Digital Watermarking: Embedding and Extraction in Signal Processing

Jaiden Gann

EE 384 Project Report

Introduction

The goal of this project is to use MATLAB to implement a watermarking system that embeds a watermarked image into another image. Through this project, two watermarking techniques are used: Least Significant Bit (LSB) and Discrete Cosine Transform (DCT). They are compared visually, using peak signal to noise ratio value (PSNR), and using the structural similarity index value (SSIM). The "*Theory*" section will contain more information on the watermarking techniques and evaluation methods. The next section "*Simulation*" will describe the steps taken to implement the watermarking system. In the "*Results*" section, the figures generated along with comments on the implementation and what you're seeing will be discussed. The references used for this project along with the code will be in the *"References"* and "*Appendix*" sections respectively.

Theory

The main objective of watermarking is to ensure the invisibility of a watermark to provide a way to secure digital content. Digital content today can be easily possessed through illegal means, duplicated, and then distributed out. The watermark should be robust against attacks such as compression, filtering, and noise addition. Theres typically two steps: embedding and extracting. Embedding inserts the chosen watermark which can be an image or text into the original or host media by modifying its contents based on an algorithm. Extraction retrieves the watermark in order to verify the authenticity of the host media.

The two domains of watermarking used in this project are Spatial and Transform. LSB substitution belongs to the Spatial Domain and is the most commonly used algorithm for that domain. It typically replaces the least significant bit of the original image with bits of the watermark image. There are some algorithms where the number of least significant bits can be specified. Since the least significant bits are being replaced the watermark is imperceptible to the human eye. From the transform domain, DCT was implemented. DCT separates the image into its frequency coefficients which can be expressed as a sum of cosine functions. In Figure 2, N is the number of data samples, x(n) is the input data sample, y(k) is the DCT coefficients, and α(k) is the scaling factor which controls the visibility of the watermark.

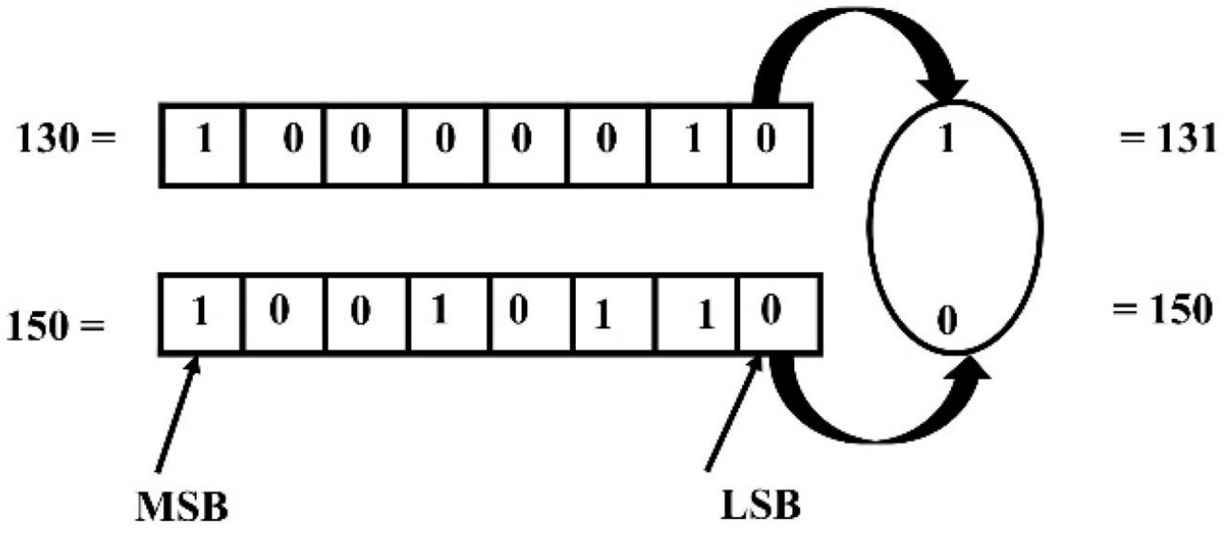


Figure 1. Example LSB technique

$$y(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)k}{2N}\right),\ k = 0, 1, \ldots, N-1$$

Figure 2. 1D DCT equation

$$x(n) = \sum_{k=0}^{N-1} \alpha(k) y(n) \cos\left(\frac{\pi(2n+1)k}{2N}\right),\ n = 0, 1, \ldots, N-1$$

Figure 3. Inverse DCT equation

To compare both algorithms, the PSNR value, SSIM value, and computation time were calculated, along with the original and watermarked image being displayed. The calculation time helps compare the complexity and speed of the two algorithms. PSNR measures the amount of distortion introduced during the watermarking process. A high PSNR means there is a higher similarity between the original and watermarked image i.e., better image quality. It is a measure of pixel intensity and does not consider human visual perception. SSIM assesses the similarity between two images by considering luminance, contrast, and structure. It ranges from -1 to 1, with 1 meaning there is a higher similarity between the two images. Both metrics use the mean square error (MSE) which measures the average of the square of the errors, where the error is the difference between the estimator and estimated outcome.

$$\mathrm{MSE} = \frac{1}{MN} \sum_{n=0}^{M} \sum_{m=1}^{N} \left[\hat{g}\left(n,m\right) - g\left(n,m\right)\right]^2$$

$$\mathrm{PSNR} = 10\log_{10}\left(\mathrm{peakval}^2\right)/\mathrm{MSE}$$

$$\mathrm{SSIM}\left(x,y\right) = \frac{\left(2\mu_x\mu_y + C_1\right)\left(2\sigma_x\sigma_y + C_2\right)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}$$

Figure 4. MSE, PSNR, and SSIM Equations

Simulation

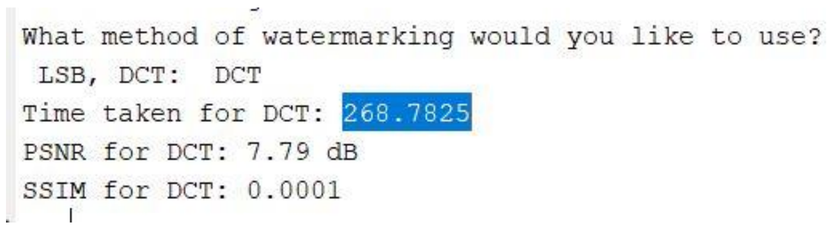
The system starts out by prompting the user to select an image to watermark by open a file explore. It then asks for the image to use as a watermark and what method the user wants to use. Based on the method inputted by the user, the correct embedding algorithm will be used and then the PSNR and SSIM are calculated.

The embedDCT function takes the watermark image and resizes to be smaller, in this case 64 by 64. Both the watermark image and original image are then changed to grayscale. Discrete cosine transform is performed on both images. Then a 64 by 64 section of the original image is replaced with each pixel being added to alpha and a DCT coefficient from the watermark. Currently, alpha is 0.5 but it can be adjusted between 0 and 1 to change the strength of the watermark as needed. The inverse DCT is then taken of the original image to get back the watermarked image, which is then saved and displayed.

The embedLSB function also takes the watermark image and resizes it to be a 64 by 64 and changes the images to grayscale. It also resizes the watermark image to match the original. The watermark image is scaled to have values between 0 and 1 and is then changed by multiplying alpha by the watermark image and adding the original image values. This is the part that replaces the bits. Currently, alpha is set to 0.7 and as with DCT can be changed to modify the strength of the watermark. In this case the difference in strength can be seen when the extracted image is displayed. The pixel values are clipped so that it stays in a range of 0 to 255. The original and watermarked image are then displayed to the user and the watermarked image is saved. The embedLSB function also extracts the watermark from the image since there is no visual difference when watermarking with LSB. To extract, the opposite steps of embedding of performed. The watermarked image’s pixel values are subtracted from the original image and then divided by alpha. The extracted watermark is rescaled to a range of 0 to 255 and then displayed.

PSNR and SSIM are calculated using the equations shown in Figure 4. The computation time is found by using MATLAB’s own stopwatch timer. All three values are displayed to the user in the command window.

To use the watermarking system, to make it easier place any images you want watermarked in the images folder within the project folder. The image to be used as the watermark photo is in the top-level structure with all the scripts. The watermarked images will also be saved there. There are images provided in the image folder, however, you should be able to download any image from the internet and the system should watermark it. If you want to use something other than a jpg such as a png, you’ll need to make it so you can view all files or filter to what you want in your file explorer. As a note, each time you run DCT and LSB respectively, they will overwrite the previous watermarked image that was saved. I did attempt to implement my own DCT and inverse DCT function, however, it took way too long to compute the values so for time I used Matlab’s builtin. Below is the results from using my own DCT and inverse DCT function. The code was also provided with the other files and in the appendix.



```
What method of watermarking would you like to use?
 LSB, DCT:  DCT
Time taken for DCT: 268.7825
PSNR for DCT: 7.79 dB
SSIM for DCT: 0.0001
```

Figure 5. Results from own DCT and inverse DCT function

Results

To show the system working, a couple images will be watermarked and the alpha value will be changed so that difference in watermark strength can be seen. Then there will brief comparison between the PSNR, SSIM, and computation time. For the first example, LSB will be used on a jpg with an alpha of 0.7.

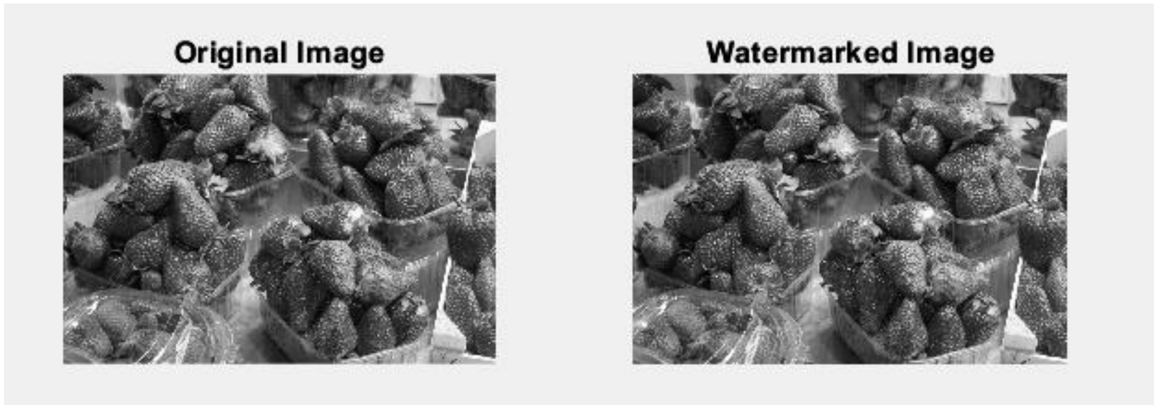


Figure 6. Original vs Watermarked Image

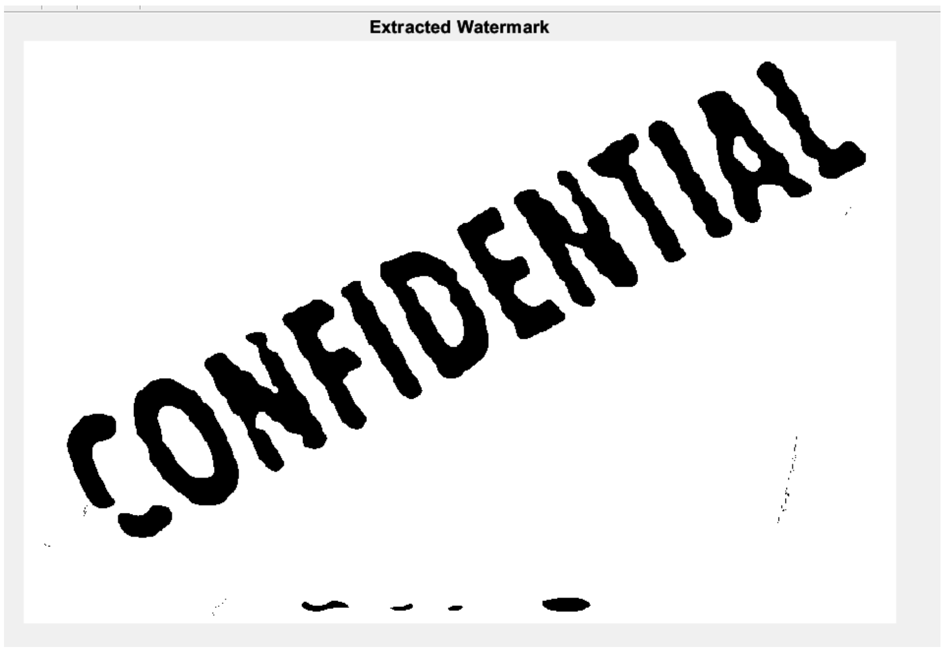
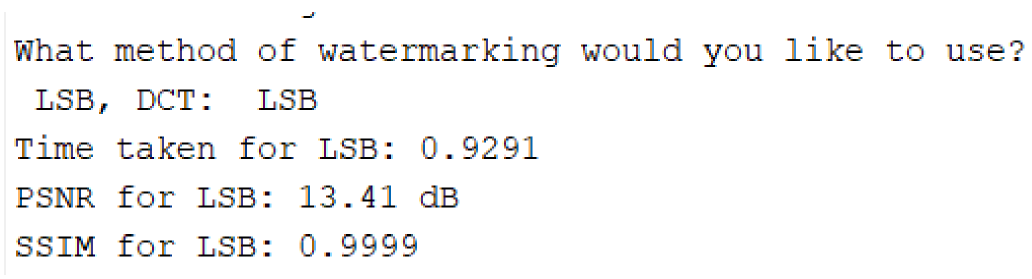


Figure 7. Extracted Watermarked



```
What method of watermarking would you like to use?
 LSB, DCT:  LSB
Time taken for LSB: 0.9291
PSNR for LSB: 13.41 dB
SSIM for LSB: 0.9999
```

Figure 8. SSIM, PSNR, Computation Time

Now we’ll change alpha to be 0.5 and use a png.

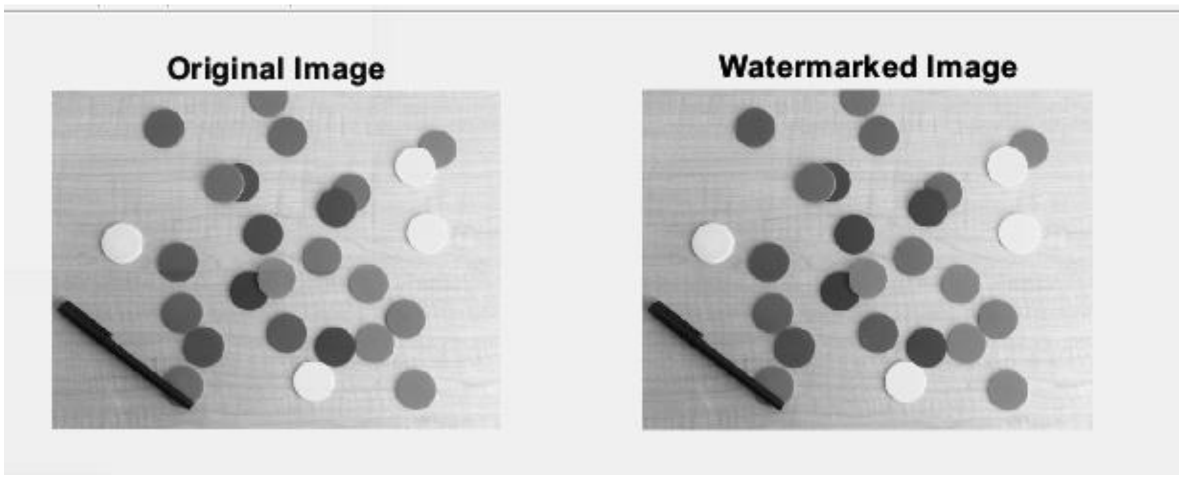


Figure 9. Original vs Watermarked Image

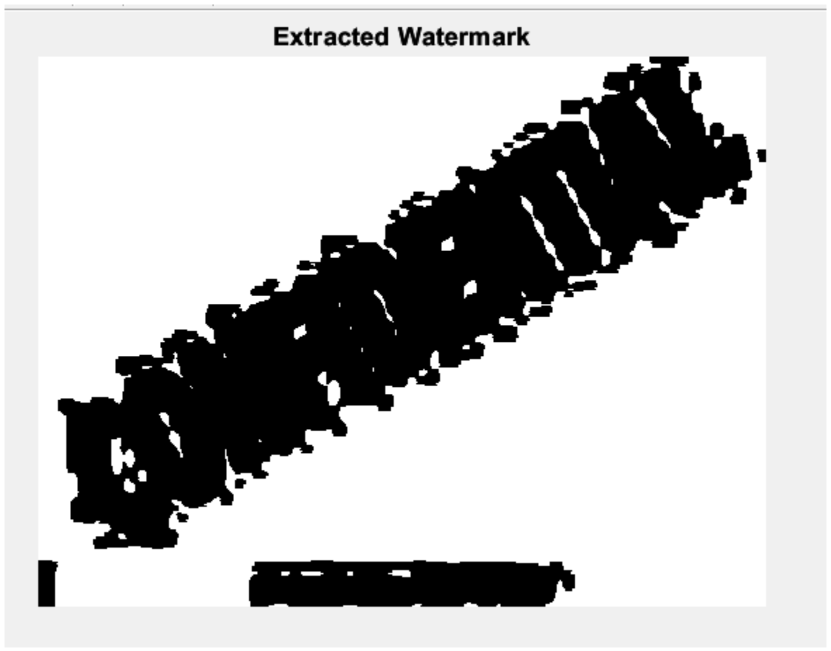
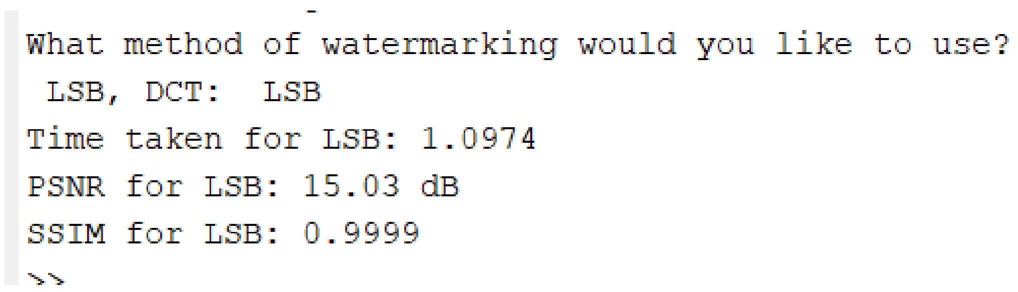


Figure 10. Extracted Watermarked



```
What method of watermarking would you like to use?
 LSB, DCT:  LSB
Time taken for LSB: 1.0974
PSNR for LSB: 15.03 dB
SSIM for LSB: 0.9999
>>
```

Figure 11. SSIM, PSNR, Computation Time

As you can see lowering alpha or in other words the strength of the watermark makes it harder to tell what was extracted from the watermarked image, which can make verifying digital content harder. The lower the alpha the better the PSNR value is, in other words the better the image quality because the watermark isn’t as strong. The SSIM for both watermarked images are close to 1 meaning between the original and watermarked image there no difference structurally. This makes sense since the least significant bits are being replaced. Now DCT will be used on a jpg with an alpha of 0.5.

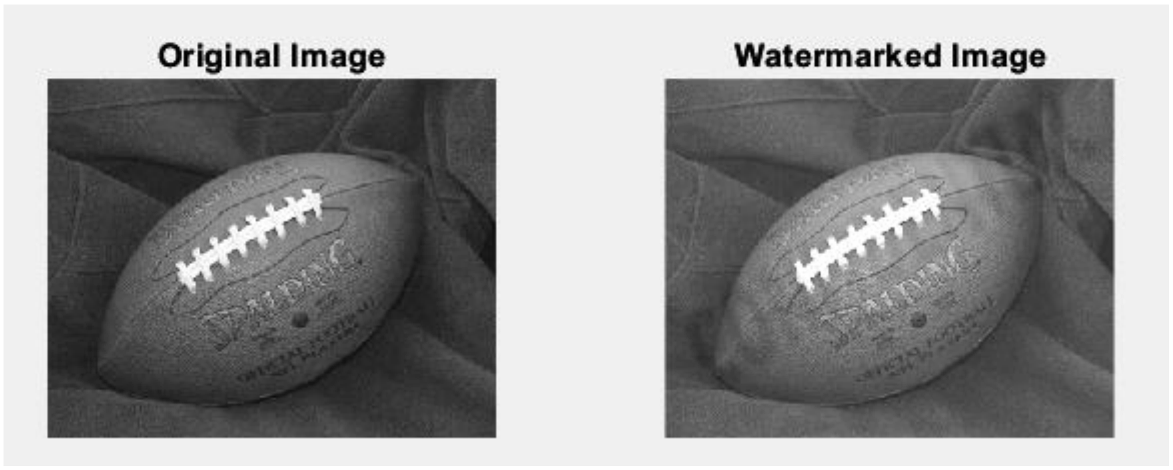
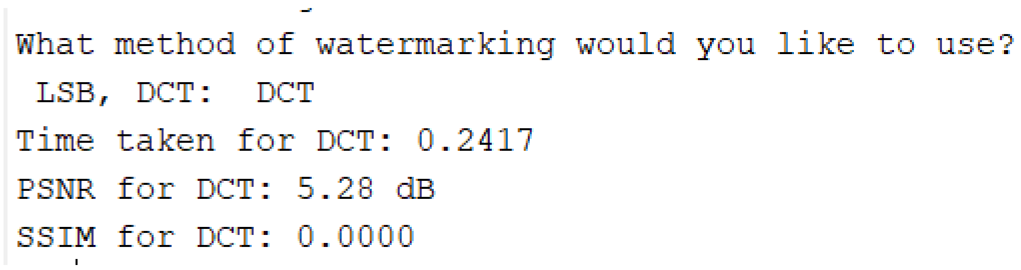


Figure 12. Original vs Watermarked Image



```
What method of watermarking would you like to use?
 LSB, DCT:  DCT
Time taken for DCT: 0.2417
PSNR for DCT: 5.28 dB
SSIM for DCT: 0.0000
```

Figure 13. SSIM, PSNR, Computation Time

Now a png will be used an alpha of 0.8.

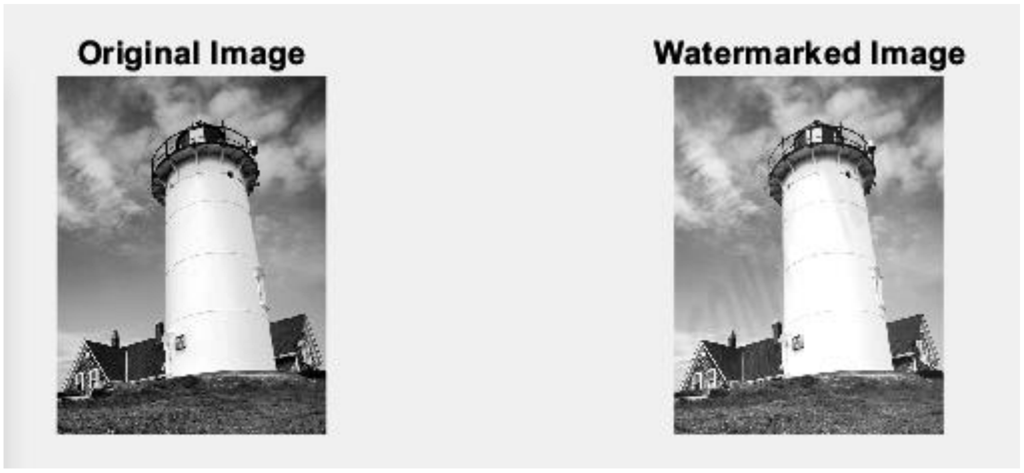
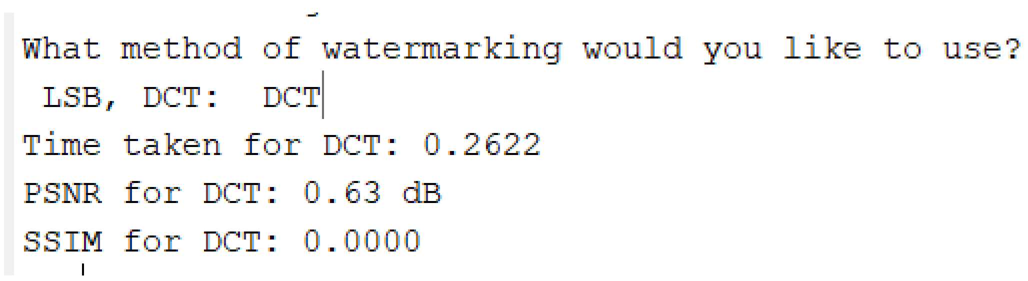


Figure 14. Original vs Watermarked Image



```
What method of watermarking would you like to use?
 LSB, DCT:  DCT
Time taken for DCT: 0.2622
PSNR for DCT: 0.63 dB
SSIM for DCT: 0.0000
```

Figure 15. SSIM, PSNR, Computation Time

Despite using a higher value of alpha, because of the difference in the photos themselves when converted to grayscale you can just faintly see the words confidential in the lighthouse image. The image quality also significantly went down when a higher alpha value was used.

Between LSB and DCT, DCT was a lot faster but at the same value of alpha LSB produced a better-quality image based off the PSNR values. LSB has the better SSIM value because in this case you can visually see that after being watermarked the images look the same, however, that is not always the case.

References

Begum, M.; Uddin, M.S. Digital Image Watermarking Techniques: A Review. *Information* **2020**, *11*, 110. https://doi.org/10.3390/info11020110

“Documentation.” *Documentation - MATLAB & Simulink,* https://www.mathworks.com/help/.

He-Jing, Wu. "A DCT Domain Image Watermarking Method Based on Matlab" International Journal of Advanced Network, Monitoring and Controls, vol.2, no.2, 2017, pp.38-45. https://doi.org/10.21307/ijanmc-2017-008

Sara, Umme, Morium Akter, and Mohammad Shorif Uddin. "Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study." *Journal of Computer and Communications* 7.3 (2019): 8-18.

Z. -X. Wang, K. -Y. Sha and X. -L. Gao, "Digital Watermarking Technology Based on LDPC Code and Chaotic Sequence," in IEEE Access, vol. 10, pp. 38785-38792, 2022, doi: 10.1109/ACCESS.2022.3166475.

Appendix A. MATLAB Code for Watermarking System

```
% Jaiden Gann
% 8/3/23
% Watermarking System

clc;clear;close all;

fprintf("Watermarking System\n\n");
fprintf("To use make sure the images you want to be watermarked are in the folder images\n\n");

% Load the original image and watermark image
fprintf("Select the image to watermark\n")
[filename,folderPath] = uigetfile({'*.jpg'; '*.png'; '*.bmp'});
if filename == 0
    disp('No file selected. Watermarking canceled.');
end
originalImage = imread(fullfile(folderPath, filename));

fprintf("Select the image to use as a watermark\n")
[filename2,folderPath2] = uigetfile({'*.jpg'; '*.png'; '*.bmp'});
watermarkImage = imread(fullfile(folderPath2, filename2));

% Ask for method to be used
%method = input("What method of watermarking would you like to use?\n LSB, DCT:  ", 's');
method = 'lsb'; % for publishing
% Run the embedding process
switch lower(method)
    case 'lsb'
        tic;
        watermarkedImage = embedLSB(originalImage, watermarkImage);
        timeElapsedLSB = toc;
        fprintf("Time taken for LSB: %.4f\n", timeElapsedLSB);
    case 'dct'
        tic;
        watermarkedImage = embedDCT(originalImage, watermarkImage);
        timeElapsedDCT = toc;
        fprintf("Time taken for DCT: %.4f\n", timeElapsedDCT);
    otherwise
        error('Inavlid watermarking method. Support methods are: LSB and DCT')
end

% Calculate PSNR
```

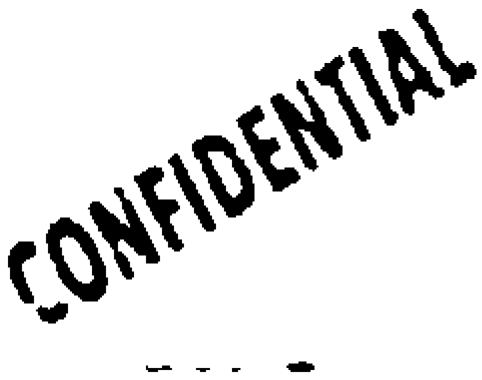
```
switch lower(method)
    case 'lsb'
        % Make images doubles for calculations
        watermarkedImage = double(watermarkedImage);
        originalImage = double(originalImage);
        % Mean Squared Error - metric of distortion
        MSE_LSB = sum(sum(sum((originalImage - watermarkedImage).^2))) / numel(originalImage);
        PSNR_LSB = 10 * log10(255^2 / MSE_LSB); %higher PSNR = better image quality
        fprintf("PSNR for LSB: %.2f dB\n", PSNR_LSB);
    case 'dct'
        % Make images doubles for calculations
        watermarkedImage = double(watermarkedImage);
        originalImage = double(originalImage);

        watermarkedImageDCT = idct2(watermarkedImage);   %reconstruct watermarked image from DCT
coefficients
        MSE_DCT = sum(sum(sum((originalImage - watermarkedImageDCT).^2))) / numel(originalImage);
        PSNR_DCT = 10*log10(255^2 / MSE_DCT);
        fprintf("PSNR for DCT: %.2f dB\n", PSNR_DCT);
    otherwise
        error('Invalid watermarking method. Supported methods are LSB and DCT')
end

switch lower(method)
    case 'lsb'
        originalImage = my_rgb2gray(originalImage);
               watermarkedImage = imresize(watermarkedImage, size(originalImage, [1,2])); %
Resize
        SSIM_LSB = calculateSSIM(originalImage, watermarkedImage);
        fprintf("SSIM for LSB: %.4f\n", SSIM_LSB);
    case 'dct'
        originalImage = my_rgb2gray(originalImage);
        watermarkedImageDCT = idct2(watermarkedImage);
               watermarkedImageDCT = imresize(watermarkedImageDCT, size(originalImage, [1,2])); %
Resize DCT coefficients
        SSIM_DCT = calculateSSIM(originalImage, watermarkedImageDCT);
        fprintf("SSIM for DCT: %.4f\n", SSIM_DCT);
    otherwise
        error('Invalid watermarking method. Supported methods are LSB and DCT');
end
```

```
Watermarking System

To use make sure the images you want to be watermarked are in the folder images

Select the image to watermark
Select the image to use as a watermark
Time taken for LSB: 1.6808
PSNR for LSB: 23.89 dB
SSIM for LSB: 0.9999
```

Original Image

Watermarked Image

Extracted Watermark



Functions

RGB to Gray Scale Funciton

```
function grayImage = my_rgb2gray(A)

    grayImage = 0.3 * A(:,:,1) + 0.6*A(:,:,2) + 0.1 *A(:,:,3);    %equation
for function
```

```
end
```

embedLSB Funciton

```
function watermarkedImage = embedLSB(originalImage, watermarkImage)
    % Load the watermark image (make sure it has the same dimensions as the
host image)
    watermarkImage = imresize(watermarkImage, [64, 64]); % Resize the
watermark image to a smaller size

    if size(originalImage, 3) == 3
        originalImage = my_rgb2gray(originalImage);
    end

    if size(watermarkImage, 3) == 3
        watermarkImage = my_rgb2gray(watermarkImage);
    end

    % Resize the watermark image to match the host image size
    watermark_image = imresize(watermarkImage, size(originalImage, [1 2]));

    % Convert the host and watermark images to double
    host_image = double(originalImage);
    watermark_image = double(watermark_image);

    % Scaling the watermark image to have values between 0 and 1
    watermark_image = watermark_image / 255;

    % Embedding the watermark into the host image (simple additive
watermarking)
    alpha = 0.7; % Watermark strength (adjust this value to control the
visibility of the watermark)
    watermarkedImage = host_image + alpha * watermark_image;

    % Clipping the pixel values to maintain the range [0, 255]
    watermarkedImage(watermarkedImage < 0) = 0;
    watermarkedImage(watermarkedImage > 255) = 255;

    % Convert the watermarked image back to uint8
    watermarkedImage = uint8(watermarkedImage);

    % Display the original image, watermark image, and watermarked image
    figure;
    subplot(2, 2, 1);
    imshow(uint8(host_image));
    title('Original Image');
    subplot(2, 2, 2);
    imshow(watermarkedImage);
    title('Watermarked Image');

    watermarkedImage = double(watermarkedImage);
    host_image = double(host_image);

    % Extraction of the watermark from the watermarked image
    extracted_watermark = (watermarkedImage - host_image) / alpha;

    % Rescaling the extracted watermark to the range [0, 255]
    extracted_watermark = (extracted_watermark - min(extracted_watermark(:)))
```

```
/ (max(extracted_watermark(:)) - min(extracted_watermark(:)));
    extracted_watermark = 255 * extracted_watermark;

    % Convert the extracted watermark to uint8
    extracted_watermark = uint8(extracted_watermark);

    % Display the extracted watermark
    figure;
    imshow(extracted_watermark);
    title('Extracted Watermark');

    % Save the watermarked image
    imwrite(watermarkedImage, 'watermarked_imageLSB.jpg');
end
```

embedDCT Funciton

```
function watermarkedImage = embedDCT(originalImage,watermarkImage)

% Load the watermark image (make sure it has the same dimensions as the host
image)
watermarkImage = imresize(watermarkImage, [64, 64]); % Resize the watermark
image to a smaller size

if size(originalImage, 3) == 3
    originalImage = my_rgb2gray(originalImage);
end

if size(watermarkImage, 3) == 3
    watermarkImage = my_rgb2gray(watermarkImage);
end

% Apply DCT to the host image
dctHost = dct2(double(originalImage));

% Apply DCT to the watermark image
dctWatermark = dct2(double(watermarkImage));

alpha = 0.8; % Watermark strength (can be adjusted as needed)

% Embedding process (modification of DC coefficients)
dctHost(1:64, 1:64) = dctHost(1:64, 1:64) + alpha * dctWatermark;

% Inverse DCT to obtain the watermarked image
watermarkedImage = uint8(idct2(dctHost));

figure;
subplot(2, 2, 1);
imshow(originalImage);
title('Original Image');

subplot(2, 2, 2);
imshow(watermarkedImage);
title('Watermarked Image');

% Save the watermarked image
```

```
imwrite(watermarkedImage, 'watermarked_imageDCT.jpg');
end
```

calculateSSIM Funciton

```
function ssim_value = calculateSSIM(originalImage, watermarkedImage)
    % Parameters for SSIM calculation
    K1 = 0.01;
    K2 = 0.03;

    % Compute means of the images
    mu_x = mean(originalImage(:));
    mu_y = mean(watermarkedImage(:));

    % Compute variances of the images
    sigma_x_sq = var(originalImage(:), 1);
    sigma_y_sq = var(watermarkedImage(:), 1);

    % Compute covariance
    cov_xy = cov(originalImage(:), watermarkedImage(:));

    % Compute SSIM
    C1 = (K1 * 255)^2;
    C2 = (K2 * 255)^2;

    numerator = (2 * mu_x * mu_y + C1) * (2 * cov_xy(1, 2) + C2);
    denominator = (mu_x^2 + mu_y^2 + C1) * (sigma_x_sq + sigma_y_sq + C2);

    ssim_value = numerator / denominator;
end
```

Published with MATLAB® R2022b

My DCT function

```
function D = my_dct2(block)

   % Function to compute the 2D Discrete Cosine Transform (DCT) of a block

   % Input:

   %   block: input 2D block (e.g., 8x8 block)

   % Output:

   %   D: 2D DCT coefficients of the block



   [M, N] = size(block);
```

```
    D = zeros(M, N);


    for u = 1:M
        for v = 1:N
            alpha_u = sqrt(1/M);
            alpha_v = sqrt(1/N);
            if u == 1
                alpha_u = sqrt(1/M)/sqrt(2);
            end
            if v == 1
                alpha_v = sqrt(1/N)/sqrt(2);
            end
            sum_val = 0;
            for x = 1:M
                for y = 1:N
                    sum_val = sum_val + block(x, y) * cos((2*x - 1)*(u-1)*pi/(2*M)) * cos((2*y - 1)*(v-
1)*pi/(2*N));
                end
            end
            D(u, v) = alpha_u * alpha_v * sum_val;
        end
    end
end
```

My inverse DCT function

```
function block = my_idct2(D)
    % Function to compute the 2D Inverse Discrete Cosine Transform (IDCT) of a block
    % Input:
```

```
    %   D: 2D DCT coefficients of the block
    % Output:
    %   block: 2D IDCT of the block (e.g., 8x8 block)

    [M, N] = size(D);
    block = zeros(M, N);

    for x = 1:M
        for y = 1:N
            sum_val = 0;
            for u = 1:M
                for v = 1:N
                    alpha_u = sqrt(1/M);
                    alpha_v = sqrt(1/N);
                    if u == 1
                        alpha_u = sqrt(1/M)/sqrt(2);
                    end
                    if v == 1
                        alpha_v = sqrt(1/N)/sqrt(2);
                    end
                    sum_val = sum_val + alpha_u * alpha_v * D(u, v) * cos((2*x - 1)*(u-1)*pi/(2*M)) * cos((2*y -
1)*(v-1)*pi/(2*N));
                end
            end
            block(x, y) = sum_val;
        end
    end
end
```