Computer Vision HW2 Report

作者：105062509 / 羅右鈞

註：此 report 原先在 dropbox paper 上編輯，直接至
https://paper.dropbox.com/doc/Computer-Vision-HW2-Report-NI2G0d4mwKCy1CRObwzSJ
就能看到更美的排版。

各題做法解釋

各題會用到的獨立函式，做法都在 README 有解釋過了，這部分在解釋的時候只會說明步驟跟用到哪些常獨立函式，並不會再重新解釋常用函式的部分。

Part 1-A

Write a program to compute the projection matrix from a set of 2D-3D point correspondences by using the least-squares (eigenvector) method for each image.

做法：

- 首先使用 clicker.m 手動選取兩張棋盤圖片上相對應的 2D, 3D 座標，分別存至 “points2D3D_1.mat”, “points2D3D_2.mat” 中
- 在程式中，讀取兩張圖的 2D, 3D 座標，並使用獨立函式 computeCameraMatrix 分別計算兩張圖的 3D 座標投影至 2D 座標的 camera projection matrix

結果：

- 第一章棋盤圖的 camera projection matrix:

  P1 =

  -9.9536e-02 3.2848e-02 -1.3797e-02 -6.2146e-01
  1.3029e-02 9.2668e-02 -1.8743e-02 -7.7039e-01
  -1.3425e-05 1.6180e-05 1.8184e-05 -5.1057e-04

- 第二章棋盤圖的 camera projection matrix

P2 =

-5.0363e-02 2.4935e-02 6.4323e-02 -7.9885e-01
-6.3572e-03 7.2696e-02 -1.3108e-02 -5.9080e-01
6.7901e-06 1.2102e-05 1.8078e-05 -4.8636e-04

Part 1-B

Decompose the two computed projection matrices from (A) into the camera calibration matrices, rotation matrices and translation vectors to obtain the camera intrinsic parameters (calibration matrix K) and extrinsic parameters (rotation matrix R and translation vector t) by using the Gram-Schmidt process.

做法：

- 讀取 part 1-A 的結果 (兩張圖的 camera projection matrix)
- 使用獨立函式 QR 對 camera projection matrix 做 RQ 分解，因為 RQ 分解跟 QR 分解差在一個是從最後一個 row 處理到第一個 row；另一個是從第一個 column 處理到最後一個 column，因此 input 需要做一些調整，最後 output 出來的 Q, R matrix 也需要做些調整才能分別對應到 rotation, intrinsic matrix
- 有了 intrinsic, rotation matrix，就能求出 translation vector，再將 rotation matrix, translation vector 組成 extrinsic matrix

結果：

- 第一張圖片的 camera matrix 分解成 intrinsic, extrinsic matrix 的結果
  Intrinsic1 =

  8.8278e-02 -5.9459e-04 5.8168e-02
  -5.2042e-18 8.8635e-02 3.5386e-02
  -6.7763e-21 3.3881e-21 2.7797e-05

Rotation1 =

-0.8070079 -0.0059659 -0.5905106
0.3398222 0.8131091 -0.4726251
-0.4829692 0.5820808 0.6541581

Translation1 =

5.0538
-1.3586
-18.3678

Extrinsic1 =

-0.80701 -0.00597 -0.59051 5.05379
0.33982 0.81311 -0.47263 -1.35858
-0.48297 0.58208 0.65416 -18.36775
0.00000 0.00000 0.00000 1.00000

- 第二張圖片的 camera matrix 分解成 intrinsic, extrinsic matrix 的結果

Intrinsic2 =

6.9780e-02 -9.3025e-05 4.9259e-02
-4.3368e-18 6.9315e-02 2.6313e-02
8.4703e-22 3.3881e-21 2.2790e-05

Rotation2 =

-0.932350 -0.016407 0.361185
-0.204819 0.847186 -0.490229
0.297947 0.531042 0.793235

Translation2 =

```
  3.61663
 -0.42192
-21.34136
```

Extrinsic2 =

```
 -0.93235   -0.01641    0.36118    3.61663
 -0.20482    0.84719   -0.49023   -0.42192
  0.29795    0.53104    0.79323  -21.34136
  0.00000    0.00000    0.00000    1.00000
```

Part 1-C

Re-project 2D points on each of the chessboard images by using the computed calibration matrix, rotation matrix and translation vector. Show the results (2 images) and compute the point re-projection root-mean-squared errors.

做法：

- 讀取圖片手點選取的 3D 座標，以及 part 2b 算出來的 intrinsic, extrinsic matrix
- 將 3D 座標多加一個維度為 1，轉成 homogeneous coordinate
- 計算新投影的 2D 座標 (predicted point) = intrinsic matrix * projection matrix * extrinsic matrix
- 此時新投影的 2D 座標還是 homogeneous coordinate，將第一個維度 (x) 跟第二個維度 (y) 除上第三個維度 (z)，就是真正的 2D 座標，最後用 plot 顯示結果，並且算出原始 2D 座標與新投影的 2D 座標之間的 RMSE。

結果：

- 黃點是手動選取的點；紅點是新投影上去的點
- 左圖 RMSE 結果：RMSE1 = 8.6262
- 右圖 RMSE 結果：RMSE2 = 9.5177

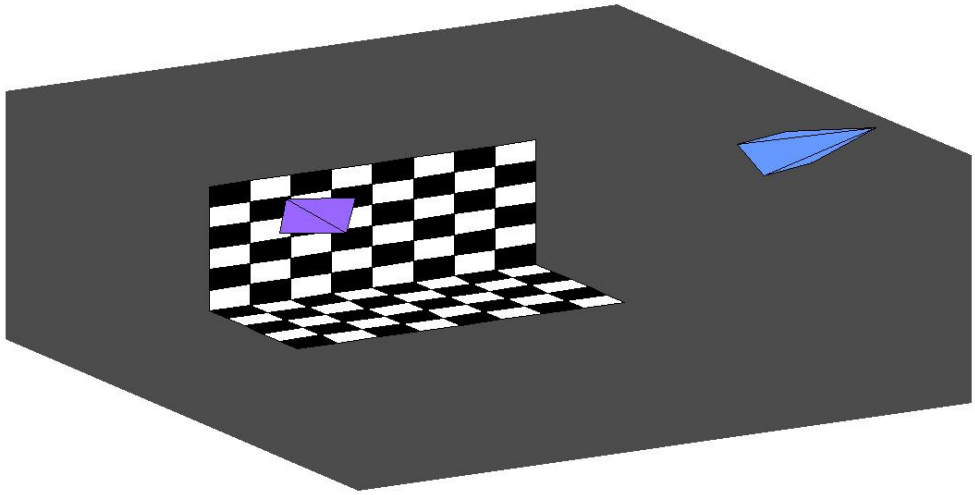
Part 1-D

Complete “visualizeCamera.m” provided by TA to plot camera poses for the computed extrinsic parameters (R, t) and then compute the angle between the two camera pose vectors.

做法：

- 此題先將 “visualizeCamera.m” 內的 cameraPoseVector, cameraPosition, angle 給算出來，然後再 “part_1d.m” 內呼叫 visualizeCamera()
- cameraPoseVector 其實就是將 world 相對於 camera 所朝的方向，也就是朝 x=0, y=0, z=1 的方向，乘上 rotation matrix 就能得到 camera 相對於 world 所朝的方向
- cameraPosition 就是 transpose(-inverse(R1)*t1)
- 兩台相機的 angle 就是 cameraPoseVector 之間的 cos 夾角
- 這題主要是參考 http://ksimek.github.io/2012/08/22/extrinsic/ 這篇文章所解釋的內容

結果：



Part 2-A

Show the homography H that maps points in the input image to the corresponding points in the left frame and right frame.

做法：

- 手動選取 Hillary (left frame), Trump (right frame) 畫像的四個 corner 點
- 使用獨立函式 computeHomography() 算出 left = H * right 的 homography

結果：

- right frame 2D 座標投影到 left frame 2D 座標的 homography
  H =

  -1.2729e-03 6.5229e-06 8.5521e-01
  -7.1388e-04 -8.3343e-04 5.1827e-01
  -1.7831e-06 4.2409e-08 4.6972e-04

Part 2-B

Warp the left photo (Hillary) to replace the right photo (Trump) and warp the original right photo to replace the left photo by using bilinear interpolation with backward warping. (1 image)

做法：

- 讀取原圖 “debate.jpg” 存至 img，再複製一張到 img_new，這是我們要編輯的圖
- 給定 frame 的四個 corner 座標，利用 roipoly() 取得 left frame 跟 right frame 內的所有座標，也就是我們的 target points
- 有了兩個 frame 內的所有座標後，再利用 homography 投影出新的 frame 內的所有座標，也就是我們的 projected points
- 利用獨立函式 backwardWrap()，求出 target points 所有的顏色，並畫在 img_new 上

結果：

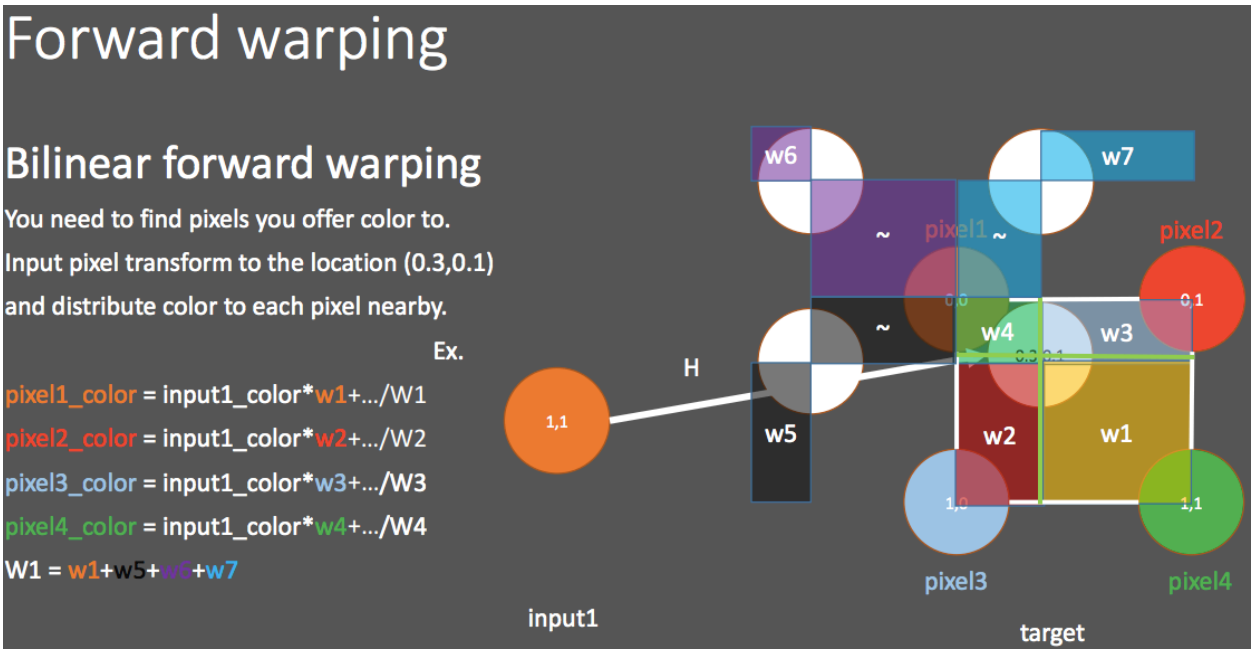
Part 2-C

Warp a new texture image (of your choice) to the two-sided wall by using bilinear interpolation with forward warping. (1 image)

做法：

- 手動選取兩面牆的四個 corner 座標
- 分別計算 texture 圖片上的四個 corner 座標投影到兩面牆的四個 corner 座標的 homography
- 再利用 homography，算出所有 texture 圖片上所有 pixel 的座標投影到兩面牆上的座標，在程式中命名為 projected points
- 跑個 for loop 去 scan 所有的 projected points，將 projected point 相對應在 texture 上的 pixel 顏色利用 bilinear interpolation 的公式，權重性地將顏色分配到它周圍四個 pixel，在這邊要注意的是，每個 pixel 可能會從很多 projected point 得到顏色，因此 RGB 有可

能會超過 1，因此需要做 normalization，也就是將某個 pixel 除上由其他 projected points 分配給他顏色時所乘上的權重加總，如下圖所示：



- 實作細節中，我們是將兩面牆的顏色記錄到另一張跟原圖 size 一樣的 3D matrix－wall，這會是一張只有兩面牆，剩下背景都是黑色的圖，然後在上述 for loop 的過程中，我們也有宣告了一個與原圖 size 一樣的 2D matrix－wall_mask，紀錄兩面牆所有的 pixel 座標
- 將原圖乘上 wall_mask (有牆的地方為 0, 沒牆的地方為 1)，即可得到一張沒有牆的圖片
- 將這張沒有牆的圖片加上 wall，即可得到把 wall 貼在原圖上的圖片，而這個時候 Hillary 跟 Trump 也被蓋掉了（他們在上一步時就被挖掉了）
- 我們拿 part 2B 時拿 roipoly() 算的 hillary_mask, trump_mask，以同樣原理，將目前我們編輯的圖乘上這兩個 masks（有 frame 的地方為 0，沒 frame 的地方為 1）之後，就可以得到一張沒有 Hillary, Trump 的圖（準確來說應該是把兩個 frames 所有的 points 都給挖掉了）
- 另外我們再拿原圖分別乘上這兩個 masks（有 frame 的地方為 1，沒 frame 的地方為 0）之後，就可以得到一張只有 Hillary 跟一張只有 Trump 的圖，程式中命名分別為 hillary, trump
- 最後將 hillary, trump 直接加回目前我們編輯的圖，就是最後結果了

結果：

Part 2-D

Discuss the results of the two warping methods (forward and backward) you implement above.

首先，backward 的精神是拿 target point/pixel 投影回原本所在位置的 projected point 的周邊四個 pixel 經過內插後而得到 color，也就是說我們可以確保 target point/pixel 只會從四個 pixel 得到 color；forward 的精神是拿原本所在位置的 pixel 投影到 projected point，然後再將顏色分配給 projected point 它周邊四個 pixel，但是不像 backward，一個 pixel 是只會從四個 pixel 得到 color，而反觀 forward 的方式，一個 pixel 如果落在很多其他 projected points 的周邊，就會被分配到過多的顏色，因此才需要做 normalization。另外值得注意的是，投影的起始範圍跟終點範圍，希望 pixel 的數量最好是能夠一致的，或是貼圖要貼在某個地方時，貼圖的尺寸至少要跟某個地方的尺寸一樣大或更大，因為如果假設要將 1000 個點投影到 10000 個位置，就會造成有些點沒有顏色，導致一堆洞洞的結果。

而 forward 之所以叫 forward，backward 之所以叫 backward，是因為：

- forward method 取得顏色的方式是，我們事先就有顏色了，再把顏色分配給投影點四周的 pixels
- backward method 取得顏色的方式是，我們的顏色必須回去從投影過來的點，它四周的 pixels 所內差出來的顏色