Introduction to Multimedia Homework #3

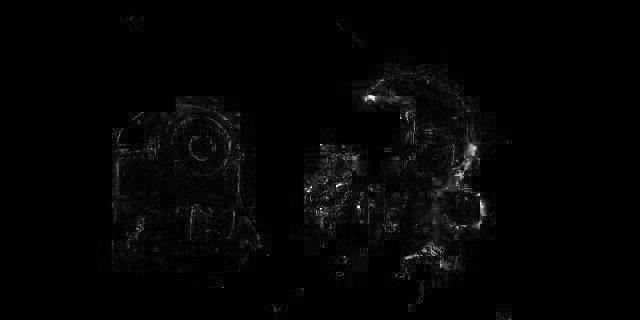
103062318蔡尚倫

Q1.( 2D logarithmic search method簡寫成2D--SEARCH)

a. Show the residual images when the target image is frame05073.jpg for all the above combinations. (8 images)文字下方代表該圖片 B=Block Size p=Search Range

**FULL SEARCH B=8 p=8**

**FULL SEARCH B=16 p=8**



**FULL SEARCH B=8 p=16**



**FULL SEARCH B=16 p=16**



**2D--SEARCH B=8 p=8**



**2D--SEARCH B=16 p=8**



**2D--SEARCH B=8 p=16**

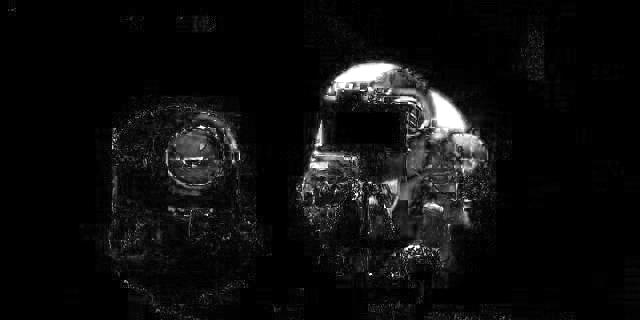


**2D--SEARCH B=16 p=16**

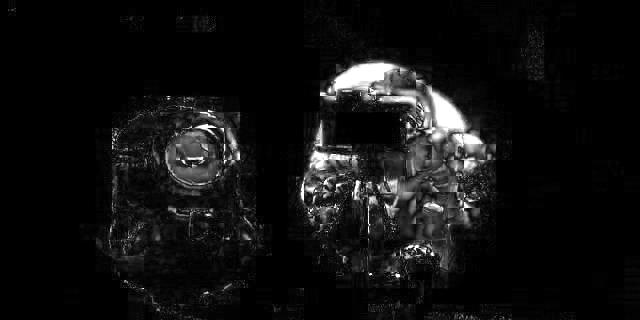


b. Show the residual images when the target image is frame05081.jpg for all the above combinations. (8 images) 文字下方代表該圖片B=Block Size p=Search Range

**FULL SEARCH B=8 p=8**

****

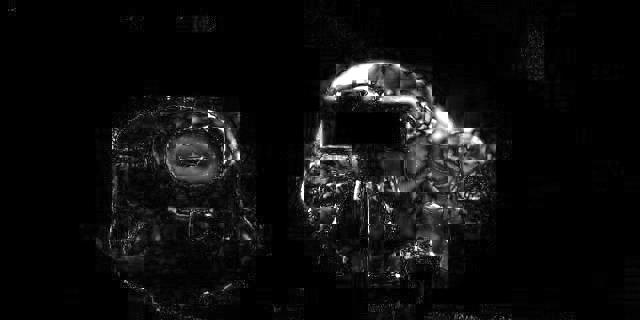
**FULL SEARCH B=16 p=8**



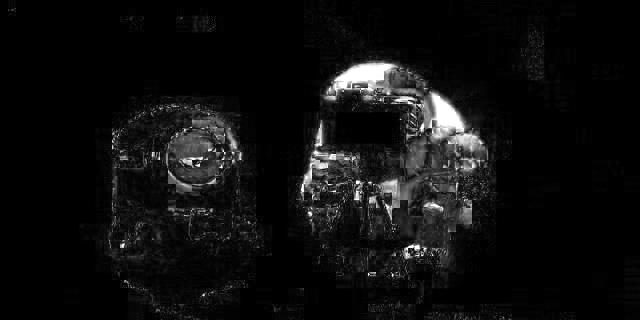
**FULL SEARCH B=8 p=16**

****

**FULL SEARCH B=16 p=16**



**2D--SEARCH B=8 p=8**



**2D--SEARCH B=16 p=8**



**2D--SEARCH B=8 p=16**



**2D--SEARCH B=16 p=16**



c. Show and compare the total SAD values for all the results in (a) and (b). B=Block Size p=Search Range

**Show**

(a)

The reference image is frame05072.jpg

The target image is frame05073.jpg

FULL SEARCH B=8 p=8 TOTAL SAD:3027.929412

FULL SEARCH B=16 p=8 TOTAL SAD:3516.972549

FULL SEARCH B=8 p=16 TOTAL SAD:2972.760784

FULL SEARCH B=16 p=16 TOTAL SAD:3497.654902

2D--SEARCH B=8 p=8 TOTAL SAD:3282.972549

2D--SEARCH B=16 p=8 TOTAL SAD:3674.929412

2D--SEARCH B=8 p=16 TOTAL SAD:3306.611765

2D--SEARCH B=16 p=16 TOTAL SAD:3678.400000

(b)

The reference image is frame05072.jpg

The target image is frame05081.jpg

FULL SEARCH B=8 p=8 TOTAL SAD:11693.690196

FULL SEARCH B=16 p=8 TOTAL SAD:13903.788235

FULL SEARCH B=8 p=16 TOTAL SAD:9041.898039

FULL SEARCH B=16 p=16 TOTAL SAD:11607.898039

2D--SEARCH B=8 p=8 TOTAL SAD:12128.490196

2D--SEARCH B=16 p=8 TOTAL SAD:13824.560784

2D--SEARCH B=8 p=16 TOTAL SAD:10623.835294

2D--SEARCH B=16 p=16 TOTAL SAD:12483.407843

**Compare**

同演算法不同Block size與不同Search range，由小到大排序SAD，大致是B=8 p=16 < B=8 p=8 < B=16 p=16 < B=16 p=8

原因應該是：

Block size如果取16的話，切割整個圖片後，如果是從一堆非常不像的Block中選SAD最小的，由於一次性選擇一個Block的關係，造成填補16\*16個都很不像的pixel，所以會與原本target image差很多，所以B=8填補效果會好於B=16。

Search range則是很直覺的，先考慮圖片前後的相關性，某個Block在frame\_i移動到frame\_i+1(這裡先不考慮切換場景的情況，因為測試圖片沒有這種情況)，如果你搜尋範圍比較大，那麼你就有越大的機會找到該Block在下一個frame的位置，所以p=16填補效果會好於p=8。

不同的演算法，由小到大排序SAD，大致是FULL SEARCH < 2D—SEARCH

原因應該是：

2D—SEARCH是為了加速尋找速度的演算法，所以無法像FULL SEARCH一樣，將範圍內的部分搜尋一遍，會有找不夠好的情況發生，就是你固定選取的參考Block不是最佳填補Block，所以填補效果FULL SEARCH會好於2D—SEARCH。

不同的target image，有小到大排序SAD，大致是frame05073<frame05081

原因應該是：

如果reference image與target image相差太多個frame，一是你的Search range不太可能找到相似的Block(Object可能已經移動很多，像是測試圖片中右邊小小兵手拿的相機)，除非你Search整張圖片才有機會，二是可能經過這麼多個frame可能早就已經切換場景了，就不太可能填補出圖片，所以填補效果frame05073會好於frame05081。

d. Show the PSNR of all the results in (a) and (b), and discuss the relation with the SAD. B=Block Size p=Search Range

**Show**

(a)

The reference image is frame05072.jpg

The target image is frame05073.jpg

FULL SEARCH B=8 p=8 PSNR:36.303438

FULL SEARCH B=16 p=8 PSNR:34.532875

FULL SEARCH B=8 p=16 PSNR:36.513541

FULL SEARCH B=16 p=16 PSNR:34.569724

2D--SEARCH B=8 p=8 PSNR:35.258490

2D--SEARCH B=16 p=8 PSNR:34.051036

2D--SEARCH B=8 p=16 PSNR:35.201534

2D--SEARCH B=16 p=16 PSNR:34.041279

(b)

The reference image is frame05072.jpg

The target image is frame05081.jpg

FULL SEARCH B=8 p=8 PSNR:24.511646

FULL SEARCH B=16 p=8 PSNR:23.288301

FULL SEARCH B=8 p=16 PSNR:27.315794

FULL SEARCH B=16 p=16 PSNR:25.205202

2D--SEARCH B=8 p=8 PSNR:24.324987

2D--SEARCH B=16 p=8 PSNR:23.461490

2D--SEARCH B=8 p=16 PSNR:25.737785

2D--SEARCH B=16 p=16 PSNR:24.476057

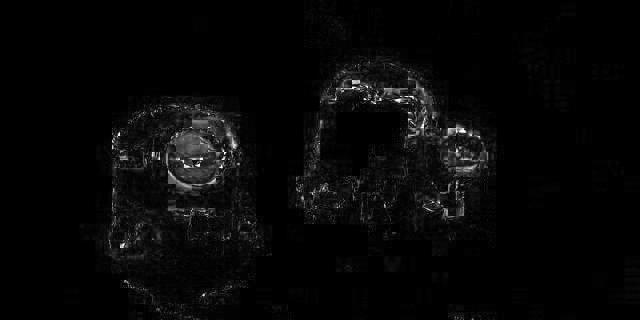
**Diucuss**

PSNR的值越大，代表兩張圖片的相似度越高，而SAD的值越大，則代表兩張圖片的相似度越低，所以PSNR與SAD是負相關的兩種數值，代表的意義都是在描述兩張圖片之間的相似程度，只是數值大小代表的意義截然不同。

Q2. ( 2D logarithmic search method簡寫成2D--SEARCH)

a. Show the residual images and total SAD value when the target image is frame05081.jpg in search range p=8, 8x8 macroblock size and 2D logarithmic search method. (1 image) 文字下方代表該圖片 B=Block Size p=Search Range

**Bi2D--SEARCH B=8 p=8 TOTAL SAD:5755.235294 PSNR:30.565988**

****

b. Discuss the results from (a) with the problem 1 of the same settings (p=8, 8x8 macroblock size, 2D logarithmic search method).

**Discuss**

Result from(a)

Bi2D--SEARCH B=8 p=8 TOTAL SAD:5755.235294 PSNR:30.565988

Problem 1 of the same settings

2D--SEARCH B=8 p=8 TOTAL SAD:12128.490196 PSNR:24.324987

由數據很明顯看出，無論是在SAD與PSNR的表現上，bi-directional prediction較優，最基本的原因可能是，能參考的填補Block更多，那麼尋找到最佳填補Block只會越好，最差也只會與單方向的Problem 1 of the same settings一樣(如果全部取用frame72的話)，那麼再從frame的變化來分析，在frame72的中找不到的最佳填補Block，可能會在frame85中找到最佳填補Block，我拿這次的測試圖片來講解，左邊的小小兵主要取到的Block來自frame72，而如果只用frame72預測右邊小小兵舉起相機的動作的話，就沒辦法比frame85右邊已經拿起相機的小小兵來的好，所以使用這種bi-directional能從前後尋找到較好的填補Block。

Q3. ( 2D logarithmic search method簡寫成2D--SEARCH)

a. Analyze the theoretical time complexity for the two search algorithms.

rows : 圖片行的數目

cols : 圖片列的數目

range : 搜尋範圍

Time complexity of FULL SEARCH

rows\*cols\*(2\*range+1)^2 = O(rows\*cols\*ranges^2)

Time complexity of 2D—SEARCH

rows\*cols\*()^2 = O(rows\*cols\*()^2)

分析按照實際的CODE，觀察最主要的迴圈，其餘的部分可以當作時間複雜度的常數，從FULL SEARCH開始，觀察最大的主迴圈有四層，所以就能計算出FULL SEARCH的時間複雜度，而2D—SEARCH則是改善range^2的時間複雜度，改善方式使用類似二分搜尋法，每次搜尋range能減半(但沒有比FULL SEARCH結果好的原因也在這)，所以效能提升大致range^2/log(range)^2倍。

b. Measure the execution time required for the two search algorithms with the two different search range sizes (p=8 and p=16).以Q1.a測試 B=Block Size p=Search Range

FULL SEARCH B=8 p=8 Elapsed time is 6.790916 seconds.

FULL SEARCH B=8 p=16 Elapsed time is 24.874815 seconds.

2D--SEARCH B=8 p=8 Elapsed time is 0.583278 seconds.

2D--SEARCH B=8 p=16 Elapsed time is 0.720939 seconds.

c. Compare and discuss the execution time with the theoretical time complexity.

B=Block Size p=Search Range

在相同的B與p條件下，明顯的2D—SEARCH跑得比FULL SEARCH快。相同演算法下，Search range較大的需要花較久的時間。由剛剛理論的效能提升為

range^2/log(range)^2，計算大概為8^2/3^2 = 7.11倍，而由時間觀察到的效能提升為6.790916/0.583278，計算大概為11.64倍，我想這會比理論還大，是因為沒考慮到Big O中的常數項以及FULL SEARCH在設計時，勢必會跑過四次迴圈，所以應該可以從Big O改成theta，而2D—SEARCH則可能更快找到相似的點(如果每次都剛好中心點的SAD最小，就能將range/2)

Q4. ( 2D logarithmic search method簡寫成2D--SEARCH)

how you implement the methods

FULL SEARCH：(fullsearch.m) 起始點用左上角的座標代表整個Block

讀進target image與reference image(再傳入前已經轉成double型態)，取得圖片(任一皆可)的rows cols heights，開始跑四層for-loop去尋找填補的Block起始點(Block左上角的座標)。前面兩層loop是i=0:block sizes:rows-blocks sizes和j=0:block sizes:rows-blocks，這裡從i=0和j=0開始的原因是方便取得Block，只需要(i+1,i+block sizes,:)，式子就不會有減有加的，另外每sizes個數跑迴圈，就能準確到達想要填補的Block起始點，最後到rows-blocks sizes，如果到rows的話會多做一次。

接著將上下左右的邊界找好，我這裡就開始判斷有沒有超過原本圖片的邊界，有的話就做修正，假如小於0，就修正為0，假如大於rows或cols，就修成為rows-block sizes或cols-block sizes(用上方(i+1,i+block sizes,:)這種方式取的時候，就保證部會超過rows或cols)。

最後跑兩層loop跑上下range，將range中所有的Block個別算出SAD，記住最小的SAD，那麼這個Block就是這Block最佳的填補Block。

算完後得到result\_img，與原本的target\_img相減以及取絕對值，並將三維度的值對應相加，得到residual\_img，這裡我將result\_img與residual\_img都傳出function。

2D—SEARCH：(Mysearch2D.m) 起始點用左上角的座標代表整個Block

前面兩層loop與FULL SEARCH相同就不贅述。

接著開始跑一個while loop，直到搜尋半徑砍半成1為止，找的方式跟助教一樣，找自己一個Block起始點與搜尋半徑上下左右四個Block的起始點，這裡當然也要判斷邊界，亦即不要超過原本圖片的邊界與Serach range的邊界，將五個Block的SAD算完後比較最小的，如果SAD最小的還是在自己這個Block的起始點的話，搜尋半徑就砍半，直到砍到剩下1就跳出loop

最後在該Block最近的8個Block比較，擁有最小SAD者，就是這個Block就是這Block最佳的填補Block。

算完後得到result\_img，與原本的target\_img相減以及取絕對值，並將三維度的值對應相加，得到residual\_img，這裡我將result\_img與residual\_img都傳出function。