Melih Sami Özaydın
502131003

09.11.2021

BM 5113 Bilgisayarla Görme

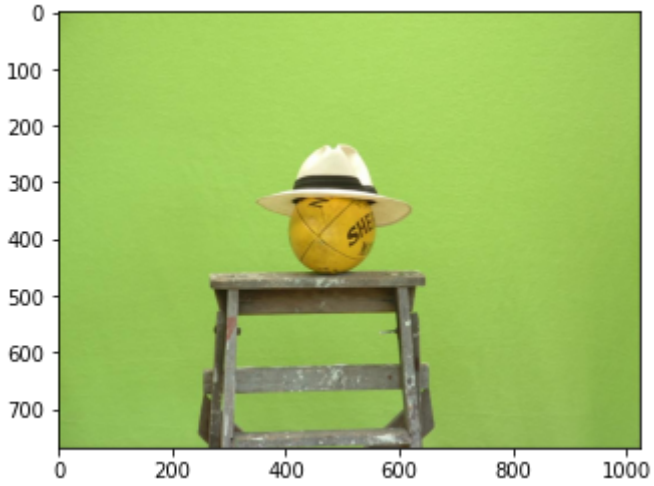
Not: İlgili kodlar ve sonuçlar https://github.com/melihozaydin/MSc_Projects linkinde erişilerbilirdir.

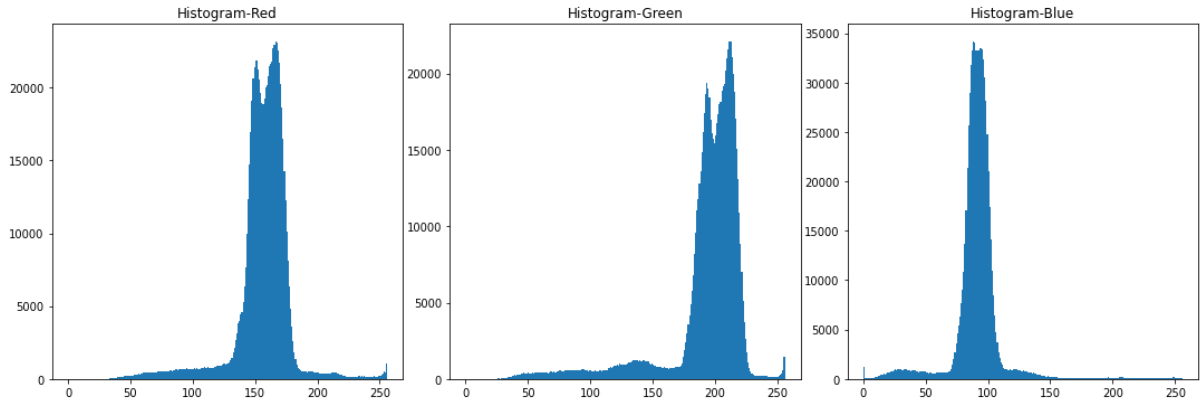
Ödev 1:

"images/greenscreen.jpg" dosyasında yer alan renkli görüntünün 3 ayrı kanalı için histogramını hesaplayıp analiz ediniz. Görüntünün içerisinde baskın olan renk için RGB kanallarında hangi yoğunluk değerinin oluştuğunu tespit ediniz. Bu yoğunluk değerlerini RGB kanallarında istediğiniz bir renk ile güncelleyerek görüntüdeki baskın rengin değişmesini sağlayınız.

Sonuçlar ve Yorum:

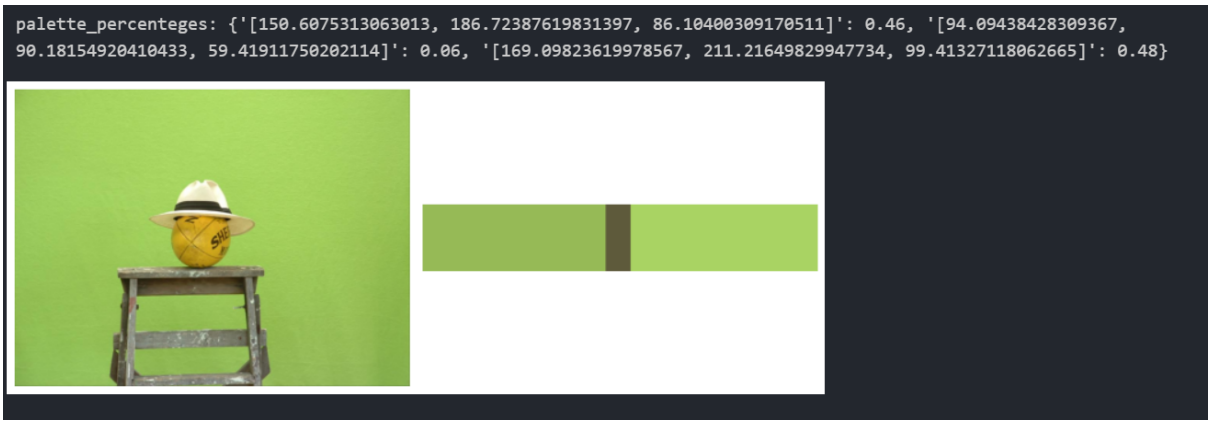
- Görüntünün BGR formatında okunup, bunun RGB formatına çevrilmesi ve kanal histogramlarının görüntülenmesi için gerekli fonksiyonlar yazıldı.



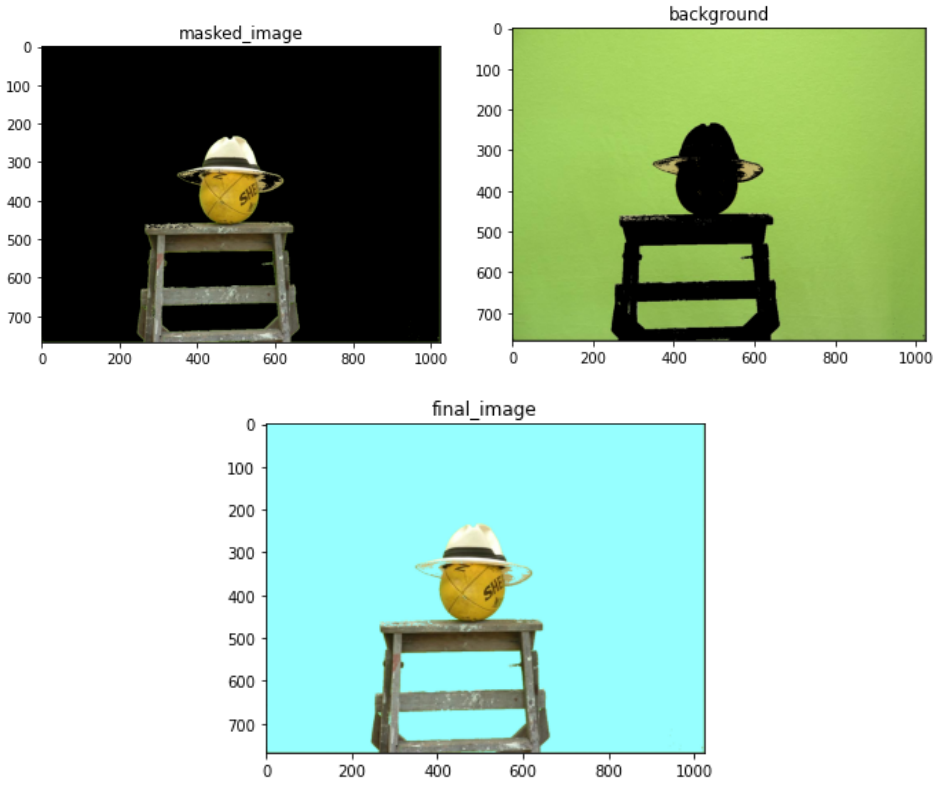


- Görüntünün içerisinde baskın olan renk için RGB kanallarında oluşan yoğunluk değerlerinin belirlenmesi için 2 alternatif metod kullanıldı.
    - Ilk olarak basitçe görsel içerisinde en çok görülen RGB renk kombinasyonunu çıkaran bir çözüm üretildi.
    - 2. Olarak, görsel içerisindeki renklesi KMeans algoritması ile K grupa gruplayarak bu grupların resimdeki baskınlık yüzdelerini veren bir fonksiyon kullanıldı.

Bu algoritmalardan 2. Çözümün daha kullanışlı olduğu düşünüldüğünden bunun sonuçları paylaşıldı.



- Yoğunluk değerlerini RGB kanallarında istenilen bir renk ile güncellemek için baskın renklerin değerlerine bakarak her renk kanalı için alt ve üst eşik seçilerek arkaplan ve önplan birbirinden ayrıldı.



Program Kaynak Kodları:

```
def show_hist(img, channel_order="RGB", title="Histogram", ret=False,
                      bins=256, cumulative=False):
    # plot histograms for all 3 channels
    #print("len(img.shape):",len(img.shape))
    # https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

    # GRAYSCALE
    if len(img.shape) < 3:
        img = np.expand_dims(img, axis=2)
        color_names = ["Gray"]
    else:
        # RGB
        color_names = ["Red", "Green","Blue"]
        if channel_order == "BGR":
            # BGR
            color_names = reversed(color_names)

    if not cumulative:
        fig, axes = plt.subplots(1, img.shape[2], figsize=(15,5))
        if img.shape[2] == 1:
            axes = [axes]
            fig.set_size_inches(7,5)

        fig.tight_layout()

        for i, ch_name, ax in zip(range(img.shape[2]), color_names, axes):
            ax.hist(img[:, :, i].ravel(), bins=bins, range=[0, 256], )
            ax.set_title(f"{title}-{ch_name}")
    else:
            fig, ax = plt.subplots(1, 1, figsize=(7,5))
            ax.hist(img.ravel(), bins=bins, range=[0, 256], cumulative=True, )
            ax.set_title(f"{title}-Cumulative")

    plt.show()
    if ret:
        return fig
```

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

```
from utils import img_read
img = img_read('../images/greenscreen.jpg')
```

```
from utils import show_hist
show_hist(img, channel_order="RGB", cumulative=False, bins=256)

def unique_count_app(a):
    colors, count = np.unique(a.reshape(-1,a.shape[-1]), axis=0, return_counts=True)
    return colors[count.argmax()]
unique_count_app(img)


def palette(clusters):
```

```
    width=300
    palette = np.zeros((50, width, 3), np.uint8)
    steps = width/clusters.cluster_centers_.shape[0]
    for idx, centers in enumerate(clusters.cluster_centers_):
        palette[:, int(idx*steps):(int((idx+1)*steps)), :] = centers
    return palette

from collections import Counter
def palette_perc(k_cluster):
    width = 300
    palette = np.zeros((50, width, 3), np.uint8)

    n_pixels = len(k_cluster.labels_)
    counter = Counter(k_cluster.labels_) # count how many pixels per cluster
    perc = {}
    for i in counter:
        perc[i] = np.round(counter[i]/n_pixels, 2)
    perc = dict(sorted(perc.items()))

    # For logging purposes
    #print(perc)
    #print(k_cluster.cluster_centers_)
    palette_percenteges = {k:v for k,v in zip([str(color.tolist()) for color in
k_cluster.cluster_centers_], perc.values()) }
    print(f"palette_percenteges: {palette_percenteges}")

    step = 0
    for idx, centers in enumerate(k_cluster.cluster_centers_):
        palette[:, step:int(step + perc[idx]*width+1), :] = centers
        step += int(perc[idx]*width+1)

    return palette_percenteges, palette




def show_img_compar(img_1, img_2):
    f, ax = plt.subplots(1, 2, figsize=(10, 10))
    ax[0].imshow(img_1)
    ax[1].imshow(img_2)
    ax[0].axis("off")  # hide the axis
    ax[1].axis("off")
    f.tight_layout()
    #f.suptitle(title, fontsize=32)
    plt.show()

from sklearn.cluster import KMeans

clt = KMeans(n_clusters=3)
clt_1 = clt.fit(img.reshape(-1, 3))

palette_percenteges, color_palette = palette_perc(clt_1)
# color_palette = palette(clt_1)
show_img_compar(img, color_palette)
```

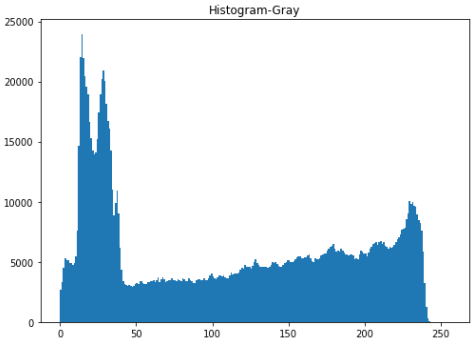
```python
def change_background(img, threshold=((0, 170, 0), (200, 255, 150)), new_background=(150, 255, 255),
convert_rgb=False):

    ## Method 1: RGB filtering
    if convert_rgb:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # bgr 2 rgb
    plt.imshow(img, cmap='gray')
    plt.title('Original')
    plt.show()

    lower_th = np.array([threshold[0]])     # [R value, G value, B value]
    upper_th = np.array([threshold[1]])

    mask = cv2.inRange(img, lower_th, upper_th)

    # Method 2: HSV filtering
    #hsv = cv2.cvtColor(img_org, cv2.COLOR_BGR2HSV)
    ## mask of green (36,25,25) ~ (86, 255,255)
    #mask = cv2.inRange(hsv, (36, 25, 25), (86, 255,255))
    #mask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))

    plt.imshow(mask, cmap='gray')
    plt.title('Mask')

    ## slice the green
    imask = mask>0
    green = np.zeros_like(img, np.uint8)
    green[imask] = img[imask]
    plt.imshow(green)
    plt.title('background')
    plt.show()

    masked_image = np.copy(img)
    masked_image[mask != 0] = [0, 0, 0]
    plt.imshow(masked_image)
    plt.title('masked_image')
    plt.show()

    if isinstance(new_background, str):
        background_image = cv2.imread(str)
        background_image = cv2.cvtColor(background_image, cv2.COLOR_BGR2RGB)
    elif isinstance(new_background, tuple) or isinstance(new_background, list):
        background_image = np.zeros_like(img)
        background_image[:] = new_background
    else:
        background_image=new_background

    # Crop background to fit the image
    crop_background = background_image[0:img.shape[0], 0:img.shape[1]]
    crop_background[mask == 0] = [0, 0, 0]

    plt.imshow(crop_background)
    plt.title('crop_background')
    plt.show()

    final_image = crop_background + masked_image
    plt.imshow(final_image)
    plt.title('final_image')
    plt.show()

    return final_image
```

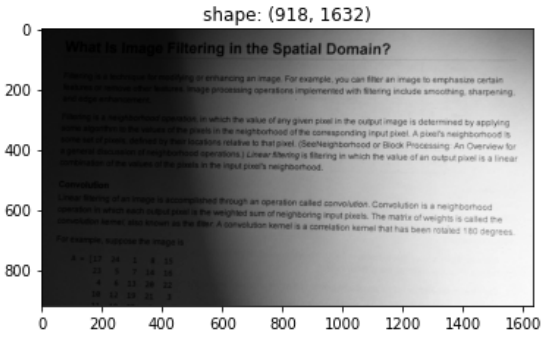
Ödev 2:

"images/png/printedtext.png" görüntüsündeki kararmanın giderilmesi için herhangi bir görüntü iyileştirme tekniği (lineer dönüşüm, gamma veya log dönüşümü, yerel veya genel histogram eşitleme) deneyip ikinci bir görüntü elde ediniz. İlk ve ikinci görüntünün histogramlarını bastırıp analiz ediniz. İyileştirilmiş görüntüde uygun bir eşik değeri belirleyip görüntüyü bölütleyiniz. Uyguladığınız iyileştirme yaklaşımının harfleri arka plandan ayırmada ne derece etkin olduğunu değerlendiriniz.

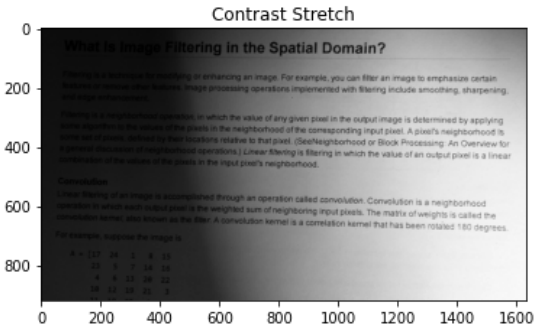
Sonuçlar ve Yorum:

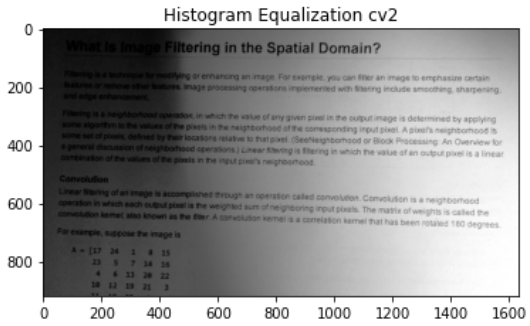
- Resim "Gray" formatında okundu ve histogramı çıkartıldı.

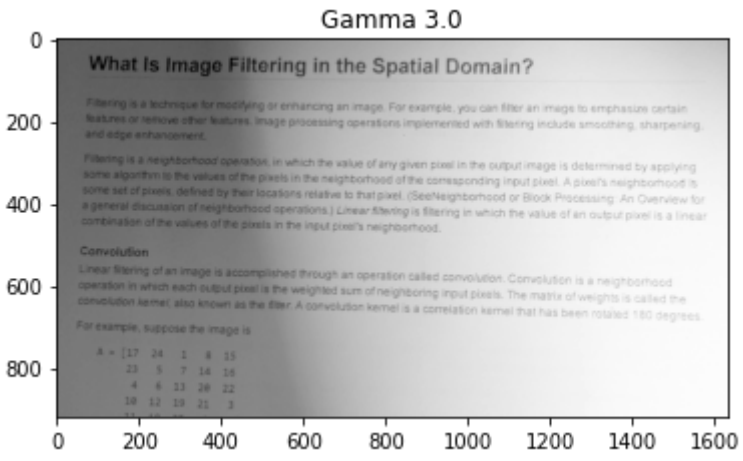




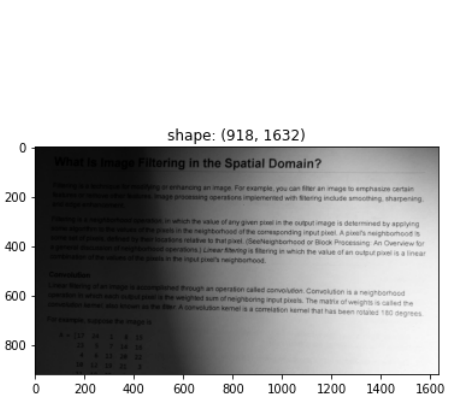
- Görüntüdeki kararmanın giderilmesi için,
  - Lineer dönüşüm olarak kontrast esnetme uygulandı.
  - Işık eşitsizliğini normalize etmek için histogram eşitleme uygulandı.
  - Karanlığı azaltmak için gamma düzeltmesi,

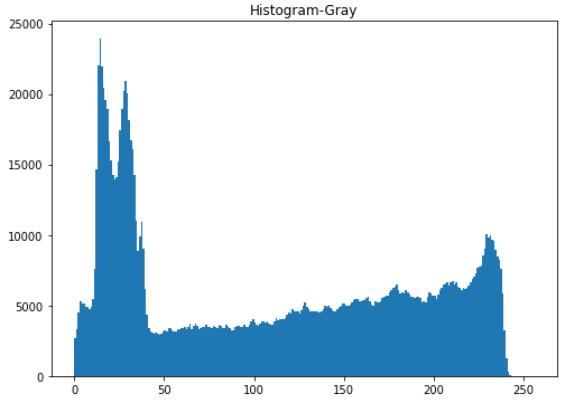


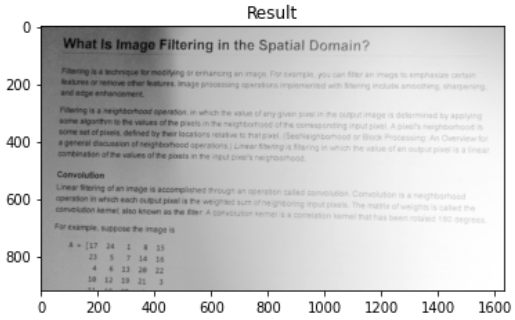


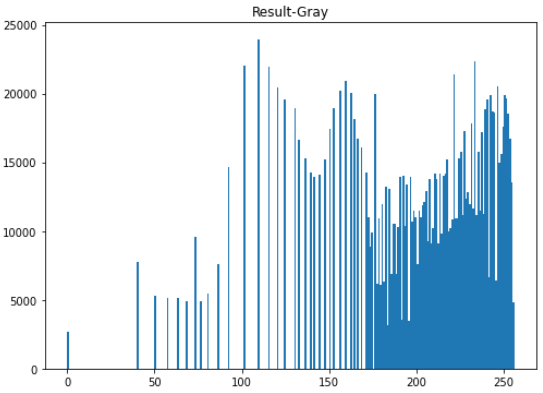


- İlk ve ikinci görüntünün histogramları bastırıp analiz edildi.

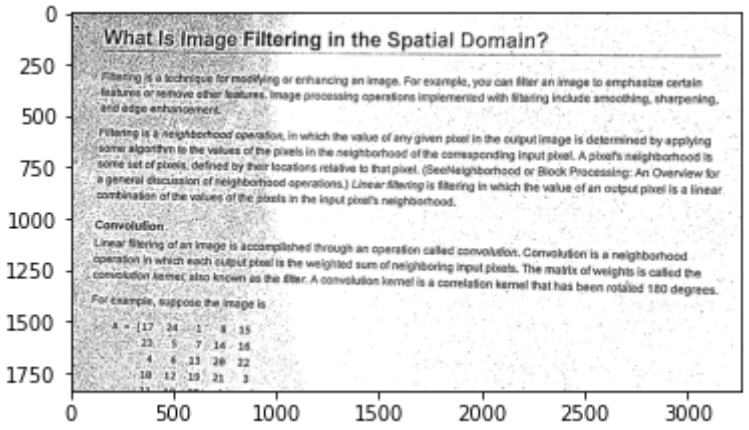








- Son olarak thresholdlama uygulandı. Burada threshold metodonun adaptif seçilmesi ile, threshold değerinin yanlış seçilmesi durumunda içerik kaybı önlendi.



- İşlemler sonucunda görseldeki yazının okunurluğunda gelişme oldu.
- Yazının görsele daha doğru hizalanması için skew correction uygulanabilir.

Program Kaynak Kodları:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np


from utils import show_hist
from utils import img_read
```

```
img_org = img_read('../images/png/printedtext.png', ret_gray=True)
show_hist(img_org)
```

```
img = np.copy(img_org)

# Linear Transformation
# Contrast stretching
def contrast_stretch(img):
    out = (img-img.min())/(img.max()-img.min())
    return ((out) * 255).astype(np.uint8)

img = contrast_stretch(img)

plt.imshow(img, cmap='gray')
plt.title('Contrast Stretch')
show_hist(img, title='Contrast Stretched', cumulative=False)
plt.show()


# Gamma Transformation
def adjust_gamma(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")
    # apply gamma correction using the lookup table
    return np.take(table, image)

gamma = 3.
img = adjust_gamma(img, gamma)

plt.imshow(img, cmap='gray')
plt.title(f'Gamma {gamma}')
show_hist(img, title=f'Gamma {gamma}')
plt.show()


# Histogram Equalization
img = cv2.equalizeHist(img)
plt.imshow(img, cmap="gray")
plt.title('Histogram Equalization cv2')
show_hist(img, title='Histogram Equalization cv2')
plt.show()
```

```
## Thresholding
def threshold_img(img, method="mean"):
    if isinstance(method, int):
        th_value = method
    elif method == "mean":
        th_value = img.mean()
    elif method == "median":
        th_value = np.median(img)

    return ((img > th_value)*255).astype(np.uint8)

img_th = threshold_img(img, method="mean")
plt.imshow(img_th, cmap='gray')
plt.show()

th , img_th = cv2.threshold(img, thresh=0, maxval= 255, type=cv2.THRESH_OTSU)
plt.imshow(img_th, cmap='gray')
plt.title('THRESH_OTSU')
plt.show()
```

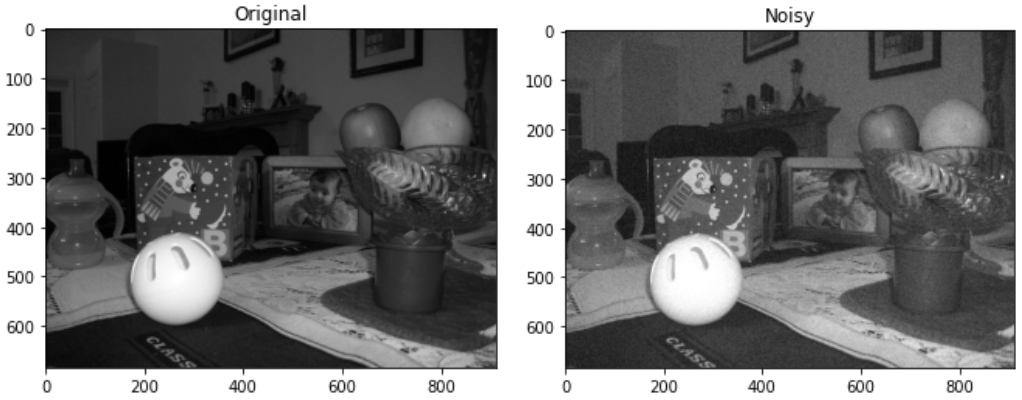
```
img_th = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2) #imgf contains Binary image
plt.title('adaptiveThreshold')
plt.imshow(img_th, cmap='gray')
plt.show()
```

Ödev 3:

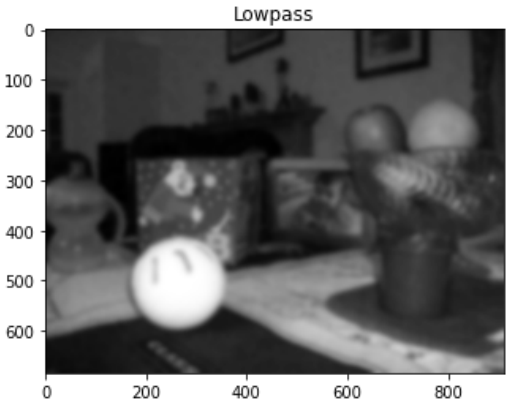
"images/png/toysflash.png" görüntüsünü gri düzey bir görüntüye çevirip üzerine Gauss gürültüsü ekleyiniz. Eklediğiniz gürültüyü temizlemek için standart sağması 5 olan bir Gauss filtre oluşturunuz ve alçak geçirgen filtreleme işlemini gerçekleştiriniz. Görüntünün orijinal, gürültü eklenmiş ve filtrelenmiş durumları için 3-B grafiğini çizdiriniz. Ardından filtrelenmiş görüntünün x ve y eksenlerindeki kısmi türevlerini hesaplayıp eğim genliği ve eğim açısı görüntülerini oluşturunuz. Açı görüntüsü içerisinden açısı π, $\pi/2$, $\pi/3$ ve $2\pi/3$ olan açıları çekip ayrı görüntüler oluşturunuz ve ekrana bastırınız. Açı görüntülerinin genlik görüntüsüne bağlı olarak nasıl oluştuğunu değerlendiriniz.

Sonuçlar ve Yorum:

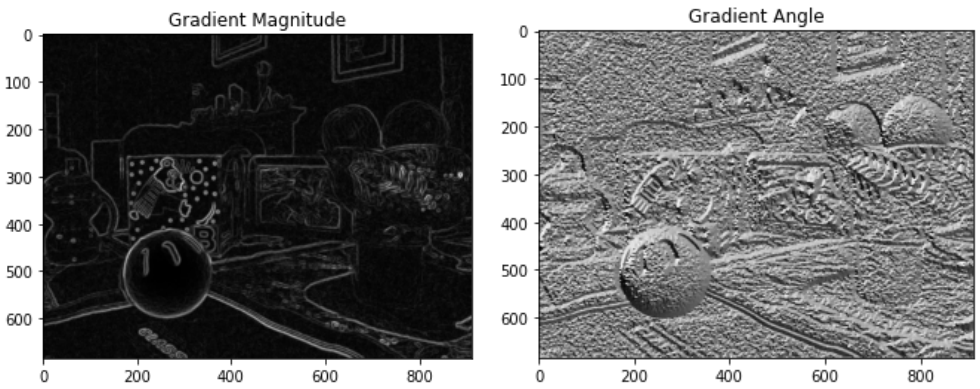
- "images/png/toysflash.png" görüntüsünü gri düzey bir görüntü olarak okunup üzerine Gauss gürültüsü eklendi.



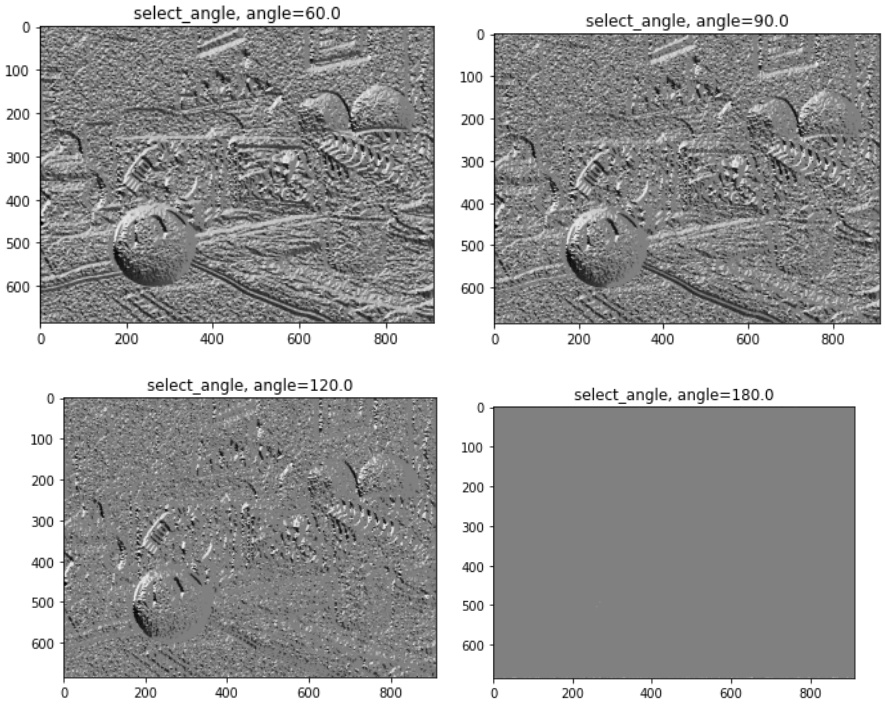
- gürültüyü temizlemek için standart sağması 5 olan bir Gauss filtre ile alçak geçiren filtreleme uygulandı.



- Filtrelenmiş görüntünün x ve y eksenlerindeki kısmi türevlerini hesaplayıp eğim genliği ve eğim açısı görüntülerini oluşturuldu.



- Açı görüntüsü içerisinden açısı π, $\pi/2$, $\pi/3$ ve $2\pi/3$ olan açıları çekip ayrı görüntüler oluşturuldu ve görselleştirildi.



- Açı görüntülerinin genlik görüntüsünün farklı detay seviyelerinde oluştuğu gözlemlendi.

Program Kaynak Kodları:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

from utils import img_read, show_hist
```

```
img_org = img_read('../images/png/toysflash.png', ret_gray=True)

def add_noise(img, noise_mult=0.2):
    noise = np.random.randint(0, 256, img.shape)
```

```
    noisy = np.clip(img + noise_mult*noise, 0, 255)
    return noisy

img_noise = add_noise(img_org,  noise_mult=0.2)
```

```
from scipy import signal

def gaussian_kernel(size=(10, 10), center=None, sigma=50.):
    """
    It produces single gaussian at expected center
    :param center:  the mean position (X, Y) - where high value expected
    :param size: The total image size (width, height)
    :param sigma: The sigma value
    :return:
    """
    if isinstance(size, int):
        size = (size, size)
    if center is None:
        center = (size[0]//2, size[1]//2)

    x_axis = np.linspace(0, size[0]-1, size[0]) - center[0]
    y_axis = np.linspace(0, size[1]-1, size[1]) - center[1]
    xx, yy = np.meshgrid(x_axis, y_axis)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))
    plt.imshow(kernel)
    plt.show()
    return kernel


gauss_kernel = gaussian_kernel(size=5)
lowpass = cv2.filter2D(img_noise, -1, gauss_kernel)

plt.imshow(lowpass, cmap='gray')
```

```
plt.imshow(img_org, cmap='gray')
plt.title('Original')
plt.show()

plt.imshow(img_noise, cmap='gray')
plt.title('Noisy')
plt.show()

plt.imshow(lowpass, cmap='gray')
plt.title('Lowpass')
plt.show()
```

```
def sobel_edge_detection(image, filter, verbose=False):
    #new_image_x = convolution(image, filter, verbose)
    new_image_x = cv2.filter2D(image, -1, filter)
    if verbose:
        plt.imshow(new_image_x, cmap='gray')
        plt.title("Horizontal Edge")
        plt.show()
```

```
    #new_image_y = convolution(image, np.flip(filter.T, axis=0), verbose)
    new_image_y = cv2.filter2D(image,-1, np.flip(filter.T, axis=0))

    if verbose:
        plt.imshow(new_image_y, cmap='gray')
        plt.title("Vertical Edge")
        plt.show()

    gradient_magnitude = np.sqrt(np.square(new_image_x) + np.square(new_image_y))

    gradient_magnitude *= 255.0 / gradient_magnitude.max()
    gradient_angle = np.arctan2(new_image_y, new_image_x)

    if verbose:
        plt.imshow(gradient_magnitude, cmap='gray')
        plt.title("Gradient Magnitude")
        plt.show()

        plt.imshow(gradient_angle, cmap='gray')
        plt.title("Gradient Angle")
        plt.show()

    return gradient_magnitude, gradient_angle


gradient_magnitude, gradient_angle = sobel_edge_detection(lowpass, np.array([[1, 0, -1], [2,
0, -2], [1, 0, -1]]), verbose=True)

x, y = pol2cart(gradient_magnitude, gradient_angle)
grad_image = np.hypot(x, y)

print(grad_image.shape)
plt.imshow(grad_image, cmap='gray')
plt.title("Gradient Image")
```

```
def select_polar_angle(angle, threshold):
    return np.where(np.abs(angle) < threshold, 0, angle)
```

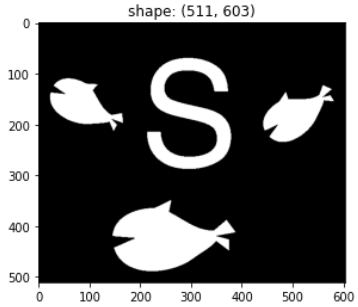
```
PI = np.pi
angles = [PI, 2*PI/3, PI/2, PI/3,]

for angle in angles:
    select_angle = select_polar_angle(gradient_angle, angle)

    plt.imshow(select_angle, cmap='gray')
    plt.title(f"select_angle, angle={(angle/PI)*180}")
    plt.show()
```

Ödev 4:

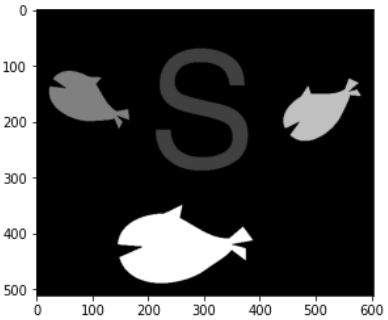
“images/sharks.png” görüntüsünü açıp ikili bir görüntü elde ediniz. Görüntü üzerinde daire şeklinde bir yapısal eleman kullanarak açma (opening) işlemi gerçekleştiriniz. Ardından görüntü üzerinde bağlı bileşen etiketleme algoritmasını çalıştırıp her bir etiketin farklı bir renkte gösterildiği etiket görüntüsünü oluşturunuz. Algoritma neticesinde görüntüde kaç adet köpek balığı sayıldığını belirtiniz.

Sonuçlar ve Yorum:

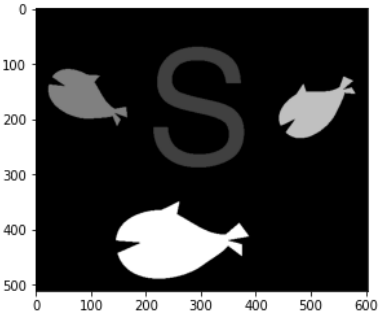
- “images/sharks.png” görüntüsünü açıp ikili bir görüntü elde edildi.



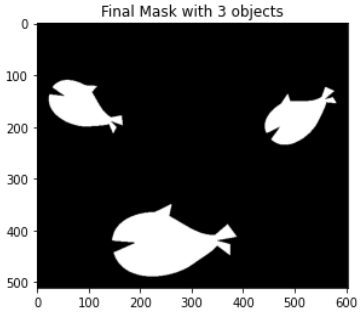
- Görüntü üzerinde daire şeklinde bir yapısal eleman kullanarak açma (opening) işlemi gerçekleştirildi.



- Görüntü üzerinde bağlı bileşen etiketleme algoritmasını çalıştırıp her bir etiketin farklı bir renkte gösterildiği etiket görüntüsünü oluşturuldu.



- Bağlı bileşen analizinin filtrelenmesi ile görüntüde kaç adet köpek balığı olduğunu belirten algoritma yazıldı.



Program Kaynak Kodları:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

from utils import img_read, show_hist
img = img_read('../images/sharks.png', ret_gray=True)
np.unique(img)
def circular_opening_morph(img, kernel_size=3, iterations=1):
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (kernel_size, kernel_size))
    return cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel, iterations=iterations)
img_open = circular_opening_morph(img, kernel_size=3, iterations=2)
plt.imshow(img_open, cmap='gray')
plt.title('Circular opening morph')
import scipy.ndimage.measurements as measurements

#Connected component labeling

def connected_components(img, connectivity=8):
    """
    Connected component labeling.
    Parameters:
    -----------
    img: 2d array
        Binary image.
    connectivity: int
        Connectivity of the graph.
    Returns:
    --------
    labels: 2d array
        Labels of the connected components.
    nbr_objects: int
        Number of objects.
    """
    labels, nbr_objects = measurements.label(img)
    print ("Number of objects:", nbr_objects)
    plt.imshow(labels, cmap='gray')
    print("unique:", np.unique(labels))
    return labels, nbr_objects

connected_components = connected_components(img)
output = cv2.connectedComponentsWithStats(img, 8, cv2.CV_32S)
(numLabels, labels, stats, centroids) = output
# initialize an output mask to store all characters parsed from
# the license plate
mask = np.zeros(img.shape, dtype="uint8")
object_count = 0

# loop over the number of unique connected component labels
# skip id=0 as it is background
for i in range(1, numLabels):
    # if this is the first component then we examine the
    # *background* (typically we would just ignore this
    # component in our loop)
```

```
    if i == 0:
        text = "examining component {}/{} (background)".format(
            i + 1, numLabels)
    # otherwise, we are examining an actual connected component
    else:
        text = "examining component {}/{}".format( i + 1, numLabels)

    # print a status message update for the current connected
    # component
    print("[INFO] {}".format(text))

    # extract the connected component statistics and centroid for
    # the current label
    x = stats[i, cv2.CC_STAT_LEFT]
    y = stats[i, cv2.CC_STAT_TOP]
    w = stats[i, cv2.CC_STAT_WIDTH]
    h = stats[i, cv2.CC_STAT_HEIGHT]
    area = stats[i, cv2.CC_STAT_AREA]
    (cX, cY) = centroids[i]
    print("area: ",area)
    print(f"width:{w}, height:{h}")


    # clone our original image (so we can draw on it) and then draw
    # a bounding box surrounding the connected component along with
    # a circle corresponding to the centroid
    output = img.copy()
    cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 3)
    cv2.circle(output, (int(cX), int(cY)), 10, (0, 0, 255), -1)


    # ensure the width, height, and area are all neither too small
    # nor too big
    keepWidth = w > 5 # and w < 50
    keepHeight = h < 150 # and h > 0
    keepArea = area > 500 # and area < 1500
    # ensure the connected component we are examining passes all
    # three tests
    if all((keepWidth, keepHeight, keepArea)):
        # construct a mask for the current connected component and
        # then take the bitwise OR with the mask
        print("[INFO] keeping connected component '{}'".format(i))
        componentMask = (labels == i).astype("uint8") * 255
        mask = cv2.bitwise_or(mask, componentMask)
        object_count += 1

plt.imshow(mask, cmap='gray')
plt.title(f"Final Mask with {object_count} objects")
plt.show()
```