

Homework 2  
Computer Vision 2022 Spring  
310554031 數據所 葉詠富

Github: <https://github.com/frankye1000/NYCU-ComputerVision/tree/master/HW2>

## ➤ Preprocess

### 1. Import package

```
import cv2
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt

from HW2 import *
```

### 2. Use HW2.py function read\_img to read 2 images(left, right).

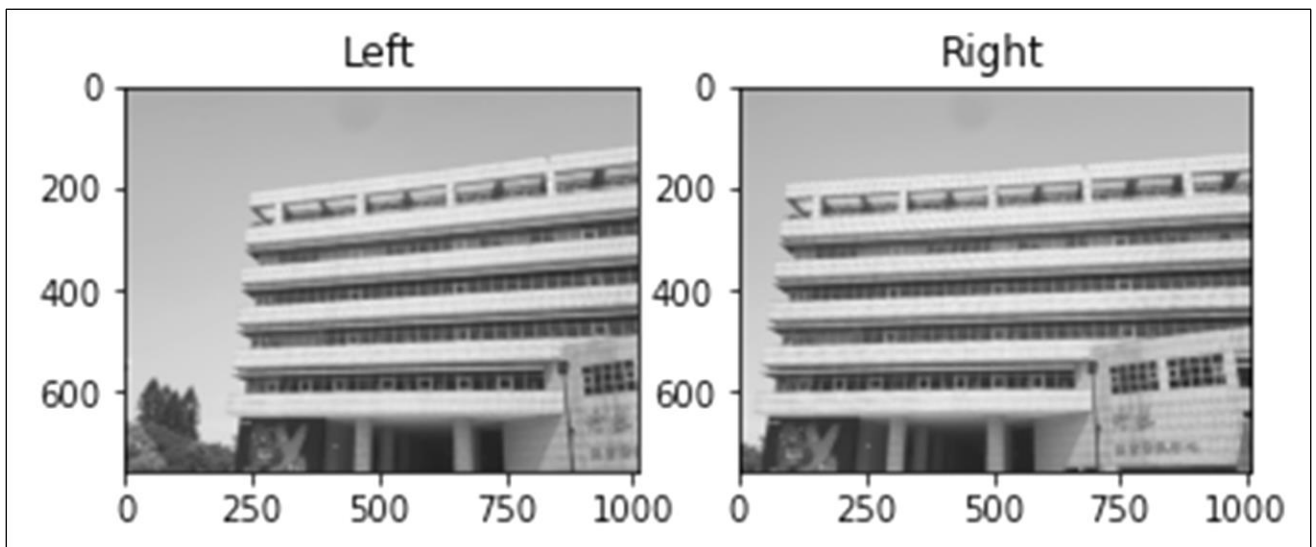
## 0. Read image

```
imgLeftName = 'm1'
imgRightName = 'm2'
imgLeft, imgLeft_gray, imgLeft_rgb = read_img('test/{}.jpg'.format(imgLeftName))
imgRight, imgRight_gray, imgRight_rgb = read_img('test/{}.jpg'.format(imgRightName))
```

```
plt.subplot(1, 2, 1)
plt.title('Left')
plt.imshow(imgLeft_gray, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Right')
plt.imshow(imgRight_gray, cmap='gray')

plt.show()
```



## ➤ Image stitching

### 1. Detecting key point(feature) on the images

Use SIFT to detect key points(red points).

```
def SIFT(img):
    SIFT_Detector = cv2.SIFT_create()
    kp, des = SIFT_Detector.detectAndCompute(img, None)

    return kp, des

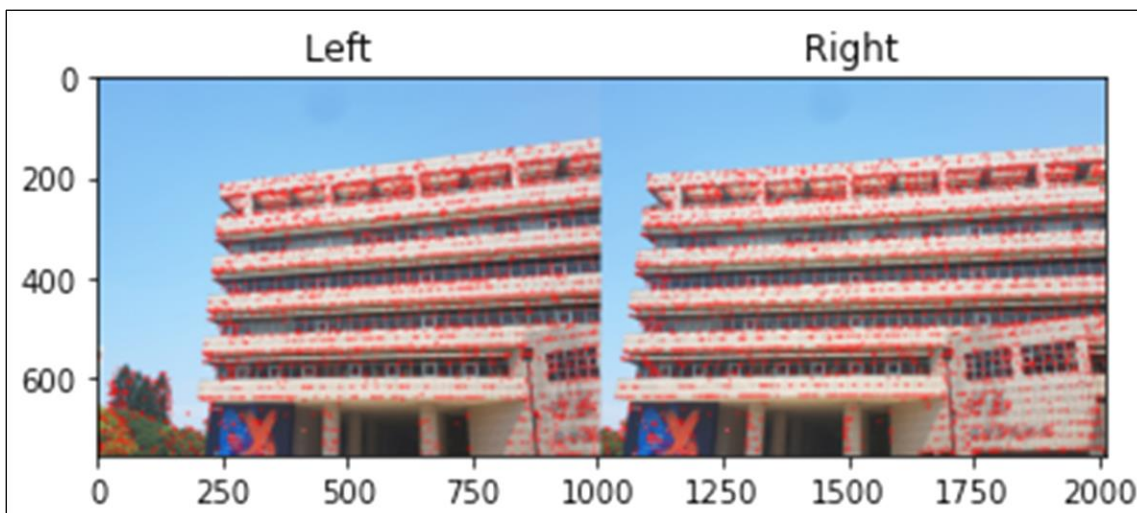
def plot_sift(gray, rgb, kp):
    tmp = rgb.copy()
    img = cv2.drawKeypoints(image=gray,
                             keypoints=kp,
                             outImage=tmp,
                             color=(255,0,0),
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG)

    return img

kpsLeft, dessLeft = SIFT(imgLeft_gray)
kpsRight, dessRight = SIFT(imgRight_gray)

kp_left_img = plot_sift(imgLeft_gray, imgLeft_rgb, kpsLeft)
kp_right_img = plot_sift(imgRight_gray, imgRight_rgb, kpsRight)
total_kp = np.concatenate((kp_left_img, kp_right_img), axis=1)

plt.title('Left                                Right')
plt.imshow(total_kp)
plt.show()
```



### 2. Finding features correspondences (feature matching)

Use KNN to calculate the Euclidean distance between each key point in the left image and each key point in the right image, and find the top two key points with the shortest distance.

```

def Knn(dessLeft, dessRight):
    ...
    Finding the K closest neighbors to the target.
    Brute-force : Comparing with the all 2-norm of SIFT feature (the 2-norm of descriptor)

    input:
        :dessLeft : left picture descriptors from SIFT
        :dessRight: right picture descriptors from SIFT
    output:
        :matches:
            [left SIFT feature index,[D1 right SIFT feature index, D1 distance],[D2 right SIFT feature index, D2 distance]]
    ...
    matches = []
    for i, desLeft in tqdm(enumerate(dessLeft)):
        D1 = [None,100000]
        D2 = [None,100000]
        for j, desRight in enumerate(dessRight):
            # Euclidean distance
            destemp = np.linalg.norm(desLeft-desRight)
            if D2[1] > destemp:
                if D1[1] > destemp:
                    D2 = D1
                    D1 = [j,destemp]
                else:
                    D2 = [j,destemp]
            matches.append([i,D1,D2])
    print("經過knn 的matches數量:", len(matches))
    return matches

```

Then set threshold for each key point through Lowe's ratio to get good match points.

```

def LoweRatioTest(matches, threshold=0.5):
    ...
    Lowe's Ratio test for eliminating bad match.
    1. For every key point P in image1 using 2NN to get 2 matched key points P1 & P2 in image2
    2. Computing the 2-norm of P1 & P2 between P named D1 , D2
    3. If D1 < threshold * D2 then P1 is a good match

    input:
        :matches : from knn
        :threshold: ratio between 0 to 1
    output:
        :good_matches: left picture coordinate & right picture coordinate
            [[x_left, y_left, x_right, y_right],....]
    ...
    good_matches = []
    for match in matches:
        kpleft_ID = match[0]
        D1_ID     = match[1][0]
        D1_dis    = match[1][1]
        D2_ID     = match[2][0]
        D2_dis    = match[2][1]

        # Lowe's Ratio test
        if D1_dis < threshold * D2_dis:
            good_matches.append(list(kpsLeft[kpleft_ID].pt + kpsRight[D1_ID].pt))

    print("經過Lowe's Ratio test 的good matches數量:", len(good_matches))
    good_matches = np.array(good_matches)
    return good_matches

```

plot good matches points.

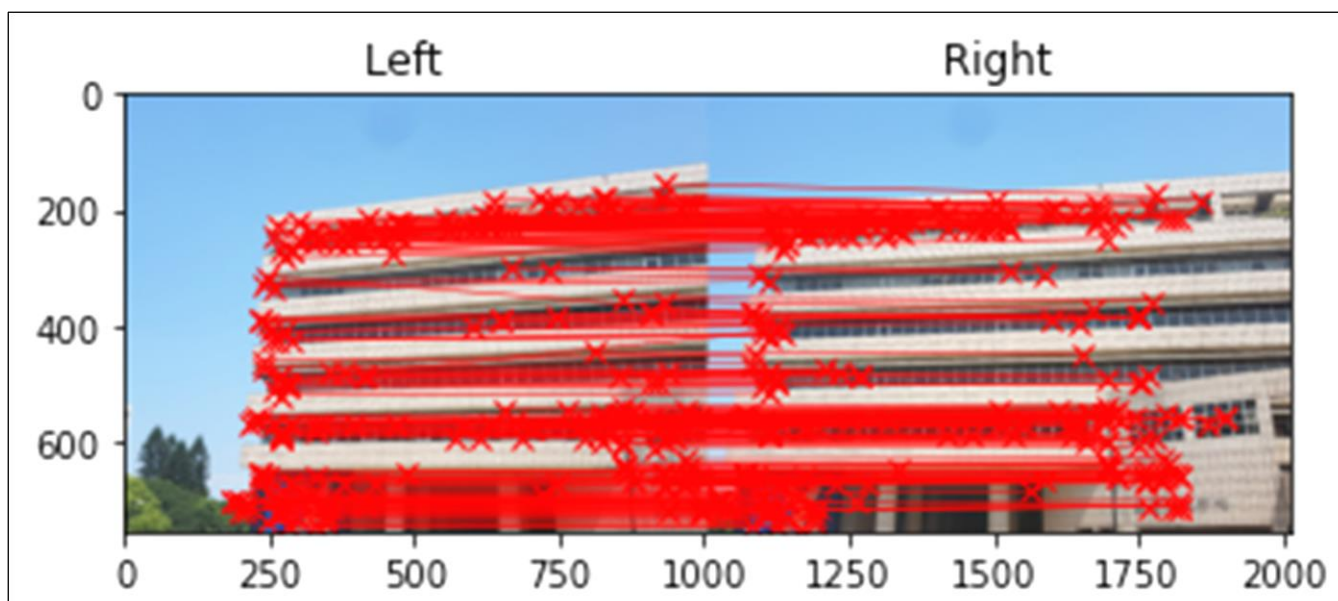
```
def plot_matches(matches, total_img):
    match_img = total_img.copy()
    offset = total_img.shape[1]/2
    fig, ax = plt.subplots()
    ax.set_aspect('equal')
    ax.imshow(np.array(match_img).astype('uint8')) # RGB is integer type

    ax.plot(matches[:, 0], matches[:, 1], 'xr')
    ax.plot(matches[:, 2] + offset, matches[:, 3], 'xr')

    ax.plot([matches[:, 0], matches[:, 2] + offset], [matches[:, 1], matches[:, 3]],
            'r', linewidth=0.5)
    plt.title('Left                                Right')
    plt.show()
```

經過 knn 的 matches 數量: 3196

經過 Lowe's Ratio test 的 good matches 數量: 227



### 3. Computing homography matrix.

Follow the step(A~E) below to get the homography matrix.

- Construct a linear system as:  $P_2 = HP_1$ ,  $P_2 = (x_2, y_2, 1)$ ,  $P_1 = (x_1, y_1, 1)$  where  $P_2$  and  $P_1$  are correspondence points,  $H$  is homography matrix.
- For perspective transformation, you can use 4 pairs of match result to solve 8 unknown variables in homography matrix.
- Using SVD decomposition to find Least Squares error solution.
- the solution = eigenvector of associated with the smallest eigenvalue ( $V$  stores the eigenvector of, stores the singular value (root of eigenvalue)) find the smallest number in and  $H$  = corresponding vector in  $V^T$ .
- Remember to normalize  $h_{33}$  to 1.

PS: SVD eigenvalue 由大到小排列，且  $h$  的解是「the eigenvector of  $A^T A$  associated with the smallest eigenvalue」，所以選  $V$  最後一個 eigenvector



```

def random_pairs(matches, n=4):
    idx = random.sample(range(len(matches)), n)
    pairs = [matches[i] for i in idx]
    return np.array(pairs)

def homography(pairs):
    """
    1. Construct a linear system as:  $P_2 = H P_1$ ,  $P_2 = (x_2, y_2, 1)$ ,  $P_1 = (x_1, y_1, 1)$  where  $P_2$  and  $P_1$  are correspondence points,
        $H$  is homography matrix.
    2. For perspective transformation, you can use 4 pairs of match result to solve 8 unknown variables in homography matrix.
    3. Using SVD decomposition to find Least Squares error solution
    4. the solution = eigenvector of associated with the smallest eigenvalue ( $V$  stores the eigenvector of,
       stores the singular value (root of eigenvalue)) find the smallest number in  $U$  and  $H$  = corresponding vector in  $V^T$ 
    5. Remember to normalize  $h_{33}$  to 1

    input:
        :pairs: 4 pairs
    output:
        :H: homography matrix(3,3)
    """
    A = np.zeros((8,9))
    pointer_a = 0
    pointer_b = 4
    for i in range(pairs.shape[0]):
        x1 = pairs[i][0]
        y1 = pairs[i][1]
        x1_b = pairs[i][2]
        y1_b = pairs[i][3]

        A[pointer_a][0] = x1
        A[pointer_a][1] = y1
        A[pointer_a][2] = 1
        A[pointer_a][6] = -x1*x1_b
        A[pointer_a][7] = -y1*x1_b
        A[pointer_a][8] = -x1_b

        A[pointer_b][3] = x1
        A[pointer_b][4] = y1
        A[pointer_b][5] = 1
        A[pointer_b][6] = -x1*y1_b
        A[pointer_b][7] = -y1*y1_b
        A[pointer_b][8] = -y1_b

        pointer_a += 1
        pointer_b += 1

    U, s, V = np.linalg.svd(A)  # SVD eigenvalue 由大到小排列，且h的解是「the eigenvector of ATA associated with the smallest
    H = V[-1].reshape(3, 3)
    H = H/H[2, 2]  # standardize to let  $w^T H[2,2] = 1$ 
    return H

```

Then, we use the RANSAC (Random Sample Consensus) algorithm to find the best Homography matrix corresponding to the two images.

Follow the step(A~E) below to get the best homography matrix.

- A. 隨機挑出要計算出 Homography  $H$  所需的最少個的點
- B. 求出 Homography  $H$  的解
- C. 將所有 Good matching 特徵配對去計算  $P = HP'$  估，並統計有多少對在容許內的誤差範圍，並稱為 Inliers
- D. 接著用這個  $H$  去算誤差  $\|P'_{\text{估}} - P'_{\text{真}}\|$ ，如果誤差比規定的門檻還小，我們可以相信這次結果是可靠的。若為可靠，那麼統計這次 Inliers 數量有沒有比最佳次數還多，有的話我們就更新最佳  $H$  和最佳 Inliers 數量
- E. 反覆進行步驟 1 到 4 反覆  $N$  次，並取得最佳  $H$

```

def get_error(points, H):
    num_points = len(points)
    all_p1 = np.concatenate((points[:, 0:2], np.ones((num_points, 1))), axis=1)
    all_p2 = points[:, 2:4]
    estimate_p2 = np.zeros((num_points, 2))
    for i in range(num_points):
        temp = np.dot(H, all_p1[i])
        estimate_p2[i] = (temp/temp[2])[0:2] # set index 2 to 1 and slice the index 0, 1
    # Compute error
    errors = np.linalg.norm(all_p2 - estimate_p2, axis=1) ** 2

    return errors

def ransac(good_matches, threshold, iters=2000):
    """
    1. 隨機挑出要計算出 Homography H 所需的最少個的點。
    2. 求出 Homography H 的解
    3. 將所有 Good matching 特徵配對去計算  $P=HP'$  估，並統計有多少對在容許內的誤差範圍，並稱為 Inliers。
    4. 接著用這個 H 去算誤差  $\|P'_{\text{估}} - P'_{\text{真}}\|$ ，如果誤差比規定的門檻還小，我們可以相信這次結果是可靠的，若為可靠，
       那麼統計這次 Inliers 數量有沒有比最佳次數還多，有的話我們就更新最佳 H 和最佳 Inliers 數量。
    5. 反覆進行步驟 1 到 4 反覆 N 次，並取得最佳 H。

    input:
        :good_matches: from knn
        :threshold    : ratio between 0 to 1
        :iters        : iters
    output:
        :best_inliers: best inliers points
        :best_H      : best H(3,3)
    """
    num_best_inliers = 0

    for i in range(iters):
        pairs = random_pairs(good_matches) # 隨機挑選四個最佳配對，計算Homography
        H = homography(pairs)

        # avoid dividing by zero
        if np.linalg.matrix_rank(H) < 3:
            continue

        errors = get_error(good_matches, H)
        idx = np.where(errors < threshold)[0]
        inliers = good_matches[idx]

        num_inliers = len(inliers)
        if num_inliers > num_best_inliers:
            best_inliers = inliers.copy()
            num_best_inliers = num_inliers
            best_H = H.copy()

    print("inliers/matches: {}/{}".format(num_best_inliers, len(good_matches)))
    return best_inliers, best_H

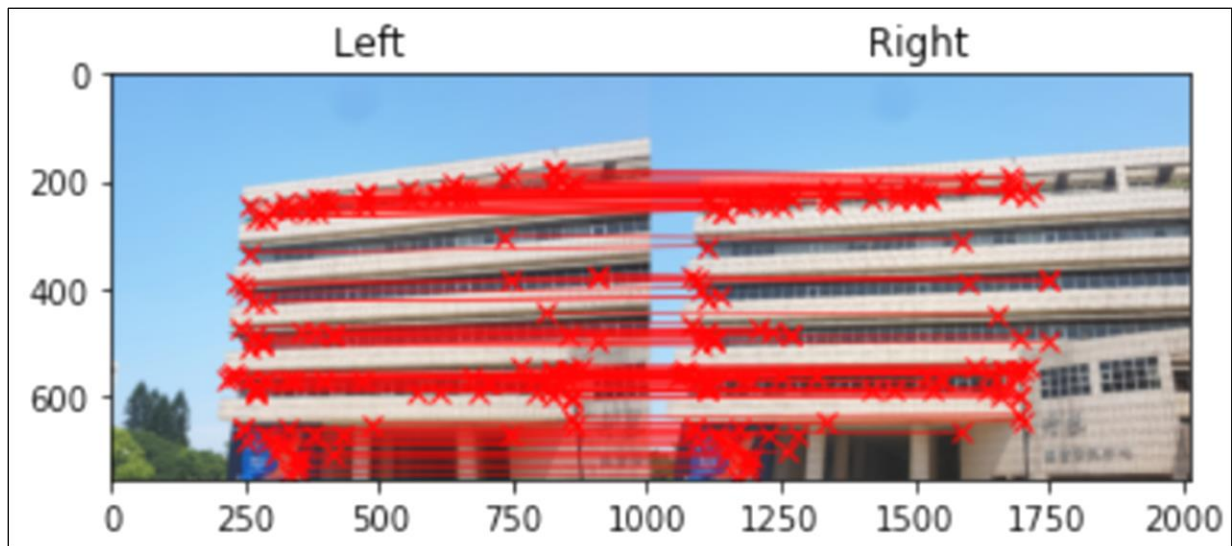
```

```

inliers, H = ransac(good_matches, 0.5, 10000)
plot_matches(inliers, total_img) # show inliers matches

inliers/matches: 121/227

```



#### 4. Stitching image (warp images into same coordinate system)

Use the Homography matrix to calculate the translation matrix  $A$ , the left image uses  $A*H$  for warp, and the right image uses  $A$  directly for warp. Finally, the right image is projected onto the left image.

The new projection logic is:

If the left pixel RGB total value is greater than the right, use the left pixel RGB, otherwise use the right.

Example:

	left image	right image
RGB	[1,2,3]	[4,5,6]
Sum(RGB)	6	15

We pick right image pixel.

PS: 這樣做 pixel 挑選投影後，比 (左圖+右圖)/2 接合處較圓滑。





```

def stitch_img(left, right, H):

    # Convert to double and normalize. Avoid noise.
    left = cv2.normalize(left.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)
    # Convert to double and normalize.
    right = cv2.normalize(right.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)

    # Left image
    height_l, width_l, channel_l = left.shape
    corners = [[0, 0, 1], [width_l, 0, 1], [width_l, height_l, 1], [0, height_l, 1]]
    corners_new = [np.dot(H, corner) for corner in corners]
    corners_new = np.array(corners_new).T # [wx', wy', w]
    x_news = corners_new[0] / corners_new[2] # x' = wx'/w
    y_news = corners_new[1] / corners_new[2] # y' = wy'/w
    y_min = min(y_news)
    x_min = min(x_news)

    translation_mat = np.array([[1, 0, -x_min], [0, 1, -y_min], [0, 0, 1]])
    H = np.dot(translation_mat, H)

    # Get height, width
    height_new = int(round(abs(y_min) + height_l))
    width_new = int(round(abs(x_min) + width_l))
    size = (width_new, height_new)

    warped_l = cv2.warpPerspective(src=left, M=H, dsize=size)

    # right image
    height_r, width_r, channel_r = right.shape
    height_new = int(round(abs(y_min) + height_r))
    width_new = int(round(abs(x_min) + width_r))
    size = (width_new, height_new)

    warped_r = cv2.warpPerspective(src=right, M=translation_mat, dsize=size)

    black = np.zeros(3) # Black pixel.

    # Stitching procedure, store results in warped_l.
    for i in tqdm(range(warped_r.shape[0])):
        for j in range(warped_r.shape[1]):
            pixel_l = warped_l[i, j, :]
            pixel_r = warped_r[i, j, :]

            if np.sum(pixel_l) >= np.sum(pixel_r):
                warped_l[i, j, :] = pixel_l
            else:
                warped_l[i, j, :] = pixel_r

    stitch_image = warped_l[:warped_r.shape[0], :warped_r.shape[1], :]
    return stitch_image

```

```

stitchImg = stitch_img(imgLeft_rgb, imgRight_rgb, H)

plt.title('{} & {}'.format(imgLeftName, imgRightName))
plt.imshow(stitchImg)
plt.imsave('test/{}.jpg'.format(imgLeftName, imgRightName), stitchImg)

```



➤ Image stitching Result

m1~m2



m1~m3



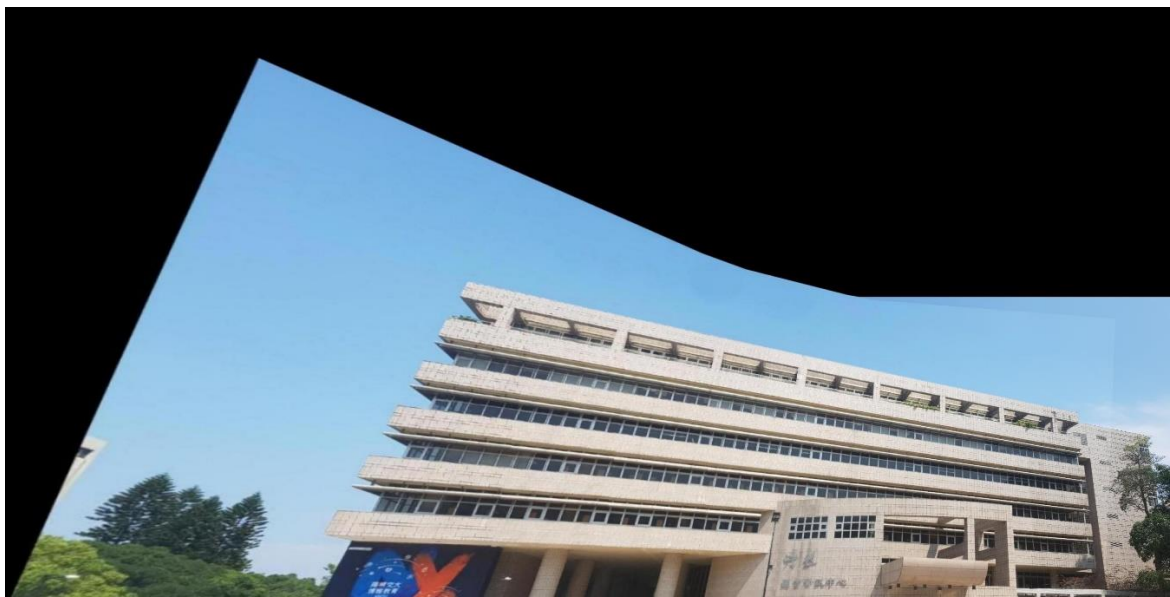
m1~m4

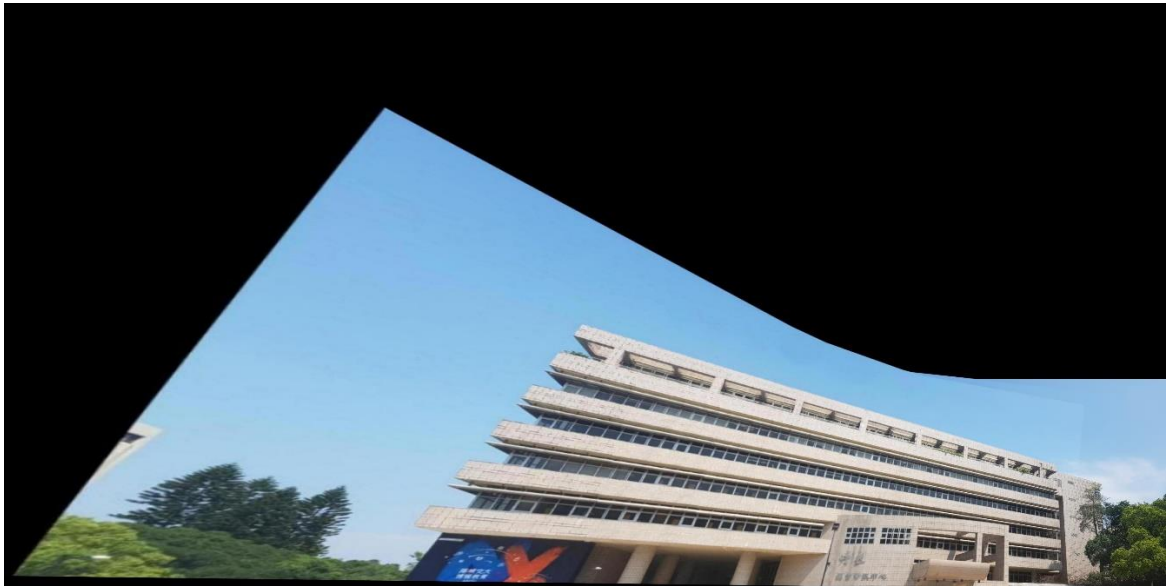



m1~m5



m1~m6



m1~m7	
m1~m8	

## ➤ Discussion & Conclusion

1. 使用 Lowe' s Ratio test 非常重要，可以找出 good matches，減少配對錯誤，和降低計算成本。
2. 在 Lowe' s ratio test 和 Ransac error 的 threshold 參數設定非常重要，會影響 ratio test 能不能找到 good match 和 ransac 的迭代次數。
3. 由上圖可以看到合併到 m1~m8 時，圖形已變形非常嚴重，可以使用 Cylindrical projection 改善情況。

## ➤ Reference

- <https://www.796t.com/content/1549421657.html>
- <https://tigercosmos.xyz/post/2020/05/cv/image-stitching/>



- <https://yungyung7654321.medium.com/python%E5%AF%A6%E4%BD%9C%E8%87%AA%E5%8B%95%E5%85%A8%E6%99%AF%E5%9C%96%E6%8B%BC%E6%8E%A5-automatic-panoramic-image-stitching-28629c912b5a>