IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 19, 2021, accepted April 3, 2021, date of publication April 8, 2021, date of current version April 22, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3071877

M-Array Based on Non-Zero Maps

XIAO ZHOU, YU KANG, TINGTING ZHANG, AND XINGANG MOU
School of Mechanical and Electronic Engineering, Wuhan University of Technology, Wuhan 430070, China
Corresponding author: Xingang Mou (sunnymou@whut.edu.cn)

ABSTRACT Two-dimensional maximum area array (m-array) is a typical pseudo-random array with specific window property - all sub-windows are globally unique. M-array constructed by Formula-Method-Based encoding algorithm features a big difference between the number of rows and columns. Enumeration-Method-Based encoding algorithm only constructs small-size m-array with high computational complexity and low success rate. Based on the characteristics of Galois Fields, a new method of m-array construction based on non-zero maps theory is proposed and the theoretical proof is presented. Given symbols p and sub-window size $m \times n$ as input, m-array with a large array size and close numbers of rows and columns is constructed by dividing and splicing m-sequence. Comparing with existing methods, the advantages of the algorithm include larger array size, more balanced number of rows and columns, and lower computational complexity. If needed, the method in this paper is flexible enough that the number of rows and columns of m-array can also be adjusted within a certain range, meaning that m-array with specific rows and columns ratio can be constructed.

INDEX TERMS M-array, non-zero maps, pseudo-random array, window property.

I. INTRODUCTION

In the field of structured light 3D reconstruction, structured light encoding is used to solve the problem of matching the corresponding point, which will directly affect the accuracy, resolution, and real-time performance of the measurement. Pseudo-random codes with pseudo-random characteristics being predetermined and repeatedly constructed include pseudo-random sequences and pseudo-random arrays. m-array is a typical pseudo-random array that is described as $U(u, v; m, n; p)$, meaning that each $m \times n$ sub-window with symbols p appears only once in $u \times v$ array U.

Pseudo-random arrays have been studied for many years. In 1976, McWilliams [1] proposed the diagonal algorithm that m-array is obtained by writing the corresponding maximal length sequence (m-sequence) down the main diagonal line of an $u \times v$ array and continuing from the opposite side whenever an edge is reached. Based on the diagonal algorithm, Lu [2] selected a square sub-array as the main body and spliced the remaining sub-windows to m-array through matching. Miao [3] tried to optimize the diagonal algorithm by central symmetrizing the sub-array.

Another classic pseudo-random array construction method is the perfect maps theory proposed by Etzion [4], which constructs m-array by shifting De Bruijn sequence. The first column of the m-array is a De Bruijn sequence, and other columns consist of cyclic shift of that sequence, with each column shifting one position in the sequence. Paterson [5], Mitchell [6], [7] and Ozturk [8] then conducted a more in-depth study on the perfect maps theory. Using the new definition of folding, Etzion [9] and Cui [10] constructed multi-dimensional pseudo-random arrays of various shapes, and further deduced the necessary and sufficient conditions for the existence of any given shape of folding.

Morano [11] proposed the piece-growing algorithm and pointed out that the m-array can be constructed more flexibly. All sub-windows are treated as independent piece resources, and the m-array is formed by splicing independent pieces together. Claes [12] proposed only testing the promising branches in the search tree, that is, when a piece cannot be spliced, only one element is backtracked at a time. Maurice [13] and Liang [14] then tried to optimize the piece-growing algorithm in different ways.

The classic m-array construction methods are mainly three types above, and other methods are constantly proposed [15]–[18].

A common problem of the diagonal algorithm and the perfect maps algorithm is that there is a fixed relation between the size of the given sub-window $m \times n$ and the size of the generated array $u \times v$, and the difference between u and v is significant especially when constructing large size m-array. The problem with the piece-growing algorithm is that due

The associate editor coordinating the review of this manuscript and approving it for publication was Michele Nappi.

to the high computational complexity, the execution time of the piece-growing algorithm increases and the success rate decreases dramatically with the increase of the array size. To solve these problems, a method based on non-zero maps theory is proposed in this paper. By dividing and splicing m-sequence according to specific rules, m-array with large array size and close number of rows and columns can be constructed in a short time.

In section II, the proposed non-zero maps theory is deduced and demonstrated. In section III, m-array based on non-zero maps theory is constructed. In section IV, the experimental results are displayed and the window property of m-array is verified. In section V, the method is compared with other m-array construction methods. Section VI summarizes the methods of this paper.

II. NON-ZERO MAPS THEORY

M-sequence is a typical pseudo-random sequence generated by linear feedback shift register (LFSR). It is also called the maximal-length shift-register sequence. A set of p-ary w-order linear feedback shift registers includes w boxes, each box has p symbols. The registers sequentially output backward to construct a cyclic sequence with period $T = p^w - 1$, that is, m-sequence [1]. The construction process is shown in Figure 1. The linear feedback shift register (LFSR) follows the fixed primitive polynomial in Galois field $GF(p^w)$:

$$h(x) = x^w + k_{w-1}x^{w-1} + \cdots + k_1x^1 + k_0 \tag{1}$$

where $0 \le k_i < p$, $k_i = 0, 1, 2, \ldots$ and $k_0 \ne 0$.

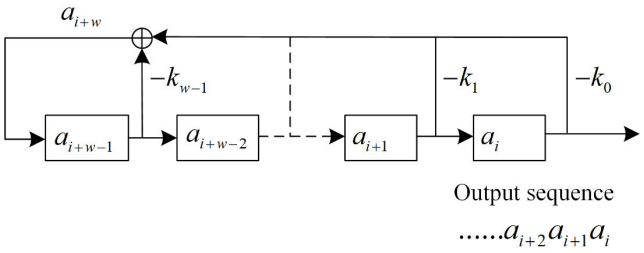

FIGURE 1. Linear feedback shift register, which sequentially outputs backward to construct M-sequence.

The feedback recursive equation describes the output sequence:

$$a_{i+w} = -k_{w-1}a_{i+w-1} - \cdots - k_1a_1 - k_0 \tag{2}$$

If the starting state is zero, all the state output by LFSR would be zero. If not, all the state output by LFSR would not be zero. Each non-zero state appears only once in a period of LFSR (suppose the starting state is not zero), $p^w - 1$ non-zero states correspond to $p^w - 1$ non-zero numbers in $GF(p^w)$.

A. SHIFT AND SPLICE

In this paper, T is used to express the period of m-sequence, A_{i+k} is defined as circularly shifting A_i to the left by k bits:

$$A_i = a_0a_1 \ldots a_{T-1}$$

$$A_{i+1} = a_0a_1 \ldots a_{T-1}\square = a_1 \ldots a_{T-1}a_0$$

$$\ldots$$

$$A_{i+k} = a_k \ldots a_{T-1}a_0 \ldots a_{k-1} \tag{3}$$

Taking $w = m \times n$, m-sequence with period $T = p^{mn} - 1$ is circularly shifted left by k and then spliced down one by one. Repeat this process until a two-dimensional array with size $m \times T$ is generated, as B_k in (4). Array B_k contains $T = p^{mn} - 1$ sub-windows with sub-window size $m \times n$ (Array B_k is a horizontal circular array).

$$B_k = \begin{pmatrix} A_i \\ A_{i+k} \\ \ldots \\ A_{i+(m-1)k} \end{pmatrix} = \begin{pmatrix} a_0a_1 \ldots\ldots a_{k-1}a_k \ldots\ldots\ldots a_{T-1} \\ a_ka_{k+1} \ldots a_{2k-1}a_{2k} \ldots a_{T-1}a_0 \ldots a_{k-1} \\ \ldots \\ a_{(m-1)k} \ldots a_{mk-1}a_{mk} \ldots\ldots\ldots a_{(m-1)k-1} \end{pmatrix} \tag{4}$$

Definition: The p-ary mn-order m-sequence is shifted and spliced according to the above process, with k as the shift number. Then a two-dimension array with size $m \times (p^{mn} - 1)$ is constructed which is called the transitional array, with $m \times n$ as the sub-window size. If each $m \times n$ sub-window in the transition array can uniquely correspond to a mn sub-sequence in m-sequence, the shift number k is called a positive factor.

When the shift number k is a positive factor, each sub-window in the transition array uniquely corresponds to a sub-sequence of the m-sequence, and each sub-sequence represents a state of LFSR. While LFSR is specified by the primitive polynomial in $GF(p^{mn})$, each state expresses a non-zero number in $GF(p^{mn})$ (the starting state is not zero). Such a one-to-one correspondence represents the maps of $GF(p^{mn})$ to the transition array, which is called non-zero maps.

1) TRANSITION ARRAY

By definition, k is an integer. Since the m-sequence is a periodic sequence, when $k \ge T$, k bits shift left is equal to $k - T$ bits shift left. When $k < 0$, k bits shift left is equal to $k + T$ bits shift left. Therefore, the actual value range of k is:

$$0 \le k < T, \quad k = 0, 1, 2, \cdots$$

In the process of non-zero maps, the shift number k is a crucial factor.

1) If the shift number $k = n$, the transition array is constructed as B_n in (5):

$$B_n = \begin{pmatrix} A_i \\ A_{i+n} \\ \ldots \\ A_{i+(m-1)n} \end{pmatrix} = \begin{pmatrix} a_0a_1 \ldots\ldots a_{n-1} \ldots\ldots\ldots a_{T-1} \\ a_na_{n+1} \ldots a_{2n-1} \ldots a_{T-1}a_0 \ldots a_{n-1} \\ \ldots \\ a_{(m-1)n} \ldots a_{mn-1} \ldots\ldots\ldots a_{(m-1)n-1} \end{pmatrix} \tag{5}$$

By connecting all rows end to end, each $m \times n$ sub-window of B_n can be unfolded to obtain a mn sub-sequence of the m-sequence in order. That is, each sub-window has a one-to-one correspondence with each sub-sequence in m-sequence. So $k = n$ is a positive factor.

2) If the shift number $0 \leq k < n$, in each $m \times n$ sub-window, the last $n-k$ elements of each row will be the same with the first $n-k$ elements of the following row. For example, one $m \times n$ sub-window of B_n in (4) is as follows:

$$\begin{pmatrix} a_0a_1 \ldots\ldots a_{k-1}a_ka_{k+1} \ldots\ldots a_{n-1} \\ a_ka_{k+1} \ldots a_{2k-1}a_{2k}a_{2k+1} \ldots a_{2n-1} \\ \ldots \\ a_{(m-1)k} \ldots a_{mk-1} \ldots\ldots\ldots a_{mn-1} \end{pmatrix}$$

The elements in each sub-window are partially repeated, resulting in the incorrect expression of some sub-window code-words and duplicate sub-windows in the transition array (Use a p-ary number defined by all elements of the sub-window to represent the sub-window code-word). This means that the process of non-zero maps of transforming the $GF(p^{mn})$ to the transition array fails, and the shift number k is not a positive factor.

3) If the shift number $n < k < T$, different values of k would cause different mapping results.

There is another case that the sub-window code-word being incorrectly expressed. Take the binary 4-order m-sequence as an example:

Example 1: Take $m = n = 2$, a binary 4-order m-sequence with period $T = 2^{mn} - 1 = 15$ is generated. When the shift number $k = 3 > n$, transition array with size 2×15 is constructed as D_3 in (6):

$$\begin{aligned} C_i &= a_0a_1a_2a_3 \ldots\ldots a_{14} \\ C_{i+3} &= a_3 \ldots\ldots a_{14}a_0a_1a_2 \\ D_3 = \begin{pmatrix} C_i \\ C_{i+3} \end{pmatrix} &= \begin{pmatrix} a_0a_1a_2 \ldots\ldots\ldots a_{14} \\ a_3a_4a_5 \ldots a_{14}a_0a_1a_2 \end{pmatrix} \end{aligned} \tag{6}$$

The primitive polynomial is:

$$f(x) = x^4 + x + 1 \tag{7}$$

The feedback recursive equation is:

$$a_{i+4} = -a_{i+1} - a_i \tag{8}$$

In (6), the last element of each 2×2 sub-window in array D_3 is defined by the first two elements. There are only three valid elements that express the sub-window code-word.

Lemma 1: If the quantity of valid elements expressing the sub-window code-word in each $m \times n$ sub-window of the transition array is smaller than mn, non-zero maps failed causing some of the numbers in $GF(p^{mn})$ are mapped incorrectly and others are mapped repeatedly.

Proof: Suppose that the quantity of valid elements in each $m \times n$ sub-window is $a (a \leq mn)$, these elements express p^a different sub-window code-words. The quantity of invalid elements in each sub-window is $mn - a$, which causes p^{mn-a} numbers in $GF(p^{mn})$ to be mapped to the same sub-window. The count of these repeated sub-windows in the transition array is p^{mn-a}.

TABLE 1. Distribution of sub-windows in transition array.

Type	Quantity [a]	Count [b]
non-zero sub-window	p^a-1	p^{mn-a}
zero sub-window	1	p^{mn-a}-1
total	p^{mn}-1	

[a]Quantity indicates the number of different sub-windows.
[b]Count refers to the number of times each sub-window appears in transition array.

For each sub-window of the transition array, there is m rows, each row is a set of n consecutive elements in m-sequence, and the interval between adjacent sets is k. The count of n consecutive 0s set in m-sequence is always one less than any other (This follows easily from that zero sub-sequence does not appear in m-sequence). The count of zero sub-window in transition array is one less than any non-zero window, is $p^{mn-a} - 1$. Table 1 shows the distribution of each sub-window in the transition array, meeting (9).

$$(p^a - 1)p^{mn-a} + p^{mn-a} - 1 = p^{mn} - 1 \tag{9}$$

Lemma 2: According to the count of zero sub-window in the transition array, whether the shift number k is a positive factor and whether the process of non-zero maps is successful would be clarified.

Proof: If the shift number k is a positive factor, the quantity of valid elements in each sub-window is $a = mn$, the count of zero sub-window is $p^{mn-a} - 1 = 0$, and the count of non-zero sub-window is $p^{mn-a} = 1$. If not, the quantity of valid elements is $a < mn$, the count of zero sub-window is $p^{mn-a} - 1 > 0$, and the count of non-zero sub-window is $p^{mn-a} > 1$.

The count of zero sub-window is 0 and the count of each non-zero sub-window is 1 in transition array, which is called the window property.

2) POSITIVE FACTOR

The run: the run is a maximal string of consecutive identical symbols. In each binary m-sequence, the number of runs of 0 is equal to the number of runs of 1 [1].

Whether the transition array contains zero sub-window depends on the runs of 0. For each p-ary mn-order m-sequence, if there are $x (x \geq m)$ sets of consecutive 0 with length $y (y \geq n)$ and k bits apart, the transition array constructed with k would contain zero sub-window, which means k is not a positive factor.

The properties of the run can be applied to calculate the positive factor. When a p-ary mn-order m-sequence is constructed, the positions of all the 0 runs are maintained, then all the positive factors can be calculated by traversing all

the 0 runs. In fact, the algorithm can be simplified without calculating all the positive factors, but only verifying the expected positive factor q (Use q to represent the expected positive factor) according to the position of each 0 run. Then, the best positive factor k close to q would be found in q area (k may be slightly larger or smaller than q).

B. ARRAY TRANSFORMATION

When the best positive factor k is found, the transition array is constructed by k with the numbers of rows and columns differing greatly.

In (10), taking k as the division factor, the transition array B_k in (4) with size $m \times T$ is divided into $h = [T/k]$ segments $s_1, s_2, s_3, \ldots, s_h$, each segment with size $m \times k$ partially overlaps with its neighbors. As T may not be an interal multiple of k, s_e represents the rest of the elements, with size $m \times (T - hk)$, where $k > T - hk \geq 0$. When the subscript value is greater than T, take the modulus of T.

$$\begin{aligned} B_{\mathrm{k}} &= s_1 + s_2 + s_3 + \ldots + s_h + s_e \\ &= \begin{pmatrix} a_0 a_1 \ldots\ldots a_{k-1} \\ a_k a_{k+1} \ldots a_{2k-1} \\ \ldots \\ a_{(m-1)k} \ldots a_{mk-1} \end{pmatrix} + \begin{pmatrix} a_k a_{k+1} \ldots a_{2k-1} \\ a_{2k} \ldots\ldots a_{3k-1} \\ \ldots \\ a_{mk} \ldots a_{(m+1)k-1} \end{pmatrix} + \ldots \\ &\quad + \begin{pmatrix} a_{(h-1)k} \ldots a_{hk-1} \\ a_{hk} \ldots a_{(h+1)k-1} \\ \ldots \\ a_{(h+m-2)k} \ldots \end{pmatrix} + s_e \end{aligned} \tag{10}$$

What can be found is that each two adjacent segments have the common $m-1$ rows, that is, the last $m-1$ rows of each segment are the same as the first $m-1$ rows of the following segment. Transition array can be transformed into a quasi m-array by splicing these h segments one by one vertically according to the overlap elements.

As shown in Figure 2, the transition array is divided into h segments $s_1, s_2, s_3, \ldots, s_h$, and a quasi m-array M_{quasi} with size $(h+m-1) \times k$ is constructed by splicing these segments one by one vertically (only the last segment is kept intact):

$$M_{quasi} = \begin{pmatrix} a_0 & a_1 & \ldots & a_{k-1} \\ a_k & a_{k+1} & \ldots & a_{2k-1} \\ a_{2k} & a_{2k+1} & \ldots & a_{3k-1} \\ \ldots & \ldots & \ldots & \ldots \\ a_{(h-1)k} & a_{(h-1)k+1} & \ldots & a_{hk-1} \\ a_{hk} & a_{hk+1} & \ldots & a_{(h+1)k-1} \\ \ldots & \ldots & \ldots & \ldots \\ a_{(h+m-2)k} & a_{(h+m-2)k+1} & \ldots & a_{(h+m-1)k-1} \end{pmatrix} \tag{11}$$

In (11), when the subscript value is greater than T, take the modulus of T. M_{quasi} retains the window property since all sub-windows originate from the transition array.

III. M-ARRAY BASED ON NON-ZERO MAPS THORY

As proposed in section II, with given symbols p and sub-window size $m \times n$, m-sequence is generated first and the positive factor k is calculated next. Then a quasi m-array would be constructed by constructing and transforming the transition array.

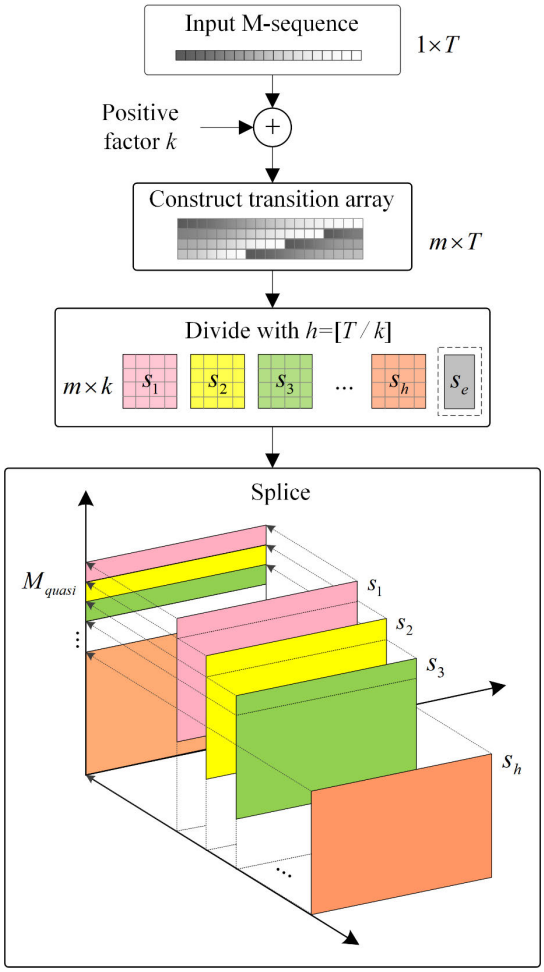

FIGURE 2. Construction of quasi m-array M_{quasi}. Taking the positive factor k as the division factor, transition array is divided and spliced to obtain quasi m-array M_{quasi}. M_{quasi} with size $(h + m - 1) \times k$ is described as (11).

A. BASE ARRAY CONSTRUCTION

What can be found is that the process can be simplified without explicitly constructing and transforming the transition array but dividing and splicing m-sequence directly, as shown in Figure 3. This follows easily from that before and after replacement, almost the same quasi m-array is obtained.

In (12), taking the positive factor k as a division factor, m-sequence A_i in (3) is divided into $h = [T/k]$ segments $t_1, t_2, t, \ldots, t_h$. As the period of m-sequence T may not be an interal multiple of k, t_e represents the rest of the elements, with length $T - hk$, where $k > T - hk \geq 0$.

$$\begin{aligned} A_i &= t_1 + t_2 + t_3 + \ldots + t_h + t_e \\ &= (a_0 a_1 \ldots a_{k-1}) + (a_k a_{k+1} \ldots a_{2k-1}) + \ldots \\ &\quad + \left(a_{(h-1)k} \ldots a_{hk-1}\right) + t_e \end{aligned} \tag{12}$$

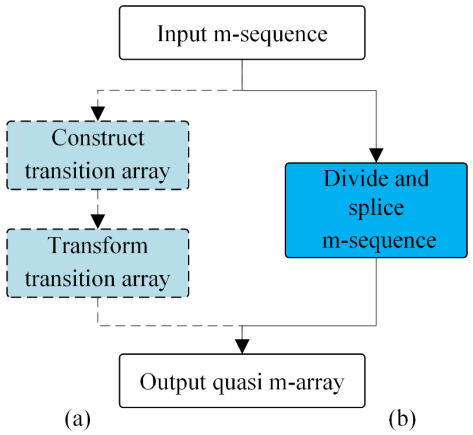

FIGURE 3. Quasi m-array construction. (a) Construction method of quasi m-array introduced in section II. (b) The process of constructing and transforming transition array is replaced by dividing and splicing m-sequence directly.

As shown in Figure 4, another quasi m-array M_{base} with size $h \times k$ is constructed by splicing these segments one by one vertically, as (14). M_{base} is named because m-array is constructed based on M_{base}.

$$M_{base} = \begin{pmatrix} a_0 & a_1 & \dots & a_{k-1} \\ a_k & a_{k+1} & \dots & a_{2k-1} \\ a_{2k} & a_{2k+1} & \dots & a_{3k-1} \\ \dots & \dots & \dots & \dots \\ a_{(h-1)k} & a_{(h-1)k+1} & \dots & a_{hk-1} \end{pmatrix} \tag{13}$$

There are minor differences between quasi array M_{quasi} in (11) and M_{base} in (13) (M_{quasi} contains more $m-1$ rows). Treating M_{base} as a base array, m-array with larger size can be constructed by adding sub-windows horizontally and vertically to M_{base}.

B. BASE ARRAY EXPANSION

The transition array is a collection of $m \times n$ sub-windows, including all non-zero sub-windows, and each of them appears once (The transition array is a circular array). M_{base} in Figure 4 contains most of the sub-windows, but not all.

Without explicitly constructing transition array, but dividing and splicing m-sequence directly, sub-windows in the last segment of transition array are abandoned. As shown in Figure 2 and Figure 4, M_{quasi} in (11) contains more $m-1$ rows than M_{base} in (13). In Figure 5 ①, the extra $m-1$ rows with the same form are found and added following the original h rows, with all elements are continuous according to m-sequence (m-sequence is a circular sequence). The number of rows reaches $h+m-1$, as Figure 5(c) M_{vert}.

The process of array transformation results in the abandonment of the sub-windows connecting two adjacent segments. As shown in (10), (11) and (13), quasi m-array M_{quasi} and M_{base} does not contain the sub-windows connecting two adjacent segments. In Figure 5 ②, the length of each row is extended to $k+n-1$ by appending extra $n-1$ elements along m-sequence, as Figure 5(d) M_{hori}.

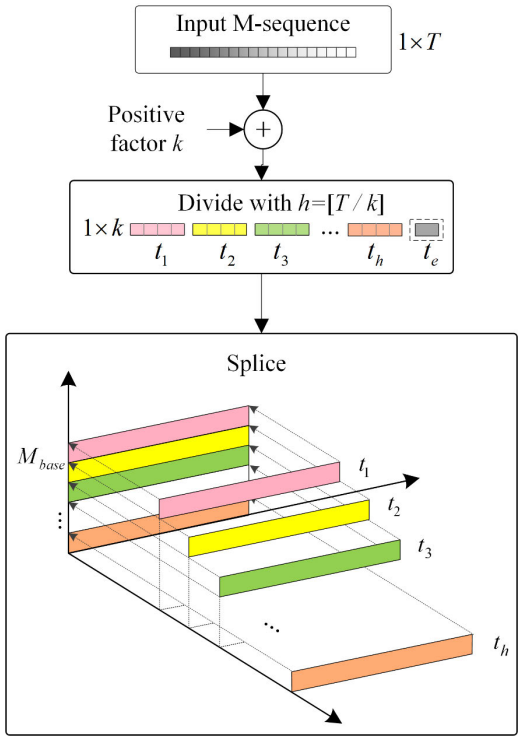

FIGURE 4. Construction of quasi m-array M_{base}. m-sequence is divided and spliced one by one to obtain M_{base}.

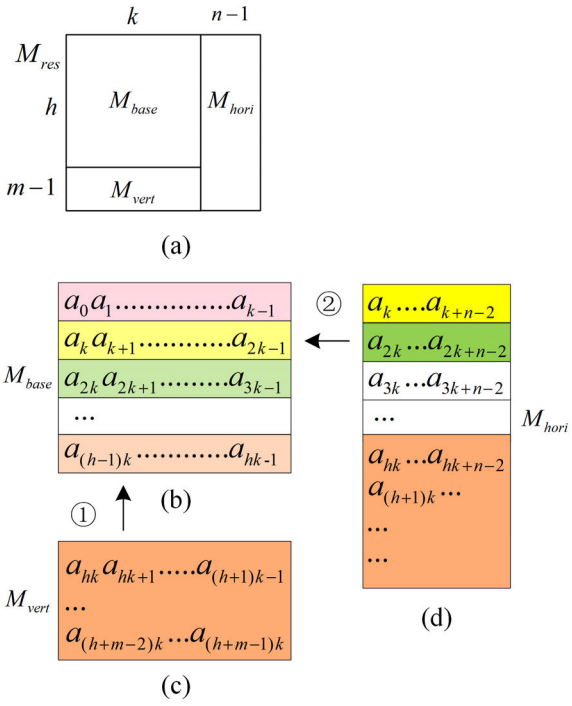

FIGURE 5. Expansion method. ① Append vertically. ② Append horizontally. When the subscript value is greater than T, take the modulus of T.

The final m-array M_{res} as Figure 5(a) is described as:

$$M_{res} = \begin{pmatrix} M_{base} & M_{hori} \\ M_{vert} & \end{pmatrix} \tag{14}$$

As shown in Figure 5, m-array is expanded vertically and horizontally to contain as many sub-windows as possible, with size $(h + m - 1) \times (k + n - 1)$. That is, for m-array $M_{res}(u, v; m, n; p)$, $u = (h + m - 1)$, $v = (k + n - 1)$.

C. CONSTRUCTION PROCESS

Non-zero maps theory constructs m-array $U(u, v; m, n; p)$ with inputting symbols p, sub-windows size $m \times n$ and expected ratio of columns and rows.

The complete construction process of m-array is shown in Figure 6.

Step 1: According to given symbols p and sub-window size $m \times n$, m-sequence with period $T = p^{mn} - 1$ is generated.

Step 2: Take q as the expected positive factor. According to the property of the run, the best division factor k would be found in q area (k may be slightly larger or smaller than q).

Step 3: Base array is constructed by dividing and splicing m-sequence, the details are shown in Figure 4.

Step 4: Complete m-array is constructed by base array expansion.

The expected positive factor q depends on the expected ratio of columns and rows. For example: When the expected ratio entered as 1:1, then $q = [\sqrt{T}]$, k is found in q area, and $h = [T/k]$ is also close to $\sqrt{T}$, an m-array close to a square is constructed by the process above; When the expected ratio entered as 16:9, then q can be taken as $[\frac{3}{4}\sqrt{T}]$, k in q area and $h = [T/k]$. Since k is close to $\frac{3}{4}\sqrt{T}$, h is close to $\frac{4}{3}\sqrt{T}$, an m-array with the ratio of rows and columns close to $16 : 9$ would be constructed.

IV. WINDOW PROPERTY VERIFICATION

In this section, an experiment is performed to verify the window property of the m-array constructed by the proposed method.

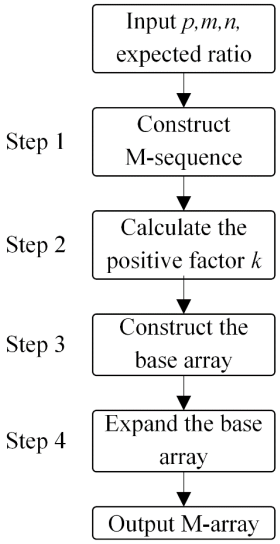

FIGURE 6. The complete construction process of m-array.

A. EXPERIMENTAL RESULTS

Some m-arrays constructed by non-zero maps theory under different conditions are shown as Figure 7 (Take m-array close to a square as a typical case).

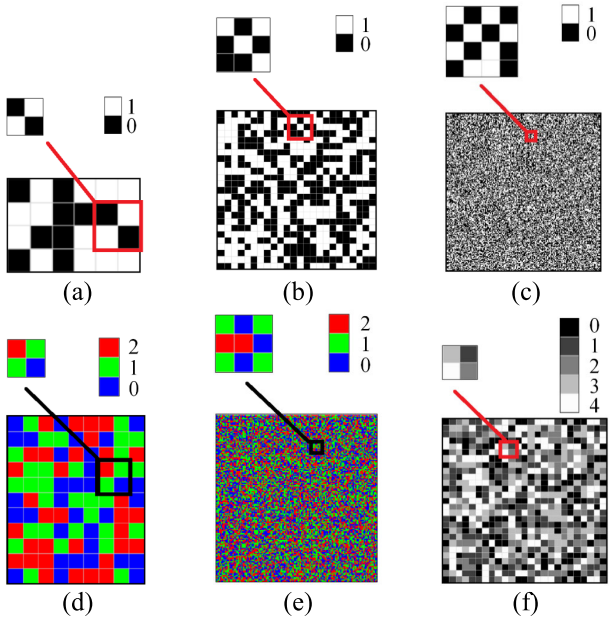

FIGURE 7. M-array under different conditions. (a) 4 × 6 m-array with sub-window size 2 × 2. (b) 25 × 24 m-array with sub-window size 3 × 3. (c) 260 × 258 m-array with sub-window size 4 × 4. (d) 11 × 9 m-array with sub-window size 2 × 2. (e) 140 × 144 m-array with sub-window size 3 × 3. (f) 27 × 25 m-array with sub-window size 2 × 2.

Take symbols $p = 2$, using a white grid to represent symbol 1, and a black grid to represent symbol 0. In Figure 7(a), given sub-window size $m \times n = 2 \times 2$, the period of m-sequence $T = 2^{mn} - 1 = 15$, division factor $k = 5$, and the number of segments $h = 3$, m-array with size 4×6 is constructed. In Figure 7(b), given sub-window size $m \times n = 3 \times 3$, $T = 511$, $k = 22$, $h = 23$, m-array size is 25×24. In Figure 7(c), given sub-window size $m \times n = 4 \times 4$, $T = 65535$, $k = 255$, $h = 257$, m-array size is 260×258.

Take symbols $p = 3$, using red, green and blue grids to represent symbol 2, 1 and 0. In Figure 7(d), given sub-window size $m \times n = 2 \times 2$, $T = 3^{mn} - 1 = 80$, $k = 8$, $h = 10$, m-array size is 11×9. In Figure 7(e), given sub-window size $m \times n = 3 \times 3$, $T = 3^{mn} - 1 = 19682$, $k = 142$, $h = 138$, m-array size is 140×144.

Take symbols $p = 5$, in Figure 7(f), sub-window size $m \times n = 2 \times 2$, $T = 5^{mn} - 1 = 624$, $k = 24$, $h = 26$, m-array size is 27×25.

B. WINDOW PROPERTY VERIFICATION

The window property of m-array proven in section II is described as each $m \times n$ sub-window appears once in an $u \times v$ m-array $U(u, v; m, n; p)$. In fact, each sub-window $m' \times n'$ with $m' \geq m$ and $n' \geq n$ is also globally unique in m-arrays.

So, the window property can be verified by checking whether each $m \times n$ sub-window is globally unique.

To verify the window property of m-array constructed by non-zero maps theory, the count of each sub-window in m-array is counted and displayed with a histogram. Whenever any sub-window appears more than once, the m-array is considered not meeting window property.

A decimal number is used to represent the code-word of each sub-window, for example, sub-window $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ (symbols $p = 2$) is decoded as 0111 in binary and 7 in decimal; sub-window $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ (symbols $p = 3$) is decoded as 9 in decimal. By scanning the m-array and recording the count of each sub-window, the pseudo-code for verifying the window property of m-array is as follows.

Algorithm 1 Verify the Window Property of a m-Array

Input: *mArray*: m-array; *m*: sub-window rows; *n*: sub-window columns; *p*: m-array symbols.
Output: true or false

```
function windowPropertyVerify (mArray, m, n, p)
    create array histogram[p^mn] ← 0
    for each sub-window in mArray
        do compute codeword
            histogram [codeword] ← histogram
              [codeword] + 1
    for each codeword in histogram
        do if histogram [codeword] > 1
            then return false
return true
```

The window property of m-arrays in Figure 7 are all verified by the algorithm in the pseudo-code above. For easy display, take the two smaller m-arrays Figure 7(a), (d) as an example, Figure 8(a), (b) shows the decoding array, and Figure 8(c), (d) shows the histogram.

In Figure 8(a), (b), sub-window code-word is decoded and displayed in the center of each 2×2 sub-window, the decoding array shows that each sub-window is globally unique and seems to be random. In Figure 8(c), (d), each non-zero sub-window code-word is decoded to appear once in m-array, with zero sub-window do not appear, which verifies the conclusion drawn in section II.

V. COMPARED WITH OTHER ALGORITHMS

In this section, the method in this paper is mainly compared with three existing algorithms. Formula-Method-Based encoding algorithms include the diagonal algorithm [1] and perfect maps algorithm [4]. Enumeration-Method-Based Encoding Algorithm refers to the piece-growing algorithm [11]. These three classic algorithms are the most representative method of its types, are widely cited and referenced. Other related methods are mostly based on these methods with minor changes to adapt to specific applications, refer to [2], [3], [12]–[14] *et al.*

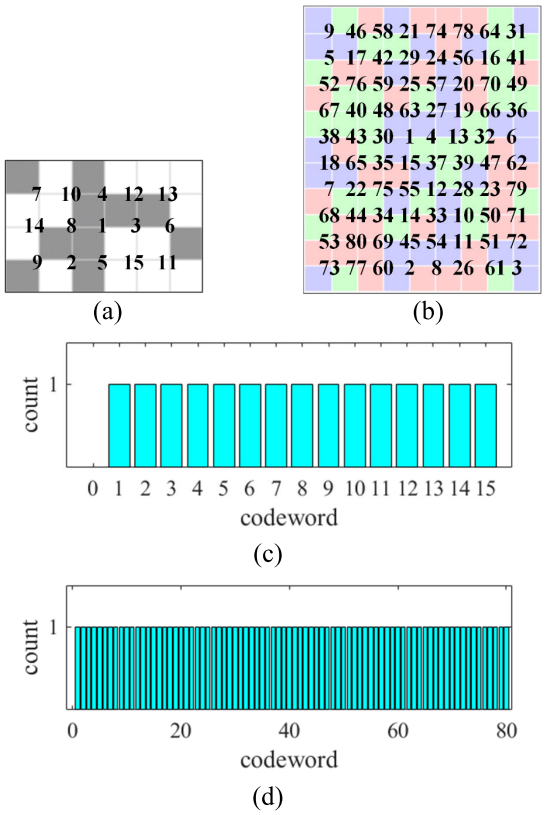

FIGURE 8. Window property verification. (a) sub-window size 2 × 2, symbols p = 2, m-array decoding. (b) sub-window size 2 × 2, symbols p = 3, m-array decoding. (c) histogram of (a). (d) histogram of (b).

A. BALANCED NUMBER OF ROWS AND COLUMNS

The diagonal algorithm [1] constructs m-array by folding an m-sequence. For an m-sequence with period $T = p^{mn} - 1$, the size of m-array $u \times v$ are $u = p^m - 1$ and $v = T/u$, where u and v are relatively prime. For example,

for $p = 2, m = 2, n = 2, T = 15$, then $u = 3, v = 5$;
for $p = 2, m = 3, n = 3, T = 511$, then $u = 7, v = 73$.

M-array is obtained by writing the m-sequence down the main diagonal line of an $u \times v$ array and continuing from the opposite side whenever an edge is reached. The diagonal algorithm is restricted because the m-array size depends on the prime factor decomposition of the period of the corresponding m-sequences [15].

Let R_1 be the ratio of columns and rows,

$$R_1 = \frac{v}{u} = \frac{p^{mn} - 1}{(p^m - 1)^2} > \frac{p^{mn} - 1}{p^{2m}} = p^{m(n-2)} - \frac{1}{p^{2m}} \approx p^{m(n-2)} \tag{15}$$

Perfect maps algorithm [4] constructed m-array by shifting De Bruijn sequence. The first column of the m-array is a De Bruijn sequence with length $u = p^m$, and other columns consist of cyclic shift of that sequence. Another De Bruijn sequence with size $v = u^{n-1}$ is obtained to difine the number of bits shifted of each column. For example,

for $p = 2, m = 2, n = 2$, then $u = 4, v = 4$;
for $p = 2, m = 3, n = 3$, then $u = 8, v = 64$.

TABLE 2. Compared with formula-method-based encoding algorithm.

Method	Symbols	Sub-window Size	M-array Size	CR-ratio [a]	size-ratio [b]
The Diagonal Algorithm	2	2×2	3×5	0.222	0.500
		3×3	7×73	1.018	0.693
		3×4	7×585	1.922	0.710
		4×4	15×4369	2.467	0.799
		5×4	31×33825	3.038	0.871
		5×5	31×1082401	4.858	0.871
	3	3×3	26×757	1.464	0.921
	5	3×3	124×15751	2.104	0.984
Perfect Maps Algorithm	2	2×2	4×4	0	0.562
		3×3	8×64	0.903	0.727
		3×4	8×512	1.806	0.746
		4×4	16×4096	2.408	0.812
		5×4	32×32768	3.010	0.875
		5×5	32×1048576	4.515	0.875
	3	3×3	27×729	1.431	0.923
	5	3×3	125×15625	2.097	0.984
Method in this paper	2	2×2	4×6	0.176	0.937
		3×3	25×24	0.018	0.988
		3×4	65×68	0.020	>0.999
		4×4	260×258	0.003	>0.999
		5×4	1027×1027	0	>0.999
		5×5	5798×5795	<0.001	>0.999
	3	3×3	140×144	0.012	0.996
	5	3×3	1400×1399	<0.001	>0.999

[a]CR-ratio is the ratio of the columns and rows of m-array, and the result is first converted to logarithm, then expressed in absolute value.
[b]size-ratio refers to the ratio of sub-windows in m-array and the total number of sub-windows, the total number of sub-windows is calculated to be p^{mn}.

Let R_2 be the ratio of columns and rows,

$$R_2 = \frac{v}{u} = \frac{u^{(n-1)}}{u} = u^{(n-2)} = p^{m(n-2)} \tag{16}$$

Non-zero maps theory constructs m-array $U(u, v; m, n; p)$ with $u = (h + m - 1)$, $v = (k + n - 1)$, when inputting symbols p, sub-window size $m \times n$ and expected ratio, as shown in Figure 6. Let R_i be the ratio of columns and rows,

$$R_i = \frac{v}{u} = \frac{k + n - 1}{h + m - 1} \tag{17}$$

When the expected ratio entered as 1:1, k and $h = [T/k]$ are close to $\sqrt{T}$. With the increasing of m, n or p, the influence of m, n on R_i decreasing, so $R_i \to 1$.

As R_1 and R_2 are power functions about p, exponential functions about m, n, the ratio of columns and rows of m-array constructed by the diagonal algorithm or perfect maps algorithm increase exponentially with the increase of symbols p and sub-window size m, n. Piece-growing algorithm constructs m-array with inputting $u \times v$, so it is not compared about the ratio of columns and rows.

A CR-ratio is calculated to show the differences on the ratio of columns and rows of m-array, detailed comparison data is shown in Table 2 (The method in this paper takes the construction of m-array close to a square as a typical case). The diagonal algorithm is restricted because m-array size depends on the prime factor decomposition of the period of the corresponding m-sequences [15], so it is not applicable to some inputs. Within the limits of the diagonal algorithm, different algorithms are compared through experiments when inputting different symbols p and sub-window size $m \times n$. The data is shown in Table 2.

According to the data in Table 2, the comparison of these three methods regarding CR-ratio is shown in Figure 9.

In Figure 9(a), the CR-ratio of the m-array constructed by the diagonal algorithm or the perfect maps algorithm increases exponentially, while the method in this paper always maintains around 0, and the larger sub-window size, the more stable the CR-ratio is. In Figure 9(b), the CR-ratio of the m-array constructed by the diagonal algorithm or the perfect maps algorithm increases significantly as the symbols increase, and is much higher than the method in this paper.

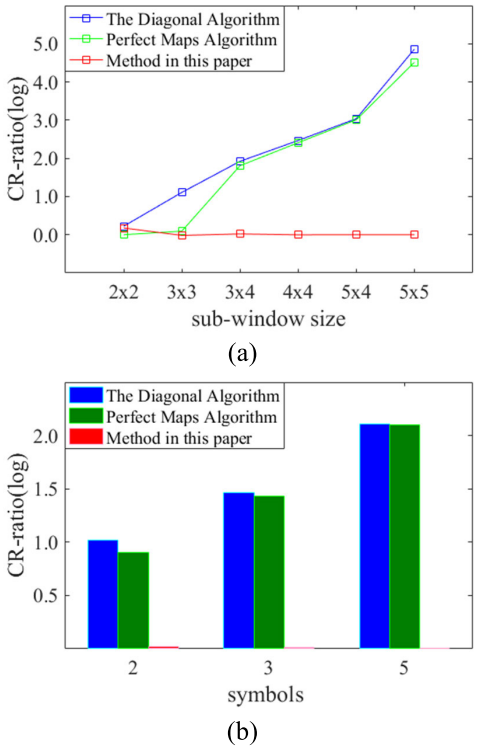

FIGURE 9. Comparison of CR-ratio. (a) Symbols p = 2 is fixed, the abscissa indicates different sub-window sizes. (b) Sub-window size m × n = 3 × 3 is fixed, the abscissa indicates different symbols.

It can be seen that an evident advantage of the method in this paper is that the ratio of columns and rows of the m-array is much more balanced than the other two methods.

B. FLEXIBLE RATIO OF ROWS AND COLUMNS

According to formula (15) and (16), R_1 and R_2 are fixed with given p, m, and n. So, when the symbol p and sub-window size $m \times n$ are given, the size of the m-arrays constructed by the diagonal algorithm or perfect maps algorithm are always fixed, and the shape are always strip-shaped. Figure 10 shows the differences on the ratio of columns and rows between these three algorithms. The diagonal algorithm and perfect maps algorithm construct m-array with fixed formula and predictable ratio of columns and rows. The method in this paper constructs m-arrays of different sizes, the selection of m-array size $u_i \times v_i$ depends on input expected ratio.

In fact, when constructing m-array, the method in this paper first calculates the positive factor k according to input expected ratio, and then the array size $u_i \times v_i$ is determined. For instance, take symbols $p = 2$, sub-window size $m \times n = 3 \times 3$, m-array constructed by the diagonal algorithm is shown as Figure 11(a), with size 7×73. M-array constructed by perfect maps algorithm is shown as Figure 11(b), with size 8×64.

Three examples of m-array constructed by the method in this paper is shown as Figure 11(c), (d), (e). m-sequence with period $T = p^{mn} - 1 = 2^{3\times3} - 1 = 511$ is constructed first.

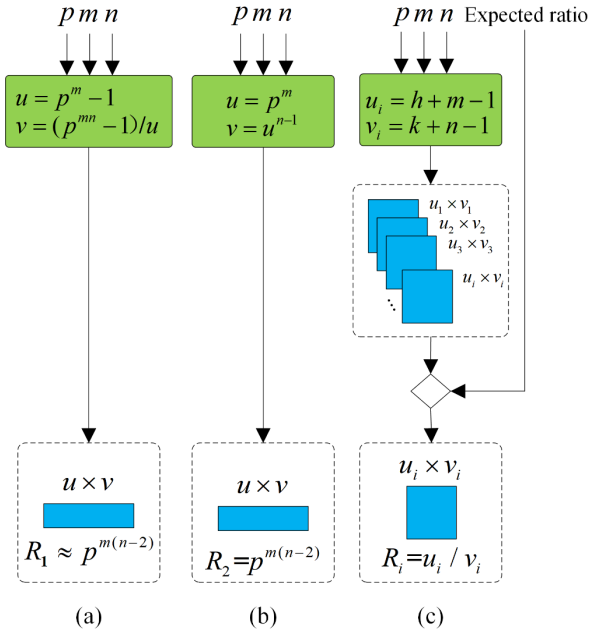

FIGURE 10. Comparison of construction formulas. (a) The diagonal algorithm. (b) Perfect maps algorithm. (c) The method in this paper.

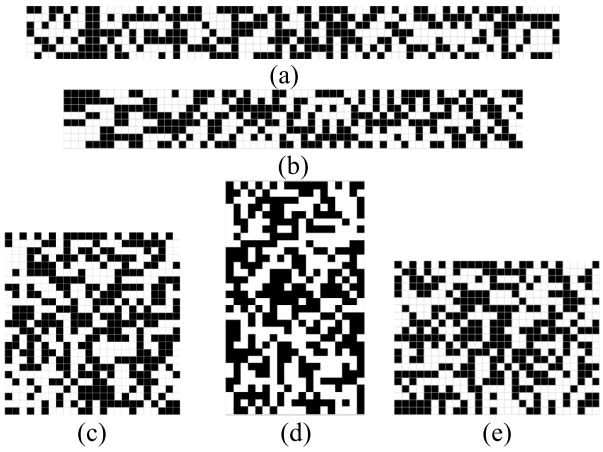

FIGURE 11. M-array by different methods when symbols p = 2 and sub-windows size 3 × 3. (a) 7 × 73 m-array constructed by the diagonal algorithm. (b) 8 × 64 m-array constructed by perfect maps algorithm. (c) 25 × 24 m-array constructed by the method in this paper. (d) 32 × 19 m-array constructed by the method in this paper. (e) 21 × 28 m-array constructed by the method in this paper.

In Figure 11(c), inputting expected ratio as 1 : 1, then $q = \left\lceil \sqrt{T} \right\rceil = 22$, the division factor k is calculated to be 23, $h = [T/k] = 22$, then m-array with size 25×24 is obtained as $u = h + m - 1$ and $v = k + n - 1$.

In Figure 11(d), inputting expected ratio as 16 : 9, then $q = \left\lceil \frac{3}{4}\sqrt{T} \right\rceil = 16$, the division factor k is calculated to be 17, $h = [T/k] = 30$, then m-array with size 32×19 is obtained.

In Figure 11(e), inputting expected ratio as 3 : 4, then $q = \left\lceil \frac{2}{\sqrt{3}}\sqrt{T} \right\rceil = 26$, the division factor k is calculated to be 26, $h = [T/k] = 19$, then m-array with size 21×28 is obtained.

The method in this paper is flexible enough that the size of m-array can be adjusted. M-array with specific expected ratio of rows and columns can be constructed by selecting the most suitable positive factor.

C. LARGER M-ARRAY SIZE

A size-ratio is calculated to compare the m-array size, which is defined as the ratio of sub-windows in m-array and the total number of sub-windows. Higher size-ratio means more sub-windows are contained in m-array, representing larger m-array size. According to the data in Table 2, the comparison of these three methods regarding size-ratio is shown in Figure 12.

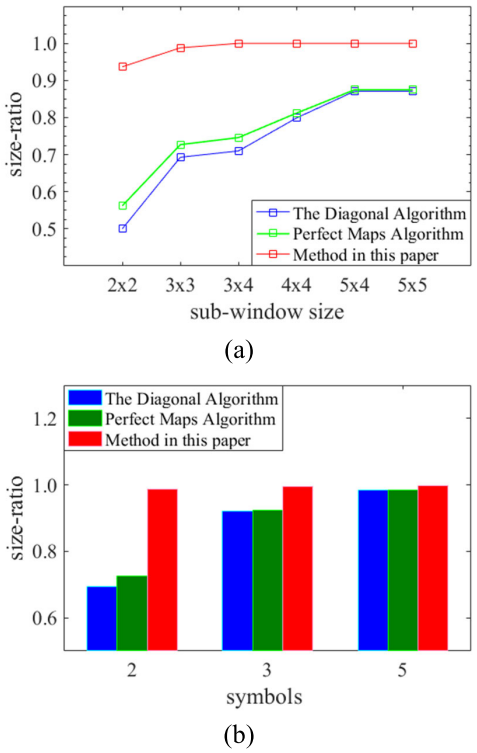

FIGURE 12. Comparison of size-ratio. (a) Symbols p = 2 is fixed, the abscissa indicates different sub-window sizes. (b) Sub-window size m × n = 3 × 3 is fixed, the abscissa indicates different symbols.

In Figure 12(a), size-ratio of these three methods gradually increases with sub-window size, the method in this paper is always close to 1 and gradually stabilizes, and the method in this paper shows the highest size-ratio. In Figure 12(b), size-ratio of these three methods increases with symbols, the method in this paper shows the highest size-ratio.

With higher size-ratio, more sub-windows are contained, the method in this paper constructs m-array with larger size than the diagonal algorithm and perfect maps algorithm.

Piece-growing algorithm is a flexible method that m-array size $u \times v$ is predetermined as input. Based on piece matching, piece-growing algorithm constructs m-array with high computational complexity. M-array is formed by splicing independent pieces together, and each sub-window in m-array is obtained by matching independent pieces until the window properties are followed. Once any one of the sub-windows cannot be spliced, the algorithm would backtrack. According to the method in this paper, m-sequence is generated first and the positive factor k is calculated next, then m-array would be generated at one time. So, the method in this paper has lower computational complexity.

TABLE 3. Compared with enumeration-method-based encoding algorithm.

Method	Sub-window size	M-array size
Morano [11]	4×4	45×45
Albitar [17]	3×3	27×29
Liang [14]	3×3	42×42
Lu [2]	3×3	48×52
Method in this paper	4×4	6565×6563
	3×3	140×144

Due to high computational complexity and random factors when matching independent pieces, the piece-growing algorithm always fails to construct large-size m-array. Take symbols $p = 3$, Morano [11] showed that to construct an m-array with size 45×45, the sub-window size needs to be at least 4×4. When the sub-window size is fixed with 3×3, Albitar [17] constructed a 27×29 m-array, Liang [14] constructed a 42×42 m-array. Lu [2] combined the diagonal algorithm with the piece-growing algorithm to construct a 48×52 m-array.

As shown in Table 3, take symbols $p = 3$, the method in this paper constructs 6565×6563 m-array when sub-window size is 4×4, 140×144 m-array when sub-window size is 3×3. The method in this paper constructs m-array much larger than piece-growing algorithm.

VI. CONCLUSION

Compared with the diagonal algorithm and perfect maps algorithm, the method in this paper construct m-array with smaller CR-ratio and larger size-ratio. Smaller CR-ratio means closer rows and columns, larger size-ratio represents larger array size, more sub-windows are included in m-array. More importantly, the method in this paper is flexible enough that the rows and columns of m-array can be adjusted within a certain range. M-array with specific aspect ratio can be constructed by selecting the most suitable division factor.

The method in this paper has lower computational complexity and can construct m-array much larger than piece-growing algorithm. When the sub-window size is fixed, the method in this paper constructs m-array with larger array size. When constructing m-array of the same size, the method in this paper requires fewer symbols or smaller sub-window size.

M-array is widely used in three-dimensional reconstruction, two-dimensional measurement, and many other fields. M-array with large array size and close number of rows and columns can be constructed in a short time, the method proposed in this paper has practical value.

REFERENCES

[1] F. J. MacWilliams and N. J. A. Sloane, "Pseudo-random sequences and arrays," *Proc. IEEE*, vol. 64, no. 12, pp. 1715–1729, Dec. 1976, doi: 10.1109/PROC.1976.10411.

[2] J. Lu, J. Han, E. Ahsan, G. Xia, and Q. Xu, "A structured light vision measurement with large size M-array for dynamic scenes," in *Proc. 35th Chin. Control Conf. (CCC)*, Chengdu, China, 2016, pp. 3834–3839, doi: 10.1109/ChiCC.2016.7553951.

[3] Y. Miao, Y. Zhao, H. Ma, M. Jiang, J. Lin, and P. Jin, "Design of diffractive optical element projector for a pseudorandom dot array by an improved encoding method," *Appl. Opt.*, vol. 58, no. 34, pp. G169–G176, Dec. 2019.

[4] T. Etzion, "Constructions for perfect maps and pseudorandom arrays," *IEEE Trans. Inf. Theory*, vol. IT-34, no. 5, pp. 1308–1316, Sep. 1988, doi: 10.1109/18.21260.

[5] K. G. Paterson, "Perfect maps," in *Proc. IEEE Int. Symp. Inf. Theory*, San Antonio, TX, USA, Jan. 1993, p. 408, doi: 10.1109/ISIT.1993.748724.

[6] C. J. Mitchell and K. G. Paterson, "Decoding perfect maps," *Des., Codes Cryptogr.*, vol. 4, pp. 11–30, Jan. 1994.

[7] C. J. Mitchell, "Aperiodic and semi-periodic perfect maps," *IEEE Trans. Inf. Theory*, vol. 41, no. 1, pp. 88–95, Jan. 1995, doi: 10.1109/18.370116.

[8] C. Ozturk, J. Nissanov, and S. Dubin, "Generation of perfect map codes for an active stereo imaging system," in *Proc. IEEE 22nd Annu. Northeast Bioeng. Conf.*, New Brunswick, NJ, USA, Mar. 1996, pp. 76–77, doi: 10.1109/NEBC.1996.503225.

[9] T. Etzion, "Sequence folding, lattice tiling, and multidimensional coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4383–4400, Jul. 2011, doi: 10.1109/TIT.2011.2146010.

[10] J. Cui, J. Pei, and P. Yang, "Existence of a folding in multidimensional coding," *Discrete Math.*, vol. 326, pp. 4–8, Jul. 2014, doi: 10.1016/j.disc.2014.02.019.

[11] R. A. Morano, C. Ozturk, R. Conn, S. Dubin, S. Zietz, and J. Nissano, "Structured light using pseudorandom codes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 3, pp. 322–327, Mar. 1998, doi: 10.1109/34.667888.

[12] K. Claes, "Structured light adapted to control a robot arm," Ph.D. dissertation, KU Leuven, Leuven, Belgium, 2008.

[13] X. Maurice, P. Graebling, and C. Doignon, "A pattern framework driven by the Hamming distance for structured light-based reconstruction with a single image," in *Proc. CVPR*, Providence, RI, USA, Jun. 2011, pp. 2497–2504, doi: 10.1109/CVPR.2011.5995490.

[14] Z. Liang, Y. Yu, and H. Xue, "A structured light encoding method for M-array technique," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dali, China, Dec. 2019, pp. 415–421, doi: 10.1109/ROBIO49542.2019.8961565.

[15] C. J. Kuo and H. B. Rigas, "2-D quasi m-arrays and gold code arrays," *IEEE Trans. Inf. Theory*, vol. 37, no. 2, pp. 385–388, Mar. 1991, doi: 10.1109/18.75260.

[16] A. M. Bruckstein, T. Etzion, R. Giryes, N. Gordon, R. J. Holt, and D. Shuldiner, "Simple and robust binary self-location patterns," *IEEE Trans. Inf. Theory*, vol. 58, no. 7, pp. 4884–4889, Jul. 2012, doi: 10.1109/TIT.2012.2191699.

[17] C. Albitar, P. Graebling, and C. Doignon, "Design of a monochromatic pattern for a robust structured light coding," in *Proc. IEEE Int. Conf. Image Process.*, San Antonio, TX, USA, Sep./Oct. 2007, pp. VI-529-VI-532, doi: 10.1109/ICIP.2007.4379638.

[18] A. Elahi, J. Lu, Q.-D. Zhu, and L. Yong, "A single-shot, pixel encoded 3D measurement technique for structure light," *IEEE Access*, vol. 8, pp. 127254–127271, 2020, doi: 10.1109/ACCESS.2020.3009025.

XIAO ZHOU received the Ph.D. degree in pattern recognition and intelligent system from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2009. From 2010 to 2011, he did his postdoctoral training with the School of Computer Science and Technology, HUST. In 2017, he was a Visiting Scholar with PATH, University of California at Berkeley, where he was a Visiting Associate Research Engineer, in 2018. He is currently an Associate Professor with the School of Mechanical and Electrical Engineering, Wuhan University of Technology, Wuhan. His teaching and research interests include machine vision and intelligent systems.

YU KANG received the B.S. degree from the Wuhan University of Technology, Wuhan, Hubei, China, in 2018, where he is currently pursuing the master's degree. His research interests include image processing and machine vision.

TINGTING ZHANG received the B.S. degree from Sun Yat-sen University, Guangdong, China, in 2009. She worked as a Researcher for companies, such as General Motors, China. From 2017 to 2020, she worked as an Associate Specialist with PATH, University of California at Berkeley, USA.

XINGANG MOU received the B.S. degree in communication engineering and the Ph.D. degree in control science and engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2001 and 2010, respectively. He is an Associate Professor with the School of Mechanical and Electrical Engineering, Wuhan University of Technology. His research interests include image processing and parallel computing.