

EGE ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



GÖRÜNTÜ İŞLEME
2021-2022 GÜZ YARIYILI
PROJE 1 RAPORU

PROJE KONUSU: Görüntü İşleme Yazılımı

HAZIRLAYAN
05170000022 – Mehmet Anıl TAYSI
05180000087 – Emel KAYACI

İçindekiler

1. Görüntü Yükleme ve Kaydetme (Load/Read, Save)	1
1.1 Görüntü Yükleme	1
1.2 Görüntü Kaydetme	3
2. Arayüz / Form Ortamı Hazırlama	5
3. Görüntü İyileştirme İşlemleri, Filtreler	6
3.1 Prewitt Filtresi	6
3.2 Hessian Filtresi.....	6
3.3 Gaussian Filtresi.....	7
3.4 Sato Filtresi	7
3.5 Meijering Filtresi.....	8
3.6 Sobel Filtresi	8
3.7 Unsharp Mask Filtresi	9
3.8 Laplacian Filtresi	9
3.9 Bilateral Filtresi.....	10
3.10 Scharr Filtresi.....	10
4. Histogram Görüntüleme ve Eşitleme	11
5. Uzaysal Dönüşüm İşlemleri.....	13
5.1 Rotate İşlemi	14
5.2 Swirl İşlemi	15
5.3 Rescale İşlemi	15
5.4 Warp Polar İşlemi	16
5.5 Resize İşlemi	16
6. Yoğunluk Dönüşüm İşlemleri.....	17
7. Morfolojik İşlemler	18
7.1 Dilation İşlemi.....	18
7.2 Erosion İşlemi	19
7.3 Skeletonize İşlemi	19
7.4 Thinning İşlemi	20
7.5 Küçük Objeleri Kaldırma İşlemi	20
7.6 Top-Hat İşlemi	21
7.7 Opening İşlemi.....	21
7.8 Closing İşlemi.....	22

7.9 Morphological Gradient İşlemi	22
7.10 Black-Hat İşlemi	23
8. Video İşleme	23
9. Özdeğerlendirme Tablosu ve İş Bölümü	25
10. Kaynakça	26

1. Görüntü Yükleme ve Kaydetme (Load/Read, Save)

1.1 Görüntü Yükleme

```
def upload_image():
    # Önceki resmi silmek için kullanılır.
    for widget in mainFrame.winfo_children():
        widget.destroy()
    # Fonksiyon sona erdikten sonra resmin ekranda
    görüntülenebilmesi için global tanımlanmalıdır.
    global loaded_image
    global image
    root.filename =
    filedialog.askopenfilename(initialdir='/', title='Select
    File', filetypes=(('png files', '.png'), ('jpeg files',
    '.jpg'), ('all files', '*..*')))
    try:
        image = Image.open(root.filename).resize((345,
    325))
        loaded_image = ImageTk.PhotoImage(image)
    except PIL.UnidentifiedImageError:
        showerror('Hata!', 'Resim formatına uygun dosya
    seçimi yapınız.')
    else:
        mainImage = tk.Label(mainFrame,
        image=loaded_image)
        mainImage.pack()
```

Kod 1. Görüntü yükleme

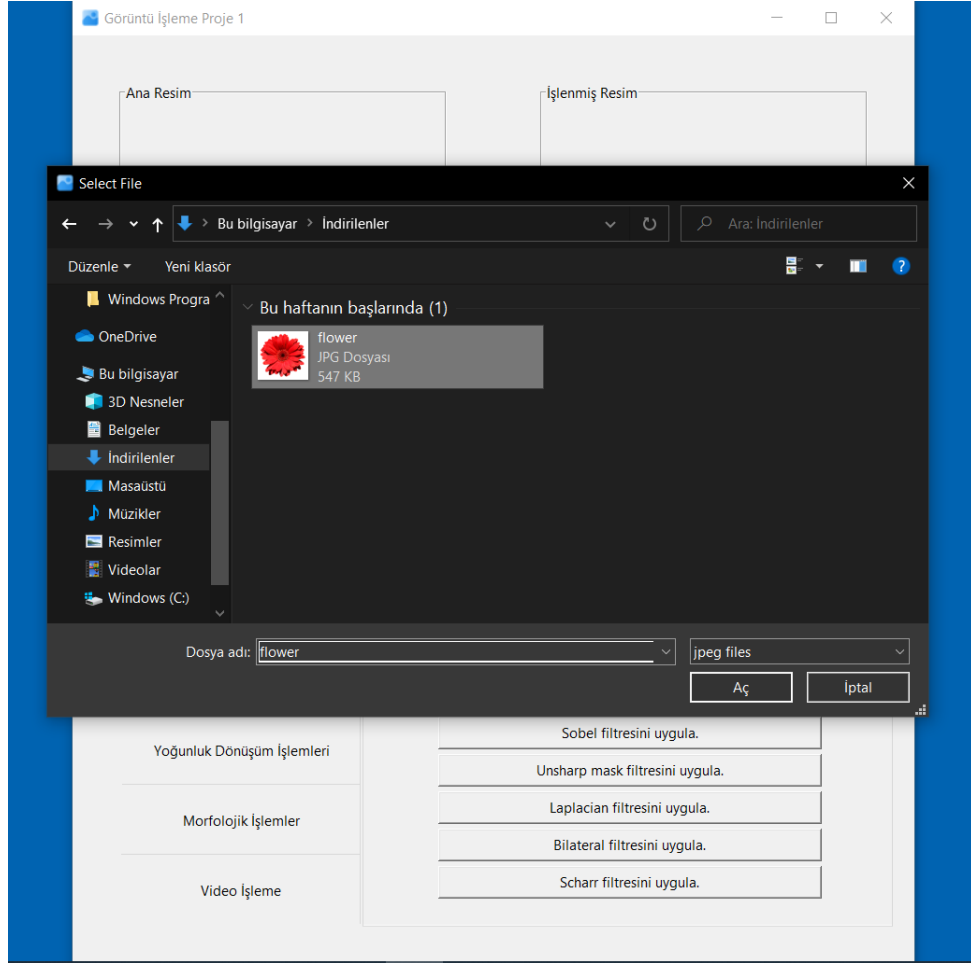
Öncelikle mainFrame olarak isimlendirilmiş ve işlenmemiş resmi içeren pencerede bir resim bulunuyorsa bunun sıfırlanması gerekmektedir. Aksi takdirde kullanıcı resim yükle butonuna basıp yeni bir resim yüklemek istediğinde bu resim eski resmin altında yer alıp diğer arayüz bileşenleriyle çakışacaktır.

Yüklenen resim global olarak tanımlanan loaded_image değişkeninde saklanmaktadır. Bu değişkenin global olarak tanımlanmasının nedeni fonksiyon sona erdikten sonra bile resmin ekranda görüntülenebilmesi içindir. Eğer böyle bir tanımlama yapılmazsa garbage collector fonksiyon bittikten sonra bu resmi gereksiz olarak görüp imha etmektedir.

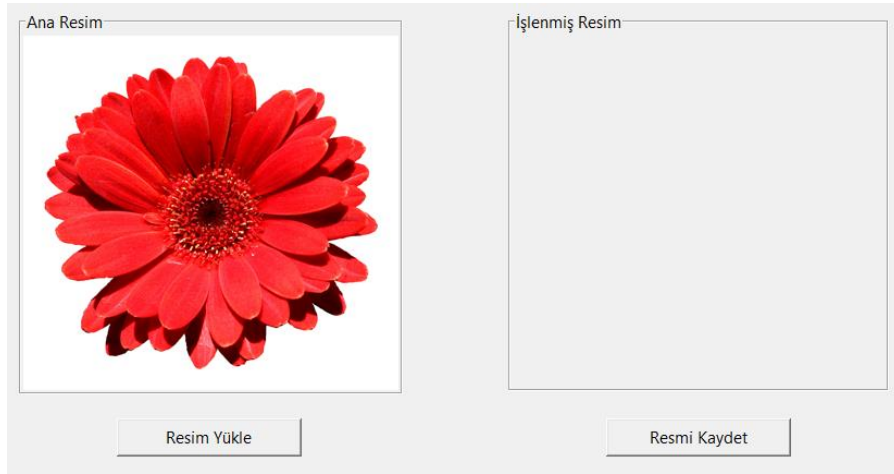
Png ve jpg en yaygın kullanılan görsel uzantıları olduğundan filedialog içerisinde yalnızca bu uzantılara sahip görsellerin tüm dosyalardan ayrıştırılıp gruplandırılarak görüntülenebilmesi için dosya tipi olarak tanımlanmışlardır.

Resmin arayüzde belirlenmiş pencere formatlarına uyabilmesi için resize işlemi uygulanıp PhotoImage objesi tipinde olması sağlanmıştır.

Aşağıda resmin pencereye yüklenmeden önce ve sonraki görüntüleri yer almaktadır.

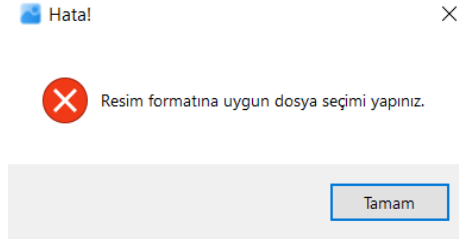


Şekil 1. Resmin seçilmesi



Şekil 2. Resmin ana resim penceresine yüklenmesi

Görsel uzantısı içermeyen dokümanlar ekranda gözükmeyeceğinden dolayı try-except bloğu içerisinde UnidentifiedImageError hatasının kontrolü yapılmıştır. Aşağıda hata yakalandığı takdirde ekranda gözükecek hata mesajı görülmektedir.



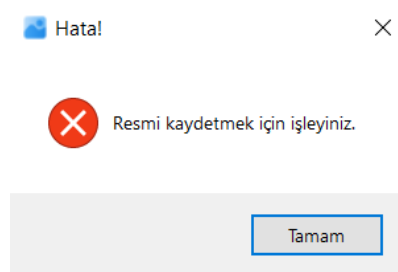
Şekil 3. Format uyumsuzluğuna dair hata mesajının gösterilmesi

1.2 Görüntü Kaydetme

```
def save_image():
    if 'processed_image' not in globals():
        showerror('Hata!', 'Resmi kaydetmek için işleyiniz.')
    else:
        rgb_im = processed_image.convert('RGB') # Png uzantılı
        dosyalar RGBA tipindedir. Jpg tipinde kaydedebilmek için alpha
        kanalını yok etmek gerekir.
        filename = filedialog.asksaveasfile(mode='w',
        defaultextension=".jpg")
        if not filename:
            return
        rgb_im.save(filename)
```

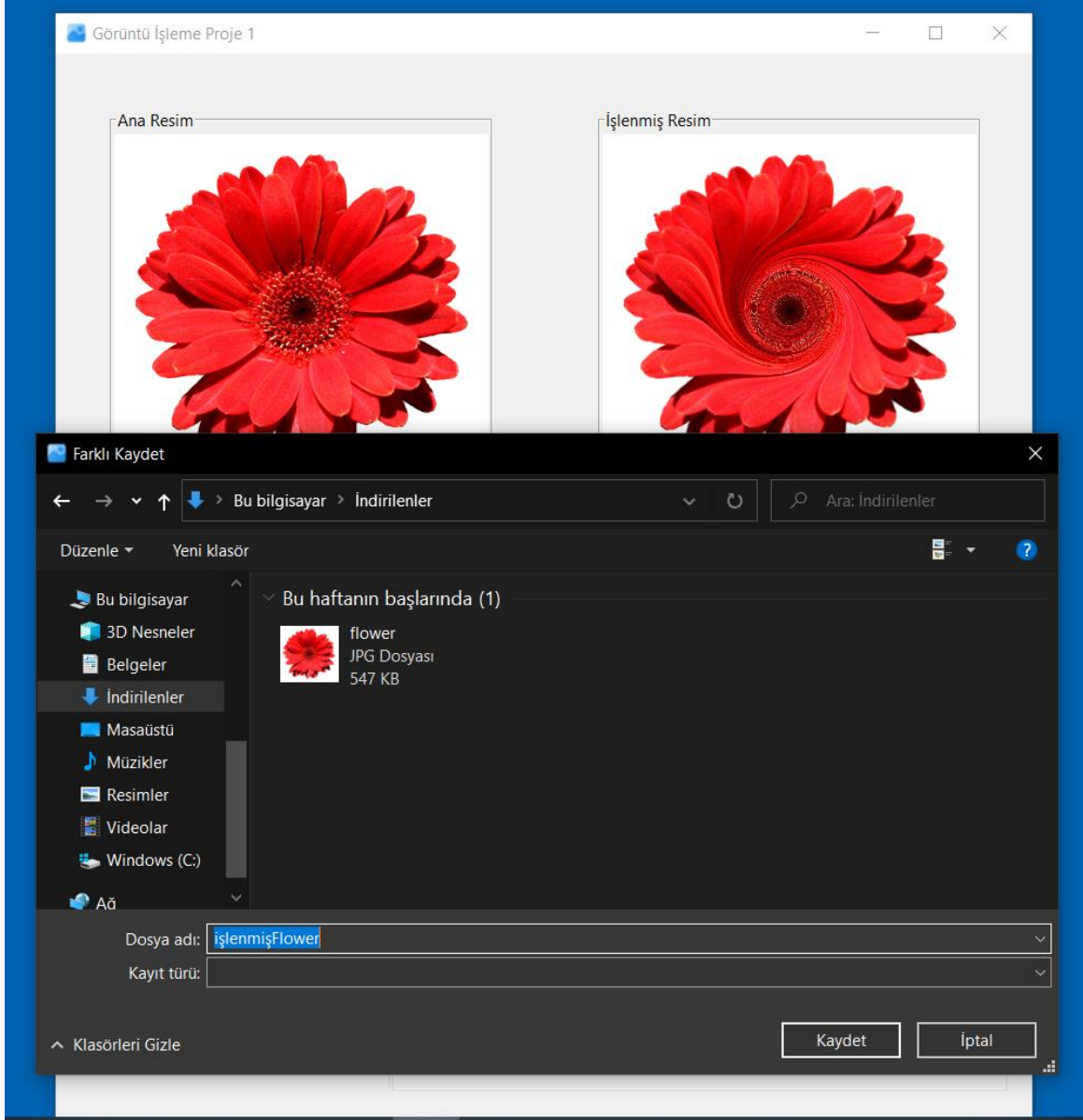
Kod 2. Görüntü kaydetme

Öncelikle resmin herhangi bir işlemle geçip geçmediğinin kontrol edilmesi gerekmektedir. Processed_image global olarak tanımlanan ve yalnızca resim işlenmişse yaratılmış bir değişkendir. Bu nedenle yaratılmış global değişkenler içerisinde yer almıyorsa resmin işlenmediğine ve bundan dolayı da kaydedilemeyeceğine dair hata mesajının gösterilmesi gerekmektedir.



Şekil 4. İşlenmemiş resmin kaydedilemeyeceğine dair hata mesajı

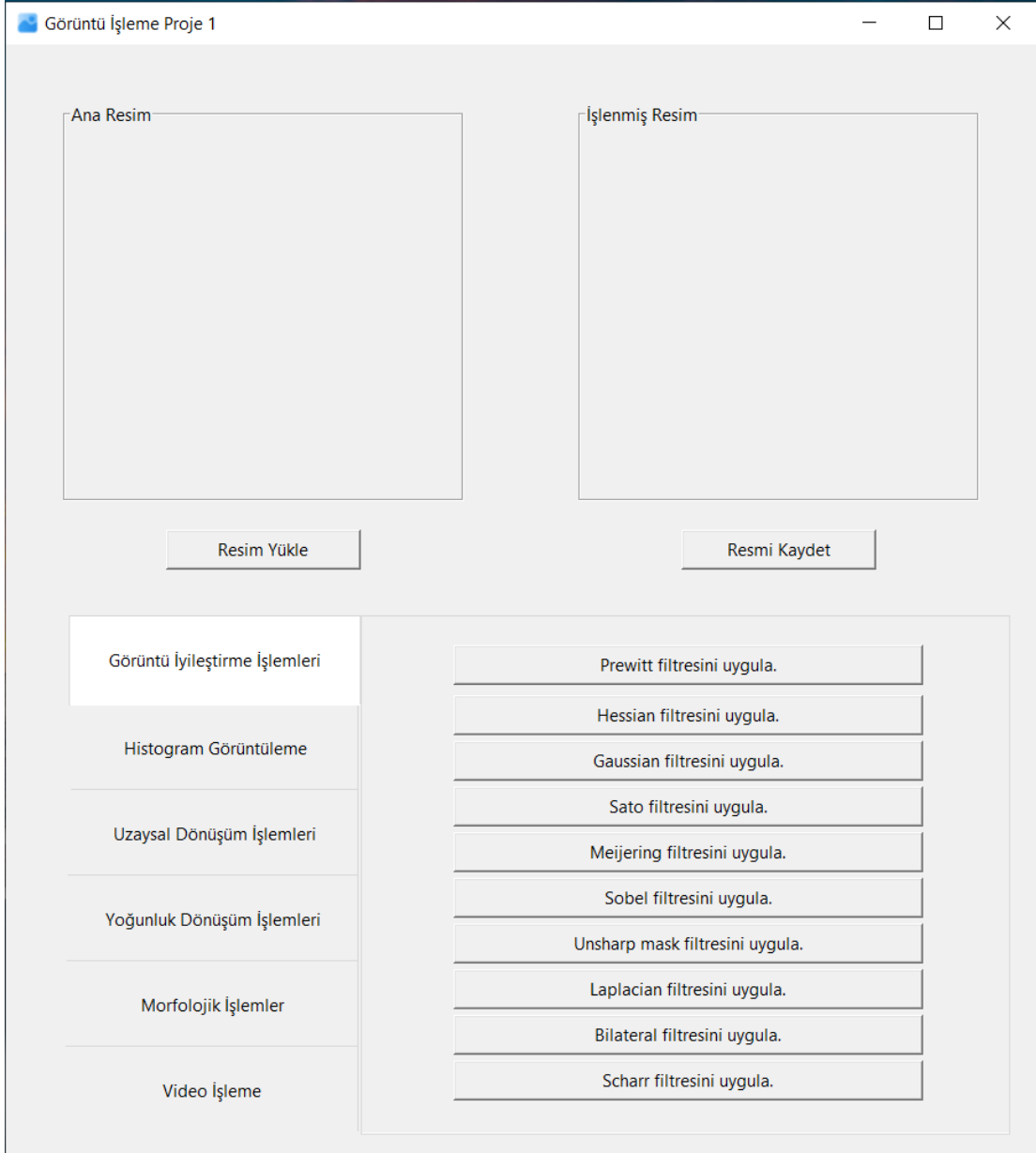
Png uzantılı dosyalar RGBA tipinde olup jpg uzantılı dosyalarda son kanal olan alpha bulunmamaktadır. Bu farklılığı ortadan kaldırmak için resimleri kaydederken yalnızca jpg tipinde kaydedilmesi tercih edildi ve png uzantılı dosyalardan alpha kanalı ortadan kaldırıldı.



Şekil 5. Resmin kaydedilmesi

2. Arayüz / Form Ortamı Hazırlama

Görüntüler eklenmeden önceki arayüz ortamı aşağıda gösterilmiştir.



Şekil 6. Arayüz ortamı

Arayüz ortamında istenen maddeler notebook olarak isimlendirilen menü halinde hazırlanmıştır. Ana resim menüler arası gezildiğinde değişmeyip yalnızca kullanıcı tekrar resim yükle butonunda yeni bir resim eklediğinde değişim gerçekleşmektedir. Tasarım hazırlanırken tkinter kütüphanesinden faydalanılmıştır.

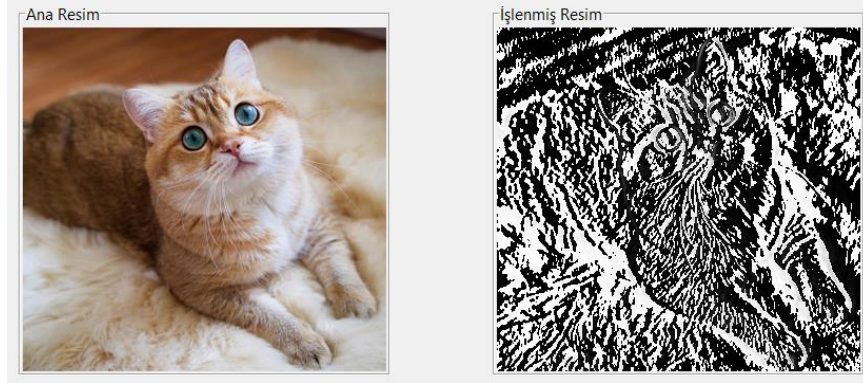
3. Görüntü İyileştirme İşlemleri, Filtreler

3.1 Prewitt Filtresi

İşlenmiş resimden de görüldüğü üzere Prewitt filtresi görüntü işlemede kenar algılama algoritmalarında kullanılır. Filtre aydınlık piksellerden karanlık olanlara doğru en büyük artışın yönüne ve bu yöndeki değişim oranına bakarak her noktadaki görüntü yoğunluğunun gradyanını hesaplar. Böylece o noktada görüntünün ne kadar pürüzsüz bir geçiş yaptığını tespit eder, böylece bu noktanın bir kenar parçası olup olmadığı hakkında bize bilgi verir.

```
processed_image_array = filters.prewitt v(np.array(img_gray))
```

Kod 3. Prewitt filtresinin uygulanması



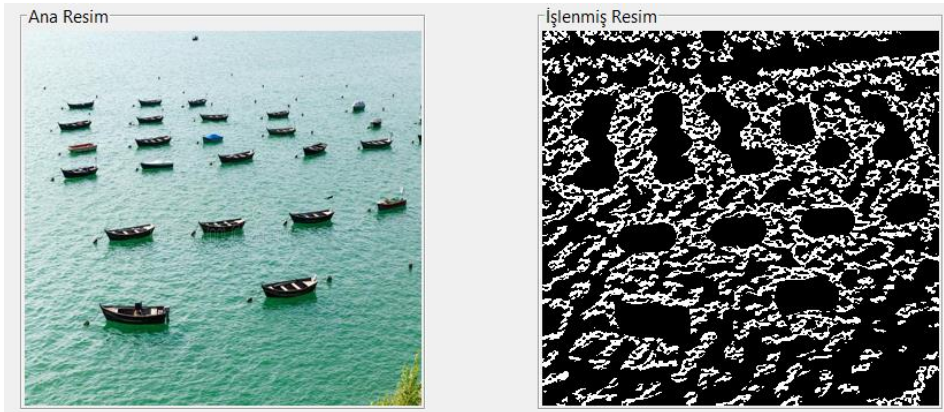
Şekil 7. Prewitt filtresinin görüntü üzerinde uygulanması

3.2 Hessian Filtresi

Hessian filtresi sürekli (continuous) kenarları algılamak için kullanılır. Gerçek hayatta gemi, nehir veya tekstil malzemesi üzerindeki kırışıklıkları algılamada yararlanır. Böylece bu tür nesnelerin tüm görüntü içerisindeki oranına ulaşarak çeşitli çıkarımlar yapılabilir.

```
processed_image_array = filters.hessian(np.array(img_gray))
```

Kod 4. Hessian filtresinin uygulanması



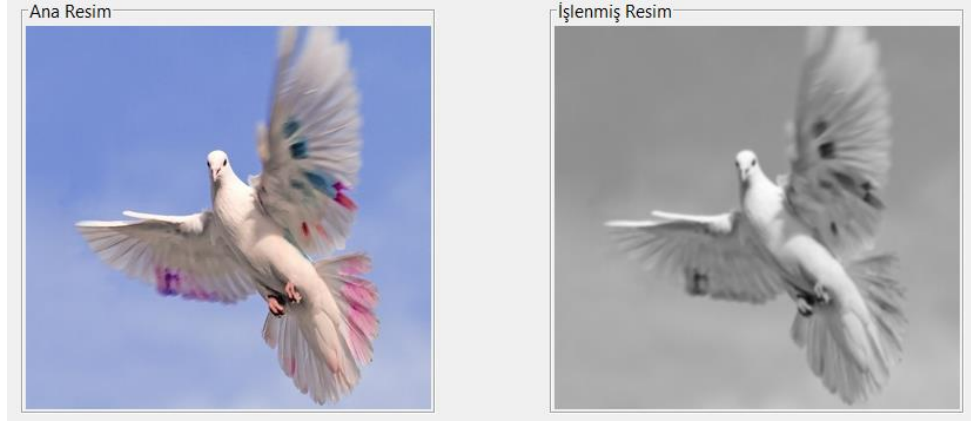
Şekil 8. Hessian filtresinin görüntü üzerinde uygulanması

3.3 Gaussian Filtresi

Gaussian filtresi görüntüyü bulanıklaştırmaya yarar. Genellikle görüntüdeki gürültüyü ve detayları azaltmak için kullanılır. Çoğu kenar algılama algoritması gürültüye duyarlı olduğundan ön işleme aşamasında görüntüye bu filtre uygulanır.

```
processed_image_array = filters.gaussian(np.array(img_gray))
```

Kod 5. Gaussian filtresinin uygulanması



Şekil 9. Gaussian filtresinin görüntü üzerinde uygulanması

3.4 Sato Filtresi

Hessian filtresi gibi sürekli (continuous) kenarları algılamak için kullanılır.

```
processed_image_array = filters.sato(np.array(img_gray))
```

Kod 6. Sato filtresinin uygulanması



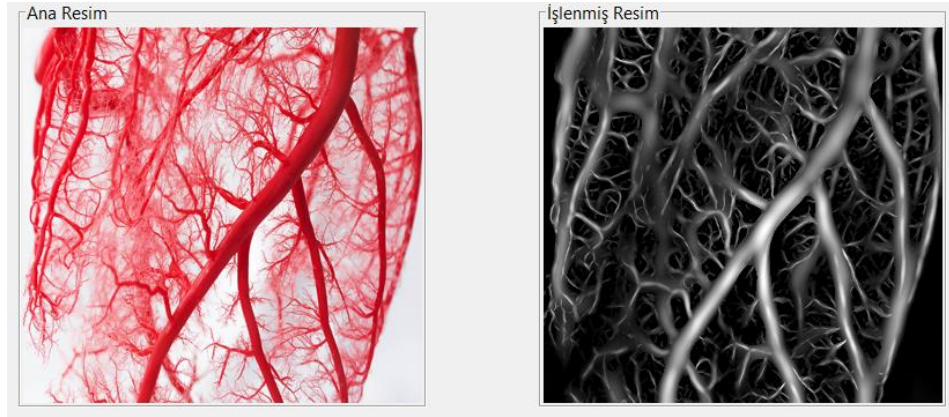
Şekil 10. Sato filtresinin görüntü üzerinde uygulanması

3.5 Meijering Filtresi

Hessian ve Sato filtreleri gibi sürekli kenar algılama filtresidir. Resimler karşılaştırıldığında işlenmiş resimdeki kan damarlarının daha net olduğu görülmektedir.

```
processed image array = filters.meijering(np.array(img_gray))
```

Kod 7. Meijering filtresinin uygulanması



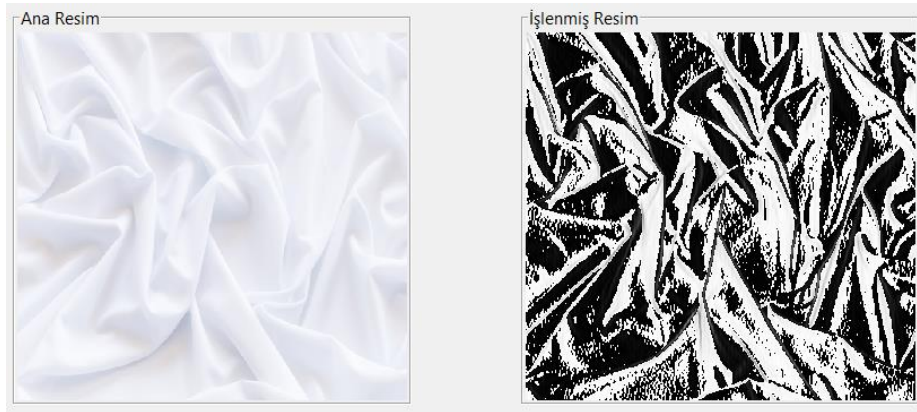
Şekil 11. Meijering filtresinin görüntü üzerinde uygulanması

3.6 Sobel Filtresi

Kenar algılama filtresidir. İşlenmiş resimde bu etkiyi kumaştaki düz yerlerin siyah, kırışıkların ise beyazla aydınlanmış olmasından gözlemleyebiliriz.

```
processed image array = ndimage.sobel(np.array(img_gray))
```

Kod 8. Sobel filtresinin uygulanması



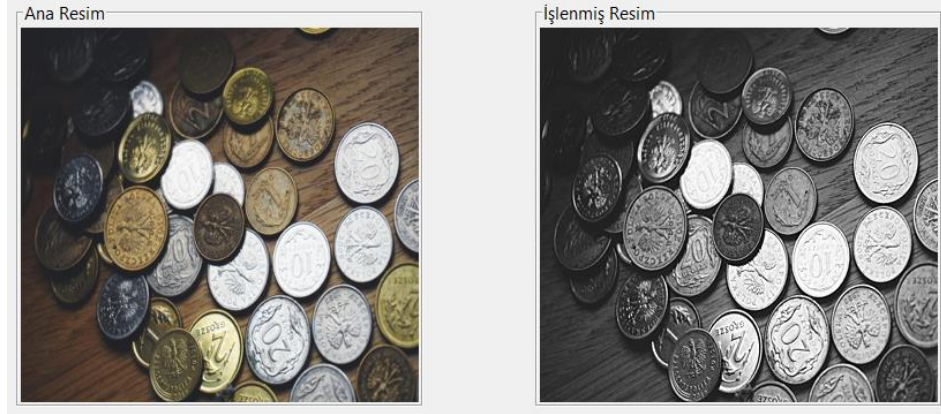
Şekil 12. Sobel filtresinin görüntü üzerinde uygulanması

3.7 Unsharp Mask Filtresi

Adı orijinal görüntünün maskesini oluşturmak için bulanık bir görüntü kullanmasından gelmektedir. Bu bulanık görüntü 3.3 maddesinde bahsedilen Gaussian filtresi ile elde edilebilir. Keskinliği azaltan bu bulanıklık maskesi daha sonra orijinal görüntüyle birleştirilir ve orijinalden daha az bulanık bir görüntü oluşturulur. Örnek resimden de görüldüğü üzere paraların üzerindeki şekil ve sayılar daha net görülmektedir.

```
processed image array = filters.unsharp_mask(np.array(img_gray))
```

Kod 9. Unsharp Mask filtresinin uygulanması



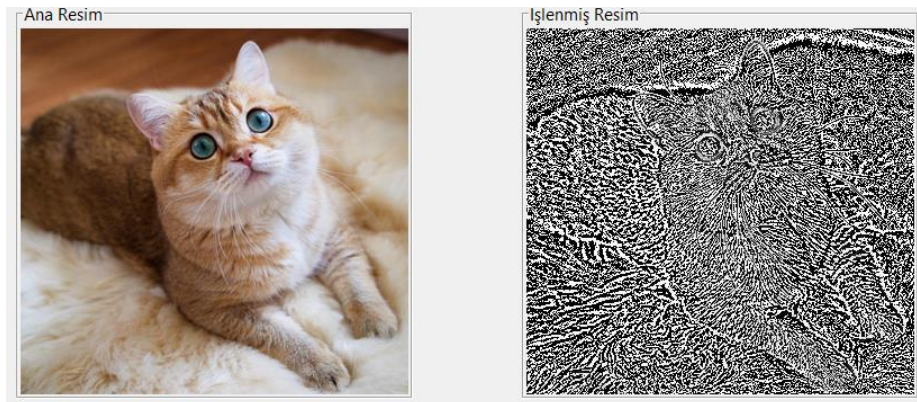
Şekil 13. Unsharp Mask filtresinin görüntü üzerinde uygulanması

3.8 Laplacian Filtresi

Laplace operatörünü kullanarak bir görüntünün kenarlarını bulmaya yarar.

```
processed image array = cv2.Laplacian(np.array(img_gray), cv2.CV_64F)
```

Kod 10. Laplacian filtresinin uygulanması



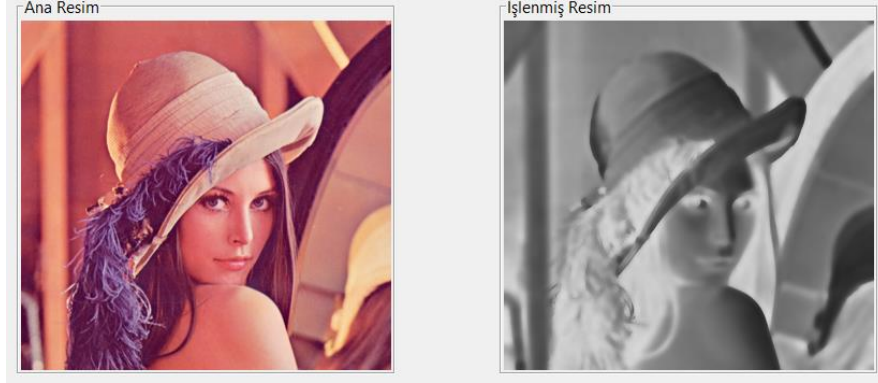
Şekil 14. Laplacian filtresinin görüntü üzerinde uygulanması

3.9 Bilateral Filtresi

Bilateral filtre, görüntüler için kenar koruyucu ve gürültü azaltıcı bir yumuşatma (smoothing) filtresidir. Her pikselin yoğunluğunu, yakınındaki piksellerden gelen ağırlıklı ortalama yoğunluk değerleriyle değiştirir.

```
processed image array = cv2.bilateralFilter(np.array(img_gray), 9, 75, 75)
```

Kod 11. Bilateral filtresinin uygulanması



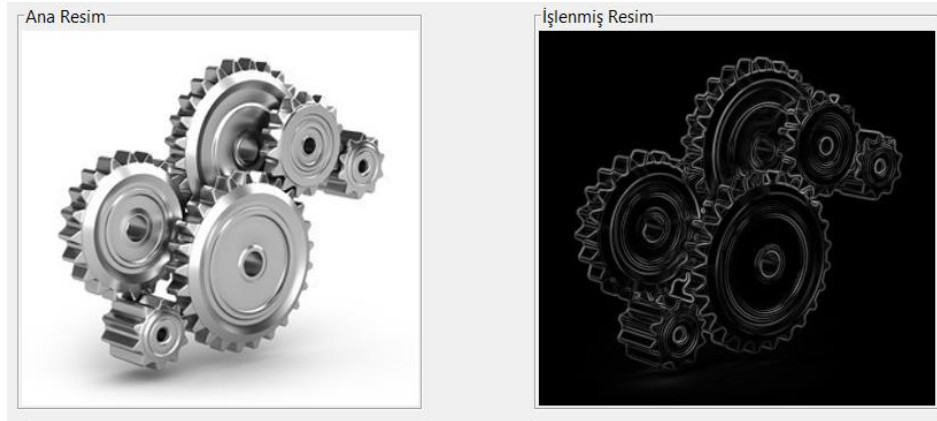
Şekil 15. Bilateral filtresinin görüntü üzerinde uygulanması

3.10 Scharr Filtresi

Scharr filtresi, 1. türevi kullanarak gradyan kenarlarını belirlemek ve vurgulamak için kullanılan bir filtreleme yöntemidir. Performans açısından Sobel filtresine benzemektedir.

```
processed image array = filters.scharr(np.array(img_gray))
```

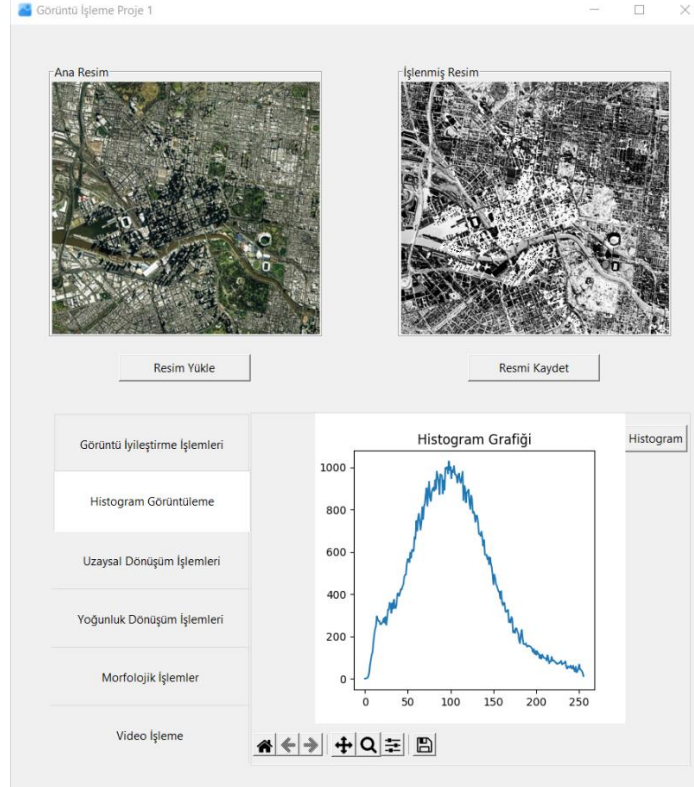
Kod 12. Scharr filtresinin uygulanması



Şekil 16. Scharr filtresinin görüntü üzerinde uygulanması

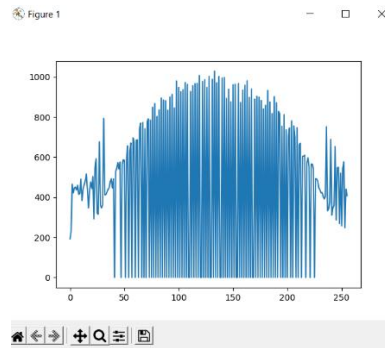
4. Histogram Görüntüleme ve Eşitleme

Histogram, sayısal bir resim içerisinde her renk değerinden kaç adet olduğunu gösteren grafiklerdir. Histogram eşitleme ile belirli bir ton etrafında toplanan histogram eğrisi (0-255) tonları arasına düzgün bir şekilde dağıtılır böylece resmin renk dağılımının homojen olarak yapılandırılması sağlanır. Şekil 17'deki menüde ana resme dair histogram grafiği görünmektedir. Bu grafikten de görüldüğü üzere renkler eşit dağılmamaktadır. Görüntü iyileştirme metotlarından biri olup kontrast eşitlemeye yardımcı olur. Çok açık veya koyu resimlerdeki ayrıntıları görmeye yardımcı olur.



Şekil 17. Histogram görüntüleme menüsü ve ana resme ait histogram grafiği

Şekil 18'de ise işlenmiş resme ait histogram grafiği görünmektedir. Renkler artık daha eşit bir dağılım göstermektedir.



Şekil 18. İşlenmiş resme ait histogram grafiği

```

def hist_equalize():
    global processed_image
    resmi_temizle()
    try:
        img_gray = ImageOps.grayscale(image)
        processed_image_array = cv2.equalizeHist(np.array(img_gray))
        hist_show()
    except NameError:
        showerror('Hata!', 'İşlem yapabilmek için resim yükleyiniz.')
    else:
        processed_image = Image.fromarray((processed_image_array *
255).astype(np.uint8))
        show_processed_image(processed_image)
        # işlenmiş görüntünün histogramı
        hist_processed = cv2.calcHist([np.array(processed_image)], [0],
None, [256], [0, 256])
        plt.plot(hist_processed)
        plt.show()

def hist_show():
    fig = Figure(figsize=(4, 4), dpi=100)
    canvas = FigureCanvasTkAgg(fig, histogramGoruntuleme)

    img_gray = ImageOps.grayscale(image)
    histr = cv2.calcHist([np.array(img_gray)], [0], None, [256], [0, 256])

    # histogram grafiginin cizdirilmesi ve GUI'ye entegre edilmesi
    hist_plot = fig.add_subplot(111)
    hist_plot.plot(histr)
    hist_plot.set_title("Histogram Grafiği")

    canvas.draw()
    canvas.get_tk_widget().pack(side=tk.LEFT)

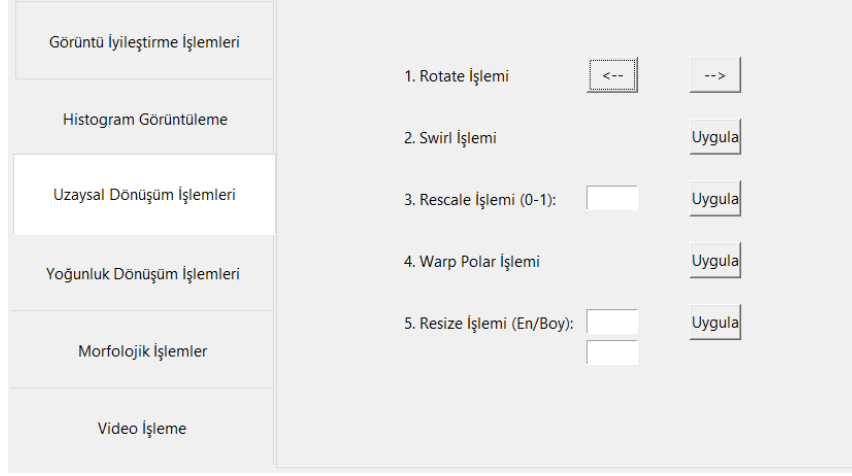
    # histogram grafigine toolbar eklenmesi
    toolbar = NavigationToolbar2Tk(canvas, histogramGoruntuleme)
    toolbar.update()
    canvas.get_tk_widget().pack(side=tk.TOP)

```

Kod 13. Histogram eşitleme işleminin gerçekleştirilmesi

5. Uzaysal Dönüşüm İşlemleri

Rotate, rescale ve resize işlemleri dışındaki dönüşümlerde parametre değerleri sabit verilmiştir. Aşağıda menüde uzaysal dönüşüm işlemlerine ayrılmış kısım gösterilmiştir.



Şekil 19. Uzaysal dönüşüm işlemleri menüsü

Kod tekrarını önlemek için her bir butona ayrı metot yazmak yerine tek bir metoda parametre olarak hangi işlemin yapıldığı string olarak aktarılmıştır. Aşağıda tüm uzaysal dönüşümleri içeren kod parçası verilmiştir. Her bir işlem yapılmadan önce eski resmin silinmesi gerektiğinden resmi_temizle metodu çağırılmaktadır.

```
def uzaysal_donusum_yap(islem_adi):
    global processed_image
    resmi_temizle()
    try:
        processed_image_array = []
        if islem_adi == "sola_dondur":
            counter.increase()
            processed_image = image.rotate(45 * counter.count)
        elif islem_adi == "saga_dondur":
            counter.decrease()
            processed_image = image.rotate(45 * counter.count)
        elif islem_adi == "swirl":
            processed_image_array = skit.swirl(np.array(image),
strength=50)
        elif islem_adi == "rescale":
            processed_image_array = skit.rescale(np.array(image),
float(textBoxolcek.get()), multichannel=True)
        elif islem_adi == "warp_polar":
            processed_image_array = skit.warp_polar(np.array(image),
output_shape=(325, 325), multichannel=True)
        else:
            processed_image_array = skit.resize(np.array(image),
(int(textBoxEn.get()), int(textBoxBoy.get())))
    except NameError:
```



```

        showerror('Hata!', 'İşlem yapabilmek için resim yükleyiniz.')
    else:
        if islem_adi != "sola_dondur" and islem_adi != "saga_dondur":
            processed_image = Image.fromarray((processed_image_array
* 255).astype(np.uint8))
            show_processed_image(processed_image)

```

Kod 14. Uzaysal dönüşüm işlemlerinin tamamını içeren kod

5.1 Rotate İşlemi

Rotate işleminde kullanıcı sol/sağ yön butonlarına basarak resmi 45 derecelik açılarla döndürebilmektedir. Butona ne kadar basıldığını tutmak için Counter sınıfı içerisinde bir count değişkeni tutulmaktadır.

```

class Counter:
    def __init__(self): # ilk başlatıldığında sayacı sıfıra ayarlar
        self.count = 0

    def increase(self):
        self.count += 1

    def decrease(self):
        self.count -= 1

counter = Counter()

```

Kod 15. Counter sınıfı

Resim 45 derecelik açıyla döndürülmesi sayaç değeri ile çarpım sonucunda elde edilmiştir. Programda PIL (Python Image Library) yardımıyla aktarılmış ana (işlenmemiş) resim image değişkeninde saklanmaktadır. Bu kütüphanede yer alan hazır rotate metodu bulunduğundan direkt kullanılmıştır.

```

if islem_adi == "sola_dondur":
    counter.increase()
    processed_image = image.rotate(45 * counter.count)
elif islem_adi == "saga_dondur":
    counter.decrease()
    processed_image = image.rotate(45 * counter.count)

```

Kod 16. Rotate işlemini gerçekleştiren kod



Şekil 20. Sola doğru 45 derece açıyla döndürülmüş işlenmiş resim

5. 2 Swirl İşlemi

Swirl, büküm veya spiral şeklinde resmin işlenmesini ifade etmektedir.

```
processed_image_array = skit.swirl(np.array(image), strength=50)
```

Kod 17. Swirl işleminin gerçekleştirilmesi



Şekil 21. Swirl işlemi uygulanmış resim

5.3 Rescale İşlemi

Rescale ölçeklendirme anlamına gelmektedir. Ölçek 1 olduğunda işlenmiş resim ile ana resim aynı olmaktadır. Resmin boyutunu azaltmak için ölçek değeri 1'den küçük, artırmak için ise 1'den büyük olmalıdır.

```
processed_image_array = skit.rescale(np.array(image),  
float(textBoxolcek.get()), multichannel=True)
```

Kod 18. Rescale işleminin gerçekleştirilmesi



Şekil 22. Resmin 0.5 oranında küçültülmesi

5.4 Warp Polar İşlemi

Görüntüyü polar veya log-polar koordinat düzlemine göre yeniden eşler.

```
processed_image_array = skit.warp_polar(np.array(image),  
output_shape=(325, 325), multichannel=True)
```

Kod 19. Warp polar işleminin gerçekleştirilmesi



Şekil 23. Warp polar işlemi uygulanmış resim

5.5 Resize İşlemi

Reshape işlemine benzer olup resmin boyutunun ayarlanmasıyla ilgilidir. Reshape işleminde ölçek verilip en/boy oranı korunurken resize işleminde en ve boy parametrelerini kullanıcı girmektedir.

```
processed_image_array = skit.resize(np.array(image),  
(int(textBoxEn.get()), int(textBoxBoy.get())))
```

Kod 20. Resize işleminin gerçekleştirilmesi



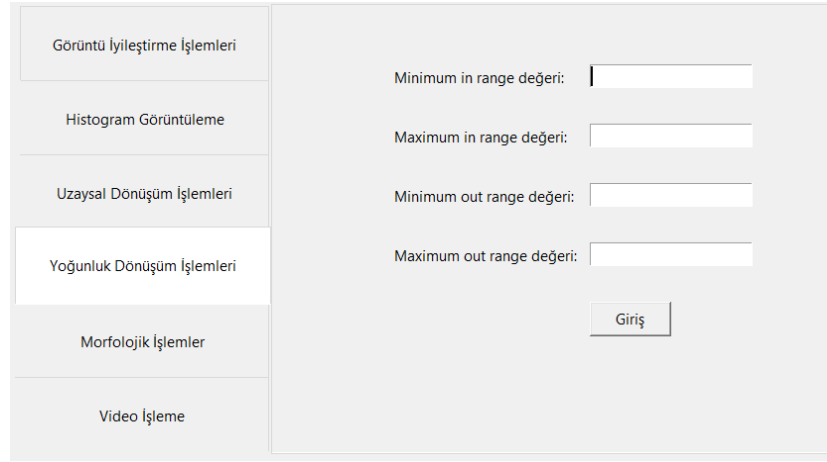
Şekil 24. En/Boy değerlerinin 300/150 olarak girilmesiyle oluşan yeni resim

6. Yoğunluk Dönüşüm İşlemleri

Bir görüntü iki boyutlu bir fonksiyon olarak tanımlanır. Herhangi bir koordinat noktasındaki f 'nin genliği (amplitude) o noktadaki görüntünün yoğunluğu olarak adlandırılır.

Örneğin 8 bitlik bir gri tonlamalı görüntüde 256 adet gri seviyesi vardır. Bu da görüntüdeki herhangi bir pikselin 0 ile 255 arasında bir yoğunluk değerine sahip olduğunu belirtir.

Yoğunluk dönüşüm işlemi iki parametre üzerinden bu genlik değerini değiştirmeyi amaçlamaktadır. Bu parametreler in range ve out range olup her biri minimum, maksimum değerleriyle tuple veri yapısını oluşturmaktadır. Aşağıdaki menüde bu değerler için giriş yerleri görünmektedir.

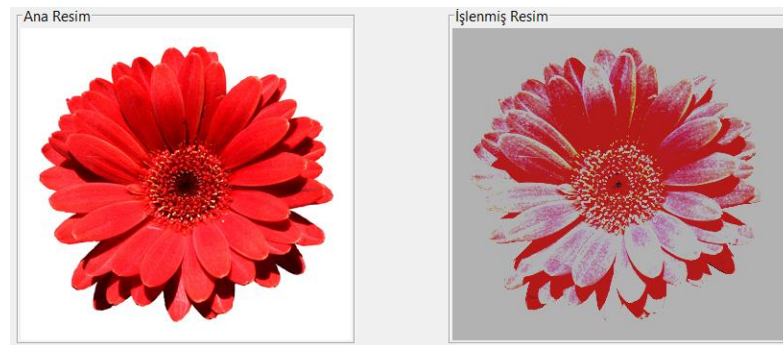


Şekil 25. Yoğunluk dönüşüm işlemleri menüsü

In range değeri: (10, 50)

Out range değeri: (0.1, 0.7)

olarak belirlenip işlenmiş resim aşağıda gösterilmiştir.



Şekil 26. Yoğunluk dönüşüm işlemi gerçekleştirilmiş resim

```

def yogunluk_donustur():
    resmi_temizle()

    global processed_image
    minInRange = textboxInRange1.get()
    maxInRange = textboxInRange2.get()
    minOutRange = textboxOutRange1.get()
    maxOutRange = textboxOutRange2.get()

    try:
        processed_image_array = skie.rescale_intensity(image,
in_range=(float(minInRange), float(maxInRange)),
out_range=(float(minOutRange), float(maxOutRange)))
    except NameError:
        showerror('Hata!', 'İşlem yapabilmek için resim yükleyiniz.')
    except ValueError:
        showerror('Hata', 'Resmin işlenebilmesi için parametre
değerlerini giriniz.')
    else:
        processed_image = Image.fromarray((processed_image_array *
255).astype(np.uint8))
        show_processed_image(processed_image)

```

Kod 21. Yoğunluk dönüştürme işleminin gerçekleştirilmesi

7. Morfolojik İşlemler

7.1 Dilation İşlemi

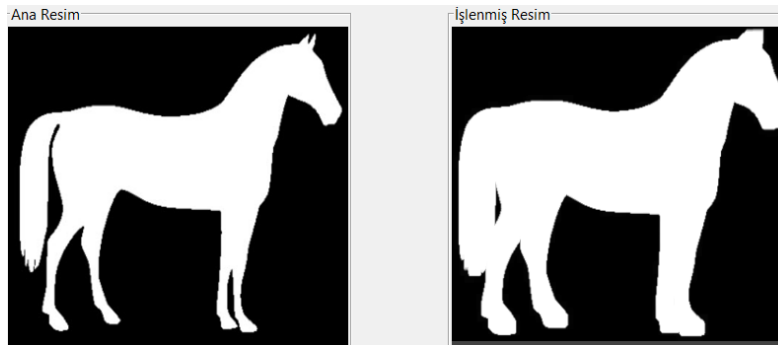
Dilation işlemi, bir görüntüdeki nesnelerin sınırlarına pikseller ekler ve bunu görüntüyü işlerken kullanılan yapılandırma ögesinin boyutuna ve şekline bağlıdır. Operatörün binary görüntü üzerindeki temel etkisi, ön plan piksellerinin (yani tipik olarak beyaz piksellerin) bölgelerinin sınırlarını kademeli olarak büyütme. Böylece ön plan piksel alanlarının boyutu büyürken bu bölgelerdeki delikler küçülür (doldurma işlemi yapılması gibi)

```

kernel = np.ones((20,20), np.uint8)
if islem_adi == "Dilation":
    processed_image_array = dilation(np.array(img_gray), kernel)

```

Kod 22. Dilation İşlemi



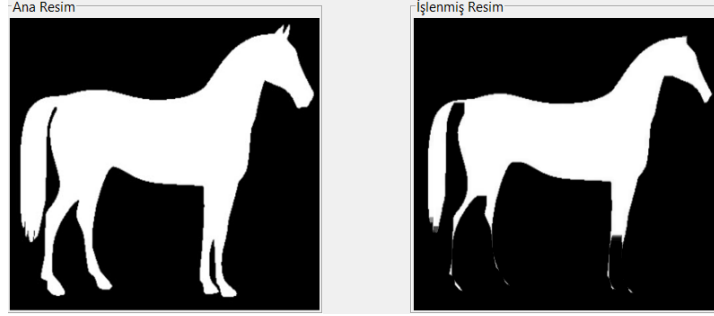
Şekil 27. Dilation işlemi gerçekleştirilmiş resim

7.2 Erosion İşlemi

Erosion işlemi temelde Dilation işleminin tam tersi işlemi yapar, bir görüntüdeki nesnelerin sınırlarındaki pikselleri iterasyon sayısına göre silme işlemi yapar. Kodda for döngüsü içinde 10 kez erosion fonksiyonunun çağırılma nedeni 10 iterasyonlu erosion'u göstermek ve görüntüdeki farkları daha net görmek içindir.

```
kernel = np.ones((20,20), np.uint8)
elif islem_adi == "Erosion":
    for i in range(10):
        processed_image_array = erosion(np.array(img_gray), kernel)
```

Kod 23. Erosion İşlemi



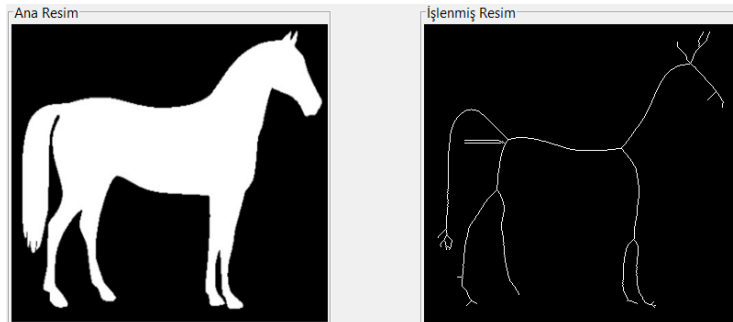
Şekil 28. Erosion işlemi gerçekleştirilmiş resim

7.3 Skeletonize İşlemi

Skeletonize işlemi, ya da diğer adıyla medial axis, çeşitli morfolojik işlemler kullanılarak bir görüntüdeki nesnelerin yapısal durumunun çıkarılmasına yarayan bir yöntemdir. Bizim uygulamamızda skimage kütüphanesinin morphology modülündeki medial_axis fonksiyonunu kullanmamızın yanında, farklı elde edilmiş yöntemleri ve yolları vardır. Üç boyutlu cisimler üzerinde dahi uygulamaları bulunmaktadır, dolayısıyla modelleme işlemlerinde kilit rol oynama kapasitesine ait bir morfolojik operasyondur.

```
elif islem_adi == "Skeletonize":
    img = img_as_bool(color.rgb2gray(np.asarray(image)))
    out = morphology.medial_axis(img)
    processed_image_array = np.array(out)*255
```

Kod 24. Skeletonize İşlemi



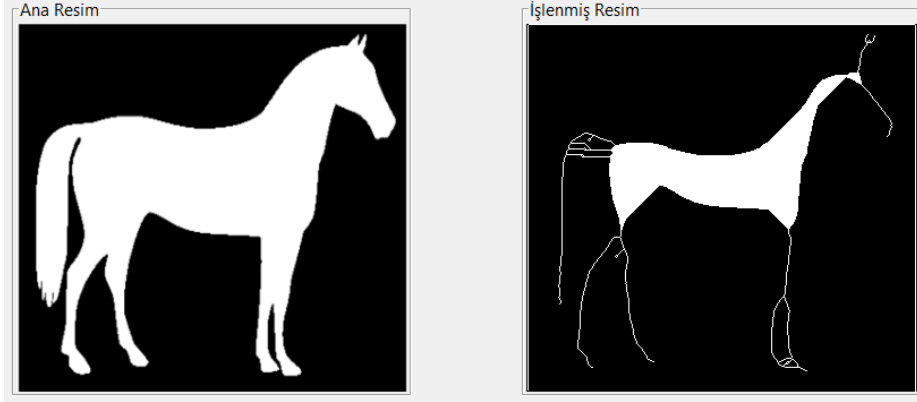
Şekil 29. Skeletonize işlemi gerçekleştirilmiş resim

7.4 Thinning İşlemi

Thinning, bir resmindeki nesnelerin inceltilmesi anlamına gelir, adı üzerindedir. Bunu sağlamak açısından diğer morfolojik operasyonlardan olan erosion ve opening gibi operasyonları arka arkaya nizami bir şekilde kullanarak tüm nesne kenarlarındaki piksellerden kurtulunup inceltme sağlanmış olur.

```
elif islem_adi == "Thinning":  
    processed_image_array = morphology.thin(img_gray, 25)*255
```

Kod 25. Thinning İşlemi



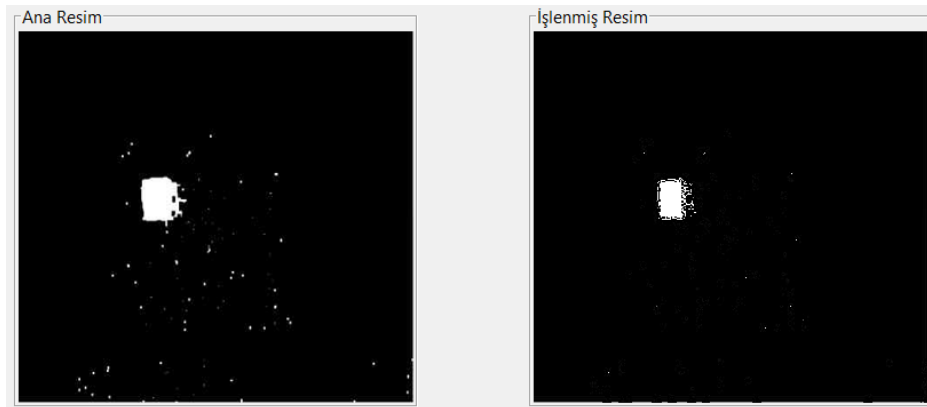
Şekil 30. Thinning işlemi gerçekleştirilmiş resim

7.5 Küçük Objeleri Kaldırma İşlemi

Görüntüyü bir takım ağır operasyonlardan geçirerek bağlantılılık (connectivity) oranına göre belli büyüklüklerdeki küçük objeleri görüntüden kaldırmış olur, ancak elbette büyük nesnelerde de ufak değişiklikler görmek mümkündür (görüntüye uygulanan manipülatif işlemlerden dolayı)

```
elif islem_adi == "RSO":  
    processed_image_array =  
    morphology.remove_small_objects(np.array(img_gray), 50)
```

Kod 26. RSO İşlemi



Şekil 31. RSO işlemi gerçekleştirilmiş resim

7.6 Top-Hat İşlemi

Top-Hat ve Black-Hat işlemleri, verilen görüntüden küçük ayrıntıları ve öğeleri çıkarmak için kullanılan yöntemlerdir. Bu iki tür dönüşümde, Top-Hat dönüşümü, giriş (input) görüntüsünün, bazı yapılandırma öğeleri tarafından opening ile farkının çıkarılmasını sağlarken, Black-Hat dönüşümü closing ile input görüntünün arasındaki farkı temel alır. Dolayısıyla, Bu iki işlem opening ve closing işlemleri ile doğrudan bağlantılıdır.

```
elif islem_adi == "TopHat":  
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))  
    processed_image_array = cv2.morphologyEx(np.array(img_gray),  
cv2.MORPH_TOPHAT, kernel)
```

Kod 27. Top-Hat İşlemi



Şekil 32. Top-Hat işlemi gerçekleştirilmiş resim

Yukarıdaki resimde de görebileceğiniz gibi, Top-Hat işlemi kullanılarak koyu bir arka plan üzerinde açık pikseller iyileştirilmiş ve çıkarılmıştır. Böylece girdideki küçük ayrıntıları gözlemlemek kolaylaşmıştır.

7.7 Opening İşlemi

Opening, önce aşındırma (erosion) ve ardından dilation işleminin yapıldığı bir işlemler dizisidir. Elde edilen görüntünün ince çıkıntılarını ortadan kaldırır. Elde edilen görüntünün iç gürültüsünü (internal noise) gidermek için kullanılır.

```
elif islem_adi == "Opening":  
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (10,10))  
    processed_image_array = cv2.morphologyEx(np.array(img_gray),  
cv2.MORPH_OPEN, kernel)
```

Kod 28. Opening İşlemi



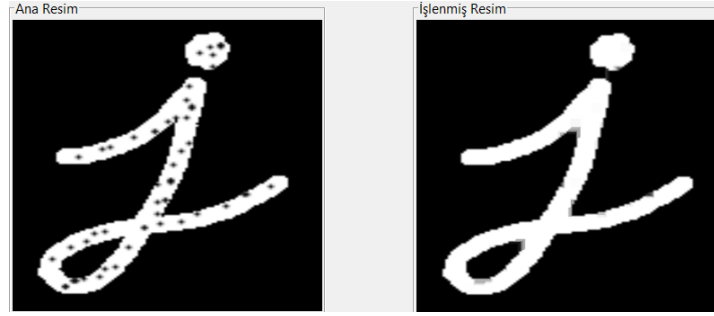
Şekil 33. Opening işlemi gerçekleştirilmiş resim

7.8 Closing İşlemi

Closing, önce dilation ve ardından aşınma (erosion) işleminin yapıldığı bir işlemler dizisidir. Elde edilen görüntüdeki küçük delikleri ortadan kaldırır. Contour'un yumuşatılması ve dar kırılmaların kaynaştırılması için kullanılır.

```
elif islem_adi == "Closing":  
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (10,10))  
    processed_image_array = cv2.morphologyEx(np.array(img_gray),  
cv2.MORPH_CLOSE, kernel)
```

Kod 29. Closing İşlemi



Şekil 34. Closing işlemi gerçekleştirilmiş resim

7.9 Morphological Gradient İşlemi

Morphological gradient, bir görüntünün genişletilmesi ve erosion'a tabii tutulması arasındaki farka eşit olan işlemdir. Ortaya çıkan görüntüdeki her piksel değeri, yakındaki piksellerin kontrast yoğunluğunu gösterir. Bu, kenar algılamada, görüntü segmentasyonunda ve bir nesnenin ana hatlarını bulmak için kullanılan oldukça etkili bir yöntemdir.

```
elif islem_adi == "MorphGrad":  
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))  
    processed_image_array = cv2.morphologyEx(np.array(img_gray),  
cv2.MORPH_GRADIENT, kernel)
```

Kod 30. Morphological gradient işlemi



Şekil 35. Morphological gradient işlemi gerçekleştirilmiş resim

7.10 Black-Hat İşlemi

Top-Hat işlemi başlığı altında 2 benzer morfolojik işlem açıklanmıştır (7.6). Bu bölümde Black-Hat için kod parçası ve görüntü örneği yer alacaktır. Beyaz arka plana karşı koyu renkli nesneleri vurgulamak için kullanılır. Aşağıdaki resimde, "TO EAST" harfleri beyaz arka plana karşı siyahtır ve bu nedenle harfler çıktıda vurgulanmıştır.

```
elif islem_adi == "BlackHat":  
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (20,20))  
    processed_image_array = cv2.morphologyEx(np.array(img_gray),  
cv2.MORPH_BLACKHAT, kernel)
```

Kod 31. Black-Hat işlemi



Şekil 36. Black-Hat işlemi gerçekleştirilmiş resim

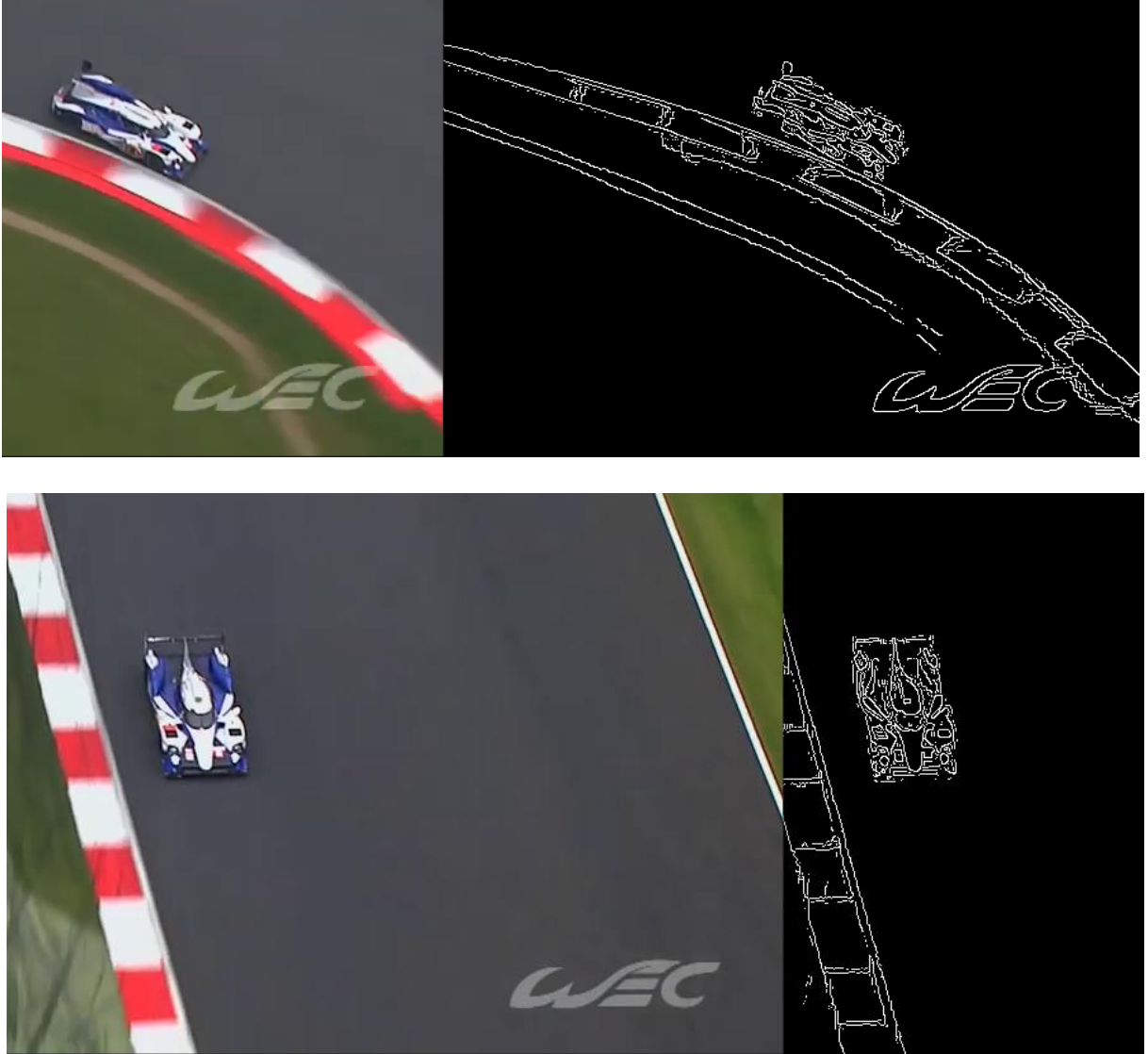
8. Video İşleme

Video işleme için Canny kenar algılama algoritması kullanıldı. Canny kenar algılama, gürültüyü önlerken görüntüdeki kenarları algılamak için kullanılan bir görüntü işleme yöntemidir. 7 adımdan oluşmaktadır. [1]

1. Renkli resmin gri tonlamalı resme dönüştürülmesi
2. Gaussian blur efektinin uygulanması (Bulanıklaştırma, görüntüyü daha fazla işlemekten önce gürültünün bir kısmını ortadan kaldırır.)
3. Yoğunluk değişimlerinin belirlenmesi
4. Non Max Suppression: Üretilen görüntü büyüklüğü kalın kenarların oluşmasına yol açar. İdeal olarak, son görüntünün ince kenarları olmalıdır. Bu nedenle, kenarları inceltmek için Non-max suppression uygulanmalıdır.
5. Double Thresholding: Non-Max suppression sonucunun mükemmel olmadığını, elde edilen bazı kenarların aslında kenar olmayabileceğini ve görüntüde biraz parazit fark ettiğimiz aşamada devreye double-thresholding işlemi girmektedir. Yüksek ve düşük eşik değerlerine göre, kenar olmaya aday bölgeleri sınıflandırır.

6. Bu aşamada, güçlü ve zayıf olarak sınıflandırdığımız kenar adaylarını belirdikten sonra, zayıf olarak sınıflandırılan kenarlardan gerçekten kenar olanlarını ayıklama kısmına girmiş oluyoruz. Bu aşamanın olmaması durumunda, çözünürlüğü yüksek görüntüdeki ufak kenarları kenar olarak sınıflandırmayıp kaybetmiş olurduk.
7. Son olarak zayıf kenar adaylarının tam olarak belirlenmesinden sonra bunları görüntüden tamamen çıkarıyoruz (0'a eşitleyip siyah yapıyoruz) böylelikle elimizde sadece kenarlardan oluşan bir görüntü kalıyor.

Program arayüzünün Video İşleme kısmından modül çalıştırılabilir, örnek video da programın içine gömülü şekilde eklenmiştir.



Şekil 37. Video işleme örnek görüntüler

```

def video_processing():
    # Kamerayı etkinleştirir
    cap = cv2.VideoCapture('toyota.mp4')
    while 1:
        ret, frame = cap.read()
        if ret:
            cv2.imshow('Video', frame)

            edges = cv2.Canny(frame, 100, 100)
            cv2.imshow('Edges', edges)

            # En az 1 milisaniye beklendiyse ve esc tuşuna basılırsa
            video işleme sonlanır
            if cv2.waitKey(1) & 0xFF == 27:
                break
    cap.release()
    cv2.destroyAllWindows()

```

Kod 32. Videonun Canny algoritması ile işlenmesi

9. Özdeğerlendirme Tablosu ve İş Bölümü

	İstenen Özellik	Puan	Var	Açıklama	Tahmini Puan
1	Görüntü Yükleme ve Kaydetme (Load/Read, Save)	10	✓	Görüntü yükleme ve kaydetme fonksiyonları belirli dosya formatlarını (png, jpg, jpeg) kabul edecek şekilde ayarlandı.	10
2	Arayüz / Form Ortamı Hazırlama	10	✓	Tkinter kütüphanesi kullanılarak arayüz ortamı hazırlandı.	10
3	Görüntü İyileştirme İşlemleri, Filtreler (10 farklı filtre içermeli)	10	✓	Farklı amaçlara hizmet eden kullanışlı 10 adet filtre programa eklendi.	10
4	Histogram Görüntüleme ve Eşikleme	10	✓	Histogram grafikleri ve equalization işlemi programa eklendi.	10
5	Uzaysal Dönüşüm İşlemleri (Resizing, Rotation, Cropping, Swirl ... gibi 5 farklı dönüşüm işlemi içermeli)	10	✓	5 farklı uzaysal dönüşüm işlemi kullanıcıya kolaylık sağlayacak şekilde eklendi.	10
6	Yoğunluk Dönüşümü İşlemleri (Değerleri kullanıcı verebilmeli)	10	✓	Kullanıcıdan girdi alınarak yoğunluk dönüşümü yapılması sağlandı.	10
7	Morfolojik İşlemler (10 farklı morfolojik işlem içermeli)	10	✓	Basit ve karmaşık düzeylerde 10 farklı morfolojik işlem programa eklendi.	10

8	Video İşleme (Videoda kenar belirleme gibi bir örnek yeterli) Herhangi bir ortamda yapabilirsiniz.	10	✓	Canny algoritması kullanarak video üzerinde kenar tespiti yapılması sağlandı, örnek video ile test edildi.	10
9	Rapor Biçimi, Düzeni, Özdeğerlendirme Raporu ve İş Bölümü Yukarıdaki 8 madde raporda ayrı başlıklar halinde verilerek açıklanmalı. Raporda her bir maddedeki her bir örnek için, işlemden önceki ve sonraki ekran görüntüleri olmalı. İlgili kod parçaları da eklenmeli.	20	✓	Rapor tüm kurallara uygun şekilde, örnek görüntü ve kod parçalarını içerecek şekilde hazırlandı. Ayrıca programın içinde de istenilen modül kolayca bulunabilir.	20
Toplam					100

Öz değerlendirme tablosunda projenin bölümleri ile ilgili gerekli tüm detaylar yer almaktadır. İş bölümü detayı olarak;

Projenin geliştirilmesi araştırılması için 4 saat (uygun GUI kütüphanesi, farklı görüntü işleme kütüphanelerinin araştırılması), geliştirme aşaması için 12 saat, test aşaması için ise 3 saat harcandı.

Proje süresince grup üyeleri olarak github platformu üzerinden projenin farklı modülleri üzerinde sürekli geliştirmeler yapıp bu geliştirmeleri kolaylıkla birbirimizle paylaşabildik.

Emel; programın arayüzünün geliştirilmesi, uzaysal dönüşüm işlemleri, yoğunluk dönüşümü ve video işleme alanına yoğunlaşmıştır. Anıl; filtrelerin belirlenmesi ve gerçekleştirimi, morfolojik işlemler, histogram equalization işlemlerine yoğunlaşmıştır. Yoğunlaştığımız alanlar dışında birbirimizin alanlarıyla ilgili sürekli bilgi alışverişi gerçekleştirdiğimiz bir proje oldu.

10. Kaynakça

1. Canny Edge Detection, <https://justin-liang.com/tutorials/canny/#gradient>
2. Top-hat and black-hat operations, <https://www.quora.com/Why-use-the-top-hat-and-black-hat-morphological-operations-in-image-processing>
3. Scikit-image, <https://scikit-image.org/>
4. Opencv morphological operations, https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html