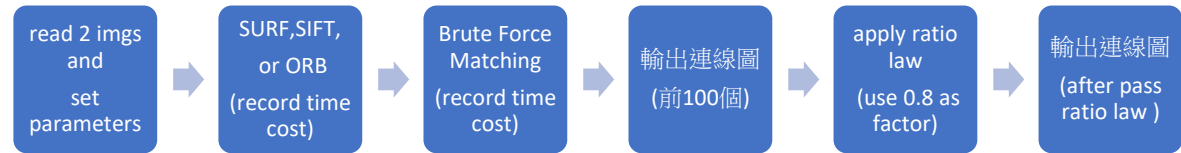


一、說明使用方法

由於 opencv-contrib-python 4.4.0.46(以及之後的各版本)在 windows 系統上，會因為專利問題，無法直接使用 SURF，需要重新編譯 opencv 成 nonFree version。因此利用 colab，安裝 3.4.2.17(該版本只能在 linux 下才可 pip 到)

<https://colab.research.google.com/drive/1WiQl0L4bGpMPgqMNVJm42iLyBgV5z8LI?usp=sharing> (colab use local temp disk)



2.Parameters (modify the picture file)

```

[ ] img0_addr = '/lab11/images/book_0.JPG' #設定img0圖片位置
    img1_addr = '/lab11/images/book_1.JPG' #設定img1圖片位置

ENABLE_GRAY_SCALE = False #控制是否要變成灰階
ENABLE_LIMIT_DRAWING_LINES_BEFORE_RATIO = False #在ratio test前的SURF、SIFT、ORB結果是否要設置只顯示前n個(distance最大越前面)，
                                                    #若輸出pairs數量比限制小，則直接輸出目前的最大pairs數量
limit_draw_lines_before_ratio_test = 25 #限制pairs輸出數量(before ratio test)
ENABLE_LIMIT_DRAWING_LINES_AFTER_RATIO = False #在ratio test後的SURF、SIFT、ORB結果是否要設置只顯示前n個(distance最大越前面)，
                                                    #若輸出pairs數量比限制小，則直接輸出目前的最大pairs數量
limit_draw_lines_after_ratio_test = 25 #限制pairs輸出數量(after ratio test)

if ENABLE_GRAY_SCALE == True:
    img0 = cv2.imread(img0_addr, cv2.IMREAD_GRAYSCALE)
    img1 = cv2.imread(img1_addr, cv2.IMREAD_GRAYSCALE)
else:
    img0 = cv2.imread(img0_addr)
    img1 = cv2.imread(img1_addr)
  
```

◀ 參數設定

```

# Perform SURF feature detection and description.
surf = cv2.xfeatures2d.SURF_create()
t1 = time.time()
kp0 = surf.detect(img0, None)
kp1 = surf.detect(img1, None)
t2 = time.time()
kp0, des0 = surf.compute(img0, kp0)
kp1, des1 = surf.compute(img1, kp1)
t3 = time.time()
# Perform brute-force KNN matching.
bf = cv2.BFMatcher() #use L2 by default
pairs_of_matches = bf.knnMatch(des0, des1, k=2)
t4 = time.time()
total_kp = len(kp0)+len(kp1)

# Perform SIFT feature detection and description.
sift = cv2.xfeatures2d.SIFT_create()
t1 = time.time()
kp0 = sift.detect(img0, None)
kp1 = sift.detect(img1, None)
t2 = time.time()
kp0, des0 = sift.compute(img0, kp0)
kp1, des1 = sift.compute(img1, kp1)
t3 = time.time()
# Perform brute-force KNN matching.
bf = cv2.BFMatcher() #use L2 by default
pairs_of_matches = bf.knnMatch(des0, des1, k=2)
t4 = time.time()
total_kp = len(kp0)+len(kp1)

# Perform ORB feature detection and description.
orb = cv2.ORB_create()
t1 = time.time()
kp0 = orb.detect(img0, None)
kp1 = orb.detect(img1, None)
t2 = time.time()
kp0, des0 = orb.compute(img0, kp0)
kp1, des1 = orb.compute(img1, kp1)
t3 = time.time()
# Perform brute-force KNN matching.
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False) #use Hamming format
pairs_of_matches = bf.knnMatch(des0, des1, k=2)
t4 = time.time()
total_kp = len(kp0)+len(kp1)
  
```

▲分別利用 cv2.xfeatures2d.SURF_create()、cv2.xfeatures2d.SIFT_create()、cv2.ORB_create()，建立 feature detection 物件，用 detect 方法去找兩張圖的特徵點，compute 方法運算尋找兩張圖過程中。以上兩步驟皆各自計時，另計算每特徵點所花費的 compute 時間。接著用 brute force knnMatch 去配對兩張圖的特徵點，其中 SURF、SIFT 是用預設的 L2 方式配對，而 ORB 則需使用 HAMMING 配對。

```

# Sort the pairs of matches by distance.
pairs_of_matches = sorted(pairs_of_matches, key=lambda x: x[0].distance)
  
```

▲將配對的使用測試圖的向量距離由小至大排序，可視為相異度越小者優先。

```

# Draw the pairs of matches.(from best to worst)
img_pairs_of_matches = cv2.drawMatchesKnn(
    img0, kp0, img1, kp1, pairs_of_matches[:limit_num], img1,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
  
```

▲畫出前 limit_num 個最好的配對(limit_num 定義在 parameter，預設是 100)

```

# Apply the ratio test.
matches = [x[0] for x in pairs_of_matches
            if len(x) > 1 and x[0].distance < 0.8 * x[1].distance]
  
```

▲使用 0.8 倍做為 ratio test 的標準(OpenCV document 上提及通常設在 0.7~0.8，預設 0.8)，即當匹配點與第二可能匹配的相似度要有一定差距，才可視為比較

好的匹配。

```
# Draw the pairs of matches.(from best to worst)(after ratio test)
img_matches = cv2.drawMatches(img0, kp0, img1, kp1, matches[:limit_num],img1,
                             None, singlePointColor=(255, 0, 0), flags=0)
```

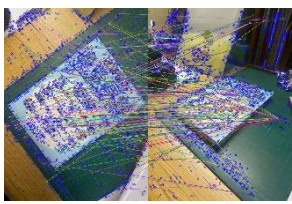


▲畫出前 limit_num 個最好的配對，並標示任何有找到的 feature point (limit_num 定義在 parameter，預設是全畫出來)

二、測量結果

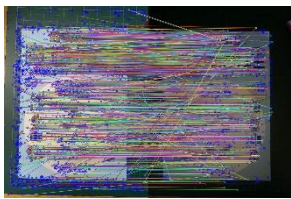
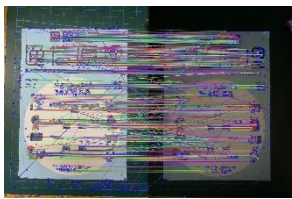
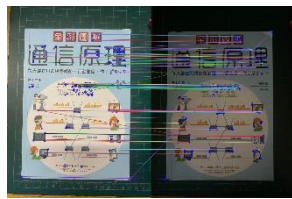
以下測試使用 iphone11 照相後，長寬皆先縮小 5 倍至 604*806 or 806*604，而 ratio test 的係數使用 0.8。

1.不同角度：

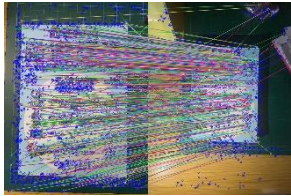
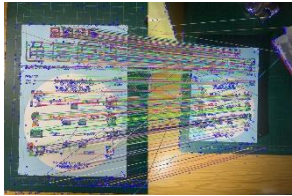
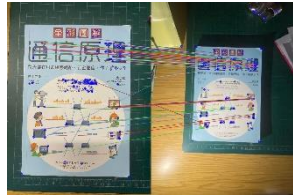
i) before ratio test line = 100 (下表使用 SURF-SURF, SIFT-SIFT, ORB-ORB)

Method	SURF with Brute Force	SIFT with Brute Force	ORB with Brute Force
Kp0/kp1	2793/2542	1329/1314	500/500
length	Total 5335	Total 2644	Total 1000
Detector	0.4291408061981201 s	0.3114180564880371 s	0.02365803718566894 s
Descriptor per kp	0.00017340775394171 s	0.00011956349082865 s	2.1173954010009e-05 s
matching	0.29621291160583496 s	0.13910770416259766 s	0.00552487373352050 s
After	193 pairs	95 pairs	11 pairs
After image			

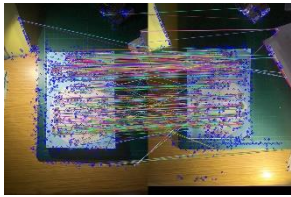
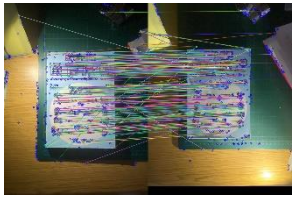

2.不同亮度： (下表使用 SURF-SURF, SIFT-SIFT, ORB-ORB)

method	SURF with Brute Force	SIFT with Brute Force	ORB with Brute Force
Kp0/kp1	4544/2432	1587/1650	500/500
length	Total 6976	Total 3237	Total 1000
Detector	0.3603181838989258 s	0.297119140625 s	0.021187305450439453 s
Descriptor per kp	0.00024199335400117 s	0.00011281787446887 s	2.08570957183837e-05 s
matching	0.4395639896392822 s	0.220916748046875 s	0.006192207336425781 s
After	1181 pairs	597 pairs	116 pairs
After image			

3.不同大小：(下表使用 SURF-SURF, SIFT-SIFT, ORB-ORB)


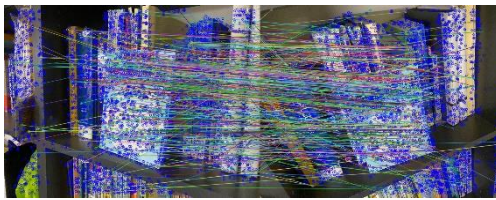
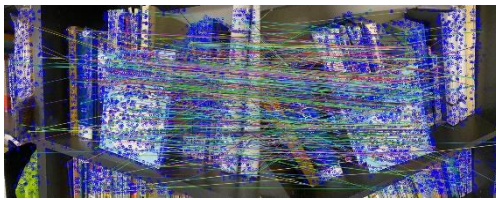

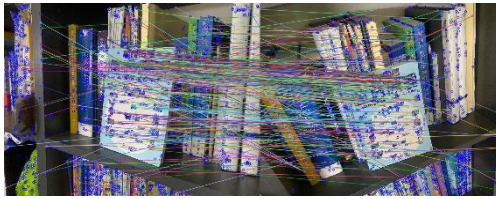
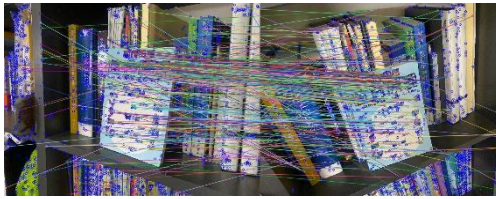



method	SURF with Brute Force	SIFT with Brute Force	ORB with Brute Force
Kp0/kp1	4544/2012	1587/1347	500/500
length	Total 6556	Total 2934	Total 1000
Detector	0.33013916015625 s	0.3015272617340088 s	0.02521681785583496 s
Descriptor per kp	0.00023396597758090 s	0.00011314050470311 s	2.45711803436279e-05 s
matching	0.36838221549987793 s	0.172621488571167 s	0.009119033813476562 s
After	751 pairs	427 pairs	61 pairs
After image			

4.不同光源方向：(下表使用 SURF-SURF, SIFT-SIFT, ORB-ORB)

method	SURF with Brute Force	SIFT with Brute Force	ORB with Brute Force
Kp0/kp1	1828/1647	1028/1099	500/500
length	Total 3475	Total 2127	Total 1000
Detector	0.24614357948303223 s	0.2515265941619873 s	0.01491999626159668 s
Descriptor per kp	0.00024488332460252 s	0.00013713372611192 s	2.09648609161376e-05 s
matching	0.12529826164245605 s	0.08694338798522949 s	0.005379199981689453 s
After	355 pairs	388 pairs	74 pairs
After image			


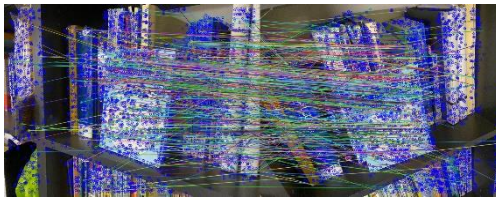

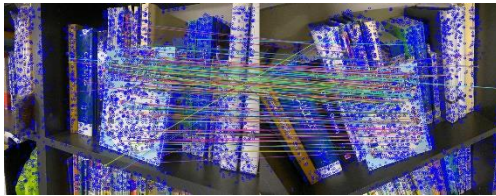
5-1.不同距離及角度：(下表使用 SURF-SURF, SIFT-SIFT, ORB-ORB)

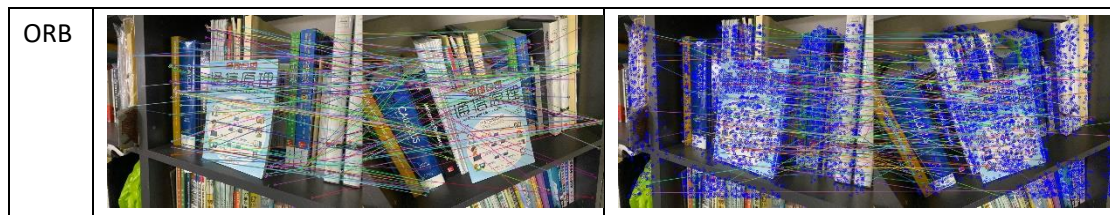
method	SURF with Brute Force	SIFTC with Brute Force	ORB with Brute Force
Kp0/kp1	5893/5606	2532/2104	500/500
length	Total 11499	Total 4636	Total 1000
Detector	0.4524261951446533 s	0.308307409286499 s	0.02257037162780761 s
Descriptor per kp	0.00017387208259399 s	8.42156545986275e-05 s	2.05910205841064e-05 s
matching	1.3407366275787354 s	0.42151641845703125 s	0.00535535812377929 s

After	364 pairs	296 pairs	29 pairs
	Before ratio test		After ratio test
SURF			
SIFT			
ORB			

6.不同距離及角度：（固定使用 SURF 的 detector）

method	SURF with Brute Force	SIFT with Brute Force	ORB with Brute Force
Kp0/kp1	5893/5606	5893/5606	5531/5272
length	Total 11499	Total 11499	Total 10803
Detector	0.4712710380554199 s	0.436826229095459 s	0.4548912048339844 s
Descriptor per SURF kp	0.00017157259459329 s	0.00073559747197149 s	5.19879623705464e-06 s
matching	1.3257741928100586 s	2.5753438472747803 s	0.631878137588501 s
After	364 lines	177 lines	135 lines

	Before ratio test (use SURF detector)	After ratio test (use SURF detector)
SURF		
SIFT		



三、分析比較

SIFT 找到特徵點的位置、尺寸並賦予特徵點方向後，可確保其移動、縮放、旋轉的不變性。此外還需要為關鍵點建立一個描述子向量，使其在不同光線與視角皆能保持不變性。

SURF 特徵 **SIFT** 特徵的加強版，差別在於 **SURF** 使用海森(Hessian)矩陣的行列式，比 **SIFT** 算法更高速。

ORB 特徵檢測具有尺度和旋轉不變性，運行時間遠優於 **SIFT** 和 **SURF**，但可以發現尋找到的特徵個數都是三者最少的，但配對表現是不錯的。

1. 不同角度：在此比較中，**ORB** 如預期的有最短的 **detector use time** 和 **descriptor time per key point** 以及 **matching time**，但是 **SIFT** 在上述後兩個層面都比 **SURF** 快約 1 倍。而這三個方法在 **after ratio test** 畫出線的數量及其特徵點數成正相關。
2. 不同亮度：和第一點相似，但這三個方法在 **after ratio test** 畫出線的數量都比第一點多。
3. 不同大小：和第一點相似，但這三個方法在 **after ratio test** 畫出線的數量介於第一點和第二點之間。
4. 不同光源方向：和前面不同的是，**SIFT after ratio test** 畫出線的數量甚至比 **SURF** 多，推論是因為 **SIFT** 對光線有不變性。
5. 不同距離及角度：**descriptor time per key point**：和前面相比，**ORB** 一樣是最快的，在 **descriptor time per key point**，**SIFT** 比 **SURF** 快約 1 倍，但在 **matching time** 中，**SIFT** 比 **SURF** 快了不只一倍。
6. 固定使用 **SURF** 的 **detector**：(比較 **descriptor time per SURF key point**) 一樣都是用 **SURF** 找特徵點，但 **ORB** 產生的特徵點數比另兩個少(推測 **ORB descriptor** 會篩選特徵)。在 **descriptor time per SURF key point**，**SURF** 比 **SIFT** 快了 3 倍以上，在 **matching time** 中 **SURF** 也比 **SIFT** 快了 1 倍(推測是特徵向量結構)。

然而與上述實驗 1~5 的 **SURF-SURF**, **SIFT-SIFT** 的 **descriptor time** 比較，會與 **descriptor time per SURF key point** 有不同的結果，前者是因為 **SURF** 的 **detector** 比 **SIFT** 能找的更多且更有效率，然而在 **descriptor** 階段會因為 **SIFT** 找到的特徵點較少，**descriptor** 階段花費時間自然也少，造成 **SURF** 會花時較 **SIFT** 多。而在實驗 6 會固定特徵點個數時，這時才可以正確比較出三者的 **descriptor** 階段效能。