EN2550 – Assignment 02

Name : Rathnayaka R.G.H.V.

Index No : 180529E

01.various types of 2-D transformations

a. 2D Rotation

Image rotates according to the angle which inputs to t. Then H matrix can be taken generally as below.

H = [[np.cos(t), np.sin(t), 0.], [-np.sin(t), np.cos(t), 0.], [0., 0., 1.]]

The plot for 2D rotation for θ = π/3 where t = np.pi/3 is given in *Figure (01)* beside.

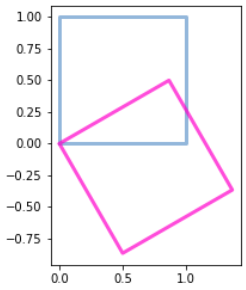

Figure (01) – 2D Rotation

b. 2D Translation

Image can be translated in both x and y directions generally as below.

H = [[1., 0., a.], [0., 1., b.], [0., 0., 1.]]

Here 'a' depicts the shift of the image in x direction and 'b' depicts the shift of the image in y direction. The plot for a=2 and b=2 is given in *Figure (02)* beside.

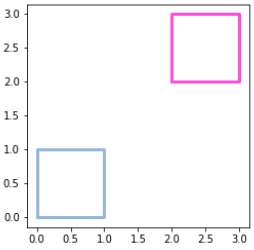

Figure (02) – 2D Translation

c. 2D Scaling

The general H matrix can be taken as below where 's' depicts the scale of the image in x direction and 't' depicts the scale of the image in y direction.

H = [[s., 0., 0.], [0., t., 0.], [0., 0., 1.]]

The plot for s=2 and t=0.5 is given below in the *Figure (03)* beside.

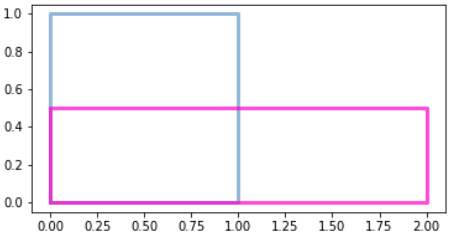

Figure (03) – 2D Scaling

d. 2D Reflection

Image can be reflected about both x axis and y axis by taking the H matrix as below.

H = [[-1., 0., 0.], [0., 1., 0.],[0., 0., 1.]] # 2D Reflection about Y-axis

H = [[1., 0., 0.], [0., -1., 0.],[0., 0., 1.]] # 2D Reflection about X-axis

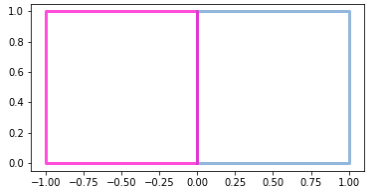

Figure (04) - 2D Reflection about Y-axis

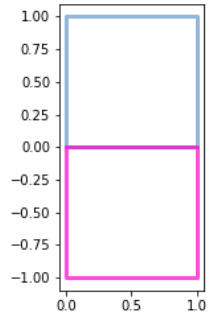

Figure (05) - 2D Reflection about X-axis

e. 2D Shear

Two-dimensional shearing can be done both horizontally and vertically. H matrix is depicted as below.

H = [[1., 0., 0.], [2., 1., 0.],[0., 0., 1.]] # 2D Vertical Shear

H = [[1., 2., 0.], [0., 1., 0.],[0., 0., 1.]] # 2D Horizontal Shear

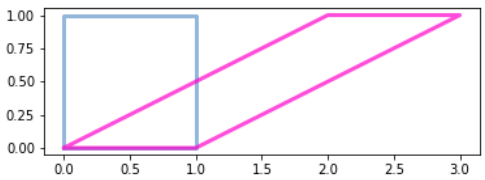

Figure (06) – 2D Horizontal Shear

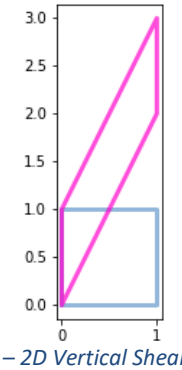

Figure (07) – 2D Vertical Shear

The code segment shown below depicts the changes that had to be done in item no.1 to do above 2D transformations.

```
t = np.pi/3
#H = [[np.cos(t), np.sin(t), 0.], [-np.sin(t), np.cos(t), 0.], [0., 0., 1.]] #(a) 2D Rotation
#H = [[1., 0., 2.], [0., 1., 2.], [0., 0., 1.]] #(b) 2D Translation
#H = [[2., 0., 0.], [0., 0.5, 0.], [0., 0., 1.]] #(c) 2D Scaling
#H = [[-1., 0., 0.], [0., 1., 0.],[0., 0., 1.]] #(d) 2D Reflection about Y-axis
#H = [[1., 0., 0.], [0., -1., 0.],[0., 0., 1.]] #(d) 2D Reflection about X-axis
#H = [[1., 0., 0.], [2., 1., 0.],[0., 0., 1.]] #(e) 2D Vertical Shearing
H = [[1., 2., 0.], [0., 1., 0.],[0., 0., 1.]] #(e) 2D Horizontal Shearing
print(H)
```

Figure (08) – All 2D Transformations

02.Warping an image using a given Homography

Here we transform Graffiti img1.ppm to img5.ppm using the code given under item no.2.

```
import cv2 as cv
import numpy as np

im1 = cv.imread('Desktop/graf/img1.ppm', cv.IMREAD_ANYCOLOR)
im5 = cv.imread('Desktop/graf/img5.ppm', cv.IMREAD_ANYCOLOR)

with open('Desktop/graf/H1to5p') as f:
    H = [[float(x) for x in line.split()] for line in f]
print(H)
H = np.array(H)

im5_warped = cv.warpPerspective(im5, np.linalg.inv(H), (1000,1000))
im5_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
```

Figure (09) – Warping image using a given Homography

H matrix for this homography and the image 1, image 5 and warped image 5 is given below in the *Figure (10)* below.

```
[[0.62544644, 0.057759174, 222.01217], [0.22240536, 1.1652147, -25.605611], [0.00049212545, -3.6542424e-05, 1.0]]
```

(a) Image 1

(b) Image 5

(c) Image 5-Warped

Figure (10) – H matrix, Image 1, Image 5 and stitched Image

03. Computing the Homogrpahy Using Mouse-Clicked Points and Warping

Homography is computed using the relevant OpenCV Stitching can be carried out. Some changes were done to the code in item no.3 for mouse clicking and selecting matching points in the two images to be stitched. Modifications which were done to the code in item no.3 is given in the *Figure (11)* below.

```
H, mask = cv.findHomography(p1,p2)
print(H)

im4_warped = cv.warpPerspective(im4,np.linalg.inv(H),(1000,1000))
im4_warped[0:im1.shape[0],0:im1.shape[1]] = im1

cv.namedWindow("Image 4-Warped",cv.WINDOW_AUTOSIZE)
cv.imshow("Image 4-Warped",im4_warped)
cv.waitKey(0)
cv.destroyAllWindows()
```

```
[[188. 138.]
 [334. 192.]
 [516. 348.]
 [682.  53.]
 [544. 544.]]
[[173. 236.]
 [282. 252.]
 [450. 337.]
 [358.  82.]
 [575. 484.]]
[[ 6.47262160e-01  6.48627243e-01 -2.66035072e+01]
 [-1.43663393e-01  9.14838133e-01  1.52866351e+02]
 [ 3.78634536e-04 -4.48359322e-05  1.00000000e+00]]
```

Figure (11) – Computing Homography using Mouse Clicked Points and Warping

The image 1, image 4 and the stitched image are clearly depicted in *Figure (12), Figure (13)* and *Figure (14)* respectively.

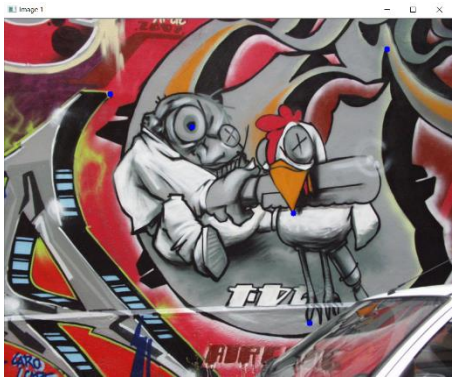

Figure (12) – image 1

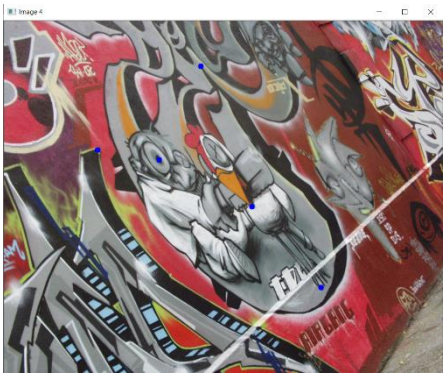

Figure (13) – image 4

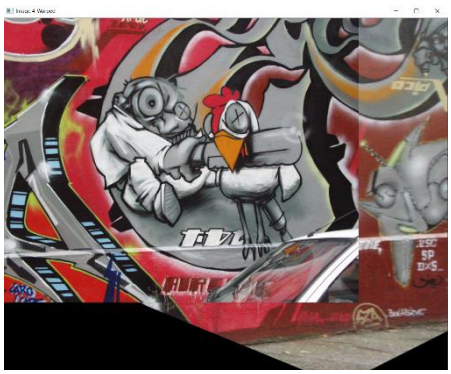

Figure (14) – Stitched image

04. Computing the Homogrpahy Using Mouse-Clicked Points without OpenCV

Homography is computed without using the OpenCV functions and Stitching can be carried out. Some changes were done to the code in item no.4 for mouse clicking and selecting matching points in the two images to be stitched. Code with these modifications is given in the *Figure (15)* below.

```
A = np.empty((2*N, 9))
for i in range(N):
    A[2*i,:]=[-p1[i][0],-p1[i][1],-1, 0, 0, 0, p1[i][0]*p2[i][0], p1[i][1]*p2[i][0], p2[i][0]]
    A[2*i+1,:]=[0, 0, 0, -p1[i][0], -p1[i][1], -1,  p1[i][0]*p2[i][1], p1[i][1]*p2[i][1], p2[i][1]]
u,s,v=np.linalg.svd(A)
L=v[-1,:]/v[-1,-1]
Homography=L.reshape(3,3)
im5_warped = cv.warpPerspective(im5, np.linalg.inv(Homography), (1000,1000))
im5_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
cv.namedWindow("Image 5 Warped",cv.WINDOW_AUTOSIZE)
cv.imshow("Image 5 Warped", im5_warped)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure (15) – Computing Homography using Mouse Clicked Points and Warping

The image 1, image 4 and the stitched image are clearly depicted in *Figure (12), Figure (13)* and *Figure (14)* respectively.

Figure (16) – image 1

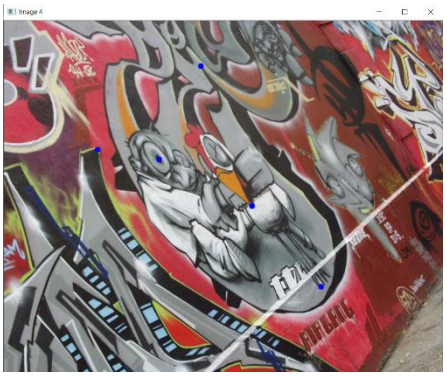

Figure (17) – image 5

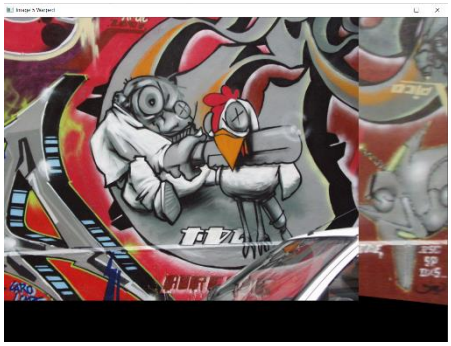

Figure (18) – Stitched image

05. (a).Stitching more than 2 images using Mouse Clicked Points

Homography is computed using the relevant OpenCV Stitching can be carried out. Some changes were done to the code in item no.3 for mouse clicking and selecting matching points in the two images to be stitched. Modifications which were done to the code in item no.3 is given in the *Figure (19)* below.

```
#Homography matrix
H4, status14 = cv.findHomography(p1,p2,cv.RANSAC)
H5, status15 = cv.findHomography(p1,p3,cv.RANSAC)
im4_warped = cv.warpPerspective(im4,np.linalg.inv(H4),(1000,1000))
im5_warped = cv.warpPerspective(im5,np.linalg.inv(H5),(1000,1000))
im4_warped[0:im1.shape[0],0:im1.shape[1]] = im1
cv.namedWindow("Image 4-Warped",cv.WINDOW_AUTOSIZE)
cv.imshow("Image 4-Warped",im4_warped)
cv.waitKey(0)
cv.namedWindow("Image 5-Warped",cv.WINDOW_AUTOSIZE)
cv.imshow("Image 5-Warped",im5_warped)
cv.waitKey(0)
im4_warped[0:im5_warped.shape[0],0:200] = im5_warped[:,0:200]
cv.namedWindow("Image Final-Warped",cv.WINDOW_AUTOSIZE)
cv.imshow("Image Final-Warped",im4_warped)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure (19) – Stitching 3 images using Mouse Clicked Points

The image 1, image 4, image 5 and the final stitched image are clearly depicted in *Figure (20), Figure (21), Figure (22) and Figure (23)* respectively.

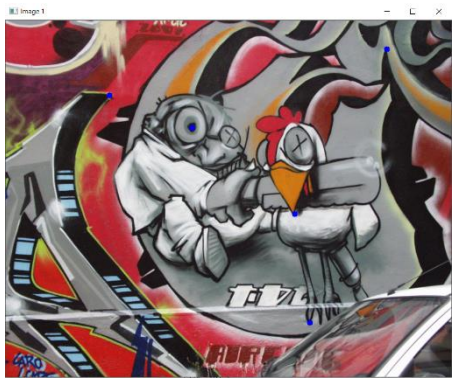

Figure (20) – image 1

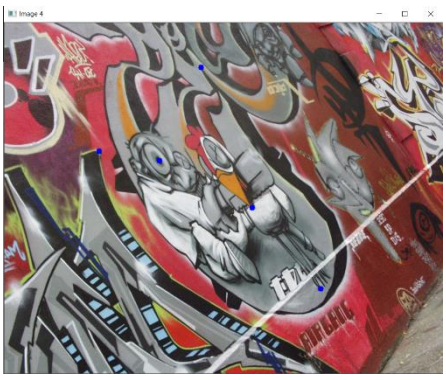

Figure (21) – image 4

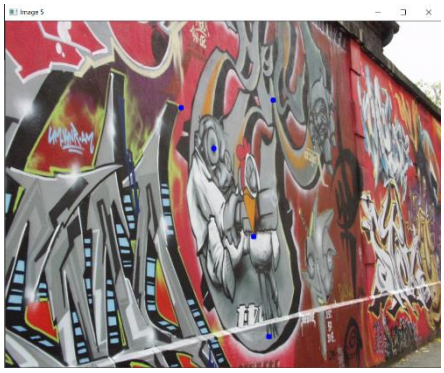

Figure (22) – image 5

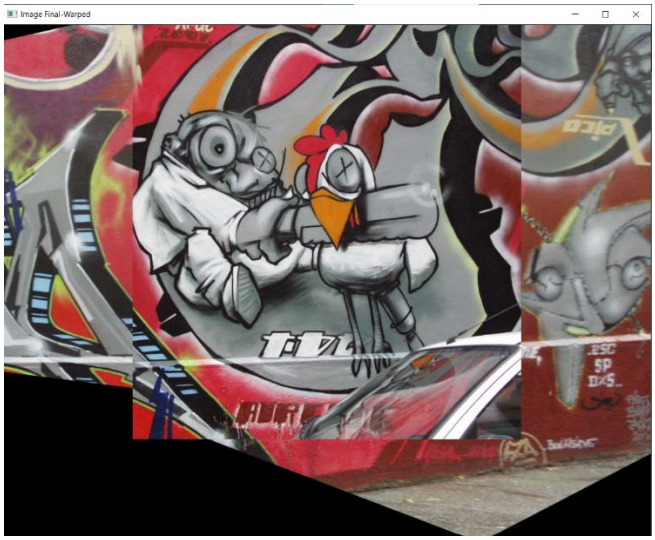

Figure (23) – Final image

05. (b).Stitching images using SIFT Features and RANSAC

SIFT features can be matches as in the *Figure (24)* beside.

Code for the transformation is also given in *Figure (25)* while the output stiched image is given in *Figure (26).*

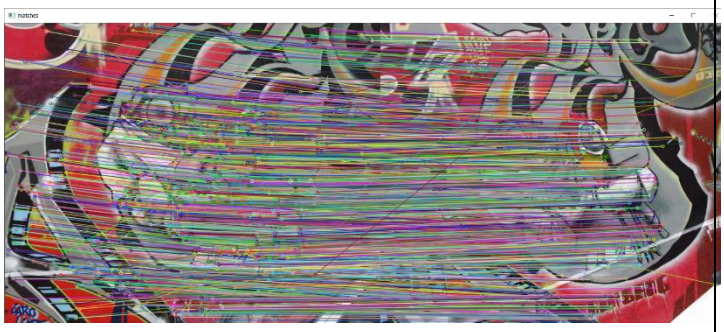

Figure (24) – Matching SIFT Features

```
im1 = cv.imread('Desktop/graf/img1.ppm', cv.IMREAD_ANYCOLOR)
img1 = cv.cvtColor(im1,cv.COLOR_BGR2GRAY)
im2 = cv.imread('Desktop/graf/img2.ppm', cv.IMREAD_ANYCOLOR)
img2 = cv.cvtColor(im2,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
#Find the keypoints and descriptors with SIFT
kp1,des1 = sift.detectAndCompute(img1,None)
kp2,des2 = sift.detectAndCompute(img2,None)
# BFMatcher with default params
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)
# Apply ratio test
good = []
for a,b in matches:
    if a.distance < 0.75*b.distance:
        good.append([a])
matches = np.asarray(good)
#Draw  good matches
imMatch = cv.drawMatchesKnn(im1,kp1,im2,kp2,matches,None,flags=2)
cv.imwrite("matches.jpg",imMatch)
cv.namedWindow("matches",cv.WINDOW_AUTOSIZE)
cv.imshow("matches",imMatch)
cv.waitKey(0)
#Find the locations of good matches
src = np.float32([ kp1[m.queryIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)
dst = np.float32([ kp2[m.trainIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)
#Find homography
H, masked = cv.findHomography(src, dst, cv.RANSAC, 5.0)
print(H)
im2_warped = cv.warpPerspective(im2, np.linalg.inv(H), (1000,1000))
im2_warped[0:im1.shape[0], 0:im1.shape[1]] = im1
cv.namedWindow("Output Image",cv.WINDOW_AUTOSIZE)
cv.imshow("Output Image",im2_warped)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure (25) – Stitching images using SIFT features and RANSAC

Figure (26) – Stitched Image obtained using SIFT Features using im1.ppm and im2.ppm