

ENGR 3340 / CSCI 6397

DIgItal Image ProcessIng

Term Project

#### LSB Steganography: Text Hiding and Image Hiding

#### INSTRUCTOR

**Dr. Mahmut KARAKAYA**

**Hunter Moore**

**B001144074**

**1. ABSTRACT**

For my term project, we chose to implement text-hiding and image-hiding within a given target image using least-significant-bit steganography. *Steganography* is the practice of embedding message data (text, image, audio, etc.) within a target data file such that it is **imperceptible** to the human visual system. This distinguishes steganography from *cryptography***,** where the user can **detect** that a hidden message has been embedded but may or may not have the means or knowledge required to extract it. Steganographic images are highly susceptible to corruption by noise and compression. We used Peak Signal to Noise Ratio (PSNR) and Mean Square Error (MSE) as evaluators for our output steganographic images.

**2. Test RESULTS**

**2.1 Text Hiding**

For our first task, we hid a text message within a target image. Hidden message text was supplied by a text file (‘message.txt’) and embedded in the target image (‘lena.png’). By targeting the least-significant bitplane, we can easily hide a message within the image without any noticeable artifacts; however, when noise is added to the image, the output message is corrupted.

Input message is: Hello! This is a secret message.

Output message is: Hello! This is a secret message.

MSE: 0.000172

MSE: 0.000000

MSE: 0.000000

PSNR: 37.653274

|  |  |
| --- | --- |
| **(a)** | **(b)** |

**Figure 1:** (a) input image ‘lena.png’ with histogram. (b) output image with histogram.

**2.2 Image Hiding**

For our second task, we embedded a watermark image (‘lena.png’) within a target cover image (‘Falcon.jpg’). Applying noise to the image does not impact the embedded image to the same extent as text hiding. Again, we used LSB steganography to embed the watermark into the target image. PSNR and MSE were used to evaluate our output images.

please select the target bitplane: 1

MSE of noiseless watermarked image: 0.166708

MSE of noiseless watermarked image: 90.014960

MSE of noiseless watermarked image: 505.959275

PSNR of noiseless watermarked image: -27.753489

MSE of noisy watermarked image: 215.943564

MSE of noisy watermarked image: 311.743578

MSE of noisy watermarked image: 811.098342

PSNR of noiseless watermarked image: -31.267110

|  |  |
| --- | --- |
| **(a)** | **(b)** |
| **(c)** | **(d)** |
| **(e)** | **(f)** |
| **(g)** |

**Figure 2:** (a) Our original cover image, (‘Falcon.jpg') and the image to be embedded (‘lena.png’). (b) Embedded image with a threshold at 150 gray levels. (c) Generated tile watermark. (d) watermarked image. (e) Watermarked image with gaussian noise (f) extracted watermark from noiseless image. (g) extracted watermark from noisy image.

**3. MATLAB CODES**

**3.1 Code for Text Hiding**

%Hunter Moore

%001144074

%Term Project: LSB Text Hiding

clc;

clear;

close all;

%----------Embedding----------

%Get the cover image

inimage = imread('lena.png');

cover = inimage;

%Get message to embed in image

file= fopen('message.txt','rb');

%Save message as a char array

message = fread(file,'char');

%Close file

fclose(file);

%Ensure that the input message does not exceed the size of the cover image

assert(numel(cover) > numel(message)\*8, 'ERROR: message is too large for cover image');

%Insert some uncommon character to append to the message text as a

%terminator.

eof = 'þ';

messageText = [message;eof];

%Each character needs to be converted into 8-bit binary form

binary = transpose(dec2bin(messageText,8));

%Find the least-significant bits to be set to one or zero.

zeroBit = find(binary == '0');

oneBit = find(binary == '1');

%Set the values of the least-significant bits

cover(zeroBit) = bitset(cover(zeroBit),1,0);

cover(oneBit) = bitset(cover(oneBit),1,1);

%----------Extraction----------

%Note that output is usually corrupted when noise is introduced.

%Increasing the level of noise increases the level of corruption.

%cover = imnoise(cover,'salt & pepper', 0.0002);

outputMessage = [];

for i = 1:8:numel(cover)

chars = bitget(cover(i:i+7),1);

chars = bin2dec(num2str(chars));

if(chars == eof)

break;

else

outputMessage(end+1) = chars;

end

end

outputMessage = char(outputMessage);

%----------Output----------

figure(1);

subplot(1,2,1);

imshow(inimage);

subplot(1,2,2);

imhist(inimage);

suptitle('Original Image');

figure(2);

subplot(1,2,1);

imshow(cover);

subplot(1,2,2);

imhist(cover);

suptitle ('Image with embedded message');

out = sprintf( 'Input message is: %s\nOutput message is: %s',message,outputMessage);

fprintf(out);

%Use MSE and PSNR to evaluate output

cover = double(cover);

inimage = double(inimage);

MSE = mse(cover,inimage);

out=sprintf('\nMSE: %f',MSE);

fprintf(out);

peaksnr = psnr(cover,inimage);

out=sprintf('\nPSNR: %f',peaksnr);

fprintf(out);

**3.2 Code for Image Hiding**

%Hunter Moore

%001144074

%Term Project: LSB Image Hiding

clc;

clear;

close all;

%This program can accept color images as inputs

%However, outputs will be grayscale.

%-----Embedding-----

inimage = imread('Falcon.jpg');

cover = inimage;

hiddenImage = imread('lena.png');

figure(1);

subplot(1,2,1);

imshow(inimage);

subplot(1,2,2);

imshow(hiddenImage,[]);

suptitle('Original Image and Hidden Image');

[row,col,chan] = size(cover);

%Check if cover image is RGB or grayscale

%Use only the red channel for color images.

if chan > 1

coverRed=cover(:,:,1);

end

[hrow,hcol,hchan] = size(hiddenImage);

%Check if hidden image is RGB or grayscale

%Use only the red channel for color images.

if hchan > 1

hiddenImage=hiddenImage(:,:,1);

end

%Generate a binary image with the given threshold to define the watermark.

threshold = 150;

binaryImage = hiddenImage < threshold;

figure(2),imshow(binaryImage,[]);

title('Thresholded Hidden Image');

%Set the target bitplane.

BP=input('please select the target bitplane: ','s') ;

BP=str2double(BP);

%Rescale hidden image in scenarios where it is larger than the cover image

if hrow > row || hcol > col

scale = min([row/hrow,col/hcol]);

binaryImage = imresize(binaryImage,scale);

[hrow, hcol] = size(binaryImage);

end

%For scenarios where the hidden image is smaller than the cover image, tile

%the hidden image across the cover image.

if hrow < row || hcol < col

wm = zeros(size(cover),'uint8');

for y = 1:col

for x =1:row

wm(x,y)=binaryImage(mod(x,hrow)+1,mod(y,hcol)+1);

end

end

wm = wm(1:row,1:col);

else

%When watermark and cover image are the same size

wm = binaryImage;

end

figure(3),imshow(wm,[]);

title('Generated Watermark');

%Generate watermarked image

wmImage = cover;

for z = 1:chan

for y= 1:col

for x = 1:row

wmImage(x,y,z) = bitset(cover(x,y),BP,wm(x,y));

end

end

end

figure(4),imshow(wmImage,[]);

title('Watermarked Image');

%Generate a noisy watermarked image

wmImageNoise = imnoise(wmImage,'gaussian',.02);

figure(5),imshow(wmImageNoise);

title('Noisy Watermarked Image');

%-----Extraction-----

%Initialize the size of extracted watermarks from noisy and noiseless

%images

extWM = zeros(size(wmImageNoise));

extWMNoise = zeros(size(wmImageNoise));

for z = 1:chan

for y = 1:col

for x = 1:row

%Extract the watermarks

extWM(x,y,z) = bitget(wmImage(x,y),BP);

extWMNoise(x,y,z)=bitget(wmImageNoise(x,y),BP);

end

end

end

%Rescale extracted watermarks.

extWM = uint8(255 \* extWM);

extWMNoise = uint8(255 \* extWMNoise);

%Display output extracted watermarks

figure(6);

imshow(extWM,[]);

figure(7);

imshow(extWMNoise,[]);

%Evaluate Output using MSE and PSNR

wmImage = double(wmImage);

wmImageNoise = double(wmImageNoise);

inimage = double(inimage);

MSE1 = mse(wmImage,inimage);

out=sprintf('\nMSE of noiseless watermarked image: %f',MSE1);

fprintf(out);

peaksnr1 = psnr(wmImage,inimage);

out=sprintf('\nPSNR of noiseless watermarked image: %f',peaksnr1);

fprintf(out);

MSE2 = mse(wmImageNoise,inimage);

out=sprintf('\nMSE of noisy watermarked image: %f',MSE2);

fprintf(out);

peaksnr2 = psnr(wmImageNoise,inimage);

out=sprintf('\nPSNR of noiseless watermarked image: %f',peaksnr2);

fprintf(out);

**4. DISCUSSION**

In this project, I used LSB steganography for text hiding and image watermarking. For future work, I would like to solve the issue for image hiding where the output images are grayscale. For this project, I decided to cut out audio hiding given the time constraints. I also deviated from my original project proposal, where I suggested using Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT) as the embedding functions. Ultimately, I found that LSB embedding is easier to implement. In the future, I may continue working to apply steganographic techniques in the frequency domain.