## EN2550 — Fundamentals of Image Processing and Machine Vision

**Student Name:** Nuwan Bandara
**Student Number:** 180066F

**Submitted Date:** March 15, 2021
**Assignment Number:** 2

**Problem 1** Using the given code (item no. 1), experiment with various types of 2-D transformations.
Image alignment is executed based on features and search sets of features match that agreed in terms of either local appearance or geometric configuration. Here, several 2D transformations are experimented using the given python implementation with different transformation matrices.

Entire code flow is executed and accessed on the Google Colaboratory platform (Colab) via (but, since Colab does not facilitate the OpenCV window sessions, the codes are experimented in local Jupyter environment and then, pasted in Colab),
`https://colab.research.google.com/drive/1KxXmIZafbYoZzdW2Sy_iTmoo-EvR60NN?usp=sharing`

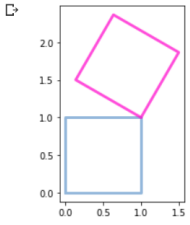
$$H_{Euclidean} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}, H_{Similarity} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$H_{SimplifiedAffine} = \begin{bmatrix} 1 & \tan\theta & t_x \\ \tan\theta & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

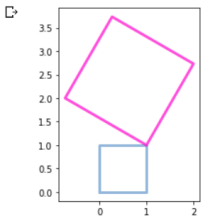

(a) Euclidean transformation matrix


(b) Similarity transformation matrix

Figure 1: Python implementations of 2D transformation matrices




(a) After Euclidean transformation

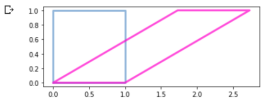



(b) After Similarity transformation

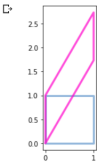
Figure 2: Results after 2D transformations

**Problem 2** Transform Graffiti (https://www.robots.ox.ac.uk/ vgg/data/affine/) img1.ppm onto img5.ppm using code in item no. 2.

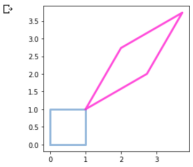
In order to apply perspective transformation, a 3x3 transformation matrix is needed. This transformation matrix could be calculated using OpenCV inbuilt *getPerspectiveTransform* by referring four points on the input image and corresponding points on the output image. Here,



(a) After Shear transformation in x-direction



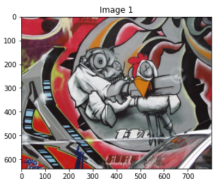
(b) After Shear transformation in y-direction



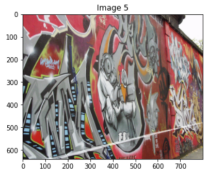
(c) After Simplified Affine transformation

Figure 3: Results after 2D transformations (Problem 1)

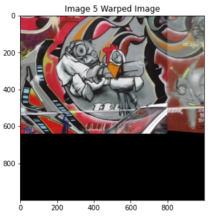
the transformation matrix is given in the *H1to5p* file and the requirement is to apply this matrix using *warpPerspective* function which gets input image, output image (im4), transformation matrix (H), size of the output image ((1000x1000)), combination of interpolation methods, pixel extrapolation method and boarder value as input parameters. Afterwards, the warped image is stitched with the input im1 image as per given in the code. Here, the variants necessarily includes coordinates of quadrangle vertices in the source image and coordinates of the corresponding quadrangle vertices in the destination image.




(a) Input Image - img1.ppm




(b) Image 5 - img5.ppm




(c) Image 5 warped image

Figure 4: Before and after transformation

**Problem 3** Item no. 3 is for mouse clicking and selecting matching points in the two images to be stitched. Compute the homography using the relevant OpenCV function and carry out stitching.

In the the projective transformation, the planar homography relates the transformation between two planes (up to a scale factor) such that,

$$H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3}$$

This homography matrix is contained with eight degrees of freedom (thus, requires four corresponding points) and in general, normalized with,

$$h_{33} = 1 \;\; or \;\; h_{11}^2 + h_{12}^2 + h_{13}^2 + h_{21}^2 + h_{22}^2 + h_{23}^2 + h_{31}^2 + h_{32}^2 + h_{33}^2 = 1 \tag{4}$$

The crucial invariant in homography is the cross ratio of four co-linear points (ratio of ratios of lengths) and if $h_{33} = 0$ and having n-points, such that,

$$\begin{bmatrix} 0^T & x_1^T & ... & -y_1^1 x_1^T \\ x_1^T & 0^T & ... & -x_1^1 x_1^T \\ ... & & & \\ ... & & & \\ 0^T & x_n^T & ... & -y_n^1 x_n^T \\ x_n^T & 0^T & ... & -x_n^1 x_n^T \end{bmatrix}_{2n*9} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}_{9*1} = \begin{bmatrix} 0 \\ 0 \\ ... \\ ... \\ 0 \end{bmatrix}_{2n*1} \tag{5}$$

In the form of $Ah = 0$. Hence, the solution would be the unit eigen vector of $A_T A$ corresponding to the smallest eigen value.

The inbuilt OpenCV function *findHomography* which gets *object Points Planar* and *image points* as input parameters could easily be used to estimate or compute the homography as implemented below.

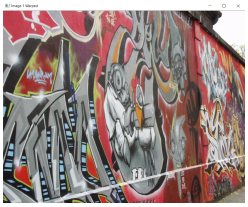
```
H, _ = cv2.findHomography(p1, p2)
print(H)
im1_warp = cv2.warpPerspective(im1, H, (im1.shape[1], im1.shape[0]))
im1_warp[0:im1.shape[0],0:im1.shape[1]] = im5
```

Figure 5: OpenCV inbuilt function implementation to compute homography

Afterwards, as in problem (3), the *warpPerspective* is utilized in order to apply the obtained transformation into stitching as per required.

(a) Input Image - img1.ppm

(b) Image 5 - img5.ppm



(c) Image 1 warped image

Figure 6: Before and after stitching

The obtained warped image suggests that the perspective transformation by computing homography (using the OpenCV inbuilt function) is working perfectly in this problem since it seems almost similar with the original image 5 with no significant distortion.

**Problem 4** Item no. 4 is similar to item no. 3. Here, compute the homography using your own code and stitch the two images.

In order to compute the homography using a custom function, the equation (5) is utilized as the reference.

In this code, first, the required matrices are defined as per zero arrays with the corresponding shapes as referenced. Afterwards, the specific values for the each element in matrix A has been assigned using the input parameters to the custom function which are the simply the corresponding mouse-clicked points.

In the assigning process of values to the corresponding elements in matrix A, an estimation has been induced to the even and odd rows in certain columns as assigning values of 0 and 1 as assumed. Afterwards, the singular value decomposition is utilized via *numpy linalg.svd* in order to obtain the necessary values for the transformation matrix H.

Afterwards, the obtained H matrix is reshaped into the shape of (3x3) since the function *warpPerspective* needs the transformation matrix to be in that shape. The rest of the code is similar with the previously applied implementation in problem (3).

```
def findH(p1, p2):
    H = np.array([0 for n in range(9)], dtype = 'float64')
    A = np.zeros([10, 9])
    row = 0
    for p_i in range(5):
        A[row*2] = np.array([p1[p_i][0], p1[p_i][1], 1,
                             0, 0, 0,
                             -1*p1[p_i][0]*p2[p_i][0], -1*p1[p_i][1]*p2[p_i][0], -1*p2[p_i][0]], dtype = int)
        A[row*2+ 1] = np.array([0, 0, 0,
                                p1[p_i][0], p1[p_i][1], 1,
                                -1*p1[p_i][0]*p2[p_i][1], -1*p1[p_i][1]*p2[p_i][1], -1*p2[p_i][1]], dtype = int)
        row += 1
    [U, D, V] = np.linalg.svd(A)
    H += V[-1]/V[-1][-1]
    print(H.shape)
    H.reshape(3, 3)
    return H
```

(a) Custom function for homography computation

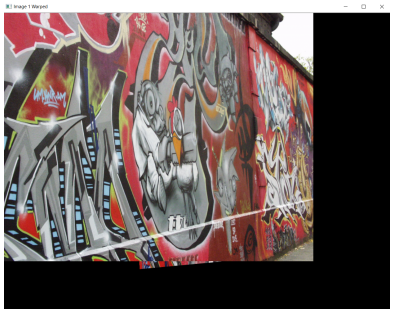
```
H = findH(p1, p2)
im1_warped = cv2.warpPerspective(im1, H.reshape(3, 3), (1000,1000))
im1_warped[0:im1.shape[0],0:im1.shape[1]] = im5
cv2.namedWindow("Image 5", cv2.WINDOW_AUTOSIZE)
cv2.imshow("Image 5", im5)
cv2.waitKey(0)
cv2.namedWindow("Image 1 Warped", cv2.WINDOW_AUTOSIZE)
cv2.imshow("Image 1 Warped", im1_warped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(b) Custom function implementation on the images

Figure 7: Python implementation of a custom function for homography computation

(a) Input Image - img1.ppm

(b) Image 5 - img5.ppm



(c) Image 1 warped image

Figure 8: Before and after stitching using a custom function

Although the above estimations are valid when computing the homoraphy, the resultant warped image suggests that there are significant discontinuities due to various reasons which may include those assumptions. Howsoever, the resultant image is at an acceptable level in

which a satisfactory warping is stitched. The black outlines are due to the destination image dimensions specified in the inbuilt *warpPerspective* function.

**Problem 5** Stitch images using SuperGlue features instead of mouse-clicked points. Compute the homography though RANSAC or MSAC. Stitch more than two images using mouse-clicked points. Handle the seams. See the original paper here.

The RANSAC algorithm could be implemented using OpenCV inbuilt *findHomography* using the input argument of *cv2.RANSAC* and *ransacReprojThreshold*.

```
H, _ = cv2.findHomography(p1, p2, cv2.RANSAC, ransacReprojThreshold=2.0)
print(H)
im1_warp = cv2.warpPerspective(im1, H, (im1.shape[1], im1.shape[0]))
im1_warp[0:im1.shape[0],0:im1.shape[1]] = im5
```

Figure 9: RANSAC algorithmic implementation of homography

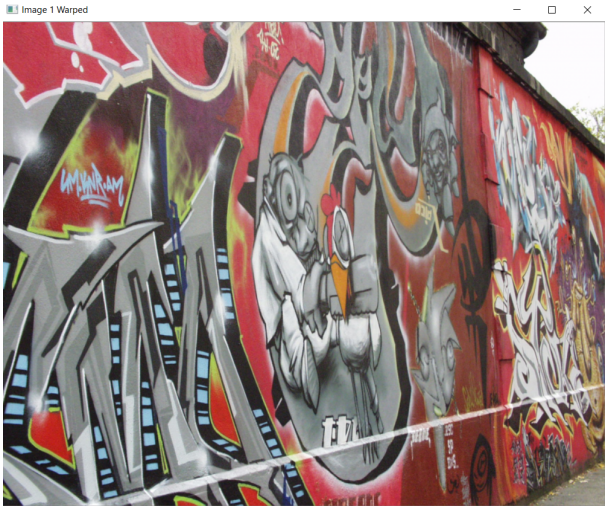



Figure 10: Result image after warped using RANSAC for the stitching in Problem (3)