**Digital Image Processing**

**Title: Computer Vision: Feature Extraction**

**Objectives:** To understand the problems of computer vision. To understand features in images and find those features using different computer vision algorithms.

**Tools Used:** Python

**Procedure:** Open IDLE and perform the following tasks.

**Task 1**

Take the image of your face. Make a copy and rotate that copy. Find features on your face using any algorithm e.g SIFT, SURF, ORB.

Hint for Code: See Slides

**Code:**

Could not use the SURF method as it was not available in cv2. I tried using older versions of cv2 as well as other variations of cv2, but still couldn’t use SURF. Hence, after 2 hours of installing and deleting different packages, I just used the SIFT method.

import cv2  
  
# Load the image  
image1 = cv2.imread(r'C:\Users\Naeem\Desktop\Jahanzeb\pics\6.jpeg')  
  
# Convert the training image to RGB  
training\_image = cv2.cvtColor(image1, cv2.COLOR\_BGR2RGB)  
  
# Convert the training image to gray scale  
training\_gray = cv2.cvtColor(training\_image, cv2.COLOR\_RGB2GRAY)  
  
# Create test image by adding Scale Invariance and Rotational Invariance  
test\_image = cv2.pyrDown(training\_image)  
test\_image = cv2.pyrDown(test\_image)  
num\_rows, num\_cols = test\_image.shape[:2]  
  
rotation\_matrix = cv2.getRotationMatrix2D((num\_cols/2, num\_rows/2), 30, 1)  
test\_image = cv2.warpAffine(test\_image, rotation\_matrix, (num\_cols, num\_rows))  
  
test\_gray = cv2.cvtColor(test\_image, cv2.COLOR\_RGB2GRAY)  
  
  
  
surf = cv2.SIFT\_create(800)  
  
train\_keypoints, train\_descriptor = surf.detectAndCompute(training\_gray, None)  
test\_keypoints, test\_descriptor = surf.detectAndCompute(test\_gray, None)  
  
# keypoints\_without\_size = np.copy(training\_image)  
# keypoints\_with\_size = np.copy(training\_image)  
#  
# cv2.drawKeypoints(training\_image, train\_keypoints, keypoints\_without\_size, color = (0, 255, 0))  
#  
# cv2.drawKeypoints(training\_image, train\_keypoints, keypoints\_with\_size, flags = cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)  
  
# Print the number of keypoints detected in the training image  
print("Number of Keypoints Detected In The Training Image: ", len(train\_keypoints))  
  
# Print the number of keypoints detected in the query image  
print("Number of Keypoints Detected In The Query Image: ", len(test\_keypoints))  
  
  
  
  
# Create a Brute Force Matcher object.  
bf = cv2.BFMatcher(cv2.NORM\_L1, crossCheck = False)  
  
# Perform the matching between the SURF descriptors of the training image and the test image  
matches = bf.match(train\_descriptor, test\_descriptor)  
  
# The matches with shorter distance are the ones we want.  
matches = sorted(matches, key = lambda x : x.distance)  
  
result = cv2.drawMatches(training\_image, train\_keypoints, test\_gray, test\_keypoints, matches, test\_gray, flags = 2)  
  
# Display the best matching points  
cv2.imshow('result',result)  
  
# Print total number of matching points between the training and query images  
print("\nNumber of Matching Keypoints Between The Training and Query Images: ", len(matches))  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Task 2**

Write your name and one life goal on a plain paper. Use that image to find handwriting on it using ORB, SIFT or FAST.

Hint for Code: See Teams.

**Code:**

SIFT part:

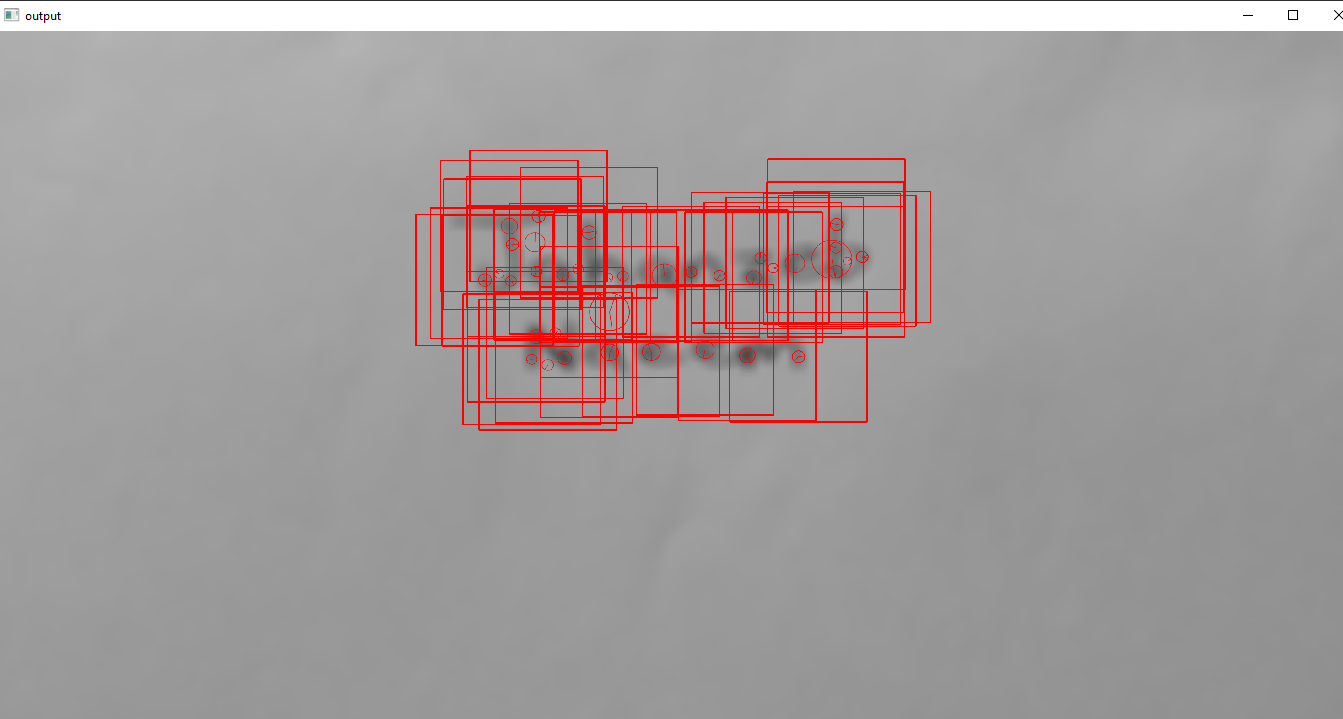
import cv2  
  
def patchExtractor(img, keypoints, w=300):  
 for i in range(len(keypoints)):  
 (x,y) = keypoints[i].pt  
 x1=int(x-w/2)  
 x2=int(x+w/2)  
 y1=int(y-w/2)  
 y2=int(y+w/2)  
 rect = cv2.rectangle(img, (x1, y1), (x2, y2), (0, 0, 255), 2)  
 return img  
  
  
cv2.namedWindow("output", cv2.WINDOW\_NORMAL)  
cv2.resizeWindow("output", 1366, 768)  
cv2.moveWindow("output", 0,0)  
  
# Read image  
img1 = cv2.imread(r'C:\Users\Naeem\Desktop\Jahanzeb\DIP\DIP Lab\images\n1.jpg')  
  
  
# Initiate SIFT detector  
sift = cv2.SIFT\_create()  
  
# Convering to Gray  
img1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)  
  
# Gaussian Blurr  
img1 = cv2.GaussianBlur(img1, (45, 45), 15, cv2.BORDER\_DEFAULT)  
cv2.imshow("output", img1)  
  
# find the keypoints and descriptors with ORB  
kp, des = sift.detectAndCompute(img1, None)  
  
  
# Create images with keypoints  
img1 = cv2.drawKeypoints(img1, kp, img1, (0, 0, 255), cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)  
cv2.imshow("output", img1)  
  
img1 = patchExtractor(img1, kp, 300)  
# cv2.imwrite('sift1AfterGaussian.jpg', img1)  
  
cv2.imshow("output", img1)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

ORB part:

import cv2  
  
def patchExtractor(img, keypoints, w=300):  
 for i in range(len(keypoints)):  
 (x,y) = keypoints[i].pt  
 x1=int(x-w/2)  
 x2=int(x+w/2)  
 y1=int(y-w/2)  
 y2=int(y+w/2)  
 rect = cv2.rectangle(img, (x1, y1), (x2, y2), (0, 0, 255), 2)  
 return img  
  
  
cv2.namedWindow("output", cv2.WINDOW\_NORMAL)  
cv2.resizeWindow("output", 1366, 768)  
cv2.moveWindow("output", 0,0)  
  
# Read image  
img1 = cv2.imread(r'C:\Users\Naeem\Desktop\Jahanzeb\DIP\DIP Lab\images\n1.jpg')  
  
# Initiate ORB detector  
orb = cv2.ORB\_create()  
  
# Converting to Gray  
img1= cv2.cvtColor(img1, cv2.COLOR\_BGR2GRAY)  
  
# find the keypoints and descriptors with ORB  
kp, des = orb.detectAndCompute(img1,None)  
  
# Create images with keypoints  
img1=cv2.drawKeypoints(img1,kp,img1,(0,0,255),cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)  
  
  
img1 = patchExtractor(img1,kp,300)  
cv2.imwrite('Orb1BeforeSharpening.jpg',img1)  
  
cv2.imshow("output", img1)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Screenshot:**

SIFT part:



ORB part:

