國立臺灣師範大學
資訊工程研究所碩士論文

指導教授： 黃 文 吉 博士

適用於數位全像顯微鏡系統影像認證之浮水印法則開發之研究

A Watermarking Algorithm for Holographic Microscopy System Authentication

研究生： 曾 瀚 逸 撰

中華民國 一 百零 二 年 七 月

中文摘要

本論文提出一個適用於數位全像顯微鏡系統(Digital Holographic Microscopy，DHM)影像認證的脆弱型浮水印法則，同時提供針對傳送錯誤或是惡意篡改擁有高敏感度，並且能有效的偵測出來。本法則的浮水印嵌入於離散餘弦轉換(Discrete Cosine Transform，DCT)域後並沒有做量化。藉由有效的增強全像圖的儲存精度，我們能夠控制因嵌入浮水印而造成的破壞。最後實驗結果也呈現本法則的浮水印在重建後可以用於過濾破壞的資料或對惡意篡改的偵測。

關鍵字：數位全像顯微鏡系統、認證、浮水印

Abstract

A fragile watermarking algorithm for hologram authentication is presented in this paper. While providing high perceptual transparency, the proposed algorithm attains high performance detection to delivery errors and malicious tampering. In the proposed algorithm, the watermark is embedded in the discrete cosine transform (DCT) domain of a hologram without quantization. The distortion produced by the watermark can be controlled. It can be effectively lowered by enhancing the precision for storing the watermarked hologram pixels. Experimental results reveal that the proposed algorithm can be used as a filter for blocking polluted or tampered holograms from reconstruction.

keyword：Holographic Microscopy System、Authentication、Watermarking

誌謝

感謝 黃文吉老師在這兩年內對我不辭辛勞的細心指導，無論是在研究上或者是待人處事方面均獲益良多，在此致上最深的謝意。同時感謝國立中央大學通訊工程系　張寶基博士、國立台灣師範大學光電科技研究所　鄭超仁博士能夠百忙之中撥冗參加本人的口試審查，並給予本人更多的建議與指導。也感謝國立台灣師範大學光電科技研究所的李易達同學以及賴信吉同學給予論文上許多的幫助與指導，使本人的論文能夠順利完成。最後感謝實驗室的同學以及一路上支持我的人，讓我能夠順利完成碩士這個學位。

目錄

附表目錄

附圖目錄

第一章 緒論

本章節主要說明本論文的研究背景、動機以及目的。最後會簡略敘述各章節重要的內容架構。

1.1 研究背景與動機目的

隨著數位全像(Digital Holography，DH)技術的到來，全像圖可以透過 charged-coupled devices (CCD) 和高速電腦來記錄以及重建。DH 技術提供高效能的 3D 影像以及展示，這些特性對其他領域的應用很具吸引力。在度量衡學、生物學以及消費性電子等其他以 DH 技術為基礎的領域，全像圖的儲存和傳輸都是必須的，所以儲存跟傳輸所造成的破壞就是個需要改善的議題。另外，當全像圖在進行無線傳送時可能發生隨機或是突發性的破壞，這些被破壞的資料往往會造成重建出來的 3D 影像不正確，使我們無法做出正確的判斷，所以對以 DH 技術為基礎的系統，資料的正確性就顯得格外重要。

為了維持資料的正確，對資料加密是一個值得採用的方法，雖然使用加密演算法效果很不錯[1, 2]，但相對地它也包含了大量的數學計算，提高了計算複雜度。此外，加密演算法並無提供資料解密之後的保護，這會讓資料暴露於危險之中，若我們使用嵌入浮水印的方法，則我們將不用考慮這些缺點。嵌入浮水印的重要目標之一，就是我們只需嵌入一些秘密資訊(浮水印)於原始的資料當中，之後任何的錯誤或修改我們都能察覺出來，而這種型態的浮水印我們稱之為脆弱型浮水印[3]，跟用於保護版權的強健型浮水印相反[4, 5, 6]。嵌入浮水印並不需要太多

的數學計算，因此我們可以即時的嵌入或是擷取出浮水印。另外，嵌入浮水印提供了全時的保護，因為當我們嵌入浮水印後，浮水印將不再是一個輕易就能切割出來的部分。

對於以 DH 技術為基礎的系統，嵌入浮水印除了確保資料還原後的正確性外，還提供一個額外的優點，就是降低接收端的計算負擔。在典型的全像圖傳輸系統中，接收端負責的工作之一就是 3D 影像的重建工作，其中包括了複雜的數學計算，例如菲涅耳轉換(Fresnel transform)和相位展開法則(phase unwrapping)。因此，若能在 3D 影像重建之前就先擷取浮水印，就能過濾掉資料破壞嚴重的全像圖，確定做還原的全像圖皆為正確的資料。另外透過嵌入浮水印這種認證機制，也可過濾掉未經授權的使用者利用我們系統來做還原的動作，確保使用的人皆為我們所授權過的，進而提升整個系統的效能。

現今的脆弱型浮水印技術可以依據他們嵌入的領域分成兩種類型[7]。第一種是直接在空間域(spatial domain)做運算[8, 9]，很多嵌入在空間域的浮水印技術都是透過把浮水印嵌入在影像中最不重要位元(Least Significant Bits，LSBs)來達到高敏感度的目標，但是這個技術並不一定適合用來偵測因為傳輸或是惡意篡改所產生的錯誤，因為當發生錯誤的位置是在最重要位元(Most Significant Bits，MSBs)，或是其他不是 LSBs 的時候，這些因為傳輸或是惡意篡改所產生的錯誤我們可能就無法偵測出來。

第二種嵌入脆弱型浮水印的演算法則是把浮水印嵌入於轉換域底下[10]，因為在轉換域，任何的改變在 MSBs 或是 LSBs 都會影響到其他的係數，也因此當

發生傳送錯誤或是惡意篡改時，我們可以比嵌入於空間域的法則更容易偵測出錯誤。然而，大部分嵌入於轉換域的浮水印法則通常都把浮水印嵌入在做完轉換計算以及量化過後的係數底下，然而因量化而產生的誤差會降低影像重建後的正確性，這也是現今嵌入於轉換域的浮水印法則不適用於嵌入對象為全像圖的主要原因之一。

本篇論文主要目的是要實現出一個適用於以 DH 系統為基礎的脆弱型浮水印嵌入技術，提供透過浮水印來對全像圖做認證和檢查資料的正確性。本篇的技術是將浮水印嵌入於 DCT 域底下。首先將欲嵌入浮水印的全像圖先切割成不重疊且等大小的區塊，之後對每個區塊做 DCT 轉換，我們為了可以彈性的控制因嵌入浮水印而造成的破壞，所以我們對每個做完 DCT 轉換的係數並沒有做量化的動作。此外，我們可以改變已嵌入浮水印的全像圖的儲存精度，較高的儲存精度可以降低影像的失真程度，因為我們將已嵌入浮水印的全像圖的資訊以較大的儲存空間來儲存。最後提出實驗數據證明本論文提出的嵌入浮水印法則擁有很高的敏感度，可以正確地偵測出因傳送或是惡意篡改所造成的錯誤。

1.2 全文架構

本篇論文共分為四章，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】浮水印法則介紹

詳細說明本論文所提出的浮水印法則及其數學分析。

【第三章】實驗成果與數據探討

呈現本論文所提出的浮水印法則成果數據以及結果討論。

【第四章】結論

對本論文做最後的總結。

第二章 浮水印法則介紹

在本章中，我們將詳細介紹本論文所提出的浮水印法則，以及透過數學分析證明其有效性與正確性。

2.1 簡介

本論文提出一個適用於以 DH 系統為基礎的脆弱型浮水印嵌入技術，提供透過浮水印來對全像圖做認證和檢查資料的正確性。整個浮水印法則我們分成兩個部份來探討，第一部份我們先介紹浮水印的嵌入流程以及浮水印的嵌入法則。我們的浮水印嵌入於 DCT 域底下，計算完後並沒有做量化的動作，而是直接對 DCT 運算過後的係數做嵌入浮水印的動作，之後再做 IDCT 計算，然後我們會依據我們所期望的儲存精度的大小對每個係數做調整，最後再四捨五入使其值為整數，以上為整個嵌入過程的初步介紹。

第二部分我們介紹浮水印的擷取，我們首先會將接收到的資料做 DCT 轉換，獲得我們藏於 DCT 域底下的浮水印資訊，最後產生出浮水印影像。接下來的小節裡，我們將仔細介紹整個浮水印的嵌入以及擷取過程，最後就是對整個法則的數學分析，證明其合理性與正確性。

2.2 浮水印嵌入法則

浮水印嵌入流程，如圖 2.1 所示

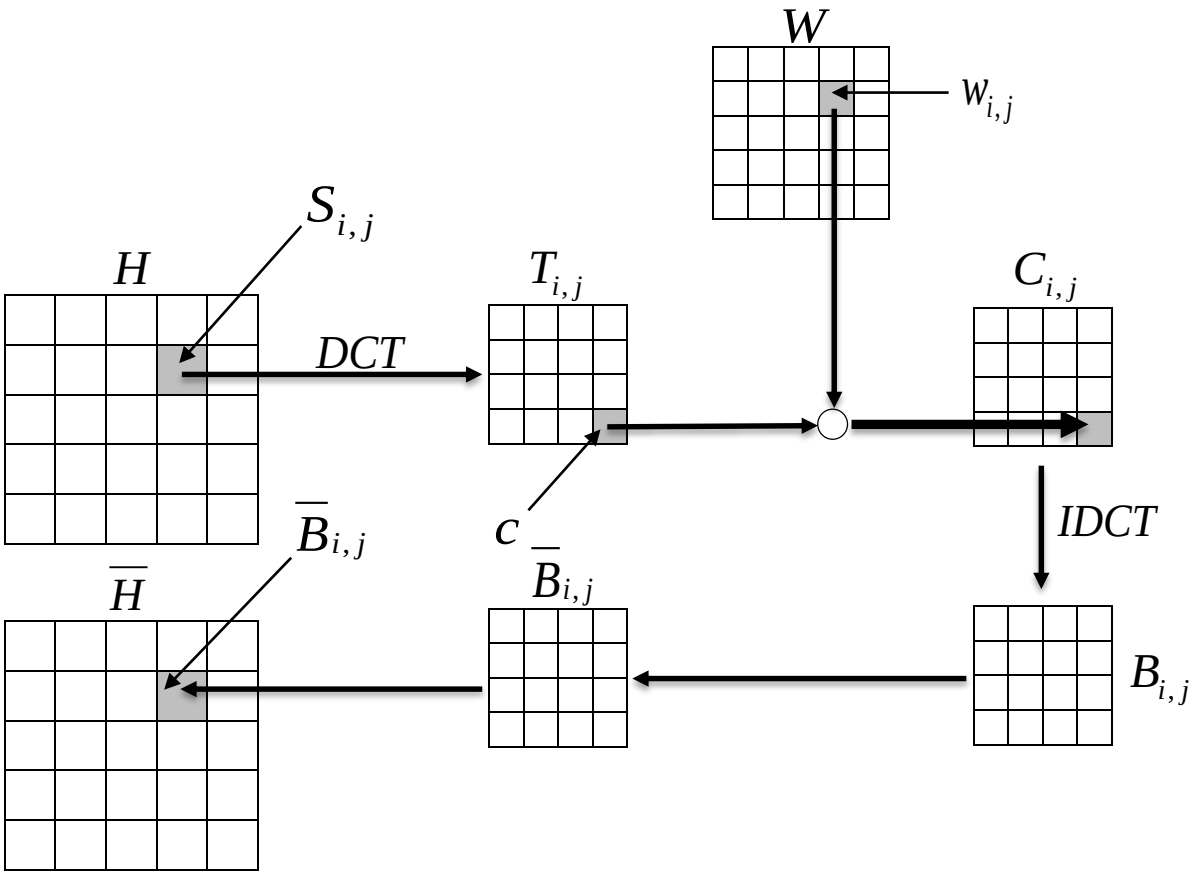


圖 2.1 浮水印嵌入架構圖

W 為一張二進制的浮水印影像，而 H 為欲嵌入浮水印的全像圖。假設 W 跟 H 的維度分別為 $2^n \times 2^n$ 跟 $2^N \times 2^N$ ，其中 $N > n$。為了嵌入浮水印，我們把全像圖 H 分割成每一個不重疊且等大小的區塊 $S_{i,j}$ ，其中 $i = 1,\ldots,2^n$，$j = 1,\ldots,2^n$，每個區塊的大小為 $2^k \times 2^k$，$k = N - n$。令 $w_{i,j}$ 為浮水印影像 W 中(i, j)-th 之間的元素。本論文提出的浮水印法則是將 $w_{i,j}$ 嵌入於 DCT 域底下的 $S_{i,j}$。令 $T_{i,j}$ 為一個取自於 $S_{i,j}$ 中做完 DCT 轉換後的係數的集合，而我們嵌入浮水印時只改變 $T_{i,j}$ 裡取

名為 c 的係數，其他的係數則維持不變。

$w_{i,j}$	p	$I(p)$	p^*
0	even		$p^* \leftarrow p$
0	odd	$I(p) < c$	$p^* \leftarrow p+1$
0	odd	$I(p) > c$	$p^* \leftarrow p-1$
1	even	$I(p) < c$	$p^* \leftarrow p+1$
1	even	$I(p) > c$	$p^* \leftarrow p-1$
1	odd		$p^* \leftarrow p$

表 2.1 p^*的處理參考表

為了把 $w_{i,j}$ 嵌入於 c 中，首先我們切割實數軸 L 成為一個不重疊且等區間的實數軸。當 $w_{i,j}$ 為 0 的時候，c 會移動到最近的指標為 *even* 的區間裡，否則，c 會移動到最近的指標為 *odd* 的區間裡。令 p 為 c 所在區間裡的指標，其計算方法為

$$p = \left\lfloor \frac{c}{L} \right\rfloor \tag{1}$$

另外，令區間 p 的中心點為 $I(p)$，其算法為

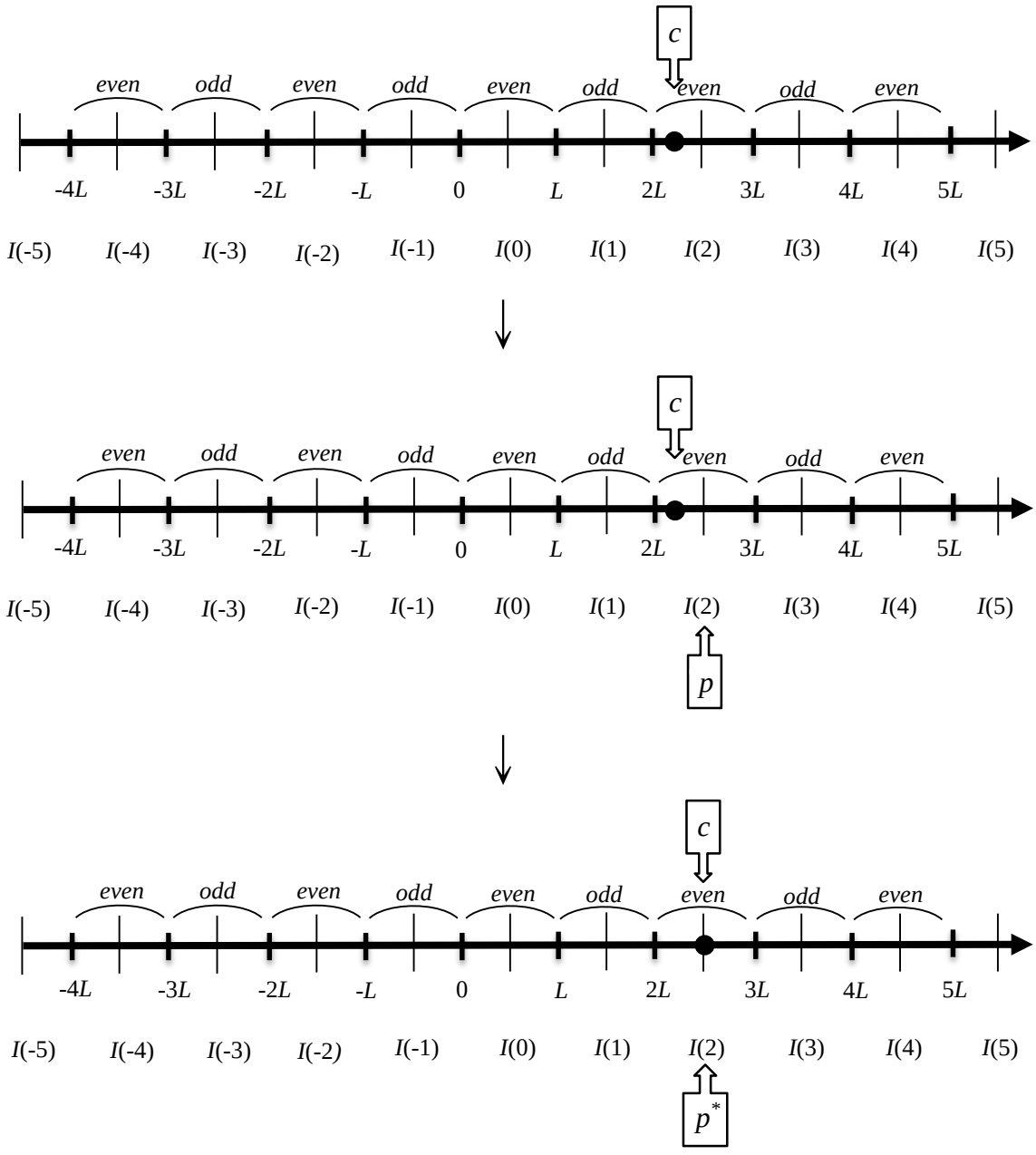
$$I(p) = p \times L + \left(\frac{L}{2} \right) \tag{2}$$

根據 $w_{i,j}$ 、c、p 和 $I(p)$的值，我們可以直接找出最近指標為 p^*的區間。表 2.1 所表示的是所有情況的處理過程。在 p^*找到之後，我們就可以計算出新的係數 c，藉由

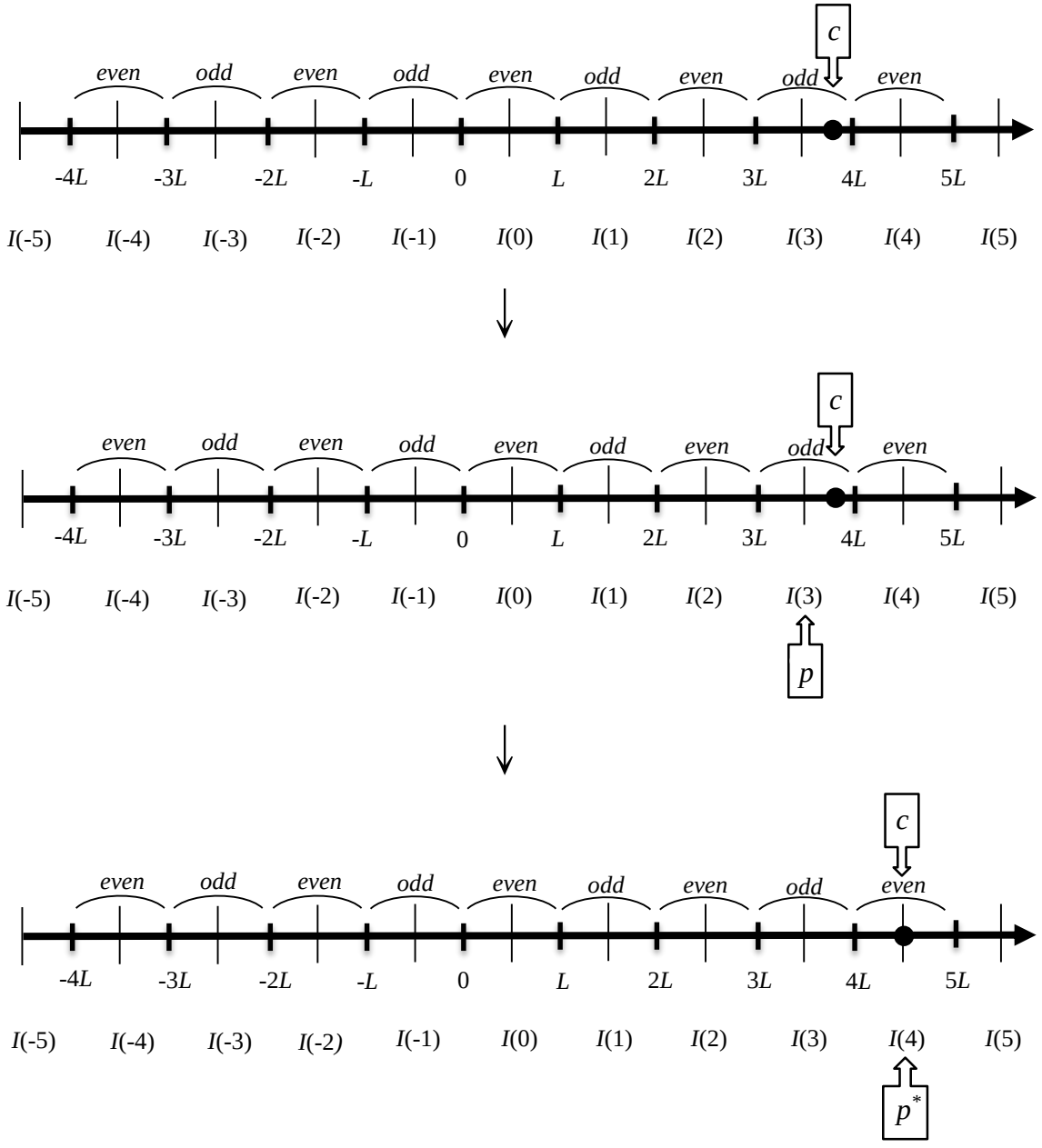
$$c = I\left(p^*\right) \tag{3}$$

經過以上步驟就完成 c 的嵌入處理。接下來我們對每個情況做詳細的介紹。

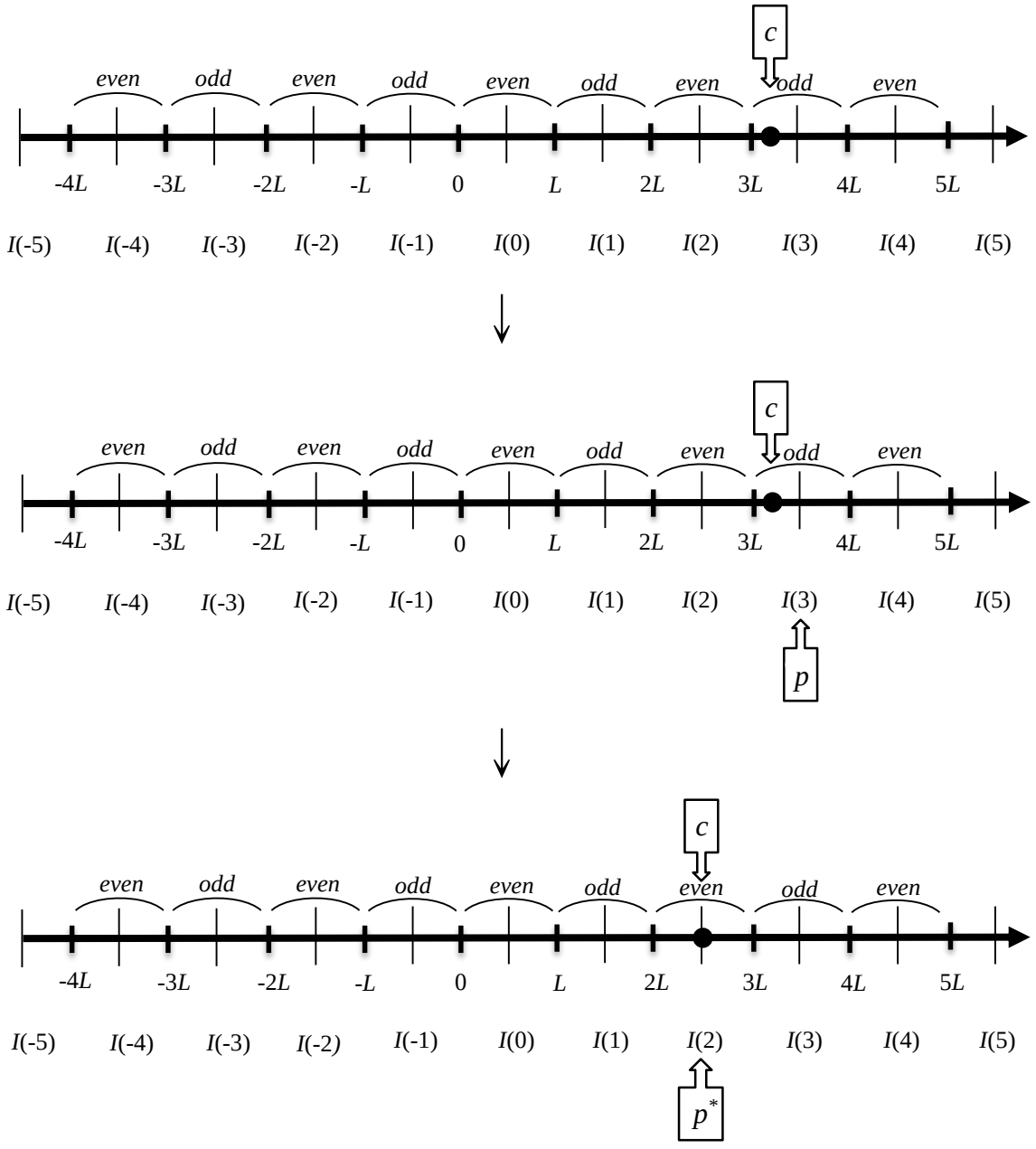
情況一. $w_{i,j} = 0$，p 為偶數區間的指標



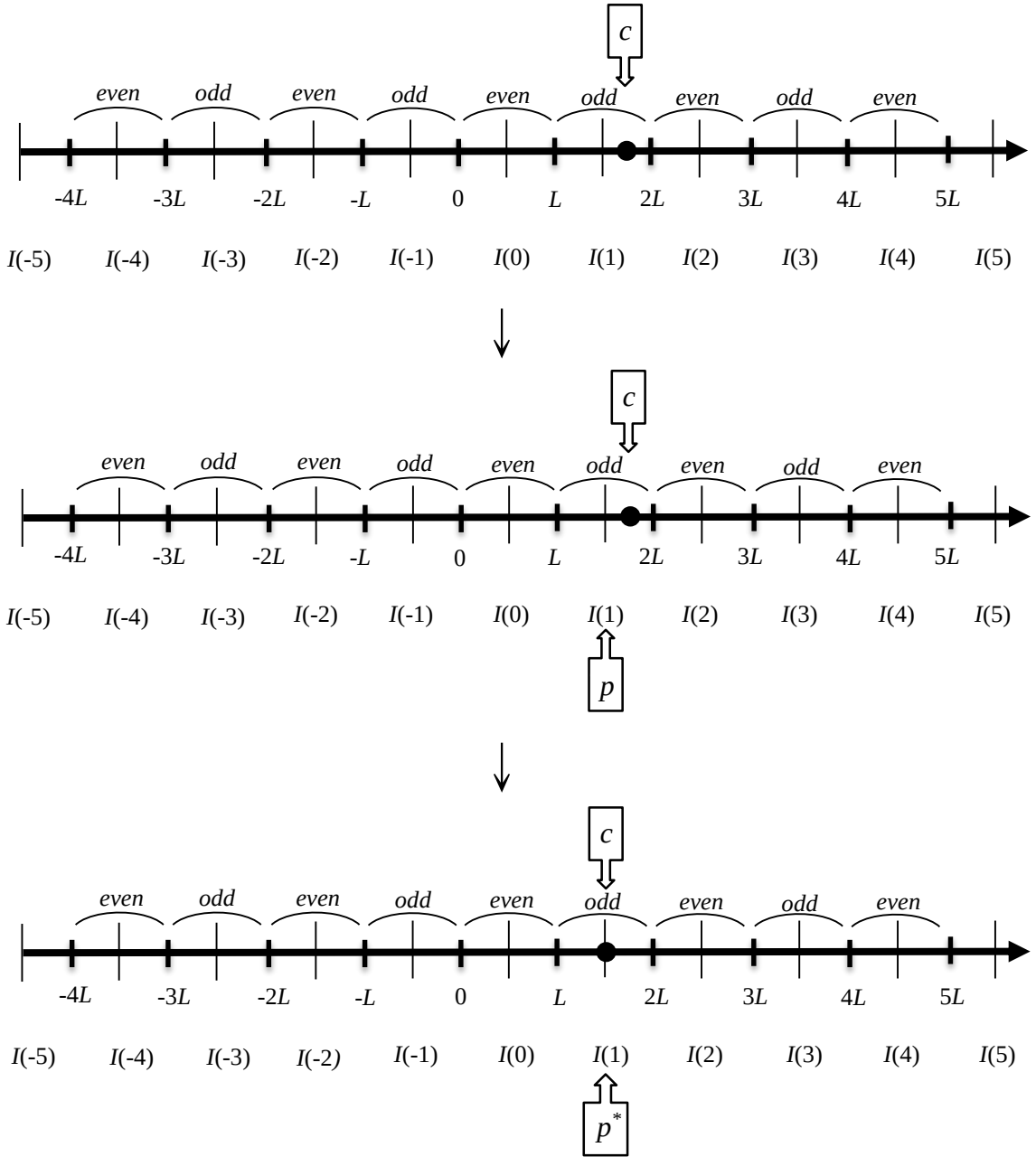
情況二. $w_{i,j} = 0$，p 為奇數區間的指標，且 $I(p) < c$



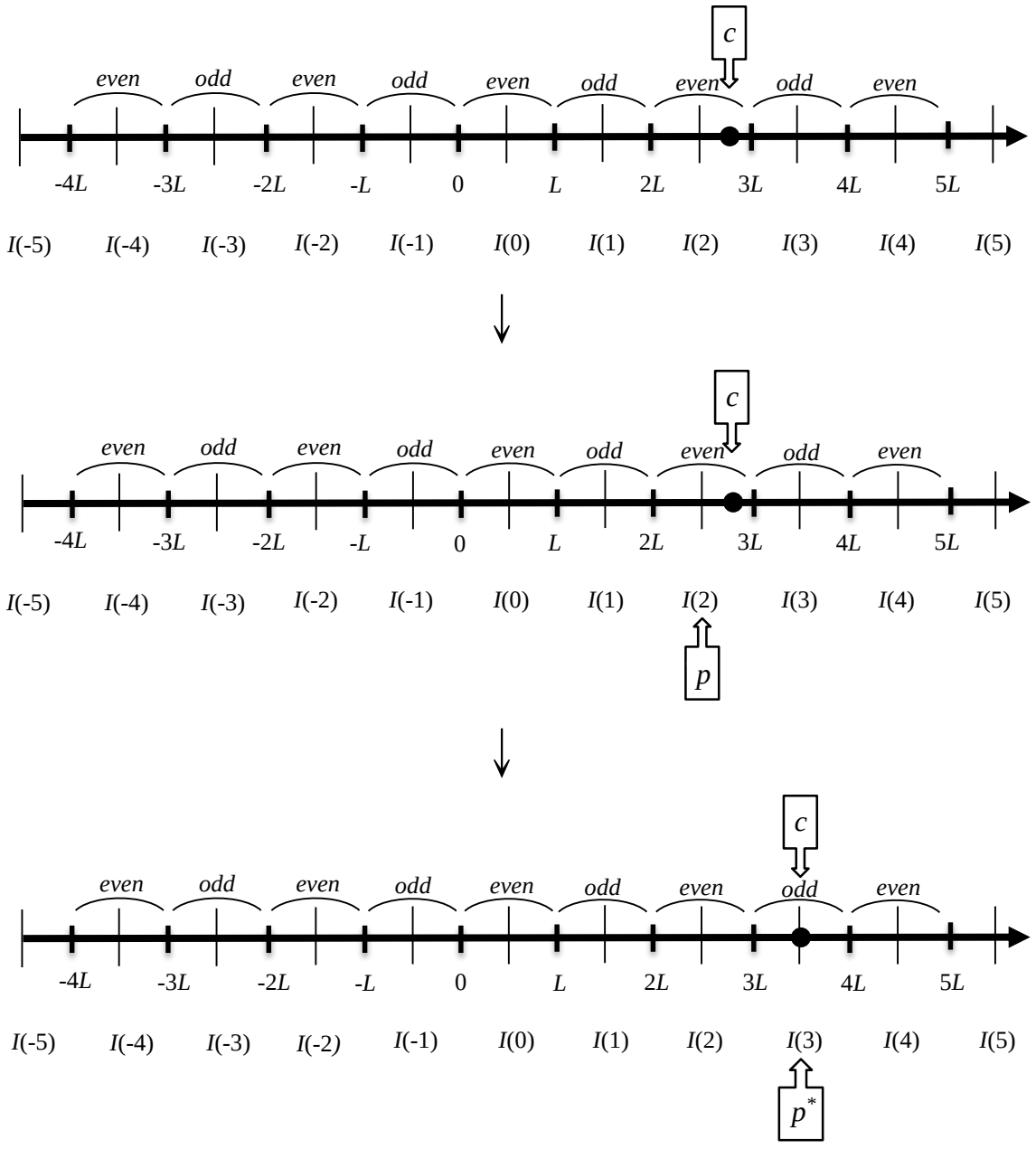
情況三. $w_{i,j} = 0$，p 為奇數區間的指標，且 $I(p) > c$



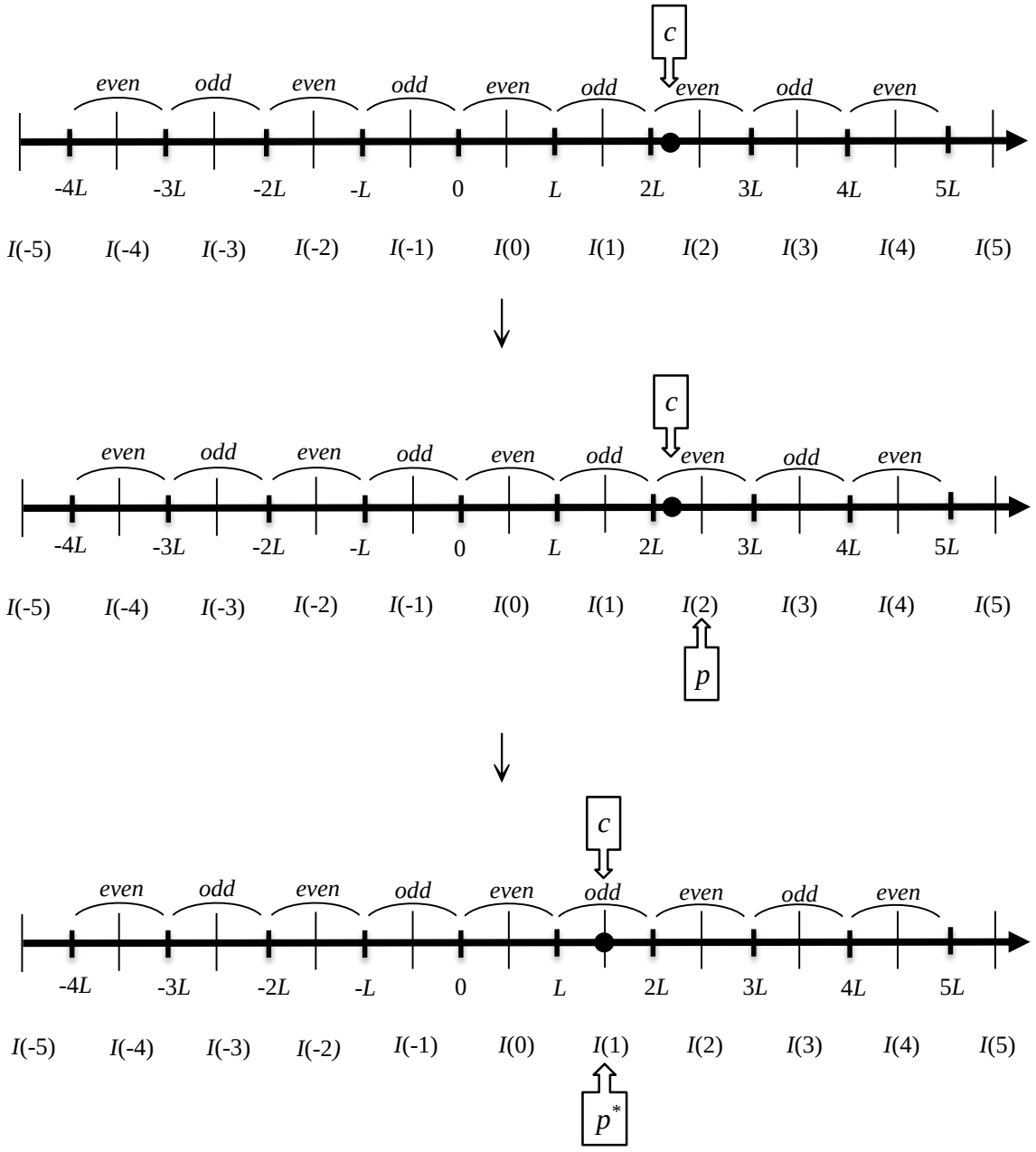
情況四. $w_{i,j}=1$，p 為奇數區間的指標



情況五. $w_{i,j} = 1$，p 為偶數區間的指標，且 $I(p) < c$



情況六. $w_{i,j} = 1$，p 為偶數區間的指標，且 $I(p) > c$



經過上述的處理過程，嵌入浮水印於 c 中才算完成。在 $T_{i,j}$ 中，$w_{i,j}$ 只嵌入於 c 中，其他的係數則保持不變，因此整個嵌入過程在 DCT 域底下只產生些微的破壞。令 $C_{i,j}$ 為嵌入浮水印後的 $T_{i,j}$，$B_{i,j}$ 為 $C_{i,j}$ 做完 IDCT 後的區塊。我們可以視

$B_{i,j}$ 為 $S_{i,j}$ 嵌入 $w_{i,j}$ 資訊之後的版本。

我們可以注意到 $B_{i,j}$ 裡的各個元素皆為實數。令 $\overline{B}_{i,j}$ 為用有限精度表示的 $B_{i,j}$。$\overline{B}_{i,j}$ 的取得來自於對 $B_{i,j}$ 裡的每個元素做四捨五入。我們設 m 為用多少個位元數來讓 $\overline{B}_{i,j}$ 的值可以用整數格式來表示，m 越高代表用更高的精度來表示嵌入浮水印後的區塊。我們定義 $\overline{H}=\{\overline{B}_{i,j}, i=1,...2^n, j=1,...,2^n\}$，因此，$\overline{H}$ 可以視為原始全像圖 H 嵌入浮水印 W 在 DCT 域底下後的版本。以下為根據 H 和 W 產生 $\overline{H}$ 的處理過程的總結

嵌入浮水印 W 在全像圖 H 中的處理程序

Step 0: 起始: 浮水印影像 W 和全像圖 H。

Step 1: 為了嵌入浮水印把全像圖 H 分割成區塊 $S_{i,j}, i=1,...,2^n, j=1,...,2^n$。

Step 2: 對 $S_{i,j}$ 做 DCT 轉換獲得 $T_{i,j}$。

從 $T_{i,j}$ 中選取 c 出來。

Step 3: 嵌入 $w_{i,j}$ 後得到 $C_{i,j}$，而 c 的處理程序根據表 2.1 及 eq.(3)。

Step 4: 對 $C_{i,j}$ 做 IDCT 計算得到 $B_{i,j}$。

對 $B_{i,j}$ 裡的全部元素做四捨五入使其可以用 m 位元來儲存。

Step 5: 得到 $\overline{H}=\{\overline{B}_{i,j}, i=1,...2^n, j=1,...,2^n\}$。

2.3 浮水印擷取法則

浮水印擷取流程，如圖 2.2 所示

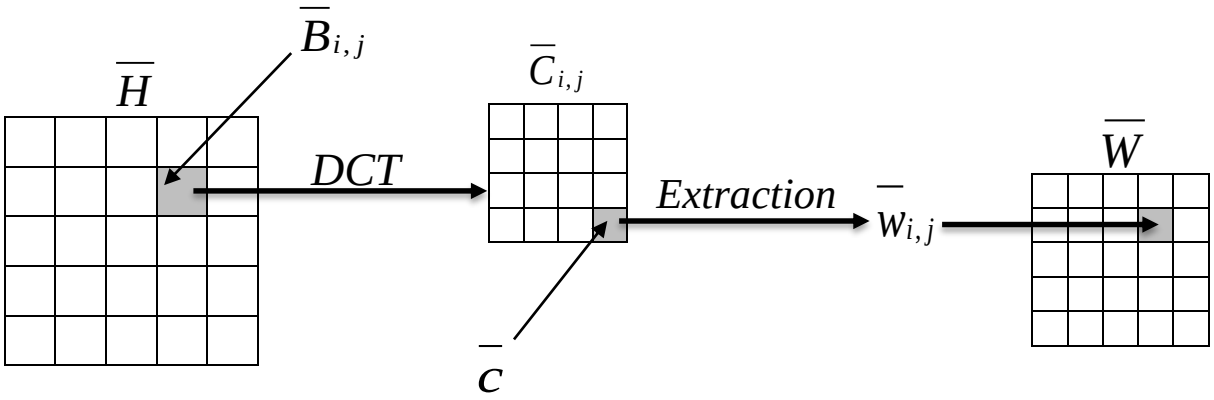


圖 2.2　浮水印擷取架構圖

令$\overline{W}$為從$\overline{H}$擷取出來的浮水印影像。為了擷取出浮水印影像$\overline{W}$，首先我們對每個區塊$\overline{B}_{i,j}, i=1,...2^n, j=1,...,2^n$做 DCT 轉換。在這裡有一點值得注意，雖然$B_{i,j}$是由$C_{i,j}$做 IDCT 計算後得到的，但$\overline{B}_{i,j}$做完 DCT 轉換後不一定等於$C_{i,j}$，原因是為了讓$\overline{B}_{i,j}$內的每一個像素都是用有限精度來表示，所以對每個像數都會做四捨五入，而其產生的誤差也存於$\overline{B}_{i,j}$之中。對一個有四捨五入誤差的$\overline{B}_{i,j}$做 DCT 轉換會導致與$C_{i,j}$算出來的 DCT 係數不同。令$\overline{C}_{i,j}$為有四捨五入誤差的$\overline{B}_{i,j}$做完 DCT 轉換後的結果。

當得到$\overline{C}_{i,j}$之後，我們會檢查$\overline{C}_{i,j}$裡的$\overline{c}$係數。其中$\overline{c}$跟c在 DCT 域底下會分享同一個位置。檢查過程包含了從$\overline{c}$係數計算出$\overline{p}$，而$\overline{p}=\left\lfloor \frac{\overline{c}}{L} \right\rfloor$。當$\overline{p}$為偶數

時，與其相對應擷取出的浮水印影像，我們定義為$\overline{w}_{i,j}$，其值為 0，否則其值為 1。當所有的區塊都檢查過後，擷取出來的浮水印影像我們命名為$\overline{W}=\{\overline{w}_{i,j}, i=1,...,2^n, j=1,...,2^n\}$。最後，我們會比較 W 跟$\overline{W}$，當$W\neq\overline{W}$時，我們就偵測出該全像圖可能遭到惡意的篡改或是因傳輸而造成的錯誤。從$\overline{H}$擷取出浮水印$\overline{W}$的處理流程我們整合如下

從全像圖$\overline{H}$擷取出浮水印$\overline{W}$的處理程序

Step 0: 起始: 全像圖$\overline{H}$。

Step 1: 為了擷取浮水印將全像圖$\overline{H}$分割成區塊$\overline{B}_{i,j}, i=1,...2^n, j=1,...,2^n$。

Step 2: 對$\overline{B}_{i,j}$做 DCT 轉換獲得$\overline{C}_{i,j}$。

從$\overline{C}_{i,j}$中選取$\overline{c}$出來。

Step 3: 根據$\overline{p}$從$\overline{c}$找出$\overline{w}_{i,j}$。

當$\overline{p}$為奇數時，$\overline{w}_{i,j}=1$，否則$\overline{w}_{i,j}=0$。

Step 4: 得到$\overline{W}=\{\overline{w}_{i,j}, i=1,...2^n, j=1,...,2^n\}$。

2.4 浮水印法則數學分析

從 eqs.(1)、(3)中，我們知道較小的 L 值可以減少因嵌入浮水印而造成的破壞。此外，用較少的位元數 m 來表示 $\overline{B}_{i,j}$ 裡的各個像素可以降低已嵌入浮水印的全像圖 $\overline{H}$ 的儲存大小，然而我們不能事先分開指定 L 值跟 m 值的參數為何。

有一個有趣的議題，就是本論文提出的浮水印法則可以根據 L 值的大小來控制 $\overline{B}_{i,j}$ 對抗攻擊以及錯誤的敏感度，當然包含了因為 m 而造成的四捨五入誤差。因為較小的 L 值可以增強浮水印法則的敏感度，所以給定一個 m 值後，必須可以提供一個 L 值的最低限度來確保不會因為四捨五入的關係而產生出錯誤的結果。

為了找出給定 m 值之後 L 值的最低限度，我們要先探討在 DCT 域底下因四捨五入所造成的影響，為了讓我們的描述更淺顯易懂，我們使用之前出現在論文中的 $B_{i,j}$ 、$\overline{B}_{i,j}$ 、$C_{i,j}$ 、$\overline{C}_{i,j}$ 、$w_{i,j}$ 和 $\overline{w}_{i,j}$ 來做表示。

因為每個區塊 B 跟 $\overline{B}$ 都包含了 2^k 個行，我們可以表示成

$$B=[b_1,...,b_{2^k}],\ \overline{B}=[\overline{b}_1,...,\overline{b}_{2^k}] \tag{4}$$

其中 b_q 跟 $\overline{b}_q$ 用來表示 B 跟 $\overline{B}$ 裡的第 q 行。每一行在區塊內都包含了 2^k 個元素。

區塊 B 中的第 q 行其四捨五入誤差平方和我們命名為 e_q，算式為

$$e_q = \sum_{u=1}^{2^k} \left(b_q(u) - \bar{b}_q(u) \right)^2 \tag{5}$$

其中 $b_q(u)$ 跟 $\bar{b}_q(u)$ 分別代表 b_q 跟 $\bar{b}_q$ 裡的第 u 個元素。我們可以把等式改寫成向量型式

$$e_q = \left(\bar{b}_q - b_q \right)^T \left(\bar{b}_q - b_q \right) \tag{6}$$

我們定義區塊 B 的四捨五入誤差的平方和為 E_B，其四捨五入誤差的平方和來自其全部的行，其算式為

$$E_B = \sum_{q=1}^{2^k} e_q = \sum_{q=1}^{2^k} \left(\bar{b}_q - b_q \right)^T \left(\bar{b}_q - b_q \right) \tag{7}$$

因為 C 和 $\overline{C}$ 分別為 B 跟 $\overline{B}$ 做完 DCT 轉換後的結果，我們可以從 B 跟 $\overline{B}$ 計算出 C 和 $\overline{C}$ 透過

$$C = ABA^T \tag{8}$$

$$\overline{C} = A\overline{B}A^T \tag{9}$$

其中 A 是 $2^k \times 2^k$ 的 DCT 矩陣 $\left\{ a_{i,j}, i = 1,...,2^k, j = 1,...,2^k \right\}$，且

$$a_{i,j} = \rho_i \cos \frac{(2j+1)i\pi}{2 \times 2^k}, \tag{10}$$

$$\rho_i = \begin{cases} \sqrt{1/2^k} & i = 0 \\ \sqrt{2/2^k} & i > 0 \end{cases} \tag{11}$$

我們下一步要證明 C 與 $\overline{C}$ 之間的平方差跟 B 的四捨五入誤差的總和會相同。我

們定義 y_q 跟 $\bar{y}_q$ 如下所示

$$y_q = Ab_q,\ \bar{y}_q = A\bar{b}_q \tag{12}$$

因此，

$$\left(\bar{y}_q - y_q\right) = A\left(\bar{b}_q - b_q\right) \tag{13}$$

從 eq.(13)，我們可以得到

$$\left(\bar{y}_q - y_q\right)^T\left(\bar{y}_q - y_q\right) = \left(\bar{b}_q - b_q\right)^T A^T A\left(\bar{b}_q - b_q\right) \tag{14}$$

因為 DCT 矩陣 A 為正交矩陣，所以 $A^T A = I$ ，接下來

$$\left(\bar{y}_q - y_q\right)^T\left(\bar{y}_q - y_q\right) = \left(\bar{b}_q - b_q\right)^T\left(\bar{b}_q - b_q\right) \tag{15}$$

現在我們定義兩個 $2^k \times 2^k$ 矩陣 Y 和 $\bar{Y}$ 如下表示

$$Y = \left[y_1,...,y_{2^k}\right],\ \bar{Y} = \left[\bar{y}_1,...,\bar{y}_{2^k}\right] \tag{16}$$

其中 y_q 和 $\bar{y}_q$ 分別表示為 Y 和 $\bar{Y}$ 中的第 q 行，如下

$$Y = AB,\ \bar{Y} = A\bar{B} \tag{17}$$

令 E_Y 為 Y 和 $\bar{Y}$ 之間的平方差，因此

$$E_Y = \sum_{q=1}^{2^k}\left(\bar{y}_q - y_q\right)^T\left(\bar{y}_q - y_q\right) \tag{18}$$

從 eqs.(7)、(15)，我們可以知道

$$E_Y = \sum_{q=1}^{2^k} \left(\bar{b}_q - b_q\right)^T \left(\bar{b}_q - b_q\right) = E_B \tag{19}$$

因此，E_Y 跟區塊 B 擁有同樣的四捨五入誤差的平方和，值得注意的一點，從 eqs.(8)、(9)、(17)，Y、$\bar{Y}$、C 和 $\bar{C}$ 之間的關係可以表示為

$$C = ABA^T = YA^T, \tag{20}$$

$$\bar{C} = A\bar{B}A^T = \bar{Y}A^T \tag{21}$$

因此，

$$C^T = AY^T,\ \bar{C}^T = A\bar{Y}^T \tag{22}$$

定義 E_C 為 C 和 $\bar{C}$ 之間的平方差。再一次地，因為 DCT 矩陣 A 為正交矩陣，所以我們可以從 eq.(22)認定

$$E_C = E_Y \tag{23}$$

結合 eqs.(19)、(23)，我們得到

$$E_C = \sum_{q=1}^{2^k} \left(\bar{b}_q - b_q\right)^T \left(\bar{b}_q - b_q\right) = E_B \tag{24}$$

以上證明了 C 和 $\bar{C}$ 之間的平方差跟區塊 B 的四捨五入誤差的平方和是一樣的。

為了我們以上概論的真實性，假設 H 裡的每一個像素值都在 $[0,256)$ 之內，我們用 8 個位元就可完整的表示出所有的像素值。給定一個有限精度的 $m(m \geq 8)$ 來表示每一個像素值，像素值的分數部分包含了 $m-8$ 個位元。單精度表示的像素值的四捨五入誤差的大小應該要小於 $2^{-(m-7)}$ (i.e. 平方後應小於 $2^{-2(m-7)}$)，所以當每一個區塊的大小皆為 $2^k \times 2^k$ 時，其最大上限 E_B 為

$$E_B \leq 2^{2k-2(m-7)} \tag{25}$$

因為從 eq.(24)我們得到 $E_C = E_B$，其最大上限跟 E_C 的最大上限一樣，這表示了 $E_C \leq 2^{2k-2(m-7)}$。C 裡的係數 c 跟 $\bar{C}$ 裡的係數 $\bar{c}$ 他們的平方差皆會小於 E_C，因為這樣，$(\bar{c}-c)^2 \leq E_C$，因此

$$(\bar{c}-c)^2 \leq 2^{2k-2(m-7)} \tag{26}$$

從 eq.(3)我們可以知道一件事，c 永遠位於以長度 L 為單位的區間的中心，因此當 c 跟 $\bar{c}$ 的平方差大於 $(L/2)^2$ 時，擷取出來的浮水印影像 $\bar{w}$ 跟當初嵌入的浮水印影像 w 就會不一樣，所以當 L 滿足下列條件

$$2^{2k-2(m-7)} \leq (L/2)^2 \tag{27}$$

根據 eq.(26) $(\bar{c}-c)^2 \leq 2^{2k-2(m-7)}$，且 $\bar{w} = w$。也就是說，eq.(27)提供了 L 的最低限度來確保當用精度 m 來表示時，浮水印 w 還是可以正確的被擷取出

第三章 實驗成果跟數據探討

本章節主要是呈現並且討論我們提出之法則的一些實驗結果。

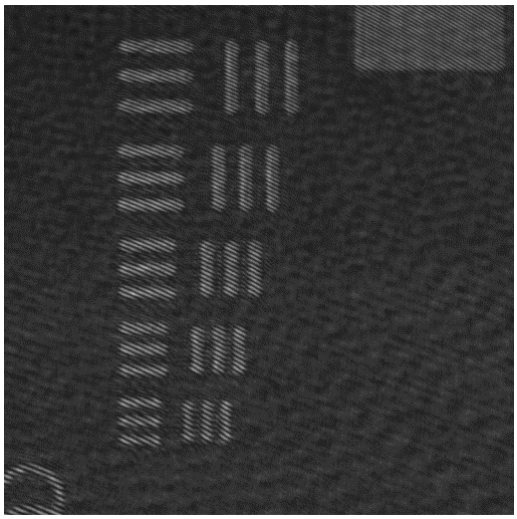
3.1 開發平台與測試對象

本論文提出的法則主要以 JAVA 語言實作，並搭配以 matlab 撰寫出的還原程式為輔。之所以用 JAVA 語言為我們實現的語言原因為我們將來是有可能實作出來應用在客戶端或是系統端上，而客戶端的裝置可能為智慧型手機或是平板電腦，這些行動裝置我們主要以 android 系統為主，選擇 android 系統的原因為其普及率是目前行動裝置上最高的，而 android 系統的應用程式主要都以 JAVA 語言實作，所以使用 JAVA 語言來實作是最為適合的。另外 JAVA 語言本身也提供了許多對我們有用的功能，整理成以下四點:

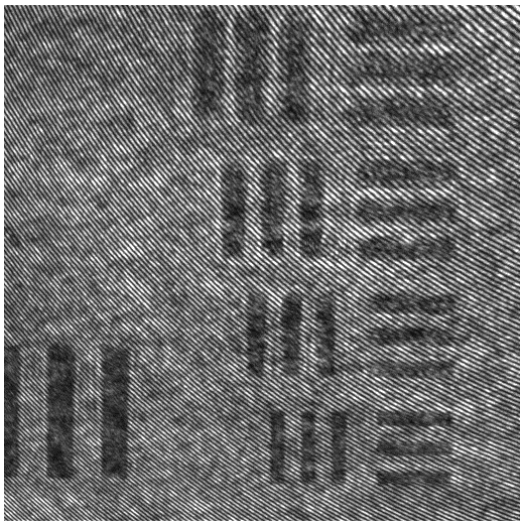
1. 跨平台: 由於 JAVA 的 JVM(Java Virtual Machine)技術，可在各種作業系統上建立起自己的平台，故擁有跨平台的特性。若是未來此浮水印法則需要在平板電腦或是手機上執行，只需設計其使用者介面，毋須再重新設計程式。
2. 簡單而不複雜: JAVA 語言本身提供許多類別與函式庫給開發者使用，以供開發者用來撰寫應用程式。若撰寫程式時使用物件導向的設計，將來若需要維護則相對地比較容易。
3. 資源回收處理(Garbage Collection)：剛剛提到 JAVA 中的 JVM 技術，不僅可達到跨平台的功能，還可以幫助開發者做程式的動態記憶體管理，讓開發者不用擔心記憶體的問題。

4. 例外處理(Exception) ：JAVA 的例外處理機制，主要用來顯示程式中有可能出現問題的地方的資訊，以供開發者修改或維護，若是執行時發生錯誤，開發者就可以容易地發現問題點，不需浪費多餘的時間來找出問題。

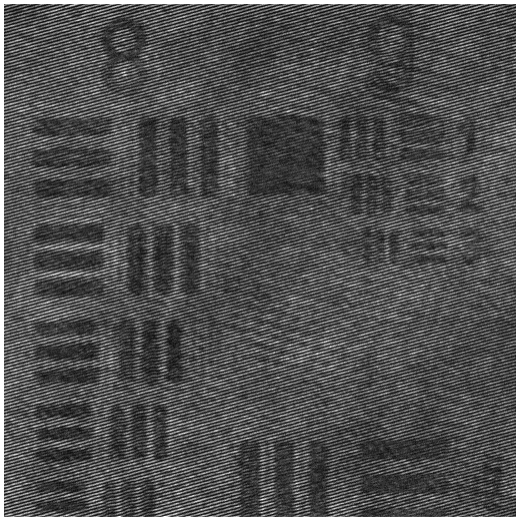
測試對象為四張 512x512 的全像圖(圖 3.1(a)、圖 3.1(b)、圖 3.1(c)、圖 3.1(d))，為了方便之後結果的呈現我們把每張全像圖都命名(Hologram 1、Hologram 2、Hologram 3、Hologram 4)，以及一張 128x128 的二元浮水印影像(圖 3.1(e))，其圖片在下頁顯示。我們使用 128x128 大小的浮水印影像主要原因在章節 2.2 中的嵌入浮水印部分中有提到我們會將讀進來的全像圖做切割的動作，而我們區塊切割的大小為 4x4，512x512 的全像圖表示會有 128x128 個區塊，所以浮水印影像用 128x128 的大小才能讓每一個位置的元素剛好都能分配到每一個 128x128 的區塊裡，以上為我們使用 128x128 大小的浮水印影像的主要原因。



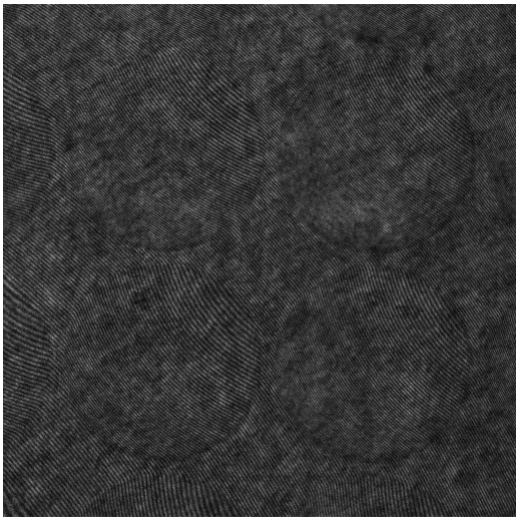
(a) Hologram 1



(b) Hologram 2



(c) Hologram 3



(d) Hologram 4

(e) 二元浮水印

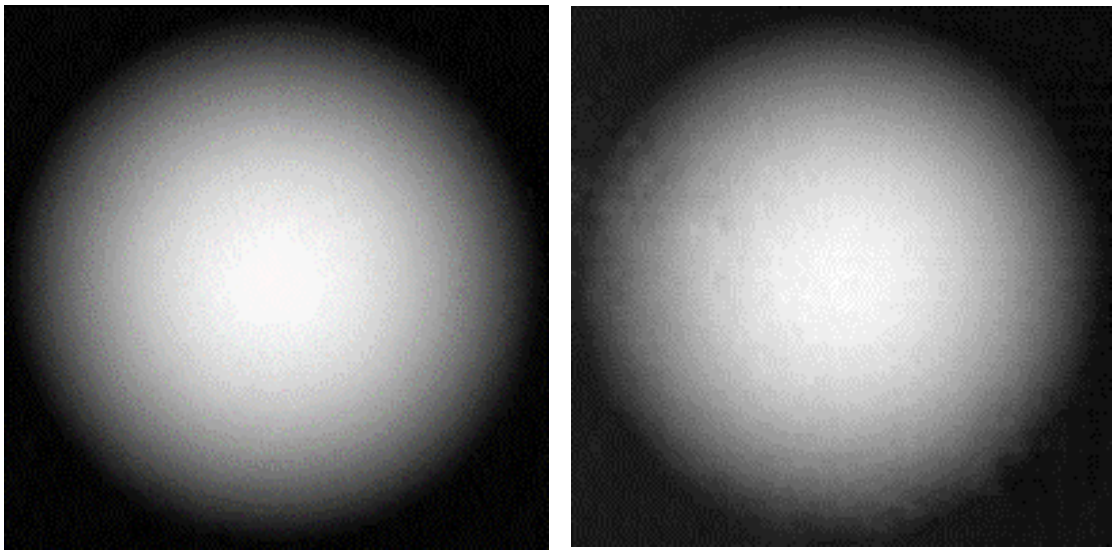
圖 3.1 測試資料及二元浮水印
其中全像圖皆為台灣師範大學光電科技研究所所提供；
(a) Hologram 1、(b) Hologram 2、(c) Hologram 3、(d) Hologram 4

3.2 實驗數據的呈現與討論

本節主要分成幾個小節來討論。3.2.1 節呈現之前章節提到過的不同影像格式對於重建品質之影響，主要測試格式為用 $m=8$ 及 JPG 兩種格式。3.2.2 節主要呈現章節 2.2 提到過的 m 跟 L 之間的關係，當對 m 跟 L 做調整後，對全像圖還原後的結果所造成的影響將在這節詳細的討論。之後我們假設了兩個情況來驗證我們浮水印法則的效能與其正確性。在 3.2.3 節模擬全像圖透過無線傳輸時發生封包破壞或是其他使接收資料不正確的情況，當我們接收到資料時，我們透過擷取浮水印來偵測我們接收到的資料是否正確，所以浮水印的呈現及討論為這小節的重點。3.2.4 節模擬全像圖遭到惡意篡改等其他情況使儲存在資料庫中的全像圖不正確，當要還原的時候，我們先擷取出浮水印來判斷全像圖是否有被破壞，若擷取出破壞嚴重的浮水印我們就捨棄這個資料，而在這個實驗中我們調整篡改的大小來觀察浮水印的變化，用以驗證浮水印的結果是否如我們所預期的一樣。

3.2.1 JPG 格式對於重建品質之影響

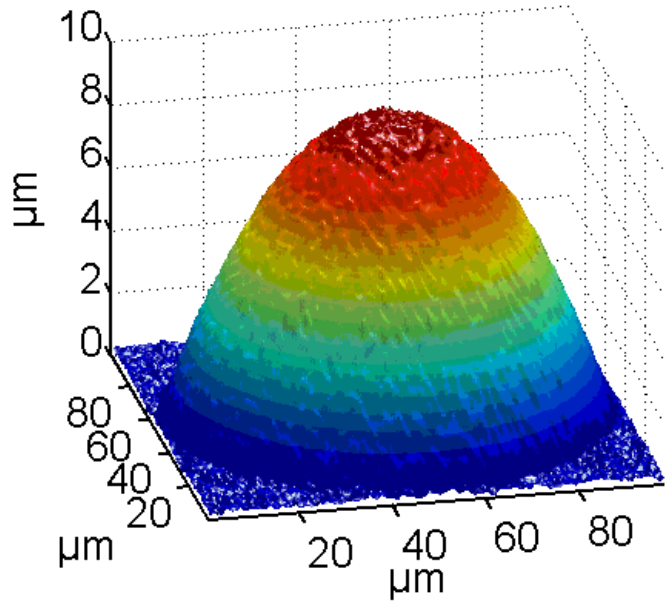
我們以 Hologram 4 來做我們的測試對象，分別用 $m=8$ 及 JPG 格式來儲存，兩者都沒有嵌入浮水印，結果呈現如圖 3.2 及圖 3.3 所示。圖 3.2 為二維相位展開影像，我們可以發現圖 3.2(a)還原出的影像半球附近的邊緣比較圓滑清楚，而圖 3.2(b)則比較模糊不清，兩者的成像品質有很大的不同，其 PSNR 為 26.8263。圖 3.3 為三維相位展開影像，我們也可以發現圖 3.3(a)還原出來的邊比較平滑且完整，圖 3.3(b)則比較崎嶇不平且有缺陷，這也證明了不同的影像格式對全像圖的重建會有很大的影響。



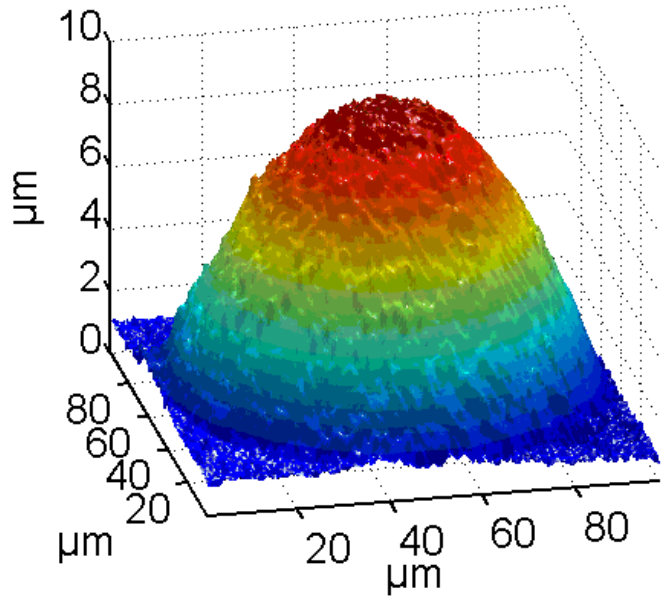
(a) $m=8$ 儲存且無嵌入浮水印還原出來的二維相位展開影像

(b) JPG 格式儲存且無嵌入浮水印還原出來的二維相位展開影像

圖 3.2 不同影像格式未嵌入浮水印還原出的二維相位展開影像



(a) $m=8$儲存且無嵌入浮水印還
原出來的三維相位展開影像



(b) JPG 格式儲存且無嵌入浮水印
還原出來的三維相位展開影像

圖 3.3 不同影像格式未嵌入浮水印還原出的三維相位展開影像

3.2.2 浮水印控制參數的討論

討論章節 2.2 提到過的 m 跟 L 之間的關係，以四張全像圖做為我們的測試對像。我們透過 PSNR 計算來驗證我們結果，而 PSNR 計算皆以還原過後的影像來測量。我們之所以要用還原過後的影像而不用未還原過後的全像圖來做測量主要原因有二:

1. 我們不知道原本存在於未還原影像的錯誤會不會因還原程式的計算而擴大，使得還原出來的影像品質不合乎我們的要求，所以如果還原過後出來的結果包含了原本加入浮水印所造成的破壞還是很好的話，就表示我們的浮水印法則對影像破壞程度很小。
2. 一般我們都以全像圖還原過後的影像來做判斷，所以還原過後的影像正不正確會比未還原的影像來的重要，所以透過對還原後的影像做 PSNR 計算，我們才知道這個還原後的影像是否能提供我們做出正確的判斷。

以上兩點為我們為什麼要使用還原過後的影像來做 PSNR 量測的主要原因。

我們發現隨著 m 值的增加 PSNR 也會隨著增加，但理論上 PSNR 並不會一直無限制的往上提升,所以我們必須知道每張全像圖還原過後的 PSNR 所能達到的最大上限，首先我們簡單介紹 PSNR 的計算，其計算過程如下

Step 0: 起始: A:為全像圖 H 還原後的結果，還原過程如圖 3.4 所示。因全像圖 H 的不同可能還原出不只一張 A，例如只還原出一張振幅影像，或是還原出振幅影像和相位展開影像。

B:為全像圖 $\overline{H}$ 還原後的結果，還原過程如圖 3.5 所示，與 A 主要差別在有嵌入浮水印。因全像圖 $\overline{H}$ 的不同可能還原出多張 B，例如只還原出一張振幅影像，或是還原出振幅影像和相位展開影像。

Step 1: 計算出兩圖的 Mean Squared Error (MSE)。

$$\mathrm{MSE} = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(A(i,j) - B(i,j)\right)^2$$

Step 2: 最後計算出 PSNR。

$$\mathrm{PSNR} = 10\cdot\log_{10}\left(\frac{255^2}{MSE}\right)$$

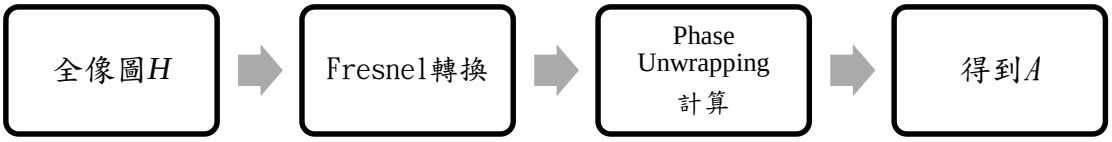


圖 3.4 全像圖 H 的還原過程

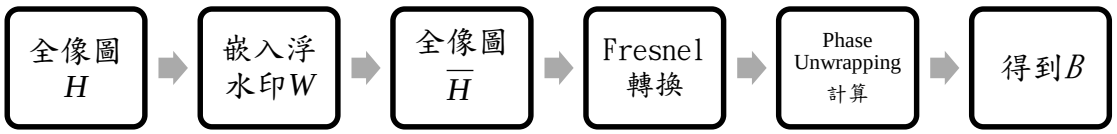


圖 3.5 全像圖 H 嵌入浮水印與其還原過程

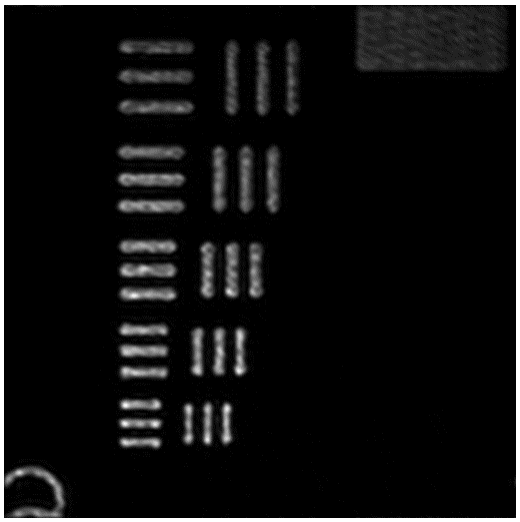
我們令嵌入浮水印的全像圖還原後的影像其 PSNR 的最大上限為 P_0。關於 P_0 的獲得，嵌入浮水印的全像圖我們用 $m=64$ 來表示，代表用雙精度的儲存格式來儲存，保留所有的小數部分。當 $m=64$，根據 eq.(27) $2^{2k-2(m-7)} \le (L/2)^2$，其 L 應該為一個極小的數值，為了方便計算，在這裡我們假設 L 為 0.0001。圖 3.8、圖 3.10、圖 3.12、圖 3.14、圖 3.16 虛線上的數字就是以上面的設定算出來的 P_0。我們希望嵌入浮水印的全像圖還原後的影像其 PSNR 越接近 P_0 越好，其中我們可以透過控制 m 來改善嵌入浮水印所造成的破壞，當用 m 個位元儲存的已嵌入浮水印的全像圖還原後的影像做完 PSNR 計算的結果是最接近 P_0，表示這張全像圖用 m 個位元來儲存是最適合的。

我們的測試皆以可以正確擷取出浮水印為前提，從 $m=8$ 開始，對 m 一路往上加到使有嵌入浮水印的全像圖還原後的影像計算出的 PSNR 最接近 P_0，而這之間也需要調整 L 來確保浮水印不會因為四捨五入的關係而遭到破壞，根據 eq.(27) $2^{2k-2(m-7)} \le (L/2)^2$，$m$ 跟 L 之間的關係如表 3.1 所示

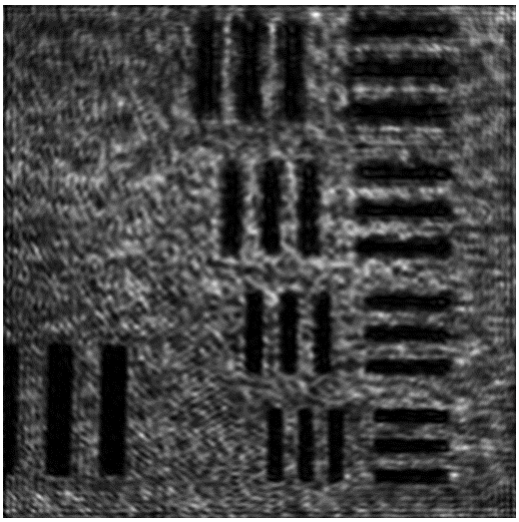
m	***L***
8	4
9	2
10	1
11	0.5
12	0.25
13	0.125
14	0.0625
15	0.03125
16	0.015625
17	0.0078125
18	0.00390625

表 3.1　*m* 與 *L* 之間的對應

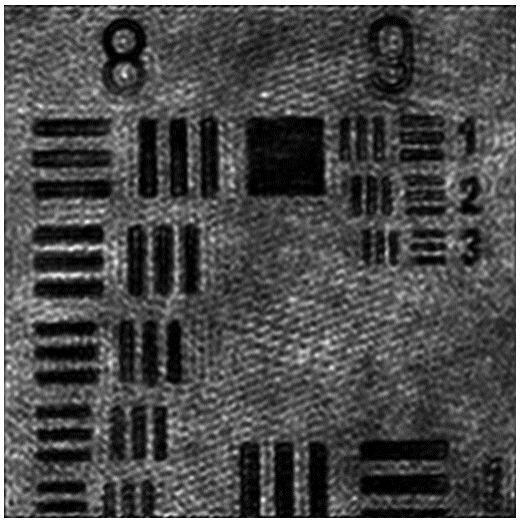
我們可以發現隨著 *m* 值的增加 *L* 則呈倍數減少，*m* 跟 *L* 之間的關係是互相對應的無法分開來指定。以下的實驗結果為呈現每張全像圖在嵌入浮水印後用不同的 *m* 個位元儲存，其還原後的影像 PSNR 值的變化。圖 3.6(a)、圖 3.6(b)、圖 3.6(c)、圖 3.6(d) 、圖 3.6(e)為每張全像圖在未嵌入浮水印時所還原出來的影像，以供後面結果比較之用。



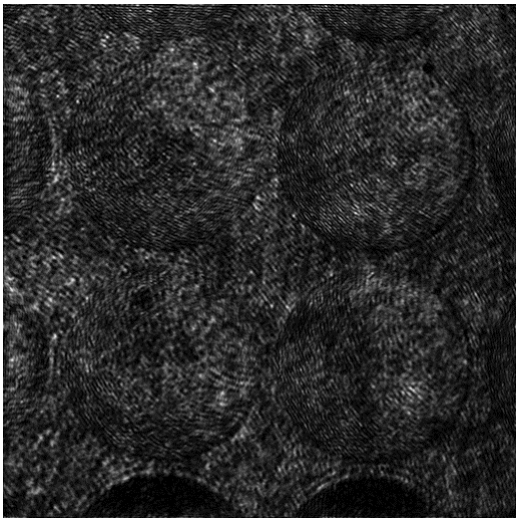
(a) Hologram 1 未加入浮水印還原後之振幅影像



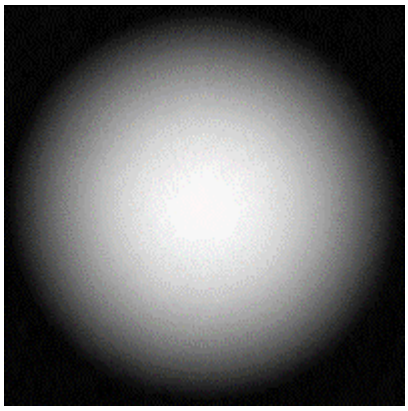
(b) Hologram 2 未加入浮水印還原後之振幅影像



(c) Hologram 3 未加入浮水印還原後之振幅影像

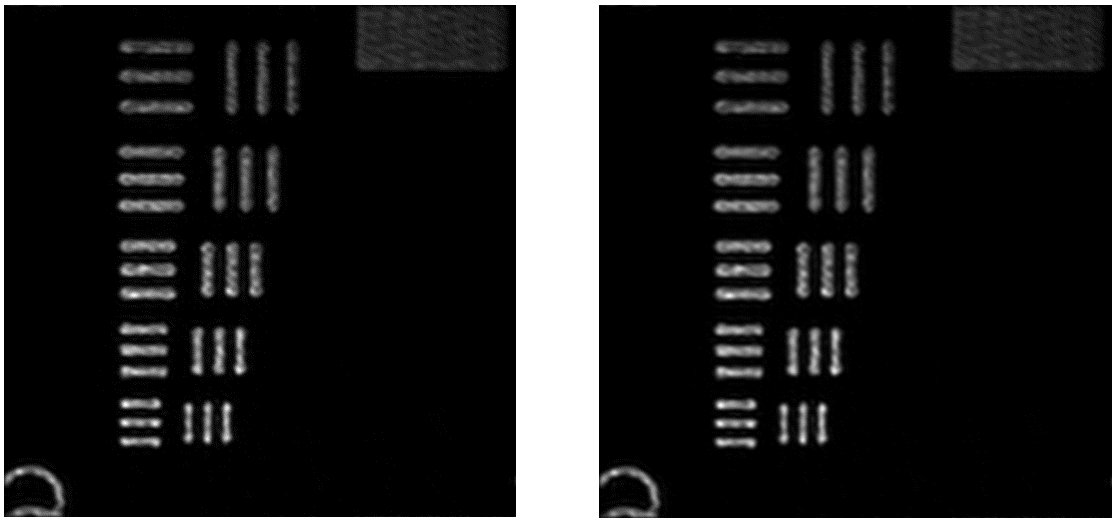


(d) Hologram 4 未加入浮水印還原後之振幅影像



(e) Hologram 4 未加入浮水印還原後之相位展開影像

圖 3.6 各張全像圖未加入浮水印還原後的影像



(a) (b)

圖 3.7 Hologram 1 嵌入浮水印後還原出的影像
(a) 用 $m=8$ 儲存後之振幅還原結果 (b) 用 $m=15$ 儲存後之振幅還原結果

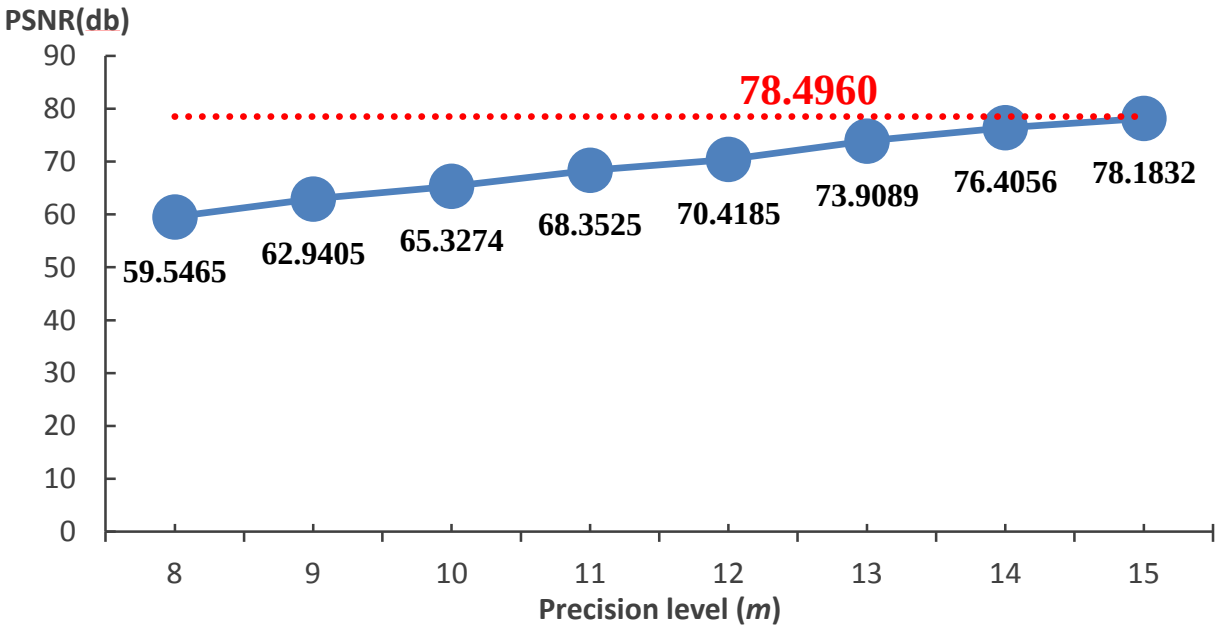
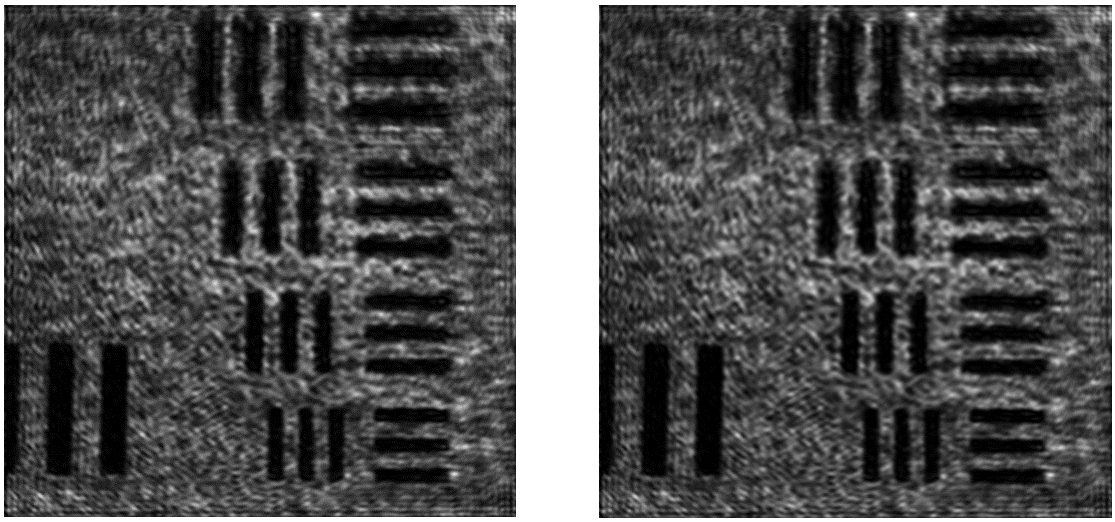


圖 3.8 Hologram 1 嵌入浮水印後用不同 m 儲存後之 PSNR



(a) (b)

圖 3.9 Hologram 2 嵌入浮水印後還原出的影像
(a) 用 $m=8$ 儲存後之振幅還原結果 (b) 用 $m=13$ 儲存後之振幅還原結果

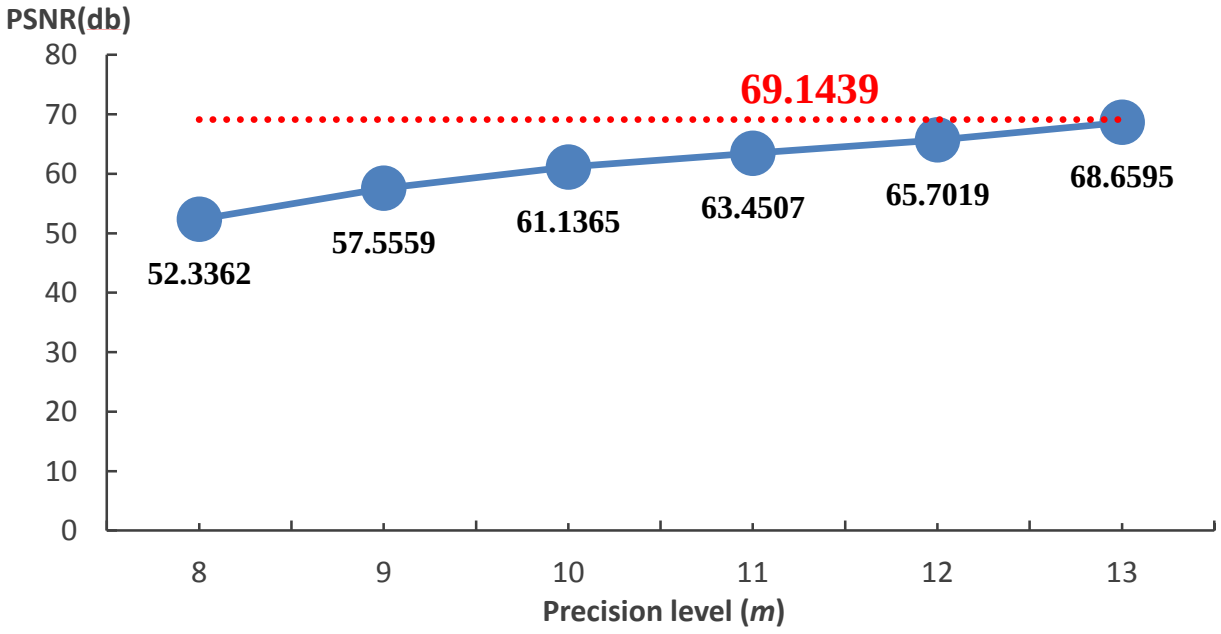
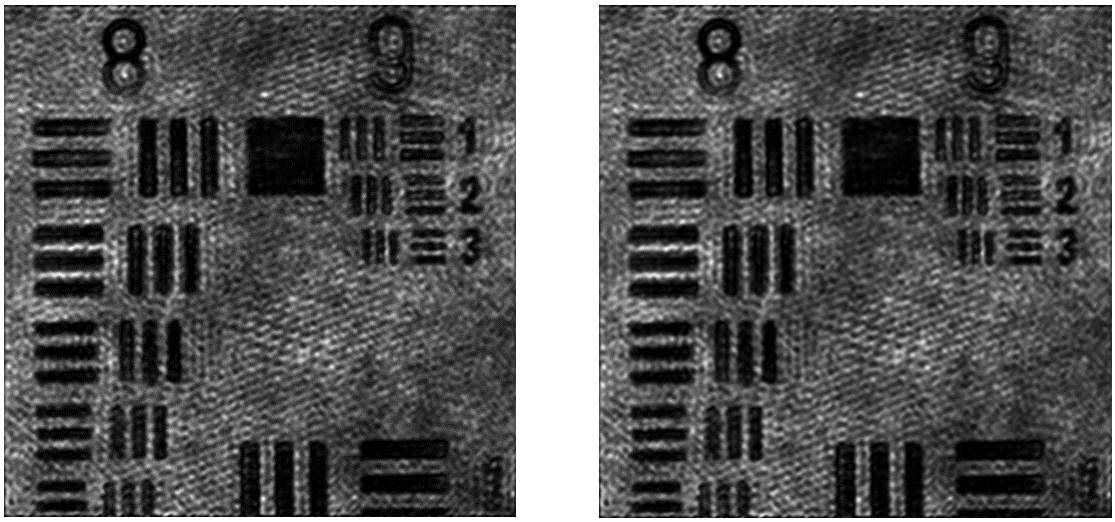


圖 3.10 Hologram 2 嵌入浮水印後用不同 m 儲存後之 PSNR



(a) (b)

圖 3.11 Hologram 3 嵌入浮水印後還原出的影像
(a) 用 $m=8$ 儲存後之振幅還原結果 (b) 用 $m=14$ 儲存後之振幅還原結果

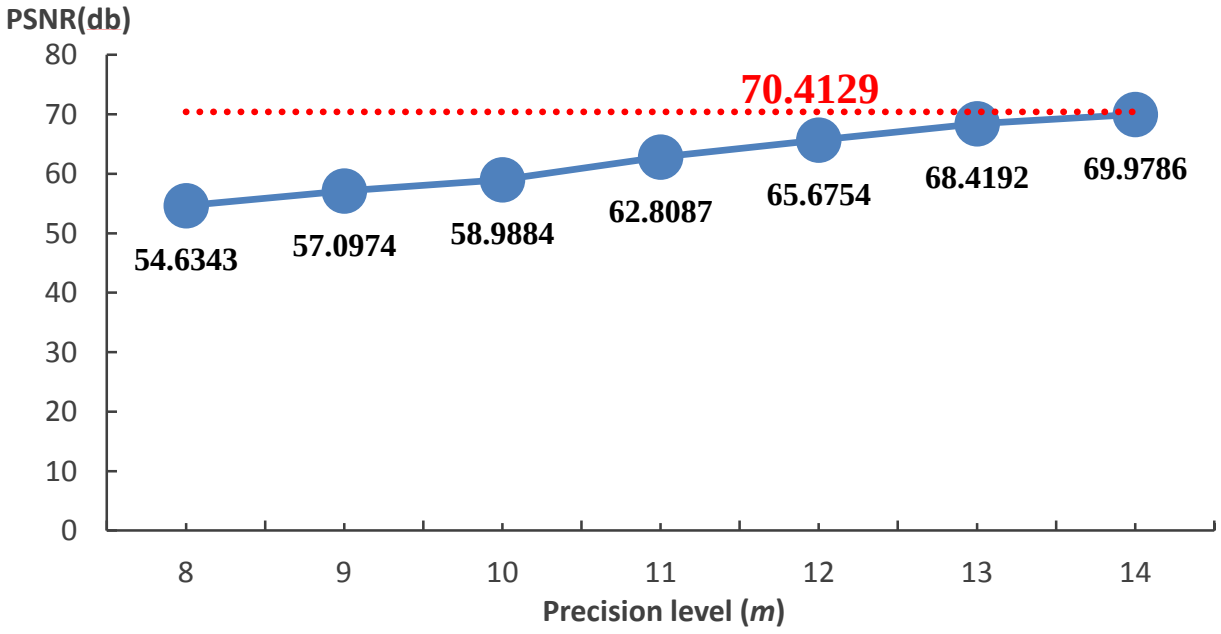
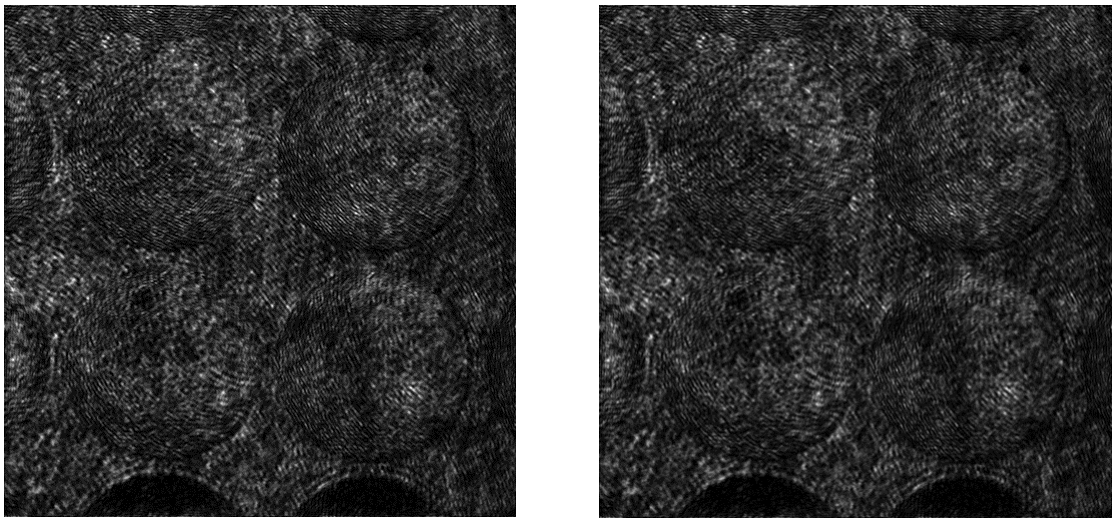


圖 3.12 Hologram 3 嵌入浮水印後用不同 m 儲存後之 PSNR



(a) (b)

圖 3.13 Hologram 4 嵌入浮水印後還原出的影像(振幅)
(a) 用 $m=8$ 儲存後之振幅還原結果 (b) 用 $m=18$ 儲存後之振幅還原結果

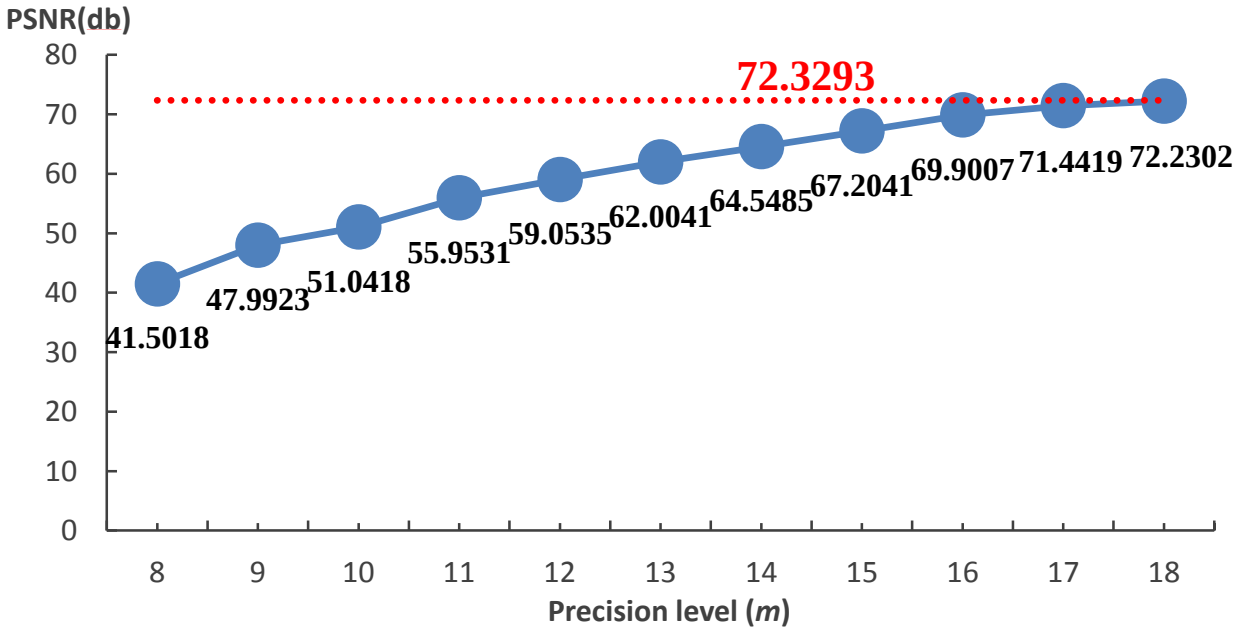
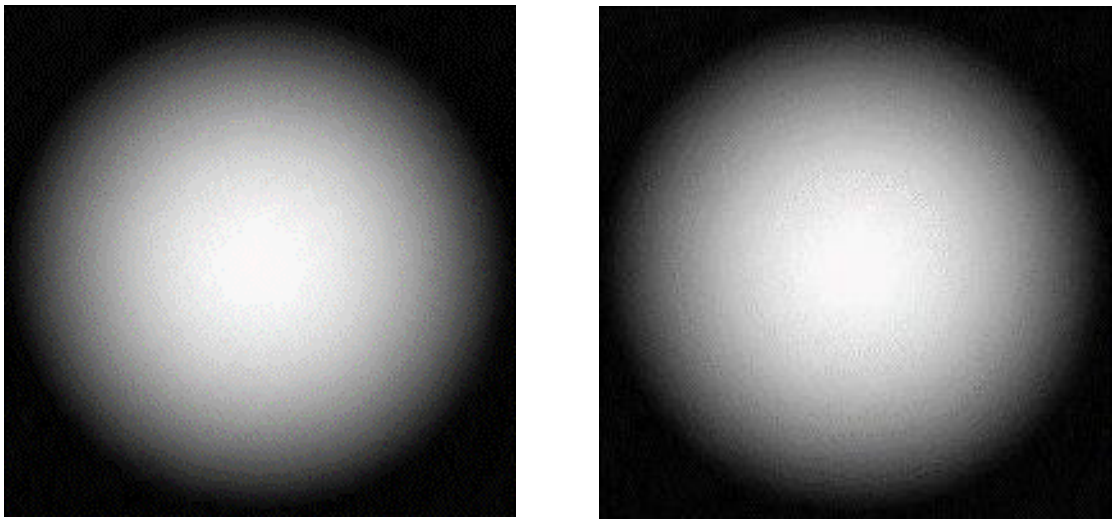


圖 3.14 Hologram 4 嵌入浮水印後用不同 m 儲存後之 PSNR(振幅)



(a) (b)

圖 3.15 Hologram 4 嵌入浮水印後還原出的影像(相位展開)
(a) 用 $m=8$ 儲存後之相位展開還原結果 (b) 用 $m=18$ 儲存後之相位展開還原結果

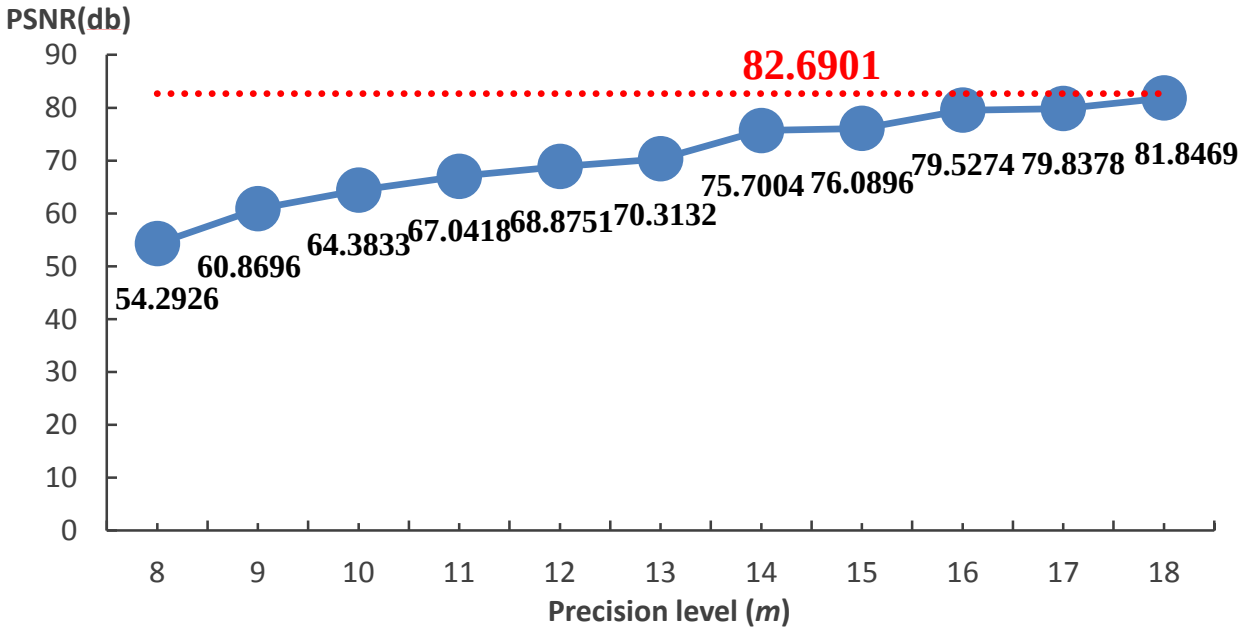


圖 3.16 Hologram 4 嵌入浮水印後用不同 m 儲存後之 PSNR(相位展開)

接下來對於 Hologram 4 還原出的相位展開影像我們做進一步的討論，與前面不一樣的地方在於前面的結果是以 PSNR 計算，這邊則以標準差(Standard deviation)來計算，標準差的公式如下所示

$s=\left(\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i-\bar{x}\right)^2\right)^{\frac{1}{2}}$，其中 $\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$，$n$ 為樣本的元素個數。

我們計算還原後的影像的方法分別為方法一及方法二，方法一如圖 3.17 所示，而方法二如圖 3.18 所示，其中方法一我們選擇平整區四個角落的原因為我們不希望計算出的結果受到拍攝物體的影響，比如形狀、表面粗糙程度等等因素，所以我們會選擇物體以外的區域。方法二為我們想測試嵌入浮水印的動作是否會對還原出的物體影像造成破壞，所以特地選擇物體影像部分。透過以上的計算結果我們可以比較兩張相位展開影像的品質好壞，標準差越低表示成像品質越好，越高則反之。使用方法一的原始無嵌入浮水印算出的標準差為 0.0558，方法二的原始無嵌入浮水印算出的標準差為 1.0723。表 3.2 及表 3.3 為我們用不同 m 儲存且有嵌入浮水印所算出來的標準差，中間為我們用不同 m 儲存所算出的標準差，而最後一個為我們用 $m=64$，L 為 0.0001 算出的標準差。我們可以發現標準差到一定程度後就不會再下降，也可以發現我們嵌入浮水印後與原始無嵌入浮水印的結果相差甚小，表示我們嵌入浮水印並沒有對影像的重建造成很嚴重的破壞。

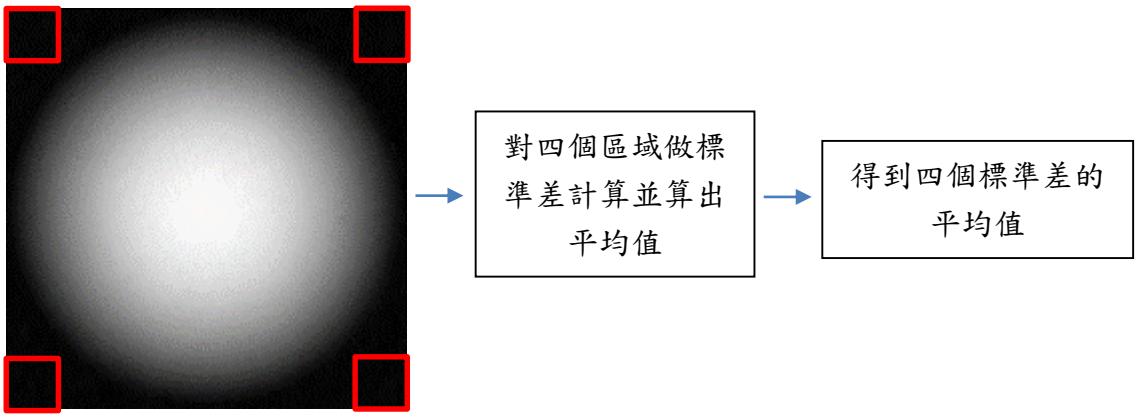
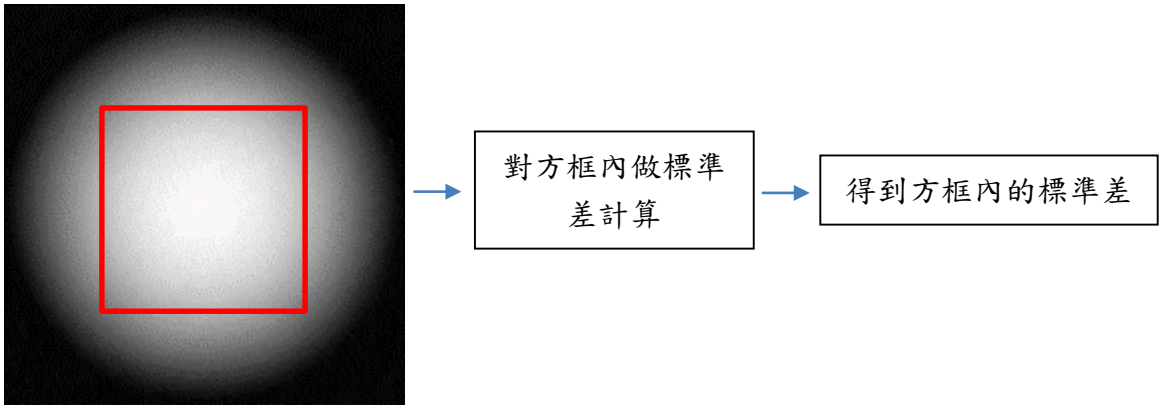


圖 3.17 方法一還原後的影像計算過程

m	Standard deviation
8	**0.0561**
9	**0.0557**
10	**0.0559**
11	**0.0558**
12	**0.0558**
13	**0.0558**
14	**0.0558**
15	**0.0558**
16	**0.0558**
17	**0.0558**
18	**0.0558**
64	**0.0558**

表 3.2 方法一用不同 *m* 儲存且有嵌入浮水印算出的標準差

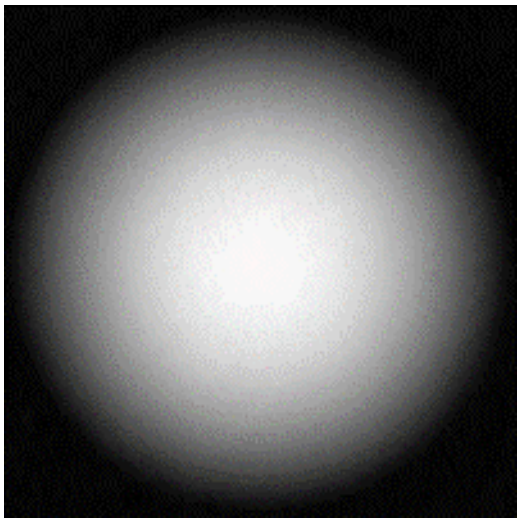


取出方框內的相位分布，如上圖方框所示

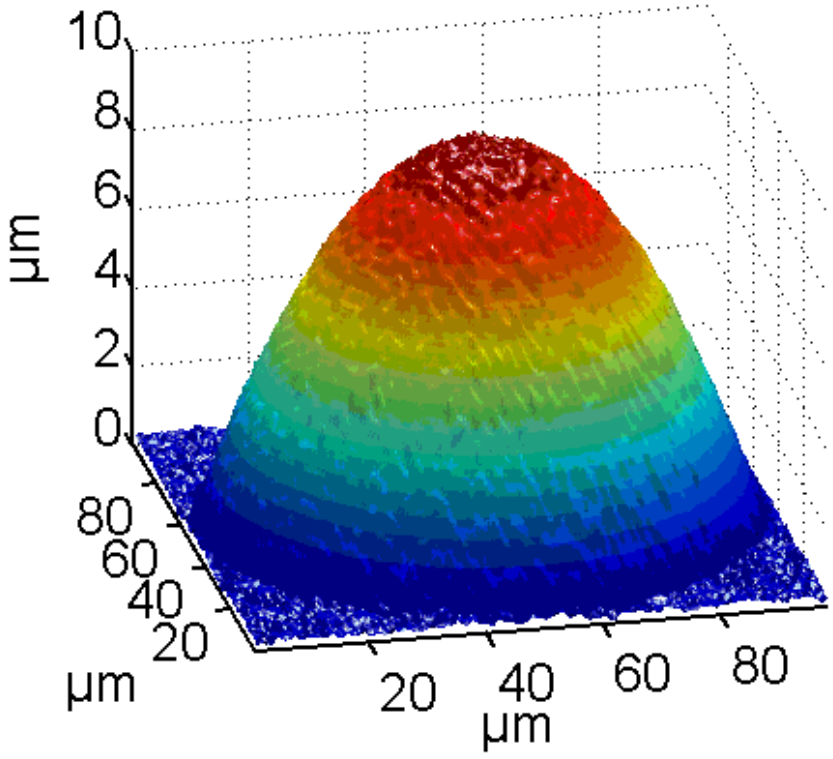
圖 3.18 方法二還原後的影像計算過程

m	Standard deviation
8	**1.0723**
9	**1.0724**
10	**1.0723**
11	**1.0723**
12	**1.0723**
13	**1.0723**
14	**1.0723**
15	**1.0723**
16	**1.0723**
17	**1.0723**
18	**1.0723**
64	**1.0723**

表 3.3 方法二用不同 *m* 儲存且有嵌入浮水印算出的標準差

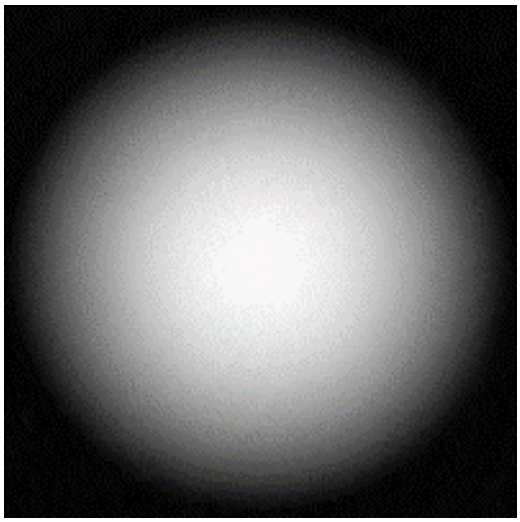


(a) 無嵌入浮水印之二維相位展開還原結果

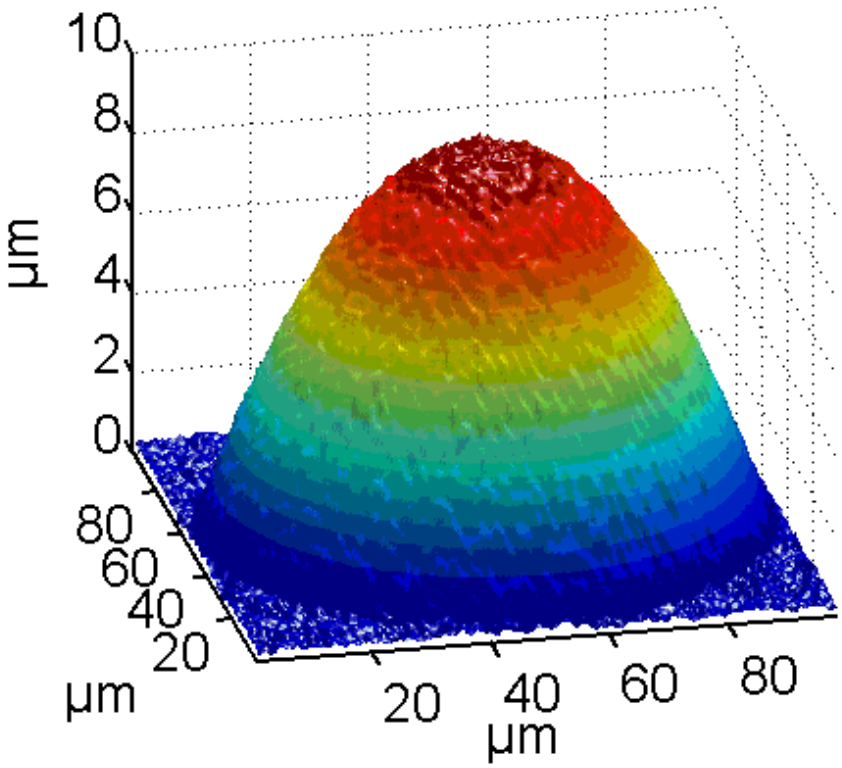


(b) 無嵌入浮水印之三維相位展開還原結果

圖 3.19　Hologram 4 無嵌入浮水印還原出的相位展開影像



(a) 用 $m=8$ 儲存且嵌入浮水印之二維相位展開還原結果



(b) 用 $m=8$ 儲存且嵌入浮水印之三維相位展開還原結果

圖 3.20 Hologram 4 嵌入浮水印後還原出的相位展開影像

3.2.3 模擬情境一

本節主要模擬使用者透過無線網路傳送全像圖時,其間發生封包破壞等任何使全像圖資料被破壞的情況。當系統接收到全像圖後會先擷取出浮水印，藉由擷取出的浮水印來判斷接收到的全像圖是否有遭到破壞,如遭到嚴重破壞則直接丟棄，否則對接收到的全像圖做還原計算。這裡我們設定每張全像圖在傳輸過程中都發生封包破壞的狀況，並設定每個位元在傳送時發生被破壞的機率為萬分之一。我們從圖 3.22(a)、圖 3.23(a)、圖 3.24(a)、圖 3.25(a)所擷取出的浮水印影像(圖 3.22(b)、圖 3.23(b)、圖 3.24(b)、圖 3.25(b))判斷接收到的全像圖都遭到嚴重破壞，所以我們就直接丟棄而不會做還原的動作。之後我們使用 Hologram 1 做為我們的測試對象，其中我們用不同的 *m* 來儲存有嵌入浮水印的 Hologram 1。從圖 3.26 我們可以觀察到隨著我們用來儲存的 *m* 值增加，其偵測出的錯誤就越多，表示本浮水印法則對於無線網路傳送所造成的破壞能夠有效的偵測出來。模擬流程如圖 3.21 所表示。我們設計此模擬情境主要訴說三個此浮水印法則提供的功能:

1. 透過嵌入浮水印這個機制可以幫系統過濾因網路傳送而遭到嚴重破壞的全像圖，確保系統還原的全像圖皆為正確的全像圖。
2. 可以避免發生駭客擷取資料修改後再上傳的情形，當上述情形發生時就會破壞我們嵌入進去的浮水印，之後我們擷取出浮水印影像就能偵測出來，可以達到保護系統的效果。

3. 若我們接收到的全像圖無法擷取出浮水印，我們就可以判斷其傳送者有可能非我們授權的用戶，進而確保使用系統的用戶皆為我們所授權的使用者。

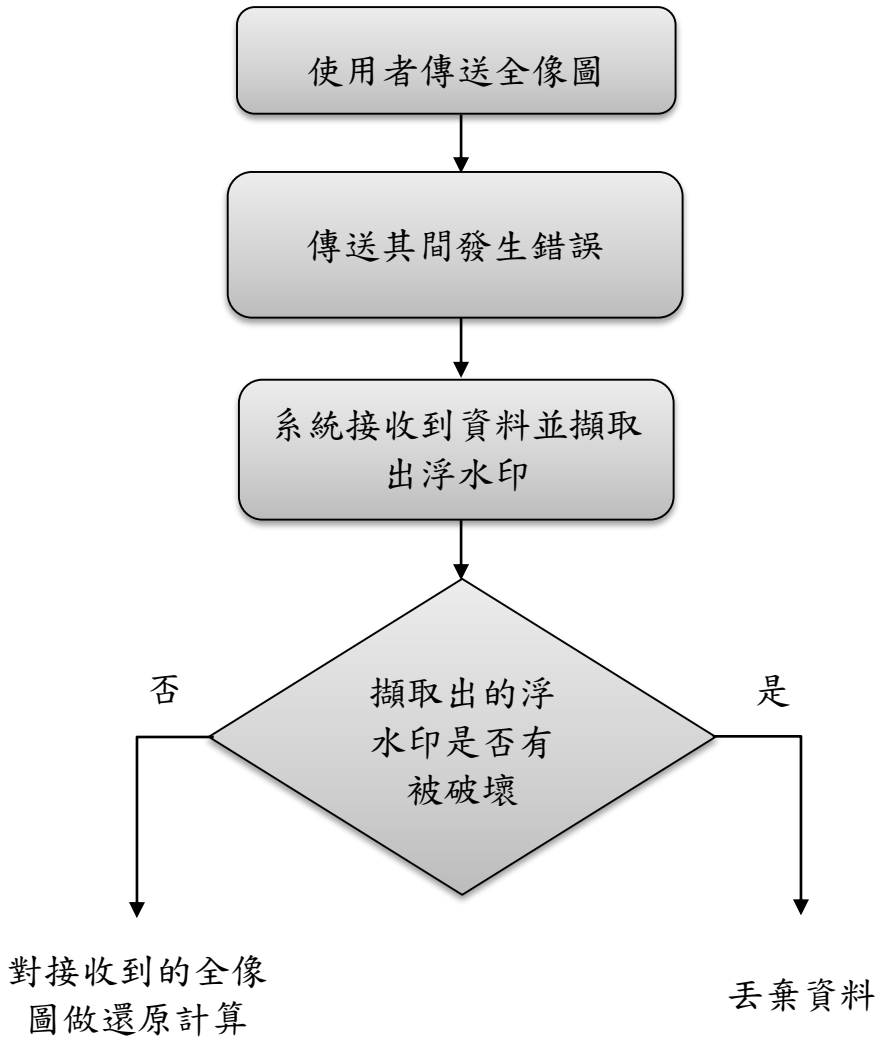
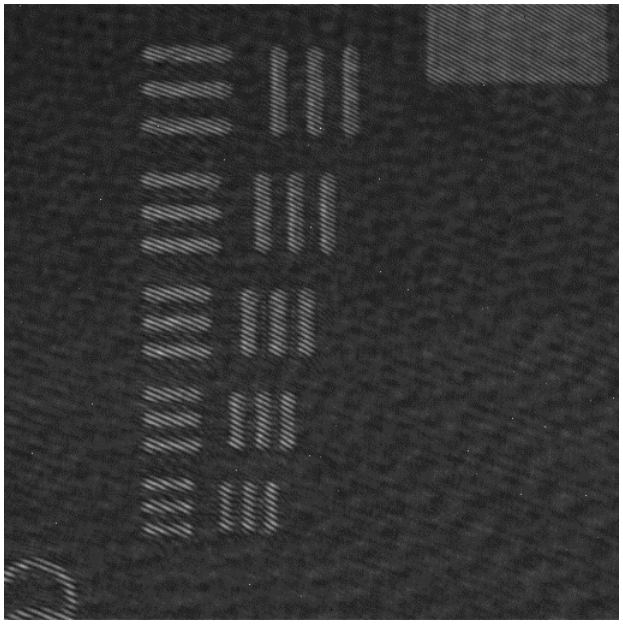


圖 3.21 情境一模擬流程圖



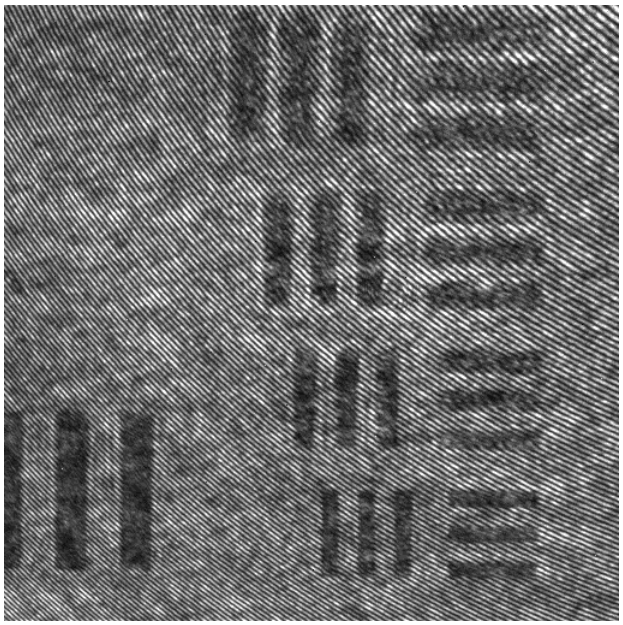
(b)

(a)

圖 3.22　Hologram 1 被破壞後的影像

(a) 遭到破壞的有嵌入浮水印但未做還原的原始影像

(b) 從圖 3.22 (a)擷取出來的浮水印



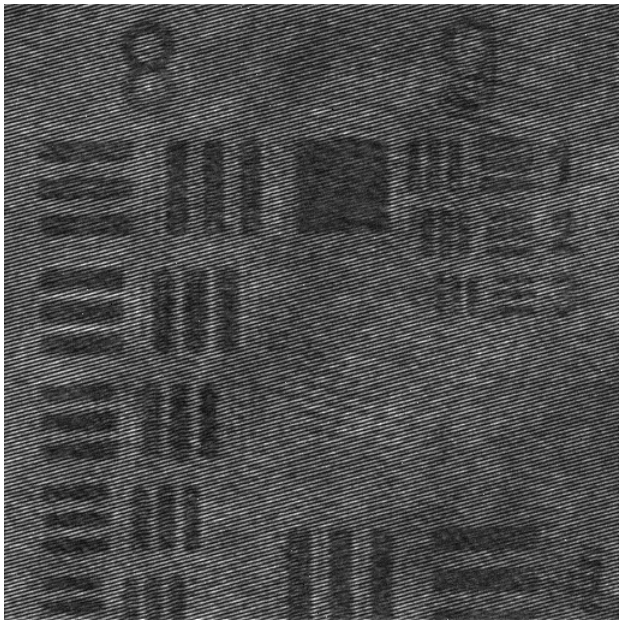
(b)

(a)

圖 3.23　Hologram 2 被破壞後的影像

(a) 遭到破壞的有嵌入浮水印但未做還原的原始影像

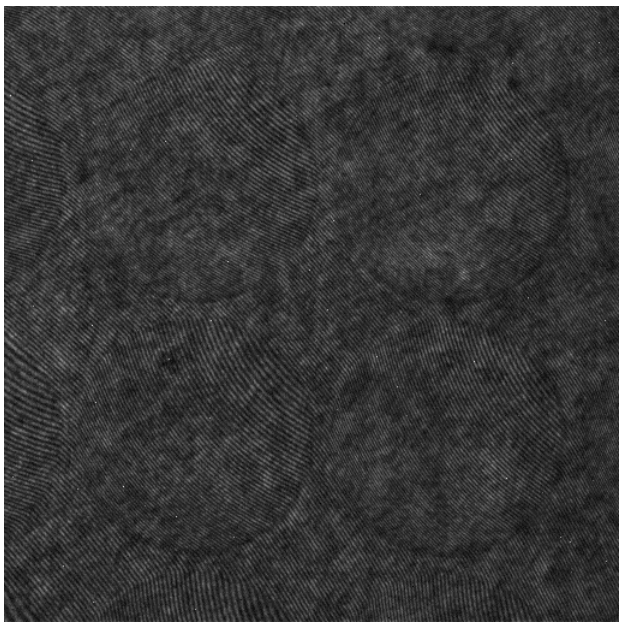
(b) 從圖 3.23 (a)擷取出來的浮水印



(a)

(b)

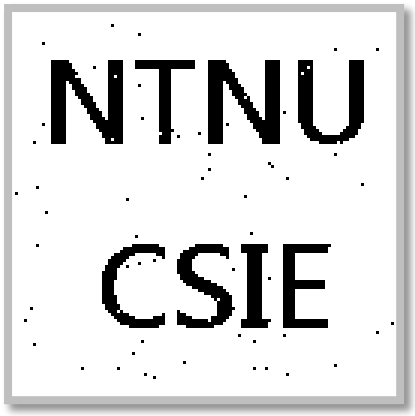
圖 3.24　Hologram 3 被破壞後的影像
(a) 遭到破壞的有嵌入浮水印但未做還原的原始影像
(b) 從圖 3.24 (a)擷取出來的浮水印



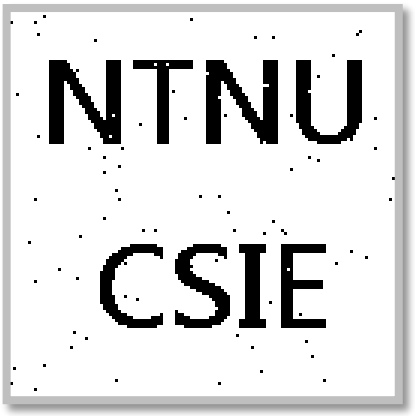
(a)

(b)

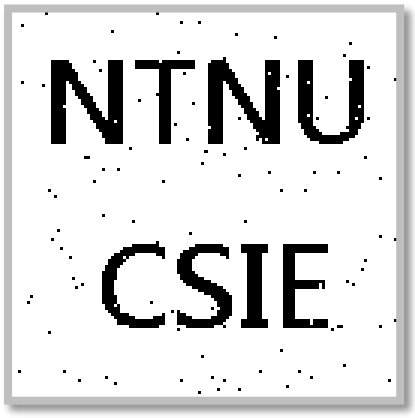
圖 3.25　Hologram 4 被破壞後的影像
(a) 遭到破壞的有嵌入浮水印但未做還原的原始影像
(b) 從圖 3.25 (a)擷取出來的浮水印



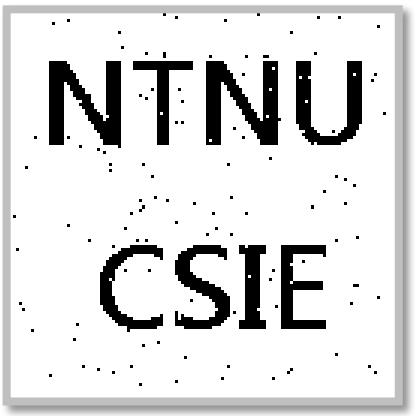
(a) 用 $m=8$ 儲存後所擷取出的浮水印影像



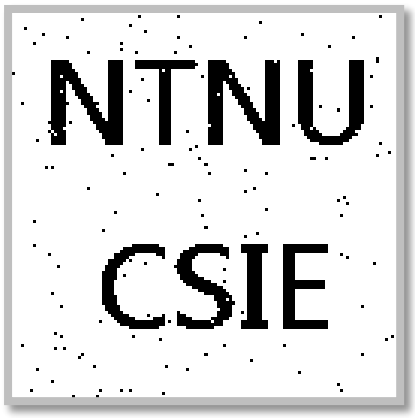
(b) 用 $m=9$ 儲存後所擷取出的浮水印影像



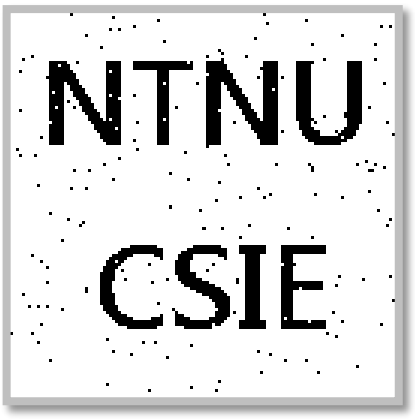
(c) 用 $m=10$ 儲存後所擷取出的浮水印影像



(d) 用 $m=11$ 儲存後所擷取出的浮水印影像



(e) 用 $m=12$ 儲存後所擷取出的浮水印影像



(f) 用 $m=13$ 儲存後所擷取出的浮水印影像

圖 3.26　Hologram 1 被破壞後擷取出的浮水印影像

3.2.4 模擬情境二

本節主要模擬全像圖儲存在資料庫中，遭到駭客篡改的情況下是否能正確偵測出來。我們以用 12 位元來儲存的全像圖來做我們測試對象，並隨機取出大約 1000 個點來做篡改。篡改的大小為±1 ~ ±5 之間的整數，因為我們是用 12 位元來儲存，所以篡改的幅度相當於除完 16 後的大小，所以實際篡改為±1/16～±5/16。我們可以從圖 3.28、圖 3.29、圖 3.30、圖 3.31 的結果觀察，隨著我們篡改的大小越大，浮水印偵測出來的錯誤也越多，但根據 eq.(27) $2^{2k-2(m-7)} \le (L/2)^2$，當我們篡改的能量不足於破壞我們嵌入的浮水印時，我們就無法透過浮水印來偵測錯誤。以下面的實驗結果而言，當我們篡改的大小為±1/16，其能量為 1/256。根據表 3.1，用 12 位元來儲存其 $L = 0.25$，套用 eq.(27) $2^{2k-2(m-7)} \le (L/2)^2$ 後我們算出浮水印所能測出的篡改能量為 1/64，所以如果全像圖被篡改的大小為±1/16 時我們是無法偵測出來的，但其篡改對全像圖的破壞程度極小，透過對其做 PSNR 計算，我們得知該全像圖還原後出來的結果也與未篡改的全像圖還原後的結果差不多，所以我們可以忽略掉這些過小的破壞，可是當篡改的大小為±2/16 時，其能量為 1/64，我們就能偵測出些許的破壞，到±5/16 時我們就可以偵測出大量的錯誤。以下的實驗結果也呈現出隨著篡改大小增大，浮水印的破壞程度就越嚴重，表示擷取出的浮水印結果符合我們的預期。之後我們用以 12 位元來儲存的 Hologram 1 做為我們的測試對象，並隨機取出大約 10000 個點來做篡改。篡改大小為±1～±5 之間的整數，因為我們是用 12 位元來儲存，所以篡改的幅度相當於

除完 16 後的大小，所以實際篡改為±1/16 ~ ±5/16。我們可以從圖 3.32 的結果觀察，在篡改的大小為±1/16 我們也能夠偵測出來，並且隨著我們篡改的大小越大，浮水印偵測出來的錯誤也越多，其中我們的篡改是改在介於 0 ~ 255 之間的像素值上，也表示我們的浮水印法則對於很小的篡改也能夠偵測出來。情境二模擬的流程如圖 3.27 所示。透過這個模擬情境我們可以知道此浮水印法則提供以下功能:

- 即使是儲存在我們資料庫中的全像圖也有可能遭到駭客入侵而被篡改，為了保護系統不要還原這些全像圖，透過擷取浮水印就可避免發生這類情形。

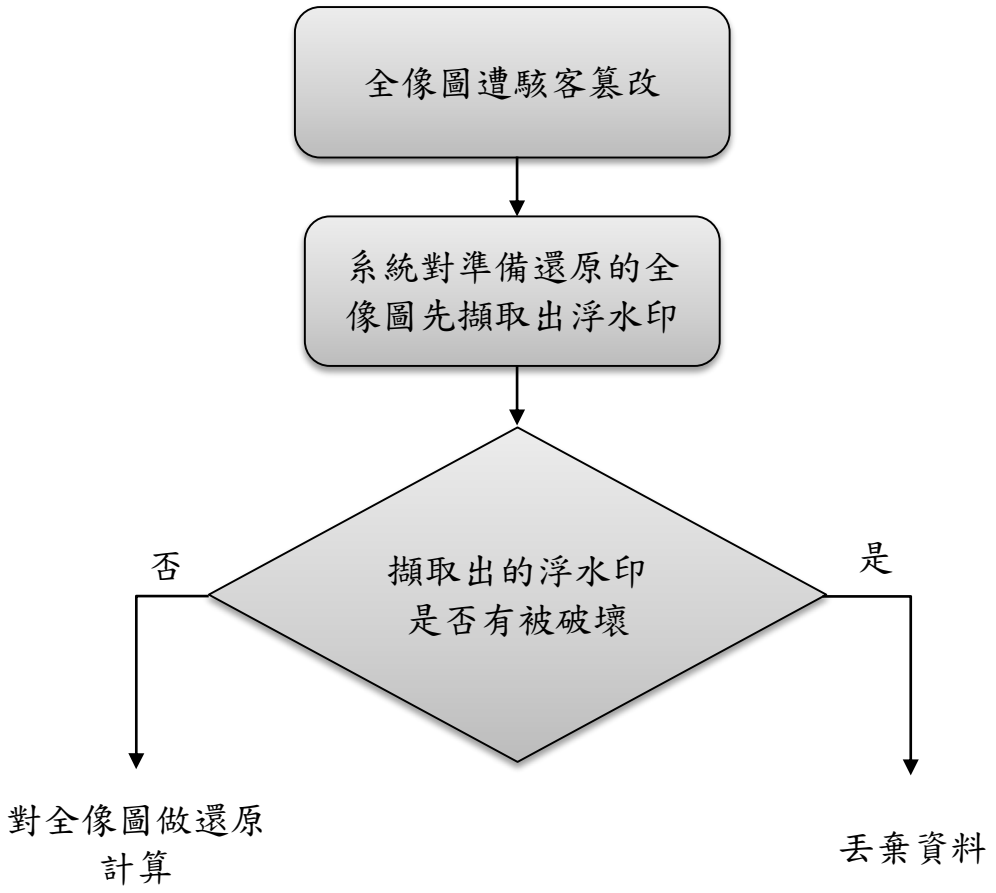


圖 3.27 情境二模擬流程圖

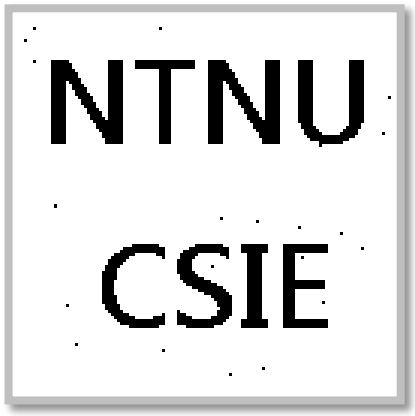
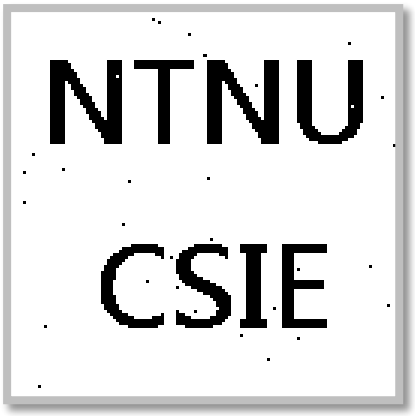
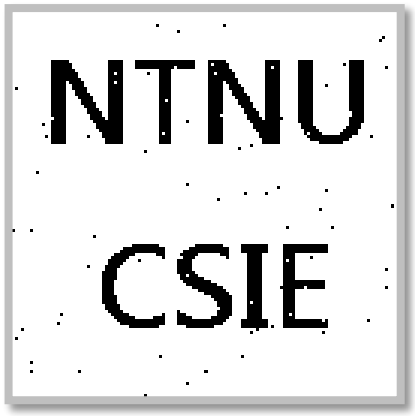
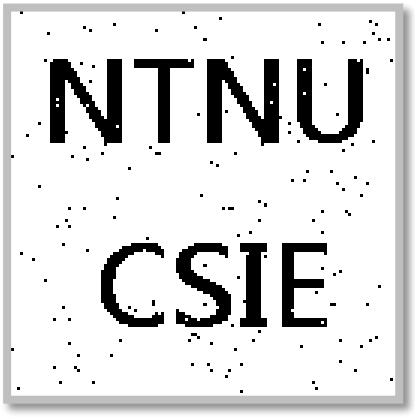
<table>
<tr><td>±1/16
  </td><td>±2/16
  </td></tr>
<tr><td>±3/16
  </td><td>±4/16
  </td></tr>
<tr><td>±5/16
  </td><td></td></tr>
</table>

圖 3.28 Hologram 1 被不同程度的篡改大小破壞後所擷取出的浮水印
(1000 個點)

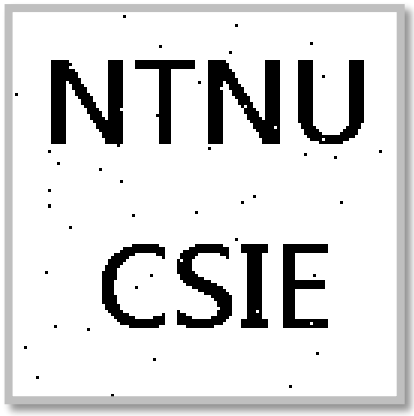
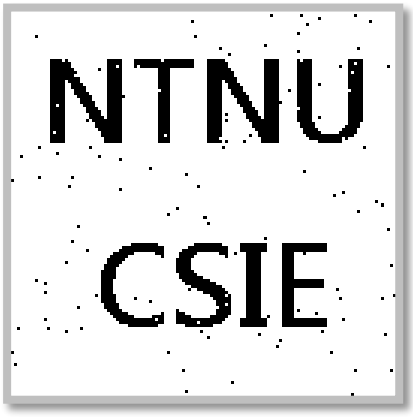
<table>
<tr><td>±1/16
  </td><td>±2/16
  </td></tr>
<tr><td>±3/16
  </td><td>±4/16
  </td></tr>
<tr><td>±5/16
  </td><td></td></tr>
</table>

圖 3.29 Hologram 2 被不同程度的篡改大小破壞後所擷取出的浮水印
(1000 個點)

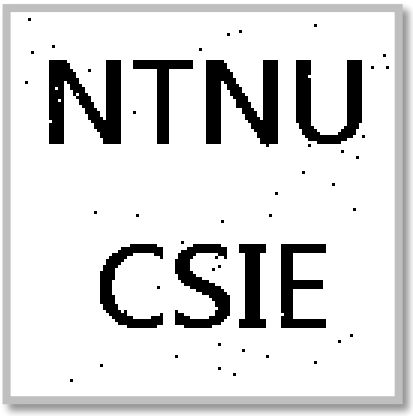
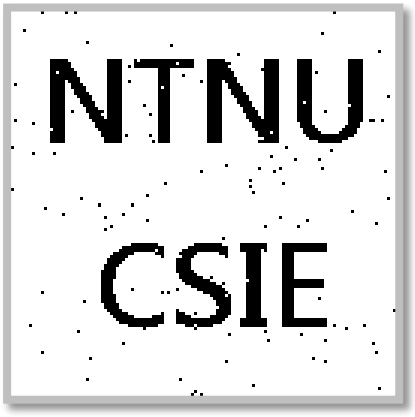
<table>
<tr><td>±1/16
  </td><td>±2/16
  </td></tr>
<tr><td>±3/16
  </td><td>±4/16
  </td></tr>
<tr><td>±5/16
  </td><td></td></tr>
</table>

圖 3.30 Hologram 3 被不同程度的篡改大小破壞後所擷取出的浮水印
(1000 個點)

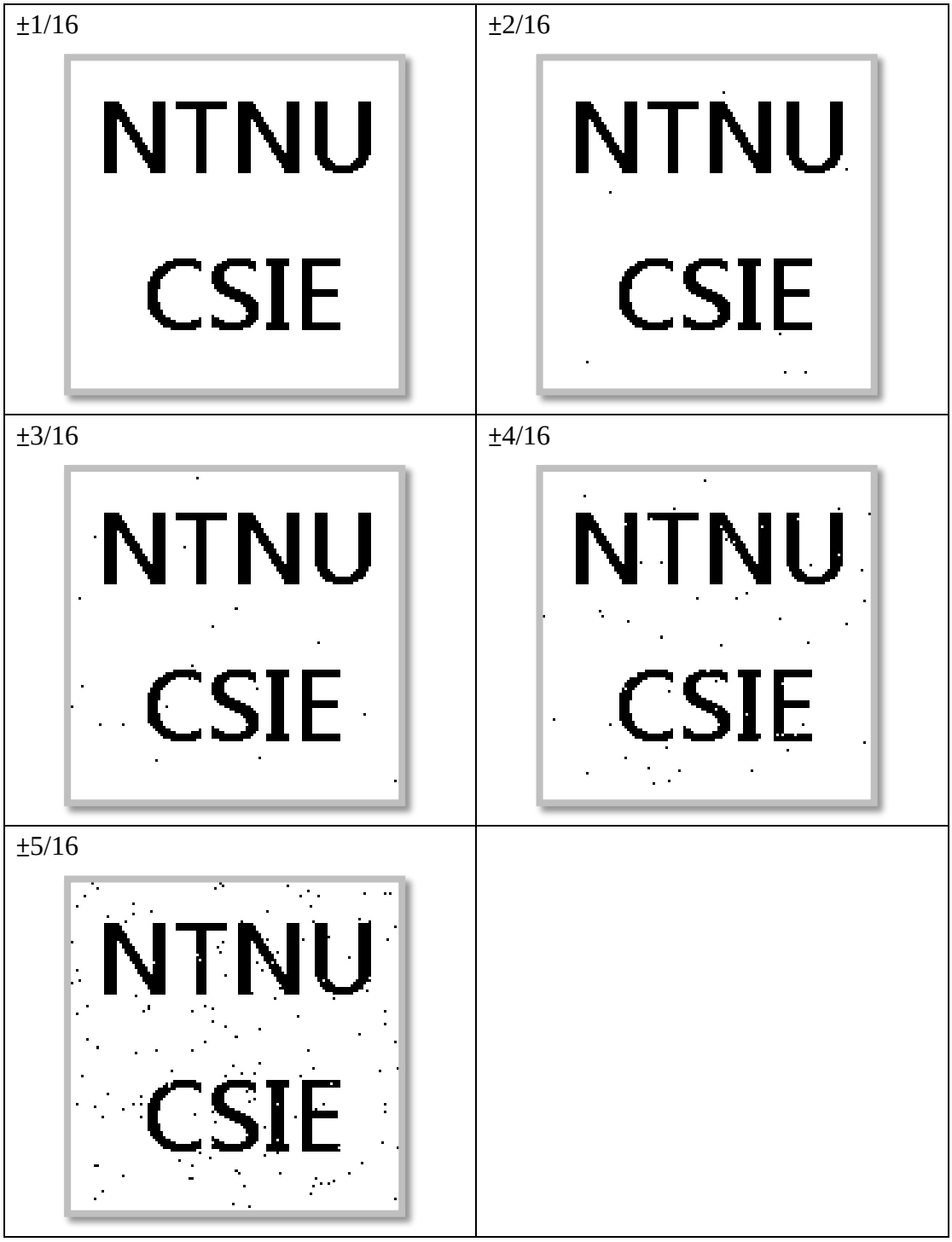


圖 3.31 Hologram 4 被不同程度的篡改大小破壞後所擷取出的浮水印
(1000 個點)

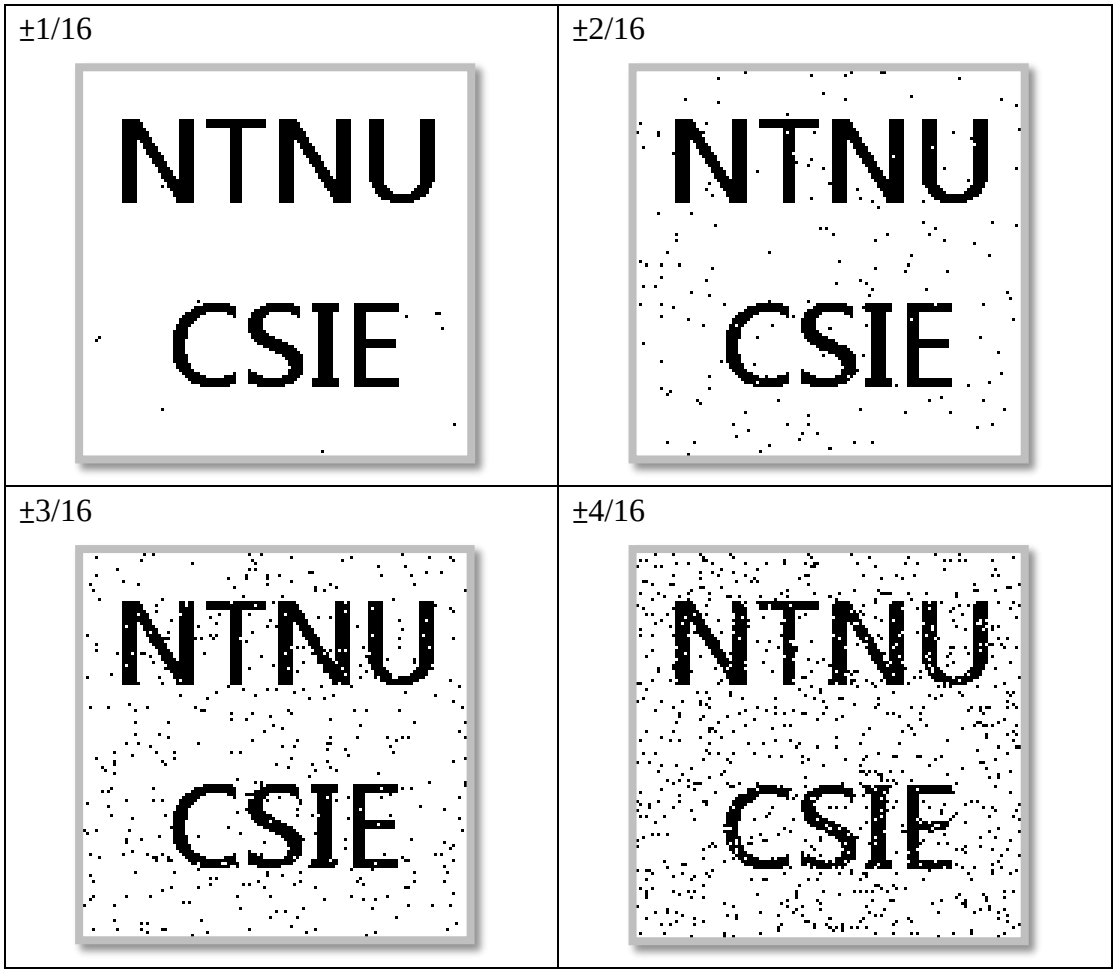


圖 3.32　Hologram 1 被不同程度的篡改大小破壞後所擷取出的浮水印 (10000 個點)

第四章　結論

本論文中所呈現的是適用於數位全像顯微鏡系統影像認證的脆弱型浮水印法則，並實際測試本法則對全像圖的破壞程度，透過數學分析我們可以根據不同的全像圖來調整我們嵌入浮水印的強度，或是使用更多的位元來保存我們的資料，藉此來降低浮水印對全像圖的破壞，並提高浮水印對篡改或破壞的敏感度，進而達到保護全像圖及提升全像圖還原系統端效能之目的。

此外，此法則並無特別複雜的數學計算，所以計算複雜度低，這點使得此法則可以達到即時嵌入及擷取之條件，並且對硬體資源要求不高，對於使用行動裝置的客戶來說，可以免除因裝置的硬體而造成的限制。

綜觀以上各個敘述，本論文所提出的浮水印法則不僅可實現於有網路傳輸需求的全像圖還原系統的保全方面，並且能實際應用於有系統授權之使用者的裝置上。

參考著作

[1] G. Sivathanu, C. P. Wright, and E. Zadok, Ensuring Data Integrity in Storage: Techniques and Applications, Proc. International Workshop on Storage Security and Survivabil-ity, pp.26-36, 2005.

[2] A. Perrig, B. Przydatek, and D. Song, SIA: Secure Information Aggregation in Sensor Networks. J. Comput. Secur. pp. 69-102, 2007.

[3] M. Wu and B. Liu, Watermarking for image authentication, in Proc. IEEE Int. Conf. Image Processing, pp. 437441, 1998.

[4] H. T. Chang and C. L. Tsan, Image watermarking by use of digital holography embedded in the discrete-cosine-transform domain, Applied Optics, pp.6211-6219, 2005.

[5] J. Li, Robust image watermarking scheme against geometric attacks using a computer-generated hologram, Applied Optics, pp.6302-6312, 2010.

[6] N. K. Nishchal, T. Pitkaaho, and T.J., Naughton, Digital Fresnel Hologram Watermarking, Proc. Euro-American Workshop on Information Optics, 2010.

[7] C. I. Podilchuk and E. J. Delp, Digital Watermarking: Algorithms and Applications, IEEE Signal Processing Magazine, pp.33-46, 2001.

[8] C.H. Huang, J.S. Pan, and H.M. Hang, Watermarking Based on Spatial Domain, Chapter 5 of the Book Intelligent Watermarking Techniques, Editors, J.S. Pan, H.C. Huang and L.C.Jain, pp.135-146, World Scientific Publishing, Singapore, 2004.

[9] I. Kamel and H. Juma, A Lightweight Data Integrity Scheme for Sensor Networks, Sensors, pp.4118-4136, 2011.

[10] M. Chen, Y. He, and R. L. Lagendijk, A Fragile Watermark Error Detection Scheme for Wireless Video Communications, IEEE Trans. Multimedia, Vol. 7, pp.201-211, 2005.

[11] C. J. Cheng, L. C. Lin and W. T. Dai, Construction and detection of digital holographic watermarks, Optics Communications, Vol. 248, pp.105-116, 2005.