

## **BLM6105-2 Uzaktan Algılama Dersi Ödev 6**

**24501100 - Aleyna Nil Uzunoğlu**

### **Giriş**

Bu çalışmada, üç farklı skimage görüntüsü üzerinde konvolüsyon filtreleri, gradyan analizi ve kenar tespiti algoritmaları uygulanmıştır. Kullanılan yöntemler görüntü işleme ve uzaktan algılama alanlarında sıkılıkla tercih edilen tekniklerdir.

Ödevin üç ana bölümünden oluşan yapısı sayesinde, hem temel filtreleme teknikleri hem de gradyan tabanlı ve model tabanlı kenar tespit yöntemleri karşılaştırılmış olarak incelenmiştir. Veri seti olarak scikit-image kütüphanesinden alınan gri seviye üç test görüntüsü: camera, astronaut, coins kullanılmıştır.

### **Adımlar**

#### *(a) Konvolüsyon Filtreleri Uygulama (Kutu, Gauss, Laplacian)*

Bu filtreler için  $k=3$  ve  $k=5$  çekirdek boyutlarında simetrik ve yansımalı padding uygulanarak konvolüsyon gerçekleştirilmiştir.

#### *(b) Gradyan Vektörleri (Magnitude & Orientation)*

- Görüntüdeki yönel değişim sobelfiltresi ile hesaplandı.
- Her piksel için:
  - Magnitude: Kenarın gücünü (parlaklık değişim şiddeti)
  - Orientation: Kenarın yönünü (açı bilgisi, derece cinsinden)

hesaplanmıştır.

#### *(c) Kenar Tespit Yöntemleri*

Aşağıdaki 5 kenar tespit algoritması her görüntüye uygulanmış ve görsel olarak karşılaştırılmıştır:

- Roberts
- Prewitt
- Sobel
- LoG (Laplacian of Gaussian)
- Canny

Ek: main.py

```
import numpy as np

import matplotlib.pyplot as plt

from skimage import data, io, color, util

from skimage.filters import gaussian, laplace

from skimage.filters import roberts, prewitt, sobel, sobel_v, sobel_h

from skimage.feature import canny

from scipy import ndimage as ndi

import os

OUTPUT_DIR = "outputs"

os.makedirs(OUTPUT_DIR, exist_ok=True)

def apply_convolution(image, filter_type='box', k_size=3, padding_mode='reflect',
                      **kwargs):

    if image.ndim > 2:

        image = color.rgb2gray(image)

        image = util.img_as_float(image)

    if filter_type == 'box':

        if k_size % 2 == 0:

            k_size += 1

        kernel = np.ones((k_size, k_size)) / (k_size * k_size)

        return ndi.convolve(image, kernel, mode=padding_mode)

    elif filter_type == 'gaussian':

        sigma = (k_size - 1) / 6.0 if k_size > 1 else 1
```

```
    return gaussian(image, sigma=sigma, mode=padding_mode, **kwargs)

elif filter_type == 'laplacian':

    return laplace(image, ksize=3, **kwargs)

else:

    raise ValueError("Desteklenmeyen filtre tipi. 'box', 'gaussian' veya 'laplacian' kullanın.")

def calculate_gradients(image):

    if image.ndim > 2:

        image = color.rgb2gray(image)

        image = util.img_as_float(image)

        grad_x = sobel_v(image)

        grad_y = sobel_h(image)

        magnitude = np.sqrt(grad_x**2 + grad_y**2)

        max_mag = np.max(magnitude)

        if max_mag > np.finfo(float).eps:

            magnitude = magnitude / max_mag

            orientation = np.arctan2(grad_y, grad_x)

    return grad_x, grad_y, magnitude, orientation

def apply_edge_detectors(image, log_sigma=1.0, canny_sigma=1.0):

    if image.ndim > 2:

        image = color.rgb2gray(image)

        image = util.img_as_float(image)

        edges = {}

        edges['Roberts'] = roberts(image)
```

```
edges['Prewitt'] = prewitt(image)

edges['Sobel'] = sobel(image)

im_gaussian = gaussian(image, sigma=log_sigma)

edges['LoG (Sigma={})'.format(log_sigma)] = laplace(im_gaussian)

edges['Canny (Sigma={})'.format(canny_sigma)] = canny(image, sigma=canny_sigma)

return edges
```

```
def plot_comparison(original, filtered_dict, main_title="Karşılaştırma", cmap='gray',
save_path=None):
```

```
    num_images = len(filtered_dict) + 1

    cols = 3

    rows = (num_images + cols - 1) // cols

    fig, axes = plt.subplots(rows, cols, figsize=(cols * 4, rows * 4))
```

```
    if num_images <= 1:
```

```
        ax = np.array([axes])
```

```
    elif rows == 1 and cols == 1:
```

```
        axes = np.array([[axes]])
```

```
    elif rows == 1 or cols == 1:
```

```
        axes = axes.reshape(rows, cols)
```

```
    if axes is None:
```

```
        print("Error: axes is None.")
```

```
    return
```

```
ax = axes.flat

ax[0].imshow(original, cmap='gray' if original.ndim == 2 else None)

ax[0].set_title("Orijinal")

ax[0].axis('off')
```

```
i = 1
```

```
for title, img in filtered_dict.items():
```

```
    if i < len(ax):
        img_cmap = cmap if img.ndim == 2 else None
        im = ax[i].imshow(img, cmap=img_cmap)
        ax[i].set_title(title)
        ax[i].axis('off')
```

```
    if 'Orientation' in title:
```

```
        fig.colorbar(im, ax=ax[i], fraction=0.046, pad=0.04)
```

```
    i += 1
```

```
else:
```

```
    break
```

```
for j in range(i, len(ax)):
```

```
    ax[j].axis('off')
```

```
fig.suptitle(main_title, fontsize=16)
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

if save_path:
    plt.savefig(save_path, bbox_inches='tight')
    print(f"Grafik kaydedildi: {save_path}")
    plt.close(fig)

else:
    plt.show()

image_names = ['camera', 'moon', 'coins']

images_to_process = {name: getattr(data, name)() for name in image_names}

# a
print("--- (a): Konvolüsyon Filtreleri")

for name, img in images_to_process.items():
    print(f"\nİşlenen Görüntü: {name}")

    filtered_a = {}

    for k in [3, 5]:
        try:
            filtered_a[f'Box (k={k})'] = apply_convolution(img, filter_type='box', k_size=k)
            filtered_a[f'Gaussian (k={k})'] = apply_convolution(img, filter_type='gaussian',
                                                               k_size=k)
        except Exception as e:
            print(f" Hata ({name}, k={k}): {e}")
```

try:

```
filtered_a[f'Laplacian (k=3)'] = apply_convolution(img, filter_type='laplacian', k_size=3)
```

except Exception as e:

```
print(f' Hata (Laplacian, {name}): {e}'")
```

if filtered\_a:

```
output_filename = os.path.join(OUTPUT_DIR, f'{name}_a_convolution.png")
```

```
plot_comparison(img, filtered_a, f'{name} - Konvolüsyon Filtreleri",
save_path=output_filename)
```

# (b): Gradyanlar

```
print("\n--- (b): Gradyanlar ---")
```

```
for name, img in images_to_process.items():
```

```
print(f'\nİşlenen Görüntü: {name}')
```

try:

```
gx, gy, mag, ori = calculate_gradients(img)
```

```
filtered_b = {
```

```
'Gradient X (Sobel_V)': gx,
```

```
'Gradient Y (Sobel_H)': gy,
```

```
'Magnitude': mag,
```

```
'Orientation': ori
```

```
}
```

```
cmaps = ['gray', 'gray', 'gray', 'hsv']
```

```
num_images = len(filtered_b) + 1

cols = 3

rows = (num_images + cols - 1) // cols

fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))

if num_images <= 1: ax = np.array([axes])

elif rows == 1 and cols == 1: axes = np.array([[axes]])

elif rows == 1 or cols == 1: axes = axes.reshape(rows, cols)

if axes is None:

    print("Error: axes is None.")

    continue

ax = axes.flat

ax[0].imshow(img, cmap='gray' if img.ndim == 2 else None)

ax[0].set_title("Orijinal")

ax[0].axis('off')

i = 1

for title, filtered_img in filtered_b.items():

    if i < len(ax):

        im = ax[i].imshow(filtered_img, cmap=cmaps[i-1])

        ax[i].set_title(title)
```

```
    ax[i].axis('off')

    fig.colorbar(im, ax=ax[i], fraction=0.046, pad=0.04)

    i += 1

else: break

for j in range(i, len(ax)): ax[j].axis('off')

fig.suptitle(f'{name} - Gradyanlar", fontsize=16)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

output_filename_b = os.path.join(OUTPUT_DIR, f'{name}_b_gradients.png')

plt.savefig(output_filename_b, bbox_inches='tight')

print(f'Grafik kaydedildi: {output_filename_b}')

plt.close(fig)

except Exception as e:

    print(f'Gradyan hesaplama/kaydetme hatası ({name}): {e}')

# (c): Kenar Belirleme Filtreleri

print("\n--- (c) Kenar Belirleme Filtreleri ---")

for name, img in images_to_process.items():

    print(f"\nİşlenen Görüntü: {name}")

    try:

        edges_c = apply_edge_detectors(img, log_sigma=1.5, canny_sigma=1.5)
```

```

output_filename_c = os.path.join(OUTPUT_DIR, f'{name}_c_edge_detection.png')

plot_comparison(img, edges_c, f'{name} - Kenar Belirleme Filtreleri', cmap='gray',
save_path=output_filename_c)

except Exception as e:

    print(f'Kenar belirleme/kaydetme hatası ({name}): {e}')

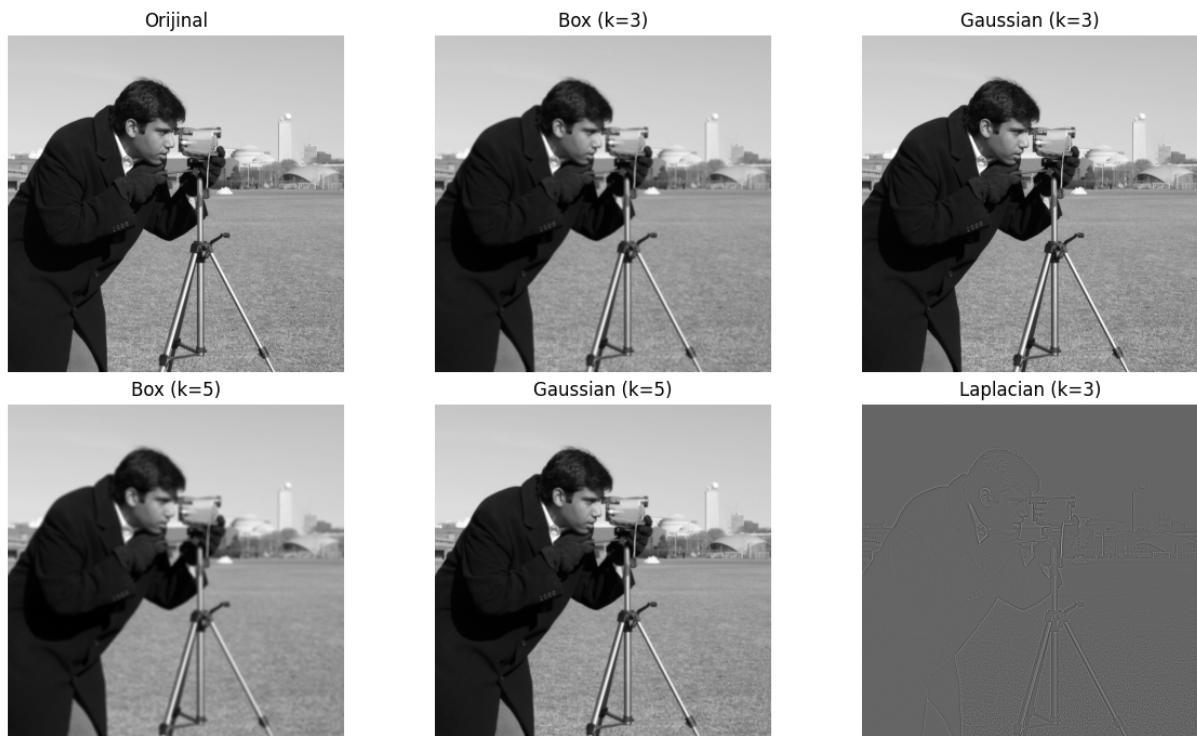
print(f'\nİşlem Tamamlandı. Çıktılar '{OUTPUT_DIR}' dizinine kaydedildi.')

```

Ek : Çıktılar

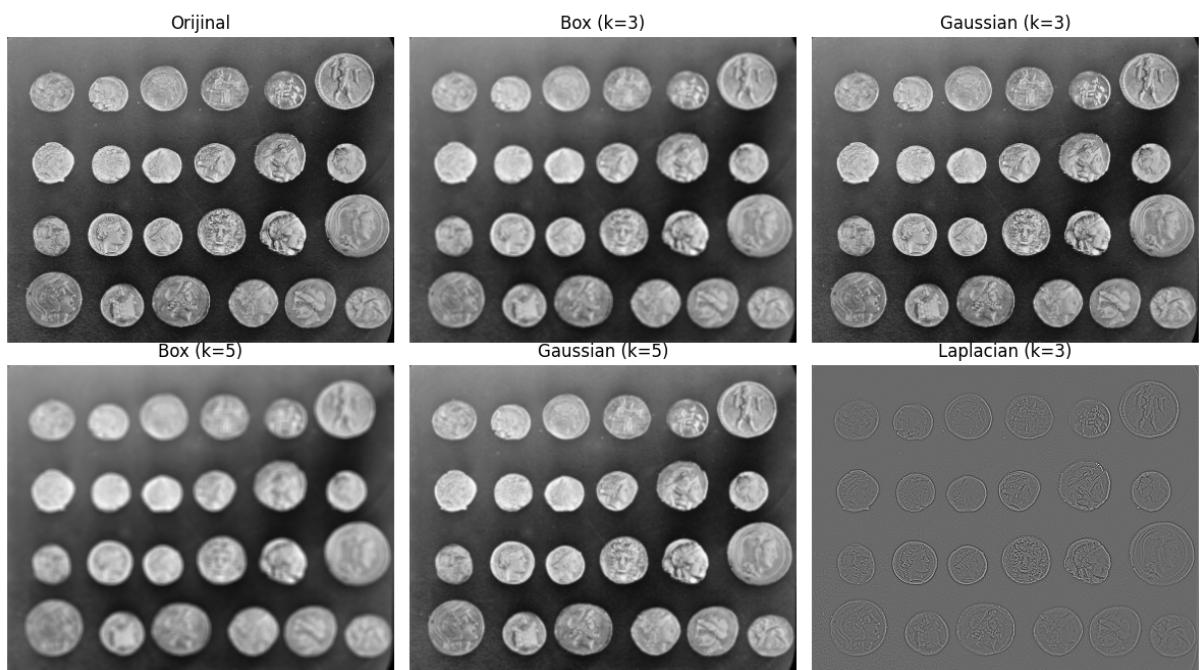
### 1) a bölümü için

camera - Konvolüsyon Filtreleri



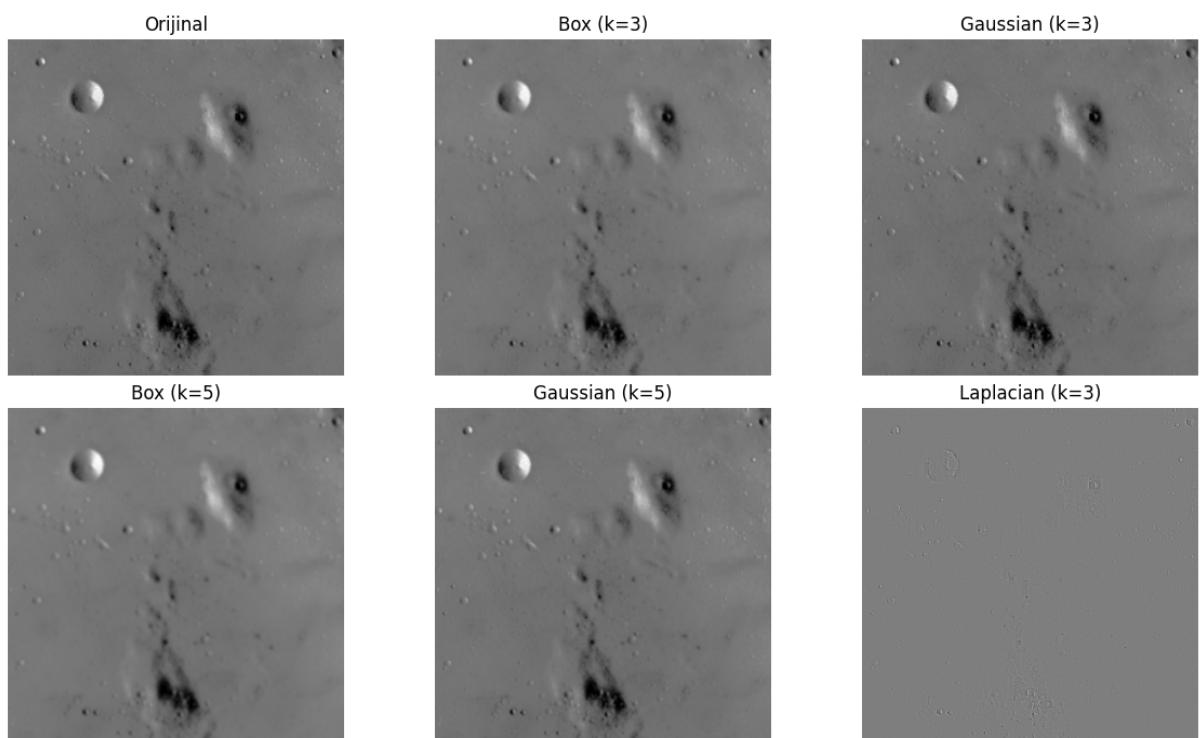
Şekil 1.1) camera görüntüsüne uygulanan konvolüsyon filtreleri

coins - Konvolüsyon Filtreleri



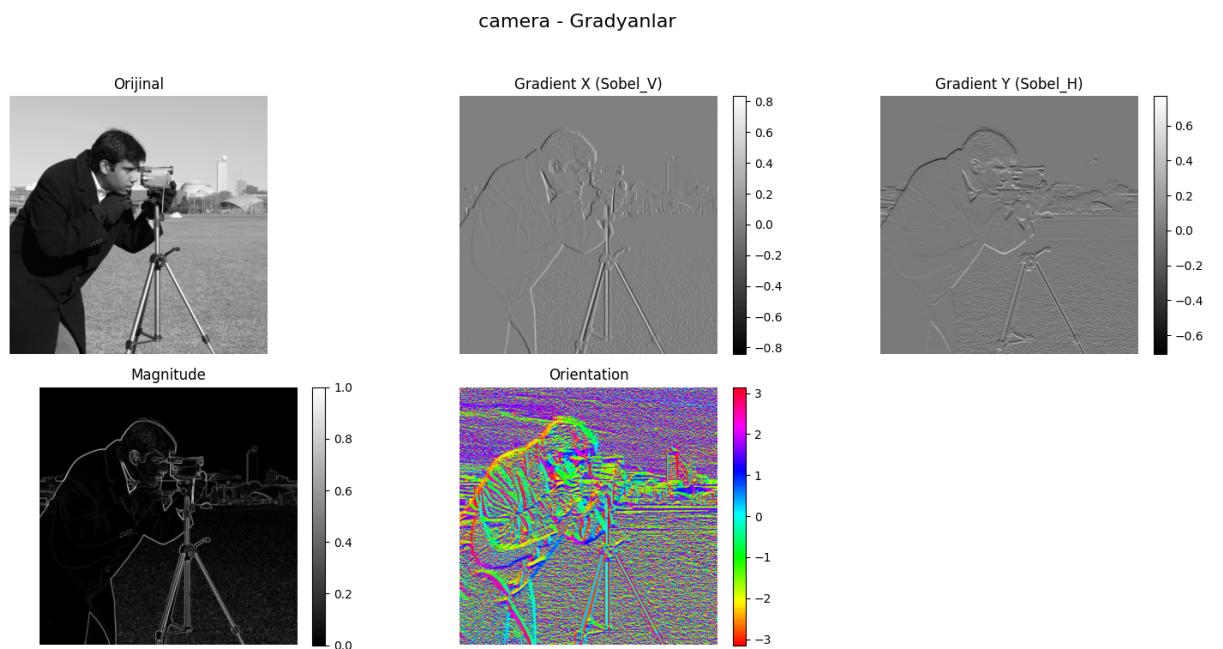
Şekil 1.2) coins görüntüsüne uygulanan konvolüsyon filtreleri

moon - Konvolüsyon Filtreleri

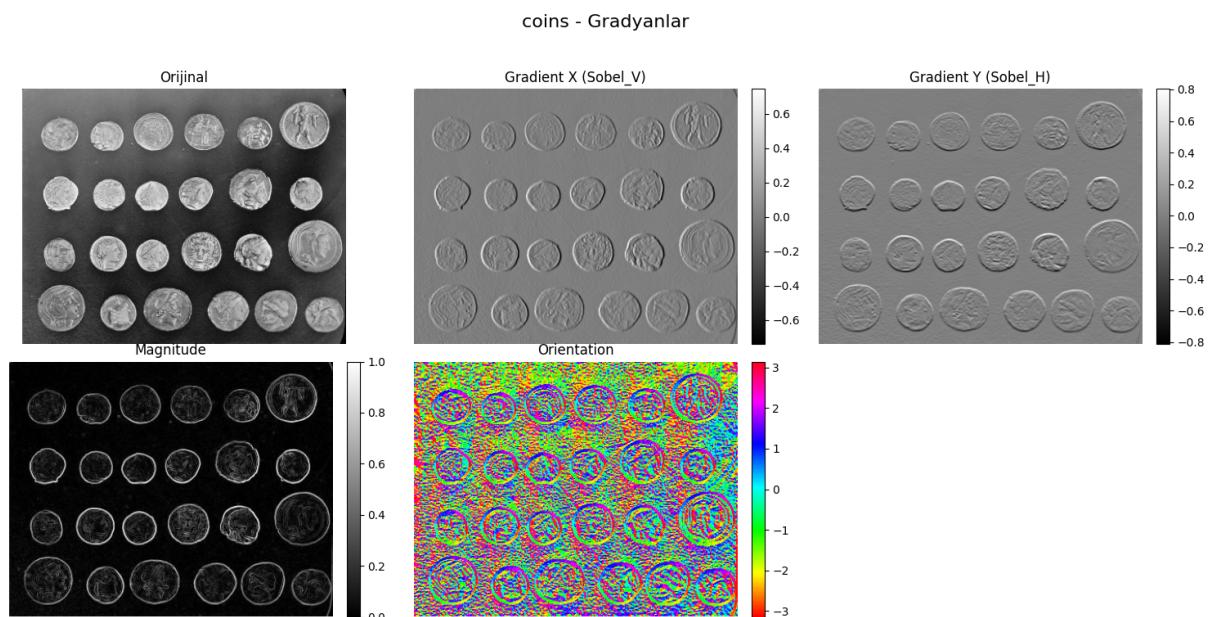


Şekil 1.3) moon görüntüsüne uygulanan konvolüsyon filtreleri

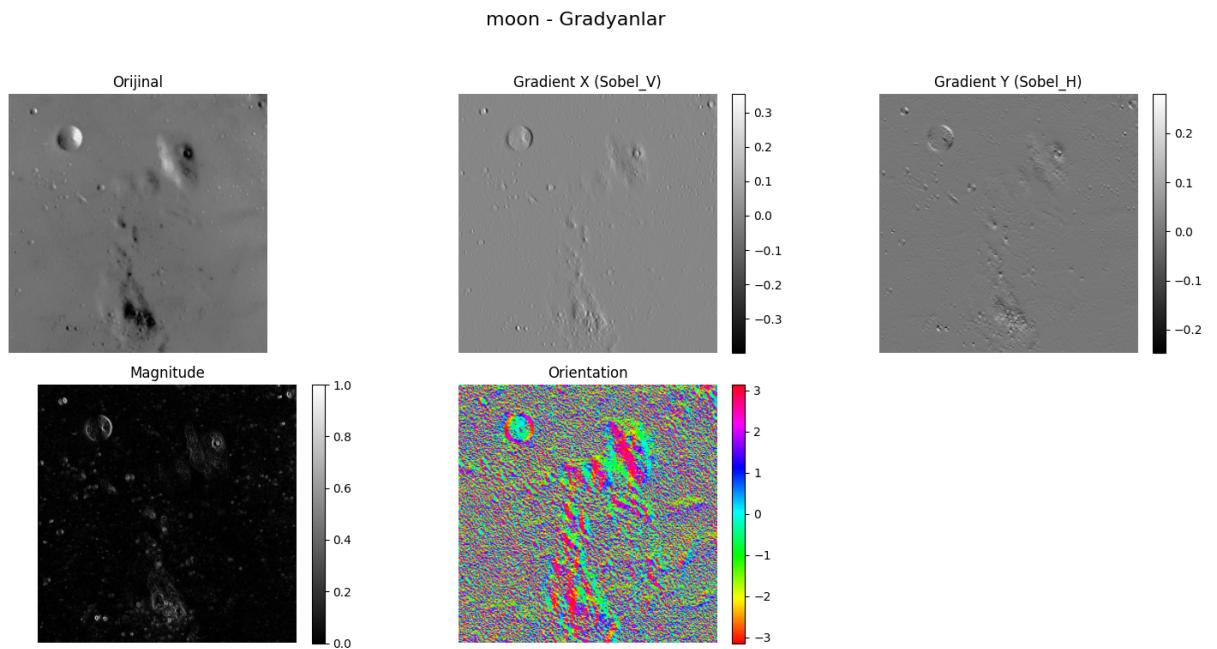
## 2) b bölümü için



Şekil 2.1) camera görüntüsünün gradyanları

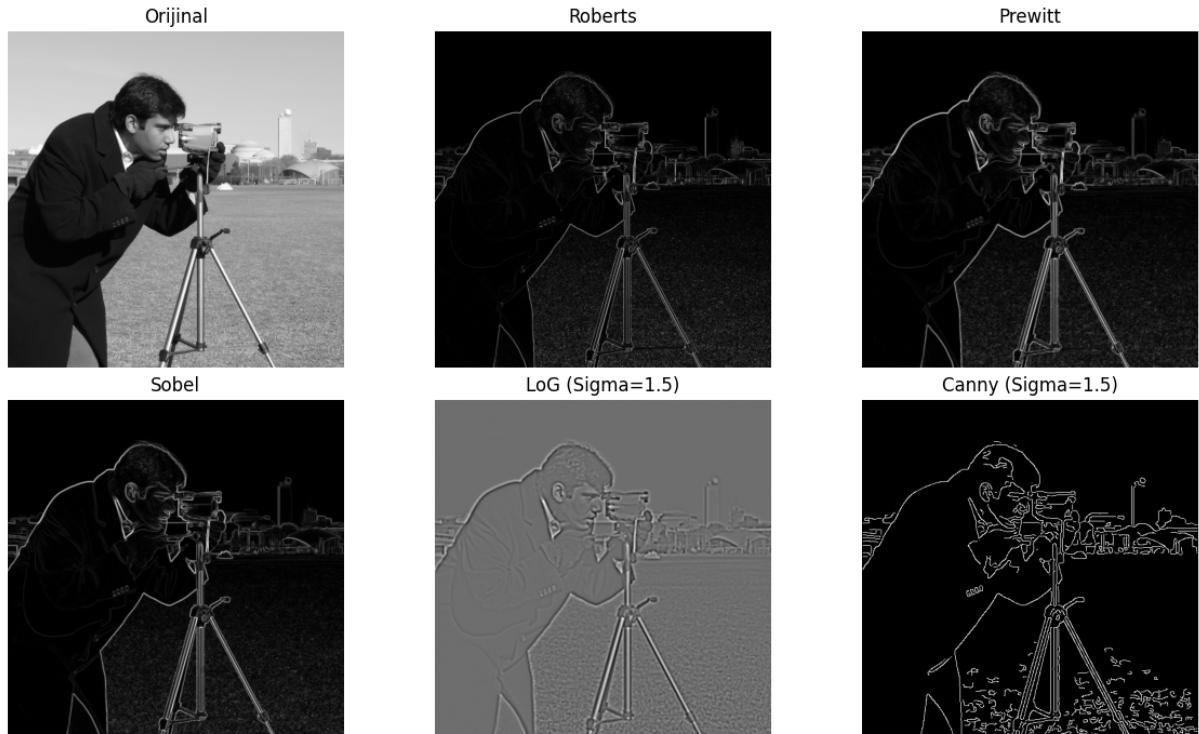


Şekil 2.2) coins görüntüsünün gradyanları



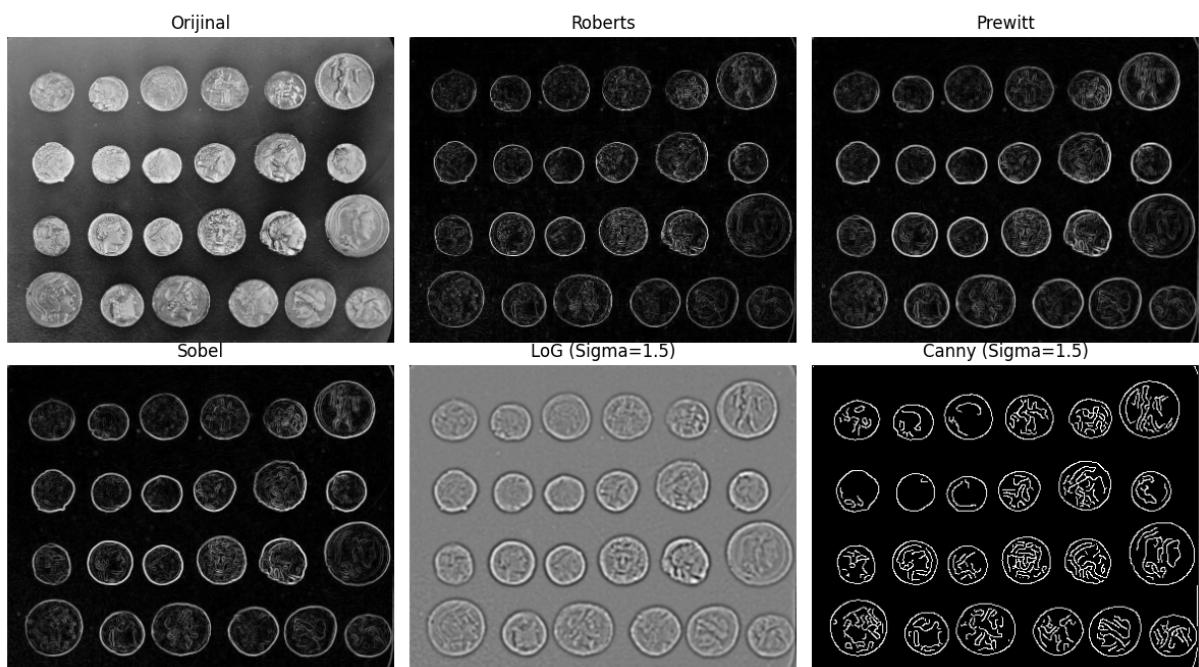
Şekil 2.3) moon görüntüsünün gradyanları

### 3) c bölümü çıktıları camera - Kenar Belirleme Filtreleri



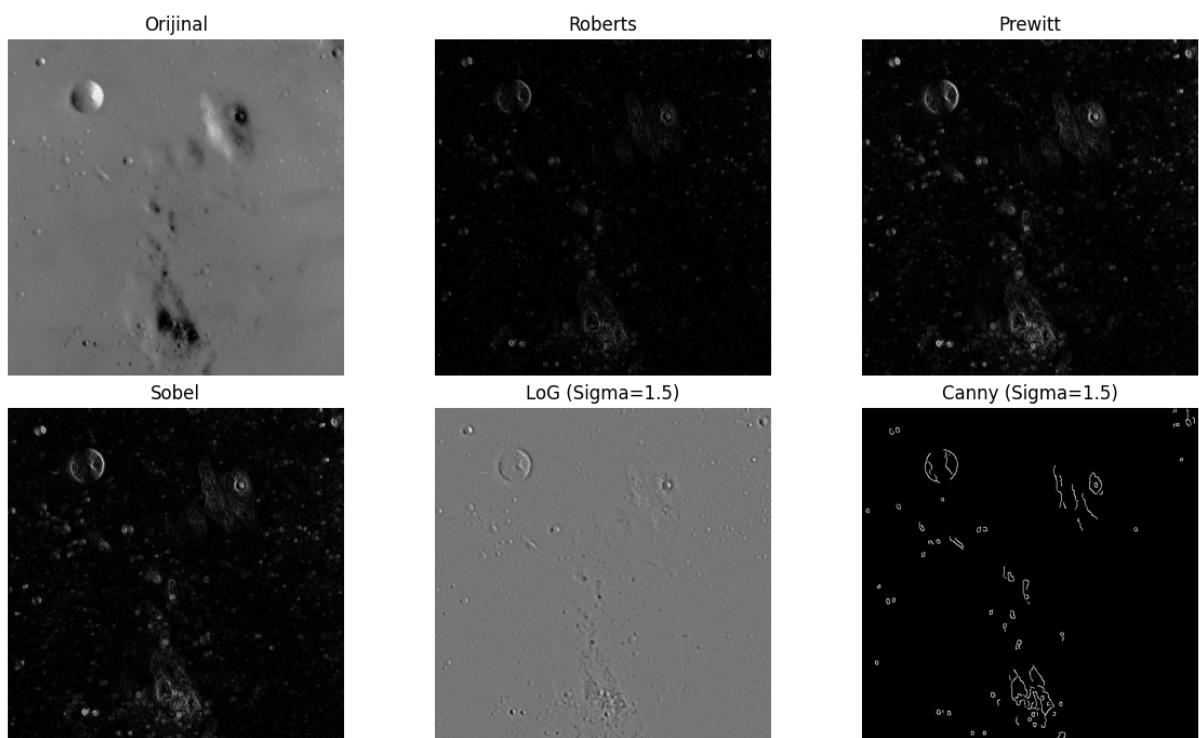
Şekil 3.1) camera görüntüsü için kenar belirleme filtreleri

coins - Kenar Belirleme Filtreleri



Şekil 3.2) coins görüntüsü için kenar belirleme filtreleri

moon - Kenar Belirleme Filtreleri



Şekil 3.3) moon görüntüsü için kenar belirleme filtreleri