CV_Assignment2.md

Name : Anvesh Chaturvedi

Roll Number : 20161094

Computer Vision Assignment 2

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from random import choice
from scipy import ndimage
import scipy
from PIL import Image
import random


THRESH_RANSAC = 0.85
NUM_ITERS = 1000
```

$$\begin{bmatrix} wx_i' \\ wy_i' \\ w \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{aligned} x_i' &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

CSE 576, Spring 2

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\underset{2n \times 9}{A} \qquad \underset{9}{h} \qquad \underset{2n}{0}$$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - 0\|^2$

- Since h is only defined up to scale, solve for unit vector $\hat{h}$
- Solution: $\hat{h}$ = eigenvector of A^TA with smallest eigenvalue
- Works with 4 or more points

calculateHomography : Computes the homography matrix using the smallest eigenvector after SVD of the matrix as defin above

```
def calculateHomography(correspondences):
    ''' Computes the homography matrix based on the correspondance points given as input'''

    num = correspondences.shape[0]
    M = np.zeros((2*num, 9))
    for i in range(num):
        corr = correspondences[i]
        p1 = np.matrix([corr.item(0), corr.item(1), 1])
        p2 = np.matrix([corr.item(2), corr.item(3), 1])
        M[2*i,0:3] = p1
        M[2*i,6:9] = - p2.item(0) * p1
        M[2*i + 1, 3:6] = p1
        M[2*i + 1, 6:9] = - p2.item(1) * p1

    U, D, V = np.linalg.svd(M)

    h = np.reshape(V[8], (3, 3))
    h = h / h[2,2]
    return h
```

RANSAC

- RANSAC is a resampling technique that generates candidate solutions by using the minimum number ob- servations (data points) required to estimate the underlying model parameters. As pointed out by Fischlerand Bolles, unlike conventional sampling techniques that use as much of the data as possible to obtain an initial solution and then proce to prune outliers, RANSAC uses the smallest set possible and proceeds to enlarge this set with consistent data points.
- The number of iterations, N , is chosen high enough to ensure that the probability p (usually set to 0.99) that at least o of the sets of random samples does not include an outlier. Let u represent the probability that any selected data point i an inlier and v = 1u the probability of observing an outlier. N iterations of the minimum number of points denoted m are required

Procedure for Computing Homography

- Select four feature pairs (at random)
- Compute homography H (exact)

- Compute inliers where ||pi', H pi|| < ε
- Keep largest set of inliers.
- Re-compute least-squares H estimate using all of the inliers.

get_num_inliers: This functions finds the number of inliers corresponing to the current homography matrix and returns it tc be further used in RANSAC.

```
def get_num_inliers(correspondances, h,  threshold):
    ''' Returns the number of inliers for currently computed homography'''
    cnt=0
    for corr in correspondances:
        point_1 = np.matrix([corr[0].item(0), corr[0].item(1), 1]).T
        point_2 = np.matrix([corr[0].item(2), corr[0].item(3), 1]).T
        p2_estimate = np.dot(h, point_1)
        p2_estimate = p2_estimate / p2_estimate.item(2)
        if np.linalg.norm(point_2 - p2_estimate) < threshold:
            cnt+=1
    return cnt
```

ransac : Perform RANSAC on the generated correspondances matrix to find a robust solution for the homography matrix

```
def ransac(corr, thresh, thresh_dist=5):
    ''' Performs RANSAC to compute a homography matrix. '''
    maxInliers = -1
    finalH = None
    num = len(corr)
    print("num: ", num)
    for i in range(NUM_ITERS):
        rand_indx = random.sample(range(num), 4)
        for j in range(4):
            if j == 0:
                random_points = corr[rand_indx[j]]
            else:
                random_points = np.vstack((random_points, corr[rand_indx[j]]))

        h = calculateHomography(random_points)

        num_inliers = get_num_inliers(corr, h, thresh_dist)

        if num_inliers > maxInliers:
            maxInliers = num_inliers
            finalH = h

        if maxInliers > (num*thresh):
            break
    return finalH
```

crop_image : This function takes input as a rgb image and returns a trimmed image with black boundaries around the ima removed.

```
def crop_image(image, threshold=0):
    ''' Crops the black boundaries around the stitched images. '''

    flatImage = np.max(image, 2)

    rows = np.where(np.max(flatImage, 0) > threshold)[0]
    if rows.size:
        cols = np.where(np.max(flatImage, 1) > threshold)[0]
        image = image[cols[0]: cols[-1] + 1, rows[0]: rows[-1] + 1]
    else:
        image = image[:1, :1]
```

```
    return image
```

stitch : This function takes input as the 2 rgb images which needs to be stitched together in the output. It generates the `sift features` for both the images and then matches the generated features using a matcher to generate a correspondance list (list of lists with each element as `[x1 y1 x2 y2]` where `(x1, y1)` and `(x2, y2)` denote the correspondance points).
It then passes on this list to the `RANSAC` function which returns a homography matrix as the output which is used to warp th second image.
Finally, the first image and the warped second image are stitched together to get the desired output.

```
def stitch(im1_rgb, im2_rgb):
    ''' Main function for stitching 2 images.
        Gets sift features for the images and constructs a list of correspondance points.
        Generates the homography matrix using RANSAC according to the correspondance points.
        Stitches the first image with warped second image.
        Crops the black boundary around stitched image.
    '''
    if len(im1_rgb.shape) != 3:
        im1_rgb = cv2.cvtColor(im1_rgb,cv2.COLOR_GRAY2RGB)
    im1 = cv2.cvtColor(im1_rgb, cv2.COLOR_RGB2GRAY)
    im1 = cv2.copyMakeBorder(im1,200,200,500,500, cv2.BORDER_CONSTANT)
    im1_rgb = cv2.copyMakeBorder(im1_rgb,200,200,500,500, cv2.BORDER_CONSTANT)

    if len(im2_rgb.shape) != 3:
        im2_rgb = cv2.cvtColor(im2_rgb,cv2.COLOR_GRAY2RGB)
    im2 = cv2.cvtColor(im2_rgb, cv2.COLOR_RGB2GRAY)

    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(im1, None)
    kp2, des2 = sift.detectAndCompute(im2, None)

    matcher = cv2.BFMatcher(cv2.NORM_L2, True)
    matches = matcher.match(des1, des2)

    correspondenceList = []

    for m in matches:
        (x1, y1) = kp1[m.queryIdx].pt
        (x2, y2) = kp2[m.trainIdx].pt
        correspondenceList.append([x1, y1, x2, y2])

    corrs = np.matrix(correspondenceList)
    out_ransac = ransac(corrs, THRESH_RANSAC)
    print("Homography Matrix : ")
    print(out_ransac)
    final_image = None

    output = np.zeros_like(im1_rgb)
    out = cv2.warpPerspective(im2_rgb, scipy.linalg.inv(out_ransac), (im1.shape[1],  im1.shape[0]))

    (x, y) = im1.shape
    for i in range(x):
        for j in range(y):
            if im1[i][j]==0 and np.sum(out[i][j])==0:
                output[i][j]=[0,0,0]
            elif im1[i][j]==0:
                output[i][j] = out[i][j]
            else:
                output[i][j] = (im1_rgb[i][j])

    final_image = np.copy(output)
    final_image =  np.uint8(final_image)
    final_image = crop_image(final_image)
```

```
    return final_image
```

stitch_images : This function gets a list of images and uses the function `stitch()` to stitch them together to finally return combined output.

```
def stitch_images(paths_list):
    '''Gets a list of images and returns a stitched output.'''
    num = len(paths_list)
    for i in range(num):
        im_input = np.array(Image.open(paths_list[i]))
        if i is 0:
            im = im_input
            continue
        im = stitch(im, im_input)
    return im
```

display_images : Function to define how the final output should be printed

```
def display_images(path_list, output):
    ''' Displays output as desired. '''
    num, i = len(path_list), 0
    while(num > 0):
        if num == 1:
            plt.figure(figsize=(8,8))
            img = Image.open(path_list[i])
            plt.title("Image " + str(i+1))
            plt.imshow(img)
            i+=1; plt.xticks([]); plt.yticks([]);
        else:
            plt.figure(figsize=(16,16))
            plt.subplot(1,2,1)
            plt.title("Image " + str(i+1))
            plt.imshow(Image.open(path_list[i]))
            i+=1; plt.xticks([]); plt.yticks([]);
            plt.subplot(1,2,2)
            plt.title("Image " + str(i+1))
            plt.imshow(Image.open(path_list[i]))
            i+=1; plt.xticks([]); plt.yticks([]);
        num-=2
    plt.figure(figsize = (16,16))
    plt.title("Final Output")
    plt.imshow(output)
    plt.xticks([]); plt.yticks([]);
    plt.show()
```

Results On Test Images

```
path_list = ['./test_images/img2_1.png', './test_images/img2_2.png', './test_images/img2_3.png',
          './test_images/img2_4.png', './test_images/img2_5.png', './test_images/img2_6.png']
output = stitch_images(path_list)
```

```
num:  1123
Homography Matrix
[[ 1.00000101e+00  8.39764615e-07 -6.83001083e+02]
 [-2.19350631e-08  1.00000133e+00 -2.10000347e+02]
 [-7.96418027e-13  2.03887609e-09  1.00000000e+00]]
num:  842
```

```
Homography Matrix
[[ 1.00000501e+00 -2.26536111e-07 -9.75004843e+02]
 [ 2.93826785e-06  9.99999838e-01 -1.98002150e+02]
 [ 1.04285012e-08 -1.64850316e-08  1.00000000e+00]]
num:  716
Homography Matrix
[[ 1.00136238e+00 -5.47595447e-04 -4.95529903e+02]
 [ 4.18770369e-04  1.00048780e+00 -3.39421434e+02]
 [ 2.82272478e-06 -2.53993259e-06  1.00000000e+00]]
num:  1489
Homography Matrix
[[ 9.99635286e-01 -1.06231494e-04 -6.00734461e+02]
 [ 1.88874693e-06  9.99588282e-01 -4.20826896e+02]
 [ 5.57915516e-09 -8.03388974e-07  1.00000000e+00]]
num:  1385
Homography Matrix
[[ 9.99991321e-01 -6.50954843e-06 -9.25988310e+02]
 [-1.52763717e-07  9.99992841e-01 -4.44996603e+02]
 [-3.47913256e-09 -5.29654248e-09  1.00000000e+00]]


display_images(path_list, output)
```

Image 1

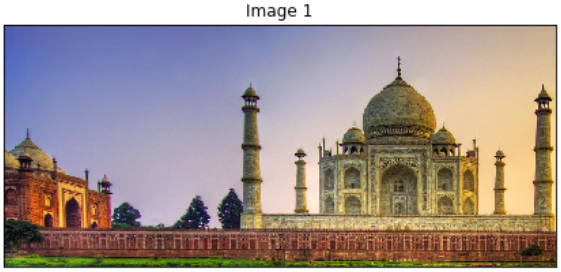


Image 2

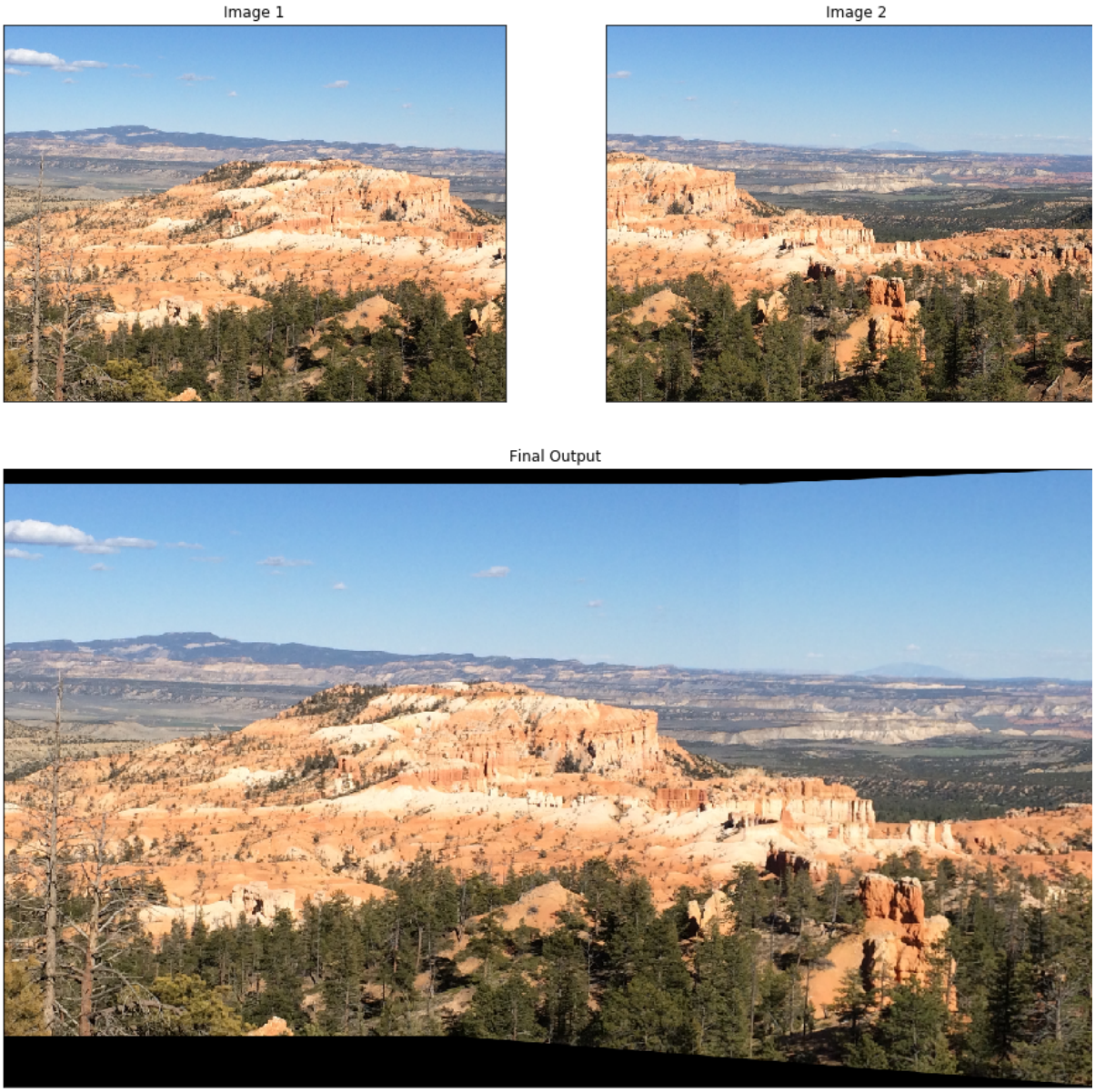
Image 3

Image 4


```
path_list = ['./test_images/img1_1.png', './test_images/img1_2.png']
output = stitch_images(path_list)
```

```
num:  5379
Homography Matrix
[[ 1.29428164e+00  1.70623197e-02 -1.47597770e+03]
 [ 5.50207438e-02  1.24721033e+00 -3.34029281e+02]
 [ 2.03165432e-04 -1.17780624e-05  1.00000000e+00]]
```

```
display_images(path_list, output)
```




```
path_list = ['./test_images/img3_1.png', './test_images/img3_2.png']
output = stitch_images(path_list)
```

```
num:  1300
Homography Matrix
[[ 2.06174203e+01 -3.77137188e-01 -2.17673655e+04]
 [ 4.98123215e+00  1.74314396e+01 -1.01521623e+04]
 [ 1.32128272e-02 -2.65379884e-04  1.00000000e+00]]
```

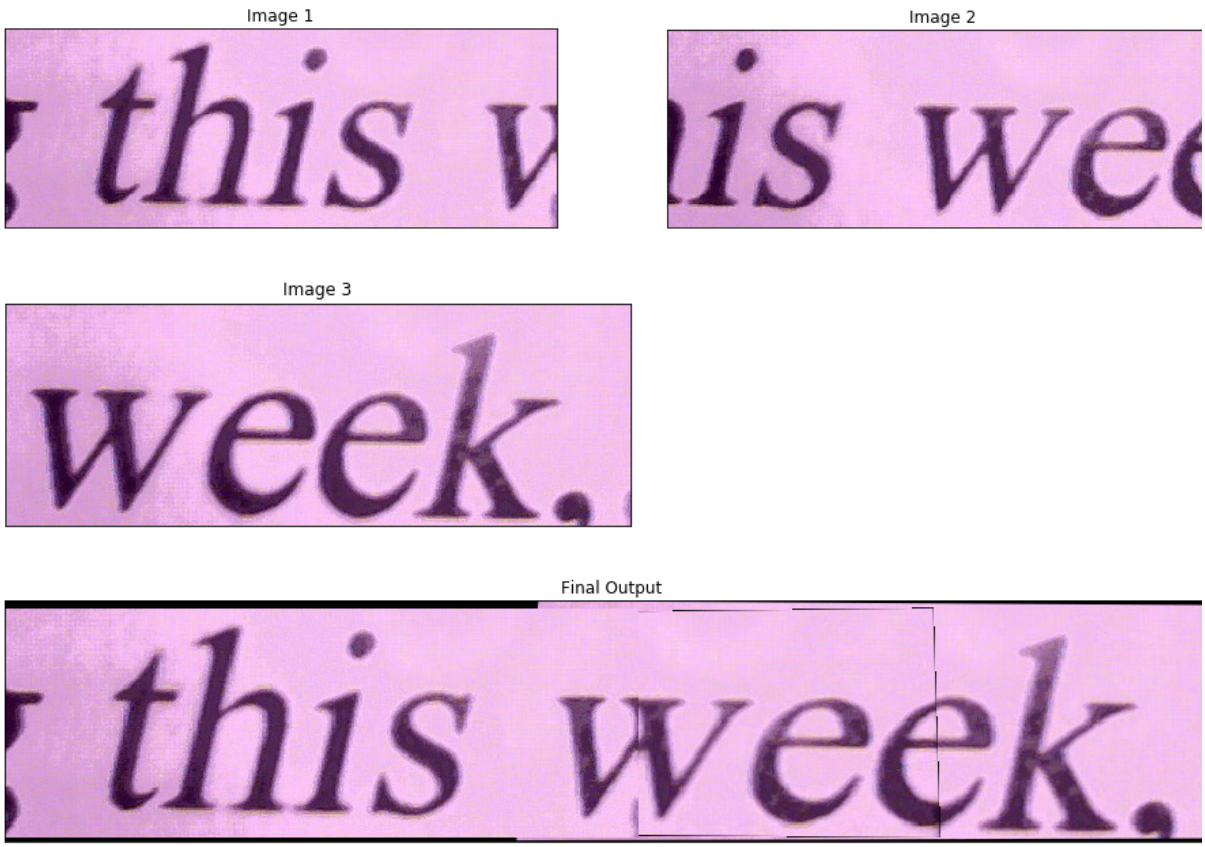
```
display_images(path_list, output)
```


```
path_list = ['./test_images/img4_1.jpg', './test_images/img4_2.jpg', './test_images/img4_3.jpg']
output = stitch_images(path_list)
```

```
num:  53
Homography Matrix
[[ 1.16055087e+00  2.21008682e-02 -8.04634582e+02]
 [ 1.42129128e-02  1.21884903e+00 -2.59950630e+02]
 [ 1.57834329e-04  1.25778802e-04  1.00000000e+00]]
num:  51
Homography Matrix
[[ 9.27075136e-01  8.82272502e-02 -8.30608739e+02]
```

```
 [-6.31604577e-03  9.94256992e-01 -1.88817829e+02]
 [-5.81099208e-05  2.88548319e-04  1.00000000e+00]]


display_images(path_list, output)
```



Results On My Images

```
path_list = ['./test_images/my_images/1_11.jpg', './test_images/my_images/1_12.jpeg']
output = stitch_images(path_list)


num:  1120
Homography Matrix :
[[ 9.07042383e-01  2.84480858e-02 -2.96935401e+02]
 [-4.66491807e-02  9.46365791e-01 -1.47261888e+02]
 [-6.21267129e-05  3.65253661e-06  1.00000000e+00]]


display_images(path_list, output)
```

Image 1

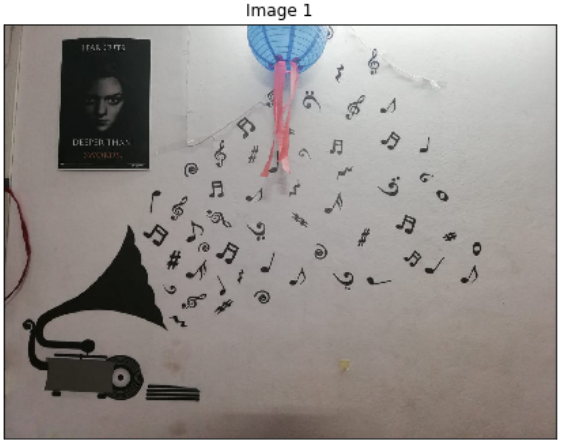
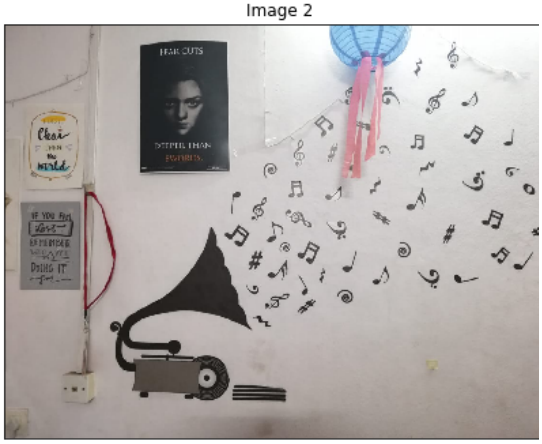
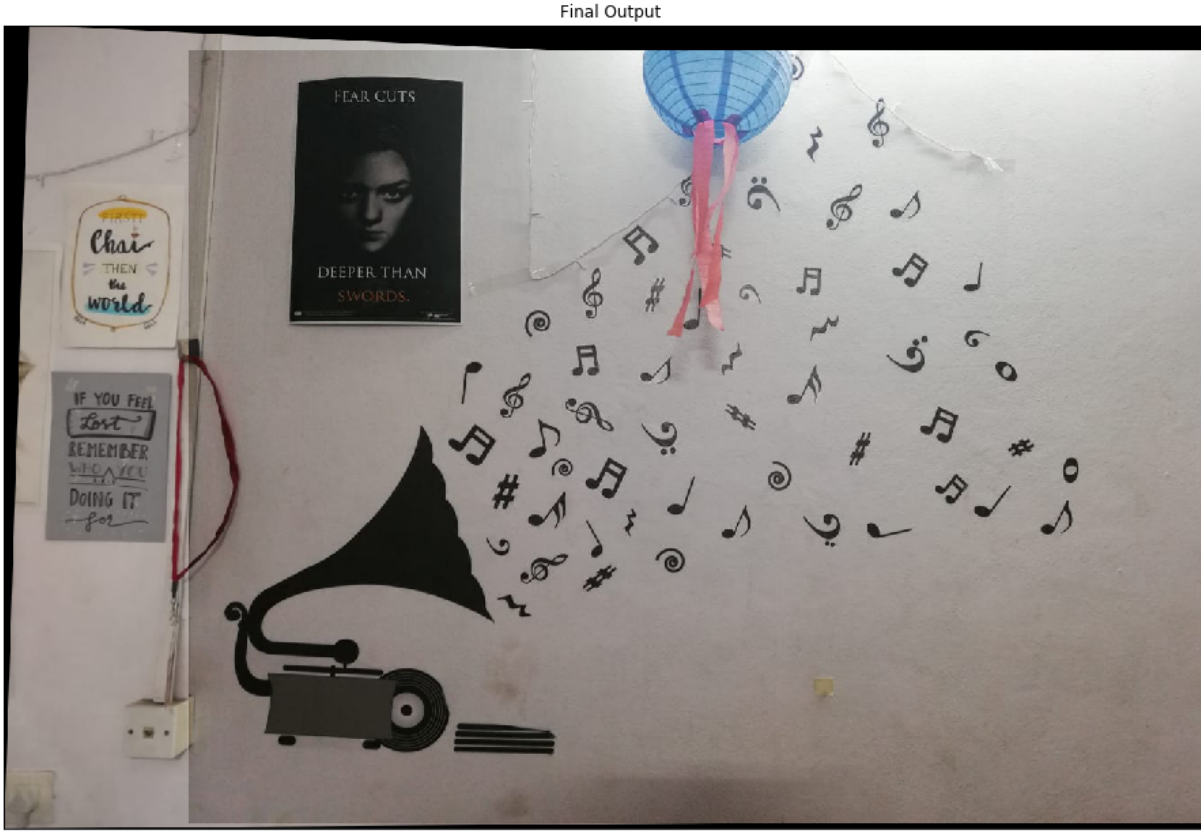


Image 2



Final Output




```
path_list = ['./test_images/my_images/2_1.jpeg', './test_images/my_images/2_2.jpeg']
output = stitch_images(path_list)
```

```
num:  1399
Homography Matrix :
[[ 1.95619872e+00 -1.13997136e-01 -1.64837733e+03]
 [ 3.46645475e-01  1.67865234e+00 -7.38309647e+02]
 [ 5.87576329e-04 -2.46915779e-05  1.00000000e+00]]
```

```
display_images(path_list, output)
```

Image 1

Image 2

Final Output