ECE 661 Computer Vision

Homework 5

Yi-Hong Liao | liao163@purdue.edu | PUID 0031850073

1. Logic

Establish the correspondence between images using SIFT and reject the outliers with RANSAC algorithm. Estimate homography using linear least square and refine it using non-linear least square (LM). Use the homographies to stitch the images and form a panoramic view.

2. Step

- Linear Least Square Estimation
  Homography H can be express as

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

For each selected points pair (x, y) and (x', y') we have two equations

$$xh_{11} + yh_{12} + h_{13} - xx'h_{31} - yx'h_{32} - x'h_{33} = 0,$$

$$xh_{21} + yh_{22} + h_{23} - xy'h_{31} - yy'h_{32} - y'h_{33} = 0.$$

Because only the ratio of H maters, we set $h_{33} = 1$. Therefore, if we selected four points on the distorted image and its four corresponding points on the undistorted image based on the physical dimension, we can write down the equations as matrix

$$AX = B,$$

where

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_{21} & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix},$$

$$X = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}, B = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}.$$

The best estimate of X can be solved using least square

$$\hat{X} = (A^T A)^{-1} A^T B.$$

Then the homography H can be estimated.

- RANSAC Algorithm
  - Set parameters
    - $\sigma = 2$
    - $\delta = 3\sigma = 6$
    - P = 0.99
    - $\epsilon = 0.25$
    - n = 6
    - N = $\frac{ln(1-P)}{ln[1-(1-\epsilon)^n]}$
    - $M = (1-\epsilon) \cdot n_{total}$
  - From all the correspondences found by SIFT, randomly chose n ($n > 4$) correspondences to estimate the homography using linear least square.
  - Applied the homography and identified the number of inliers (the correspondences that have Euclidean distance between estimate point and real point < δ).
  - Repeat the above estimation N trials and select the homography that results in the most inliers.
  - Estimate the homography using all the inliers in the best trial.
- Non-Linear Least Square Estimation
  For the non-linear least square estimation, I wrote my own Levenberg-Marquardt (LM) algorithm. The goal of the non-linear least square is to minimize

$$C(p) = \|X - f(p)\|^2$$

Where p is the vector consists of the homography parameters we want to refine. We want to find δ_p in each iteration so that $p + \delta_p$ can lead to the local minimum of C. What LM does is combine gradient descent (GD) and Gauss-Newton (GN) method. LM at each iteration k consists of first setting a value for damping coefficient μ_k and then setting

$$\delta_p = \left(J_f^T J_f + \mu I\right)^{-1} J_f^T \epsilon(p_k)$$
$$p_{k+1} = p_k + \delta_p$$

I calculate the Jacobian matrix J_f by differentiate f(p) with respect to p for each measurement. Then determine

$$\mu_{k+1} = \mu_k \cdot max\{1/3, 1 - (2\rho - 1)^3\}.$$

When increasing μ, LM behaves more like GD. If $\mu = 0$, LM behaves as GN.

- Image Stitching
  I first use the homography to transform the four corner points of the domain image into range image plane. Then, I filled out the pixels inside the polygon form by the transformed four corner points in the range image by using the inverse of the homography. In this work, the middle image is set as the common reference frame for all 5 images. All images are transformed into middle image coordinate.

3. Result

• Theory question

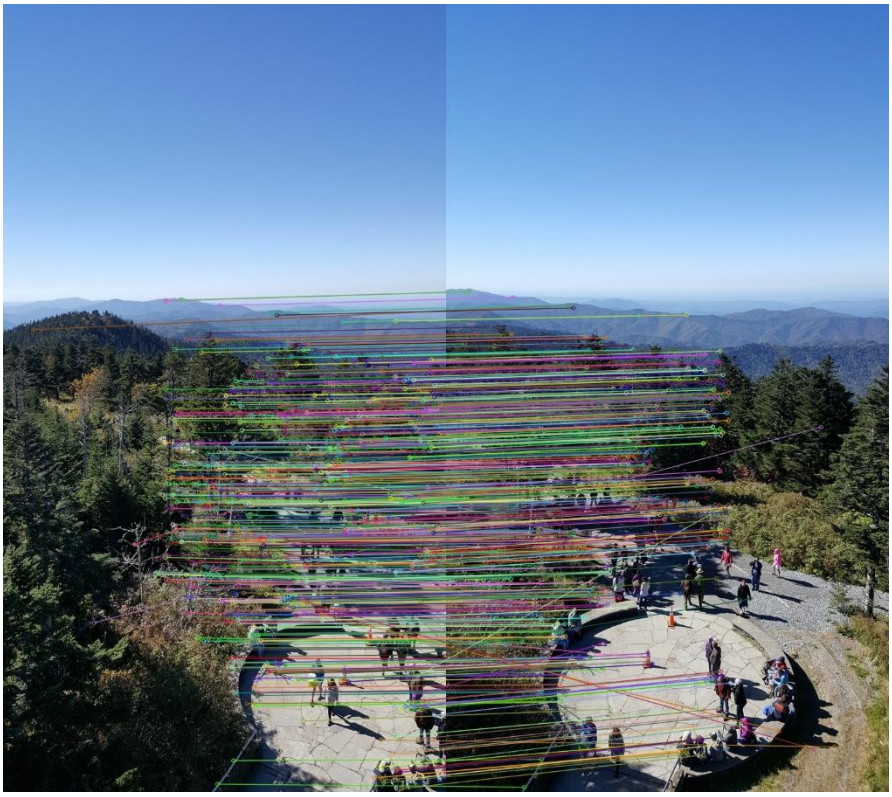
- In RANSAC algorithm, we estimate the homography many times using the minimum correspondences needed. In every trial, we find the number of the inliers. The inliers are the estimated points that has Euclidean distance to the real corresponding points smaller than a threshold. After the trials, the homography with the most inliers is selected and its inliers are used to estimate the homography one last time. Then the final inliers and outliers are determined by the Euclidean distance again using this final homography.
- Gradient Descent (GD) works best when the estimation is far from the local minimum, but the convergence rate drops significantly when it's close to the local minimum. On the other hand, Gauss-Newton (GN) can provide a one step solution when the estimation is close to local minimum, but it's more unstable. The Levenberg-Marquardt (LM) algorithm combines the best of GD and GN by introducing a damping factor μ. By adjusting μ, the algorithm shifts between GD and GN. When the estimation starts, μ is set to be large and the LM behaves like GD. When the estimation is close to local minimum, μ decrease the GN dominates the convergence.

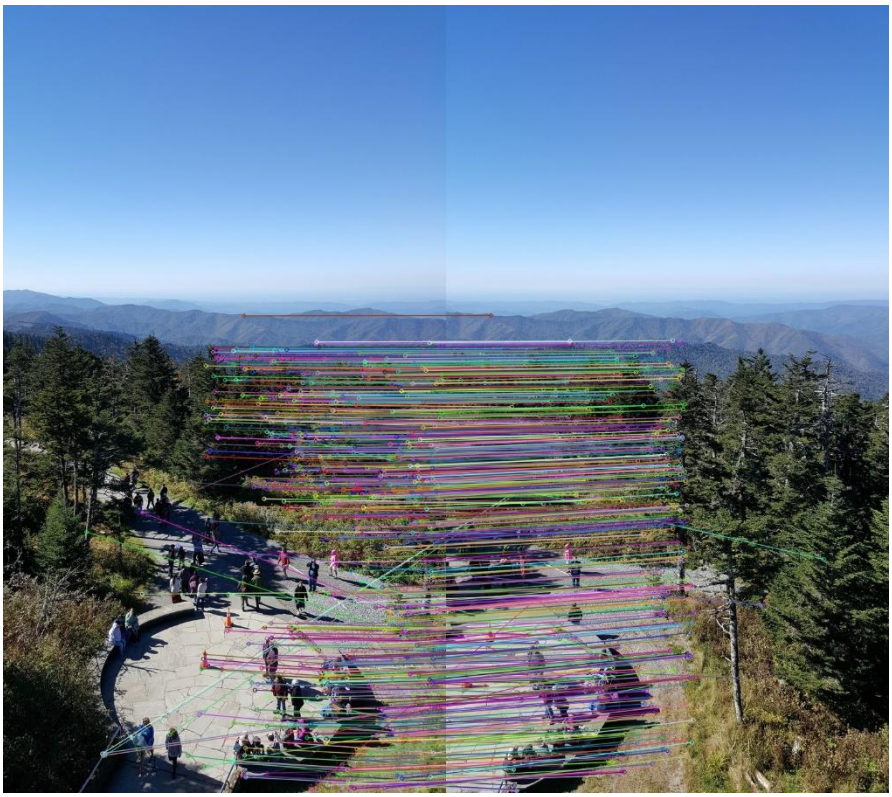
- **Input images**

Figure 1: Input image set 1

Figure 2: Input image set 2

- **Correspondence between images**





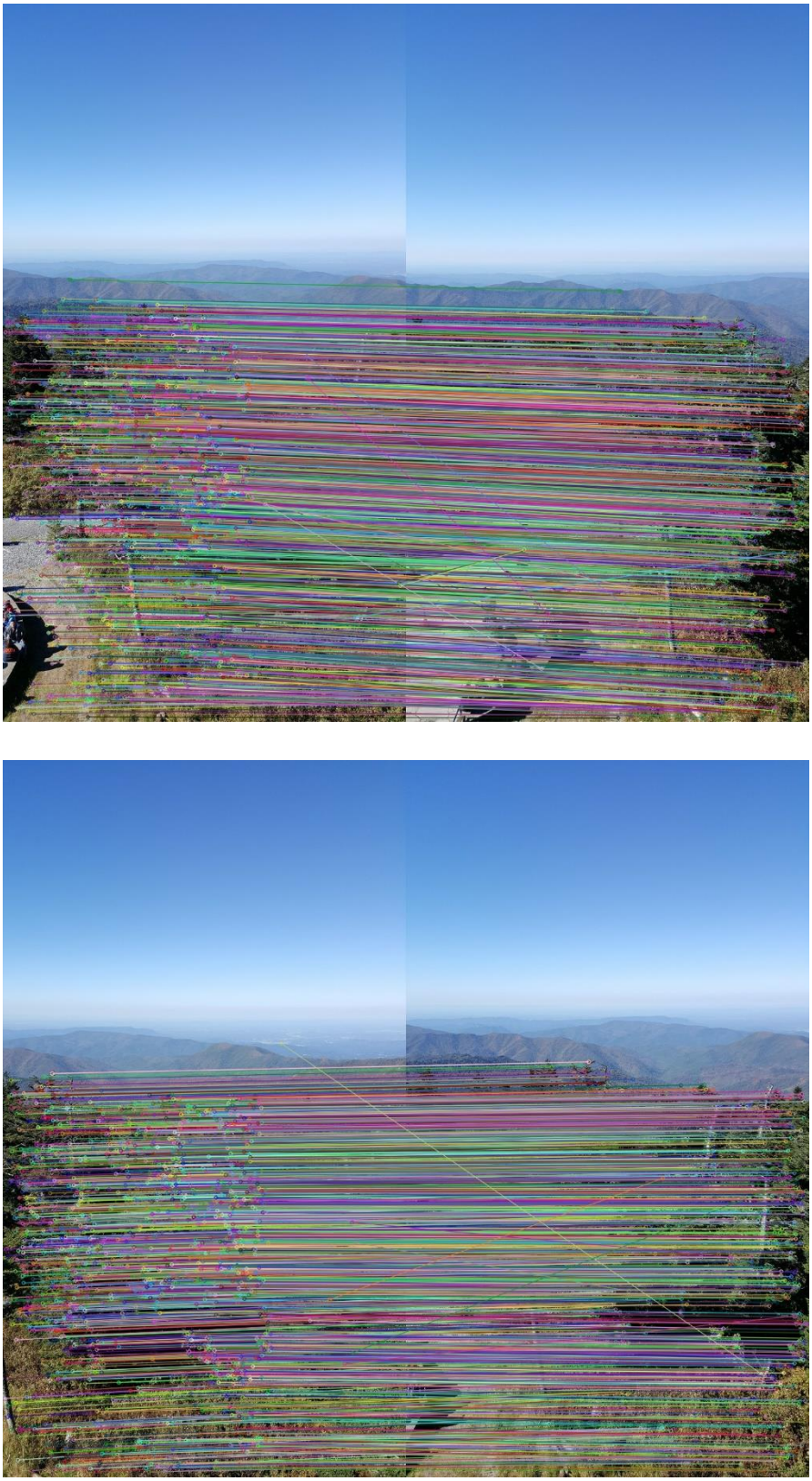
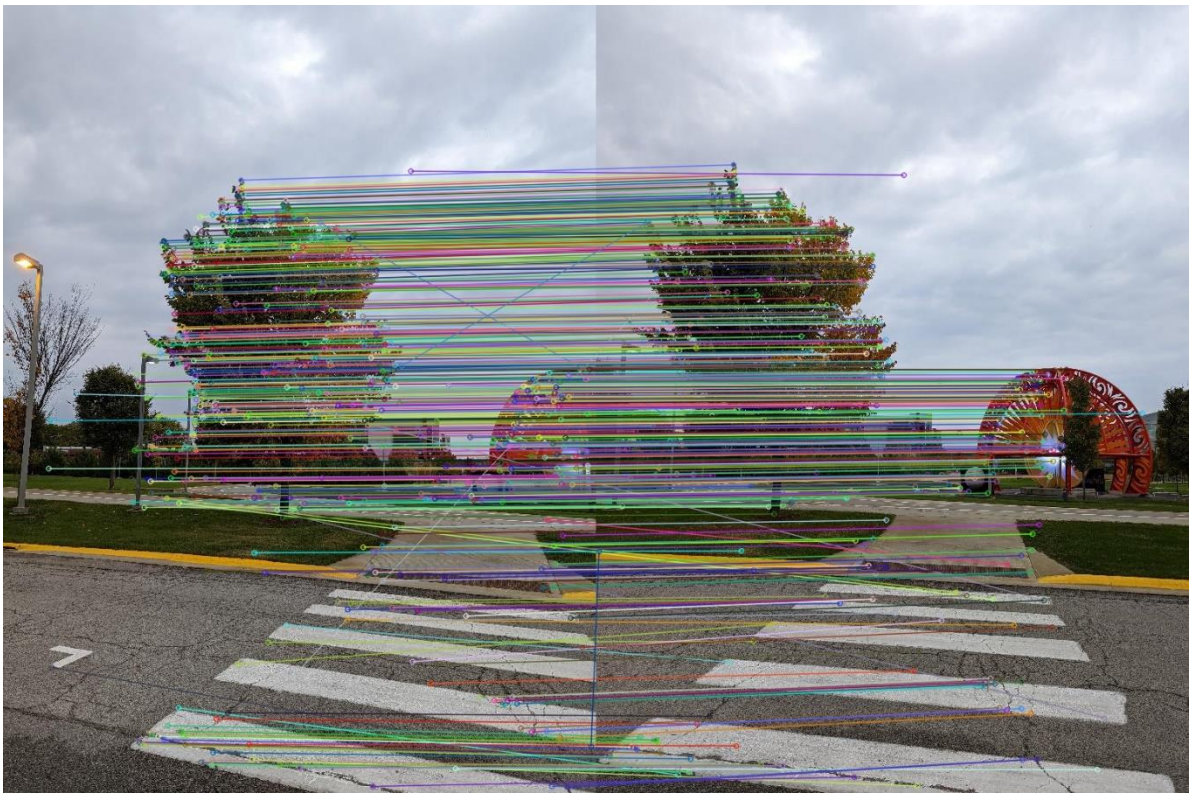
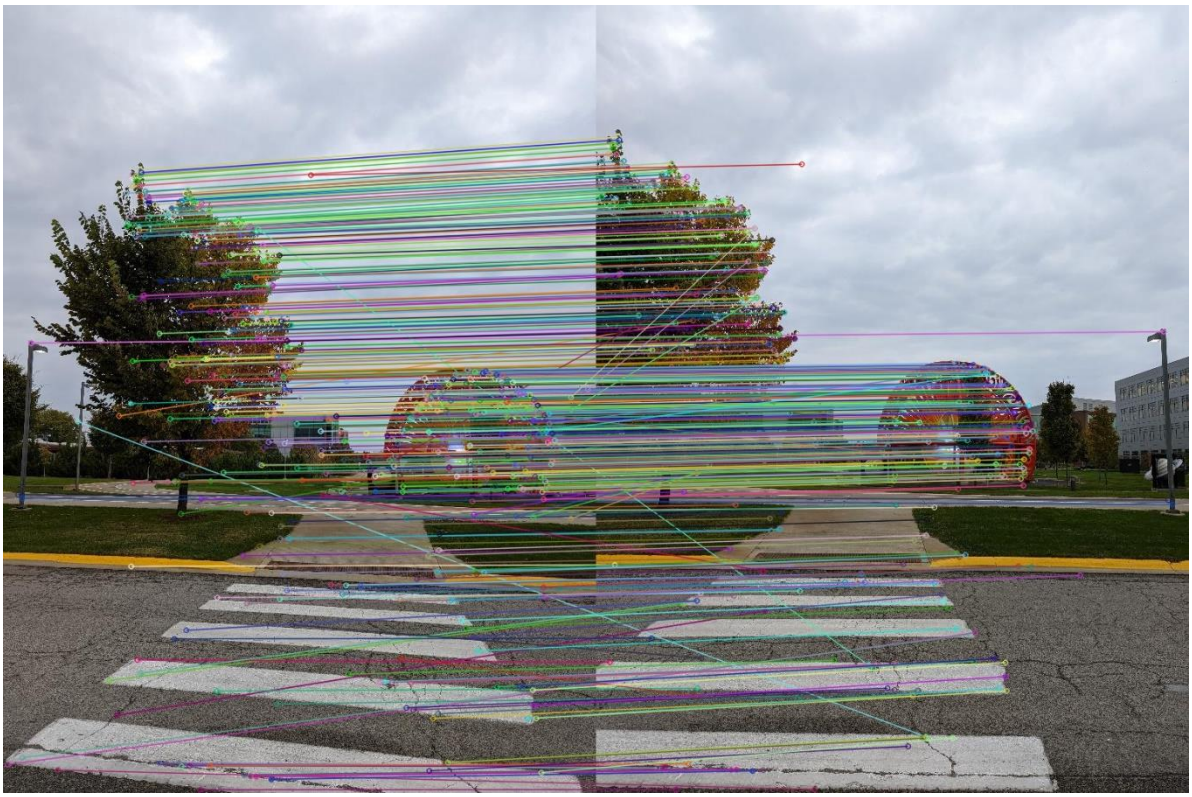


Figure 3: Image correspondences in image set 1





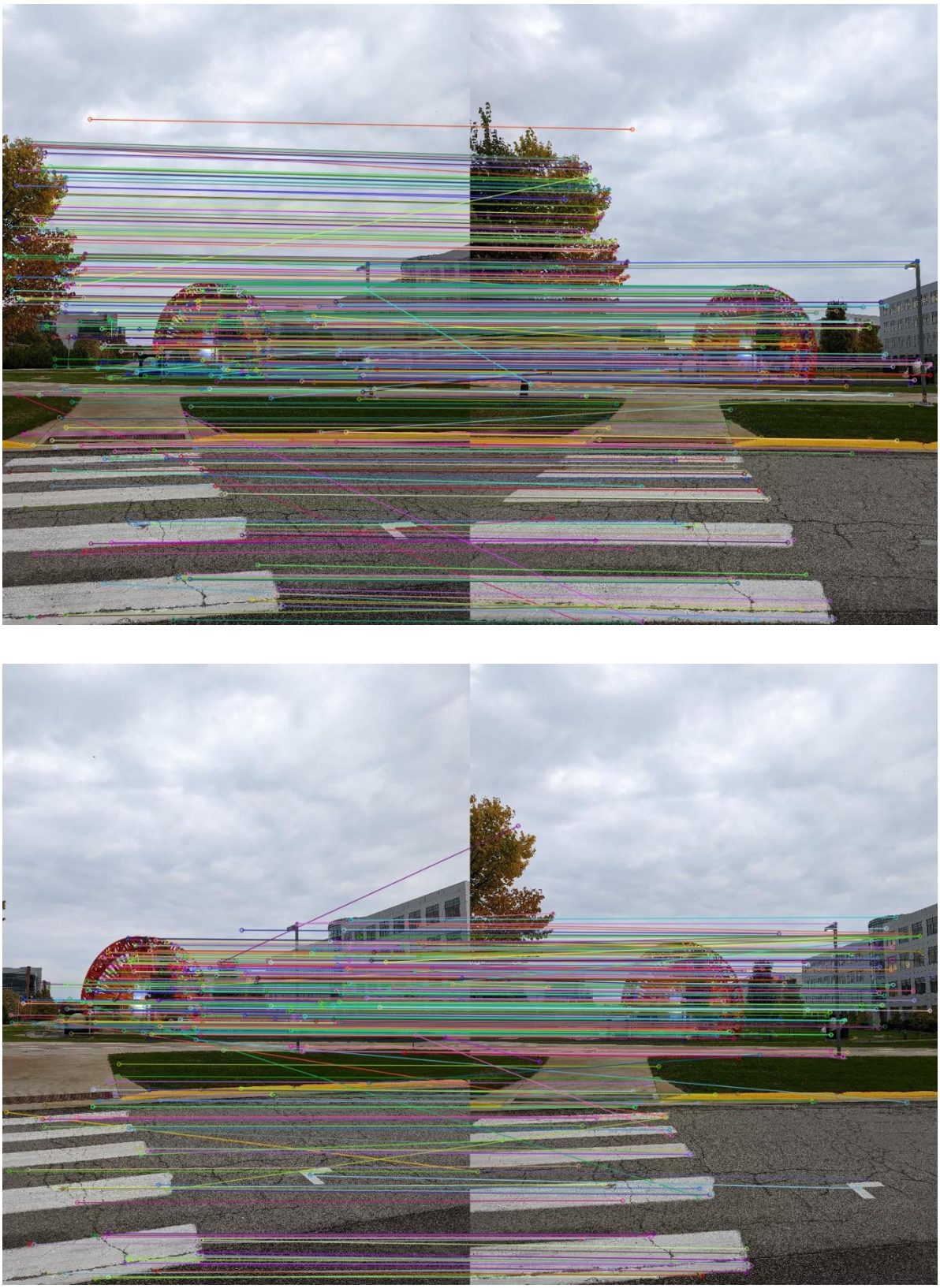
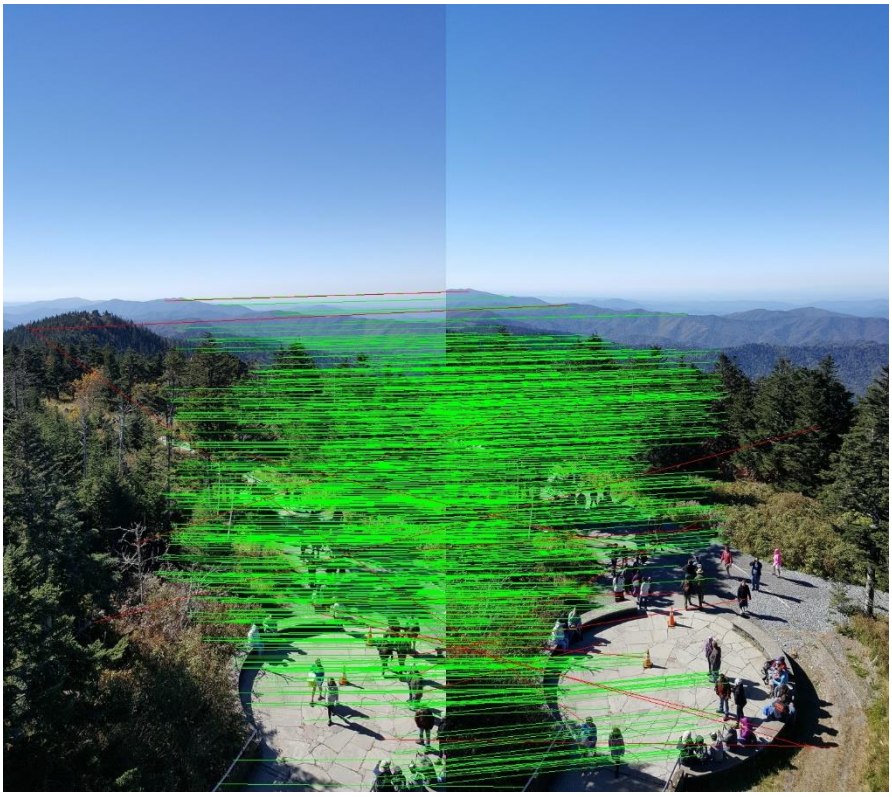
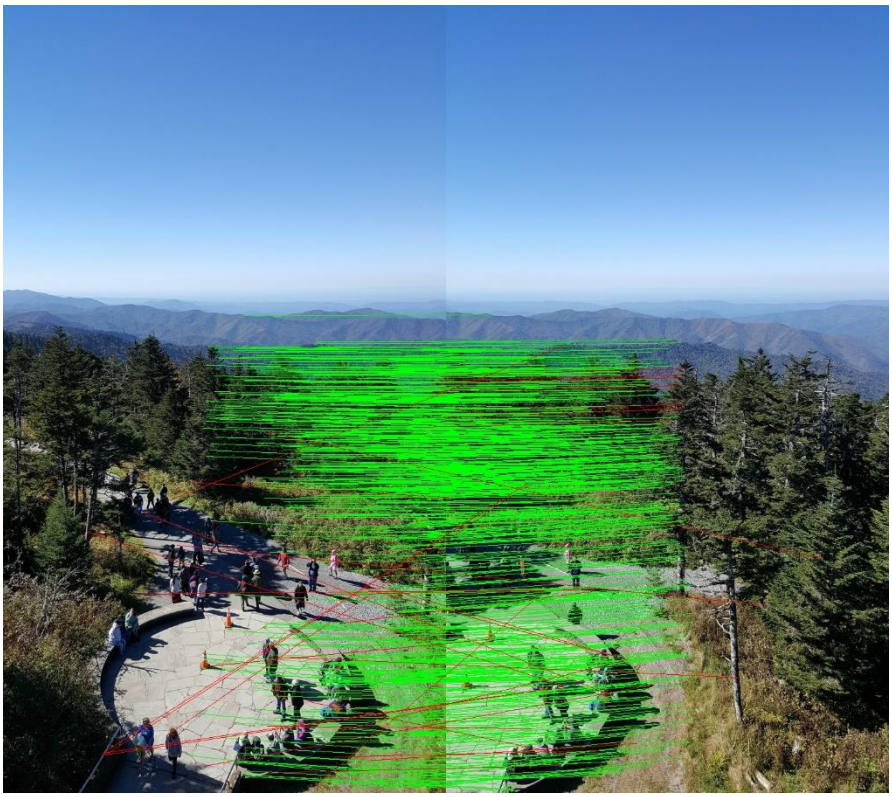


Figure 4: Image correspondences in image set 2

- **Set of inliers (green lines) and outliers (red lines)**





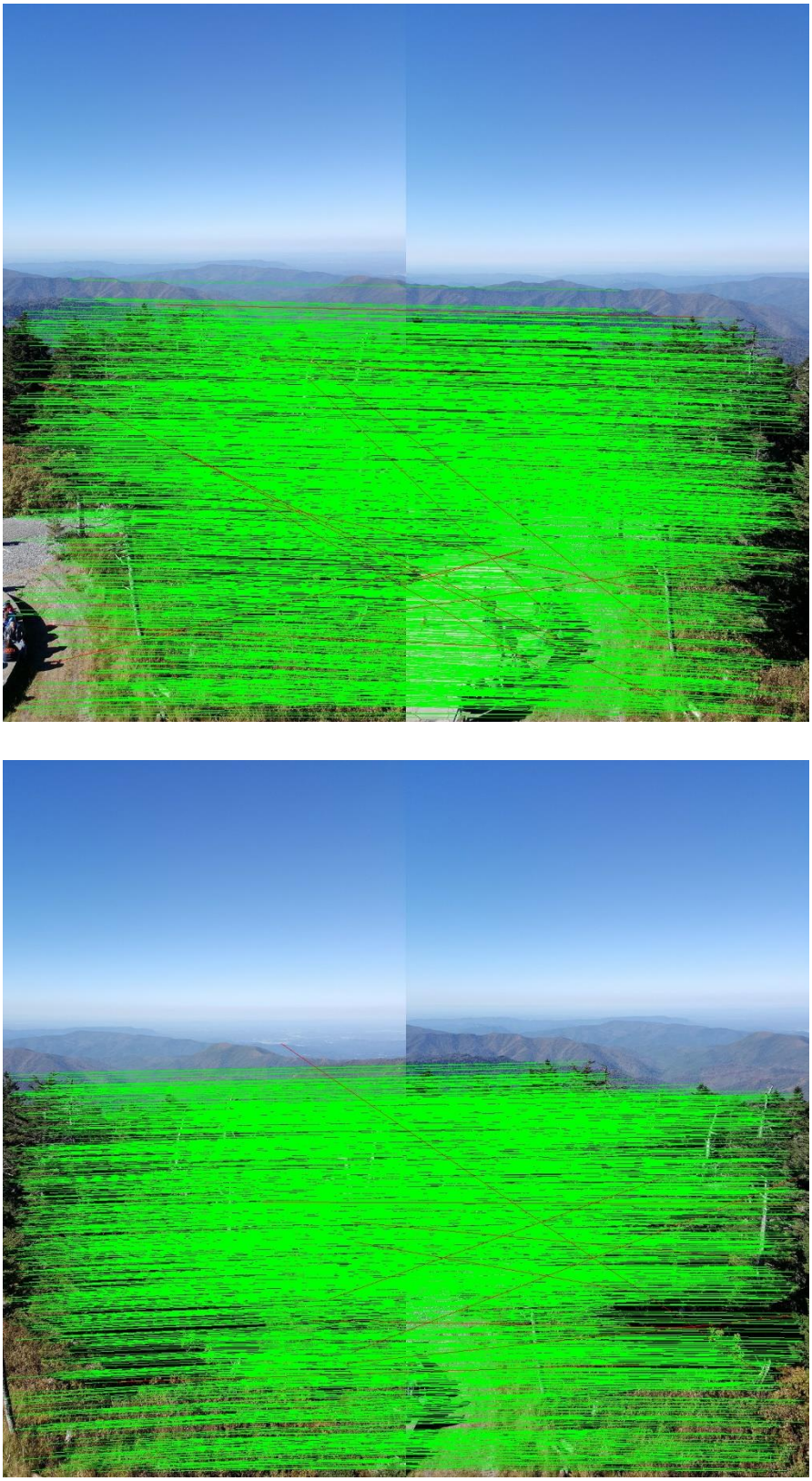
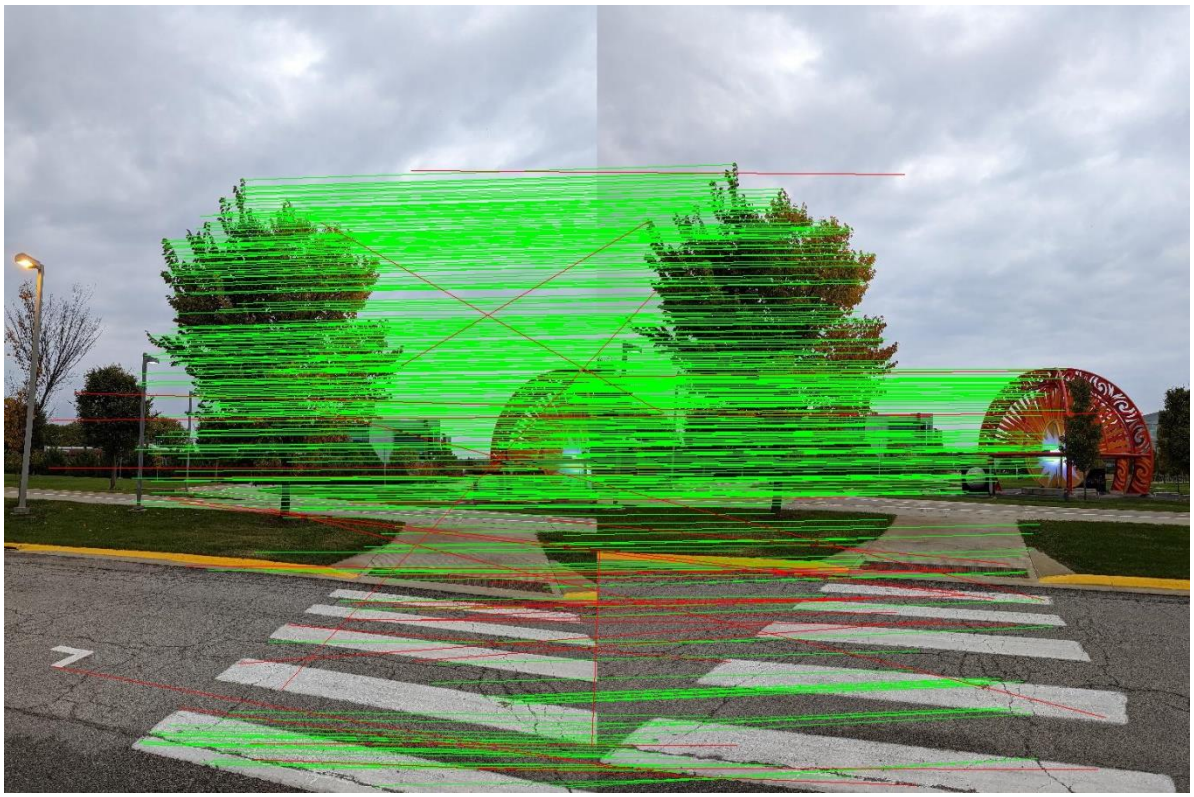
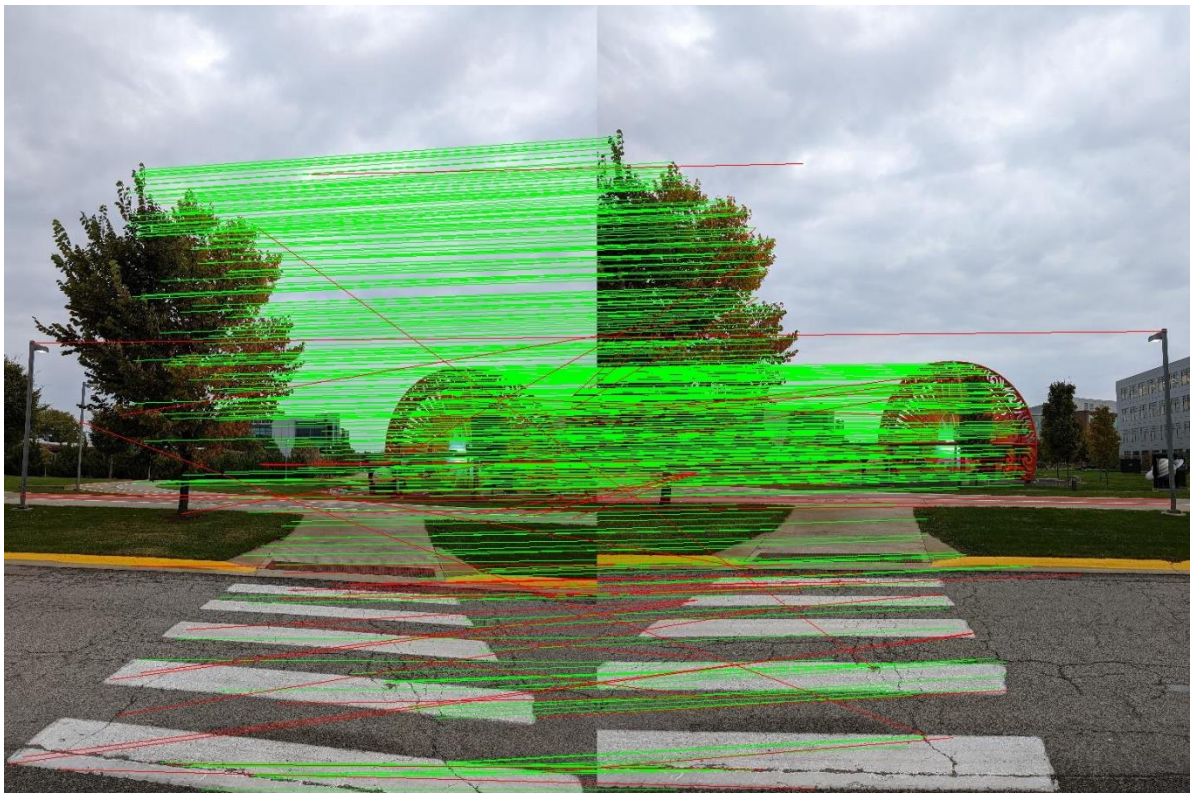


Figure 5: Correspondence outliers and inliers of image set 1





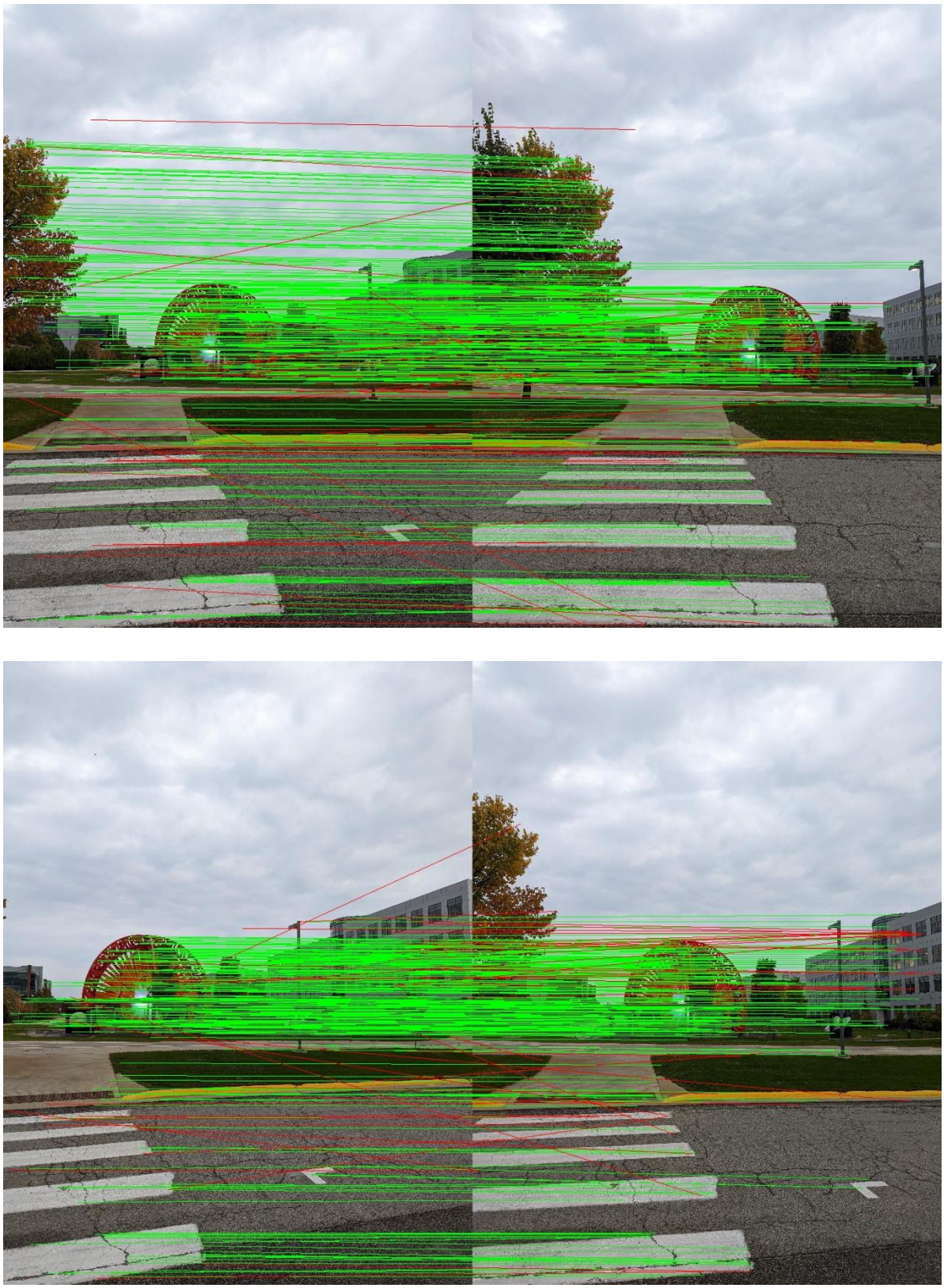


Figure 6: Correspondence outliers and inliers of image set 2

- **Panorama**

Figure 7: Panorama of image set 1

Figure 8: Panorama of image set 2

- **Geometric error of full set of inliers**

Image Set 1				
	Image 0/1	Image 1/2	Image 3/2	Image 4/3
RANSAC	1186.86	2157.83	2886.61	3275.82
LM	1179.38	2135.64	2875.16	3258.82

Image Set 2				
	Image 0/1	Image 1/2	Image 3/2	Image 4/3
RANSAC	1446.28	1159.88	1046.13	728.25
LM	1444.83	1157.67	1043.60	728.14

4. Source Code

```
#!/usr/bin/env python
# coding: utf-8

# In[248]:


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


# In[249]:


def calculate_homography(X, Xp):
    n = len(X)
    A = np.zeros((n*2, 8))
    B = np.zeros((n*2, 1))
    for i in range(n):
        A[2*i][0] = X[i][0]
        A[2*i][1] = X[i][1]
        A[2*i][2] = 1
        A[2*i][6] = -X[i][0]*Xp[i][0]
        A[2*i][7] = -X[i][1]*Xp[i][0]
        A[2*i+1][3] = X[i][0]
        A[2*i+1][4] = X[i][1]
        A[2*i+1][5] = 1
        A[2*i+1][6] = -X[i][0]*Xp[i][1]
        A[2*i+1][7] = -X[i][1]*Xp[i][1]
        B[2*i] = Xp[i][0]
        B[2*i+1] = Xp[i][1]
    H = np.linalg.inv(A.transpose().dot(A)).dot(A.transpose()).dot(B)
    H = np.append(H, 1)
    H = H.reshape(3, 3)
    return H

def find_SIFT_correspondence(img1, img2):
    # This code is modified from OpenCV document:
    # https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Initiate SIFT detector
    sift = cv2.SIFT_create()

    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1_gray, None)
    kp2, des2 = sift.detectAndCompute(img2_gray, None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()
```

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

    # Apply ratio test
    kp1_out = []
    kp2_out = []
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            kp1_out.append(kp1[m.queryIdx].pt)
            kp2_out.append(kp2[m.trainIdx].pt)
            good.append([m])

    img3 =
cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv2.DrawMatchesFlags_
NOT_DRAW_SINGLE_POINTS)
    return kp1_out, kp2_out, img3


def find_inliers(H, kp1, kp2, delta):
    n = len(kp1)
    kp1_h = np.insert(kp1, 2, 1, axis=1)
    kp2_h = np.insert(kp2, 2, 1, axis=1)
    kp1p_h = H.dot(np.transpose(kp1_h))
    kp1p_h = np.transpose(kp1p_h/kp1p_h[2, :])
    dist2 = np.sum((kp2_h-kp1p_h)**2, axis=1)
    inliers_idx = np.where(dist2 <= delta**2)
    outliers_idx = np.where(dist2 > delta**2)

    return inliers_idx[0], inliers_idx[0].size, outliers_idx[0],
outliers_idx[0].size

def draw_inliers_outliers(img1, img2, kp1_inliers, kp2_inliers,
kp1_outliers, kp2_outliers):
    kp1_inliers = kp1_inliers.astype(int)
    kp2_inliers = kp2_inliers.astype(int)
    kp1_outliers = kp1_outliers.astype(int)
    kp2_outliers = kp2_outliers.astype(int)

    w = img1.shape[1]
    img = np.concatenate((img1, img2), axis=1)
    color_inliers = (0, 255, 0)
    color_outliers = (0, 0, 255)
    for i in range(0, len(kp1_inliers)):
        #convert data types int64 to int
        cv2.line(img, tuple(kp1_inliers[i]), (kp2_inliers[i][0]+w,
kp2_inliers[i][1]), color_inliers, 1)

    for i in range(0, len(kp1_outliers)):
        #convert data types int64 to int
        cv2.line(img, tuple(kp1_outliers[i]), (kp2_outliers[i][0]+w,
kp2_outliers[i][1]), color_outliers, 1)
    return img

def RANSAC(kp1, kp2, sigma, p, epsil, n):
    delta = 3*sigma
    nTot = len(kp1)
    N = math.ceil(math.log(1 - p) / math.log(1 - ((1 - epsil)**n)))
```

```
    M = math.ceil ((1 - epsil)*nTot)
    print('delta: ', delta)
    print('nTot: ', nTot)
    print('N: ', N)
    print('M: ', M)

    max_num_inliers = 0
    for i in range(0, N):
        n_idxs = np.random.choice(range(0, nTot), n)
        X = []
        Xp = []
        for i in n_idxs:
            X.append(kp1[i])
            Xp.append(kp2[i])
        H = calculate_homography(X, Xp)
        inliers_idx, num_inliers, outliers_idx, num_outliers =
find_inliers(H, kp1, kp2, delta)
        if num_inliers > max_num_inliers and num_inliers >= M:
            max_num_inliers = num_inliers
            best_inliers_idx = inliers_idx
            best_outliers_idx = outliers_idx
    kp1_inliers = np.array(kp1)[best_inliers_idx]
    kp2_inliers = np.array(kp2)[best_inliers_idx]
    kp1_outliers = np.array(kp1)[best_outliers_idx]
    kp2_outliers = np.array(kp2)[best_outliers_idx]

    H_new = calculate_homography(kp1_inliers, kp2_inliers)
    return H_new, kp1_inliers, kp2_inliers, kp1_outliers, kp2_outliers

def calculate_error(H, kp1, kp2):
    kp1_h = np.insert(kp1, 2, 1, axis=1)
    kp1p_h = H.dot(np.transpose(kp1_h))
    kp1p_h = np.transpose(kp1p_h/kp1p_h[2, :])

    return kp2.flatten() - kp1p_h[:, 0:2].flatten()

def levenberg_marquardt(H0, kp1, kp2):
    m = len(kp1)
    kp1 = np.array(kp1)
    kp2 = np.array(kp2)
    error = calculate_error(H0, kp1, kp2)

    H = H0
    J = np.zeros((2*m, 9))
    cost_change = 1
    prev_cost = 0
    new_cost = 1
    tau = 0.5
    iteration = 0
    while cost_change > 0.00005:
        for i in range(m):
            x = kp1[i][0]
            y = kp1[i][1]
            num1 = H[0][0]*x + H[0][1]*y + H[0][2]
            num2 = H[1][0]*x + H[1][1]*y + H[1][2]
            den = H[2][0]*x + H[2][1]*y + H[2][2]
            J[i*2][0] = x/den
```

```
            J[i*2][1] = y/den
            J[i*2][2] = 1/den
            J[i*2][6] = -x*num1*(den**(-2))
            J[i*2][7] = -y*num1*(den**(-2))
            J[i*2][8] = -num1*(den**(-2))

            J[i*2+1][3] = x/den
            J[i*2+1][4] = y/den
            J[i*2+1][5] = 1/den
            J[i*2+1][6] = -x*num2*(den**(-2))
            J[i*2+1][7] = -y*num2*(den**(-2))
            J[i*2+1][8] = -num2*(den**(-2))
        error = calculate_error(H, kp1, kp2)
        new_cost = error.dot(error)

        if iteration == 0:
            u = tau*max(np.diag((J.T).dot(J)))
        else:
            cost_change = prev_cost - new_cost
            rho =
cost_change/((delta_p.T).dot(J.T).dot(error)+(delta_p.T).dot(u*np.identity
(9)).dot(delta_p))
            u = u*max(1/3, 1-(2*rho-1)**3)

        delta_p =
np.linalg.inv((J.T).dot(J)+u*np.identity(9)).dot(J.T).dot(error)
        delta_p_m = delta_p.reshape(3, 3)
        prev_cost = new_cost
        H = H + delta_p_m
        iteration = iteration + 1

    return H

def cost_function(h, kp1, kp2):
    H=h.reshape((3,3))
    kp1_h = np.insert(kp1, 2, 1, axis=1)
    kp2_h = np.insert(kp2, 2, 1, axis=1)
    kp1p_h = H.dot(np.transpose(kp1_h))
    kp1p_h = np.transpose(kp1p_h/kp1p_h[2, :])
    error = kp2_h-kp1p_h
    return error.flatten()

def optimize_homography(img1, img2):
    kp1, kp2, img_SIFT = find_SIFT_correspondence(img1, img2)

    H_RANSAC, kp1_inliers, kp2_inliers, kp1_outliers, kp2_outliers =
RANSAC(kp1, kp2, 2, 0.99, 0.25, 6)

    error_RANSAC = calculate_error(H_RANSAC, kp1_inliers, kp2_inliers);
    print("Geometric error of RANSAC: ", error_RANSAC.dot(error_RANSAC))

    img_in_outliers = draw_inliers_outliers(img1, img2, kp1_inliers,
kp2_inliers, kp1_outliers, kp2_outliers)

    H_nonlinear = levenberg_marquardt(H_RANSAC, kp1_inliers, kp2_inliers)

    error_lm = calculate_error(H_nonlinear, kp1_inliers, kp2_inliers);
```

```
    print("Geometric error of LM: ", error_lm.dot(error_lm))

#     h_RANSAC = H_RANSAC.flatten()
#     H_nonlinear = optimize.least_squares(cost_function, h_RANSAC, args =
(kp1_inliers, kp2_inliers), method = 'lm').x
#     H_nonlinear = H_nonlinear.reshape(3, 3)
#     print(H_RANSAC)
#     print(H_nonlinear/H_nonlinear[2, 2])

    return H_nonlinear, img_SIFT, img_in_outliers

def combine_image(imgs, Hs, ds):
    num_imgs = len(imgs)
    center_img_idx = int((num_imgs-1)/2)
    h = imgs[0].shape[0]
    w = imgs[0].shape[1]

    img_ROI = np.array([[0, 0, 1], [0, h-1, 1], [w-1, h-1, 1], [w-1, 0,
1]])

    Hs2 = []
    imgs2 = []

    for k in range(0, center_img_idx):
        H = np.identity(3)
        imgs2.append(imgs[center_img_idx+k+1])
        for l in range(0, k+1):
            H = H.dot(Hs[center_img_idx+l])
        Hs2.append(H)

    for k in range(0, center_img_idx):
        H = np.identity(3)
        imgs2.append(imgs[center_img_idx-k-1])
        for l in range(0, k+1):
            H = Hs[center_img_idx-l-1].dot(H)
        Hs2.append(H)

    # expand image
    img_old = imgs[center_img_idx]
    total_ROI = img_ROI
    x_increment = 0
    y_increment = 0
    for k in range(0, num_imgs-1):
        H = Hs2[k]

        pointHp = H.dot(np.transpose(img_ROI))
        pointHp = np.transpose(pointHp/pointHp[2, :])
        ROI = np.around(pointHp).astype(int)

        # clip ROI
        ROI[0, 1] = max(0, ROI[0, 1])
        ROI[1, 1] = min(h-1, ROI[1, 1])
        ROI[2, 1] = min(h-1, ROI[2, 1])
        ROI[3, 1] = max(0, ROI[3, 1])

        total_ROI = np.append(total_ROI, ROI, axis=0)
```

```
        ROI_max = np.amax(ROI, axis=0)
        ROI_min = np.amin(ROI, axis=0)
        total_max = np.amax(total_ROI, axis=0)
        total_min = np.amin(total_ROI, axis=0)

        x_offset = max(0, -ROI_min[0])
        y_offset = max(0, -ROI_min[1])

        x_increment = max(0, -ROI_min[0]-x_increment)
        y_increment = max(0, -ROI_min[1]-y_increment)

        ROI_offset = ROI
        ROI_offset[:, 0] = ROI[:, 0]+x_offset
        ROI_offset[:, 1] = ROI[:, 1]+y_offset

        ho = total_max[1]-total_min[1]+1
        wo = total_max[0]-total_min[0]+1
        print("new image height and width", ho, wo)
        img_new = np.zeros([np.int32(np.floor(ho/ds))+1,
np.int32(np.floor(wo/ds))+1, 3], dtype='uint8')
        img_new[y_increment:img_old.shape[0]+y_increment,
x_increment:img_old.shape[1]+x_increment, :] = img_old
        ROI_mask = np.zeros([ho, wo])
        cv2.fillPoly(ROI_mask, pts = np.int32([ROI_offset[:, 0:2]]), color
= 255)

        for i in range(0, wo, ds):
            for j in range(0, ho, ds):
                if ROI_mask[j,i] > 0:
                    pointH = np.array([i-x_offset, j-y_offset, 1])
                    pointHp = np.linalg.inv(H).dot(pointH)
                    pointHp = pointHp/pointHp[2]
                    pointHp = np.around(pointHp).astype(int)
                    if pointHp[0] < w and pointHp[0] >= 0 and pointHp[1] <
h and pointHp[1] >= 0:
                        img_new[np.int32(np.floor(j/ds)),
np.int32(np.floor(i/ds))] = imgs2[k][pointHp[1], pointHp[0]]
        img_old = img_new
        print("merge!")

    return img_new


# In[250]:


if __name__ == '__main__':
    path = r'C:\Users\yhosc\Desktop\ECE661\HW5\HW5-Images\\'
    outputPath = r'C:\Users\yhosc\Desktop\ECE661\HW5\\'
    imgs = []
    for i in range(0, 5):
        img = cv2.imread(path+f'{i}.jpg')
        imgs.append(img)
    H01, img_SIFT_01, img_in_outliers_01 = optimize_homography(imgs[0],
imgs[1])
    H12, img_SIFT_12, img_in_outliers_12 = optimize_homography(imgs[1],
imgs[2])
```

```
    H32, img_SIFT_32, img_in_outliers_32 = optimize_homography(imgs[3],
imgs[2])
    H43, img_SIFT_43, img_in_outliers_43 = optimize_homography(imgs[4],
imgs[3])
    Hs = [H01, H12, H32, H43]

    panorama = combine_image(imgs, Hs, 1)
    cv2.imwrite(outputPath+'panorama1.jpg', panorama)
    cv2.imwrite(outputPath+'SIFT1_01.jpg', img_SIFT_01)
    cv2.imwrite(outputPath+'SIFT1_12.jpg', img_SIFT_12)
    cv2.imwrite(outputPath+'SIFT1_32.jpg', img_SIFT_32)
    cv2.imwrite(outputPath+'SIFT1_43.jpg', img_SIFT_43)
    cv2.imwrite(outputPath+'ransac1_01.jpg', img_in_outliers_01)
    cv2.imwrite(outputPath+'ransac1_12.jpg', img_in_outliers_12)
    cv2.imwrite(outputPath+'ransac1_32.jpg', img_in_outliers_32)
    cv2.imwrite(outputPath+'ransac1_43.jpg', img_in_outliers_43)

    imgs = []
    for i in range(0, 5):
        img = cv2.imread(path+'0'+f'{i}.jpg')
        imgs.append(img)
    H01, img_SIFT_01, img_in_outliers_01 = optimize_homography(imgs[0],
imgs[1])
    H12, img_SIFT_12, img_in_outliers_12 = optimize_homography(imgs[1],
imgs[2])
    H32, img_SIFT_32, img_in_outliers_32 = optimize_homography(imgs[3],
imgs[2])
    H43, img_SIFT_43, img_in_outliers_43 = optimize_homography(imgs[4],
imgs[3])
    Hs = [H01, H12, H32, H43]

    panorama = combine_image(imgs, Hs, 1)
    cv2.imwrite(outputPath+'panorama2.jpg', panorama)
    cv2.imwrite(outputPath+'SIFT2_01.jpg', img_SIFT_01)
    cv2.imwrite(outputPath+'SIFT2_12.jpg', img_SIFT_12)
    cv2.imwrite(outputPath+'SIFT2_32.jpg', img_SIFT_32)
    cv2.imwrite(outputPath+'SIFT2_43.jpg', img_SIFT_43)
    cv2.imwrite(outputPath+'ransac2_01.jpg', img_in_outliers_01)
    cv2.imwrite(outputPath+'ransac2_12.jpg', img_in_outliers_12)
    cv2.imwrite(outputPath+'ransac2_32.jpg', img_in_outliers_32)
    cv2.imwrite(outputPath+'ransac2_43.jpg', img_in_outliers_43)
```