

VFX project2

Image Stitching

b02901004 王士元
b02901101 林士庭

1 Introduction

全景圖是在日常生活中相當廣泛的功能，尤其是在許多相機之中均支援此功能，一般使用上會需要借助人工來進行適當地對齊工作。然而如何透過電腦視覺的方式來進行自動產生，此時便需要從不同的影像中抓出具有重要訊息的點(例如影像中corner出現的位置)作為影像拼貼時重要的參考資訊。

在本次作業中，我們會利用抓取特徵點(feature points)的方式，將來自不同照片的feature points進行matching，根據matching的結果找出適當的homography，然後將這些照片按照順序進行拼貼的工作，以此方式來完成全景圖(panorama)。

我們在作業中實現了下列功能：

- **Cylindrical projection**
- **Feature detection& descriptor**
Multi-scale oriented patches
- **Feature matching**
KD-tree
- **Image Stitching**
Affine transform, Ransac
- **Blending**
Linear blending, Poisson blending
- **Drift**
Global warping

2 Cylindrical projection

由於相機在拍攝時，會將視野區域投影到一個平面上。然而在本次的作業中，我們所希望獲得的為 360° 的panorama，因此我們可以將最終的影像視為是在圓柱上所觀察到的，然而經過相機的捕捉之後，只將部分的圓柱影像投射至平面上。為了要會付原本圓柱的影像，便需要針對影像重新投影。

在投影時，我們會針對影像進行下列修正：

$$\begin{aligned}x' &= f\theta = f \tan^{-1} \frac{x}{f} \\y' &= fh = f \frac{y}{\sqrt{x^2 + f^2}}\end{aligned}\quad (1)$$

where x, y is the coordinate of original image and f is focal length.

(1)式可以理解為，我們先將原始的平面影像中心視為在距離相機為focal length的地方，將影像中各點視為是圓柱中心的光線和平面之交點，藉此回推該點對應自單位圓柱上何處，在進而將圓柱半徑放大至focal length。

2.1 Implementation

實作上我們會發現，雖然同樣使用(1)式進行座標轉換，但實際上有兩種截然不同的思考方向。

1. Forward method

這個方法將原始影像上的grid points直接進行座標轉換至新的影像上的座標，這樣一來所得到的新影像將會是座落在off-grid points上，若我們在此處要進行bilinear interpolation得出新影像上grid points，則必須要搜尋鄰近的4個點，這對於計算上會是很大的負擔。若我們只單純對座標進行quantization到整數，則很有可能會發生部分pixel沒有其對應值的問題。為了要解決這個問題，我們的做法是針對座標quantize之後的影像，朝向上、右、右上進行shift之後再行疊加，藉此方法來恢復丟失的pixel value。但若直接給予相同的權重，則效果上等同於對影像進行average blur，因此我們會將大部分權重給予未位移的影像，並且於疊加後再次進行normalization。

2. Backward method

這個方法下直接考慮在新影像上的每個grid points座標，藉由inverse projection的方式回算其對應在舊影像上的座標，即便所得出的結果不是整數點，也可以直接最接近的4個整數點進行bilinear interpolation。如此一來即可確保新影像中每個pixel均有值。

3 Multi-Scale Oriented Patches[1]

3.1 Image pyramid

在進行corner response的偵測之前，我們會先對原始的影像進行數次的 downsampling以及blurring，希望藉此獲得來自不同scale之下的資訊。Pyramid中每一層影像之間的關係可以描述如下：

$$\begin{aligned}P'_l(x, y) &= P_l(x, y) * g_{\sigma_p}(x, y) \\P_{l+1}(x, y) &= P'_l(sx, sy) \\ \text{with } P_0(x, y) &= I(x, y)\end{aligned}\quad (2)$$

其中 l 代表pyramid階級， g_{σ_p} 代表standard deviation σ_p 的gaussian kernel，此處根據 paper我們採用1.0， s 是downsampling factor，此處我們所採用的為2，pyramid的高度為3。 $I(x, y)$ 是原始灰階影像。

3.2 Feature detection

在這裡所採用的feature detection method是利用Harris corner matrix進行計算，首先，我們會先針對image計算gradient image，並將其進行blurring，此一步驟可以視為是獲得較為”smooth”的gradient。接著再計算Harris matrix，同樣地，再次進行blurring獲得最後的smoothed Harrix matrix，上述過程可以表示成：

$$\nabla_{\sigma_d} P_l(x, y) = \nabla P_l(x, y) * g_{\sigma_d}(x, y) \quad (3)$$

$$\mathbf{H}_l(x, y) = \nabla_{\sigma_d} P_l(x, y) \nabla_{\sigma_d} P_l(x, y)^T * g_{\sigma_i}(x, y) \quad (4)$$

其中gaussian kernel的standard deviation採用paper建議的 $\sigma_d = 1.0$, $\sigma_i = 1.0$ 。

計算出Harris matrix後，我們可以根據harmonic mean的方式計算corner response，

$$f_{HM}(x, y) = \frac{\det \mathbf{H}_l(x, y)}{\text{tr } \mathbf{H}_l(x, y)} \quad (5)$$

根據paper，在計算完corner response後，被認定為feature point的點必須符合(1)3 × 3區域內局部最大值(2)大於某一個threshold，此處我們採用10.0。實作中我們可以利用dilation的方式取出3 × 3區域中的最大值。

3.3 Adaptive non-maximal suppression

為了要避免feature points在空間上分佈過於聚集在某些特定區域之中，並且增加其分佈均勻的程度，因此會針對一個區域內限制其feature point僅能由最大的response作為代表，而接下來的問題就是要如何在給定所希望的feature points數目下，計算此區域的半徑限制。概念上，我們將半徑由0逐漸擴增，直到目標的數目達成。然而在實作上，將半徑由大到小遞減較為適當，由於一旦被納入為合法的feature point，則接下來均會存在於候選名單之中不被踢除。

我們實作中主要包含幾個部分。首先，我們先將各個local maximal candidate依照其值進行排序，這樣一來便可以建立各點之間大小的先後關係。接著，計算各點之間的距離，並從中紀錄各點所對應的大於其值且距離最近的對應點。根據和對應點之間的距離進行排序，選出其中距離最大的候選點，直到滿足數目限制，此時，被選入的候選點中，距離對應點最小者之距離即為所求之半徑限制。如同alg.1所示：

在此處我們根據paper的設定，取出最多500個feature points。

3.4 Subpixel refinement

在取出確定數目的feature points之後，我們會再將feature points的位置進行更加精確的定位以及修正，在此處會利用2D quadratic interpolation的方式，估計出對於 f_{HM} 在3 × 3區域內local maximum發生的位置，將此點視為是feature points在subpixel accuracy下的位置。而後再透過rounding的方式，重新校正feature points的位置至grid points上。

Algorithm 1 Adaptive non-maximal suppression

Input: Candidates: response image with only local maximum

Input: \max_N : required number of feature points

Output: a list of location of feature points with \max_N elements

```

1: procedure ANMS(Candidates,  $\max\_N$ )
2:   for each  $i$  in Candidates do
3:     generate  $(\mathbf{x}_i, v_i)$  pair.            $\triangleright \mathbf{x}_i$  is positional vector,  $v_i$  is response value
4:   end for
5:   sort  $(\mathbf{x}_i, v_i)$  according to  $v_i$ .
6:   for each  $i$  in  $(\mathbf{x}_i, v_i)$  do
7:     find  $d_i = \min_{k \in N_i} \|\mathbf{x}_i - \mathbf{x}_j\|$             $\triangleright N_i = \{k \in \text{Candidates} | v_i < v_k\}$ 
8:     generate  $(\mathbf{x}_i, v_i, d_i)$ .
9:   end for
10:  sort  $(\mathbf{x}_i, v_i, d_i)$  according to  $d_i$  in descending order.
11:  Choose top- $\max_N$  candidates.
12:  return  $\mathbf{x}_i$ 
13: end procedure

```

$f_{-1,1}$	$f_{0,1}$	$f_{1,1}$
$f_{-1,0}$	$f_{0,0}$	$f_{1,0}$
$f_{-1,-1}$	$f_{0,-1}$	$f_{1,-1}$

Figure 1: For sub-pixel accuracy, compute derivatives in 3×3 region

依照Figure 1所示，我們可以計算second-order Taylor's expansion如下：

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0}^T (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \left. \frac{\partial^2 f}{\partial \mathbf{x}^2} \right|_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) \quad (6)$$

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \frac{f_{1,0} - f_{-1,0}}{2} \\
\frac{\partial f}{\partial y} &= \frac{f_{0,1} - f_{0,-1}}{2} \\
\frac{\partial^2 f}{\partial x^2} &= f_{1,0} - 2f_{0,0} + f_{-1,0} \\
\frac{\partial^2 f}{\partial y^2} &= f_{0,1} - 2f_{0,0} + f_{0,-1} \\
\frac{\partial^2 f}{\partial x \partial y} &= \frac{f_{-1,-1} - f_{-1,1} - f_{1,-1} - f_{1,1}}{4}
\end{aligned} \quad (7)$$

對於式(6)取導數為0的點作為其local maximum在subpixel refinement下的位置，則

$$\begin{aligned}
 0 &= \frac{\partial f}{\partial \mathbf{x}} = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} + \frac{1}{2} \left(\left. \frac{\partial^2 f}{\partial \mathbf{x}^2} \right|_{\mathbf{x}=\mathbf{x}_0}^T + \left. \frac{\partial^2 f}{\partial \mathbf{x}^2} \right|_{\mathbf{x}=\mathbf{x}_0} \right) (\mathbf{x} - \mathbf{x}_0) \\
 &= \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0} + \left. \frac{\partial^2 f}{\partial \mathbf{x}^2} \right|_{\mathbf{x}=\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) \\
 \mathbf{x}_m &= (\mathbf{x} - \mathbf{x}_0) = - \left. \frac{\partial^2 f}{\partial \mathbf{x}^2} \right|_{\mathbf{x}=\mathbf{x}_0}^{-1} \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}_0}
 \end{aligned} \tag{8}$$

藉由偏移量 \mathbf{x}_m ，我們可以使用0.5作為threshold來判斷是否進行feature points location的修正。

3.5 Feature descriptor

在挑選出適當的feature points之後，需要利用descriptor針對該feature point的特性進行適當的參數化。在此處，我們所考慮的項目包含feature point的orientation以及周遭環境的影像結構(patch)。

3.5.1 Orientation

在計算每個feature points的方向時，我們會再次透過smooth gradient來獲得。

$$\begin{aligned}
 \mathbf{u}_l(x, y) &= \nabla_{\sigma_o} P_l(x, y) \\
 [\cos \theta, \sin \theta] &= \frac{\mathbf{u}_l}{\|\mathbf{u}_l\|}
 \end{aligned} \tag{9}$$

實作中我們依照paper，採用 $\sigma_o = 4.5$ 。

3.5.2 Patch

為了要能夠描述feature points周遭的影像，首先，我們會沿著local orientation的方向取出local intensity patches，但同時也希望取出來的descriptor對於location不要過於敏感，因此會採用在較低的sampling frequency下取出patches的方式來達成。

在實作中我們根據paper，取出 8×8 的patches，每個patch的sample點相距5個pixels，換言之，整個區域共包含 40×40 的pixels資訊。另外，在paper中提出多種取sample的方式，而我們所採用的方法如下：

$$P_l''(x, y) = P_l(x, y) * g_{2\sigma_p}(x, y) \tag{10}$$

根據paper的解釋，採用額外的gaussian blurring將可以減少aliasing的產生。另外，由於在進行sample時必須考慮到靠近邊界區域的feature points可能會缺乏足夠的pixels進行descriptors的產生，因此與邊界距離在20 pixels以內的區域將不會偵測feature points。

取出來的descriptor還會再經過normalization的步驟，這可以使得取出來的 descriptors免於intensity上受到affine的影響。

3.6 Results

Figure 2a中是將feature point location標示在其所detected的pyramid level的結果，點的位置即是feature points location，其中第一層pyramid的部分feature points則加上descriptors表示之。在Figure 2b則是將來自三層pyramid的所有feature points一起標示在原圖上，橘、綠、藍分別代表來自的pyramid level由下至上表示。



(a) Image pyramid with descriptors



(b) feature points from different level of pyramid

Figure 2: Results of our MSOP features

4 Feature Matching

在對任兩張圖片 I_1, I_2 抽取特徵點後，為了找出兩張圖片的重疊區域，我們會對這些特徵點作一對一的 matching。直覺上來說，我們要做的事情就是對 I_1 中的每個特徵點 f_1^i 搜尋其在 I_2 空間上的 nearest neighbor n_1^i ，並設定一個 threshold 來判斷 f_1^i 與 n_1^i 是否為一合理的 matching。然而若我們直接使用暴力搜尋，可能得掃過全部的特徵點才能找到 nearest neighbor，因此在這邊我們使用 KD-tree 的結構來增加尋找 nearest neighbor 的速度。

4.1 KD-Tree

KD-Tree 基本上即是一個推廣到 K dimension 的 binary search tree。在 binary search tree 中，所有資料皆為單一數值，因此我們可將資料視為一群在一維數線上的點，而在每一個 node 左枝的資料點在數線上皆會位於此 node 的左側。在 KD-Tree 中，所有的資料則為 K 維的向量，而相較於 BST 中每個 node 會將數線切成左右兩半，KD-Tree 的每個 node i 則會將 K 維空間對維度 d_i 以 K-1 維的超平面切成兩部份，也就是說我們將資料空間切成了多個子空間，並將其空間上的性質 encode 到了樹的結構中，因此可將搜尋的速度由 $O(N)$ 加快至 $O(\log N)$ 。在本次作業中，我們使用 scipy 中的 KDTree() 來達成此項功能。

在對一 query point p 搜尋其在一群資料點中的 nearest neighbor 時，我們亦可利用 KD-Tree。步驟如下：

1. 先對欲搜尋之資料點集合建立 KD-Tree，並可得到此數的 root
2. 以 Alg.2 作 $\text{NNS}(p, \text{root}, n_p, r)$ ，即可得到 p 在資料點集合中的 nearest neighbor n_p ，而若欲得到 k nearest neighbors，也可利用與 Alg.2 幾乎相同的演算法得到

4.2 Matching

在使用 KD-Tree 後，我們即可很有效率地對 f_1^i 尋找其在 $\{f_2^m\}$ 中的 nearest neighbor n_1^i ，並得到其距離 r_1^i 。但是因為兩張圖可能只有一小部分是重疊的，且有些 feature 可能會非常接近使得演算法出錯，因此每個 (f_1^i, n_1^i) 的配對並非都是正確的配對，而在這邊我們使用了兩種方式來篩選出可信度較高的 matching pairs。

4.2.1 Cross Checking

對一個正確的 matching pair (f_1^i, n_1^i) 來說， f_1^i 與 n_1^i 對彼此理論上就是最好的配對，也就是說他們都會是互相的 nearest neighbor。利用這個概念，在我們對 f_1^i 在 $\{f_2^m\}$ 中找到 nearest neighbor n_1^i 後，我們可以對 n_1^i 同樣以 KD-Tree 搜尋其在 $\{f_1^m\}$ 中的 nearest neighbor，若此 nearest neighbor 與 f_1^i 並不相同，則我們就不會將這兩點 f_1^i, n_1^i 配對在一起。

4.2.2 Ratio Test

另外，在 I_1 上的一個特徵點 f_1^i 可能因為 I_1, I_2 有許多相似的區域使得其在 I_2 上有很多相近的特徵點，這些相近特徵點之 feature 與 f_1^i 之距離亦會只有很小的差距，造成我們即使使用 KD-Tree 找到 nearest neighbor，也會因為可能存在的雜訊等因素而很難有很高的信心保證這一對 pair 就是正確的 matching。因此在經過 cross checking 之後，我們還會在利用 ratio test 來做篩選。步驟如 Alg.3。

Algorithm 2 Nearest Neighbor Search

Input: p : query point, n : node, n_p : nearest point, r : the distance from p to n_p

Output: n_p, r

```
1: procedure NNS( $p, n, n_p, r$ )
2:   if  $n$  is a leaf then
3:      $r' \leftarrow \text{Distance}(p, n)$ 
4:     if  $r'$  is less than  $r$  then
5:        $r \leftarrow r'$ 
6:        $n_p \leftarrow n$ 
7:     end if
8:   else if  $n$  is a hyperplane then
9:     if  $p$ 's value on  $d_n$  is less than  $n$ 's value on  $d_n$  then  $\triangleright$  Need to search left child
10:      NNS( $p, n$ 's left child,  $n_p, r$ )
11:      if  $p$ 's value on  $d_n + r$  is bigger than  $n$ 's value on  $d_n$  then
12:        NNS( $p, n$ 's right child,  $n_p, r$ )
13:      end if
14:    else  $\triangleright$  Need to search right child
15:      NNS( $p, n$ 's right child,  $n_p, r$ )
16:      if  $p$ 's value on  $d_n - r$  is less than  $n$ 's value on  $d_n$  then
17:        NNS( $p, n$ 's right child,  $n_p, r$ )
18:      end if
19:    end if
20:  end if
21: end procedure
```

也就是說，若 f_1^i 在 $\{f_2^m\}$ 中的最相似點與第二相似的點與 f_1^i 的相似度(r)相差很大，我們就認為這樣的 pair 在 matching 的過程中較沒有受到雜訊等因素的因素的影響，則可以被選入我們最後的 matching pairs 中。

Algorithm 3 Ratio Test

Input: (f_1^i, f_2^j) : a matching pair passed cross checking, $\{f_1^m\}$: feature points in I_1 , $\{f_2^m\}$: feature points in I_2

Output: whether (f_1^i, f_2^j) has passed the ratio test

- 1: Find the two nearest neighbor of f_1^i in $\{f_2^m\}$, $(n_1^i)_1$ and $(n_1^i)_2$, which have distance $(r_1^i)_1, (r_1^i)_2$ from f_1^i respectively, $(r_1^i)_1 \leq (r_1^i)_2$
 - 2: Find the two nearest neighbor of f_2^j in $\{f_1^m\}$, $(n_2^j)_1$ and $(n_2^j)_2$, which have distance $(r_2^j)_1, (r_2^j)_2$ from f_2^j respectively, $(r_2^j)_1 \leq (r_2^j)_2$
 - 3: **if** $(r_1^i)_1 < 0.7 \times (r_1^i)_2$ and $(r_2^j)_1 < 0.7 \times (r_2^j)_2$ **then**
 - 4: **return** True
 - 5: **else**
 - 6: **return** False
 - 7: **end if**
-

4.3 Feature Matching Result

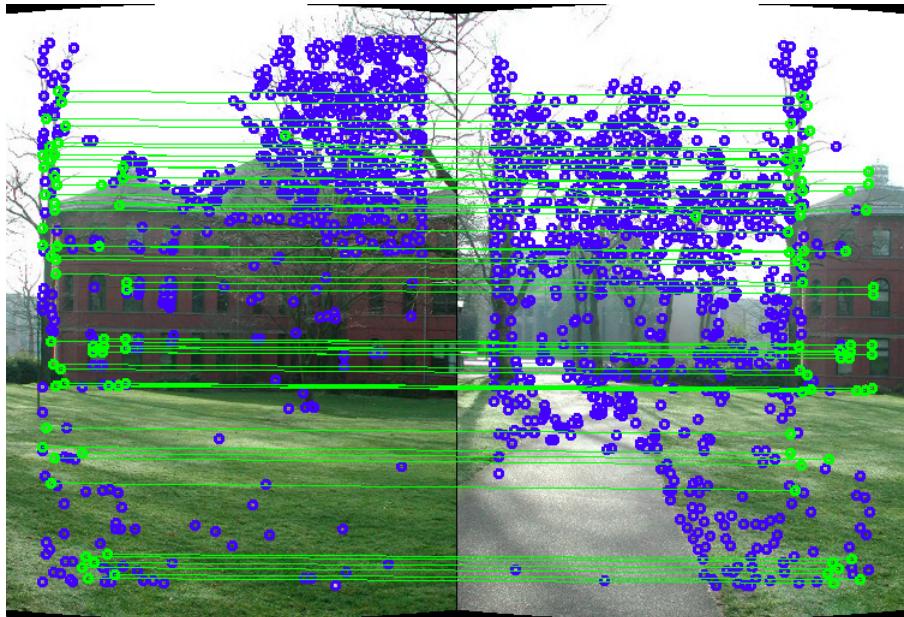


Figure 3: Matching result for two adjacent images, in which blue circles are detected feature points, green circles are matched feature points

5 Image Stitching

在有了 I_1 與 I_2 間的 matching pairs 後，我們即可利用這些 pairs 計算兩張圖之間座標點的關係，再利用得到的關係式將 I_1 warp 至 I_2 的座標系中進行 stitching。然而即使已使用 non-maximal suppression、subpixel refinement、cross checking 與 ratio test 來使得我們的 matching pairs 是足夠正確的，但其中不可避免的仍會有一些 false positive，或是特徵點的位置有些偏移。因此在這邊我們使用 Ransac algorithm 來讓我們的 warping function 更加 robust。

5.1 Ransac

因為 false positive 及特徵點位置的偏移，我們基本上無法找到一個 global warping function 使得每個 matching 的第一個元素之座標都可準確的 map 到第二個元素的座標上，因此我們通常會尋找可以讓 total error 達到最小的 warping function。然而，這樣的計算非常容易受到 outlier 的影響而產生極大的誤差。為避免這樣的問題，我們可以利用 matching pairs 數量通常會較解 warping function 之係數所需的 data 數量來得多這一特性，在解 warping function 時只從 matching pairs 中 sample 一定數量的 pairs 出來進行計算，並以其他沒 sample 到的 pairs 判斷此 warping function 是否對大部分的點都是符合的。以此方式計算多次，我們即可得到一較可靠得 warping function。而詳細的演算法則如 Alg.4，其中的 threshold 我們則是使用 0.1。

Algorithm 4 Ransac Algorithm

Input: $\{v_1^i\}, \{v_2^j\}$: matching pairs' coordinate in I_1 and I_2 respectively, which are ordered such that (v_1^i, v_2^j) is a matching pair

Input: n : how many samples to choose, k : how many times to run

Output: warp func

```
1:  $N \leftarrow |\{v_1^i\}|$ 
2: max_cnt  $\leftarrow -1$ 
3: min_mean_error  $\leftarrow \text{INF}$ 
4: for round in  $\{1, \dots, k\}$  do
5:   sample  $\leftarrow$  samples of  $n$  index from  $\{1, \dots, N\}$ 
6:   not_sample  $\leftarrow \{1, \dots, N\} \setminus \text{sample}$ 
7:   tmp_warp_func  $\leftarrow \text{SolveWarpFunc}(\{v_1^m \mid m \in \text{sample}\}, \{v_2^m \mid m \in \text{sample}\})$ 
8:   sum_error  $\leftarrow 0$ 
9:   cnt  $\leftarrow 0$ 
10:  for c in not_sample do
11:    error  $= \|v_2^c - \text{ApplyWarpFunc}(v_1^c, \text{tmp\_warp\_func})\|_2$ 
12:    if error < threshold then
13:      cnt  $\leftarrow \text{cnt} + 1$ 
14:      sum_error  $\leftarrow \text{sum\_error} + \text{error}$ 
15:    end if
16:  end for
17:  mean_error  $\leftarrow \text{Mean}(\text{sum\_error})$ 
18:  if cnt > max_cnt or (cnt == max_cnt and mean_error < min_mean_error) then
19:    max_cnt  $\leftarrow \text{cnt}$ 
20:    min_mean_error  $\leftarrow \text{mean\_error}$ 
21:    warp_func  $\leftarrow \text{tmp\_warp\_func}$ 
22:  end if
23: end for
24: return warp_func
```

5.2 Warping Function

在本次的作業中，因為我們的圖已經先進行過 cylindrical projection 投射到相同平面上，因此我們設定 warping function 為 Affine Transformation 的形式。而 I_1 與 I_2 上

的一對 matching feature point 座標的關係則可寫成式-12

$$\begin{bmatrix} \mathbf{v}_2^i \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & dx \\ m_3 & m_4 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^i \\ 1 \end{bmatrix} \quad (11)$$

而要解出其中的六個參數，我們可以將每個matching feature point 座標的關係式組成一個 linear system

$$\begin{bmatrix} x_1^1 & y_1^1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1^1 & y_1^1 & 1 \\ & & & \vdots & & \\ x_1^N & y_1^N & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1^N & y_1^N & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ dx \\ m_3 \\ m_4 \\ dy \end{bmatrix} = \begin{bmatrix} x_2^1 \\ y_2^1 \\ \vdots \\ x_2^N \\ y_2^N \end{bmatrix} \quad (12)$$

在經過 ransac algorithm 的計算後，我們即可得到 I_2 對 I_1 的 global warping(affine) function。

6 Blending

Image透過適當的warping移動至其正確的位置以及形狀後，我們需要針對兩張影像的重疊區域進行處理，若是直接進行疊加或取代均會使得影像交界處相當不自然或是仍然有明顯的邊界區。因此，我們需要透過Image blending的過程來處理交界區的影像。

6.1 Linear blending

Linear blending的想法是給予兩張圖片上每個pixels適當的weights，使得同樣的pixels總和為1，但是又兼顧到影像的連續性，即是根據pixels較為接近哪一方，給予來自該圖片的pixel values較高的weights。

在實作上，我們考慮x方向作為分配weights的考量依據，根據每個重疊區域內的pixel和兩張圖片的水平距離來分配。

Algorithm 5 Linear blending

Input: Image_left: left image of blending region.

Input: Image_right: right image of blending region.

Output: weights for left and right images

```
1: procedure LINEARBLENDING(Image_left, Image_right)
2:   H $\leftarrow$ Height of blending region
3:   for  $y = 1 : H$  do
4:     length  $\leftarrow$ width of blending region at current  $y$ .
5:     bound_l  $\leftarrow$  left boundary of blending region at current  $y$ .
6:     bound_r  $\leftarrow$  right boundary of blending region at current  $y$ .
7:     for  $x = bound_l : bound_r$  do
8:       if  $(x, y)$  lies in blending region then
9:         weight_l[x, y]  $\leftarrow \frac{bound_r - x}{length}$ 
10:        weight_r[x, y]  $\leftarrow \frac{x - bound_l}{length}$ 
11:       end if
12:     end for
13:   end for
14: end procedure
```

Alg.5簡單地呈現出linear blending如何去處理weights分配。而實作中，須額外考慮到的問題是影像重疊區可能並不是convex set，以致於在Alg.5中所計算到的 $length \neq bound_r - bound_l$ ，實際計算中必須將這個問題進行適當的修正，否則影像得到的weights將會是discontinuous。

6.2 Poisson blending[2]

雖然使用linear blending，是一個簡單且有效的blending手段，但面臨到image本身無法完美matching，例如鬼影的產生時，會使得結果仍然不盡理想。若我們借用Poisson editing[2]在處理seamless cloning的優點，同時兼顧到整張影像以及重疊區域的變化，可以讓我們得到更加自然的疊合效果。

原作者提出此方法的理念在於，使用Laplacian operator所取出的2階變化量對於人的感知具有相當的影響力，並且疊加在原影像上並不會產生過於突兀的變化；另一方面，當給定一個區域其Laplacian以及boundary上的scalar function，此時內部區域根據Poisson's equation可以得到唯一的解。

6.2.1 Problem description

如Figure 4考慮一張image S ，待處理的interpolation的區域是 Ω ，並且給定區域外的已知pixel values function f^* 。對於每一個pixel p ，定義 N_p 為 p 四個方向上相鄰的pixel。可以將此一關係表示為: pair (p, q) such that $q \in N_p$ 。而 Ω 的boundary則定義為 $\partial\Omega = \{p \in S \setminus \Omega | N_p \cap \Omega \neq \emptyset\}$ 。目標是要找出 Ω 區域的pixel value，即 $f|_{\Omega} = \{f_p, p \in \Omega\}$ 。

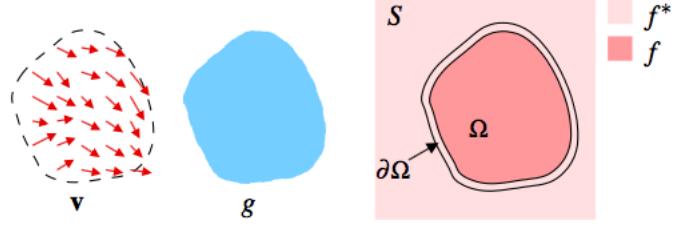


Figure 4: For image S , f is an unknown function on Ω , and f^* is a known function. \mathbf{v} is a guidance field on Ω , which might be gradient of g .

對於一個Dirichlet boundary problem，我們可以將此一離散的Poisson equation表示成optimization problem：

$$\min_{f|_{\Omega}} \sum_{(p,q) \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2, \text{ with } f_p = f_p^*, \forall p \in \partial\Omega \quad (13)$$

此處 v_{pq} 可以視為是guidance field $\mathbf{v}(\frac{p+q}{2})$ 在 $[p, q]$ 方向上的projection，也就是 $v_{pq} = \mathbf{v}(\frac{p+q}{2}) \cdot \vec{pq}$ 。並且，他的解將會滿足下列linear equation。

$$\forall p \in \Omega, |N_p|f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq} \quad (14)$$

此處的 $|N_p|$ 代表的意義是pixel p 的周遭有多少有效的pixel，也就是尚未超出影像邊界區域的pixel數目。

6.2.2 Implementation

在實作上，我們會將 g 視作是另一張image的pixel value，也就是參與blending的另一張image。因此，可以得到：

$$\begin{aligned} \mathbf{v} &= \nabla g \\ v_{pq} &= g_p - g_q, \quad \forall (p, q) \end{aligned} \quad (15)$$

進而將之整理成下列演算法：

Algorithm 6 Poisson blending

Input: Image_left: left image of blending region.

Input: Image_right: right image of blending region.

Output: Blended image

```
1: procedure POISSONBLENDING(Image_left, Image_right)
2:   Compute and locate blending region  $\Omega$ 
3:   for each  $p \in \Omega$  do
4:      $|N_p| \leftarrow 0$ ,  $v_{pq} \leftarrow 0$ ,  $l_p \leftarrow \text{image\_left}[p]$ ,  $r_p \leftarrow \text{image\_right}[p]$ 
5:     for each  $q \in N_p$  do
6:       if  $q$  is a valid pixel then            $\triangleright$  Valid means pixel has meaningful value.
7:          $|N_p| += 1$ ,  $l_q \leftarrow \text{image\_left}[q]$ ,  $r_q \leftarrow \text{image\_right}[q]$ ,  $f_q^* = 0$ 
8:         if  $q$  lies in  $\Omega$  then
9:            $v_{pq} += l_p - l_q$                    $\triangleright$  Guidance field is gradient image.
10:          else if  $q$  lies in  $\partial\Omega$  then
11:             $v_{pq} += (l_p - l_q) \text{ or } (r_p - r_q)$ 
12:             $f_q^* \leftarrow l_q \text{ or } r_q$ 
13:           $\triangleright$  Depending on which image has valid pixel.
14:        end if
15:      end if
16:    end for
17:    Construct linear equation of pixel  $p$ .
18:  end for
19:  for each channel do
20:     $x \leftarrow$  Solve linear equation.
21:  end for
22:   $f \leftarrow x$ 
23: end procedure
```

6.3 Results



(a) Linear blending



(b) Poisson blending

Figure 5: Results of our blending

Figure 5 is our implementation result, we can observe that in linear blending, due to the lack of perfect overlap between two images, there are obvious ghosting artifacts; however, after using poisson blending, we can get a more natural-looking image.

7 Drift

對於matching所產生的drift問題，我們在此處的解決方式是將第一張以及最後一張進行matching，根據計算出來的y方向shift，使用：

$$y' = y + \left(\frac{y_{shift}}{\text{total width}} \times x \right) \quad (16)$$

作為最終的warping。

此功能必須在影像是包含360度之下才能使用。

8 Final results



(a) our artifact(using translation and Poisson blending)



(b) parrington data(using translation, Poisson blending and solving drift)

Figure 6: Results of our panorama

References

- [1] Matthew Brown, Richard Szeliski, and Simon Winder, *Multi-Image Matching using Multi-Scale Oriented Patches*, Technical Report, Microsoft Research.
- [2] Patrick Perez, Michel Gangnet, Andrew Blake, *Poisson Image Editing*, Microsoft Research UK.