3DCV Hw1

R10922134 賴奕善

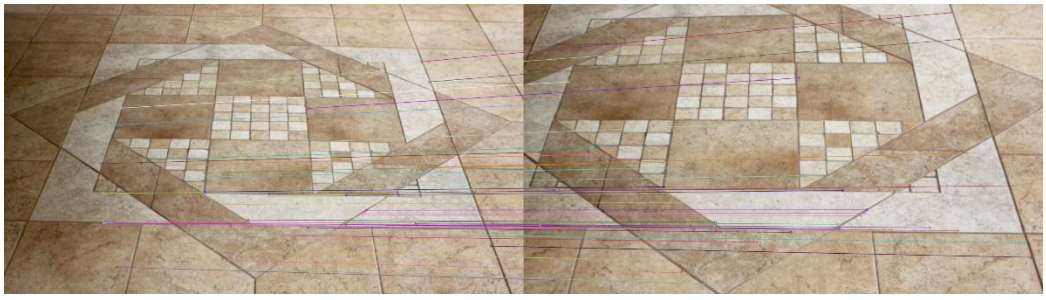
Problem 1:

- Screenshots: sample k correspondences1-0.png & 1-1.png
  - 1-0.png & 1-1.png
    - k = 4

    - k = 8

    - k = 20

    - k = 50

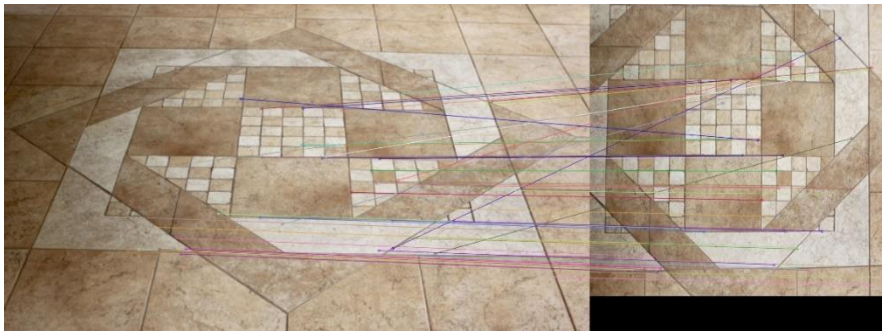


- 1-0.png & 1-2.png
  - k = 4

  - k = 8

  - k = 20

  - k = 50



- Error:

1-0.png & 1-1.png

k	4	8	20	50
Non-normalized	482.31	41.81	43.11	43.24
Normalized	171.57	22.63	22.48	22.44

1-0.png & 1-2.png

k	4	8	20	50
Non-normalized	495.01	681.62	674.87	728.38
Normalized	869.78	3450.74	1219.04	305

- Discussion & Observation:
  - Collinear:
    從 1-0.png & 1-1.png, k=4 的實驗中可以看出當三點共線或接近共線時，計算出的 homography matrix 會不準確，因為三點共線是互相 linear dependency 的。需要最少 4 組點才可以解 homography matrix，若有三點共線，共線的三組點只等價於兩組點，並不足夠解 homography matrix.
  - 1-0.png & 1-1.png vs 1-0.png & 1-2.png 比較:
    可以很簡單的發現 1-1.png 從 1-0.png 變化的幅度比 1-2.png 還要小，在看兩組之間 error 的差異，推論出當變化的幅度較大會導致計算出的 homography matrix 更不準確。
  - Normalization:
    從 1-0.png & 1-1.png 表格可以發現有經過 normalization 的 error 都有明顯降低(482 to 171 and 41 to 22)，證明對點做 normalization 是有效的。
  - 相同紋理(texture)的影響:
    當整張圖片有重複的結構，例如本圖有許多重複的正方形以及三角形，當圖片變化的幅度很大時，在找 match point 時會變得很困難，因為有很多特徵相似，但位置不同的點。因為 SIFT 沒辦法學到點的相對位置的資訊，所以在整張圖片有相同紋理時，找到 match point 的效果不佳。
  - 如何找到去除 outlier?
    可以使用 RANSAC，先隨機取 4 組點算 homography matrix，找出最多 inlier 的 4 組點，用所以的 inlier 算 homography ，可以有效去除 outliers.

Command:

Python 1.py [img1] [img2] [grountruth] [screenshot_dir_path] [norm]

[norm] : input “norm” if you want to do normalization, otherwise, ignore it.

1-0.png & 1-1.png

```
(3DCV) PS C:\Users\r1092\source\repos\homework1-jeffLAI123> python 1.py .\images\1-0.png .\images\1-1.png .\groundtruth_
correspondences\correspondence_01.npy .\screenshot\1\
sample 4 points
Error between groudtruth:  483.31558262554654
sample 8 points
Error between groudtruth:  41.81070953868666
sample 20 points
Error between groudtruth:  43.11068582983872
sample 50 points
Error between groudtruth:  43.243764195319606
```

```
(3DCV) PS C:\Users\r1092\source\repos\homework1-jeffLAI123> python 1.py .\images\1-0.png .\images\1-1.png .\groundtruth_
correspondences\correspondence_01.npy .\screenshot\1\ norm
sample 4 points
average_distance to origin:  1.3915979864100496
average_distance to origin:  1.6160170260362432
Error between groudtruth:  153.94631003704424
sample 8 points
average_distance to origin:  1.3763928853899714
average_distance to origin:  1.5856472517551008
Error between groudtruth:  21.305913198328952
sample 20 points
average_distance to origin:  1.4814714350504774
average_distance to origin:  1.6362672733690786
Error between groudtruth:  22.944970311946296
sample 50 points
average_distance to origin:  1.6062262064663084
average_distance to origin:  1.7194991882350157
Error between groudtruth:  24.491823001916636
```

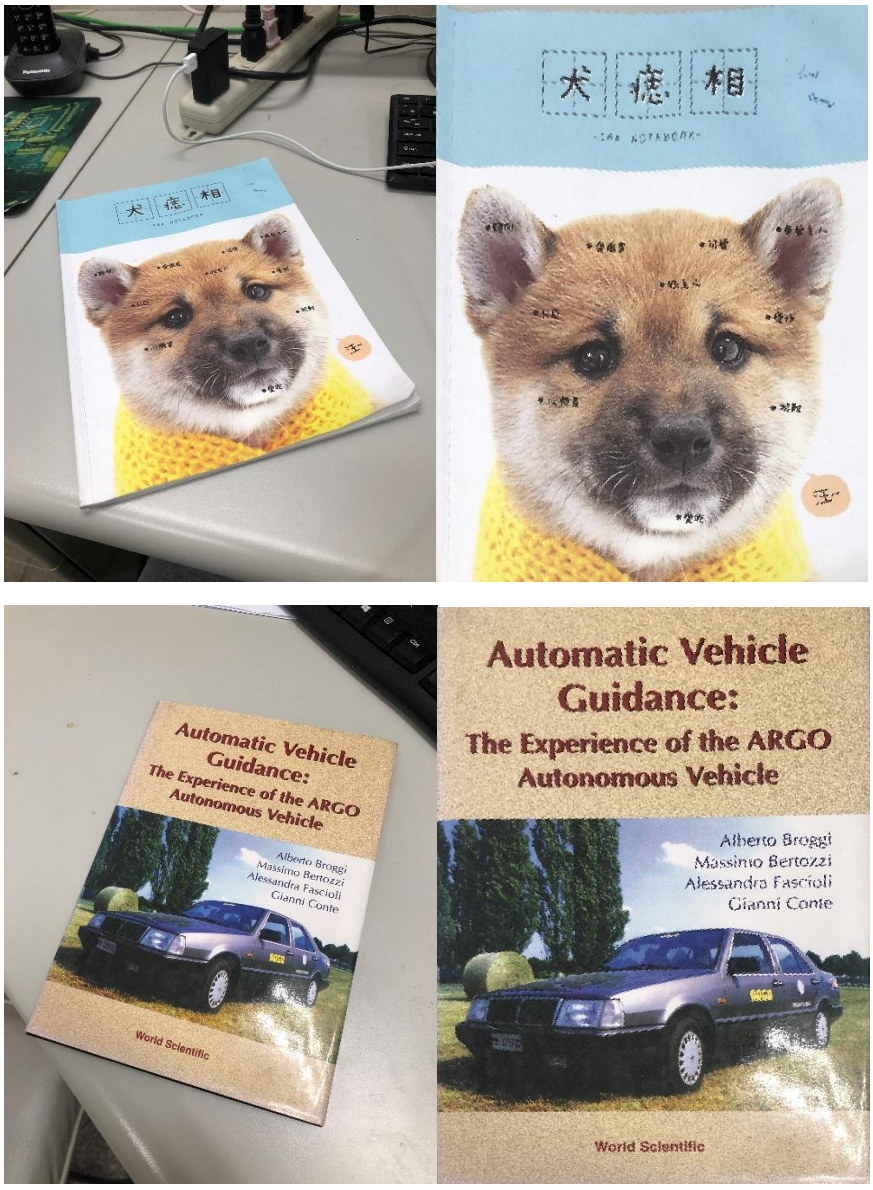
1-0.png & 1-2.png

```
(3DCV) PS C:\Users\r1092\source\repos\homework1-jeffLAI123> python .\1.py .\images\1-0.png .\images\1-2.png .\groundtrut
h_correspondences\correspondence_02.npy .\screenshot\2\
sample 4 points
Error between groudtruth:  495.0170522501813
sample 8 points
Error between groudtruth:  681.6264391761998
sample 20 points
Error between groudtruth:  674.8751078714266
sample 50 points
Error between groudtruth:  728.3846133917406
```

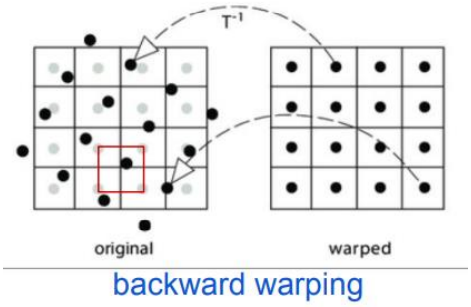
```
(3DCV) PS C:\Users\r1092\source\repos\homework1-jeffLAI123> python .\1.py .\images\1-0.png .\images\1-2.png .\groundtrut
h_correspondences\correspondence_02.npy .\screenshot\2\ norm
sample 4 points
average_distance to origin:  1.4380723136565892
average_distance to origin:  1.5639011274154617
Error between groudtruth:  869.7819075382763
sample 8 points
average_distance to origin:  1.0532516216142631
average_distance to origin:  1.6371723768444386
Error between groudtruth:  3450.742982990591
sample 20 points
average_distance to origin:  1.6675072251599992
average_distance to origin:  1.6898974227540147
Error between groudtruth:  1219.0414525115928
sample 50 points
average_distance to origin:  1.4044889171290498
average_distance to origin:  1.8060636341907659
Error between groudtruth:  305.80542981491647
```

Problem 2:

- Input & output



- Method:
  把書本的四個角由左上角開始，順時鐘依序選取，將 4 個角對應到新圖片的邊界四個角(`[0,0], [0,479] , [639,479], [639,0]`)，用這 4 組對應點計算 homography matrix (from source image to destination image). 最後將新圖片的每個 pixel 的座標乘上 homography matrix 的反矩陣，從原圖取值，產生轉正後的圖片。



- How to execute:
  Cmd: python 2.py [img]
  輸入上述 command ，在跳出的圖片中從左上開始順時鐘選取 4 個角，按下 Esc，即可得到轉正後的圖片。

Environment: see the 3DCV.yaml for conda env

- Python: 3.7.11
- cv2: 4.5.5