Computer Vision HW4 Report

105062506 羅右鈞

Functions

find_matches.m

功能：來找出兩張圖片的對應點。

做法：

1. 首先使用 VLFeat 所提供的 `vl_sift()`，找出兩張圖片的 SIFT feature points $P = (p_1, p_2, \dots, p_m)$, $Q = (q_1, q_2, \dots, q_n)$，每個 feature point 帶有 128 維的 descriptor。
2. 以 Euclidean distance 來找出與 p_i 最接近的對應點 q_j，而且最近的點與第二近的點之間的距離要大於一定的門檻值，如果未達門檻值，則放棄當前這組對應點：
   $vec_dist(p_1, q_1,q_2,\dots,q_n) = dist(p_descriptor_{1}, q_descriptor_{1,2, \dots ,n})$
   $\ \frac{1^{st}min(vec_dist) - 2^{nd}min(vec_dist)}{1^{st}min(vec_dist)} > distance_ratio_threshold$

computeHomography.m

功能：給定對應點 (至少四組)，計算對應點之間的 homography。

註：延續作業二用來計算 homography 所使用的 `computeHomography.m`。

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{aligned} x_i' &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y_i' &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

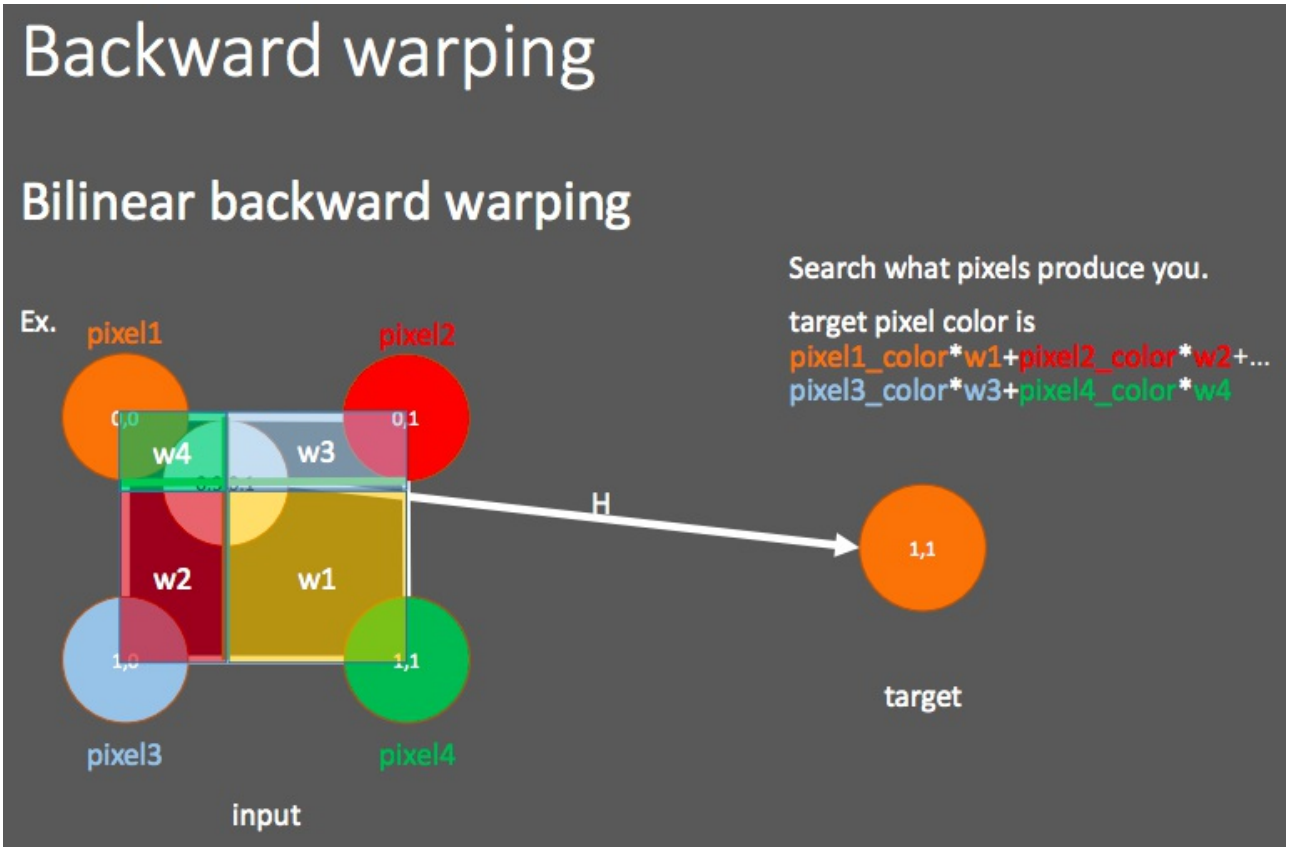
$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

做法：建立 $Ah = 0$ 後，我們將問題 format 成 $\arg\min|Ah|$, subject to $|h| = 1$ 的問題，h 其實就等於 eigenvector with smallest eigenvalue of $A^{ \mathrm{ T } }A$，然後再將 h vector 重組成 $H^{3\times3}$ matrix。

backwardWrap.m

功能：使用 backward wrapping 來內插顏色。

註：延續作業二用來計算 homography 所使用的 `backwardWrap.m`。



做法：找出 projected point 附近四個點的顏色，用 bilinear interpolation 的公式內差出 target point 的顏色。但這次因為需要 alpha blending，因此之前是直接將內插出來的顏色直接 assign 給目標圖片的 pixel 位置，這次需要考慮目標圖片的 pixel 的顏色，與內插出來的顏色用以下公式來內插：
$Current_{overlapped} = (1-\alpha)\cdot Transformed_{overlapped} + \alpha\cdot Current_{overlapped}$
其中 α 設定為 0.8。

ransacHomography

功能：使用 RANSAC 演算法求 homography。

做法：

1. 從全部的對應點中隨機抽四組出來，並用 `computeHomography()` 來計算 homography H。在這邊的對應點主要是用 `vl_sift()` 求出來的 SIFT 特徵點。
2. 再用計算出來的 H，將對一組點投影到其對應點，計算 $distance(transformed, ground_truth)$，而距離小於一定門檻值 $inlier_distance_threshold$ 則是視為 inliers。
3. 重複上述步驟 1, 2，直到跑 $max_iteration$ 次為止，從中選定最好的H，也就是能使得 inliers 的數量最大的 H，但數量至少也要達到一定的門檻值 $inlier_min_size = 0.2 \times size(corresponding_points)$，否則重新跑 $max_iteration$ 次。
4. 最後，再拿所有已視為 inliers 的那些點，用 `computeHomography()` 來計算更加精準的 H，並回傳。

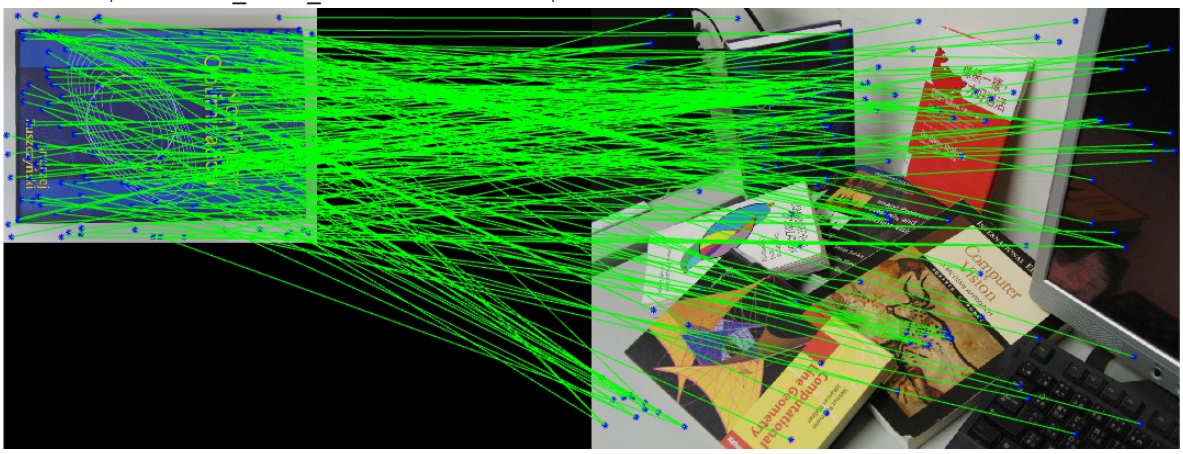
Results

Part 1-A

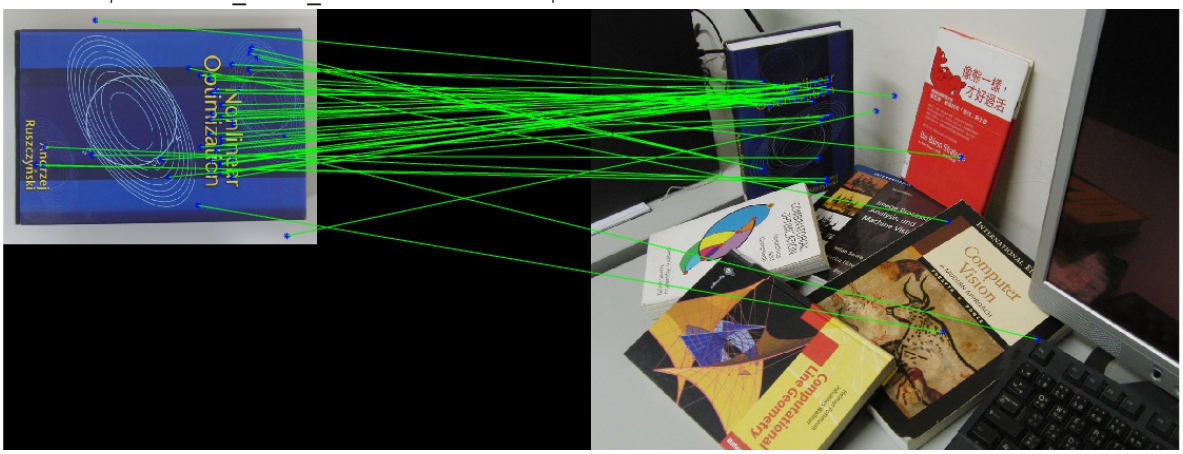
做法：`find_matches()` 分別找出三本不同的書與場景間的對影點，$distance_ratio_threshold$ 分別設為 0.07, 0.35。

結果：

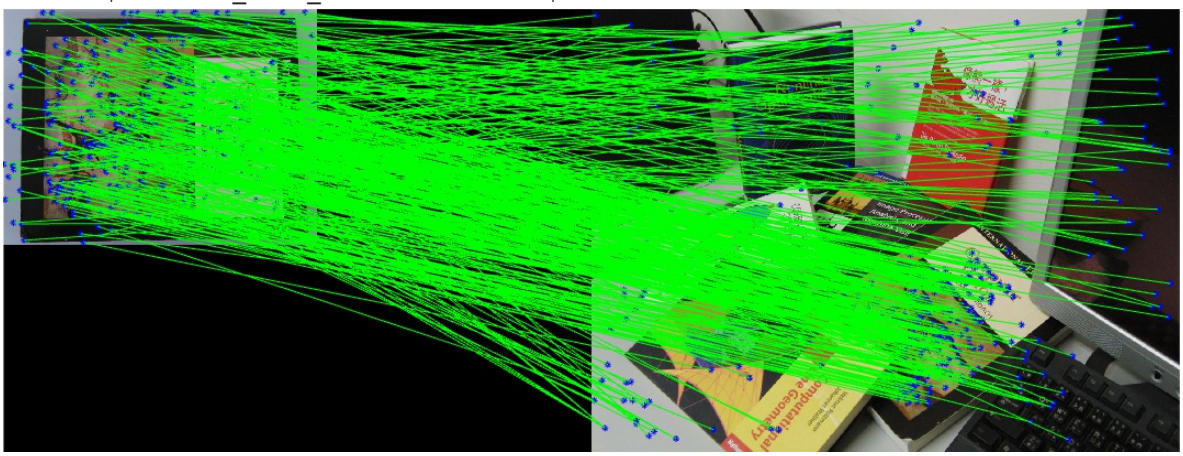
- book1: $distance_ratio_threshold = 0.07$



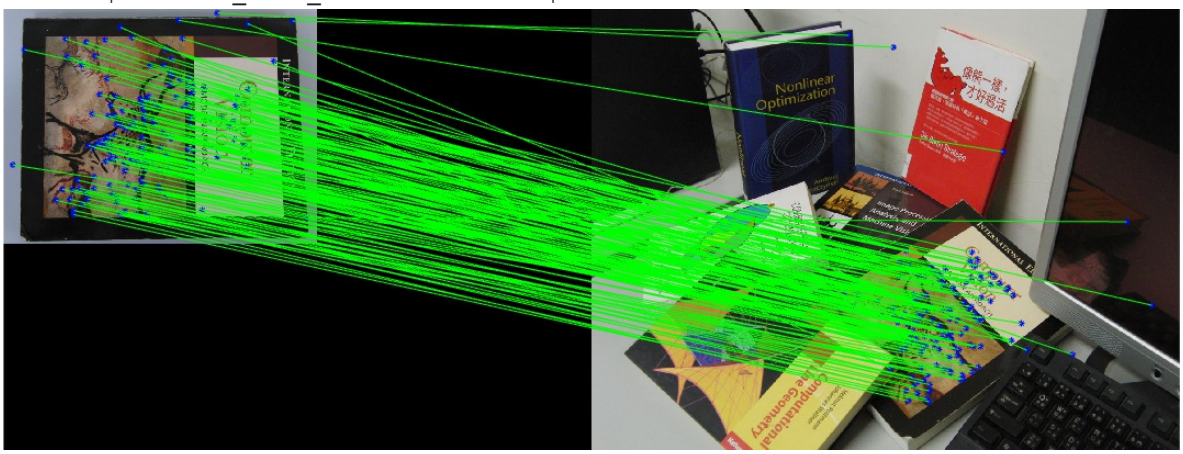
- book1: $distance_ratio_threshold = 0.35$



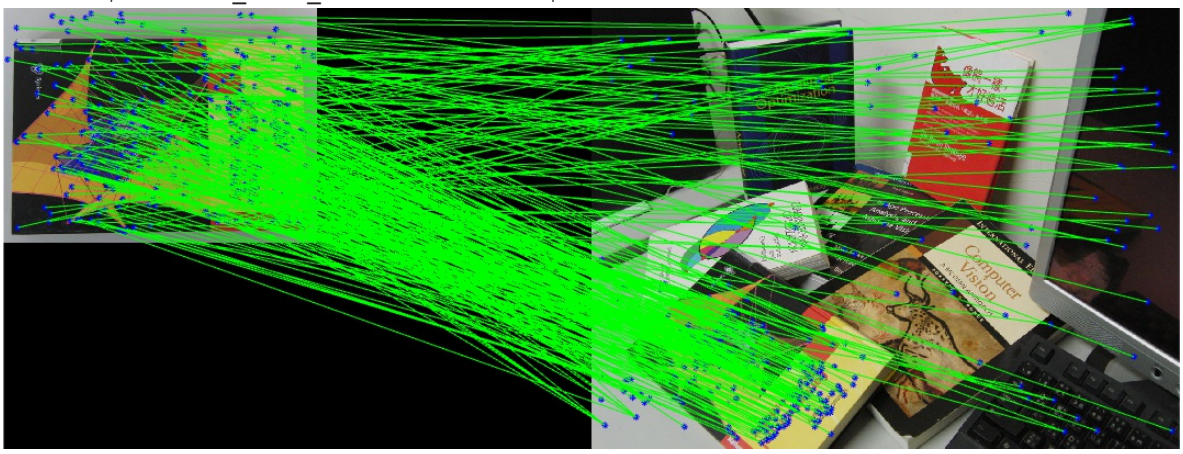
- book2: $distance_ratio_threshold = 0.07$



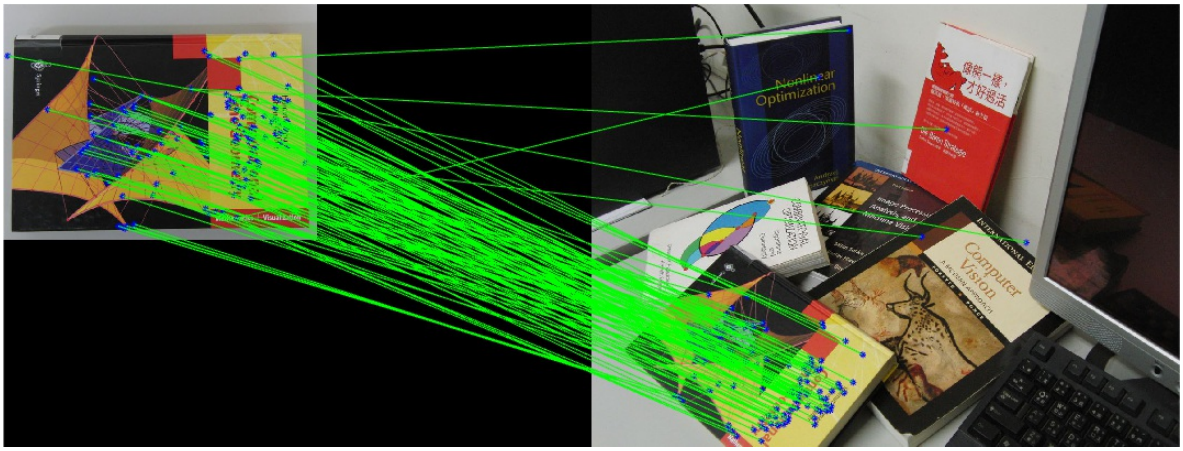
- book2: $distance_ratio_threshold = 0.35$



- book3: $distance_ratio_threshold = 0.07$



- book3: $distance_ratio_threshold = 0.35$

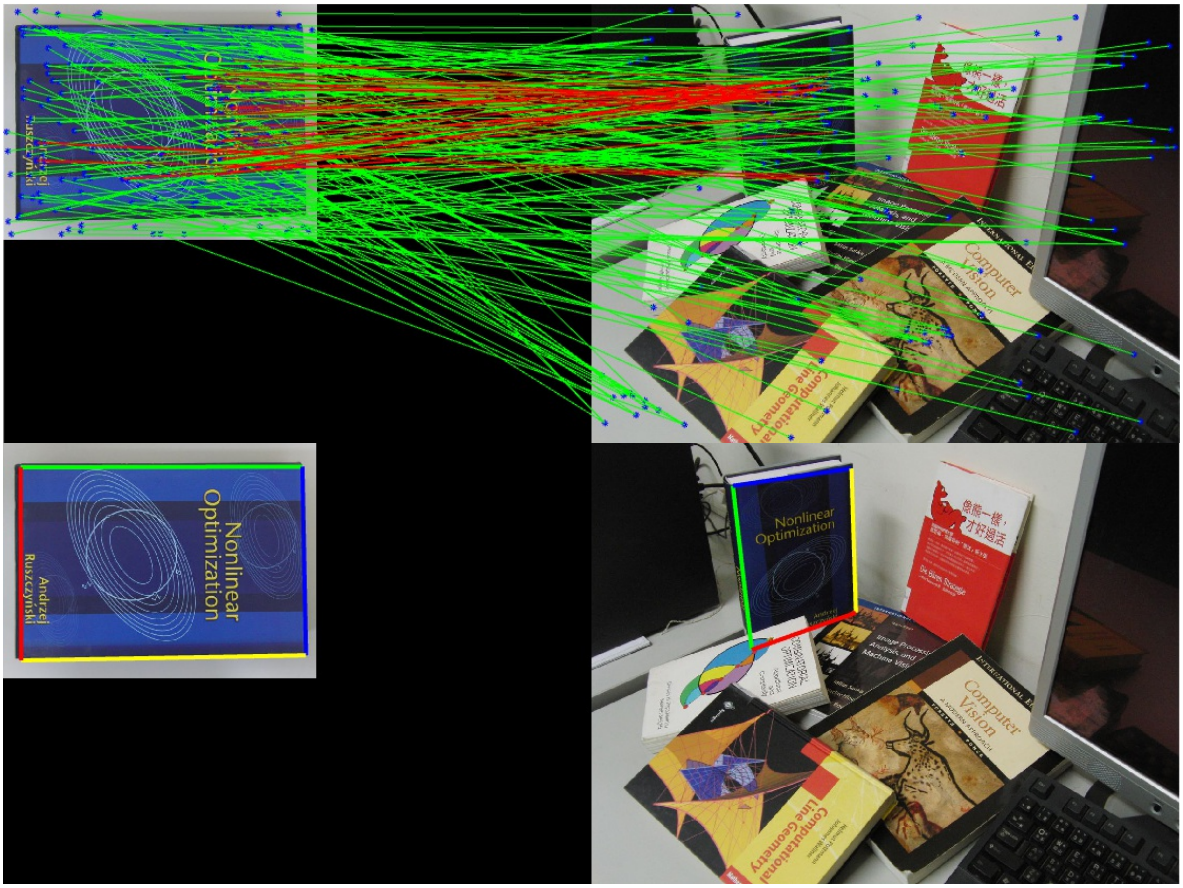


Part 1-B

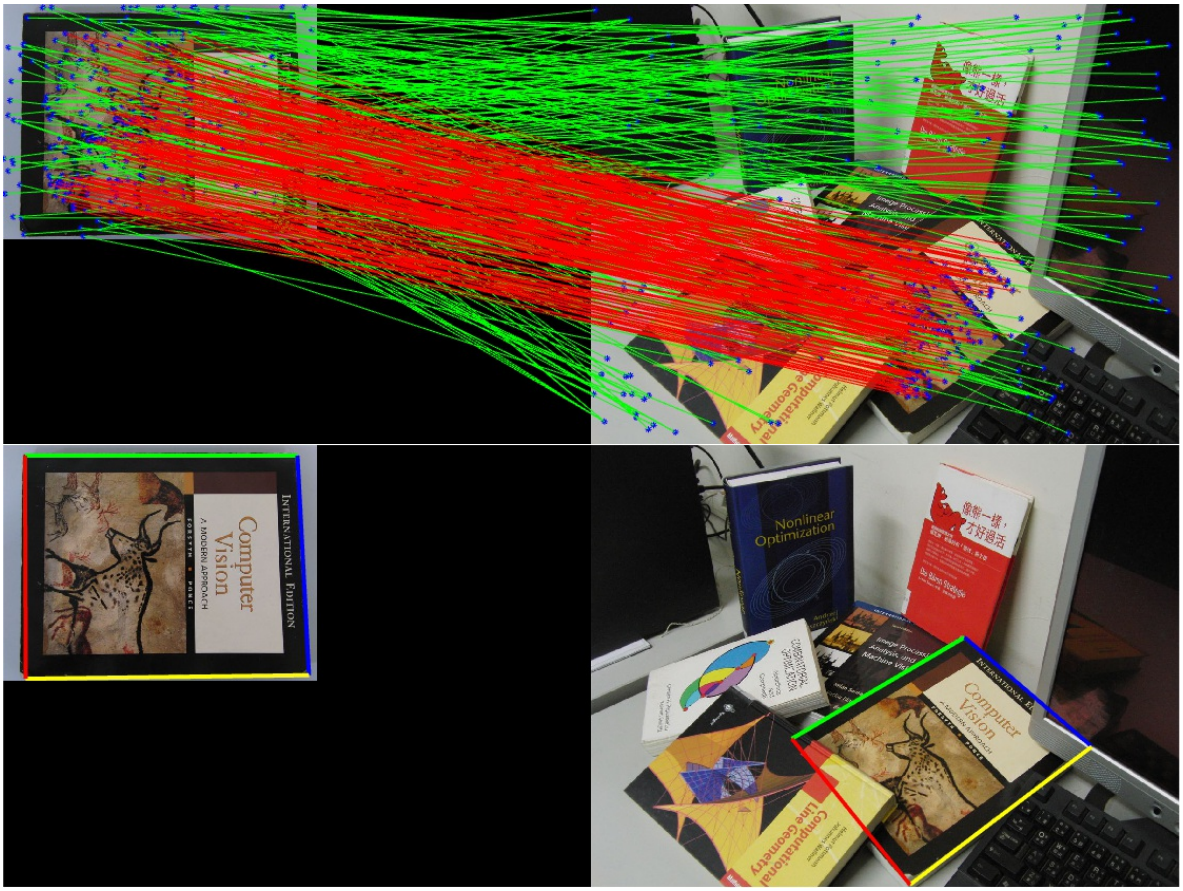
做法：利用上題所找出來的對應點 (使用 $distance_ratio_threshold = 0.07$ 來確保有兩百組以上的對應點)，使用 `ransacHomography()` 來找出對應點中的 inliers 與 outliers，以及 inliers 間的 homography H，其中 `ransacHomography()` 的參數 $inlier_distance_threshold = 30$。最後再使用 H 將書上先前手動點選的 boundaries transform 到場景中。

結果：

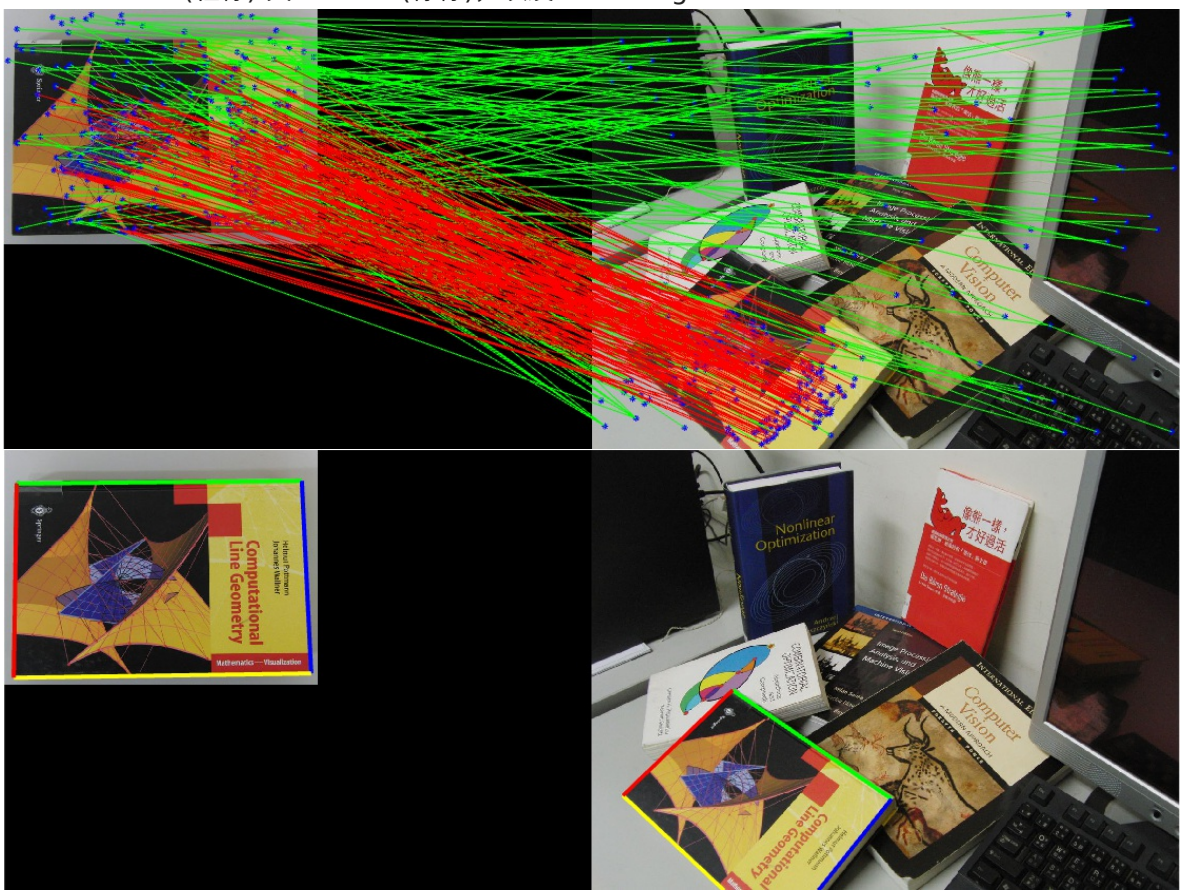
- book1: inliers(紅線) 與 outliers(綠線)，以及 matching boundries:



- book2: inliers(紅線) 與 outliers(綠線)，以及 matching boundries:



- book3: inliers(紅線) 與 outliers(綠線)，以及 matching boundries:



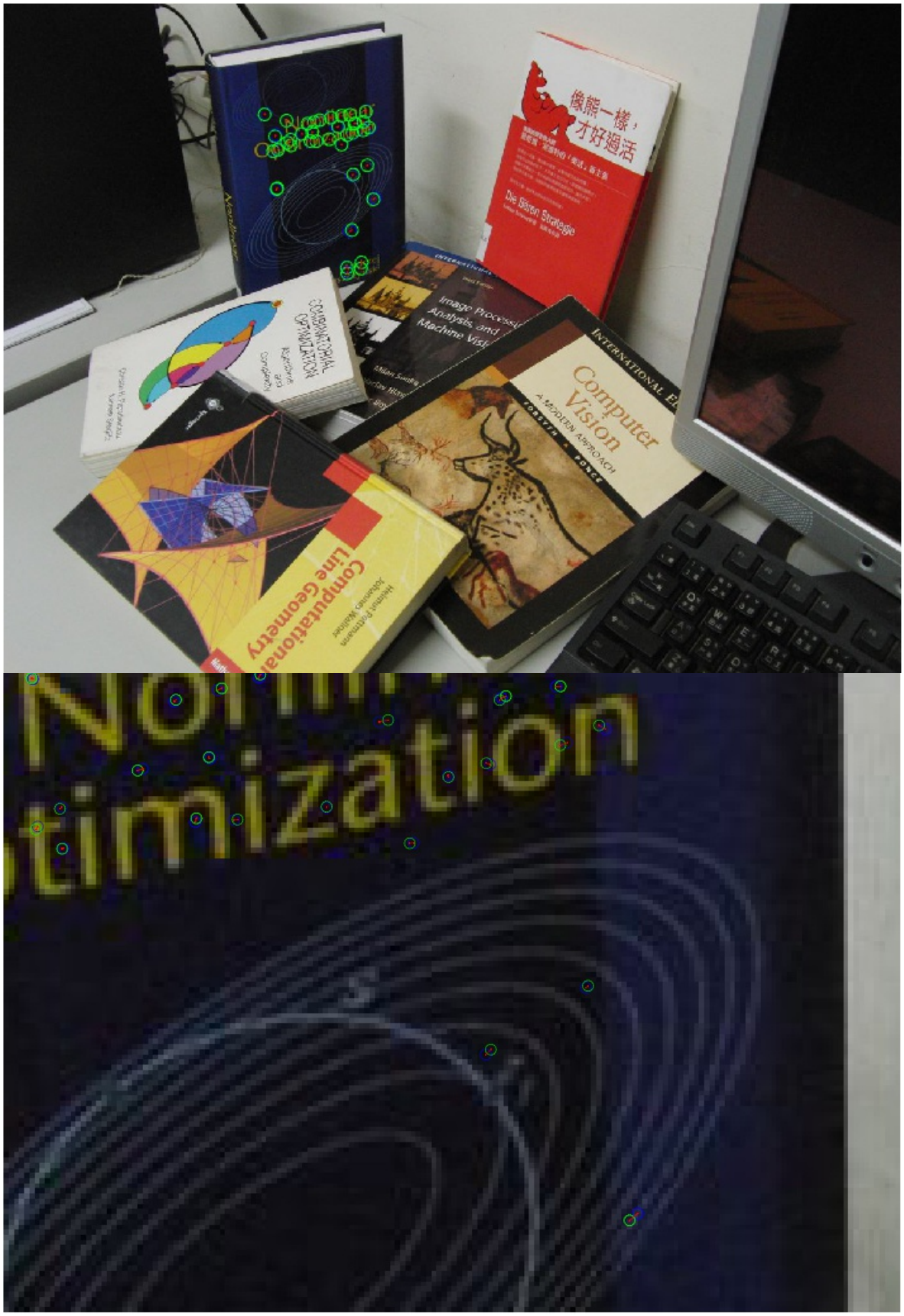
討論：在實作過程中，我嘗試許多不同的 $inlier_distance_threshold$ 來找出 inliers，如果 $inlier_distance_threshold$ 越近，找到符合門檻的 inliers 則越少，還有 $inlier_min_size$ 設定越多，則算出來的 H 也會越準，但是不能設定太高，否則原本 inliers 沒有這麼多的話，那永遠也達不到門檻。由於 RANSAC 是 random-based 的，每次算出的 H 可能不會比上一次的好，因此我們就讓它做 $max_iteration$ 次並從中挑選最好的 H，在程式中我設定 $max_iteration = 1000$。

Part 1-C

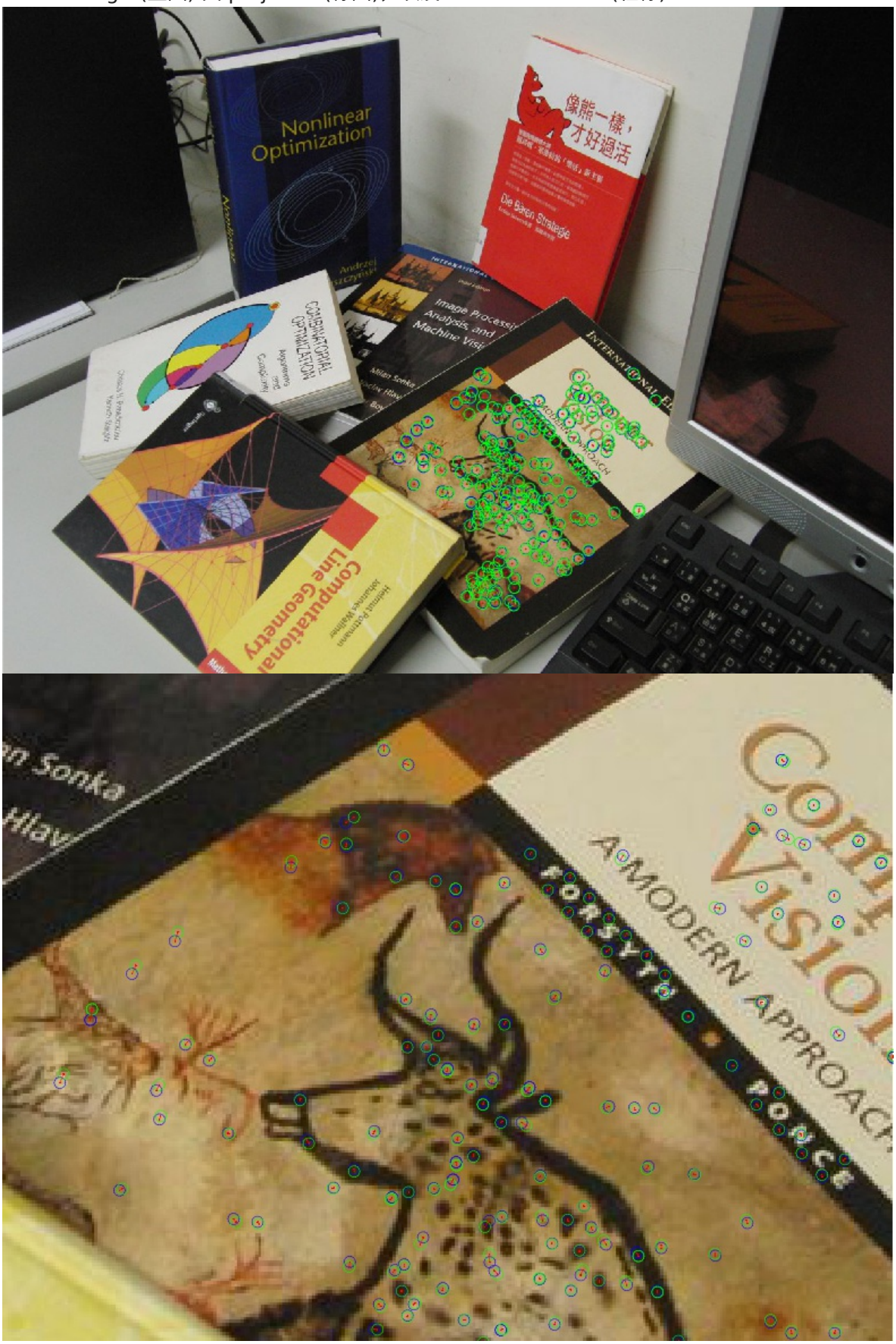
做法：利用上題所找出來的 homography H，將書上的 inliers transform 到場景中，並且跟場景中原本 inliers 的對應點一起畫出來。

結果：

- book1: target(藍圈) 與 projected(綠圈)，以及 deviation vector(紅線)



- book2: target(藍圈) 與 projected(綠圈)，以及 deviation vector(紅線)



- book3: target(藍圈) 與 projected(綠圈)，以及 deviation vector(紅線)

Part 2

做法：

1. 首先使用 `find_matches()` 分別找出 `left.jpg` 與 `center.jpg` 以及 `right.jpg` 與 `center.jpg` 的 SIFT 對應點。
2. 使用 `ransacHomography()` 分別找出 `left.jpg` 與 `center.jpg` 以及 `right.jpg` 與 `center.jpg` 的 inliers 對應點，以及 inliers 之間的 homography H。
3. 對 `left.jpg`, `center.jpg` 與 `right.jpg` 做 padding (做 padding 的原因是要以防之後用 backward wrapping 內插顏色時，會有超出原圖範圍外的點)，然後用步驟二算出來的 homography，將 padded 過的圖片中真正有原圖的地方的四個角落點 (boundary points)，用 H 投影求得即將要繪製的結果圖上的四個角落點 (projected boundary points)。不過 `center.jpg` 不需要投影，所以它的 projected boundary points 就是原圖的 boundary points，但為了維持程式碼的一致性，就當作它也是 projected 過的。
4. 由於我們要將 `left.jpg` 與 `right.jpg` 拼接至 `center.jpg`，因此我們勢必要對拼接過的結果圖做一些 shift，否則 `left.jpg` 拼接到 `center.jpg` 的會有些點落在 negative 的區域，也就是超出圖片(1,1)-(W,H)範圍外了，還有甚至一些點可能會落在超出原圖高度的地方。因此我們利用 projected boundary points 來找出 X 方向 negative 的點 shift 到 $x = 1$ 的 shift amount，以及找出 Y 方向最大的 shift amount。
5. 找出 X 方向與 Y 方向的 shift amount 之後，對 `left.jpg`, `center.jpg`, `right.jpg` 即將要繪製的結果圖上的 projected boundary points 做 shifting，得到合法的圖片範圍內的 shifted projected boundary points。
6. 我們使用 built-in function `poly2mask()`，利用 shifted projected boundary points 找出 boundary 內所有的 pixel 位置。這些 pixel 位置就是我們要內插顏色的目標位置 shifted target points。
7. 接著，我們也利用 shifted target points，扣掉原本的 shift amount 後來 unshift 回去得到 target points，再用 homography 將這些 target points 投影回 padded 過的原圖上，即 projected target points。這些 projected target points 就是內插顏色至 shifted target points 所需要的點。由於 `center.jpg` 沒有 transform 過，所以只要 unshift 它的 shifted target points 就可。
8. 使用 `backWrap()` 來依序對 `left.jpg`, `center.jpg`, `right.jpg` 內插至預設好大小的圖片上。

結果：

- 以下為 backward wrapping 中 alpha blending 的 alpha=0.5 的結果，由於經過 transformed 後的圖經由原先 padded 過的圖內差出來，因此可能會有些點落在原圖之外，特別是在邊界的地方常發生這樣的狀況，因此邊界內插出來的顏色會是接近深色的顏色，因此在 stitching 的時候，如果

alpha 值設定較低，也就是內插的顏色佔的比重較重時，則會出現下面結果圖中有一點點黑色邊界的狀況出現。

- 為了修復以上狀況，只要將 alpha 調高 (下面結果圖為 alpha=0.8)，也就是內插的顏色佔的權重低一點，就可以減少上述狀況的發生。

討論：
步驟二 ransacHomography() 的部分，如果 $inlier_distance_threshold$ 設定低一點，其實結果跟上面的結果並沒有太大的差別，因為求出來的 homography 還算不錯，不過當設定很大的時候，有就是說 inliers 的限制變低了，造成求出來的 homography 沒有比較好好，stitching 時圖片邊界變得更加明顯了，以下為結果圖：


ROWING BOATS
MOTOR BOATS
PEDAL BOATS