ECE 661 Homework 5

Suyash Ail
sail@purdue.edu

October 6, 2020

The task of this homework is to find robust homography between images using RANSAC and Non-Linear Least Squares Algorithm. The images are joined together to form a panoramic image.

1 Theory Questions

Question1

- The point to point correspondence using the :east Squares algorithm performs well only under certain situations. The performance is highly dependent on the correct correspondence between the two interest points. But even using highly accurate feature descriptors through SIFT and SURF we still end up with false pairings.
- We need a mechanism by which we can quantify which of the correpondences are bad and which are good. RANSAC tries to address this situation via a very trivial technique.
- The algorithm decides best or worst fit using random consensus of the points. We first select the least amount of data that is needed to achieve a homography estimate. Then using this estimate we project all the interest points of the domain image onto the range image. Then based on a simple distance measure,we compute the support that all the data points provide. If the number of support is greater than a given threshold, then that estimate is considered a good estimate and all the support points are good correspondences.
- this procedure is repeated multiple times with different set of data as initial points. we keep track of the total support each iteration generates. Then the iteration with the most number of supports is the most robust estimate. All the support of this estimate are the inliers or the good correspondences and other points are bad correspondences or outliers.

Question 2

- In GD, the best path to take to the minimum is along the direction of the steepest descent and usually this takes several iterations and is usually slow. We also tend to end up at local minima or never converge.

- As per Gauss-Newton (GN), if we start at point $\vec{(p)}$ we try to find a best possible step, $\vec{(\delta_p)}$ such that we directly reach the minimum. This is a very rudientary estimate of the jump to take in order to reach the minima and often fails to do so when the point is very close to the minima.

- In LM, we try to solve the following equation,

$$(J_{\vec{f}}^T J_{\vec{f}} + \mu I)\vec{(\delta_p)} = J_{\vec{f}}^T \vec{\epsilon}(\vec{p_k})$$

  where μ is a damping factor

- When $\mu = 0$, LM basically returns same solution as GN and when μ is larger than diagonal elements of $J_{\vec{f}}^T J_{\vec{f}}$, we get a solution close to that of GD

- Thus LM tries to reach the minima by taking big steps at a time using the GN algorithm (small μ) and once it reaches closer to the minima, it computes the cost of jump, by computing the cost function. If the cost of the jump is high, the μ is doubled which steers the algorithm towards GD, which is more stable and guaranteed to converge. Hence LM combines the best features of both GD and GN to achieve optimization.

2 Logic

2.1 Linear Least Squares

- The least squares algorithm is used to compute the homography between two images having more than 4 points of correspondence.

- A Homography H transforms a point X in the domain plane to X' in the range plane i.e

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

- H can be written as:

$$H = \begin{bmatrix} a_1 1 & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

  where $a_{33} = 1$ is set to be 1 as the information is in the ratios.

- Similarly, the points X : (x,y) and X' : (x',y') in their homogeneous coordinate representations can be written as,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, X' = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix}$$

  where X is the domain of H and X' is the range of H.

- From (1), we can obtain the coordinates of X' as follows,

$$x' = a_{11}x + a_{12}y + a_{13} - a_{31}xx' - a_{32}yx'$$

$$y' = a_{21}x + a_{22}y + a_{23} - a_{31}xy' - a_{32}yy'$$

- Thus, considering we have n points of the form (x_i, y_i), we can write equations for all (x'_i, y'_i) in terms of 8 unknowns. These 8 unknowns are the elements of the required Homography matrix H i.e

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ . \\ . \\ x'_n \\ y'_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx'_n & -y_nx'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_ny'_n & -y_ny'_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{31} \end{bmatrix}$$

- In simpler terms, this can be written as follows,

$$X' = A\hat{h}$$

- Now, in order to obtain a least squares approximation of , when we have 4 or more point correspondences between image pairs, the following equation holds true,

$$\hat{h} = (A^T A)^{-1} A^T X'$$

- After estimating using LLS, we can obtain the required Homography estimate by simply appending a 1 to and reshaping it into a 3x3 matrix

2.2 RANSAC

RANSAC stands for "RANdom SAmple Consensus". The algorithm is used to detect false pairs in the correspondences.

The steps for RANSAC algorithm are as follows:

1. Set the decision threshold δ as 3.
2. The number of trials to conduct, N is given by

$$N = \frac{ln(1-p)}{ln(1-(1-\in)^n)}$$

   where p is the probability that at least for one set of points there exists a trial which is independent of outliers.n is the number of correspondences that are selected randomly out the set.
3. The minimum value for the size of the inlier set to be considered aceptable is given by

$$M = (1-\in)\eta_{total}$$

   where η_{total} is the total number of correspondences.
4. N trials are carried out by randomly sampling n pairs of correspondences in each trial and homography is determined using these points. Then the total number of inliers are calculated for the approximated homography.

5. For determining which points are outliers, we follow these steps:
   (a) HX is computed for all points X
   (b) the difference between HX and X' is computed as $(\Delta x, \Delta y)$.
   (c) Squared distances between X' and expected HX are calculated as
   $$d^2 = \Delta x^2 + \Delta y^2$$
   (d) All correspondences with $d^2 < \delta^2$ are considered inliers.
6. The largest such inlier set is used to recompute the homography matrix.
7. The homography obtained from the largest inlier set is used further refines using the Non-Linear Least Squares algorithm.

2.3 Non-Linear Least Squares

The non-Linear Least Squares is used to further refine the homography estimate. We use the Levenberg-Marquardt Algorithm in this example. The algorithm is discussed below:

The non linear optimization is written as :

$$min_p ||\vec{E}||^2$$

where $\vec{E}$ is given by

$$\begin{bmatrix} e_1(\vec{p}) \\ e_2(\vec{p}) \\ e_3(\vec{p}) \\ . \\ . \end{bmatrix}$$

1. Start with an initial guess $\vec{p_0}$ for $\vec{p_k}$.
2. Determine the Jacobian of , $J_{\vec{E}}$ at $\vec{p_0}$.
3. Determine and $J_{\vec{E}}$ where
   $$J_{\vec{E}} = \begin{bmatrix} \frac{\partial e_1}{\partial p_1} & \frac{\partial e_1}{\partial p_2} & . & . & . & \frac{\partial e_1}{\partial p_n} \\ \frac{\partial e_2}{\partial p_1} & \frac{\partial e_2}{\partial p_2} & . & . & . & \frac{\partial e_2}{\partial p_n} \\ . & . & . & . & . & . \\ \frac{\partial e_n}{\partial p_1} & \frac{\partial e_n}{\partial p_2} & . & . & . & \frac{\partial e_n}{\partial p_n} \end{bmatrix}$$
4. Compute step increment as
   $$\vec{\delta_p} = -(J^T J + \mu_k I)^{-1}) J^T \vec{E}(\vec{p_k})$$
5. compute $\vec{p_{k+1}} = \vec{p_k} + \vec{\delta_k}$
6. Compute ρ and use it to compute the μ_{k+1} for the next iteration.
7. Make $\mu_k = \mu_{k+1}$
8. Repeat from step 3 until $||\vec{\delta_p}|| < \xi$, where ξ is the stopping criterion

I have made use of the standard scipy library to carry out the NLLS function using LM optimization.

2.4 Image Mosaicing

For the five images we compute the pair wise homographies H_{ij} between i^{th} and j^{th} image. We will be computing homographies H_{12},H_{23},H_{34},H_{45}. Then we project all the images onto the plane of the image 3. For this we need to find the mapping of image1 and image5 to image3. We get them by cascading the two homographies.

$$H_{13} = H_{12}H_{23}$$

$$H_{53} = H_{54}H_{43}.$$

With H_{13},H_{23},H_{43},H_{53}, we project all the images onto the plane of image3.

3 Results

Best Parameters for SIFT:
sigma=1.4
n features = 1000
contrastThreshold=0.1
edgeThreshold=10

Best Parameters for RANSAC:
$\sigma = 1$
$\delta = 3\sigma$
n=6
p=0.999
$\in$=0.4

For LM algorithm i am using the built in scipy library with the default parameters.

3.1 Input Images

Figure 1: Image 1

Figure 2: Image 2

Figure 3: Image 3

Figure 4: Image 4

Figure 5: Image 5

3.2 Correspondences

Figure 6: Correspondences between Image 1 and Image 2

Figure 7: Correspondences between Image 2 and Image 3

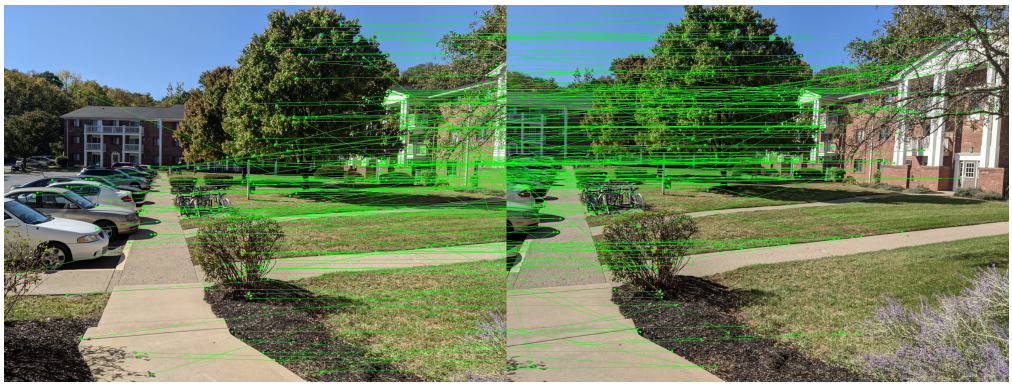

Figure 8: Correspondences between Image 3 and Image 4

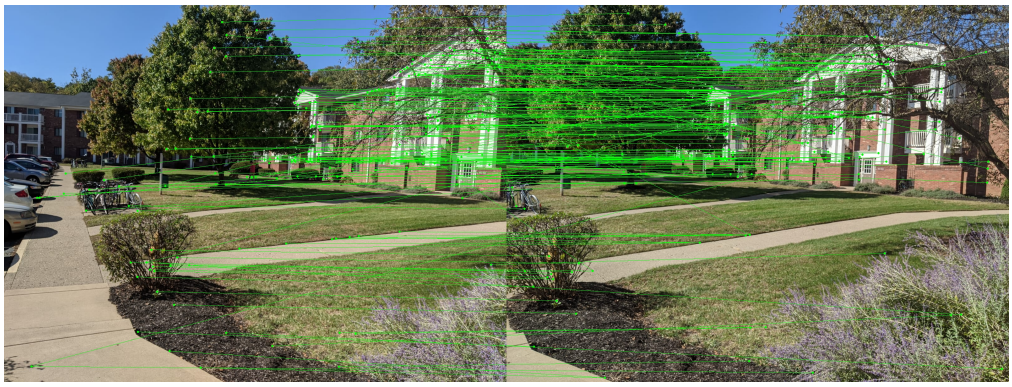

Figure 9: Correspondences between Image 4 and Image 5

3.3 Inliers and Outliers

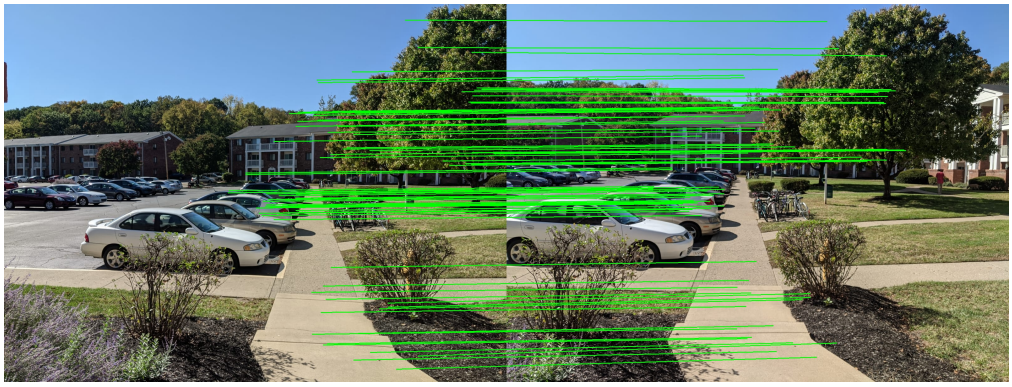

Figure 10: Inliers for Image 2 and Image3 (84)

Figure 11: Outliers for Image 2 and Image 3 (16)

Figure 12: Inliers for Image 4 and Image 5 (77)

Figure 13: Outliers for Image 4 and Image 5 (23)

3.4 Image Mosaic

Figure 14: Panorama Image

I also tried out this technique on another set of images. These photos were taken at a seashore in India. This is a very good edge case as all the images captured by the phone have different illumination due to the iPhone6's sensitivity to light. Here is the result of the reconstruction

Figure 15: Beach image

I could not include the fifth image because the algorithm couldnt find any best matches

between the 4th and 5th image and randomly computing the homography completely blowed out the results.

4 Code Listings

```
# -*- coding: utf-8 -*-
"""hw5_Suyash_Ail.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1_FY9GJzVAyYT578FBlYeewtJCvOT9Jeq
"""

import matplotlib.pyplot as plt
import numpy as np
import cv2
import math

# the newer version of cv2 does not contain SIFT and SURF. So we need to downgrade cv2
#!pip install opencv-python==3.4.2.16
#!pip install opencv-contrib-python==3.4.2.16

"""# SIFT"""

#get the keypoints and descriptors of the two images using SIFT.
def get_sift_keypoints(image1,image2):
  im1=cv2.cvtColor(image1, cv2.COLOR_RGB2GRAY)
  im2=cv2.cvtColor(image2, cv2.COLOR_RGB2GRAY)

  sift = cv2.xfeatures2d.SIFT_create(nfeatures=1000,sigma=1.4)#nOctaveLayers=5,contrastThreshold=0.1,edgeThreshold=10,sigma=1.4)
  keypoints_sift1, descriptors1 = sift.detectAndCompute(im1, None)
  keypoints_sift2, descriptors2 = sift.detectAndCompute(im2, None)

  img = cv2.drawKeypoints(image1, keypoints_sift1, None)
  plt.figure(figsize=(10,10))
  plt.subplot(1,2,1)
  plt.imshow(img)
  img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
  cv2.imwrite("kp1.JPG",img)

  img = cv2.drawKeypoints(image2, keypoints_sift2, None)
  plt.subplot(1,2,2)
  plt.imshow(img)
  img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
  cv2.imwrite("kp2.JPG",img)

  return keypoints_sift1,keypoints_sift2,descriptors1,descriptors2

#########################################################
#define ssd metric as the sum of squared differences. to keep values from blowing out we take root of the distance.
def SSD(des1, des2):
    return (np.sum((des1-des2)**2))
```

```
49
50 #find correspondence between two images using ssd
51 def Correspondence(des1, des2, kp1, kp2): #(#,128), (##,128),
52     n1=des1.shape[0]
53     n2=des2.shape[0]
54     dist=[]   # distance between every descriptor
55     points=[] # best match points
56     for i in range(n1):
57         ssd = np.array([SSD(des1[i,:], des2[j,:]) for j in range(n2)]) #send
       descriptors to ssd to compute distances
58         #print(dist)
59         j = np.argmin(ssd)
60         dist.append(np.min(ssd))
61         points.append((i,j))
62     # Select 100 pairs to eliminate outliers
63     BestPairs = [x for _,x in sorted(zip(dist,points))]  # Incresing order
64     SelectedPairs = BestPairs[0:int(np.min([100,len(BestPairs)]))]
65     int_corrs = np.zeros((len(SelectedPairs), 4))
66     kp1_xy_corrs = np.asarray([(kp1[i].pt[0],kp1[i].pt[1]) for i, _ in
       SelectedPairs])
67     kp2_xy_corrs = np.asarray([(kp2[j].pt[0],kp2[j].pt[1]) for _, j in
       SelectedPairs])
68     return np.concatenate((kp1_xy_corrs,kp2_xy_corrs),axis=1)
69
70 '''
71 im1 = cv2.imread('/content/drive/My Drive/1.jpg')
72 im2 = cv2.imread('/content/drive/My Drive/2.jpg')
73 im3 = cv2.imread('/content/drive/My Drive/3.jpg')
74 im4 = cv2.imread('/content/drive/My Drive/4.jpg')
75 im5 = cv2.imread('/content/drive/My Drive/5.jpg')
76
77 '''
78 '''
79 im1 = cv2.imread('/content/drive/My Drive/hw5/5.jpeg')
80 im2 = cv2.imread('/content/drive/My Drive/hw5/4.jpeg')
81 im3 = cv2.imread('/content/drive/My Drive/hw5/3.jpeg')
82 im4 = cv2.imread('/content/drive/My Drive/hw5/2.jpeg')
83 im5 = cv2.imread('/content/drive/My Drive/hw5/1.jpeg')
84 '''
85 im1 = cv2.imread('/content/drive/My Drive/hw5_images (1)/1 (1).jpeg')
86 im2 = cv2.imread('/content/drive/My Drive/hw5_images (1)/2 (1).jpeg')
87 im3 = cv2.imread('/content/drive/My Drive/hw5_images (1)/3 (1).jpeg')
88 im4 = cv2.imread('/content/drive/My Drive/hw5_images (1)/4 (1).jpeg')
89 im5 = cv2.imread('/content/drive/My Drive/hw5_images (1)/5 (1).jpeg')
90
91
92
93
94 im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
95 im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)
96 im3 = cv2.cvtColor(im3, cv2.COLOR_BGR2RGB)
97 im4 = cv2.cvtColor(im4, cv2.COLOR_BGR2RGB)
98 im5 = cv2.cvtColor(im5, cv2.COLOR_BGR2RGB)
99
100 kp1,kp2,des1,des2=get_sift_keypoints(im1,im2)
101 cor1=Correspondence(des1,des2,kp1,kp2)
102 print(len(cor1))
103
```

```
print(cor1[0])

def DrawPairs(input_image1, input_image2, Corres):
    left =input_image1
    right = input_image2
    output = np.concatenate((left, right), axis = 1) # (800, 1200, 3)
    for idx in range(Corres.shape[0]):
        w1 = int(Corres[idx,0])   # x-coordinate of image 1
        h1 = int(Corres[idx,1])   # y coordinate of image 1
        w2 = int(Corres[idx,2] + left.shape[1]) # x coordinate of image 2 (add image1 width to get points in image 2 plane)
        h2 = int(Corres[idx,3])   #y coordinate of image 2
        cv2.line(output, (w1,h1),(w2,h2), (0,0,255), 2)  # draw corresponding lines
        plt.imshow(output.astype(int))
    output = cv2.cvtColor(output,cv2.COLOR_RGB2BGR)
    cv2.imwrite("pair_image.jpg", output)

DrawPairs(im1,im2,cor1[:50,:])

"""# LLS"""

def LLS(p1,p2):
    H = np.zeros((3,3)) #initialize H matrix
    #p1=corrs[:,0:2]  #get the points of image1 from the correspondance matrix
    #p2=corrs[:,2:]  #get the points of image2 from the correspondence matrix


    # we will use svd to find the elements of H matrix. A matrix will be a (2n x 9) matrix which will be decomposed
    A = np.zeros((len(p1)*2,9))
    #get the elements of the A matrix (homework 2)
    for i in range(len(p1)):
        A[i*2]=[0, 0, 0, -p1[i,0], -p1[i,1], -1, p2[i,1]*p1[i,0], p2[i,1]*p1[i,1], p2[i,1]]
        A[i*2+1]=[p1[i,0], p1[i,1], 1, 0, 0, 0, -p2[i,0]*p1[i,0], -p2[i,0]*p1[i,1],-p2[i,0]]

    # Find SVD decomposition of A matrix
    U,D,V = np.linalg.svd(A)      #V is actually V.T
    V_T = np.transpose(V) #Need to take transpose because rows of V are eigen vectors
    H_elements = V_T[:,-1] #Last column is the solution as we need to minimize the least squares

    #Fill the Homography Matrix
    H[0] = H_elements[0:3] / H_elements[-1]
    H[1] = H_elements[3:6] / H_elements[-1]
    H[2] = H_elements[6:9] / H_elements[-1]
    return H

H1=LLS(cor1[:,:2],cor1[:,2:])
print(H1)

"""# Inliers count done"""

#project the points from one image to the plane of second image and find out the error in the projection.
```

```
#use the abs() function to compute error. If error < threshold then points considered inlier points
def inlier_outlier_count(H,corrs, delta):
  #get the x and y coordinates of corresponding points
  #x1=corrs[:,0]
  #y1=corrs[:,1]
  #x2=corrs[:,2]
  #y2=corrs[:,3]

  p1=corrs[:,:2]
  p2=corrs[:,2:]
  p1HC = np.concatenate((p1,np.ones((p1.shape[0],1)), np.float)), axis=1)
  pred_p2 = np.matmul(H,p1HC.T).T #predicted points in the dest image
  pred_p2 = pred_p2 // pred_p2[:,2].reshape(-1,1)
  pred_error = (pred_p2[:,:2] - p2) ** 2
  pred_error = np.sqrt(np.sum(pred_error, axis=1))
  InList  = np.where(pred_error < delta)
  count = InList[0].shape[0]
  return count, InList

i1,i2=inlier_outlier_count(H1,cor1,1.5)
print(i1)

"""# RANSAC DONE"""

def RANSAC(corrs,delta,n,p,epsilon):
  #Seperate source and destination images XY coordinates

    p1 = corrs[:,0:2]
    p2 = corrs[:,2:]

    #Number of trials for determining homography
    N = int(math.log(1 - p)/math.log(1-(1-epsilon)**n))

    #Minimum value of inliner set considered acceptable, Not delta
    M = int(len(p1) * (1-epsilon))

    #Initialize Homography Matrix
    sol_list = []

    #Loop over the total number of trials
    for i in range(N):
        #Randomly select n number of correspondences
        rand_corr_idx = np.random.randint(0,len(p1),n)
        rand_src_xy = p1[rand_corr_idx,:]
        rand_dest_xy = p2[rand_corr_idx,:]

        #Calculate Homography by SVD for n selected correspondences
        H_trial = LLS(rand_src_xy,rand_dest_xy)

        #Count the number of Inliners
        inlier_count,_ = inlier_outlier_count(H_trial,corrs,delta)

        if inlier_count > M:
            sol_list.append([H_trial,inlier_count])

    # Get a list of all possible homographies which satisfy threshold criterion
```

```
    n_soln = np.asarray(sol_list)
    #if len(n_soln)==0:
      #best_H=LLS(p1,p2)
    #else:
    best_inlier_count = np.argmax(n_soln[:,-1])
    #Homograhy with the maximum inlier support
    best_H = n_soln[best_inlier_count,0]
    return best_H

H1_ran=RANSAC(cor1,3,6,0.999,0.4)

#non linear least squares
#(Written by Manu Ramesh and me together)
from scipy.optimize import least_squares
from scipy.optimize import minimize

def objective(H,pts1,pts2):
  H = H.reshape(3,3)
  pts1HC = np.concatenate((pts1,np.ones((pts1.shape[0],1)), np.float)), axis=1)
  pred_pts2 = np.matmul(H,pts1HC.T).T #predicted points in the dest image
  pred_pts2 = pred_pts2 // pred_pts2[:,2].reshape(-1,1)

  #calculate the squared error between predicted and actual
  pred_error = (pred_pts2[:,:2] - pts2) ** 2

  pred_error = np.sqrt(np.sum(pred_error, axis=1))

  return pred_error

nlls_H_solution = least_squares(objective, np.squeeze(H1_ran.reshape(-1,1)),
    method='lm',args=(cor1[:,:2], cor1[:,2:]))

nlls_H = nlls_H_solution.x.reshape(3,3)
print(nlls_H)

def getH(img1,img2):
  #function combining all function to get the homography estimate using Least
    Squares, RANSAC and Non Linear Least Squares

  #get keypoints and descriptors between two images
  kp1, kp2, des1, des2 = get_sift_keypoints(img1,img2)

  #get correspondence between the sift keypoints
  corrs = Correspondence(des1, des2,kp1, kp2)
  pts1 = corrs[:,:2]
  pts2 = corrs[:,2:]

  #get Homography using RANSAC
  ransac_H = RANSAC(corrs,3,6,0.999,0.4)

  #Refine Homography using NLLS---Doesnt change anything actually!!
  nlls_H_solution = least_squares(objective, np.squeeze(ransac_H.reshape(-1,1)
    ), method='lm',args=(pts1, pts2))
  nlls_H = nlls_H_solution.x.reshape(3,3)

  return nlls_H

H = getH(im4,im5)
```

```
265
266 def new_image_dims(inImg, H):
267   #get the shape of the input image in the mapped coordinates.
268   #get the image coordinates in world plane
269   width=inImg.shape[0]-1
270   height=inImg.shape[1]-1
271   print("width(x)=",width)
272   print("height(y)=",height)
273   img_coords=np.array([[0,0,1],[0,height,1],[width,0,1],[width,height,1]])
274   new_img_coords=np.zeros([4,2])
275   for i in range(4):
276     new_coords=np.matmul(H,img_coords[i])
277     new_coords=new_coords/new_coords[2]
278     new_coords=np.rint(new_coords).astype(int)
279     new_img_coords[i,:]=new_coords[0:2]
280
281   #print(new_img_coords)
282   x_max = np.max(new_img_coords[:,0])
283   y_max = np.max(new_img_coords[:,1])
284   x_min = np.min(new_img_coords[:,0])
285   y_min = np.min(new_img_coords[:,1])
286
287   return [x_max, y_max, x_min, y_min]
288
289 def getImage(inImage,H,outImg,lims):
290   x_max,y_max,x_min,y_min = lims
291   offsetX=x_min
292   offsetY=y_min
293   image=inImage
294   world_image=outImg
295   scaleFactor=1
296   H_inv_=np.linalg.inv(H)
297   for i in range(0,world_image.shape[1]-1): #X-cooridnate, col
298         for j in range(0,world_image.shape[0]-1): #Y-coordinate, row
299             k1 = i/scaleFactor + offsetX
300             k2 = j/scaleFactor + offsetY
301             X_domain = [k1,k2]
302             X_domain = np.array(X_domain)
303             X_domain = np.append(X_domain,1)
304             X_range = np.matmul(H_inv_, X_domain)
305             X_range = X_range/X_range[-1]
306             X_range = np.rint(X_range)
307             X_range = X_range.astype(int)
308             if(X_range[0] > 0 and X_range[1] > 0 and X_range[0] < image.shape
    [1] and X_range[1] < image.shape[0]):
309                 world_image[j][i] = image[X_range[1]][X_range[0]]
310
311   return world_image
312
313
314
315
316
317 H12 = getH(im1,im2)
318 H23 = getH(im2,im3)
319 H3 = np.eye(3)
320 H34 = getH(im3,im4)
321 H45 = getH(im4,im5)
```

```
322
323 H1 = np.matmul(H12,H23)    #H13
324 H1 = H1/H1[-1,-1]
325
326 H2 = H23/H23[-1,-1]       #H23
327
328 H43 = np.linalg.inv(H34)
329 H4 = H43/H43[-1,-1]
330
331 H35 = np.matmul(H34,H45)
332 H53 = np.linalg.inv(H35)
333 H5 = H53/H53[-1,-1]
334
335 '''
336 H1=getH(im1,im3)
337 H2=getH(im2,im3)
338 H3=np.eye(3)
339 H4=getH(im4,im3)
340 H5=getH(im5,im3)
341 '''
342
343 img1_lims = np.array(new_image_dims(im1, H1)).reshape(1,-1)
344 img2_lims = np.array(new_image_dims(im2, H2)).reshape(1,-1)
345 img3_lims = np.array(new_image_dims(im3, H3)).reshape(1,-1)
346 img4_lims = np.array(new_image_dims(im4, H4)).reshape(1,-1)
347 img5_lims = np.array(new_image_dims(im5, H5)).reshape(1,-1)
348
349 lims = np.concatenate((img1_lims,img2_lims, img3_lims, img4_lims, img5_lims))
350 x_max = np.max(lims[:,0]); y_max = np.max(lims[:,1])
351 x_min = np.min(lims[:,2]); y_min = np.min(lims[:,3])
352 blankImg = np.zeros((np.int(np.ceil(y_max-y_min)), np.int(np.ceil(x_max-x_min)
    ),3), np.uint8)
353 print(blankImg.shape)
354 '''
355 imgOut1 = get_warp_image(im1,H1,blankImg,[x_max, y_max, x_min, y_min])
356 imgOut1 = get_warp_image(im2,H2,imgOut1,[x_max, y_max, x_min, y_min])
357 imgOut1 = get_warp_image(im3,H3,imgOut1,[x_max, y_max, x_min, y_min])
358 imgOut1 = get_warp_image(im4,H4,imgOut1,[x_max, y_max, x_min, y_min])
359 imgOut1 = get_warp_image(im5,H5,imgOut1,[x_max, y_max, x_min, y_min])
360 '''
361 imgOut1 = getImage(im1,H1,blankImg,[x_max, y_max, x_min, y_min])
362 print("2nd")
363 imgOut1 = getImage(im2,H2,imgOut1,[x_max, y_max, x_min, y_min])
364 print("3rd")
365 imgOut1 = getImage(im3,H3,imgOut1,[x_max, y_max, x_min, y_min])
366 print("4th")
367 imgOut1 = getImage(im4,H4,imgOut1,[x_max, y_max, x_min, y_min])
368 print("5th")
369 imgOut1 = getImage(im5,H5,imgOut1,[x_max, y_max, x_min, y_min])
370
371
372 #imgOut1, LUT = warpImage4(im1,H1,blankImg,[x_max, y_max, x_min, y_min])
373 #imgOut1, LUT = warpImage4(im2,H2,imgOut1,[x_max, y_max, x_min, y_min])
374 #imgOut1, LUT = warpImage4(im3,H3,imgOut1,[x_max, y_max, x_min, y_min])
375 #imgOut1, LUT = warpImage4(im4,H4,imgOut1,[x_max, y_max, x_min, y_min])
376 #imgOut1, LUT = warpImage4(im5,H5,imgOut1,[x_max, y_max, x_min, y_min])
377 plt.imshow(imgOut1.astype(int))
378
```

```
379 plt.figure(figsize=(10,10))
380 #plt.imshow(cv2.cvtColor(imgOut1,cv2.COLOR_BGR2RGB))
381 plt.imshow(imgOut1)
382
383 plt.imsave("launch_apts_pana.jpg",imgOut1)
```