Project 2 of CSE 473/573

Haowei Zhou 50248857

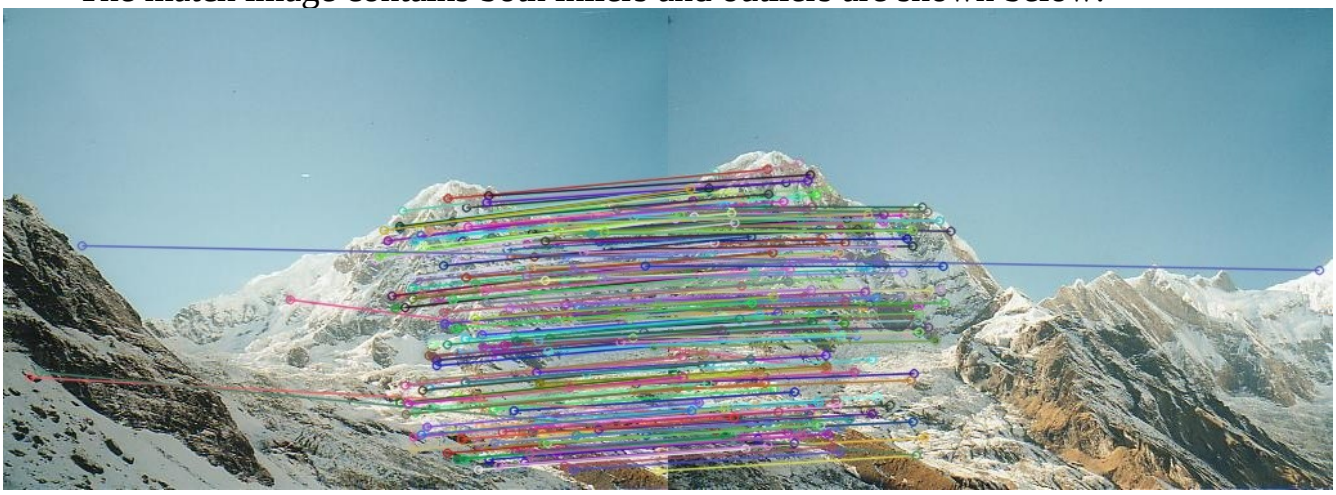
1. Image Features and Homography

1.1 Key Points

The key points of left mountain and right mountain are shown below:

1.2 Match Image

The match image contains both inliers and outliers are shown below:



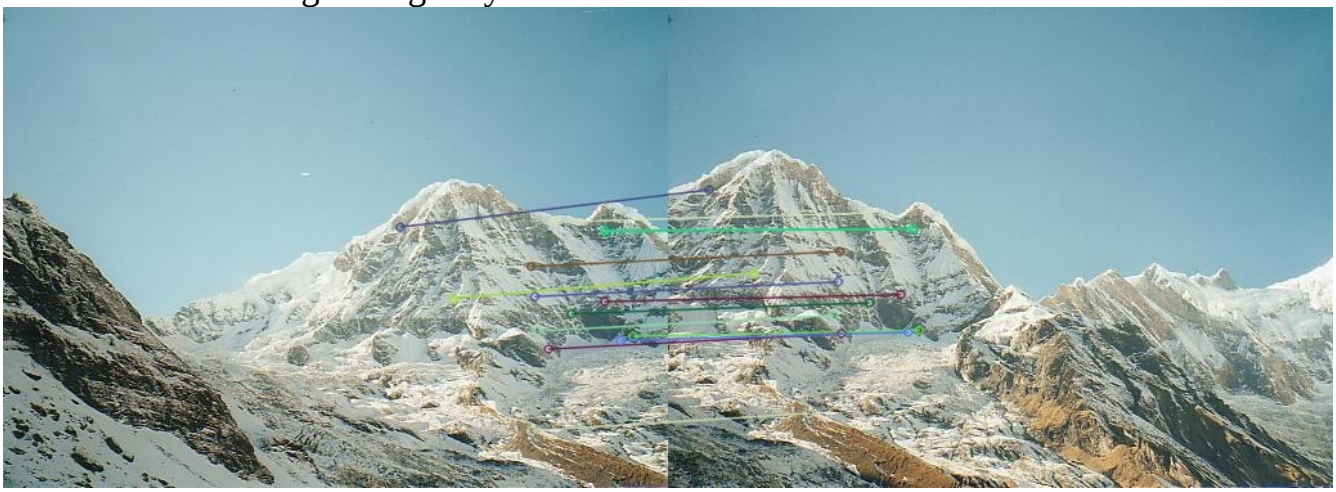
1.3 Homography Matrix H

The homography matrix is shown below:

[[1.59239961e+00 -2.92376916e-01 -3.96644342e+02]
[4.50342898e-01 1.43500407e+00 -1.91260571e+02]
[1.21743590e-03 -5.86467972e-05 1.00000000e+00]]

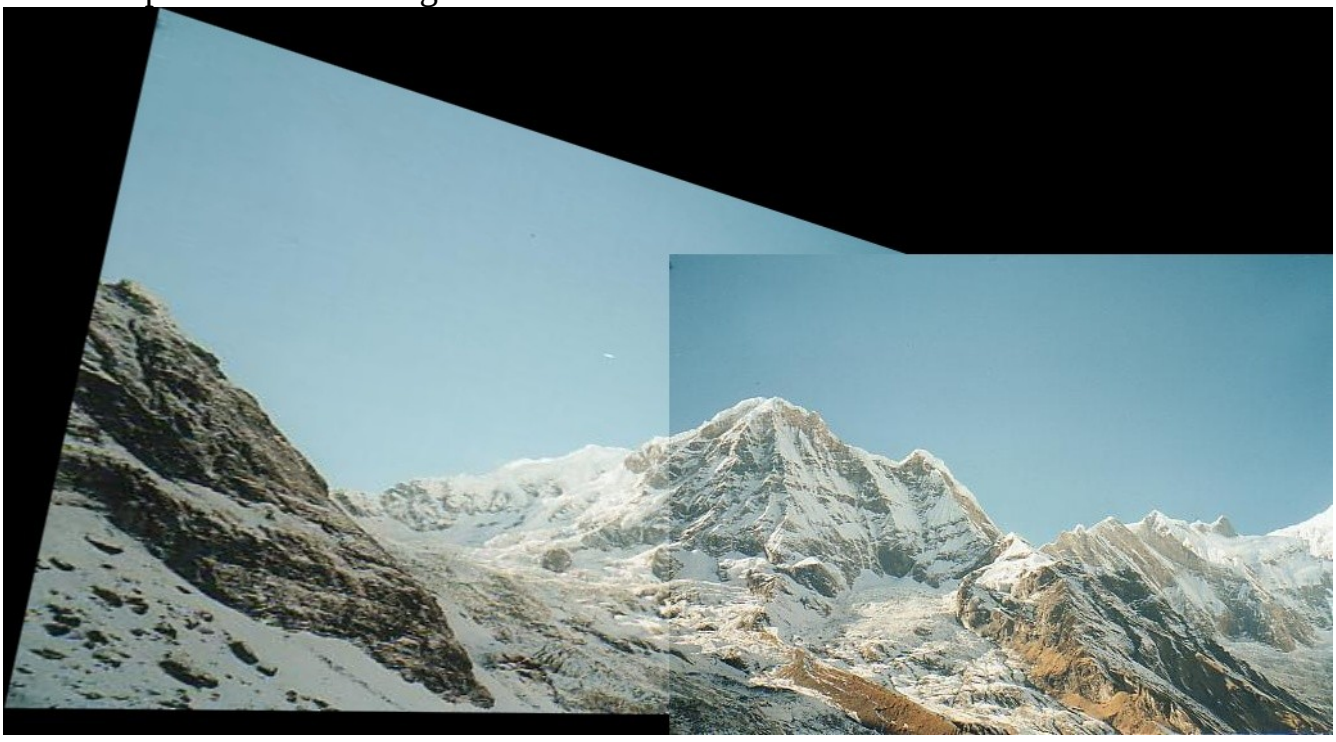
1.4 Match Image Using Only Inliers

The match image using only inliers are shown below:



1.5 Panorama Snitching Result

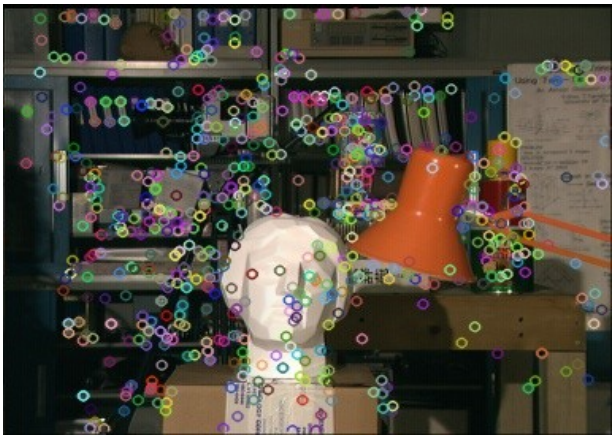
The panorama snitching result is shown below:



2. Epipolar Geometry

2.1 Key Points and Match Image

The key points of left image and right image are shown below:



The match image contains both inliers and outliers are shown below:

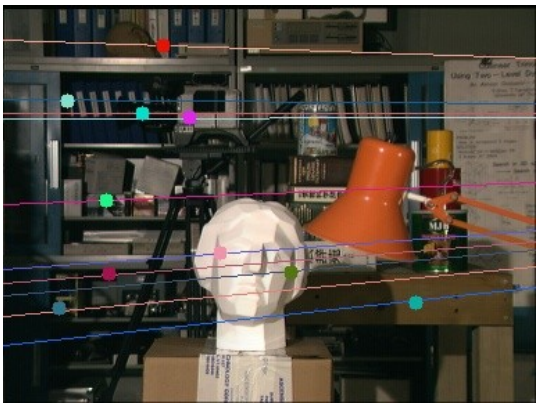
2.2 Fundamental Matrix

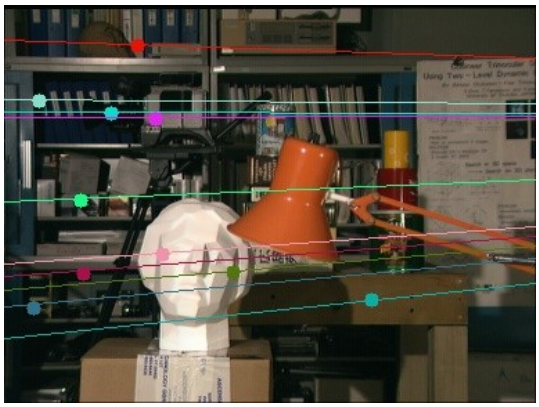
The fundamental matrix is shown below:

$$\begin{bmatrix} -1.07832959e-07 & -7.11948899e-04 & 5.57647085e-02 \\ 7.15827498e-04 & -8.90557705e-05 & -1.11432709e+00 \\ -5.58854787e-02 & 1.10821089e+00 & 1.00000000e+00 \end{bmatrix}$$

2.3 Epilines in Left and Right Image

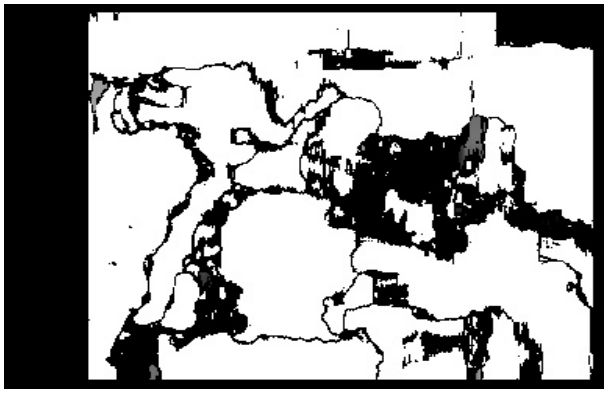
The epilines in left and right images are shown below:





2.4 Disparity Map

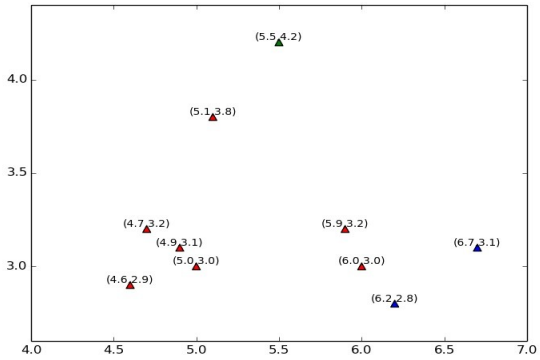
The disparity map is shown below:



3. K-means Clustering

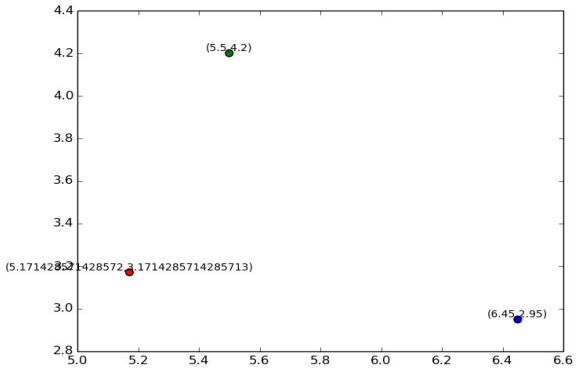
3.1 Classification of the First Iteration

The classification of one iteration result is: A((4.6,2.9),(4.7,3.2),(4.9,3.1),(5.0,3.0),(5.1,3.8),(5.9,3.2),(6.0,3.0)), B((5.5,4.2)), C((6.2,2.8),(6.7,3.1)) the image is shown below:



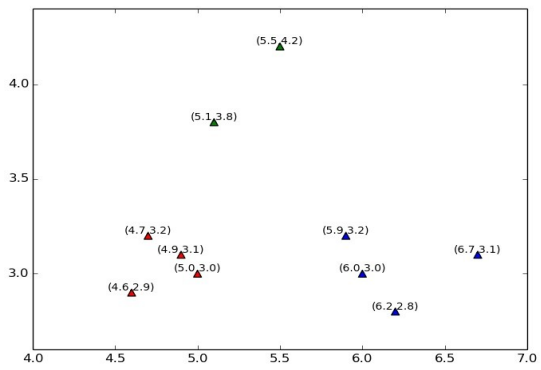
3.2 New Center of the First iteration

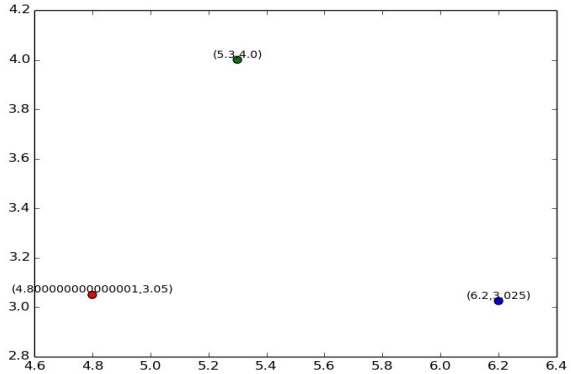
The new center is A(5.17,3.17), B(5.5,4.2), C(6.45,2.95), the result is shown below:



3.3 Classification and New Center for the Second iteration

The classification and new center results are shown below:





3.4 Color Quantization

The color quantization results which k=3,5,10,20 are shown below:

4. Bonus

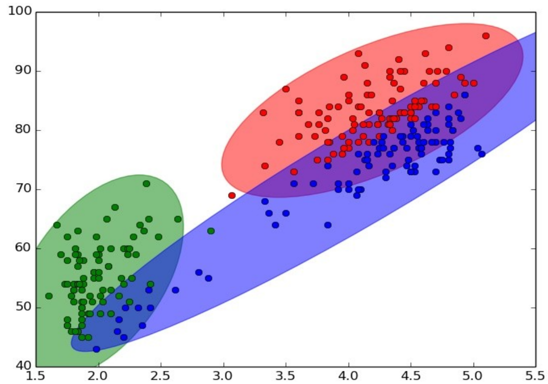
4.1 Mean Value of the First Iteration

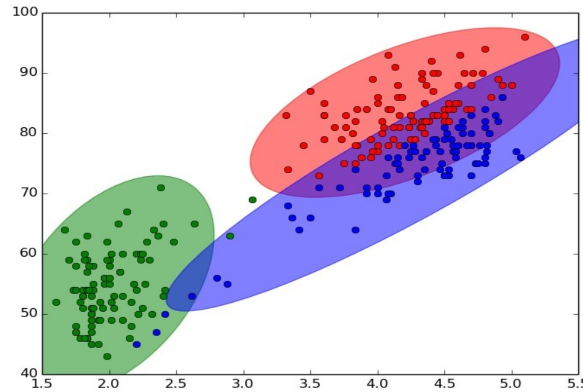
The mean value after the first iteration is:

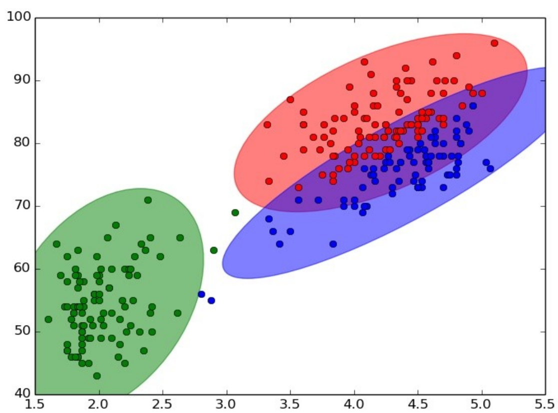
[[5.3165079 3.21527292]
 [5.61129795 3.38505311]
 [5.60443565 3.14420061]]

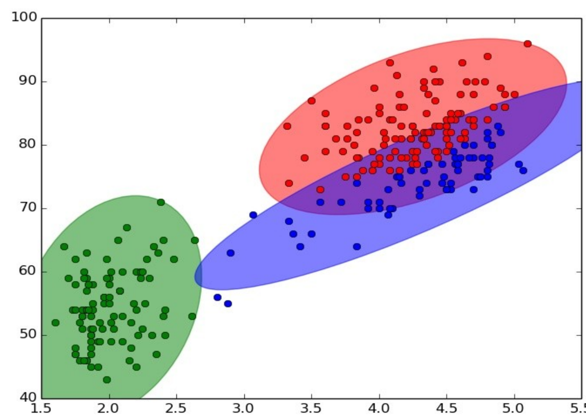
4.2 First Five Iteration Results

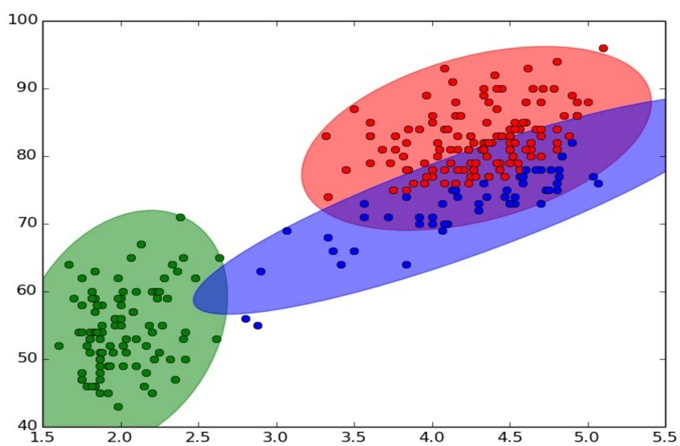
The first five iteration results are :


100
90
80
70
60
50
40
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5


100
90
80
70
60
50
40
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5


100
90
80
70
60
50
40
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5


100
90
80
70
60
50
40
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5


100
90
80
70
60
50
40
1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5

5. Code

5.1 Task1

```
UBIT = "haoweizh"
import numpy as np
import cv2
import os
np.random.seed(sum([ord(c) for c in UBIT]))

def mkdir(path):
    folder = os.path.exists(path)
    if not folder:
        os.makedirs(path)

def generate_Keypoints(imgpath,outputpath):
    img = cv2.imread(imgpath)
    sift = cv2.xfeatures2d.SIFT_create()
    kp,des = sift.detectAndCompute(img,None)
    imgkp = cv2.drawKeypoints(img,kp,img)
    cv2.imwrite(outputpath,imgkp)
    img = cv2.imread(imgpath)
    return img,kp,des

def draw_match(img1,kp1,des1,img2,kp2,des2,outputmatch):
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2,k=2)
    good1 = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good1.append(m)
    good2 = np.expand_dims(good1,1)
    img = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good2,None,flags=2)
    cv2.imwrite(outputmatch,img)
    return good1,good2,img

def get_homography_matrix(kp1,kp2,good1):
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good1 ]).reshape(-1,1,2)
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good1 ]).reshape(-1,1,2)
    H,mask = cv2.findHomography(src_pts,dst_pts,cv2.RANSAC,5.0)
    print H
    return H,mask


def draw_inliers(ranmatchnum,img1,kp1,img2,kp2,good1,H,mask,inlierpath):
    matchesMask = mask.ravel().tolist()
    h,w,d = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv2.perspectiveTransform(pts,H)
    masklen = len(matchesMask)
    partmatchesMask = []
    partgood1 = []
    for i in range(0,ranmatchnum):
        index = np.random.randint(0,masklen)
        partmatchesMask.append(matchesMask[index])
        partgood1.append(good1[index])
    draw_params = dict(matchesMask = partmatchesMask,flags = 2)
    img = cv2.drawMatches(img1,kp1,img2,kp2,partgood1,None,**draw_params)
    cv2.imwrite(inlierpath,img)
```

```
def transfer(H,dx,dy):
    transfermat = np.array([[1,0,dx],[0,1,dy],[0,0,1]])
    return np.dot(transfermat,H)

def splice(img1,img2,H,panopath):
    dx = 700
    dy = 700
    transfermat = transfer(H,dx,dy)
    wrap = cv2.warpPerspective(img1,transfermat,(img1.shape[1]+img2.shape[1]+dx,img2.shape[0]+img2.shape[0]+dy))
    wrap[dx:img1.shape[0]+dx,dy:img1.shape[1]+dy] = img2
    rows,cols = np.where(wrap[:,:,0] != 0)
    min_row,max_row = min(rows),max(rows)+1
    min_col,max_col = min(cols),max(cols)+1
    wrap = wrap[min_row:max_row,min_col:max_col,:]
    cv2.imwrite(panopath,wrap)

if __name__ == "__main__":
    folder = "part1_result"
    mkdir(folder)
    imgpath1 = "data/mountain1.jpg"
    outputpath1 = "part1_result/task1_sift1.jpg";
    img1,kp1,des1 = generate_Keypoints(imgpath1,outputpath1);
    imgpath2 = "data/mountain2.jpg"
    outputpath2 = "part1_result/task1_sift2.jpg";
    img2,kp2,des2 = generate_Keypoints(imgpath2,outputpath2);
    outputmatch = "part1_result/task1_matches_knn.jpg"
    good1,good2,img = draw_match(img1,kp1,des1,img2,kp2,des2,outputmatch)
    H,mask = get_homography_matrix(kp1,kp2,good1)
    inlierpath = "part1_result/task1_matches.jpg"
    ran_match_number = 15
    draw_inliers(ran_match_number,img1,kp1,img2,kp2,good1,H,mask,inlierpath)
    panopath = "part1_result/task1_pano.jpg"
    splice(img1,img2,H,panopath)
```

5.2 Task2

```
UBIT = "haoweizh"
import numpy as np
import cv2
import os
np.random.seed(sum([ord(c) for c in UBIT]))

def mkdir(path):
    folder = os.path.exists(path)
    if not folder:
        os.makedirs(path)

def generate_Keypoints(imgpath,outputpath):
    img = cv2.imread(imgpath)
    sift = cv2.xfeatures2d.SIFT_create()
    kp,des = sift.detectAndCompute(img,None)
    imgkp = cv2.drawKeypoints(img,kp,img)
    cv2.imwrite(outputpath,imgkp)
    img = cv2.imread(imgpath)
    return img,kp,des

def draw_match(img1,kp1,des1,img2,kp2,des2,outputmatch):
    bf = cv2.BFMatcher()
```

```
    matches = bf.knnMatch(des1,des2,k=2)
    good1 = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good1.append(m)
    good2 = np.expand_dims(good1,1)
    img = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good2,None,flags=2)
    cv2.imwrite(outputmatch,img)
    return good1,good2,img,matches

def get_fundamental_matrix(kp1,kp2,matches):
    pts1 = []
    pts2 = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            pts1.append(kp1[m.queryIdx].pt)
            pts2.append(kp2[m.trainIdx].pt)
    pts1 = np.int32(pts1)
    pts2 = np.int32(pts2)
    F,mask = cv2.findFundamentalMat(pts1,pts2,cv2.RANSAC)
    print F
    return F,mask,pts1,pts2

def drawlines(img1,img2,lines,pts1,pts2):
    r,c,d = img1.shape
    for r,pt1,pt2 in zip(lines,pts1,pts2):
        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)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2

def get_epiline(numpairs,img1,img2,pts1,pts2,leftpath,rightpath):
    pts1 = pts1[mask.ravel()==1]
    pts2 = pts2[mask.ravel()==1]
    ptsnum = pts1.shape[0]
    partpts1 = np.zeros(shape=(0,2),dtype='int32')
    partpts2 = np.zeros(shape=(0,2),dtype='int32')
    for i in range(0,numpairs):
        index = np.random.randint(0,ptsnum)
        pt1 = pts1[index]
        pt1 = pt1[np.newaxis,:]
        pt2 = pts2[index]
        pt2 = pt2[np.newaxis,:]
        partpts1 = np.r_[partpts1,pt1]
        partpts2 = np.r_[partpts2,pt2]
    lines1 = cv2.computeCorrespondEpilines(partpts2.reshape(-1,1,2),2,F)
    lines1 = lines1.reshape(-1,3)
    img3,img4 = drawlines(img1,img2,lines1,partpts1,partpts2)
    lines2 = cv2.computeCorrespondEpilines(partpts1.reshape(-1,1,2),1,F)
    lines2 = lines2.reshape(-1,3)
    img5,img6 = drawlines(img2,img1,lines2,partpts2,partpts1)
    cv2.imwrite(leftpath,img3)
    cv2.imwrite(rightpath,img5)

def disparity(img1,img2,disparitypath):
    stereo = cv2.StereoBM_create(numDisparities=48,blockSize=15)
```

```
    disparity = stereo.compute(img1,img2)
    cv2.imwrite(disparitypath,disparity)

if __name__ == "__main__":
    folder = "part2_result"
    mkdir(folder)
    imgpath1 = "data/tsucuba_left.png"
    outputpath1 = "part2_result/task2_sift1.jpg";
    img1,kp1,des1 = generate_Keypoints(imgpath1,outputpath1);
    imgpath2 = "data/tsucuba_right.png"
    outputpath2 = "part2_result/task2_sift2.jpg";
    img2,kp2,des2 = generate_Keypoints(imgpath2,outputpath2);
    outputmatch = "part2_result/task2_matches_knn.jpg"
    good1,good2,img,matches = draw_match(img1,kp1,des1,img2,kp2,des2,outputmatch)
    F,mask,pts1,pts2 = get_fundamental_matrix(kp1,kp2,matches)
    leftpath = "part2_result/task2_epi_left.jpg"
    rightpath = "part2_result/task2_epi_right.jpg"
    numpairs = 10
    get_epiline(numpairs,img1,img2,pts1,pts2,leftpath,rightpath)
    disparitypath = "part2_result/task2_disparity.jpg"
    img1 = cv2.imread(imgpath1,0)
    img2 = cv2.imread(imgpath2,0)
    disparity(img1,img2,disparitypath)
```

5.3 Task3

```
UBIT = "haoweizh"
import numpy as np
import os
import sys
import cv2
from matplotlib import pyplot as plt
np.random.seed(sum([ord(c) for c in UBIT]))

X = 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]])

center = np.array([
[6.2,3.2],
[6.6,3.7],
[6.5,3.0]])

color = ['r','g','b']

def mkdir(path):
    folder = os.path.exists(path)
    if not folder:
        os.makedirs(path)

def reclassify(savepath):
```

```
    kind2index = dict()
    for i in range(0,X.shape[0]):
        dist = sys.maxint
        kind = -1
        for j in range(0,center.shape[0]):
            d = (X[i][0]-center[j][0])**2+(X[i][1]-center[j][1])**2
            if d<dist:
                dist = d
                kind = j
        if kind2index.has_key(kind)==False:
            kind2index[kind] = []
        kind2index[kind].append(X[i])
    ax = plt.subplot()
    for elem in kind2index:
        x_list = []
        y_list = []
        pts = kind2index[elem]
        for pt in pts:
            x_list.append(pt[0])
            y_list.append(pt[1])
            plt.text(pt[0],pt[1],"("+str(pt[0])+","+str(pt[1])+")",ha='center',va='bottom',fontsize=10)
        ax.scatter(x_list,y_list,c=color[elem],marker='^',s=50,alpha=1)
    plt.savefig(savepath)
    plt.clf()
    return kind2index

def recompute_mean(kind2index,meanpath):
    ax = plt.subplot()
    for elem in kind2index:
        x_list = [0.0]
        y_list = [0.0]
        pts = kind2index[elem]
        for pt in pts:
            x_list[0] = x_list[0]+pt[0]
            y_list[0] = y_list[0]+pt[1]
        x_list[0] = x_list[0]/len(pts)
        y_list[0] = y_list[0]/len(pts)
        center[elem][0] = x_list[0]
        center[elem][1] = y_list[0]
        plt.text(x_list[0],y_list[0],"("+str(x_list[0])+","+str(y_list[0])+")",ha='center',va='bottom',fontsize=10)
        ax.scatter(x_list,y_list,c=color[elem],s=50,alpha=1)
    plt.savefig(meanpath)
    plt.clf()

def initcenter(img,k):
    r,c,d = img.shape
    center = np.zeros(shape=(0,d))
    for i in range(0,k):
        x = np.random.randint(0,r)
        y = np.random.randint(0,c)
        p = img[x][y]
        core = p[np.newaxis,:]
        center = np.r_[center,core]
    return center

def classifyimg(center,img):
    result = dict()
    r,c,d = img.shape
    for i in range(0,r):
```

```
        for j in range(0,c):
            dist = sys.maxint
            kind = -1
            for e in range(0,center.shape[0]):
                dt = 0
                for k in range(0,d):
                    dt += (img[i][j][k]-center[e][k])**2
                if dt<dist:
                    dist = dt
                    kind = e
            if result.has_key(kind)==False:
                result[kind] = []
            result[kind].append([i,j])
    return result

def recompute_center(img,classify,center):
    r,c,d = img.shape
    newcenter = np.zeros(shape=(0,d))
    for elem in classify:
        pts = classify[elem]
        ct = np.array([0,0,0])
        for pt in pts:
            for t in range(0,d):
                ct[t] = ct[t]+img[pt[0]][pt[1]][t]
        for t in range(0,d):
            ct[t] = ct[t]/len(pts)
        ct = ct[np.newaxis,:]
        newcenter = np.r_[newcenter,ct]
    return newcenter

def compare(oldcenter,newcenter):
    return (oldcenter==newcenter).all()

def color_quant(imgpath,outpath,k):
    img = cv2.imread(imgpath)
    oldcenter = initcenter(img,k)
    flag = True
    while flag==True:
        classify = classifyimg(oldcenter,img)
        newcenter = recompute_center(img,classify,oldcenter)
        if compare(oldcenter,newcenter)==True:
            r,c,d = img.shape
            for elem in classify:
                pts = classify[elem]
                for pt in pts:
                    for t in range(0,d):
                        img[pt[0]][pt[1]][t] = newcenter[elem][t]
            flag = False
        else:
            oldcenter = newcenter
    cv2.imwrite(outpath,img)
    print "save image "+str(k)

if __name__ == "__main__":
    folder = "part3_result"
    mkdir(folder)
    savepath = "part3_result/task3_iter1_a.jpg"
    kind2index = reclassify(savepath)
    meanpath = "part3_result/task3_iter1_b.jpg"
```

```
    recompute_mean(kind2index,meanpath)
    savepath = "part3_result/task3_iter2_a.jpg"
    kind2index = reclassify(savepath)
    meanpath = "part3_result/task3_iter2_b.jpg"
    recompute_mean(kind2index,meanpath)
    imgpath = "data/baboon.jpg"
    outpath = "part3_result/task3_baboon_3.jpg"
    color_quant(imgpath,outpath,3)
    outpath = "part3_result/task3_baboon_5.jpg"
    color_quant(imgpath,outpath,5)
    outpath = "part3_result/task3_baboon_10.jpg"
    color_quant(imgpath,outpath,10)
    outpath = "part3_result/task3_baboon_20.jpg"
    color_quant(imgpath,outpath,20)
```

5.4 bonus1

```
import numpy as np
import copy
from scipy.stats import multivariate_normal

X = 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]])

u = np.array([
[6.2,3.2],
[6.6,3.7],
[6.5,3.0]])

var = np.array([
[[0.5,0],
[0,0.5]],
[[0.5,0],
[0,0.5]],
[[0.5,0],
[0,0.5]]
])

pi = np.array([1.0/3,1.0/3,1.0/3])

def Expectation(localu,localvar,localpi):
    probability = []
    for i in range(0,len(X)):
        total = 0.0
        for j in range(0,len(u)):
            total += localpi[j]*multivariate_normal.pdf(X[i],mean=localu[j],cov=localvar[j])
        prob = []
        for j in range(0,len(u)):
            up = localpi[j]*multivariate_normal.pdf(X[i],mean=localu[j],cov=localvar[j])
            prob.append(up/total)
```

```
        probability.append(prob)
    return probability

def Maximization(probability,localpi):
    newu = copy.deepcopy(u)
    newvar = copy.deepcopy(var)
    newpi = copy.deepcopy(pi)
    for j in range(0,len(newu)):
        Nk = 0.0
        up = [0.0,0.0]
        for i in range(0,len(X)):
            Nk += probability[i][j]
            up += probability[i][j]*X[i]
        newu[j] = up/Nk
        varup = np.array([[0.0,0.0],[0.0,0.0]])
        for i in range(0,len(X)):
            delta = X[i]-newu[j]
            delta_axis = delta[np.newaxis,:]
            varup += probability[i][j]*np.dot(np.transpose(delta_axis),delta_axis)
        newvar[j] = varup/Nk
        newpi[j] = Nk/len(X)
    return newu,newvar,newpi

def GMMOneiter():
    localu = copy.deepcopy(u)
    localvar = copy.deepcopy(var)
    localpi = copy.deepcopy(pi)
    probablity = Expectation(localu,localvar,localpi)
    newu,newvar,newpi = Maximization(probablity,localpi)
    print newu

if __name__ == "__main__":
    GMMOneiter()
```

5.5 bonus2

```
import numpy as np
import os
import copy
from scipy.stats import multivariate_normal
from error_ellipse import plot_point_cov
import matplotlib.pyplot as plt

u = np.array([
[4.0,81],
[2.0,57],
[4.0,71]])

var = np.array([
[[1.3,13.98],
[13.98,184.82]],
[[1.3,13.98],
[13.98,184.82]],
[[1.3,13.98],
[13.98,184.82]]
])

pi = np.array([1.0/3,1.0/3,1.0/3])
```

```
rgbcolor = ['r','g','b']

def mkdir(path):
    folder = os.path.exists(path)
    if not folder:
        os.makedirs(path)

def readfile(path):
    alldata = []
    with open(path) as file:
        line = file.readline()
        while line:
            oneline = line.split()
            data = []
            for i in range(1,len(oneline)):
                data.append(float(oneline[i]))
            alldata.append(data)
            line = file.readline()
    return np.array(alldata)

def Expectation(localu,localvar,localpi,alldata):
    probability = []
    for i in range(0,len(alldata)):
        total = 0.0
        for j in range(0,len(u)):
            total += localpi[j]*multivariate_normal.pdf(alldata[i],mean=localu[j],cov=localvar[j])
        prob = []
        for j in range(0,len(u)):
            up = localpi[j]*multivariate_normal.pdf(alldata[i],mean=localu[j],cov=localvar[j])
            prob.append(up/total)
        probability.append(prob)
    return probability


def Maximization(probability,localpi,alldata):
    newu = copy.deepcopy(u)
    newvar = copy.deepcopy(var)
    newpi = copy.deepcopy(pi)
    for j in range(0,len(newu)):
        Nk = 0.0
        up = [0.0,0.0]
        for i in range(0,len(alldata)):
            Nk += probability[i][j]
            up += probability[i][j]*alldata[i]
        newu[j] = up/Nk
        varup = np.array([[0.0,0.0],[0.0,0.0]])
        for i in range(0,len(alldata)):
            delta = alldata[i]-newu[j]
            delta_axis = delta[np.newaxis,:]
            varup += probability[i][j]*np.dot(np.transpose(delta_axis),delta_axis)
        newvar[j] = varup/Nk
        newpi[j] = Nk/len(alldata)
    return newu,newvar,newpi

def OneIter(localu,localvar,localpi,alldata,count):
    probability = Expectation(localu,localvar,localpi,alldata)
    newu,newvar,newpi = Maximization(probability,localpi,alldata)
    kind2index = classify(probability)
    draw(alldata,kind2index,count)
```

```
    return newu,newvar,newpi

def classify(probability):
    kind2index = dict()
    for i in range(0,len(probability)):
        kind = -1
        maxprob = 0.0
        for j in range(0,len(probability[i])):
            if probability[i][j]>maxprob:
                maxprob = probability[i][j]
                kind = j
        if kind not in kind2index:
            kind2index[kind] = []
        kind2index[kind].append(i)
    return kind2index

def draw(alldata,kind2index,count):
    for elem in kind2index:
        data = []
        for index in kind2index[elem]:
            data.append(alldata[index])
        data = np.array(data)
        x,y = np.array(data).T
        plt.plot(x, y, 'ro',color=rgbcolor[elem])
        plot_point_cov(data, nstd=3, alpha=0.5, color=rgbcolor[elem])
    plt.savefig("bonus_result/task3_gmm_iter"+str(count)+".jpg")
    plt.clf()

if __name__ == "__main__":
    folderpath = "bonus_result"
    mkdir(folderpath)
    alldata = readfile("data.txt")
    localu = copy.deepcopy(u)
    localvar = copy.deepcopy(var)
    localpi = copy.deepcopy(pi)
    count = 1
    while count<=5:
        newu,newvar,newpi = OneIter(localu,localvar,localpi,alldata,count)
        localu = newu
        localvar = newvar
        localpi = newpi
        count += 1
```