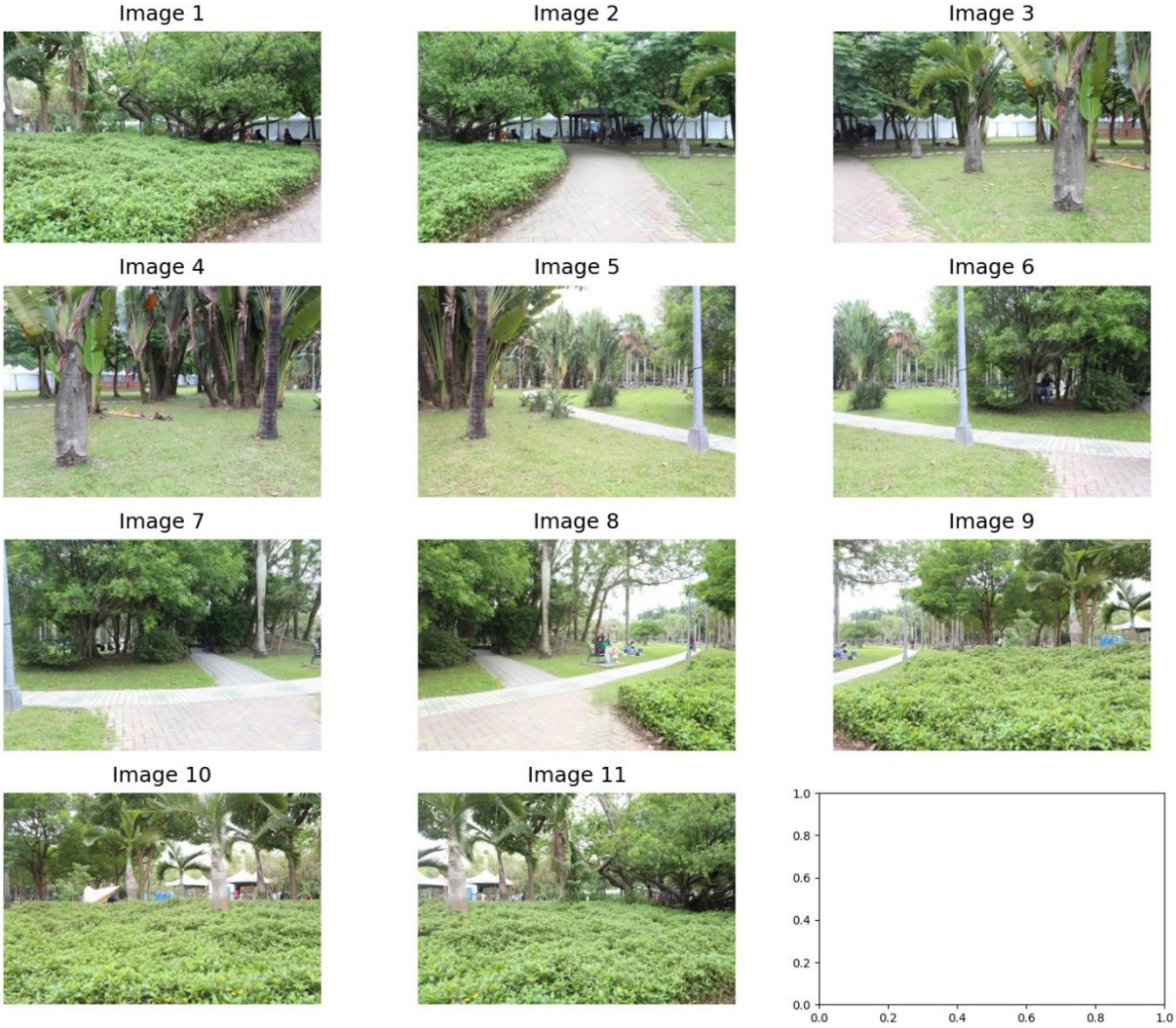
Digital Visual Effects - Project 2

第 12 組 – 方郁婷 張婷淇

April 29, 2023

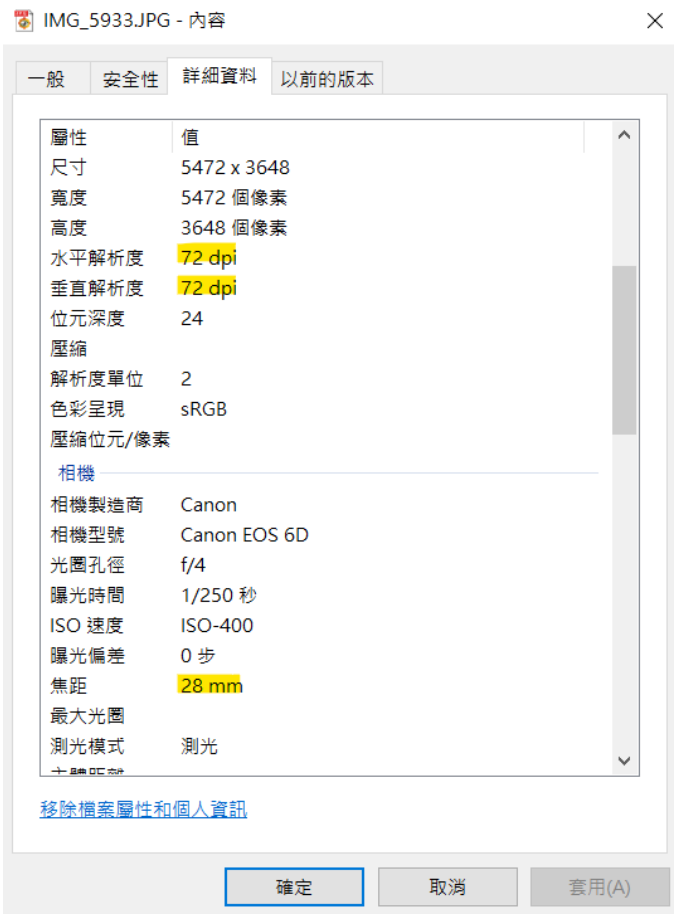
Introduction

以下是我們拍攝的照片(大安森林公園)



圓柱投影 (Cylindrical projection)

我們使用老師課程網站上提到的 autostitch 計算焦距，但算出來的焦距不是很準(太小了)，我們後來查看照片的詳細資料，藉由解析度與焦距推出我們的焦距為 28*72=2016



Steps:

1. 我們會先取得原始影像的高、寬、通道數，然後初始化一個與原始影像大小相同、像素值為 0 的圓柱投影影像
2. 建立原圖 x, y 座標矩陣，並將座標系統轉換為極座標
3. 根據極座標計算在圓柱投影座標系統上的 x, y 座標
4. 轉換回原始影像座標系統
5. 檢查邊界，並將像素資訊填入圓柱投影影像
6. 找出影像中最左邊與最右邊的點，並將圓柱投影影像周圍的黑邊去掉

Feature detection: Harris Corner Detection

Step: (根據 PPT 的步驟)

1. 先對圖片做 GaussianBlur 降噪
2. 取圖片 x 方向、y 方向的微分 Ix、Iy
3. 依照 PPT 的公式算出 Harris matrix，使用 3x3 的 Gaussian kernel 作為 window 取 sum

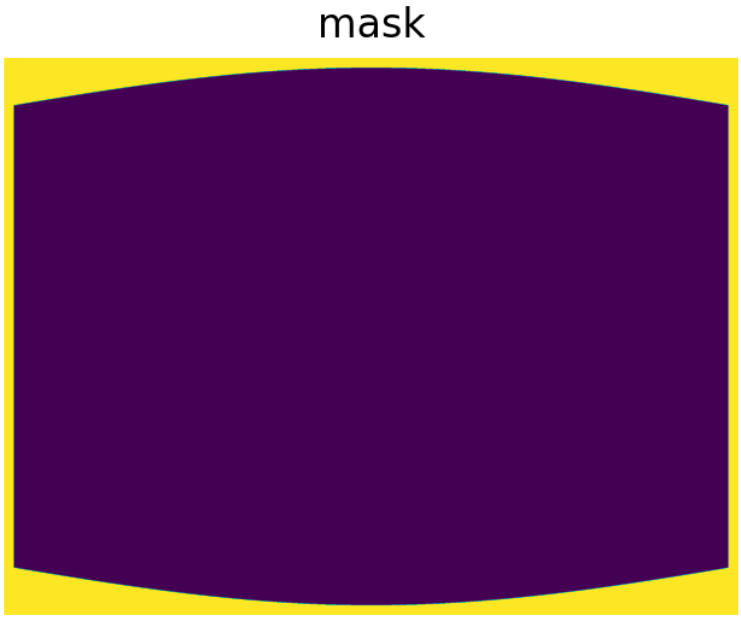
$$\mathbf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

4. 藉由 PPT 的公式計算 R，其中 k 的部份我們是取 0.05

$$R = \det \mathbf{M} - k(\operatorname{trace} \mathbf{M})^2$$

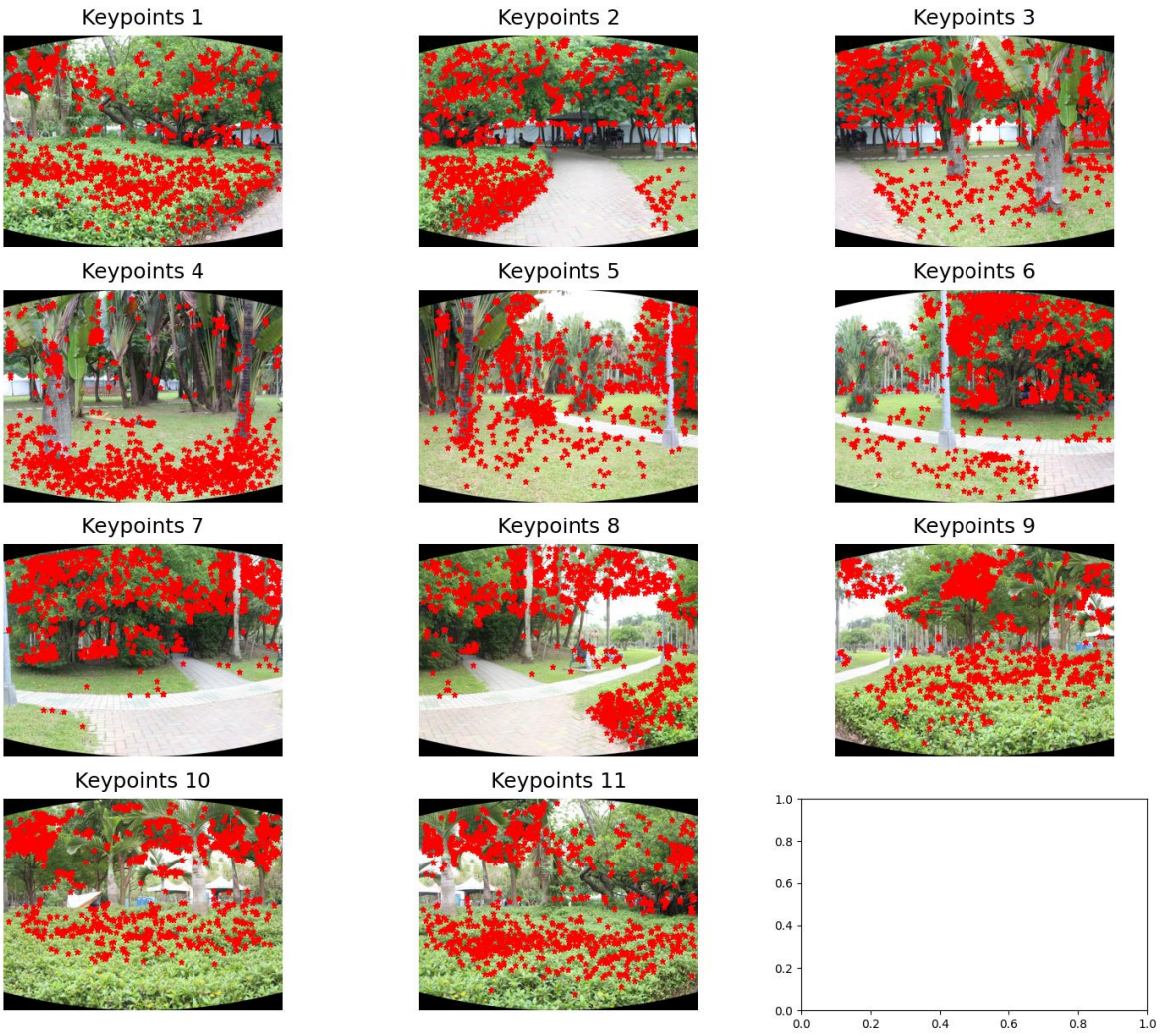
$$\det \mathbf{M} = \lambda_1 \lambda_2$$
$$\operatorname{trace} \mathbf{M} = \lambda_1 + \lambda_2$$

5. 之後我們用 non-maximum suppression 對 R 做處理，再挑選前 N 大的 R 點，代表我們要挑選 N 個特徵點，這裡為了避免取道邊界的點，我們在做 non-maximum suppression 之前，會用圓柱投影的邊界往內到 2 倍 window 之間做一個 mask，將此範圍的 R 設為 0



6. 取得特徵點的座標，並且存入 keypoints 裡

以下是我們將每張圖的 keypoints 畫出的樣子



Feature matching

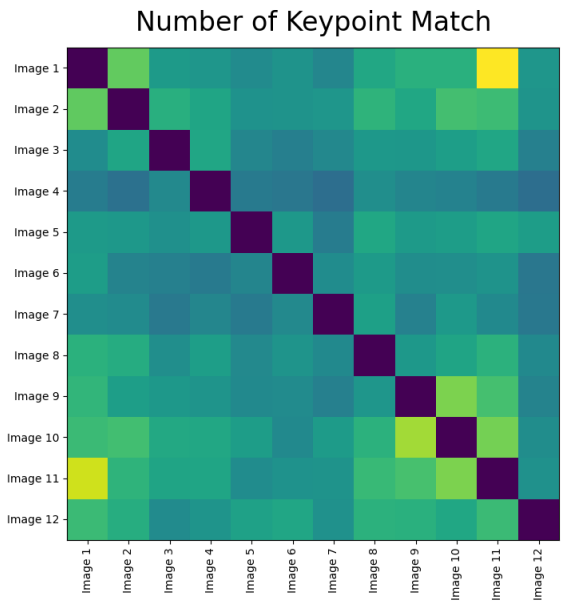
A. Simplest method

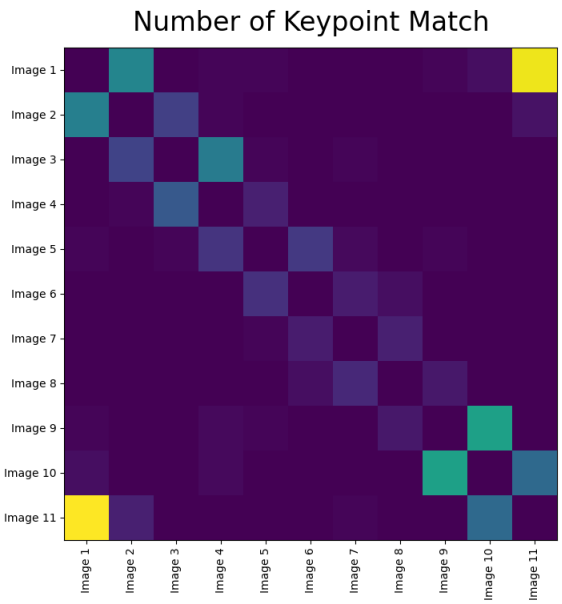
在 match 之前，需要先找出每個點的 description，我們使用最簡單的作法

1. 將此點的 patch 拉成一維向量
2. 為了減少相對亮度差的影響，我們將 patch 減掉平均亮度再處以標準差

B. SIFT feature description (Bonus)

Simplest method 對於老師網站上的範例照片雖然已足夠，但在我們自己拍的照片上效果卻不是很好，因此我們也有實作老師上課講到的 SIFT feature description 方法





左圖: Simplest method 完全沒有辨識度

右圖: SIFT feature description 能夠正確 match point(雖然 match 有點少)

我們推測是因為我們的照片太多特徵點都是樹葉了，如果用太簡單的 feature description 會無法區分特徵點

Step: (SIFT feature description 的算法步驟)

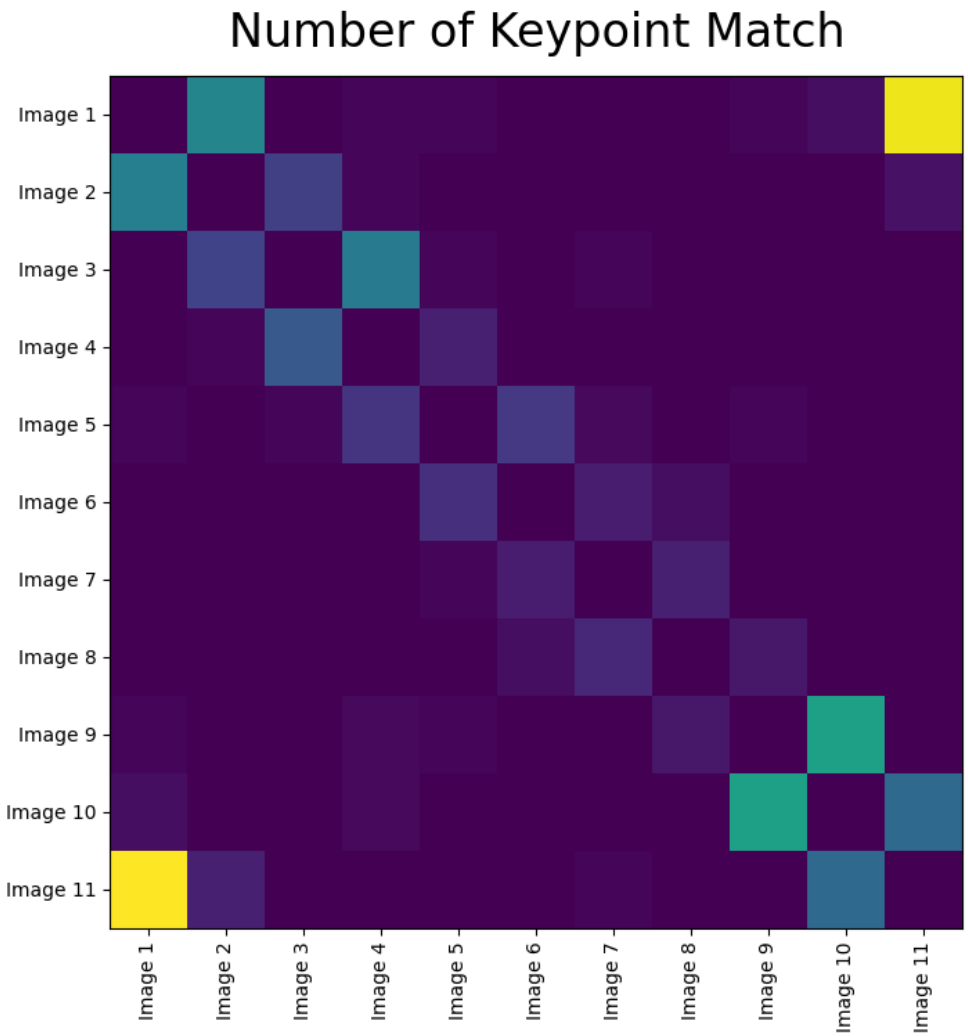
1. 利用 x 方向、y 方向的微分 Ix、Iy 計算出 theta
2. 將 16x16 的 patch 分成每 4x4 個 patch 進行投票
3. 將 theta 分為八種角度，計算每個 pixel 的投票數
4. 最後總共會有 4x4x8=128 維的向量(4x4 個 patch 與 8 種角度)
5. 對向量做一些後處理 → Normalized, clip values larger than 0.2, renormalize

Match

1. 我們使用的暴力法，依序算出每個點與其他點之間的距離，之後排序，選取距離自己最近的點
2. 因為怕有些特徵點不具有辨識度，容易判斷錯誤，所以我們會將特徵點的距離與第二進平均特徵點的距離作比較，如果小於 ratio 才選，ratio 介於 0~1 之間，設越大代表門檻越低(best_dists/second_best_dists.mean() < ratio)
3. 之後會將兩點在 keypoints 的 idx 存入 matchs 裡面

Image matching

為了要知道我們要跟哪張圖拼接，我們會遍歷所有組合，算出兩兩一張圖的 match 個數，並且作統計，以下是影像之間的 match 個數圖



也許是因為我們的特徵點都是樹葉(太相似了)，因此 match 到的點都蠻少的(最少只有 3 個點)

Step: (Image matching 的算法如下)

1. 我們會隨便選一張圖當作起始點，選擇與這張圖 match 個數最多的另一張圖做合併，並切將此張圖作為起始點繼續找與這張圖 match 個數最多的另一張圖做合併
2. 已合併過的圖會被標記，以免被重複合併
3. 我們使用 RANSAC 算法，藉由一對 match 的 point pair 估計 offset_x 與 offset_y
4. 在合併之前，我們會先確認 match 的個數至少有 3 個，且在 match 的點中，符合算出的 offset 的 pair 至少大於 4 成
5. 藉由(len(all_matches)>= 4 and best_count > len(all_matches) * 0.4) 算法，就算我們混入了不同場景的圖也依然可以成功拼接。

Image 1

Image 2

Image 3

Image 4

Image 5

Image 6

Image 7

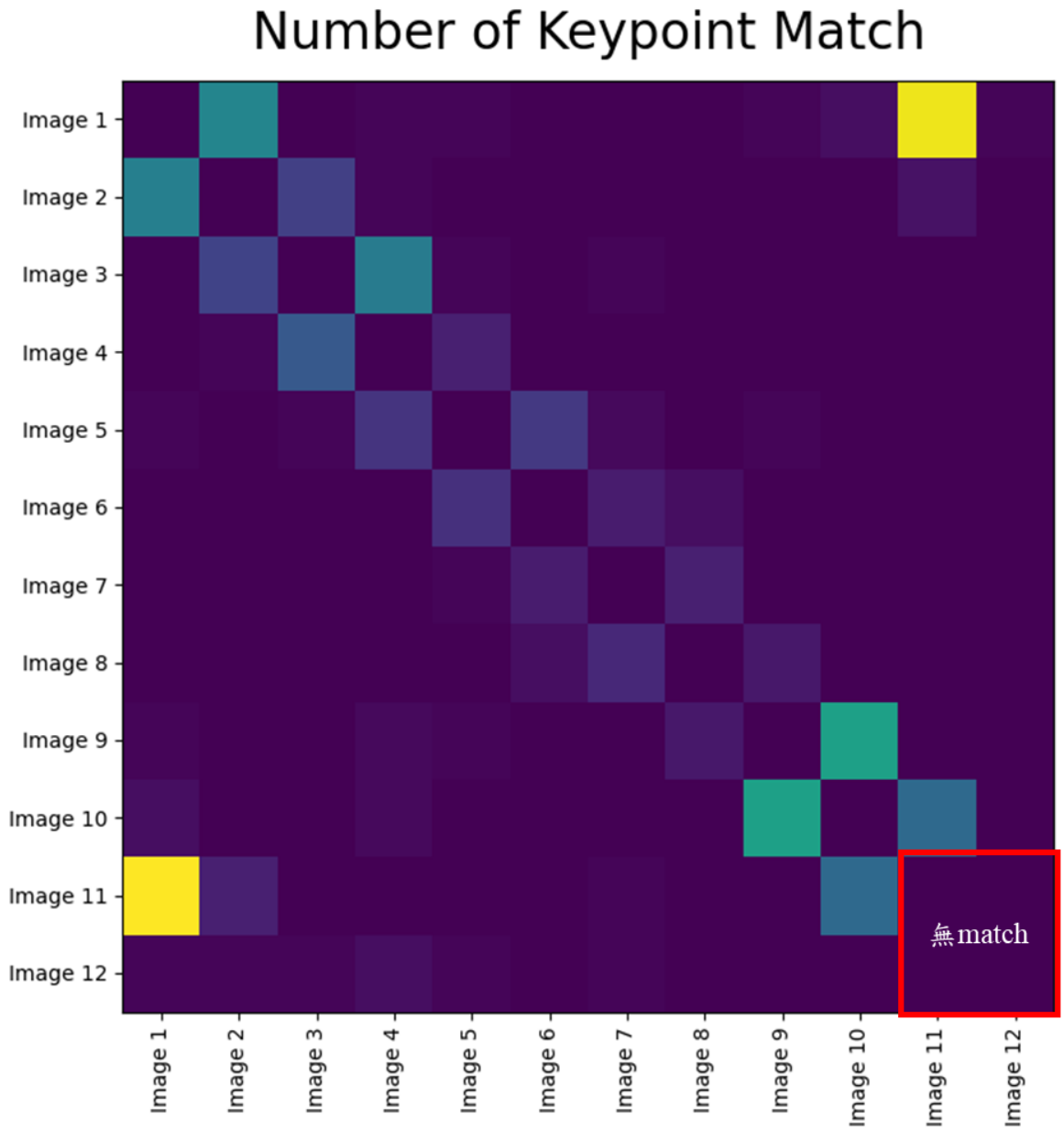
Image 8

Image 9

Image 10

Image 11

Image 12



Blending

我們在進行影像拼接時，會先創建一個 used 矩陣，用來判斷是否拼接過這張影像，然後選擇未使用過的影像中，與目前影像之間具有最多匹配點的影像進行拼接，從而產生一張合理的全景圖。

Steps:

1. 進行兩張影像的拼接時，會輸入兩張影像與兩張影像分別的 mask，其 mask 代表是否要保留影像中的像素值(也就是判斷影像中哪部分有重疊)
2. 根據原本影像大小與位移建立新的長與寬的影像，確保能放進混合拼接後的影像

3. 接著會對透過位移轉換矩陣將影像與 mask 進行平移，這樣就能將兩張影像疊合在一起
4. 再根據水平位移量的正負情況分別計算一個權重矩陣，其代表混合區域要參考第一張影像與第二張影像的比例，將兩張影像按照權重矩陣進行混合，就能得到最後的結果

以下是進行 blending 的過程結果圖

Stitch Image 11 with Image 1

End-to-end alignment (Bonus)

Steps:

1. 找出影像最左邊與最右邊中，第一個像素點不為 0 的位置，並計算全域偏移量 (透過兩點相減)
2. 計算每個 column 的像素偏移量，是透過全域偏移量除以影像寬度得到的
3. 若偏移量大於等於 0，則從影像左邊開始，每個像素的位置向上移動，若偏移量小於 0，則從影像右邊開始，每個像素的位置向上移動
4. 找出對齊後影像的上、下、左、右邊界
5. 依據邊界剪裁影像，得到最後對齊的影像

因為我們的照片沒什麼位移，因此我們使用老師網站上的範例照片作為例子，以下是我們做完 blending 的結果圖。(還未進行 end-to-end alignment)

以下是我們做完 end-to-end alignment 的結果圖

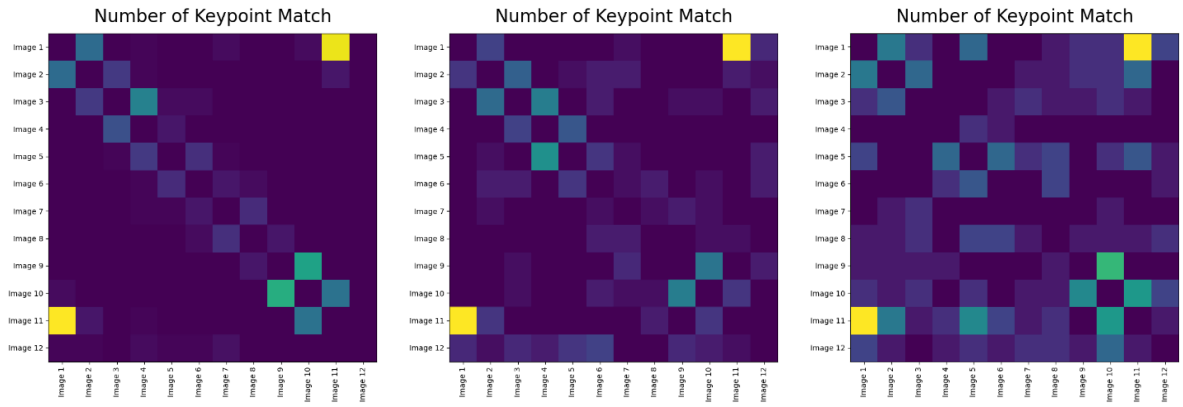
因為影像解析度較低，我們是使用以下的參數設定來進行處理

```
args = {
        'input_directory' : '../data/input/3/' ,
        'scale'           : 1 ,
        'focal_length'    : 2016 ,
        'output_image'    : '../result.jpg',
        'window'          : 16,
        'N'               : 256,
        'descriptors_sift' : True,
        'R_window_scale' : 1
}
```

Discussion

為了方便計算和縮短測試時間，一開始我們將圖片縮小了一半 (scale=0.5)。但是當我們將縮放比例設為 1 時，發現無法完全 match 所有圖片，最多只能成功 match 兩張。我們推測這是因為樹葉的特徵點太多了，沒有辨識度，導致無法精確匹配。

當影像被縮小時，一個 patch 能看到的範圍變大，因此特徵點比較有辨識度。但是縮小影像的缺點是鬼影現象更加明顯。這可能是因為當影像被縮小時，點的位置變得不那麼精確，導致對齊不夠好。



Scale: 0.5　　Scale: 0.8　　Scale: 1

Scale: 0.5

Scale: 0.8

Scale: 1

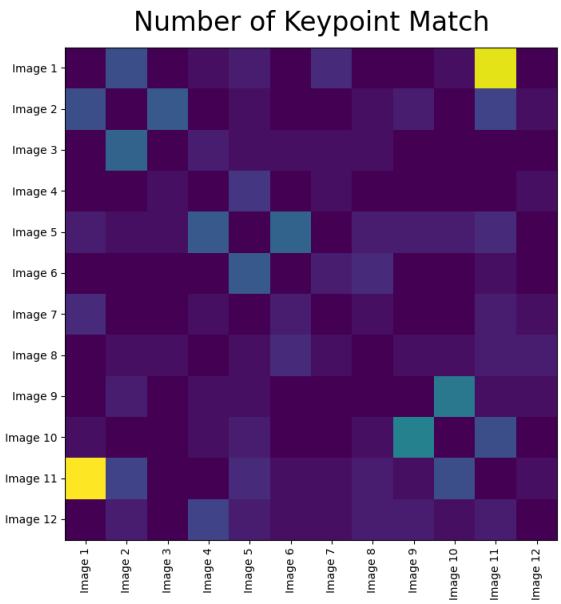
最後我們發現了一個盲點，那就是當同個焦距，不同 scale 的影像在做圓柱投影時，投影的結果有些不同

Scale: 0.5

Scale: 0.8

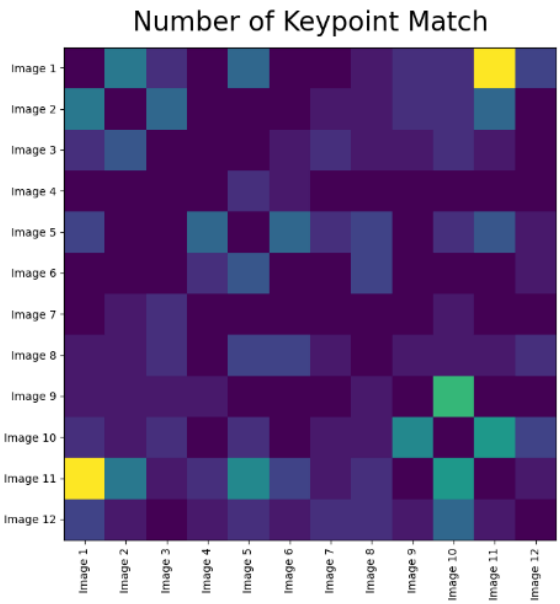
Scale: 1

因此我們讓投影的焦距會隨著 scale 縮放(focal_length* scale)，但改成這樣後，變成 match 的效果不好



Scale = 0.5

focal_length = 2016*0.5



Scale = 1

focal_length = 2016

Scale = 0.5

focal_length = 2016*0.5

Scale = 1
focal_length = 2016*1

可以看到鬼影(草叢那裡)比較少了，所以鬼影推測是由焦距調不好產生的，但不知道為什麼做圓柱投影改的越劇烈，match 的效果就越不好，推測是因為變形太劇烈，導致特徵點不好 match。

最終我們選擇折衷的參數
Scale = 0.8，focal_length = 2016*0.8

```
args = {
    'input_directory' : '../data/input/1/' ,
    'scale'           : 0.8 ,
    'focal_length'    : 2016 ,
    'output_image'    : '../result.jpg',
    'window'          : 16,
    'N'               : 2048,
    'descriptors_sift' : True,
    'R_window_scale' : 2
}
```

我們最後完成的結果圖如下:

Reference

- https://yungyung7654321.medium.com/python%E5%AF%A6%E4%BD%9C%E8%87%AA%E5%8B%95%E5%85%A8%E6%99%AF%E5%9C%96%E6%8B%BC%E6%8E%A5-automatic-panoramic-image-stitching-28629c912b5a
- https://github.com/dastratakos/Homography-Estimation/blob/main/imageAnalysis.py
- https://zhuanlan.zhihu.com/p/34761031
- https://github.com/j40903272/VFX2018
- https://www.analyticsvidhya.com/blog/2021/06/feature-detection-description-and-matching-of-images-using-opencv/
- https://github.com/jnfem112/VFX2022SPRING
- https://www.geekering.com/programming-languages/python/brunorsilva/harris-corner-detector-python/
- https://github.com/shuoenchang/NTU-Digital_Visual_Effects-VFX
- https://github.com/qhan1028/Image-Stitching
- https://github.com/KenYu910645/VFX2022/tree/main/hw2_image_stiching