INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA** **Campus Birigui** **Bacharelado em Engenharia de Computação**
Disciplina: Processamento Digital de Imagens	Filtragem Espacial
Professor: Prof. Dr. Murilo Varges da Silva	**Data:** 07/09/2023
Nome do aluno: Leonardo Reneres dos Santos	**Prontuário:** BI3009131

Questões

- **Implementar a operação de convolução.**
- **Utilizando OPENCV, scipy função convolve e implementação manual.**
- **Implementar seguintes máscaras:**
  - Média
  - Guassiano
  - Laplaciano
  - Sobel X
  - Sobel Y
  - Gradiente (Sobel X + Sobel Y)
  - Laplaciano somado a imagem original
- **Utilizar as imagens já disponibilizadas: biel, lena, cameraman, etc.**

Respostas:

```
import matplotlib.pyplot as plt

import cv2

import numpy as np

from PIL import Image

from imageio import imread
```

```
from scipy import ndimage

import numpy as np



def plotar(im, im2, name):



    fig = plt.figure()

    plt1 = plt.subplot(1,2,1)

    plt2 = plt.subplot(1,2,2)



    plt1.title.set_text('original')

    plt2.title.set_text(name)



    plt1.imshow(im, cmap='gray', vmin=0, vmax=255)

    plt2.imshow(im2, cmap='gray', vmin=0, vmax=255)

    plt.show()







def main():



    identity = np.array((

    [0, 0, 0],

    [0, 1, 0],

    [0, 0, 0]), dtype="int")



  # mean
```

```
    mean = np.array((
    [0.1111, 0.1111, 0.1111],
    [0.1111, 0.1111, 0.1111],
    [0.1111, 0.1111, 0.1111]), dtype="float")


# gauss
    gauss = np.array((
    [0.0625, 0.125, 0.0625],
    [0.1250, 0.250, 0.1250],
    [0.0625, 0.125, 0.0625]), dtype="float")


  # construct the Laplacian kernel used to detect edge-like regions of
an image
    laplacian = np.array((
    [0,  1, 0],
    [1, -4, 1],
    [0,  1, 0]), dtype="int")


  # construct the Sobel x-axis kernel
    sobelX = np.array((
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]), dtype="int")


  # construct the Sobel y-axis kernel
```

```
    sobelY = np.array((

    [-1,0,1],

    [-2,0,2],

    [-1,0,1]), dtype="int")



  # boost

    boost = np.array((

    [ 0,  -1,  0],

    [-1, 5.7, -1],

    [ 0,  -1,  0]), dtype="float")



    kernel = np.array((

    [1,  0, -1],

    [0, 0, 0],

    [-1,  0, 1]), dtype="int")



    print("Selecione a imagem: 1 , 2 ou 3")

    im = input()



    if im == '1':

        im1 =cv2.imread("lena_gray_512.tif")

        im1c =np.array( Image.open('lena_gray_512.tif'))

    elif im == '2':

        im1 = cv2.imread('biel.png')

        im1c = np.array( Image.open('biel.png'))
```

```
elif im == '3':
    im1 = cv2.imread('cameraman.tif')
    im1c = np.array(Image.open('cameraman.tif'))
else:
    im1 = cv2.imread('lena_gray_512.tif')
    im1c =np.array( Image.open('lena_gray_512.tif'))



# im1f = ndimage.convolve(im1c, kernel, mode='reflect')
# im1f = np.where(im1f < 0, 0, im1f)
# print(im1)
# imo = Image.fromarray(im1f.astype(np.uint8))
# imo.show()


mask = identity.copy()
name = 'Identity'
plotar(im1, im1, name)


while (True):
    key = input()
    if key == 'q':
        break
    elif key == 'i':
        mask = identity.copy()
        name = 'Identity'
```

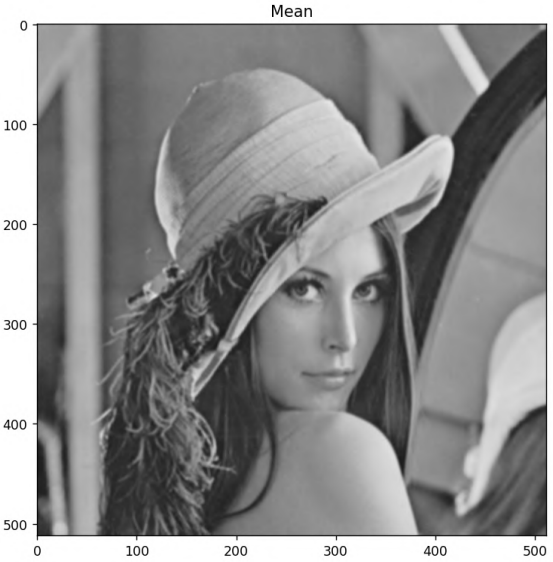
```
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'm':
            mask = mean.copy()
            name = 'Mean'
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'g':
            mask = gauss.copy()
            name = 'Gauss'
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'l':
            mask = laplacian.copy()
            name = 'Laplacian'
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key =='x':
            mask = sobelX.copy()
            name = 'SobelX'
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'y':
            mask = sobelY.copy()
            name = 'SobelY'
```

```
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'b':
            mask = boost.copy()
            name = 'Boost'
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=mask)
            plotar(im1, im1f, name)
        elif key == 'G':
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=sobelX)
            im2 = cv2.filter2D(src=im1, ddepth=-1, kernel=sobelY)
            imf = im1f + im2
            name = 'Gradiente'

            plotar(im1, imf, name)

        elif key == 'O':
            im1f = cv2.filter2D(src=im1, ddepth=-1, kernel=laplacian)
            im2f = cv2.filter2D(src=im1, ddepth=-1, kernel=identity)
            imf = im2f + im1f
            name = 'Laplacian + Identity'

            fig = plt.figure()
            plt1 = plt.subplot(1,3,1)
            plt2 = plt.subplot(1,3,2)
            plt3 = plt.subplot(1,3,3)
```

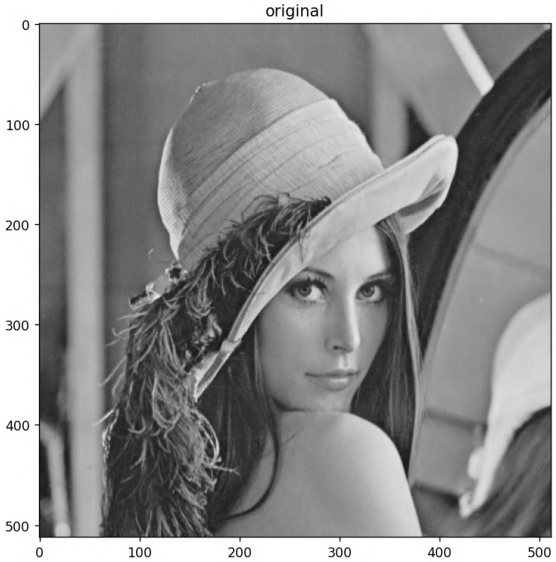
```
            plt1.title.set_text('original')
            plt2.title.set_text('Laplacian')
            plt3.title.set_text(name)

            plt1.imshow(im1, cmap='gray', vmin=0, vmax=255)
            plt2.imshow(im1f, cmap='gray', vmin=0, vmax=255)
            plt3.imshow(imf, cmap='gray', vmin=0, vmax=255)

            plt.show()
        elif key == 'C':
            name = 'Convolution'
            im1f = ndimage.convolve(im1c, kernel, mode='reflect')
            print(im1f)
            plotar(im1, im1f, name)
        elif key == 'S':
            name = 'Convolution and zero replace'
            im1f = ndimage.convolve(im1c, kernel, mode='reflect')
            im1f = np.where(im1f < 0, 0, im1f)
            print(im1f)
            plotar(im1, im1f, name)

if __name__ == "__main__":
    main()
```

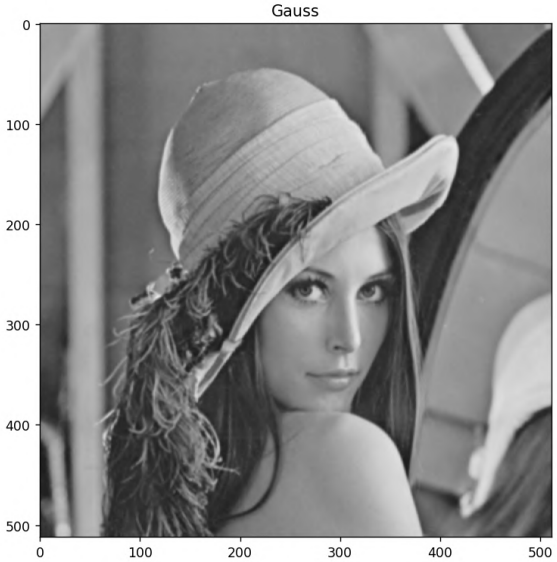
Mean:


original
0
0
0
0
0
0
0
100
200
300
400
500

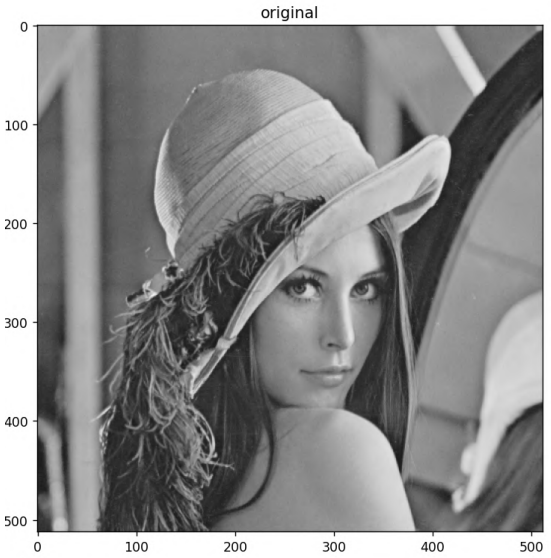

Mean
0
100
200
300
400
500
0
100
200
300
400
500

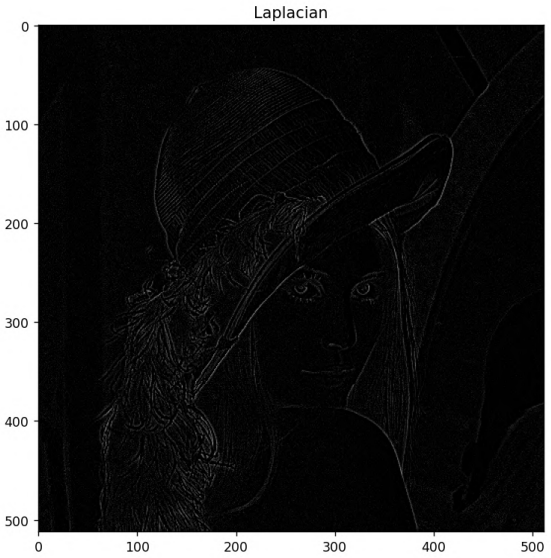
Gauss:


original
0
100
200
300
400
500
0
100
200
300
400
500

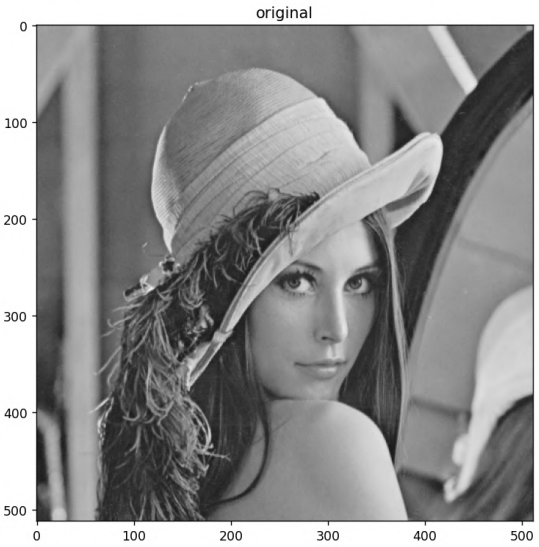

Gauss
0
100
200
300
400
500
0
100
200
300
400
500

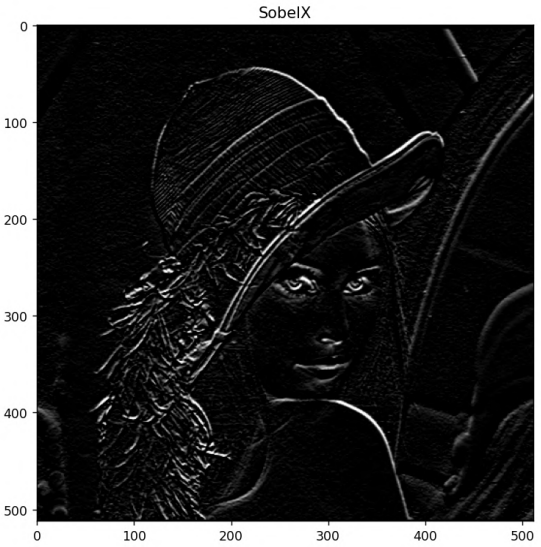
Laplacian:



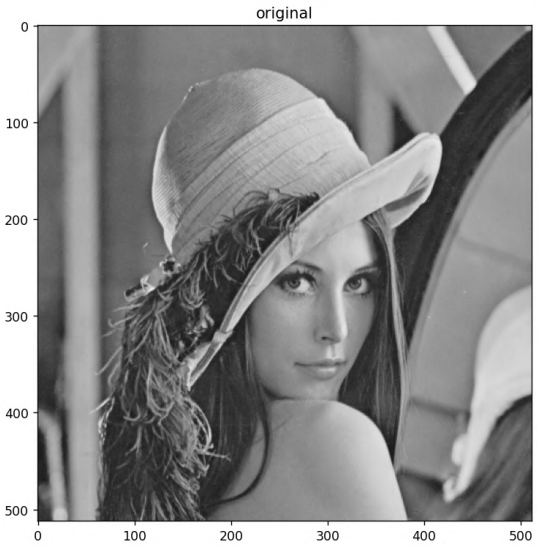


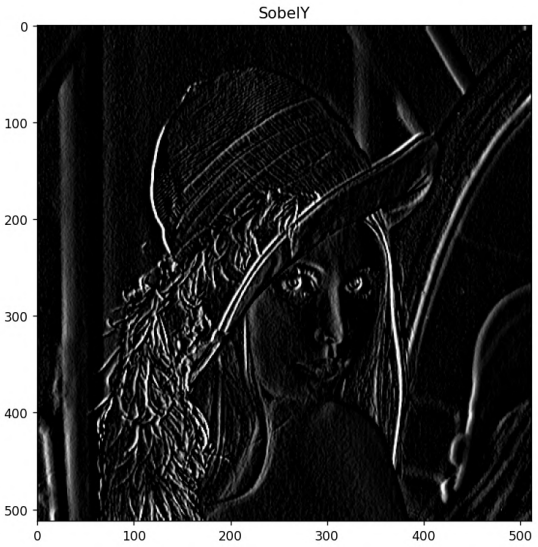
Sobelx:



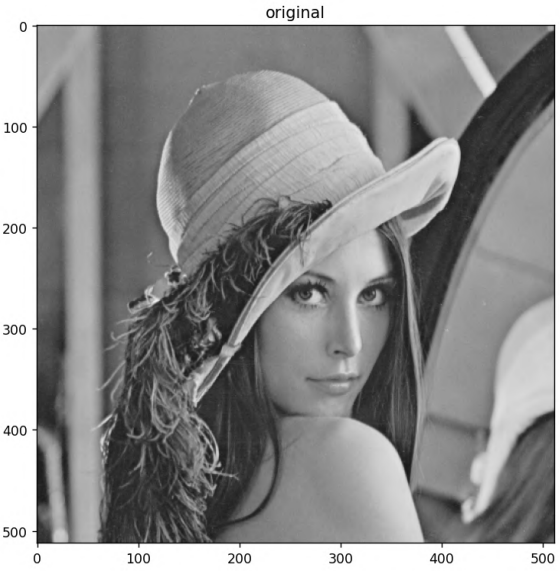


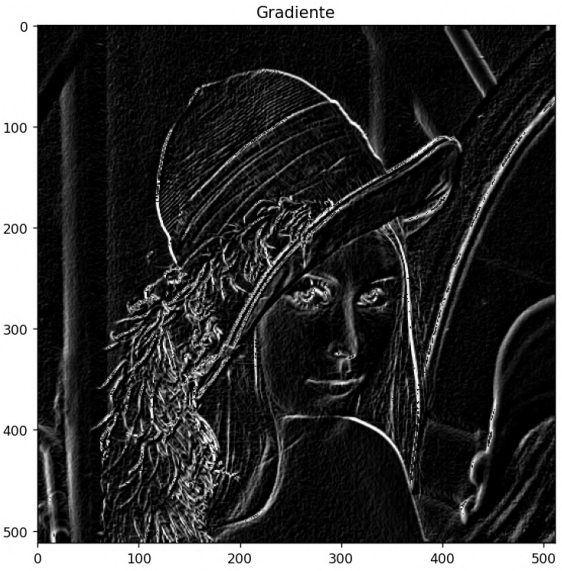
SobelY:



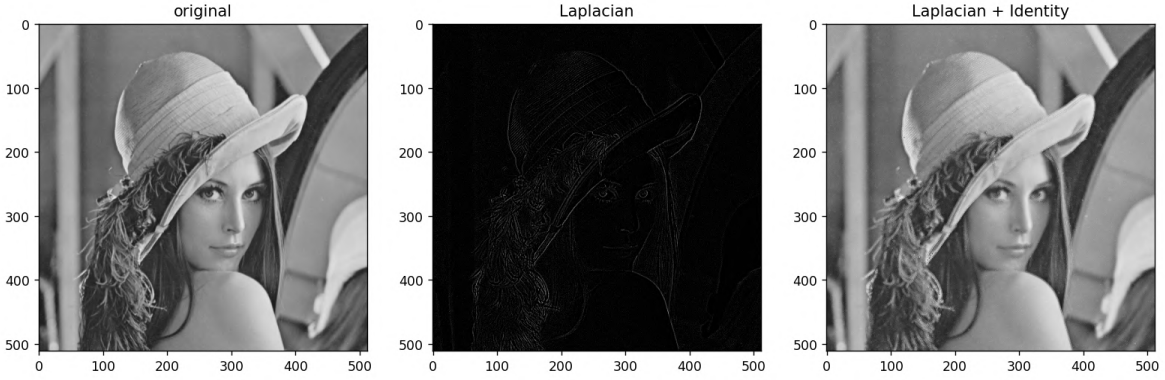


Gradiente:





Laplacian+Identity:



Convolution: