

Digital Visual Effects - Project 2

第 28 組 - 李澤謙 陳宏昇

April 27, 2022

Implementation

以下為我們於本次作業中所使用的照片：

Image 1



Image 2



Image 3



Image 4

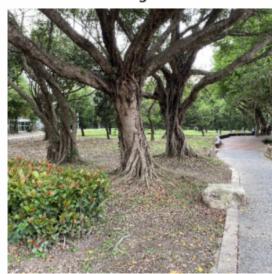


Image 5



Image 6



Image 7



Image 8



Image 9



Image 10



Image 11



Image 12



其中可以注意到我們的照片並沒有按照拍攝時的順序（意即場景的由左至右或由右至左）所排列，並且我們有加入一張不屬於該場景的照片，目的是希望能夠達到 recognizing panoramas，意即，希望實作出的作法要能夠找出哪些照片屬於同一個場景，並找出這些照片正確的拼接順序，以產生一張 panorama。此外，由於原圖的解析度非常高，若是直接使用原圖來進行處理所需要的執行時間會非常久，因此我們有將圖片的大小縮小 5 倍，再進行後續的處理。

Cylindrical Projection

首先，我們會將所有照片都進行 cylindrical projection。在 focal length 的選擇上，根據照片所記錄的資訊顯示其 focal length 為 26mm，而我們在網路上找到了以下公式：

$$\text{focal length (in pixel)} = \frac{\max(\text{height}, \text{width}) \times \text{focal length (in mm)}}{\text{sensor size (in mm)}}$$

我們所拍攝的照片大小皆為 604×604 ，而我們查到 iPhone 11 的 sensor size 為 $1/2.55 \text{ inch} \approx 9.96078431 \text{ mm}$ ，將這些數據代入以上公式之後，可得 focal length 大約為 1578 個 pixel。在得到 focal length 之後，我們使用了 inverse warping 來進行 cylindrical projection：首先，若原照片的長寬分別為 h 和 w ，focal length 為 f ，則經過 cylindrical projection 所得到的照片其長寬分別為 $\hat{h} = h$ 和 $\hat{w} = 2f \cdot \tan^{-1}(\frac{w}{2f})$ ，接著，對於 cylindrical projection 所得到的照片中的每一點，若其

座標為 (\hat{x}, \hat{y}) ，則其在原照片中所對應到的 x 座標為 $x = f \cdot \tan(\frac{\hat{x}}{f})$ ，而 y 座標為 $y = \frac{\hat{y}}{f} \sqrt{x^2 + f^2}$ (在此是將照片的正中間作為座標原點，而右方為 x 軸正向，上方為 y 軸正向)，最後，我們會使用原照片之中與 (x, y) 最相鄰的四個點來進行 bilinear interpolation，以此得到 cylindrical projection 之後的照片中 (\hat{x}, \hat{y}) 位置上的 pixel 數值。下圖為 cylindrical projection 所得到的結果：

Image 1



Image 2



Image 3



Image 4



Image 5



Image 6



Image 7



Image 8



Image 9





Keypoint Detection and Keypoint Description

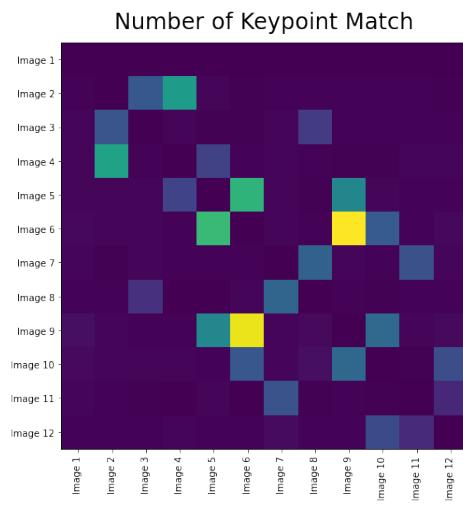
我們參考了這篇文章來實作 SIFT 演算法，以達成 keypoint detection 和 keypoint description。下圖為 SIFT 演算法所找出的 keypoint：



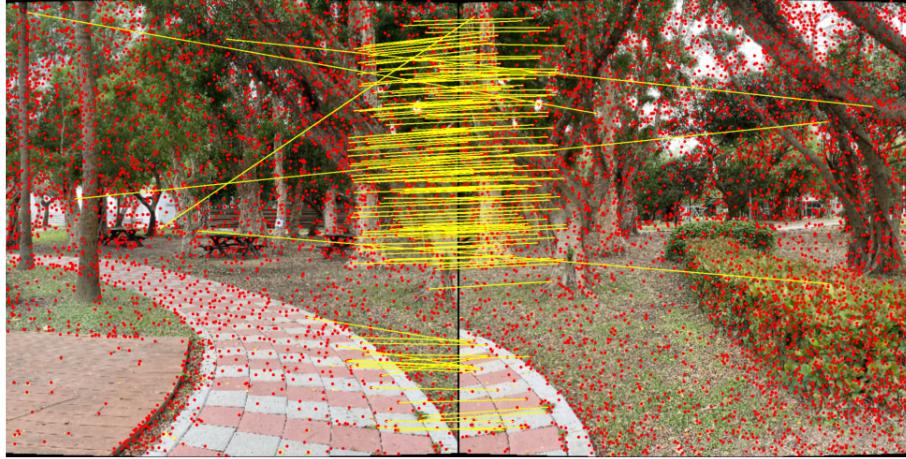


Keypoint Matching

在得到每張照片的 keypoint 與其 descriptor 之後，我們使用了 OpenCV 所提供的 knnMatch 函式，對任意兩張照片之中的 keypoint 進行 matching，並使用 Lowe's ratio test 對 matching 的結果進行篩選，以下為任意兩張照片之間所 match 到的 keypoint 數目：



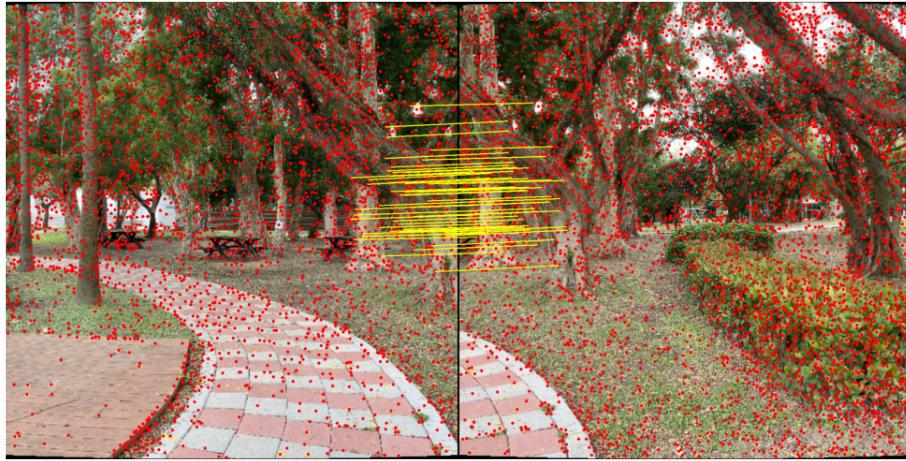
由此可以看出，在原場景之中越相近的兩張照片其 match 到的 keypoint 數目也會越多。而下圖為第 2 張照片和第 3 張照片之間 keypoint matching 的結果：



由此可以看出在原場景之中相近的兩張照片，其重疊區域之中的 keypoint 幾乎都會得到正確的 matching，但兩張照片之中仍然會有少部份錯誤的 keypoint matching。

Image Matching

我們使用了 Recognizing Panoramas 論文中的方式來進行 image matching。對於每一張照片 I ，我們會找出與其 keypoint matching 數量最多的前 N 張照片，令其分別記為 J_1, J_2, \dots, J_N ，這 N 張照片即為最有可能與照片 I 拼接在一起的照片，接著，對於其中一張照片 J_n ，我們會使用 RANSAC 演算法找出照片 I 和照片 J_n 之中哪一些 keypoint matching 為 inlier/outlier，並同時找出這兩張照片之間的相對平移量，在 RANSAC 演算法的實作上，我們會執行 M 次的迴圈，在每次的迴圈之中，我們會隨機選取一組對應的 keypoint，計算這兩個 keypoint 之間在水平與垂直方向上的平移量，並將其它組的 keypoint 皆移動該平移量，再計算有多少組 keypoint 在平移之後兩者之間的距離小於所指定的 threshold，我們會紀錄哪一個平移量可以使最多組 keypoint 通過 threshold，將該平移量定為照片 I 和照片 J_n 之間的相對平移量，並將有通過 threshold 的 keypoint matching 視為 inlier，最後，假設照片 I 和照片 J_n 之中 keypoint matching 的數量為 n_f ，當中 inlier 的數量為 n_i ，我們使用了 Recognizing Panoramas 論文中的公式，若 $n_f > 0.59 + 0.22n_i$ ，則我們便將照片 I 和照片 J_n 視為可以拼接在一起的一組照片，反之便將其視為無法拼接在一起。在參數的選擇上，我們使用了 $N = 2$ 、 $M = 1000$ 、threshold 為 5。下圖為第 2 張照片和第 3 張照片經過 RANSAC 演算法之後所保留的 keypoint matching：



由此可以看出雖然保留下來的 keypoint matching 很少，但錯誤的 keypoint matching 確實有被移除掉。

Image Blending

在找出所有可以拼接在一起的照片之後，我們會任選一張照片作為起始，並不斷找出可以與當前的 panorama 進行拼接的照片，將其以 linear blending 的方式拼接到當前的 panorama 中，以此得到更大的 panorama，直到沒有更多的照片可以拼接近來為止。下圖為拼接完的結果：



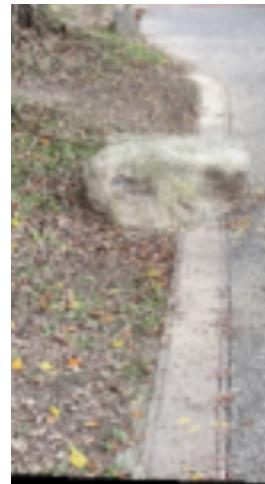
End-to-End Alignment

最後，我們會找出拼接完的照片之中，第一個 column 之中最上方與最下方不為黑色的 pixel，將其分別對應到照片的左上角及左下角，以及最後一個 column 之中最上方與最下方不為黑色的 pixel，將其分別對應到照片的右上角及右下角，以此進行 perspective transformation，達到 end-to-end alignment，以下為最終所得到的 panorama：



Discussion

在最終的 panorama 之中，可以發現照片拼接處會出現如下的鬼影：



推測其可能是因為我們假設兩張照片之間的 motion 僅有 translation 所致，若是假設兩張照片之間有更複雜的 motion，可能可以使得兩張照片對齊得更好，進而減少更多的鬼影，此外，我們在拼接照片時僅使用了最簡單的 linear blending，若是使用了更複雜的 blending 方式，也有可能可以使得照片拼接處看起來更為自然。此外，由於我們在拍攝照片時有使用腳架，因此我們的照片之間沒有產生很嚴重的垂直方向位移，雖然因為腳架擺設的方式導致照片從左至右有逐漸向下的垂直位移，但其可以用 perspective transformation 做大致的校正，然而在使用助教所提供的沒有使用腳架所拍攝的照片時，其拍攝照片時垂直方向的位移極大，導致拼接完的照片中的黑色部分沒有辦法使用 perspective transformation 移除，此時可能就要透過實作 bundle adjustment 或 rectangular panoramas 才能解決。以上即為我們在實作本次作業時所發現的問題。