GİTHUB LİNKİ:

https://github.com/EsraTosun/GoruntuIsleme-VizeOdevi

ESRA TOSUN

Vize Ödevi

**Kontrast Güçlendirme ve S-eğrisi Metodu**

Kontrast güçlendirme ve S-eğrisi metodu görüntü işleme alanında önemli bir tekniktir. Bu metod, bir görüntünün kontrastını artırmak veya azaltmak için kullanılır. Kontrast, bir görüntüdeki parlaklık farklılıklarının miktarını ifade eder. Yüksek kontrastlı bir görüntüde, parlak ve koyu bölgeler arasındaki fark belirgin şekilde görünürken, düşük kontrastlı bir görüntüde bu fark daha azdır.

S-eğrisi metodu, bir görüntünün piksel değerlerini dönüştürmek için kullanılan bir yöntemdir. Bu yöntemde, bir S-eğrisi (sigmoid) fonksiyonu kullanılarak piksel değerleri bir eğri boyunca yeniden düzenlenir. Bu eğri, görüntünün kontrastını ayarlamak için önceden tanımlanmış bir şekle sahiptir. S-eğrisi fonksiyonu, genellikle bir giriş aralığındaki piksel değerlerini belirli bir çıkış aralığına dönüştürmek için kullanılır.

1. **Standart Sigmoid Fonksiyonu:**
   * Standart sigmoid fonksiyonu, görüntüdeki piksel değerlerini belirli bir sigmoid fonksiyonu kullanarak dönüştürür. Bu işlem, piksel değerlerini daha dinamik bir aralığa genişleterek kontrastı artırır. Standart sigmoid, belirli bir eşik değeri etrafında yumuşak bir geçiş sağlar.
2. **Yatay Kaydırılmış Sigmoid Fonksiyonu:**
   * Yatay kaydırılmış sigmoid fonksiyonu, standart sigmoid fonksiyonuna bir kaydırma parametresi ekler. Bu parametre, görüntünün kontrastını artırmak veya azaltmak için kullanılır. Bu yöntem, sigmoid fonksiyonunu yatay olarak kaydırarak kontrastı ayarlar.
3. **Eğimli Sigmoid Fonksiyonu:**
   * Eğimli sigmoid fonksiyonu, standart sigmoid fonksiyonuna bir eğim parametresi ekler. Bu parametre, sigmoid fonksiyonunun eğimini ayarlar ve böylece kontrastı kontrol eder. Eğimli sigmoid, görüntünün kontrastını artırmak veya azaltmak için daha keskin bir geçiş sağlar.

**İşlem Adımları:**

1. **Resim Yükleme:**
   * Belirtilen dosya yolu kullanılarak bir resim yükleniyor.
   * Resim, PyQt5'in QPixmap sınıfıyla yüklenip QLabel içinde görüntüleniyor.

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Resim1:**İşlem yapmak için görüntü seçme

1. **Kontrast Güçlendirme Fonksiyonları:**
   * **standard\_sigmoid**, **shifted\_sigmoid**, ve **tilted\_sigmoid** fonksiyonları, sigmoid fonksiyonlarını kullanarak kontrastı güçlendiriyor. Bu fonksiyonlar, sigmoid fonksiyonun farklı parametrelerini kullanarak kontrastı değiştiriyor.
   * **s\_curve\_contrast** fonksiyonu, verilen bir resmi sigmoid fonksiyonuyla dönüştürerek kontrastı güçlendiriyor.

**3.1 standard\_sigmoid Fonksiyonu:**

* + Bu fonksiyon, standart sigmoid fonksiyonunu kullanarak kontrastı güçlendiriyor.
  + Sigmoid fonksiyonu, giriş değerlerini (burada görüntü pikselleri) [0, 1] aralığına dönüştürerek kontrastı artırıyor.
  + Sigmoid fonksiyonunun formülü: 11+𝑒−𝑥1+*e*−*x*1​, burada 𝑥*x* giriş değerini temsil ediyor.
  + Parametre olarak sadece 𝑥*x* alır ve giriş değerlerini standart sigmoid fonksiyonuna uygular.

**3.2 shifted\_sigmoid Fonksiyonu:**

* + Bu fonksiyon, sigmoid fonksiyonunun girişini yatay olarak kaydırarak kontrastı değiştiriyor.
  + Standart sigmoid fonksiyonundan farklı olarak, 𝑥*x* değerleri sigmoid fonksiyonuna 𝑐*c* parametresiyle çarpılarak kaydırılıyor.
  + Formülü: 11+𝑒−𝑐⋅(𝑥−0.5)1+*e*−*c*⋅(*x*−0.5)1​
  + Bu kaydırma işlemi, sigmoid fonksiyonunun merkezini değiştirerek kontrastı etkili bir şekilde değiştirir.
  + Varsayılan olarak 𝑐=1*c*=1 olarak belirlenmiştir.

**3.3 tilted\_sigmoid Fonksiyonu:**

* + Bu fonksiyon, sigmoid fonksiyonunun eğimini değiştirerek kontrastı ayarlar.
  + Standart sigmoid fonksiyonundan farklı olarak, sigmoid fonksiyonunun giriş değerlerini 𝑎*a* parametresiyle çarpar.
  + Formülü: 11+𝑒−𝑎⋅(𝑥−0.5)1+*e*−*a*⋅(*x*−0.5)1​
  + 𝑎*a* parametresi, sigmoid fonksiyonunun eğimini kontrol eder. Daha büyük 𝑎*a* değerleri, daha dik bir eğim ve daha yüksek kontrast sağlar.
  + Varsayılan olarak 𝑎=1*a*=1 olarak belirlenmiştir.

**3.4 s\_curve\_contrast Fonksiyonu:**

* + Bu fonksiyon, verilen bir resmi sigmoid fonksiyonuyla dönüştürerek kontrastı güçlendirir.
  + İlk olarak, görüntü tek boyuta dönüştürülür ve sigmoid fonksiyonuna uygulanır.
  + Daha sonra, sonuç [0, 1] aralığına ölçeklenir ve piksel değerlerine dönüştürülür.
  + Son olarak, dönüştürülmüş görüntü yeniden şekillendirilir ve uint8 türüne dönüştürülerek döndürülür.
  + Bu fonksiyon, farklı sigmoid fonksiyonlarını kullanarak kontrastı güçlendirmek için kullanılır.

1. **Sonuçların Görselleştirilmesi:**

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Resim2:** İşlemden sonra oluşan görüntüler

* + Matplotlib kütüphanesi kullanılarak, orijinal resim ve farklı kontrast güçlendirme yöntemlerinin sonuçları dört farklı alt görselde gösteriliyor.
  + Her bir alt görselde, orijinal resim ve bir kontrast güçlendirme yönteminin sonucu görüntüleniyor.

**Yol Çizgilerini Bulma İşlemi:**

**Hough Dönüşümü Metodu :**

Hough Dönüşümü, bir görüntüdeki doğrusal yapıları algılamak için kullanılan bir tekniktir. Bu yöntem, özellikle kenar algılama gibi işlemler sonrasında elde edilen kenar piksellerini kullanarak doğrusal yapıları tespit etmek için etkilidir. Genellikle, düz çizgilerin tespitinde kullanılır, ancak aynı prensipler eğri veya halkaları tespit etmek için de uygulanabilir.

Hough Dönüşümü, bir görüntüdeki her bir piksel için bir parametrik uzayda (tipik olarak x-y uzayı yerine eğim-başlangıç noktası uzayı) bir nokta oluşturur. Bu noktaların birleştirilmesi, görüntüde doğrusal yapıları temsil eden çizgileri veya eğrileri belirler. Hough Dönüşümü, bu parametrik uzayda doğrusal yapıları temsil eden çizgilerin birikimini bulmak için kullanılan bir akümülasyon (birleşim) yöntemi kullanır.

**Hough Dönüşümü ve Uygulama :**

Bu uygulamada, öncelikle bir görüntü üzerinde kenar algılama (Canny Edge Detection) işlemi uygulanır. Kenar algılama sonrasında, Hough Dönüşümü metodu (cv2.HoughLinesP) kullanılarak görüntüdeki doğrusal yapılar tespit edilir. Bu yöntem, kenar algılama sonucunda elde edilen kenar piksellerinden doğrusal yapıları bulmak için kullanılır.

**Uygulama Adımları ve Önemi:**



**Resim 3:** İşlem yapmak için görüntü seçme

1. **Ön İşleme (Preprocessing) Adımları:**

Görüntü, gürültüyü azaltmak ve işleme süresini iyileştirmek için Gaussian bulanıklığı ve morfolojik işlemler ile ön işleme yapılır.

* + **Gaussian Bulanıklığı (Gaussian Blur):**
    - *Ne Yapıldı:* Görüntüye 5x5 boyutunda bir Gaussian bulanıklığı uygulandı.
    - *Neden Kullanıldı:* Gürültüyü azaltarak daha temiz bir görüntü elde edildi. Bu, kenar tespitini iyileştirir ve yanlış pozitifleri azaltır.
  + **Morfolojik İşlemler (Morphological Operations):**
    - *Ne Yapıldı:* Aşınma (Erosion) ve genişletme (Dilation) işlemleri uygulandı.
    - *Aşınma (Erosion):* Küçük detayları ve gürültüyü azalttı, kenarları netleştirdi.
    - *Genişletme (Dilation):* Kenarları birleştirdi ve tamamladı, kesintili kenarları düzgün hale getirdi.

1. **Kenar Tespiti (Edge Detection):**

Canny Edge Detection yöntemiyle görüntü üzerinde kenarlar tespit edilir. Bu adım, görüntüdeki doğrusal yapıların temelini oluşturur.

**Canny Kenar Tespiti (Canny Edge Detection):**

* + - *Ne Yapıldı:* Ön işleme adımları uygulanan görüntüde kenarlar tespit edildi.
    - *Neden Kullanıldı:* Keskin kenarları belirledi ve kenarları izole etti. Bu, Hough dönüşümü için gerekli olan doğru çizgilerin belirlenmesine yardımcı oldu.

1. **Hough Dönüşümü (Hough Transformation):**

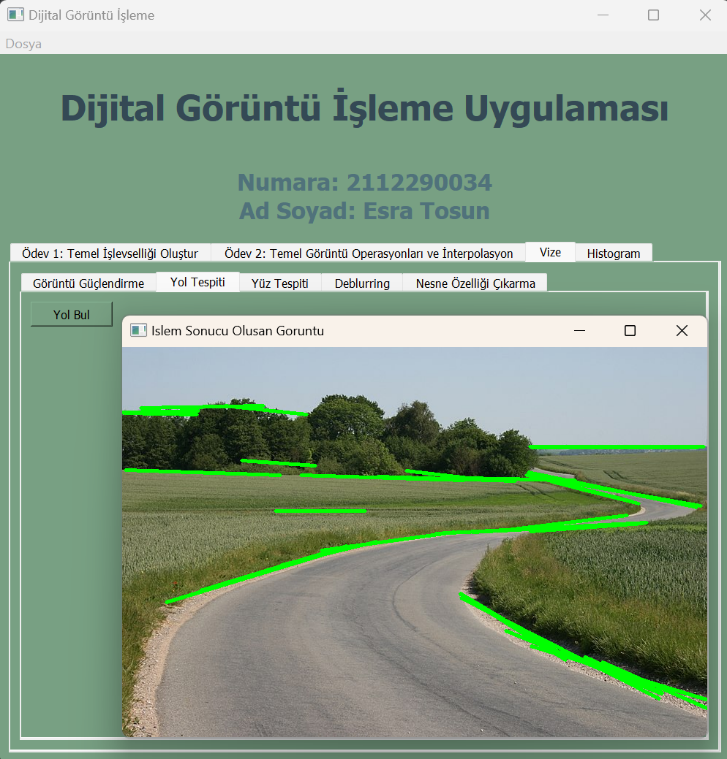
Kenarlar üzerinde Hough Dönüşümü uygulanarak doğrusal yapılar (çizgiler) tespit edilir. Bu adım, görüntüdeki yolların, hatların veya diğer doğrusal yapıların tespitini sağlar.

* + *Ne Yapıldı:* Kenarlar tespit edildikten sonra, Hough dönüşümü uygulandı.
  + *Neden Kullanıldı:* Doğru çizgilerin tespit edilmesini sağladı. Bu, görüntüdeki yolları veya çizgileri belirlemek için kullanıldı.

1. **Sonuçlar ve Görselleştirme:**

Tespit edilen doğrusal yapılar, orijinal görüntü üzerine çizilerek görselleştirilir. Bu adım, kullanıcıların doğrusal yapıları görsel olarak incelemesini ve doğruluğunu değerlendirmesini sağlar.

* + *Ne Yapıldı:* Hough dönüşümü sonucunda tespit edilen doğru çizgiler, orijinal görüntü üzerine çizildi.
  + *Neden Kullanıldı:* Yol çizgilerinin görsel olarak belirlenmesini sağladı. Bu, görüntüdeki yolların ve çizgilerin belirlenmesi için kullanıldı.



**Resim4:** İşlem sonucu oluşan görüntü

1. **Parametre Değerleri ve Seçimleri:**

• **Gaussian Bulanıklığı ve Morfolojik İşlemler İçin Kernel Boyutu:** Her ikisi de 5x5 olarak seçildi. Bu boyut, hem gürültüyü azaltmak hem de kenarları belirginleştirmek için uygun görüldü.

• **Canny Kenar Tespiti İçin Eşik Değerleri:** 50 (düşük eşik) ve 150 (yüksek eşik) olarak belirlendi. Bu değerler, görüntüdeki keskin kenarları belirlemek için uygun seçildi.

• **Hough Dönüşümü İçin Parametreler:** Threshold=50, minLineLength=100, maxLineGap=50 olarak belirlendi. Bu değerler, doğru çizgilerin belirlenmesi ve gereksiz çizgilerin filtrelenmesi için uygun olarak seçildi.

**Yüz Resminde Gözleri Tespit Eden Uygulama**

Bu uygulama, OpenCV kütüphanesini kullanarak yüz ve göz tespiti yapar. İlk olarak, görüntüye yüzleri tespit etmek için bir Haar Cascade sınıflandırıcısı kullanılır. Daha sonra, her bir yüz bölgesinde gözleri tespit etmek için ayrı bir Haar Cascade sınıflandırıcısı kullanılır. Gözler tespit edildikten sonra, tespit edilen gözlerin çevresine dikdörtgen çizilir.

**Hough Transform Metodu**

Hough dönüşümü, hattı veya daireyi algılamak için kullanılan bir görüntü işleme tekniğidir. Hough dönüşümü, matematiksel olarak belirlenmiş bir modelin (örneğin, bir doğru veya dairenin) belirsiz bir şekilde görüntüdeki piksel değerlerine dönüştürülmesini sağlar. Bu, çizginin veya dairenin parametrelerinin bir uzayında bir noktaya karşılık gelir. Daha sonra, bu uzayda birikimli bir oy kullanarak, orijinal görüntüdeki çizgilerin veya dairelerin bulunmasını sağlar.

**1. Giriş**

Bu rapor, verilen kodun işlevselliğini ve içeriğini analiz ederek, yüz resmindeki gözleri tespit eden uygulamayı ve Hough Transform metodunu inceler. İşlemler adım adım açıklanmış ve sonuçlar görsellerle desteklenmiştir.

**2. Kod İncelemesi**

**1. CascadeClassifier Sınıfının Oluşturulması**

Haar Cascade sınıflandırıcılarını kullanmak için öncelikle **CascadeClassifier** sınıfını kullanmalıyız. Bu sınıf, eğitilmiş sınıflandırıcıları yüklemek ve nesne tespiti yapmak için gereklidir.

* face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')
* eye\_cascade = cv2.CascadeClassifier('haarcascade\_eye.xml')

Burada, **haarcascade\_frontalface\_default.xml** dosyası yüz tespiti için kullanılacak sınıflandırıcıyı, **haarcascade\_eye.xml** dosyası ise göz tespiti için kullanılacak sınıflandırıcıyı içerir.

**2. Yüz Tespiti (face\_cascade)**

Yüz tespiti için Haar Cascade sınıflandırıcısını kullanırken, genellikle **detectMultiScale()** fonksiyonunu kullanırız.

* faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
* **gray**: Görüntüyü gri tonlamaya dönüştürür. Haar Cascade sınıflandırıcıları genellikle gri tonlamalı görüntülerle daha iyi çalışır.
* **scaleFactor**: Görüntüyü küçültmek için kullanılır. Daha küçük değerler daha hassas ancak daha yavaş sonuçlar sağlar.
* **minNeighbors**: Her bir tespit için gereken komşu dikdörtgenlerin sayısını belirtir. Bu parametre, yanlış pozitif tespitleri azaltmaya yardımcı olur.

**3. Göz Tespiti (eye\_cascade)**

Göz tespiti için de yüz tespitiyle aynı mantığı kullanırız, ancak bu sefer yüzün ROI'su (Region of Interest) üzerinde çalışırız.

* eyes = eye\_cascade.detectMultiScale(roi\_gray)

**roi\_gray:** Yüz bölgesinin gri tonlamalı versiyonudur. Bu bölgede göz tespiti yapılır.Diğer parametreler genellikle varsayılan değerlerle bırakılır.

1. **Tespit Edilen Nesnelerin Çevrelenmesi**

Nesnelerin etrafına dikdörtgen çizmek için **cv2.rectangle()** fonksiyonunu kullanırız.

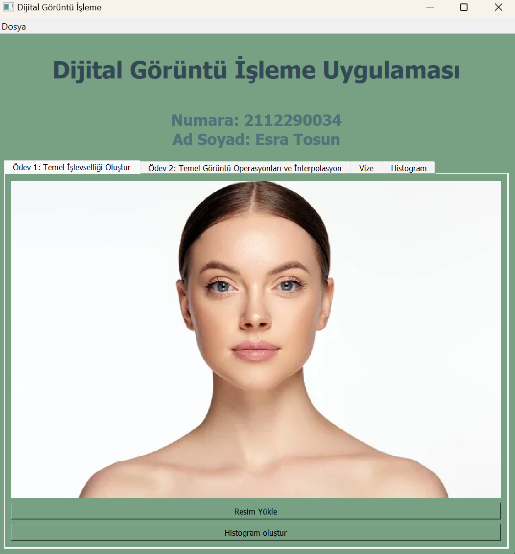
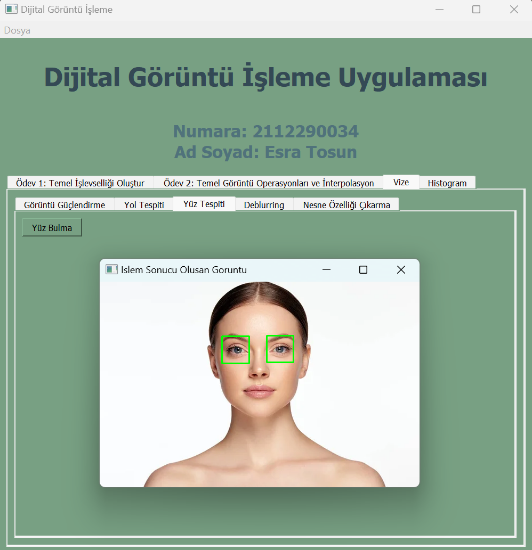
* cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
* img: Dikdörtgenin çizileceği görüntü.
* **x , y:** Dikdörtgenin sol üst köşesinin koordinatları.
* **w, h:** Dikdörtgenin genişliği ve yüksekliği.
* **(255,0,0):** Dikdörtgenin rengi (mavi, yeşil, kırmızı).
* **2:** Dikdörtgenin kalınlığı.

**3. Hough Transform Metodu**

* **Tanımı**: Hough dönüşümü, doğru veya daire gibi şekillerin tespiti için kullanılan bir görüntü işleme tekniğidir. Bu dönüşüm, matematiksel model parametrelerinin bir uzayında birikimli bir oy kullanarak şekillerin algılanmasını sağlar.
* **Uygulama**: Kod içerisinde doğrudan Hough dönüşümü uygulanmamıştır. Ancak, göz tespiti için kullanılan Haar Cascade sınıflandırıcısı zaten bir tespit algoritmasıdır ve Hough dönüşümünü içermeyebilir.

**4. Sonuçlar**

* **Yüz ve Göz Tespiti**: Kod, başarıyla yüzleri ve bu yüzlerin içerisindeki gözleri tespit etmektedir. Tespit edilen yüzler ve gözler, yeşil dikdörtgenlerle vurgulanarak görsel olarak sunulmuştur.

**Resim5:** Seçilen Resim **Resim6:** İşlem sonucu oluşan görüntü

**Deblurring algoritması**

Deblurring (bulanıklık giderme) algoritmaları, bir görüntüdeki bulanıklığı azaltmak veya tamamen gidermek için kullanılan yöntemlerdir. Görüntülerdeki bulanıklık, genellikle hareket bulanıklığı (motion blur), lens bulanıklığı, odaklanma sorunları veya diğer çeşitli faktörlerden kaynaklanabilir. Deblurring algoritmaları, bu tür bulanıklıkları düzeltmek için matematiksel ve hesaplamalı teknikler kullanır.

1. **Görüntüyü Yükleme:**
   * İşlemin başında, belirli bir dosya yolundan bir görüntü yüklenir. Bu adım, işlenecek olan görüntünün programa alınması için gereklidir. Görüntü, hareket bulanıklığını gidermek için temel veri kaynağımızdır.
2. **Hareket Bulanıklığı Çekirdeği Oluşturma:**
   * Hareket bulanıklığını gidermek için bir çekirdek oluşturulur. Bu çekirdek, hareketin yönüne ve boyutuna bağlı olarak belirlenir. Hareket bulanıklığını doğru bir şekilde düzeltebilmek için doğru çekirdek boyutu ve açısı önemlidir.
3. **Çekirdeği Normalize Etme:**
   * Oluşturulan çekirdek normalize edilir. Bu, filtreleme işlemini daha doğru ve tutarlı hale getirir. Normalize edilmemiş bir çekirdek, beklenen sonuçları üretmeyebilir veya istenmeyen yan etkilere neden olabilir.
4. **Hareket Bulanıklığını Giderme:**
   * Oluşturulan çekirdek kullanılarak görüntü üzerine filtreleme işlemi uygulanır. Bu işlem, görüntüdeki hareket bulanıklığını azaltır veya ortadan kaldırır. Özellikle sabit bir çekim açısında veya hareketli bir nesnenin izlendiği durumlarda oluşan bulanıklığı gidermek için önemlidir.
5. **Kontrast ve Parlaklık Ayarlama:**
   * Filtrelenmiş görüntünün kontrastı ve parlaklığı ayarlanır. Bu adım, filtreleme işlemi sonrasında elde edilen görüntünün daha net ve anlaşılır olmasını sağlar. Ayrıca, filtreleme işlemi sırasında kaybedilen kontrastın geri kazanılmasına yardımcı olur.
6. **Sonuçların Gösterilmesi:**
   * İşlemlerin sonucu, orijinal ve düzeltilmiş görüntülerin görsel olarak karşılaştırılmasıyla gösterilir. Bu, kullanıcıya işlemlerin etkisini açıkça gösterir ve yapılan düzeltmelerin doğruluğunu kontrol etmeye olanak tanır.

**Kullanılan Methodlar:**

1. **cv2.imread()**: Bu method, belirtilen dosya yolundan bir görüntüyü okur. Kodda, başlangıçta belirtilen görüntüyü yüklemek için kullanılmıştır.
2. **cv2.filter2D()**: Bu method, belirli bir filtre çekirdeği kullanarak bir görüntüyü filtreler. Kodda, hareket bulanıklığını gidermek için bu yöntem kullanılmıştır.
3. **cv2.convertScaleAbs()**: Bu method, bir görüntüyü belirli bir alfa ve beta değeriyle ölçeklendirir ve bir dizi görüntüyü birleştirir. Kodda, filtrelenmiş görüntünün kontrastını ve parlaklığını ayarlamak için kullanılmıştır.
4. **cv2.getRotationMatrix2D()**: Bu method, bir döndürme matrisi döndürür. Kodda, hareket bulanıklığı çekirdeği oluşturmak için kullanılmıştır.
5. **cv2.warpAffine()**: Bu method, bir görüntüyü bir döndürme matrisiyle döndürür. Kodda, hareket bulanıklığı çekirdeği oluşturmak için kullanılmıştır.
6. **plt.figure()**, **plt.subplot()**, ve **plt.imshow()**: Bu matplotlib methodları, grafikler oluşturmak ve görüntüleri göstermek için kullanılmıştır.

**İşlemler ve Seçimler:**

1. Görüntü yükleme: Başlangıçta, **cv2.imread()** methodu kullanılarak belirtilen dosya yolundan bir görüntü yüklenmiştir.



**Resim7:** Görüntü seçimi

1. Hareket bulanıklığı çekirdeği oluşturma: **deblur\_motion\_blur** fonksiyonunda, belirli bir boyutta ve açıda bir dizi oluşturularak hareket bulanıklığı çekirdeği oluşturulmuştur. Çekirdek, kernel boyutu ve hareket açısı parametreleri kullanılarak belirlenmiştir.
2. Çekirdeği normalize etme: Oluşturulan çekirdek, normalize edilerek filtreleme işlemi için hazırlanmıştır.
3. Hareket bulanıklığını giderme: **cv2.filter2D()** methodu kullanılarak görüntü üzerine filtreleme işlemi uygulanmış ve hareket bulanıklığı giderilmiştir.
4. Kontrast ve parlaklık ayarlama: **cv2.convertScaleAbs()** methodu kullanılarak filtrelenmiş görüntünün kontrastı ve parlaklığı ayarlanmıştır.
5. Sonuçların gösterilmesi: **plt.figure()**, **plt.subplot()**, ve **plt.imshow()** methodları kullanılarak orijinal ve düzeltilmiş görüntülerin görsel olarak karşılaştırılması sağlanmıştır.

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

Açıklama otomatik olarak oluşturuldu

**Resim8:** İşlem sonucu olucan görüntü

Bu işlemler, hareket bulanıklığını gidermek için kullanılan bir algoritmanın uygulanmasını sağlar. Hareket bulanıklığı, genellikle sabit bir çekim açısında hareket eden bir kamera nedeniyle oluşur ve görüntüdeki netliği azaltır. Bu algoritma, bu tür bulanıklığı azaltmak için yaygın olarak kullanılan bir yaklaşımı uygular.

**Nesne Özelliği Çıkarma Nedir?**

Nesne Özelliği Çıkarma, bir görüntü üzerindeki nesnelerin tanımlayıcı özelliklerini elde etmek için kullanılan bir işlemdir. Bu özellikler, nesnelerin morfolojik yapısı, renk dağılımı, yoğunluk profili vb. olabilir. Nesne özellikleri, görüntü analizi, nesne tanıma, nesne takibi gibi uygulamalarda önemli bilgiler sağlar.

**İşlem Adımları ve Önemi:**

1. **Görüntüyü Yükleme:**
   * İşlemin başında, bir görüntü yüklenir. Bu görüntü, analiz edilecek olan verinin kaynağıdır.

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Resim9:** Görüntü Yükleme

1. **Renk Aralığının Tanımlanması:**
   * Verilen görseldeki "koyu yeşil" bölgelerin tanımlanması için bir renk aralığı belirlenir. Bu aralık, HSV (ton, doygunluk, değer) renk uzayında belirli bir aralığı ifade eder.
2. **Renk Uzayının Değiştirilmesi:**
   * Belirlenen renk aralığına göre maske oluşturmak için görüntünün renk uzayı BGR'den HSV'ye dönüştürülür. HSV renk uzayı, renk aralıklarının daha doğru belirlenmesine ve nesnelerin renklerine bağlı olarak sınıflandırılmasına olanak tanır.
3. **Maske Oluşturma:**
   * Belirlenen renk aralığına göre bir maske oluşturulur. Bu maske, görüntüdeki "koyu yeşil" bölgeleri belirtir.
4. **Kontur Bulma ve Çizme:**
   * Oluşturulan maske kullanılarak konturlar (sınırlar) bulunur ve bu konturlar görüntü üzerine çizilir. Bu adım, nesnelerin şekillerini belirlemek için kullanılır.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Resim10:** İşlem sonucu oluşan görüntü

1. **Özelliklerin Hesaplanması:**
   * Her bir kontur için belirli özellikler hesaplanır. Bu özellikler, nesnelerin boyutu, merkezi konumu, uzunluğu, genişliği, enerjisi, entropisi, ortalama ve medyan renk değerleri gibi olabilir.
2. **Veri Analizi ve Depolama:**
   * Hesaplanan özellikler bir veri yapısında saklanır ve analiz edilir. Bu veri, genellikle bir DataFrame gibi yapılandırılmış bir forma dönüştürülerek daha kolay işlenebilir hale getirilir.
3. **Excel Tablosu Oluşturma:**
   * Hesaplanan özellikler bir Excel dosyasına yazılır. Bu dosya, elde edilen verilerin saklanması, paylaşılması ve daha sonra referans alınması için kullanılır.

Nesne özelliği çıkarma, bir görüntüdeki nesnelerin tanımlayıcı özelliklerini elde etmek için önemli bir adımdır. Bu özellikler, nesnelerin tanınması, sınıflandırılması ve takibi gibi birçok uygulamada kullanılabilir. Özellikle görüntü analizi ve makine öğrenimi gibi alanlarda yaygın olarak kullanılır.

**Kullanılan Methodlar:**

1. **cv2.imread():**
   * Bu yöntem, bir görüntü dosyasını okumak için kullanılır. Bu kodda, işlenecek olan görüntüyü yüklemek için kullanılmıştır.
2. **cv2.cvtColor():**
   * Görüntü üzerinde renk dönüşümü yapmak için kullanılır. Bu kodda, görüntünün renk uzayını BGR'den HSV'ye dönüştürmek için kullanılmıştır. HSV renk uzayı, renklerin daha doğru belirlenmesine olanak tanır.
3. **cv2.inRange():**
   * Belirli bir renk aralığına göre maske oluşturmak için kullanılır. Bu kodda, "koyu yeşil" bölgeleri belirlemek için kullanılmıştır.
4. **cv2.findContours() ve cv2.drawContours():**
   * Görüntü üzerindeki konturları bulmak ve çizmek için kullanılır. Bu kodda, oluşturulan maske kullanılarak "koyu yeşil" bölgelerin sınırları belirlenmiş ve görüntü üzerine çizilmiştir. Bu yöntem, nesnelerin şekillerini belirlemek için kullanılmıştır.
5. **cv2.moments() ve cv2.HuMoments():**
   * Nesnelerin geometrik özelliklerini (merkezi, alan, momentler vb.) hesaplamak için kullanılır. Bu kodda, her bir kontur için momentler hesaplanmış ve bu momentler kullanılarak enerji ve entropi gibi özellikler elde edilmiştir.
6. **np.mean() ve np.median():**
   * Bir görüntünün belirli bir bölgesinin ortalama ve medyanını hesaplamak için kullanılır. Bu kodda, "koyu yeşil" bölgelerin ortalama ve medyan renk değerlerini hesaplamak için kullanılmıştır.
7. **pd.DataFrame() ve df.to\_excel():**
   * Pandas kütüphanesi ile bir veri çerçevesi oluşturmak ve bu verileri bir Excel dosyasına yazmak için kullanılmıştır. Bu yöntem, hesaplanan nesne özelliklerini bir yapı içinde düzenlemek ve saklamak için kullanılmıştır.