

README.md

# 影像拼接

## 自行計算 Affine Transformation, Bilinear Interpolation 與 Linear Blending

本文與程式上傳至GitHub，連結 QR Code



### 原始影像

輕井澤王子飯店前

左



右



## 拼接結果

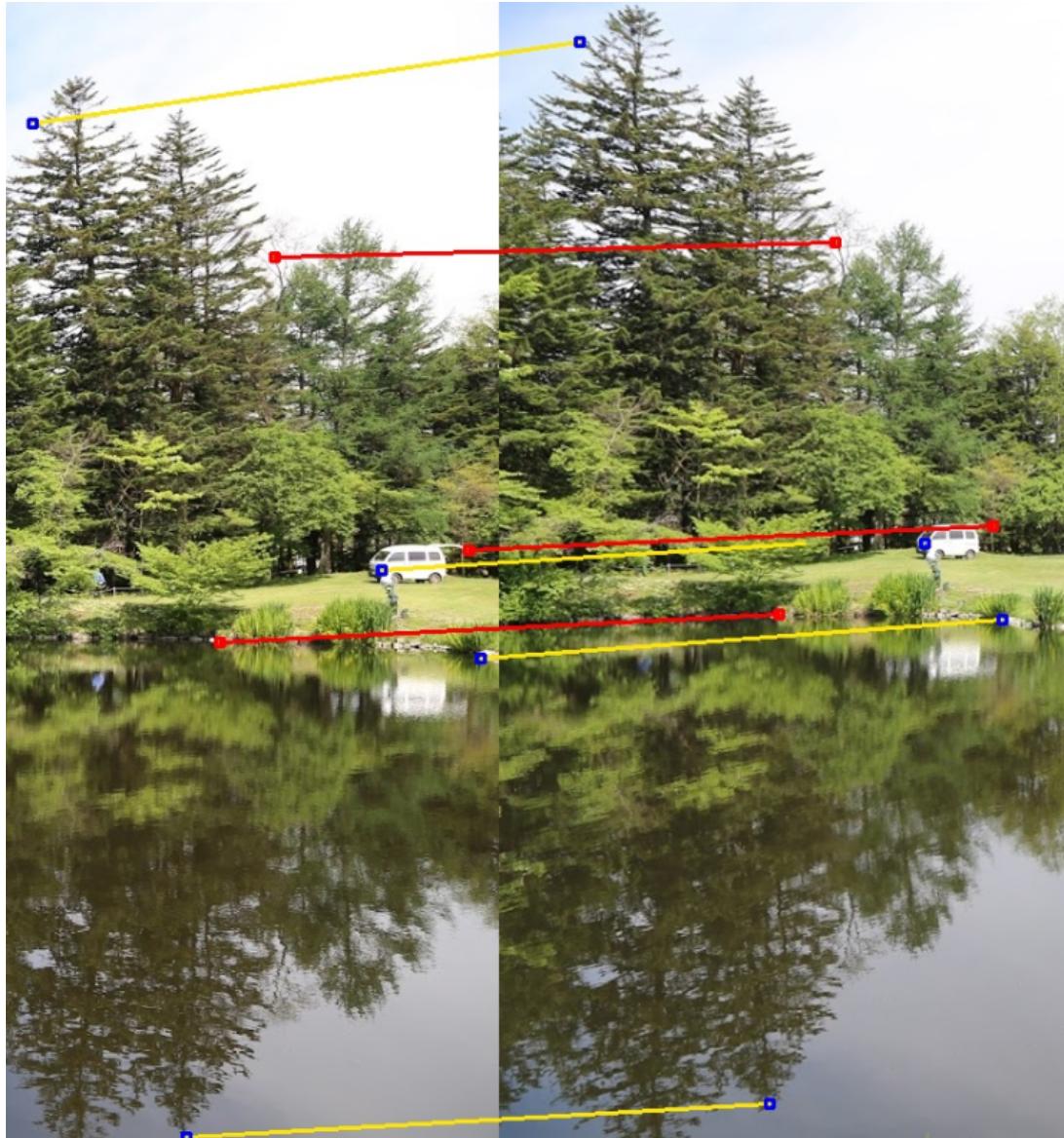


## 手動人工在兩張照片中選擇 7 個特徵點

		(左)IMG_1265.JPG	(右)IMG_5732.JPG	A 組合	B 組合	C 組合	D 組合
1	車尾	(1257, 411)	(369, 393)	V	V	V	
2	石頭	(1071, 480)	(209, 459)	V			
3	樹枝Y	(1112, 192)	(251, 181)	V			
4	樹稍	(931, 92)	(60, 31)		V		V

		(左)IMG_1265.JPG	(右)IMG_5732.JPG	A 組合	B 組合	C 組合	D 組合
5	樹頂倒影	(1046, 850)	(202, 825)		V		V
6	右邊石頭	(1266, 492)	(376, 463)			V	V
7	車燈	(1192, 426)	(318, 406)			V	

每三點可決定一個 Affine Transformation，進行四個組合測試，看哪個組合拼接效果比較好



## Affine Transformation

### 座標轉換

$(x, y)$  乘以座標轉換矩陣，得到新的空間座標  $(x', y')$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

要解  $a, b, c, d, e, f$ ，需要 6 個關係式，而三個點會有 6 個式子。可再整理成下面的聯立方程，然後解出 6 個未知數。

以組合A為例：

$$\begin{bmatrix} 369 \\ 393 \\ 209 \\ 459 \\ 251 \\ 181 \end{bmatrix} = \begin{bmatrix} 1257 & 411 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1257 & 411 & 0 & 1 \\ 1071 & 480 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1071 & 480 & 0 & 1 \\ 1112 & 192 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1112 & 192 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$$

numpy.libalg package 可以解線性方程

```
target = np.array(
    [[1257, 411, 0, 0, 1, 0],
     [0, 0, 1257, 411, 0, 1],
     [1071, 480, 0, 0, 1, 0],
     [0, 0, 1071, 480, 0, 1],
     [1112, 192, 0, 0, 1, 0],
     [0, 0, 1112, 192, 0, 1]]) #左圖座標
src = np.array([369, 393, 209, 459, 251, 181]) #右圖座標

param = np.linalg.solve(target, src) # 解聯立方程 - 求出座標轉換矩陣
print(param)
```

解出  $[a, b, e, d, e, f] = [8.51061314e-01 -2.46752991e-02 3.42931473e-03 9.65765979e-01 -6.90395771e+02 -1.78981257e+01]$

然後，我們可以用這個座標轉換矩陣來計算左圖的某一點，是右圖的哪一點。使用 numpy.matmul() 矩陣乘法即可。

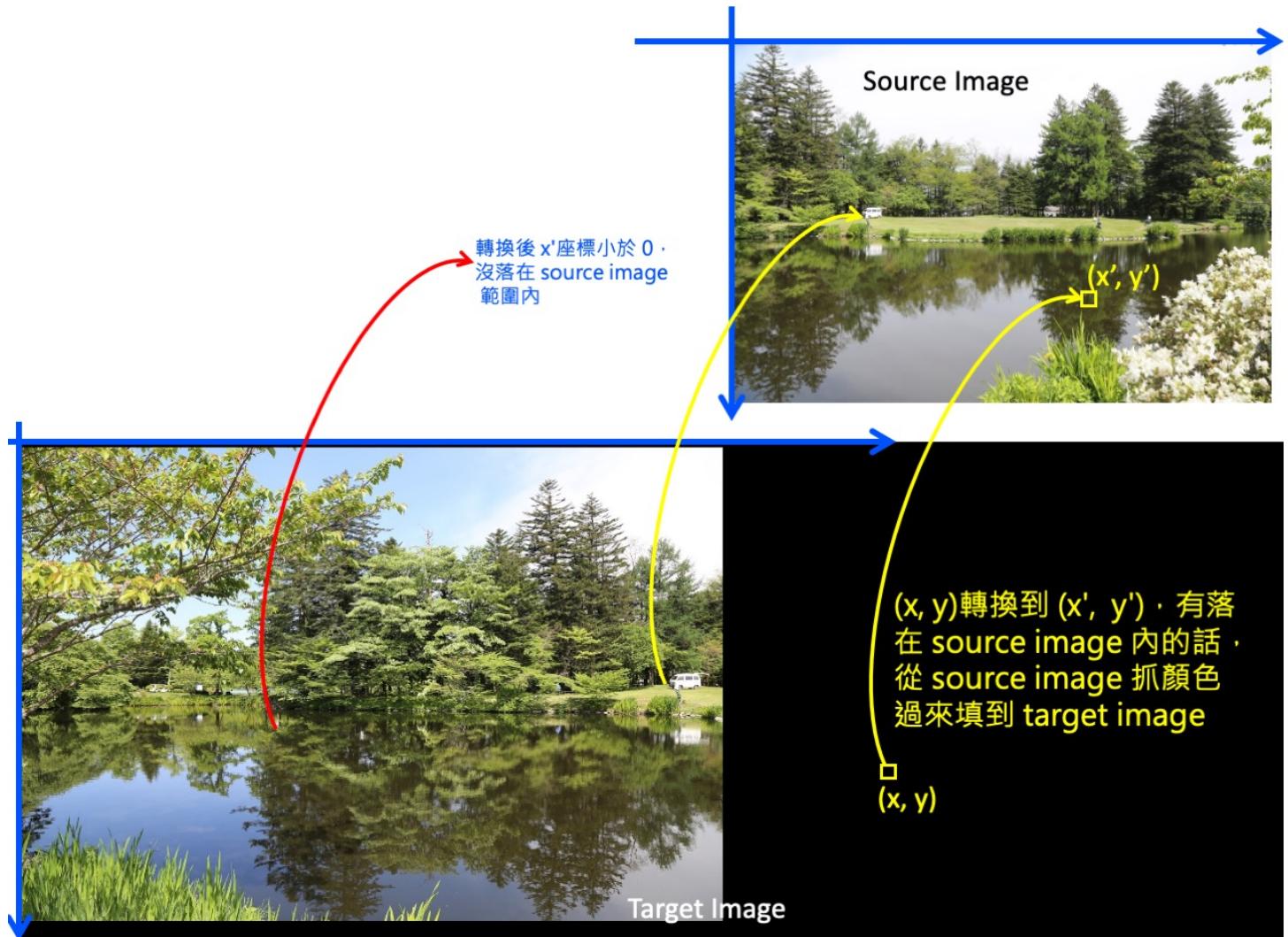
```
target_point = np.array([[1257, 411, 0, 0, 1, 0], [0, 0, 1257, 411, 0, 1]])
source_point = np.matmul(target_point, param) # 計算轉換後之座標, [369, 393]
```

## Inverse Mapping

先做一張大一點的黑色影像，把左邊(pic1)填進來。先叫它 target。之後我們用 Inverse Mapping 方式把右邊填上。



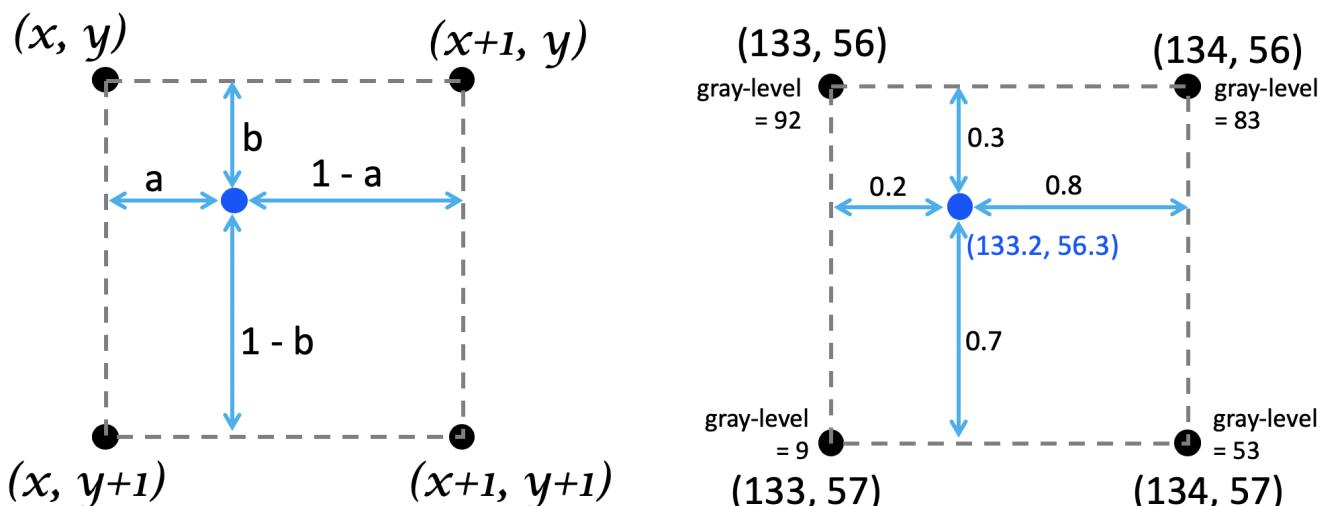
Inverse Mapping 指由要合成照片的空間(target)上的每一個點，去算它在 Source (pic2) 座標系的哪個位置。如果可以落在 source 的影像範圍內，就取它的顏色過來填到 target 上。可以避免 Forward Mapping 時 target 有空隙沒填到顏色。



## Bilinear Interpolation

由 target 空間轉換到 source 空間時，算出來的座標會是小數。它被四個點圍繞。

這時，到底要取 source 座標的哪個點的顏色來填到 target，可以用 Bilinear Interpolation 方法。



$$\text{bilinear\_color} = f(x, y)*(1-a)*(1-b) + f(x+1, y)*a*(1-b) + f(x, y+1)*(1-a)*b + f(x+1, y+1)*a*b$$

假設 target image 某一點座標轉換後，落在 source image 的 (133.2, 56.3)，則要填回的 target image 的顏色等於

```
# target_color = 左上角鄰居顏色 * 左上角權重 +
#   右上角鄰居顏色 * 右上角權重 +
#   左下角鄰居顏色 * 左下角權重 +
#   右下角鄰居顏色 * 右下角權重

target_color = 92 * 0.8 * 0.7 +
  83 * 0.2 * 0.7 +
  9 * 0.8 * 0.3 +
  53 * 0.2 * 0.3
```

## Linear Blending

---

交集處全用 Source Image 顏色蓋上去。

拼起來後，交界處很糟糕！接的很不順。





交集處，左圖顏色、右圖顏色各乘 0.5，再相加。好一點。



Linear Blending: 交集處，若點靠近左圖，左圖顏色的權重就取大一點，若靠近右圖，右圖顏色的權重就取大一點。

Linear Blending 好很多。交界處看不太出來了，雖然還有些鬼影。





## 拼接結果，裁切掉黑色部份

