CV ASSIGNMENT-2

R. Navanith, S20160010071

QUESTION 1:-

Stitiching Pairs of Images :-

STEP 1:-

Finding SIFT Keypoints and Descriptors :

```
im1 = cv2.imread('uttower_left.JPG')
im2 = cv2.imread('uttower_right.JPG')
gray1= cv2.cvtColor(im1,cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(im2,cv2.COLOR_BGR2GRAY)
print(im1.shape)
sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(gray1,None)
kp2, des2 = sift.detectAndCompute(gray2,None)
```

The two images ‘uttower_left.JPG’ and ‘uttower_right.JPG’ are the images to stitch together. The images are converted to grayscale images stored in the variables gray1, gray2 respectively. This piece of code finds the keypoints and descriptors of two images. Stored in the variables kp1, des1, kp2, des2 respectively.

STEP 2:-

Selecting the putative matched points:-

```
def detect_putative_points(des1, des2, putative_points):

    descdistance = scipy.spatial.distance.cdist(des1,des2,'sqeuclidean')
    print(descdistance.shape)
    ind = np.unravel_index(np.argsort(descdistance.ravel())[:300], descdistance.shape)
    ind = np.asarray(ind)
    print(ind.shape)
    print(ind[0])
    x1 = []
    y1 = []
    x2 = []
    y2 = []
    for i in range(ind.shape[1]):
        x1,y1 = kp1[ind[0][i]].pt
        x2,y2 = kp2[ind[1][i]].pt
        putative_points.append([x1,y1,x2,y2])
```

The above code finds the putative matched points. At first 'descdistance' stores the distance between the descriptors (128 dimensional). Among those distances the first minimum 300 distances key points are stored in 'ind'. The 300 best putative matched points are stored in 'putative_points'.

STEP 3 :-

Getting the Homography matrix using RANSAC Method :

```
def ransac_and_homography(matched_points, thresh,kp1,kp2):

    maxInliers = []
    finalH = []
    for i in range(1000):

        rand_point1 = matched_points[random.randrange(0, len(matched_points))]
        rand_point2 = matched_points[random.randrange(0, len(matched_points))]
        rand_Four = np.vstack((rand_point1, rand_point2))
        rand_point3 = matched_points[random.randrange(0, len(matched_points))]
        rand_Four = np.vstack((rand_Four, rand_point3))
        rand_point4 = matched_points[random.randrange(0, len(matched_points))]
        rand_Four = np.vstack((rand_Four, rand_point4))

        A = []
        for point in rand_Four:
            x1 = np.matrix([point.item(0), point.item(1), 1])
            x2 = np.matrix([point.item(2), point.item(3), 1])

            a1 = [-x1.item(0), -x1.item(1), -x1.item(2), 0, 0, 0, x2.item(0) * x1.item(0), x2.item(0) * x1.item(1), x2.item(2)]
            a2 = [0, 0, 0, -x1.item(0), -x1.item(1), -x1.item(2), x2.item(1) * x1.item(0), x2.item(1) * x1.item(1), x2.item(1)]

            A.append(a1)
            A.append(a2)
        A = np.matrix(A)
        u, s, v = np.linalg.svd(A)
        h = np.reshape(v[8], (3, 3))
        h = (1/h.item(8)) * h
```

```
        inliers = []
        for i in range(len(matched_points)):
            p1 = np.transpose(np.matrix([matched_points[i][0], matched_points[i][1], 1]))
            estimatep2 = np.dot(h, p1)
            estimatep2 = (1/estimatep2.item(2))*estimatep2

            p2 = np.transpose(np.matrix([matched_points[i][2], matched_points[i][3], 1]))
            error = p2 - estimatep2
            #d = geometricDistance(matched_points[i], h)
            if np.linalg.norm(error) < 3:
                inliers.append(matched_points[i])

        if len(inliers) > len(maxInliers):
            maxInliers = inliers
            finalH = h
        #print("Corr size: ", len(matched_points), " NumInliers: ", len(inliers), "Max inliers: ", len(maxInliers))

        if len(maxInliers) > (len(matched_points)*thresh):
            break
    return finalH, maxInliers
```

From the 300 putative matched points The best 4 matched points are need to get the homography matrix. This piece of code iterated through 1000 times getting different 4 random matched points and then find the homography for these random four points. And then find the inliers for every random matched points. The 4 matched points having maximum number of inliers are taken for to calculate the homgraphy matrix. So this function gives the best Homography matrix. The below is the homography matrix.

```
[[ 7.25391740e+02  1.56289111e+02 -4.45633610e+05]
 [ 2.09993075e+02  7.42692243e+02 -2.59088684e+05]
 [ 4.58599381e-01  4.75299232e-01  1.00000000e+00]]
```

STEP 4 :-

Warping one image onto the other :-

```
dst = cv2.warpPerspective(im2,H,(im2.shape[1] + im1.shape[1], im1.shape[0]))
plt.imshow(dst)
dst[0:im1.shape[0], 0:im1.shape[1]] = im2
cv2.imshow("output.jpg",dst)
```

This code warps one image into the other using the homography matrix.

After Warping the final image is