Chapter 5: Proposed New Techniques

5.1 Concealogram:

Concealogram is the Spatial Domain Technique where hiding of digital image in image carried out using LSB Insertion method or other similar algorithms. We propose a method of encoding visual information in digital image. The algorithm puts together two data files of two images, one that we want to print as an observable picture and the other that we want to conceal within the observable picture. The obtained resultant image is termed as the Concealogram. In addition to this the mathematical representation of the hidden image is scrambled with a mathematical key. Once an image is encoded, only an authorized person, who has the key, can reveal the hidden image. Concealogram method is achieved with three different approaches as discussed below...

5.1.1 Four bit data hiding method

As we know each pixel contains 8 bits. The right most bit of pixel is called Least Significant Bit (LSB) and left most bit is called Most Significant Bit (MSB). In Four bit data hiding method, which embeds first four MSB bits (MSB,MSB-1,MSB-2,MSB-3) of secret image into last four LSB bits (LSB, LSB+1, LSB+2,LSB+3) of cover image. So last 4 bits of cover image is replace with first four bits of secret image. Now next pixel of cover image is taken and again 4 MSB bits of next pixel of secret image is inserted into 4 LSB bits of cover image.

<table>
<thead>
<tr>
<th>Cover image</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 1</td>
<td>Pixel 2</td>
<td>Pixel 3</td>
</tr>
<tr>
<td>1 0 1 1 0 0 1 1</td>
<td>1 1 0 1 0 1 1 0</td>
<td>0 1 0 0 1 0 0 1</td>
</tr>
<tr>
<td>LSB+3</td>
<td>LSB+2</td>
<td>LSB+1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secret image</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 1</td>
<td>Pixel 2</td>
<td>Pixel 3</td>
</tr>
<tr>
<td>1 0 0 0 0 0 1 1</td>
<td>0 1 0 1 1 1 0 0</td>
<td>1 1 0 0 1 1 0 0</td>
</tr>
<tr>
<td>MSB</td>
<td>MSB-1</td>
<td>MSB-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stego image</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 1</td>
<td>Pixel 2</td>
<td>Pixel 3</td>
</tr>
<tr>
<td>1 0 1 1 1 0 0 0</td>
<td>1 1 0 1 0 1 0 1</td>
<td>0 1 0 0 1 1 0 0</td>
</tr>
</tbody>
</table>

Figure 5.1: Concealogram: Data embedding using 4 bit technique
In this method only 4 MSB bits of each pixel of secret image is considered. Because of Lower bits of pixel do not contain more information, so here we omit remaining four bits (LSB, LSB+1, LSB+2, LSB+3) of secret image. The data to be hidden is to be converted in binary form. In figure 5.1 gray shade shows that the last 4 LSB bit of cover image is replaced with first 4 MSB bits of secret image. Image processing fundamental said that (‘LSB contain less information’), so if we change LSB data it likely that it will not disturb the quality of image. In proposed method we omit last 4 bits LSB of each Pixel from secret image.

5.1.2 Three bit data hiding method

Three bit data hiding method embeds first three MSB bits (MSB, MSB-1, MSB-2) of secret image into last three LSB bits (LSB, LSB+1, LSB+2) of cover image as shown in figure 5.2. Again three bits of secret image (MSB-3, MSB-4, MSB-5) is stored in 3 LSB bits of next pixel of cover image. Cover image embeds three bit data in each pixel byte. The data to be hidden is converted to binary form. In Figure 5.2 gray shade shows that the last 3 LSB bit of cover image is replaced with first 3 MSB bits of secret image. From MSB-3, next 3 bit from Pixel 1 of Secret image (Shown by Darker gray shade) is stored in LSB from Pixel 2 of cover image. In proposed method we omit last 2 bits LSB of each Pixel from secret image, which is having minimal information.

| Cover image | | |
|-------------|--|--|---|
| Pixel 1     | Pixel 2 | Pixel 3 |
| 1 0 1 1 0 0 1 1 | 1 1 0 1 0 1 1 0 | 0 1 0 0 1 0 0 1 |
| LSB+2       | LSB+1   | LSB   |

| Secret image | | |
|-------------|--|--|---|
| Pixel 1     | Pixel 2 | Pixel 3 |
| 1 0 0 0 0 0 1 1 | 0 1 0 1 1 1 0 0 | 1 1 0 0 1 1 0 0 |
| MSB          | MSB-1   | MSB-2   | MSB-3 | MSB-4 | MSB-5 |

| Stego image | | |
|-------------|--|--|---|
| Pixel 1     | Pixel 2 | Pixel 3 |
| 1 0 1 1 0 1 0 0 | 1 1 0 1 0 0 0 0 | 0 1 0 0 1 0 1 0 |

Figure 5.2: Concealogram: Data embedding using 3 bit technique

5.1.3 Two bit data hiding method

Likewise above two methods this method embeds first two MSB bits (MSB, MSB-1) of secret image into last two LSB bits (LSB, LSB+1) of cover image as shown in figure 5.3. Again two bits of secret image (MSB-2, MSB-3) are stored in 2 LSB bits of next pixel of cover
image. Again two bits of secret image (MSB-4, MSB-5) are stored in 2 LSB bits of next pixel of cover image and so on. So we are hiding total six MSB bits of secret image into cover image. Cover image pixel embeds two bit data. Here only two bit of cover is changed after storing the secret bit so PSNR of this method is very good but we need to compromise with capacity.

<table>
<thead>
<tr>
<th>Cover image</th>
<th>Pixel 1</th>
<th>Pixel 2</th>
<th>Pixel 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 0</td>
<td>1 1 0 1 0 1 1 0</td>
<td>1 0 0 1 0 0 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secret image</th>
<th>Pixel 1</th>
<th>Pixel 2</th>
<th>Pixel 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 1</td>
<td>1 0 1 1 1 1 0 1</td>
<td>1 1 0 0 1 1 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stego image</th>
<th>Pixel 1</th>
<th>Pixel 2</th>
<th>Pixel 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 0</td>
<td>1 1 0 1 0 1 0 0</td>
<td>1 0 0 1 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.3: Concealogram: Data embedding using 2 bit technique

5.2 Pixel Pair Differencing Technique

Based on the embedding domains, image steganography algorithms can be classified into two types, one is embedding in the spatial domain, such as LSB (Least Significant Bit) based [80, 81] approaches and PVD-based [82, 83] approaches, and other is embedding in the transform domain, such as F5 [84] and outguess [85].

The LSB-based steganography is one of famous approaches in the spatial domain, in which the least significant bits of a cover image along a pseudo-random route are changed according to the secret bit stream, which is to be embedded. These methods regard that all pixels within an image can tolerate equal amounts of changes without causing visual artifacts to an observer. However, this is not true especially for the images having smoother and/or regular regions [86].

Our human vision is sensitive to slight changes in the smooth regions, while it can tolerate more severe changes in the edge regions, the PVD based methods have been proposed to enhance the embedding capacity without introducing obvious visual artifacts into stego images. In PVD-based schemes, the number of embedded bits is determined by the difference between the pixel and its neighbor.
PROPOSED NEW TECHNIQUES

pixels (between the first and the second pixel as well as the second and the third pixel, and so on) row-by-row or column-by-column is used for embedding data.

The larger the difference amount is, the more secret bits can be embedded. Usually, PVD-based approaches can achieve more imperceptible results compared to those typical LSB-based approaches with the same embedding capacity.

Encoding Process

Cover image I is an 8-bit grayscale image with size M×N. Take first row and denote \( P_i \) as the pixel value of \( i^{th} \) column in the original image. We are denoting \( P^*_i \) as the pixel value of \( i^{th} \) column in stego image.

At the start we take \( P^*_1 = P_1 \). The adjacent pixel value difference is

\[
d_i = P^*_i - P_{i+1}
\]

We have used \texttt{mod} function in MATLAB to find remainder.

\[
\text{If } d_i \text{ is odd then } \texttt{mod}(d_i/2) = 1
\]

\[
\text{If } d_i \text{ is even then } \texttt{mod}(d_i/2) = 0
\]

After checking \( d_i \) get the message bit \( m \). Based on values of \( m \) and \( d_i \) encoding is done in such a way which makes \( d_i \) value odd if \( m \) is ‘1’ and even if \( m \) is ‘0’. This process can be described as

For \( m = 1 \)

\[
\begin{align*}
\text{If } & \texttt{mod}(d_i/2) = 1 \quad (d_i \text{ is odd}) \quad P^*_{i+1} = P_{i+1} \\
\text{If } & \texttt{mod}(d_i/2) = 0 \quad (d_i \text{ is even}) \quad P^*_{i+1} = P_{i+1} + 1
\end{align*}
\]

For \( m = 0 \)

\[
\begin{align*}
\text{If } & \texttt{mod}(d_i/2) = 1 \quad (d_i \text{ is odd}) \quad P^*_{i+1} = P_{i+1} + 1 \\
\text{If } & \texttt{mod}(d_i/2) = 0 \quad (d_i \text{ is even}) \quad P^*_{i+1} = P_{i+1}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Table 5.1: Test 8x8 Image Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>161 160 160 161 159 159 162 163</td>
</tr>
<tr>
<td>158 159 161 162 160 163 167 165</td>
</tr>
<tr>
<td>161 161 162 157 158 160 162 165</td>
</tr>
<tr>
<td>159 158 157 159 160 165 163 167</td>
</tr>
<tr>
<td>160 160 162 162 163 164 163 168</td>
</tr>
<tr>
<td>161 161 162 160 162 167 170 169</td>
</tr>
<tr>
<td>160 160 164 161 168 168 170 172</td>
</tr>
<tr>
<td>162 161 159 165 171 166 169 170</td>
</tr>
</tbody>
</table>
Let us consider an image block as shown in Table 5.1. Let message sequence to be embedded is ‘1110100’. Data is hidden from left to right (i.e., MSB to LSB) as shown in figure 5.4. Algorithm 5.1 shows encoding process for Pixel Pair Differencing technique.

**Algorithm 5.1**: The encoding process of Pixel Pair Differencing

1: **Input**: Message $M$, Cover image $C$
2: **Output**: Stego Image $S$
3: $S = C$
4: **For** $i = 1, ..., l(M)$
5: \[ d = S_i - S_{i+1} \]
6: \[ \text{if } mod(d/2) \neq M, \text{ then} \]
7: \[ S_{i+1} \leftarrow S_{i+1} + 1 \]
8: **end if**
9: **end for**

**Decoding Process**

The decoding is done in reverse way. We have to find the adjacent pixel value difference of stego image using equation 5.4.
\[ d^*_i = P^*_i - P^*_{i+1} \]  \hspace{1cm} (5.4)

Now we have received message bit ‘1’ (if \( d^*_i \) is odd) or ‘0’ (if \( d^*_i \) is even). This is described as

Received message bit = \( \text{mod} \left( d^*_i / 2 \right) \)  \hspace{1cm} (5.5)

Figure 5.5: Pixel pair differencing decoding process

**Algorithm 5.2:** The decoding process of Pixel Pair Differencing

1: Input: Stego image \( S \), Message length \( l \)
2: Output: Message \( M \)
3: for \( i = 1, \ldots, l(S) \) do
4: \quad \( d = S_i - S_{i+1} \)
5: \quad \( M'_i \leftarrow \text{mod}(d/2) \)
6: end for

The decoding is done in reverse way as shown in algorithm 5.2. We have to find the adjacent pixel value difference of stego image as

\[ d_i = S_i - S_{i+1} \]  \hspace{1cm} (5.4)

Now we have received message bit 1 (if \( d_i \) is odd) or 0 (if \( d_i \) is even) as described by equation 5.5.

Received message bit = \( \text{mod} \left( d_i / 2 \right) \)  \hspace{1cm} (5.5)

Figure 5.5 shows the Decoding process of pixel pair differencing technique.
5.3 Multiple LSB based on Pixel Value

In gray scale image, pixel is of 8 bits & its value can be anything from 0 to 255. In this method we hide multiple LSB based on pixel value. Data embedding process based on pixel value $P(i,j)$ located at $i^{th}$ row and $j^{th}$ column:

1. $P(i,j) \geq 192$ : Hide 3 data bits in 3 LSB of $P(i,j)$.
2. $192 > P(i,j) \geq 64$ : Hide 2 data bits in 2 LSB of $P(i,j)$.
3. $P(i,j) < 64$ : Hide 1 data bit in LSB of $P(i,j)$.

This process continues till we hide all data bits, after which we get Stego image $S$ as shown in figure 5.6. For data embedding, first we convert text data into ASCII bits. As shown in Figure 5.6, let us have message “hi”, after conversion to ASCII bits of ‘h’ (1001000) followed by those of ‘i’ (1101001). Total message bits will be 10010001101001. Now based on pixel value we hide message bits into LSB of pixel. From embedding process it’s clear that we are only changing LSB bits, which won’t change the three defined ranges of pixel values since there is no change in MSBs of pixel value.

Algorithm 5.3: The encoding process of Multiple LSB based on Pixel Value

1: Input: Message $M$, Cover image $C$
2: Output: Stego Image $S$
3: $S = C$
4: $i=1$
5: do
6: if $S_i \geq 192$
7: Hide 3 bits of $M$ into 3 LSBs of $S_i$
8: else if $S_i \geq 64$
9: Hide 2 bits of $M$ into 2 LSBs of $S_i$
10: else
11: Hide 1 bit of $M$ into LSB of $S_i$
12: end if
13: $i=i+1$
14: while $i \leq l(M)$
**PROPOSED NEW TECHNIQUES**

<table>
<thead>
<tr>
<th>Cover Image Row:</th>
<th>194</th>
<th>192</th>
<th>189</th>
<th>131</th>
<th>51</th>
<th>63</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Data: Hl, Message ASCII bits: 100100011101001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(1)</td>
<td>11000010</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
<td>131</td>
<td>51</td>
</tr>
<tr>
<td>P(2)</td>
<td>100 (100100011101001) in 3 LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(3)</td>
<td>10111101</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
<td>131</td>
<td>51</td>
</tr>
<tr>
<td>P(4)</td>
<td>10000011</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
<td>131</td>
<td>51</td>
</tr>
<tr>
<td>P(5)</td>
<td>00111111</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
<td>131</td>
<td>51</td>
</tr>
<tr>
<td>P(6)</td>
<td>00111101</td>
<td>101</td>
<td>01011101</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
</tr>
<tr>
<td>P(7)</td>
<td>00111101</td>
<td>101</td>
<td>01011101</td>
<td>192</td>
<td>192</td>
<td>192</td>
<td>189</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stego Image Row:</th>
<th>196</th>
<th>196</th>
<th>189</th>
<th>130</th>
<th>90</th>
<th>62</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(1)</td>
<td>11000010</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>S(2)</td>
<td>100 (100100011101001) in 3 LSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S(3)</td>
<td>10111101</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>S(4)</td>
<td>10000011</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>S(5)</td>
<td>00111111</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>S(6)</td>
<td>00111101</td>
<td>101</td>
<td>01011101</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
</tr>
<tr>
<td>S(7)</td>
<td>00111101</td>
<td>101</td>
<td>01011101</td>
<td>196</td>
<td>196</td>
<td>189</td>
<td>130</td>
</tr>
</tbody>
</table>

Figure 5.6: Data embedding using Multiple LSB based on pixel value

Figure 5.7: Outcome of Multiple LSB based on pixel value (sequential)

Figure 5.7 shows result of Multiple LSB based on pixel value using sequential approach (i.e. hiding data row by row in sequence). Fig.3(c) and (d) shows LSB bits of original and stego image respectively. In fig 3(d) we can see pattern in LSB plane which shows presence of data bits in it. To avoid this we can hide data at random pixel locations using random number key generator. Result of randomize approach is shown in Figure 5.8.
Algorithm 5.4: The decoding process of Multiple LSB based on Pixel Value

1: **Input:** Stego image $S$, Message length $l$
2: **Output:** Message $M$
3: $i = 1$
4: **do**
5: 
6:  
7:  
8:  
9:  
10:  
11: 
12: $i = i + 1$
13: **while** $i \leq l(M)$

For data extraction, we get pixel value $S(i,j)$ of stego image and perform following process:

1. $S(i,j) \geq 192$: extract 3 data bits from LSB of $S(i,j)$.
2. $192 > S(i,j) \geq 64$: extract 2 data bits from LSB of $S(i,j)$.
3. $S(i,j) < 64$: extract 1 data bit from LSB of $S(i,j)$. 
PROPOSED NEW TECHNIQUES

Algorithm 5.4 shows decoding process of Multiple LSB based on pixel value technique. Line 5 to 11 indicates extraction of message bits based on pixel value. Figure 5.9 shows data extraction process for Multiple LSB based on pixel value technique, which is reverse of embedding process. Here we continue extraction of pixel LSBs till we finish total number of message bits. After receiving message bits we need to make group of 7 bits & get characters corresponding to it. Later by combining all characters we get sentence. In case of randomize approach; we need same random location key, which is used for data hiding.

5. 4 Multiple LSB based on MSB digits technique

In this method we hide multiple LSB based on number of ones in first three MSB bits of pixel’s binary value. This method is similar to previous one in the sense that we are not hiding anything into MSB digits. The difference is that it depends on 3 MSB bits rather than pixel value. Let’s consider \( N \) as total number of 1’s in three MSB digits. Data embedding process based on \( N \) for pixel value \( P(i,j) \) located at \( i^{th} \) row and \( j^{th} \) column is as follow:

1. **N=0**: Don’t hide anything in LSB i.e. skip this pixel
2. **N=1**: Hide 1 data bit in LSB of \( P(i,j) \).
3. **N=2**: Hide 2 data bits in 2 LSB of \( P(i,j) \).
4. **N=3**: Hide 3 data bits in 3 LSB of \( P(i,j) \).
Algorithm 5.5: The encoding process of Multiple LSB based on MSB digits

1: Input: Message $M$, Cover image $C$

2: Output: Stego Image $S$

3: $S = C$

4: $i = 1$

5: do

6: $N = \text{Total no of 1 in 3 MSBs of } S_i$

7: if $N = 0$

8: don’t hide anything

9: else

10: $\text{Hide } N \text{ bits of } M \text{ into LSBs of } S_i$

11: end if

12: $i = i + 1$

13: while $i \leq l(M)$

Figure 5.10: Data embedding using Multiple LSB based on MSB digits
Proposed New Techniques

Simplified way to describe data embedding is “we hide N bits in LSB of $P(i,j)$”. Algorithm 5.5 shows the encoding process of Multiple LSB based on MSB bits. Line 6 to 11 shows data embedding process based on 3 MSB bits of any pixel. After hiding all data bits which we get Stego image $S$ as shown in figure 5.10. As described in Multiple LSB based on pixel value technique, first we convert text data into ASCII bits.

From embedding process it’s clear that we are only changing LSB bits, which won’t change three MSB of pixel binary value. This method differs with pixel value method in a manner that we are not looking at pixel value but at three MSB bits of pixel binary value.

Figure 5.11: Outcome of Multiple LSB based on MSB digits (sequential)

Figure 5.12: Outcome of Multiple LSB based on MSB digits (randomize)
Figure 5.11 and figure 5.12 show the result of Multiple LSB based on three MSB digits of pixel using sequential & randomize approach respectively.

For data extraction, we get pixel value $S(i,j)$ of stego image & based on total number of 1’s in three MSB of it, we perform following process:

1. $N=0$: Don’t extract anything i.e. skip this pixel
2. $N=1$: extract 1 data bit from LSB of $S(i,j)$.
3. $N=2$: extract 2 data bits from LSB of $S(i,j)$.
4. $N=3$: extract 3 data bits from LSB of $S(i,j)$.

Algorithm 5.6: The decoding process of Multiple LSB based on MSB digits

1:  **Input**: Stego image $S$, Message length $l$
2:  **Output**: Message $M$
3:  $i=1$
4:  do
5:     $N=$Total no of 1 in 3 MSBs of $S_i$
6:     if $N=0$
7:         don’t extract anything
8:     else
9:         Extract $N$ bits of $M$ from LSBs of $S_i$
10:    end if
11:    $i=i+1$
12:   while $i \leq |M|$

Algorithm 5.6 shows decoding process of Multiple LSB based on MSB digits. Line 5 to 10 indicates extraction of message bits based on 3 MSB bits of pixel.

Figure 5.13 shows data extraction process for Multiple LSB based on MSB bits technique, which is reverse of embedding process. Extraction of message bits from pixel LSBs based on 3 MSB bits are carried out till we finish total number of message bits.

After receiving message bits we need to make group of 7 bits & get characters corresponding to it. Later by combining all characters we get sentence. In case of randomize approach; we need same random location key, which is used for data hiding.
5.5 Pixel Swap Modification Techniques

This transform domain technique is modification of Pixel Swap Embedding technique discussed in section 4.2.6 of chapter 4. The basic algorithm is described in as dividing the image into 8x8 blocks and calculating the DCT of the block. As described in algorithm 4.11, a block of 8x8 encodes a ‘1’ if \(\text{DCT}(u_1,v_1) > \text{DCT}(u_2,v_2)\) & encodes ‘0’ if \(\text{DCT}(u_1,v_1) < \text{DCT}(u_2,v_2)\).

What to do for \(\text{DCT}(u_1,v_1) = \text{DCT}(u_2,v_2)\) has not been mentioned in algorithm 4.11. Modification in Pixel Swap Embedding technique was required because one get error message at receiver side for an image whenever \(\text{DCT}(u_1,v_1) = \text{DCT}(u_2,v_2)\).

Algorithm 5.7 shows encoding process of the Pixel Swap Modification technique. Line 13 to 18 shows embedding process for \(\text{DCT}(u_1,v_1) = \text{DCT}(u_2,v_2)\), in which values of \(\text{DCT}(u_1,v_1)\) & \(\text{DCT}(u_2,v_2)\) are changed as per message bit. For message bit ‘1’ we make \(\text{DCT}(u_1,v_1) > \text{DCT}(u_2,v_2)\) & for message bit ‘0’ we make \(\text{DCT}(u_1,v_1) < \text{DCT}(u_2,v_2)\). So after data embedding, situation of \(\text{DCT}(u_1,v_1) = \text{DCT}(u_2,v_2)\) doesn’t occur. In algorithm 5.7, \(x\) is a threshold value which represents the trade-offs between image quality and robustness.
At receiver side bit ‘1’ extracted if $DCT(u_2, v_1) > DCT(u_2, v_2)$ & bit ‘0’ is extracted if $DCT(u_2, v_1) < DCT(u_2, v_2)$. So the decoding process will be same as shown in algorithm 4.12 of chapter 4.

**Algorithm 5.7:** The encoding process of the Pixel Swap Modification technique

1: for $i = 1, \ldots, l(M)$ do
2:   choose one cover-block $b_i$
3:   $B_i = D\{b_i\}$
4:   if $m_i = 0$ then
5:     if $B_i(u_1, v_1) > B_i(u_2, v_2)$ then
6:       swap $B_i(u_1, v_1)$ and $B_i(u_2, v_2)$
7:     end if
8:   end if
9:   else
10:  if $B_i(u_1, v_1) < B_i(u_2, v_2)$ then
11:    swap $B_i(u_1, v_1)$ and $(u_2, v_2)$
12:  end if
13:  end if
14:  if $B_i(u_1, v_1) = B_i(u_2, v_2)$ then
15:    if $m_i = 1$ then
16:      $B_i(u_1, v_1) = B_i(u_2, v_2) + x$
17:    else
18:      $B_i(u_2, v_2) = B_i(u_1, v_1) + x$
19:    end if
20:    adjust both values so that $|B_i(u_1, v_1) - B_i(u_2, v_2)| > x$
21:  $b'_i = D^{-1}\{Bi\}$
22: end for
23: create stego-image out of all $b'_i$

One can also do encoding in other way i.e. message bit ‘1’ can be encoded if $DCT(u_2, v_1) < DCT(u_2, v_2)$ & ‘0’ if can be encoded if $DCT(u_2, v_1) > DCT(u_2, v_2)$. Though the embedding capacity is low, the quality of image is good. Advantage of this technique is its more robust to attacks, such as compression, cropping etc.