Multimedia HW3

1.

首先我實作 full search algorithm 來針對 frame439.jpg，依據 frame437.jpg 做 motion estimate，並找出 motion vectors。

這裡我用 function 來實作。

```
function [Predict_frame, totalSAD, MotionVector, residual] = FullSearch(Reference_frame,Target_frame,p,macroblock_size)

[height, width, rgb] = size(Target_frame);
Target_frame = im2double(Target_frame);
Reference_frame = im2double(Reference_frame);
%%宣告predict frame
Predict_frame = zeros(height,width,3);
%%宣告motion vector，因為是overlapping，所以有(height-macroblock_size+1)*(width-macroblock_size+1)那麼多個
%MotionVector = zeros(2,(height-macroblock_size+1)*(width-macroblock_size+1));
%%宣告motion vector，因為是non overlapping，所以有height*width/(macroblock_size^2)那麼多個
MotionVector = zeros(2,(height*width)/(macroblock_size^2));
index = 1;
totalSAD = 0;
```

宣告一個叫做 FullSearch 的 function，傳入的參數有 reference frame、target frame、p(search range)、macroblock size。

回傳的參數是 predict frame 的資料。

一開始先去將 frame 的長寬，以及有幾個 channel 讀出來（rgb 為 3）。接著呼叫 im2double 將 frame 的資料從 uint8 轉成 double，並宣告長寬和 target frame 一樣的陣列，rgb 三個 channel 皆為 0。

宣告紀錄 motion vector 的陣列，大小為(height*width)/(macroblock_size)^2，因為 target frame 的面積為 height*width，去切 n*n 的 macroblock，共可以切 height*width/n^2 個 macroblock。

Index 表示做 motion estimate 到 target frame 第幾個上 macroblock 了，初設為 1，totalSAD 初設為 0，是用來記錄整個過程選取的 block 與實際 block 之間的 SAD 總共為多少。

```
for y = 1:macroblock_size:height-macroblock_size+1
    for x = 1:macroblock_size:width-macroblock_size+1
        %%初始化search range內的每一個BLOCK的SAD值，使其為double最大的數(避免有些沒算到結果minSAD為它)
        %SAD = ones(2*p+1,2*p+1)*realmax;
        %%初始化minSAD的值，使其為double最大的數
        minSAD = realmax;
        %%初始化MotionVector為到最左上角的那一個BLOCK的向量，套用講義第80頁的座標系統
        MotionVector(1, index) = (x-p)-x;
        MotionVector(2, index) = (y-p)-y;
        %用overlapping的方式在search range中找SAD最小的BLOCK
        for i = -p:p
```

接著準備開始實際進行 full search 的演算法，這裡用雙層迴圈，外圈為 y 軸，內圈為 x 軸，將整個圖片切成各個 macroblock 跑一遍，根據 macroblock 的大小，每次迴圈增加的量為 macroblock size，x 和 y 表示該 macroblock 最左上角的 pixel 的座標。並在每次去 search 之前，去初設 minSAD 的數值，使其為

realMAX，這樣做的原因是因為一開始讓這個數為 double 最大，那麼之後要比較的時候幾乎任何數都會比他小。又，初設該 macroblock 對應之 MotionVector 為(-p,-p)。

```
for i = -p:p
    for j = -p:p
        y_insearch = y+i;
        x_insearch = x+j;
        sad_value = 0;
        if y_insearch < 1 || y_insearch > height-macroblock_size+1 || x_insearch < 1 || x_insearch > width-macroblock_size+1
            continue;
        else
            for a = 1:3
                for b = 1:macroblock_size
                    for c = 1:macroblock_size
                        sad_value = sad_value + abs(Reference_frame(y_insearch+b-1,x_insearch+c-1,a) - Target_frame(y+b-1,x+c-1,a));
                    end
                end
            end
        end
        %SAD(i+p+1,j+p+1) = sad_value;
        if sad_value < minSAD
            minSAD = sad_value;
            MotionVector(1,index) = x+j-x;
            MotionVector(2,index) = y+i-y;
        end
    end
```

接著，開始進行 search 的動作。一樣是用雙層迴圈去做，範圍從-p 至 p，將 search range 中的每一個 block 都跑一遍。

在每一次的 search 中，初設 y_insearch 為 y+i，x_insearch 為 x+j，表示目前在搜尋的該 macroblock 在 reference frame 中的座標為何，並去判斷這個座標是否有超出圖片的許可範圍，也就是是否小於 1 或大於(height 或 width)-macroblocksize+1。若超出這個範圍，則不去算，換下一個 macroblock；若沒有超出，則開始去算選到的這個 macroblock 中的 rgb 值和 target frame 要 match 的 macroblock 的 rgb 值的 SAD，加總起來為 sad_value。

計算完 sad_value 之後，便拿這個值去和 minSAD 做比較，如果較小，表示這個 target frame 的 macroblock 在其 search range 中目前為止現在 match 到的結果是最好的，所以去將 minSAD 改為算出來的 sad_value，並讓其對應的 MotionVector 為 search range 中目前這個 macroblock 的座標減掉 target frame 該 macroblock 的座標。如此將 search range 中的每個 macroblock 做完，會得到最佳的 match。

在 search range 搜索完後，接著就要根據得到的最佳 motion vector 去得出 predict frame 在(x,y)位置的 macroblock 的 rgb 值要怎麼填。

用了以下式子：

```
Predict_frame(y:y+macroblock_size-1,x:x+macroblock_size-1,:) =
Reference_frame(y+MotionVector(2,index):y+MotionVector(2,index)+macro
block_size-
1,x+MotionVector(1,index):x+MotionVector(1,index)+macroblock_size-
1,:);
```

在這裡做的就是讓 predict frame 在(x,y)位置的 macroblock 的 rgb 值為 reference

frame 上在(x+motionvector_x,y+motionvector_y)的 macroblock 的 rgb 值。
Assign 完之後，讓 totalSAD 加上目前的 minSAD，並讓 index+1，也就是 MotionVector 的 index，並去算下一個位置的 macroblock 的 motion vector，並讓 predict frame 該位置的 rgb 值為對應 reference frame 的 macroblock 的 rgb 值。
全部的 macroblock 的算完後，我們就得到了整張 predict frame 的資料，並用此去算 residual。

```
residual = sum(abs(Target_frame-Predict_frame),3);
```

最後將資料回傳。接著回到 script 中，將回傳的資料作輸出、算 psnr 或者 plot 出 motion vector 的圖形。
輸出 predict frame 和 residual 圖的部分用 imwrite 去輸出。

```
imwrite(predict_frame1, "full_search_predict_img439_8_8.jpg");
imwrite(residual1, "full_search_residual_predict_img439_8_8.jpg");
```

算 psnr 值的部分用第一次作業的公式，不過這次要先將圖片的資料從 double 轉成 uint8 的表示方式。在算 error 的時候是用 target frame(img439)減掉 predict frame 的值。

```
SE = 0;
predict_frame1 = im2uint8(predict_frame1);
for rgb = 1:3
    for i = 1:height
        for j = 1:width
            SE = SE + (double(img439(i,j,rgb)) - double(predict_frame1(i,j,rgb)))^2;
        end
    end
end
MSE = SE/(height*width*3);
PSNR1 = 10 * log10(255^2/MSE);
```

在畫 motion vector 的部分，我是先用 meshgrid 依據 macroblock 的大小切出網格，然後將 frame437（reference frame）用 imshow 表示出來，這裡把他 show 出來主要是要來表示這張圖上的 macroblock 經過怎麼樣的移動會組成 predict frame 的圖。接著呼叫 quiver 將 motion vector 畫出來。

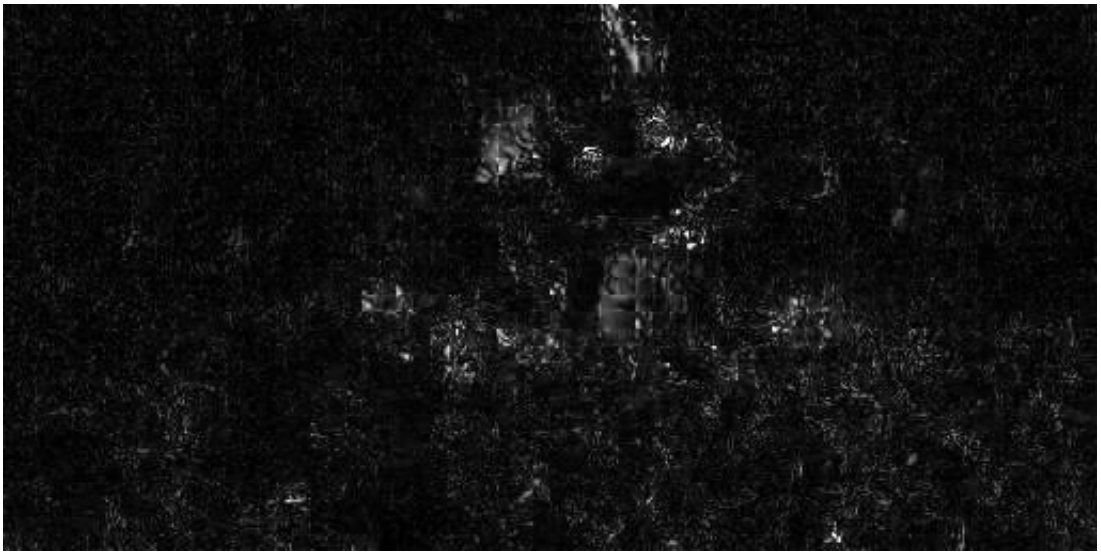
```
[X2,Y2] = meshgrid(1:16:size(img437,2),1:16:size(img437,1));
motion_x2 = zeros(size(X2,1),size(X2,2));
for i=1:size(X2,1)
    for j=1:size(X2,2)
        motion_x2(i,j) = MotionVector2(1,(i-1)*size(X2,2)+j);
    end
end
motion_y2 = zeros(size(Y2,1),size(Y2,2));
for i=1:size(Y2,1)
    for j=1:size(Y2,2)
        motion_y2(i,j) = MotionVector2(2,(i-1)*size(Y2,2)+j);
    end
end
imshow(img437);
hold on;
quiver(X2(:),Y2(:),motion_x2(:),motion_y2(:));
hold off;
```

以下是做出來的結果表示：

Search range size 8, macroblock size 8

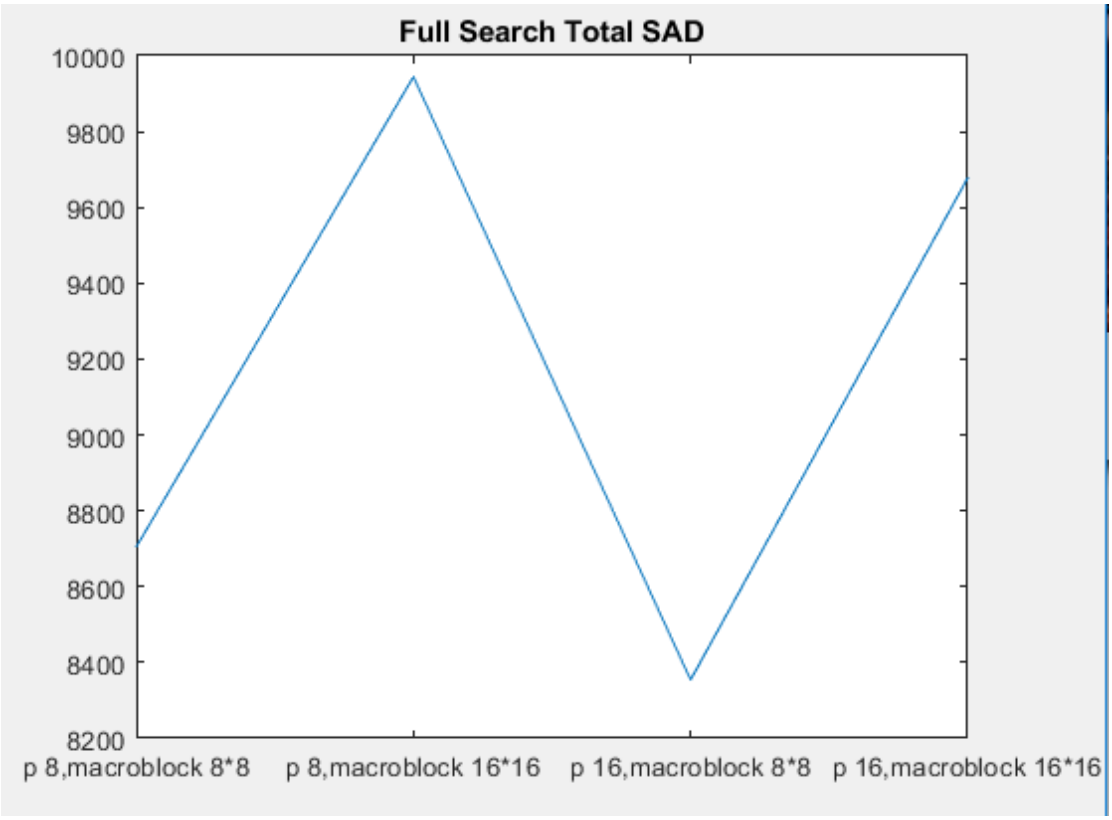
Search range size 8, macroblock size 16

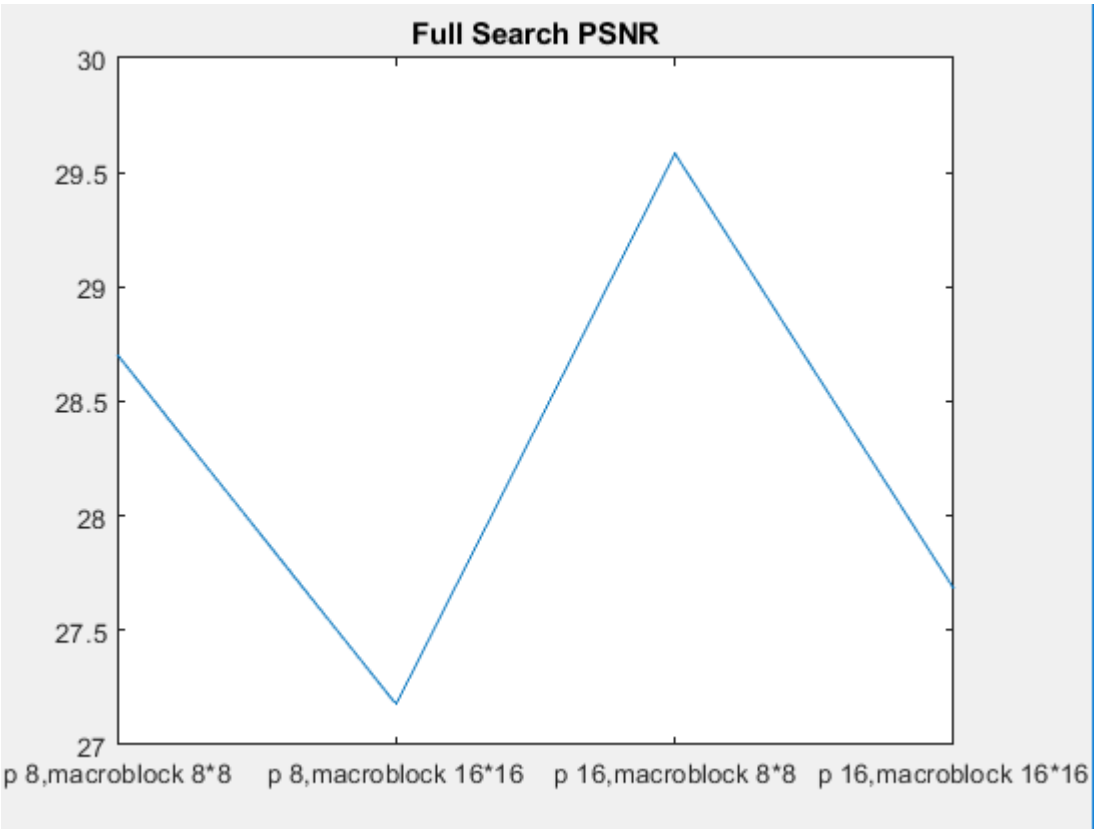
Search range size 16, macroblock size 8



Search range size 16, macroblock size 16

最後根據每一個結果的 totalSAD 和 PSNR 值，plot 出結果。





可以看到在 search range 相同的情況下，macroblock size 越小，則預測的越準，誤差越小，這是因為我們將整個圖片切成較多較細的 macroblock 去做 predict，

所以做出來的效果會比較好；在 macroblock size 相同的情況下，search range 越大，則預測的越準，誤差越小，這是因為我們在 search 的時候比對 reference frame 的 macroblock 數量較多，所以較容易找到 SAD 更小的 macroblock 來填 predict frame 上 macroblock 的值。
綜合實驗結果，得到 p = 16，macroblock size = 8 的時候 psnr 最大，total SAD 最小，接著是 p = 8， macroblock size = 8，然後 p = 16，macroblock size = 16，最後最差的結果是 p = 8，macroblock size = 16。此實驗結果亦符合我根據理論的推測。

接著實作 3 step Search 的演算法。大致上其變數都和 Full Search 雷同，主要差異是在 search 的過程。
我們首先根據要做 predict 的 macroblock，將它當作中心點，然後在固定距離內再取其他八個點，看哪一個點對到的 macroblock 和要 predict 的 macroblock 之間的 SAD 值最小。

```
if p == 8
    w = p/2;
    for i = -w:w:w
        for j = -w:w:w
            y_insearch = y+i;
            x_insearch = x+j;
            sad_value = 0;
            if y_insearch < 1 || y_insearch > height-macroblock_size+1 || y_insearch < y-p || y_insearch > y+p || x_insearch < 1 || x_inse
                continue;
            else
                for a = 1:3
                    for b = 1:macroblock_size
                        for c = 1:macroblock_size
                            sad_value = sad_value + abs(Reference_frame(y_insearch+b-1,x_insearch+c-1,a) - Target_frame(y+b-1,x+c-1,a));
                        end
                    end
                end
            end
```

同樣的如果 search range 中的該 macroblock 超出範圍則忽略它，換算下一個。
接著將九個點所代表的 macroblock 和要 predict 的 macroblock 之間的 SAD 算完之後，會得到一組 motion vector，表示要 predict 的 macroblock 移動該 vector 的向量後所到的 reference frame 上的 macroblock 和它之差距最小。

```
                if sad_value < minSAD
                    minSAD = sad_value;
                    MotionVector(1,index) = x_insearch-x;
                    MotionVector(2,index) = y_insearch-y;
                end
            end
        end
        minSAD = realmax;
        center_x = x + MotionVector(1,index);
        center_y = y + MotionVector(2,index);
        w = p/4;
```

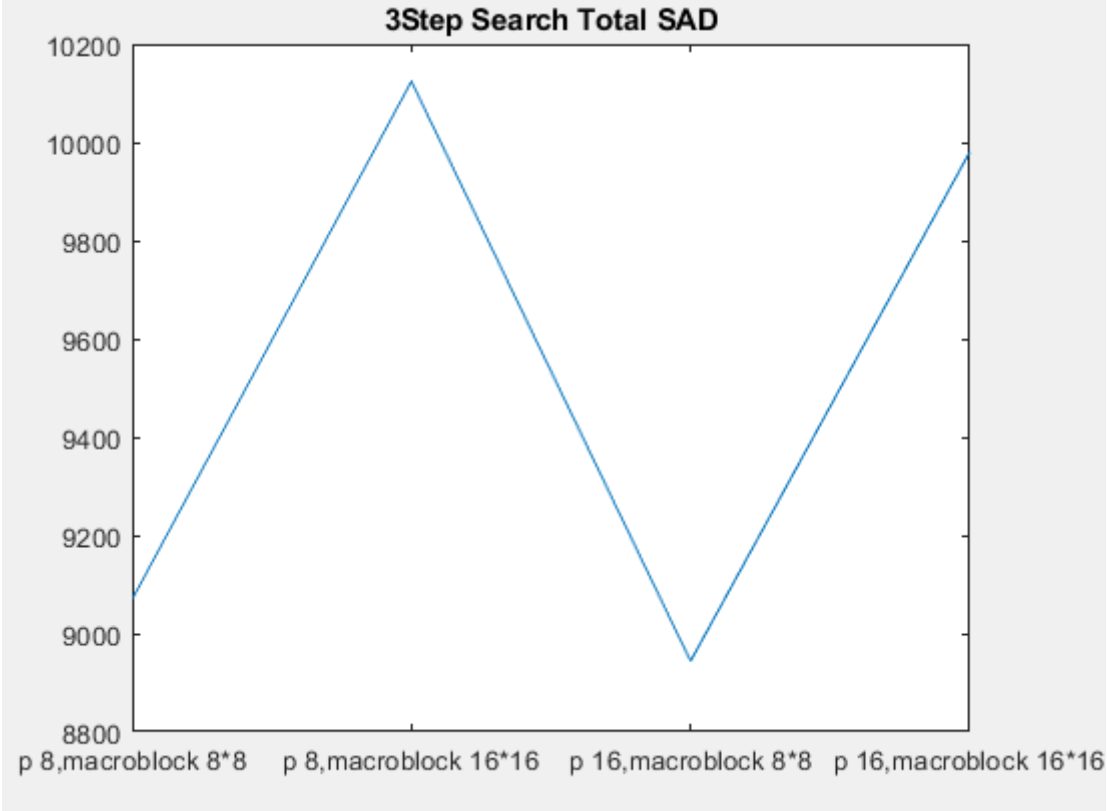
```
for i = -w:w:w
    for j = -w:w:w
        y_insearch = center_y+i;
        x_insearch = center_x+j;
        sad_value = 0;
        if y_insearch < 1 || y_insearch > height-macroblock_size+1 || y_insearch < y-p || y_insearch > y+p || x_insearch < 1 || x_ins
            continue;
        else
            for a = 1:3
                for b = 1:macroblock_size
                    for c = 1:macroblock_size
                        sad_value = sad_value + abs(Reference_frame(y_insearch+b-1,x_insearch+c-1,a) - Target_frame(y+b-1,x+c-1,a));
                    end
                end
            end
        end
        if sad_value < minSAD
            minSAD = sad_value;
            MotionVector(1,index) = x_insearch-x;
            MotionVector(2,index) = y_insearch-y;
        end
    end
```

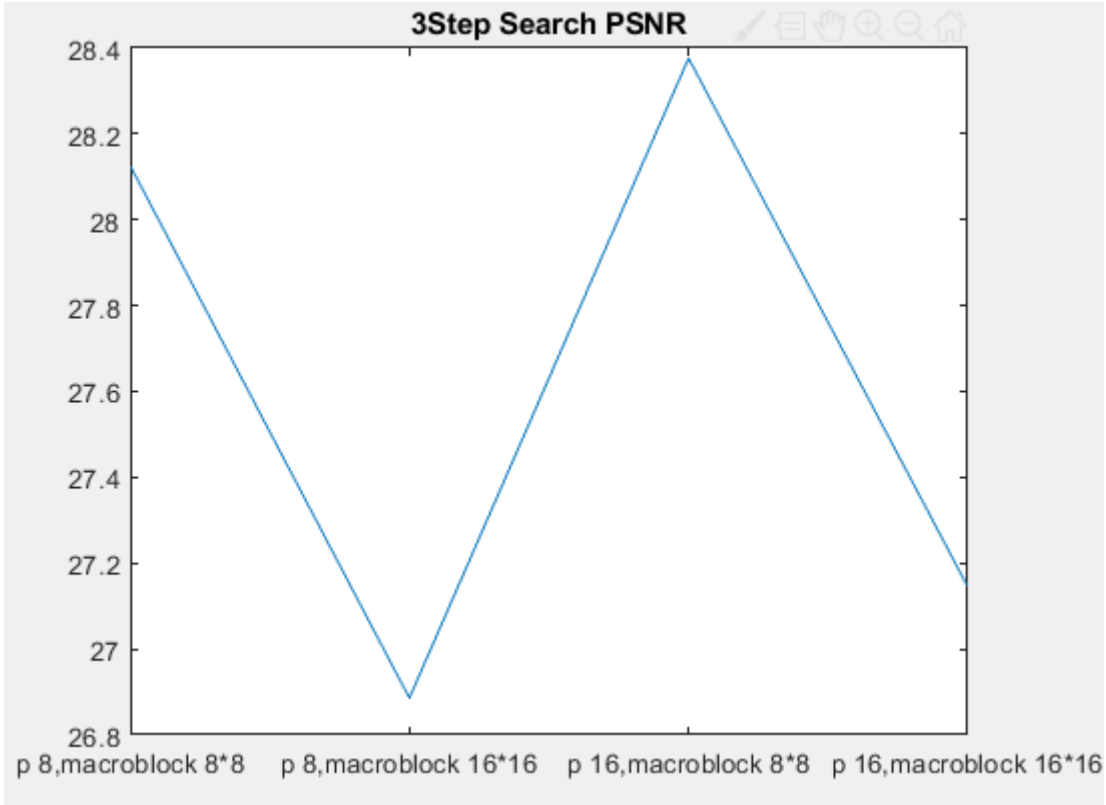
接著以該點為中心，用剛剛距離除以 2 的距離再去取 8 個點，並重複再算一次，又會得到一組 motion vector，表示要 predict 的 macroblock 移動該 vector 的向量後所到的 reference frame 上的 macroblock 和它之差距最小。
接著我們再接著依照這個步驟去算，直到最後中心點到其他八個點的距離為 1 的時候。
以下是做出來的結果表示：
3 Step Search search range size 8, macroblock size 8

3 Step Search search range size 8, macroblock size 16

3 Step Search search range size 16, macroblock size 8

3 Step Search search range size 16, macroblock size 16

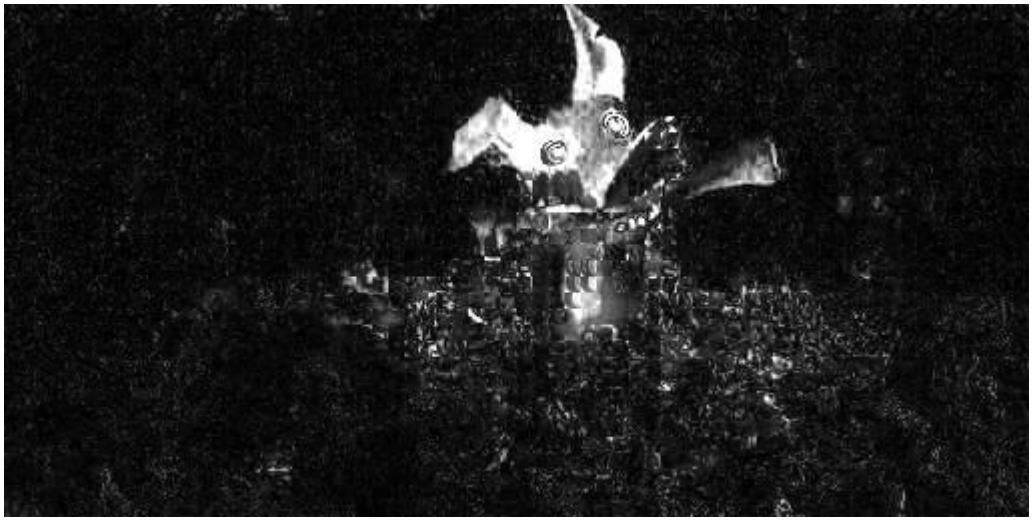




可以看到在 search range 相同的情況下，macroblock size 越小，則預測的越準，誤差越小，這是因為我們將整個圖片切成較多較細的 macroblock 去做 predict，所以做出來的效果會比較好；在 macroblock size 相同的情況下，search range 越大，則預測的越準，誤差越小，這是因為我們在 search 的時候比對 reference

frame 的 macroblock 數量較多，所以較容易找到 SAD 更小的 macroblock 來填 predict frame 上 macroblock 的值。
綜合實驗結果，得到 p = 16，macroblock size = 8 的時候 psnr 最大，total SAD 最小，接著是 p = 8， macroblock size = 8，然後 p = 16，macroblock size = 16，最後最差的結果是 p = 8，macroblock size = 16。此實驗結果亦符合我根據理論的推測。
雖然兩個演算法的在各自的算法內都和預期的一樣，不過在這裡可以再拿兩個演算法的結果出來比，可以發現 full search 的結果比 3step search 的結果作出來要好一點。這也符合我的預期，因為 full search 是在 search range 之中一個一個的找最適合的 macroblock，但 3step search 卻忽略掉了一些部份，所以做出來的效果會是 full search 較好，然而在運算量上，3 step search 的運算量會比 3 step search 少。

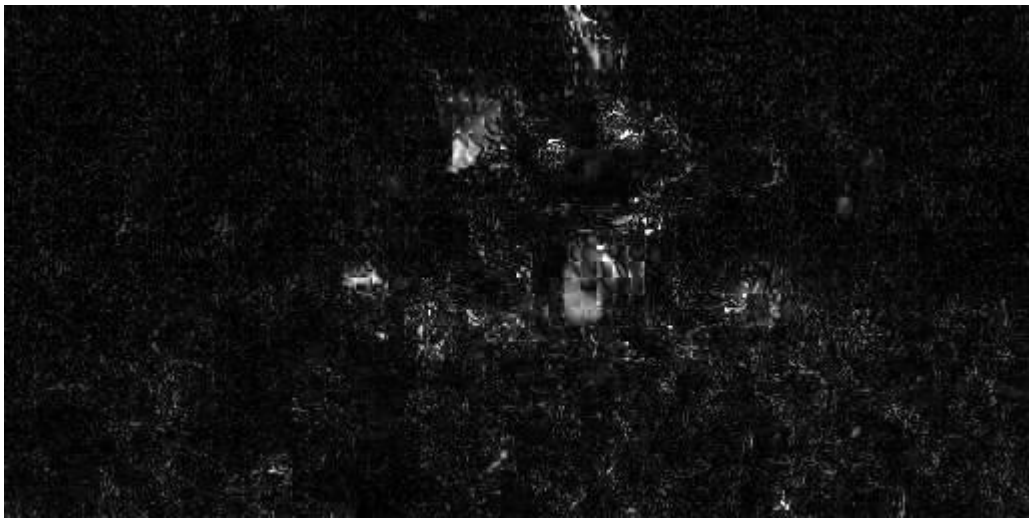
2
第二題拿 frame432 去 predict frame439 做出來的結果如下：



PSNR 值為：

```
B小題,用432frame來當reference的psnr為:
   23.0777
```

而若是用 frame437 來做 predict，其結果為：



PSNR 值為：

```
Full Search with search range size 8 and macroblock size 8:
   28.7054
```

這兩個作法相較之下，是用 frame437 去做預測的效果較好，可以看到做出來的圖狗是完整的，而 frame432 做出來的狗不完整；且看 residual 圖可以發現用 frame432 做，residual 很大，圖中有很大一部份是白的；在看 PSNR，用 frame437 做的 PSNR 值為 28.7054，用 frame432 做的 PSNR 值為 23.0777。
這符合我的預測，因為我要預測的圖是 frame439，而 frame437 離 frame439 較近，故兩張圖的差距變化較小，用來預測會較準確；反之因為 frame432 離 frame439 較遠，故差距變化較大，用演算法來預測做出來的效果並不好。

3.

以下是測出來的運算時間：

```
Full Search (range p=8 , block=8x8 )
Time : 1.28125
```

```
Full Search (range p=16 , block=8x8 )
Time : 3.48438
```

```
3 Step Search (range p=8 , block=8x8 )
Time : 0.640625
```

```
3 Step Search (range p=16 , block=8x8 )
Time : 0.796875
```

可以看到在相同的演算法下，當 search range 越大則運算所需時間越高。
同時也可以看到若 search range 相同的情況下，3 step search 的運算時間比 full search 的運算時間少很多。
接著來看一下和理論時間複雜度的比較
Full search 的理論時間複雜度為：
O((Height_of_frame*Width_of_frame/(block_size^2))*(2p+1)^2)，由於此處 block size 相同，故只須看 search range 的比較。可以看到 p=8 的時候，運算時間約為 1.3 秒，而 p=16 時運算時間約為 3.5 秒，之間的比值為 2.69。由於 O()中是 (2p+1)^2，理論上比值要約為 4，但這裡為 2.69 的原因我認為是因為當 macroblock 超出 search range 就跳過不算，所以在 p = 16 時有一些 macroblock 的運算跳過，這些跳過的數量比 p = 8 的時候多，所以此處比值才會比理論值小一點。
3 Step search 的理論時間複雜度為：
O((Height_of_frame*Width_of_frame/(block_size^2))*9logp)，，由於此處 block size 相同，故只須看 search range 的比較。可以看到 p=8 的時候，運算時間約為 0.64 秒，而 p=16 時運算時間約為 0.8 秒，之間的比值為 1.25。由於 O()中是 9logp，比值約為 1.33，在這裡可以看到實際測出來的時間和理論上的時間複雜度之間的差距差不多。