ECE 66100 HW5 Report

Zhengxin Jiang
(jiang839@purdue.edu)

October 13, 2022

1 Theory Questions

Differentiate inliers and outliers

To differentiate between the inliers and the outliers, we perform a large number of trails. For each of the trails, we randomly pick a few interest point pairs and compute an estimated homography based on the picked points. Then we apply the estimated homography to the whole set of the correspondences and check whether each estimated point lies within the threshold range of the actual point. With a relatively large number of trails, we can pick the best trail that gives the most inliers.

LM algorithm

The LM algorithm takes advantages from both GD and GM by introducing the damping coefficient μ. The damping coefficient μ is adjusted during the iterations. When the algorithm starts, LM takes the step which is close to GD. When the training is close to the minimum, the damping coefficient is set smaller so that the step is more close to GN.

2 Implementation

RANSAC algorithm

The RANSAC algorithm finds the homography between two images by first finding the interest point correspondences, then rejecting the outliers of the correspondences and using the inliers to estimate the homography.

For interest point extraction and correspondences, SIFT is used to find the interest points and the brute-force matcher in OpenCV is used to find the correspondences. Since the brute-force matcher usually finds thousands of correspondences between two images, the correspondences found by the brute-force matcher are sorted based on the distance and the top 500 correspondences with lowest distaced are used in further steps.

To reject outliers in the correspondences, N random trails are performed as described in the theory question. The selection of N needs to satisfy the probability that at least one trail picks all inliers for homography estimation. This gives

$$N = \frac{ln(1-p)}{ln[1-(1-\epsilon)^n]}$$

With $n = 6$, $\epsilon = 0.4$, $p = 0.99$, N is given around 100.

The final step is using the inliers to calculate the homography using linear least-square method or further LM refinement.

Least-Squares method

In this assignment I choose to use the homogeneous system for the linear least-square method. We use the system similar with which is used in hw2 but this time with n correspondences:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ & & & & \vdots & & & & \\ -x_n & -y_n & -1 & 0 & 0 & 0 & x_nx_n' & y_nx_n' & x_n' \\ 0 & 0 & 0 & -x_n & -y_n & -1 & x_ny_n' & y_ny_n' & y_n' \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The homography $\mathbf{h}$ is solved using SVD. The last equation of the system is to prevent the trivial solution $\mathbf{h} = 0$.

LM Refinement

As desribed in the theory question, the LM algorithm takes advantages from both gradient descent and Gaussian-Newton. When the training starts, the LM algorithm takes steps as gradient descent. When the training is approaching the minima, the LM algorithm takes steps more like Gaussian-Newton. In this assignment the optimizer from scipy is used to implement LM Refinement.

3 Results on Given Images

Correspondences between adjacent images

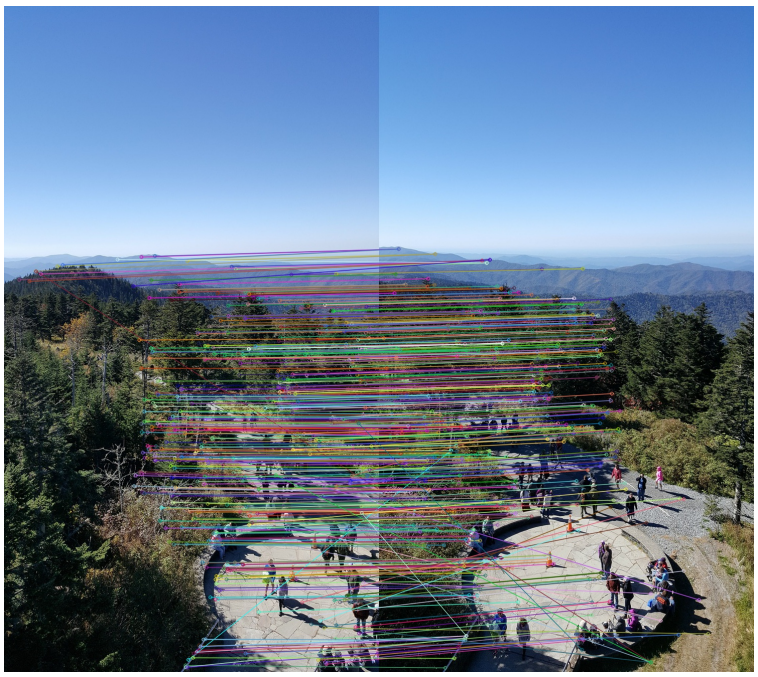

Figure 1: Correspondences between image 1 and 2

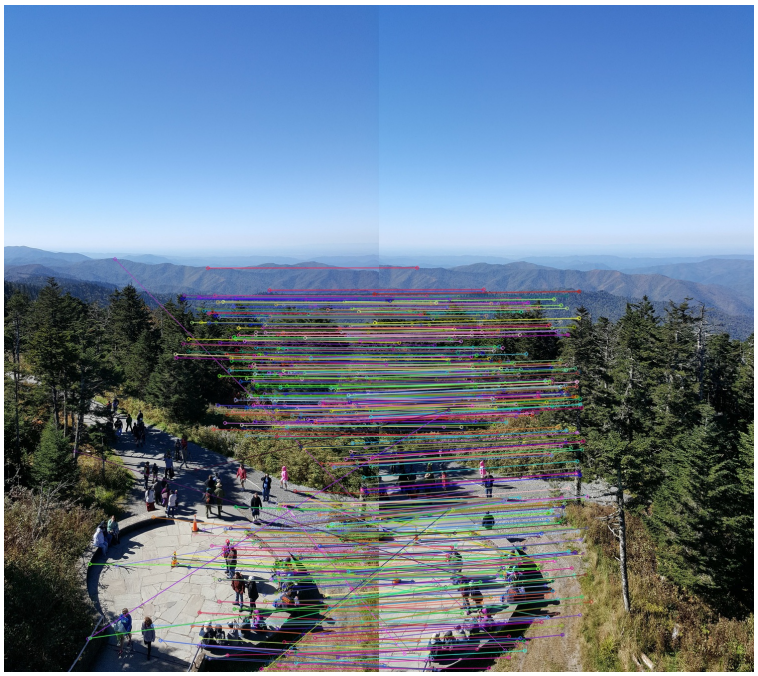

Figure 2: Correspondences between image 2 and 3

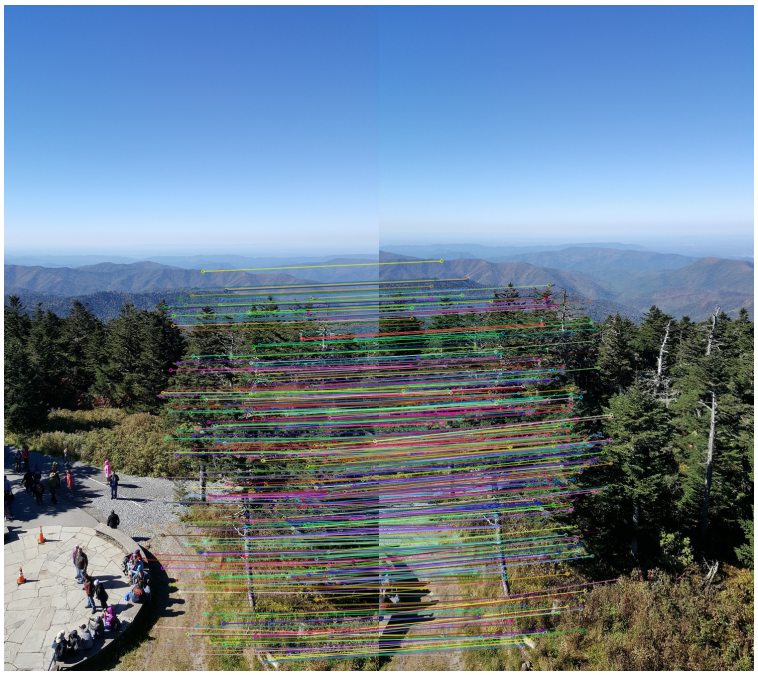

Figure 3: Correspondences between image 3 and 4

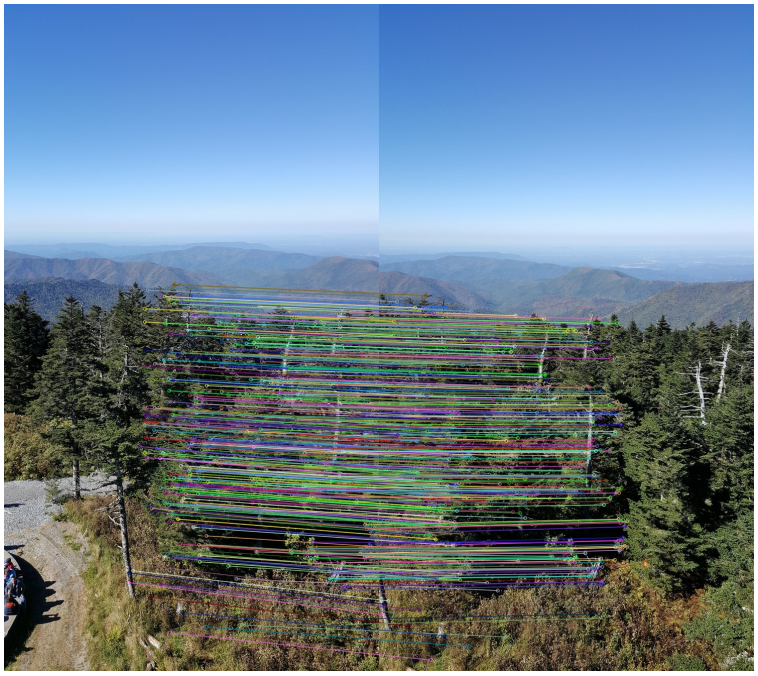

Figure 4: Correspondences between image 4 and 5

Inliers and outliers

In all result images, inliers are marked as green lines and outliers are marked as red lines.

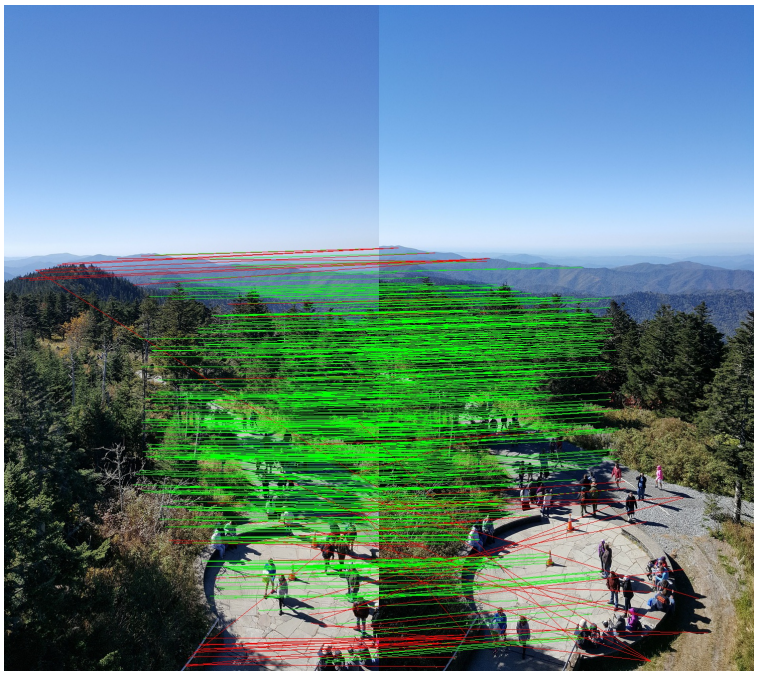

Figure 5: Inliers and outliers for image 1 and 2

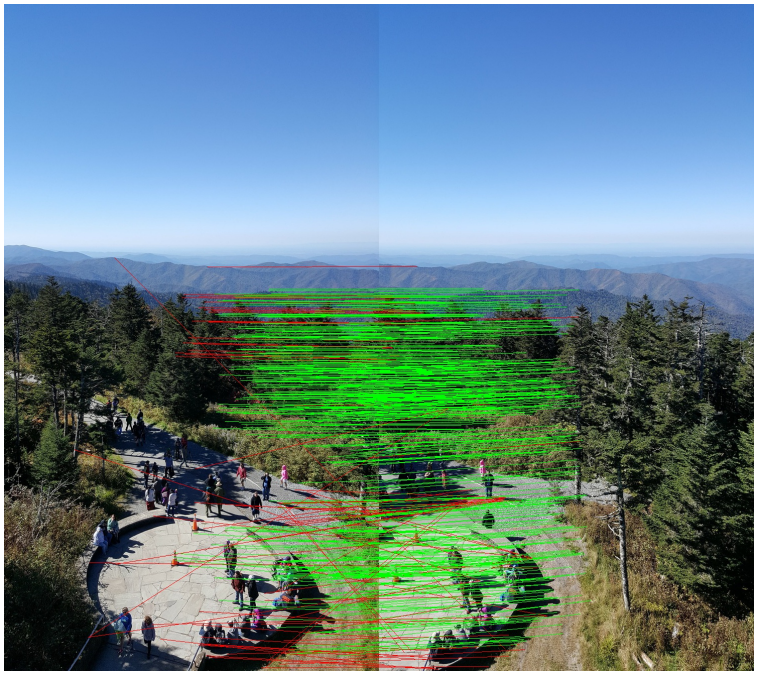

Figure 6: Inliers and outliers for image 2 and 3

Panorama

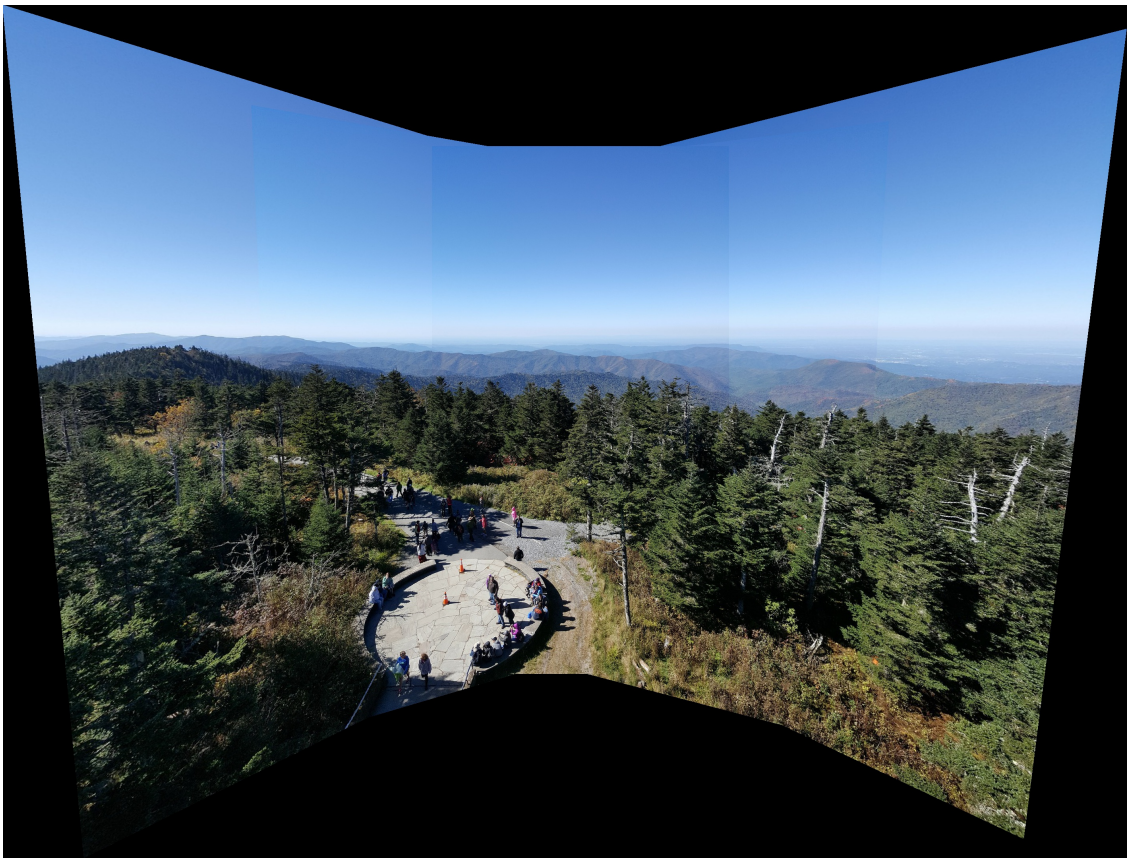

Figure 7: Panorama for the given image set

4 Results on Self-took Images

Original images

My self-took images are five images of my desktop monitor.

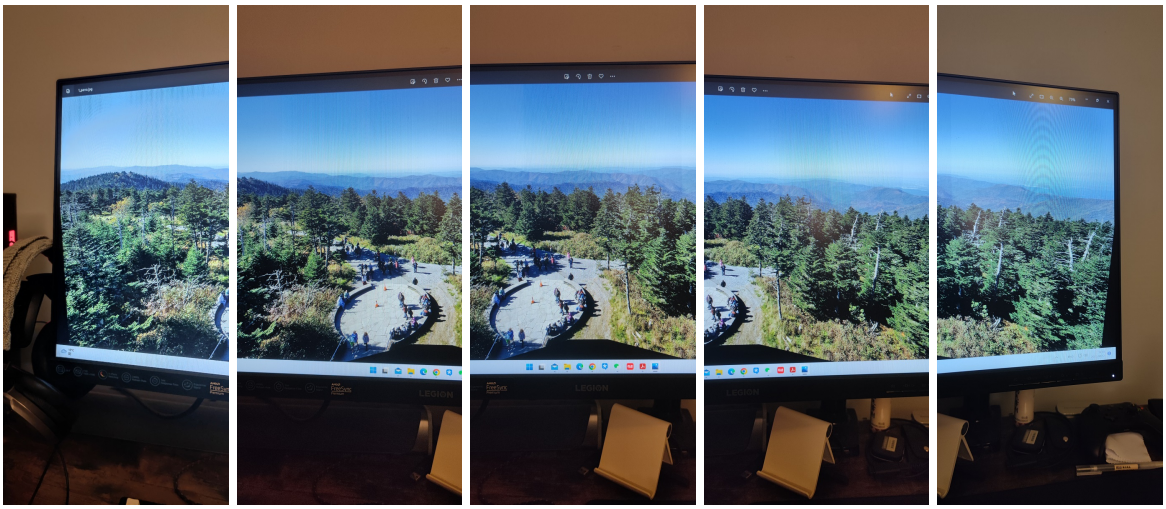

Figure 8: The set of my self-took images

Correspondences between adjacent images

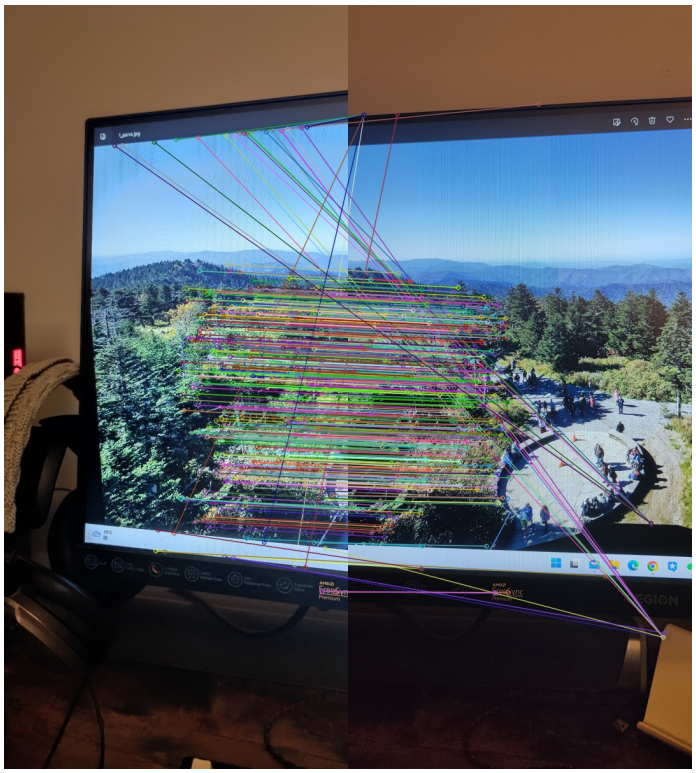

Figure 9: Correspondences between image 1 and 2

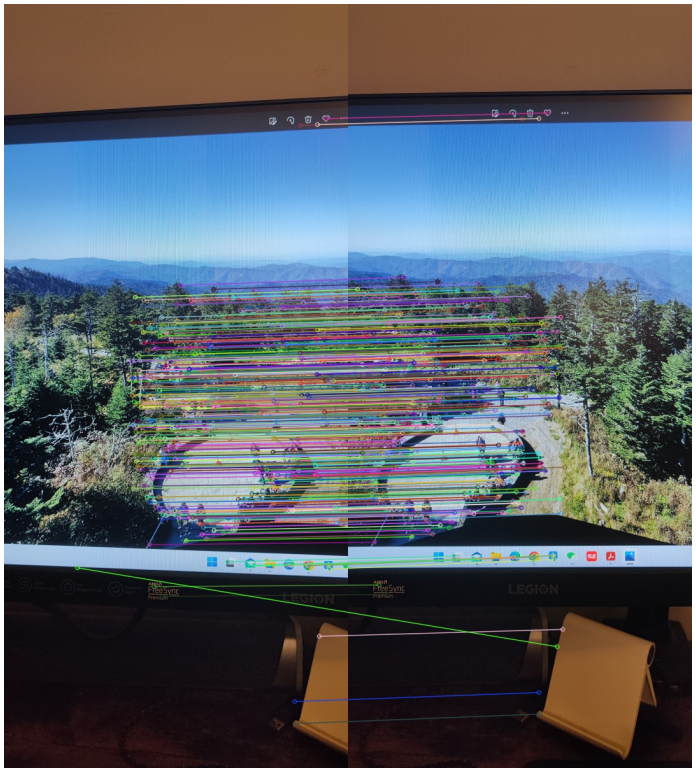

Figure 10: Correspondences between image 2 and 3

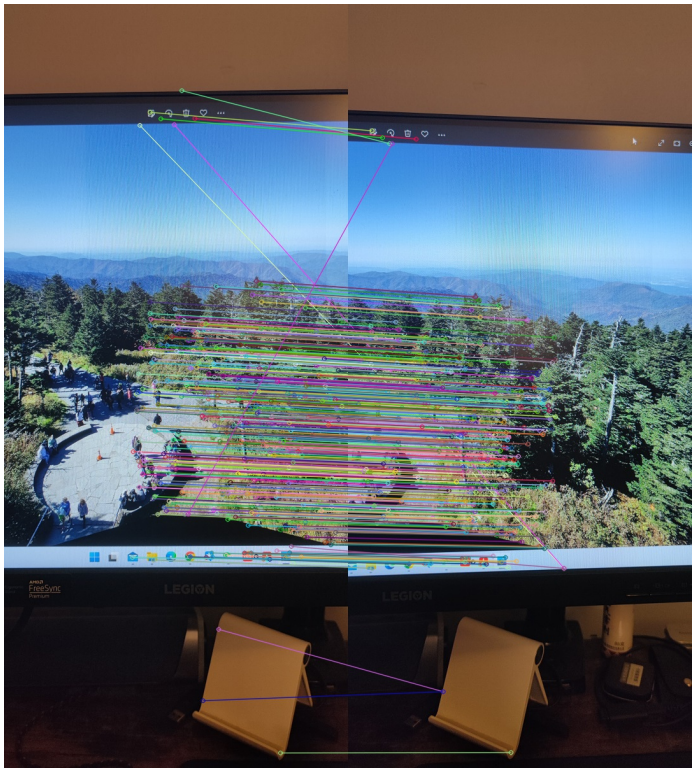

Figure 11: Correspondences between image 3 and 4

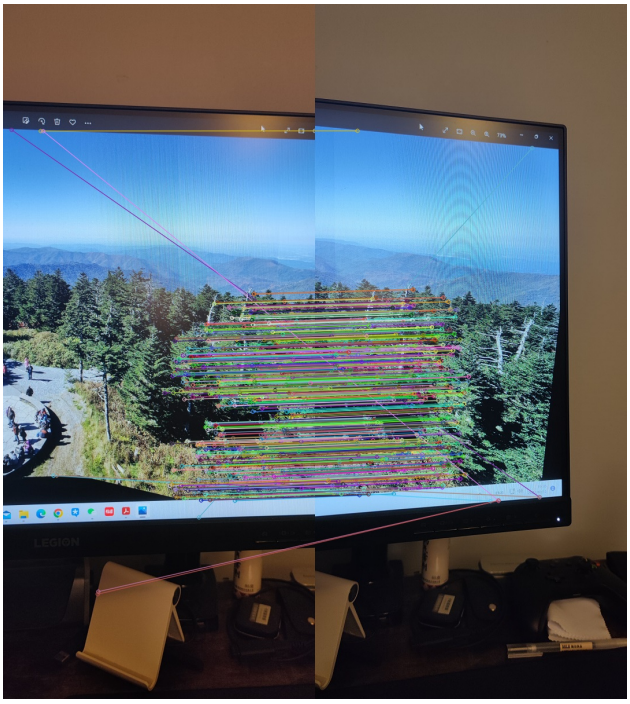

Figure 12: Correspondences between image 4 and 5

Inliers and outliers

In all result images, inliers are marked as green lines and outliers are marked as red lines.

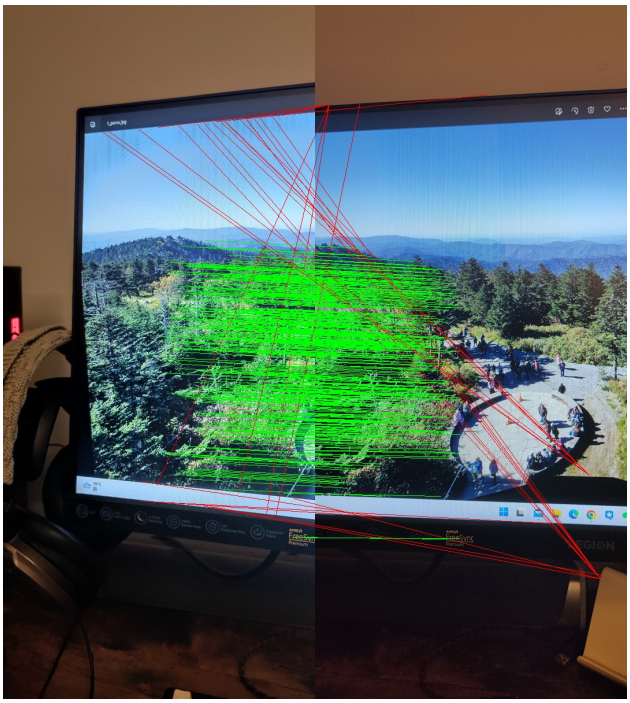

Figure 13: Inliers and outliers for image 1 and 2

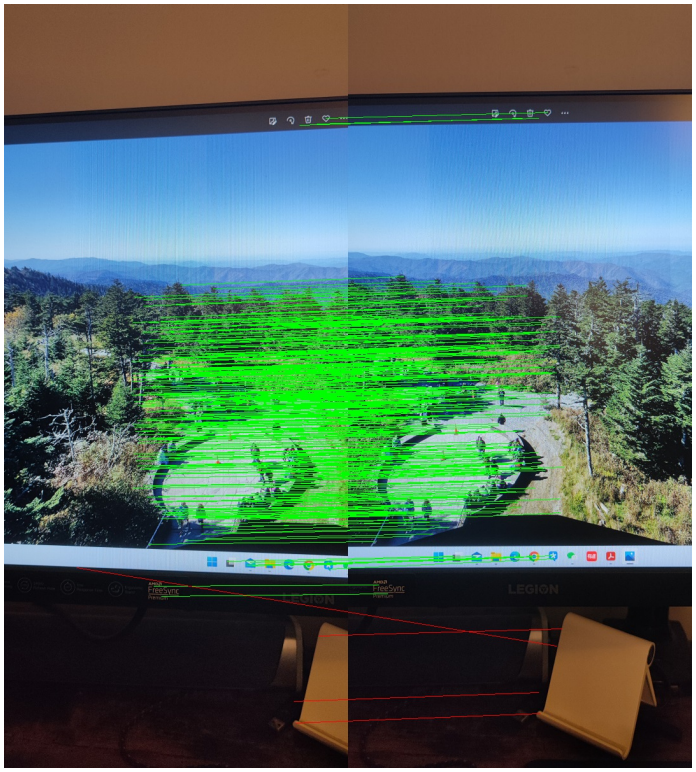

Figure 14: Inliers and outliers for image 2 and 3

Panorama

Figure 15: Panorama for the self-took image set

5 Optimal set of parameters

For RANSAC number of trails, $N = 100$ is used, which should be higher than the requirement to make $p > 0.99$.

For the inlier outlier threshold, $\delta = 3.0$ is used.

For the sample size of one trail, 6 samples are picked every time.

The above parameters give very good results in practice.

6 Source code

```
# ECE661 HW5
# Zhengxin Jiang
# jiang839

import cv2
import numpy as np
import matplotlib . pyplot as plt
import math
import scipy
from scipy import optimize


### function definetion ###

# The SIFT method
def sift(img1, img2):

    sift1 = cv2.xfeatures2d.SIFT_create()
    sift2 = cv2.xfeatures2d.SIFT_create()

    kp1, des1 = sift1.detectAndCompute(img1,None)
    kp2, des2 = sift2.detectAndCompute(img2,None)

    bf = cv2.BFMatcher()
    matches = bf.match(des1,des2)
    matches = sorted(matches, key = lambda x:x.distance)

    return kp1, kp2, matches


# draw interest points correspondences using SIFT
def drawCorrespondences(img1, img2):

    kp1, kp2, matches = sift(img1, img2)
    img_corr = cv2.drawMatches(img1,kp1,img2,kp2,matches[:500],None,flags=2)

    return img_corr


# The function takes two arrays of points and return the homography from 1 to 2
def findHomograpy_LeastSquare(points_1, points_2):

    P = np.zeros((len(points_1)*2+1, 9))

    # construct a homogenious system
    for i in range(len(points_1)):

        P[2*i, 0] = -points_1[i][0]
        P[2*i, 1] = -points_1[i][1]
        P[2*i, 2] = -1
        P[2*i, 6] = points_1[i][0]*points_2[i][0]
        P[2*i, 7] = points_1[i][1]*points_2[i][0]
```

```
        P[2*i, 8] = points_2[i][0]

        P[2*i+1, 3] = -points_1[i][0]
        P[2*i+1, 4] = -points_1[i][1]
        P[2*i+1, 5] = -1
        P[2*i+1, 6] = points_1[i][0]*points_2[i][1]
        P[2*i+1, 7] = points_1[i][1]*points_2[i][1]
        P[2*i+1, 8] = points_2[i][1]

    P[-1, -1] = 1

    b = np.zeros(len(points_1)*2+1)
    b[-1] = 1

    # solving H using least square method
    H = np.linalg.lstsq(P, b, rcond=None)[0]
    H = H/H[-1]
    H = np.reshape(H, (3,3))

    return H


# Cost function for LM
def costFunc(H, p1, p2):

    p1 = np.append(p1, np.ones((len(p1),1)), axis=1).T
    p2 = np.append(p1, np.ones((len(p2),1)), axis=1).T

    p2_est = H.dot(p1)
    p2_est = (p2_est/p2_est[-1])

    err = np.abs(np.subtract(p2_est.T - p1.T))**2

    return err

# Estimate the homography from image1 to image2 using RANSAC
def homography_ransac(img1, img2, LM = False, drawliers = False, outimgname = ''):

    kp1, kp2, matches = sift(img1, img2)

    # parameters
    N_trails = 100
    delta = 3.0
    sample_size = 6
    total_matches = 500

    # perform N trails to find inliers
    trail_inliernum = []
    trail_matchidx = []
    for trail in range(N_trails):
        # randomly pick some samples
        sample_idx = np.random.randint(total_matches, size = sample_size)

        sample_p1 = []
```

```
        sample_p2 = []
        for i in range(sample_size):
            sample_p1.append(kp1[matches[sample_idx[i]].queryIdx].pt)
            sample_p2.append(kp2[matches[sample_idx[i]].trainIdx].pt)

        sample_H = findHomograpy_LeastSquare(sample_p1, sample_p2)

        # calculate distance using the estimated homography
        dist = []
        for i in range(total_matches):

            p1 = kp1[matches[i].queryIdx].pt
            p2 = kp2[matches[i].trainIdx].pt

            est_p2 = sample_H.dot(p1+(1,))
            est_p2 = est_p2/est_p2[-1]

            dist.append(np.sqrt((est_p2[0]-p2[0])**2 + (est_p2[1]-p2[1])**2))

        # find the inliers under the threshold
        matchidx = np.flatnonzero(np.array(dist) < delta)
        trail_inliernum.append(len(matchidx))
        trail_matchidx.append(matchidx)

    # pick the best trail with most inliers
    best_trail = np.argmax(trail_inliernum)

    # calculate the homography
    p1 = []
    p2 = []
    p1_LM = []
    p2_LM = []

    for match_idx in trail_matchidx[best_trail]:
        p1.append(kp1[matches[match_idx].queryIdx].pt)
        p2.append(kp2[matches[match_idx].trainIdx].pt)
        if LM == True:
            p1_LM.append(list(kp1[matches[match_idx].queryIdx].pt))
            p2_LM.append(list(kp2[matches[match_idx].trainIdx].pt))

    H_ransac = findHomograpy_LeastSquare(p1, p2)

    if LM == True:
        H_ransac = np.flatten(H_ransac)
        H_ransac = optimize.least_squares(costFunc, H_ransac, args=(p1_LM, p2_LM), method
            ='lm').x
        H_ransac = np.reshape(H_ransac, (3,3))

    # draw inliers and outliers if required
    if drawliers == True:

        img_comb = np.concatenate((img1, img2), axis=1)

        for i in range(total_matches):
```

```
            dp1 = kp1[matches[i].queryIdx].pt
            dp2 = kp2[matches[i].trainIdx].pt
            dp2 = list(dp2)
            dp2[0] += img1.shape[1]

            # draw green lines for inliers and red lines for outliers
            if i in trail_matchidx[best_trail]:
                cv2.line(img_comb,(int(dp1[0]),int(dp1[1])),(int(dp2[0]),int(dp2[1]))
                    ,(0,255,0))
            else:
                cv2.line(img_comb,(int(dp1[0]),int(dp1[1])),(int(dp2[0]),int(dp2[1]))
                    ,(0,0,255))

        cv2.imwrite('C:/Users/jzx/OneDrive␣-␣purdue.edu/ECE661/hw5/result␣images/'+
            outimgname, img_comb)

    return H_ransac


# help function of constructPanoramic
# get four corner coords after an image is restored through a homography
def getMaxWorldCoord(img, H):

    maxcoord = np.array([[0, 0], [0, img.shape[0]], [img.shape[1], img.shape[0]], [img.
        shape[1], 0]])
    maxcoord = np.append(maxcoord, np.ones((4,1)), axis=1)
    maxcoord_world = H.dot(maxcoord.T)
    maxcoord_world = (maxcoord_world/maxcoord_world[-1]).astype(int)

    return maxcoord_world

# help function of constructPanoramic
# The function creates an blank image with the same size as the input image
def getBlankImage(width, height):

    blankimg = np.zeros((min(height, 50000), min(width, 50000), 3), dtype=np.uint8)

    return blankimg

# Given 5 pictures and homographies between the center picture
# and other pictures, construct and return a panoramic picture.
def constructPanoramic(img1, img2, img3, img4, img5, H31, H32, H34, H35):

    # use the origin images to calculate the size of the canvas
    maxcoord_3 = np.array([[0, 0], [0, img3.shape[0]], [img3.shape[1], img3.shape[0]], [
        img3.shape[1], 0]])
    maxcoord_3 = np.append(maxcoord_3, np.ones((4,1)), axis=1).T.astype(int)

    maxcoord_world1 = getMaxWorldCoord(img1, np.linalg.inv(H31))
    maxcoord_world2 = getMaxWorldCoord(img2, np.linalg.inv(H32))
    maxcoord_world4 = getMaxWorldCoord(img4, np.linalg.inv(H34))
    maxcoord_world5 = getMaxWorldCoord(img5, np.linalg.inv(H35))
```

```
maxcoord_world = np.concatenate((maxcoord_world1, maxcoord_world2, maxcoord_world4,
    maxcoord_world5, maxcoord_3), axis=1)

# offset in the recovered image coordinates
offset_x = min(maxcoord_world[0])
offset_y = min(maxcoord_world[1])

# calculated size of the recovered image
new_width = max(maxcoord_world[0]) - min(maxcoord_world[0])
new_height = max(maxcoord_world[1]) - min(maxcoord_world[1])

new_img = getBlankImage(new_width, new_height)

# pixel replacement
for i in range(new_img.shape[1]):
    for j in range(new_img.shape[0]):

        x = i + offset_x
        y = j + offset_y

        proj_coord = H31.dot([x, y, 1])
        x_proj = round(proj_coord[0]/proj_coord[2])
        y_proj = round(proj_coord[1]/proj_coord[2])
        if 0 <= x_proj and x_proj < img1.shape[1] and 0 <= y_proj and y_proj < img1.
            shape[0]:
            new_img[j, i] = img1[y_proj, x_proj]

        proj_coord = H35.dot([x, y, 1])
        x_proj = round(proj_coord[0]/proj_coord[2])
        y_proj = round(proj_coord[1]/proj_coord[2])
        if 0 <= x_proj and x_proj < img5.shape[1] and 0 <= y_proj and y_proj < img5.
            shape[0]:
            new_img[j, i] = img5[y_proj, x_proj]

        proj_coord = H32.dot([x, y, 1])
        x_proj = round(proj_coord[0]/proj_coord[2])
        y_proj = round(proj_coord[1]/proj_coord[2])
        if 0 <= x_proj and x_proj < img2.shape[1] and 0 <= y_proj and y_proj < img2.
            shape[0]:
            new_img[j, i] = img2[y_proj, x_proj]

        proj_coord = H34.dot([x, y, 1])
        x_proj = round(proj_coord[0]/proj_coord[2])
        y_proj = round(proj_coord[1]/proj_coord[2])
        if 0 <= x_proj and x_proj < img4.shape[1] and 0 <= y_proj and y_proj < img4.
            shape[0]:
            new_img[j, i] = img4[y_proj, x_proj]

        x_proj = x
        y_proj = y
        if 0 <= x_proj and x_proj < img3.shape[1] and 0 <= y_proj and y_proj < img3.
            shape[0]:
            new_img[j, i] = img3[y_proj, x_proj]
```

```
    return new_img


# construct a panoramic using 5 images
def panoramic(img1, img2, img3, img4, img5, LM = False):

    H_32 = homography_ransac(img3, img2, LM = LM)
    H_34 = homography_ransac(img3, img4, LM = LM)

    H_21 = homography_ransac(img2, img1, LM = LM)
    H_31 = H_32.dot(H_21)
    H_31 = H_31/H_31[-1, -1]

    H_45 = homography_ransac(img4, img5, LM = LM)
    H_35 = H_34.dot(H_45)
    H_35 = H_35/H_35[-1, -1]

    pano = constructPanoramic(img1, img2, img3, img4, img5, H_31, H_32, H_34, H_35)

    return pano



if __name__ == '__main__' :

    result_path = 'C:/Users/jzx/OneDrive␣-␣purdue.edu/ECE661/hw5/result␣images/'

# # images for task 1
# img_1 = cv2.imread('HW5-Images/0.jpg')
# img_2 = cv2.imread('HW5-Images/1.jpg')
# img_3 = cv2.imread('HW5-Images/2.jpg')
# img_4 = cv2.imread('HW5-Images/3.jpg')
# img_5 = cv2.imread('HW5-Images/4.jpg')


# ### Task1 ###
# pimg = panoramic(img_1, img_2, img_3, img_4, img_5)
# cv2.imwrite(result_path + '1_pano' + '.jpg', pimg)

# corr_12 = drawCorrespondences(img_1, img_2)
# cv2.imwrite(result_path + '1_corr_12' + '.jpg', corr_12)
# corr_23 = drawCorrespondences(img_2, img_3)
# cv2.imwrite(result_path + '1_corr_23' + '.jpg', corr_23)
# corr_34 = drawCorrespondences(img_3, img_4)
# cv2.imwrite(result_path + '1_corr_34' + '.jpg', corr_34)
# corr_45 = drawCorrespondences(img_4, img_5)
# cv2.imwrite(result_path + '1_corr_45' + '.jpg', corr_45)

# _ = homography_ransac(img_1, img_2, drawliers = True, outimgname = '1_liers_12.jpg')
# _ = homography_ransac(img_2, img_3, drawliers = True, outimgname = '1_liers_23.jpg')
# _ = homography_ransac(img_3, img_4, drawliers = True, outimgname = '1_liers_34.jpg')
# _ = homography_ransac(img_4, img_5, drawliers = True, outimgname = '1_liers_45.jpg')
```

```
    ### Task2 ###
    img_1 = cv2.imread('HW5-Images/21.jpg')
    img_2 = cv2.imread('HW5-Images/22.jpg')
    img_3 = cv2.imread('HW5-Images/23.jpg')
    img_4 = cv2.imread('HW5-Images/24.jpg')
    img_5 = cv2.imread('HW5-Images/25.jpg')

    pimg = panoramic(img_1, img_2, img_3, img_4, img_5)
    cv2.imwrite(result_path + '2_pano' + '.jpg', pimg)

# corr_12 = drawCorrespondences(img_1, img_2)
# cv2.imwrite(result_path + '2_corr_12' + '.jpg', corr_12)
# corr_23 = drawCorrespondences(img_2, img_3)
# cv2.imwrite(result_path + '2_corr_23' + '.jpg', corr_23)
# corr_34 = drawCorrespondences(img_3, img_4)
# cv2.imwrite(result_path + '2_corr_34' + '.jpg', corr_34)
# corr_45 = drawCorrespondences(img_4, img_5)
# cv2.imwrite(result_path + '2_corr_45' + '.jpg', corr_45)

# _ = homography_ransac(img_1, img_2, drawliers = True, outimgname = '2_liers_12.jpg')
# _ = homography_ransac(img_2, img_3, drawliers = True, outimgname = '2_liers_23.jpg')
# _ = homography_ransac(img_3, img_4, drawliers = True, outimgname = '2_liers_34.jpg')
# _ = homography_ransac(img_4, img_5, drawliers = True, outimgname = '2_liers_45.jpg')
```