Reversible Watermark Using the Difference Expansion of A Generalized Integer Transform

Adnan M. Alattar, *Member, IEEE*

***Abstract*— A reversible watermarking algorithm with very high data hiding capacity has been developed for color images. The algorithm allows the watermarking process to be reversed, which restores the exact original image. The algorithm hides several bits in the difference expansion of vectors of adjacent pixels. The required general reversible integer transform and the necessary conditions to avoid underflow and overflow are derived for any vector of arbitrary length. Also, the potential payload size that can be embedded into a host image is discussed, and a feedback system for controlling this size is developed. In addition, to maximize the amount of data that can be hidden into an image, the embedding algorithm can be applied recursively an across the color components. Simulation results using spatial triplets, spatial quads, cross-color triplets, and cross-color quads are presented and compared with the existing reversible watermarking algorithms. These results indicate that the spatial quad-based algorithm allows for hiding the largest payload at the highest signal-to-noise ratio (SNR).**

***Index Terms*—Difference Expansion, Data Embedding, Reversible Watermark, Steganography.**

I. Introduction

IN digital watermarking or steganography, a hardly noticeable noise-like signal can be embedded into a digital medium, such as an image, audio, or video data, to protect it from illicit use and alteration, to authenticate its content and origin, or to enhance its value and enrich its information content [1]. Unlike metadata, which is often appended to the digital file, a watermark is bound into the fabric of the media and cannot be removed or destroyed easily.

The watermarking process usually introduces irreversible degradation of the original medium. Although this degradation is slight, it may not be acceptable to some applications, such as military uses, medical uses, and multimedia archiving of valuable original works. However, these applications may tolerate the addition of such noise if the watermark can be removed after decoding. This removal restores the original medium without any reference to information beyond what is available in the watermarked medium itself.

The amount of tolerance to the watermark varies from one application to another. Although some applications desire high signal-to-noise ratio (SNR), many others accept low SNR. For example, Mintzer, *et al.* [2], used completely visible but removable patterns (low SNR) with their digital library application to promote image sale on the Internet. The user can download a free marked image as a “teaser”, but he or she must purchase a “vaccine” program to restore a high quality image from the teaser image.

If the embedding algorithm and all embedding parameters are available to the reader, it might be possible to calculate the watermark and subtract it from the marked medium to recover the original medium, once the watermark is detected and the payload is read. Honsinger, *et al.* [3], used this approach and an invertible addition to devise a low capacity reversible watermark algorithm. He restricted the embedder to be additive and non-adaptive and included the watermarking parameters in the payload. Unfortunately, watermarking algorithms are often highly adaptive and employ some kind of non-linearity to optimize their performance. Macq [4] extended Honsinger’s technique to the Patchwork algorithm proposed by Bender, *et al.*[5], for a robust watermark. Both of Macq and Honsinger’s algorithms suffer from a wraparound effect that causes a salt-and-pepper visual artifact.

Vleeschouwer, *et al.* [6], used a circular interpretation of bijective transformations to embed a reversible watermark and reduce the salt-and-pepper artifact. Also, Fridrich, *et al.* [7], devised a reversible watermark that does not suffer from the salt-and-pepper artifact. That algorithm compresses one of the least significant bit planes of the host image, appends an image hash and payload, encrypts the result, and finally, replaces the original bit plane with the result. The process can be reversed to obtain the original host image. Fridrich also extended the technique to JPEG-compressed images [8] and GIF and PNG palette images [9]. The capacity of the algorithms of Vleeschouwer and Fridrich is low; it is around 100 bits/image.

Celik, *et al.* [10], enhanced Fridrich’s approach and devised a low distortion reversible watermark that is capable of embedding as high as 0.7 bits/pixel. His algorithm first quantizes each pixel by a quantizer of L step size, compresses the quantization noise and appends a payload to it, then adds an L-ary representation of the result to the quantized image. Also, Tian [11] used a difference expansion transform of a pair of pixels to devise a high capacity and low distortion reversible watermark. His algorithm divides the image into pairs of pixels then embeds one bit into the difference of the pixels of each pair from those pairs that are not expected to cause an overflow or underflow. The location map that

Manuscript received March 3, 2003.

A. Alattar is with Digimarc Corporation, Tualatin, OR 97062 USA (phone: 503-495-4631; fax: 503-885-9880; e-mail:aalattar@digimarc.com).

indicates the modified pairs is compressed and included in the payload. Tian's algorithm is capable of embedding as high as 2 bits/pixel.

Kalker, *et al.*[12], derived capacity bounds for a reversible watermark and proposed using error correction codes to make the reversible watermark robust to ordinary processing [13]. He presented toy examples to illustrate the idea of robust reversible watermarks.

In this paper, we extend Tian's algorithm using difference expansion of vectors, instead of pairs, to increase the hiding ability and the computation efficiency of the algorithm. This approach allows the algorithm to embed several bits in every vector in a single pass through the image data.

In the next section, we define the generalized integer difference expansion transform. In Section III, we describe the proposed embedding and recovery algorithms for a reversible watermark. In Section IV, we discuss the size of the payload that can be embedded using the proposed algorithm. In Section V, we present an algorithm to adjust the internal thresholds of the algorithm to embed the desired payload size into a host image. In Section VI, we discuss the idea of recursive embedding and embedding across color components to hide more data into a host image. In Section VII, we present simulation results of the proposed algorithm using triplets and quad vectors. And finally, in the last section, we summarize our conclusions.

II. Generalized Difference Expansion

Vector: For the purpose of this paper, the vector $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$ is formed from N pixel values chosen from N different locations within the same color component according to a predetermined order. This order may serve as a security key. The simplest way to form this vector is to consider every set of $a \times b$ adjacent pixel values as shown in Fig. 1 as a vector. If w and h are the width and the height of the host image, respectively, then $1 \le a \le h$, $1 \le b \le w$, and $a + b \ne 2$.

For simplicity, we require that each color component be treated independently and, hence, have its own set of vectors. Also, we require that vectors do not overlap each other; i.e., each pixel exists in only one vector. These requirements may be removed at the expense of complicating the watermarking algorithm due to the extra caution required to determine the processing order of the overlapped vectors.

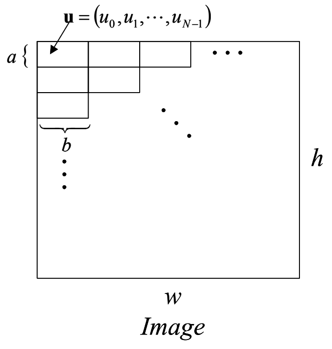

Fig. 1. Vector configuration in an image.

Reversible Integer Transform: The forward difference expansion transform, $f(\cdot)$, for the vector $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$ is defined as:

$$v_0 = \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \qquad (1)$$
$$v_1 = u_1 - u_0$$
$$\vdots$$
$$v_{N-1} = u_{N-1} - u_0,$$

where $\lfloor \cdot \rfloor$ is the least nearest integer, and a_i is a constant integer. Obviously, v_0 is the weighted average of the entities of the vector $\mathbf{u}$, while $v_1, v_2, \cdots, v_{N-1}$ are the differences between $u_1, u_2, \cdots, u_{N-1}$ and u_0, respectively.

The inverse difference expansion transform, $f^{-1}(\cdot)$, for the transformed vector, $\mathbf{v} = (v_0, v_1, \cdots, v_{N-1})$ is defined as:

$$u_0 = v_0 - \left\lfloor \frac{\sum_{i=1}^{N-1} a_i v_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \qquad (2)$$
$$u_1 = v_1 + u_0$$
$$\vdots$$
$$u_{N-1} = v_{N-1} + u_0.$$

Proof: To prove that equation (2) is the inverse of equation (1), one can substitute v_0, v_1, …, v_{N-1} from equation (1) into u_0 of equation (2). This gives

$$u_0 = \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor - \left\lfloor \frac{\sum_{i=1}^{N-1} a_i (u_i - u_0)}{\sum_{i=0}^{N-1} a_i} \right\rfloor$$

$$\begin{aligned} u_0 &= \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor - \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} - \frac{\sum_{i=0}^{N-1} a_i}{\sum_{i=0}^{N-1} a_i} u_0 \right\rfloor \\ &= \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor - \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor + u_0 = u_0. \end{aligned} \tag{3}$$

Now, the reversibility concerning u_1, u_2,..., u_{N-1} can be proven by simple mathematical manipulation of v_1, v_2,..., v_{N-1} in equation (1).

Definition 1: The vector $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$ is said to be expandable if, for all b_1, b_2,..., $b_{N-1} \in \{0,1\}$, $\mathbf{v} = f(\mathbf{u})$ can be modified to produce $\tilde{\mathbf{v}} = (v_0, \tilde{v}_1, \cdots, \tilde{v}_{N-1})$ according to equation (4) below without causing overflow or underflow in $\tilde{\mathbf{u}} = f^{-1}(\tilde{\mathbf{v}})$.

$$\begin{aligned} v_0 &= \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \\ \tilde{v}_1 &= 2 \times v_1 + b_1 \\ &\vdots \\ \tilde{v}_{N-1} &= 2 \times v_{N-1} + b_{N-1} \end{aligned} \tag{4}$$

It should be noticed here that the above modification only changes the differences of the vector $\mathbf{u}$. Each of $\tilde{v}_1$, $\tilde{v}_2$,..., $\tilde{v}_{N-1}$ is a one-bit left-shifted version of the original value v_1, v_2 ..., v_{N-1}, respectively, but potentially with a different least significant bit (LSB). The weighted average v_0 of $\mathbf{u}$ remains unchanged.

To prevent overflow and underflow, one must make sure that the following conditions hold:

$$\begin{aligned} &0 \le v_0 - \left\lfloor \frac{\sum_{i=1}^{N-1} a_i \tilde{v}_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \le 255 \\ &0 \le \tilde{v}_1 + v_0 - \left\lfloor \frac{\sum_{i=1}^{N-1} a_i \tilde{v}_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \le 255 \\ &\vdots \\ &0 \le \tilde{v}_{N-1} + v_0 - \left\lfloor \frac{\sum_{i=1}^{N-1} a_i \tilde{v}_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \le 255. \end{aligned} \tag{5}$$

Definition 2: The vector $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$ is said to be changeable if, for all values of b_1, b_2,..., $b_{N-1} \in \{0,1\}$, $\mathbf{v} = f(\mathbf{u})$ can be modified to produce $\tilde{\mathbf{v}} = (v_0, \tilde{v}_1, \cdots, \tilde{v}_{N-1})$ according to equation (6) below without causing overflow or underflow in $\tilde{\mathbf{u}} = f^{-1}(\tilde{\mathbf{v}})$ by satisfying equation (5).

$$\begin{aligned} v_0 &= \left\lfloor \frac{\sum_{i=0}^{N-1} a_i u_i}{\sum_{i=0}^{N-1} a_i} \right\rfloor \\ \tilde{v}_1 &= 2 \times \left\lfloor \frac{v_1}{2} \right\rfloor + b_1 \\ &\vdots \\ \tilde{v}_{N-1} &= 2 \times \left\lfloor \frac{v_{N-1}}{2} \right\rfloor + b_{N-1} \end{aligned} \tag{6}$$

Again, the above modification only changes the differences of the vector $\mathbf{u}$. $\tilde{v}_1$, $\tilde{v}_2$,..., $\tilde{v}_{N-1}$ are the same as the original v_1, v_2,..., v_{N-1}, but potentially with different LSBs. The weighted average v_0 of $\mathbf{u}$ remains unchanged.

Notice that a changeable vector remains changeable even after changing the LSBs of its v_1, v_2,..., v_{N-1}. Also, from definitions 1 and 2, it can be observed that an expandable vector is also changeable.

It should be mentioned here that the difference expansion transform used by Tian [11] is a special case of the above transform and can be obtained by setting $N = 2$ and $a_0 = a_1 = \cdots = a_{N-1} = 1$.

III. Algorithm for Reversible Watermark

Let $I(i, j, k)$ be an RGB image, and assume that:

1. The pixel values in the red component, $I(i, j, 0)$, are

arranged into the set of $1\times N$ vectors $U_R = \{\mathbf{u}_l^R, l = 1\cdots L\}$ using the security key K_R.

2. The pixel values in the green component, $I(i,j,1)$, are arranged into the set of $1\times N$ vectors $U_G = \{\mathbf{u}_n^G, n = 1\cdots N\}$ using the security key K_G.
3. The pixel values in the blue component, $I(i,j,2)$, are arranged into the set of $1\times N$ vectors $U_B = \{\mathbf{u}_p^B, p = 1\cdots P\}$ using the security key K_B.

Although it is not necessary, usually all color components in the image have the same dimensions and are processed using the same difference transform. This makes the number of vectors in the sets U_R, U_G, and U_B be the same (i.e., $L = N = P$). Also, let the set $U = \{\mathbf{u}_r, r = 1\cdots R\}$ represent any of the above set of vectors U_R, U_G, and U_B, and let K represent its associated security key. In addition, let $V = \{\mathbf{v}_r, r = 1\cdots R\}$ be the transformation of V under the difference expansion transform $f(\cdot)$ (i.e., $V = f(U)$ and $U = f^{-1}(V)$). Lastly, let $\mathbf{u}_r = (u_0, u_1, \cdots, u_{N-1})$ and its difference expansion transform be $\mathbf{v}_r = (v_0, v_1, \cdots, v_{N-1})$.

The vectors in U now can be classified into three groups according to the definitions given in Section II, above. The first group, S_1, contains all expandable vectors whose $v_1 \le T_1$, $v_2 \le T_2$,..., $v_{N-1} \le T_{N-1}$, where T_1, T_2,..., T_{N-1} are predefined thresholds. The second group, S_2, contains all changeable vectors that are not in S_1. The third group, S_3, contains the rest of the vectors (not changeable). Also, let S_4 denote all changeable vectors (i.e., $S_4 = S_1 \bigcup S_2$).

Let us now identify the vectors of S_1 using a binary location map, M, whose entries are 1s and 0s, where the 1 symbol indicates the S_1 vectors, and the 0 symbol indicates the S_2 or S_3 vectors. Depending on how the vectors are formed, the location map can be 1- or 2-dimensional. For example, if vectors are formed from 2x2 adjacent pixels, the location map forms a binary image that has one-half the number of rows and one-half the number of columns as the original image. However, if a random key is used to identify the locations of the entries of each vector, then the location map is a binary stream of ones and zeros. In this case, the security key and an indexing table are needed to map the zeros and ones in this stream to the actual locations in the image. Such a table must be predefined and assumed to be known to both the embedder and the reader.

A. Embedding a Reversible Watermark

The embedding algorithm can be summarized using the following steps:

For every $U \in \{U_R, U_G, U_B\}$, do the following:

1. Form the set of vectors U from the image $I(i,j,k)$ using the security key K.
2. Calculate V using the reversible integer transform, $f(\cdot)$ (see equation (1)).
3. Use V, equations (4) and (6), and the conditions in equation (5) to divide U into the sets S_1, S_2, and S_3.
4. Form the location map, M; then compress it using a lossless compression algorithm, such as JBIG or an arithmetic compression algorithm, to produce sub-bitstream B_1. Append a unique identifier, end-of-stream (EOS), symbol to B_1 to identify the end of B_1.
5. Extract the LSBs of v_1, v_2,..., v_{N-1} of each vector in S_2. Concatenate these bits to form sub-bitstream B_2.
6. Assume the watermark to be embedded forms a sub-bitstream B_3, and concatenate sub-bitstreams B_1, B_2, and B_3 to form the bitstream B.
7. Sequence through the member vectors of S_1 and S_2 as they occur in the image and through the bits of the bitstream B in their natural order. For S_1, expand the vectors as described in equation (4). For S_2, expand the vectors as in equation (6). The values of b_1, b_2,..., b_{N-1} are taken sequentially from the bitstream.
8. Calculate the inverse reversible integer transform of the resulting vectors using $f^{-1}(\cdot)$ (see equation (2)) to produce the watermarked S_1^w and S_2^w.
9. Replace the pixel values in the image, $I(i,j,k)$, with the corresponding values from the watermarked vectors in S_1^w and S_2^w to produce the watermarked image $I^w(i,j,k)$.

It should be noted here that the size of bitstream B must be less than or equal to $N-1$ times the size of the set S_4. To meet this condition, the values of the threshold T_1, T_2,..., T_{N-1} must be set properly. Also, note that the algorithm is not limited to RGB images. Using the RGB space in the previous discussion was merely for illustration purposes, and using the algorithm with other type of color images is straightforward.

B. Reading a Watermark and Restoring the Original Image

To read the watermark and restore the original image, the following steps must be followed:

For every $U \in \{U_R, U_G, U_B\}$, do the following:

1. Form the set of vectors U from the image $I^w(i,j,k)$ using the security key K.
2. Calculate V using the reversible integer transform, $f(\cdot)$ (see equation (1)).
3. Use V, equation (6), and the conditions in equation (5) to divide the vectors in T into the two sets $\hat{S}_4$ and S_3. $\hat{S}_4$ has the same vectors as S_4, which was constructed during embedding, but the values of the entities in each vector may be different. Similarly, S_3 is the same set constructed during embedding, since it contains non-changeable vectors.
4. Extract the LSBs of $\tilde{v}_1$, $\tilde{v}_2$,…, $\tilde{v}_{N-1}$ of each vector in $\hat{S}_4$, and concatenate them to form the bitstream B, which is identical to that formed during embedding.
5. Identify the EOS symbol and extract sub-bitstream B_1. Then, decompress B_1 to restore the location map M, and, hence, identify the member vectors of the set S_1 (expandable vectors). Collect these vectors into set $\hat{S}_1$.
6. Identify the member vectors of S_2. They are the members of $\hat{S}_4$ who are not members of $\hat{S}_1$. Form the set $\hat{S}_2 = \hat{S}_4 - \hat{S}_1$.
7. Sequence through the member vectors of $\hat{S}_1$ and $\hat{S}_2$ as they occur in the image and through the bits of the bitstream B in their natural order after discarding the bits of B_1. For $\hat{S}_1$, restore the original values of v_1, v_2,…, v_{N-1} as follows:
$$v_1 = \left\lfloor \frac{\tilde{v}_1}{2} \right\rfloor, \quad v_2 = \left\lfloor \frac{\tilde{v}_2}{2} \right\rfloor, \cdots, v_{N-1} = \left\lfloor \frac{\tilde{v}_{N-1}}{2} \right\rfloor. \tag{7}$$
For $\hat{S}_2$, restore the original values of v_1, v_2,…, v_{N-1} according to equation (6). The values of b_1, b_2,…, b_{N-1} are taken sequentially from the bitstream.
8. Calculate the inverse reversible integer transform of the resulting vectors using $f^{-1}(\cdot)$ (see equation (2)) to restore the original S_1 and S_2.
9. Replace the pixel values in the image $I^w(i,j,k)$ with the corresponding values from the restored vectors in S_1 and S_2 to restore the original image $I(i,j,k)$.
10. Discard all the bits in the bitstream B, which were used to restore the original image. Form the sub-bitstream B_3 from the remaining bits. Read the payload and authenticate the image using the watermark contained in B_3.

IV. Payload Size

To be able to embed data into the host image, the size of the bitstream B must be less than or equal to $N-1$ times the size of the set S_4. This means that

$$\|S_1\| + \|S_2\| = \frac{\|B_1\| + \|B_2\| + \|B_3\|}{N-1}, \tag{8}$$

where $\|x\|$ indicates number of elements in x. But $\|B_2\| = (N-1)\|S_2\|$; hence, equation (8) can be reduced to

$$\|B_3\| = (N-1)\|S_1\| - \|B_1\|. \tag{9}$$

For Tian's algorithm, the bitstream size is $\|B_3\| = \|S_1\| - \|B_1\|$, which can be obtained from equation (9) by setting $N = 2$.

Equation (9), above, indicates that the size of the payload that can be embedded into a given image depends on the number of expandable vectors that can be selected for embedding and on how well their location map can be compressed.

With $w \times h$ host image, the algorithm would generate $\frac{w \times h}{N}$ vectors. Only a portion, α $(0 \le \alpha \le 1)$, of these vectors can be selected for embedding; i.e., $\|S_1\| = \alpha \frac{w \times h}{N}$. Also, the algorithm would generate a binary map, M, containing $\frac{w \times h}{N}$ bits. This map can be compressed losslessly by a factor β $(0 \le \beta \le 1)$. This means that $\|B_1\| = \beta \frac{w \times h}{N}$. Ignoring the unchangeable vectors (i.e., assuming $\|S_3\| = 0$) and using equation (9), the potential payload size (in bits) becomes

$$\begin{aligned} \|B_3\| &= (N-1)\alpha \frac{w \times h}{N} - \beta \frac{w \times h}{N} \\ &= \left(\frac{N-1}{N}\alpha - \frac{1}{N}\beta \right) \times w \times h. \end{aligned} \tag{10}$$

Equation (10), above, indicates that the algorithm is effective when N and the number of selected expandable vectors are reasonably large. In this case, it does not matter if the binary map, M, is difficult to compress (which is because its size is very small). But, when each vector is formed from N consecutive pixels (row- or column-wise) in the image, and when N is large, the number of expandable vectors may decrease substantially; consequently, the values of the thresholds T_1, T_2,…, T_{N-1} must be increased to maintain the same number of selected expandable vectors. This increase causes a decrease in the quality of the embedded image. Such a decrease can be ignored by many applications, since the embedding process is reversible and the original image can be obtained at any time. In this case, the algorithm becomes more suitable for low SNR embedding than for high SNR embedding. To maximize $\|B_1\|$ for high SNR embedding, N either must be kept relatively small or each vector must be formed from adjacent pixels in the two-dimensional area in the image. The quad ($N = 4$) structure given in the next section satisfies both requirements simultaneously.

When $\alpha \le \frac{\beta}{N-1}$, the payload size in equation (10) becomes negative. In this case, nothing can be embedded into the image. This scenario is less likely to happen with natural images. Most lossless compression algorithms can achieve a 2:1 compression ratio easily (i.e., $\beta = \frac{1}{2}$). In this case, α must be greater than $\frac{1}{2(N-1)}$ to be able to embed a non–zero payload. This ratio can be satisfied easily when $N > 2$.

For Tian's algorithm, where $N = 2$, the payload size becomes

$$\|B_3\| = \left(\frac{\alpha}{2} - \frac{\beta}{2}\right) \times w \times h. \tag{11}$$

Equation (11), above, suggests that the ratio of selected expandable pairs, α, has to be much higher than the achievable compression ratio, β, for Tian's algorithm to be effective. When the compression ratio of the binary map is more than the ratio of selected expandable pairs, Tian's algorithm cannot embed anything into the host image. However, since Tian uses pairs of pixels as vectors, the correlation of the pixels in each pair is expected to be very high in natural images. This correlation makes the pair easier to satisfy smaller thresholds and, hence, to produce a large portion of selected expandable pairs. The major drawback of Tian's algorithm is the size of the binary map. To almost double the amount of data that can be embedded into the host image, Tian applies his algorithm row-wise, then column-wise.

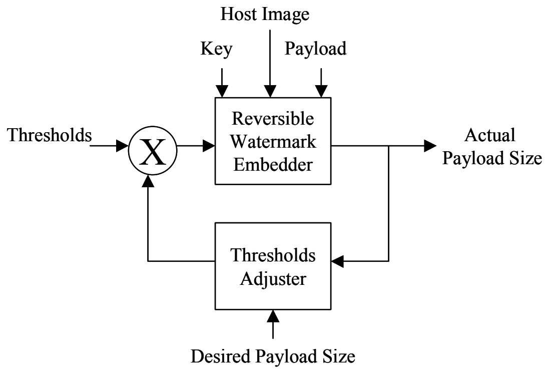

Fig. 2. Feedback system for adjusting the thresholds.

V. Data Rate Controller

For a given vector size, N, the payload size that can be embedded into an image and the quality of the resulting image is solely determined by the host image itself and by the value of the thresholds used. However, most practical applications require the embedding of a fixed-size payload regardless of the nature of the host image. Hence, an automatic data-rate controller is necessary to adjust the value of the thresholds properly and to compensate for the effect of the host image. The simple iterative feedback system depicted in Fig. 2 can be used for this purpose.

If $T(n) = [T_1(n), T_2(n), \cdots T_{N-1}(n)]$ is the threshold's vector at the nth iteration, and if C is the desired payload length, then the following proportional feedback controller can be used:

$$T(n) = T(n-1) - \lambda(C - \|B_3\|)T(n-1), \tag{12}$$

where $0 < \lambda < 1$ is a constant that controls the speed of convergence. $T(0)$ is a preset value that reflects the relative weights between the entities of the vector used in the difference expansion transform.

VI. Recursive and Cross-Color Embedding

A. Recursive Embedding

Applying the algorithm recursively as in Fig. 3 can increase its hiding capacity. This recursive application is possible because the proposed watermark embedding is reversible, which means that the input image can be recovered exactly after embedding. However, the difference between the

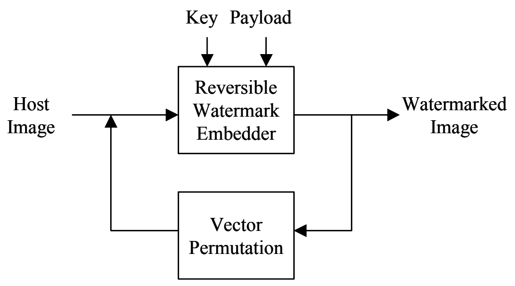

Fig. 3. Recursive embedding of the reversible watermark.

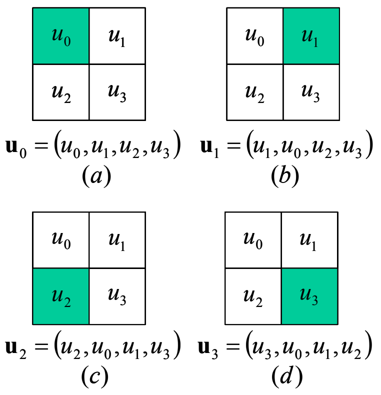
u_0 u_1 u_2 u_3
$\mathbf{u}_0=(u_0,u_1,u_2,u_3)$
(a)
$\mathbf{u}_1=(u_1,u_0,u_2,u_3)$
(b)
$\mathbf{u}_2=(u_2,u_0,u_1,u_3)$
(c)
$\mathbf{u}_3=(u_3,u_0,u_1,u_2)$
(d)

Fig. 4. Quad permutation for recursive embedding.

original image and the embedded image increases with every application of the algorithm. At some point this difference becomes unacceptable for the intended application. However, since the original image always can be recovered exactly, most applications have a high tolerance to this error.

One way to reduce the error when the algorithm is applied recursively is to use permutations of the entities of the input vector, which is depicted in Fig. 4 for quad vectors. The figure suggests four difference quad structures, each of which can be used in a different iteration for a total of four iterations. For $\mathbf{u}_0$, the difference expansion of equation (1) is performed based on u_0, so the closer u_0 is to u_1, u_2, and u_3, the smaller the difference is and, hence, the smaller the embedding error is. Similarly, for $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$, the difference expansion will be based on u_1, u_2, and u_3, respectively. This use of permutations allows the algorithm to exploit the correlation within a quad completely.

B. *Cross-Color Embedding*

To hide even more data, the algorithm can be applied across color components after it is applied independently to each color component. In this case, the vector $\mathbf{u}$ contains the color components (R,G,B) of each pixel arranged in a predefined order. This cross-color application can be done either as cross-color triple $\mathbf{u}=(R,G,B)$, as cross-color quad $\mathbf{u}=(R,G,G,B)$, or as a permutation thereof. For the cross-color quad arrangement, the integer difference expansion transform becomes

$$\begin{aligned} v_0 &= \left\lfloor \frac{R+2G+B}{4} \right\rfloor \\ v_1 &= R-G \\ v_2 &= B-G \end{aligned} \tag{13}$$

$$\begin{aligned} G &= v_0 - \left\lfloor \frac{v_1+v_2}{4} \right\rfloor \\ R &= v_1 + G \\ B &= v_2 + G, \end{aligned} \tag{14}$$

which is the reversible component transform proposed in JPEG2000 for color conversion from RGB to YUV [14].

Although, the spirit of the payload size analysis of Section IV applies to the cross-color vectors, the results need some modification. This modification is needed because only two bits are embedded per cross-color triplet or quad, rather than $N-1$, and the number of vectors, in this case, equals the area of the location map, which equals the area of the original image. Hence,

$$\begin{aligned} \|B_3\| &= 2\|S_1\| + \|B_1\| \\ \|B_3\| &= (2\alpha-\beta)\times w \times h \end{aligned} \tag{15}$$

VII. Experimental Results

We implemented the algorithm detailed in Section III and tested it with spatial triplets, spatial quads, cross-color triplets, and cross-color quads with $a_0 = a_1 = \cdots = a_{N-1} = 1$. In all cases, we used a random binary sequence derived from a uniformly distributed noise as a watermark signal. We tested the algorithm with the 512x512 RGB images: *Lena*, *Baboon*, and *Fruits*. We set $T_1 = T_2 = T_3$ in all experiments.

A. *Spatial Triplets*

A spatial triplet is a 1x3 or 3x1 vector formed from three consecutive pixel values in the same color component row- or column-wise, respectively. We applied the algorithm recursively to each color component: first to the columns and then to the rows. The payload size embedded into each of the test images (all color components) is plotted against the peak signal-to-noise ratios (PSNR) of the resulting watermarked image in Fig. 5. The plot indicates that the achievable

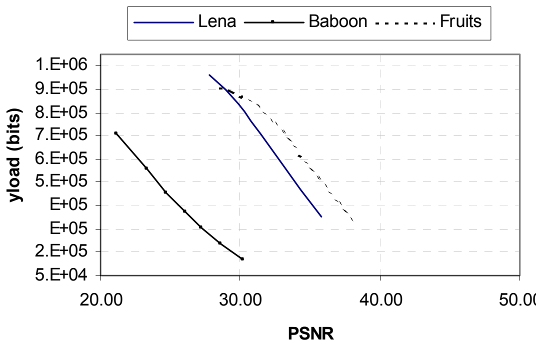

Fig. 5. Embedded payload size vs. PSNR for colored images embedded using spatial triplet-based algorithm.

embedding capacity depends on the nature of the image itself. Some images can bear more bits with lower distortion in the sense of PSNR than others. Images with a lot of low frequency contents and high correlation, like *Lena* and *Fruits*, produce more expandable triplets with lower distortion (in the PSNR sense) than high frequency images, such as *Baboon*, and, hence, can carry more watermark data at higher PSNR.

The proposed algorithm performs slightly better with *Fruits* than with *Lena*. With *Fruits,* the algorithm is able to embed 858 kbits with an image quality of 28.52 dB. The algorithm is also able to embed 288 kbits with reasonably high image quality of 37.94 dB. Nevertheless, the performance of the algorithm is lower with *Baboon* than with *Lena* or *Fruits*. With *Baboon* the algorithm is able to embed 656 kbits at 21.2 dB and 115 kbits at 30.14 dB.

The visual quality of the watermarked images is shown in Fig. 6 and Fig. 7 for *Lena* and *Baboon* embedded at very low, low, and medium PSNR. In general, the embedded images hardly can be distinguished from the original. However, a sharpening effect can be observed when the original and the embedded images are displayed alternatively. This effect is more noticeable at lower PSNR than at higher PSNR. It is also more noticeable for a high frequency image, such as *Baboon,* than for *Lena* and *Fruits*.

B. *Spatial Quads*

A spatial quad was assembled from 2x2 adjacent pixels in the same color component as shown in Fig. 4a. We applied the algorithm to each color component independently. The payload size embedded into each of the test images (all color components) is plotted against PSNR in Fig. 8. Again, the plot indicates that the achievable embedding capacity depends on the nature of the image itself. The algorithm performs with *Fruits* and *Lena* much better than *Baboon*, and it performs slightly better with *Fruits* than with *Lena*. With *Fruits,* the algorithm is able to embed 867 kbits with image quality of 33.59 dB. It is also able to embed 321 kbits with high image quality of 43.58 dB. Nevertheless, with *Baboon* the algorithm is able to embed 802 kbits at 24.73 dB and 148 kbits at 36.6 dB.

Comparing Fig. 8 with Fig. 5 reveals the achievable payload size for the spatial quad-based algorithm is about 300,000 bits higher than for the spatial triplets-based algorithm at the same PSNR, and the PSNR is about 5 dB higher for spatial quad-based algorithm than for spatial triplet-based algorithm at the same payload size. Also, the spatial quad-based algorithm has finer control over the payload size

Fig. 6. *Lena* embedded using spatial triplet-based algorithm (a) original, (b) 27.76 dB embedded with 910,802 bits, (c) 31.44 dB embedded with 660,542, (d) 35.80 dB embedded with 305,182 bits.

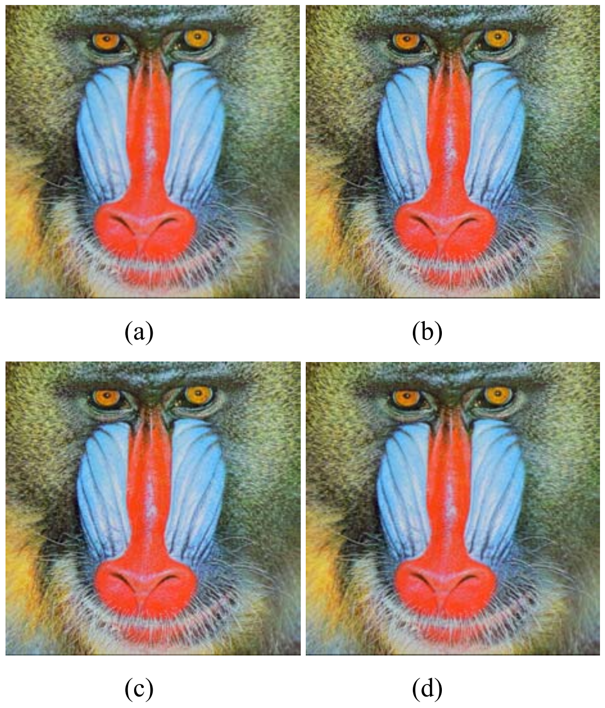

Fig. 7. *Baboon* embedded using spatial triplet-based algorithm (a) original, (b) 21.20 dB embedded with 656,296 bits, (c) 24.74 dB embedded with 408, 720 bits, (d) 30.14 dB embedded with 115,026 bits.

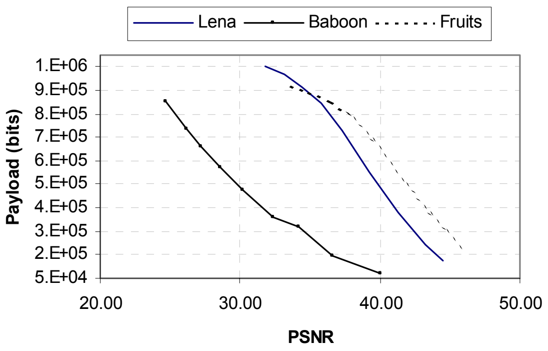

Fig. 8. Embedded payload size vs. PSNR for colored images embedded using spatial quad-based algorithm.

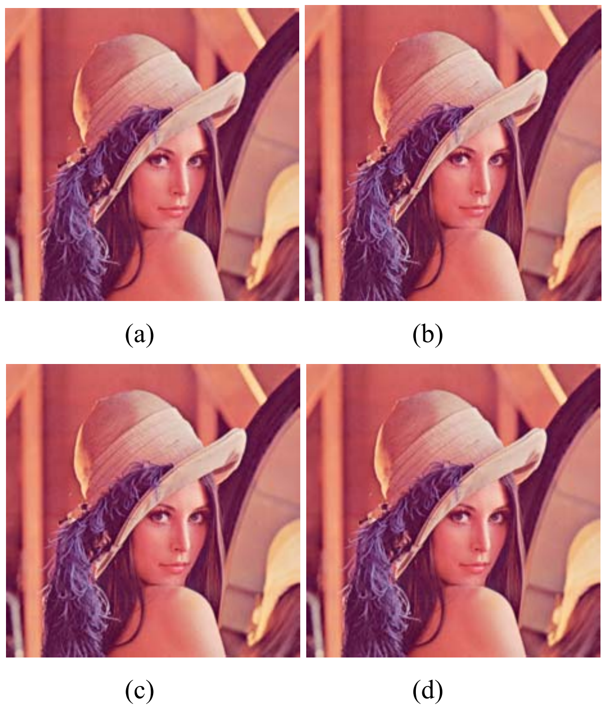

Fig. 9. *Lena* embedded using spatial quad-based algorithm (a) original, (b) 31.78 dB embedded with 955,558 bits, (c) 37.34 dB embedded with 680,053, (d) 44.56 dB embedded with 122,440 bits.

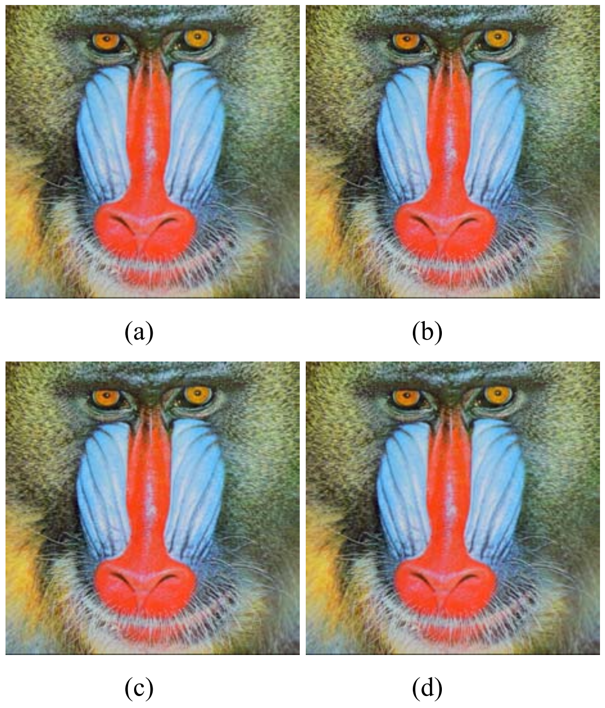

Fig. 10. *Baboon* embedded using spatial quad-based algorithm (a) original, (b) 24.73 dB embedded with 802,219 bits, (c) 30.19 dB embedded with 423,511, (d) 40.00 dB embedded with 71,695 bits.

and the PSNR than the spatial triplet-based algorithm. For example, it was possible to produce images at PSNRs in the 38 dB to 46 dB range with spatial quad-based algorithm, but not with spatial triplet-based algorithm. This result is because 2x2 spatial quads have higher correlation than 1x3 spatial triplets and because the single location map used by the spatial quad-based algorithm is smaller than each of the two location maps used by the spatial triplet-based algorithm (one location map for each pass).

The visual quality of the watermarked image is shown in Fig. 9 and Fig. 10 for *Lena* and *Baboon* embedded at low, medium, and high SNR. In general, the quality of the embedded images is better than that obtained by the algorithm using spatial triplets. Also, the sharpening effect is less noticeable.

C. *Cross-Color Triplets and Cross-Color Quads*

The cross-color triplets and quads were formed from the RGB values of each pixel, as described in Section VI. Fig. 11 and Fig. 12 plot the size of the payload embedded into each of the test images against PSNR for cross-color triplets and cross-color quads, respectively. Both figures show that the achievable payload size and the PSNR using cross-color vectors are much lower than those using spatial vectors. Hence, for a given PSNR level, it is better to use spatial vectors than cross-color vectors.

Also, Fig. 11 and Fig. 12 clearly show that the cross-color triplet-based algorithm has almost the same performance as the cross-color quad-based algorithm with all test images except *Lena* at PSNR grater than 30. While the cross-color triplet-based algorithm was able to embed small payloads at these higher PSNRs, the cross-color quad-based algorithm was not. Upon closer inspection of the *Lena* image, we noticed that the blue channel of *Lena* is very close to the green channel. Also, upon further inspection of the cross-color

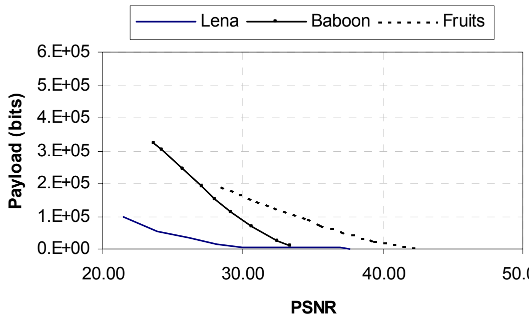

Fig. 11. Embedded payload size vs. PSNR for colored images embedded using cross-spectral triplet-based algorithm.

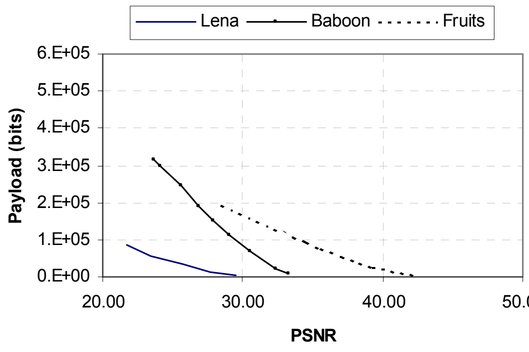

Fig. 12. Embedded payload size vs. PSNR for colored images embedded using cross-spectral quad-based algorithm.

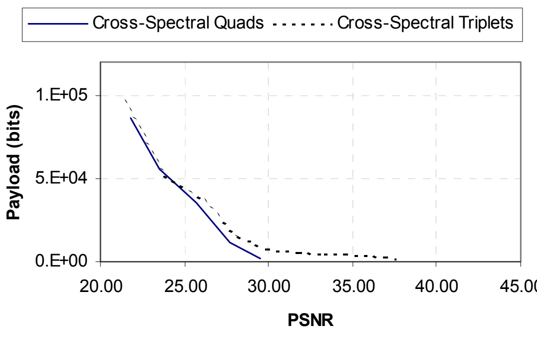

Fig. 13. Payload size vs. PSNR for *Lena* colored image using cross-spectral triplet-based and quad-based algorithms.

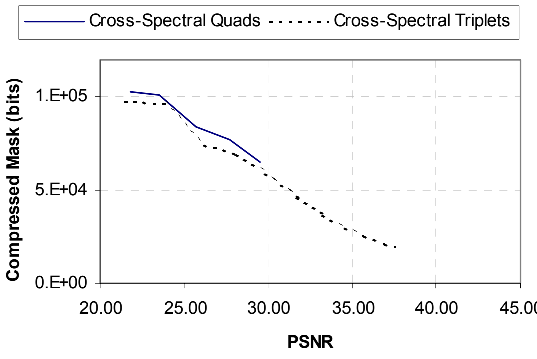

Fig. 14. Size of compressed map vs. PSNR for *Lena* colored image using cross-spectral triplet-based and quad-based algorithms.

triplet-based and cross-color quad-based transforms, we noticed that when the red or blue channel is close in value to the green channel, the dynamic range of G after expansion according to equation (5) becomes wider for the cross-color quad-based transform than for the cross-color triplet-based transform. Hence, in this case, the cross-color triplet-based algorithm has the potential of producing more expandable vectors and a location map of less entropy than the cross-color quad-based transform. And, indeed, this was the case with the *Lena* image as can be seen in Fig. 13 and Fig. 14.

D. Comparison with Other Algorithms in the Literature

We also compared the performance of the proposed algorithm with that of Tian's described in [11] using grayscale *Lena* and *Barbara* images. Recall that Tian's algorithm uses spatial pairs rather than spatial triplets and spatial quads. The results are plotted in Fig. 15 for the spatial triplet-based algorithm and in Fig. 16 for the spatial quad-based algorithm. As expected, Fig. 15 indicates that our spatial triplet-based algorithm outperforms Tian's at low PSNR, but Tian's algorithm out performs ours at high PSNR. In contrast, Fig. 16 indicates that our spatial quad-based algorithm outperforms Tian's at PSNR higher than 35dB, but Tian's algorithm marginally outperforms ours at lower PSNR. Moreover, our spatial quad-based algorithm is applied once to the image data, while Tian's is applied twice: The first time it is applied column-wise and the second time row-wise. Having to apply the algorithm only once makes our algorithm more efficient.

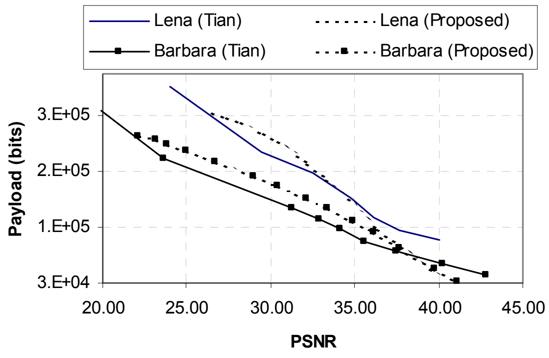

Fig. 15. Comparison results between the proposed spatial triplet-based algorithm and Tian's using grayscale images.

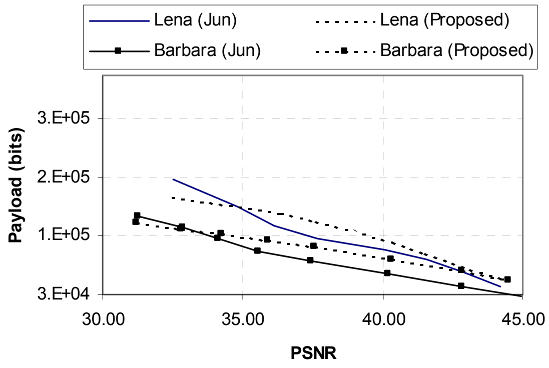

Fig. 16. Comparison results between the proposed spatial quad-based algorithm and Tian's using grayscale images.

We also compared our proposed algorithm with that of Celik [10] using grayscale *Lena* and *Barbara* images. The results are plotted in Fig. 17 for the spatial triplet-based algorithm and in Fig. 18 for the spatial quad-based algorithm. Fig. 17 indicates that our spatial triplet-based algorithm also outperforms Celik's at low PSNR, but our algorithm has similar performance to Celik's at high PSNR. In contrast, Fig. 18 indicates that our quad-based algorithm is superior to Celik's at almost all PSNRs.

VIII. CONCLUSIONS

In this paper, a very high capacity algorithm based on the difference expansion of vectors of an arbitrary size has been developed for embedding a reversible watermark with low image distortion. Test results of the spatial triplet-based and spatial quad-based algorithms indicate that the amount of data one can embed into an image depends highly on the nature of the image. The test results also indicate that the performance of the spatial quad-based algorithm is superior to that of the spatial triplet-based algorithm. These results also show that applying the algorithm across the color components has inferior performance to applying the algorithm spatially; hence, cascading cross-color with spatial applications would be useful only when there is a need to hide a large amount of data without regard to the quality of the watermarked image. However, since spatial quad-based embedding provides a very high embedding capacity at low image distortion, and since

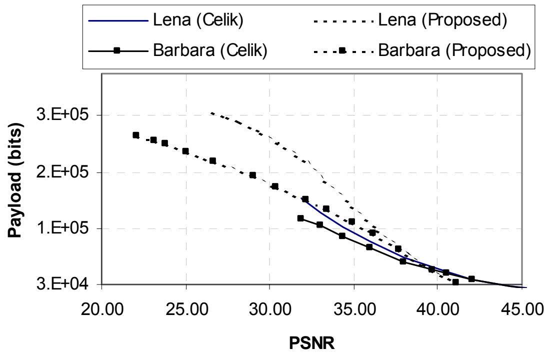

Fig. 17. Comparison results between the proposed spatial triplet-based algorithm and Celik's using gray scale images.

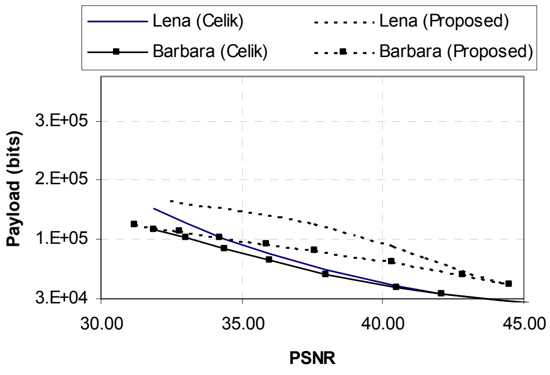

Fig. 18. Comparison results between the proposed spatial quad-based algorithm and Celik's using gray scale images.

the performance of this embedding is better than that of Tian's and Celik's algorithms, spatial quad-based embedding would be sufficient for most applications.

ACKNOWLEDGMENT

The author thanks Jun Tian, John Stach, Steve Decker, Joel Meyer, Duane Proefrock, and Kyle Smith at Digimarc Corporation for their helpful discussion and feedback.

REFERENCES

[1] I. J. Cox, M. L. Miller, J. A. Bloom, *Digital Watermarking*, Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[2] F. Mintzer, J. Lotspiech, and N. Morimoto. (Dec. 1997). Safeguarding digital library contents and users: digital watermarking. *D-lib magazine*.[Online]. Available: *http://www. Dlib.org*

[3] C. W. Honsinger, P. W. Jones, M. Rabbani, and J. C. Stoffel, "Lossless recovery of an original image containing embedded data," U.S. Patent 6,278,791, 2001.

[4] B. Macq, "Lossless multiresolution transform for image authenticating watermark," in *Proc. of EUSIPCO*, Tampere, Finland, Sept. 2000.

[5] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Systems Journal*, vol. 35, no. 3-4, pp. 313-336, 1996.

[6] C. De Vleeschouwer, J. F. Delaigle, and B. Macq, "Circular interpretation of histogram for reversible watermarking*,"* in *Proc. of IEEE 4th Workshop on Multimedia Signal Processing*, Oct. 2001.

[7] J. Fridrich, M. Goljan, and Rui Du, "Invertible authentication," in *Proc. SPIE Photonics West, Security and Watermarking of Multimedia Contents III*, vol. 3971, San Jose, California, Jan. 2001, pp. 197-208.

[8] J. Fridrich, M. Goljan, and R. Du, "Lossless data embedding—new paradigm in digital watermarking," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 2, pp. 185-196, Feb. 2002.

[9] J. Fridrich, M. Goljan, and R. Du, "Lossless data embedding for all image formats*,"* in *Proc. SPIE Photonics West, Electronic Imaging 2002, Security and Watermarking of Multimedia Contents*, vol. 4675, San Jose, California, Jan. 2002, pp. 572–583.

[10] M. U. Celik, G. Sharma, A. M. Tekalp, and E. Saber, "Reversible data hiding," in *Proc. of the IEEE International Conference on Image Processing*, vol. II, Sept. 2002, pp. 157-160.

[11] J. Tian, "Reversible watermarking by difference expansion," in *Proc. of Workshop on Multimedia and Security: Authentication, Secrecy, and Steganalysis,* J. Dittmann, J. Fridrich, and P. Wohlmacher, Eds., Dec. 2002, pp. 19-22.

[12] T. Kalker and F.M. Willems, "Capacity bounds and code constructions for reversible data-hiding," in *Proc. of Electronic Imaging 2003, Security and Watermarking of Multimedia Contents V*, Santa Clara, California, Jan. 2003.

[13] T. Kalker and F. M. Willems, "Capacity bounds and constructions for reversible data-hiding," in *Proc. of the International Conference on Digital Signal Processing*, June 2002, pp.71-76.

[14] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*. Kluwer Academic Publishers, 2002, pp. 422-423.

Adnan M. Alattar (M'86) was born in Khanyounis, Palestine in 1961. He graduated with a BS degree in electrical engineering from the University of Arkansas, Fayetteville, Arkansas in 1984. He also earned his MS and PhD degrees in electrical engineering from North Carolina State University, Raleigh, North Carolina in 1985 and 1989, respectively.

He was a Senior R&D Engineer at Intel Corporation from 1989 to1995. He was an Assistant Professor at King Fahd University for Petroleum and Minerals from 1995 to 1998. Since 1998, he has been a Senior R&D Engineer at Digimarc Corporation, Tualatin, Oregon. He holds 11 US and 2 European patents in the area of digital watermarking and video compression, and he is the author of several technical papers. His areas of interest include Digital Watermarking, Video Compression, and Image and Signal Processing.