School of Electrical and Computer Engineering
Purdue University, WL, IN, USA

Nahian Ibn Hasan
Email: hasan34@purdue.edu
PUID: 0032764564
ECE66100 - Computer Vission
Fall 2022
Homework-5

October 12, 2022

1 Objective

In this homework, we will learn to implement a fully automated approach for robust homography estimation and subsequently refine it using Nonlinear Least-Squares minimization approaches such as Levenberg-Marquardt (LM) algorithm.

2 Theoretical Questions

2.1 Question 1

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

2.2 Answer

At the first stage of RANSAC, we randomly select n correspondences without replacement. Next, we calculate Homography using these n correspondences. For our calculation we selected $n = 20$. Next, apply the homography to all interest points in the image 1. We already know the ground truth transformed points. Hence, we can threshold the distance between the estimated and ground truth projected interest points in image 2. For our case, the threshold is $100 \times \sigma$, where $\sigma = 2$. If the distance is less than this threshold, we call the correspondence as inliers, otherwise, outliers.

2.3 Question 2

As you will see in Lecture 13, 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.

2.4 Answer

GD method becomes very slow as the solution gets closer to the actual solution, because of smaller step sizes. On the other hand, GN is fast but it can fail if the Jacobian is not of full rank. Additionally, the initial guess should bbe close to the actual solution. Levenberg-Marquardt (LM) method combines both of GD nad GN method. At high level, LM behaves like GD as the solution is far from actual. It starts to behave like GN as the solution gets closer to actual. Adding a damping factor to the Jacobian can combine these two. For high damping factor, the solution is similar to GD and for low damping factor, it's similar to GN. The damping factor is inversely related to distance between current solution and actual solution.

3 Task 1: Linear Least Square Homography Estimation

3.1 Algorithm

Given a point x in the planar scene and the corresponding point in the x' in the image plane, the homographic transformation can be written as

$$X' = HX. \tag{1}$$

Here, x and x' are represented in homographic coordinates and H is the homographic matrix. Therefore,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}; X' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}; H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \tag{2}$$

The expression in equation 1 becomes

$$x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \tag{3a}$$
$$x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \tag{3b}$$
$$x'_3 = h_{31}x_1 + h_{32}x_2 + h_{33}x_3 \tag{3c}$$

Now, we can transform these homographic coordinates to physical coordinates by the following transformations-

$$x = \frac{x_1}{x_3}; y = \frac{x_2}{x_3}; x' = \frac{x'_1}{x'_3}; y' = \frac{x'_2}{x'_3} \tag{4}$$

Here, (x, y) is the physical scene coordinate and (x', y') is the physical image plane coordinate. Hence, using equations in 3c and 4, we get

$$x' = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3} \tag{5a}$$
$$y' = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3} \tag{5b}$$

Or,

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{6a}$$
$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{6b}$$

Or,

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - h_{33}x' = 0 \tag{7a}$$
$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' - h_{33}y' = 0 \tag{7b}$$

Let's assume $h_{33} = 1$. Now, if we have 4 pairs of physical coordinates in the physical scene and physical image plane, we can evaluate equation 7b at those four pairs and end-up with 8 equations, which can be solved later to find out the coefficients of homographic matrix H. Let's assume the four pairs are $\{(x_1, y_1), (x'_1, y'_1)\}$, $\{(x_2, y_2), (x'_2, y'_2)\}$, $\{(x_3, y_3), (x'_3, y'_3)\}$ and $\{(x_4, y_4), (x'_4, y'_4)\}$. Hence, the resultant equations can be expressed in matrix notation as follows-

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} \tag{8}$$

Or,

$$Ah = b. \tag{9}$$

The system of linear equations in 8 can be solved as $h = A^{-1}b$.

However, if the matrix A is not square, in other words, the homography is estimated with more than 4 points, we cannot inverse the matrix A. Then the least square minimization must satisfy

$$(A^T A)h = A^T b \tag{10}$$

Then we would have to use the pseudoinverse of A. The pseudoinverse of A is A^+ calculated as-

$$A^+ = (A^T A)^{-1} A^T \tag{11}$$

Therefore,

$$h = A^+ b \tag{12}$$

4 RANSAC Algorithm

While using more than 4 points for homography estimation, there are some correspondences which are false or outliers. Using Random Sampling and Consensus Method (RANSAC), we can identify these outliers and inliers. But before applying the RANSAC, we need to find out the interest points using any kinfd of interest point detectors such as SIFT or SURF. Next, the corresponding interest points between two sets of interest points are detected using suitable distance metrics such as SUM of Squared Differences (SSD) or Normalized Cross Correlation (NCC). After that we can apply the RANSAC algorithm. At the first stage, we randomly select n correspondences without replacement. Next, we calculate Homography using these n correspondences. For our calculation we selected $n = 20$. Next, apply the homography to all interest points in the image 1. We already know the ground truth transformed points. Hence, we can threshold the distance between the estimated and ground truth projected interest points in image 2. For our case, the threshold is $100 \times \sigma$, where $\sigma = 2$. Repeat this process N times, where $N = \frac{ln(1-p}{ln(1-(1-\epsilon)^n)}, p = 0.99, \epsilon = 0.25$. Find the largest inlier set among these N trials. The largest set should have at least $M = (1 - \epsilon)n$ correspondences.

5 Non-Linear Least Square Method using Levenberg-Marquardt Method

The goal of non-linear minimization is to minimize a cost function of type $C(p) = ||X - f(p)||^2$. X is the ground truth result, $f(p)$ is the estimated results. In our case, p represents the eight elements of the H matrix. p is initially set to the output from the RANSAC algorithm. $f(p)$ is found by applying the homography to the domain image. The LM method combines the GS and GN method. The minimal solution from the GN method is like -

$$\delta_P = (J_f^T J_f)^{-1} J_f^T \epsilon(p) \tag{13}$$

Here, J_f is the jacobian of f with repspect to p. If $J_f^T J_f$ is diagonal, GN and GD provide the same result. With a damping factor of μ, the results of GD can be incorporated in GN as follows-

$$(J_f^T J_f + \mu I)\delta_p = J_f^T \epsilon(p) \tag{14}$$

Therefore, at step k, the result from LM method is -

$$\delta_p = (J_f^T J_f + \mu I)^{-1} J_f^T \epsilon(p_k) \tag{15}$$

The new minimized value will be $p_{k+1} = p_k + \delta_p$. The μ for the next iteration is calculated as follows-

$$\mu_{k+1} = \mu_k . max\{\frac{1}{3}, 1 - (2\rho_{k+1} - 1)^3\} \tag{16}$$

Here,

$$\rho_{k+1} = \frac{C(p) - C(p_{k+1})}{\delta_p^T J_f^T \epsilon(p_k) + \delta^T \mu_k I \delta_p} \tag{17}$$

After a certain number of iterations, the resultant p is the final result of the LM method.

6 Results

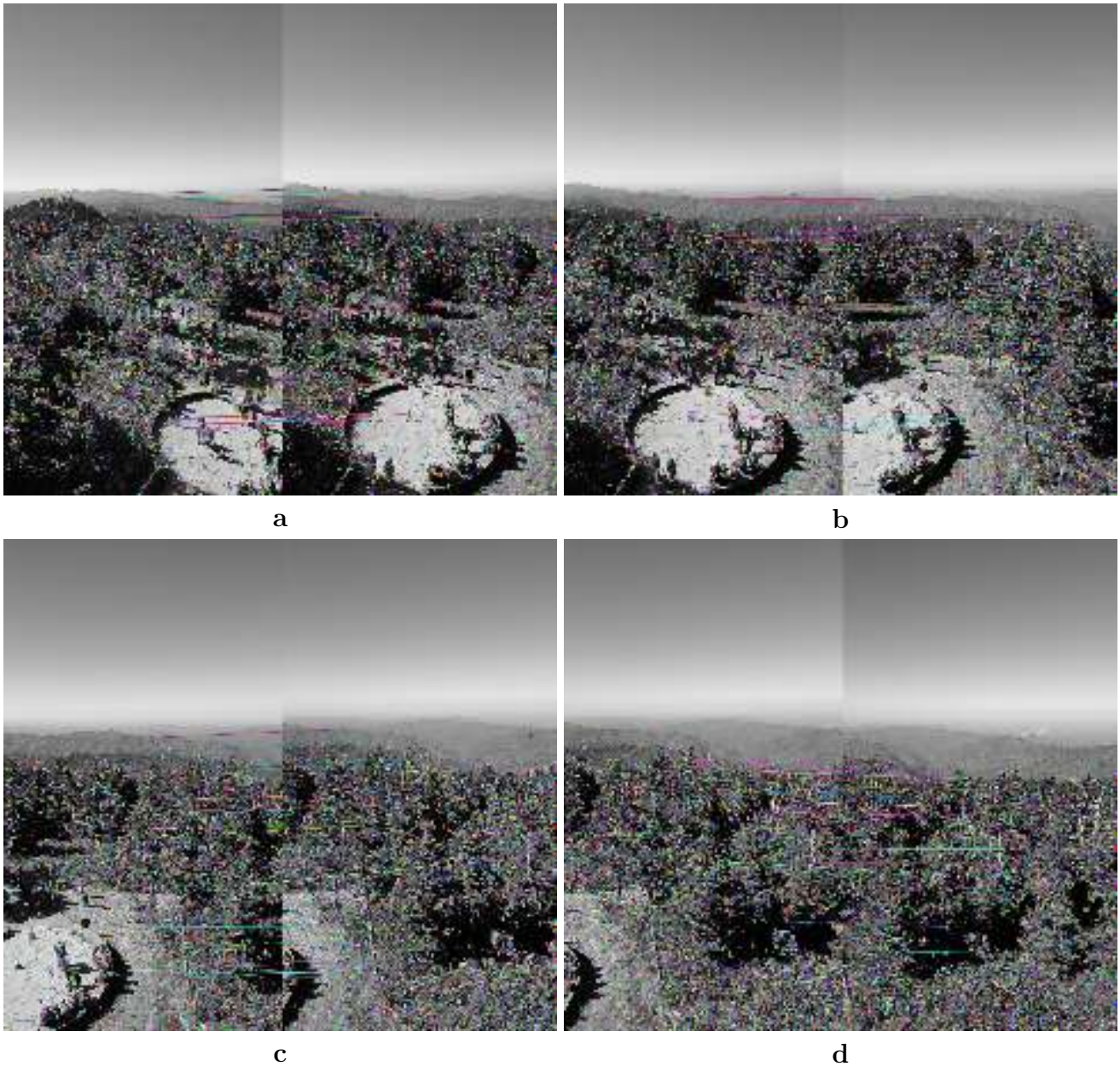
6.1 Input Images

a **b** **c**

d **e**

Figure 1: Input Images

6.2 Image Correspondences (Pairwise)

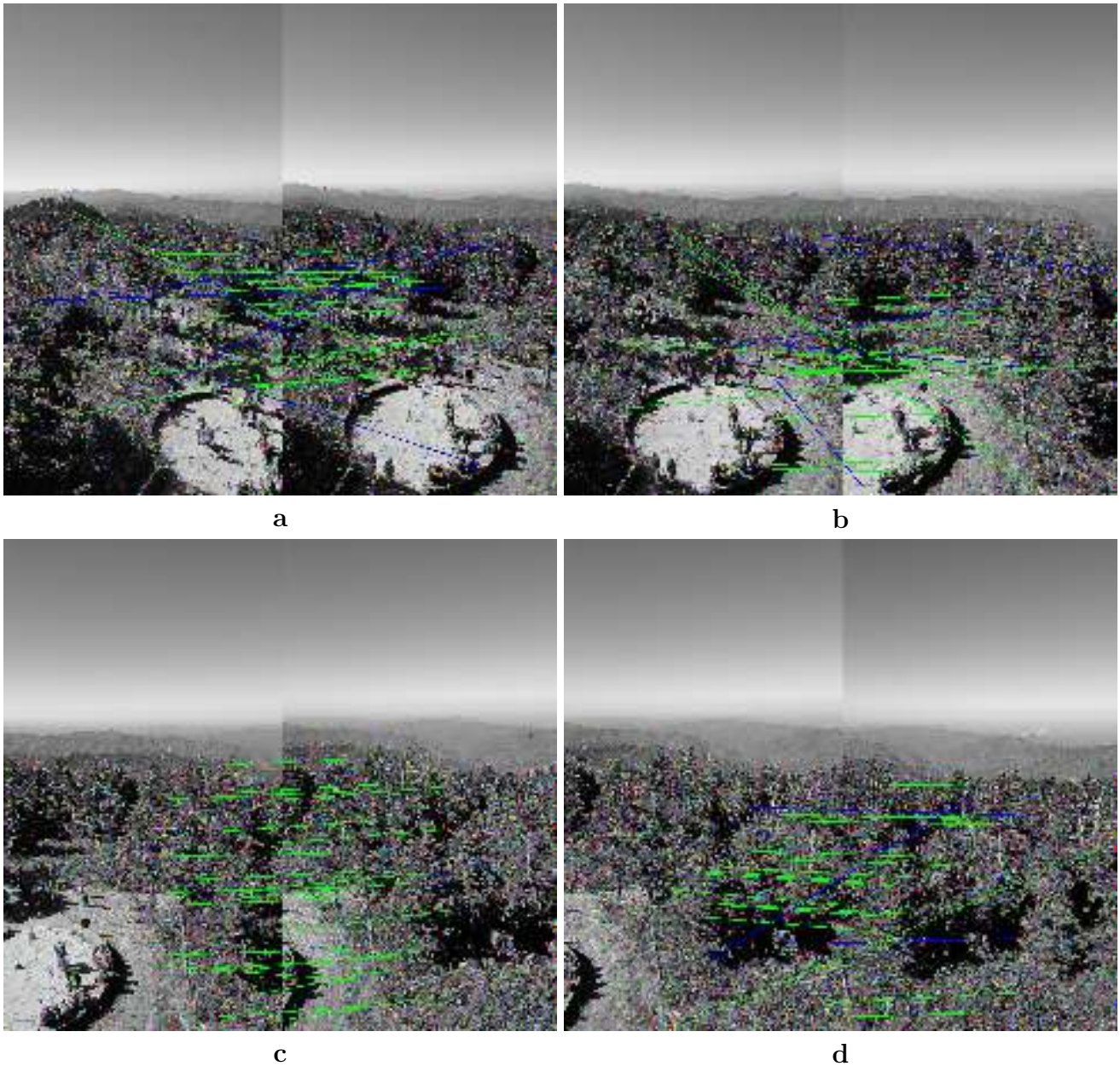


a **b**

c **d**

Figure 2: Image Correspondences between (a)image 1 and 2, (b)image 2 and 3, (c)image 3 and 4, (d)image 4 and 5. Sample 25 correspondences are shown.

6.3 Best Inlier Correspondences from RANSAC (Pairwise)



a **b**

c **d**

Figure 3: Image Inlier Correspondences between (a)image 1 and 2, (b)image 2 and 3, (c)image 3 and 4, (d)image 4 and 5. Inliers are calculated using RANSAC algorithm. The blue correspondences are outliers and the green lines are inliers.

6.4 Panorama Formation

Figure 4: Panorama Image

7 Custom Image Results

7.1 Input Images

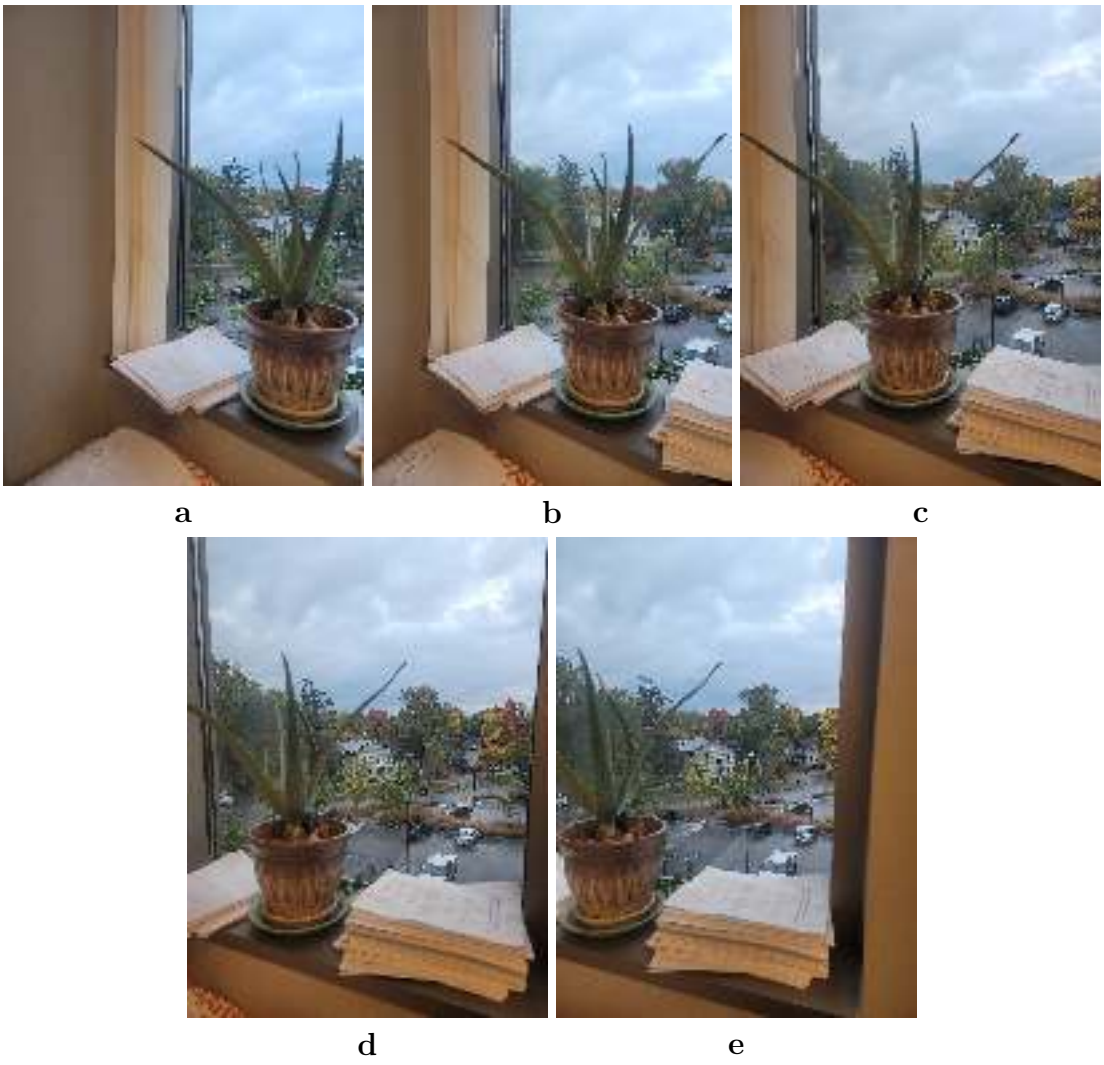
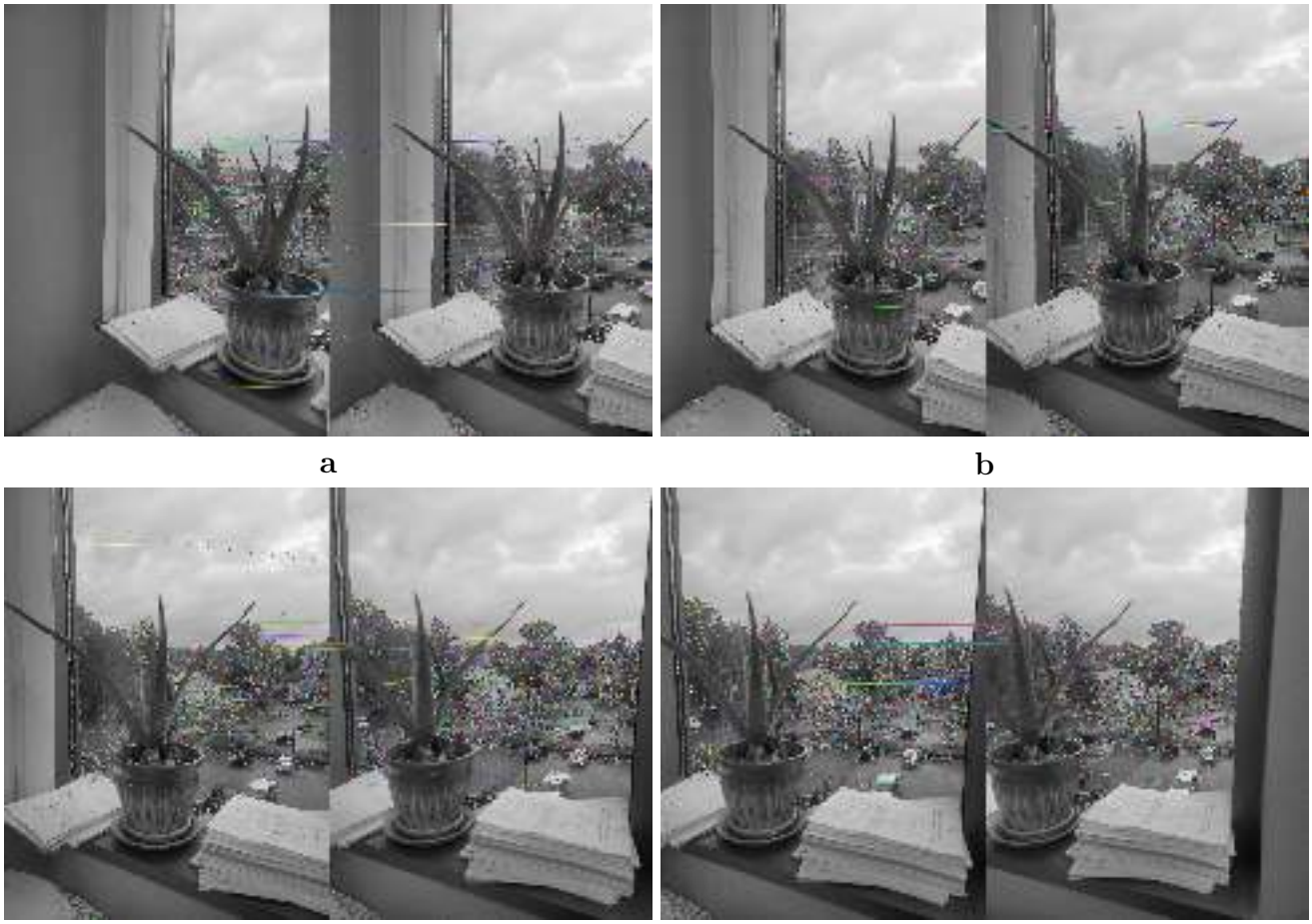


Figure 5: Input Images

7.2 Image Correspondences (Pairwise)

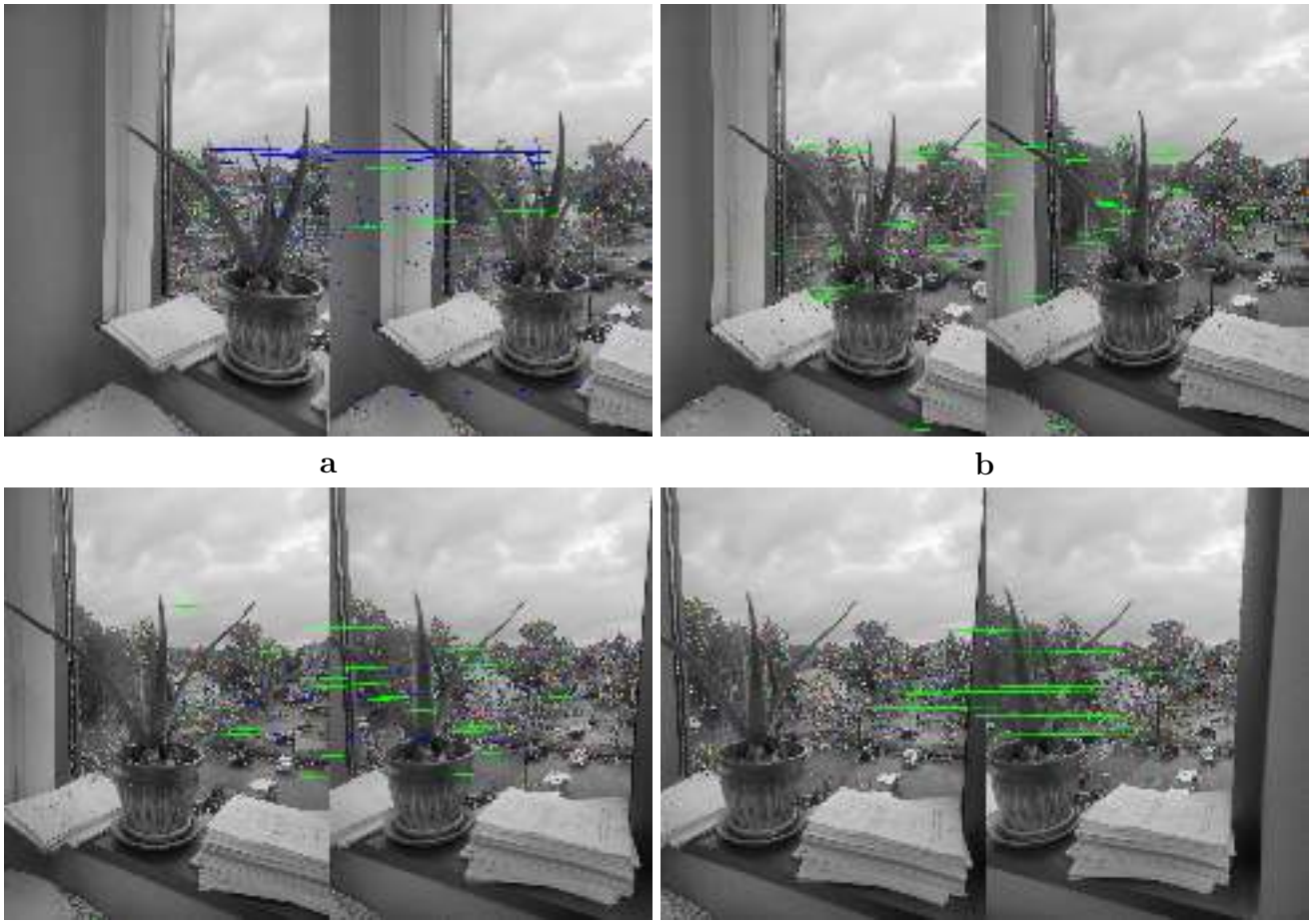


a **b**

c **d**

Figure 6: Image Correspondences between (a)image 1 and 2, (b)image 2 and 3, (c)image 3 and 4, (d)image 4 and 5. Sample 25 correspondences are shown.

7.3 Best Inlier Correspondences from RANSAC (Pairwise)



a **b**

c **d**

Figure 7: Image Inlier Correspondences between (a)image 1 and 2, (b)image 2 and 3, (c)image 3 and 4, (d)image 4 and 5. Inliers are calculated using RANSAC algorithm. The blue correspondences are outliers and the green lines are inliers.

7.4 Panorama Formation

Figure 8: Panorama Image

8 Source Code

```
import cv2
import numpy as np

def LLSM(points_1,points_2):
        A = np.zeros((2*points_1.shape[0],8))
        b = np.zeros((2*points_1.shape[0],1))
        for ix in range(0,points_1.shape[0]):
                A[2*ix,:] = np.array([points_1[ix,0],points_1[ix,1],1,0,0,0,-points_1[ix,0]*poin
                A[2*ix+1,:] = np.array([0,0,0,points_1[ix,0],points_1[ix,1],1,-points_1[ix,0]*po
                b[2*ix] = points_2[ix,0]
                b[2*ix+1] = points_2[ix,1]

        H = np.matmul(calculate_pseudo_inverse(A),b)
        H = np.reshape(np.append(H,1),(3,3))
        return H

def calculate_pseudo_inverse(A):
        return np.matmul(np.linalg.inv(np.matmul(A.T,A)),A.T)
import cv2
import os
import numpy as np

def image_read(file_path):
        image = cv2.imread(file_path)
        dimensions = image.shape#[height. weidth,channels]
        return image, dimensions

def opencv_interest_points(image,detector,out_file):
        gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        if detector.upper()=="SIFT":
                det = cv2.SIFT_create()
        kp,des = det.detectAndCompute(gray,None)
        img=cv2.drawKeypoints(gray,kp,image)
        cv2.imwrite('sift_keypoints_'+out_file+'.jpg',img)
        return img,kp,des

def opencv_correspondence(img_1,kp_1,des_1,img_2,kp_2,des_2,out_file,N):
        #N      : how many correspondences to be kept
        bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)# create BFMatcher object
        matches = bf.match(des_1,des_2)# Match descriptors.
        matches = sorted(matches, key = lambda x:x.distance)# Sort them in the order of their dis
        img3 = cv2.drawMatches(img_1,kp_1,img_2,kp_2,matches[:N], None,flags=2)# Draw first 10 m
        cv2.imwrite('opencv_correspondences_'+out_file+'.jpg',img3)
        return img3,matches

def draw_correspondences(img_1,img_2,point_set_1,point_set_2,point_set_3,point_set_4,name):
        stacked_image = np.concatenate((img_1,img_2),1)
        for ix in range(0,point_set_1.shape[0]):
                pt1 = tuple(np.round(point_set_1[ix]).astype(int))
                pt2 = tuple(np.round(np.array(point_set_2[ix] + [img_1.shape[1],0])).astype(int)
                cv2.line(stacked_image,pt1,pt2,(0, 255, 0), 2)
                cv2.circle(stacked_image,pt1,3,(0, 0, 255),-1)
                cv2.circle(stacked_image,pt2,3,(0, 0, 255),-1)
        for ix in range(0,point_set_3.shape[0]):
                pt1 = tuple(np.round(point_set_3[ix]).astype(int))
                pt2 = tuple(np.round(np.array(point_set_4[ix] + [img_1.shape[1],0])).astype(int)
                cv2.line(stacked_image,pt1,pt2,(255, 0, 0), 2)
                cv2.circle(stacked_image,pt1,3,(0, 0, 255),-1)
```

```
                cv2.circle(stacked_image,pt2,3,(0, 0, 255),-1)
        cv2.imwrite('Image_Correspondences_'+name+'.jpg',stacked_image)

def cost_function(H,point_set_1,point_set_2):
        Range_Points = list()
        F= list()
        for ix in range(point_set_1.shape[0]):
                Range_Points.append(point_set_2[ix,0])
                Range_Points.append(point_set_2[ix,1])

                f1 = np.array(H[0]*point_set_1[ix,0]+H[1]*point_set_1[ix,1]+H[2] / H[6]*point_set
                f2 = np.array(H[3]*point_set_1[ix,0]+H[4]*point_set_1[ix,1]+H[5] / H[6]*point_set

                F.append(f1)
                F.append(f2)

        return np.array(Range_Points) - np.array(F)

def form_line(pt1,pt2):
        if len(pt1)<3:
                pt1.append(1)
        if len(pt2)<3:
                pt2.append(1)
        t = np.cross(pt1,pt2)
        if t[2]!= 0:
                line = t/t[2]
        else:
                line = t
        return line

def form_intersection(line_1,line_2):
        t = np.cross(line_1,line_2)
        if t[2]!= 0:
                intersection = t/t[2]
        else:
                intersection = t
        return intersection

def stich_frame(panorama,Frame,H,side,previous_min_X=0):
        [height,width,channels] = Frame.shape
        Frame_corners = np.array([[0,0,1],[0,height-1,1],[width-1,height-1,1],[width-1,0,1]])
        temp = np.dot(H,Frame_corners.T)
        Frame_corners_transformed = (np.round(temp/temp[2,:]).astype(int)).T
        frame_top_edge = form_line([0,1],[10,1])
        frame_bottom_edge =form_line([0,panorama.shape[0]],[10,panorama.shape[0]])
        if side.upper() == 'RIGHTSIDE':
                panorama_edge = form_line(Frame_corners_transformed[2,:].tolist(),Frame_corners_t
                top_intersection = form_intersection(frame_top_edge,panorama_edge)

                bottom_intersection = form_intersection(frame_bottom_edge,panorama_edge)
                max_X = np.round(max(top_intersection[0],bottom_intersection[0])).astype(int)
                min_X = np.min(Frame_corners_transformed[:,0])
                expanded_panorama = np.zeros((panorama.shape[0],np.abs(max_X-panorama.shape[1]),3
                print(min_X)
                print(max_X)
                print(panorama.shape[0])
                print(np.abs(max_X-min_X))
                new_frame = np.zeros((panorama.shape[0],np.abs(max_X-min_X),3),np.uint8)
                panorama = np.concatenate((panorama,expanded_panorama),1)
                indices = np.indices((new_frame.shape[1],new_frame.shape[0]))
```

```
            print('indices = ',indices.shape)
            x_indices = indices[0,:,:].reshape(new_frame.shape[1]*new_frame.shape[0],1)+min_X
        elif side.upper() == 'LEFTSIDE':
            panorama_edge = form_line(Frame_corners_transformed[0,:].tolist(),Frame_corners_t
            top_intersection = form_intersection(frame_top_edge,panorama_edge)
            bottom_intersection = form_intersection(frame_bottom_edge,panorama_edge)
            min_X = np.round(min(top_intersection[0],bottom_intersection[0])).astype(int)
            max_X = np.max(Frame_corners_transformed[:,0])
            expanded_panorama = np.zeros((panorama.shape[0],abs(min_X-previous_min_X),3))
            new_frame = np.zeros((panorama.shape[0],abs(abs(min_X)+max_X),3))
            panorama = np.concatenate((expanded_panorama,panorama),1)
            indices = np.indices((new_frame.shape[1],new_frame.shape[0]))
            x_indices = indices[0,:,:].reshape(new_frame.shape[1]*new_frame.shape[0],1)+min_X
        else:
            print("Please specify the direction of stitching, rightside or leftside")
            return

        y_indices = indices[1,:,:].reshape(new_frame.shape[1]*new_frame.shape[0],1)
        z_indices = np.ones((new_frame.shape[1]*new_frame.shape[0],1),np.uint8)
        final_indices = np.concatenate((x_indices,y_indices,z_indices),1)
        new_indices = (np.dot(np.linalg.pinv(H).astype(float),final_indices.T)).T
        new_indices[:,0] = new_indices[:,0]/new_indices[:,2]
        new_indices[:,1] = new_indices[:,1]/new_indices[:,2]
        new_indices[:,2] = new_indices[:,2]/new_indices[:,2]
        new_indices = new_indices.astype(int)

        final_indices = final_indices[new_indices[:,0] >= 0]
        new_indices = new_indices[new_indices[:,0] >= 0]
        final_indices = final_indices[new_indices[:,1] >= 0]
        new_indices =new_indices[new_indices[:,1] >= 0]
        final_indices = final_indices[new_indices[:,0] < width]
        new_indices = new_indices[new_indices[:,0] < width]
        final_indices = final_indices[new_indices[:,1] < height]
        new_indices =new_indices[new_indices[:,1] < height]

        print('PN = ',panorama.shape)
        print('F = ',Frame.shape)
        print(new_indices.shape)
        if not side.upper()=='Rightside':
            final_indices[:,0] = final_indices[:,0]-min_X
        for ix in range(new_indices.shape[0]):
            panorama[final_indices[ix,1]][final_indices[ix,0]] = Frame[new_indices[ix,1]][ne

        return panorama,min_X

import cv2
import numpy as np
import random
import math
import Linear_Least_Square_Homography as LSM

def calculate_inliers_outliers(H,delta,point_set_1,point_set_2):
        transformed_points_1 = np.zeros(point_set_1.shape)
        for ix in range(0, point_set_1.shape[0]):
            tx = np.dot(H,np.array(np.append(point_set_1[ix,:],1)).T)
            transformed_points_1[ix] = np.round(np.array([tx[0]/tx[2],tx[1]/tx[2]]))

        distances = np.sqrt(np.sum((point_set_2 - transformed_points_1)**2,1))
        idx = distances<=delta
        inliers_1 = point_set_1[idx]
```

```
        inliers_2 = point_set_2[idx]
        idx = distances>delta
        outliers_1 = point_set_1[idx]
        outliers_2 = point_set_2[idx]
        return inliers_1,inliers_2,outliers_1,outliers_2

def Ransac(Int_points_1,Int_points_2,n,sigma,p,epsilon):
        #n = number of randomly selected correspondences for ransac
        #delta = distance threshold between transformed points and original points
        delta = 100*sigma
        num_correspondences = Int_points_1.shape[0]
        N = math.ceil(math.log(1-p)/math.log(1-(1-epsilon)**n))
        M = math.ceil((1-epsilon)*n)
        Inliers = list()
        Outliers = list()
        Optimum_inlier_len = -1
        Optimum_inliers = []
        Optimum_outliers = []
        #print("M = ",M)
        #print("N = ",N)
        for ix in range(N):
                indx = random.sample(range(num_correspondences), n)
                point_set_1 = Int_points_1[indx,:]
                point_set_2 = Int_points_2[indx,:]
                H = LSM.LLSM(point_set_1,point_set_2)
                inliers_1,inliers_2,outliers_1,outliers_2 = calculate_inliers_outliers(H,delta,p
                if len(inliers_1) > Optimum_inlier_len:
                        Optimum_inlier_len = len(inliers_1)
                        Optimum_inliers_1 = inliers_1
                        Optimum_inliers_2 = inliers_2
                        Optimum_outliers_1 = outliers_1
                        Optimum_outliers_2 = outliers_2

        #print("Optimum Inliers 1 = ",Optimum_inliers_1,'\nOptimum Inliers 2 = ',Optimum_inliers
        return Optimum_inliers_1,Optimum_inliers_2,Optimum_outliers_1,Optimum_outliers_2
import os
import utils as UL
import Linear_Least_Square_Homography as LSM
import numpy as np
import Ransac as RN
from scipy import optimize
import cv2

def Main():
        detector = "sift"
        num_Images = 5
        Image_List = list()
        H_list = np.zeros((num_Images,3,3))#list of pairwise homographies
        for kx in range(0,num_Images-1):
                img_file_0 = './HW_5_Images/'+str(kx)+'.jpg'
                img_file_1 = './HW_5_Images/'+str(kx+1)+'.jpg'

                img_0,_ = UL.image_read(img_file_0)
                img_1,_ = UL.image_read(img_file_1)


                int_points_img_0,kp_0,des_0 = UL.opencv_interest_points(img_0,detector,'img_'+str
                int_points_img_1,kp_1,des_1 = UL.opencv_interest_points(img_1,detector,'img_'+str

                correspondence_img,correspondences = UL.opencv_correspondence(int_points_img_0,k
```

```
            corresponding_points = np.zeros((len(correspondences),2))
            for ix in range(0,len(correspondences)):
                    corresponding_points[ix,0] = correspondences[ix].queryIdx
                    corresponding_points[ix,1] = correspondences[ix].trainIdx

            corresponding_kp_0 = np.zeros((len(correspondences),2))
            corresponding_kp_1 = np.zeros((len(correspondences),2))
            for ix in range(0,corresponding_points.shape[0]):
                    corresponding_kp_0[ix,:] = kp_0[int(corresponding_points[ix,0])].pt
                    corresponding_kp_1[ix,:] = kp_1[int(corresponding_points[ix,1])].pt

            H = LSM.LLSM(corresponding_kp_0,corresponding_kp_1)
            Optimum_inliers_1,Optimum_inliers_2,Optimum_outliers_1,Optimum_outliers_2 = RN.R
            UL.draw_correspondences(img_0,img_1,Optimum_inliers_1,Optimum_inliers_2,Optimum

            #Optimize least squares
            H = LSM.LLSM(Optimum_inliers_1,Optimum_inliers_2)
            H = optimize.least_squares(UL.cost_function,np.reshape(H,[1,9]).squeeze(),args=[
            H = np.reshape(H.x,[3,3]).squeeze()

            #print("H matrx "+str(kx)+'_'+str(kx+1),H)
            H_list[kx] = H
    #Add the last image in the list
    for kx in range(0,num_Images):
            img,_ = UL.image_read('./HW_5_Images/'+str(kx)+'.jpg')
            Image_List.append(img)

    stitchy=cv2.Stitcher.create()
    (dummy,output)=stitchy.stitch(Image_List)
    cv2.imshow('cv2_panaroma',output)
    cv2.waitKey(0)
    cv2.imwrite('cv2_panorama.jpg',output)
    #Transform H matrices to the anchor image

    anchor_img = 3
    H_to_mid = np.eye(3)
    for ix in range(anchor_img,len(Image_List)):
            H_to_mid = np.matmul(H_to_mid,np.linalg.pinv(H_list[ix]))
            H_list[ix] = H_to_mid
    H_to_mid = np.eye(3)
    for ix in range(anchor_img-1,-1,-1):
            H_to_mid = np.matmul(H_to_mid,H_list[ix])
            H_list[ix] = H_to_mid
    H_list = np.insert(H_list,anchor_img,np.eye(3),0)

    print(H_list)
    #Create panorama
    print(len(Image_List))
    panorama = Image_List[anchor_img-1]
    for ix in range(anchor_img,num_Images):
            panorama,_ = UL.stich_frame(panorama,Image_List[ix],H_list[ix],'Rightside')
            cv2.imwrite("Panorama_right_"+"%d.jpg"%(ix),panorama)
    previous_min_X = 0
    for ix in range(anchor_img-1,-1,-1):
            panorama,previous_min_X = UL.stich_frame(panorama,Image_List[ix],H_list[ix],'Lef
            cv2.imwrite("Panorama_left_"+"%d.jpg"%(ix),panorama)
Main()
```