# 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)

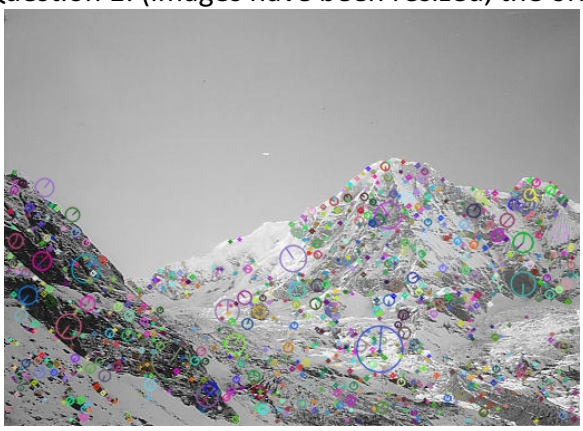


fig1.1 task1_sift1.jpg

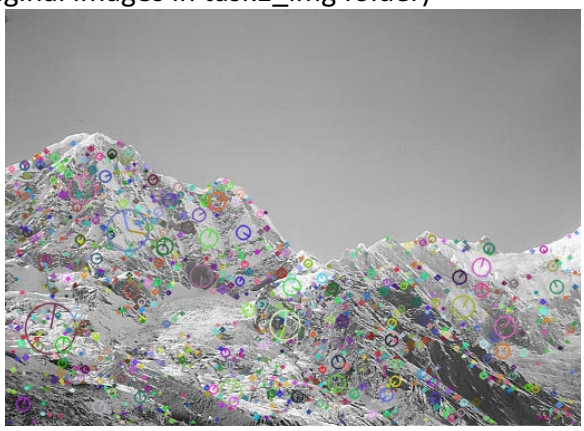
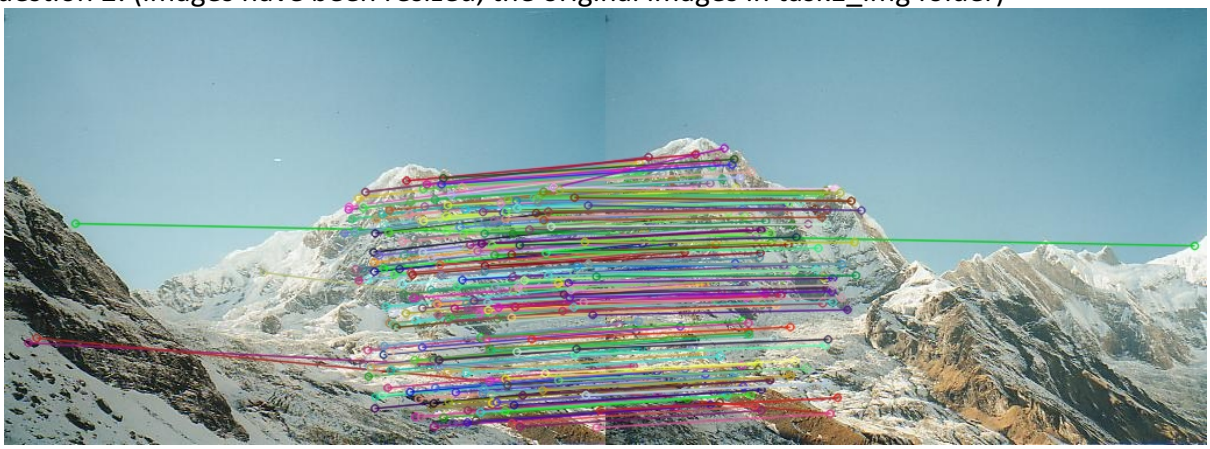


fig1.2 task1_sift2.jpg

Question 2: (images have been resized, the original images in task1_img folder)



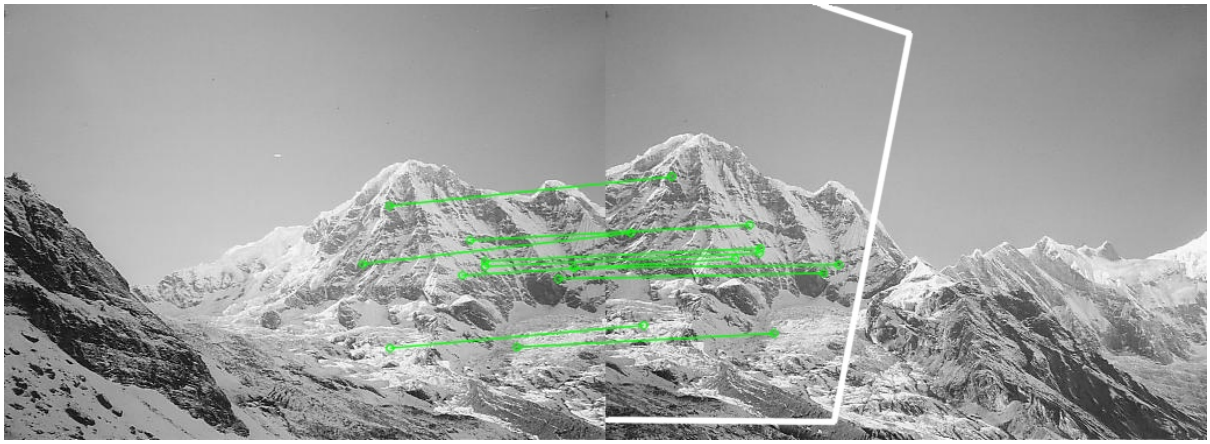
task1_matches_knn.jpg

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

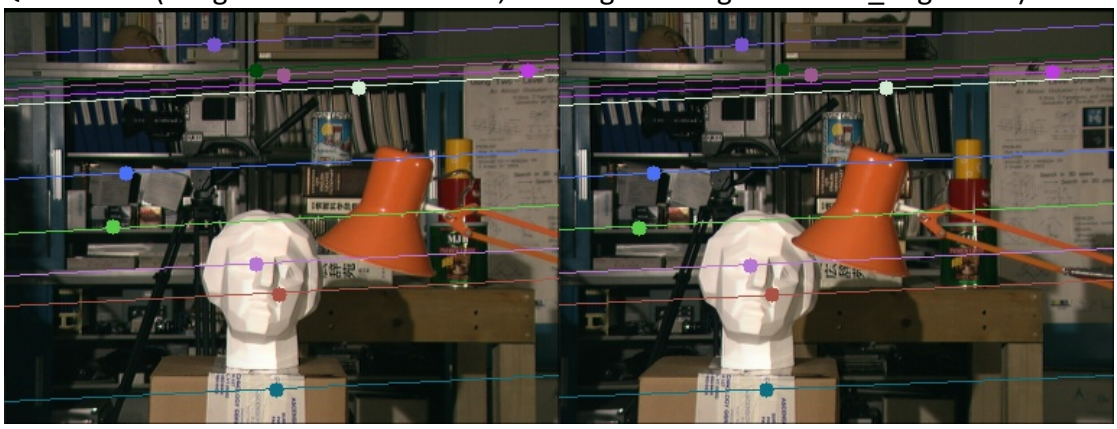
task2_sift2.jpg

task2_matches_knn.jpg

Question 2:

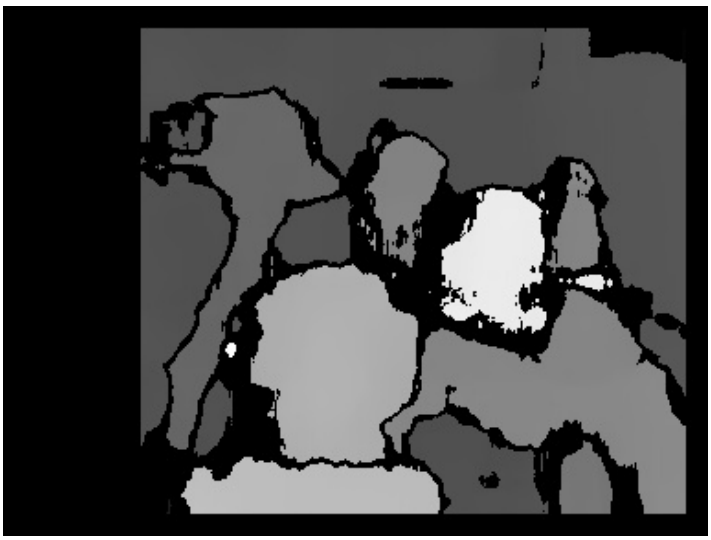
```
fundamental matrix:
[[-2.12607354e-06 -8.10713687e-05  7.47530309e-02]
 [ 4.60726414e-05  3.79326900e-05  1.32728554e+00]
 [-7.52042326e-02 -1.32608913e+00  1.00000000e+00]]
```

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:**

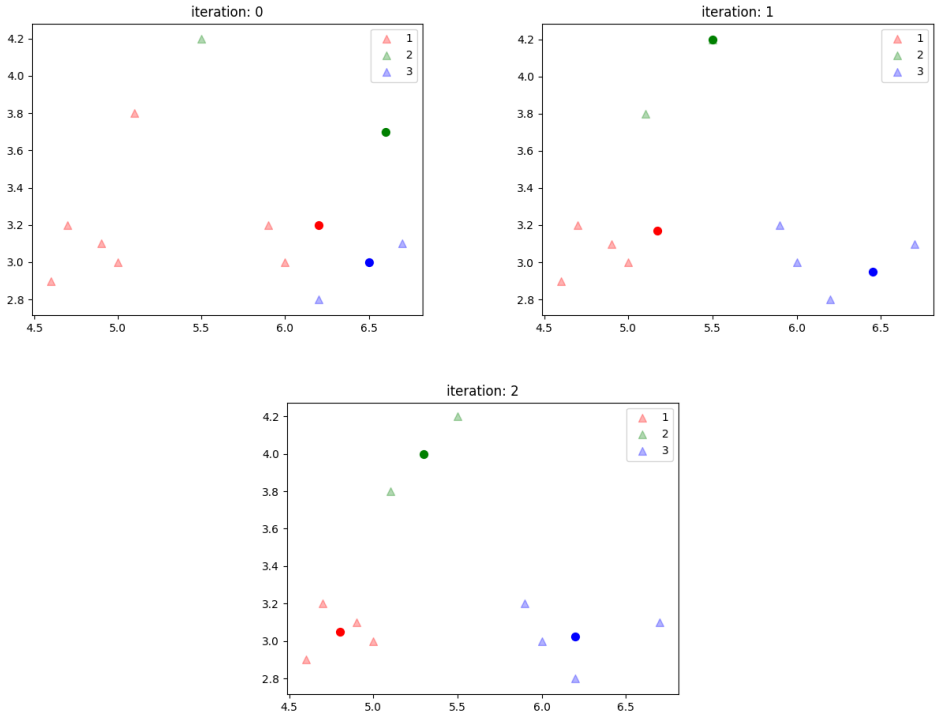
cluster: [5.17142857 3.17142857], cluster: [5.5 4.2], cluster: [6.45 2.95]

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:*

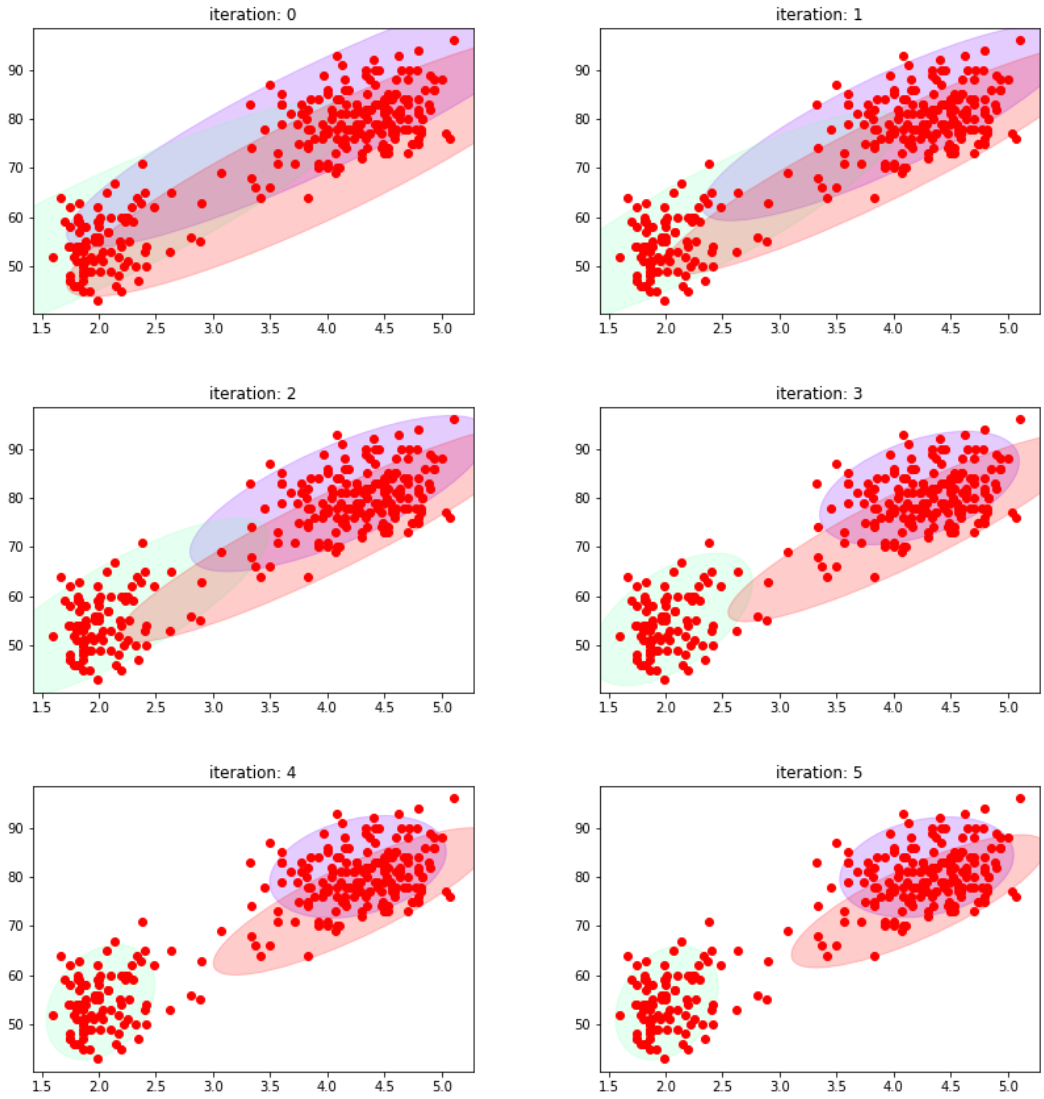
```
import numpy as np
import cv2
from io import StringIO
import task_3_k_mean as k_mean

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~