ifade edilecektir.

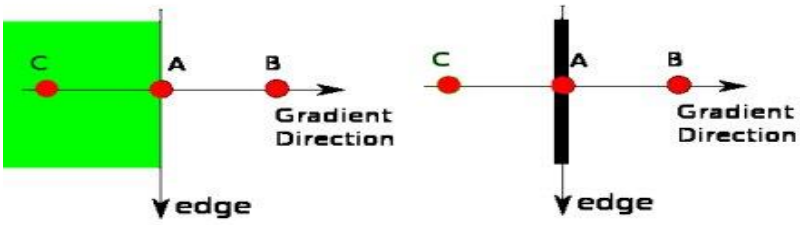
1.3.1 Temel Şerit Takip ve Tespit Sisteminde Kullanılan Algoritmalar

1.3.1.1Canny Köşe Bulma Algoritması

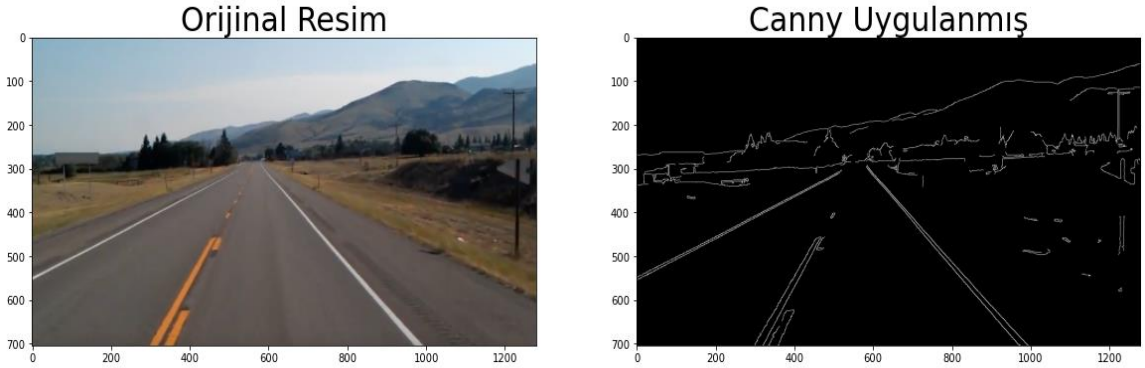
Canny Köşe Bulma Algoritması John F. Canny tarafından geliştirilmiş bir köşe bulma algoritmasıdır. Bu algoritma sırasıyla kendi içerisinde tanımlanan sıra ile öncelikle verilen resmi Gauussian Bulanıklaştırma metodu ile 5x5 oranında bulanıklaştırır. Daha sonra görüntü yatay yönde birinci türevi elde etmek amaçlı hem yatay hem de dikey yönde filtrelenir. Sonrasında her bir pikselin gradyan yönünde yerel bir maksimum olup olmadığı kontrol edilir. En son aşamada histerik bir eşik alınarak fonksiyon içerisinde yerel olarak verilen minimum ve maksimum değerlere göre köşe olup olmadıkları kabul edilir. (Bkz. Şekil 1.1 ve Şekil 1.2) [5]

Köşe Gradyanı (G) = $\sqrt{G_x^2 + G_y^2}$

Açı $(\theta) = tan^{-1}\left(\frac{G_x}{G_{\dot{y}}}\right)$



Şekil 1.1 Vektörlerin Gösterimi



Şekil 1.2 Orijinal Resim Üzerinde Canny Algoritması Uygulanmış

1.3.1.2 Hough Dönüşüm Algoritması

Hough Dönüşümü, Paul VC Hough tarafından patentlenmiş bir algoritmadır ve orijinal olarak fotoğraflardaki karmaşık çizgileri tanımak için icat edilmiştir (Hough, 1962). Algoritma, başlangıcından bu yana, belirli türlerdeki daireler ve dörtgenler gibi diğer şekilleri tanıyabilmek için değiştirildi ve geliştirildi. Hough Dönüşümü algoritmasının nasıl çalıştığını anlamak için dört kavramı anlamak önemlidir: kenar görüntüsü, Hough Alanı ve kenar

noktalarının Hough Uzayında eşlenmesi, bir çizgiyi temsil etmenin alternatif bir yolu ve çizgilerin nasıl algılandığı.

Algoritmanın çalışma prensibi;

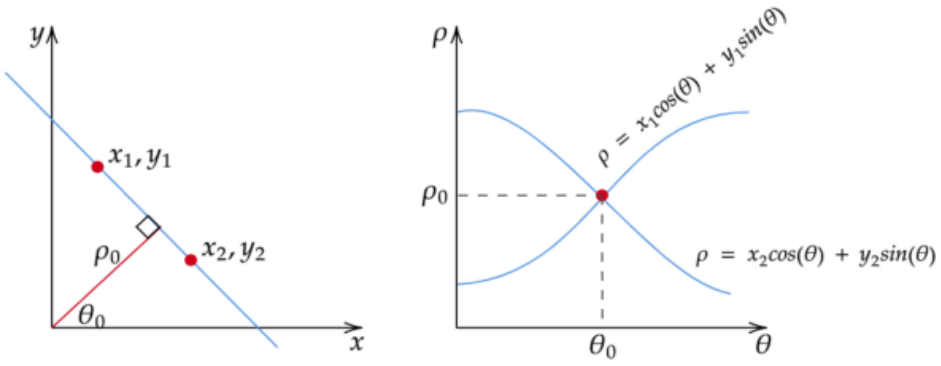
1 P ve θ aralığına karar verin. Çoğunlukla, θ aralığı [0, 180] derece ve ρ [- d, d] 'dir, burada d , kenar görüntüsünün köşegeninin uzunluğudur. P ve θ aralığını nicelemek önemlidir, bunun anlamı sonlu sayıda olası değerin olması gerektiğidir.

2 Hough Space'i boyutla (num_rhos, num_thetas) temsil eden biriktirici adı verilen bir 2D dizi oluşturun ve tüm değerlerini sıfır olarak başlatın.

3 Orijinal görüntü üzerinde kenar algılama gerçekleştirin. Bu, seçtiğiniz herhangi bir kenar algılama algoritmasıyla yapılabilir.

4 Kenar görüntüsündeki her piksel için, pikselin bir kenar pikseli olup olmadığını kontrol edin. Bu bir kenar piksel ise, tüm olası θ değerlerinde döngü yapın, karşılık gelen ρ'yu hesaplayın, toplayıcıda θ ve ρ indeksini bulun ve bu indeks çiftleri üzerindeki toplayıcı tabanını artırın.

5 Akümülatördeki tüm değerler arasında döngü yapın. Değer belirli bir eşikten büyükse, ρ ve θ indeksini alın, indeks çiftinden ρ ve θ değerini alın, daha sonra y = ax + b biçimine geri dönüştürülebilir. [6]

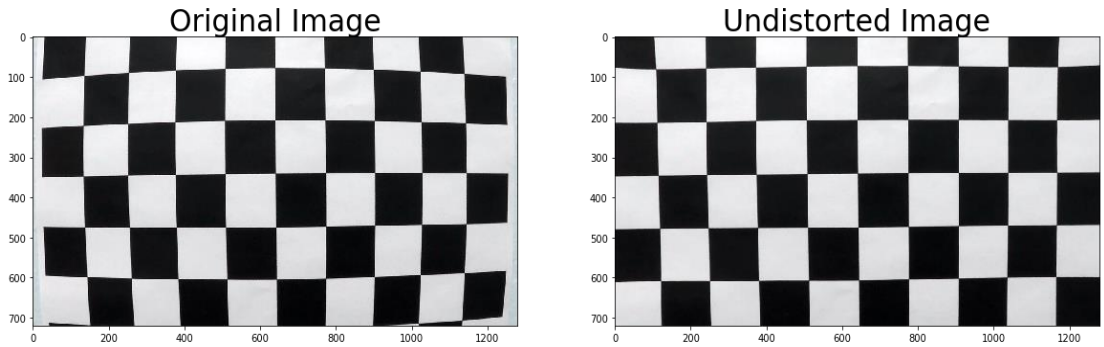


Şekil 1.3 Düz bir çizgi ve Ona karşılık gelen Hough dönüşümü

1.3.2 Gelişmiş Şerit Takip ve Tespit Sisteminde Kullanılan Algoritmalar

1.3.2.1 Görüntüdeki Açısal Bozulmanın Kaldırılması (Undistortion)

Bu algoritmanın uygulanmasının amacı halihazırda kameradan elde edilen görüntünün insan gözünün aldığı görüntünün haricinde görüntüyü belirli bir bozulma veya çarpıtma ile almasıdır. Bu dijital görüntü ise dijital ortama aktarıldığında üzerindeki bozulma sebebi ile tespit edilecek veride hata payı olması riski taşır. Bu yüzden en doğru görüntüyü elde etme amaçlı bu algoritma görüntüye uygulanır. (**Bkz. Şekil 1.4**)



Şekil 1.4 Görüntüdeki Bozulmaya Bir Örnek [7]

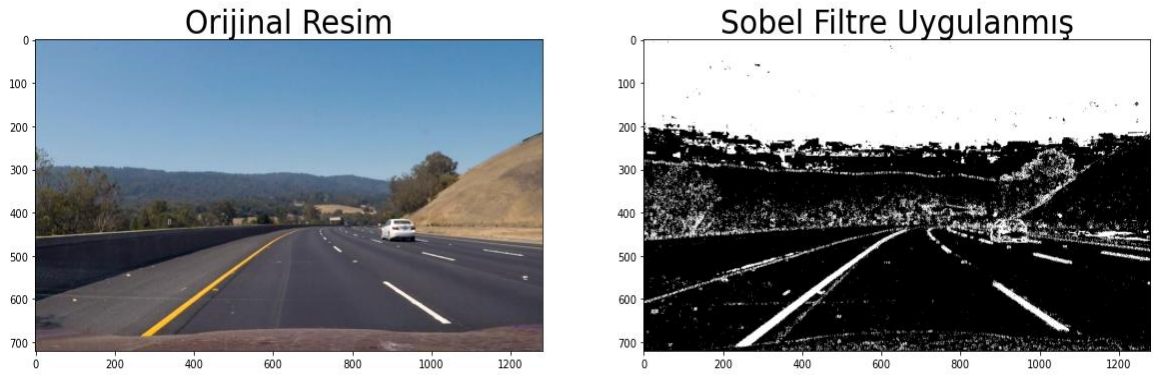
1.3.2.2 Sobel Filtre Algoritması

Sobel operatör ve Sobel-Feldman operatörü olarakta adlandırılabilir. Bu algoritma Canny köşe bulma algoritmasına benzer şekilde çalışır fakat buna karşın uygulama kısmında yatay ve dikey çizgiler ayırt edilebildiği gibi daha iyi bir çıktı sağlar. Algoritma dijital görüntü üzerinde 3x3 lük iki ayrı kernel uygular ve sonuçları birleştirerek ekrana getirir. Bunlardan bir tanesi X kernel diğeri ise Y kernel dir. X yatay çizgilerde köşe ararken Y dikey çizgilerde köşe arar. (Bkz. Şekil 1.5, Şekil 1.6) [8] [9]

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Şekil 1.5 Sobel Filtre Kernel Çarpımı

Bunlardan X kernel yatay çizgilerde köşe ararken Y kernel dikey çizgilerde köşe arar. Bunu ise aradaki piksellerin eşik değer değişimlerinden çıkartır. Eğer yatayda çizilen çizgilerde piksel eşik değişimi oluyorsa bunları yatay köşe, dikey çizilen çizgilerde piksel eşik değişimi oluyorsa bunları dikey köşe olarak alır.



Şekil 1.6 Orijinal Resim Üzerinde Sobel Filtre Uygulanmış

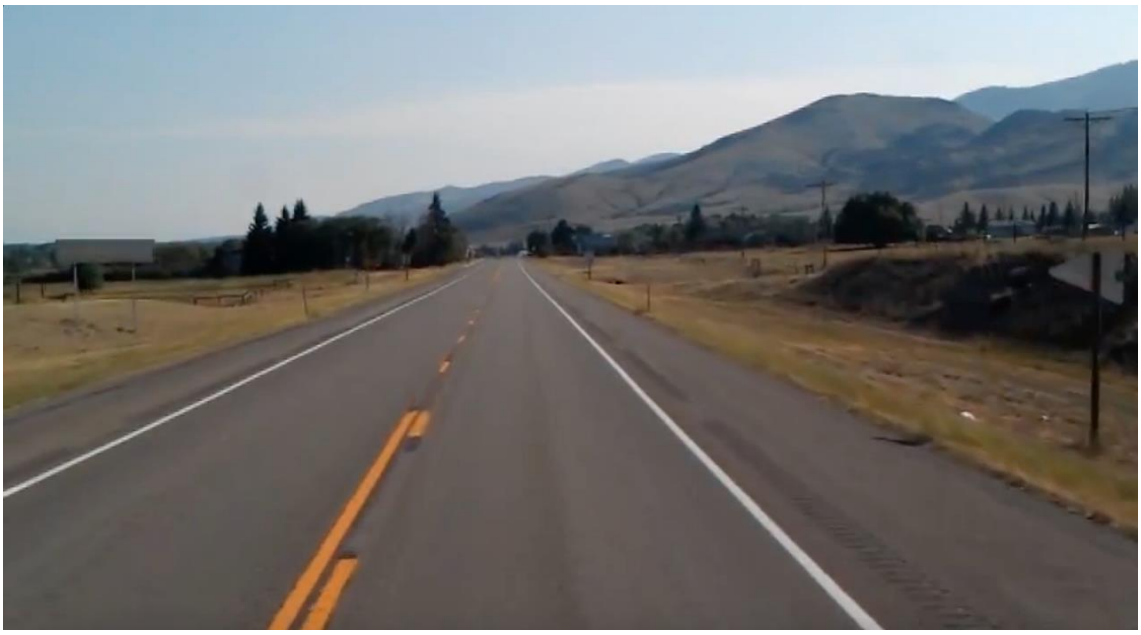
1.3.2.3 Kayan Pencere Algoritması

Bu çalışmada kullanılan algoritmalardan birisi de Kayan Pencere Algoritması (Sliding Window Algorithm) dır. Bu algoritma dijital görüntü üzerinde bulunması istenen alan üzerinde belirli bir boyutta pencereler oluşturarak bu pencerelerin içinde barınan piksel değerlerini alır. Her bir pencere pikselleri aldıktan sonra görüntü üzerinde bir adım kayar ve yeni bulunan pikselleri de alır. Tabi ki bu çalışmada bu işlem yapılırken belirlenen piksel değerleri belirli dizilerin içlerinde toplanarak daha sonra şerit tanımada kullanılmak amaçlı birleştirilmiştir. Bu algoritma nesne algılamanın yanı sıra nesne tespiti amaçlı Makine Öğrenmesi uygulamalarında da kullanılmaktadır.

2. TEMEL ŞERİT TAKİP UYGULAMASI

2.1 ANA YOL GÖRÜNTÜNÜN EKLENMESİ

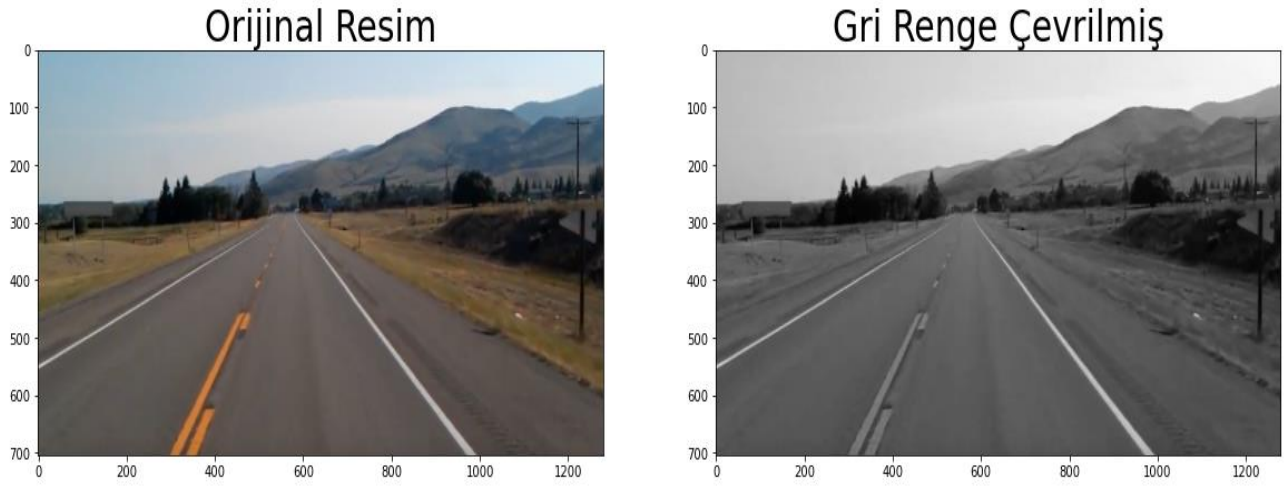
Bu çalışmada kullanılacak olan dijital görüntüyü öncelikle Python derleyicisi içerisinde içeri aktarmak gerekir. Bu sebeple OpenCv kütüphanesini ve diğer gerekli kütüphaneleri içeri aktardıktan sonra bir OpenCv kütüphane kodu olan *cv2.imread("dst.jpg")* şeklinde dijital görüntü içeri aktarılır bu aşamadan sonra bu görüntü üzerindeki şeritleri bulma çalışmamız benzer OpenCv kütüphane kodları ile devam edecektir. (Bkz. Şekil 2.1)



Şekil 2.1 İçeriye Aktarılan Görüntü

2.2 GÖRÜNTÜNÜN GRİ RENGE ÇEVİRİLMESİ

Bu aşamada içeriye aktarılan dijital görüntü gri renge çevrilir. Bunun yapılma amacı dijital görüntünün içerisinde şerit kesiti ile resmin içerisinde bulunan gürültüleri azaltmak ve RGB renk uzayında gelen görüntüdeki renk karmaşasını ortadan kaldırmaktır. Bu aşamayı uygulamak için bir OpenCv kodu olan görüntüyü gri renge çevirme amaçlı kullanılan *cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)* kodu kullanılır.(Bkz. Şekil2.2)

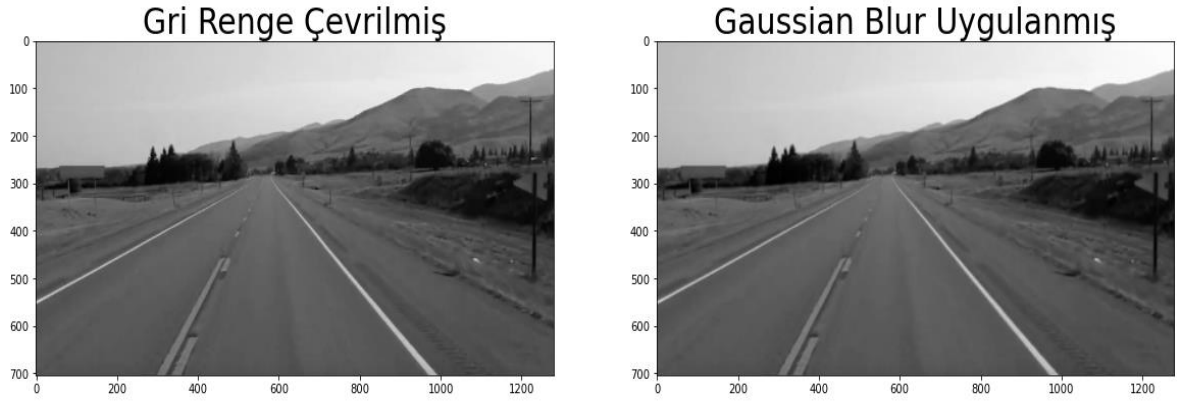


Şekil 2.2 Orijinal Resim Üzerinde Gri Renk Algoritması Uygulanmış

2.3 GÖRÜNTÜYE GAUSSİAN BULANIKLAŞTIRMA UYGULANMASI

Gri renge dönüştürülen görüntü üzerindeki gürültüleri yine azaltma amaçlı dijital görüntüye Gaussian Bulanıklaştırma uygulanır. Bu teknik dijital görüntünün üzerine uygulandığı zaman görüntü üzerindeki tüm renk değerlerini girilen kernel değerler özelinde eşitlemeye çalışarak görüntünün renk kanalında daha iyi bir okunma sağlar. Bu yüzden görüntü üzerinde şeritlerin tespitini de kolaylaştıran unsurlardan birisidir. Bu tekniğin uygulanması için OpenCv

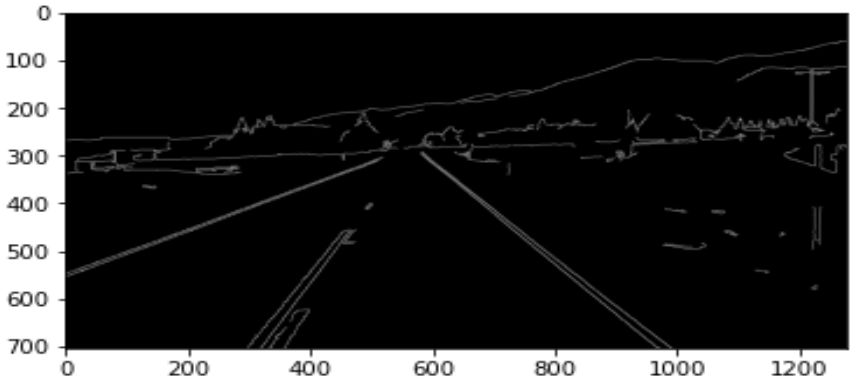
kütüphanesinde barınan kodlardan birisi kullanılır. Bu kod *cv2.GaussianBlur(image,(5,5),0)* şeklindedir. Burada kernel değerleri *5* seçilmiştir ve yerel olarak verilen bir değerdir. *0* değeri ise sigmaX ve sigmaY değerleri içindir (Bkz. Şekil 2.3)



Şekil 2.3 Gri Renge Çevrilmiş ve Üzerinde Gaussian Blur Uygulanmış Resim

2.4 CANNY KÖŞE BULMA ALGORİTMASININ UYGULANMASI

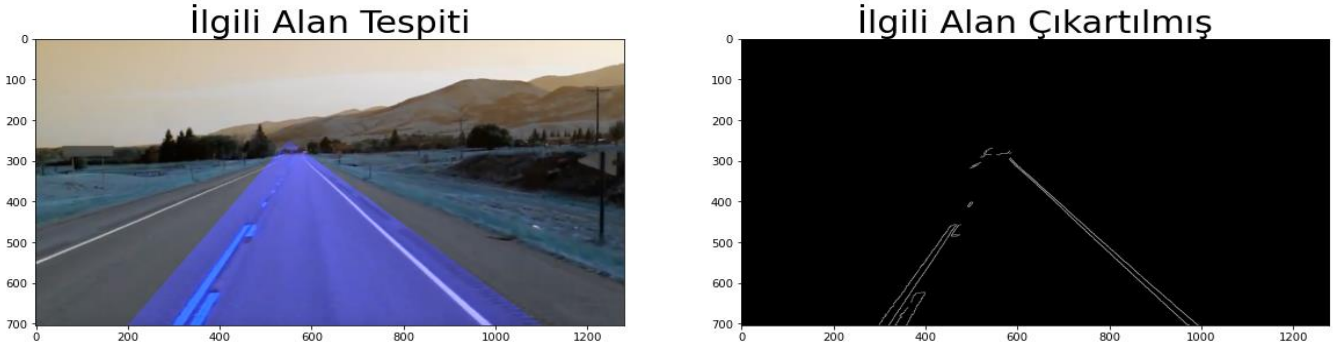
Uygulamanın en önemli aşamalarından biri olan bu kısımda gürültülerden arındırılmış dijital görüntü üzerinde köşelerin tespit edilmesi amacı ile Canny Köşe Bulma Algoritması uygulanmıştır. Bu algoritma ile Şeridin Tespiti amaçlı görüntüdeki köşeler çıkartılmıştır. Bu alanı çıkartmak için bir OpenCv kodu olan *cv2.Canny(image, 50, 150)* burada *image* adlı değişken kaynak görüntüyü ifade ederken *50* minumum değişim değerini *150* ise maksimum değişim değerini ifade eder. (Bkz. Şekil 2.4)



Şekil 2.4 Canny Algoritması Uygulanmış Görüntü

2.5 İLGİLİ ALANIN GÖRÜNTÜ ÜZERİNDEN ÇIKARILMASI

Dijital görüntü üzerinde köşe tespiti yapıldıktan sonra sırada ilgilenilen alanı diğer bir deyişle şeritlerin olduğu kısmı alan üzerinden çıkartmak gereklidir. Çalışmaya bu görüntü üzerinden alınan kesit üzerinden devam edilecektir. Bu kesit seçilirken ise rastgele değil belirli ölçümler üzerine çıkartılmıştır. Alan dijital görüntü üzerinde belli bir üçgen alan belirlenerek çıkartılmıştır. Bu alanın çıkarılması için girilen yükseklik ve genişlik değerleri dijital cetveller yardımı ile tespit edilmiştir ve kullanılmıştır. (Bkz. Şekil 2.5)



Şekil 2.5 İlgili Alan Çıkartılmış Resim

2.6 BU ALAN ÜZERİNDE HOUGH ÇİZGİ DÖNÜŞÜMÜ UYGULANMASI

İlgili alan çıkartıldıktan sonra tespit edilen alan üzerinde noktaların tespiti için Hough Çizgi Dönüşümü yapılır. Bu algoritma görüntü üzerinde nokta kabul ettiği çizgi ve belirtileri alır ve bazı noktaları tespit ederek bunları çıktı verir. Bunun uygulanması içinde bir OpenCv kodu olan *cv2.HoughLinesP(image, 2, np.pi/180, 100, np.array([]), minLineLength=40,maxLineGap=5)* bu kod içerisinde *image* değişkeni kaynak görüntüyü *2* rho değerini *np.pi/180* theta değerini *100* threshold değerini *np.array([])* lines değerini *minLineLength* istenilen minimum çizgi uzunluğunu *maxLineGap* ise çizgiler arası istenilen boşluk değerini belirtir. Seçilen değerler için kesin sayılar yoktur tasarıma ve görüntüye göre değişiklik gösterebilir. Bu değerler aşağıda gösterilmiştir.

[[705 419 927 641]]
[[747 472 885 615]]
[[320 702 452 482]]
[[586 302 708 428]]
[[358 610 456 459]]
[[591 303 705 417]]
[[355 615 385 570]]
[[900 613 976 689]]
[[887 618 970 703]]
[[736 459 778 503]]
[[300 700 325 660]]]

2.7 ELDE EDİLEN NOKTALARIN BİRLEŞTİRİLMESİ

Çalışmada sıradaki adım ise bu elde edilen noktaların birleştirilmesi ve sağ ve sol şerit olarak ayırılması amaçlı noktaların bir araya getirilmesidir. Bu sebeple yerel bir algoritma yazılmıştır. Sağ ve sol şerit adına iki boş dizi oluşturulmuş daha sonrasında bir döngü içerisinde denklem birinci dereceden bir denkleme dönüştürülerek temel y=mx+n üzerinden kod ilerletilmiş ve çalışma içerisinde y ekseninin şerit ilerlerken ters ilerlediği düşünülerek

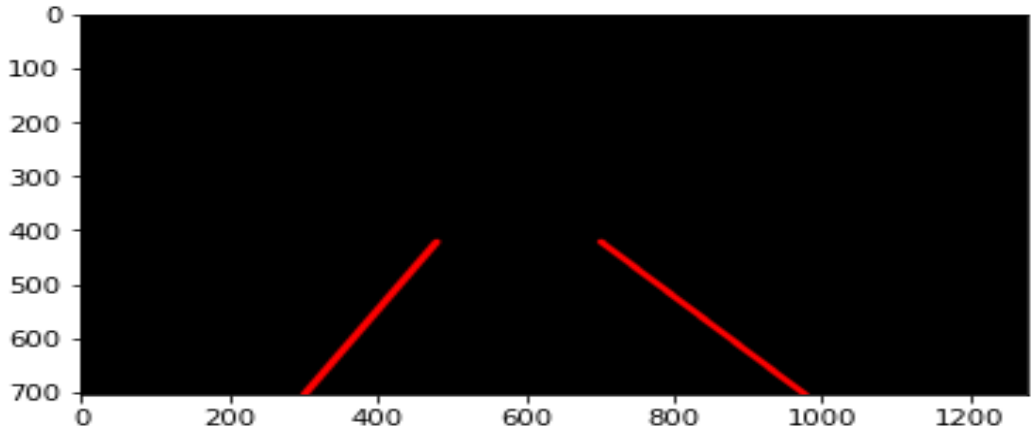
azalan yönde olanlar x artan yönde olanlar y kabul edilmiş ve 4 nokta 2 koordinat üzerinden değerler sağ ve sol şerit olmak üzere kendi dizilerine eklenmiştir. Bu işlemler gerçekleşirken çeşitli OpenCv ve Python kodları kullanılmıştır. Sonuç olarak aşağıda gösterilen ortalama değerler çıkartılmıştır.

[[[302, 704, 481, 422]], [[979, 704, 703, 422]]]

Bu değerler sırasıyla sol şerit değerlerini ve sağ şerit değerlerini ifade etmektedir.

2.8 BİRLEŞTİRİLEN NOKTALARIN ŞERİT OLARAK ÇİZİLMESİ

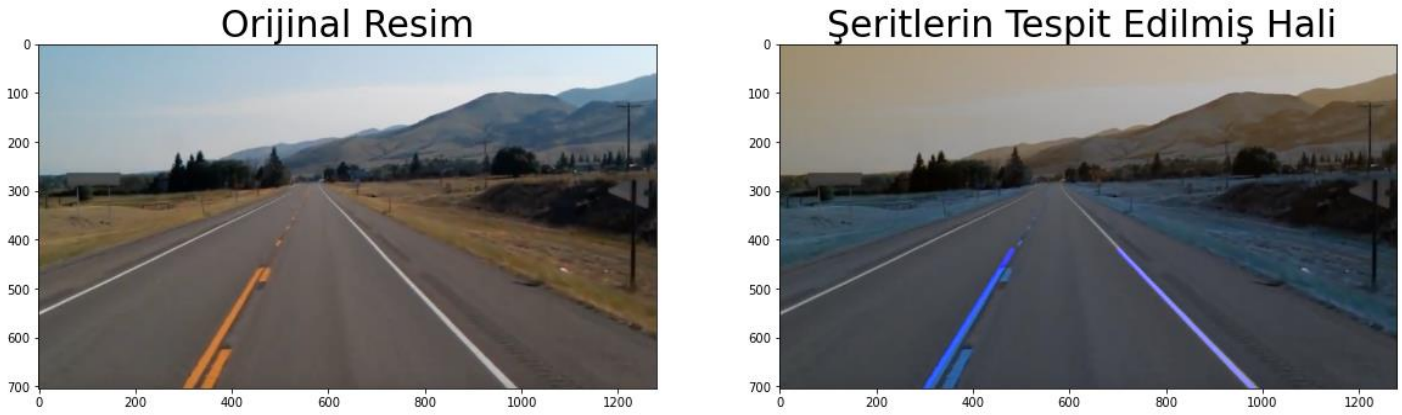
Sıradaki aşama ise en son elde edilen iki şeridin çizilmesidir. Bunun için elde edilen iki nokta özelinde girdiler verilerek şerit ekrana çizdirilebilir. Bir OpenCv kodu ile bunu gerçekleyebiliriz. *cv2.line(image,(x1,y1),(x2,y2),(255,0,0),10)* burada verilen değerler bir önceki aşamada elde edilen değerler üzerinedir. Burada image değişkeni kaynak görüntüyü, *(x1,y1),(x2,y2)* değerleri elde edilen değerleri, *(255,0,0)* renk kodunu ve *10* değeri ise çizginin kalınlık değerini belirler. (Bkz. 2.8)



Şekil 2.8 Şeritlerin Çizdirilmesi

2.9 ANA GÖRÜNTÜ İLE ŞERİT GÖRÜNTÜSÜNÜN BİRLEŞTİRİLMESİ

Son aşamada elde edilen iki görüntü birleştirilir bunun için OpenCv kütüphanesinden bir kod kullanılır. *cv2.addWeighted(image1,0.8,image2 ,1,1)* bu kod ile iki görüntü bir araya getirilerek tek bir görüntü oluşturulur. Bu kod içerisinde kullanılan *image1* değişkeni birinci görüntüyü *image2* değişkeni ise ikinci görüntüyü temsil etmektedir. Sırasıyla *0.8* ve *1* değerleri ise görünürlük değerleridir. Diğer *1* değeri ise gama değerini ifade eder.(Bkz. Şekil 2.9)



Şekil 2.9 Temel Şerit Takip Sisteminin Son Hali

3. GELİŞMİŞ ŞERİT TAKİP UYGULAMASI

3.1 ANA ŞERİT GÖRÜNTÜSÜNÜ EKLEME

Burada yine çalışmaya öncelikli olarak OpenCv ve diğer gerekli kütüphaneler eklendikten sonra kaynak dijital görüntü içeriye aktarılarak başlanır. Bu dijital görüntünün aktarılması işlemi için yine cv2.imread("dst.jpg") komutu kullanılır. (Bkz. Şekil 3.1)

Şekil 3.1 Gelişmiş Şerit Takip Sistemi İçin Orijinal Resim

3.2 GÖRÜNTÜ ÜZERİNDEKİ BOZULMAYI GİDERME

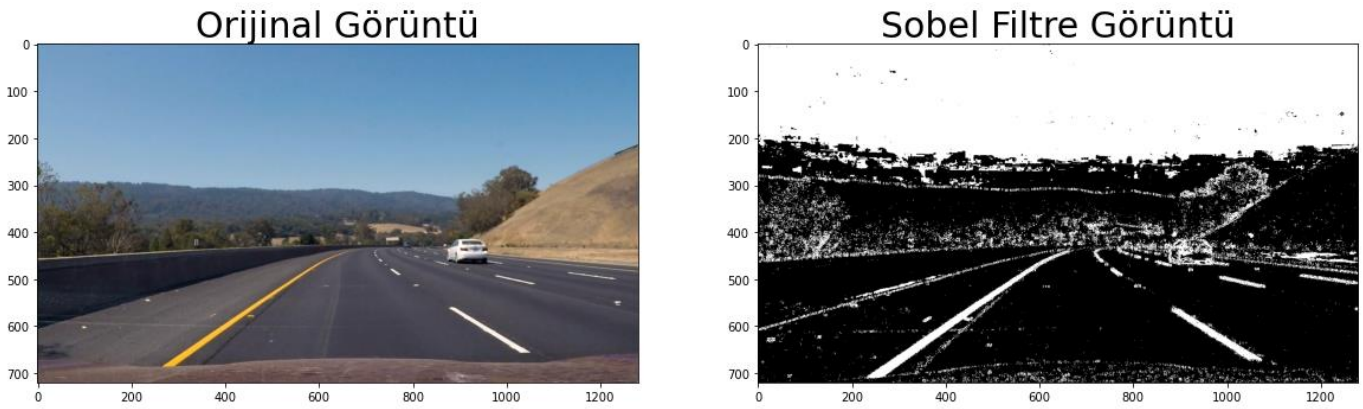
Kameraların aldığı ve çalışmada kullanılan görüntülerde çarpıklık ve bozulma olduğundan dolayı bunların düzeltilmesi gerekir bu yüzden görüntü örnek bir algoritma ile düzeltilebilir. Bunun için çalışmanın bir kısmında belirtilen Undistortion Algoritması uygulanır. Bu algoritmanın çalışma prensibi Kullanılan Algoritmalar kısmında anlatılmıştır. Burada bu

algoritma yine Şeritlerin Tespiti amaçlı görüntünün düzeltilmesi için Dijital görüntüye uygulanır. (Bkz. Şekil 3.2)

Şekil 3.2 Görüntü Düzeltme İşleminin Uygulanmış Hali

3.3 GÖRÜNTÜYÜ HSL' YE ÇEVİRME VE SOBEL FİLTRE UYGULAMASI

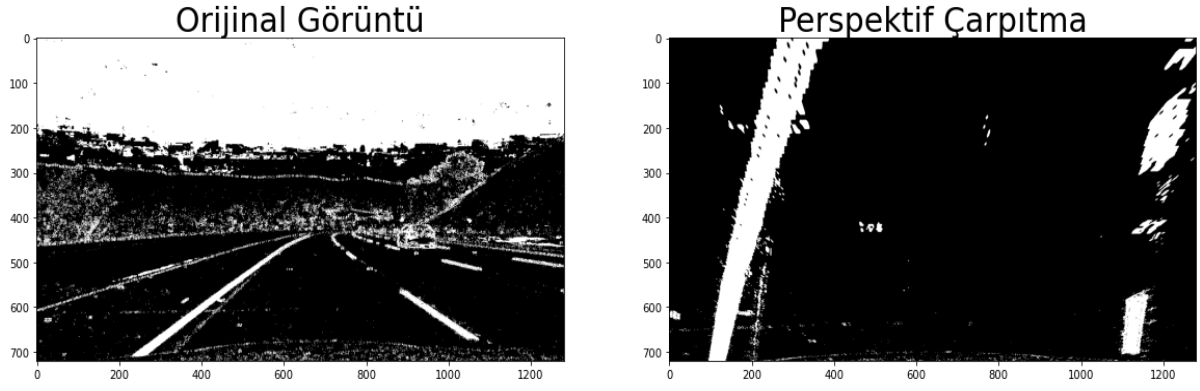
Çalışma esnasında burada görüntüdeki gürültüleri giderme amaçlı görüntü RGB renk uzayından HSL renk uzayına çevrilmiştir. Buradaki amaç görüntü üzerine uygulanacak diğer işlemlerden önce görüntü üzerindeki gürültüleri temizlemek ve uygulanacak filtrelerin daha iyi bir sonuç vermesini sağlamaktır. HSL sırasıyla Hue(Renk), Saturation(Doygunluk) ve Ligthness(Açıklık) kelimelerinin birleşimidir. Görüntü HSL renk uzayına çevrildikten sonra Sobel Filtre uygulanır. Bu uygulama için görüntünün HSL uzayındaki hali h, l ve s kanallarına ayrılarak filtre için l kanalı kullanılır. Daha sonra Sobel Filtreyi uygulamak için bir OpenCv kodu olan *cv2.Sobel(dst, cv2.CV_64F, 1, 0)* kodu kullanılır. Bu kod sayesinde x değişkeninin türevi alınır ve yatay çizgiler üzerinden Şerit Tespiti için çalışılır. (Bkz. Şekil3.3)



Şekil 3.3 Sobel Görüntü Çıkartılmış

3.4 GÖRÜNTÜ ÜZERİNDE PERSPEKTİF ÇARPITMA UYGULANMASI

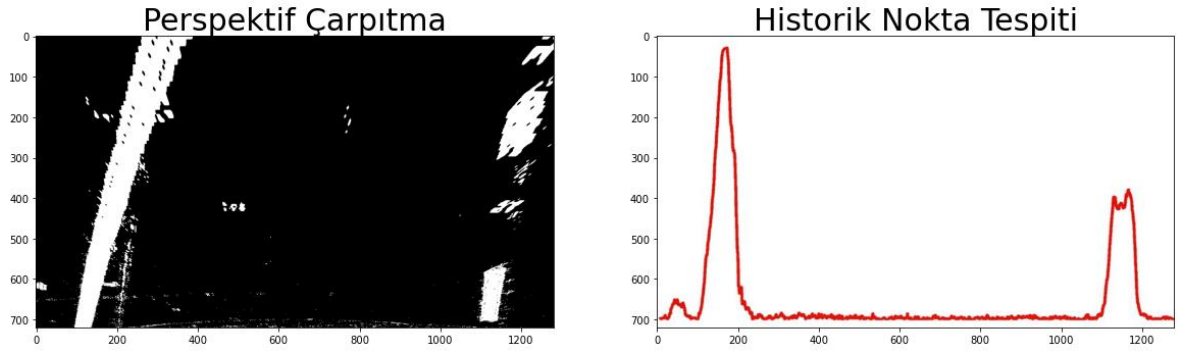
Çalışmanın sıradaki aşamasında Sobel Filtre üzerinden çıkan görüntüyü ilgi alanına göre biçimlendirmek olacaktır. İstenilen şey görüntü içerisinde şerit görüntüsünü ele almak ve sanki ona kuş bakışı bakılıyormuş gibi bir görüntü elde etmektir. Bunun için görüntü üzerinde şeridi içeren belirli bir alan belirlenmelidir. Bu yüzden alanı belirlemek için çeşitli koordinatlar dijital cetveller yardımıyla belirlenerek verilir. Daha sonra istenilen görüntü elde edilir. (Bkz. Şekil 3.4)



Şekil 3.4 Perspektif Çarpıtma Uygulanmış

3.5 HİSTORİK NOKTA TESPİTİ YAPILMASI

Çalışmanın bu aşamasında görüntü üzerindeki şeritlerin daha iyi tespit edilebilmesi amaçlı görüntü üzerine histogram algoritması uygulanılarak yüksek eşik değerli piksellerin ortaya çıkarılması sağlanır. Bu çıkan noktalar Perspektif Çarpıtma uygulamasından çıkan görüntü üzerinden sağlandığından en yüksek eşik değerli noktalar Sağ ve Sol şeridi tespit edilebilmesini sağlar bu da dijital görüntü üzerinde diğer oluşan noktalardan çıkış görüntüsünü arındıracaktır. (Bkz. Şekil 3.5)



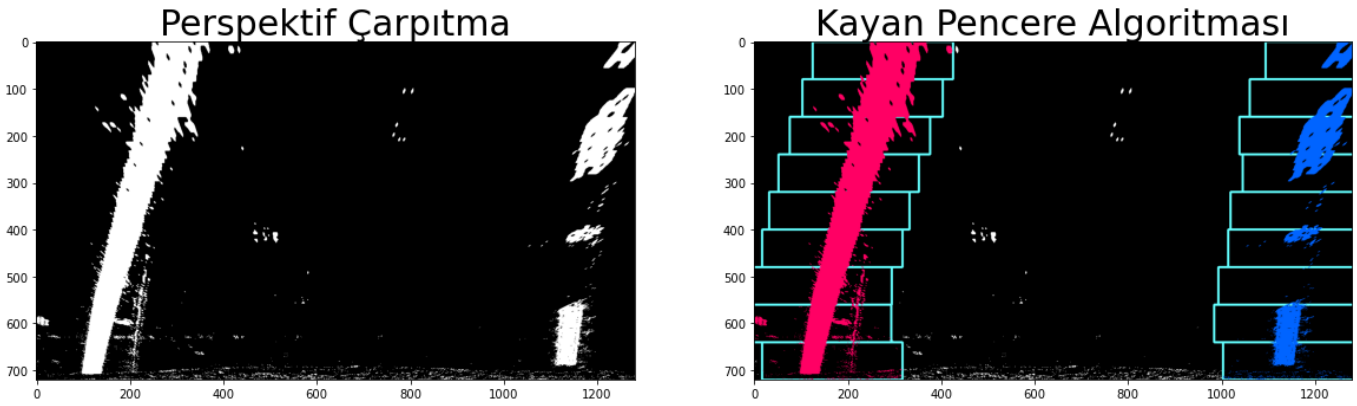
Şekil 3.5 Histerik Nokta Tespiti İşlemi

Burada sağ tarafta gösterilen şekilde yükselen ölçüm değerleri görüldüğü üzere sağ ve sol şerit noktalarında tepe yapmıştır istenilen sonuç elde edilmiştir.

3.6 KAYAN PENCERE ALGORİTMASININ UYGULANMASI(SWA)

Çalışmanın bu adımında Historik Nokta Tespitinden sonra elde edilen piksel noktaları üzerinde Kayan Pencere Algoritması uygulanarak bu pikseller tam anlamıyla tespit edilmiştir. Algoritmanın çalışma mantığı Kullanılan Algoritmalar kısmında anlatılmıştır. Buna göre verilen pencere boyutunda bir dikdörtgen pencere çıkan noktalar üzerinde ilerleyerek nokta tespiti yapacak ve bu aldığı noktaları Sağ ve Sol şerit olarak dizilere

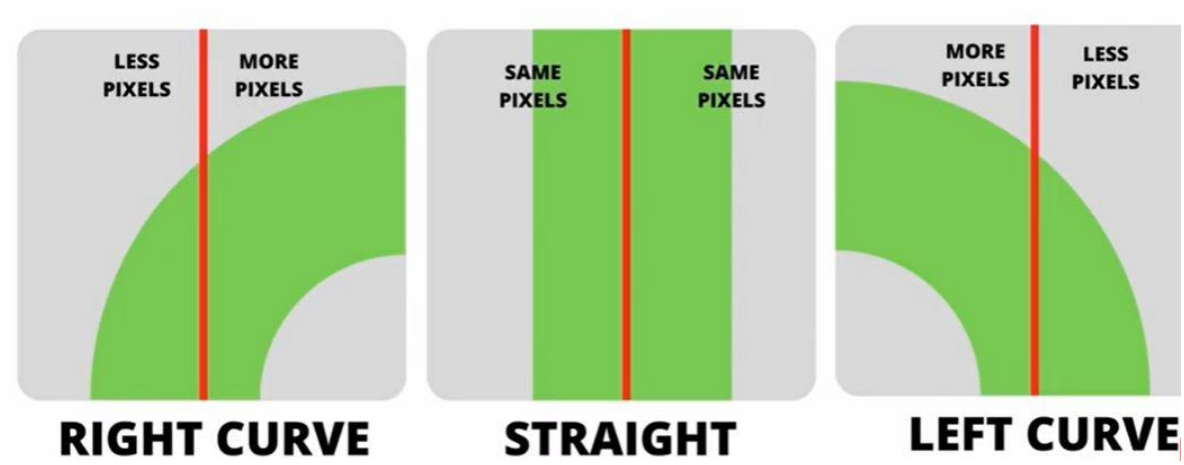
atayacak ve bunları kaydedecektir. Bu noktalar tespit edilirken yazılan kodlar belirli olmayıp algoritmanın mantığınca yazılabilir. (Bkz. Şekil 3.6)



Şekil 3.6 Kayan Pencere Algoritması Uygulanmış

3.7 EĞRİ BULMA ALGORİTMASININ UYGULANMASI

Bu kısımda son olarak uygulanan algoritma Eğri Bulma Algoritmasıdır. Bu algoritma sayesinde şeritlerin kıvrılma durumlarında şerit yine algılanabilecektir. Bu algoritmanın çalışma mantığı bir merkez noktası belirleyerek bu merkez noktası etrafındaki piksel değişimlerini almaktır. Daha sonra bu alınan piksellerdeki değişim miktarına bakarak bu değişim oranında şeritlerin sağa ve sola kıvrıldıklarını tespit eder ve değişim oranı miktarını da yoldaki kıvrılma oranı olarak kaydeder. Bu algoritma yazılırken belirli bir mantık üzerine geliştirilmiştir. OpenCv içerisindeki tek bir hazır koddan yararlanmaz. Bkz. Şekil 3.7.1, Şekil 3.7.2)



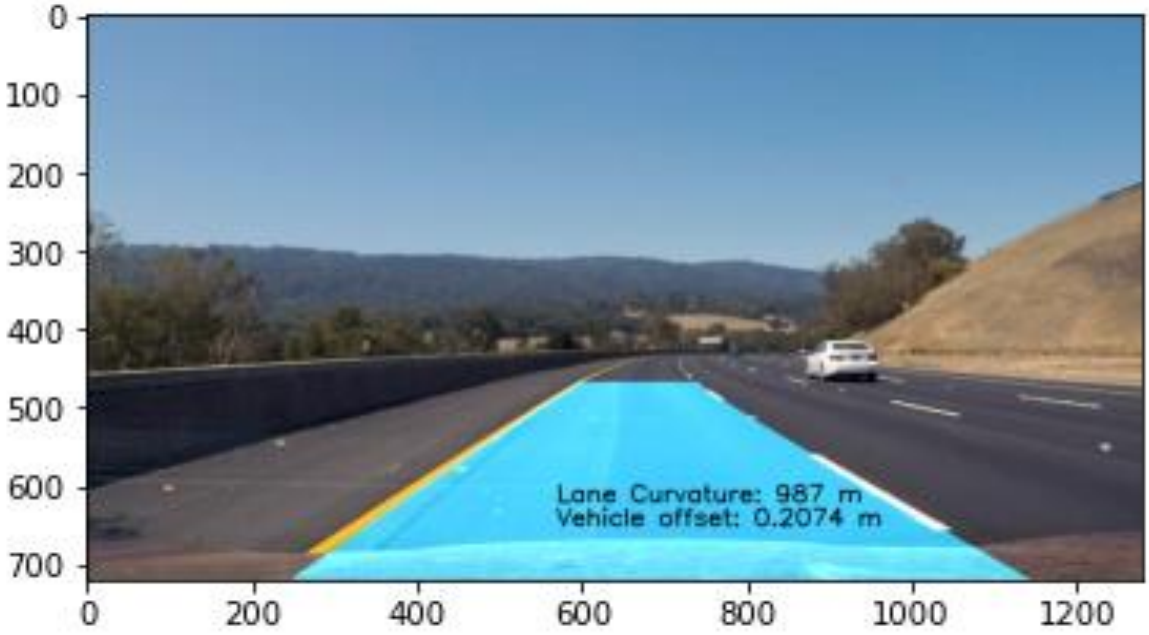
Şekil 3.7.1 Çeşitli Şerit Görüntüleri

Şekil 3.7.2 Offset Metrikleri Çıkartılmış

Görüldüğü üzere şeridin eğimi ve aracın offset' i şekilde belirtilmiştir

3.8 ŞERİTLERİN ÇİZDİRİLMESİ VE ANA GÖRÜNTÜ İLE EKRANA GETİRİLMESİ

Son olarak şerit görüntüsü ile şeritlerin kıvrılma derecesi birleştirilerek görüntünün son hali elde edilir. Bunun için cv2.addWeighted("src1",0.8,"src2",1) fonksiyonu kullanılır ve tespit edilen şerit görüntüsü ekrana getirilir. Bu uygulama tekil bir görüntüden çok bir video içinde denenebilir. Tek yapılması gereken bir kaynak görüntü yerine videonun parçalanarak frame ler halinde kaynak kabul ettirilmesidir.(Bkz. Şekil 3.8)

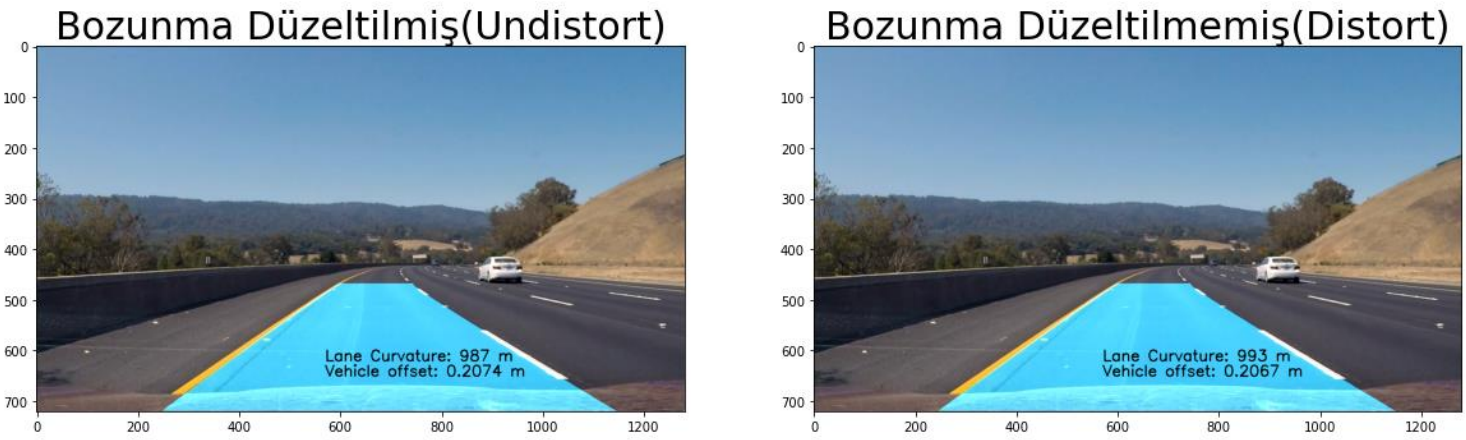


Şekil 3.8 Gelişmiş Şerit Takip Sisteminin Son Hali

4. YAPILAN İKİ UYGULAMANIN KARŞILAŞTIRILMASI

4.1 GÖRÜNTÜDEKİ BOZUNMANIN GİDERİLMESİNİN SONUCU

Çalışma esnasında normal bir kameranın bir insan gözüne oranla 3 boyutu hatalı şekilde algıladığından bahsedilmiştir. Kameralar görüntüyü normal bir görüntüye oranla bozuk şekilde algılıyor (Distortion) ve bunun sonucunda dijital görüntü üzerinde Şerit Takip ve Tespiti yapılırken şeritler gerçekte olduğu konuma göre hatalı bir konumda olduğundan reel bir uygulama karşısında yazılan algoritmalar hatalar çıkarabilir. Bu yüzden ilk içeriye aktarılan görüntü üzerinde bozulma giderme işleminin uygulanması eskisine oranla bize daha iyi bir sonuç vermektedir. (Bkz. Şekil 4.1)

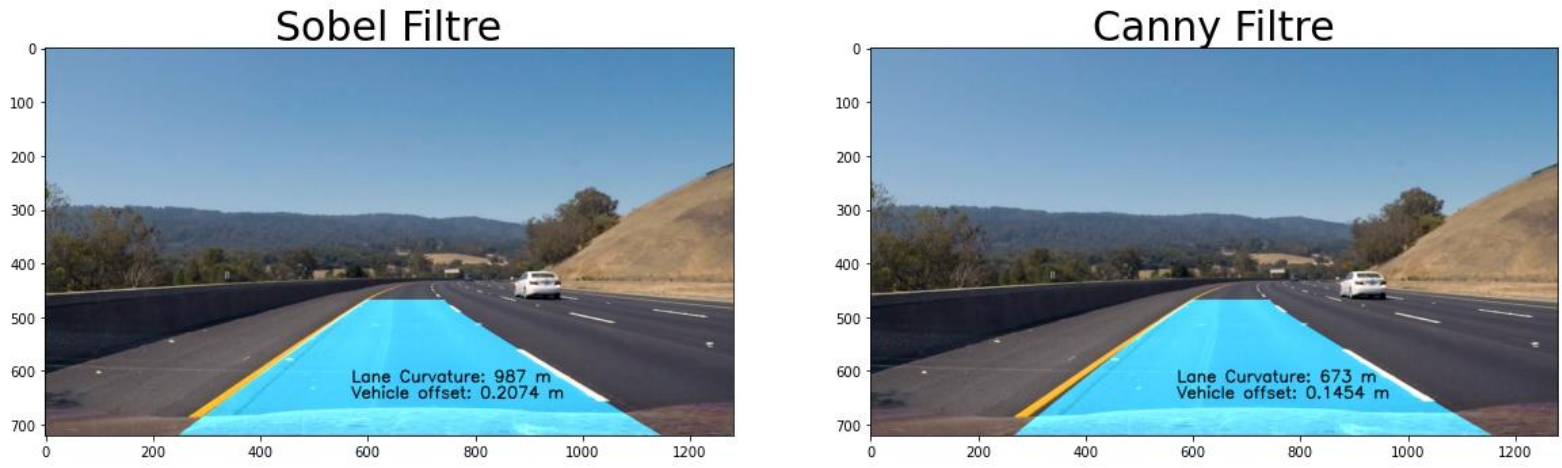


Şekil 4.1 Bozulma Oranları Karşılaştırılması

Yukarıda görüldüğü üzere Lane Curvature (Yol Kavisi) ve Vehicle Offset (Araç Ofseti) değerleri bozulma giderilmez ise sırasıyla +%0.08 ve -%0.04 oranlarında hatalı çıkmaktadır.

4.2 CANNY VE SOBEL KÖŞE BULMA ALGORİTMALARININ KARŞILAŞTIRILMASI

Çalışmada esnasında sırasıyla Temel Şerit Takip ve Tespit Sistemi içerisinde uygulanan Canny Köşe Bulma Algoritması ve Gelişmiş Şerit Takip Sistemi içerisinde uygulanan Sobel Köşe bulma algoritması arasındaki en büyük farklardan birisi Canny algoritmasında köşelerin tespiti için çizilen çizgilerin veya belirleme kernellerinin yatay veya dikey olarak bağımsız ayarlanamamasıdır. Bu yüzden Gelişmiş Sistem diye adlandırılan sistemde yatay ve dikey olarak köşeleri bağımsız olarak belirleyebildiğimiz Sobel Filtre kullanılmıştır. Bunun sebebi şeritlerin görüntülerde dikine şekilde ilerlemesi bu yüzden köşe tespiti için çizgilerin yatay olarak çizilmesidir. Yine değişim miktarları incelenebilir. (Bkz. Şekil 4.2)



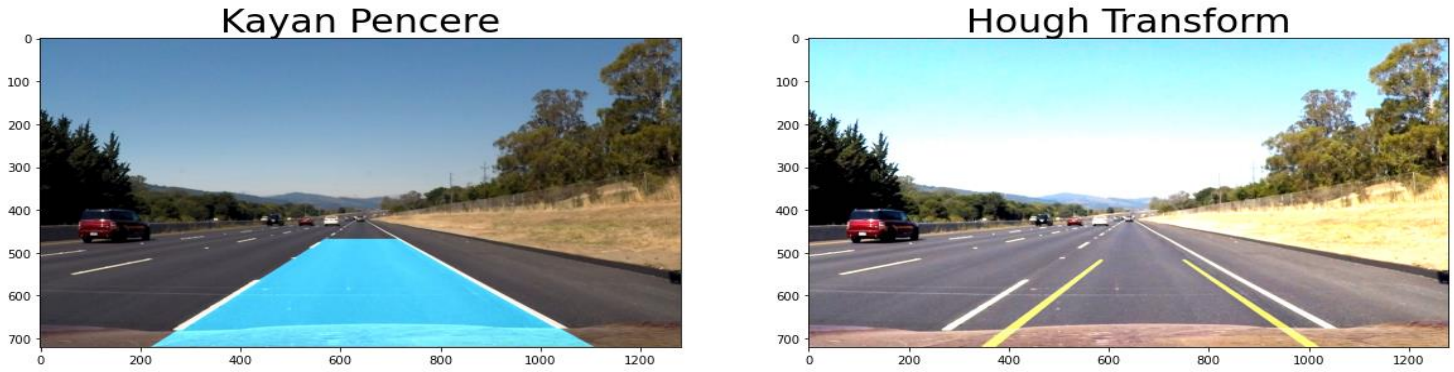
Şekil 4.2 Canny ve Sobel Filtre Uygulamalarının Karşılaştırılması

Yukarıda görüldüğü üzere Lane Curvature (Yol Kavisi) ve Vehicle Offset (Araç Ofseti) değerleri arasında iki şekilde kullanılan algoritmalar birebir aynısı iken çok ciddi farklar oluşmuştur. Bunun sebebi Canny Filtresinden çıkan çıktının hem yatay hem de dikey köşeleri

içermesi ve bunun yanında Sobel filtre içinde kullanılan HSL uzayı ile Sobel Filtrenin daha gürültüsüz bir görüntü üzerinde çalışmasıdır.

4.3 HOUGH TRANSFORM VE KAYAN PENCERE ALGORİTMALARININ KARŞILAŞTIRILMASI

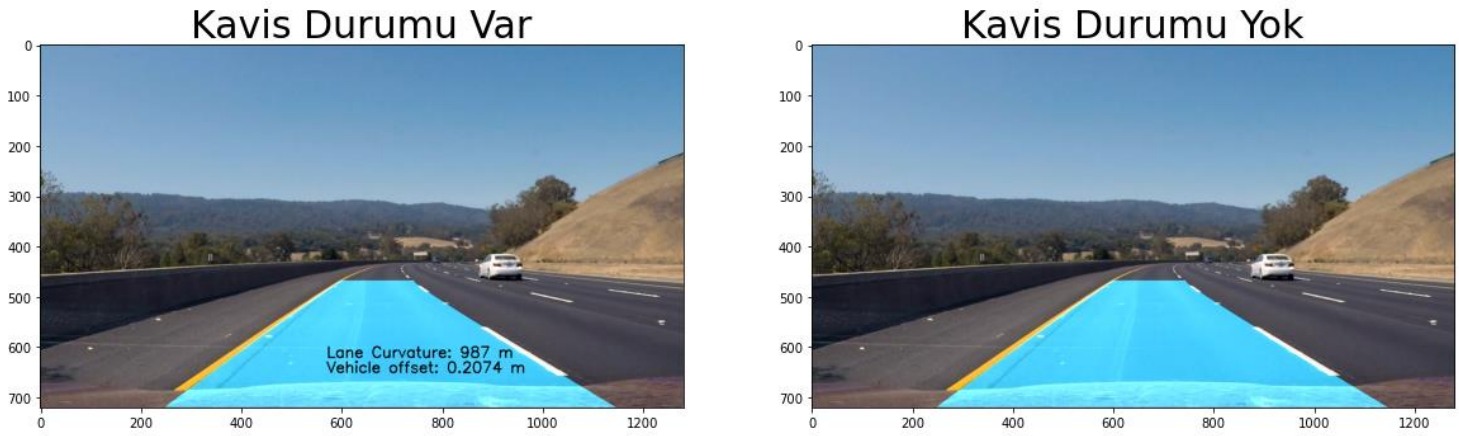
Yine çalışma sırasında Temel Şerit Takip ve Tespit Sistemi içerisinde uygulanan Hough Nokta Dönüşüm Algoritması ve Gelişmiş Şerit Takip ve Tespit Sistemi içerisinde uygulanan Kayan Pencere Algoritması arasındaki en büyük farklardan birisi Hough Dönüşümünün çıktı üzerinde daha tahmini bir çalışma yapmasıdır. Demek istenilen noktalara belirli bir algoritmaya göre tahmini olarak alınmasıdır. Bunun yanı sıra Kayan Pencere Algoritması öncesinde çıktı üzerinde öncesinde Sobel Filtre ile yapılan gürültü temizleme işlemleri ve Historik Nokta Tespiti ile tam olarak pencerelerin nerelere hizalanacağının tahmin edilmesi büyük farklara sebep olabilir. Fakat burada sadece algoritmaları karşılaştırmak amaçlı iki algoritma dışında geri kalan tüm etmenler aynıdır. Ayrıca kavis durumu hesaba katılmadığı için tamamı ile düz bir şerit görüntüsü seçilmiştir. (Bkz. Şekil 4.3)



Şekil 4.3 Kayan Pencere ve Hough Transform Uygulamalarının Karşılaştırılması

4.4 YOLDAKİ KAVİS DURUMU

Çalışmada yer alan Temel Tespit Sisteminin bir diğer eksi yönlerinden birisi de yoldaki kavis durumlarında veri aktaramamasıdır. Fakat aracın hangi yöne doğru hizalanması gerektiğini eğer kavis durumunu bilmiyorsak ilerleyen algoritmalarımızda buna karar veremeyiz bu sebeple büyük önem arz etmektedir. (Bkz. Şekil 4.4)



Şekil 4.4 Kavis Durumlarının Karşılaştırılması

5. SONUÇ VE ÖNERİLER

5.1 SONUÇ

Sonuç olarak iki sistemde uygulanarak her birisinin eksi ve artı yönleri gözlemlenmiştir. Bunun yanı sıra her bir algoritmanın sonuçları teker teker gözlemlenmiştir. Sonuç olarak Gelişmiş Şerit Takip ve Tespit Sistemi diye adlandırılan sistemin Temel Şerit Takip ve Tespit Sistemine göre birçok yönden avantajlı ve kullanışlı olduğu ayrıca reel yaşamda daha uygulanabilir olduğu sonucuna varılmıştır.

5.2 ÖNERİLER

Bu çalışmada her ne kadar Gelişmiş Şerit Takip Sistemi öne çıkarılsa da bu sisteminde birçok eksik yönü vardır. Reel hayata uygulandığında bu daha iyi gözlemlenebilir. Örneğin bu sistemde Kayan Pencere Algoritmasında yerel olarak verilen eşik değerleri bir CNN algoritması ve veri seti ile kolayca tespit edilip otomatikleştirilebilir. Bunun yanında Kalman Filtresinin uygulanması da çok daha iyi sonuçlar doğurabilir. Bu sebeple CV' nin temel parçalarından birisi olan Makine Öğrenmesi uygulamalarının ve algoritmalarının bu sistemlere uygulanması daha doğru ve reel hayata uygulanabilir sonuçlar doğuracaktır.

5.3 GELECEK ÇALIŞMALAR

Bu çalışmada ilerleyen safhalarda literatürde yer alan çeşitli tez ve makaleler incelenecek bu çalışmalarda burada yapılan çalışmalara benzer kısımlardaki performans metrikleri esas alınarak mevcut sistemlerin nasıl daha ileriye taşınabileceği konusunda çalışmalar yapılacaktır. Bunun üzerine konu bazlı yeni makale veya tezler öne sürülebilir.

6. KAYNAKÇA

https://tr.wikipedia.org/wiki/Bilgisayarl%C4%B1_g%C3%B6r%C3%BC [1]

https://www.datasciencearth.com/bilgisayarli-goru-computer-vision/ [2]

https://tr.wikipedia.org/wiki/OpenCV [3]

https://mesutpiskin.com/blog/opencv-nedir.html [4]

https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html [5]

https://ichi.pro/tr/hough-donusumu-ile-hat-algilama-145144313222473 [6]

https://www.hackster.io/kemfic/curved-lane-detection-34f771 [7]

https://en.wikipedia.org/wiki/Sobel_operator [8]

https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html [9]

7. ÖZGEÇMİŞ

Emre DOĞAN 1998’ de İstanbul’ da doğdu. İlköğretimini İsmet Paşa İlkokulunda tamamladıktan sonra lise öğrenimine Abdurrahman Nermin Bilimli Teknik ve Meslek Lisesinde devam etti. 2017 yılında Trakya Üniversitesi Elektrik Elektronik Mühendisliği bölümünde okumaya hak kazandı. Bu mühendislik alanının elektronik bölümünden Görüntü İşleme ve Yapay Zekâ gibi alanlarıyla daha fazla ilgilenmektedir. Ekip çalışmasına yatkın, sorumluluk sahibi ve yaratıcı çözümler bulabilen bir kişiliğe sahiptir.

İletişim Bilgileri:

E-Posta: edoan1998@gmail.com

Adres: Başak Mah. Vadipark Evleri B10 /19 Başakşehir İstanbul