CSE 573 Computer Vision & Image Processing

Project 2 Report

UBitName : abhave | personNumber : 50289049 | Date : November 5, 2018

**Task 1- Image Features and Homography**

Source Code:

import cv2

import numpy as np

import random

UBIT = 'abhave'

np.random.seed(sum([ord(c) for c in UBIT]))

#1st part

def doSIFT1():

img1 = cv2.imread('mountain1.jpg')

gray1= cv2.cvtColor(img1,cv2.COLOR\_BGR2GRAY)

org1=gray1

sift = cv2.xfeatures2d.SIFT\_create()

kp1,des1= sift.detectAndCompute(gray1,None)

gray1=cv2.drawKeypoints(gray1,kp1,gray1)

cv2.imwrite('task1\_sift1.jpg',gray1)

return gray1,kp1,org1,des1

def doSIFT2():

img2 = cv2.imread('mountain2.jpg')

gray2= cv2.cvtColor(img2,cv2.COLOR\_BGR2GRAY)

org2=gray2

sift = cv2.xfeatures2d.SIFT\_create()

kp2,des2 = sift.detectAndCompute(gray2,None)

gray2=cv2.drawKeypoints(gray2,kp2,gray2)

cv2.imwrite('task1\_sift2.jpg',gray2)

return gray2,kp2,org2,des2

#2nd part

def knn(img1,img2,kp1,kp2,org1,org2,des1,des2):

bf = cv2.BFMatcher()

matches = bf.knnMatch(des1,des2, k=2)

#print(matches)

new = []

new1=[]

for m,n in matches:

if m.distance < 0.75\*n.distance:

new.append([m])

new1.append(m)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,new,img1,flags=2)

cv2.imwrite('task1\_matches\_knn.jpg',img3)

M=homographyDrawMatchesAndWarp(new1,img1,img2,kp1,kp2)

return img3,M

#3rd ,4th and 5th part

def homographyDrawMatchesAndWarp(new,img1,img2,kp1,kp2):

src\_pts = np.float32([ kp1[m.queryIdx].pt for m in new]).reshape(-1,1,2)

dst\_pts = np.float32([ kp2[m.trainIdx].pt for m in new]).reshape(-1,1,2)

M, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC,5.0)

matchesMask = mask.ravel().tolist()

length=len(matchesMask)

x=random.randint(0,length-10)

y=x+10

draw\_params = dict(matchColor = (0,255,0), singlePointColor = None,matchesMask = matchesMask[x:y],flags = 2)

img3 = cv2.drawMatches(img1,kp1,img2,kp2,new[x:y],None,\*\*draw\_params)

cv2.imwrite('task1\_matches.jpg',img3)

img1 = cv2.imread('mountain1.jpg')

img2 = cv2.imread('mountain2.jpg')

r1, c1 = img1.shape[:2]

r2, c2 = img2.shape[:2]

list\_of\_points\_1 = np.float32([[0,0], [0,r1], [c1, r1], [c1,0]]).reshape(-1,1,2)

temp\_points = np.float32([[0,0], [0,r2], [c2, r2], [c2,0]]).reshape(-1,1,2)

list\_of\_points\_2 = cv2.perspectiveTransform(temp\_points, M)

list\_of\_points = np.concatenate((list\_of\_points\_1, list\_of\_points\_2), axis=0)

[xmin, ymin] = np.int32(list\_of\_points.min(axis=0).ravel() - 0.5)

[xmax, ymax] = np.int32(list\_of\_points.max(axis=0).ravel() + 0.5)

translation\_dist = [-xmin, -ymin]

H\_translation = np.array([[1, 0, translation\_dist[0]], [0, 1, translation\_dist[1]], [0,0,1]])

output\_img = cv2.warpPerspective(img1, H\_translation.dot(M), (xmax - xmin, ymax - ymin))

output\_img[translation\_dist[1]:r1+translation\_dist[1],translation\_dist[0]:c1+translation\_dist[0]] = img2

cv2.imwrite('task1\_pano.jpg',output\_img)

return M

def main():

img1,kp1,org1,des1=doSIFT1()

img2,kp2,org2,des2=doSIFT2()

img3,M=knn(img1,img2,kp1,kp2,org1,org2,des1,des2)

print('The homography matrix is as follows : ')

print(M)

main()

Output:

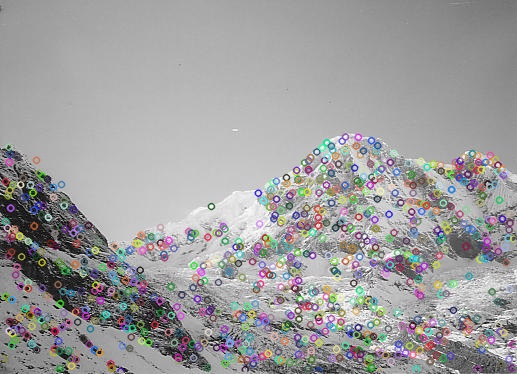
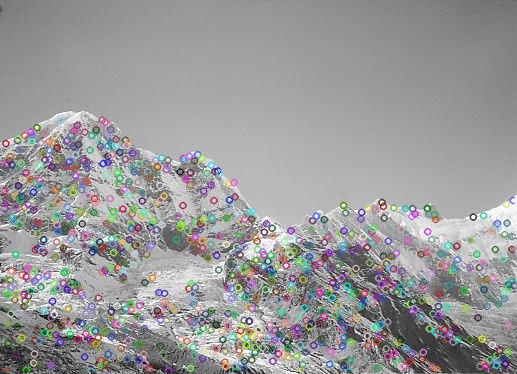
 

Figure 1: Image for showing key points for first image Figure 2: Image for showing key points for first image

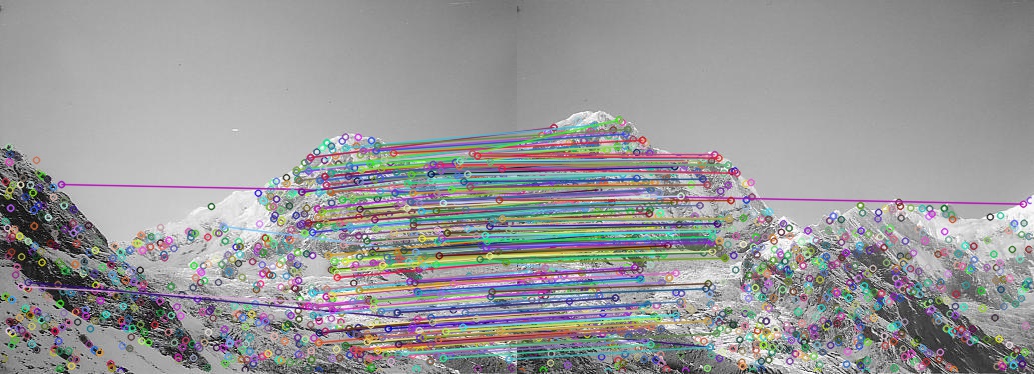


Figure 3: Image for showing all matches using KNN. task1\_matches\_knn.jpg

The homography matrix is as follows :

[ 1.58720376e+00 -2.91747553e-01 -3.95226519e+02]

[ 4.48097764e-01 1.43063310e+00 -1.90273584e+02]

[ 1.20808480e-03 -6.07787702e-05 1.00000000e+00]

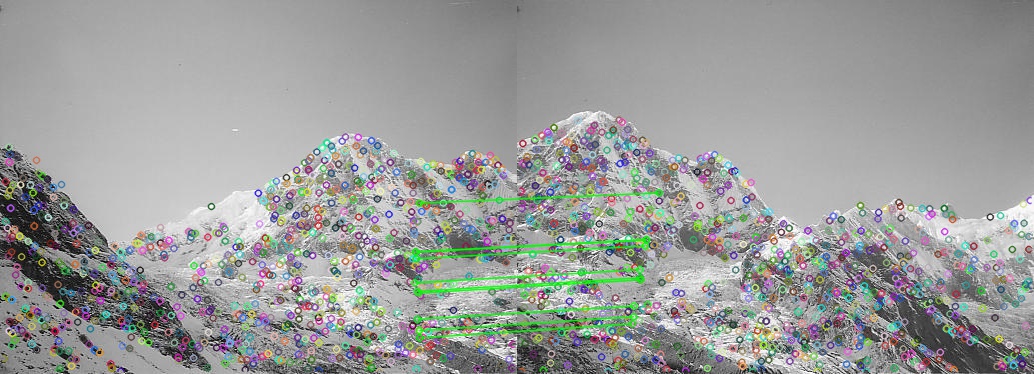


Figure 4: Image for showing match image for around 10 random matches using only inliers. task1\_matches.jpg



Figure 5: Warp the first image to the second image using H. task1\_pano.jpg

**Task 2 - Epipolar Geometry**

Source Code:

# -\*- coding: utf-8 -\*-

"""

Created on Fri Nov 2 23:06:16 2018

@author: abhis

"""

import cv2

import numpy as np

import random

UBIT = 'abhave'

np.random.seed(sum([ord(c) for c in UBIT]))

#1st part

def doSIFT1():

#1st part

img1 = cv2.imread('tsucuba\_left.png')

gray1= cv2.cvtColor(img1,cv2.COLOR\_BGR2GRAY)

org1=gray1

sift = cv2.xfeatures2d.SIFT\_create()

kp1,des1= sift.detectAndCompute(gray1,None)

gray1=cv2.drawKeypoints(gray1,kp1,gray1)

cv2.imwrite('task2\_sift1.jpg',gray1)

return gray1,kp1,org1,des1

def doSIFT2():

img2 = cv2.imread('tsucuba\_right.png')

gray2= cv2.cvtColor(img2,cv2.COLOR\_BGR2GRAY)

org2=gray2

sift = cv2.xfeatures2d.SIFT\_create()

kp2,des2 = sift.detectAndCompute(gray2,None)

gray2=cv2.drawKeypoints(gray2,kp2,gray2)

cv2.imwrite('task2\_sift2.jpg',gray2)

return gray2,kp2,org2,des2

def knnAndepiline(img1,img2,kp1,kp2,org1,org2,des1,des2):

bf = cv2.BFMatcher()

matches = bf.knnMatch(des1,des2,k=2)

new= []

new2=[]

for m,n in matches:

if m.distance < 0.75\*n.distance:

new.append([m])

new2.append(m)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,new,img1,flags=2)

cv2.imwrite('task2\_matches\_knn.jpg',img3)

F=fundamental(new2,img1,img2,kp1,kp2)

return img3,F

#2nd part

def fundamental(new,img1,img2,kp1,kp2):

org1 = cv2.imread('tsucuba\_left.png')

org2 = cv2.imread('tsucuba\_right.png')

src\_pts = np.int32([ kp1[m.queryIdx].pt for m in new ])

dst\_pts = np.int32([ kp2[m.trainIdx].pt for m in new ])

#print(src\_pts.shape,dst\_pts.shape)

F, mask = cv2.findFundamentalMat(src\_pts, dst\_pts, cv2.RANSAC)

#print(F)

src\_pts = src\_pts[mask.ravel()==1]

dst\_pts= dst\_pts[mask.ravel()==1]

orgsrc=src\_pts

orgdst=dst\_pts

length=len(src\_pts)

list=[(255,100,0),(0,0,3),(255,0,0),(255,255,0),(0,0,255),(255,0,255),(255,255,255),(0,255,255),(100,255,0),(255,205,100)]

#print(len(list))

for i in range(0,len(list)):

x=random.randint(0,length-1)

y=x+1

src\_pts=orgsrc[x:y]

dst\_pts=orgdst[x:y]

#print(dst\_pts)

# right image and drawing its lines on left image

lines1 = cv2.computeCorrespondEpilines(dst\_pts, 2,F)

lines1 = lines1.reshape(-1,3)

img5,img6 = drawlines(org1,org2,lines1,src\_pts,dst\_pts,color=list[i])

# left image and drawing its lines on right image

lines2 = cv2.computeCorrespondEpilines(src\_pts, 1,F)

lines2 = lines2.reshape(-1,3)

img3,img4 = drawlines(org2,org1,lines2,dst\_pts,src\_pts,color=list[i])

cv2.imwrite('task2\_epi\_right.jpg',img3)

cv2.imwrite('task2\_epi\_left.jpg',img5)

return F

def drawlines(img1,img2,lines,src\_pts,dst\_pts,color):

r,c = img1.shape[:2]

for r,pt1,pt2 in zip(lines,src\_pts,dst\_pts):

#color = tuple(np.random.randint(0,255,3).tolist())

x0,y0 = map(int, [0, -r[2]/r[1] ])

x1,y1 = map(int, [c, -(r[2]+r[0]\*c)/r[1] ])

img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)

return img1,img2

#4th part

def disparityMap():

img1 = cv2.imread('tsucuba\_left.png')

img2 = cv2.imread('tsucuba\_right.png')

window\_size=5

stereo = cv2.StereoSGBM\_create(numDisparities=32, blockSize=15,speckleWindowSize=10,

speckleRange=1,uniquenessRatio=3,preFilterCap=2,disp12MaxDiff=2,

minDisparity=0,P1=8\*3\*window\_size\*\*2,P2=32\*3\*window\_size\*\*2)

disparity = stereo.compute(img1,img2)

cv2.imwrite('task2\_disparity.jpg',disparity)

def main():

img1,kp1,org1,des1=doSIFT1()

img2,kp2,org2,des2=doSIFT2()

img3,F=knnAndepiline(img1,img2,kp1,kp2,org1,org2,des1,des2)

print('The fundamental matrix is as follows : ')

print(F)

disparityMap()

main()

Output:

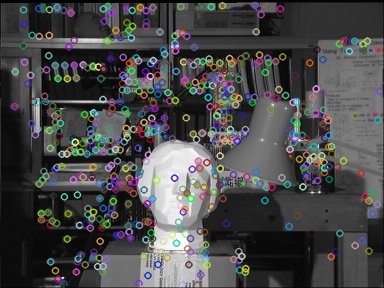
 

Figure 1: Image for showing key points for first image Figure 2: Image for showing key points for first image



Figure 3: Image for showing all matches using KNN. task2\_matches\_knn.jpg

The fundamental matrix is as follows:

[-2.12607354e-06 -8.10713687e-05 7.47530309e-02]

[ 4.60726414e-05 3.79326900e-05 1.32728554e+00]

[-7.52042326e-02 -1.32608913e+00 1.00000000e+00]

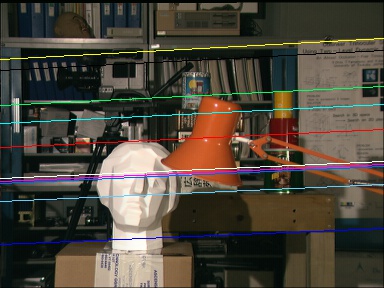
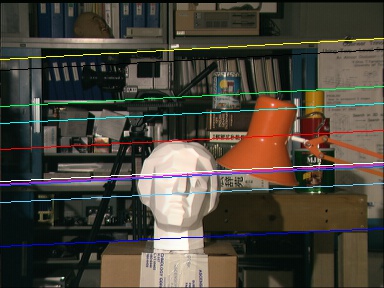
 

Figure 4: Image for showing epilines Figure 5: Image for showing epilines.



Figure 6: Image for showing disparity map.

**Task 3 - K-means Clustering**

Source Code:

# -\*- coding: utf-8 -\*-

"""

Created on Sat Nov 3 23:45:26 2018

@author: abhis

"""

import numpy as np

import math

from matplotlib import pyplot as plt

import cv2

UBIT = 'abhave'

np.random.seed(sum([ord(c) for c in UBIT]))

list=[[5.9, 3.2],[4.6,2.9],[6.2,2.8],[4.7,3.2],[5.5,4.2],[5.0,3.0],[4.9,3.1],[6.7,3.1],[5.1,3.8],[6.0,3.0]]

#x=np.array(list)

k=3

n=[[6.2,3.2],[6.6,3.7],[6.5,3.0]]

#red #green #blue

list=np.array(list)

n=np.array(n)

def iteration(list,n):

list1=[]

list2=[]

list3=[]

for i in range(0,len(list)):

x1=list[i][0]

y1=list[i][1]

row=[]

for j in range(0,3):

x2=n[j][0]

y2=n[j][1]

dist=math.sqrt((x2-x1)\*\*2+(y2-y1)\*\*2)

row.append(dist)

x=np.min(row)

if(row[0]==x):

list1.append([x1,y1])

elif(row[1]==x):

list2.append([x1,y1])

else:

list3.append([x1,y1])

l1=np.asarray(list1,dtype='float64')

l2=np.asarray(list2,dtype='float64')

l3=np.asarray(list3,dtype='float64')

return l1,l2,l3

def reomputeCentroid(list):

x=len(list)

#print(x)

new=[]

sumx=0

sumy=0

for i in range(0,x):

sumx=sumx+list[i][0]

sumy=sumy+list[i][1]

#print(sumx,sumy)

sumx,sumy=sumx/x,sumy/x

new.append(sumx)

new.append(sumy)

return new

def colorPoints1(list1,list2,list3,n,str):

plt.scatter(list1[:,0],list1[:,1], edgecolors='red', facecolor='red',marker="^")

plt.scatter(list2[:,0],list2[:,1],edgecolors='green', facecolor='green',marker="^")

plt.scatter(list3[:,0],list3[:,1],edgecolors='blue', facecolor='blue',marker="^")

plt.scatter(n[0][0],n[0][1],edgecolors='red', facecolor='red',marker='+')

plt.scatter(n[1][0],n[1][1],edgecolors='green', facecolor='green',marker='+')

plt.scatter(n[2][0],n[2][1],edgecolors='blue', facecolor='blue',marker='+')

plt.savefig(str)

plt.show()

plt.close()

def baboon(k):

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

n=[]

for i in range(0,k):

n.append(img[0,i])

n=np.array(n)

l1,l2,l3=baboonIteration(img,n)

#print(img[0][2][0])

#print(n)

new=[]

new.append(l1)

new.append(l2)

new.append(l3)

return new

def baboonIteration(img,n):

list1=[]

list2=[]

list3=[]

for i in range(0,len(img)):

for x in range(0,len(img)):

x1=img[i][x][0]

y1=img[i][x][1]

z1=img[i][x][2]

row=[]

for j in range(0,3):

x2=n[j][0]

y2=n[j][1]

z2=n[j][2]

dist=math.sqrt((x2-x1)\*\*2+(y2-y1)\*\*2+(z2-z1)\*\*2)

row.append(dist)

x=np.min(row)

if(row[0]==x):

list1.append([x1,y1,z1])

elif(row[1]==x):

list2.append([x1,y1,z1])

else:

list3.append([x1,y1,z1])

l1=np.asarray(list1,dtype='float64')

l2=np.asarray(list2,dtype='float64')

l3=np.asarray(list3,dtype='float64')

return l1,l2,l3

def main():

str='task3\_iter1\_a.png'

list1,list2,list3=iteration(list,n)

colorPoints1(list1,list2,list3,n,str)

print('The clusters are as follows :')

print(list1)

print(list2)

print(list3)

n[0]=reomputeCentroid(list1)

n[1]=reomputeCentroid(list2)

n[2]=reomputeCentroid(list3)

print('The updateded centroids are as follows :')

print(n)

str='task3\_iter1\_b.png'

colorPoints1(list1,list2,list3,n,str)

str='task3\_iter2\_a.png'

list1,list2,list3=iteration(list,n)

colorPoints1(list1,list2,list3,n,str)

print('The clusters are as follows :')

print(list1)

print(list2)

print(list3)

str='task3\_iter2\_b.png'

n[0]=reomputeCentroid(list1)

n[1]=reomputeCentroid(list2)

n[2]=reomputeCentroid(list3)

print('The updateded centroids are as follows :')

print(n)

colorPoints1(list1,list2,list3,n,str)

main()

#img=baboon(3)

#img=np.array(img).reshape(512,512,3)

#cv2.imshow('new.jpg',img)

Output:

After 1st iteration classification vector is as follows:

Cluster 1= [[5.9 3.2] [4.6 2.9] [4.7 3.2] [5.0 3.0] [4.9 3.1] [5.1 3.8] [6.0 3.0]]

Cluster 2= [[5.5 4.2]]

Cluster 3= [[6.2 2.8] [6.7 3.1]]

Updated Ui values are as follows:

U1= [5.17142857 3.17142857]

U2= [5.5 4.2]

U3= [6.45 2.95]

After 2nd iteration classification vector is as follows:

Cluster1= [[4.6 2.9] [4.7 3.2] [5.0 3.0] [4.9 3.1]]

Cluster2= [[5.5 4.2] [5.1 3.8]]

Cluster 3= [[5.9 3.2] [6.2 2.8] [6.7 3.1] [6.0 3.0]]

Updated Ui values are as follows:

U1= [4.8 3.05]

U2= [5.3 4.0]

U3= [6.2 3.025]

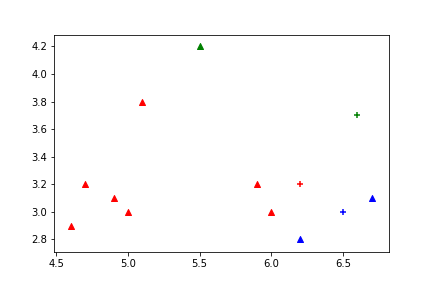
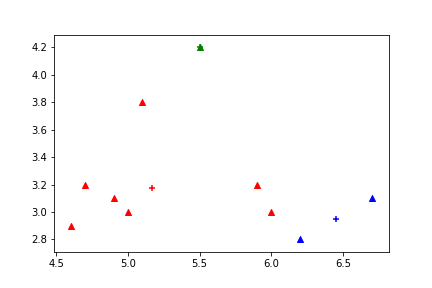
 

Figure 1: task3\_iter1\_a.png Figure 2: task3\_iter1\_b.png

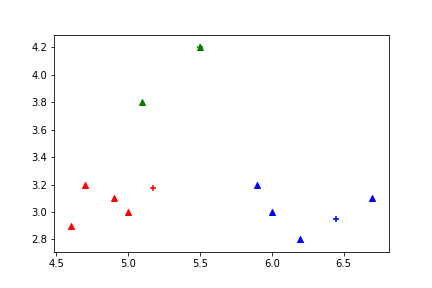
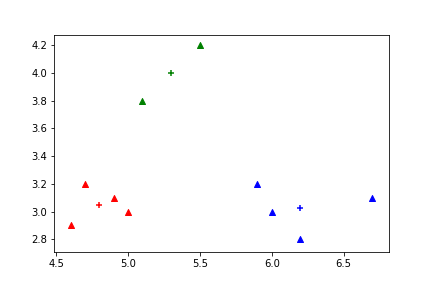
 

Figure 3: task3\_iter2\_a.png Figure 4: task3\_iter2\_b.png

References:

[1] – https://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_feature2d/py\_feature\_homography/py\_feature\_homography.html

[2] – <https://docs.opencv.org/3.2.0/da/de9/tutorial_py_epipolar_geometry.html>

[3] – https://docs.opencv.org/3.1.0/da/df5/tutorial\_py\_sift\_intro.html

[4] – https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_feature2d/py\_matcher/py\_matcher.html

[5] – https://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_calib3d/py\_depthmap/py\_depthmap.html

[6] – https://www.kaggle.com/asymptote/homography-estimate-stitching-two-imag

[7] – https://docs.opencv.org/3.4/d2/d85/classcv\_1\_1StereoSGBM.html