REVERSIBLE IMAGE WATERMARKING BASED ON A GENERALIZED INTEGER TRANSFORM

Xudong Chen, Xiaolong Li, Bin Yang and Yingmin Tang

Institute of Computer Science and Technology, Peking University, Beijing 100871, China

ABSTRACT

In this paper, a recently proposed reversible image watermarking algorithm based on reversible contrast mapping (RCM) is further developed. The integer transform, RCM, which is originally defined on a pair of integers, is extended to integer array of arbitrary length. Based on this generalization, the embedding capacity can be significantly increased. Meanwhile, the embedding distortion is well controlled by a suitable selection of embeddable pixel blocks. In fact, the hidden data is priorly embedded into the blocks that introduce less distortion. Furthermore, the proposed scheme does not need additional data compression, and it has low computational complexity. The experiments verify that the novel scheme outperforms the original RCM-based method and some state-of-the-art reversible watermarking algorithms, especially the ones based on integer transform.

Index Terms— Information hiding, reversible watermarking, reversible contrast mapping (RCM), integer transform

1. INTRODUCTION

Digital watermarking is the process of hiding information into digital data such as image, audio or video files, for various purposes including copyright protection, content authentication, forensic tracking and so on. Usually, the watermark embedding procedure destroys the cover medium and results in distortion. Although the distortion is slight, it may still lead to misinterpretation of medical images or military images during image analysis. Reversible watermarking, which enables original image recovery, provides a method for these applications in which any distortion of the image is intolerable [1].

A range of reversible watermarking techniques have been presented in the literature. For instance, the integer transform based methods [2–5], histogram shifting based methods [6, 7], and compression technique based methods [8, 9], etc. In [2], Tian proposed a method based on the difference expansion of adjacent pixels, and one bit is embedded into a pixel pair. Thereafter, Alattar extended Tian’s scheme by employing a triple or quad of pixels rather than a pair, and the experimental results have shown that it outperforms Tian’s method by increasing embedding rate while holding low distortion [3]. In Tian and Alattar’s methods, a location map is used to solve the underflow/overflow problem. Note that, the location map is usually huge in size and should be compressed by a certain lossless compression algorithm. Thus the size of the compressed location map determines the performance of a method. Therefore, as pointed out by Sachnev et al. [10], the reversible watermarking techniques with smaller, or in some cases, no location maps, are desirable.

Recently, a reversible watermarking scheme based on reversible contrast mapping (RCM) is proposed by Coltuc et al. [4], in which the location map is not needed. The scheme does not need additional lossless data compression, and the computational complexity is extremely low for both data embedding and extraction. This important feature makes it appropriate for real time applications. However, the same as Tian’s approach does, this method can embed only one bit into a pixel pair. Thus, in a single pass embedding, its embedding rate cannot exceed 0.5 bit per pixel (BPP).

In this paper, we extend Coltuc et al.’s work and propose a novel reversible image watermarking scheme based on a generalized integer transform. The proposed method uses a block that contains \( n \) pixels, and \((n-1)\) bits are embedded into each suitably selected block, where \( n \) is a positive integer. Comparing with Coltuc et al.’s method, our solution can provide a higher BPP while giving a better peak signal to noise ratio (PSNR). The experiments verify that our method outperforms some recently proposed reversible watermarking algorithms, especially the ones based on integer transform. In addition, there are no data compression steps in our method, which results in fast data embedding and extraction. The rest of this paper is organized as follows. In Section 2, Coltuc et al.’s method is briefly reviewed, and the novel method is introduced in detail. The experiments including comparisons with some state-of-the-art algorithms will be given in Section 3. The final conclusion is drawn in last section.

2. THE PROPOSED METHOD

We first introduce the RCM, a transform defined on a pair of integers. More specifically, the RCM transforms a pair of pixel values \((x, y)\) into \((x’, y’)\) in the following way:

\[
x’ = 2x - y, \quad y’ = 2y - x.
\]  

(1)

To prevent the overflow/underflow problem, the RCM is restricted to a subset of \([0, 255]^2\): \(\{x, y\} \in [0, 255]^2 : 0 \leq 2x - y \leq 255, 0 \leq 2y - x \leq 255\}. The inverse RCM is defined by,

\[
x = \lfloor 2(f(x') + f(y'))/3 \rfloor, \quad y = \lfloor (f(x') + 2f(y'))/3 \rfloor,
\]  

(2)

where \([\alpha]\) is the smallest integer greater than or equal to \(\alpha\), \(f\) is a function defined by

\[
f(x) = 2\lfloor x/2 \rfloor,
\]

and \([\alpha]\) is the greatest integer smaller than or equal to \(\alpha\). The important property of RCM is, if either \(x\) or \(y\) in Eq.(1) is even, one can exactly recover \(x\) and \(y\) by using Eq.(2). It means that one can freely modify the least significant bits (LSBs) of \(x’\) and \(y’\) while holding the original integer pair recovery. In Coltuc et al.’s scheme, for the LSBs of \((x’, y’)\), one is used as a flag-bit, and the other one is for embedding data. It is clear that the upper bound of the embedding rate of this method is 0.5 BPP. In fact, one can increase the embedding rate by extending the RCM to integer array of arbitrary length. We show the details below.

---

Corresponding author: Yingmin Tang, e-mail: tym@founder.com
2.1. A generalization of RCM

We start with the following integer transform which maps an array of \( n \) integers \( x = (x_1, x_2, ..., x_n) \) to \( y = (y_1, y_2, ..., y_n) = T(x) \):

\[
y_i = 2x_i - x_{i+1}, \quad i = 1, 2, ..., n.
\]

(3)

Here, we take \( x_{n+1} = x_1 \) as convention. Obviously, the integer transform in Eq. (3) is just the RCM if \( n = 2 \), and one can get easily its inverse:

\[
x_i = \frac{1}{2} \left( x_i + x_{i+1} \right), \quad i = 1, 2, ..., n.
\]

(4)

We define then another integer transform \( T^* \) which maps an integer array \( y = (y_1, y_2, ..., y_n) \) to \( x = (x_1, x_2, ..., x_n) \):

\[
x_i = \left\lceil \frac{1}{4} \left( 2^{n-i}y_{i+1} - f(y_{i-1}) \right) \right\rceil, \quad i = 1, 2, ..., n.
\]

(5)

Recall here that the function \( f \) is defined by \( f(x) = 2\lfloor x/2 \rfloor \).

Similar to the RCM, one can prove the following important property of the above defined transforms \( T \) and \( T^* \).

**Theorem 1.** Suppose that \( x = (x_1, x_2, ..., x_n) \) is in \( \mathbb{Z}^n \). If at least one of \( x_i \) \( (1 \leq i \leq n) \) is even, we have \( T^*(T(x)) = x \).

**Proof.** Assume \( (y_1, y_2, ..., y_n) = T(x) \). One can prove that, at least one of \( y_i \) \( (1 \leq i \leq n) \) is even. Thus,

\[
0 \leq \sum_{j=1}^{n} 2^{n-j} (y_{i+1} - f(y_{i-1})) \leq \sum_{j=1}^{n} 2^{n-j} = 1.
\]

The theorem is then proved, by comparing Eq. (4) with Eq. (5). \( \square \)

The theorem demonstrates that one can recover the original integer array \( x \) even if the LSBs of \( y = T(x) \) are modified. As a result, the LSBs of \( y \) can be explored for reversible data embedding.

We now give several notations. Let

\[
A = \{ x = (x_1, x_2, ..., x_n) \in \mathbb{Z}^n : 0 \leq x_i \leq 255, \quad i = 1, 2, ..., n \}
\]

be the set of all possible pixel value arrays with length \( n \) of a gray-scale image, and we consider a subset of \( A \):

\[
D = \{ x \in A : 1 \leq 2f(x_i) - f(x_{i+1}) \leq 253, \quad i = 1, 2, ..., n \}.
\]

By the definition of \( D \), we know that, \( x \in D \Rightarrow T(x) \in A \). Thus, the pixel block whose value belongs to \( D \) can be used for data embedding. In addition, changing the LSBs of \( x \) will not affect the relationship between \( x \) and \( D \). Take now a subset \( S \subseteq D \), where

\[
S = \{ x \in D : g(x) \leq t \},
\]

\[
g(x) = \left( \sum_{i=1}^{n} (f(x_i) - f(x_{i+1}))^2 \right)^{\frac{1}{2}},
\]

and \( t > 0 \) is a predefined threshold. The same as the set \( D \), the relationship between \( x \) and \( S \) will not be affected when the LSBs of \( x \) are modified. In addition, we remark here that in our scheme, the distortion of watermark embedding is mainly caused by the integer transform (see Eq. (3)), and this distortion (for a given pixel block whose value is \( x \)) is approximately

\[
\sum_{i=1}^{n} (y_i - x_i)^2 \approx \sum_{i=1}^{n} (x_i - x_{i+1})^2 \approx (g(x))^2.
\]

(6)

---

Fig. 1. A pixel labeling strategy for a 4 × 4 sized gray-scale image “Lena”, with different block size.

In consequence, one can control the embedding distortion by selecting a threshold \( t \) appropriately.

We then define

\[
S_1 = \{ x \in S : \exists i \in \{1, 2, ..., n\} \text{ s.t. } x_i \text{ is even} \},
\]

\[
S_2 = S - S_1 = \{ x \in S : \forall i \in \{1, 2, ..., n\} \Rightarrow x_i \text{ is odd} \},
\]

and \( S_3 = A - S \). Similar to Coltuc et al.’s approach, the pixel block whose value belongs to \( S_1 \) will be used for data embedding by using the generalized RCM, and the pixel block whose value belongs to \( S_2 \) will also be used for data embedding by simply replacing the LSBs. In particular, a flag-bit is used to determine whether the block is in \( S_1 \) or \( S_2 \). We now further decompose \( S_3 \) to get a partition, \( S_3 = S_{3,1} \cup S_{3,2} \), with

\[
S_{3,1} = \{ x \in S_3 : T^*(x) \in S_1 \},
\]

\[
S_{3,2} = S_3 - S_{3,1} = \{ x \in S_3 : T^*(x) \notin S_1 \}.
\]

The main advantage of the decomposion is that one can exactly identify the block whose value belongs to \( S_{3,2} \) without any modification. Thus the number of flag-bits (used for the blocks in \( S_3 \)) is decreased, and meanwhile the embedding capacity is increased.

In summary, according to the above definitions, one gets a partition of \( A \), \( A = S_1 \cup S_2 \cup S_3 = S_1 \cup S_2 \cup S_{3,1} \cup S_{3,2} \). The four sets \( S_1, S_2, S_{3,1} \) and \( S_{3,2} \) will be treated separately in the following data embedding procedure.

2.2. The watermark embedding procedure

We now give the detailed data embedding procedure.

**Step 1:** Divide the host image into non-overlapping blocks, and each block contains \( n \) pixels.

**Step 2:** For a given pixel block with value \( x = (x_1, x_2, ..., x_n) \), where \( x_i \) is the value of the \( i \)-th pixel in the block. Record the LSB of \( x_1 \) if \( x \in S_{3,1} \). We get a binary sequence when all the divided blocks are scanned in a certain order. The sequence will be embedded into the image as a part of watermark.

**Step 3:** Consider a given pixel block whose value is \( x = (x_1, x_2, ..., x_n) \). The block is modified according to four cases.

**Case 1:** When \( x \in S_1 \), take \( (y_1, y_2, ..., y_n) = T(x) \), and we replace \( x \) by \( (f(y_1), f(y_2) + w_1, ..., f(y_n) + w_{n-1}) \), where \( w_i (1 \leq
Fig. 3. Performance comparisons on four standard $512 \times 512$ sized gray-scale images: “Lena”, “Baboon”, “Airplane” (F-16) and “Barbara”.

$i \leq n - 1$ is the corresponding watermark bit. In this case, the pixel values are first transformed and then we replace the LSBs of the resulting integer array by the watermark bits. In particular, the first LSB of the transformed integer array is set to 0, which serves as a flag-bit. Thus, $(n - 1)$ bits are embedded in this block.

Case 2: When $x \in S_2$, we replace $x$ by $(f(x_1) + 1, f(x_2) + w_1, ..., f(x_n) + w_{n-1})$, where $w_i (1 \leq i \leq n - 1)$ is the watermark bit. Here, the LSB of $x_1$ is set to 1 and the LSBs of $x_i (2 \leq i \leq n)$ are replaced by the watermark bits. The same as the above case does, $(n - 1)$ bits are embedded in this block.

Case 3: When $x \in S_{3,1}$, replace $x$ by $(f(x_1) + 1, x_2, ..., x_n)$. In this case, only the first LSB is used as a flag-bit, and there are no embedded data in this block.

Case 4: When $x \in S_{3,2}$, nothing will be changed.

Step 4: Repeat Step 3 for all divided pixel blocks and we get the watermarked image.

Denote now $z = (z_1, z_2, ..., z_n)$ the watermarked pixel value of $x$. By the definitions in subsection 2.1 and Thm.1, one can prove,

- $x \in S_1 \Rightarrow$ the LSB of $z_1$ is 0 and $T^\ast(z) = x \in S_1$.
- $x \in S_2 \Rightarrow$ the LSB of $z_1$ is 1 and $z \in S_2$.
- $x \in S_{3,1} \Rightarrow$ the LSB of $z_1$ is 1, $z \notin S$ and $T^\ast(z) = T^\ast(x) \in S_1$.
- $x \in S_{3,2} \Rightarrow z \notin S$ and $T^\ast(z) = T^\ast(x) \notin S_1$.

The above properties will be used for original image recovery.

Finally, before closing this subsection, we point out that the embedding capacity of the proposed scheme is $(n-1)N - N'$, where $N$ ($N'$, resp.) is the number of the pixel blocks whose values belong to $S$ ($S_{3,1}$, resp.). The method is then applicable if and only if $(n-1)N > N'$. In fact, this condition can be simply satisfied when the threshold $t$ is large enough. For instance, $t \geq 3$ is sufficient for embedding data, when $n = 4$ and the host image is the $512 \times 512$ sized gray-scale image “Lena”.

2.3. The watermark extraction procedure

The watermark extraction and the original image restoration procedures are given in this subsection.

Step 1: The same as the data embedding procedure does, divide the watermarked image into non-overlapping blocks such that each block contains $n$ pixels.

Step 2: Consider a given pixel block whose value is $z = (z_1, z_2, ..., z_n)$. We discuss according to the parity of $z_1$.

Case 1: When $z_1$ is even, we know that the original pixel value belongs to $S_1 \cup S_{3,2}$.
- If $T^\ast(z) \in S_1$, then the original pixel value belongs to $S_1$. Recover the original pixel value as $T^\ast(z)$, and extract the watermark bits as $w_{i-1} = z_i - f(z_i)$ for each $i > 1$.
- If $T^\ast(z) \notin S_1$, then the original pixel value belongs to $S_{3,2}$. Recover the original pixel value as $z$ and there are no hidden data to be extracted.

Case 2: When $z_0$ is odd, we know that the original pixel value belongs to $S_2 \cup S_3$.
- If $z \in S$, then the original pixel value belongs to $S_2$. Recover the original pixel value as $(f(z_1) + 1, f(z_2) + 1, ..., f(z_n) + 1)$, and extract the watermark bits as $w_{i-1} = z_i - f(z_i)$ for each $i > 1$.
- Otherwise, the original pixel value belongs to $S_3 = S_{3,1} \cup S_{3,2}$. If $T^\ast(z) \in S_3$, then the original pixel value belongs to $S_{3,1}$, and the restoration of this type of blocks will be given in Step 3 below. If $T^\ast(z) \notin S_1$, we know that the original pixel value belongs to $S_{3,2}$, recover the original pixel value as $z$, and there are no hidden data to be extracted.

Step 3: Repeat Step 2 for all divided blocks. Then, in order to complete the original image restoration, we replace the first LSB of the pixel blocks whose values belong to $S_{3,1}$ by the corresponding extracted hidden bits.
3. EXPERIMENTAL RESULTS

The experiments are reported in this section.

First, we point out that the $i$-th pixel and the $(i+1)$-th pixel in a divided block should be connected, for each $i \in \{1, 2, \ldots, n\}$. In this way, the estimated embedding distortion $g(x)$ (see Eq.(6)) might be minimized as far as possible because neighboring pixels are often highly correlated. For instance, a pixel labeling strategy is plotted in Fig.1, for $n = 16$ with $4 \times 4$ sized block.

Let us then see Fig.2. The figure shows the performance of the proposed method for different block size $n \in \{2, 4, 8, 16, 32\}$, for “Lena”. From the figure, we see that with increasing $n$, the method can embed more data and its performance becomes better. However, this gain is slight when $n$ is large enough. Therefore, we take simply $n = 16$ in the following experiments to present the performance of the proposed method. More specifically, for data embedding, the image is firstly divided into blocks with size $4 \times 4$ and then the pixels in each block are labeled as shown in Fig.1.

We now see Fig.3. It shows the comparisons of the following schemes, for four standard $512 \times 512$ sized gray-scale images: “Lena”, “Baboon”, “Airplane” (F-16) and “Barbara”. (1) Our method, for the threshold $t$ varying from its minimal value (the integer such that the embedding rate is positive) to 400. (2) Tian’s method [2]. (3) Alattar’s method [3], for pixel quads. (4) Coltuc et al.’s method [4]. (5) Weng et al.’s method [5]. (6) Tai et al.’s method [7]. Here, we implement the algorithms (2)-(6) ourself, and only the results of single pass embedding are presented. In addition, arithmetic coding is used for compressing the location map for the methods (2), (3), (5) and (6). From this figure, we see that the proposed method is better than the integer transform based methods (2)-(5). However, in the case of high embedding rate (for instance, larger than 0.7 for “Airplane”), our method performs worse than Tai et al.’s method. The little glitch will be fixed by using multiple embedding (or, in other words, repeated embedding).

Note that the proposed method is dependent on the threshold $t$. Thus, for two pass embedding, there are several choices to select the thresholds $(t_1, t_2)$, where $t_1$ and $t_2$ are the thresholds used for the first and the second pass embedding, respectively. We have tested the following three cases: (1) $t_1$ is fixed as a large value and $t_2$ varies; (2) $t_2$ is fixed as a large value and $t_1$ varies; (3) $t_1$ and $t_2$ are taken the same value. In fact, the experiments show that the three cases achieve approximately the same performance. We then take simply $t_1 = t_2$ for two pass embedding. The comparison results are shown in Fig.4. The figure confirms that our method gives a better performance than that of Tai et al.’s method. In particular, for our method, the repeated embedding performs better than single pass embedding when the embedding rate is large (for instance, when the embedding rate is larger that 0.6, for “Airplane”). The interesting fact should be investigated in the future work, in order to arrive at the best embedding performance.

4. CONCLUSION

In this work, we have presented a novel reversible image watermarking scheme based on a generalized integer transform. The experiments have shown that grouping more pixels in a block and then taking the block as an embedding unit can significantly increase the embedding rate. In addition, the novel method enables us to select the blocks that introduce less distortion to embed data priorly, and the embedding distortion is then well controlled. With these improvements, the novel scheme obtains a better performance than the other integer transform based reversible watermarking algorithms.

Fig. 4. Performance comparison between our method (multiple embedding) and Tai et al.’s method [7].

5. REFERENCES