Assignment A02

A. Thieshanthan, 180641N

March 16, 2021

Abstract

Assignment a02 covers 2D tranformations, computing homography, using homography to transform images and stitching images to make panorama.This report includes the results, interpretation of the work given in assignment a02. Important code pieces and image results are provided. The link for entire code can be found at appendix section.

1 2D Transformations

Making changes to an image with a set of rules is called transformations. Commonly applied 2D transformations are translation, rotation, scaling, reflection and shear. Each type of translation can be done sequentially to achieve a certain end results. To shorten this process cartesian point coordinates should be converted to homogenous coordinates. This conversion can be done with any cartesian coordinate $P(x, y)$ converted to homogenous coordinate $P\prime(x, y, h)$. In the code provided in item 1 of assignment description four points are provided in the homogenous coordinates. The transformation H is also provided. Figure 1 represents various kinds of transformation achieved manipulating the homography matrix given in the code.

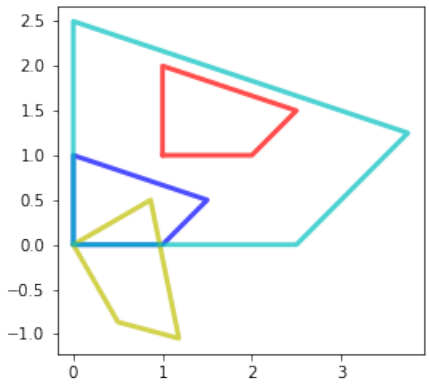


Figure 1: 2D transformations. Blue - original, Yellow - Rotated, Red - translated, Cyan - scaled

2 Warping Graffii Images

Here the homography matrix is provided for each image in the dataset. Therefore transforming the image is relatively easy with using opencv's cv2.warpPerspective() method. This method requires the image to be transformed, homography matrix and output image size as parameters. The source images downloaded from `https://www.robots.ox.ac.uk/~vgg/data/affine/` are provided in figure 2 and the results are provided in figure 3.

(a) Image 1

(b) Image 5

Figure 2: Images retrived from `https://www.robots.ox.ac.uk/~vgg/data/affine/`. Image size = (640,800)

Figure 3: Figure 2b is transformed and stitched with figure 2a.

To warp the image and stitch a basic algorithm was implemented. warp_image() function take image to be transformed, homography and shape of the output image. Cartesian coordinates are converted homogenous coordinate and then multiplied with the homography. For mapping points opencv's cv2.remap() method used for map the coordinates of the image using bilinear interpolation.

```
def warp_image(img, H, shape = (2000,2000,3)):
        m,n,ch = shape #canvas size
        indy, indx = np.indices((m, n), dtype=np.float32)
        lin_homg_ind = np.array([indx.ravel(), indy.ravel(),
                                 np.ones_like(indx).ravel()])
        #warping
        map_ind = np.matmul(H, lin_homg_ind)
        x, y = map_ind[:-1]/map_ind[-1]
        x = x.reshape(m,n).astype(np.float32)
        y = y.reshape(m,n).astype(np.float32)
        warped_img = cv.remap(img, x, y, cv.INTER_LINEAR)

        return warped_img
```

stitch() function takes the images to be stitched and pad as parameters. Pad is added to

ensure complete transformed images are stitched. The following code uses the above functions.

```
def stitch(img1, img2, pad = (200, 200)):
        '''Stitch img1 onto img2
        img2 - reference image
        '''
        m,n,c = img2.shape
        m += pad[0]
        n += pad[1]
        canvas = np.zeros((m, n, c))
        canvas[0:img2.shape[0], 0:img2.shape[1]] = img2
        for i in range(m):
                for j in range(n):
                        if np.all(img1[i][j]) != 0:
                                canvas[i,j] = img1[i,j]
        return canvas
```

The following code uses the above functions.

```
im1_warped = warp_image(im1, np.linalg.inv(H), (1200,1200,3))
im15 = stitch(im1_warped, im5, pad = 100)
```

3 Selecting matching points by mouse clicking and computing homography using opencv

With the code provided in item 3, matching points can be selected from clicking on the matching points in each image. Opencv's cv.findHomography() method calculates the homography when two arrays of matching points are given. When the number of matching points are increased, the Sum Squared difference(SSD) value between the calculated homography matrix and VGG's homography matrix will reduced. Not exactly clicking on the matching pixel affects the computation. The computed Homography matrices have the SSD values of **39.09** for 5 points with the VGG's provided homography matrix. Accurately selecting the matching points will increase the accuracy of the computation of the homography matrix.

4 Computing Homography without opencv

Points that were used in section 3 are used for the computations in this section as well. Which enables the comparison between calculated homographies.

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix} = c \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

by eliminating c the following equation can be derived,

$$A\vec{h} = \vec{0} \tag{2}$$

where,

$$A = \begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -\hat{x}_i x_i & -\hat{x}_i y_i & -\hat{x}_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -\hat{y}_i x_i & -\hat{y}_i y_i & -\hat{y}_i \end{pmatrix}$$

and

$$\vec{h}^T = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}$$

This defines a least square problem minimizing $\|A\vec{h} - \vec{0}\|^2 = 0$. Since $\vec{h}$ is only defined up to scale we can solve for unit vector $\hat{\vec{h}}$. The solution for this is the right singular vector of A corresponding to the smallest singular value. This is solved using np.linalg.svd() method. The resulting array is divided by the last element and reshaped into a 3×3 matrix.

The homography computed in this method has SSD value of **42.57** which is higher to opencv's implementation of finding homography. This might be a result of higher optimization in opencv. Resulting stitched image is provided in figure 4b.

(a) Homography calculated using 5 matching points

(b) Homography calculated by svd

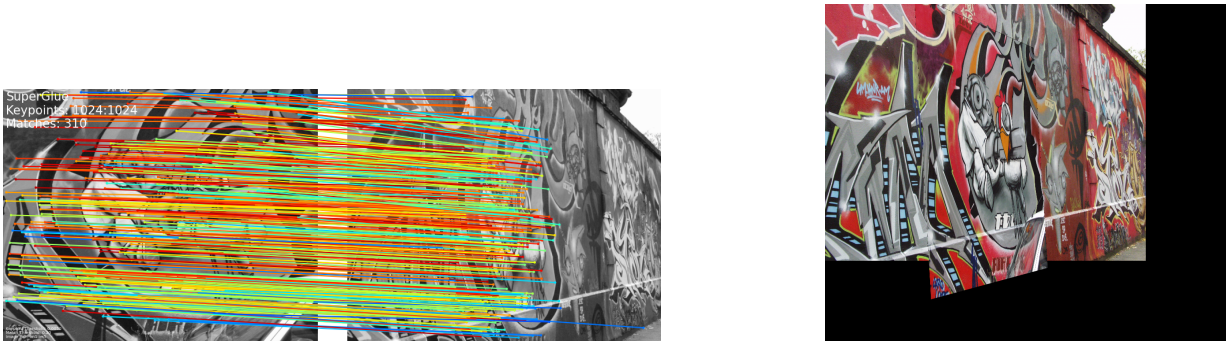
Figure 4: Results of question 3 and 4. These results have visible mismatches when stitching. edges of Image 1 can be clearly seen in both images. Increasing the number of matching points using superglue finds a better homography matrix thus better visual results in figure 5b

5 Stitching Images using SuperGlue algorithm

SuperGlue [1] is an image matching algorithm which utilizes a graph neural network to identify and match features between a pair of images. Using the provided pretrained model of SuperGlue we get a matching set of points between 2 images.

Running SuperGlue on figure 2a and figure 2b gives a set of 310 matching pairs of points. Then using those points to find homography with RANSAC method provides a homography matrix which has a SSD value of **9.88** with the VGG's homography. Running SuperGlue on a pair of images will result in a .npz file which contains key points from both images and whether they are matching. By processing this file we can extract the matching points and use them to calculate homography with RANSAC. Then the image warping and stitching is carried out using the functions mentioned in the above sections.

```
def find_matching_points(m):
        q1 = []
        q2 = []
        for i in range(len(m['matches'])):
```



(a) Feature matching with SuperGlue **(b)** Stitched image using SuperGlue

Figure 5: Using SuperGlue for feature matching and stitching image

```
                if mat['matches'][i] !=-1:
                        q1.append(m['keypoints0'][i])
                        q2.append(m['keypoints1'][m['matches'][i]])
        q1 = np.asarray(q1)
        q2 = np.asarray(q2)
        return q1, q2

points12 = find_matching_points(m1)
h12, s= cv.findHomography(points12[0], points12[1], method= cv.RANSAC)
mount2 = warp_image(img2, (h12), (2000,2000,3))
mount2 = stitch(mount2, img1, pad = (50,350))
```

Output of this code is provided in figure 6

Figure 6: Stitching 2 images using SuperGlue

References

[1] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperGlue: Learning feature matching with graph neural networks," in *CVPR*, 2020. [Online]. Available: `https://arxiv.org/abs/1911.11763`.

Appendix

The entire code is provided at github repository at `https://github.com/thieshanthan/EN2550_assignment_a01`.