Introduction to Multimedia HW4 Report

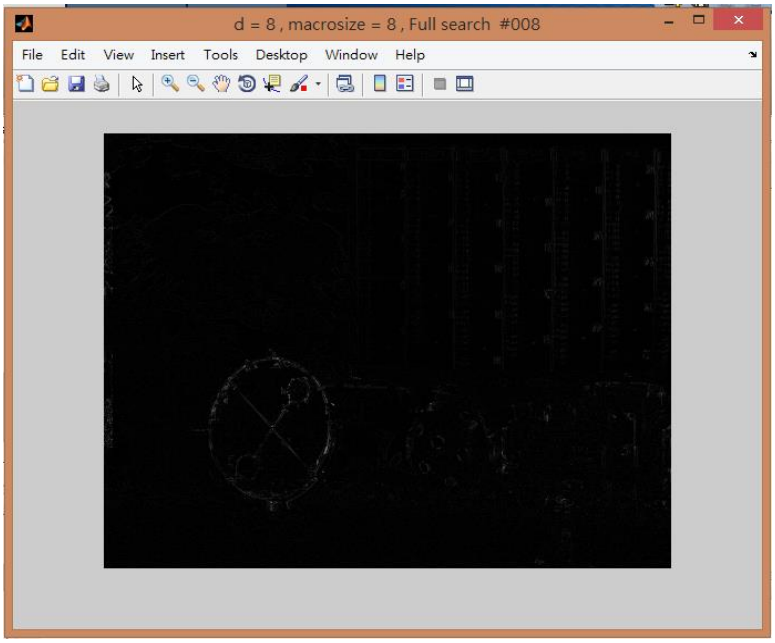
102062209 邱政凱

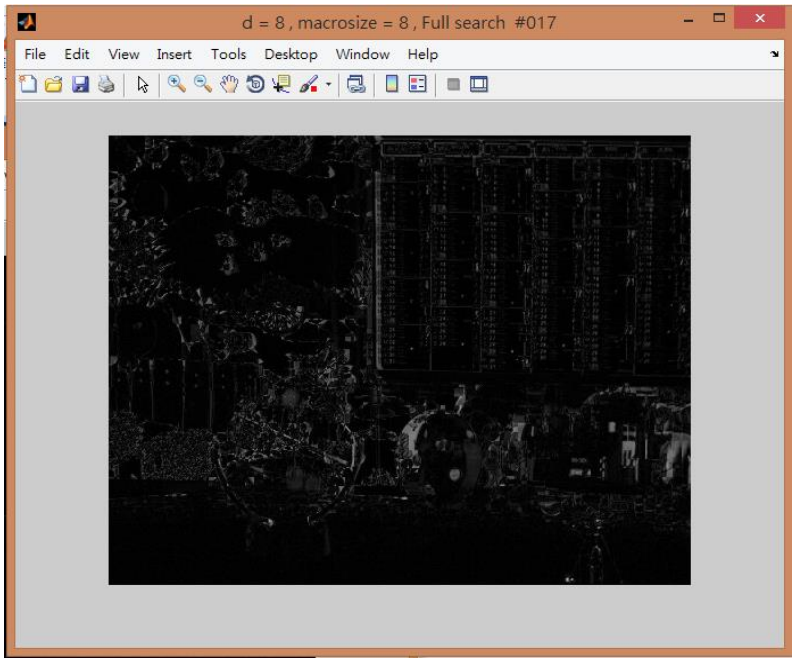
這次的作業要我們實作影像壓縮中常用到的 Motion Estimate。作業給的範例圖片我覺得還挺有意思的，少少幾張圖中就包含了各種方向和方式的物體移動。這次實作上最麻煩的部分就是題目要求要比較的東西太多了，儘管 code 幾乎一樣只是參數換一換，但是總共 2*2*2(兩種 search range、兩種 macro block size、兩種 search method)共八種方式去針對全部 11 張圖片去做 Motion Estimate 和 Error Prediction 也著實讓人費了一番功夫。

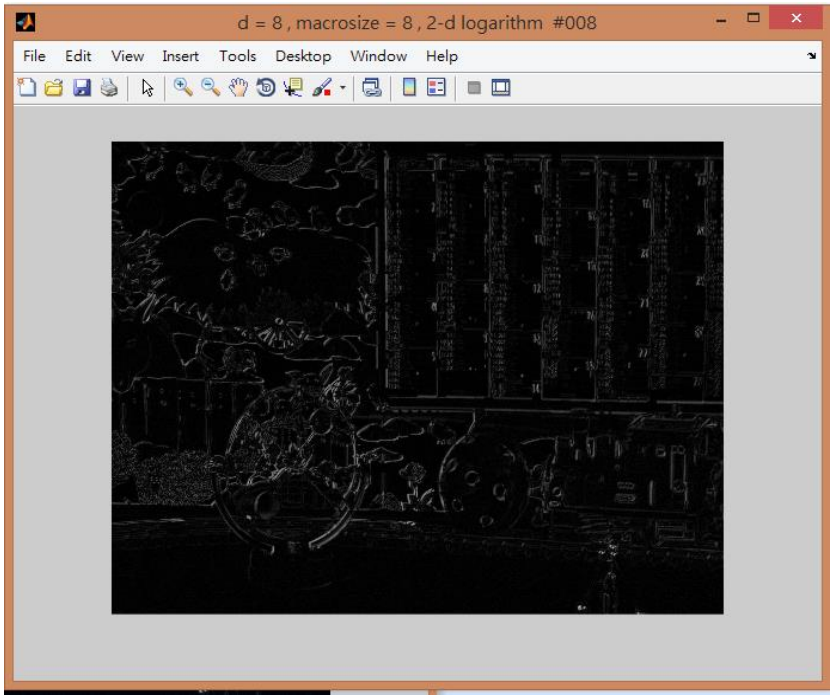
題目(a)要求我們去比較各種不同參數和演算法下的 SAD (Sum of Absolute Difference)值。先拿使用 Full Search Algorithm 的四項的 SAD 拿來比對，會發現幾乎對每個 frame 來說，d=16，macro block size=8 的 SAD 是最小的，而 d=8 ，macro block size=16 的 SAD 值是最大的，另外兩個則介於中間；若看的是所有 frame 加總的 SAD 值，會發現 d=8,size=16 > d=8,size=8 > d=16,size=16 > d=16,size=8。接著拿使用 2-D logarithm search algorithm 的四項的 SAD 來做比對，會發現若觀察的是最前面幾個 frame 的話結果會有點不一樣，SAD 的分布較沒有規律；不過到了後半的 frames，會發現 SAD 值的分布跟之前一樣全部變成 d=8,size=16 > d=8,size=8 > d=16,size=16 > d=16,size=8；而

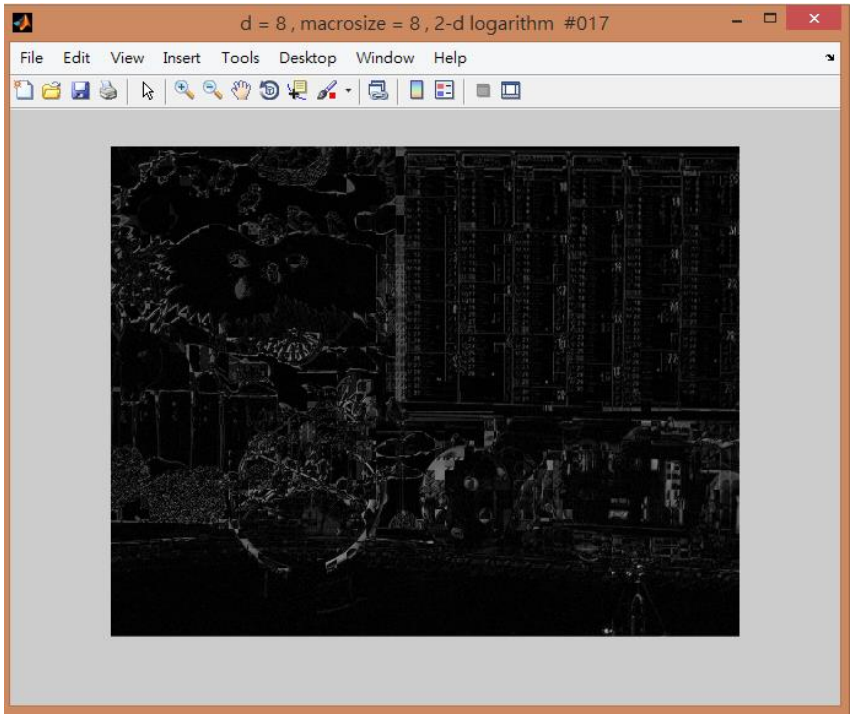
看所有 frames 的 SAD 加總值的話結果也跟上述一樣。

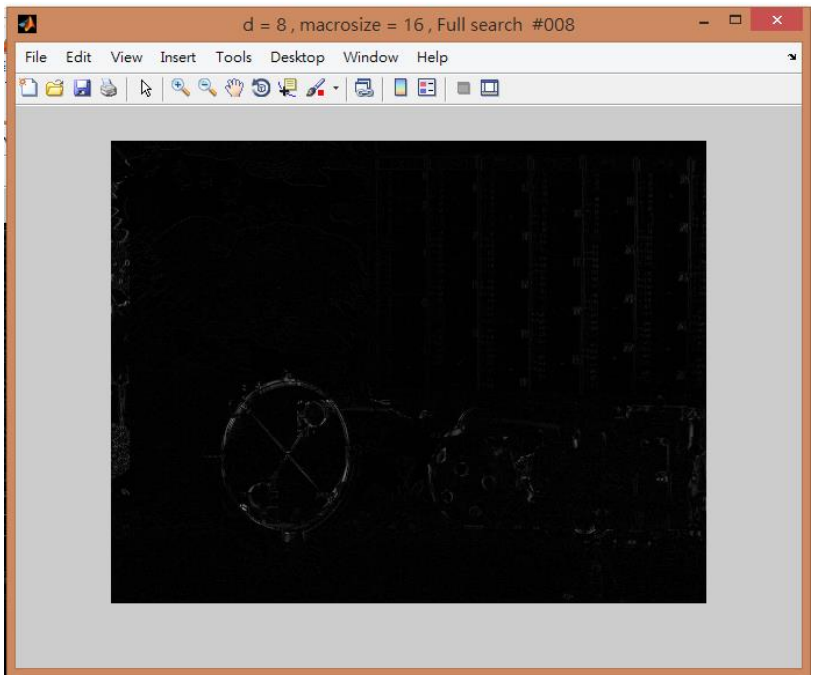
題目所要求的 Residual image 請參考如下。(視窗名稱有顯示它是代表哪一種算法跟參數的結果)

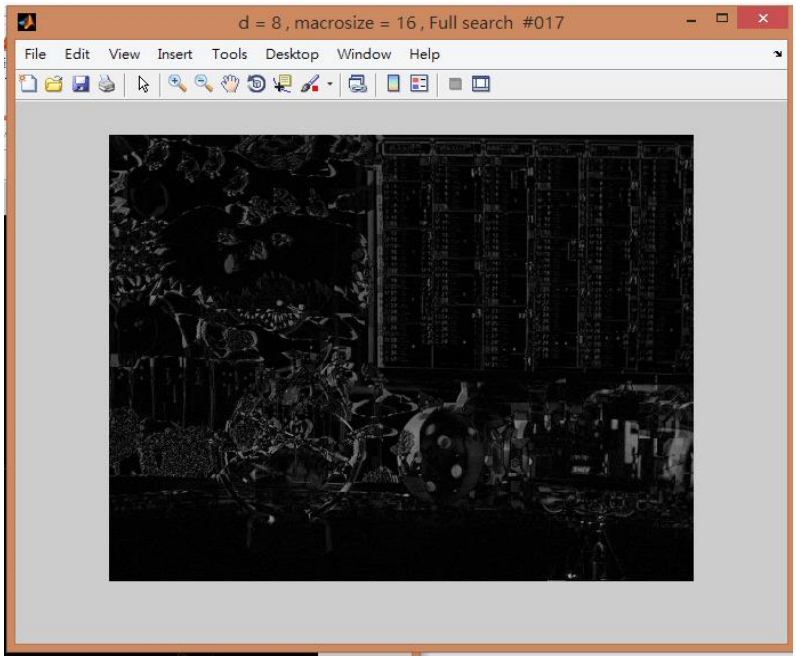


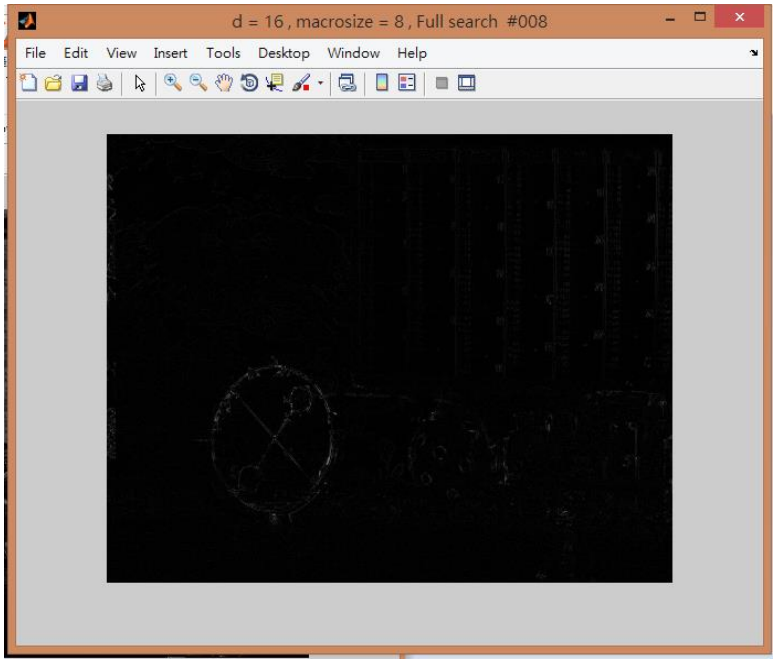


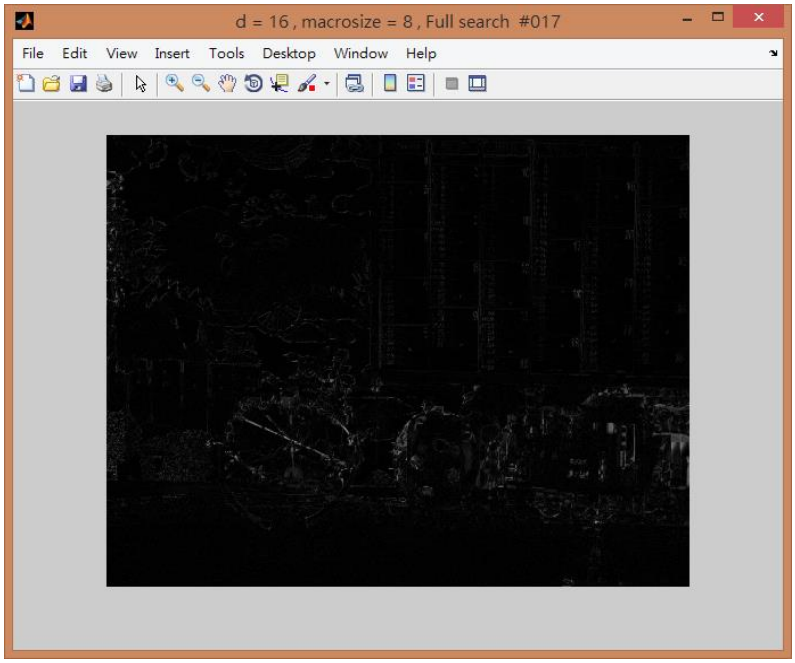

d = 8 , macrosize = 8 , 2-d logarithm #008
File Edit View Insert Tools Desktop Window Help

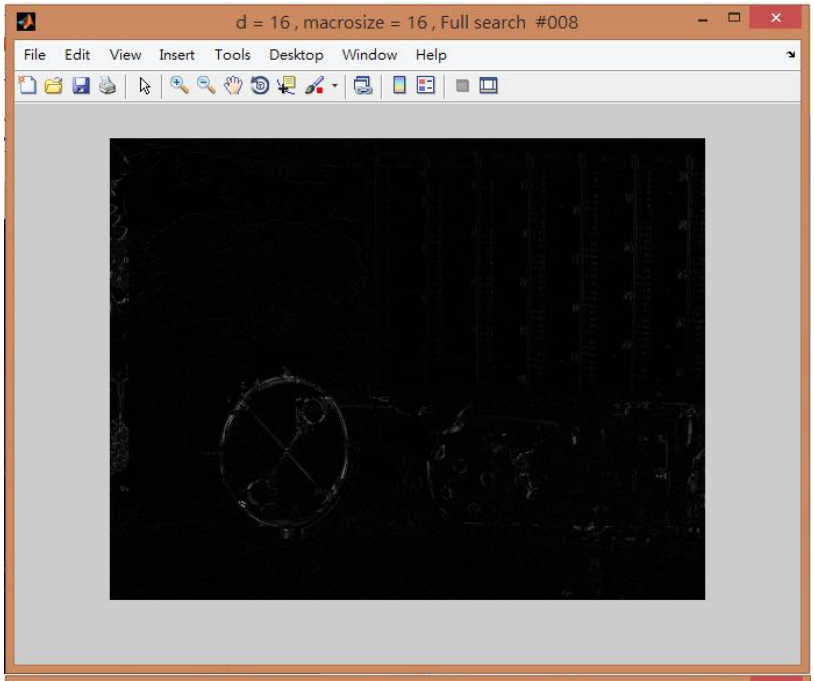

d = 8 , macrosize = 8 , 2-d logarithm #017
File Edit View Insert Tools Desktop Window Help

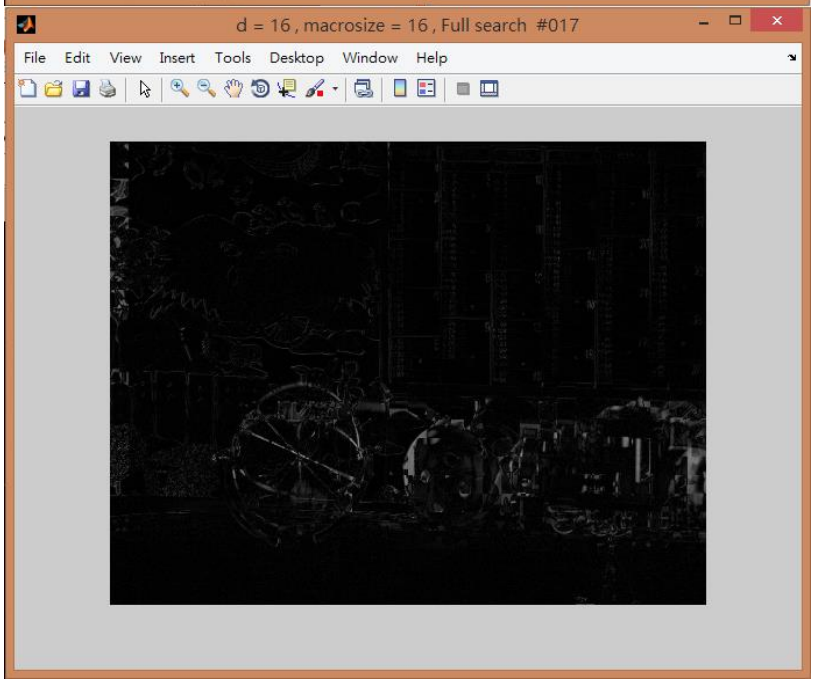

d = 8 , macrosize = 16 , Full search #008
File Edit View Insert Tools Desktop Window Help

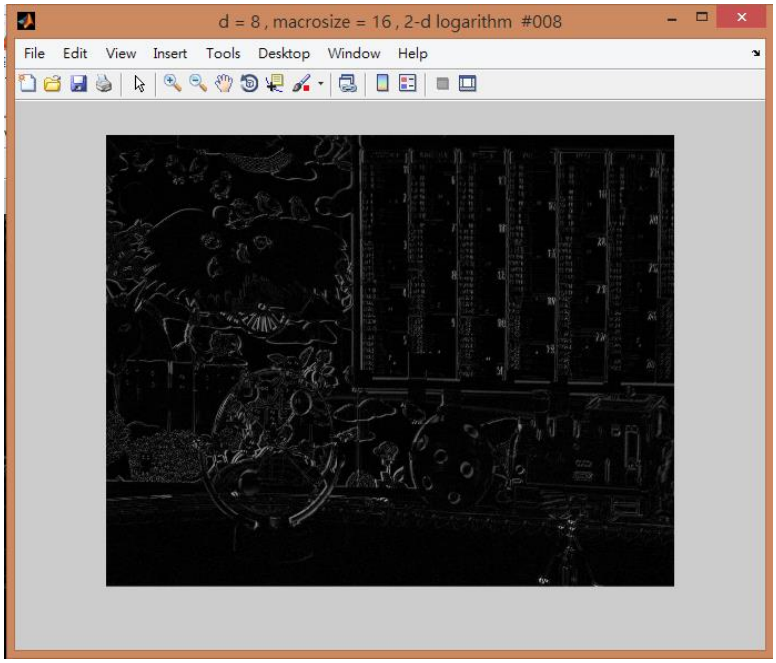

d = 8 , macrosize = 16 , Full search #017
File Edit View Insert Tools Desktop Window Help

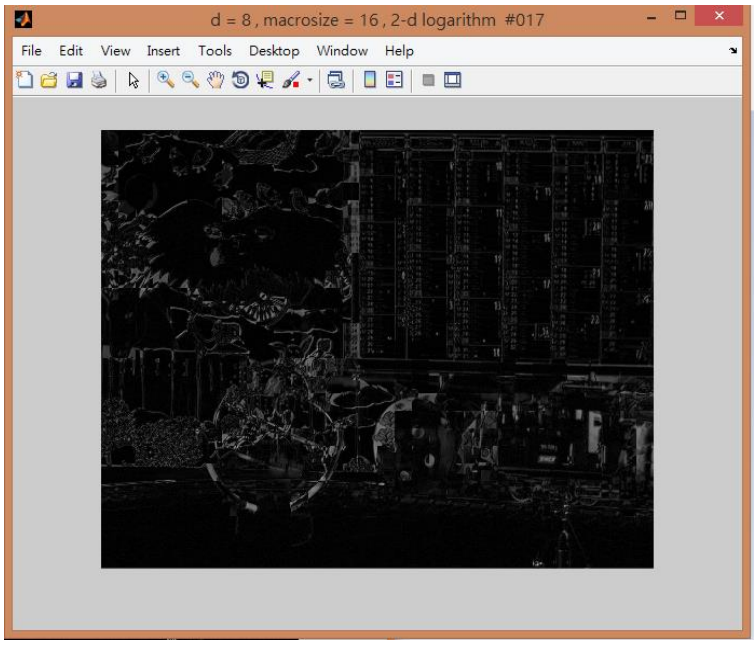

d = 16 , macrosize = 8 , Full search #008
File Edit View Insert Tools Desktop Window Help

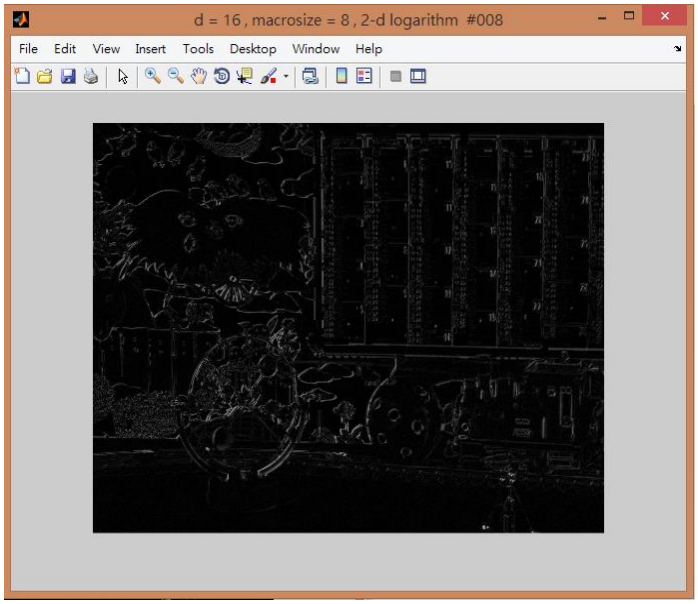

d = 16 , macrosize = 8 , Full search #017
File Edit View Insert Tools Desktop Window Help

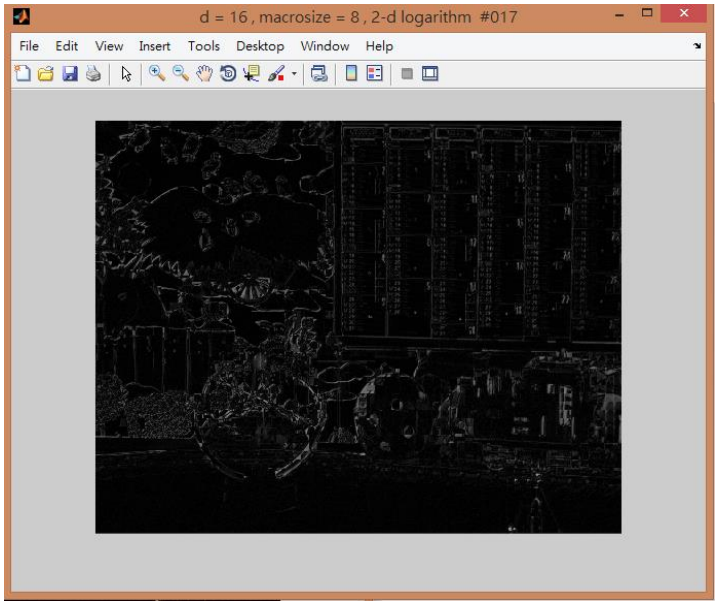

d = 16 , macrosize = 16 , Full search #008
File Edit View Insert Tools Desktop Window Help

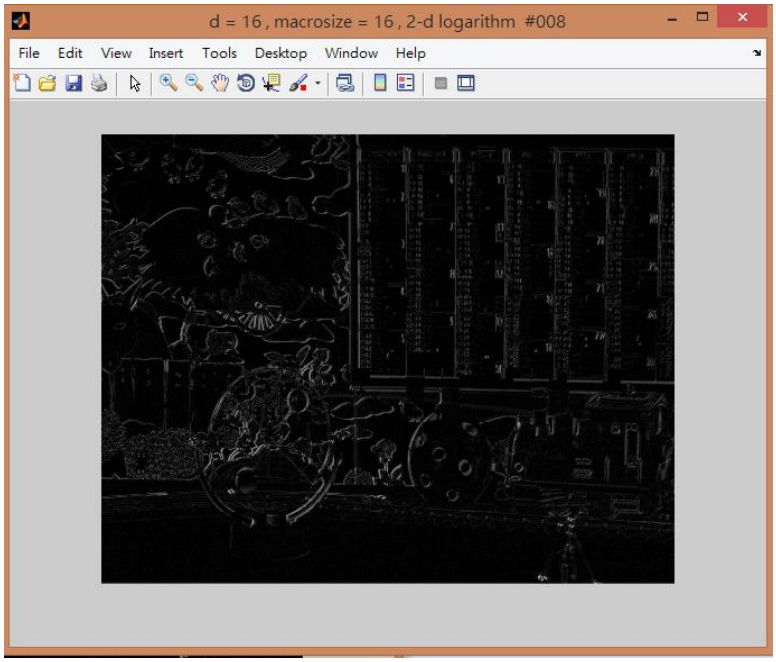

d = 16 , macrosize = 16 , Full search #017
File Edit View Insert Tools Desktop Window Help

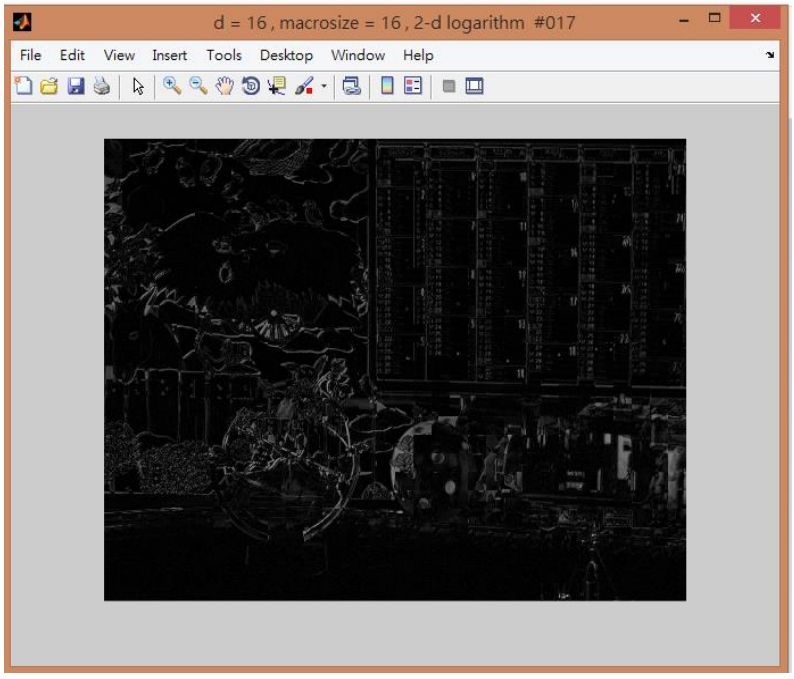

d = 8 , macrosize = 16 , 2-d logarithm #008
File Edit View Insert Tools Desktop Window Help


d = 8 , macrosize = 16 , 2-d logarithm #017
File Edit View Insert Tools Desktop Window Help


d = 16 , macrosize = 8 , 2-d logarithm #008
File Edit View Insert Tools Desktop Window Help


d = 16 , macrosize = 8 , 2-d logarithm #017
File Edit View Insert Tools Desktop Window Help

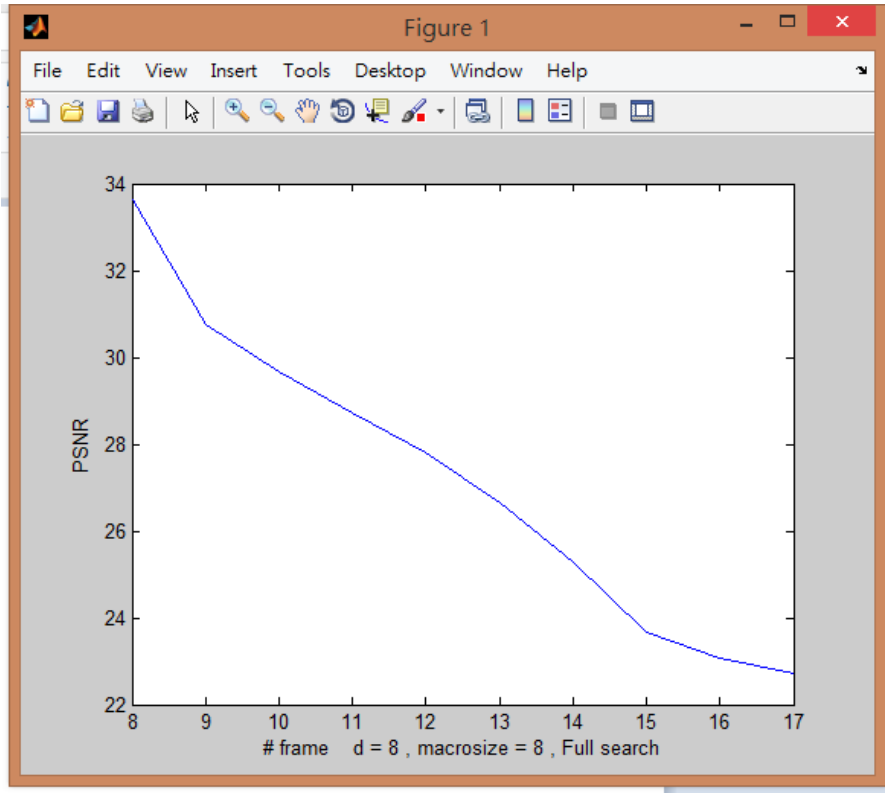




觀察後不難發現不論對哪種參數和演算法而言，#017 的 residual image 整體都比#008 的來得明顯，可以直接推論為應該是因為跟 Reference frame 相比，#017 幾乎所有物體都已經有了大量的偏移，不好從 reference 去拼湊出新的 target frame。

(b)小題則叫我們畫出所有的 PSNR curve。結果如下：

(參數及使用演算法已經標示在 xlabel 的地方)



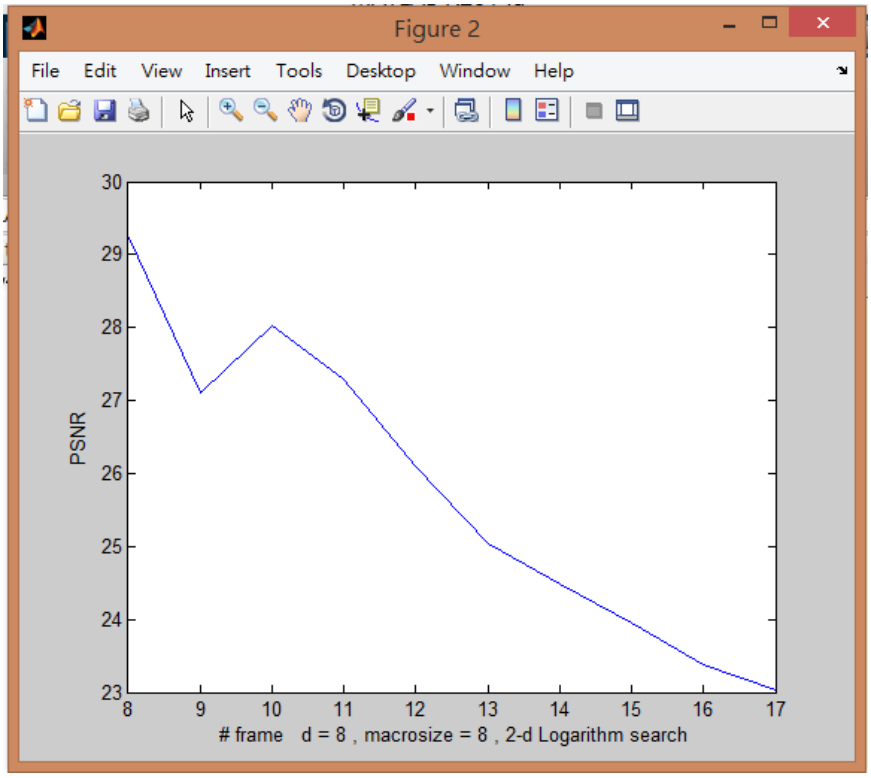


Figure 2
File Edit View Insert Tools Desktop Window Help
PSNR
30
29
28
27
26
25
24
23
8 9 10 11 12 13 14 15 16 17
frame d = 8 , macrosize = 8 , 2-d Logarithm search

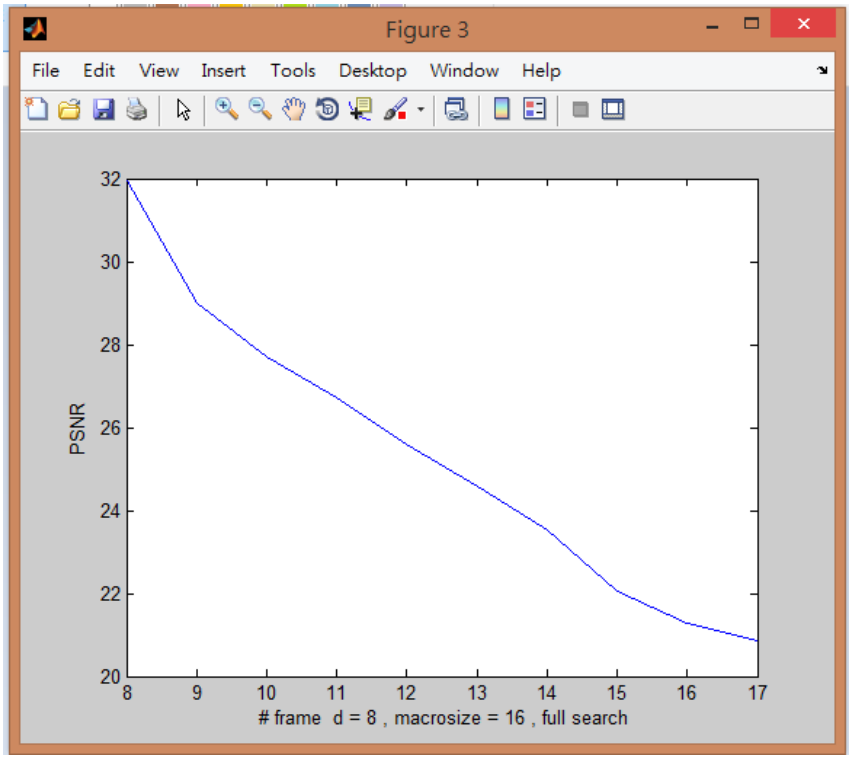


Figure 3
File Edit View Insert Tools Desktop Window Help
PSNR
32
30
28
26
24
22
20
8 9 10 11 12 13 14 15 16 17
frame d = 8 , macrosize = 16 , full search

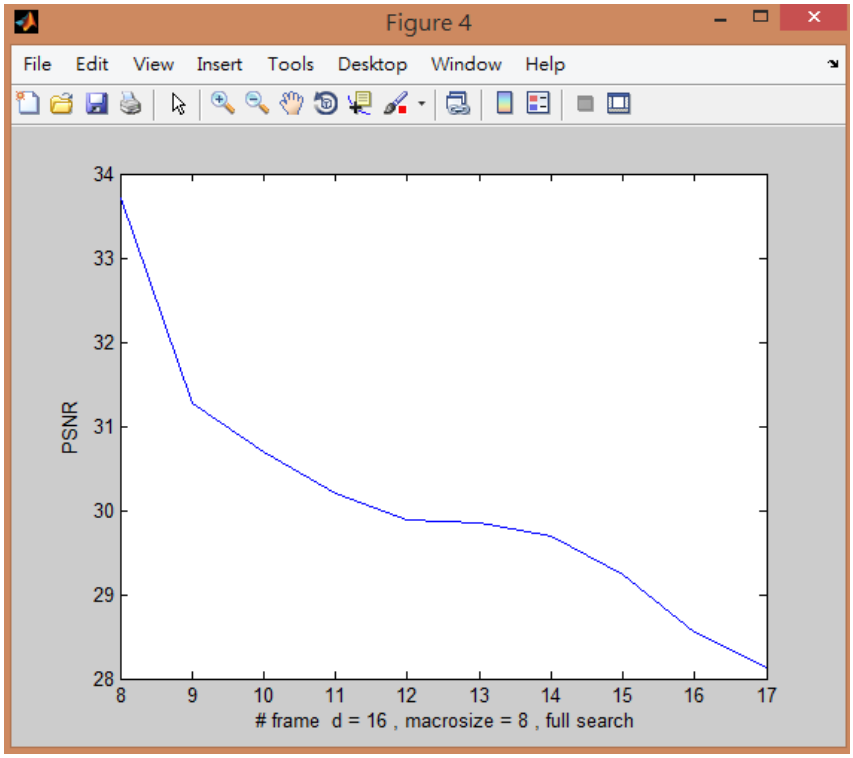


Figure 4
File Edit View Insert Tools Desktop Window Help
PSNR
34
33
32
31
30
29
28
8 9 10 11 12 13 14 15 16 17
frame d = 16 , macrosize = 8 , full search

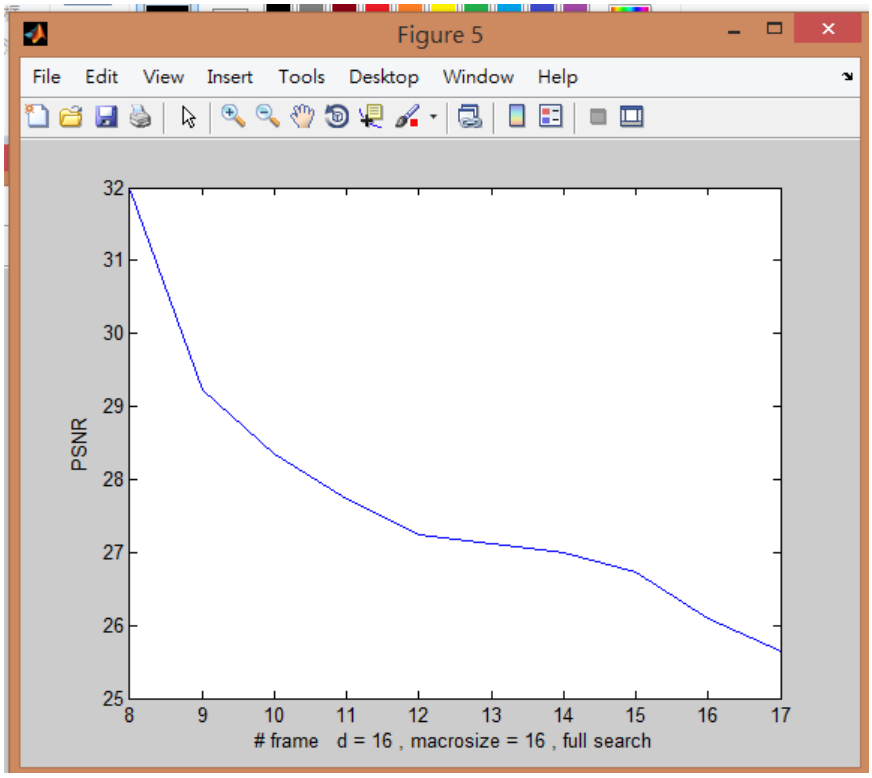


Figure 5
File Edit View Insert Tools Desktop Window Help
PSNR
32
31
30
29
28
27
26
25
8 9 10 11 12 13 14 15 16 17
frame d = 16 , macrosize = 16 , full search

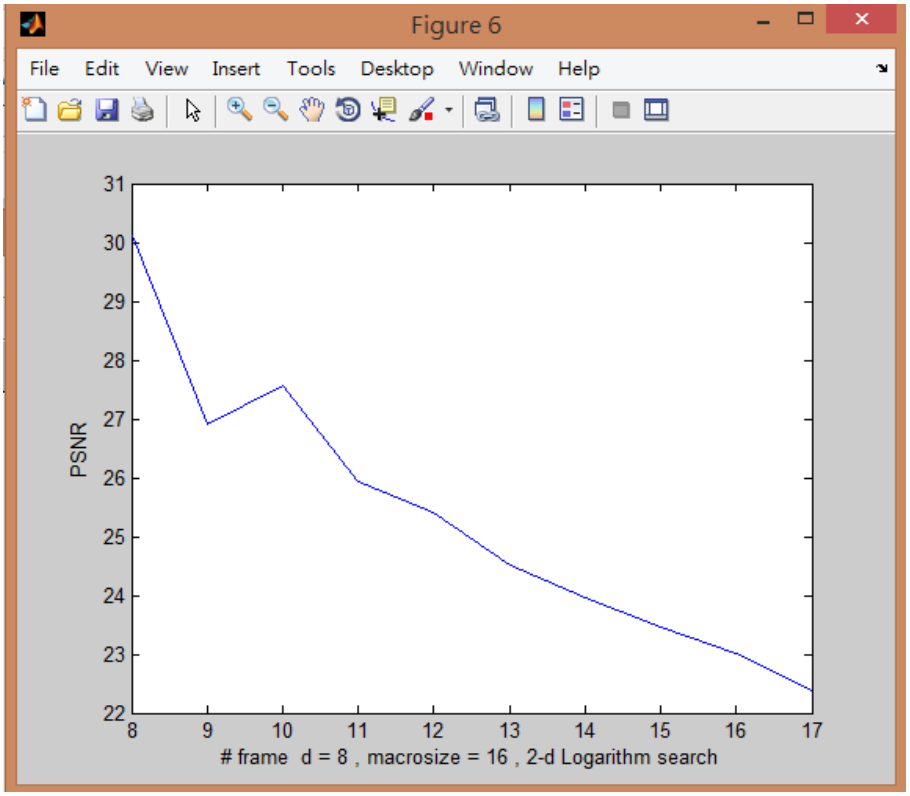


Figure 6
File Edit View Insert Tools Desktop Window Help
PSNR
31
30
29
28
27
26
25
24
23
22
8 9 10 11 12 13 14 15 16 17
frame d = 8 , macrosize = 16 , 2-d Logarithm search

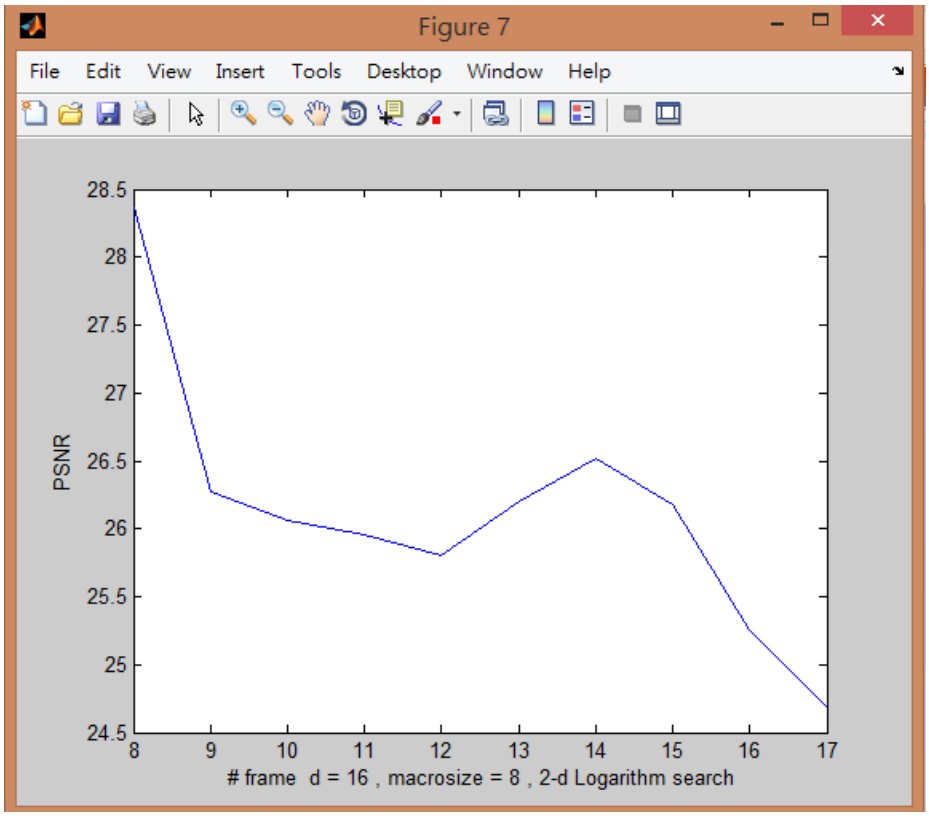
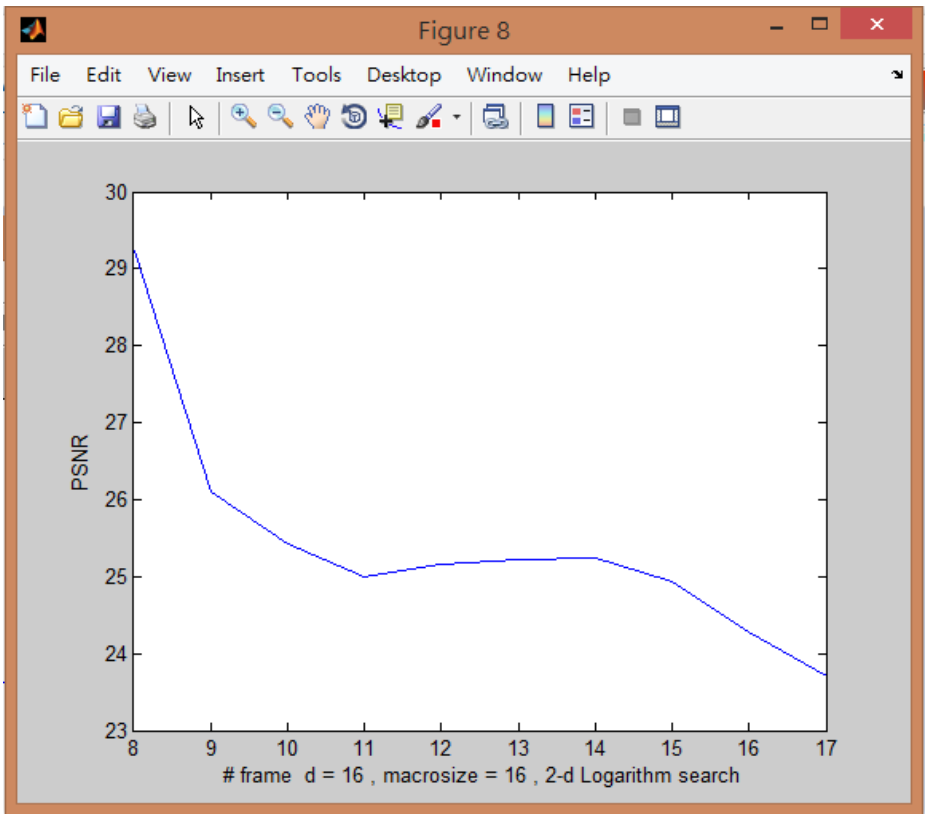


Figure 7
File Edit View Insert Tools Desktop Window Help
PSNR
28.5
28
27.5
27
26.5
26
25.5
25
24.5
8 9 10 11 12 13 14 15 16 17
frame d = 16 , macrosize = 8 , 2-d Logarithm search



從以上結果觀察，可以很快地發現一個所有 figure 都共有的一項特徵，就是 PSNR 值隨著 frame 增加整體而言呈現降低的趨勢。這點可以直接跟 residual image 的比較一樣推斷為後面的 frame 場景內物體跟 reference frame 的變化越來越大，越來越難從 reference frame 去拼湊出需要的 target frame。另外就是可以發現，整體而言 Full Search Algorithm 的 PSNR 比 2-D Logarithm Search Algorithm 的 PSNR 還要來的高，畢竟 2-D Logarithm Search Algorithm 為了速度上的考量捨棄了一些精確度，整體 PSNR 偏低是可以理解的。另外如果全部 Figure 都看後面幾個 frames 的話都可以發現，對兩種演算法而言 PSNR 值都呈現 d=8,size=16 > d=8,size=8 > d=16,size=16 > d=16,size=8

的情況。我認為合理的解釋方式是：Macro block size 可以想成解析度，越大塊解析度差(所以還原度會較差)；d=search range 是搜尋的範圍，搜尋的範圍越大自然越容易找到 best match 的 macro block，還原度自然也會越佳。儘管對於 2-D Logarithm search 的前幾個 frame 來說結果有一點誤差，但我個人認為從畫面有較多變化的後面幾個 frame 來衡量 Motion Estimate 的還原度還是比較有指標性的。

(C)小題則叫我們比較兩者在執行時間上的差異。我把四種參數跟兩種演算法總共八種的執行時間都記錄了下來如下：

(我是在每次執行時都把全部 10 個 frames 的 motion vector 找出來)

d = 8, macro size = 8, full search → 41.705708
d = 8, macro size = 16, full search → 13.379291
d = 16, macro size = 8, full search → 119.043467
d = 16, macro size = 16, full search → 57.129899
d = 8, macro size = 8, 2-d logarithm search → 4.122734
d = 8, macro size = 16, 2-d logarithm search → 3.233076
d = 16, macro size = 8, 2-d logarithm search → 7.116270
d = 16, macro size = 16, 2-d logarithm search → 3.456050

以上全部單位都是秒。不過因為每台電腦性能不一我的結果也會跟理論值有誤差。

整體而言可以很顯而易見的發現 2-D Logarithm search 的速度實在是比 Full search 快太多啦。Full search 理論上要執行(2d+1)^2 次的搜尋和 N^2(N=macroblock 邊長)次的 SAD 計算，假設圖片邊長 a*b 時間複雜度應該是 O((2d+1)^2)*O(N^2)*O(a*b/N^2)~= O((2d+1)^2)*

O(a*b)。而 2-d logarithm 每一輪搜尋完搜尋範圍都會成指數函數縮小(每次縮小二分之一)，也就是搜尋所需的步驟相對 full search 只須對數函數個次數的步驟就可以了，O(N^2)*O(log2(d))*O(a*b/N^2)~= O(log2(d))*O(a*b)，節省的時間相當可觀。

而論同一個演算法裡頭，可以發現 macro block 越大，執行時間越短(macro block 邊長變成兩倍，需要計算的 motion vector 數量變為四分之一)。而 d = search range 越大，執行時間也越多(這應該相當直覺，就是要找多一點的地方來比對，所需步驟當然會比較多)。