LAB 1A:operations on image:

```
import cv2
img= cv2.imread('/Users/pandu/Documents/PCV/PCV Lab/Pycharm Projects/download1.jpeg')
print(img.shape)
print(img[0][0])
print(img.dtype)
gray_img= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2.imshow('gray image',gray_img)
print(gray_img)
resized_img= cv2.resize(gray_img,(500,500))
print(resized_img.shape)
img[0:150,0:300]=[255,0,0]
cv2.imshow('color_change',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

LAB 1B: PROPERTIES OF A VIDEO:

```
import cv2
import numpy as np
cap = cv2.VideoCapture("/Users/pandu/Documents/PCV/PCV Lab/Pycharm Projects/video (2160p).mp4")
print("width of frames:'{}'".format(cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
print("height of frames:'{}' ".format(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
print("frames per second:'{}'".format(cap.get(cv2.CAP_PROP_FPS)))
print("frame count:'{}'".format(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
print("current frame number:'{}'".format(cap.get(cv2.CAP_PROP_POS_FRAMES)))

while cap.isOpened():
 ret, frame = cap.read()
 if ret:
   cv2.imshow('video (2160p).mp4', frame)
   if cv2.waitKey(2555) & 0xFF == ord('q'):
    break
   else:
    break
cap.release()
cv2.destroyAllWindows()
```

LAB 2A:IMAGE NEGATION:

```
#ngeation of image

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

# Read an image
image = cv2.imread("C:/Users/MAHE/Desktop/nature-images..jpg")
```

```
# Apply log transformation method


negative_image = 255 - image
Hori = np.concatenate((image, negative_image), axis=1)

plt.imshow(Hori)
plt.show()
```

LAB 2B:LOG TRANFORMATION:

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

# Read an image
image = cv2.imread("C:/Users/MAHE/Desktop/nature-images..jpg")

i =0

c = (255)/ np.log(1 + np.max(image))

log_image = c * (np.log(image + 1))

    # Specify the data type so that
    # float value will be converted to int

log_image = np.array(log_image, dtype = np.uint8)
Hori = np.concatenate((image, log_image), axis=1)

    # Display both images

plt.imshow(Hori)
plt.show()
```

LAB 3A: linear piece transformation:

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

img = cv2.imread("C:/Users/MAHE/Desktop/Screenshot 2023-08-29 103207.png",0)

# To ascertain total numbers of
# rows and columns of the image,
# size of the image
```

```
m,n = img.shape

# To find the maximum grey level
# value in the image
L = img.max()

T1 = 130
T2 = 180
T = 150

img_thresh_back = np.zeros((m,n), dtype = int)
for i in range(m):

 for j in range(n):

  if T1 < img[i,j] < T2:
   img_thresh_back[i,j]= 255
  else:
   img_thresh_back[i,j] = img[i,j]

cv2.imwrite(’Cameraman_Thresh_Back.png’, img_thresh_back)
```

LAB 3B: HISTOGRAM EQUALISATION:

```
# import Opencv
import cv2

# import Numpy
import numpy as np

# read a image using imread
img = cv2.imread("C:/Users/MAHE/Desktop/Screenshot 2023-08-29 103207.png", 0)

m,n = img.shape
L = img.max()
arr = [0]*256
for i in range(m):
    for j in range(n):
        arr[img[i,j]] = arr[img[i,j]] + 1

print(arr)

equ = cv2.equalizeHist(img)
cv2.imwrite(’Cameraman_equalized.png’, equ)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

LAB4:low, light pass filters:

```
import cv2
import numpy as np
import random
# mean filter
img_m = cv2.imread("C:/Users/MAHE/Desktop/Screenshot 2023-08-29 103207.png",0)
kernel = np.ones((5,5),np.float32)/25
mean_filtered = cv2.filter2D(img_m,-1,kernel)
cv2.imwrite(’original_image.png’,img_m)
cv2.imwrite(’mean_filtered.png’,mean_filtered)
#adding salt and pepper noise
def add_noise(img):

    # Getting the dimensions of the image
    row , col = img.shape

    # Randomly pick some pixels in the
    # image for coloring them white
    # Pick a random number between 300 and 10000
    number_of_pixels = random.randint(300, 10000)
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord=random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord=random.randint(0, col - 1)

        # Color that pixel to white
        img[y_coord][x_coord] = 255

    # Randomly pick some pixels in
    # the image for coloring them black
    # Pick a random number between 300 and 10000
    number_of_pixels = random.randint(300 , 10000)
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord=random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord=random.randint(0, col - 1)

        # Color that pixel to black
        img[y_coord][x_coord] = 0

    return img
#median_filter

img_med = add_noise(img_m)
cv2.imwrite(’noised_image.png’,img_med)
median_filtered = cv2.medianBlur(img_med,3)
cv2.imwrite(’medain_filtered.png’,median_filtered)
img_l = cv2.imread("C:/Users/MAHE/Desktop/Screenshot 2023-08-29 103207.png",0)
img_lap = cv2.Laplacian(img_l,-1)
cv2.imwrite(’laplacian_filtered.png’,img_lap)
```

LAB5:SOBEL AND CANNY EDGE DETECTION:

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread(’C:/Users/MAHE/Downloads/OIP (1).jpg’)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 0)

sobel_x = cv2.Sobel(blur, cv2.CV_8U, 1, 0, ksize=3)
sobel_y = cv2.Sobel(blur, cv2.CV_8U, 0, 1, ksize=3)

edges_1 = cv2.bitwise_or(sobel_x, sobel_y)
edges = cv2.Canny(edges_1, 315, 415)

cv2.imshow(’Original Image’, img)
cv2.imshow(’Edge-Detected Image’, edges)
cv2.imshow(’Sobel horizontal’, sobel_x)
cv2.imshow(’Sobel vertical’, sobel_y)

cv2.imwrite(’output.jpg’, edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

LAB6:GEOMETRIC TRANFORMATION:

```
import cv2 as cv
img=cv.imread(’/content/square.jpeg’)
cv.imwrite(’square.png’,img)

angle=60
h,w=img.shape[:2]
M=cv.getRotationMatrix2D((w/2,h/2),angle,1)
rotated_img=cv.warpAffine(img,M,(w,h))
cv.imwrite(’rotated.png’,rotated_img)

h,w=img.shape[:2]
shrink_img=cv.resize(img,(80,80),interpolation=cv.INTER_AREA)
enlarge_img=cv.resize(img,None,fx=5,fy=5,interpolation=cv.INTER_CUBIC)
cv.imwrite(’shrinked.png’,shrink_img)
cv.imwrite(’enlarge.png’,enlarge_img)

import numpy as np
h,w=img.shape[:2]
matrix=np.float32([[1,0,25],[0,1,25]])
translated_img=cv.warpAffine(img,matrix,(h,w))
cv.imwrite(’translated.png’,translated_img)
```

```
h,w=img.shape[:2]
matrix2=np.float32([[1,0.5,0],[0,1,0],[0,0,1]])
sheard_img=cv.warpPerspective(img,matrix2,(int(w*1.5),int(h*1)))
cv.imwrite(’sheared.png’,sheard_img)
```

LAB7: SIFT ALGORITHM

```
import cv2 as cv
import matplotlib.pyplot as plt

img1 = cv.imread("download1.jpeg")
img2 = cv.imread("download2.jpeg")

img_1 = cv.cvtColor(img1,cv.COLOR_BGR2GRAY)
img_2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
m,n = img_1.shape
img_2 = cv.resize(img2,[m,n])

m,n = img_1.shape
sift = cv.SIFT_create()

keypoints1,descriptors1 = sift.detectAndCompute(img_1,None)
keypoints2,descriptors2 = sift.detectAndCompute(img_2,None)

bf = cv.BFMatcher(cv.NORM_L1,crossCheck = True)

matches = bf.match(descriptors1,descriptors2)

matches = sorted(matches,key = lambda x:x.distance)

matched_img = cv.drawMatches(img1,keypoints1,img2,keypoints2,matches,img2,flags = 2)

cv.imwrite("matched.png",matched_img)
print(’total points’,(len(keypoints1)+len(keypoints2))/2)
print(’matched key points’,len(matches))
print(’accuracy’,len(matches)/((len(keypoints1)+len(keypoints2))/2))
```

LAB8:FACE DETECTION:

```
import cv2

# Load the pre-trained face detection model
face_cascade = cv2.CascadeClassifier(’haarcascade_frontalface_default.xml’)

# Read the input image or stream from a camera
cap = cv2.VideoCapture(0)  # Use default camera

while True:
    ret, frame = cap.read()  # Capture frame by frame

    if not ret:
        break

    # Convert the input image to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect the faces in the image
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

    # Draw a rectangle around each detected face
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the output image with detected faces
    cv2.imshow('Face Detection', frame)

    # Wait for user input to exit or continue processing
    if cv2.waitKey(1) == ord('q'):
        break

# Release the resources and close the windows
cap.release()
cv2.destroyAllWindows()

--------------------------------------------------------------------------
```

LAB9: Intensity transformations

```
#image neagation

import cv2 as cv
img = cv.imread()
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

neg = cv.bitwise_not(gray)
cv.imshow('real',img)
cv.imshow('negative',neg)

#log transformation
import numpy as np

c = 20
log = c*np.log(1+gray)
#adding two images side by side
result = cv.hconcat([img,log])
cv.imshow('result',result)

--------------------------------------------------------------------------
```

LAB10:Image Deionising

```
import cv2 as cv
img = cv.imread()
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
```

```
#Define kernel matrix
kernel = np.ones((5, 5), np.float32) / 25

#Apply mean filtered

mean = cv.filter2D(img,-1,kernel)

#Apply median filter

median = cv.medianBlur(img,5)

#apply laplacian filter

laplacian = cv.Laplacian(img,cv.CV_64F)
or
laplacian = cv.Laplacaian(img,-1)

cv.imshow(’mean’,mean)
cv.imshow(’median’,median)
cv.imshow(’laplacian’,laplacian)
```

LAB11:KLT FEATURE TRACKER

```
import cv2
import numpy as np


cap = cv2.VideoCapture(r’video (2160p) (2).mp4’)  # Replace ’input_video.mp4’ with the path to your video file

feature_params = dict(maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

lk_params = dict(winSize=(15, 15), maxLevel=2, criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

mask = np.zeros_like(old_frame)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate optical flow using Lucas-Kanade algorithm
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # Draw the tracks
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel().astype(int)
        c, d = old.ravel().astype(int)
        mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)
        frame = cv2.circle(frame, (a, b), 5, (0, 255, 0), -1)

    img = cv2.add(frame, mask)
    cv2.imshow(’KLT Feature Tracking’, img)

    if cv2.waitKey(1) & 0xFF == ord(’q’):
        break

    # Update previous frame and points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

cv2.destroyAllWindows()
cap.release()
```

LAB12:TRACKING MOVING OBJECTS

```
import cv2
cap = cv2.VideoCapture("/Users/pandu/Documents/PCV/PCV Lab/Pycharm Projects/video (2160p) (2).mp4")
ret, frame = cap.read()

x,y,w,h = cv2.selectROI(frame)

track_window = (x, y, w, h)

roi = frame[y:y+h, x:x+w]

hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

roi_hist = cv2.calcHist([hsv_roi], [0], None, [180], [0, 180])

cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)

term_criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)

while True:
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to the HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Calculate the back projection of the histogram
    dst = cv2.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)

    # Apply mean shift to get the new location
    ret, track_window = cv2.meanShift(dst, track_window, term_criteria)

    # Draw the new tracking window on the frame
    x, y, w, h = track_window
    img2 = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow(’Mean Shift Object Tracking’, img2)

    # Press ’q’ to exit the video
    if cv2.waitKey(1) & 0xFF == ord(’q’):
        break

cap.release()
cv2.destroyAllWindows()
```

LAB13:CAR DETECTION

```
import cv2

haar_cascade = ’cars.xml’
video = ’cars.mp4’
cap = cv2.VideoCapture(video)
car_cascade = cv2.CascadeClassifier(haar_cascade)

# Set the desired width and height for the resized video
width = 640
height = 480

while True:
    # reads frames from a video
    ret, frames = cap.read()
    if not ret:
        break  # Break the loop if the video ends

    # Resize the frame
    frames = cv2.resize(frames, (width, height))

    gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)

    # Detects cars of different sizes in the input image
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)

    # To draw a rectangle in each car
    for (x, y, w, h) in cars:
        cv2.rectangle(frames, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # Display frames in a window
    cv2.imshow(’video’, frames)

    # Wait for Esc key to stop
    if cv2.waitKey(33) == 27:
        break

# Release the video capture object and close the window
cap.release()
cv2.destroyAllWindows()
```
