**Homework #2 –** **Object Detection & Image Warping**

**Due date: 2016/05/17**

**0456648 多工碩一 楊柏漢**

**繳交日期 : 2016/05/17**

**文件列表:**

1. **hw2.m**

**(程式檔，包含KNN,RANSAC, build homography matrix model, warping…)**

1. **result.dir**

**(作業生成結果)**

1. **object\_11.bmp~object\_22.bmp+target.bmp**

**(作業測資)**

1. **hw2\_0456648.docx與hw2\_0456648.pdf**

**(本檔，作業porject敘述，怕圖檔跑掉多付一個pdf供閱讀)**

1. **其他一大堆**

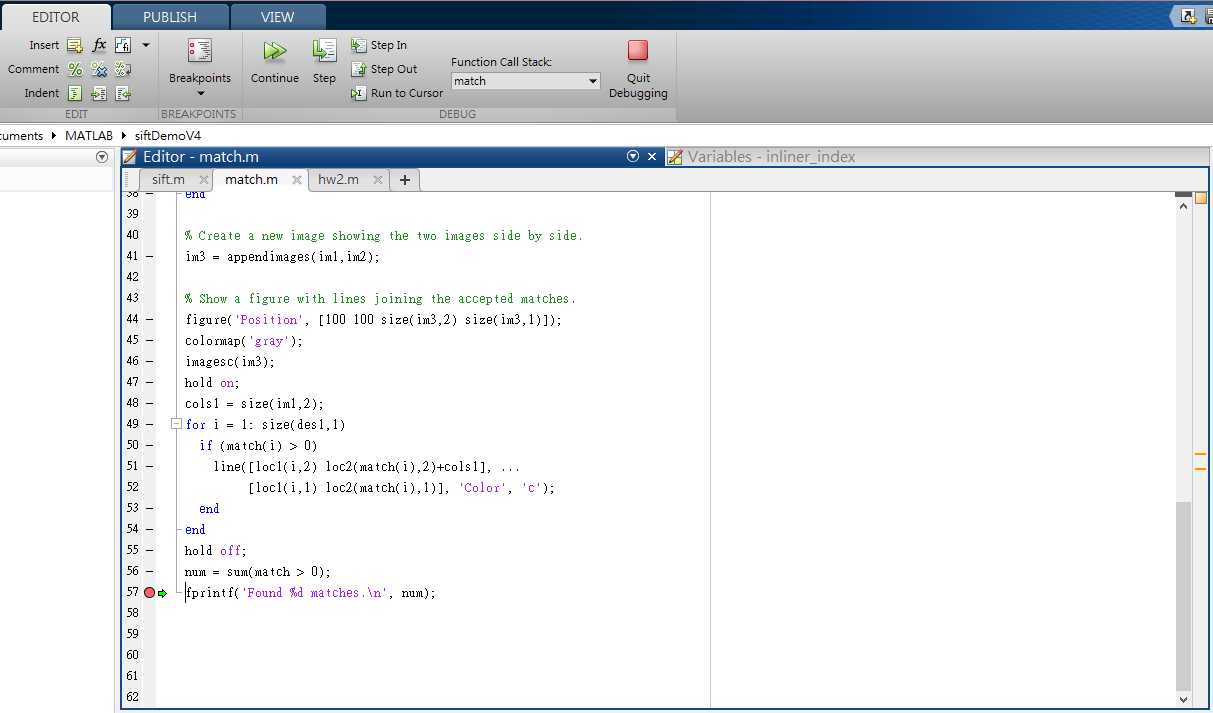
**(David Lowe的 SIFT matlab code and another thing…)**

**Technical discussion**

本次作業為image detection和warping的實現，利用的軟體為matlab，原因是因為對matlab比對OpenCV熟(掩面)

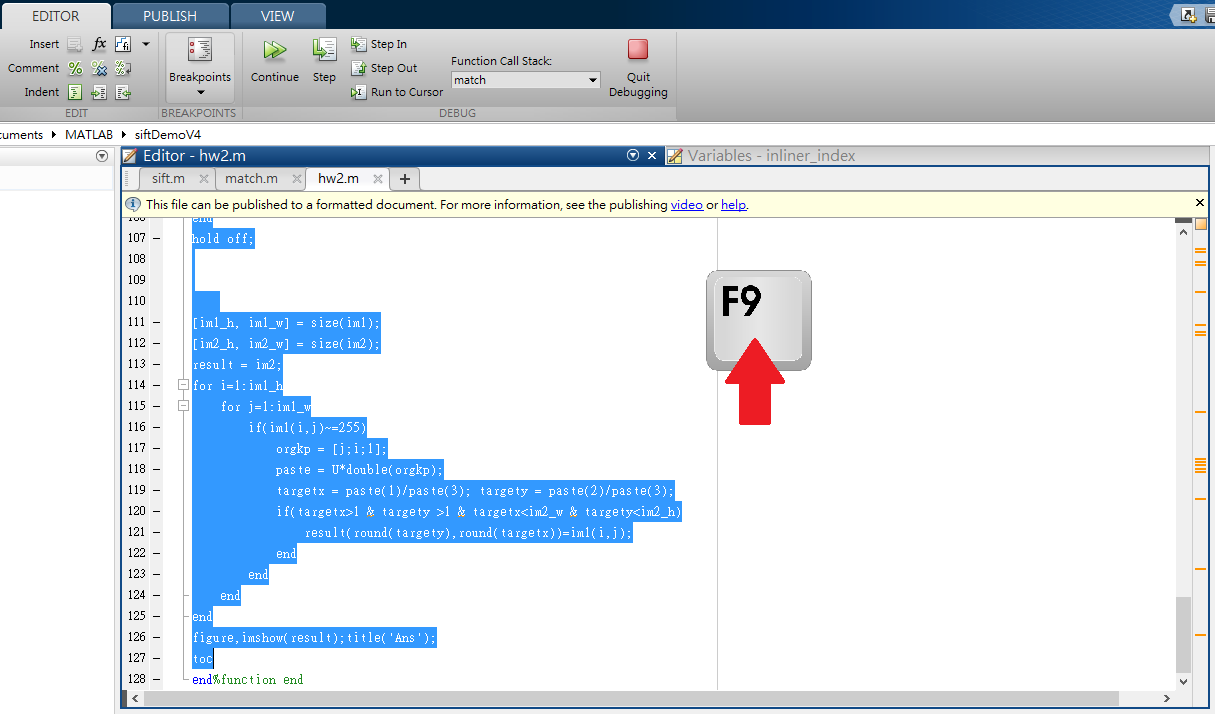
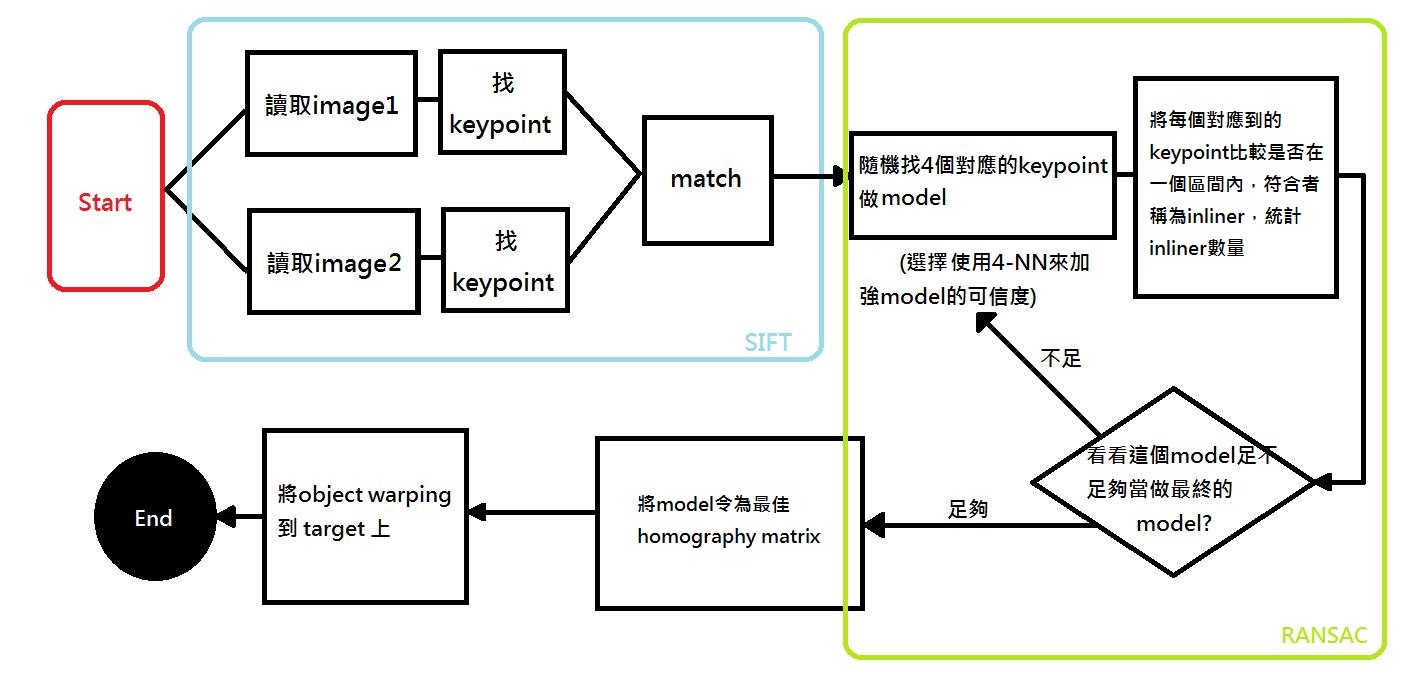
完成的項目有1. SIFT(套用David Lowe的code) 2. KNN 3. RANSAC 4. Homography matrix 5. Warping，以下將一一解說。

**How to Run?**

本次作業套用了David Lowe的code，但該code會在程式結束時無形中將佔存的變數清空，所以我在執行時在match.m的程式末標記了一個暫停點，如下：

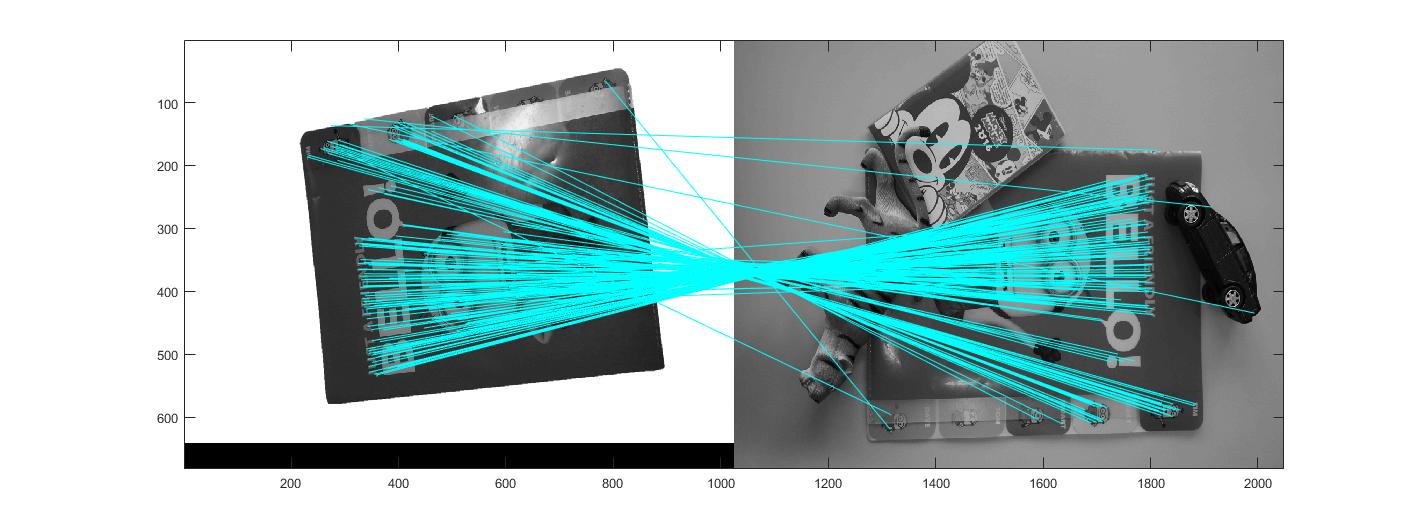
在command line下指令: match(‘object\_22.bmp’, ‘target.bmp’);

分別輸入object和target，執行後會卡在標記得暫停點，之後點開hw2.m執行全部程式碼(我是把function內全部框起來[tic & toc間]按f9)，如下：

**流程圖:**

1. SIFT

這部分使用David Lowe的matlab code (來源：<http://www.cs.ubc.ca/~lowe/keypoints/>)

會針對兩張圖中來找出對應最相近的的keypoint與該descriptor來配對，並畫出SIFT的連線配對，如下：

並且生成矩陣match分別對應左圖的點對應到右圖的哪個點(內容存放為index，需要到loc1與loc2做查表找到x,y軸位置)，由圖中可以看到SIFT仍會有對應不佳的點，會造成之後建model時可能造成”選錯點”的情形。

1. KNN

節錄程式碼：

random\_seed = randi([1,count\_index-1],1,4);%隨機找尋match中任意4點

random\_seed = sort(random\_seed);

NNpdist = [];

NN\_index = [];

for j = 1:4

for i = 1 : count\_index-1

NNdist = [loc2(match\_index(2,random\_seed(j)),2),loc2(match\_index(2,random\_seed(j)),1);loc2(match\_index(2,i),2),loc2(match\_index(2,i),1)];

NNpdist(i) = pdist(NNdist); %計算每一個點與其他對應的keypoint距離

end

[NNB NNI]= sort(NNpdist); %計算各點距離後依大小排序

NN\_index(j,1:4) = NNI(1:4); %取最短距離的四點(包含第一點為自己對應的點)

end

利用random\_seed隨機找尋match中的任意四點，將四點各自與其他keypoint correspondences比較距離，挑出最小的4個做為KNN的4個最近點，之後model將以4\*4\*4\*4種方法去chain model

1. Homographic matric & RANSAC

將前面KNN做出的model做成homographic matrix，依照講義找出 min( ||Ux|| ) ，找出U^T \* U的eigenvalue 與其 eigenvector，挑出最小的eigenvalue，將它的eigenvector排列成3\*3的矩陣當作model，之後要進行RANSAC。

將每個點丟進上面計算的matrix去看看對應到的點是不是跟原本的在附近(我設定歐式距離50以內判斷為inliner)，如果為inliner，計算inliner數量(inliner\_counter ++)，並記錄inliner對應到的位置在哪供我們後面檢查使用。

每一次的model計算inliner後都會比較目前inliner最大值，如超越最大值則將記錄全數更新。

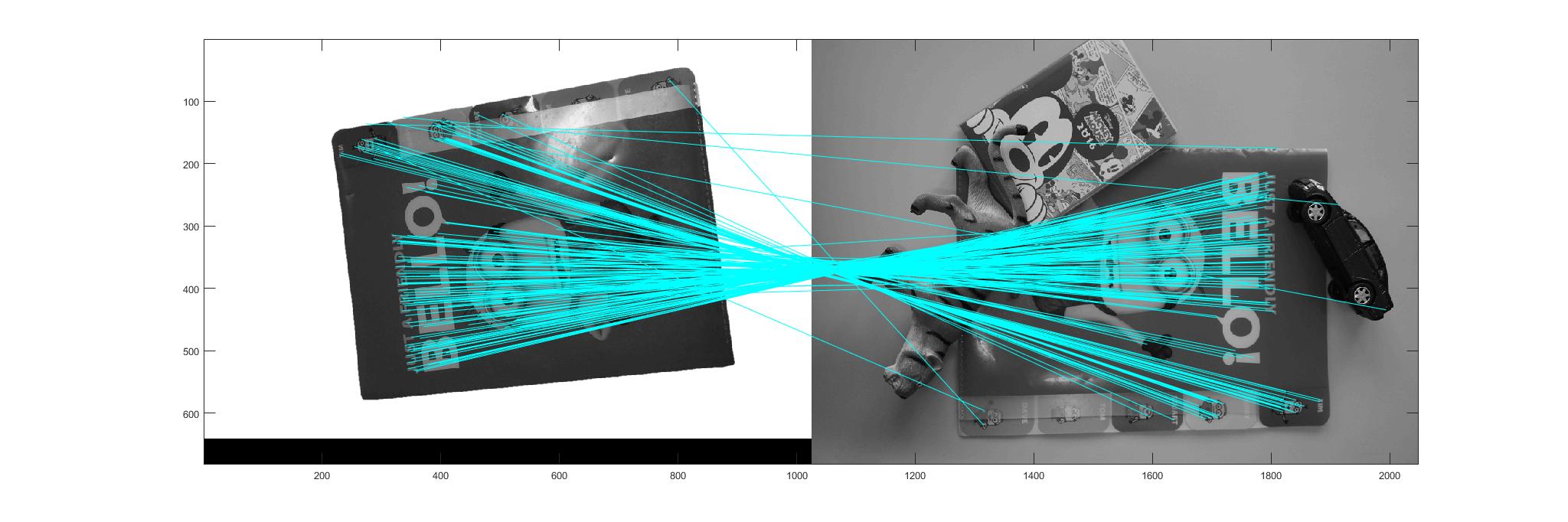
P.S. 如果途中發現有某個model計算inliner超過總共keypoint的7成，則直接判定它為最佳model，否則將重複執行2000次(所以最高計算model次數為2000\*4\*4\*4\*4)

4. Warping

最後我們會得到一個最佳的model，我們把原本的object中不等於255的點都丟進該model中，並將結果貼到target上。

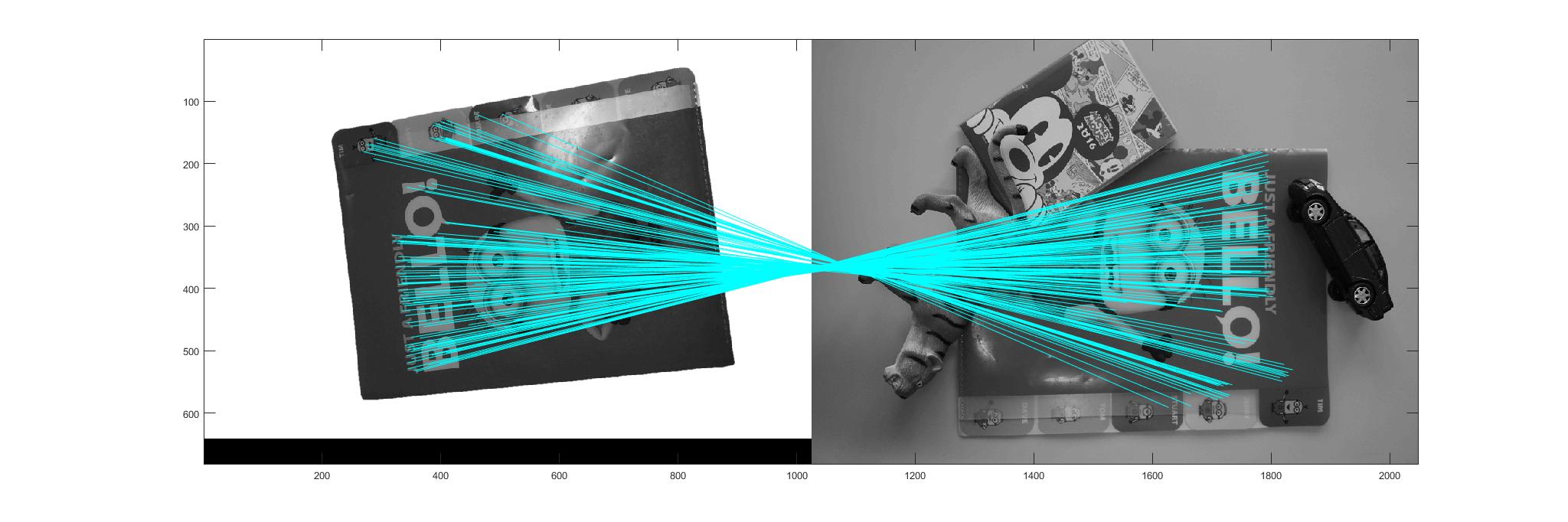
**Discussion of results**

以object\_22.bmp對應到target.bmp為例

SIFT後的結果 :

經過程式執行後可以找到229個inliner (總數246中佔229/246=0.93，高於9成的inliner)

，花費程式執行時間為26.421284 seconds

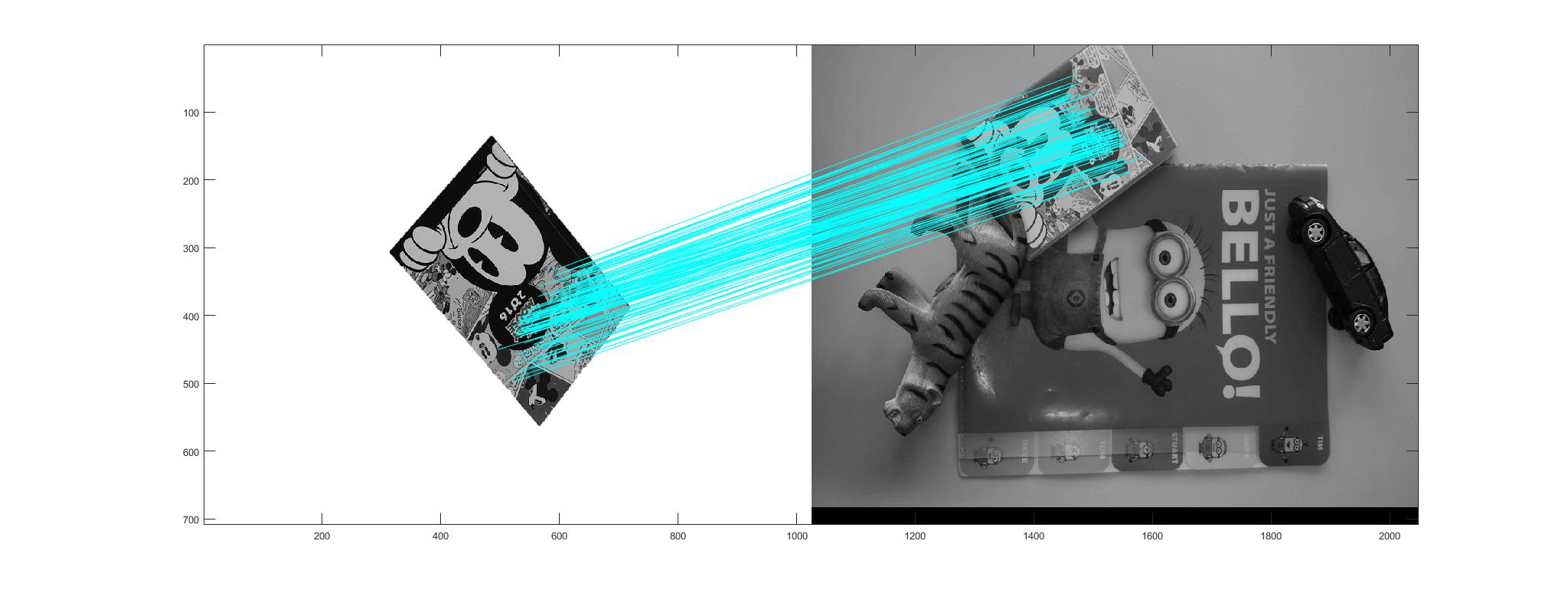
將對應的到點拿出來畫線檢查:

肉眼可看到錯誤的點都不會被選到，與我們預期的效果相同。

做Warping將object貼到target中：

還是稍有誤差，可能要再提高門檻求得更好的model。(實驗9成5的inliner結果與此結果差不多)

做第二個物體的比較: (object\_12與target)：

耗時: 超過20分鐘，跑完2000個iteration找到132個inliner(佔總數246個keypoint的132/246=0.536)

Warping的結果：

可以看到keypoint的點跟前一張比較，inliner確實又對到該點，只是我們肉眼看很明顯地知道它剛好對反了180度(所以上上一張圖只出現了米奇筆記本的上半部matching)，看來還是要支持超過7~8成的inliner會比較可信。或者多次執行看有沒有機會找到更好的點。

**Conclusion**

該改善的項目1. 加速演算法 2. 選擇更合理的挑選seed方法 3. 選擇其他algorithm

1. 如果演算法夠快說不定能夠跑完C(249,4)種方法，暴力硬解，雖然我覺得這不是一個很好的研究方向，比較對第二個方法感興趣。
2. 在做完SIFT之後可以找個方法預先挑出較合理的keypoint correspondences，可以提高選到好的點的機率進而做出更好的效果，但小弟資質不足，頻頻想出一些不合理的挑點方式until deadline(崩潰)。
3. 利用非SIFT的方式找keypoint correspondesnces，例如SURF等等。