MACHINE VISION

DIGITAL ASSIGNMENT – 1

SRIVIDHYA S

20BPS1119

**QUESTION:**

Write a MATLAB program for displaying Image Restoration techniques.

**CODE:**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**def** show\_image(image, title**=**'Image', cmap\_type**=**'gray'):

plt**.**imshow(image, cmap**=**cmap\_type)

plt**.**title(title)

plt**.**axis('off')

**def** plot\_comparison(img\_original, img\_filtered, img\_title\_filtered):

fig, (ax1, ax2) **=** plt**.**subplots(ncols**=**2, figsize**=**(10, 8), sharex**=True**, sharey**=True**)

ax1**.**imshow(img\_original, cmap**=**plt**.**cm**.**gray)

ax1**.**set\_title('Original')

ax1**.**axis('off')

ax2**.**imshow(img\_filtered, cmap**=**plt**.**cm**.**gray)

ax2**.**set\_title(img\_title\_filtered)

ax2**.**axis('off')

**Image in-painting**

**import** sklearn **as** sk

**import** skimage

**from** skimage **import** data

**from** skimage.morphology **import** disk, binary\_dilation

**from** skimage.restoration **import** inpaint

image\_orig **=** data**.**astronaut()

*# Create mask with six block defect regions*

mask **=** np**.**zeros(image\_orig**.**shape[:**-**1], dtype**=**bool)

mask[20:60, 0:20] **=** 1

mask[160:180, 70:155] **=** 1

mask[30:60, 170:195] **=** 1

mask[**-**60:**-**30, 170:195] **=** 1

mask[**-**180:**-**160, 70:155] **=** 1

mask[**-**60:**-**20, 0:20] **=** 1

*# add randomly positioned small point-like defects*

rstate **=** np**.**random**.**default\_rng(0)

**for** radius **in** [0, 2, 4]:

*# larger defects are less common*

thresh **=** 3 **+** 0.25 **\*** radius *# make larger defects less common*

tmp\_mask **=** rstate**.**standard\_normal(image\_orig**.**shape[:**-**1]) **>** thresh

**if** radius **>** 0:

tmp\_mask **=** binary\_dilation(tmp\_mask, disk(radius, dtype**=**bool))

mask[tmp\_mask] **=** 1

*# Apply defect mask to the image over the same region in each color channel*

image\_defect **=** image\_orig **\*** **~**mask[**...**, np**.**newaxis]

image\_result **=** inpaint**.**inpaint\_biharmonic(image\_defect, mask, channel\_axis**=-**1)

fig, axes **=** plt**.**subplots(ncols**=**3, nrows**=**1)

ax **=** axes**.**ravel()

ax[0]**.**set\_title('Original image')

ax[0]**.**imshow(image\_orig)

ax[1]**.**set\_title('Defected image')

ax[1]**.**imshow(image\_defect)

ax[2]**.**set\_title('Inpainted image')

ax[2]**.**imshow(image\_result)

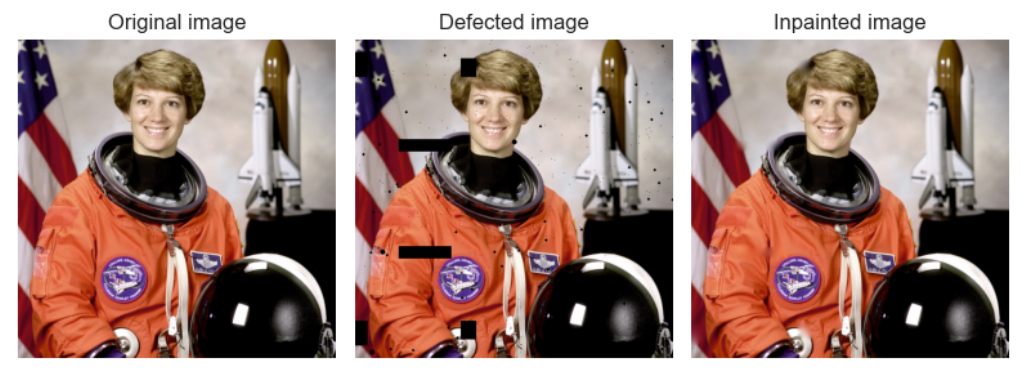
**for** a **in** ax:

a**.**axis('off')

fig**.**tight\_layout()

plt**.**show()

**OUTPUT:**



**Types of Noise**

**CODE:**

import skimage

import imageio as iio

import matplotlib.pyplot as plt

img = iio.imread("peppers.png")

def plotnoise(img, mode, r, c, i):

plt.subplot(r,c,i)

if mode is not None:

gimg = skimage.util.random\_noise(img, mode=mode)

plt.imshow(gimg)

else:

plt.imshow(img)

plt.title(mode)

plt.axis("off")

plt.figure(figsize=(18,24))

r=4

c=2

plotnoise(img, None, r,c,1)

plt.title('Original')

plotnoise(img, "Gaussian", r,c,2)

plotnoise(img, "Salt", r,c,3)

plotnoise(img, "Pepper", r,c,4)

plotnoise(img, "s&p", r,c,5)

plt.show()

**OUTPUT:**







**Image denoising**

**CODE:**

# Import the module and function

from skimage.util import random\_noise

import imageio as iio

img =iio.imread("peppers.png")

# Add noise to the image

noisy\_image = random\_noise(img)

# Show original and resulting image

plot\_comparison(img, noisy\_image, 'Noisy image')

**OUTPUT:**



**Total Variation Denoising Filter**

**CODE:**

from skimage.restoration import denoise\_tv\_chambolle

#Apply total variation filter denoising

denoised\_image = denoise\_tv\_chambolle(noisy\_image,weight=0.1,channel\_axis=-1)

#Show denoised image

plot\_comparison(noisy\_image,denoised\_image,'Denoised image')

**OUTPUT:**



**Bilateral Filter**

**CODE:**

from skimage.restoration import denoise\_bilateral

# Apply bilateral filter denoising

denoised\_image1 = denoise\_bilateral(noisy\_image, channel\_axis=-1)

# Show original and resulting images

plot\_comparison(noisy\_image,denoised\_image1,'Denoised image')

**OUTPUT:**



**Gaussian Filter**

**CODE:**

from skimage import data

from skimage import filters

from skimage import restoration

coins = data.coins()

gaussian\_image = filters.gaussian(coins, sigma=2)

plt.subplot(141)

plt.imshow(coins[10:80, 300:370], cmap='gray', interpolation='nearest')

plt.axis('off')

plt.title('Image')

plt.subplot(142)

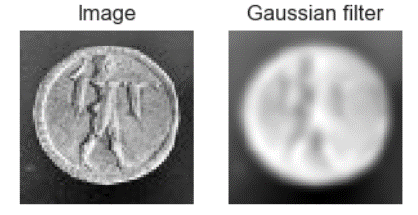
plt.imshow(gaussian\_image[10:80, 300:370], cmap='gray',interpolation='nearest')

plt.axis('off')

plt.title('Gaussian filter')

plt.show()

**OUTPUT:**



**Median Filter**

**CODE:**

coins = data.coins()

median\_image = filters.median(coins, np.ones((3, 3)))

plt.subplot(141)

plt.imshow(coins[10:80, 300:370], cmap='gray', interpolation='nearest')

plt.axis('off')

plt.title('Image')

plt.subplot(142)

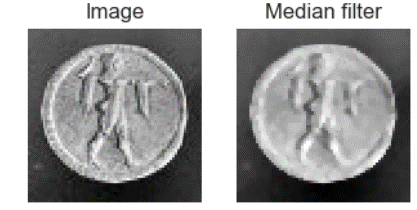
plt.imshow(median\_image[10:80, 300:370], cmap='gray',interpolation='nearest')

plt.axis('off')

plt.title('Median filter')

plt.show()

**OUTPUT:**



**Mean Filter**

**CODE:**

import cv2

coins = data.coins()

figure\_size = 9

new\_image = cv2.blur(coins[10:80, 300:370],(figure\_size, figure\_size))

plt.subplot(141), plt.imshow(coins[10:80, 300:370], cmap='gray'),plt.title('Original')

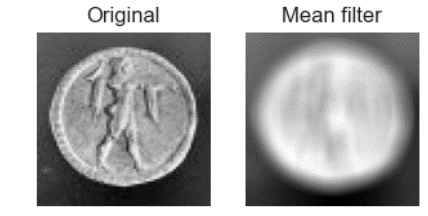
plt.xticks([]), plt.yticks([])

plt.subplot(142), plt.imshow(new\_image, cmap='gray'),plt.title('Mean filter')

plt.xticks([]), plt.yticks([])

plt.show()

**OUTPUT:**



**CODE:**

import cv2

plt.style.use('seaborn')

image = cv2.imread('peppers.png')

dst = cv2.fastNlMeansDenoisingColored(image, None, 11, 6, 7, 21)

row, col = 1, 2

fig, axs = plt.subplots(row, col, figsize=(15, 5))

fig.tight\_layout()

axs[0].imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

axs[0].set\_title('Original')

plt.axis('off')

axs[1].imshow(cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB))

axs[1].set\_title('Denoised Image')

plt.show()

**OUTPUT:**



**Vertical vs Horizontal Masks (Sobel Operator)**

**CODE:**

def plot\_comparison(original, filtered, title\_filtered):

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(8, 6), sharex=True, sharey=True)

ax1.imshow(original, cmap=plt.cm.gray)

ax1.set\_title('original')

ax1.axis('off')

ax2.imshow(filtered, cmap=plt.cm.gray)

ax2.set\_title(title\_filtered)

ax2.axis('off')

import cv2

import numpy as np

from matplotlib import pyplot as plt

img0 = cv2.imread('downloads/original.png',)

# converting to gray scale

gray = cv2.cvtColor(img0, cv2.COLOR\_BGR2GRAY)

# remove noise

img = cv2.GaussianBlur(gray,(3,3),0)

# convolute with proper kernels

sobelx = cv2.Sobel(img,cv2.CV\_64F,1,0,ksize=5) # x

sobely = cv2.Sobel(img,cv2.CV\_64F,0,1,ksize=5) # y

plt.subplot(3,1,1),plt.imshow(img,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(3,1,2),plt.imshow(sobelx,cmap = 'gray')

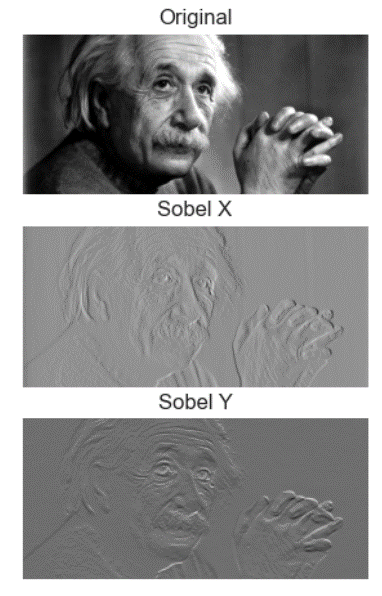
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])

plt.subplot(3,1,3),plt.imshow(sobely,cmap = 'gray')

plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()

**OUTPUT:**



**Laplacian Operator**

**CODE:**

coins = data.coins()

new\_image = cv2.Laplacian(coins[10:80, 300:370],cv2.CV\_64F)

plt.figure(figsize=(11,6))

plt.subplot(141), plt.imshow(coins[10:80, 300:370], cmap='gray'),plt.title('Original')

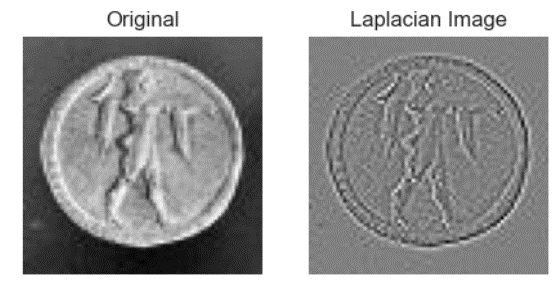
plt.axis('off')

plt.subplot(142), plt.imshow(new\_image, cmap='gray'),plt.title('Laplacian Image')

plt.axis('off')

plt.show()

**OUTPUT:**



**High Pass Filter**

**CODE:**

# High Pass Filtering is an edge detection operation

#read image

coins = data.coins()

#edge detection filter

kernel = np.array([[0.0, -1.0, 0.0],

[-1.0, 4.0, -1.0],

[0.0, -1.0, 0.0]])

kernel = kernel/(np.sum(kernel) if np.sum(kernel)!=0 else 1)

#filter the source image

img\_rst1 = cv2.filter2D(coins,-1,kernel)

plt.subplot(2,2,1),plt.imshow(coins,cmap = 'gray')

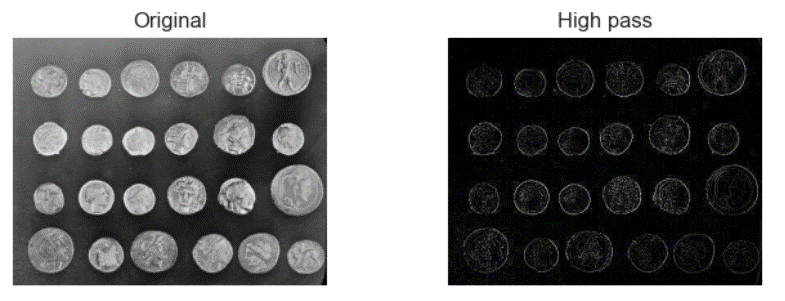
plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(img\_rst1,cmap = 'gray')

plt.title('High pass'), plt.xticks([]), plt.yticks([])

plt.show()

**OUTPUT:**



**Low Pass Filter**

**CODE:**

#read image

coins = data.coins()

#prepare the 5x5 shaped filter

kernel = np.array([[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1]])

kernel = kernel/sum(kernel)

#filter the source image

img\_rst = cv2.filter2D(coins,-1,kernel)

plt.subplot(2,2,1),plt.imshow(coins,cmap = 'gray')

plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(img\_rst,cmap = 'gray')

plt.title('Low pass'), plt.xticks([]), plt.yticks([])

plt.show()

**OUTPUT:**

