hough tranform

```
#@title hough tranform
import cv2
import numpy as np

img = cv2.imread('line_hough.jpg')

sob_8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=3)

sobel_64 = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)
abs_64 = np.absolute(sobel_64)
sobel_8u = np.uint8(abs_64)

cv2.imshow('a1',sobel_8u)
cv2.waitKey(0)

img = sobel_8u
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
sk = cv2.ximgproc.thinning(th, None, 1)

k1 = np.array(([0, 0, 0], [-1, 1, -1], [-1, -1, -1]), dtype="int")
k2 = np.array(([0, -1, -1], [0, 1, -1], [0, -1, -1]), dtype="int")
k3 = np.array(([-1, -1, 0],  [-1, 1, 0], [-1, -1, 0]), dtype="int")
k4 = np.array(([-1, -1, -1], [-1, 1, -1], [0, 0, 0]), dtype="int")

k5 = np.array(([-1, -1, -1], [-1, 1, -1], [0, -1, -1]), dtype="int")
k6 = np.array(([-1, -1, -1], [-1, 1, -1], [-1, -1, 0]), dtype="int")
k7 = np.array(([-1, -1, 0], [-1, 1, -1], [-1, -1, -1]), dtype="int")
k8 = np.array(([0, -1, -1], [-1, 1, -1], [-1, -1, -1]), dtype="int")

o1 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k1)
o2 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k2)
o3 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k3)
o4 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k4)
out1 = o1 + o2 + o3 + o4

o5 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k5)
o6 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k6)
o7 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k7)
o8 = cv2.morphologyEx(sk, cv2.MORPH_HITMISS, k8)
out2 = o5 + o6 + o7 + o8

out = cv2.add(out1, out2)
pts = np.argwhere(out == 255)
loose_ends = img.copy()
for pt in pts:
    loose_ends = cv2.circle(loose_ends, (pt[1], pt[0]), 3, (0,255,0), -1)
pts = list(map(tuple, pts))
final = img.copy()

for i, pt1 in enumerate(pts):
  min_dist = max(img.shape[:2])
  sub_pts = pts.copy()
  del sub_pts[i]
```

```
  for pt2 in sub_pts:
    dist = int(np.linalg.norm(np.array(pt1) - np.array(pt2)))
    if dist < min_dist:
      min_dist = dist
      pt_2 = pt2
  final = cv2.line(final, (pt1[1], pt1[0]), (pt_2[1], pt_2[0]), (0, 0, 255), thickness = 2)

cv2.imshow('Final Image',final)
cv2.waitKey(0)
```

K means algo

```
#@title K means algo
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread("seg3.tif")

img2 = img.reshape((-1,3))

img2 = np.float32(img2)


criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

k = 3

attempts = 3


ret,label,center=cv2.kmeans(img2, k, None, criteria, attempts, cv2.KMEANS_PP_CENTERS)

center = np.uint8(center)

res = center[label.flatten()]
res2 = res.reshape((img.shape))
cv2.imwrite("segmented.jpg", res2)


titles = ['original image', 'segmentation image']
images = [img, res2]

for i in range(2):
    plt.subplot(1, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```

Segmentation color

```
#@title Segmentation color

from skimage import io, measure
```

```
import cv2
from matplotlib import pyplot as plt
import numpy as np
from scipy import ndimage as nd
from skimage import img_as_ubyte

img = io.imread('color.png')
plt.imshow(img)

#In the input image, we have bunch of colors, we segment image on pixel based on the specific color
hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
#hsv demension is same as input image

mask = cv2.inRange(hsv, (100, 90, 90), (120, 255, 255))
#at center dark blue, at edge -light blue H is 100 to 120
#sat - 90 to 255 and V is from 90 to 2355
#mask =cv2.inRange(hsv, (0,0,100), (180, 70, 255)) for white color

plt.imshow(mask)

#in the output i have some holes - to close it i use binary closing operation
#(dilation followed by erosion)
closed_mask = nd.binary_closing(mask, np.ones((5,5)))
plt.imshow(closed_mask)

#after segmentation each object given a unique label value
label_image = measure.label(closed_mask)
plt.imshow(label_image)


from skimage.color import label2rgb
image_label_overlay = label2rgb(label_image, image=img)
#it takes the original image and overlay the label image on top of that in RGB
#in the input image the blue balls are in different color and all other colors of ball are black and

plt.imshow(image_label_overlay)
img_as_8byte = img_as_ubyte(image_label_overlay)

#To calculate image properties (area, centroid, convex_area, label, intensity image, major axis leng
#import this data as pandas-compatible table.

props = measure.regionprops_table(img_as_8byte, img, properties=['label', 'area'])

import pandas as pd
df = pd.DataFrame(props)
print(df.head())
```

Watershed algo

```
#@title Watershed algo
import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage import color
```

```
from skimage import color

img1 = cv2.imread("watershed.jpg")
img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)


#Threshold image to binary using OTSU. ALl thresholded pixels will be set to 255
ret1, thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

cv2.imshow('Threshold', thresh)
cv2.waitKey(0)

print(ret1)

# Morphological operations to remove small noise - opening
#To remove holes we can use closing
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 1)

cv2.imshow('opening', opening)
cv2.waitKey(0)


from skimage.segmentation import clear_border
opening = clear_border(opening) #Remove edge touching objects

cv2.imshow('opening', opening)
cv2.waitKey(0)

# finding sure background by dilation
sure_bg = cv2.dilate(opening,kernel,iterations=2)

cv2.imshow('sure_bg', sure_bg)
cv2.waitKey(0)

# Finding sure foreground area using distance transform and thresholding
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,3)

cv2.imshow('dist_transform', dist_transform)
cv2.waitKey(0)


print(dist_transform.max()) #gives about 38.200073
ret2, sure_fg = cv2.threshold(dist_transform,0.001*dist_transform.max(),255,0)

cv2.imshow('sure_fg', sure_fg)
cv2.waitKey(0)

# Unknown ambiguous region is nothing but bkground - foreground
sure_fg = np.uint8(sure_fg)

unknown = cv2.subtract(sure_bg,sure_fg)

cv2.imshow('unknow', unknown)
cv2.waitKey(0)


#Now we create a marker and label the regions inside.
# For sure regions, both foreground and background will be labeled with positive numbers.
# Unknown regions will be labeled 0.
```

```
ret3, markers = cv2.connectedComponents(sure_fg)

#So let us add 10 to all labels so that sure background is not 0, but 10
markers = markers+10

# Now, mark the region of unknown with zero
markers[unknown==255] = 0   #if my unknow is 255 then my marker is 0
plt.imshow(markers, cmap='jet')

#Now we are ready for watershed filling.
markers = cv2.watershed(img1,markers)

#boundaries to -1 after watershed.
img1[markers == -1] = [255,0,255]

img2 = color.label2rgb(markers, bg_label=0)

cv2.imshow('Overlay on original image', img1)
cv2.imshow('Colored', img2)
cv2.waitKey(0)
```

image Segmentation SEGHE

```
#@title image Segmentation SEGHE

from skimage import io
from matplotlib import pyplot as plt
import numpy as np

img = io.imread("noise.png",0)


from skimage.restoration import denoise_nl_means, estimate_sigma
from skimage import img_as_ubyte, img_as_float


float_img = img_as_float(img)
sigma_est = np.mean(estimate_sigma(float_img, multichannel=True))


denoise_img = denoise_nl_means(float_img, h=15 * sigma_est, fast_mode=False,
                               patch_size=5, patch_distance=3, multichannel=True)

denoise_img_as_8byte = img_as_ubyte(denoise_img)

segm1 = (denoise_img_as_8byte <= 25)
segm2 = (denoise_img_as_8byte > 25) & (denoise_img_as_8byte <= 75)
segm3 = (denoise_img_as_8byte > 75) & (denoise_img_as_8byte <= 125)
segm4 = (denoise_img_as_8byte > 125)


all_segments = np.zeros((denoise_img_as_8byte.shape[0], denoise_img_as_8byte.shape[1], 3)) #nothing


all_segments[segm1] = (1,0,0)
all_segments[segm2] = (0,1,0)
all_segments[segm3] = (0,0,1)
```

```
all_segments[segm4] = (1,1,0)
plt.imshow(all_segments)


from scipy import ndimage as nd

segm1_opened = nd.binary_opening(segm1, np.ones((3,3)))
segm1_closed = nd.binary_closing(segm1_opened, np.ones((3,3)))

segm2_opened = nd.binary_opening(segm2, np.ones((3,3)))
segm2_closed = nd.binary_closing(segm2_opened, np.ones((3,3)))

segm3_opened = nd.binary_opening(segm3, np.ones((3,3)))
segm3_closed = nd.binary_closing(segm3_opened, np.ones((3,3)))

segm4_opened = nd.binary_opening(segm4, np.ones((3,3)))
segm4_closed = nd.binary_closing(segm4_opened, np.ones((3,3)))

all_segments_cleaned = np.zeros((denoise_img_as_8byte.shape[0], denoise_img_as_8byte.shape[1], 3)) #

all_segments_cleaned[segm1_closed] = (0,0,1)
all_segments_cleaned[segm2_closed] = (0,1,0)
all_segments_cleaned[segm3_closed] = (1,0,0)
all_segments_cleaned[segm4_closed] = (1,1,0)

plt.imshow(all_segments_cleaned)
```

fft

```
#@title fft

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

x = np.arange(256)
y = np.sin(2 * np.pi * x / 4)
y += max(y)
img = np.array([[y[j]*127 for j in range(256)] for i in range(256)], dtype=np.uint8) # create 2-D ar

plt.imshow(img)

img = cv2.imread('color.png', 0) # load an image

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

dft_shift = np.fft.fftshift(dft)

magnitude_spectrum = 20 * np.log((cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))+1)
fig = plt.figure(figsize=(12, 12))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(img)
ax1.title.set_text('Input Image')
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(magnitude_spectrum)
ax2.title.set_text('FFT of image')
plt.show()
```

image ft

```
#@title image ft
import cv2
from matplotlib import pyplot as plt
import numpy as np


img = cv2.imread('color.png', 0)
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))

rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.ones((rows, cols, 2), np.uint8)
r = 50
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 0


"""

rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.zeros((rows, cols, 2), np.uint8)
r = 200
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 1
"""
# Band Pass Filter - Concentric circle mask, only the points living in concentric circle are ones
"""rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.zeros((rows, cols, 2), np.uint8)
r_out = 80
r_in = 10
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = np.logical_and(((x - center[0]) ** 2 + (y - center[1]) ** 2 >= r_in ** 2),
                           ((x - center[0]) ** 2 + (y - center[1]) ** 2 <= r_out ** 2))
mask[mask_area] = 1"""


fshift = dft_shift * mask

fshift_mask_mag = 20 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]))


f_ishift = np.fft.ifftshift(fshift)

img_back = cv2.idft(f_ishift)

img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
```

```
fig = plt.figure(figsize=(12, 12))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(img, cmap='gray')
ax1.title.set_text('Input Image')
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(magnitude_spectrum, cmap='gray')
ax2.title.set_text('FFT of image')
ax3 = fig.add_subplot(2,2,3)
ax3.imshow(fshift_mask_mag, cmap='gray')
ax3.title.set_text('FFT + Mask')
ax4 = fig.add_subplot(2,2,4)
ax4.imshow(img_back, cmap='gray')
ax4.title.set_text('After inverse FFT')
plt.show()
```

face

```
#@title face
import cv2
import face_recognition
from simple_facerec import SimpleFacerec

img=cv2.imread('modi.png')

#rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

cv2.imshow("Image", img)

#cv2.imshow("Image", rgb_img)

cv2.waitKey(0)


#First step for Face recongnition is to encode the image.

#encode the imported image and comapre with the other images in the folder


img_encoding = face_recognition.face_encoings(img)[0]


# Repeat the step for another image


img1=cv2.imread('D:/VIT/Fall 22-23/Vision and Image Processing/lab/Frec/images/dhoni.png'  )

#rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

cv2.imshow("second Image", img1)

#cv2.imshow("Image", rgb_img)


#img1_encoding = face_recognition.face_encoings(img1)[0]
```

```
#cv2.waitKey(0)


# Step 3: comparison of images

#comapre the face, if both the images are same it will print true.

result = face_recognition.compare_faces([img_encoding], img1_encoding)

print("Result:", result)


"""step4: encode all the faces in the dataset  (why - through the webcam video streaming)

if it find a match in the dataset, it shows the name )




#Encode the face from the folder"""

sfr = SimpleFacerec()
sfr.load_encoding_images("images/")  # images/ is the name of the folder contains image


"""Step 4: Face recognition in real-time on a webcam"""

# https://pysource.com/wp-content/uploads/2021/08/source-code-face-recognition.zip

""" From the link above download  simple_facerec.py  (keep this .py file in the image dateset folder


#Step5:   Take webcame stream

cap = cv2.VideoCapture(2)

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



    #Step 5: Face location and face recognition


#Identify the face passing the frame of the webcam to the function

# detect_known_faces(frame)  - it will give name of the person.


face_locations, face_names = sfr.detect_known_faces(frame)

for face_loc, name in zip(face_locations, face_names):
    y1, x2, y2, x1 = face_loc[0], face_loc[1], face_loc[2]


#step 6: Show name and rectangle
```

```
cv2.putText(frame, name, (x1, y1 -10)), cv2.FONT_HERSHEY_DUPLEX, 1, (0, 0, 200), 2)

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 200), 4)

cv2.imshow("FRAME", frame)

key = cv2.waitKey(1)

if key == 27;:
    break
```