影像處理 - 合併照片

黃凱鴻 5105056017

簡述

本次的目的是將兩張略有差異,但仍有重疊之處的照片,合併成同一張。這兩張照片之間的關係可能會有平移及旋轉。為了簡化計算量,不考慮視角的差異,意即視角都定為無窮遠處。在找出兩張照片的對應關係後,再利用 Bilinear 演算法,得到對應位置的像素值,最後得到合併後的新圖片。

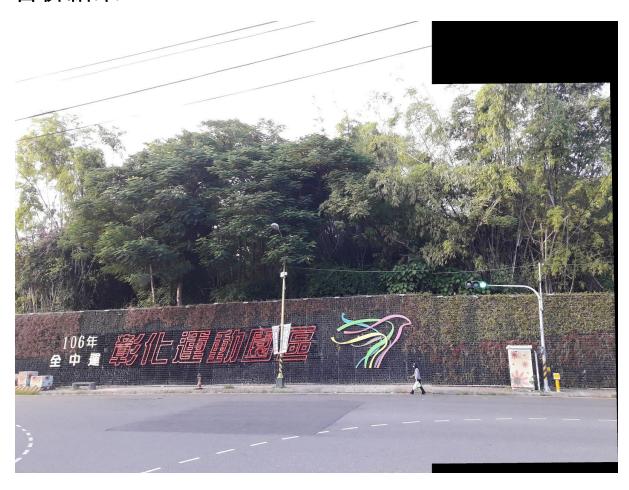
原始圖片

來源:自己手動拍攝 位置:彰化高中旁圍牆





合併結果



程式解說

開發語言及套件

Python PIL numpy

1. 挑選對應位置

在挑選時,有特意選取範圍較大的區域,以減低誤差。

```
img1 = Image.open("pic_left.jpg")
img2 = Image.open("pic_right.jpg")

### 在兩張影片上手動找出三個相同物品的pixel位置
pList1 = [(2779,2747),(1139,2173),(1772,1389)]
pList2 = [(1651,2357),(55,1758),(692,965)]
```

2. 找出兩組座標之間的對應 function

將點座標拓展為 3x3 的 matrix,令第一組為 A, 第二組為 B,則 Ax = B。利用 numpy 的 solve 方法,找出 A^{-1} ,則 $x = A^{-1}B$ 將其包裝為一個 function,可以輸入任意一座標 p = (a, b),得到 $p \cdot x$ 的解,也就是 pList1 內的座標可以轉換到 pList2。

```
def createTransform(pList1,pList2):
### 產生transformation function
primary = np.float32(pList1)
secondary = np.float32(pList2)

n = primary.shape[0] # dim
pad = lambda x: np.hstack([x, np.ones((x.shape[0], 1))]) #增加維度成 NxN Matrix
unpad = lambda x: x[:,:-1] # 把維度降回
X = pad(primary)
Y = pad(secondary)

A = np.linalg.solve(X,Y)

transform = lambda x: unpad(np.dot(pad(x), A))
return transform
```

除了list1 => list2 的 transform function之外,再建一個反向的 inverse function,用途是 list2 => list1 的轉換,在後面會說明。

```
transform = createTransform(pList1,pList2)
invers = createTransform(pList2,pList1)

### 測試用
transform(np.float32([[2779,2747],[1651,2357]]))

### 測試用
invers(np.float32([[2008,2285],[967,2439]]))
```

3. Bilinear

定義一個function 叫 bilinear,用途是輸入一組座標以及一張圖片,回傳該座標在圖片上的 RGB,若是超出圖片範圍則回傳 (0, 0, 0)。取法是使用 bilinear 演算法,取最近的四個點做線性內插。

```
def bilinear(targetTuple,rgb_im):
    x = targetTuple[0]
    y = targetTuple[1]

if (x < 0 or x >= rgb_im.width or y < 0 or y >= rgb_im.height):
        return (0,0,0)
    if(x==rgb_im.width-1 or y==rgb_im.height-1):
        return rgb_im.getpixel((x,y))

x1 = int(x)
    x2 = min(x1+1,rgb_im.width-1)
    y1 = int(y)
    y2 = min(y1+1,rgb_im.height-1)

x11 = np.float32([(x-x1)*p for p in rgb_im.getpixel((x1, y2))]) + np.float32([(x2-x)*p for p in rgb_im.getpixel((x1, y1))])
    #print(x11)
    x22 =np.float32( [(x-x1)*p for p in rgb_im.getpixel((x2, y2))]) + np.float32([(x2-x)*p for p in rgb_im.getpixel((x1, y2))])
    #print(x22)
    y12 = (y-y1) * x22 + (y2-y) * x11

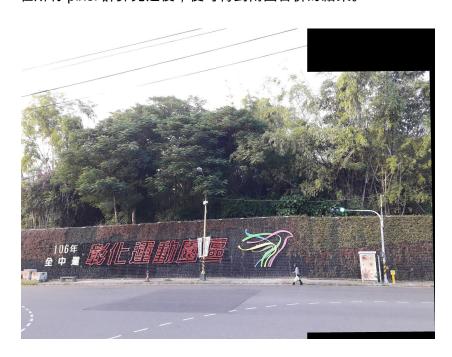
    return y12
```

4. 計算並輸出合併圖片

先用前面提到的 inverser function,計算第二張圖的邊界會落在哪裡。並跟第一張圖的邊界做比較,以得到合併之後圖片大小(取最大值)。

假設合併後圖片長寬為 w,h,則可對 w * h 的所有 pixel 做計算。若是 pixel 落在第一張圖內,為了減少計算量,便直接使用第一張圖的 RGB,反之則計算經過 transform 之後的座標 (i,j)。(i,j) 便是第二張圖上對應的座標點,再使用 bilinear 來得到其 RGB值,填入新圖片。

在所有 pixel 計算完之後,便可得到兩圖合併的結果。



問題檢討

Q:接縫處不完美

A:在第二步挑選對應位置時,是人工尋找的,會有誤差,無法保證是真實物體上的同一點。 這個誤差會隨著數值放大而增加(線性關係),兩張照片的視角差也不同。

再加上照片上並非所有位置都適合尋找對應,如天空對人眼來說都是藍色,樹葉可能被風吹動而有微小的偏差,能尋找的只剩下下方的固定物體。於是最後得到的 transformation 便存在了誤差,導致在合併時找到的 pixel 並非最正確的。