CHAPTER 3 PROPOSED METHOD

This chapter describes about the adaptive algorithm. It was introduced to use in the retrieval process to enhance the exiting image watermarking scheme [8]. Also, the development of adaptive algorithm was described.

3.1 Adaptive Algorithm Version 1

To analyze the factors that influence the watermark retrieval performance, the equation (2.3) is rewritten by

$$\begin{aligned} w'(i,j) &= B(i,j) + w(i,j)sL'(i,j) \\ &- \frac{1}{8}\Big(\sum_{m=-1}^{1}\sum_{n=-1}^{1} B(i+m,j+n) - B(m,n)\Big) \\ &- \frac{1}{8}\Big(\sum_{m=-1}^{1}\sum_{n=-1}^{1} w(i+m,j+n)sL'(i+m,j+n) - w(m,n)sL'(m,n)\Big) \end{aligned} \tag{3.1}$$

Reasonably, from the equation (2.3), it is that large variation in pixels around *(i, j)* can cause a bias in the original pixel prediction process, which results in an erroneous original pixel value. However, we can minimize this bias by removing all surrounding pixels around *(i, j)* that most differs from the remaining. For instance, in case of having two extreme pixel values within the eight surrounding pixels, those two pixels should be removed from the prediction process. Based on the first assumption previously made, the prediction accuracy of an original pixel depends mainly on its nearby pixel values. In other words, it can be said that a small difference in variance (σ^2) between the original surrounding pixels and the watermarked surrounding pixels gives a higher accuracy of the original pixel prediction than a large one. Therefore, if the variance of eight pixels around $B'(i, j)$, $\sigma_{B'}^{\ 2}$, is much more different than that of eight pixels around

B(i, j), σ_B^2, one pixel around *B'(i, j)* that affects the variance most should be removed in order to make the new variance close to σ_B^2. Based on this concept, σ_B^2 is thus referred to as a prediction threshold in the original pixel prediction process. Since in the watermark retrieval, the blind detection is considered and the real value of σ_B^2 cannot be determined. In this thesis, we hence used the variance of the watermarked image as σ_B^2. It should be noted that in nature the variance of the entire image is usually much higher than the variance of the small image area. The practical threshold is then derived in accordance with the variance of the entire watermarked image. From the above discussions, we propose a new watermark retrieval method based on the removal of surrounding pixel(s) that most affects its variance, compared to the threshold derived from the watermarked image's variance. If one surrounding pixel around *(i, j)* is already removed and the resultant variance of the remaining pixels around *(i, j)* is still much different from the threshold, we will remove another surrounding pixel that most affects its variance again. The process continues until the resultant variance of the remaining pixels around *(i, j)* is close enough to the threshold. However, the maximum number of the pixel removal is limited to 4 pixels to sustain the numbers of surrounding pixels left for the prediction process. The steps of our proposed watermark retrieval method are as follows:

1. Predict an original pixel from the watermarked image at coordinate *(i, j)*, the eight surrounding pixels around *(i, j)* are stored in the *temp* array.
2. Sort out the pixel values in the *temp* array in order from the smallest value to the largest one.
3. Compare the variance of *temp* to the variance of the watermarked image. If the *temp* variance is higher, remove one pixel at the first or last position in the *temp* array. The pixel to be removed depends on the new variance obtained after

removing it. That is, if removing the pixel at the first position gives the new variance less than removing the pixel at the last position, the pixel at the last position is removed.

4. Repeat step three until $\sigma_{B'}{}^2$ is lower than the threshold, or four surrounding pixels are removed.

The pseudo code to implement our proposed watermark retrieval method is given below.

```
Function adaptive original pixel prediction at coordinate (i,j)
optimum threshold - suitable value derived from image variance
neighbor[] - array of surrounding pixels sorted in order from the smallest value to the largest one
neighbor size - number of surrounding pixels used for the prediction
criterion1 - (size of neighbor[] >= neighbor size)
criterion2 - variance of neighbor[] > optimum threshold
v1 - variance of neighbor[] after removing one pixel at first position
v2 - variance of neighbor[] after removing one pixel at last position

While (criterion1 and criterion2)
    If (v1>v2)
        temp[] = neighbor[]*the one without the pixel at first position
    Else
        temp[] = neighbor[] *the one without the pixel at last position
    End if
    neighbor[] = temp[]
End while
Return ( sum(neighbor[]) / size of neighbor[] )
```

Detail, algorithm decision can be precise with suitable threshold. More detail about analysis of threshold and results will be described in chapter 4.

3.2 Adaptive Algorithm Version 2

To analyze the adaptive algorithm version 1, there are some conflictions about an assumption of the retrieval process that is the retrieval process can be achieved by the assumption that a pixel value at a given coordinate (i, j) can be estimated by the average of its nearby pixel values and the summation of w around (i, j) is close to zero. In case the number of the removal pixel is odd, it is impossible that the summation of *w* around (i, j) is close to zero.

To hold the summation, the number of the removal pixel will be couple. Moreover, the standard deviation will be replaced with variance because variance value is greater than pixel value for more. The standard deviation can represent the appearance of the extreme pixel value.

The pseudo code to implement the adaptive algorithm version 2 is given below.

```
Function adaptive original pixel prediction at coordinate (i,j)
threshold - suitable value
neighbor[] - array of surrounding pixels sorted in order from the smallest value to the largest one
sd – Standard deviation of neighbor array
neighbor[] = neighbor[]*remove the one couple extreme pixels(min and max of neighbor array)
calculate sd for neighbor
If (sd> threshold)
        neighbor[] = neighbor[]*the one couple extreme pixels(min and max of neighbor array)
End if
Return (sum(neighbor[]) / size of neighbor[])
```

3.3 Adaptive Algorithm Version 3

To support the original adaptive concept, the adaptive algorithm version 3 can be produced by combination between the adaptive algorithms versions 1 and 2.

The pseudo code to implement the adaptive algorithm version 3 is given below.

```
Function adaptive original pixel prediction at coordinate (i,j)
threshold - suitable value
neighbor[] - array of surrounding pixels sorted in order from the smallest value to the largest one
sd – Standard deviation of neighbor array
If (sd> threshold )
    neighbor[] = neighbor[]*the one couple extreme pixels(min and max of neighbor array)
    calculate sd for neighbor
    If (sd> threshold )
        neighbor[] = neighbor[]*the one couple extreme pixels(min and max of neighbor array)
        calculate sd for neighbor
        If (sd> threshold )
            neighbor [] = neighbor []*the one couple extreme pixels(min and max of neighbor array)
        End if
    End if
End if
Return (sum (neighbor []) / size of neighbor [])
```