VFX project2

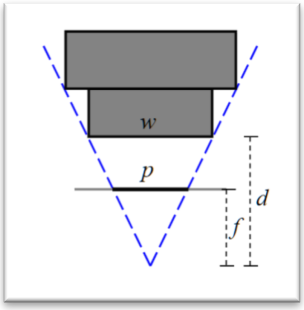
B02902015 梁智泓，B02902099 葉政杰

Cylindrical Warping:

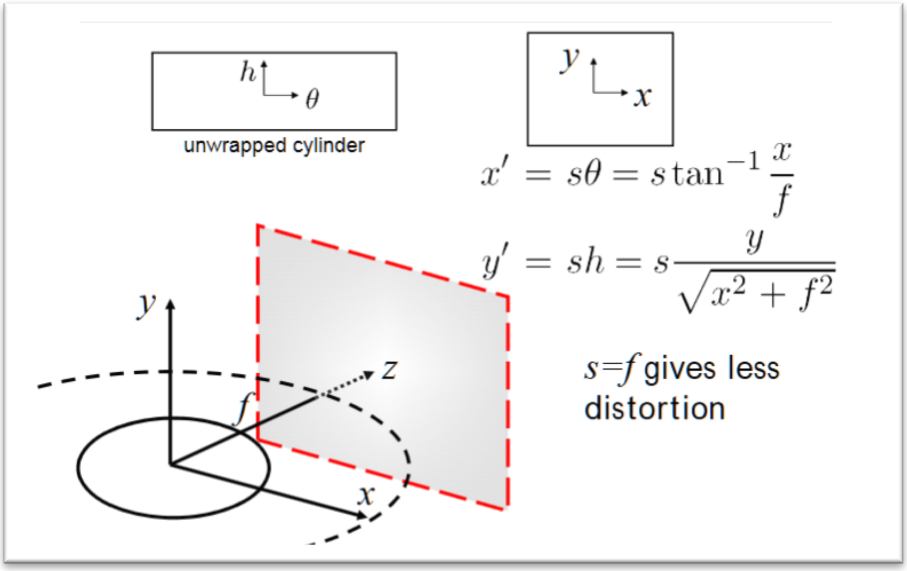
Purpose: 為了解決拼貼出的 rectilinear panorama 在照片數量上升的時候會產生嚴重的 warping 而導致效果不好，決定先採把每一張照片做 cylindrical warping 後再去拼貼，也就是老師上課提到的 cylindrical panorama 以獲得變形量較小的拼貼結果。

Implement: 要實作這部分必須先利用一些程式如 AutoStich 或是單純利用土法煉鋼測量法來計算 focal length，才能去解投影到圓柱表面上的相關座標。

土法煉鋼法：



再來借用老師上課投影片的圖表示，先利用 x 以及 focal length 去表示 theta、在用 x, y, focal length 來表示 h，便可以解出投射點 x', y'。特別注意的是 s 的值取用 focal length 時會有好事發生。



實際上 implement 時則可以選擇先將整張照片平移到中心為原點再來解投影點，最後再加回剛開始的平移較為方便。

Before Warping	After Warping

Feature Detection:

做完 Image Warping 之後，就開始對每張照片找出 feature point，然而因為 SIFT 的難度過高，也跑得較慢，所以我們最後決定使用 Harris corner feature detection。

找出 feature point 之後，會有太多我們不要的點，所以我們採用三個方式去過濾掉我們不需要的點，分別是

1. Reject Boundary: 將圖片周邊（邊框設為 6）的 feature point 過濾掉
2. Reject Low Contrast: 將圖片中對比度過低的點過濾掉
3. Reject Edge: 將圖片中是邊線的給過濾掉

完成之後，就能夠大幅度的降低 feature point 的數目，也讓每個 feature point 帶有更多的資訊。

Harris feature detection	After Rejection

Feature Description:

找出所有 feature point 之後，必須要對每個點做一個 description，讓點與點之間得以辨識，而一開始，我們是使用 SIFT 去實作。然而結果卻不盡理想，且運算需要的時間也過多（平均一張照片約 15min）。於是，我們自己發想了一個比較簡單但 robust 的 feature description，就是利用每個 feature point 周圍 13*13 的 RGB 點，表示為一個 508 (13*13*3)dim 的 vector，作為為每個點 description，這樣的好處是它的運算速度非常快，相當容易 implement，且 robust。

Feature Matching:

找到一張照片之中的所有 feature point 與各自的 description 之後，接下來是要對兩張照片之間找出彼此 match 的點，而我們是使用 L2-norm 當作兩點間的 distance，每個點對另外一張的每個點算出距離之後，由小到大做排序，找到 rank1 且跟 rank1 與 rank2 的差距夠大（rank1/rank2 < 0.8）時便可以做為 match pair。

Ransac:

找出所有的 match 點之後，在利用 ransac 將不合理的 match pair 給過濾掉，作法是任取兩個 match pair，計算在 range 中的 pair，做了多次之後，最後輸出在 range 中點最多的 pairs

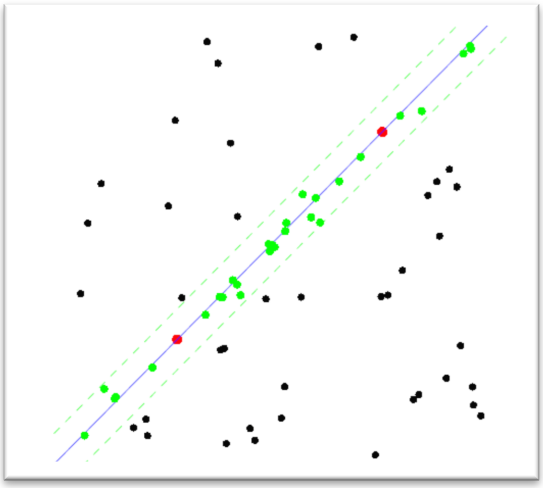


Image Matching:

Purpose: 透過上一步驟解出的 feature_point matching 來釐清兩張照片之間的關係，例如照片二可能是照片一經過了某一個 transform 之後的結果。換句話說即為解出一個 transform_matrix 可針對特徵點的配對來黏合兩張圖片。

Implement: 解一個 linear system，如以下圖為例：

$$\begin{pmatrix} x_1 & y_1 & 0 & 0 \\ 0 & 0 & x_1 & y_1 \\ x_2 & y_2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 0 & 0 \\ 0 & 0 & x_n & y_n \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{21} \\ m_{22} \end{pmatrix} = \begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ \vdots \\ x_n' \\ y_n' \end{pmatrix}$$

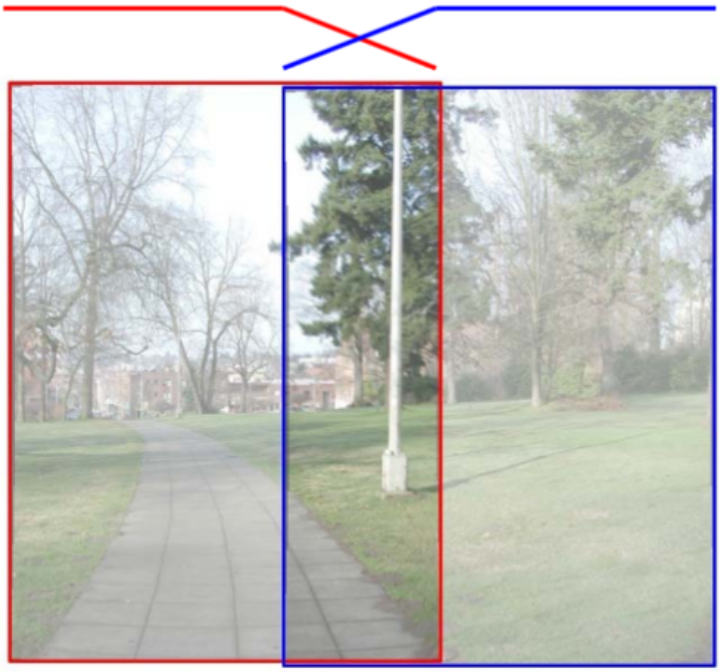
其中 x1, y1 表示在圖一的座標點，x1’, y1’表示在圖二的座標點
m11, m12, m21, m22 即是一個 M 矩陣來表示兩組座標點之間的變換。
通常有越多的 equation 解起來雖然會越複雜但結果可能較佳。
而針對 M 矩陣選擇的 degree of freedom 不同也會有更 robust 的變換型式能夠實現，但相對整個 system 也會更難 solve。

在此需特別注意老師上課投影片的這組 system 只是個範例，跟助教討論後也覺得這張圖例完全做不到我們最想要也最基本的 translation 啊(囧)，於是最後實作時選擇採用了助教推薦版本，解起來不太慢結果也還行。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Blending:

Purpose: 解出了兩張照片的 transformation 後最後一步便是把兩張圖片按照這個解來黏合，而 blending 主要想解決的問題有 parallax, lens distortion, scene motion, exposure difference 等等，但這裡假設我們只有曝光度差異需要處理。
Implement: 將疊合到的部分做兩組圖片的線性組合，scalar 分別為 1-to-0 以及 0-to-1 的斜直線等分，簡單相加後可以得到較為平滑的結果。借用老師上課的投影片，線性組合即是指圖上紅線以及藍線斜下來的部分。



Result:

相關的結果，都在 blending_result 的 folder 裡面

What we found:

1. 我們發現，blending 出來的結果與 input 的照片的好壞其實有不小的關係，當照片中的物體不夠遠，且 focal length 不夠長時，就會導致 Warping 及 blending 的結果會不夠好。
2. 除此之外，若照片中有移動的物體時，就會導致 feature matching 的結果會有問題以致最後在 blending 時會有鬼影的狀況發生。