**FILTERS:**

**# Libraries**  
import cv2

import numpy as np

import matplotlib.pyplot as plt

**# Loading Image**

image = cv2.imread(r".\\sunset.png")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x, y = gray.shape[:2]

**Creating and Applying Low Pass Average Filter**

kernel = np.ones([3, 3], dtype = int)

kernel = kernel / 9

out1 = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)

out2 = cv2.blur(src=image, ksize=(3,3))  # 5x5, 7x7, 11x11

plt.imshow(image, 'gray')

plt.title('Input Image')

plt.show()

plt.imshow(out1, 'gray')

plt.title('Low Pass Average Filter')

plt.show()

plt.imshow(out2, 'gray')

plt.title('Low Pass Average Filter - In built function')

plt.show()

**# Low Pass Median Filter**

out = cv2.medianBlur(src=image, ksize = 3)  # 5, 7, 9, 11

plt.imshow(image, 'gray')

plt.title('Input Image')

plt.show()

plt.imshow(out, 'gray')

plt.title('Low Pass Median Filter')

plt.show()

**# High Pass Filter**

kernel = np.array([[-1/9, -1/9, -1/9],

                   [-1/9, 8/9, -1/9],

                   [-1/9, -1/9, -1/9]])

# another kernal without 1/9

out = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)

plt.imshow(image, 'gray')

plt.title('Input Image')

plt.show()

plt.imshow(out, 'gray')

plt.title('High Pass Filter')

plt.show()

**# High Boost Filter**

kernel = np.ones([3, 3], dtype = int)

kernel = kernel / 9

blur\_image = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)

out = cv2.addWeighted(image, 2, blur\_image, -1, 0) #

plt.imshow(image, 'gray')

plt.title('Input Image')

plt.show()

plt.imshow(out, 'gray')

plt.title('High Boost Filter')

plt.show()

**# NOISE:**

**# Adding Random Noise**

# Read Image

img = cv2.imread('lily.jpg') # Color image

# Convert the image to grayscale

img\_gray = img[:,:,1]

plt.imshow(img\_gray,'gray')

plt.title('Original Image')

plt.show()

# Genearte noise with same shape as that of the image

noise = np.random.normal(0, 50, img\_gray.shape)

# Add the noise to the image

img\_noised = img\_gray + noise

# Clip the pixel values to be between 0 and 255.

img\_noised = np.clip(img\_noised, 0, 255).astype(np.uint8)

plt.imshow(img\_noised,'gray')

plt.title('Noise Added Image')

plt.show()

**#Salt and Paper Noise**

# Read Image

img = cv2.imread('lily.jpg') # Color image

# Convert the image to grayscale

img\_gray = img[:,:,1]

plt.imshow(img\_gray,'gray')

plt.title('Original Image')

plt.show()

# Get the image size (number of pixels in the image).

img\_size = img\_gray.size

# Set the percentage of pixels that should contain noise

noise\_percentage = 0.1  # Setting to 10%

# Determine the size of the noise based on the noise precentage

noise\_size = int(noise\_percentage\*img\_size)

# Randomly select indices for adding noise.

random\_indices = np.random.choice(img\_size, noise\_size)

# Create a copy of the original image that serves as a template for the noised image.

img\_noised = img\_gray.copy()

# Create a noise list with random placements of min and max values of the image pixels.

noise = np.random.choice([img\_gray.min(), img\_gray.max()], noise\_size)

# Replace the values of the templated noised image at random indices with the noise, to obtain the final noised image.

img\_noised.flat[random\_indices] = noise

plt.imshow(img\_noised,'gray')

plt.title('Noise Added Image')

plt.show()

**# Gaussian Noise**

# Load the image

image = cv2.imread('.\lily.jpg')

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x,y = gray.shape[:2]

plt.imshow(gray,'gray')

plt.title('Original Image')

plt.show()

# Generate random Gaussian noise

mean = 0

stddev = 180

noise = np.zeros(img.shape, np.uint8)

cv2.randn(noise, mean, stddev)

# Add noise to image

noisy\_img = cv2.add(img, noise)

# Save noisy image

plt.imshow(noisy\_img,'gray')

plt.title('Noise Added Image')

plt.show()

**#Add impulse Noise and use Filter to remove it**

# Load the image

image = cv2.imread('.\lily.jpg')

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x,y = gray.shape[:2]

plt.imshow(gray,'gray')

plt.title('Original Image')

plt.show()

imp\_noise=np.zeros((x,y),dtype=np.uint8)

cv2.randu(imp\_noise,0,255)

imp\_noise=cv2.threshold(imp\_noise,245,255,cv2.THRESH\_BINARY)[1]

in\_img=cv2.add(gray,imp\_noise)

fig=plt.figure(dpi=200)

fig.add\_subplot(1,3,1)

plt.imshow(img,cmap='gray')

plt.axis("off")

plt.title("Original")

fig.add\_subplot(1,3,2)

plt.imshow(imp\_noise,cmap='gray')

plt.axis("off")

plt.title("Impulse Noise")

fig.add\_subplot(1,3,3)

plt.imshow(in\_img,cmap='gray')

plt.axis("off")

plt.title("Combined")

**HARRIS Corner Detection**

import cv2

import matplotlib.pyplot as plt

import os

import sys

import numpy as np

import scipy.ndimage.filters as filters

def h\_fun(img, kernel\_size=3):

    """Calculates Harris operator array for every pixel"""

    Ix = cv2.Sobel(img, cv2.CV\_32F, 1, 0, ksize=kernel\_size)

    Iy = cv2.Sobel(img, cv2.CV\_32F, 0, 1, ksize=kernel\_size)

    Ix\_square = Ix \* Ix

    Iy\_square = Iy \* Iy

    Ixy = Ix \* Iy

    Ix\_square\_blur = cv2.GaussianBlur(Ix\_square, (kernel\_size, kernel\_size), 0)

    Iy\_square\_blur = cv2.GaussianBlur(Iy\_square, (kernel\_size, kernel\_size), 0)

    Ixy\_blur = cv2.GaussianBlur(Ixy, (kernel\_size, kernel\_size), 0)

    det = Ix\_square\_blur \* Iy\_square\_blur - Ixy\_blur \* Ixy\_blur

    trace = Ix\_square\_blur + Iy\_square\_blur

    k = 0.05

    h = det - k\*trace\*trace

    h = h / np.max(h)

    return h

def find\_max(image, size, threshold):

    """Finds maximum of array"""

    data\_max = filters.maximum\_filter(image, size)

    maxima = (image == data\_max)

    diff = (image > threshold)

    maxima[diff == 0] = 0

    return np.nonzero(maxima)

def draw\_points(img, points):

    plt.figure()

    plt.imshow(img)

    plt.plot(points[1], points[0], '\*', color='r')

    # plt.show()

**#Input Image**

IMG\_NAME1 = 'house5.jpg'

img1\_color = cv2.imread(IMG\_NAME1)

img1\_color = cv2.cvtColor(img1\_color, cv2.COLOR\_BGR2RGB)

img1 = cv2.imread(IMG\_NAME1, cv2.IMREAD\_GRAYSCALE)

**#Just Change Kernel Size and Threshold**

KERNEL\_SIZE = 3

THRESHOLD = 0.1

# Find maximums

h1 = h\_fun(img1, KERNEL\_SIZE)

m1 = find\_max(h1, KERNEL\_SIZE, THRESHOLD)

draw\_points(img1\_color, m1)

plt.show()

**Prewitt**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread(r"C:\Users\HP\Downloads\olympic.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

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

plt.title('Input Image')

plt.show()

kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])

# img2= cv2.GaussianBlur(gray,(5,5),0)#gaussian Image

img\_prewittx = cv2.filter2D(gray, -1, kernelx) # Horizontal

img\_prewitty = cv2.filter2D(gray, -1, kernely) # Vertical

img\_prewitt = img\_prewittx + img\_prewitty

plt.imshow(img\_prewittx, cmap='gray')

plt.title('Prewitt Horizontal Edge Kernel')

plt.show()

plt.imshow(img\_prewitty, cmap='gray')

plt.title('Prewitt Vertical Edge Kernel')

plt.show()

plt.imshow(img\_prewitt, cmap='gray')

plt.title('Prewitt Both Edges Kernel')

plt.show()

**Sobel –**

# Sobel Edge Detector Operator

image = cv2.imread(r"C:\Users\HP\Downloads\olympic.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

plt.imshow(gray, 'gray')

plt.title('Input Image')

plt.show()

img2= cv2.GaussianBlur(gray,(5,5),0)#gaussian Image

img\_sobelx = cv2.Sobel(img2,cv2.CV\_8U,0,1,ksize=3)

img\_sobely = cv2.Sobel(img2,cv2.CV\_8U,1,0,ksize=3)

img\_sobel = img\_sobelx + img\_sobely

plt.imshow(img\_sobelx, 'gray')

plt.title('Sobel Horizontal Edge Kernel')

plt.show()

plt.imshow(img\_sobely, 'gray')

plt.title('Sobel Vertical Edge Kernel')

plt.show()

plt.imshow(img\_sobel, 'gray')

plt.title('Sobel Both Edges Kernel')

plt.show()

**Robert**

# robert Operator

image = cv2.imread(r"C:\Users\HP\Downloads\roman.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

plt.imshow(gray, 'gray')

plt.title('Input Image')

plt.show()

kernel\_Roberts\_x = np.array([[1, 0],[0, -1]])

kernel\_Roberts\_y = np.array([[0, -1],[1, 0]])

img2= cv2.GaussianBlur(gray,(5,5),0)#gaussian Image

x = cv2.filter2D(img2, cv2.CV\_16S, kernel\_Roberts\_x)

y = cv2.filter2D(img2, cv2.CV\_16S, kernel\_Roberts\_y)

absX = cv2.convertScaleAbs(x)

absY = cv2.convertScaleAbs(y)

roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

plt.imshow(roberts, 'gray')

plt.title('roberts Kernel')

plt.show()

**ORB**

import cv2

import matplotlib.pyplot as plt

# Load and convert the image to grayscale

image = cv2.imread(r"C:\Users\HP\Downloads\olympic.jpeg")

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

# Initialize ORB detector

orb = cv2.ORB\_create()

# Detect keypoints and compute descriptors

keypoints, descriptors = orb.detectAndCompute(gray, None)

# Draw the keypoints on the image

img\_with\_keypoints = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0), flags=0)

# Display the image with keypoints

plt.imshow(cv2.cvtColor(img\_with\_keypoints, cv2.COLOR\_BGR2RGB))

plt.title('ORB Keypoints')

plt.show()

**SURF**

import cv2

import matplotlib.pyplot as plt

# Load and convert the image to grayscale

image = cv2.imread(r"C:\Users\HP\Downloads\olympic.jpeg")

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

# Initialize the SURF detector

surf = cv2.xfeatures2d.SURF\_create(400) # 400 is the Hessian threshold

# Detect keypoints and compute descriptors

keypoints, descriptors = surf.detectAndCompute(gray, None)

# Draw the keypoints on the image

img\_with\_keypoints = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0), flags=0)

# Display the image with keypoints

plt.imshow(cv2.cvtColor(img\_with\_keypoints, cv2.COLOR\_BGR2RGB))

plt.title('SURF Keypoints')

plt.show()

**# Canny Edge Detector**

from skimage import feature, exposure

# reads an input image

img = cv2.imread(r"C:\Users\hp\Downloads\temple.jpeg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)                     #BGR 2 RGB for plotting using matplotlib

print("input image dimensions", gray.shape)

width = 100

height = 100

dim = (width, height)

# resize image

gray = cv2.resize(gray, dim)

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

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.title('Input Image')

plt.show()

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

plt.imshow(blurred, cmap = 'gray')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.title('Blurred Image')

plt.show()

**# Canny with Aperature sie**

image = cv2.imread(r"C:\Users\hp\Downloads\temple.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x,y = gray.shape[:2]

plt.imshow(gray, 'gray')

plt.title('Input Image')

plt.show()

# Setting All parameters

t\_lower = 100  # Lower Threshold

t\_upper = 200  # Upper threshold

aperture\_size = 5  # Aperture size

# Applying the Canny Edge filter

# with Custom Aperture Size

edge = cv2.Canny(gray, t\_lower, t\_upper, apertureSize=aperture\_size)

# Convert the image data to a floating-point format

#edge = edge.astype(np.int8)

plt.imshow(edge, 'gray')

plt.title('Input Image')

plt.show()

**# Canny with L2 Grandient**

image = cv2.imread(r"C:\Users\hp\Downloads\temple.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x,y = gray.shape[:2]

plt.imshow(gray, 'gray')

plt.title('Input Image')

plt.show()

t\_lower = 100 # Lower Threshold

t\_upper = 200 # Upper threshold

aperture\_size = 5 # Aperture size

L2Gradient = True # Boolean

# Applying the Canny Edge filter with L2Gradient = True

edge = cv2.Canny(gray, t\_lower, t\_upper, L2gradient = L2Gradient )

plt.imshow(edge, 'gray')

plt.title('Input Image')

plt.show()

**### Canny wiht aperature size and L2Gradient**

image = cv2.imread(r"C:\Users\hp\Downloads\temple.jpeg")

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

x,y = gray.shape[:2]

plt.imshow(gray, 'gray')

plt.title('Input Image')

plt.show()

# Defining all the parameters

t\_lower = 100 # Lower Threshold

t\_upper = 200 # Upper threshold

aperture\_size = 5 # Aperture size

L2Gradient = True # Boolean

# Applying the Canny Edge filter

# with Aperture Size and L2Gradient

edge = cv2.Canny(gray, t\_lower, t\_upper,

                 apertureSize = aperture\_size,

                 L2gradient = L2Gradient )

plt.imshow(edge, 'gray')

plt.title('Input Image')

plt.show()

**SEGMENTATION**

**## 1. Watershed Segmentation**

import cv2

import matplotlib.pyplot as plt

import numpy as np

from skimage import exposure

**# Read Image**

# reads an input image

img = cv2.imread("C:\\Users\\kambl\\ComputerVision\_Lab\\Sample Images\\coins.jpg")

plt.imshow(img,'gray')

plt.title('Original Image')

plt.show()

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

plt.imshow(gray,'gray')

plt.title('Gray Image')

plt.show()

# Applying Thresholding

#Threshold Processing

ret, bin\_img = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

plt.imshow(bin\_img,'gray')

plt.title('Thresholded Binary Image')

plt.show()

**# Noise Removal- Morphological Gradient Processing**

# noise removal

#kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))

kernel = np.ones((3,3),np.uint8)

bin\_img = cv2.morphologyEx(bin\_img, cv2.MORPH\_OPEN, kernel, iterations=1)

plt.imshow(bin\_img,'gray')

plt.title('Noise Removal Binary Image')

plt.show()

**# Black background and foreground of the image**

# Create subplots with 4 row and 1 columns

fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(8, 8))

# sure background area

sure\_bg = cv2.dilate(bin\_img, kernel, iterations=3)

axes[0].imshow(sure\_bg, 'gray')

axes[0].set\_title('Sure Background')

# Distance transform

dist = cv2.distanceTransform(bin\_img, cv2.DIST\_L2, 5)

axes[1].imshow(dist, 'gray')

axes[1].set\_title('Distance Transform')

#foreground area

ret, sure\_fg = cv2.threshold(dist, 0.5 \* dist.max(), 255, cv2.THRESH\_BINARY)

sure\_fg = sure\_fg.astype(np.uint8)

axes[2].imshow (sure\_fg, 'gray')

axes[2].set\_title('Sure Foreground')

# unknown area

unknown = cv2.subtract(sure\_bg, sure\_fg)

axes[3].imshow(unknown,  'gray')

axes[3].set\_title('Unknown')

plt.tight\_layout()

plt.show()

# **Place markers on local minima**

# Marker labelling

# sure foreground

ret, markers = cv2.connectedComponents(sure\_fg)

# Add one to all labels so that background is not 0, but 1

markers += 1

# mark the region of unknown with zero

markers[unknown == 255] = 0

fig, ax = plt.subplots(figsize=(6, 6))

ax.imshow(markers, cmap="tab20b")

ax.axis('off')

plt.show()

# **Apply Watershed Algorithm to Markers**

# watershed Algorithm

markers = cv2.watershed(img, markers)

fig, ax = plt.subplots(figsize=(5, 5))

ax.imshow(markers, cmap="tab20b")

ax.axis('off')

plt.show()

labels = np.unique(markers)

coins = []

for label in labels[2:]:

# Create a binary image in which only the area of the label is in the foreground

#and the rest of the image is in the background

    target = np.where(markers == label, 255, 0).astype(np.uint8)

  # Perform contour extraction on the created binary image

    contours, hierarchy = cv2.findContours(target, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE )

    coins.append(contours[0])

print("Number of coins: ", len(coins))

# Draw the outline

img = cv2.drawContours(img, coins, -1, color=(0, 0, 255), thickness=2)

fig, ax = plt.subplots(figsize=(5, 5))

ax.imshow(img)

ax.axis('off')

plt.show()