EASTERN MEDITERRANEAN UNIVERSITY

Computer Engineering Department
Software Engineering

CMSE 492 Spring 2021
Selected Topics in Software Engineering II

Assoc. Prof. Dr. Alexander Chefranov
Assistant Felix Babalola
Assistant Nada Kollah

Lab Report 1
Implementation of Wang's Method

Group 4:
Taha Ünsal 17300199
Ahmad Turjman 16700452
Kiana Kamkar 17701884
Olcay Ergül 18001478
Souhail Zakar 17700710
Ata Oğuz Kuzu 17001836

Famagusta, TRNC
15.04.2021

Outline

Problem Definition

20.03.2021
Task:
1. Implement Wang's algorithm [1] in any programming language/operating system
2. Test your implementation using Seminar 23.03.2021 examples: "Embed the following secret
binary data, S=' 1 1 0 1 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1', into the cover image, CI=(190,
255, 161, 100, 1, 159) by Wang's method using T=160, ml=16, mu=32".
3. Test your implementation on 4 host 512x512 images (Mandrill, Peppers, Jet, and Lena) and 4
secret 256x256 images (Mandrill, Peppers, Jet, and Lena) used in [1]
4. For each of 16 variants of embedding secret into the covers, calculate PSNR and compare your
results versus [1, Table 1, p. 112]. In the case of discrepancies, fix your problems, or prove that
your results are correct.
5. Defend the Lab on April 8, Thursday, 18.30-20.20, Teams (upload your report, run your
program, and explain your work done).
6. Report shall have
6.1. Cover page (University, Department, Course, Semester, Year, City, Country, Lab subject,
Team members, Lecturer, Lab assistant)
6.2. Outline
6.3. Problem definition (see items 1-4 above)
6.4. Wang's method description
6.5. Description of Wang's method implementation in your programming language/operating
system

6.5.1. Description of the host/secret images you use and their sources
6.5.2. Description of preprocessing phase 1 implementation
6.5.3. Description of embedding phase 2 implementation (m_l, m_u, and T must be
parameters, not literals)
Description of extraction phase 3 implementation
Description of PSNR and embedding capacity calculation

6.6. Description of the tests conducted and their results, screenshots of them
6.7. Comparison of your results versus [1, Table 1, p. 112].
6.8. Conclusion
6.9. References
6.10. Appendices with the code developed
6.11. Archived file (zip, or rar) with all Lab related materials (report, images used, test results,

sources, executables). It shall be possible to install your program from the archive, run it on
your examples, and view results you got.

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, paper

Grading policy: report – 50%, explanations – 50%

Description of Wang's Method

Steganograph is the field of protecting messages by hiding them. In digital images, we can apply methods in order to embed secret data inside them without the alternation being noticeable by human vision. Least significant bit method alters the last bit of each pixel in the image in order to embed a secret data inside digital images. Since last bits play the least role on the appearance of the images, image is modified minimally while storing the secret data. In this report, we will apply the Least Significant Bit method in order to embed and extract data inside pixels.

Monochrome images consist of pixels. Each pixel has a luminosity value between 0 and 255. This value is stored by 8 bits in memory. When these bits are concatenated into a single long bit string and then embedded inside another image bit by bit, we can later extract our image back from the host image.

Phase 1: Preprocessing

Input: Secret image S
Output: Secret bit string Bs

Step 1: Order all pixels in S in a top to bottom and left to right way.
Step 2: Represent each pixel in an 8-bit long bit string, concatenate all the strings into a single long bit string Bs.
Step 3: Output the bit string Bs.

Phase 2: Embedding

Input: Host image C, bit string Bs
Output: Stego image SI

Step 1: Set the threshold value T and two modulus mu and ml where mu > ml. Then compute the residue RES and the possible embedded capacity EC as the following procedure:

If Pc(i) >= T
    EC = floor(log2(mu))
    RES = Pc(i) mod mu
Else
    EC = floor(log2(ml))
    RES = Pc(i) mod ml

Step 2: Compute difference D = | RES - DEC| where DEC is the decimal value of EC bit-length string that is fetched from Bs.
Step4: Embed DEC into pixel Pc(i) in order to obtain Ps(i) which is the intensity of the i-th pixel after embedding DEC by following the algorithm in Shiuh-Jeng Wang's paper at phase 2 step 4.

Phase 3: Extraction

Input: Stego Image SI, Threshold T, modulus mu and ml where mu>ml
Output: Extracted bit string of Bs', original secret image S'

Step 1: Compute RES and EC as following cases:

- Case I: Ps(i) < T
    - Compute RES = Ps(i) mod ml
    - Compute EC = floor(log2(ml))
- Case 2: Ps(i) >= T
    - Compute RES = Ps(i) mod mu
    - Compute EC = floor(log2(mu))

Step 2: Translate the RES into the bit representations with EC bit length.
Step 3: Repeat steps 1-2 until all the embedded bits of Bs' are recovered.

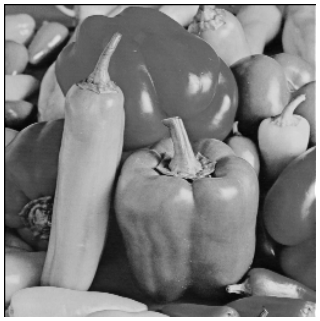
Description of the Implementation

Description of Host/Secret Images

Host and secret images are monochrome PNG images. PNG images are lossless formats having 4 channels red, green, blue, alpha. Our images have 255 alpha value. And equal amount of red, green, blue in order to stay as a shade of gray. Our host image is 512 by 512 and the secret image is 256 by 256 as stated in the problem.

It was rather difficult to obtain the test images as the internet was crowded with variations of them having various file extensions. We managed to find a couple images with suitable color scheme, extension and size for us. Following were the test images:


1. Zelda (512x512)


2. Peppers (256x256)


3. Lena (256x256)

512x512 images were used as host images and 256x256 ones were used as secret images as instructed in the task.

Description of Phase 1: Preprocessing

Image names to be manipulated were prompted to the user at the start of the program. In case of encoding; host image, secret image, and output file name is being asked. In case of extracting; stegano image and output file name is being asked. If user inputs a lossy format as output file name, the program automatically converts it to a lossless format PNG since it is very challenging to preserve embedded data along with compression.

Lastly, a calculation is made to ensure our secret image fits into the host image. Host image consists of 512x512 pixels. Each pixel is assumed to have 1 channel. Therefore, the image can hold a total of 512x512x1 bits. Secret image however, consists of 256x256 pixels, 1 channel and 8 bits per channel. Therefore, the number of bits to be embedded is calculated as 256x256x8 which does not fit into the host image. In order to overcome this, instead of embedding all bits of the secret image, we only embed a certain number of bits. And we choose the bits that matter the most. Therefore we go with most significant bits. We start by testing if all bits can be embedded in the host image. If not, we decrease the number of bits to be embedded and test again. We repeat this cycle until the secret image could be fit into the host image.

As an example;
256x256x8 > 512x512x1
256x256x7 > 512x512x1
256x256x6 > 512x512x1
256x256x5 > 512x512x1
256x256x4 = 512x512x1

Therefore, embedding only 4 most significant bits will fit into the host image.

Description of Phase 2: Embedding

If the user chooses the embedding method, the program calls the embedding function. Embedding function iterates over secret image pixels. For each pixel, it will get the value and convert it to a bit string. Selected number of most significant bits will be extracted from the current pixel. Then for each of these bits, a pixel from the host image will be fetched and the current bit will be embedded into that pixel in the host image. This process concludes the embedding part.

Embedding pseudocode:

For each pixel in secret image:
    Get current pixel value
    Convert it to bit string
    Get selected number of most significant bits from the bit string

For each bit in bit string:
    Fetch a pixel from host image
    Convert it to bit string
    Replace the least significant bit of host image pixel with the current bit

Description of Phase 3: Extraction

After the user chooses extracting from the menu, extraction function iterates over stega images pixels. It converts the current pixel value into a bit string. Then it extracts the least significant bit of the string and adds it to a buffer. Every 8 bits might feel like they form a secret image pixel here, however we may have not actually embedded all the bits of the secret image pixels in order to fit the secret image into the host image. As we have previously discussed, we are selecting some number of most significant bits from the secret image pixels. Therefore the rest of its bits are not embedded and lost. They would get substituted by zeros. Therefore, once this function iterates as many as the number of most significant bits teared from the pixels of the secret image, we obtain all the embedded data of a secret image bit. We will append (8 - number of most significant bits) number of zeros after it in order to complete it to a byte.

Example:

4 Most significant bits => 1010 0011, 0110 1001 will be 1010, 0110
And when extracting, instead of reading 8 bits by 8 bits, we will read as groups of most significant beats teared. Then append zeros to it. In this case: 1010 0000, 0110 0000

Each time we obtain a secret image pixel, we add it into the previous one. Gathering all pixels of the secret image forms itself at the end, successfully extracting our secret image back.

Description of PSNR and Embedding Capacity Calculation

Calculation of embedding capacity was discussed in the “Description of the Phase 2: Embedding” section. And PSNR was calculated by getting the mean square error of the original image and the stegano image. Checking whether it is 0.
If not, calculating 20 * log(255 / sqrt(mean square error)).

Description of the Tests Conducted

Smaller 256x256 images have been embedded into the 512x512 host image. And then they were extracted back using the same program. Results are shown in the following section. Program was tested using images with PNG extension due to being lossless.

Comparison of the Results

Host Image 1: Zelda (512x512)

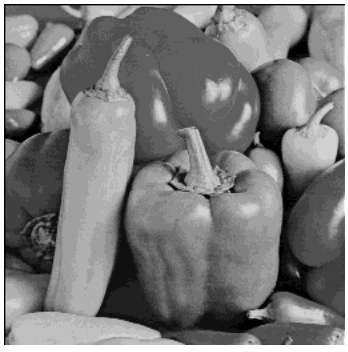
Secret Image 1: Lena (256x256)

Secret Image 2: Peppers (256x256)

Stegano Image 1: Lena embedded in Zelda (512x512)

Stegano Image 2: Peppers embedded in Zelda (512x512)

Extracted Image 1: Lena (256x256)



Extracted Image 2: Peppers (256x256)

Conclusion

We have embedded secret images into host images with minimum manipulation of the host image. Only the least significant bits of the host images got affected, therefore difference is not perceivable by the human eye. Resultant stega image had a very low level of noise which i s calculated by PSNR. Some data from the secret image was needed to be compromised in order to fit it into the host image. But at the end, the image was successfully recovered with minimum data loss meanwhile still being able to complete the steganography.

References

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

Appendices

Source Code

```
import cv2 as cv
import numpy as np
import math

def embed_get_files():
      print('Enter host image name with extension: ', end = '')
      host_image_name = input()
      host_image = cv.imread(host_image_name)
      #host_image = cv.imread('zelda512.png')

      print('Enter secret image name with extension: ', end = '')
      secret_image_name = input()
      secret_image = cv.imread(secret_image_name)
      #secret_image = cv.imread('lena256.png')

      print('Enter output image name: ', end = '')
      output_image_name = input()
      #output_image_name = 'stegano.png'

      output_image_name_check, output_image_extension =
output_image_name.split('.')
```

```
     if output_image_name_check == 'jpeg' or output_image_name_check
== 'jpg':
          output_image_name = output_image_name + '.png'
          print('Output file name changed to: ', output_image_name)

     # Check how many MSB's of secret image can be embedded given
host images size
     hostrows,   hostcols,   hostchannels   = host_image.shape
     secretrows, secretcols, secretchannels = secret_image.shape

     max_msb = 8
     while (hostrows * hostcols < secretrows * secretcols *
max_msb):
          max_msb -= 1

     if max_msb == 0:
          print('Error! max_msb = 0')
          exit(1)

     return host_image, secret_image, output_image_name, max_msb

def extract_get_files():
     print('Enter steganography image name with extension: ', end =
'')
     steg_image = cv.imread( input() )
     #steg_image = cv.imread('stegano.png')

     print('Enter output image name: ', end = '')
     output_image_name = input()
     #output_image_name = 'extracted.png'

     output_image_name_check, output_image_extension =
output_image_name.split('.')
     if output_image_name_check == 'jpeg' or output_image_name_check
== 'jpg':
          output_image_name = output_image_name + '.png'
          print('Output file name changed to: ', output_image_name)

     return steg_image, output_image_name

def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 255.0
```

```
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr

def embed(host_image, secret_image, max_msb):
     hostrows,   hostcols,   hostchannels   = host_image.shape
     secretrows, secretcols, secretchannels = secret_image.shape

     if hostrows != 512 or hostcols != 512:
          print('Host image size should be 512x512')
          exit(1)

     if (secretrows != 256 or secretcols != 256):
          print('Secret image size should be 256x256')
          exit(1)

     curr_host_row = 0
     curr_host_col = 0

     for row in range(0, secretrows):
          for col in range(0, secretcols):
               r, g, b = secret_image[col, row]
               binval = '{0:08b}'.format(r)
               bin_MSBs = binval[0: max_msb: 1]

               for bit in bin_MSBs:
                    hr, hb, hg = host_image[curr_host_col,
curr_host_row]
                    hostbinval = '{0:08b}'.format(hr)
                    newbin = hostbinval[0: 7: 1] + bit
                    new = int(newbin, 2)

                    host_image[curr_host_col, curr_host_row] =
(new, new, new)

                    curr_host_col += 1
                    if (curr_host_col == hostcols):
                         curr_host_col = 0
                         curr_host_row += 1

     return host_image

def extract(steg_image):
     max_msb = 4
     stegrows, stegcols, stegchannels = steg_image.shape
```

```
	if stegrows != 512 or stegcols != 512:
		print('Stegano image size should be 512x512')
		exit(1)

	new_img_layer = np.zeros((256, 256, 3), dtype = np.uint8)

	curr_new_img_row = 0
	curr_new_img_col = 0

	traveled_pixel_count = 0
	LSBs_buffer = ''

	for row in range(0, stegrows):
		for col in range(0, stegcols):
			sr, sb, sg = steg_image[col, row]
			binval = '{0:08b}'.format(sr)
			lsb = binval[7]
			LSBs_buffer += lsb
			traveled_pixel_count += 1

			if traveled_pixel_count % max_msb == 0:
				padding_size = 8 - max_msb
				for i in range(0, padding_size):
					LSBs_buffer += '0'

				dec_pixel_val = int(LSBs_buffer, 2)
				new_img_layer[curr_new_img_col,
curr_new_img_row] = (dec_pixel_val, dec_pixel_val, dec_pixel_val)

				LSBs_buffer = ''

				curr_new_img_col += 1
				if curr_new_img_col == 256:
					curr_new_img_col = 0
					curr_new_img_row += 1

	return new_img_layer

def main():
	print('1 - Embed\n2 - Extract\nPlease enter mode: ', end = '')
	mode = input()

	while (mode != '1' and mode != '2'):
		print('Undefined input! Plase try again: ', end = '')
		mode = input()
```

```
    if (mode == '1'):
        host_image, secret_image, output_image_name, max_msb =
embed_get_files()
        embedded_image = embed(host_image, secret_image, max_msb)
        cv.imwrite(output_image_name, embedded_image)
        print('PSNR: ', PSNR(host_image, embedded_image))

    if (mode == '2'):
        steg_image, output_image_name = extract_get_files()
        cv.imwrite(output_image_name, extract(steg_image))

if __name__ == '__main__':
    main()
```