國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：鍾 國 亮　博士

吳 榮 根　博士

應用於 H.264 中，區塊匹配的快速演算法

A Fast Block-Matching Algorithm

for H.264/AVC

研究生：　林　致　弘　撰

中華民國　九十九　年　一 月

中文摘要

H.264/AVC 是目前最新的影像壓縮標準，其中的動態評估(Motion Estimation)會占據 70%左右的編碼時間，因此這部分是一個很重要的改良課題。這篇論文將提出一個新的演算法，它將結合 DAS(Directional Asymmetric Search)以及 SPS(Search Pattern Switching)兩種方式的特性，並改良搜尋 Pattern 的部分，使新的演算法可以比 DAS 以及 SPS 還快速，並保持著不錯的影像品質。

Abstract

H.264/AVC is the latest video coding standard at this moment. Motion estimation, which is a part of this standard, occupies about 70% of the total encoding time. Therefore, it is an important section that needs to be improved. In this thesis, we propose a new fast block-matching algorithm, which combines the Directional Asymmetric Search algorithm (DAS) and the Search Pattern Switch algorithm (SPS) and improves the motion estimation section with adding search patterns in Directional Asymmetric Search. Our algorithm can cut the total encoding time compare with DAS and SPS, but still keep the similar PSNR and bitrates.

目 錄

附圖目錄

附表目錄

第一章 緒論

本章主要是說明本論文的研究背景、動機以及目的。最後會概述各章節的主要內容架構。

1.1 研究背景

H.264 是目前最新的影像壓縮標準[1][2]，可以提供高效能的影像壓縮，它包含了許多新的技術，例如 Variable Block Size，在 H.264/AVC 中，有七種不同的 Block Size 可以使用；Quarter-pixel Accurate，可以提供到四分之一個 Pixel 的精準度；Multiple Reference Frame Technique，可以參考一張或多張的 Frames。

目前在 Encode 端的部分，最花費時間的就是 Motion Estimation 的部分了，這部分會占掉整個 Encode 過程 70%左右的時間，由於這部分實在太耗時間，於是有很多人提出了加速 Motion Estimation 的演算法 (Fast Block-Matching Algorithm)，例如:Diamond Search [3] 是先利用 Large Diamond Search Pattern (LDSP，共有九個點，圖 1.1 上黑色之點)，在 Search Window 內的中心開始搜尋，若 Minimum Matching Error 發生的位置不為 LDSP 的中心點，則以 LDSP 上發生

Minimum Matching Error 的那點作為下一次 LDSP 的中心點，直到 LDSP 的中心點為 Minimum Matching Error，才轉換成 Small Diamond Search Pattern (SDSP，共五個點，圖 1.1 中白色四點所圍成之菱形)，SDSP 只做一次，用以選出 best matching block。

Three-Step Search [4] 是 N-Step Search 的一種，N-Step Search 每次搜尋的距離為 2^{N-1}，逐漸遞減。如圖 1.2 是 Three-Step Search 的一個範例，一開始先搜尋以原點為中心，包含原點以及距離為 4 的九個點 (圖 1.2 上黑色的點)，求得最小點後，再以最小點為中心，找周圍距離為 2 的八個點 (圖 1.2 上灰色的點)，再以所求的八個點以及中心點的最小點為中心，再找周圍的八個點，最後的最小點即為所求。

BBGDS (Block-Based Gradient Descent Search) [5] 則是一種遞迴式的搜尋，1)一開始先以原點為中心點，搜尋周圍的八個點，若中心點為最小點，則搜尋結束，若中心點不為最小點，2)則以最小點為新的搜尋的中心點，在執行步驟一的工作，直到中心點為最小點。過程可見圖 1.3。

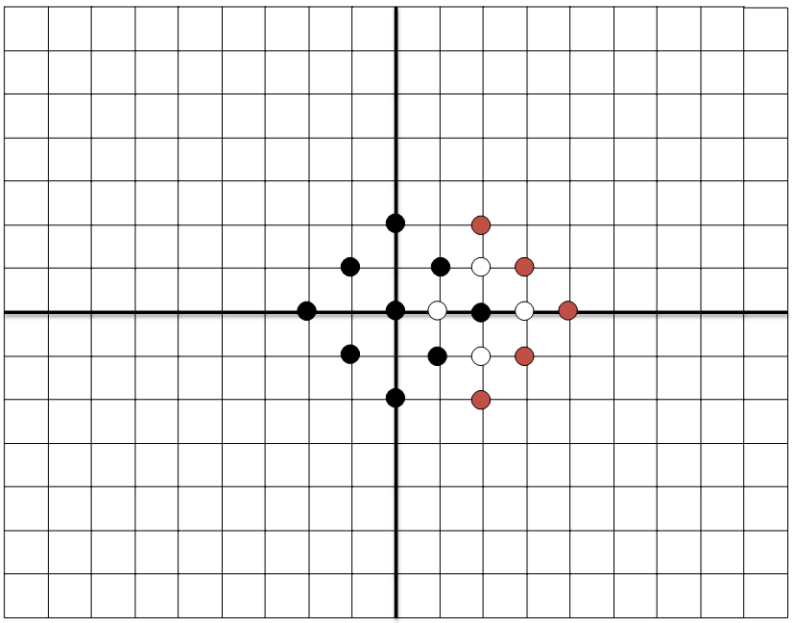

圖 1.1 Diamond Search(DS)

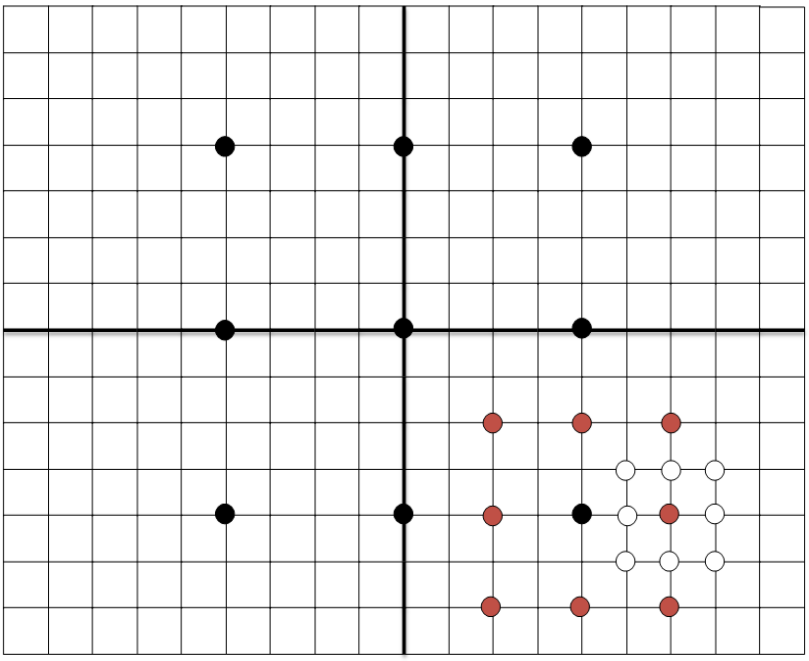

圖 1.2 Three-Step Search(3SS)

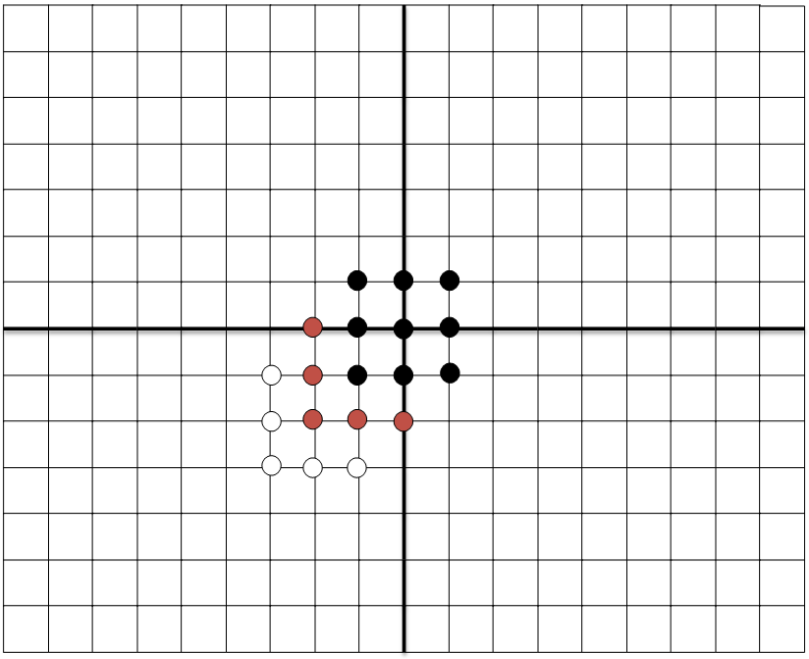

圖 1.3 Block-Based Gradient Descent Search(BBGDS)

1.2 研究動機與目的

由上一節的討論中，我們知道 Encode 端大部分的時間都會消耗在 Motion Estimation 上面。所以有許多的研究都在研究如何改善 Motion Estimation，主要需要改善的部分是在整體編碼的時間、PSNR (Peak Signal to Noise Ratio)以及 Bitrates 這三大部分。

PSNR 是 Peak Signal to Noise Ratio 的縮寫。它的定義如下:

$$PSNR_{dB} = 10\log_{10}\frac{(2^n-1)^2}{MSE} \tag{1}$$

MSE 是 Mean Square Error 的縮寫。它的定義如下:

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(C_{ij}-R_{ij})^2 \tag{2}$$

上面公式(2)中，以 Size N × N 的 Block 做說明，C_{ij} 及 R_{ij} 分別代表 Current Frame 以及 Reference Frame 上的 Pixel；i、j 為其在 Block 中的相對位置。Bitrates 則是以每秒壓縮的 Bits 數為單位。

這篇論文的實驗會利用一種稱為 DAS (Directional Asymmetric Search) [6] 的 BMA (Block-Matching Algorithm)，它的特性是可以快速得完成 Motion Estimation，而且在 Bitrates 跟 PSNR 上也有不錯的表現。我們想利用它快速完成 Motion Estimation 的特性，在上面加以改進，配合 SPS (Search Pattern Switching) [7] 可以評估 MV (Motion Vector)遠近的方法，便可以在三大項目上有所兼顧。

實驗對照組的 BMA 除了上述的兩個方法，還有 UMHexagonS (Unsymmetrical-cross Multi-Hexagon-grid Search)，UMHexagonS 是目前比較常使用以及常被拿來比較的方法，它不僅有效的加速了 Motion Estimation 的時間，也有相當不錯的 PSNR 以及 Bitrates，是一種很不錯的 BMA，也是目前 H.264 實驗的系統 JM 中的其中一個 BMA，我們實驗的結果就是希望也能夠跟 UMHexagonS 來做比較。

1.3 全文架構

本論文共分為五章，以下為各章的內容概述:

【第一章】緒論

說明本論文的研究背景、研究動機、研究目的和全文的架構。

【第二章】過去的相關研究

介紹 Directional Asymmetric Search 及 Search Pattern Switching 兩種方法。

【第三章】新的演算法

介紹我們所提出的新演算法以及演算法的流程。

【第四章】實驗結果以及數據探討

呈現本論文所提出的成果數據以及數據討論。

【第五章】結論

對本論文做最後的總結。

第二章 過去的相關研究

本章將討論本論文所使用到的基礎背景，將介紹兩種 BMA (Block-Matching Algorithm)，Directional Asymmetric Search 以及 Search Pattern Switching，讓大家能對後面的實驗有些瞭解。

2.1 Directional Asymmetric Search

Center-biased 這個特性，已經廣泛的被使用在許多研究中了 [8] ，這個特性是說，絕大部分的 Motion Vector 都會很靠近 Zero Motion (見表 2.1)，所以好的搜尋策略就是從 Search Window 中心開始搜尋。

另外，在欲搜尋最小點(Local Minimum 或 Global Minimum)的周圍，大部分都會呈現嚴格遞減的狀況，也就是說，越接近最小點時，Error 會越小。於是就利用這個特性，以最大點往最小點的方向 (見圖 2.1)作為 DAS (Directional Asymmetric Search)的方向，之後會在這個方向上選出下次欲搜尋的 Pattern。

表 2.1 Motion Vector 之分佈

	MV <= 5	MV > 5
Foreman	80.86 %	19.14 %
Coastguard	96.96 %	3.04 %
Silent	95.99 %	4.01 %
Football	57.34 %	47.66 %
News	98.36 %	1.64 %
Stefan	76.07 %	23.93 %
Mobile	97.78 %	2.22 %

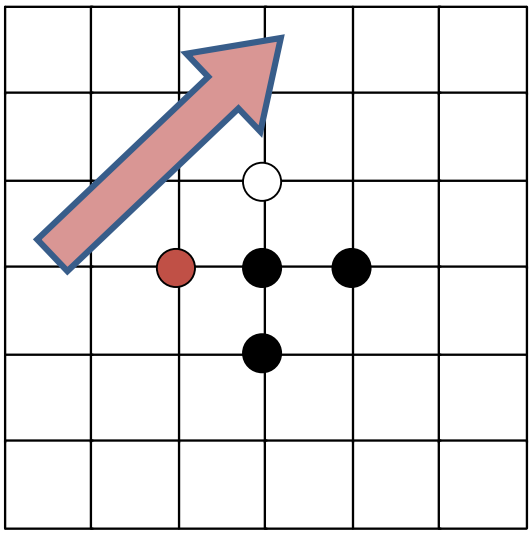

圖 2.1　圖中灰色的點為 Error 最大點，白色點為最小點，箭頭的方向即為 DAS 的方向。

DAS 由 13 種 Pattern 所組成，其中 12 種是看方向性來決定的，剩下的一種則是最基礎的起始 Pattern。一開始先看在 Search Window 中心，由五個點所組成的十字架型的 Pattern，如果 Minimum Block Distortion(MBD)發生在十字型 Pattern 的中央的話，則停止搜尋；如果不是的話，則以 Pattern 中最大 Block Distortion 的點往最小 Block Distortion 之點為方向，加入新的 Pattern (方向性與 Pattern 的關連請見圖 2.2)，再以最小 Block Distortion 的點跟新加入的 Pattern 做比較，如果原先最小 Block Distortion 的點依然為最小的點，則搜尋終止；否則將從最小 Block Distortion 的點及新加入的 Pattern 中，挑出最大及最小 Block Distortion 的點，決定出新的 DAS 之方向，繼續做比較，直到 Minimum Block Distortion 發生在原來最小 Block Distortion 之點上。

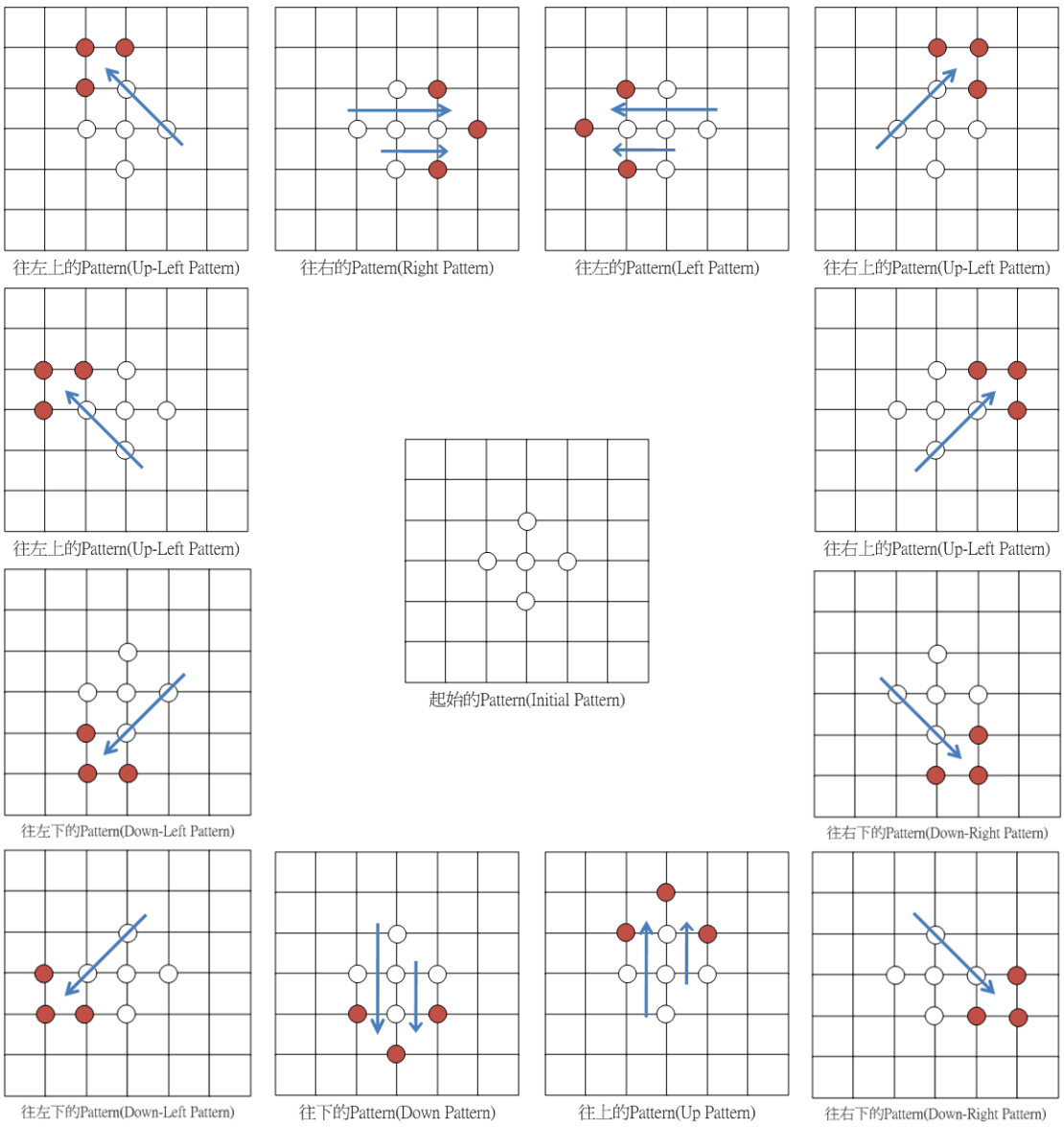

圖 2.2　13 種 DAS 的 Pattern(白色的點代表第一次搜尋的點，紅色點代表第二次搜尋所加的點，箭頭即為 DAS 的方向)

最後我們整理一下整個 DAS 搜尋的步驟：

1) 在 Search Window 的中央，建立起始的 Pattern(如圖 2.2 中的白色十字型 Pattern)。

2) 計算十字型 Pattern 的五個點之 Block Distortion，若 Minimum Block Distortion 發生在 Pattern 中央的話，則 Motion Vector 設為 0(Zero Motion Vector)；否則進行步驟三。

3) 將步驟二 Block Distortion 最小的點設為中心點，並以 Block Distortion 最大點往最小點的方向，做為 DAS 之方向，來選擇應加入的 Pattern。

4) 根據 DAS 的方向，加入對應的點之後(可見圖 2.2 紅色的點)，計算新加入的三個點，在與之前 Block Distortion 最小之點做比較，若前一步之 Block Distortion 最小點仍為 Minimum Block Distortion，則搜尋終止，並將 Minimum Block Distortion 之點設為 Motion Vector；否則便重複步驟三及步驟四。

此外，根據許多研究我們知道，許多 Stationary Blocks 或 Quasi-Stationary Blocks，它們的 Block Distortion 是很低的。所以當找到一個擁有非常小的 Block Distortion 的 Block 時，其實可以不用再去搜尋其他的 Block 了，因為即使找到更小的 Block Distortion，也不會有太顯著的改善，反而會消耗掉許多時間。所以這裡設了一個參數 T_{best} 為門檻值，在每次計算 Block Distortion 的時候，只要小於 T_{best} 這個門檻值，就停止搜尋，並且將小於 T_{best} 的這點設為 Motion Vector，如此便可以加快搜尋的速度。我們在這裡會依 Block Size 的不同而有變化，在 16 × 16 的 Block 下，T_{best} 設為 256。

2.2 Search Pattern Switching Algorithm

根據研究，越接近 Global Minimum Block Distortion 的時候，此時 Block Distortion 會產生嚴格遞減的狀況，也就是說，越接近 Global Minimum 時，Block Distortion 減少的狀況就會越快，圖 2.3 可以解說這種狀況。越接近 Global Minimum，減少的速率就越快(可見圖 2.3 右邊的斜線，斜率較陡)；比較遠離 Global Minimum 的地方，減少的速度就較緩慢(斜率較為平緩)。

由上述可知，利用 Search Window 中央的中心點，計算它與周圍幾個點的差異，作為斜率，可以來推測 Global Minimum 的遠近。第一步先找出 Search Window 的中心點，並計算出它的 Block Distortion，先將這個點的 Block Distortion 命名為 D_A，再來計算它周圍的四個點(上下左右的四個點)，並選出其中 Block Distortion 的最小點，將其 Distortion 值命名為 D_B，於是，可以利用(D_B-D_A)，去計算斜率；之後再以斜率的陡峭或是平緩，來判斷 Global Minimum 離目前計算的點之遠近。

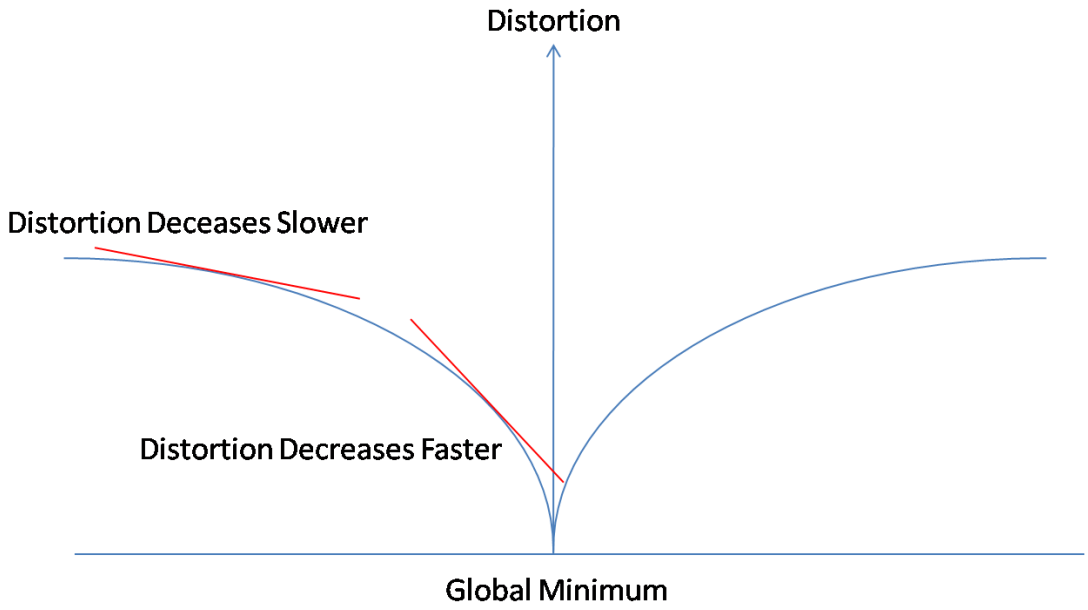

圖 2.3　靠近 Global Minimum 時，Block Distortion 減少的狀況

但(D_B-D_A)的大小會跟著影片的不同而有所變化；所以(D_B-D_A)無法取出一個定值來給予參考，判斷此時斜率是大或者是小。例如:在 Football 這個範例影片中，D_B 的值為 10000、D_A 的值為 9000(D_B-D_A)的值為 1000；而在 Foreman 這段影片中，D_B 的值為 1800、D_A 的值為 800，(D_B-D_A)的值也為 1000。但可以看得出在 Football 中，Block Distortion 減少的斜率是很平緩的，而在 Foreman 中，則是很陡峭的。所以用(D_B-D_A)這個數值來判斷並不是一個好的方法。

由於 D_B 與 D_A 之間的差，是與影片有關的，所以要定義一個新變數 EDR (Error Descent Rate)，它是利用 D_B 與 D_A 之間的比例，來計算斜率。這樣 EDR 便可以不受到不同影片的影響。定義 $EDR = D_B/D_A$，例如在鄰近 Search Window 中心的四個點中，選出其中 Block Distortion 最小之點，它的 Distortion 即為 D_B =800，再計算出 Search Window 中心點的 Block Distortion，D_A =1000，那麼這裡的 EDR = 800/1000 = 0.8。而根據實驗，當 EDR > 0.9 時，便可以推測此時 Global Minimum 是在較遠處，反之，Global Minimum 是在 Search Window 中心點附近。

Search Pattern Switching (SPS) Algorithm 便利用 EDR 來將 MV 分類，判斷 Global Minimum 的遠近後，再去決定用何種方法來搜尋。當 EDR ≧ 1 時，便是鄰近 Search Window 中心點的四個點的 Block Distortion 皆大於中心點，根據之前的研究，這種狀況下 Global Minimum 很可能在中心點鄰近的部分，這時候會使用 Zero Motion Vector，這樣可以使得演算法可以在編碼的品質跟速度上都取得平衡。當 EDR < 1 時，若 EDR 大於門檻值 0.9，便認為 Global Minimum 不在 Search Window 中心點附近，所以這時便會使用範圍較大的 Block Matching Algorithm (BMA)，例如:3SS(Three-Step Search)；若 EDR < 1，但 EDR 小於等於門檻值 0.9，我們便認為 Global Minimum 不會離 Search Window 中心太遠，則便會選用範圍

較小的 BMA，例如: BBGDS(Block-Based Gradient Descent Search)。

SPS Algorithm 也可以使用其他 Pattern 的 BMA，例如範圍較大的演算法，可以用 4SS(Four-Step Search)來取代 3SS；範圍較小的演算法，可以用 DS (Diamond Search)來替換 BBGDS；所以 SPS Algorithm 是可以結合其他不同演算法的，我們就要利用這個特性去結合之前提過的 DAS(Directional Asymmetric Search)。

第三章 新的演算法

在本章中，我們將介紹我們所提出的新演算法，介紹它們的方法以及流程。共有兩個方法將在本章介紹；方法一主要是希望提升編碼影像的 Quality，而方法二則是在維持住 PSNR 以及 Bitrates 的情況下，提升編碼的速度。

3.1 方法一

上一章節所提到的 DAS(Directional Asymmetric Search)，雖然是一種很快速的 Block Matching Algorithm，但由於它是單方向性，且終止條件容易滿足，所以雖然 DAS 搜尋時間快速，可是它在 Bitrates 跟 PSNR 的表現上，就會稍微差一點，而且因為終止條件容易滿足，很容易陷入 Local Minimum 的狀況；也就是說，BMA 可能只是找到區域的最小值(Local Minimum)，而不是 Search Window 內的最小值(Global Minimum)。

所以為了解決這種狀況。我們決定加入 SPS(Search Pattern Switching) Algorithm 的概念進來，利用它裡面的 EDR(Error Descent Rate)，來判斷 Global Minimum 的遠近，再來使用 DAS。因為如果 DAS 的起始位置離 Global Minimum

的距離很遠的話，那麼很有可能在搜尋的途中，就會因為陷入 Local Minimum 的狀況，而導致搜尋終止，所以 DAS 的起始點不宜離 Local Minimum 太遠，而使用 EDR 便可以解決這個狀況。

因此在我們的演算法中，第一步便先在 Search Window 的中心，設一個小型 Diamond Search 的 Pattern(圖 2.2 之起始 Pattern)，並計算這五個點的 Block Distortion，若中心點的 Block Distortion 比周圍的四個點的還小，就會使用 Zero Motion Vector，用以加速演算法的進行。若鄰近的四個點中，有存在一點之 Block Distortion 比中心點的 Distortion 還小，則開始計算 Error Descent Rate，將鄰近四點中，最小點之 Distortion 設為 D_B，將中心點之 Distortion 設為 D_A，令 EDR = D_B/D_A。計算出 EDR 之後，再看看 EDR 有沒有超過門檻值(0.9)，若沒有超過門檻值的話，就可以直接在中心點開始做 DAS(之前做小型 Diamond Search 時，已可將最大及最小點記錄起來，用以決定 DAS 之方向)，但若 EDR 超過門檻值(0.9)，則代表 Global Minimum 是在較遠處，此時我們就要決定，DAS 要從哪裡開始執行起。

於是根據不同的測試影片，去觀察當 EDR 超過門檻值(0.9)時，DAS 的起點應從哪裡開始，才會有較好的效果。根據實驗，當 EDR 超過門檻值(0.9)時，選

擇的 Pattern 會如同圖 3.1。將圖 3.1 上的八個點分別計算完之後，取出其中 Block Distortion 最小的點，與上一步所找出之最小 Block Distortion 之點做比較，若上一步所找之點較小，則以上一步所找的點，將其設為 DAS 的起始位置；反之，則將 Pattern 上最小 Block Distortion 之點設為下一次 DAS 的起始位置。

找到下一次 DAS 的起始位置後，一樣要先做小型的 DS Pattern，看中心點位置是否最小，如果是就以目前中心點位置為 Motion Vector；若最小 Distortion 位置不為中心點，則找出最大及最小 Distortion 位置，決定 DAS 方向，重複 DAS，直到滿足終止條件為止。但在最後，在利用 DAS 找出的最小 Distortion 之點後，再搜尋周圍的八個點，類似像 BBGDS 那樣，以確保剛好因為 DAS 特性而剛好漏掉搜尋的點。

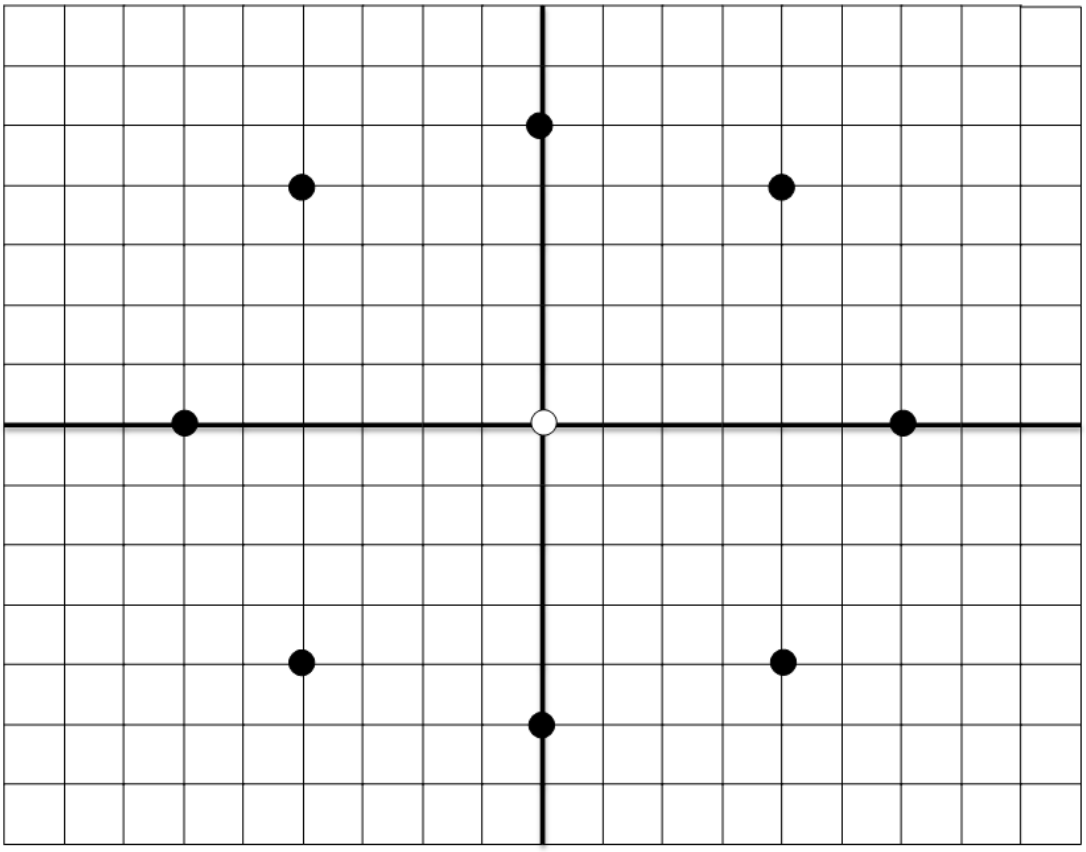

圖 3.1　當 EDR 超過門檻值時，所要搜尋的 Pattern。

3.2 方法二

DAS(Directional Asymmetric Search)是一種相當快速的演算法，單方向性以及每次只需加入三個點的 Pattern 是它快速的主因。我們想改變加入三個點的 Pattern 這部分，以再一次的提升速度。

每次當需要加入新 Pattern 的時候，先計算這次 DAS 方向上的那點，可見圖 3.2；圖 3.2 的方向性是往上，所以應加入的是白色的這三點，因為此時 DAS 的方向性是往上，所以我們先計算編號為 1 的白色點，如果白色點 1 的 Block Distortion 小於上一步之最小點的話，則點 2 及點 3 便不用再計算，將上一步之最小點位置設為這次的最大點位置，點 1 位置設為最小點位置，計算 DAS 的方向。

若點 1 大於上一步所求出之最小點，則依序計算點 2 及點 3；若點 2 小於上一步之最小點，則一樣將上一步之最小點設為最大點位置，將點 2 設為最小點位置，計算 DAS 的方向。

若點 2 也大於上一步所求之最小點，則依序計算點 3，此時便如同 DAS 一樣，在四個點中找出最大最小位置，求其方向性。重複上述的步驟，直到最小點位置不再變化為止。

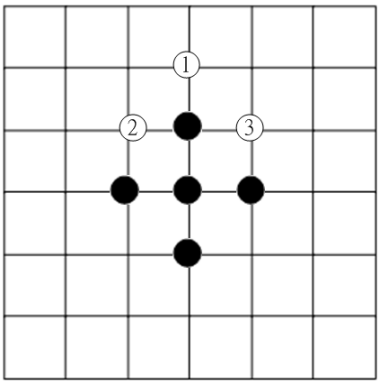

圖 3.2 新的 Pattern 加入順序

因為若點 1 小於上一步之最小點，則方向性依然為上一步驟之 DAS 的方向，可以符合我們一般的認知。若點 1 大於上一步之最小點，則有三種狀況 一)點 2 比點 1 大 二)點 2 比上一步之最小點小 三)點 2 介於兩者之間；而只要點 2 大於上一步之最小點，便一定會計算點 3，這樣便跟 DAS 的 Pattern 一樣了。而點 2 可能比點 3 小，所以偏差的機率只有六分之一。

但在點 1 較上一步最小點還大的狀況下，我們無法去評估，到底先做點 2 好還是點 3 好，所以我們選擇，在發現點 1 大於上一 步最小點後，便做 DAS 的 Pattern，來較完整的估測方向性。

為了解決 MV 太大，DAS 會跑太久，或陷入到 Local Minimum 的狀況，我們決定結合 SPS(Search Pattern Switching)的概念，在起始的小型 Diamond Pattern 做完之後，令中心點之 Distortion 為 D_A，周圍四點最小點的 Distortion 設為 D_B，如果周圍四點皆比中心點之 Distortion 大的話，則最後傳出 Zero Vector；否則計算(D_B / D_A)，用以判斷我們演算法的起始位置。

若(D_B / D_A)小於等於門檻值 0.9 的話，則以 Search Window 中心展開我們的演算法；若(D_B / D_A)大於門檻值的話，則往 DAS 方向以及旁邊的方向跳出，如下圖，圖 3.3 所示，會先測試位於 DAS 方向上白色的點，若白色點比起始 Pattern 的最小點還小的話，則會在該點上作類似 DAS 在該方向性的 Pattern，如圖 3.4 所示；若白色點比起始 Pattern 最小點大的話，則會依序做灰色及黑色的點；若比起始 Pattern 最小點小，則加入該方向性的四個點的 Pattern，若三點皆比起始

Pattern 之點大，則還是從起始 Pattern 展開演算法；否則，計算遠方新加入四點之 Pattern，求得方向性，開始展開我們的新演算法。

這裡會加入跟 DAS 一樣的 Distortion 的門檻值，在計算的過程中，只要有任何點低於門檻值，搜尋便會終止，理由一樣是因為在搜尋下去的效果實在有限，即時終止可以加快搜尋的速度。

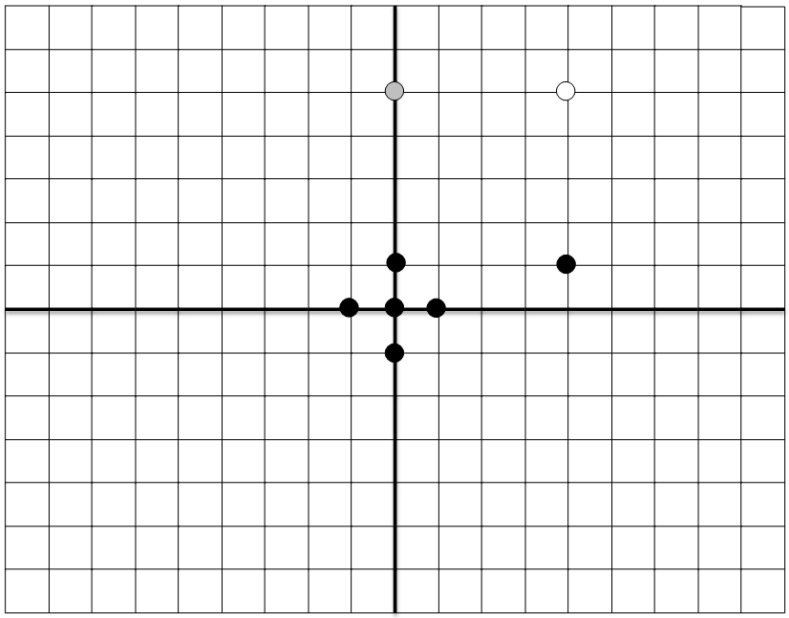

圖 3.3 當判斷為 MV 在遠方時，所展開之 Pattern

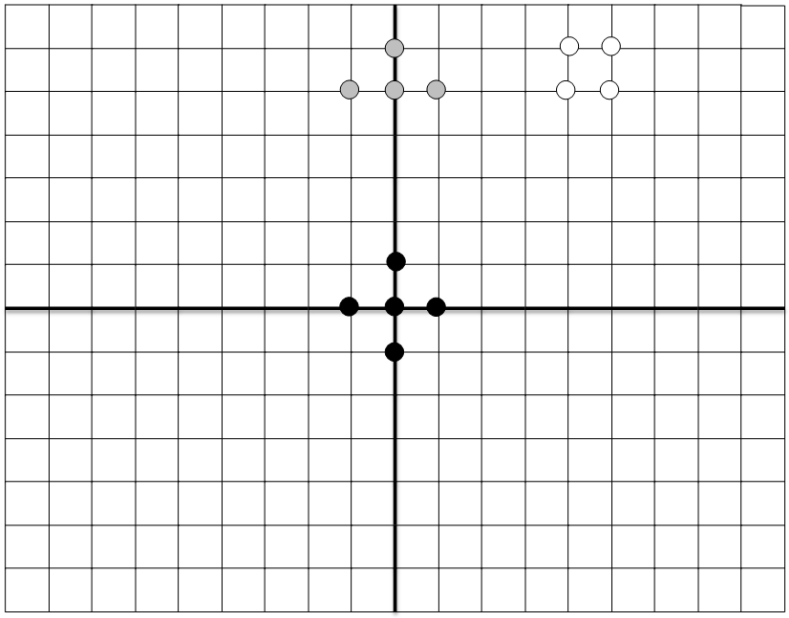

圖 3.4 決定為白點(或灰點)時，所做之 Pattern

3.3 演算法流程

最後我們來整理一下整個流程：

方法一

1) 先在 Search Window 中央做一個小型的 Diamond Search，記錄下五個點中的最大以及最小點，若最小點在 Diamond 形狀 Pattern 的中央，則搜尋終止，使用 Zero Motion Vector；反之，則令中央周圍四點之最大 Distortion 為 D_B，Pattern 中央點之 Distortion 為 D_A，設 EDR = D_B / D_A，檢查 EDR 是否超過門檻值(這裡設為 0.9)。若 EDR 小於等於門檻值，進入步驟 2)。

2) 利用之前記錄下的最大及最小點，以最大點往最小點的方向為 DAS 的方向，並在該方向上加入對應的 Pattern，計算新加入的三個點，並與上一步的最小點 Distortion 做比較，若上一步之最小 Distortion 之點仍為最小點，則終止搜

尋，進入步驟 5)；若 Minimum Distortion 發生在新加入的 Pattern 上，則重複之前加入 Pattern 並比較的動作，直到最小 Distortion 之位置不再變化，進入步驟 5)。

3) 若 EDR 大於門檻值，則展開圖 3.1 的 Pattern，計算這八個點，並與上一步 DS 中之 Distortion 最小點作比較，若上一步之最小點仍為 Minimum Distortion，則以該點做為起始位置；若 Minimum Distortion 發生在圖 3.1 之 Pattern 的八個點上，則以該點作為中心點，做一個小型的 Diamond Search。

4) 在 Diamond 形狀的 Pattern 中，若中心點為 Minimum Distortion，則跳到步驟 5)。若 Minimum Distortion 發生在中心點周圍四個點上，則執行步驟 2)。

5) 以之前找到的 Minimum Distortion 之點為中心，搜尋周圍鄰近的八個點，跟中心點做比較，以最後的 Minimum Distortion 之點為 Motion Vector。

方法二

1) 先計算小型 Diamond Search Pattern 的五個點。

2) 以中心點之 Distortion 為 DA，周圍四點最小之 Distortion 設為 D_B，以(D_B / D_A)跟門檻值做比較，決定展開演算法之位置。若周圍四點之 Distortion 皆大於中心點，則最後傳出 Zero Motion Vector。

3) 若(D_B / D_A)小於等於門檻值則從起始 Pattern 最小點位置展開演算法。

4) 若(D_B / D_A)大於門檻值，則跳出一定距離，在 DAS 方向以及鄰近兩個方向(見圖 3.3)上作測試，若 DAS 方向上該點小於上一步之最小點，則在該點位置作 DAS 該方向性的 Pattern(見圖 3.4)；若 DAS 方向上該點大於上一步之最小點，則依逆時鐘順序測試圖 3.3 上鄰近方向之點，剩下步驟與上面相同。若三點皆大於起始 Pattern 之最小值，則以起始 Pattern 為新演算法展開位置。

第四章實驗結果以及數據探討

4.1 實驗結果

要評估我們演算法的效能的話，就要在 H.264 的實做軟體 JM 14.2 上面實作。我們將使用 Intel Core 2 6320 1.86GHz 的 CPU 還有 1GB 的 RAM，以及在 MS 的 Window XP 下，使用 Visual Studio 2005 來做為我們的實驗環境。我們將會以 tempete、mobile、stefan、foreman、akiyo、news 及 football 等數個 CIF(352 x 288，4:2:0)測試影片來做測試。這邊的 GOP(Group Of Pictures)設為 IPPP，壓縮量化參數 QP(Quantisation Parameter)設為 28。

作為評估效能的項目有 PSNR(Peak Signal to Noise Ratio)、Bitrates 以及 Total Encoding Time；實驗的對照組則以 DAS、SPS 以及 UMHexagonS 做為主要的比較對象，因為 UMHexagonS 是 JM 裡面的一種預設的 BMA，而且它的搜尋速度以及效果都相當優秀，所以也選用它來做為我們的對照演算法。

方法一

表 4.1 Full Search 與 UMHexagonS 的比較

Full Search				UMHeaxagonS		
	PSNR	Bitrates	Time	PSNR	Bitrates	Time
Football	35.61	1544.57	642.369	35.61	1547.95	66.334
Mobile	35.32	2116.12	783.900	35.52	2117.59	85.704
Tempete	35.86	1378.75	701.288	35.85	1377.77	78.022

表 4.2 新演算法實驗結果

新演算法			
	PSNR	Bitrates	Time
Football	35.63	1560.86	50.000
Mobile	35.52	2126.13	67.552
Tempete	35.85	1378.66	60.691

方法二

表 4.3 新演算法實驗結果

		新演算法	
	psnr	**bitrates**	**time**
akiyo	**40.43**	**223.82**	**41.873**
news	**38.92**	**425.59**	**45.194**
foreman	**37.59**	**530.78**	**49.208**
football	**35.63**	**1566.96**	**44.862**
tempete	**35.86**	**1379.39**	**55.796**
mobile	**35.32**	**2130.01**	**62.354**
stefan	**36.64**	**1397.72**	**25.041**
coastguard	**35.86**	**1149.98**	**56.075**

表 4.4 DAS 實驗數據

		DAS	
	psnr	**bitrates**	**time**
akiyo	**40.43**	**223.86**	**45.88**
news	**38.92**	**425.84**	**49.402**
foreman	**37.6**	**533.95**	**54.021**
football	**35.63**	**1568.27**	**49.109**
tempete	**35.85**	**1378.66**	**60.445**
mobile	**35.32**	**2130.67**	**66.945**
stefan	**36.65**	**1399.2**	**27.083**
coastguard	**35.86**	**1149.97**	**60.819**

表 4.5 與 DAS 比較之加速百分比

	akiyo	**news**	**foreman**	**football**	**tempete**	**mobile**	**stefan**	**coastguard**
$\triangle T$	**8.73%**	**9.31%**	**8.91%**	**8.65%**	**7.69%**	**6.86%**	**7.54%**	**7.80%**

表 4.6 SPS 實驗數據

		SPS	
	psnr	**bitrates**	**time**
akiyo	**40.44**	**224.82**	**45.138**
news	**38.92**	**431.52**	**49.192**
foreman	**37.63**	**571.49**	**58.288**
football	**35.64**	**1602.33**	**51.479**
tempete	**35.86**	**1397.11**	**61.526**
mobile	**35.32**	**2165.35**	**68.636**
stefan	**36.68**	**1454.64**	**28.158**
coastguard	**35.88**	**1171.68**	**64.386**

表 4.6 UMHexagonS 實驗數據

		UMHexagonS	
	psnr	**bitrates**	**time**
akiyo	**40.44**	**223.59**	**48.867**
news	**38.92**	**423.66**	**56.539**
foreman	**37.59**	**512.32**	**70.925**
football	**35.61**	**1547.95**	**66.163**
tempete	**35.85**	**1377.77**	**77.859**
mobile	**35.32**	**2117.59**	**85.402**
stefan	**36.63**	**1369.1**	**33.438**
coastguard	**35.86**	**1145.35**	**86.442**

表 4.8 與 UMHexagonS 比較之加速百分比

	akiyo	**news**	**foreman**	**football**	**tempete**	**mobile**	**stefan**	**coastguard**
$\triangle T$	**14.31%**	**20.07%**	**30.62%**	**32.19%**	**28.34%**	**26.99%**	**25.11%**	**35.13%**

4.2 數據探討

方法一

實驗的結果在表 4.1~4.2，可以看到我們的演算法在 PSNR 上面可以跟 UMHexagonS 達到一樣的水準，而 Bitrates 在稍差(不到 1%)的狀況下，快了 17%~20%的 Total Encoding Time。

我們使用 UMHexagonS 做為對照組，由表 4.1 可得知，UMHexagonS 效能非常逼近 Full Search，PSNR 部分與 Full Search 近乎一樣，Bitrates 部分也是微微的落後，相差非常小的距離；但 UMHexagonS 可以提升相當多的效率，所以 UMHexagonS 可以算是一種相當優秀的演算法，因此使用它做為對照組。

方法二

實驗的結果在表 4.3~表 4.6，我們的新演算法可以在 PSNR 幾乎一樣，而 Bitrates 損失不到 1%的狀況下，在整體編碼時間上，對於 DAS 有 8%左右的提升。

第五章 結論

實驗的結果顯示我們的方法有很好的成效，分述如下:

方法一

我們提出一種新的 Block Motion Algorithm，它利用 DAS (Directional Asymmetric Search)可以快速完成搜尋的特性，結合了 SPS (Search Pattern Switching)可以估測 Motion Vector 遠近的功能，來克服 DAS 容易碰到 Local Minimum 的狀況，可以找到更好或是最好的 Motion Vector。新的演算法可以在跟 UMHexagonS 比較時，有相同的 PSNR 以及差距不到 1%的 Bitrates 狀況下，快了 17%~20%的 Encoding Time。

方法二

提供了一種更快速的 BMA，利用 SPS 估測 MV 的優點，可以避免做太長的 DAS 或陷入 Local Minimum 的狀況，保留大部分 DAS 的特性，在更節省計算點數的狀況下，我們可以獲得比 DAS 快 8%及比 UMHexagonS 快 26%左右速度的提升，與 DAS 比較 Bitrates 會升高非常微小的幅度；與 UMHexagonS 約略會升高 1% Bitrates。

Motion Estimation 的搜尋演算法已經被許多人研究過了，算是一個很成熟的研究題目，未來我們想再進一步提升的話，除了更新穎的創意之外，再來就要好好觀察跟 Motion Vector 有關的資訊，才能更有效的估計到 Motion Vector 的位置，進一步的去想出更有效的演算法，去提升搜尋的速度。

參考文獻

[1] “Advanced Video Coding for Generic Audiovisual Services (ITU-T Rec.H.264 | ISO/IEC 144496-10 AVC),” Joint Video Team of ISO/IEC and ITU-T, 2005.

[2] Iain E. G.Richardson , “H.264 and MPEG-4 Video Compression,”wiley , Baker & Taylor Books, 2003.

[3]Shan Zhu and Kai-Kuang Ma, “A new diamond search algorithm for fast block-matching motion estimation,” *Image Processing, IEEE Transactions*, Volume: 9, Issue: 6, Feb, 2000 Page(s):287-290.

[4] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, “Motion compensated interframe coding for video conferencing, ” in *Proc. Nat. Telecommun, Conf., NO, L.A., Nov-Dec 1981*, pp. G5.3.1-G.5.3.5.

[5] L. K. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding, ” *IEEE Trans. on Circuits and Systems for Video Technology,* vol. 6, pp. 419-422, Aug 1996.

[6] C.M. Kuo, Y.H. Kuan, C.H. Hsieh and Y.H. Lee , “A Novel Prediction-Based Directional Asymmetric Search Algorithm for Fast Block-Matching Motion Estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 19, No. 6, 2009, pp893-899.

[7] K.H. Ng, L.M. Po, K.M. Wong, C.W. Ting and K.W. Cheung, “A Search Patterns Switching Algorithm for Block Motion Estimation,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 19, No. 5, 2009, pp753-759.

[8]C.H. Cheung and L.M. Po, “Novel Cross-Diamond-Hexagonal Search Algorithms for Fast Block Motion Estimation,” *Multimedia, IEEE Transactions on,* Volume 7, Issue 1, Feb. 2005 Page(s):16 – 22

[9] H. Zeng, C. Cai, and K.K. Ma**,** “Fast Mode Decision for H.264/AVC Based on Macroblock Motion Activity,” *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 19, No. 4, 2009, pp.1-10.