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…)

2. result.dir

   (作業生成結果)

3. object_11.bmp~object_22.bmp+target.bmp

   (作業測資)

4. 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)

5. 其他一大堆

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

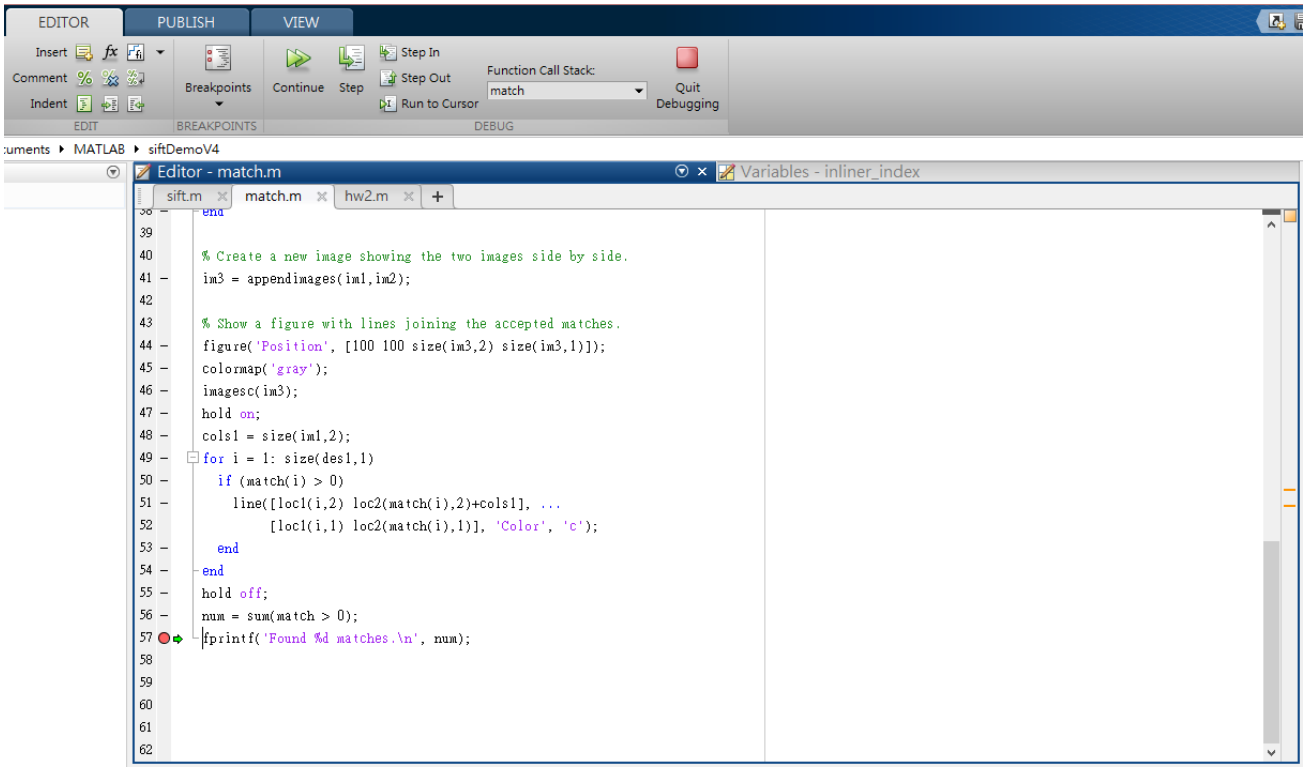
Technical discussion

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。

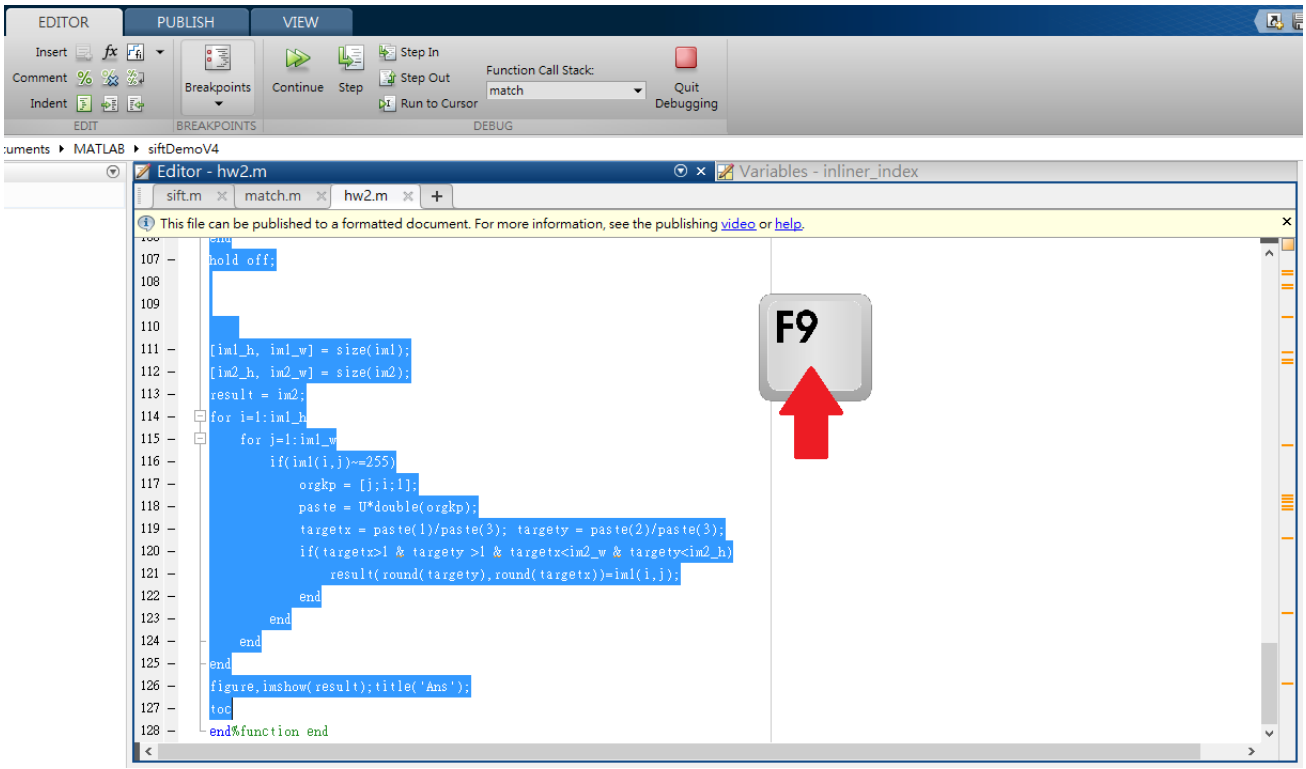
How to Run?

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：

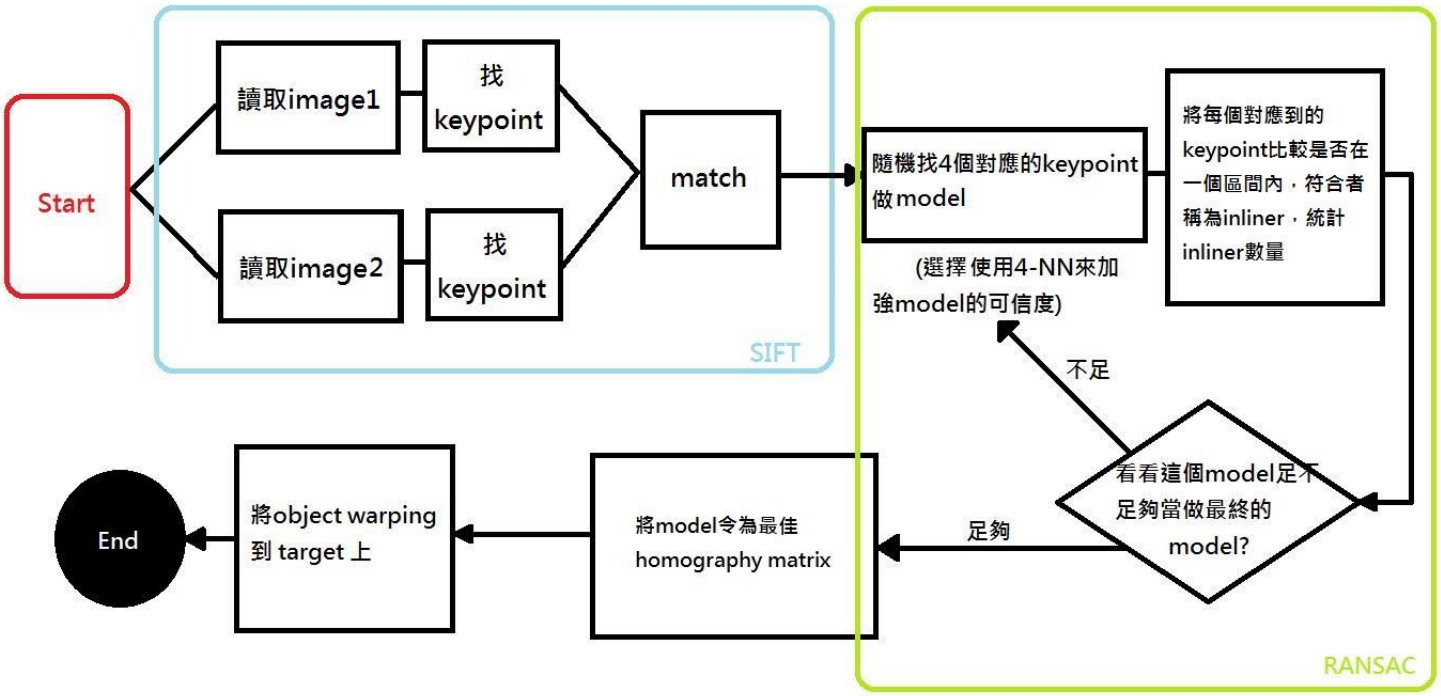


在 command line 下指令: match(‘object_22.bmp’ , ‘target.bmp’);

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：


EDITOR
PUBLISH
VIEW
Insert
Comment
Indent
Breakpoints
Continue
Step
Step In
Step Out
Run to Cursor
Function Call Stack:
match
Quit Debugging
EDIT
BREAKPOINTS
DEBUG
MATLAB ▸ siftDemoV4
Editor - hw2.m
Variables - inliner_index
sift.m
match.m
hw2.m
This file can be published to a formatted document. For more information, see the publishing video or help.
hold off;
[im1_h, im1_w] = size(im1);
[im2_h, im2_w] = size(im2);
result = im2;
for i=1:im1_h
for j=1:im1_w
if(im1(i,j)~=255)
orgkp = [j;i;1];
paste = U*double(orgkp);
targetx = paste(1)/paste(3); targety = paste(2)/paste(3);
if(targetx>1 & targety >1 & targetx<im2_w & targety<im2_h)
result(round(targety),round(targetx))=im1(i,j);
end
end
end
end
figure,imshow(result);title('Ans');
toc
end%function end
F9

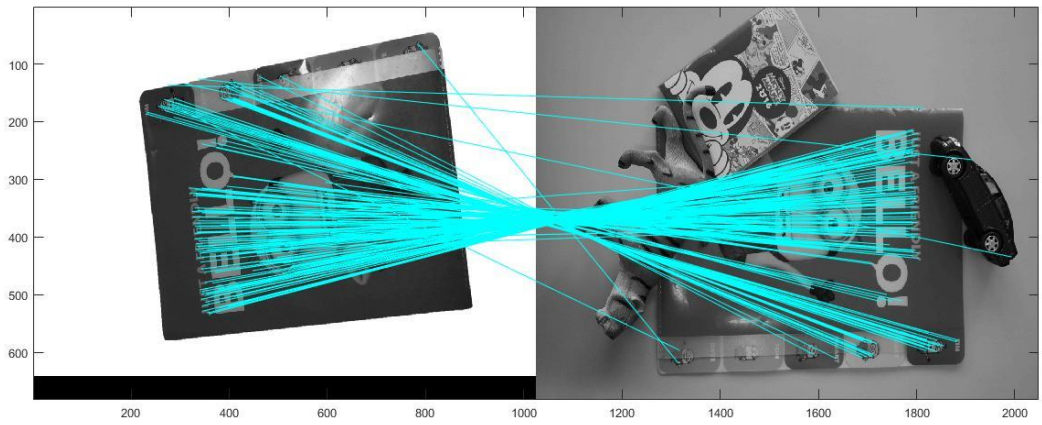
流程圖:


Start
讀取image1
找 keypoint
讀取image2
找 keypoint
match
SIFT
隨機找4個對應的keypoint做model
(選擇使用4-NN來加強model的可信度)
將每個對應到的keypoint比較是否在一個區間內，符合者稱為inliner，統計inliner數量
不足
看看這個model足不足夠當做最終的model?
足夠
將model令為最佳homography matrix
將object warping到 target 上
End
RANSAC

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。

2. 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

3. 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)

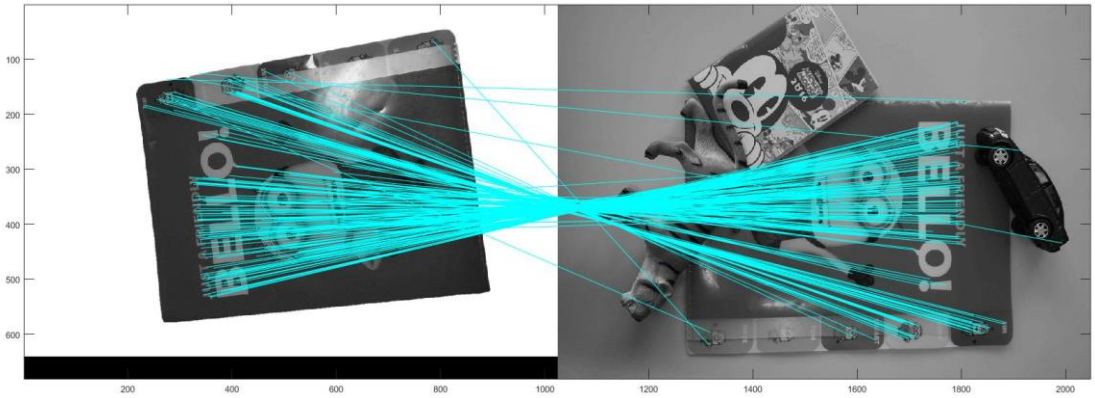
4. Warping

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

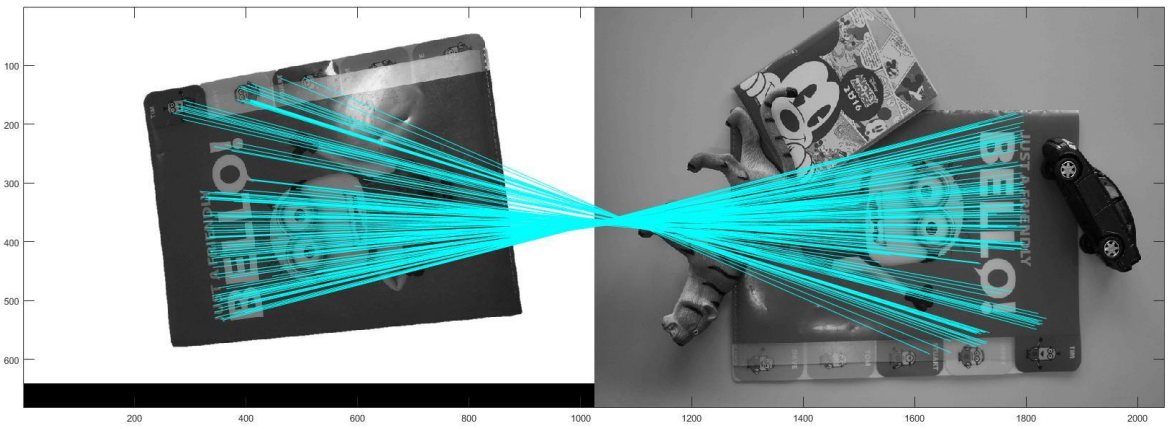
以 object_22.bmp 對應到 target.bmp 為例

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:



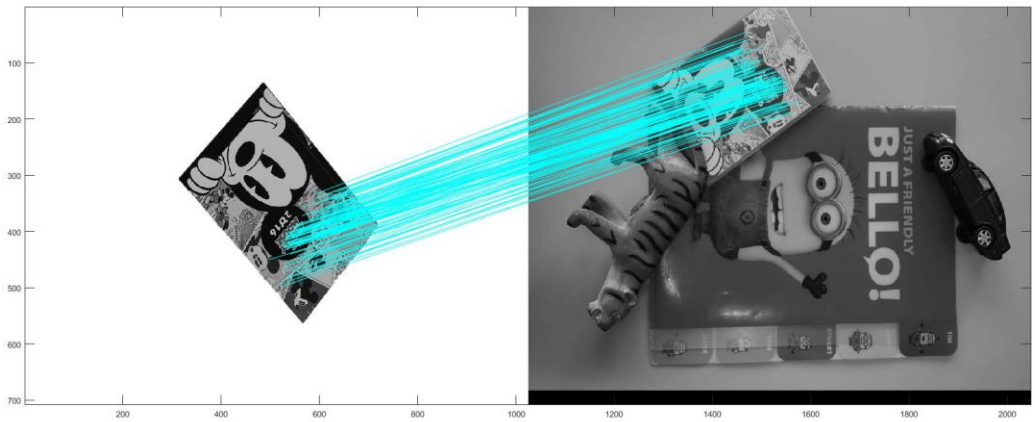
肉眼可看到錯誤的點都不會被選到，與我們預期的效果相同。

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：

Ans

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。