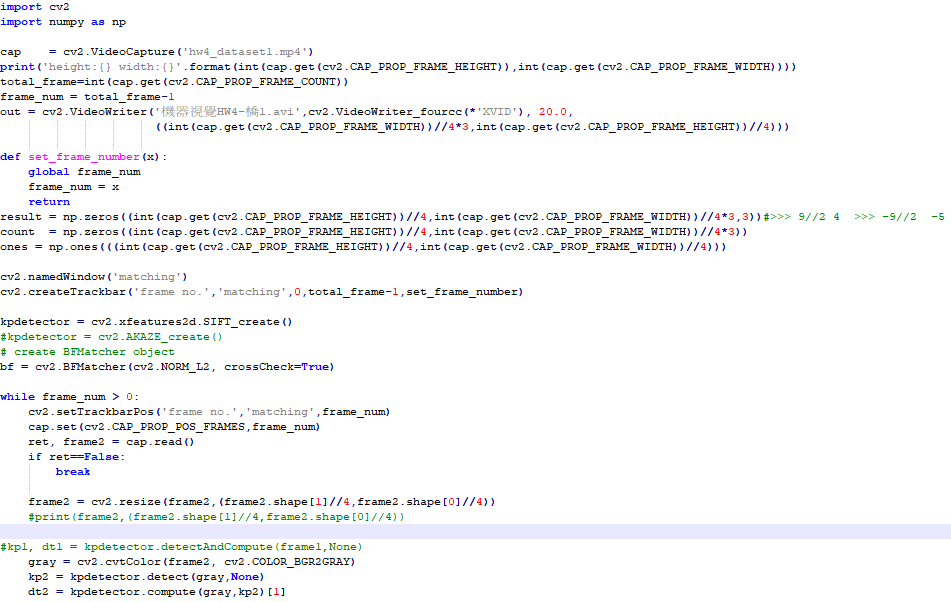
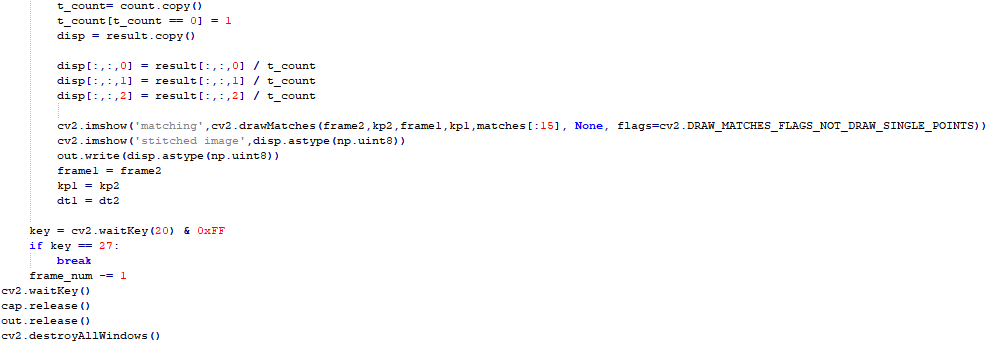
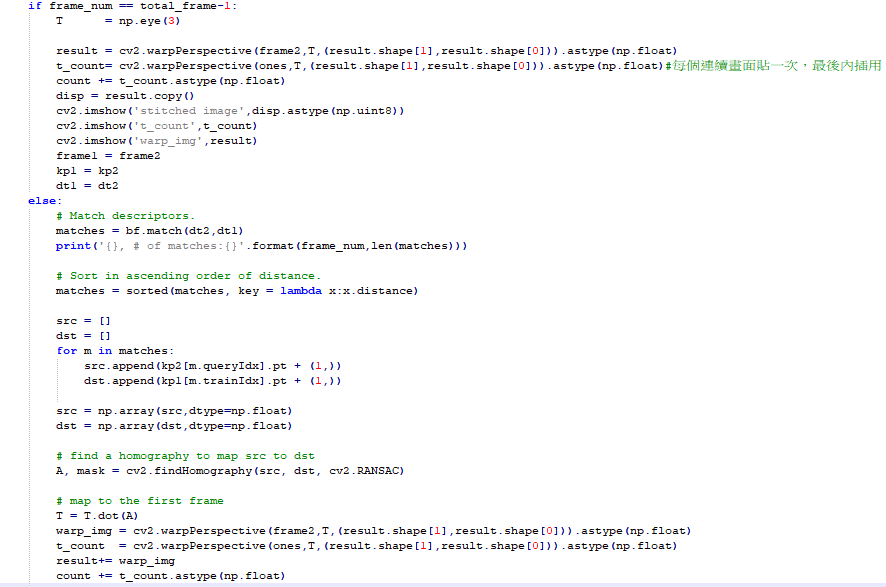
|  |
| --- |
| 機器視覺作業報告 |
| 影像拼接作業 |
|  |
|  |
| 姓名:翁偉恆  學號:00453048  日期:2019/6/9 |
|  |

|  |
| --- |
|  |

# 方法

以下為dataset1從左到右影片轉全景程式



xfeatures2d.SIFT\_create()為創造SIFT物件的指標，SIFT是一種尋找 “影像尺度不變” 特徵的物件。

BFMatcher()為創建BF匹配器物件，第一個參數為距離測試類型，cv.NORM\_L2是指用的方式算距離，第二個參數為是否要crossCheck，crossCheck是指只有 “圖片A中的第i個特徵點與圖片B中的第j個特徵點距離最近，並且圖片B中的第j個特徵點到圖片A中的第i個特徵點也是最近時(圖片A中沒有其他點到j的距離更近)” ，才會返回最佳匹配(i,j) 。

xfeatures2d.SIFT.detect()在影像中找到特徵點。

xfeatures2d.SIFT. compute()計算每個點的特徵描述，例如：位置、尺度、方向…等。

warpPerspective()第一個參數為輸入圖片，第二個參數為透視轉換矩陣，透視轉換矩陣的格式為

使x(t+1)= x(t)+ ∆x與y(t+1)= y(t)+ ∆y

第三個參數為輸出圖像的大小，t為第幾張圖像。

BFMatcher.match()第一個參數與第二個參數為兩個特徵點矩陣，用暴力法在第一幅圖像中取每個特徵點依次與第二幅圖像的每個特徵點進行距離測試，最後返回距離最近的特徵點

findHomography()第一個參數與第二個參數為兩個輸入圖片，他會幫你算出這兩張圖片之間的透視轉換矩陣

findHomography()的原理與warpPerspective()第二個參數的原理大同小異，只是warpPerspective()第二個參數為，而findHomography()是第一個參數為，第二個參數，輸出為。

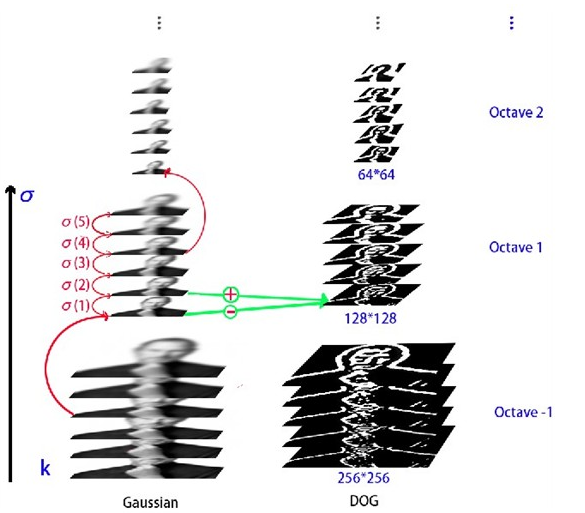
SIFT()原理：

1. 首先對欲偵測的圖片，進行多次高斯模糊。
2. 把原來的圖片減掉模糊後的圖片(如下圖)。
3. 把模糊後的圖片+ ” 原來的圖片減掉模糊後” 。
4. 繼續迭代，迭代次數如下圖，其中是基準層尺度，o為組octave的索引，s為組內層的索引。K為組內總層數的倒數，關鍵點的尺度坐標σ就是按關鍵特徵點所在的組和組內的層。



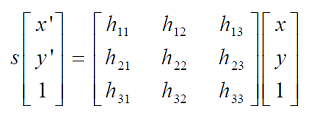


1. 迭代到一定次數時停止，並算出每個像素的值，像素有極大值的地方為特徵點。



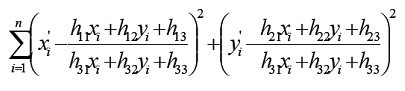
RANSAC 演算法：

RANSAC演算法用利用轉換矩陣(如下圖)，找出較匹配轉換角度的特徵點，通常令=1來標準化矩陣。由於矩陣有8個未知參數，則至少需要8個線性方程求解，對應到點位置信息上，一組點對可以列出兩個方程，則至少包含4組匹配點對。



類似

RANSAC演算法從匹配特徵點中隨機抽出4個樣本並保證這4個樣本之間不共線，計算出轉換矩陣，然後利用這個模型測試所有數據，計算滿足這個模型數據點的個數與投影誤差(即代價函數，如下圖)，若對應的代價函數最小，則此模型為最優模型。



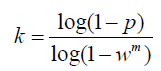
步驟：

1. 隨機從特徵點集中隨機抽出4個樣本特徵點 (此4個樣本之間不能共線)，計算出變換矩陣H。

2. 計算特徵點集中所有特徵點與矩陣H的投影誤差，若誤差小於門檻值，則加入有效特徵點集 I 。

3. 如果當前有效特徵點集I元素個數大於最優特徵點集I\_best ，則更新 I\_best = I，同時更新迭代次數k。

4. 如果迭代次數大於k，則退出 ; 否則迭代次數加1，並重複上述步驟。

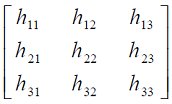


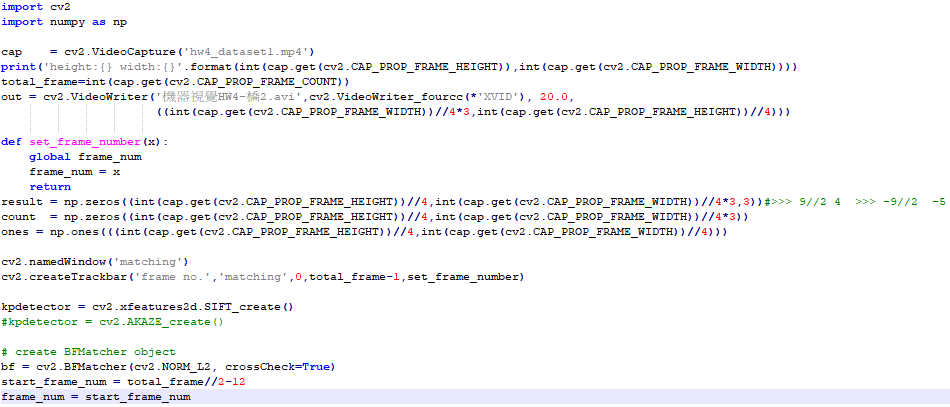
其中，p為可信度，一般取0.995；w為有效特徵點的比例 ; m為計算矩陣所需要的最少樣本數=4。

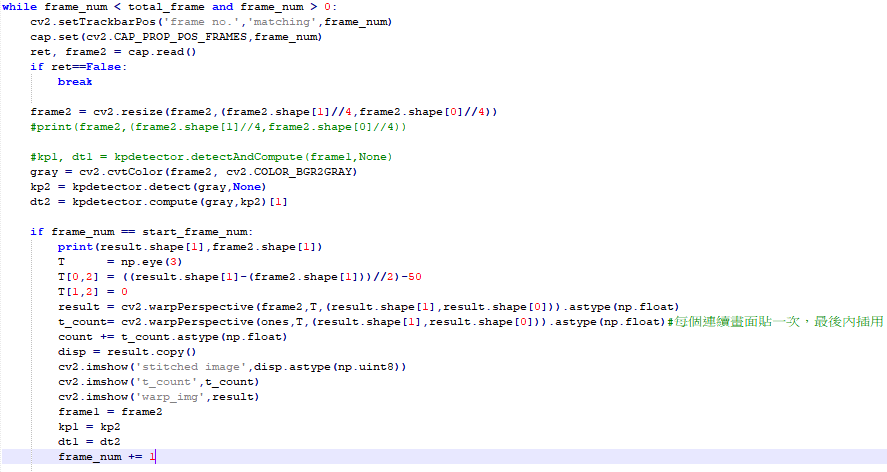
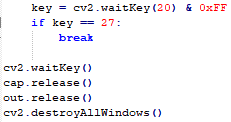
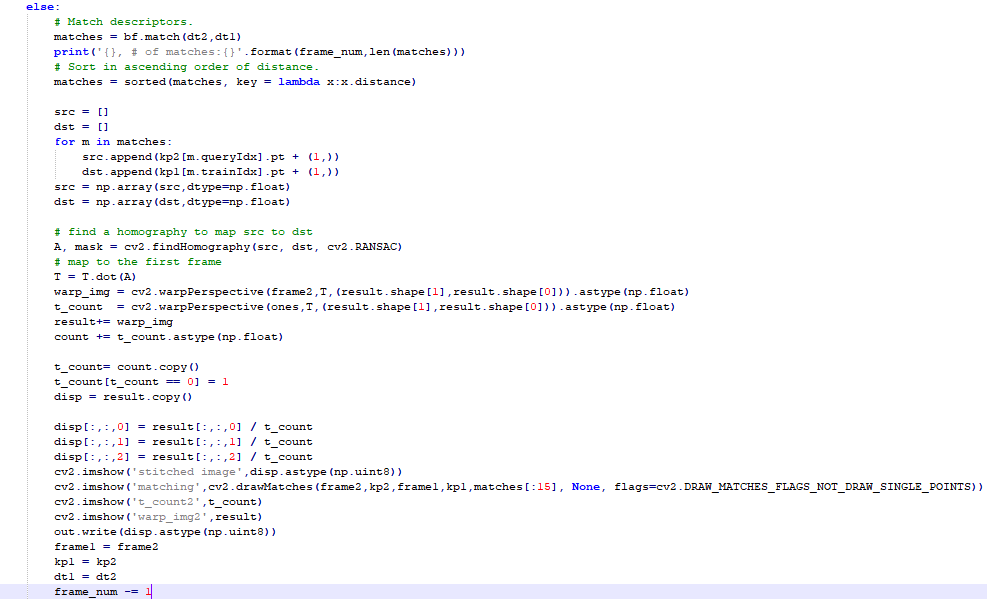
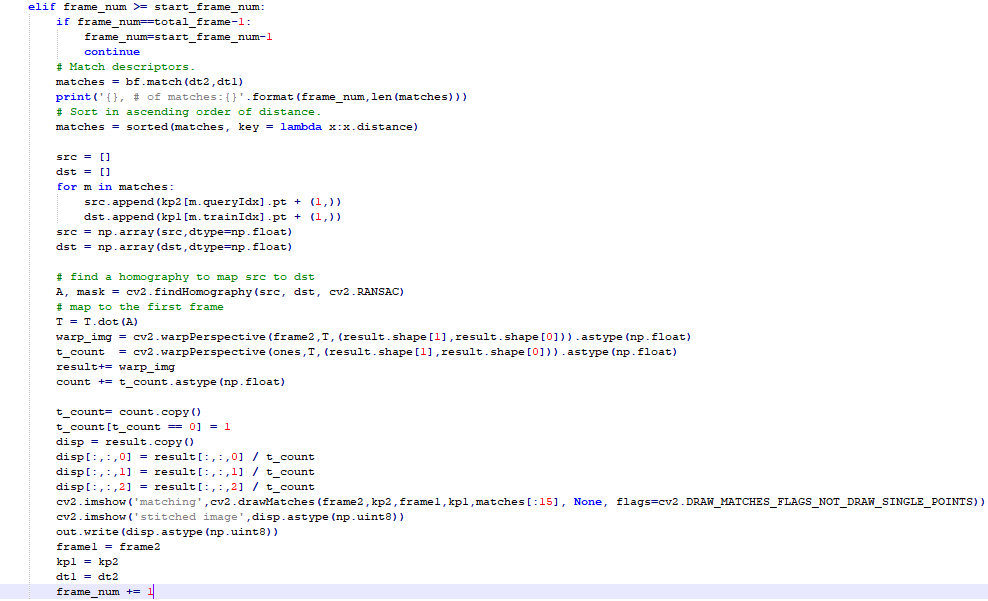
簡易演算法說明：

1. 將影片切成多張圖片
2. 選定最左邊那張為初始圖片，並尋找初始圖片的特徵點
3. 建立合適的把放出使位置的矩陣T0(程式碼中為frame\_num==total\_frame-1時，裡面的T)，將初始圖片放到合適的初始位置
4. 讀取第二張圖片，並尋找第二張圖片的特徵點
5. 比對兩張圖片的特徵點位置
6. 找出兩張圖片之間的透視轉換矩陣(程式碼中為A)
7. 找出與前一張圖片位置的透視轉換矩陣(T=T0\*A) (程式碼中為非frame\_num==total\_frame-1的T)
8. 疊合，以此類推到整堆圖片疊合結束

上圖程式的findHomography()就是利用RANSAC演算法，找出投影誤差最小的



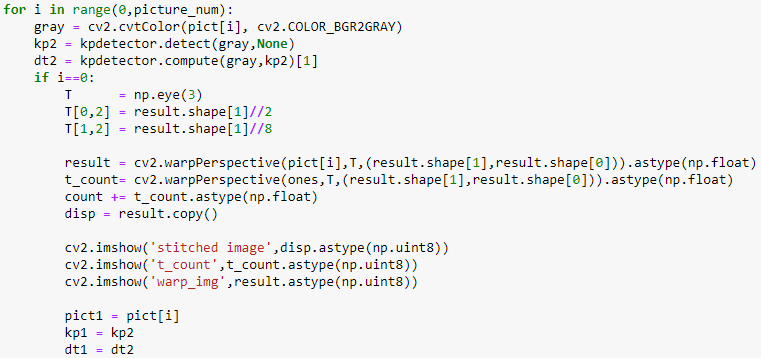
以下為dataset1從中間到兩邊影片轉全景程式

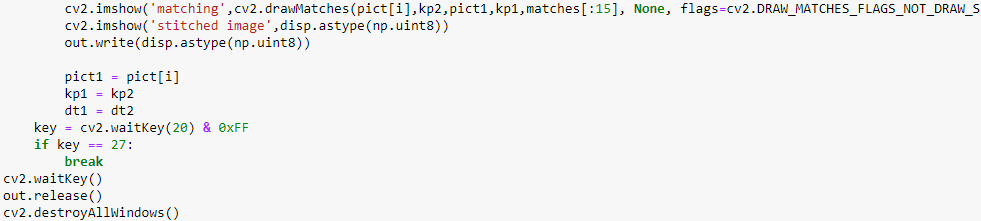
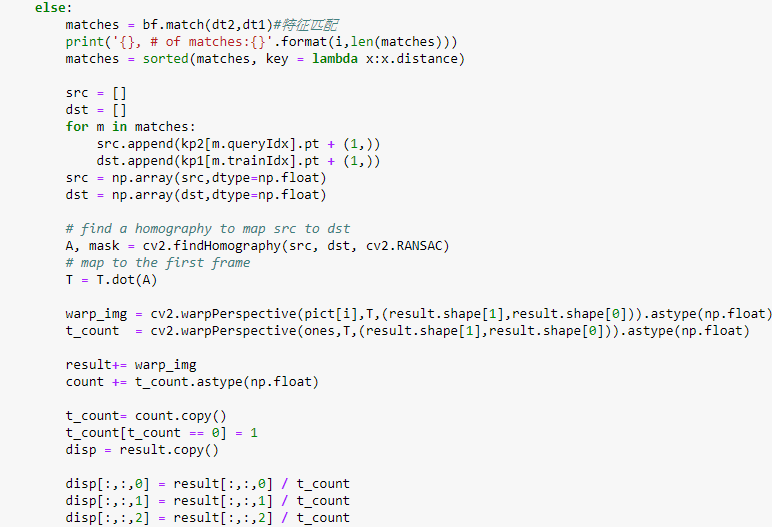
 

簡易演算法說明：

1. 將影片切成多張圖片
2. 選定最中間那張為初始圖片，並尋找初始圖片的特徵點
3. 建立合適的把放出使位置的矩陣T0(程式碼中為frame\_num==total\_frame//2時，裡面的T)，將初始圖片放到合適的初始位置
4. 讀取第二張圖片，並尋找第二張圖片的特徵點
5. 比對兩張圖片的特徵點位置
6. 找出兩張圖片之間的透視轉換矩陣(程式碼中為A)
7. 找出與前一張圖片位置的透視轉換矩陣(T=T0\*A) (程式碼中為非frame\_num== total\_frame//2的T)
8. 疊合
9. 以此類推疊合到所有比中間左邊的圖片結束
10. 回到最中間那張初始圖片
11. 再繼續疊合到所有比中間右邊的圖片結束

以下為dataset2還原披薩程式





程式碼中的t\_count為權重矩陣，像素值會介於1~2之間，1代表 ( 重疊後圖片的像素值 / 1 ) ，2代表 ( 重疊後圖片的像素值 / 2 )，主要是因為圖片拼接時，重疊的部分BGR通道值會直接相加，導致拼接後重疊地方的BGR通道像素值，分別會是 ” 原來圖片的像素值和後來的像素值較大者 ” 的1~2倍左右，所以重疊出來的部分需除以t\_count復原，但這可能會使原先的圖片亦被t\_count減弱權重，造成模糊。

由於DSC\_1169.JPG與其他張照片的匹配性較差，所以還原時捨棄此張照片。以上順序是我個人認為還原較不失真，也就是最後幾張照片與第一張照片重疊性及大小失和性較小的排序。

簡易演算法說明：

1. 選定初始圖片，並尋找初始圖片的特徵點(本次選DSC\_1167.JPG為初始圖片)
2. 建立合適的把放出使位置的矩陣T0(程式碼中為i==0時，裡面的T)，將初始圖片放到合適的初始位置
3. 讀取第二張圖片，並尋找第二張圖片的特徵點
4. 比對兩張圖片的特徵點位置
5. 找出兩張圖片之間的透視轉換矩陣(程式碼中為A)
6. 找出與前一張圖片位置的透視轉換矩陣(T=T0\*A) (程式碼中為非i==0的T)
7. 除以權重矩陣t\_count
8. 疊合，以此類推到整堆圖片疊合結束

# 結果

<https://www.youtube.com/watch?v=F2vHk1bgdWs&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=3>

以上連結為dataset1 ”從左到右” 影片轉全景的影片，個人覺得邊右邊的影像明顯被拉大了，這可能和warpPerspective()一直帶入同一個轉換矩陣T有關係，假設其中有一次的轉換矩陣T把原圖稍微拉開了一下，則下一次匹配特徵點時，”下一次原圖的特徵點” 就會和 “這次被拉開距離的特徵點” 進行比對，所對應出來的關係就會以 “這次被拉開距離的特徵點” 為欲被拼接圖片的特徵點，使得找 ”下一次原圖的特徵點” 可能會較容易找到相差距離較大的特徵點，繼續拉大距離，繼續迭代，所以到最後的展開圖會漸進式的變寬，產生失真。

<https://www.youtube.com/watch?v=wYG6OdlMPP8&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=4>

以上連結為dataset1 “從中間到兩邊” 影片轉全景的影片，這次兩邊的拉寬失真有明顯的取得平衡，使得左邊的失真不會太大，右邊的失真也不會太大，值得注意的是，frame\_num為中值的那張照片，不一定式展開全景模式後的中間那部分，這要看原始影像的瞬時攝影移動速率來決定，而這個範例中，如果初始影像取 ” frame\_num為中值的那張照片” 的話，左邊的影像全部展開成全景模式後，還是沒辦法把輸出影像左邊填滿的，如下圖。所以初始圖片的選取要看輸入影像為何，而做適當的選擇。

<https://www.youtube.com/watch?v=8v7GMfjvXaQ&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=5>

以上連結為dataset2還原披薩的影片，順序為：

DSC\_1169.JPG

DSC\_1162.JPG

DSC\_1163.JPG

DSC\_1164.JPG

DSC\_1165.JPG

DSC\_1166.JPG

DSC\_1167.JPG

DSC\_1168.JPG

DSC\_1158.JPG

DSC\_1159.JPG

DSC\_1160.JPG

DSC\_1161.JPG

這份影片為了考慮到DSC\_1169.JPG，也就是中間那一張的圖片，由於中間那張的特徵點與其他張圖片的特徵點匹配性較低，所以我個人把它安排在第一張或最後一張，然後其他張以逆時針方向合併，我個人測出來的的結果是，DSC\_1169.JPG只能與DSC\_1160.JPG、DSC\_1161.JPG、DSC\_1162.JPG、DSC\_1163.JPG、DSC\_1165.JPG、DSC\_1166.JPG匹配，而在這些能夠匹配的圖片中，個人覺得以DSC\_1169.JPG為第一張，DSC\_1162.JPG為第二張，接著逆時針拼接的順序較不失真，只是最後幾張拼接的時候，會與原來的圖片重疊，導致原來的圖片會被後來的圖片，相加除以權重後，模糊化。

<https://www.youtube.com/watch?v=b2dH9R79PkU&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=6>

以上連結為dataset2還原披薩的影片，順序為：

DSC\_1159.JPG

DSC\_1160.JPG

DSC\_1161.JPG

DSC\_1162.JPG

DSC\_1163.JPG

DSC\_1164.JPG

DSC\_1165.JPG

DSC\_1166.JPG

DSC\_1167.JPG

DSC\_1168.JPG

DSC\_1158.JPG

這份影片捨棄了DSC\_1169.JPG，也就是中間那一張的圖片，我個人想試試看，如果把所有圖片之中，與其他圖片匹配性最差的DSC\_1169.JPG刪除之後，拼接過後的影片是否有明顯的改善，拿去試了一下，結果最後幾還是有重疊到，不過重疊到的面積有稍微變小，至少披薩的餡感覺沒有重疊到，這對我來說已經算是一個不錯的進展，還有就是披薩很明顯的變扁了，然而奇怪的是，我個人覺得稍微扁一點的披薩更有真實感，所以我就沒把 “ 變扁” 這件事當成極大的失真，因此這次嘗試仍然被我歸類為nice try

<https://www.youtube.com/watch?v=7hgZO61K7Jc&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=7>

以上連結為dataset2還原披薩的影片，順序為：

DSC\_1167.JPG

DSC\_1168.JPG

DSC\_1158.JPG

DSC\_1159.JPG

DSC\_1160.JPG

DSC\_1161.JPG

DSC\_1162.JPG

DSC\_1163.JPG

DSC\_1164.JPG

DSC\_1165.JPG

DSC\_1166.JPG

DSC\_1169.JPG

這是一個很特別的測試，不是說他的效果有多好，而是在這一筆測試中，匹配性較差的DSC\_1169.JPG，竟然能完全沒有影響力，也就是說，去掉DSC\_1169.JPG，和保留DSC\_1169.JPG的輸出結果，用眼睛看幾乎一模一樣，這可能是因為DSC\_1169.JPG是最後一張拼接上去的圖片，再加上拼接到DSC\_1169.JPG的時候，匹配的特徵點，剛好有與其他拼接好圖片的特徵點有較高的匹配率，因此剛好完美的完全卡在原圖的某段區域，而後來的圖和原來拼接的圖又看起來一模一樣，相加在除以權重矩陣後，大約等於原圖，所以才有如此特別的性質，但可惜的是這個順序除了DSC\_1169.JPG之外，其他最後幾張有嚴重重疊，因此輸出結果不太好看。

<https://www.youtube.com/watch?v=VaD3KKlQM7k&list=PLBTefedmJ955eKkALh2rPBbSodZzOiqbt&index=8>以上連結為dataset2還原披薩的影片，在經過無數次的try and error之後，我個人覺得匹配時捨棄掉DSC\_1169.JPG這張圖片會始還原出的披薩比較不那麼失真，個人對失真的定義為 ”後幾張照片與第一張照片重疊”、“匹配後，最後幾張照片與第一張照片大小有明顯差異” 以及 “還原後，披薩的傾斜角度”，很不幸的是，這範例中，無論怎麼還原，都無法一起減小以上三種情狀，角度對上了，最後幾張就和第一張披薩重疊了，或者最後幾張照片與第一張照片大小減少了，披薩就重疊了，因此經過地毯式嘗試後，個人認為

DSC\_1160.JPG

DSC\_1161.JPG

DSC\_1162.JPG

DSC\_1163.JPG

DSC\_1164.JPG

DSC\_1165.JPG

DSC\_1166.JPG

DSC\_1167.JPG

DSC\_1168.JPG

DSC\_1158.JPG

DSC\_1159.JPG

的拼接順序比較稍微能在以上三種失真中取得平衡，但結果就見仁見智，個人認為 ”照片稍微傾斜” 為所有失真中傷害最小的

# 結論

說明心得，例如完成甚麼，學到那些技巧。

我覺得本次作業非常有意義，能讓我們了解如何用opencv找到特徵點，以及其背後的演算法，並且還運用特徵點，完成一些影像拼接的應用。在數學方面，讓我最為印象深刻的是透視轉換矩陣的原理，它能淺顯易懂的表示出兩張圖片的位移、縮放、和旋轉的關係。在本次作業中，我深深的了解到輸入的重要性，特別是還原披薩的那部分，那初始圖片的選擇讓我try and error了好久，特別是只拍到披薩中間的那一張，它上一張圖片和下一張圖片的選擇區間較大，且為特徵點比對方向較雜亂的部分，所以會嚴重影響輸出的形狀，此為調整輸出最為關鍵的部分。

# 參考文獻

(如果有參考文獻, 請列出)

<http://monkeycoding.com/?tag=sift>

<https://zhuanlan.zhihu.com/p/36123150>

<https://www.jianshu.com/p/95c4890c486b>

<https://www.jianshu.com/p/ed57ee1056ab>

<https://blog.csdn.net/GAN_player/article/details/78285771>

<https://www.cnblogs.com/Jessica-jie/p/8622449.html>

<https://blog.csdn.net/zjc156m/article/details/80241405>

<https://blog.csdn.net/pi9nc/article/details/23302075>

<https://blog.csdn.net/luoshixian099/article/details/50217655>