# ECE 661: Homework 5

Xingguang Zhang

(Fall 2020)

## Theory Questions

**1.** Conceptually speaking, how do we differentiate between the inliers and the outliers when using RANSAC for solving the homography estimation problem using the interest points extracted from two different photos of the same scene?

- Given two images 1 and 2, we have interest point pairs $X$ and $Y$ on them, respectively. Assume the calculated homographic transform from $X$ to $Y$ is $H$ in a trail, we have $Y' = HX$, we can determine if a pair $X_i$ and $Y_i$ is an inlier or outlier by the distance between $Y_i'$ and $Y_i$, if $D(Y_i', Y_i) > \epsilon$, where $\epsilon$ is the distance threshold, we say the $i$-th pair is an outlier, otherwise, we claim they are an inlier.

**2.** As you will see in Lecture 12, the Gradient-Descent (GD) is a reliable method for minimizing a cost function, but it can be excruciatingly slow. At the other extreme, we have the much faster Gauss-Newton(GN) method but it can be numerically unstable. Explain in your own words how the Levenberg-Marquardt (LM) algorithm combines the best of GD and GN to give us a method that is reasonably fast and numerically stable at the same time.

- We focus on the increment vector $\delta$ of all these methods. For GD, it's $\delta = 2\mu J^T \epsilon$, for GN, it's $\delta = (J^T J)^{-1} J^T \epsilon$, for LM, it's $\delta = (J^T J + \mu I)^{-1} J^T \epsilon$. GD is stable, but when the parameter is near the optimal solution, the first order derivative can be small and the step size will be small, so it can be very slow. When the parameter is near the solution, GN can bring us directly to the solution, but in the early steps, when the parameter is far away from the solution, or when $J^T J$ is singular, the GN may suffer from the instability. But if we use LM method, and set a large $\mu$ in the initial steps, the $\mu I$ will be dominant in $J^T J + \mu I$, so $\delta \approx \mu^{-1} J^T \epsilon$, it will be act like GD and stably move to the solution. When $\mu$ damps over steps, the $J^J$ will be the dominant term and it will act just like GN, leading us to the finally solution quickly.

## Programming Tasks

### Algorithm description

#### 1. Linear Least-Squares Minimization

Linear Least-Squares Minimization is used to compute the homographic transform matrix $H$ when we have more than 4 pairs of interest points. Given a pair of interest points, the projective transformation $Y = HX$ projects the source coordinates in homogeneous representational space $X = (x, y, 1)$ into target coordinates $Y = (u, v, 1)$. $H$ is a non-singular matrix and can be represented as:

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

Where $a_{ij}$ are parameters that need to be solved, and we can set $a_{33} = 1$ because we want the projection by homographic. Thus, for each pair of $X$ and $Y$, we have:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

We can form 3 functions by this, they are:

$$\begin{aligned} a_{11}x + a_{12}y + a_{13} &= u \\ a_{21}x + a_{22}y + a_{23} &= v \\ a_{31}x + a_{32}y + 1 &= 1 \end{aligned}$$

Or,

$$\begin{aligned} a_{11}x + a_{12}y + a_{13} &= a_{31}xu + a_{32}yu + u \\ a_{21}x + a_{22}y + a_{23} &= a_{31}xv + a_{32}yv + v \end{aligned}$$

Then we have the equation that can be easily combined with equations from other pairs of points:

$$\begin{aligned} a_{11}x + a_{12}y + a_{13} - a_{31}xu - a_{32}yu &= u \\ a_{21}x + a_{22}y + a_{23} - a_{31}xv - a_{32}yv &= v \end{aligned}$$

It can be represented by matrix multiplication for coordinate pairs $(x_i, y_i)$ and $(u_i, v_i)$:

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_iu_i & -y_iu_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_iv_i & -y_iv_i \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

Suppose now we have $n$ pairs of associated points, the equations can be represented by:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ & & & & & \cdots & & \\ & & & & & \cdots & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nu_n & -y_nu_n \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_nv_n & -y_nv_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \cdots \\ \cdots \\ u_n \\ v_n \end{bmatrix} \tag{1}$$

Define

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ & & & & & \cdots & & \\ & & & & & \cdots & & \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix} \quad P = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} \quad C = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \cdots \\ \cdots \\ u_n \\ v_n \end{bmatrix}$$

Since A has rank 8, but we have more than 8 equations, there's no solution for this, but there exists an optimal solution. We can solve the optimal $P$ by minimizing $\|AP - C\|^2$, this gives us:

$$P = (A^T A)^{-1} A^T C \tag{2}$$

## 2. RANSAC algorithm

RANSAC stands for "Random Sample Consensus". Suppose we have a certain amount of point pairs, and there are some false correspondences in them. RANSAC algorithm will automatically separate the correct correspondences (inliers) and false correspondences (outliers) apart.
The RANSAC runs for $N$ trails, in each trail,

(1) We randomly select a small amount (say, $n$) of pairs and compute $H$ using linear least-square minimization.

(2) After we get $H$, we need to determine inliers and outliers based on the distance between projected point coordinates $Y' = HX$ and the target point coordinates $Y$. For each $Y'_i = (x'_i, y'_i)$ and $Y = (x, y)$,
if $(x'_i - x_i)^2 + (y'_i - y_i)^2 <= d^2$, the $i$-th point pair is an inlier, otherwise it's an outlier. Here $d$ is the distance threshold.

(3) Record the number of inliers, the $H$ associated with the largest number is the best homographic transform matrix.

We need to find the amount of trails before the above steps in order to make sure we can hit the correct $H$ at least once in all trails. Given the worst probability of false correspondence is $\epsilon$, and we want the probability to get the correct $H$ after $N$ trails is $P$, the amount of trails $N$ can be computed as:

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

Once we find the $H$ associated with the most inliers, we can compute the optimal $H$ with utilizing all inliers by the linear least-squares minimization.

## 3. Levenberg-Marquardt Algorithm to refine homography

The linear least-square method with redundant correspondences can produce homography $H$ which minimizes $\|HX - Y\|^2$ by minimizing $\|AP - C\|^2$. The projection result is noisy because of the imaging distortion, the resolution of images and the SIFT corner detection. To further refine $H$, we directly minimize $\|HX - Y\|^2$ by a powerful non-linear optimization method, Levenberg-Marquardt (LM) algorithm.
Assume $H' = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} & a_{31} & a_{32} & a_{33} \end{bmatrix}^T$, we define our optimization as:

$$\min_{H'} E(H') = \min_{p} \left\| f(H') - Y \right\|^2$$

Where

$$f(P) = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix} \qquad Y = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \qquad E(H') = \begin{bmatrix} e_1(H') \\ e_2(H') \\ e_3(H') \\ e_4(H') \\ \vdots \\ e_{2n-1}(H') \\ e_{2n}(H') \end{bmatrix}$$

The steps of LM algorithm are described below:

(1) Initialize the parameter vector $H'$ by the result $H$ from RANSAC.. Set parameters $\tau$ and $\zeta$.

(2) Compute the Jacobian matrix of $E$ w.r.t. parameter vector $P$, it is:

$$J_E(H') = \begin{bmatrix} \frac{\partial e_1}{\partial a_{11}} & \frac{\partial e_1}{\partial a_{12}} & \cdots & \frac{\partial e_1}{\partial a_{33}} \\ \frac{\partial e_2}{\partial a_{11}} & \frac{\partial e_2}{\partial a_{12}} & \cdots & \frac{\partial e_2}{\partial a_{33}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{2n}}{\partial a_{11}} & \frac{\partial e_{2n}}{\partial a_{12}} & \cdots & \frac{\partial e_{2n}}{\partial a_{33}} \end{bmatrix}$$

For the $(2i-1)$-th and $2i$-th rows in $J_E$, associated with $X_i = (x_i, y_i, 1)$ and $Y_i' = (x_i', y_i', w_i)$, the parameters in $J_E$ are:

$$\begin{aligned}
\frac{\partial e_{2i-1}}{\partial a_{11}} &= \frac{x_i}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{x_i}{w_i} \\
\frac{\partial e_{2i-1}}{\partial a_{12}} &= \frac{y_i}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{y_i}{w_i} \\
\frac{\partial e_{2i-1}}{\partial a_{13}} &= \frac{1}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{1}{w_i} \\
\frac{\partial e_{2i-1}}{\partial a_{21}} &= \frac{\partial e_{2i-1}}{\partial a_{22}} = \frac{\partial e_{2i-1}}{\partial a_{23}} = 0 \\
\frac{\partial e_{2i-1}}{\partial a_{31}} &= \frac{-x_i(a_{11}x_i + a_{12}y_i + a_{13})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{x_i x_i'}{w_i^2} \\
\frac{\partial e_{2i-1}}{\partial a_{32}} &= \frac{-y_i(a_{11}x_i + a_{12}y_i + a_{13})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{y_i x_i'}{w_i^2} \\
\frac{\partial e_{2i-1}}{\partial a_{33}} &= \frac{-(a_{11}x_i + a_{12}y_i + a_{13})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{x_i'}{w_i^2} \\
\frac{\partial e_{2i}}{\partial a_{11}} &= \frac{\partial e_{2i}}{\partial a_{12}} = \frac{\partial e_{2i}}{\partial a_{13}} = 0 \\
\frac{\partial e_{2i}}{\partial a_{21}} &= \frac{x_i}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{x_i}{w_i} \\
\frac{\partial e_{2i}}{\partial a_{22}} &= \frac{y_i}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{y_i}{w_i} \\
\frac{\partial e_{2i}}{\partial a_{23}} &= \frac{1}{a_{31}x_i + a_{32}y_i + a_{33}} = \frac{1}{w_i} \\
\frac{\partial e_{2i}}{\partial a_{31}} &= \frac{-x_i(a_{21}x_i + a_{22}y_i + a_{23})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{x_i y_i'}{w_i^2} \\
\frac{\partial e_{2i}}{\partial a_{32}} &= \frac{-y_i(a_{21}x_i + a_{22}y_i + a_{23})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{y_i y_i'}{w_i^2} \\
\frac{\partial e_{2i}}{\partial a_{33}} &= \frac{-(a_{21}x_i + a_{22}y_i + a_{23})}{(a_{31}x_i + a_{32}y_i + a_{33})^2} = -\frac{y_i'}{w_i^2}
\end{aligned}$$

Use the initial $J_E(H')$, we get the initial damping coefficient $\mu_0 = \tau \times max(diag(J_E^T J_E))$

(3) Start the iteration until $||\delta_k||_2 <= \zeta$, in iteration $k$, we implement steps (4)-(9)

(4) Update the Jacobian $J_E = J_E(H_k')$, and calculate the increment to $H_k'$:

$$\delta_k = -(J_E^T J_E + \mu_k I)^{-1} J_E E(H_k')$$

(5) Compute $H'_{k+1} = H'_k + \delta_k$

(6) Compute the errors (costs): $C(H'_k) = \|E(H'_k)\|_2$ and $C(H'_{k+1}) = \left\|E(H'_{k+1})\right\|_2$

(7) Compute the ratio of the actual change in the cost function

$$\rho = \frac{C(H'_{k+1}) - C(H'_k)}{-\delta_k^T J_E^T E(H'_k) + \delta_k^T \mu_k I \delta_k}$$

(8) Update the damping coefficient for the next step

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

(9) Update $H'_{k+1}$ as $H'_k$ if $C(H'_{k+1}) > C(H'_k)$

### Implement steps and Parameters

The image sequence we used in the experiments is shown below:

Figure 1: Image 1

Figure 2: Image 2

Figure 3: Image 3

Figure 4: Image 4

Figure 5: Image 5

### 1. Extract interest points by SIFT method and matching point pairs

The SIFT method is implemented by utilizing the OpenCV library. We called **cv2.xfeatures2d.SIFT_create** method to create SIFT operators and the parameters are set as: n_feature=1000, nOctaveLayers=4, sigma=2.
To match the correspondences, we used NCC method to compute the corresponding scores just like in homework 4. We collected all matched pairs and their scores, we finally accept 100 pairs with the top-100 highest scores.

### 2. Use RANSAC with LLS algorithm to compute homography

The parameters are set as:

- $P = 0.999$
- $\epsilon = 0.1$. The largest false matching probability is selected based on the last matching step.
- $n = 5$
- $d = 3$. If distance threshold is too large, the outliers can't be all filtered out, if the distance threshold is too small, a lot of inliers will also be falsely classified as ourliers, we can only select a small area on the image pairs and this will make the major area be transformed inaccurately later.

### 3. LM Method to refine the homography

The parameters are set as $\tau = 0.5$ and $\zeta = 10^{-23}$.

### 4. Stitch all images together

Iterate all image pairs we get $H_{12}$, $H_{23}$, $H_{34}$, $H_{45}$.
We select an anchor image from all images which will be directly copied on the finally concatenated image, then we compute the homographies that transform other images to it. Assume it's the 3rd image, we need to compute $H_{13}$, $H_{23}$, $H_{43}$, $H_{53}$.
Determine the range of the stitched image. Project 4 vertices on other 4 images to the anchor image space to find the range. We need to extract the offsets on both height and width directions, and if the range is too large, we also need to find a resize parameter to map the large range to a relatively small one. Just like what we did in homework3, transform all 5 images to the stitched image.

## Results

**1. Paired images – the inliers are connected by green lines and the ourliers are connected by red lines. All interest points are marked by red circles**

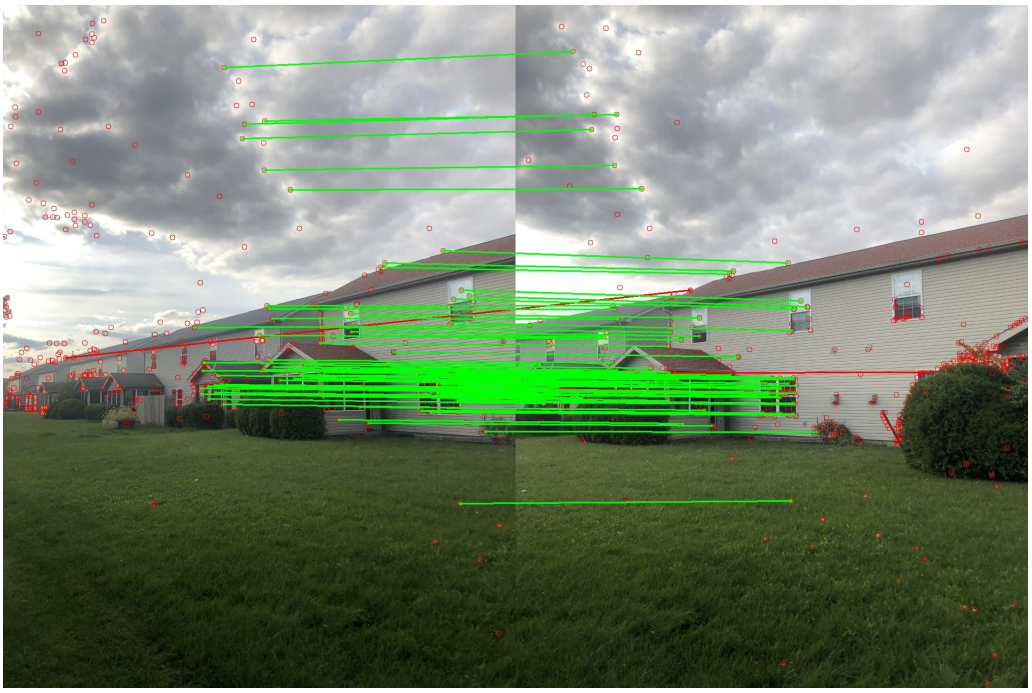

Figure 6: Correspondences found between image 1 and 2

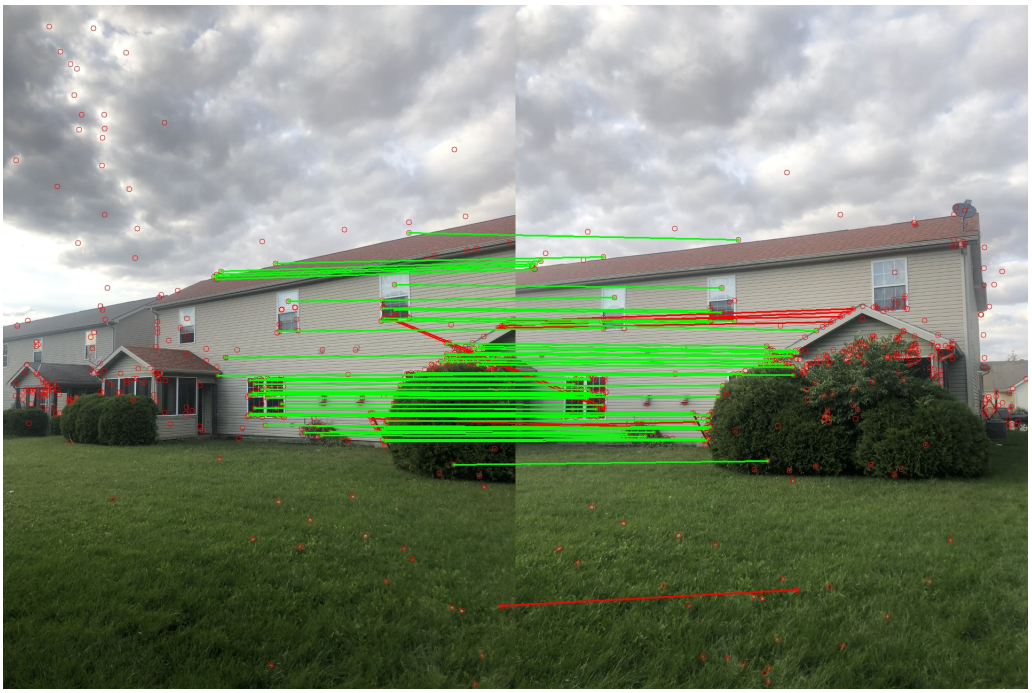

Figure 7: Correspondences found between image 2 and 3

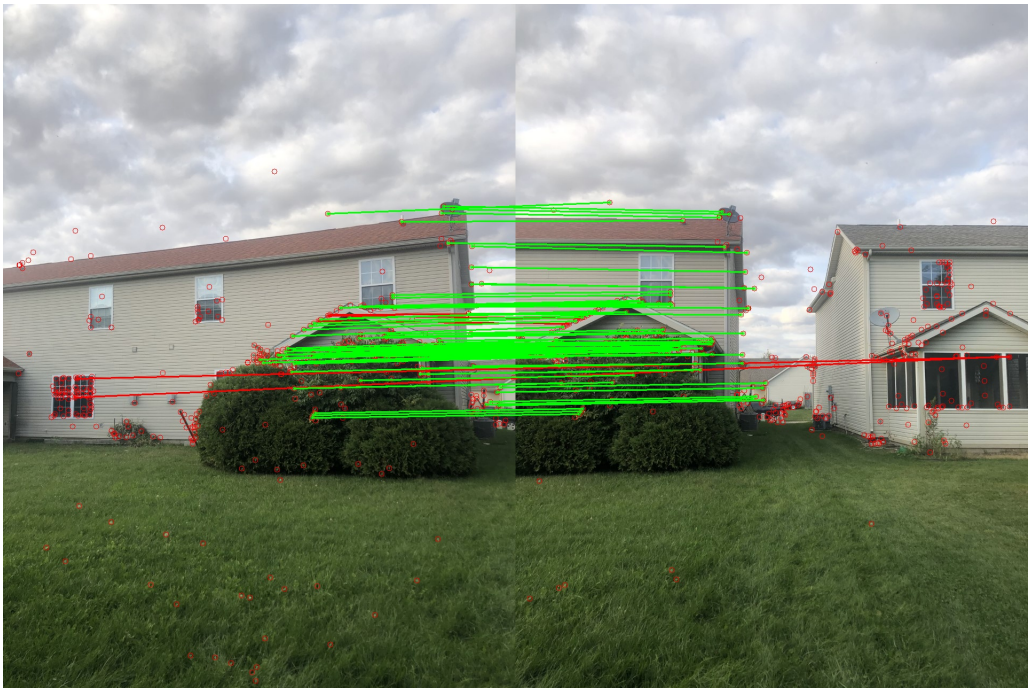

Figure 8: Correspondences found between image 3 and 4

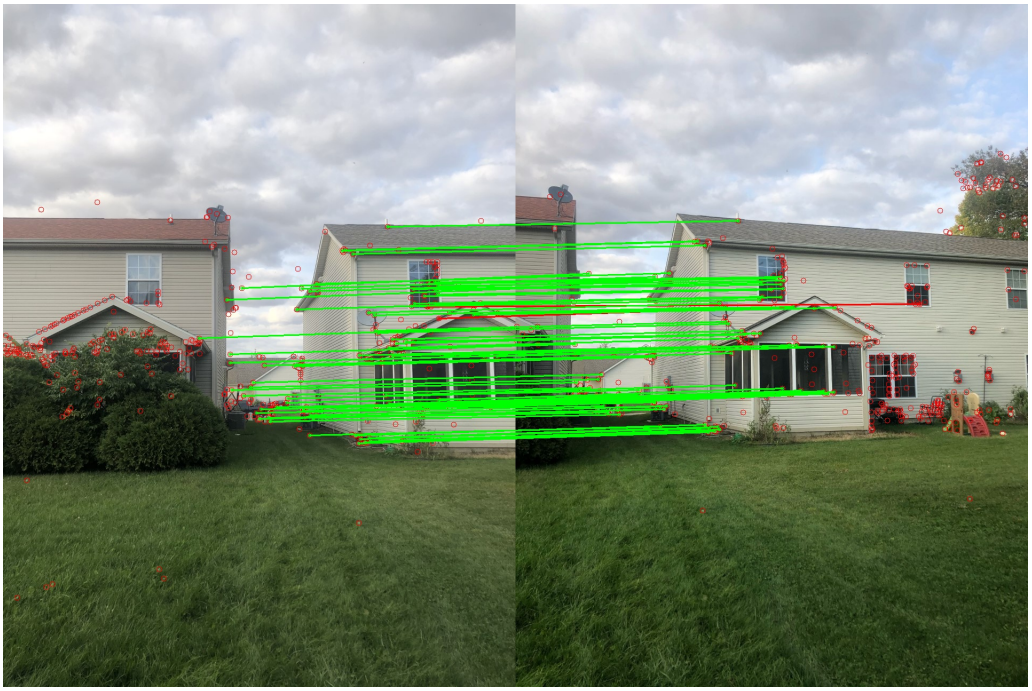

Figure 9: Correspondences found between image 4 and 5

**2. Stitched images**

Figure 10: Panorama before LM refinement

Figure 11: Panorama after LM refinement

## Code

```
import numpy as np
import cv2
import os
import math

#
# ------------------------ Step1: interest point matching ---------------------
#
def DrawMarker(Img, pList, color=(0,0,255)):
    result = np.copy(Img)
    for j, i in pList:
        result = cv2.circle(result, (j, i), 5, color, 1)
    return result


def cpMatching(f1, f2, method):
    d1 = f1.shape[0]
    d2 = f2.shape[0]
    if method == 'SSD':
        f11 = np.sum(f1*f1, axis=1).reshape((d1, 1))
        f12 = np.dot(f1, f2.T)
        f22 = np.sum(f2*f2, axis=1).reshape((1, d2))
        cp = -f11.repeat(d2, axis=1) + 2 * f12 - f22.repeat(d1, axis=0)
    elif method == 'NCC':
        m1 = np.sum(f1, axis=1, keepdims=True)/f1.shape[1]
        m2 = np.sum(f2, axis=1, keepdims=True)/f2.shape[1]
        nf1 = f1 - m1
        nf2 = f2 - m2
        numerator = np.dot(nf1, nf2.T)
        denom_1 = np.sqrt(np.sum(nf1*nf1, axis=1).reshape((d1, 1)))
        denom_2 = np.sqrt(np.sum(nf2*nf2, axis=1).reshape((1, d2)))
        denominator = np.dot(denom_1, denom_2)
        cp = numerator / denominator
    else: print("\'Method\' has to be SSD or NCC")
    f1_Matching = np.argmax(cp, axis=1)
    f2_Matching = np.argmax(cp, axis=0)
    return cp, f1_Matching, f2_Matching


def matchedImg(img1, img2, pList1, pList2, idx1, idx2):
    def drawLines(img, p_start, p_end, color=(0,255,0)):
        image = np.copy(img)
        for i in range(len(p_start)):
            image = cv2.circle(image, p_start[i], 3, color, 1)
            image = cv2.circle(image, p_end[i], 3, color, 1)
            image = cv2.line(image, p_start[i], p_end[i], color, 2)
        return image

    im1 = np.copy(img1)
    im2 = np.copy(img2)

    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    if h1 > h2:
        im2 = cv2.copyMakeBorder(im2, 0, h1-h2, 0, 0, cv2.BORDER_REPLICATE)
        h = h1
```

```
    elif h1 < h2:
        im1 = cv2.copyMakeBorder(im1, 0, h2-h1, 0, 0, cv2.BORDER_REPLICATE)
        h = h2
    else: h = h1

    if w1 > w2:
        im2 = cv2.copyMakeBorder(im2, 0, 0, 0, w1-w2, cv2.BORDER_REPLICATE)
        w = w1
    elif w1 < w2:
        im1 = cv2.copyMakeBorder(im1, 0, 0, 0, w2-w1, cv2.BORDER_REPLICATE)
        w = w2
    else: w = w1

    concatIm = np.concatenate((im1, im2), axis=1)
    npList2 = [(j+w, i) for (j, i) in pList2]
    concatIm = DrawMarker(concatIm, pList1, color=(255,0,0))
    concatIm = DrawMarker(concatIm, npList2, color=(255,0,0))

    bestMatch_1 = [pList1[i] for i in idx1]
    bestMatch_2 = [npList2[i] for i in idx2]
    print('detected '+str(len(pList1))+ ' corners in image 1')
    print('detected '+str(len(pList2))+ ' corners in image 2')
    print('found '+str(len(bestMatch_1))+ ' matched corners between image 1 and 2'
    )
    matchedIm = drawLines(concatIm, bestMatch_1, bestMatch_2, color=(255,0,0))
    return concatIm, matchedIm


def SIFTmatching(img1, img2, nfeatures=100, max_pairs=100):
    g_1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
    g_2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)

    sift = cv2.xfeatures2d.SIFT_create(nfeatures,
                                       nOctaveLayers=4,
                                       contrastThreshold=0.05,
                                       edgeThreshold=10,
                                       sigma=2)
    kp1, des1 = sift.detectAndCompute(g_1, None)
    kp2, des2 = sift.detectAndCompute(g_2, None)
    pList1 = [(int(kp1[i].pt[0]), int(kp1[i].pt[1])) for i in range(len(kp1))]
    pList2 = [(int(kp2[i].pt[0]), int(kp2[i].pt[1])) for i in range(len(kp2))]
    cp, idn_1, idn_2 = cpMatching(des1, des2, method='NCC')
    pidxList1, pidxList2, score = [], [], []
    for i, idx2 in enumerate(idn_1):
        if i==idn_2[idx2]:
            pidxList1.append(i)
            pidxList2.append(idx2)
            score.append(cp[i, idx2])
    zipped = list(zip(score, pidxList1, pidxList2))
    zipped.sort(reverse=True)
    score, pidxList1, pidxList2 = zip(*zipped)
    concatIm, matchedIm = matchedImg(img1, img2, pList1, pList2, pidxList1[:
    max_pairs], pidxList2[:max_pairs])
    matchedcorners1 = np.asarray([(kp1[i].pt[0], kp1[i].pt[1]) for i in pidxList1
    [:max_pairs]])
    matchedcorners2 = np.asarray([(kp2[i].pt[0], kp2[i].pt[1]) for i in pidxList2
    [:max_pairs]])
    return concatIm, matchedIm, matchedcorners1, matchedcorners2
```

```
#
# ----------------------------- Step2: RANSAC -----------------------------
#
def findHomoproj(source, target):
    def F_unit(source_point, target_point):
        x, y = source_point[0], source_point[1]
        x_, y_ = target_point[0], target_point[1]
        return np.asarray([[x, y, 1, 0, 0, 0, -x*x_, -y*x_],
                           [0, 0, 0, x, y, 1, -x*y_, -y*y_]])
    F_list = [F_unit(source[i], target[i]) for i in range(source.shape[0])]
    F = np.concatenate(F_list, axis=0)
    T_span = target.reshape((-1,1))
    H_param = np.dot(np.linalg.pinv(F), T_span)
    H = np.ones((9, 1))
    H[:8, :] = H_param
    return H.reshape((3, 3))


def findInliers(src, tgt, H, dt):
    nps = src.shape[0]
    X = np.concatenate((src, np.ones((nps,1))), axis=1)
    Y_ = np.dot(H, X.T).T
    Y_ = Y_ / Y_[:,2].reshape((nps,1))
    e = tgt - Y_[:,:2]
    e_norm = np.linalg.norm(e, axis=1)
    idx_in = np.where(e_norm<=dt)
    num_in = idx_in[0].shape[0]
    return idx_in, num_in


def markInliers(img, sp, tp, idx, color=(0,255,0)):
    image = np.copy(img)
    w = int(img.shape[1]/2)
    for i in idx:
        p_start = (int(sp[i,0]), int(sp[i,1]))
        p_end = (int(tp[i,0] + w), int(tp[i,1]))
        image = cv2.circle(image, p_start, 3, color, 1)
        image = cv2.circle(image, p_end, 3, color, 1)
        image = cv2.line(image, p_start, p_end, color, 2)
    return image


def RANSAC(src, tgt, dt, n, p, ep):
    n_total = src.shape[0]
    M = int((1 - ep) * n_total)
    N = int(math.log(1-p) / math.log(1-(1-ep)**n))
    max_inlier = 0
    found_idx = ()
    for i in range(N):
        idx = np.random.randint(0, n_total, n)
        src_trial = src[idx]
        tgt_trial = tgt[idx]
        H_homo = findHomoproj(src_trial, tgt_trial)
        idx_in, num_in = findInliers(src, tgt, H_homo, dt)
        if num_in > max_inlier:
            max_inlier = num_in
            found_idx = idx_in
    H_RANSAC = findHomoproj(src[found_idx], tgt[found_idx])
    if max_inlier >= M:
```

```
        print('found at most', max_inlier, 'inliers')
    else:
        print('No enough inliers detected! Require at least', M, 'but found',
    max_inlier)
    return H_RANSAC, found_idx[0]


#
# ------------------------- Step3: Levenberg-Marquardt -----------------------
#
def findJacobian(source, target, H):
    def J_unit(source_point, target_point):
        x, y = source_point[0], source_point[1]
        x_, y_, w_ = target_point[0], target_point[1], target_point[2]
        w_2 = w_ ** 2
        return np.asarray([[x/w_, y/w_, 1/w_, 0, 0, 0, -x*x_/w_2, -y*x_/w_2, -x_/
    w_2],
                           [0, 0, 0, x/w_, y/w_, 1/w_, -x*y_/w_2, -y*y_/w_2, -y_/w_2
    ]])

    nps = source.shape[0]
    source_rep = np.concatenate((source, np.ones((nps,1))), axis=1)
    tgt_rep = np.dot(H, source_rep.T).T
    J_list = [J_unit(source[i], tgt_rep[i]) for i in range(nps)]
    J = np.concatenate(J_list, axis=0)
    tgt_norm = tgt_rep[:,:2] / tgt_rep[:,2].reshape((nps,1))
    E = target - tgt_norm.reshape((-1,))
    return J, E


def getError(src, tgt, H):
    nps = src.shape[0]
    source_rep = np.concatenate((src, np.ones((nps,1))), axis=1)
    tgt_rep = np.dot(H, source_rep.T).T
    tgt_norm = tgt_rep[:,:2] / tgt_rep[:,2].reshape((nps,1))
    E = tgt - tgt_norm.reshape((-1, ))
    e = np.dot(E.T, E)
    return e


def projTransform(H, source, flatten=True):
    nps = source.shape[0]
    source_rep = np.concatenate((source, np.ones((nps,1))), axis=1)
    t_homo = np.dot(H, source_rep.T).T
    t_norm = t_homo[:,:2] / t_homo[:,2].reshape((nps,1))
    if flatten:
        return t_norm.reshape((-1,))
    else:
        return t_norm

def LM_Refine(src_pts, tgt, H_init, ep, tau=0.5):
    I = np.identity(9)
    tgt = tgt.reshape((-1,))

    #Initialize parameters
    J, ek = findJacobian(src_pts, tgt, H_init)
    mu_init = tau * np.max(np.diagonal(np.dot(J.T, J)))
    mu = mu_init
    H_k = H_init
```

```
    step = 0
    errors = []
    delta_norm = 1
    while delta_norm >= ep:
        Cp = getError(src_pts, tgt, H_k)

        J, ek = findJacobian(src_pts, tgt, H_k)
        delta = np.dot(np.dot(np.linalg.inv(np.dot(J.T, J) + mu * I), J.T), ek)
        delta_norm = np.linalg.norm(delta)

        H_k1 = H_k + delta.reshape((3, 3))
        Cp_next = getError(src_pts, tgt, H_k1)
        errors.append(Cp_next)

        rho_num = Cp - Cp_next
        rho = rho_num / (np.dot(np.dot(delta, J.T), ek) + np.dot(np.dot(delta, mu
   * I), delta))

        mu_next = mu * max(1 / 3, 1 - (2 * rho - 1) ** 3)
        print("step =", step, "  mu =", mu, "  delta_norm =", delta_norm, " Cp =",
    Cp)

        step += 1
        mu = mu_next
        if mu == np.inf:
            break
        if rho_num >= 0:
            H_k = H_k1

    H_NonLinear = H_k
    return H_NonLinear, errors


#
# ------------------------- Step4: stitch images together ----------------------
#
def filterH(H_list, center):
    H_num = len(H_list)
    left = H_list[:center]
    right = [np.linalg.inv(H) for H in H_list[center:]]
    if center >= 2:
        left.reverse()
        for i in range(1, center):
            left[i] = np.dot(left[i], left[i-1])
        left.reverse()
    if center < H_num-1:
        for i in range(1, H_num - center):
            right[i] = np.dot(right[i], right[i-1])
    return left + [np.eye(3)] + right

def findRange(h, w, H_list):
    original_range = np.asarray([[0,0], [w, 0], [w, h], [0, h]])
    ranges = []
    for H in H_list:
        new_range = projTransform(H, original_range, flatten=False).astype(int)
        ranges.append(new_range)
    return ranges
```

```
def projRange(source, target, H, srange, offsetH, offsetW):
    mask = np.zeros_like(source)
    srange -= np.asarray([[offsetW, offsetH]])
    mask = cv2.fillPoly(mask,[srange],(255,255,255))
    for i in range(source.shape[0]):
        for j in range(source.shape[1]):
            if any(mask[i][j]) > 0:
                coor_source = np.asarray([[j+offsetW, i+offsetH]])
                coor_target = projTransform(H, coor_source, flatten=False).squeeze
    ()
                source[i,j,:] = target[int(coor_target[1]), int(coor_target[0]),:]
    return source, mask


def stitch(images, H_list, center=2):
    h, w = images[0].shape[0], images[0].shape[1]
    new_list = filterH(H_list, center)
    ranges = findRange(h-1, w-1, new_list)
    vertices = np.concatenate(ranges, axis=0)

    l, r = np.amin(vertices[:, 0]), np.amax(vertices[:, 0])
    t, b = np.amin(vertices[:, 1]), np.amax(vertices[:, 1])
    new_W = r-l
    new_h = b-t
    offsetW = l
    offsetH = t
    template = np.zeros((new_h, new_W, 3))

    for img_idx in range(len(images)):
        srange = ranges[img_idx]
        template, mask = projRange(template, images[img_idx],
                                   np.linalg.inv(new_list[img_idx]),
                                   srange, offsetH, offsetW)
    result = cv2.cvtColor(template.astype('uint8'), cv2.COLOR_RGB2BGR)
    return result


if __name__ == "__main__":
    folder_task = '/home/xingguang/Documents/ECE661/hw5/images'
    images = []
    out_path = []
    for i in range(1,6):
        path = os.path.join(folder_task, str(i)+'.jpg')
        out_path.append(os.path.join(folder_task, 'matched'+str(i)+'.jpg'))
        images.append(cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB))

    dt = 10
    n = 6
    p = 0.999
    ep = 0.1
    thr = 1e-23
    H_list = []
    for i in range(4):
        _, matchedIm, corners1, corners2 = SIFTmatching(images[i], images[i+1],
    500, 100)
        H, in_idx = RANSAC(corners1, corners2, dt, n, p, ep)
        filteredIm = markInliers(matchedIm, corners1, corners2, in_idx)
        H_refined, errors = LM_Refine(corners1[in_idx], corners2[in_idx], H, thr)
        H_list.append(H_refined)
```

```
        cv2.imwrite(out_path[i], cv2.cvtColor(filteredIm, cv2.COLOR_RGB2BGR))

    result = stitch(images, H_list, center=2)
    cv2.imwrite('/home/xingguang/Documents/ECE661/hw5/images/panoramic.jpg',
    result)
```