Eastern Mediterranean University

Department of Computer Engineering

Software Engineering Program

CMSE 492 Term Project

Famagusta, North Cyprus

Spring 2017/2018 Semester

Students: Group 3

1. Talal Mahdy (147139)
2. Charles Brown (147118)
3. Abdoulgwad Elsheredi (147597)
4. Balto Mohammad (147697)

Instructor: Assoc. Prof. Dr. Alexander Chefranov
Lab Coordinator: Ms. Nivine Samarji

May 20, 2018

1. Outline

The aim of this project is to develop a software tool that implements 3 methods of data embedding-Wang's method [1], Wu and Tsai's method [2], and Hybrid method [3], so that embedding is followed by extraction, and the data extracted is always verified versus the data embedded.

2. Problem Definition

1. Implement Wang's method, Wu and Tsai's , and Hybrid algorithm in Java Language
2. Compare methods [1-3] using host images specified in [1-3] and secret messages specified in [1] (256x256 gray scale images).
   - 2.1. Embedding capacity
   - 2.2. PSNR
   - 2.3. Image quality index
   - 2.4. Time for embedding (extraction) of one secret image into (from) one cover (stego) image
3. Find the best parameters for each method
   - 3.1. m_l, m_u, T for [1]
     - 3.1.1. Try T=32*I, i=1,..,7
     - 3.1.2. Try m_l=8+4*I, i=0,..,7, keeping $m_l<2^{\log_2 T}$, $m_l<m_u$
     - 3.1.3. Try m_u=8+8*I, i=0,..7, keeping $m_u<2^{\log_2 (256-T)}$
   - 3.2. Ranges for [2]
     - 3.2.1. Try two ranges used in [2] and {[0,31],[[32,63],[64,127],[128,255]}.
   - 3.3. Threshold and ranges for [3]
     - 3.3.1. Try thresholds, T=32*I, i-1,..,7
     - 3.3.2. Try ranges: [3, Table 1]; [3, Figs. 1] with (6,8,10,12,14) bits embedding into respective range; [3, Fig.2, a] with.(8, 10) bits for embedding; [3, Fig. 2, b] with (8, 10, 12) bits for embedding
4. Show results of your study in tabular form as it is made in [1-3]

3. Method description

Wang's Method Description

Wang's method is a technique which is used to hide or conceal the fact that there is a hidden message which is to be sent. These techniques are known as data embedding and they are different than Cryptography. In cryptography, it is clear that there is some hidden data which require a special key to access. But in Embedding, the user will not be aware that there is a secret data since the media of transfer will usually be in the form of images or videos with secrets embedded in them. To implement Wang's Algorithm, we have to go through 3 phases of the algorithm. The first phase is called Bit String Transformation where we input a secret image to embed data into, and we get a String of Bits as an output. In the second phase, we input the cover image, the bit string, a threshold value, and two modulus numbers to get a secret image as an output. And finally in the third phase, we input the secret image, the modulus values and the threshold value to get the extracted bit string back.

Wu and Tsai's

The Wu and Tsai method proposes an algorithm that allows users to embed the LSBs of the pixels of a gray-valued image without distorting the image such that the changes made go unnoticed by user's vision. In the Wu and Tsai's method we simply divide the cover image into a number of non-overlapping two-pixel blocks. Whereby, each block is ordered according to the difference (d) of the gray values of the two pixels in the block. A small difference value (d) indicates that the block is in a smooth area and conversely a large one indicates that it is in an edged area. However, the pixels in edged areas may tolerate larger changes of pixel values than those in smooth areas, hence Wu and Tsai's proposal to embed more data in edged areas in comparison to smooth areas. Moreover, it is in this way we keep the changes in the resulting stego-image unnoticeable.

Khodaei Hybrid method description

The Khodaei Hybrid method proposes an algorithm based on the concept that sharp edge areas can tolerate larger changes than smooth areas. Similar to the PVD method (Wu and Tsai 2003), we use the difference value of pixels to determine how many secret bits can be embedded into LSBs of two consecutive pixels. In estimating the number of secret embedding bits, a Range table R_i is designed with continuous ranges from 0 to 255 that is introduced in Table 1 [1]. We have R_iϵ $[l_i, u_i]$ where l_i is the lower bound ofR_i, and u_i is the upper bound ofR_i. The range table R_i has four ranges R_1= [0, 15], R_2= [16, 63], R_3 = [64,127], and R_4 = [128, 255]. Khodaei Hybrid method consists of two main phases, the embedding and extracting phases.

5. Description of methods implemented in JAVA programming language

5.1. Wang's method

The Software program is designed as a menu driven program and the design flow of the program with an example goes like this:

```
1. Welcome to Wang's Algorithm Implementation!
   Phase 1: Bit String Transformation. Please make your choice regarding the
            secret you would like to embed.

      1. I would like to enter my own secret text.
      2. I would like to embed the Mandrill 256x256 image
      3. I would like to embed the Peppers 256x256 image
      4. I would like to embed the Jet 256x256 image
      5. I would like to embed the Lena 256x256 image

         =>User selects 1

      Enter Secret String to be embedded: ABCD

2. Phase 2: Please make your choice regarding the Cover Image you would like
            to embed the secret in.

      1. I would like to enter my own Cover Image Pixels.
      2. I would like to embed my secret in the Mandrill 512x512 image
      3. I would like to embed my secret in the Peppers 512x512 image
      4. I would like to embed my secret in the Jet 512x512 image
      5. I would like to embed my secret in the Lena 512x512 image

            =>User selects 1

      Enter number of rows: 2
      Enter number of columns: 3

      Enter Pixels separated by space. Enter -1 to stop taking pixel inputs:
      0 110 158 163 190 250 -1

      Secret String: ABCD
      Secret String as Bit Stream Bs: 01000001010000100100001101000100
      Cover Image Pixels: 0 110 158 163 190 250

      Secret Embedding.
      Enter Threshold Value T: 160
      Enter Upper Modulus number mu: 16
      Enter Lower Modulus number ml: 8
      Secret Image Pixels: 2 112 154 168 196 248
      Phase 2 is Over.

3. Phase 3: Extraction of Secret.

      Extracted Bit Stream: 01000001010000100100O
      Phase 3 done. PSNR Calculation:
```

```
MSE: 14.833333333333334
PSNR: 36.41841604606641
Image is in good condition since PSNR is greater than or equal to 30.
```

5.2. Khodaei Hybrid method

5.2.1. Description of the host/secret images used and their sources

Fig1. 256* 256 Lena grayscale image

Image Source - https://www.researchgate.net/figure/8-bit-256-x-256-Grayscale-Lena-Image_fig1_3935609

Fig2. 512* 512 Lena grayscale image

Image Source - https://www.researchgate.net/figure/lenatif-512x512-grayscale-and-stego-image_fig4_266200292

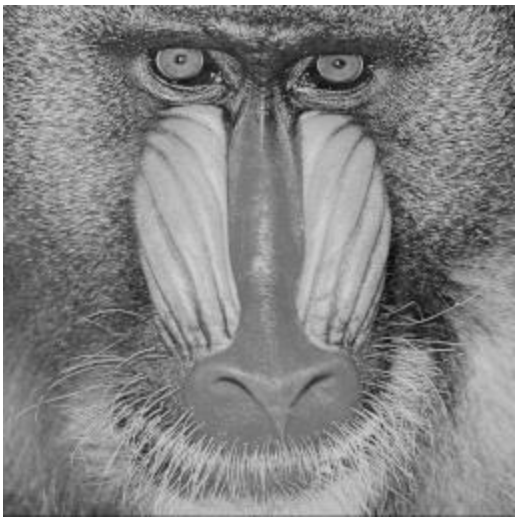


Fig3. Mandrill 256x256 grayscale image

Image Source - http://www.ece.northwestern.edu/~faisal/d20/baboon.bmp

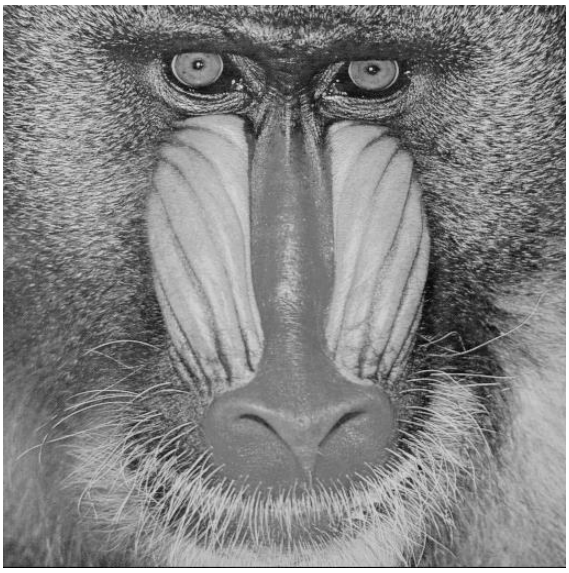


Fig4. Mandrill 512x512 grayscale image

http://decsai.ugr.es/cvg/CG/images/base/47.gif

Fig5. Peppers 256x256 gray scale image

Image Source – http://www.ee.columbia.edu/~sfchang/course/dip/demos/sampleexample.html

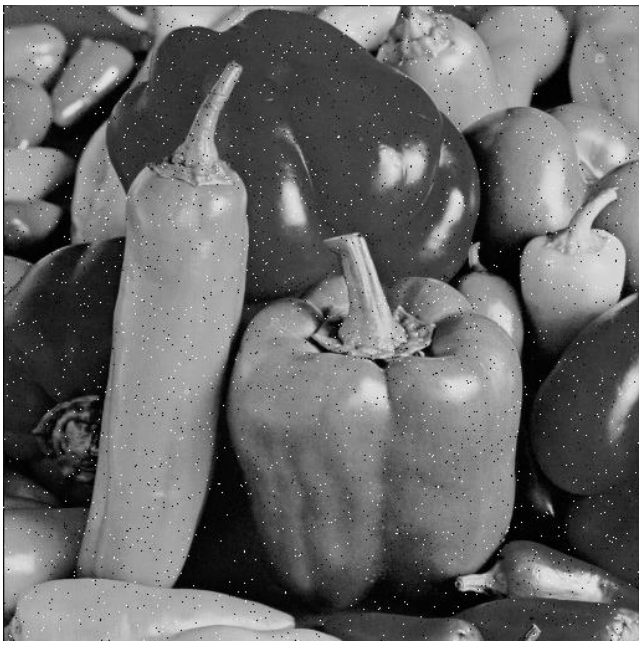


Fig6. Peppers 512x512 grayscale image

Image Source - http://www.hlevkin.com/TestImages/pepper.bmp

Note:

The Jet, Zelda and Scene images were not found.

5.2.2. Description of the obtaining secret binary data stream

The proposed method uses the image with 256 gray-scales as a cover image. To hide secret data into cover-image pixels, divide the cover image into several non-overlapping blocks with two consecutive pixels in raster scan manner, say p_i and p_{i+1}. Then, embed n bits of the secret data into say p_i and p_{i+1} of each block to the following two cases.

Note:

Ranges are R1= [0, 15], R2= [16, 63], R3= [64,127], R4= [128,255]

A difference value in Range:

- R1 represents 8 bits from secret data to be embedded
- R2 represents 10 bits from secret data to be embedded
- R3 represents 12 bits from secret data to be embedded
- R4 represents 14 bits from secret data to be embedded

Calculate difference value (d) by $p_{i+1} - p_i$

Check range of d to determine how many bit(s) of secret data to be embedded.

5.2.3. Description of embedding phase

Having found d and determined number of embedding bits (n) through the range table [1]. Then, embed n bits of the secret data into p_i and p_{i+1} of each block to the following two cases.

Case 1. If the values of both pixels p_i and $p_{i+1}$are less than 192 (i.e., p_i < 192 and p_{i+1}< 192), read n = 6 bits of secret data and embed k= 3 first bits of it into k LSBs of pixels p_i to obtain p_i' and k =3 next bits of it into k LSBs of pixels p_{i+1} to obtain p'_{i+1}.

Case 2. If one of the pixel values p_i and p_{i+1} or both are larger or equal to 192 (i.e., p_i ≥ 192 or p_{i+1} ≥ 192), hide the secret data into two successive pixels of the cover image according to the following procedure: Step 1. Calculate the difference value d_i between two consecutive pixels in the block by

$$d_i = | p_i - p_{i+1} |$$

Step 2. Refer to the range table R_i and find out the range to which di belongs. Now, obtain the number of embedding bits n and read n bits of secret data and embed k = n/2 first bits of the secret data into k LSBs of pixel p_i to obtain p_i' and k = n/2 next bits of it into k LSBs of pixel p_{i+1} to get p'_{i+1}. Step 3. Calculate the new difference value d0 i , which is given by

$$d_i' = | p_i' - p'_{i+1}.|$$

Step 4. If d_i and d_i' belong to different ranges, re-adjust p_i' and p'_{i+1}|. Carry out the readjusting process as follows.

1. Compute the modified values of p_i' and p'_{i+1} by

$$p_i'' = p_i' + 2^k;$$

$$p'''_i = p_i' - 2^k;$$

$$p''_{i+1} = p'_{i+1} + 2^k;$$

$$p'''_{i+1} = p'_{i+1} - 2^k;$$

1. Select the optimal value of p0 i and p0 iþ1 by

(p_i', p'_{i+1})= optimal $\{(p_i', p''_{i+1}) || (p_i', p'''_{i+1}) || (p_i'', p'_{i+1}) || (p_i'', p''_{i+1})$

$|| (p_i'', p'''_{i+1}) || (p_i''', p'_{i+1}) || (p_i''', p''_{i+1}) || (p_i''', p'''_{i+1})\}$

To choose the optimal values of p_i' and p'_{i+1}, it is essential to notice that the optimal selected values are the values that have minimum difference with original values p_i' and p'_{i+1}. For example, if $|p_i - p_i'|$ and $|p_{i+1} - p''_{i+1}|$ are minimum in comparison with the other seven values, we choose (p_i', p''_{i+1}) as optimal value and replace it instead of (p_i', p'_{i+1}). Also, one of these values or both of them are greater than or equal to 192. Moreover, d_i and - d_i' belong to the same range and also- $p_i' \leq 255$ and - $p'_{i+1} \leq 255$.

5.2.4. Description of extraction phase

To extract secret data bits from stego-image pixels, the following steps are accomplished.
Step 1. Divide the stego-image into some non-overlapping blocks of two consecutive pixels in raster scan manner, say p_i' and p'_{i+1}.
Step 2. Extract secret data bits from LSBs of two successive pixels as follows.
Case 1. If the values of both pixels p_i' and p'_{i+1} are less than 192 (i.e., $p_i' < 192$ and $p'_{i+1} < 192$), extract k= 3 bits from k LSB bits of pixel $p_i'$and again k = 3 bits from k LSB bits of pixel p'_{i+1}.
Case 2. If one of the pixel values pi and piþ1 or both of them are larger than or equal to 192 (i.e., $p_i \geq 192$ or $p_{i+1} \geq 192$), extract the secret data bits from two successive pixels of the stego-image as follows.
1. Calculate the difference value d_i' between two consecutive pixels in the block by

$$d_i' = |p_{i+1} - p''_{i+1}|$$

2. Apply the range table$R_i$and obtain the range to which d_i'' belongs.
3. Get the number of n secret embedded bits into two pixels p_i' and p'_{i+1} and extract k = n/2 secret bits from k LSBs of $p_i'$and also exploit k = n/2 secret bits from k LSBs of p'_{i+1}.

5.2.5. Description of PSNR, embedding capacity calculation, quality index, number of pixels embedded consistently

We employ the peak signal-to-noise ratio (PSNR) to evaluate the distortion of the cover images after embedding secret data. If the PSNR value is larger than 30 dB, the distortion of the stego-image is imperceptible to the human eye (Lee et al. 2008). The PSNR value between the cover image P and the stego-image P' can be computed as

PSNR = 10 X Log ($\frac{(255)^2}{MSE}$) where MSE is defined by,

$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}(P_i - P_i')^2,$$

Where m is the number of pixels in P and P'. We utilize the universal image quality index to indicate the quality of stego-images (Wang and Bovik 2002). The quality index Q is based on statistic measure and is calculated by

$$Q = \frac{4\sigma_{xy}\bar{x}\bar{y}}{(\sigma_x^2 + \sigma_y^2)[(\bar{x})^2 + (\bar{y})^2]},$$

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,\ \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i,$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2,$$

$$\sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2,$$

$$\sigma_{xy}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}).$$

Here, N is the size of the cover image, x_i is the value of the pixels in the cover image, y_i is the value of the pixels in the stego-image, $\bar{x}$ is the mean value of x_i, and $\bar{y}$ is the mean value of y_i.

5.3. Wu's and Tsai's method

5.3.1. Description of the host/secret images used and their sources

Fig1. 256* 256 Lena grayscale image

Image Source - https://www.researchgate.net/figure/8-bit-256-x-256-Grayscale-Lena-Image_fig1_3935609

Fig2. 512* 512 Lena grayscale image

Image Source - https://www.researchgate.net/figure/lenatif-512x512-grayscale-and-stego-image_fig4_266200292

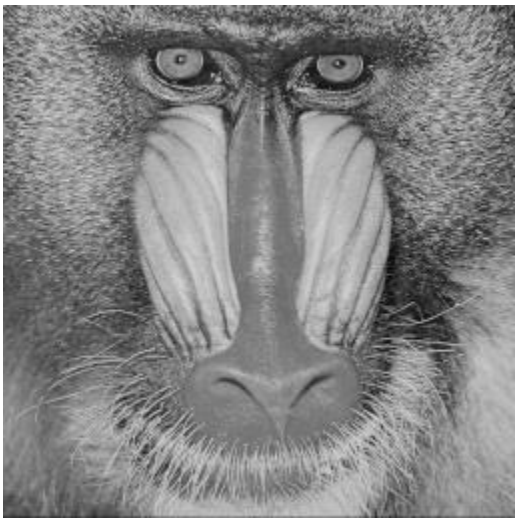


Fig3. Mandrill 256x256 grayscale image

Image Source - http://www.ece.northwestern.edu/~faisal/d20/baboon.bmp

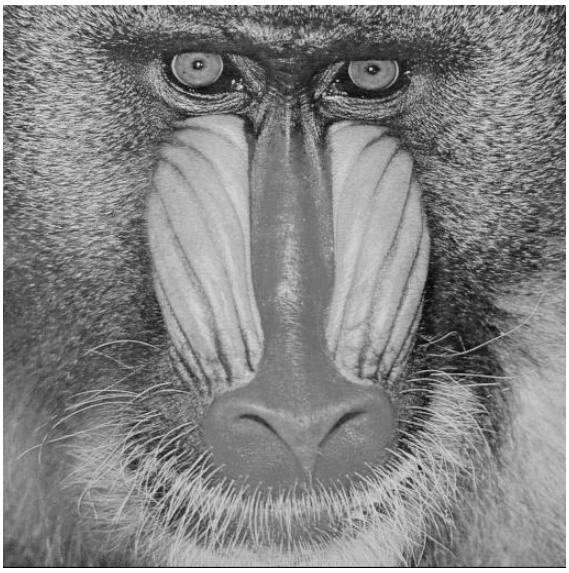


Fig4. Mandrill 512x512 grayscale image

http://decsai.ugr.es/cvg/CG/images/base/47.gif

Fig5. Peppers 256x256 gray scale image

Image Source – http://www.ee.columbia.edu/~sfchang/course/dip/demos/sampleexample.html

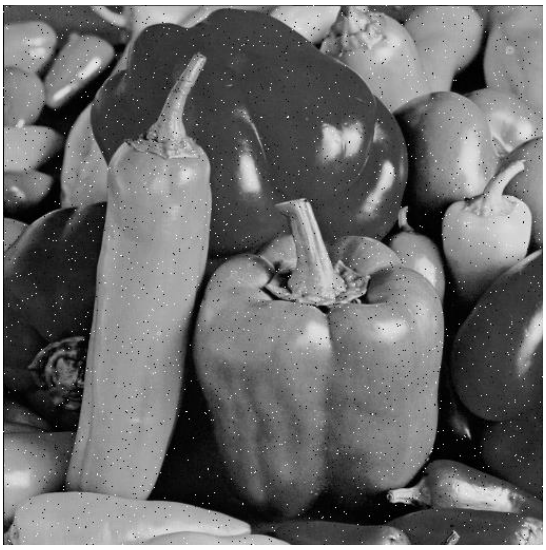


Fig6. Peppers 512x512 grayscale image

Image Source - http://www.hlevkin.com/TestImages/pepper.bmp

5.3.2. Description of obtaining secret binary data stream

Select a two-pixel block from Cover Image C. Assume gray value difference d of the two pixels is in Range. Convert given Secret image pixels to 8 bits.

Note:

Ranges are R1= [0,7], R2= [8,15], R3= [16,31], R4= [32,63], R5= [64,127], R6= [128,255]

A difference value in Range:

- R1 represents 3 bits from secret data
- R2 represents 4 bits from secret data
- R3 represents 4 bits from secret data
- R4 represents 4 bits from secret data
- R5 represents 4 bits from secret data
- R6 represents 4 bits from secret data

This is obtained by the formula $log_2(u_k - l_k + 1)$,with $u_k and\ l_k$ representing the lower and the upper bound respectively.

Calculate difference value (d) by $p_{i+1} - p_i$

Check range of d to determine how many bit(s) of secret data to be embedded.

5.3.3. Description of checking the falling-off-boundary condition using inverse difference function

Let d' be the l_k of d's Range values. Then Calculate m, m=d'-d, checking whether d is odd or even,

- If odd apply $p_i - celing(\frac{m}{2})$, $p_{i+1} - floor(\frac{m}{2})$
- If even apply $p_i - floor(\frac{m}{2})$, $p_{i+1} - celing(\frac{m}{2})$

Check resultant (p_i, p_{i+1}) if any is negative or >255 then embedding cannot happen

5.3.4. Description of embedding using inverse difference function

Calculate d' according to d:

d'= $l_k + bit(s)$, d>= 0
d'= $l_k + bit(s)$, d<0

Then calculate m, m=d'-d followed by checking whether d is odd or even,

- If odd apply $p_i - celing(\frac{m}{2})$, $p_{i+1} - floor(\frac{m}{2})$
- If even apply $p_i - floor(\frac{m}{2})$, $p_{i+1} - celing(\frac{m}{2})$
-

5.3.5. Description of extraction implementation including checking the falling-off-boundary condition using inverse difference function

For extraction calculate d*= - $p_{i+1} - p_i$, then obtain the bits that were previously embedded by the following:

- Bit(s) = d* $- l_k$, d* >= 0
- Bit(s) = -d*$- l_k$, d*<0

5.3.6. Description of RMSE, PSNR, and embedding capacity calculation

RMSE is the square rooted Mean Square Error. The MSE is the cumulative squared error between the compressed and the original image whereas PSNR represents a measure of the peak error, if its greater or equal to 30 then image is in good condition. Embedding Capacity. The embedding capacity (EC) in the code is calculated by first obtaining the pixels of the embedded image (EI) using the EI.get(). Then using bit-count to obtain how many bits each pixel contains and summing them up to get Embedding Capacity. Below is a code extract of the aforementioned processes.

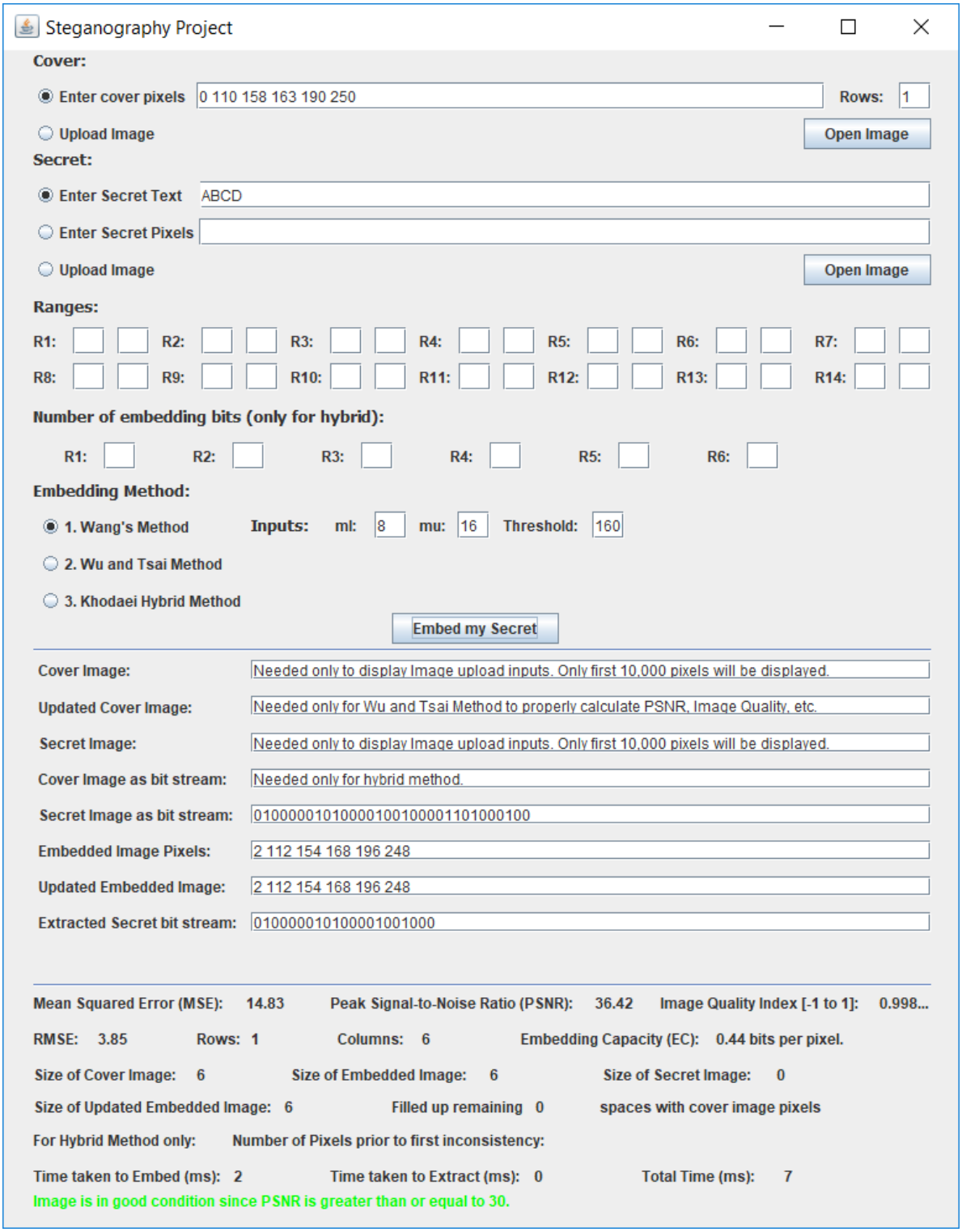
```
for(int i=0;i<CIUpdated.size();i++)
{
	MSEnumerator+=(Math.pow(Math.abs(CIUpdated.get(i)-EI.get(i)),2));
}
	MSE = MSEnumerator/(rows*columns);
	RMSE = Math.sqrt(MSE);
	System.out.println("\nMSE: " +MSE +"\nRMSE: " +RMSE);
	for(int i=0;i<EI.size();i++)
	{
		EC+=bitCount(EI.get(i));
	}
	System.out.println("Embedding Capacity: " +EC);

	PSNR = 10*Math.log10(255*255/MSE);
	System.out.println("PSNR: " +PSNR);
	if(PSNR>=30)
		System.out.println("Image is in good condition since PSNR is greater
   than or equal to 30.");
	else
		System.out.println("Image is in bad condition since PSNR is less than
   30.");
```

6.0. Description of the tests conducted and their results, screenshots of them

6.0.1. Wang's method:

Inputs: Cover pixels: 0,110,158,163,190,250, Secret Text: ABCD, Ml: 8 , MU:16 Threshold:160

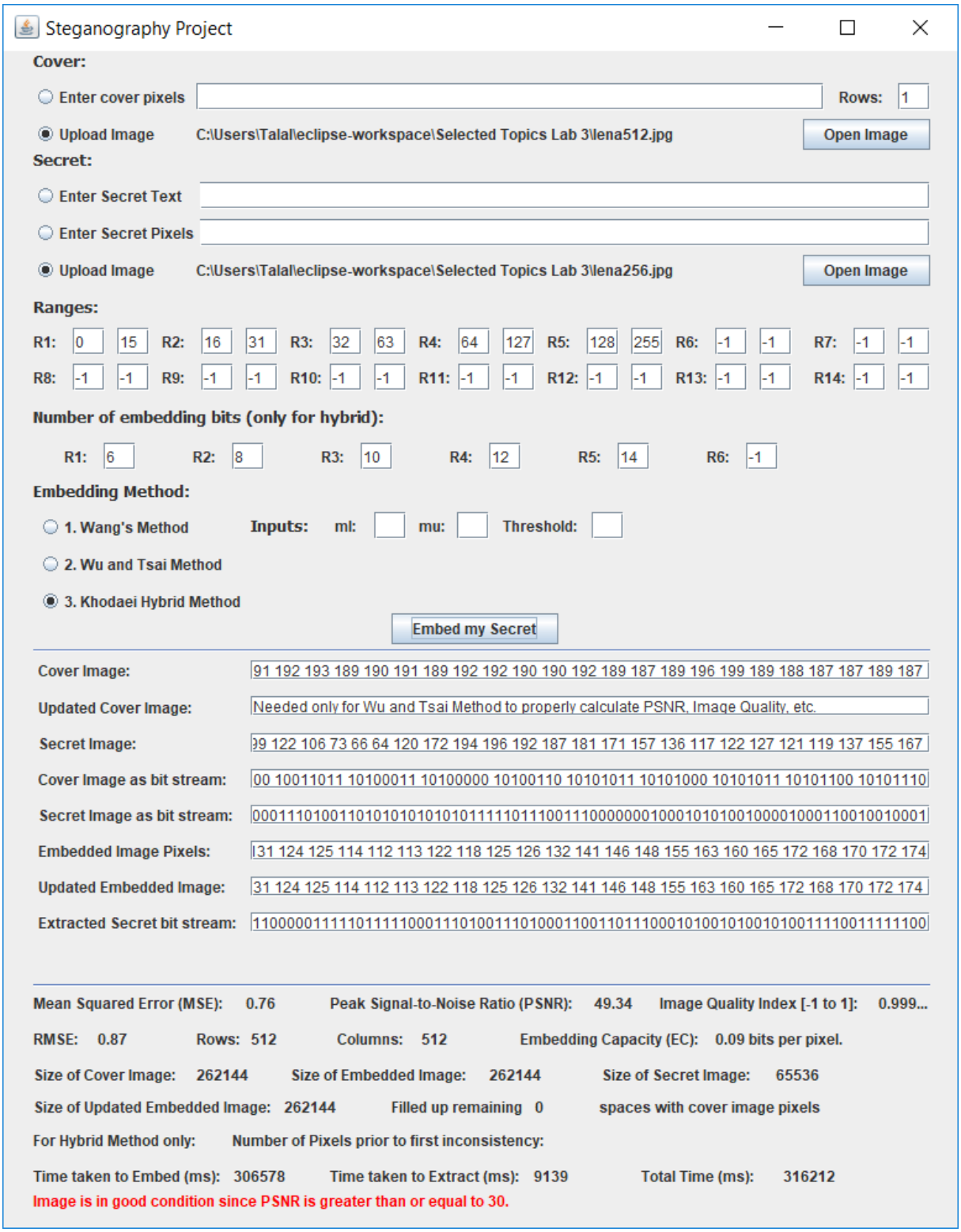
Expected results: 2, 112,154,168,196,248



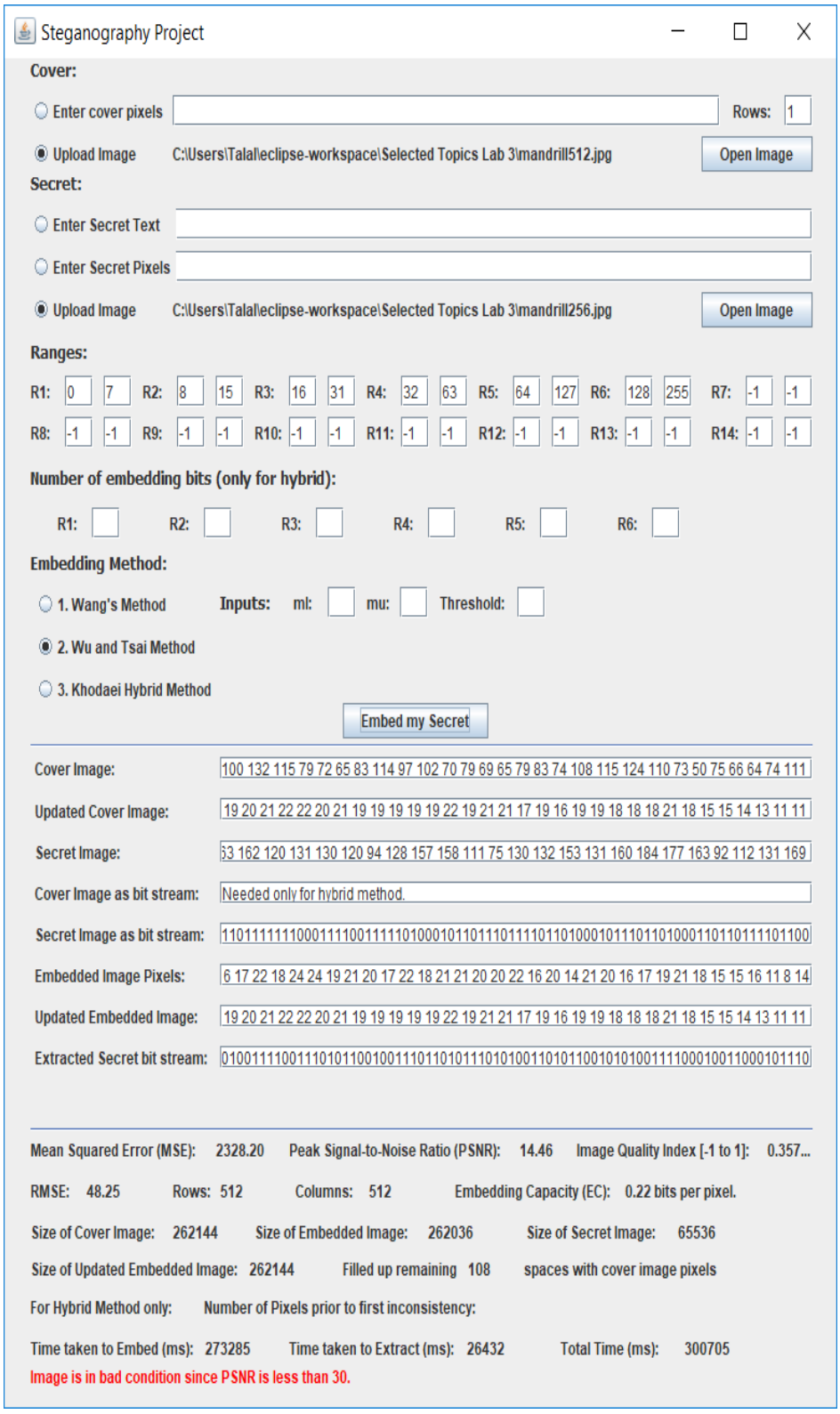
6.0.2. Wu's and Tsai's method

Inputs: Secret: Lena Image 256x256, Cover Image: Lena Image 512x512

Result (screenshot)



6.0.3. Khodaei's Hybrid method



7. Comparison of methods

7.1. Wang Method

The following results were obtained as a result of tests on Lena grayscale image Peppers grayscale, Mandrill grayscale with the covers being the same grayscale with pixels 512x512 and the secret being the same grayscale with pixels 256x256.

Table 2. The comparisons of the results between the adaptive LSB method (Yang, Weng, and Wang 2008) and the modulus function method (Lee and Chen 2010).

	Capacity (bit)			PSNR (db)		
Cover Image	Modulus Function	Adaptive LSB	Proposed method	Modulus function	Adaptive LSB	Proposed method
Elaine	786432	800188	837814	37.89	37.56	37.74
Lena	786432	817032	839028	37.91	37.74	37.67
Peppers	786432	807388	822042	37.92	37.05	37.13
Scene	786432	836338	851518	36.28	35.86	36.02
Zelda	786432	789836	799836	37.93	38.12	38.19
Tiffany	786432	811592	834812	37.93	36.68	36.81
Average	786432	810395	830841	37.64	37.17	37.26

Results:

Case (i)	Threshold(T)	ml	mu	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
1	32	12	16	37.78	0.995	0.25	156081
2	64	16	24	34.27	0.989	0.25	155087
3	96	20	32	32.73	0.985	0.25	151435
4	128	24	40	30.85	0.978	0.25	205572
5	160	28	48	29.28	0.969	0.25	169688
6	192	32	56	27.73	0.957	0.25	176020
7	224	36	64	30.31	0.975	0.25	151006

Peppers Image Results:

Case (i)	Threshold(T)	ml	mu	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
1	32	12	16	37.78	0.998	0.25	48322
2	64	16	24	33.84	0.996	0.25	48757
3	96	20	32	33.53	0.995	0.25	47252
4	128	24	40	31.99	0.994	0.25	47234
5	160	28	48	31.70	0.993	0.25	47602
6	192	32	56	31.70	0.992	0.25	46970
7	224	36	64	30.95	0.992	0.25	46396

Mandrill Image Results:

Case (i)	Threshold(T)	ml	mu	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
1	32	12	16	37.78	0.996	0.25	50205
2	64	16	24	34.08	0.993	0.25	54024
3	96	20	32	32.76	0.990	0.25	48798
4	128	24	40	31.99	0.988	0.25	49351
5	160	28	48	30.76	0.984	0.25	50259
6	192	32	56	32.06	0.988	0.25	48365
7	224	36	64	31.74	0.987	0.25	48407

7.2. Wu's and Tsai's method

Table 2
Values of RMSEs and PSNRs of stego-images in which a file consisting of the text of this article is embedded using two sets of range widths in the embedding process

Cover image	Embedding using the range widths of 8, 8, 16, 32, 64, and 128		Embedding using the range widths of 2, 2, 4, 4, 4, 8, 8, 16, 16, 32, 32, 64, and 64	
	RMSE	PSNR	RMSE	PSNR
Lena	2.07	41.79	0.97	48.43
Jet	2.28	40.97	1.09	45.67
Peppers	2.09	41.73	1.20	47.19
Baboon	3.25	37.90	1.59	44.10

Pepper:

Case (Ranges)	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
0,31,32,63,64,127,128,255	11.51	0.292	0.292	93250
0,31,32,63,64,127,128,255	10.12	0.062	0.19	91459
0,1,2,3,4,7,8,11,12,15,16,23,24,31,32,47,48,63,64,95,96,127, 128,192,255	10.22	0.086	0.10	87887

Mandrill:

Case (Ranges)	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
0,31,32,63,64,127,128,255	14.55	0.372	0.23	97718
0,7,8,15,16,31,32,63,64,127,128,255	14.46	0.357	0.357	98086
0,1,2,3,4,7,8,11,12,15,16,23,24,31,32,47,48,63,64,95,96,127,128,192,255	43.18	0.999	0.14	94982

Pepper:

Case (Ranges)	PSNR	Image Quality	Embedding Capacity	Total Time (ms)
0,31,32,63,64,127,128,255	16.57	0.416	0.24	357724
0,7,8,15,16,31,32,63,64,127,128,255	18.94	0.654	0.19	333652
0,1,2,3,4,7,8,11,12,15,16,23,24,31,32,47,48,63,64,95,96,127,128,192,255	49.34	0.99	0.09	316212

7.3. Khodaei's Hybrid method:

Table 3. The comparisons of the results between the adaptive LSB method (Yang, Weng, and Wang 2008) and the modulus function method (Lee and Chen 2010).

	Image Quality (Q)		
Cover Image	Modulus Function	Adaptive LSB	Proposed method
Elaine	0.9975	0.9972	0.9973
Lena	0.9976	0.9969	0.9971
Peppers	0.9980	0.9975	0.9977
Scene	0.9985	0.9978	0.9980
Zelda	0.9969	0.9968	0.9969
Tiffany	0.9892	0.9889	0.9891
Average	0.9962	0.9958	0.9960

Table 2. The comparisons of the results between the adaptive LSB method (Yang, Weng, and Wang 2008) and the modulus function method (Lee and Chen 2010).

	Capacity (bit)			PSNR (db)		
Cover Image	Modulus Function	Adaptive LSB	Proposed method	Modulus function	Adaptive LSB	Proposed method
Elaine	786432	800188	837814	37.89	37.56	37.74
Lena	786432	817032	839028	37.91	37.74	37.67
Peppers	786432	807388	822042	37.92	37.05	37.13
Scene	786432	836338	851518	36.28	35.86	36.02
Zelda	786432	789836	799836	37.93	38.12	38.19
Tiffany	786432	811592	834812	37.93	36.68	36.81
Average	786432	810395	830841	37.64	37.17	37.26

Pepper:

Case (Ranges)	Embedding Bits	PSNR	Image Quality	Embedding Capacity
0,15,16,63,64,127,128,255	8,10,12,14	48.51	0.99	0.01
0,15,16,31,32,63,64, 127,128,255	6,8,10,12,14	48.51	0.99	0.01
0, 15,16 ,255	8,10	48.51	0.99	0.01
0,15,16,31,32,255	8,10,12	48.51	0.99	0.01

Lena:

Case (Ranges)	Embedding Bits	PSNR	Image Quality	Embedding Capacity
0,15,16,63,64,127,128,255	8,10,12,14	49.34	0.99	0.09
0,15,16,31,32,63,64, 127,128,255	6,8,10,12,14	49.34	0.99	0.09
0, 15,16 ,255	8,10	49.34	0.99	0.09
0,15,16,31,32,255	8,10,12	49.34	0.99	0.09

Mandrill:

Case (Ranges)	Embedding Bits	PSNR	Image Quality	Embedding Capacity
0,15,16,63,64,127,128,255	8,10,12,14	14.46	0.357	0.22
0,15,16,31,32,63,64, 127,128,255	6,8,10,12,14	14.46	0.357	0.22
0, 15,16 ,255	8,10	14.46	0.357	0.22
0,15,16,31,32,255	8,10,12	14.46	0.357	0.22

8. Results Interpretation:

The results show an interesting trend for Wang's method in the sense that whenever the "I" value or the case number or the threshold increases the PSNR value decreases gradually. However, Wang's method has no effect on the embedding capacity always, it remains constant. Moreover, in the case of Wu's and Tsai's method, the more ranges included as the inputs of the algorithm the less the PSNR and image quality will be. The difference between Wang's and Wu's method has a varying embedding capacity as a result. Wu's method implements a checking algorithm before embedding the secret, this makes it more efficient. In the case of Hybrid method, we observed a pattern whereby the PSNR values where almost identical across varying ranges and the given embedding bits. This is because the ranges are close to each other i.e [8,10] and [8,10,12], so we aren't trying to embed various differing bits as in Wang's and Wu and Tsai's method.

9.0. Conclusion

In conclusion, we implemented three data embedding techniques- Wang's, Wu's and Tsai's and Hybrid method. Performed several test cases and compared the results obtained through factors such as PSNR, Embedding quality and time taken to embed and extract.

References

1. S.-J. Wang, Steganography of capacity required using modulo operator for embedding secret image, Applied Mathematics and Computation, 164 (2005), 99-116, doi:10.1016/j.amc.2004.04.059, http://cmpe.emu.edu.tr/en/CourseLoad.aspx?id=CMSE492&page=lecturenotes
2. D.-C. Wu, W.-H. Tsai, A steganographic method for images by pixel-value differencing, Pattern Recognition Letters 24 (2003) 1613–1626, http://cmpe.emu.edu.tr/en/CourseLoad.aspx?id=CMSE492&page=lecturenotes
3. M. Khodaei, B. S. Bigham, K. Faez, Adaptive, data hiding, using pixel-value-differencing and LSB substitution, Cybernetics and Systems, 47:8 (2016) 617-628, DOI: 10.1080/01969722.2016.1214459,, http://cmpe.emu.edu.tr/en/CourseLoad.aspx?id=CMSE492&page=lecturenotes

Appendix

```
import java.awt.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.awt.event.ActionEvent;

public class UIMain {

	private JFrame frmSteganographyProject;
	private JTextField txtFieldCover;
	private JTextField txtFieldSecretText;
	private JTextField txtFieldSecretPixels;
	private final ButtonGroup buttonGroupCover = new ButtonGroup();
	private final ButtonGroup buttonGroupSecret = new ButtonGroup();
	private JTextField txtFieldR1_1;
	private JTextField txtFieldR1_2;
	private JTextField txtFieldR2_1;
	private JTextField txtFieldR2_2;
	private JTextField txtFieldR3_1;
	private JTextField txtFieldR3_2;
	private JTextField txtFieldR4_1;
	private JTextField txtFieldR4_2;
```

```
    private JTextField txtFieldR5_1;
    private JTextField txtFieldR5_2;
    private JTextField txtFieldR6_1;
    private JTextField txtFieldR6_2;
    private JTextField txtFieldMl;
    private JTextField txtFieldMu;
    private JTextField txtFieldThreshold;
    private JTextField txtFieldRows;
    private final ButtonGroup buttonGroupMethod = new ButtonGroup();


    static ArrayList<Integer> CI = new ArrayList<>(); //Cover Image
    static ArrayList<Integer> SI = new ArrayList<>(); //Secret Image
    static ArrayList<Integer> EI = new ArrayList<>(); //Embedded Image
    static ArrayList<Integer> CIUpdated = new ArrayList<>(); //Updated Cover Image (Wu)
    static ArrayList<Integer> EIUpdated = new ArrayList<>(); //Updated Embedded Image
(Hybrid)
    static List<Integer> RA = new ArrayList<>(); //Re-Adjust Numbers (Hybrid)
    static List<String> CIStream = new ArrayList<>(); //Cover Image Stream (Hybrid)
    static int mu, ml, T, rows, columns, pixel, EC, D, RES, DEC, AV, p, q, startStr,
pixelCounter, n, k, p1, q1, p2, q2, p3, q3, numberBeforeInconsist, coverBits;
    static int label1, label2, label3, label4, label5, label6, label7, label8, minLabel;
    static int beginBs, endBs;
    static int width, height, rgb; //image variables
    static int R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1,
R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1,
R14_2;
    static int bit1, bit2, bit3, bit4, bit5, bit6;
    static int checkResult=1, dCheck, numOfBitsCheck, bCheck, dPrimeCheck,
pixelOneCheck, pixelTwoCheck, pixelOneEmbedCheck, pixelTwoEmbedCheck, beginBsCheck,
endBsCheck; //checking variables
    static int secretNum, numOfBits, d, b, dPrime, pixelOne, pixelTwo, pixelOneEmbed,
pixelTwoEmbed; //embedding variables
```

```
static int dExtracted, bExtracted; //extraction variables

static double m, mCheck, MSE, RMSE, MSEnumerator, PSNR, embeddingCapacity, Q, N, xBar=0, yBar=0, sigmaX=0, sigmaY=0, sigmaXY=0;

static String secretText, Bs, Cs, subBs, extractedBs, tempExtractedBs, embeddedPixels, coverPixels, secretPixels, coverAsBits, subBsCheck, intToBin, updatedCover, updatedEmbedded, extraction, tempP1, tempP2, tempString, tempString1;

static StringBuilder str = new StringBuilder();

static StringBuilder str1 = new StringBuilder();

private JTextField txtFieldR1Bits;

private JTextField txtFieldR2Bits;

private JTextField txtFieldR3Bits;

private JTextField txtFieldR4Bits;

private JTextField txtFieldR5Bits;

private JTextField txtFieldR6Bits;

private JTextField txtFieldR7_1;

private JTextField txtFieldR7_2;

private JTextField txtFieldR8_1;

private JTextField txtFieldR8_2;

private JTextField txtFieldR9_1;

private JTextField txtFieldR9_2;

private JTextField txtFieldR10_1;

private JTextField txtFieldR10_2;

private JTextField txtFieldR11_1;

private JTextField txtFieldR11_2;

private JTextField txtFieldR12_1;

private JTextField txtFieldR12_2;

private JTextField txtFieldR13_1;

private JTextField txtFieldR13_2;

private JTextField txtFieldR14_1;

private JTextField txtFieldR14_2;
```

```
public static double log2(double a)
{
            return Math.log(a)/Math.log(2);
}

public static String stringToBinary(String s)
{
  byte[] bytes = s.getBytes();
  StringBuilder binary = new StringBuilder();
  for (byte b : bytes)
  {
     int val = b;
     for (int i = 0; i < 8; i++)
     {
        binary.append((val & 128) == 0 ? 0 : 1);
        val <<= 1;
     }
  }
  return binary.toString();
}

public static int binaryToInteger(String binary)
{
    char[] numbers = binary.toCharArray();
    int result = 0;
    for(int i=numbers.length - 1; i>=0; i--)
        if(numbers[i]=='1')
            result += Math.pow(2, (numbers.length-i - 1));
    return result;
}
```

```
public static String intToBinary(int n, int numOfBits)
{
        String binary = "";
        for(int i = 0; i < numOfBits; ++i, n/=2)
        {
          switch (n % 2)
          {
            case 0:
              binary = "0" + binary;
              break;
            case 1:
              binary = "1" + binary;
              break;
          }
        }

        return binary;
}

    public static int rangeFinder(int d, int R1_1, int R1_2, int R2_1, int R2_2, int R3_1, int R3_2, int R4_1, int R4_2, int R5_1, int R5_2, int R6_1, int R6_2, int R7_1, int R7_2, int R8_1, int R8_2, int R9_1, int R9_2, int R10_1, int R10_2, int R11_1, int R11_2, int R12_1, int R12_2, int R13_1, int R13_2, int R14_1, int R14_2)
    {
        int numOfBits=0;

        if((d>=R1_1 && d<=R1_2) || (d<=R1_1 && d>=-R1_2))
            numOfBits= (int) log2(R1_2-R1_1+1);
        else if((d>=R2_1 && d<=R2_2) || (d<=-R2_1 && d>=-R2_2))
```

```
            numOfBits= (int) log2(R2_2-R2_1+1);
        else if((d>=R3_1 && d<=R3_2) || (d<=-R3_1 && d>=-R3_2))
            numOfBits= (int) log2(R3_2-R3_1+1);
        else if((d>=R4_1 && d<=R4_2) || (d<=-R4_1 && d>=-R4_2))
            numOfBits= (int) log2(R4_2-R4_1+1);
        else if((d>=R5_1 && d<=R5_2) || (d<=-R5_1 && d>=-R5_2))
            numOfBits= (int) log2(R5_2-R5_1+1);
        else if((d>=R6_1 && d<=R6_2) || (d<=-R6_1 && d>=-R6_2))
            numOfBits= (int) log2(R6_2-R6_1+1);
        else if((d>=R7_1 && d<=R7_2) || (d<=-R7_1 && d>=-R7_2))
            numOfBits= (int) log2(R7_2-R7_1+1);
        else if((d>=R8_1 && d<=R8_2) || (d<=-R8_1 && d>=-R8_2))
            numOfBits= (int) log2(R8_2-R8_1+1);
        else if((d>=R9_1 && d<=R9_2) || (d<=-R9_1 && d>=-R9_2))
            numOfBits= (int) log2(R9_2-R9_1+1);
        else if((d>=R10_1 && d<=R10_2) || (d<=-R10_1 && d>=-R10_2))
            numOfBits= (int) log2(R10_2-R10_1+1);
        else if((d>=R11_1 && d<=R11_2) || (d<=-R11_1 && d>=-R11_2))
            numOfBits= (int) log2(R11_2-R11_1+1);
        else if((d>=R12_1 && d<=R12_2) || (d<=-R12_1 && d>=-R12_2))
            numOfBits= (int) log2(R12_2-R12_1+1);
        else if((d>=R13_1 && d<=R13_2) || (d<=-R13_1 && d>=-R13_2))
            numOfBits= (int) log2(R13_2-R13_1+1);
        else if((d>=R14_1 && d<=R14_2) || (d<=-R14_1 && d>=-R14_2))
            numOfBits= (int) log2(R14_2-R14_1+1);

        return numOfBits;
}
```

```
public static int maxRangeFinder(int d, int R1_1, int R1_2, int R2_1, int R2_2, int R3_1, int R3_2, int R4_1, int R4_2, int R5_1, int R5_2, int R6_1, int R6_2, int R7_1, int R7_2, int R8_1, int R8_2, int R9_1, int R9_2, int R10_1, int R10_2, int R11_1, int R11_2, int R12_1, int R12_2, int R13_1, int R13_2, int R14_1, int R14_2)
{
    int maxRange=0;

    if(d>=0)
    {
    if(d>=R1_1 && d<=R1_2)
        maxRange=R1_2;
    else if(d>=R2_1 && d<=R2_2)
        maxRange=R2_2;
    else if(d>=R3_1 && d<=R3_2)
        maxRange=R3_2;
    else if(d>=R4_1 && d<=R4_2)
        maxRange=R4_2;
    else if(d>=R5_1 && d<=R5_2)
        maxRange=R5_2;
    else if(d>=R6_1 && d<=R6_2)
        maxRange=R6_2;
    else if(d>=R7_1 && d<=R7_2)
        maxRange=R7_2;
    else if(d>=R8_1 && d<=R8_2)
        maxRange=R8_2;
    else if(d>=R9_1 && d<=R9_2)
        maxRange=R9_2;
    else if(d>=R10_1 && d<=R10_2)
        maxRange=R10_2;
    else if(d>=R11_1 && d<=R11_2)
        maxRange=R11_2;
```

```
else if(d>=R12_1 && d<=R12_2)
        maxRange=R12_2;
else if(d>=R13_1 && d<=R13_2)
        maxRange=R13_2;
else if(d>=R14_1 && d<=R14_2)
        maxRange=R14_2;
}
else
{
        if(d<=R1_1 && d>=-R1_2)
                maxRange=-R1_2;
        else if(d<=-R2_1 && d>=-R2_2)
                maxRange=-R2_2;
        else if(d<=-R3_1 && d>=-R3_2)
                maxRange=-R3_2;
        else if(d<=-R4_1 && d>=-R4_2)
                maxRange=-R4_2;
        else if(d<=-R5_1 && d>=-R5_2)
                maxRange=-R5_2;
        else if(d<=-R6_1 && d>=-R6_2)
                maxRange=-R6_2;
        else if(d<=-R7_1 && d>=-R7_2)
                maxRange=-R7_2;
        else if(d<=-R8_1 && d>=-R8_2)
                maxRange=-R8_2;
        else if(d<=-R9_1 && d>=-R9_2)
                maxRange=-R9_2;
        else if(d<=-R10_1 && d>=-R10_2)
                maxRange=-R10_2;
        else if(d<=-R11_1 && d>=-R11_2)
```

```
                maxRange=-R11_2;
            else if(d<=-R12_1 && d>=-R12_2)
                maxRange=-R12_2;
            else if(d<=-R13_1 && d>=-R13_2)
                maxRange=-R13_2;
            else if(d<=-R14_1 && d>=-R14_2)
                maxRange=-R14_2;
        }
        return maxRange;
    }

    public static int minRangeFinder(int d, int R1_1, int R1_2, int R2_1, int R2_2, int R3_1, int R3_2, int R4_1, int R4_2, int R5_1, int R5_2, int R6_1, int R6_2, int R7_1, int R7_2, int R8_1, int R8_2, int R9_1, int R9_2, int R10_1, int R10_2, int R11_1, int R11_2, int R12_1, int R12_2, int R13_1, int R13_2, int R14_1, int R14_2)
    {
        int minRange=0;

        if((d>=R1_1 && d<=R1_2) || (d<=R1_1 && d>=-R1_2))
            minRange=R1_1;
        else if((d>=R2_1 && d<=R2_2) || (d<=-R2_1 && d>=-R2_2))
            minRange=R2_1;
        else if((d>=R3_1 && d<=R3_2) || (d<=-R3_1 && d>=-R3_2))
            minRange=R3_1;
        else if((d>=R4_1 && d<=R4_2) || (d<=-R4_1 && d>=-R4_2))
            minRange=R4_1;
        else if((d>=R5_1 && d<=R5_2) || (d<=-R5_1 && d>=-R5_2))
            minRange=R5_1;
        else if((d>=R6_1 && d<=R6_2) || (d<=-R6_1 && d>=-R6_2))
            minRange=R6_1;
        else if((d>=R7_1 && d<=R7_2) || (d<=-R7_1 && d>=-R7_2))
```

```
                minRange=R7_1;
        else if((d>=R8_1 && d<=R8_2) || (d<=-R8_1 && d>=-R8_2))
                minRange=R8_1;
        else if((d>=R9_1 && d<=R9_2) || (d<=-R9_1 && d>=-R9_2))
                minRange=R9_1;
        else if((d>=R10_1 && d<=R10_2) || (d<=-R10_1 && d>=-R10_2))
                minRange=R10_1;
        else if((d>=R11_1 && d<=R11_2) || (d<=-R11_1 && d>=-R11_2))
                minRange=R11_1;
        else if((d>=R12_1 && d<=R12_2) || (d<=-R12_1 && d>=-R12_2))
                minRange=R12_1;
        else if((d>=R13_1 && d<=R13_2) || (d<=-R13_1 && d>=-R13_2))
                minRange=R13_1;
        else if((d>=R14_1 && d<=R14_2) || (d<=-R14_1 && d>=-R14_2))
                minRange=R14_1;

        return minRange;
    }

    public static int rangeFinderHybrid(int d, int bit1, int bit2, int bit3, int bit4, int bit5, int
bit6, int R1_1, int R1_2, int R2_1, int R2_2, int R3_1, int R3_2, int R4_1, int R4_2, int R5_1, int
R5_2, int R6_1, int R6_2)
    {
        int numOfBits=0;

        if(d>=0 && d<=15)
                numOfBits=8;
        else if(d>=16 && d<=63)
                numOfBits=10;
        else if(d>=64 && d<=127)
                numOfBits=12;
```

```
        else if(d>=128 && d<=255)
                numOfBits=14;

        return numOfBits;
}


/**
 * Launch the application.
 */
public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
                public void run() {
                        try {
                                UIMain window = new UIMain();
                                window.frmSteganographyProject.setVisible(true);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        });
}

/**
 * Create the application.
 */
public UIMain() {
        initialize();
}
```

```
/**
 * Initialize the contents of the frame.
 */
private void initialize() {
	frmSteganographyProject = new JFrame();
	frmSteganographyProject.setTitle("Steganography Project");
	frmSteganographyProject.setBounds(100, 100, 790, 999);
	frmSteganographyProject.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frmSteganographyProject.getContentPane().setLayout(null);

	JPanel panel = new JPanel();
	panel.setBounds(12, 0, 748, 947);
	frmSteganographyProject.getContentPane().add(panel);
	panel.setLayout(null);

	JLabel lblCover = new JLabel("Cover:");
	lblCover.setFont(new Font("Tahoma", Font.BOLD, 13));
	lblCover.setBounds(12, 0, 56, 16);
	panel.add(lblCover);

	JRadioButton rdbtnCover = new JRadioButton("Enter cover pixels");
	buttonGroupCover.add(rdbtnCover);
	rdbtnCover.setSelected(true);
	rdbtnCover.setBounds(12, 25, 129, 25);
	panel.add(rdbtnCover);

	txtFieldCover = new JTextField();
	txtFieldCover.setBounds(144, 26, 508, 22);
	panel.add(txtFieldCover);
	txtFieldCover.setColumns(10);
```

```
JLabel lblRows = new JLabel("Rows:");
lblRows.setBounds(664, 29, 36, 16);
panel.add(lblRows);

txtFieldRows = new JTextField();
txtFieldRows.setText("1");
txtFieldRows.setColumns(10);
txtFieldRows.setBounds(712, 26, 26, 22);
panel.add(txtFieldRows);

JRadioButton rdbtnUploadCover = new JRadioButton("Upload Image");
buttonGroupCover.add(rdbtnUploadCover);
rdbtnUploadCover.setBounds(12, 55, 127, 25);
panel.add(rdbtnUploadCover);

JLabel lblFileCover = new JLabel("");
lblFileCover.setBounds(144, 59, 437, 16);
panel.add(lblFileCover);

final JFileChooser fileDialog = new JFileChooser();
JButton btnCover = new JButton("Open Image");
btnCover.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                int returnVal =
fileDialog.showOpenDialog(frmSteganographyProject);

if (returnVal == JFileChooser.APPROVE_OPTION) {
  java.io.File file = fileDialog.getSelectedFile();
  lblFileCover.setText(file.getAbsolutePath());
```

```
}
            }
 });
 btnCover.setBounds(635, 55, 103, 25);
 panel.add(btnCover);

 JLabel lblSecret = new JLabel("Secret:");
 lblSecret.setFont(new Font("Tahoma", Font.BOLD, 13));
 lblSecret.setBounds(12, 80, 56, 16);
 panel.add(lblSecret);

 JRadioButton rdbtnSecretText = new JRadioButton("Enter Secret Text");
 buttonGroupSecret.add(rdbtnSecretText);
 rdbtnSecretText.setSelected(true);
 rdbtnSecretText.setBounds(12, 105, 135, 25);
 panel.add(rdbtnSecretText);

 txtFieldSecretText = new JTextField();
 txtFieldSecretText.setBounds(146, 106, 592, 22);
 panel.add(txtFieldSecretText);
 txtFieldSecretText.setColumns(10);

 JRadioButton rdbtnSecretPixels = new JRadioButton("Enter Secret Pixels");
 buttonGroupSecret.add(rdbtnSecretPixels);
 rdbtnSecretPixels.setBounds(12, 135, 135, 25);
 panel.add(rdbtnSecretPixels);

 txtFieldSecretPixels = new JTextField();
 txtFieldSecretPixels.setBounds(146, 136, 592, 22);
 panel.add(txtFieldSecretPixels);
```

```
txtFieldSecretPixels.setColumns(10);

JRadioButton rdbtnUploadSecret = new JRadioButton("Upload Image");
buttonGroupSecret.add(rdbtnUploadSecret);
rdbtnUploadSecret.setBounds(12, 165, 127, 25);
panel.add(rdbtnUploadSecret);

JLabel lblFileSecret = new JLabel("");
lblFileSecret.setBounds(144, 169, 437, 16);
panel.add(lblFileSecret);

JButton btnSecret = new JButton("Open Image");
btnSecret.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                int returnVal =
fileDialog.showOpenDialog(frmSteganographyProject);

if (returnVal == JFileChooser.APPROVE_OPTION) {
  java.io.File file = fileDialog.getSelectedFile();
  lblFileSecret.setText(file.getAbsolutePath());
}
        }
});
btnSecret.setBounds(635, 165, 103, 25);
panel.add(btnSecret);

JLabel lblRanges = new JLabel("Ranges:");
lblRanges.setFont(new Font("Tahoma", Font.BOLD, 13));
lblRanges.setBounds(12, 199, 56, 16);
panel.add(lblRanges);
```

```
JLabel lblR1 = new JLabel("R1:");
lblR1.setBounds(12, 227, 20, 16);
panel.add(lblR1);

txtFieldR1_1 = new JTextField();
txtFieldR1_1.setBounds(44, 224, 26, 22);
panel.add(txtFieldR1_1);
txtFieldR1_1.setColumns(10);

txtFieldR1_2 = new JTextField();
txtFieldR1_2.setColumns(10);
txtFieldR1_2.setBounds(80, 224, 26, 22);
panel.add(txtFieldR1_2);

JLabel lblR2 = new JLabel("R2:");
lblR2.setBounds(116, 227, 20, 16);
panel.add(lblR2);

txtFieldR2_1 = new JTextField();
txtFieldR2_1.setColumns(10);
txtFieldR2_1.setBounds(148, 224, 26, 22);
panel.add(txtFieldR2_1);

txtFieldR2_2 = new JTextField();
txtFieldR2_2.setColumns(10);
txtFieldR2_2.setBounds(184, 224, 26, 22);
panel.add(txtFieldR2_2);

JLabel lblR3 = new JLabel("R3:");
```

```
lblR3.setBounds(220, 227, 20, 16);
panel.add(lblR3);

txtFieldR3_1 = new JTextField();
txtFieldR3_1.setColumns(10);
txtFieldR3_1.setBounds(252, 224, 26, 22);
panel.add(txtFieldR3_1);

txtFieldR3_2 = new JTextField();
txtFieldR3_2.setColumns(10);
txtFieldR3_2.setBounds(288, 224, 26, 22);
panel.add(txtFieldR3_2);

JLabel lblR4 = new JLabel("R4:");
lblR4.setBounds(324, 227, 20, 16);
panel.add(lblR4);

txtFieldR4_1 = new JTextField();
txtFieldR4_1.setColumns(10);
txtFieldR4_1.setBounds(356, 224, 26, 22);
panel.add(txtFieldR4_1);

txtFieldR4_2 = new JTextField();
txtFieldR4_2.setColumns(10);
txtFieldR4_2.setBounds(392, 224, 26, 22);
panel.add(txtFieldR4_2);

JLabel lblR5 = new JLabel("R5:");
lblR5.setBounds(428, 227, 20, 16);
panel.add(lblR5);
```

```
txtFieldR5_1 = new JTextField();
txtFieldR5_1.setColumns(10);
txtFieldR5_1.setBounds(460, 224, 26, 22);
panel.add(txtFieldR5_1);

txtFieldR5_2 = new JTextField();
txtFieldR5_2.setColumns(10);
txtFieldR5_2.setBounds(496, 224, 26, 22);
panel.add(txtFieldR5_2);

JLabel lblR6 = new JLabel("R6:");
lblR6.setBounds(532, 227, 20, 16);
panel.add(lblR6);

txtFieldR6_1 = new JTextField();
txtFieldR6_1.setColumns(10);
txtFieldR6_1.setBounds(564, 224, 26, 22);
panel.add(txtFieldR6_1);

txtFieldR6_2 = new JTextField();
txtFieldR6_2.setColumns(10);
txtFieldR6_2.setBounds(600, 224, 26, 22);
panel.add(txtFieldR6_2);

JLabel lblR7 = new JLabel("R7:");
lblR7.setBounds(644, 227, 20, 16);
panel.add(lblR7);

txtFieldR7_1 = new JTextField();
```

```
txtFieldR7_1.setColumns(10);
txtFieldR7_1.setBounds(676, 224, 26, 22);
panel.add(txtFieldR7_1);

txtFieldR7_2 = new JTextField();
txtFieldR7_2.setColumns(10);
txtFieldR7_2.setBounds(712, 224, 26, 22);
panel.add(txtFieldR7_2);

JLabel lblR8 = new JLabel("R8:");
lblR8.setBounds(12, 256, 20, 16);
panel.add(lblR8);

txtFieldR8_1 = new JTextField();
txtFieldR8_1.setColumns(10);
txtFieldR8_1.setBounds(44, 253, 26, 22);
panel.add(txtFieldR8_1);

txtFieldR8_2 = new JTextField();
txtFieldR8_2.setColumns(10);
txtFieldR8_2.setBounds(80, 253, 26, 22);
panel.add(txtFieldR8_2);

JLabel lblR9 = new JLabel("R9:");
lblR9.setBounds(116, 256, 20, 16);
panel.add(lblR9);

txtFieldR9_1 = new JTextField();
txtFieldR9_1.setColumns(10);
txtFieldR9_1.setBounds(148, 253, 26, 22);
```

```
panel.add(txtFieldR9_1);

txtFieldR9_2 = new JTextField();
txtFieldR9_2.setColumns(10);
txtFieldR9_2.setBounds(184, 253, 26, 22);
panel.add(txtFieldR9_2);

JLabel lblR10 = new JLabel("R10:");
lblR10.setBounds(220, 256, 30, 16);
panel.add(lblR10);

txtFieldR10_1 = new JTextField();
txtFieldR10_1.setColumns(10);
txtFieldR10_1.setBounds(252, 253, 26, 22);
panel.add(txtFieldR10_1);

txtFieldR10_2 = new JTextField();
txtFieldR10_2.setColumns(10);
txtFieldR10_2.setBounds(288, 253, 26, 22);
panel.add(txtFieldR10_2);

JLabel lblR11 = new JLabel("R11:");
lblR11.setBounds(324, 256, 30, 16);
panel.add(lblR11);

txtFieldR11_1 = new JTextField();
txtFieldR11_1.setColumns(10);
txtFieldR11_1.setBounds(356, 253, 26, 22);
panel.add(txtFieldR11_1);
```

```
txtFieldR11_2 = new JTextField();
txtFieldR11_2.setColumns(10);
txtFieldR11_2.setBounds(392, 253, 26, 22);
panel.add(txtFieldR11_2);

JLabel lblR12 = new JLabel("R12:");
lblR12.setBounds(428, 256, 30, 16);
panel.add(lblR12);

txtFieldR12_1 = new JTextField();
txtFieldR12_1.setColumns(10);
txtFieldR12_1.setBounds(460, 253, 26, 22);
panel.add(txtFieldR12_1);

txtFieldR12_2 = new JTextField();
txtFieldR12_2.setColumns(10);
txtFieldR12_2.setBounds(496, 253, 26, 22);
panel.add(txtFieldR12_2);

JLabel lblR13 = new JLabel("R13:");
lblR13.setBounds(532, 256, 30, 16);
panel.add(lblR13);

txtFieldR13_1 = new JTextField();
txtFieldR13_1.setColumns(10);
txtFieldR13_1.setBounds(564, 253, 26, 22);
panel.add(txtFieldR13_1);

txtFieldR13_2 = new JTextField();
txtFieldR13_2.setColumns(10);
```

```
txtFieldR13_2.setBounds(600, 253, 26, 22);
panel.add(txtFieldR13_2);

JLabel lblR14 = new JLabel("R14:");
lblR14.setBounds(644, 256, 30, 16);
panel.add(lblR14);

txtFieldR14_1 = new JTextField();
txtFieldR14_1.setColumns(10);
txtFieldR14_1.setBounds(676, 253, 26, 22);
panel.add(txtFieldR14_1);

txtFieldR14_2 = new JTextField();
txtFieldR14_2.setColumns(10);
txtFieldR14_2.setBounds(712, 253, 26, 22);
panel.add(txtFieldR14_2);

JLabel lblNumberOfEmbedding = new JLabel("Number of embedding bits (only for hybrid):");
lblNumberOfEmbedding.setFont(new Font("Tahoma", Font.BOLD, 13));
lblNumberOfEmbedding.setBounds(12, 288, 293, 16);
panel.add(lblNumberOfEmbedding);

JLabel lblR1Bits = new JLabel("R1:");
lblR1Bits.setBounds(37, 320, 20, 16);
panel.add(lblR1Bits);

txtFieldR1Bits = new JTextField();
txtFieldR1Bits.setColumns(10);
txtFieldR1Bits.setBounds(69, 317, 26, 22);
```

```
panel.add(txtFieldR1Bits);

JLabel lblR2Bits = new JLabel("R2:");
lblR2Bits.setBounds(141, 320, 20, 16);
panel.add(lblR2Bits);

txtFieldR2Bits = new JTextField();
txtFieldR2Bits.setColumns(10);
txtFieldR2Bits.setBounds(173, 317, 26, 22);
panel.add(txtFieldR2Bits);

JLabel lblR3Bits = new JLabel("R3:");
lblR3Bits.setBounds(245, 320, 20, 16);
panel.add(lblR3Bits);

txtFieldR3Bits = new JTextField();
txtFieldR3Bits.setColumns(10);
txtFieldR3Bits.setBounds(277, 317, 26, 22);
panel.add(txtFieldR3Bits);

JLabel lblR4Bits = new JLabel("R4:");
lblR4Bits.setBounds(349, 320, 20, 16);
panel.add(lblR4Bits);

txtFieldR4Bits = new JTextField();
txtFieldR4Bits.setColumns(10);
txtFieldR4Bits.setBounds(381, 317, 26, 22);
panel.add(txtFieldR4Bits);

JLabel lblR5Bits = new JLabel("R5:");
```

```
lblR5Bits.setBounds(453, 320, 20, 16);
panel.add(lblR5Bits);

txtFieldR5Bits = new JTextField();
txtFieldR5Bits.setColumns(10);
txtFieldR5Bits.setBounds(485, 317, 26, 22);
panel.add(txtFieldR5Bits);

JLabel lblR6Bits = new JLabel("R6:");
lblR6Bits.setBounds(557, 320, 20, 16);
panel.add(lblR6Bits);

txtFieldR6Bits = new JTextField();
txtFieldR6Bits.setColumns(10);
txtFieldR6Bits.setBounds(589, 317, 26, 22);
panel.add(txtFieldR6Bits);

JLabel lblInputs = new JLabel("Inputs:");
lblInputs.setFont(new Font("Tahoma", Font.BOLD, 13));
lblInputs.setBounds(188, 376, 56, 16);
panel.add(lblInputs);

JLabel lblMl = new JLabel("ml:");
lblMl.setBounds(256, 376, 26, 16);
panel.add(lblMl);

txtFieldMl = new JTextField();
txtFieldMl.setColumns(10);
txtFieldMl.setBounds(288, 373, 26, 22);
panel.add(txtFieldMl);
```

```
JLabel lblMu = new JLabel("mu:");
lblMu.setBounds(324, 376, 26, 16);
panel.add(lblMu);

txtFieldMu = new JTextField();
txtFieldMu.setColumns(10);
txtFieldMu.setBounds(355, 373, 26, 22);
panel.add(txtFieldMu);

JLabel lblThreshold = new JLabel("Threshold:");
lblThreshold.setBounds(392, 376, 62, 16);
panel.add(lblThreshold);

txtFieldThreshold = new JTextField();
txtFieldThreshold.setColumns(10);
txtFieldThreshold.setBounds(464, 373, 26, 22);
panel.add(txtFieldThreshold);

JLabel lblEmbeddingMethod = new JLabel("Embedding Method:");
lblEmbeddingMethod.setFont(new Font("Tahoma", Font.BOLD, 13));
lblEmbeddingMethod.setBounds(12, 348, 129, 16);
panel.add(lblEmbeddingMethod);

JRadioButton rdbtnWang = new JRadioButton("1. Wang's Method");
buttonGroupMethod.add(rdbtnWang);
rdbtnWang.setSelected(true);
rdbtnWang.setBounds(16, 373, 149, 25);
panel.add(rdbtnWang);
```

```
JRadioButton rdbtnWu = new JRadioButton("2. Wu and Tsai Method");
buttonGroupMethod.add(rdbtnWu);
rdbtnWu.setBounds(16, 403, 162, 25);
panel.add(rdbtnWu);

JRadioButton rdbtnHybrid = new JRadioButton("3. Khodaei Hybrid Method");
buttonGroupMethod.add(rdbtnHybrid);
rdbtnHybrid.setBounds(16, 433, 175, 25);
panel.add(rdbtnHybrid);

JLabel lblCoverImage = new JLabel("Cover Image:");
lblCoverImage.setBounds(16, 493, 94, 16);
panel.add(lblCoverImage);

JTextField txtOutput1CoverPixels = new JTextField("Needed only to display Image upload inputs. Only first 10,000 pixels will be displayed.");
txtOutput1CoverPixels.setBounds(188, 493, 550, 16);
panel.add(txtOutput1CoverPixels);

JLabel lblUpdatedCover = new JLabel("Updated Cover Image:");
lblUpdatedCover.setBounds(16, 523, 140, 16);
panel.add(lblUpdatedCover);

JTextField txtOutput2UpdatedCover = new JTextField("Needed only for Wu and Tsai Method to properly calculate PSNR, Image Quality, etc.");
txtOutput2UpdatedCover.setBounds(188, 522, 550, 16);
panel.add(txtOutput2UpdatedCover);

JLabel lblSecretImage = new JLabel("Secret Image:");
lblSecretImage.setBounds(16, 552, 82, 16);
panel.add(lblSecretImage);
```

```
JTextField txtOutput3SecretPixels = new JTextField("Needed only to display Image upload inputs. Only first 10,000 pixels will be displayed.");
txtOutput3SecretPixels.setBounds(188, 552, 550, 16);
panel.add(txtOutput3SecretPixels);

JLabel lblCoverImageAs = new JLabel("Cover Image as bit stream:");
lblCoverImageAs.setBounds(16, 581, 162, 16);
panel.add(lblCoverImageAs);

JTextField txtOutput4CoverBits = new JTextField("Needed only for hybrid method.");
txtOutput4CoverBits.setBounds(188, 581, 550, 16);
panel.add(txtOutput4CoverBits);

JLabel lblSecretImageAs = new JLabel("Secret Image as bit stream:");
lblSecretImageAs.setBounds(16, 610, 162, 16);
panel.add(lblSecretImageAs);

JTextField txtOutput5SecretBits = new JTextField("");
txtOutput5SecretBits.setBounds(188, 610, 550, 16);
panel.add(txtOutput5SecretBits);

JLabel lblEmbeddedImagePixels = new JLabel("Embedded Image Pixels:");
lblEmbeddedImagePixels.setBounds(16, 639, 149, 16);
panel.add(lblEmbeddedImagePixels);

JTextField txtOutput6EmbedPixels = new JTextField("");
txtOutput6EmbedPixels.setBounds(188, 639, 550, 16);
panel.add(txtOutput6EmbedPixels);
```

```
JLabel lblUpdatedEmbedded = new JLabel("Updated Embedded Image:");
lblUpdatedEmbedded.setBounds(16, 668, 162, 16);
panel.add(lblUpdatedEmbedded);

JTextField txtOutput7UpdatedEmbedded = new JTextField("Needed only to properly calculate PSNR, Image Quality, etc.");
txtOutput7UpdatedEmbedded.setBounds(188, 668, 550, 16);
panel.add(txtOutput7UpdatedEmbedded);

JLabel lblExtractedSecretAs = new JLabel("Extracted Secret bit stream:");
lblExtractedSecretAs.setBounds(16, 697, 162, 16);
panel.add(lblExtractedSecretAs);

JTextField txtOutput8ExtractedBits = new JTextField("");
txtOutput8ExtractedBits.setBounds(188, 697, 550, 16);
panel.add(txtOutput8ExtractedBits);

JLabel lblNote = new JLabel("");
lblNote.setForeground(Color.RED);
lblNote.setBounds(12, 726, 688, 16);
panel.add(lblNote);

JSeparator separator_1 = new JSeparator();
separator_1.setBounds(12, 755, 726, 2);
panel.add(separator_1);

JLabel lblMeanSquareError = new JLabel("Mean Squared Error (MSE):");
lblMeanSquareError.setBounds(12, 762, 162, 16);
panel.add(lblMeanSquareError);
```

```
JLabel lblOutput9MSE = new JLabel("");
lblOutput9MSE.setBounds(184, 762, 56, 16);
panel.add(lblOutput9MSE);

JLabel lblPsnr = new JLabel("Peak Signal-to-Noise Ratio (PSNR):");
lblPsnr.setBounds(252, 762, 211, 16);
panel.add(lblPsnr);

JLabel lblOutput10PSNR = new JLabel("");
lblOutput10PSNR.setBounds(466, 762, 46, 16);
panel.add(lblOutput10PSNR);

JLabel lblRows_Out = new JLabel("Rows:");
lblRows_Out.setBounds(144, 791, 36, 16);
panel.add(lblRows_Out);

JLabel lblOutput11Rows = new JLabel("");
lblOutput11Rows.setBounds(188, 791, 56, 16);
panel.add(lblOutput11Rows);

JLabel lblColumns = new JLabel("Columns:");
lblColumns.setBounds(258, 791, 56, 16);
panel.add(lblColumns);

JLabel lblOutput12Columns = new JLabel("");
lblOutput12Columns.setBounds(326, 791, 56, 16);
panel.add(lblOutput12Columns);

JLabel lblEC = new JLabel("Embedding Capacity (EC):");
lblEC.setBounds(406, 791, 156, 16);
```

```
panel.add(lblEC);

JLabel lblOutput13EC = new JLabel("");
lblOutput13EC.setBounds(564, 791, 174, 16);
panel.add(lblOutput13EC);

JLabel lblSizeOfCover = new JLabel("Size of Cover Image:");
lblSizeOfCover.setBounds(12, 820, 125, 16);
panel.add(lblSizeOfCover);

JLabel lblOutput14CoverSize = new JLabel("");
lblOutput14CoverSize.setBounds(144, 820, 56, 16);
panel.add(lblOutput14CoverSize);

JLabel lblSizeOfEmbedded = new JLabel("Size of Embedded Image:");
lblSizeOfEmbedded.setBounds(220, 820, 156, 16);
panel.add(lblSizeOfEmbedded);

JLabel lblOutput15EmbedSize = new JLabel("");
lblOutput15EmbedSize.setBounds(381, 820, 56, 16);
panel.add(lblOutput15EmbedSize);

JLabel lblSizeOfUpdated = new JLabel("Size of Updated Embedded Image:");
lblSizeOfUpdated.setBounds(12, 846, 209, 16);
panel.add(lblSizeOfUpdated);

JLabel lblOutput16UpdateEmbed = new JLabel("");
lblOutput16UpdateEmbed.setBounds(215, 846, 56, 16);
panel.add(lblOutput16UpdateEmbed);
```

```
JLabel lblFilledUpRemaining = new JLabel("Filled up remaining");
lblFilledUpRemaining.setBounds(302, 846, 109, 16);
panel.add(lblFilledUpRemaining);

JLabel lblOutput17RemainSpace = new JLabel("");
lblOutput17RemainSpace.setBounds(418, 846, 46, 16);
panel.add(lblOutput17RemainSpace);

JLabel lblSpacesWithCover = new JLabel("spaces with cover image pixels");
lblSpacesWithCover.setBounds(470, 846, 194, 16);
panel.add(lblSpacesWithCover);

JLabel lblForHybridMethod = new JLabel("For Hybrid Method only:");
lblForHybridMethod.setBounds(12, 873, 149, 16);
panel.add(lblForHybridMethod);

JLabel lblNumberOfPixels = new JLabel("Number of Pixels prior to first inconsistency:");
lblNumberOfPixels.setBounds(173, 873, 261, 16);
panel.add(lblNumberOfPixels);

JLabel lblOutput18PriorToFirst = new JLabel("");
lblOutput18PriorToFirst.setBounds(434, 873, 56, 16);
panel.add(lblOutput18PriorToFirst);

JLabel lblImageQualityIndex = new JLabel("Image Quality Index [-1 to 1]:");
lblImageQualityIndex.setBounds(519, 762, 175, 16);
panel.add(lblImageQualityIndex);

JLabel lblOutput19ImageQuality = new JLabel("");
```

```
lblOutput19ImageQuality.setBounds(696, 762, 42, 16);
panel.add(lblOutput19ImageQuality);

JLabel lblPSNRComment = new JLabel("");
lblPSNRComment.setForeground(Color.GREEN);
lblPSNRComment.setBounds(12, 922, 726, 16);
panel.add(lblPSNRComment);

JLabel lblRunTimeEmbed = new JLabel("Time taken to Embed (ms):");
lblRunTimeEmbed.setBounds(12, 902, 162, 16);
panel.add(lblRunTimeEmbed);

JLabel lblOutput20RunTimeEmbed = new JLabel("");
lblOutput20RunTimeEmbed.setBounds(174, 902, 75, 16);
panel.add(lblOutput20RunTimeEmbed);

JLabel lblRunTimeExtract = new JLabel("Time taken to Extract (ms):");
lblRunTimeExtract.setBounds(252, 902, 160, 16);
panel.add(lblRunTimeExtract);

JLabel lblOutput21RunTimeExtract = new JLabel("");
lblOutput21RunTimeExtract.setBounds(417, 902, 75, 16);
panel.add(lblOutput21RunTimeExtract);

JLabel lblTotalTime = new JLabel("Total Time (ms):");
lblTotalTime.setBounds(504, 902, 103, 16);
panel.add(lblTotalTime);

JLabel lblOutput22TotalTime = new JLabel("");
lblOutput22TotalTime.setBounds(619, 902, 75, 16);
```

```
panel.add(lblOutput22TotalTime);

JLabel lblRmse = new JLabel("RMSE:");
lblRmse.setBounds(12, 791, 46, 16);
panel.add(lblRmse);

JLabel lblOutputRMSE = new JLabel("");
lblOutputRMSE.setBounds(64, 791, 56, 16);
panel.add(lblOutputRMSE);

JLabel lblSizeOfSecret = new JLabel("Size of Secret Image:");
lblSizeOfSecret.setBounds(472, 820, 129, 16);
panel.add(lblSizeOfSecret);

JLabel lblOutputSecretSize = new JLabel("");
lblOutputSecretSize.setBounds(613, 820, 56, 16);
panel.add(lblOutputSecretSize);

JSeparator separator = new JSeparator();
separator.setBounds(12, 484, 726, 2);
panel.add(separator);

/**
 * Real Application here.
 */

JButton btnEmbed = new JButton("Embed my Secret");
btnEmbed.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                long startTotal = System.currentTimeMillis();
```

```
if(!rdbtnUploadCover.isSelected())
    txtOutput1CoverPixels.setText("Needed only to display Image upload inputs. Only first 10,000 pixels will be displayed.");
if(!rdbtnUploadSecret.isSelected())
    txtOutput3SecretPixels.setText("Needed only to display Image upload inputs. Only first 10,000 pixels will be displayed.");
if(!rdbtnWu.isSelected())
    txtOutput2UpdatedCover.setText("Needed only for Wu and Tsai Method to properly calculate PSNR, Image Quality, etc.");
if(!rdbtnHybrid.isSelected())
{
    txtOutput4CoverBits.setText("Needed only for hybrid method.");
    txtOutput7UpdatedEmbedded.setText("Needed only for Wu and Tsai Method to properly calculate PSNR, Image Quality, etc.");
}

CI.clear();
SI.clear();
EI.clear();
EIUpdated.clear();
CIUpdated.clear();
CIStream.clear();
RA.clear();
str = new StringBuilder();
str1 = new StringBuilder();
Bs="";
Cs="";
extractedBs="";
embeddedPixels="";
coverPixels="";
secretPixels="";
```

```
coverAsBits="";
secretText="";
updatedCover="";
updatedEmbedded="";
extraction="";
coverBits=0;
pixelCounter=0;
beginBs=0;
endBs=0;
MSEnumerator=0;
xBar=0;
yBar=0;
sigmaX=0;
sigmaY=0;
sigmaXY=0;

lblNote.setText("");

//Cover Radios
if(rdbtnCover.isSelected())
{
	String coverNumbers = txtFieldCover.getText();
	if(rdbtnHybrid.isSelected())
	{
		for (String s : coverNumbers.split("\\s"))
		{
			CI.add(Integer.parseInt(s));
			intToBin = intToBinary(CI.get(CI.size()-1),
8);
			CIStream.add(intToBin);
```

```
                    Cs+=(CIStream.get(CI.size()-1) +" ");
                    coverBits+=8;
                }
                txtOutput4CoverBits.setText(Cs.trim());
            }
            else
            {
                for (String s : coverNumbers.split("\\s"))
                {
                    CI.add(Integer.parseInt(s));
                    coverBits+=8;
                }
            }
            rows = Integer.parseInt(txtFieldRows.getText());
            columns = CI.size()/rows;
        }
        else if(rdbtnUploadCover.isSelected())
        {

            BufferedImage image;
            try
            {
               File input = new File(lblFileCover.getText());
               image = ImageIO.read(input);
               rows = image.getHeight();
               columns = image.getWidth();


               int k=0;
               if(rdbtnHybrid.isSelected())
```

```
{
        for(int i=0; i<rows; i++)
          {
                for(int j=0; j<columns; j++)
                {

                rgb = image.getRGB(j,i);
                CI.add(rgb & 0xFF);
                coverBits+=8;
                if(CI.size()<10000)
                {
                 coverPixels += (CI.get(k) +" ");

                 k++;
                }
                intToBin = intToBinary(CI.get(CI.size()-1),
8);
                                CIStream.add(intToBin);

Cs+=(CIStream.get(CIStream.size()-1) +" ");
                }
           }
        txtOutput4CoverBits.setText(Cs.trim());
}
else
{
        for(int i=0; i<rows; i++)
          {
                for(int j=0; j<columns; j++)
                {

                rgb = image.getRGB(j,i);
```

```
                                CI.add(rgb & 0xFF);
                                coverBits+=8;
                                if(CI.size()<10000)
                                {
                                 coverPixels += (CI.get(k) +" ");
                                 k++;
                                }
                               }
                          }
                    }
                 } catch (Exception e) {}
                txtOutput1CoverPixels.setText(coverPixels);

}
else
{}

//Secret Radios
if(rdbtnSecretText.isSelected())
{
        secretText = txtFieldSecretText.getText();

        if(!rdbtnWang.isSelected())
        {
                for(int i=0;i<secretText.length();i++)
                        SI.add((int) secretText.charAt(i));
        }
}
else if(rdbtnSecretPixels.isSelected())
{
        String secretNumbers = txtFieldSecretPixels.getText();
```

```
                int j=0;
                for (String s : secretNumbers.split("\\s"))
                {
                        SI.add(Integer.parseInt(s));
                        secretText += Character.toString((char)
SI.get(j).intValue());
                        j++;
                }
        }
        else if(rdbtnUploadSecret.isSelected())
        {
                BufferedImage image;
                try
                {
                   File input = new File(lblFileSecret.getText());
                   image = ImageIO.read(input);
                   height = image.getHeight();
                   width = image.getWidth();


                   int k=0;
                   if(rdbtnWang.isSelected()) //if method 1, convert
SI arraylist to secretText String
                   {
                        for(int i=0; i<height; i++)

                                {
                                        for(int j=0; j<width; j++)
                                                {
                                                        rgb =
image.getRGB(j,i);
```

```
                                                    SI.add(rgb &
0xFF);

        if(SI.size()<10000)

        secretPixels += (SI.get(k) +" ");
                                                    secretText +=
Character.toString((char) SI.get(k).intValue());
                                                    k++;
                                                }
                                            }
                                        }
                                        else
                                        {
                                            for(int i=0; i<height; i++)

                                            {
                                                for(int j=0; j<width; j++)
                                                    {
                                                        rgb =
image.getRGB(j,i);
                                                        SI.add(rgb & 0xFF);
                                                        if(SI.size()<10000)
                                                            secretPixels +=
(SI.get(k) +" ");
                                                        k++;
                                                    }
                                            }
                                        }
                                            }catch (Exception e) {}
                                    txtOutput3SecretPixels.setText(secretPixels);

                                }
```

```
else
{}

//Setup TextFields
if(rdbtnWu.isSelected() || rdbtnHybrid.isSelected())
{
        if(txtFieldR1_1.getText().equals(""))
                txtFieldR1_1.setText("-1");
        if(txtFieldR1_2.getText().equals(""))
                txtFieldR1_2.setText("-1");
        if(txtFieldR2_1.getText().equals(""))
                txtFieldR2_1.setText("-1");
        if(txtFieldR2_2.getText().equals(""))
                txtFieldR2_2.setText("-1");
        if(txtFieldR3_1.getText().equals(""))
                txtFieldR3_1.setText("-1");
        if(txtFieldR3_2.getText().equals(""))
                txtFieldR3_2.setText("-1");
        if(txtFieldR4_1.getText().equals(""))
                txtFieldR4_1.setText("-1");
        if(txtFieldR4_2.getText().equals(""))
                txtFieldR4_2.setText("-1");
        if(txtFieldR5_1.getText().equals(""))
                txtFieldR5_1.setText("-1");
        if(txtFieldR5_2.getText().equals(""))
                txtFieldR5_2.setText("-1");
        if(txtFieldR6_1.getText().equals(""))
                txtFieldR6_1.setText("-1");
        if(txtFieldR6_2.getText().equals(""))
                txtFieldR6_2.setText("-1");
```

```
if(txtFieldR7_1.getText().equals(""))
        txtFieldR7_1.setText("-1");
if(txtFieldR7_2.getText().equals(""))
        txtFieldR7_2.setText("-1");
if(txtFieldR8_1.getText().equals(""))
        txtFieldR8_1.setText("-1");
if(txtFieldR8_2.getText().equals(""))
        txtFieldR8_2.setText("-1");
if(txtFieldR9_1.getText().equals(""))
        txtFieldR9_1.setText("-1");
if(txtFieldR9_2.getText().equals(""))
        txtFieldR9_2.setText("-1");
if(txtFieldR10_1.getText().equals(""))
        txtFieldR10_1.setText("-1");
if(txtFieldR10_2.getText().equals(""))
        txtFieldR10_2.setText("-1");

if(txtFieldR11_1.getText().equals(""))
        txtFieldR11_1.setText("-1");
if(txtFieldR11_2.getText().equals(""))
        txtFieldR11_2.setText("-1");
if(txtFieldR12_1.getText().equals(""))
        txtFieldR12_1.setText("-1");
if(txtFieldR12_2.getText().equals(""))
        txtFieldR12_2.setText("-1");
if(txtFieldR13_1.getText().equals(""))
        txtFieldR13_1.setText("-1");
if(txtFieldR13_2.getText().equals(""))
        txtFieldR13_2.setText("-1");
if(txtFieldR14_1.getText().equals(""))
```

```
            txtFieldR14_1.setText("-1");
        if(txtFieldR14_2.getText().equals(""))
            txtFieldR14_2.setText("-1");

        R1_1 = Integer.parseInt(txtFieldR1_1.getText());
        R1_2 = Integer.parseInt(txtFieldR1_2.getText());
        R2_1 = Integer.parseInt(txtFieldR2_1.getText());
        R2_2 = Integer.parseInt(txtFieldR2_2.getText());
        R3_1 = Integer.parseInt(txtFieldR3_1.getText());
        R3_2 = Integer.parseInt(txtFieldR3_2.getText());
        R4_1 = Integer.parseInt(txtFieldR4_1.getText());
        R4_2 = Integer.parseInt(txtFieldR4_2.getText());
        R5_1 = Integer.parseInt(txtFieldR5_1.getText());
        R5_2 = Integer.parseInt(txtFieldR5_2.getText());
        R6_1 = Integer.parseInt(txtFieldR6_1.getText());
        R6_2 = Integer.parseInt(txtFieldR6_2.getText());
        R7_1 = Integer.parseInt(txtFieldR1_1.getText());
        R7_2 = Integer.parseInt(txtFieldR1_2.getText());
        R8_1 = Integer.parseInt(txtFieldR2_1.getText());
        R8_2 = Integer.parseInt(txtFieldR2_2.getText());
        R9_1 = Integer.parseInt(txtFieldR3_1.getText());
        R9_2 = Integer.parseInt(txtFieldR3_2.getText());
        R10_1 = Integer.parseInt(txtFieldR4_1.getText());
        R10_2 = Integer.parseInt(txtFieldR4_2.getText());
        R11_1 = Integer.parseInt(txtFieldR5_1.getText());
        R11_2 = Integer.parseInt(txtFieldR5_2.getText());
        R12_1 = Integer.parseInt(txtFieldR6_1.getText());
        R12_2 = Integer.parseInt(txtFieldR6_2.getText());
        R13_1 = Integer.parseInt(txtFieldR6_1.getText());
        R13_2 = Integer.parseInt(txtFieldR6_2.getText());
```

```
            R14_1 = Integer.parseInt(txtFieldR6_1.getText());
            R14_2 = Integer.parseInt(txtFieldR6_2.getText());

}


//Embedding Method Radios
//Wang Method (1st)
if(rdbtnWang.isSelected())
{
            long startEmbedTime = System.currentTimeMillis();
            ml = Integer.parseInt(txtFieldMl.getText());
            mu = Integer.parseInt(txtFieldMu.getText());

            T = Integer.parseInt(txtFieldThreshold.getText());

            Bs = stringToBinary(secretText);
            beginBs=0;
            endBs=0;
            for(int i=0;i<CI.size();i++)
            {
                        if(CI.get(i)>T)
                        {
                                    EC = (int)Math.floor(log2(mu));
                                    RES = CI.get(i) % mu;
                        }
                        else
                        {
                                    EC = (int)Math.floor(log2(ml));
                                    RES = CI.get(i) % ml;
                        }
```

```
endBs += EC;
if(endBs>Bs.length())
{
        endBs=Bs.length();
        subBs = Bs.substring(beginBs, endBs);
        if(endBs-beginBs != 0)
                lblNote.setText("Last " +(endBs-beginBs)+" secret bits "+subBs+" could not be embedded because their length "+subBs.length()+" is less than EC = "+EC+" bits we are trying to embed.");
        break;
}
subBs = Bs.substring(beginBs, endBs);
DEC = binaryToInteger(subBs);
beginBs = endBs;
D = Math.abs(RES - DEC);

if(CI.get(i)<T) //Case 1
{
        if(CI.get(i)<(ml/2))
                EI.add(DEC);
        else if((ml/2)<=CI.get(i) && CI.get(i)<(T-(ml/2)))
        {
                if(D>(ml/2))
                {
                        AV = ml-D;
                        if(RES > DEC)
                                EI.add(CI.get(i)+AV);
                        else
                                EI.add(CI.get(i)-AV);
                }
```

```
                else
                {
                        AV = D;
                        if(RES>DEC)
                                EI.add(CI.get(i)-AV);
                        else
                                EI.add(CI.get(i)+AV);
                }
        }
        else if((T-ml/2)<=CI.get(i) && CI.get(i)<T)
        {
                EI.add((CI.get(i)-RES)+DEC);
        }
}

else //Case 2
{
        if(CI.get(i)>(255-(mu/2)+1))
                EI.add((255-mu+1)+DEC);
        else if(T+(mu/2)<CI.get(i) &&
CI.get(i)<=(255-(mu/2)+1))
        {
                if(D>(mu/2))
                {
                        AV = mu-D;
                        if(RES>DEC)
                                EI.add(CI.get(i)+AV);
                        else
                                EI.add(CI.get(i)-AV);
                }
```

```
                                        else
                                        {
                                                AV = D;
                                                if(RES>DEC)
                                                        EI.add(CI.get(i)-AV);
                                                else
                                                        EI.add(CI.get(i)+AV);
                                        }
                                }
                                else if(T<=CI.get(i) &&
CI.get(i)<=(T+(mu/2)))
                                {
                                        EI.add(CI.get(i)-RES+DEC);
                                }
                        }
                        embeddedPixels += (EI.get(i) +" ");
                }
                txtOutput5SecretBits.setText(Bs);

                txtOutput6EmbedPixels.setText(embeddedPixels.trim());

                long endEmbedTime = System.currentTimeMillis();
                long totalEmbedTime = endEmbedTime-startEmbedTime;
                System.out.println(totalEmbedTime);

        lblOutput20RunTimeEmbed.setText(String.valueOf(totalEmbedTime));
                //extraction method1

                for(int i=0;i<EI.size();i++)
                {
                        if(EI.get(i)<T) //Case 1
```

```
                    {
                        RES = EI.get(i)%ml;
                        EC = (int)Math.floor(log2(ml));
                    }

                    else //Case 2
                    {
                        RES = EI.get(i)%mu;
                        EC = (int)Math.floor(log2(mu));
                    }
                    extraction += intToBinary(RES,EC);
                }

lblOutput13EC.setText(String.valueOf(extractedBs.length()/coverBits));
                long EndExtractTime = System.currentTimeMillis();
                long totalExtractTime = EndExtractTime - endEmbedTime;

lblOutput21RunTimeExtract.setText(String.valueOf(totalExtractTime));
            }

            //Wu & Tsai Method (2nd)
            else if(rdbtnWu.isSelected())
            {
                long startEmbedTime = System.currentTimeMillis();
                for(int i=0;i<SI.size();i++)
                    Bs += Integer.toBinaryString(SI.get(i));
                txtOutput5SecretBits.setText(Bs);

                beginBsCheck=0;
                beginBs=0;
                endBsCheck=0;
```

```
endBs=0;
for(int i=0;i<CI.size();i+=2)
{
        //checking
        pixelOneCheck=CI.get(i);
        pixelTwoCheck=CI.get(i+1);
        dCheck=pixelTwoCheck-pixelOneCheck;
        numOfBitsCheck=rangeFinder(dCheck, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1, R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1, R14_2);
        endBsCheck += numOfBitsCheck;
        if(endBsCheck>Bs.length())
        {
                endBsCheck=Bs.length();
        }
        beginBsCheck=endBs;
        subBsCheck = Bs.substring(beginBsCheck, endBsCheck);

        bCheck = Integer.parseInt(subBsCheck, 2);
        dPrimeCheck = maxRangeFinder(dCheck, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1, R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1, R14_2);
        mCheck = dPrimeCheck - dCheck;
        pixelOneEmbedCheck =  (int) ((dCheck % 2 != 0) ? (pixelOneCheck-Math.ceil(mCheck/2)) : (pixelOneCheck-Math.floor(mCheck/2)));
        pixelTwoEmbedCheck =  (int) ((dCheck % 2 != 0) ? (pixelTwoCheck+Math.floor(mCheck/2)) : (pixelTwoCheck+Math.ceil(mCheck/2)));

        if((pixelOneEmbedCheck < 0 || pixelOneEmbedCheck > 255) || (pixelTwoEmbedCheck < 0 || pixelTwoEmbedCheck > 255))
        {
                endBsCheck = beginBsCheck;
```

```
                    continue; //checking failed, continue to next
pixels
                }

                //checking passed
                else
                {
                    pixelOne=CI.get(i);
                    pixelTwo=CI.get(i+1);
                    CIUpdated.add(pixelOne);
                    updatedCover +=
(CIUpdated.get(CIUpdated.size()-1) +" ");
                    CIUpdated.add(pixelTwo);
                    updatedCover +=
(CIUpdated.get(CIUpdated.size()-1) +" ");
                    d=pixelTwo-pixelOne;
                    numOfBits=rangeFinder(d, R1_1, R1_2,
R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1, R7_2, R8_1, R8_2, R9_1,
R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1, R14_2);
                    endBs += numOfBits;
                    if(endBs>Bs.length())
                    {
                        endBs=Bs.length();
                        subBs = Bs.substring(beginBs,
endBs);
                        if(endBs-beginBs != 0)
                            lblNote.setText("Last "
+(endBs-beginBs)+" secret bits "+subBs+" could not be embedded because their length
"+subBs.length()+" is less than numOfBits = "+numOfBits+" we are tring to embed.");
                        break;
                    }
                    subBs = Bs.substring(beginBs, endBs);
                    b = Integer.parseInt(subBs, 2);
```

```
                                        dPrime = (d>=0) ? (minRangeFinder(d, R1_1,
R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1, R7_2, R8_1, R8_2,
R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1, R14_2)+b) : -
1*(minRangeFinder(d, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2,
R7_1, R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2,
R14_1, R14_2)+b);
                                        m = dPrime - d;
                                        pixelOneEmbed =  (int) ((d % 2 != 0) ?
(pixelOne-Math.ceil(m/2)) : (pixelOne-Math.floor(m/2)));
                                        pixelTwoEmbed =  (int) ((d % 2 != 0) ?
(pixelTwo+Math.floor(m/2)) : (pixelTwo+Math.ceil(m/2)));
                                        EI.add(pixelOneEmbed);
                                        embeddedPixels += (EI.get(EI.size()-1) +" ");
                                        EI.add(pixelTwoEmbed);
                                        embeddedPixels += (EI.get(EI.size()-1) +" ");
                                        beginBs=endBs;
                                }
                        }
                        txtOutput6EmbedPixels.setText(embeddedPixels);

                        long endEmbedTime = System.currentTimeMillis();
                        long totalEmbedTime = endEmbedTime-startEmbedTime;

        lblOutput20RunTimeEmbed.setText(String.valueOf(totalEmbedTime));
                        //Extraction
                        for(int i=0;i<EI.size();i+=2)
                        {
                                dExtracted = EI.get(i+1)-EI.get(i);
                                bExtracted = (dExtracted>=0) ? (dExtracted-
minRangeFinder(dExtracted, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2,
R6_1, R6_2, R7_1, R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2,
R13_1, R13_2, R14_1, R14_2)) : (-dExtracted-minRangeFinder(dExtracted, R1_1, R1_2, R2_1,
R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2, R7_1, R7_2, R8_1, R8_2, R9_1, R9_2,
R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1, R13_2, R14_1, R14_2));
```

```
                                        intToBin = intToBinary(bExtracted,
rangeFinder(dExtracted, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1,
R6_2, R7_1, R7_2, R8_1, R8_2, R9_1, R9_2, R10_1, R10_2, R11_1, R11_2, R12_1, R12_2, R13_1,
R13_2, R14_1, R14_2));
                                        extraction+=intToBin;
                                    }
                                    txtOutput8ExtractedBits.setText(extraction);

        lblOutput13EC.setText(String.valueOf(extraction.length()/coverBits));
                                    long EndExtractTime = System.currentTimeMillis();
                                    long totalExtractTime = EndExtractTime - endEmbedTime;

        lblOutput21RunTimeExtract.setText(String.valueOf(totalExtractTime));
                                }

                                else if(rdbtnHybrid.isSelected())
                                {
                                    long startEmbedTime = System.currentTimeMillis();
                                    if(txtFieldR1Bits.getText().equals(""))
                                        txtFieldR1Bits.setText("-1");
                                    if(txtFieldR2Bits.getText().equals(""))
                                        txtFieldR2Bits.setText("-1");
                                    if(txtFieldR3Bits.getText().equals(""))
                                        txtFieldR3Bits.setText("-1");
                                    if(txtFieldR4Bits.getText().equals(""))
                                        txtFieldR4Bits.setText("-1");
                                    if(txtFieldR5Bits.getText().equals(""))
                                        txtFieldR5Bits.setText("-1");
                                    if(txtFieldR6Bits.getText().equals(""))
                                        txtFieldR6Bits.setText("-1");

                                    bit1 = Integer.parseInt(txtFieldR1Bits.getText());
```

```
bit2 = Integer.parseInt(txtFieldR2Bits.getText());
bit3 = Integer.parseInt(txtFieldR3Bits.getText());
bit4 = Integer.parseInt(txtFieldR4Bits.getText());
bit5 = Integer.parseInt(txtFieldR5Bits.getText());
bit6 = Integer.parseInt(txtFieldR6Bits.getText());


for(int i=0;i<SI.size();i++)
{
        intToBin = intToBinary(SI.get(i), 8);
        Bs+=intToBin;
}
txtOutput5SecretBits.setText(Bs);

//embedding
for(int i=0;i<CI.size();i+=2)
{
        p=CI.get(i);
        q=CI.get(i+1);
        tempP1=CIStream.get(i);
        tempP2=CIStream.get(i+1);
        if(p<192 && q<192) //Case 1
        {
                n=6;
                k=3;
                str.append(tempP1);
                str1.append(tempP2);
                startStr=8-k;
                endBs+=k;
```

```
                    if(endBs>Bs.length() || endBs+k>Bs.length())
                    {
                        endBs=Bs.length();
                        subBs = Bs.substring(beginBs,
endBs);

                        if(endBs-beginBs != 0)
                            lblNote.setText("Last "
+(endBs-beginBs)+" secret bits "+subBs+" could not be embedded because their length
"+subBs.length()+" is less than n = "+n+" bits we are tring to embed.");
                        break;
                    }

                    str.replace(startStr, 8, Bs.substring(beginBs,
endBs));
                    str1.replace(startStr, 8, Bs.substring(endBs,
endBs+k));
                    beginBs=endBs+k;
                    endBs+=k;
                    p1=Integer.parseInt(str.toString(), 2);
                    q1=Integer.parseInt(str1.toString(), 2);
                    str = new StringBuilder();
                    str1 = new StringBuilder();
                    EI.add(p1);
                    EI.add(q1);
                    pixelCounter+=2;
                }
                else //Case 2
                {
                    d=Math.abs(q-p);
                    n=rangeFinderHybrid(d, bit1, bit2, bit3, bit4,
bit5, bit6, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2);
```

```
k=n/2;
str.append(tempP1);
str1.append(tempP2);
startStr=8-k;
endBs+=k;

if(endBs>Bs.length() || endBs+k>Bs.length())
{
        endBs=Bs.length();
        subBs = Bs.substring(beginBs,
endBs);
        if(endBs-beginBs != 0)
                lblNote.setText("Last "
+(endBs-beginBs)+" secret bits "+subBs+" could not be embedded because their length
"+subBs.length()+" is less than n = "+n+" bits we are tring to embed.");

        for(int j=0;j<EI.size();j++)
        {
                EIUpdated.add(EI.get(j));
        }


        for(int j=EI.size();j<CI.size();j++)
        {
                EIUpdated.add(CI.get(j));
                updatedEmbedded +=
(EIUpdated.get(EIUpdated.size()-1) +" ");
        }
        numberBeforeInconsist = EI.size();
        break;
}
```

```
                                                str.replace(startStr, 8, Bs.substring(beginBs,
endBs));
                                                str1.replace(startStr, 8, Bs.substring(endBs,
endBs+k));
                                                beginBs=endBs+k;
                                                endBs+=k;
                                                p1=Integer.parseInt(str.toString(), 2);
                                                q1=Integer.parseInt(str1.toString(), 2);
                                                dPrime=Math.abs(q1-p1);
                                                str = new StringBuilder();
                                                str1 = new StringBuilder();
                                                if(rangeFinderHybrid(dPrime, bit1, bit2,
bit3, bit4, bit5, bit6, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1,
R6_2)==n) //no re-adjust
                                                {
                                                        EI.add(p1);
                                                        EI.add(q1);
                                                        pixelCounter+=2;
                                                }
                                                else //re-adjust
                                                {
                                                        p2=(int) (p1+Math.pow(2, k));
                                                        p3=(int) (p1-Math.pow(2, k));
                                                        q2=(int) (q1+Math.pow(2, k));
                                                        q3=(int) (q1-Math.pow(2, k));

                                                        if((p1<=255 && q2<=255) &&
(p1>=192 || q2>=192) && rangeFinderHybrid(Math.abs(p1-q2), bit1, bit2, bit3, bit4, bit5, bit6,
R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2)==n) //p1,q2
                                                                label1=Math.abs(Math.abs(p-
p1)+Math.abs(q-q2));
                                                        else
```

```
label1=256;
RA.add(label1);

if((p1<=255 && q3<=255) &&
(p1>=192 || q3>=192)) //p1,q3
label2=Math.abs(Math.abs(p-
p1)+Math.abs(q-q3));
else
label2=256;
RA.add(label2);

if((p2<=255 && q1<=255) &&
(p2>=192 || q1>=192)) //p2,q1
label3=Math.abs(Math.abs(p-
p2)+Math.abs(q-q1));
else
label3=256;
RA.add(label3);

if((p2<=255 && q2<=255) &&
(p2>=192 || q2>=192)) //p2,q2
label4=Math.abs(Math.abs(p-
p2)+Math.abs(q-q2));
else
label4=256;
RA.add(label4);

if((p2<=255 && q3<=255) &&
(p2>=192 || q3>=192)) //p2,q3
label5=Math.abs(Math.abs(p-
p2)+Math.abs(q-q3));
else
label5=256;
```

```
                                                    RA.add(label5);


                                                    if((p3<=255 && q1<=255) &&
(p3>=192 || q1>=192)) //p3,q1
                                                        label6=Math.abs(Math.abs(p-
p3)+Math.abs(q-q1));
                                                    else
                                                        label6=256;
                                                    RA.add(label6);


                                                    if((p3<=255 && q2<=255) &&
(p3>=192 || q2>=192)) //p3,q2
                                                        label7=Math.abs(Math.abs(p-
p3)+Math.abs(q-q2));
                                                    else
                                                        label7=256;
                                                    RA.add(label7);


                                                    if((p3<=255 && q3<=255) &&
(p3>=192 || q3>=192) && rangeFinderHybrid(Math.abs(p3-q3), bit1, bit2, bit3, bit4, bit5, bit6,
R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2)==n) //p3,q3
                                                        label8=Math.abs(Math.abs(p-
p3)+Math.abs(q-q3));
                                                    else
                                                        label8=256;
                                                    RA.add(label8);


                                                    if(label1==256 && label2==256 &&
label3==256 && label4==256 && label5==256 && label6==256 && label7==256)
                                                    {
                                                        numberBeforeInconsist = i+1;
                                                        lblNote.setText("An
inconsistency occured while re-adjusting cover pixels "+CI.get(i)+" and "+CI.get(i+1)+".
Embedding operation stopped at pixel "+(i+1)+".");
```

```
                break;
            }
            minLabel =
RA.indexOf(Collections.min(RA));


            if(minLabel==0)
            {
                EI.add(p1);
                EI.add(q2);
            }
            else if(minLabel==1)
            {
                EI.add(p1);
                EI.add(q3);
            }
            else if(minLabel==2)
            {
                EI.add(p2);
                EI.add(q1);
            }
            else if(minLabel==3)
            {
                EI.add(p2);
                EI.add(q2);
            }
            else if(minLabel==4)
            {
                EI.add(p2);
                EI.add(q3);
            }
```

```
                                        else if(minLabel==5)
                                        {
                                                EI.add(p3);
                                                EI.add(q1);
                                        }
                                        else if(minLabel==6)
                                        {
                                                EI.add(p3);
                                                EI.add(q2);
                                        }
                                        else if(minLabel==7)
                                        {
                                                EI.add(p3);
                                                EI.add(q3);
                                        }
                                        pixelCounter+=2;
                                }
                        }
                        embeddedPixels += (EI.get(i) +" "+EI.get(i+1) +" ");
                }
                txtOutput6EmbedPixels.setText(embeddedPixels.trim());

                long endEmbedTime = System.currentTimeMillis();
                long totalEmbedTime = endEmbedTime-startEmbedTime;

lblOutput20RunTimeEmbed.setText(String.valueOf(totalEmbedTime));

                //Extraction
                for(int i=0;i<EI.size();i+=2)
                {
```

```
p=EI.get(i);
q=EI.get(i+1);

if(p<192 && q<192) //Case 1
{
        n=6;
        k=3;
        tempP1=intToBinary(p,8);
        tempP2=intToBinary(q,8);
        tempString=tempP1.substring(8-k, 8);
        tempString1=tempP2.substring(8-k, 8);
        p1=Integer.parseInt(tempString,2);
        p2=Integer.parseInt(tempString1,2);
        extraction+=intToBinary(p1, k);
        extraction+=intToBinary(p2, k);
}

else //Case 2
{
        d=Math.abs(q-p);
        n=rangeFinderHybrid(d, bit1, bit2, bit3, bit4,
bit5, bit6, R1_1, R1_2, R2_1, R2_2, R3_1, R3_2, R4_1, R4_2, R5_1, R5_2, R6_1, R6_2);
        k=n/2;
        tempP1=intToBinary(p,8);
        tempP2=intToBinary(q,8);
        tempString=tempP1.substring(8-k, 8);
        tempString1=tempP2.substring(8-k, 8);
        p1=Integer.parseInt(tempString,2);
        p2=Integer.parseInt(tempString1,2);
        extraction+=intToBinary(p1, k);
```

```
                                        extraction+=intToBinary(p2, k);
                                    }
                                }

        lblOutput18PriorToFirst.setText(String.valueOf(numberBeforeInconsist));
                                long EndExtractTime = System.currentTimeMillis();
                                long totalExtractTime = EndExtractTime - endEmbedTime;

        lblOutput21RunTimeExtract.setText(String.valueOf(totalExtractTime));
                            }


                            EIUpdated.addAll(EI);
                            updatedEmbedded = embeddedPixels;
//                          EIUpdated.addAll(new ArrayList<Integer>
(CI.subList(EIUpdated.size(), CI.size())));
                            if(rdbtnWu.isSelected())
                            {
                                for(int i=EI.size();i<CI.size();i++)
                                {
                                    EIUpdated.add(CI.get(i));
                                    updatedEmbedded +=
(EIUpdated.get(EIUpdated.size()-1) +" ");
                                    CIUpdated.add(CI.get(i));
                                    updatedCover += (CIUpdated.get(CIUpdated.size()-
1) +" ");
                                }
                                txtOutput2UpdatedCover.setText(updatedCover);
                            }
                            else
                            {
                                for(int i=EI.size();i<CI.size();i++)
                                {
```

```
                                        EIUpdated.add(CI.get(i));
                                        updatedEmbedded +=
(EIUpdated.get(EIUpdated.size()-1) +" ");
                                }
                        }


                        //PSNR Calculation
                        for(int i=0;i<EIUpdated.size();i++)
                        {
                                MSEnumerator+=(Math.pow(Math.abs(CI.get(i)-
EIUpdated.get(i)), 2));
                        }
                        MSE = MSEnumerator/(rows*columns);
                        RMSE = Math.sqrt(MSE);
                        PSNR = 10*Math.log10(255*255/MSE);
                        if(PSNR>=30)
                                lblPSNRComment.setText("Image is in good condition since
PSNR is greater than or equal to 30.");
                        else
                        {
                                lblPSNRComment.setForeground(Color.RED);
                                lblPSNRComment.setText("Image is in bad condition since
PSNR is less than 30.");
                        }


                        //image quality calculation
                        N = EIUpdated.size();
                        for(int i=0;i<N;i++)
                        {
                                xBar+=CI.get(i);
                        }
```

```
xBar=xBar/N;
for(int i=0;i<N;i++)
{
	yBar+=EIUpdated.get(i);
}
yBar=yBar/N;
for(int i=0;i<N;i++)
{
	sigmaX+=Math.pow(CI.get(i)-xBar, 2);
}
sigmaX=sigmaX/(N-1);
for(int i=0;i<N;i++)
{
	sigmaY+=Math.pow(EIUpdated.get(i)-yBar, 2);
}
sigmaY=sigmaY/(N-1);
for(int i=0;i<N;i++)
{
	sigmaXY+=(CI.get(i)-xBar)*(EIUpdated.get(i)-yBar);
}
sigmaXY=sigmaXY/(N-1);
Q=4*sigmaXY*xBar*yBar/((sigmaX+sigmaY)*(Math.pow(xBar,
2)+Math.pow(yBar, 2)));

txtOutput6EmbedPixels.setText(embeddedPixels.trim());
txtOutput7UpdatedEmbedded.setText(updatedEmbedded);
txtOutput8ExtractedBits.setText(extraction);
lblOutput11Rows.setText(String.valueOf(rows));
lblOutput12Columns.setText(String.valueOf(columns));
```

```
        lblOutput13EC.setText(String.format("%.2f",(double)extraction.length()/coverBits) +" bits per pixel.");
                                lblOutput14CoverSize.setText(String.valueOf(CI.size()));
                                lblOutput15EmbedSize.setText(String.valueOf(EI.size()));

        lblOutput16UpdateEmbed.setText(String.valueOf(EIUpdated.size()));
                                lblOutput17RemainSpace.setText(String.valueOf(EIUpdated.size()-
EI.size()));
                                lblOutput10PSNR.setText(String.format("%.2f", PSNR));
                                lblOutput9MSE.setText(String.format("%.2f", MSE));
                                lblOutputRMSE.setText(String.format("%.2f", RMSE));
                                lblOutput19ImageQuality.setText(String.valueOf(Q));
                                lblOutputSecretSize.setText(String.valueOf(SI.size()));

                                long endTotal = System.currentTimeMillis();
                                long totalTime = endTotal-startTotal;
                                lblOutput22TotalTime.setText(String.valueOf(totalTime));


                                txtOutput1CoverPixels.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput2UpdatedCover.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput3SecretPixels.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput4CoverBits.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput5SecretBits.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput6EmbedPixels.addFocusListener(new
CursorAtStartFocusListener());
                                txtOutput7UpdatedEmbedded.addFocusListener(new
CursorAtStartFocusListener());
```

```
                txtOutput8ExtractedBits.addFocusListener(new
CursorAtStartFocusListener());
                    }
                });

                btnEmbed.setBounds(302, 455, 135, 25);
                panel.add(btnEmbed);
        }
}
```