HW 1

陳翰雯 - B08902092

Problem 1: Homography Estimation

Q1-1. Feature Matching

The local feature detection and matching is done through the use of SIFT and BFMatcher:

`sift = cv.SIFT_create()`

`sift.detectAndCompute()`

`cv.BFMatcher()`

Outliers are rejected through the use of Ratio Test:

`if m.distance < 0.75 * n.distance then ...`

Fig 1. Correspondence between 1-0.png and 1-1.png, k=4

Fig 2. Correspondence between 1-0.png and 1-1.png, k=8

Fig 3. Correspondence between 1-0.png and 1-1.png, k=20

Fig 4. Correspondence between 1-0.png and 1-2.png, k=4

Fig 5. Correspondence between 1-0.png and 1-2.png, k=8

Fig 6. Correspondence between 1-0.png and 1-2.png, k=20

Q1-2. Error comparison for Direct Linear Transform

k	1-1.png	1-2.png
4	138.6954997414783	344.5914141100657
8	1.9192956673091424	530.7090221202224
20	0.873630218781328	552.5337657388872

Q1-3. Error comparison for Normalized Direct Linear Transform

k	1-1.png	1-2.png
4	138.6954997414783	344.5914141100657

k	1-1.png	1-2.png
8	1.853424335180171	543.8543578528142
20	0.873630218781328	2273.492847400319

Observations:

As k increases ($k = 4$ to $k = 8$ to $k = 20$):

- In the case of 1-1.png, as number of k increases, the error decreases
- In the case of 1-2.png, as the number of k increases, the error increases too. This may be due to the presence of many outliers

When normalization is applied:

- In the case of 1-1.png, it shows that normalized DLT has lower or equal error value than normal DLT even if difference is not very significant.
- In the case of 1-2.png, it shows that normalized DLT showed to have higher value than normal DLT. In the case where $k = 20$, it showed to have a significant difference in error value

This somehow concludes that the SIFT feature matching does not work well with 1-2.png

Bonus - ORB with FLANN based Matcher

The detailed method could be found in the function `def experiment()`

k	1-1.png (ORB)	1-1.png (SIFT)	1-2.png (ORB)	1-2.png (SIFT)
4	680.1663637737864	138.6954997414783	**9.899332386285263**	344.5914141100657
8	3.3587352863808064	1.9192956673091424	558.2143048840554	530.7090221202224
20	2.6960548855249713	0.873630218781328	20810.299508553824	552.5337657388872

This method does not really work well with 1-1.png, we could see that SIFT perform a lot better.

In the case of 1-2.png, it does perform a lot better when $k = 4$, we see a significant difference in the error value (highlighted in green). However, as k increases the error value spikes up due to the presence of outliers.

Even though ORB provided a lot better results than SIFT in the case of 1-2.png, the overall error value is still high. This means that this method is still not a considerable one.

Fig 7. Correspondence between 1-0.png and 1-1.png, k=4 (ORB)

Fig 8. Correspondence between 1-0.png and 1-1.png, k=8 (ORB)

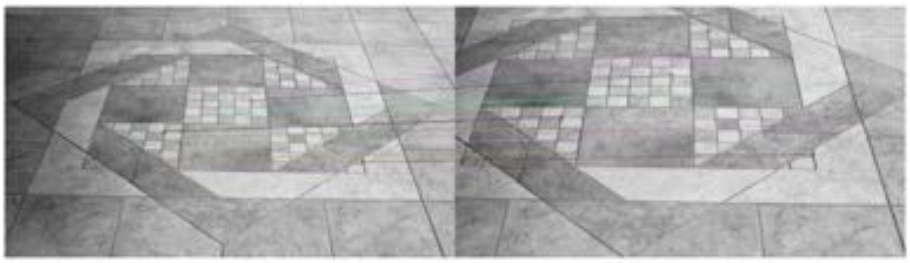

Fig 9. Correspondence between 1-0.png and 1-1.png, k=20 (ORB)

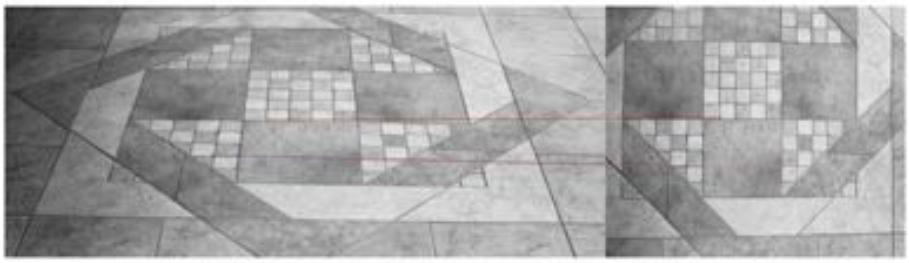

Fig 10. Correspondence between 1-0.png and 1-2.png, k=4 (ORB)

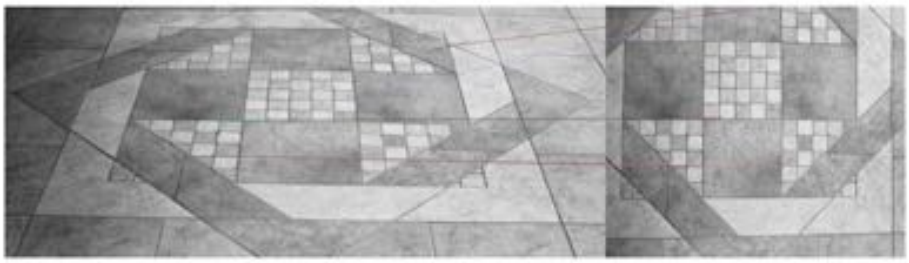

Fig 11. Correspondence between 1-0.png and 1-2.png, k=8 (ORB)

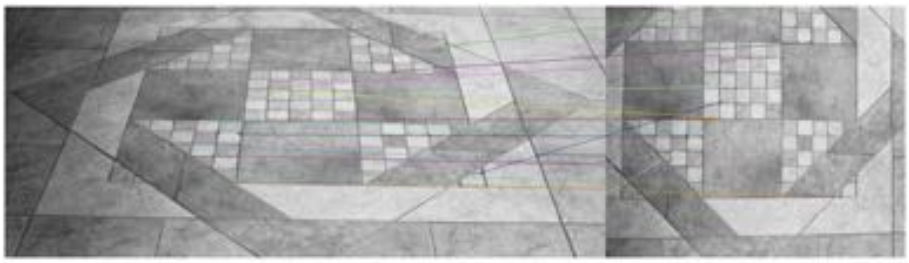

Fig 12. Correspondence between 1-0.png and 1-2.png, k=20 (ORB)

Bonus - Applying RANSAC

Here, we tried different k combinations for a total of 10000 trials and choose the best combinations with the lowest error values.

k	1-1.png (RANSAC)	1-1.png (SIFT)	1-2.png (RANSAC)	1-2.png (SIFT)
4	0.6065823493597492	138.6954997414783	0.6896945296468507	344.5914141100657
8	0.6163048323931887	1.9192956673091424	0.9013562549735729	530.7090221202224
20	0.6973880053343726	0.873630218781328	1.0711231590911232	552.5337657388872

This method showed to have worked well with both 1-1.png and 1-2.png. Even though the results will not be exactly the same as we tried different random combinations, however, overall, it showed to have small error values in comparison to using SIFT and ORB only. Hence, using RANSAC could be considerable method.

Fig 13. Correspondence between 1-0.png and 1-1.png, k=4 (RANSAC)

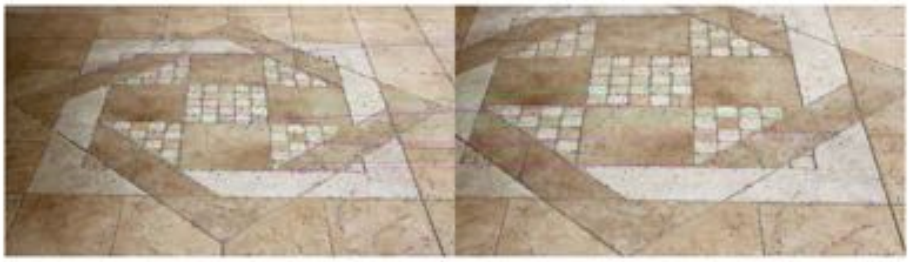

Fig 14. Correspondence between 1-0.png and 1-1.png, k=8 (RANSAC)

Fig 15. Correspondence between 1-0.png and 1-1.png, k=20 (RANSAC)

Fig 16. Correspondence between 1-0.png and 1-2.png, k=4 (RANSAC)

Fig 17. Correspondence between 1-0.png and 1-2.png, k=8 (RANSAC)

Fig 18. Correspondence between 1-0.png and 1-2.png, k=20 (RANSAC)

Problem 2: Document Rectification

We first select 4 corners of the document to be rectified. Once coordinates are found, we then find the homography between the 4 corners of the document and the 4 corners of the image size (The image size we select is 800x600). This is so that the document can fill the whole 800x600 space.

initial coordinates of document: `[[297, 191], [574, 410], [305, 743], [28, 516]]`

final coordinates: `[[0, 0], [shape[1]-1, 0], [shape[1]-1, shape[0]-1], [0, shape[0]-1]]`

Once homography is found, we then do backward warping as follows:

We first create an empty image

```
new_image = np.zeros((shape[0], shape[1], 3), dtype='uint8')
```

We then find the inverse of the homography matrix. (This is to map back the coordinates of the new image to the original image)

`inv_H = np.linalg.inv(H)`

`inv_H = inv_H/inv_H[2][2]`

We then loop through every pixel to find the respective coordinates in the original image

```
for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # Backward Warping, finding the position in the original image
            mul = np.matmul(inv_H, [j,i,1])
            mul = mul/mul[-1] # dividing by z

            y = mul[0]
            x = mul[1]
```

Once we found the respective `x,y` coordinates in the original image, we then do bilinear interpolation to decide on the color intensity of each pixel.

The main equation used: $x = w_a a + w_b b + w_c c + w_d d$

`new_img[i][j] = interpolation(img, x, y)`

```
def interpolation(img, x, y):
    x1 = math.floor(x)
    x2 = math.floor(x+1)
    y1 = math.floor(y)
    y2 = math.floor(y+1)

    x_d1 = x-float(x1)
    x_d2 = float(x2)-x
    y_d1 = y-float(y1)
    y_d2 = float(y2)-y

    img=np.array(img)
    result = 0

    if x1 < 0 or x2 >= img.shape[0] or y1 < 0 or y2 >= img.shape[1]:
        return 0, 0
    else:
        wa = x_d2*y_d2
        wb = x_d1*y_d2
        wc = x_d1*y_d1
        wd = x_d2*y_d1

        a = img[x1][y1].astype(float)
        b = img[x2][y1].astype(float)
        c = img[x2][y2].astype(float)
        d = img[x1][y2].astype(float)

        result += a * wa
        result += b * wb
        result += c * wc
        result += d * wd
```

```
    result.astype(int)
    return result
```

Fig 19. Document before rectification (2.png)

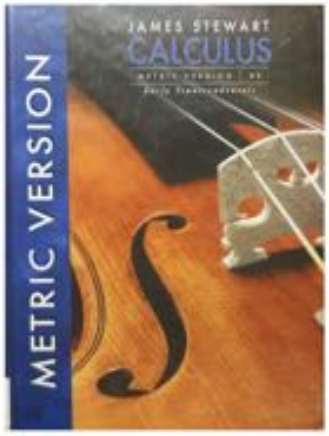

Fig 20. Document after rectification (2_warp.png)

How to execute:

1.py:

`python 1.py [path to image1] [path to image2] [path to groundtruth correspondence]`

2.py:

`python 2.py [path to document image]`

Environment

Python = 3.7.12

Matplotlib = 3.5.1

Numpy = 1.21.4

OpenCV = 4.5.4.60