Project 2 of CSE 473/573

Shaoming Xu UB# 50247057

# Task 1

Question 1: (images have been resized, the original images in task1\_img folder)

fig1.1 task1\_sift1.jpg fig1.2 task1\_sift2.jpg

Question 2: (images have been resized, the original images in task1\_img folder)



task1\_matches\_knn.jpg

Question 3:

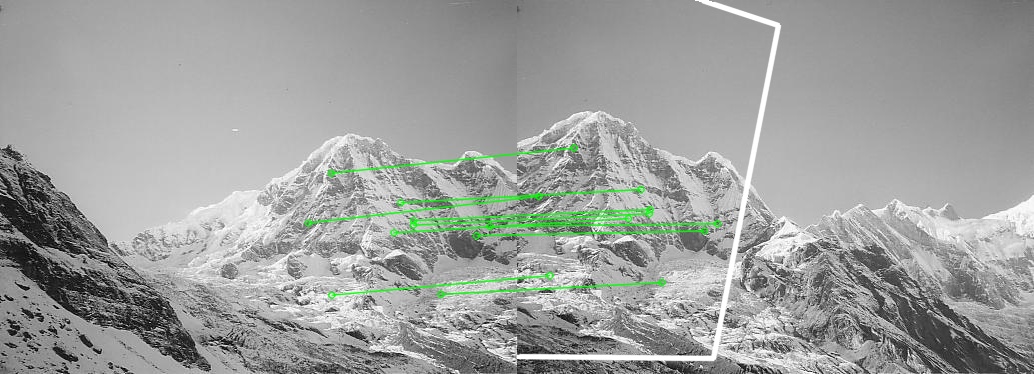
homography matrix:

[[ 1.58799966e+00 -2.91541838e-01 -3.95539425e+02]

[ 4.48199617e-01 1.43139761e+00 -1.90370131e+02]

[ 1.20864262e-03 -5.94920214e-05 1.00000000e+00]]

Question 4: (images have been resized, the original images in task1\_img folder)



task1\_matches.jpg

Question 5: (images have been resized, the original images in task1\_img folder)



task1\_pano.jpg

### Task 1 code

import cv2

import numpy as np

import imutils

import matplotlib.pyplot as plt

UBIT = '50247057'

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

def sift\_match(img\_l,img\_r,task="task1"):

img\_l\_gray = cv2.cvtColor(img\_l, cv2.COLOR\_BGR2GRAY)

img\_r\_gray = cv2.cvtColor(img\_r, cv2.COLOR\_BGR2GRAY)

# Initiate SIFT detector

sift = cv2.xfeatures2d.SIFT\_create()

# find the keypoints and descriptors with SIFT

kp\_l,des\_l = sift.detectAndCompute(img\_l\_gray,None)

kp\_r,des\_r = sift.detectAndCompute(img\_r\_gray,None)

cv2.imwrite("../"+task+"\_img/"+task+"\_sift1.jpg", cv2.drawKeypoints(img\_l\_gray, kp\_l, None,flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS))

cv2.imwrite("../"+task+"\_img/"+task+"\_sift2.jpg", cv2.drawKeypoints(img\_r\_gray, kp\_r, None,flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS))

# FLANN parameters

FLANN\_INDEX\_KDTREE = 0

index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

search\_params = dict(checks=50) # or pass empty dictionary

flann = cv2.FlannBasedMatcher(index\_params,search\_params)

matches = flann.knnMatch(des\_l,des\_r,k=2)

good = []

for m,n in matches:

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

good.append([m])

img\_res = cv2.drawMatchesKnn(img\_l,kp\_l,img\_r,kp\_r,good,None,flags=2)

cv2.imwrite("../"+task+"\_img/"+task+"\_matches\_knn.jpg",img\_res)

return kp\_l, des\_l, kp\_r, des\_r, good

def draw\_match\_img(img\_l, img\_r, kp\_l, des\_l, kp\_r, des\_r, good, seed):

img\_l\_gray = cv2.cvtColor(img\_l, cv2.COLOR\_BGR2GRAY)

img\_r\_gray = cv2.cvtColor(img\_r, cv2.COLOR\_BGR2GRAY)

## Use the FLANN

MIN\_MATCH\_COUNT = 10

if len(good) > MIN\_MATCH\_COUNT:

src\_pts = np.float32([ kp\_l[m[0].queryIdx].pt for m in good ]).reshape(-1,1,2)

dst\_pts = np.float32([ kp\_r[m[0].trainIdx].pt for m in good ]).reshape(-1,1,2)

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

matchesMask = mask.ravel().tolist()

h,w= img\_l\_gray.shape

pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)

dst = cv2.perspectiveTransform(pts,M)

img\_r\_gray = cv2.polylines(img\_r\_gray,[np.int32(dst)],True,255,3, cv2.LINE\_AA)

else:

print("Not enough matches are found - %d/%d" % (len(good),MIN\_MATCH\_COUNT))

matchesMask = None

rand = np.random.RandomState(seed)

index = rand.permutation(len(good))[0:10];

draw\_params = dict(matchColor = (0,255,0), # draw matches in green color

singlePointColor = None,

matchesMask = [matchesMask[i] for i in index], # draw only inliers

flags = 2)

img5 = cv2.drawMatches(img\_l\_gray,kp\_l,img\_r\_gray,kp\_r, list(map(lambda x: x[0], [good[i] for i in index])) ,None,\*\*draw\_params)

cv2.imwrite('../task1\_img/task1\_matches.jpg',img5)

return M

def warpTwoImages(img1, img2, H):

'''

Cite: learn from

https://stackoverflow.com/questions/13063201/how-to-show-the-whole-image-when-using-opencv-warpperspective

warp img2 to img1 with homograph H'''

h1,w1 = img1.shape[:2]

h2,w2 = img2.shape[:2]

pts1 = np.float32([[0,0],[0,h1],[w1,h1],[w1,0]]).reshape(-1,1,2)

pts2 = np.float32([[0,0],[0,h2],[w2,h2],[w2,0]]).reshape(-1,1,2)

pts2\_ = cv2.perspectiveTransform(pts2, H)

pts = np.concatenate((pts1, pts2\_), axis=0)

[xmin, ymin] = np.int32(pts.min(axis=0).ravel() - 0.5)

[xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 0.5)

t = [-xmin,-ymin]

Ht = np.array([[1,0,t[0]],[0,1,t[1]],[0,0,1]]) # translate

result = cv2.warpPerspective(img2, Ht.dot(H), (xmax-xmin, ymax-ymin))

result[t[1]:h1+t[1],t[0]:w1+t[0]] = img1

return result

if \_\_name\_\_ == "\_\_main\_\_":

UBIT = '50247057'

seed = sum([ord(c) for c in UBIT])

filename\_1 = '../task1\_img/mountain1.jpg'

filename\_2 = '../task1\_img/mountain2.jpg'

img\_l = cv2.imread(filename\_1) # trainImage

img\_r = cv2.imread(filename\_2) # queryImage

img\_r\_gray = cv2.cvtColor(img\_r, cv2.COLOR\_BGR2GRAY)

img\_l\_gray = cv2.cvtColor(img\_l, cv2.COLOR\_BGR2GRAY)

kp\_l, des\_l, kp\_r,des\_r, good = sift\_match(img\_l, img\_r, task="task1")

M = draw\_match\_img(img\_l, img\_r, kp\_l, des\_l, kp\_r, des\_r, good, seed)

print("homography matrix:")

print(M)

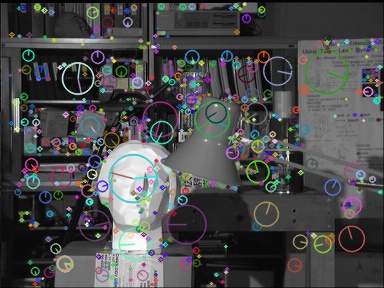
## wrap left images

res = warpTwoImages(img\_r, img\_l, M)

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

# Task2

Question 1: (images have been resized, the original images in task2\_img folder)

task2\_sift1.jpg task2\_sift2.jpg



task2\_matches\_knn.jpg

Question 2:

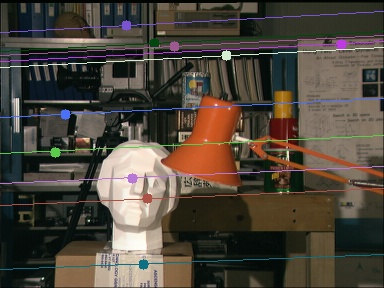
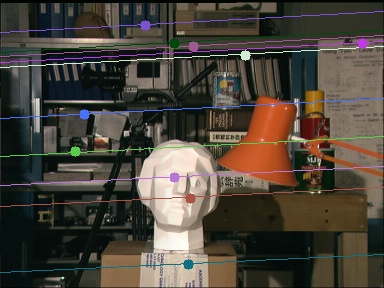
fundamental matrix:

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

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

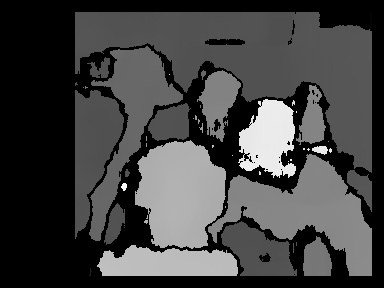
[-7.52042326e-02 -1.32608913e+00 1.00000000e+00]]

Question 3: (images have been resized, the original images in task2\_img folder)



task2\_epi\_left.jpg task2\_epi\_right.jpg

Question 4: (images have been resized, the original images in task2\_img folder)



task2\_disparity.jpg

### Task2 code

import cv2

import numpy as np

import imutils

import matplotlib.pyplot as plt

from task\_1 import sift\_match

UBIT = '50247057'

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

"""

Cite: I learn a lot from opencv python tutorial for task2 part programming.

https://docs.opencv.org/3.4/d9/db7/tutorial\_py\_table\_of\_contents\_calib3d.html

"""

def drawlines(img1,img2,lines,pts1,pts2,seed=30):

'''

cite: https://docs.opencv.org/3.2.0/da/de9/tutorial\_py\_epipolar\_geometry.html

img1 - image on which we draw the epilines for the points in img2

lines - corresponding epilines

'''

r,c,d = img1.shape

rand = np.random.RandomState(seed)

index = rand.permutation(len(lines))[0:10]

for r,pt1,pt2 in np.asarray(list(zip(lines,pts1,pts2)))[index]:

color = tuple(rand.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)

img1 = cv2.circle(img1,tuple(pt1),5,color,-1)

img2 = cv2.circle(img2,tuple(pt2),5,color,-1)

return img1,img2

if \_\_name\_\_ == "\_\_main\_\_":

UBIT = '50247057'

seed = sum([ord(c) for c in UBIT])

img1 = cv2.imread('../task2\_img/tsucuba\_left.png') # left image

img2 = cv2.imread('../task2\_img/tsucuba\_right.png') # right image

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

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

##question 1

kp1, des1, kp2, des2, good = sift\_match(img1,img2,task="task2")

## question 2

pts2 = [kp2[m[0].trainIdx].pt for m in good]

pts1 = [kp1[m[0].queryIdx].pt for m in good]

pts1 = np.int32(pts1)

pts2 = np.int32(pts2)

F, mask = cv2.findFundamentalMat(pts1,pts2,cv2. FM\_RANSAC)

# We select only inlier points

pts1 = pts1[mask.ravel()==1]

pts2 = pts2[mask.ravel()==1]

print("fundamental matrix:")

print(F)

##question 3

# Find epilines corresponding to points in right image (second image) and

# drawing its lines on left image

lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)

lines1 = lines1.reshape(-1,3)

img5,img6 = drawlines(img1,img2,lines1,pts1,pts2, seed=seed)

# Find epilines corresponding to points in left image (first image) and

# drawing its lines on right image

lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)

lines2 = lines2.reshape(-1,3)

img3,img4 = drawlines(img2,img1,lines2,pts2,pts1,seed=seed)

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

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

## question 4

stereo = cv2.StereoBM\_create(numDisparities=64, blockSize=25)

disparity = stereo.compute(img1\_gray,img2\_gray)

norm\_image = cv2.normalize(disparity, None, alpha = 0, beta = 1, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_32F)

cv2.imwrite('../task2\_img/task2\_disparity.jpg',(norm\_image\*255).astype(np.uint8))

# Task3

### Part a: K-means Clustering

There are total two iterations for the given data as shows below. The triangle represents data, the circle represents the cluster center. I show the center and data in same image.

**Iteration 0:**

This shows the clusters’ center location which represents the mu in report.

Cluster 1: [6.2 3.2], Cluster 2: [6.6 3.7], Cluster 3: [6.5 3.0 ]

classification vector: [1 1 3 1 2 1 1 3 1 1]

**Iteration 1:**

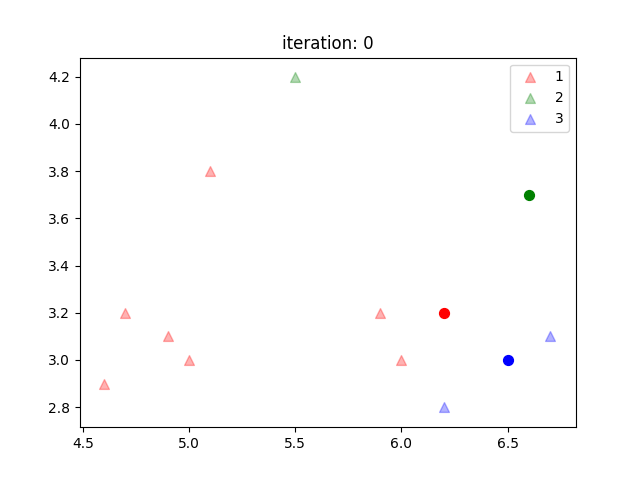
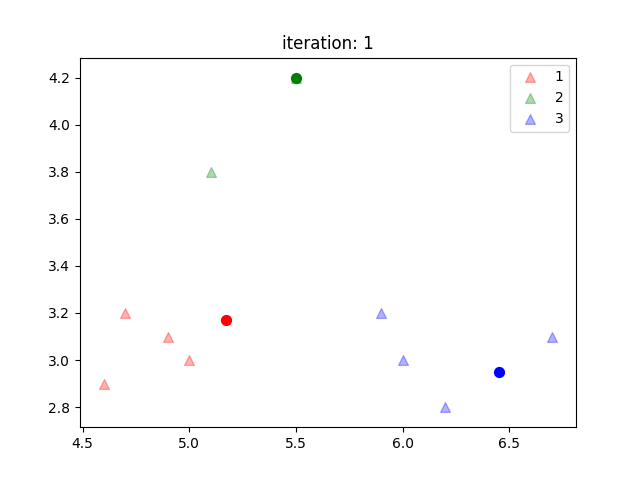
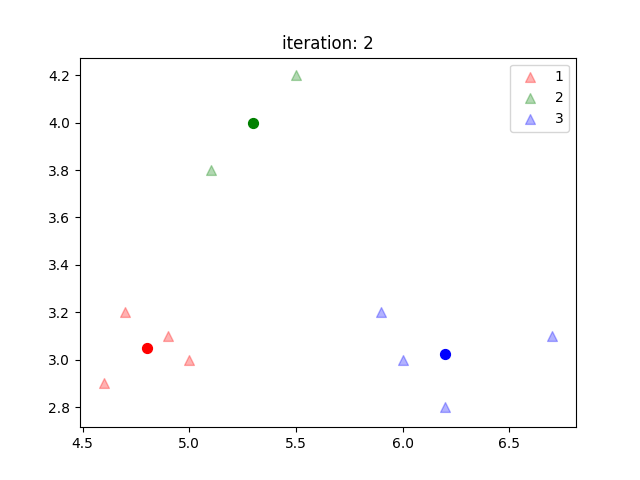
cluster: [5.17142857 3.17142857], cluster: [5.5 4.2], cluster: [6.45 2.95]

classification vector: [3 1 3 1 2 1 1 3 2 3]

**Iteration 2:**

cluster: [4.8 3.05], cluster: [5.3 4. ], cluster: [6.2 3.025]

classification vector: [3 1 3 1 2 1 1 3 2 3]

### Part a: K-means Clustering Code:

import numpy as np

import cv2

from io import StringIO

import matplotlib

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import mylibrary as mylib

from mylibrary import euclidean\_distance

import os

UBIT = '50247057'

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

class Center:

def \_\_init\_\_(self, center, pts):

self.center = center

self.pts = pts

def set\_center(self, center):

self.center = center

def set\_pts(self, pts):

self.pts = pts

def \_\_eq\_\_(self, other):

return np.all(self.center == other.center)

def \_\_repr\_\_(self):

return "cluster: " + str(self.center)

def init\_given(data,k,seed):

centers = np.array([[6.2,3.2],

[6.6,3.7],

[6.5,3.0]])

centers = list(map(lambda x: Center(x,np.array([])), centers))

centers = assign\_to\_center(data, centers,k)

return centers

def init\_centers\_random(data, k, seed=20):

centers = data[np.random.RandomState(seed=seed).permutation(data.shape[0])[0:k]]

centers = list(map(lambda x: Center(x,np.array([])), centers))

centers = assign\_to\_center(data, centers,k)

return centers

def assign\_to\_center(data, centers, k):

dis\_matrix = np.empty((0,data.shape[0]))

for center in centers:

dis\_matrix = np.vstack((dis\_matrix, np.sum(np.square(data - center.center), axis=1)))

belongs = np.argmin(dis\_matrix, axis=0)

for i in range(k):

centers[i].pts = np.where(belongs == i)[0]

return centers

def update\_centers(data, centers, k):

not\_updated = True

new\_centers = []

for center in centers:

new\_centers.append(Center(np.mean(data[center.pts],axis=0), np.array([])))

return assign\_to\_center(data,new\_centers, k)

def plot\_k\_mean(data, centers, itr, save\_path="../task3\_img/k\_mean"):

try:

os.makedirs(save\_path)

except FileExistsError:

print("use existing folder:", save\_path)

label\_set = set(np.arange(len(centers)))

color\_map = dict(zip(label\_set, cm.rainbow(np.linspace(0, 1, len(label\_set)))))

for label in label\_set:

index = centers[label].pts

plt.scatter(data[index][:,0], data[index][:,1], s=20, c=color\_map[label],

alpha=0.3, label=label)

plt.scatter(centers[label].center[0], centers[label].center[1], s=100, c=color\_map[label],

alpha=1.0, marker='x')

plt.title("iteration: "+ str(itr))

plt.legend(loc='best')

#plt.show()

plt.savefig(save\_path+"/iteration\_" + str(itr) + ".png")

plt.close()

def plot\_k\_mean\_a(data, centers, itr, save\_path="../task3\_img/k\_mean"):

try:

os.makedirs(save\_path)

except FileExistsError:

print("use existing folder:", save\_path)

label\_set = set(np.arange(len(centers)))

#color\_map = dict(zip(label\_set, cm.rainbow(np.linspace(0, 1, len(label\_set)))))

color\_map = dict(zip(label\_set, ['red','green','blue']))

for label in label\_set:

index = centers[label].pts

plt.scatter(data[index][:,0], data[index][:,1], s=50, c=color\_map[label],

alpha=0.3, label=label+1,marker='^')

plt.scatter(centers[label].center[0], centers[label].center[1], s=50, color=color\_map[label],

alpha=1.0, marker='o')

plt.title("iteration: "+ str(itr))

plt.legend(loc='best')

plt.savefig(save\_path+"/iteration\_" + str(itr) + ".png")

plt.close()

def k\_mean(data, k, init\_fun, max\_itr=50, seed=20, need\_plot=False, plot\_fun=None, show\_info=False):

itr = 0

centers = init\_fun(data,k,seed)

while itr <= max\_itr:

print("iteration :", itr)

if show\_info:

classification\_vector = np.zeros(len(data),dtype=np.int)

for i, center in enumerate(centers):

print(center)

classification\_vector[center.pts] = i + 1

print(classification\_vector)

if need\_plot:

plot\_fun(data, centers, itr)

new\_centers = update\_centers(data,centers,k)

centers\_1 = np.asarray(list(map(lambda x: x.center ,centers)))

centers\_2 = np.asarray(list(map(lambda x: x.center ,new\_centers)))

if np.all(centers\_1 == centers\_2):

break

centers = new\_centers

itr += 1

print("total iteration", itr)

return centers

if \_\_name\_\_ == "\_\_main\_\_":

UBIT = '50247057'

seed = sum([ord(c) for c in UBIT])

data\_given = np.array([[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]])

data\_a = data\_given

k = 3

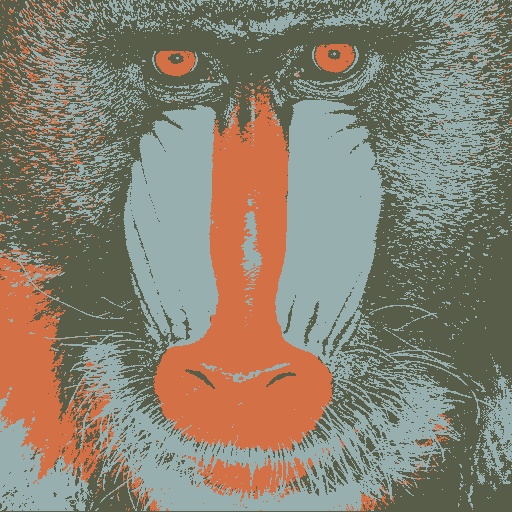
max\_itr = 100

itr = 0

centers = k\_mean(data\_a,k,init\_fun=init\_given ,need\_plot=True,plot\_fun=plot\_k\_mean\_a, show\_info=True)

## Color Quantization:

For the purpose to save paper, I have resized the image in report. If you want to see the original image, please go to task3\_img folder. Thank you

** **

**K = 3 k = 5**

** **

**k = 10 k=20**

#### Color Quantization code:

import numpy as np

import cv2

from io import StringIO

import task\_3\_k\_mean as k\_mean

UBIT = '50247057'

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

def to\_pixel\_list(img):

pixel\_list = np.empty((0,img[0].shape[1]))

for row in img:

pixel\_list = np.append(pixel\_list, row, axis=0)

return pixel\_list

def get\_img(centers,shape):

row\_num, col\_num, \_ = shape

new\_img = np.empty(shape)

for center in centers:

pixel = center.center.astype(np.uint8)

locs = list(map(lambda x: (int(x/col\_num), int(x%col\_num)) ,center.pts))

for loc in locs:

new\_img[loc] = pixel

return new\_img.astype(np.uint8)

def quantized\_img(img, k, init\_fun = k\_mean.init\_centers\_random, max\_itr=10000, seed=20):

pixel\_list = to\_pixel\_list(img)

centers = k\_mean.k\_mean(pixel\_list, k, init\_fun, max\_itr=100000, seed=seed)

return get\_img(centers,img.shape)

if \_\_name\_\_ == "\_\_main\_\_":

UBIT = '50247057'

seed = sum([ord(c) for c in UBIT])

img = cv2.imread("../data/baboon.jpg")

ks = [3,5,10,20]

for k in ks:

print("k =",k)

new\_img = quantized\_img(img,k,seed=seed)

cv2.imwrite("../task3\_img/task3\_baboon\_"+str(k)+".jpg",new\_img)

print()

### Bonus: Gaussian Mixture Model

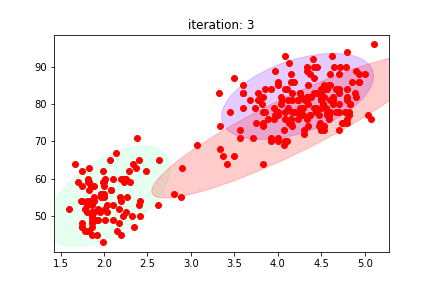
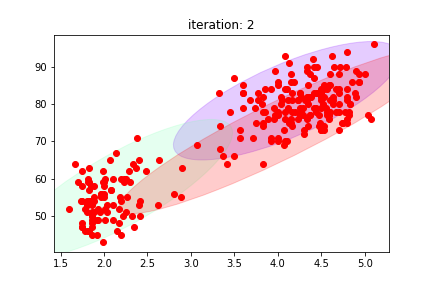
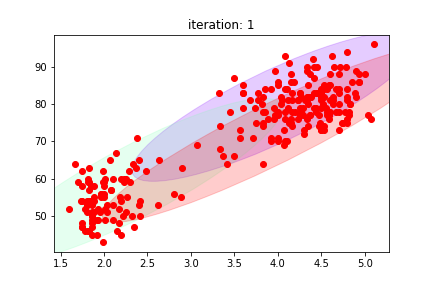
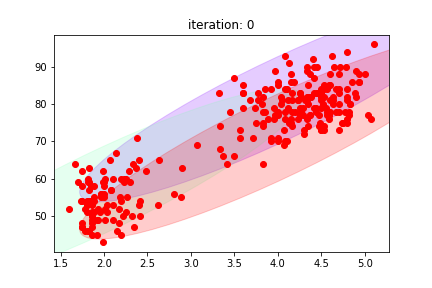
**Question a: what is the u\_i after the first iteration?**

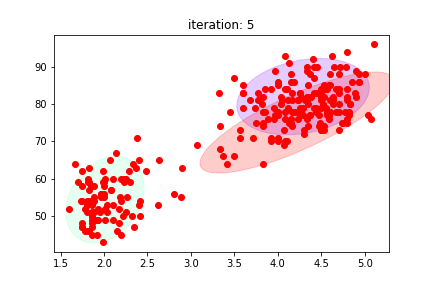
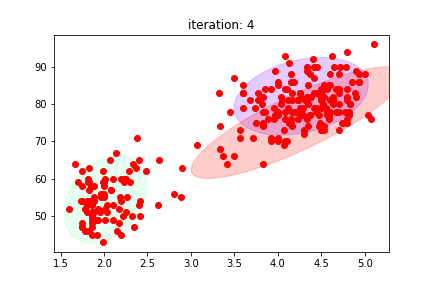
[5.3165079 3.21527292]

[5.61129795 3.38505311]

[5.60443565 3.14420061]

**Question b: Include first five plots for faithful data.**





### Bonus: Gaussian Mixture Model code:

import numpy as np

from io import StringIO

import matplotlib

import matplotlib.pyplot as plt

import matplotlib.cm as cm

from matplotlib.patches import Ellipse

from scipy.stats import multivariate\_normal

import os

UBIT = '50247057'

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

class Point:

def \_\_init\_\_(self, loc, r\_ks):

self.loc = loc

self.r\_ks = r\_ks.copy()

def set\_r\_ks(self, r\_ks):

self.r\_ks = r\_ks.copy()

def set\_r\_k(self, k, val):

self.r\_ks[k] = val

def show\_r\_ks(self):

res = "\n "

for i, val in enumerate(self.r\_ks):

res += "r\_" + str(i) + "=" + str(val) +"\n "

return res + "\n\n"

def \_\_repr\_\_(self):

return "pts:" + str(self.loc) + "\nr\_ks is:" + self.show\_r\_ks()

class Distribution:

def \_\_init\_\_(self, mean, sigma, pi\_k):

self.mean = mean.copy()

self.sigma = sigma.copy()

self.pi\_k = pi\_k

def set\_mean(self, mean):

self.mean = mean.copy()

def set\_sigma(self, sigma):

self.sigma = sigma.copy()

def set\_pi\_k(self, pi\_k):

self.pi\_k = pi\_k

def \_\_eq\_\_(self, other):

return np.all(abs(self.mean - other.mean) < 0.001)

def \_\_repr\_\_(self):

return "mean:\n" + str(self.mean) + "\nsigma:\n" + str(self.sigma) +"\npi\_k:\n" + str(self.pi\_k) + "\n\n\n"

def preprocess(data\_given, means, sigmas, pi\_k):

if len(means) != len(sigmas):

raise Exception("means number must equal to sigmas number!")

r\_ks = np.zeros(len(means))

distributions = list(map(lambda x: Distribution(x[0], x[1], pi\_k), zip(means,sigmas)))

data = list(map(lambda x: Point(x, r\_ks), data\_given))

return data, distributions

def e\_step(data, distributions):

def set\_r\_ks(point, distributions):

a\_s = []

b = 0

for distribution in distributions:

a = distribution.pi\_k \* multivariate\_normal.pdf(point.loc, distribution.mean, distribution.sigma)

a\_s.append(a)

b += a

a\_s = np.asarray(a\_s)

r\_ks = a\_s / b

point.set\_r\_ks(r\_ks)

return point

return list(map(lambda x: set\_r\_ks(x, distributions), data))

def m\_step(data, distributions):

r = np.asarray(list(map(lambda x: x.r\_ks, data)))

a = np.sum(r,axis=0)

#get pi\_ks

pi\_ks = a / r.shape[0]

#get means

locs = np.asarray(list(map(lambda x: x.loc, data)))

means = []

for r\_k in r.T:

means.append(np.sum(np.asarray(list(map(lambda x : x[0] \* x[1], zip(locs, r\_k)))), axis=0))

means = np.asarray(means)

means = np.asarray(list(map(lambda x : x[0] / x[1], zip(means, a))))

# get sigmas

sigmas = []

for k, mean in enumerate(means):

x\_u = locs - mean

v\_1 = x\_u.T @ np.asarray(list(map(lambda x: x[0] \* x[1], zip(x\_u, r.T[k]))))

v\_2 = a[k]

sigmas.append(v\_1 / v\_2)

sigmas = np.asarray(sigmas)

# return new list of Distribution objects

return list(map(lambda x: Distribution(x[0],x[1],x[2]), zip(means, sigmas, pi\_ks)))

def det(A):

if A.shape[0] != A.shape[1]:

raise Exception("must be square matrix")

if A.shape == (2,2):

return A[0,0] \* A[1,1] - A[0,1] \* A[1,0]

res = 0

for i, row in enumerate(A.T):

res += row[0] \* pow(-1,i) \* det(np.delete(np.delete(A, 0, 0), i, 1))

return res

def gaussian\_mixture\_model(data\_given, init\_fun, max\_itr=100, trace\_plot=False, data\_name="anony", export\_path="../task3\_img/"):

means, sigmas, pi\_k = init\_fun()

data, distributions = preprocess(data\_given, means, sigmas, pi\_k)

if trace\_plot:

save\_path = export\_path + data\_name

try:

os.makedirs(save\_path)

except FileExistsError:

print("use existing folder:", save\_path)

for i in np.arange(max\_itr):

print("iteration:", i)

if i == 1 and data\_name == "bonus\_a":

print("means")

for val in [distribution.mean for distribution in distributions]:

print(val)

if trace\_plot:

plot\_gmm(data\_given, distributions, i, save\_path)

try:

data = e\_step(data, distributions)

except Exception as inst:

print("\nEncounter exception")

print(inst)

break

new\_distributions = m\_step(data, distributions)

if new\_distributions == distributions:

break

if not np.all(np.asarray(list(map(lambda x: det(x.sigma), new\_distributions))) != 0):

print("\n\nEncounter singular matrix exception!!!")

break

distributions = new\_distributions

return data, distributions

def plot\_gmm(data\_given, distributions, itr,save\_path="../task3\_img/anony"):

# Plot the raw points...

x, y = data\_given.T

plt.plot(x, y, 'ro')

label\_set = set(np.arange(len(distributions)))

color\_map = dict(zip(label\_set, cm.rainbow(np.linspace(0, 1, len(label\_set)))))

# Plot a transparent 3 standard deviation covariance ellipse

for k, label in enumerate(label\_set):

plot\_cov\_ellipse(distributions[k].sigma, distributions[k].mean, 2, None, alpha=0.2,color=color\_map[label], )

plt.title("iteration: "+ str(itr))

plt.savefig(save\_path+"/iteration\_" + str(itr) + ".png")

plt.close()

def plot\_cov\_ellipse(cov, pos, nstd=2, ax=None, \*\*kwargs):

"""

adopt from: https://github.com/joferkington/oost\_paper\_code/blob/master/error\_ellipse.py

"""

def eigsorted(cov):

vals, vecs = np.linalg.eigh(cov)

order = vals.argsort()[::-1]

return vals[order], vecs[:,order]

if ax is None:

ax = plt.gca()

vals, vecs = eigsorted(cov)

theta = np.degrees(np.arctan2(\*vecs[:,0][::-1]))

# Width and height are "full" widths, not radius

width, height = 2 \* nstd \* np.sqrt(vals)

ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, \*\*kwargs)

ax.add\_artist(ellip)

return ellip

def get\_faithful\_data(filename):

with open(filename) as f:

raw\_data = np.genfromtxt(StringIO(f.read()), dtype='str')

data = raw\_data[:,1:].astype("float")

label = raw\_data[:,0].astype("int")

return data, label

def init\_means\_sigmas\_piks\_a():

means = np.array([[6.2, 3.2],

[6.6, 3.7],

[6.5, 3.0]])

sigmas = np.array([[[0.5,0],[0,0.5]],

[[0.5,0],[0,0.5]],

[[0.5,0],[0,0.5]]])

pi\_k = 1 / 3

return means, sigmas, pi\_k

def init\_means\_sigmas\_piks\_faithful():

means = np.array([[4.0, 81],

[2.0, 57],

[4.0, 71]])

sigmas = np.array([[[1.30,13.98],[13.98,184.82]],

[[1.30,13.98],[13.98,184.82]],

[[1.30,13.98],[13.98,184.82]]

])

pi\_k = 1 / 3

return means, sigmas, pi\_k

if \_\_name\_\_ == "\_\_main\_\_":

print("\*\*\*\*\*\*\*\*\*\*\*\* question a \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

data\_a = np.array([[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]])

data, distributions = gaussian\_mixture\_model(data\_a, init\_means\_sigmas\_piks\_a, trace\_plot=True, data\_name="bonus\_a")

print("\*\*\*\*\*\*\*\*\*\*\*\* question b \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

faithful\_data, \_ = get\_faithful\_data("../data/faithful.dat")

data, distributions = gaussian\_mixture\_model(faithful\_data, init\_means\_sigmas\_piks\_faithful,

max\_itr=10, trace\_plot=True, data\_name="bonus\_b\_faithful\_data")

I have deleted all my comments in code to make the report short, but it still reaches 16 pages. Sorry for any inconvenience for you~