Math 414 Final Project – Steganography

Tyler Harmon

Introduction

Steganography, roughly meaning “covered writing” is a method of hiding a piece of information inside of another (Lo, Topiwala, & Wang, 1998). Take, for example, a letter with a secret message hidden inside. This may simply be that the first letter in every line can be taken and turned into a comprehensible sentence, or it may follow a more complex pattern. The letter itself may be innocuous and have a coherent message without knowing the key, but the interpreted message can be completely different than the plaintext one. Steganography is not limited to a particular medium or file type, and can vary in complexity.

In this project, the goal is to hide files of various types and sizes in images using their Haar and Baubechies decompositions. Then, the reconstructed images will be compared to their original counterparts to see how the technique affects the quality of the image.

Mathematical Background

Steganography shares many similarities with its cousin cryptography, but there are a few differences. Primarily, the purpose of the different techniques is usually different. While steganography tries to conceal a message, cryptography focuses on security. Cryptography assumes a bad-actor can intercept whatever encrypted message is being sent and tries to keep them from recovering the actual data. From a functional perspective, cryptography typically secures data by altering the message (image, text, or other data) sent using processes that are difficult to reverse without the proper key (typically some sort of backdoor that is usable only when one has the proper signature). In contrast to this, steganography tends to leave the message relatively unchanged, but instead breaks it up and embeds the chunks of message into the medium. Both of these techniques could be used in tandem to both obscure and secure a secret message being sent.

Some steganographic techniques for hiding data within images apply data directly to the individual pixel values of the image. However, this can affect the image quite drastically. For

example, if two adjacent pixels are the same color in the original image but the adjacent bits of data in the hidden file are sufficiently different, the pixels in the reconstructed image will vary. This can make a passerby or recipient who is looking at the image suspicious, especially if the original image is available for quick visual comparison.

In contrast to this pixel-by-pixel technique, the wavelet-based technique doesn't affect pixels on an individual basis. Instead, the coefficients generated by the decomposition are altered to contain the data. Since the decomposition "smooths" the value of adjacent units, the changes caused by reconstruction after a file is hidden are dispersed over a wider region of the image.

The MRAs used in this project are the Haar and Daubechie II. The Haar decomposition has compact support and is discrete, which is fine since the digital images are pixelated anyways. This wavelet is a simple baseline to test this processing on. The Daubechie MRAs are used often in image compression and processing for their ability to approximate the image well when reconstructing. The db2 wavelet is a continuous wavelet with compact support, which makes it one of the ideal wavelets for our processing (Boggess & Narcowich, 2009).

To extend the wavelet decomposition from 1-D to 2-D, additional steps are taken. Transforms are taken horizontally (row-wise), vertically (column-wise), and diagonally across the image to create three separate transformed images, which are synthesized to get the combined coefficients for the decomposed image.

Application

This application runs through the typical MRA processing steps: sampling and decomposition, processing, and reconstruction. The processing step, however, is based almost entirely on the file to be hidden as opposed to the coefficients calculated by the MRA. The Update_Coefs.m program can load a .mat file created in the 2-D wavelet analysis tool, embed another file inside the coefficients, and save the .mat file to be reconstructed by the wavelet analyzer. This can also extract a file from coefficients that have a file embedded in them.

The program leverages the binary encoding of files to embed the files. Many common types of digital media use 8-bit values (one Btye) per pixel/audio sample/character (eg. .png,

.m4a, and .txt files, respectively). The least significant bits of coefficient values are replaced with substrings of the binary file to reduce the impact the swapped bits have on the decomposed images. For example, the bitstring “11111111” represents the value 255 in most file representations. If we take replace the last 4 bits with “0000” the outcome (“11110000”) is 240, but if we instead replace the most significant 4 bits, the outcome (“00001111”) is 15. A 4-bit substring was chosen as the size to replace to easily subdivide the Bytes in the hidden file; concatenating the last 4 bits of a two adjacent coefficients will yield one Byte of data from the hidden file. Reducing the number of bits replaced per coefficient reduces the effect of the change on the resulting image, but requires many more coefficients to be changed to store the entire file. No compression or modification is done to the hidden file other than separating the bits, so no loss of quality is accrued through this processing.

To recover the file from the coefficients, a 32-bit value is extracted from the first 8 coefficients of the decomposition. This was written in during the embedding process to determine how many values must be read to reconstruct the entire file. Then each proceeding pair of coefficients are taken together: the odd-indexed coefficient contains the most significant 4 bits of the file and the even-indexed coefficient contains the least significant 4 bits. These sets of 4 bits are concatenated to recreate the Byte of data from the file. All of these values are saved together into one binary file and, by opening the file using the proper file extension, the file can be viewed as if it was the original.

Test Images

To test the effect of steganography on the images reconstructed, different images with very different styles were deconstructed and files of varying sizes were hidden inside them. The following images were selected as targets to embed other files within:

Piet Mondrian-esque Modernist Painting

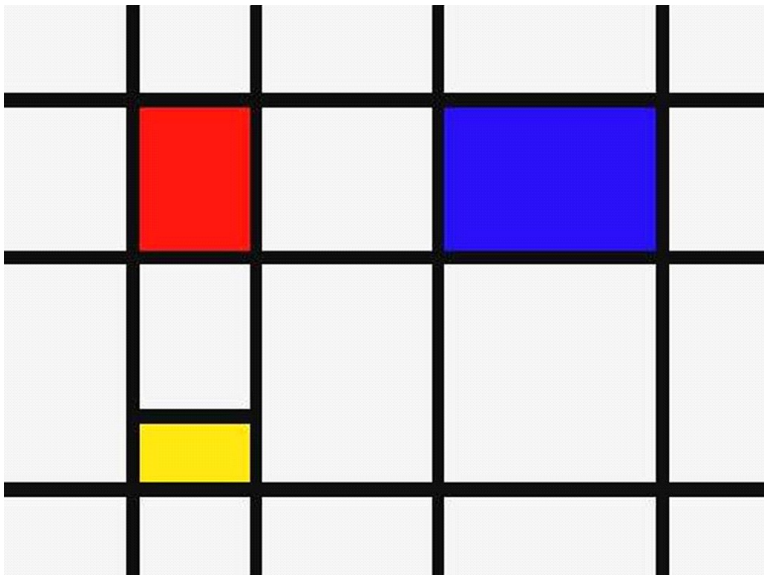

Figure 1: Painting in the style of Piet Modrian; Simple shapes and blocks of color

This image was selected to be the simplest of those tested. The lines are distinct and the colors are constant across the shapes. Thus, changes to a small region of pixels (as would occur with pixel-by-pixel steganography) should stand out quite easily.

Multi-Colored Gradient

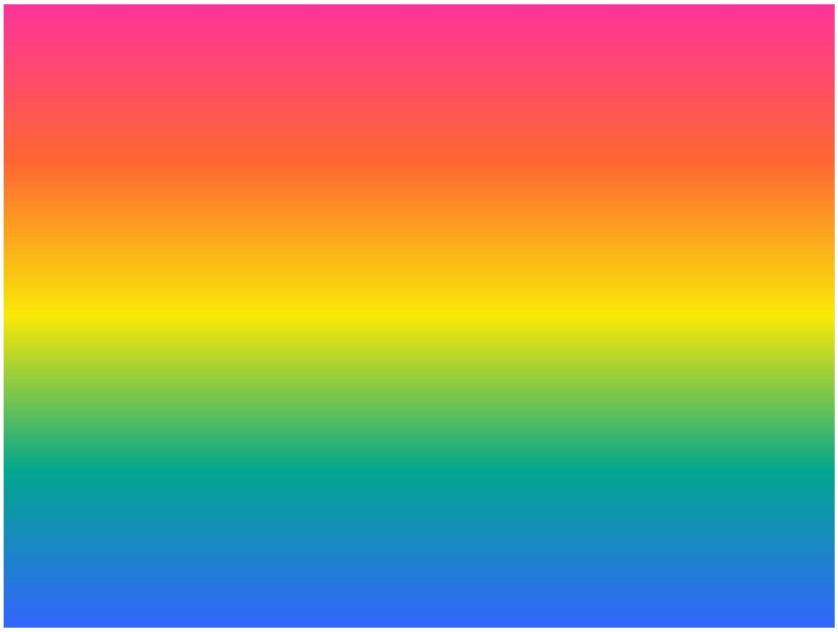

Figure 2: Smooth gradient across multiple colors

This image was generated in Microsoft PowerPoint to directly contrast Figure 1. While the former has clear-cut regions of color, the latter changes gradually with no hard lines.

A Sunday Afternoon on the Island of La Grande Jatte, Georges Seurat

Figure 3: A Sunday Afternoon on the Island of La Grande Jatte. From The Art Institute of Chicago (https://www.artic.edu/artworks/27992/a-sunday-on-la-grande-jatte-1884)

Besides being a phenomenal painting, this image was selected because of the style in which it was created. It uses a technique called “pointillism”, where the entire picture is created using small dots of paint that, when seen together, create the illusion of continuous shapes (not unlike a wavelet approximation). These individual dots make for plenty of inherent noise within the image, meaning it should be all the more difficult to distinguish differences between the original and altered versions.

Test Files

Files of varying size and type were embedded within the test images. Using different file types was done to show the generality of this process, and the increasing amount of data hidden illustrates the visual effect it causes.

Text File

```
hello, world!

1234567890-=qwertyuiop[]asdfghjkl;'zxcvbnm,./!@#$%^&*()_+QWERTYUIOP{}ASDFGHJKL:"ZXCVBNM<>?
```

Figure 4: Sample text file embedded using steganography

This file is the smallest and simplest. Comprised of only 107 bytes of data, this file can be hidden quickly and surreptitiously.

Image File

Figure 5: Texas A&M University logo in .png format

While still a relatively simple image, the 400x400 size means there are 160,000 values in in an uncompressed state, exponentially larger than the text file.

Audio File

Audio files such as songs run through data at a very high rate, so just a few seconds of music can translate into a much larger file than that of a document or image (941KB as opposed to 1KB text document and 18KB image). To test this, a 25-second clip of Saint-Saens's "Carnival

of the Animals – Le Cygne", performed by Yo-Yo Ma, Eugene Ormandy, and Yan Pascal Tortelier was used. This was close to the largest file that could be stored in the test images without increasing the number of bits stored in each coefficient.

Procedure

For each test image, eight decompositions were created. First, the first-level decomposition coefficients were generated using both the Haar and db2 MRAs in MATLAB's Wavelet Analyzer tool. Then for each of these decompositions, each of the test files was embedded in the coefficients and reconstructed to observe changes to the approximations of the images.

Results

The steganography processing algorithm was able to successfully embed the test files into each of the images' decompositions. In addition, the smaller files were completely hidden inside the test images. In Figure 6, both the images synthesized and the histograms representing the coefficients in the original and altered image are almost identical.

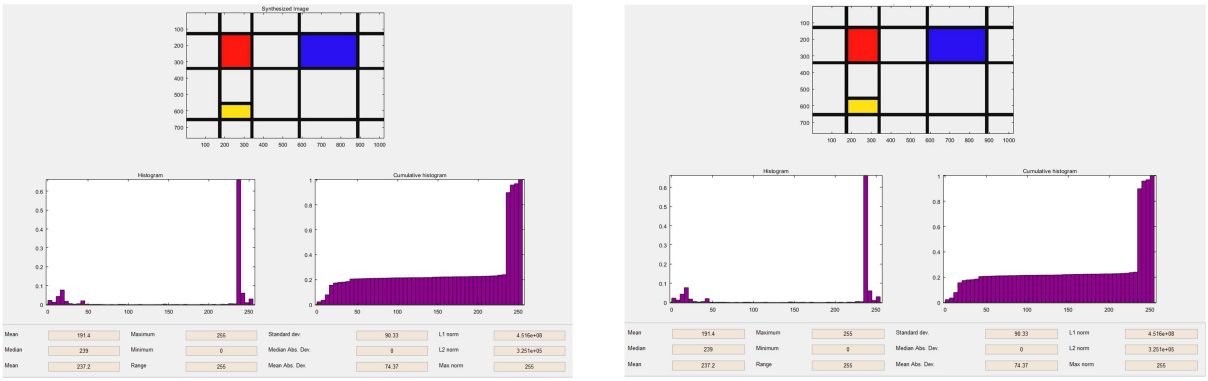

Figure 6: Original db2 decomposition (left) vs. db2 decomposition with text file embedded (right)

Figure 7 shows the test TAMU logo image after being extracted from the gradient's coefficients. This file appears exactly the same as the original. Since the values of the .png were not modified, no change in the output should appear.

Figure 7: Texas A&M logo extracted

For the text and image files hidden, there was little visual difference between the synthesized images recovered from the coefficients. Figure 8 below illustrates a small difference between the preprocessed image and the postprocessed gradient image. In the altered image, there is a faint dark bar that runs vertically down the left side. Looking at the histograms from both tests (Figure 9), they are fairly similar save for the bar at the 250 mark; here, the proportion changes from <0.01 to slightly more than 0.02. This change contributed to the change in shade of the narrow bar in the reconstructed image.

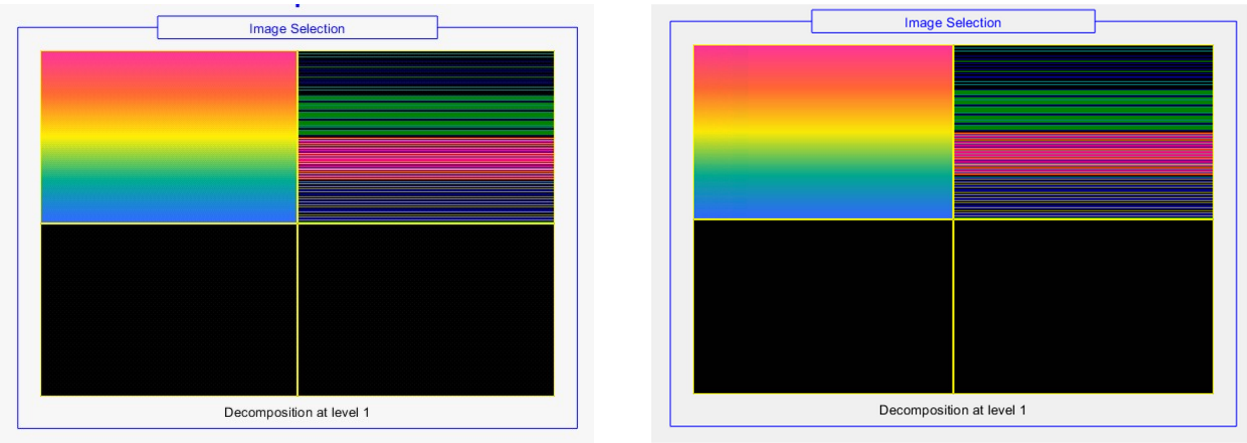

Figure 8: Haar decomposition of original (left) vs. Haar decomposition of logo-embedded (right)

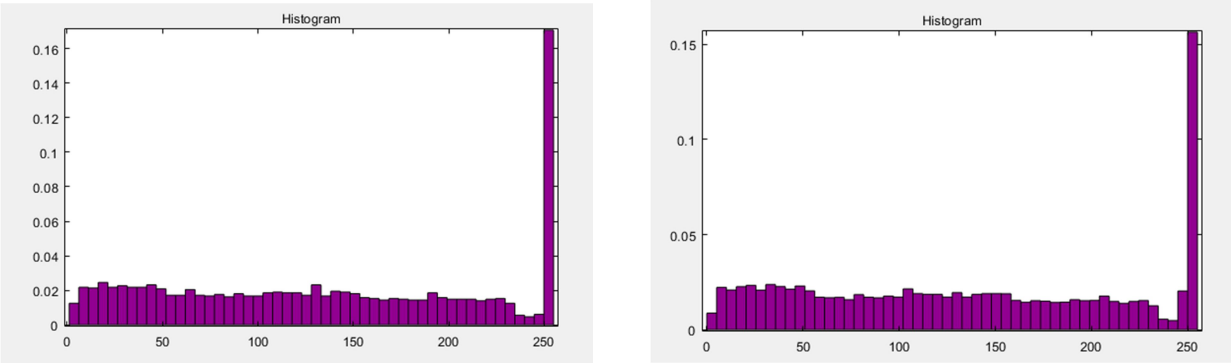

Figure 9: Histogram of original image (left) vs. logo-embedded image (right)

When the audio file was embedded in the test images, the actual hiding was much less successful. With such a large file compared to the size of other test files (and also the file size of the decomposition matrices themselves), almost half of the data in the decomposition had been changed into bits from the audio file. The final reconstructed, approximated images still had much of the same appearance as the originals, but the 1st-level images were quite different. Figure 10 compares the two images from the Seurat piece before and after the audio file was added. The synthesized image at level 1 in the modified image looks as though it was displayed on an old CRT screen with bad signal. Figure 11 shows the decompositions of both of these images in better detail. The decomposition in the original picks out small details, like the edges of shapes, but the second image is filled with so much noise that the decompositions are unintelligible without synthesizing the image. This phenomenon is especially present in the histograms of the coefficients. Figure 12 compares the histogram of the coefficients from the original and processed images. The original has a somewhat bell-curved shape while the processed image has a spike and the first and last bins with the majority of the data in the first and last bin. Although the final approximated image is fairly similar to the original, the underlying decomposition is unrecognizable.

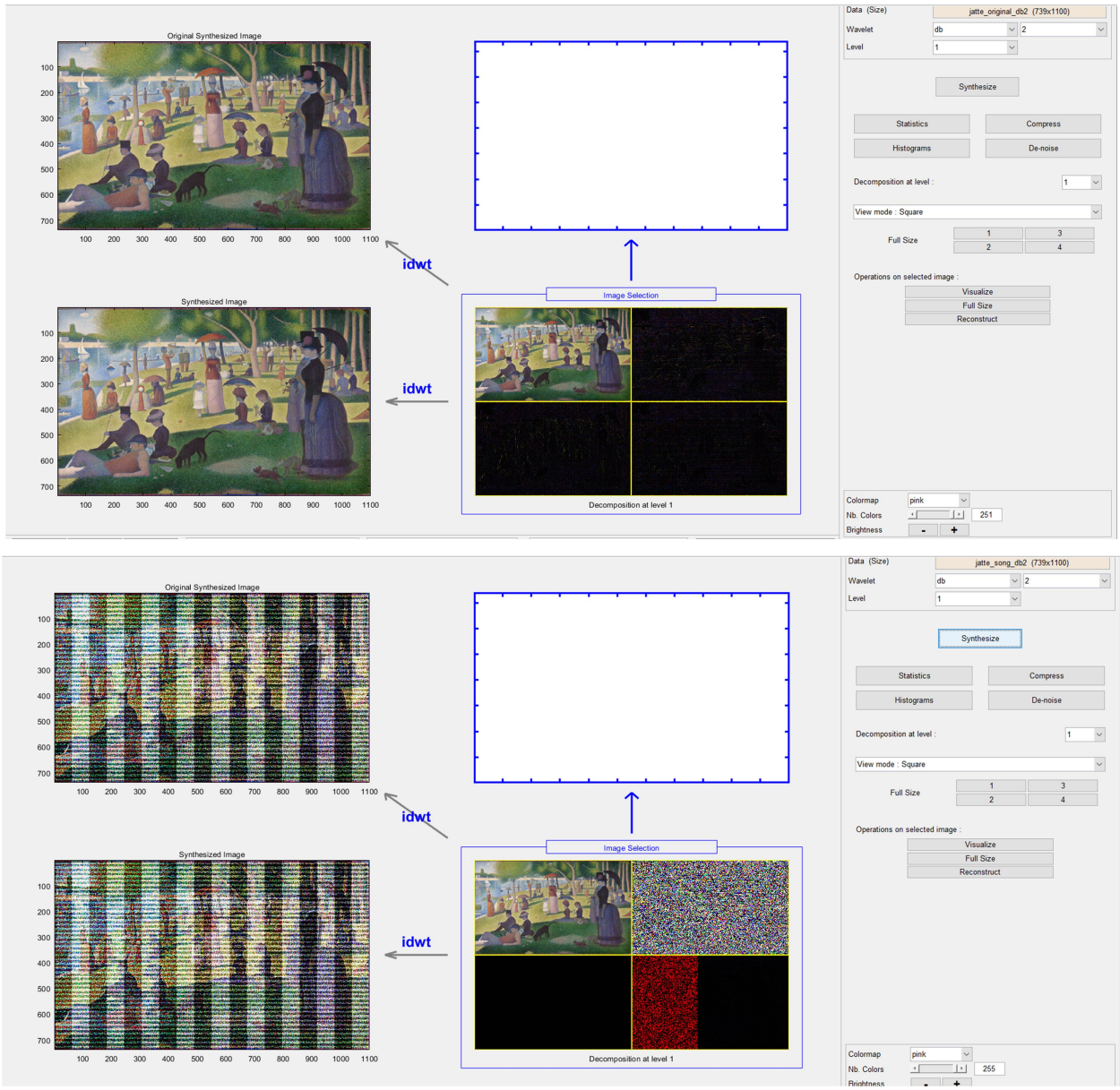

Figure 10: Original db2 decomposition (above) vs. reconstruction with audio embedded (below)

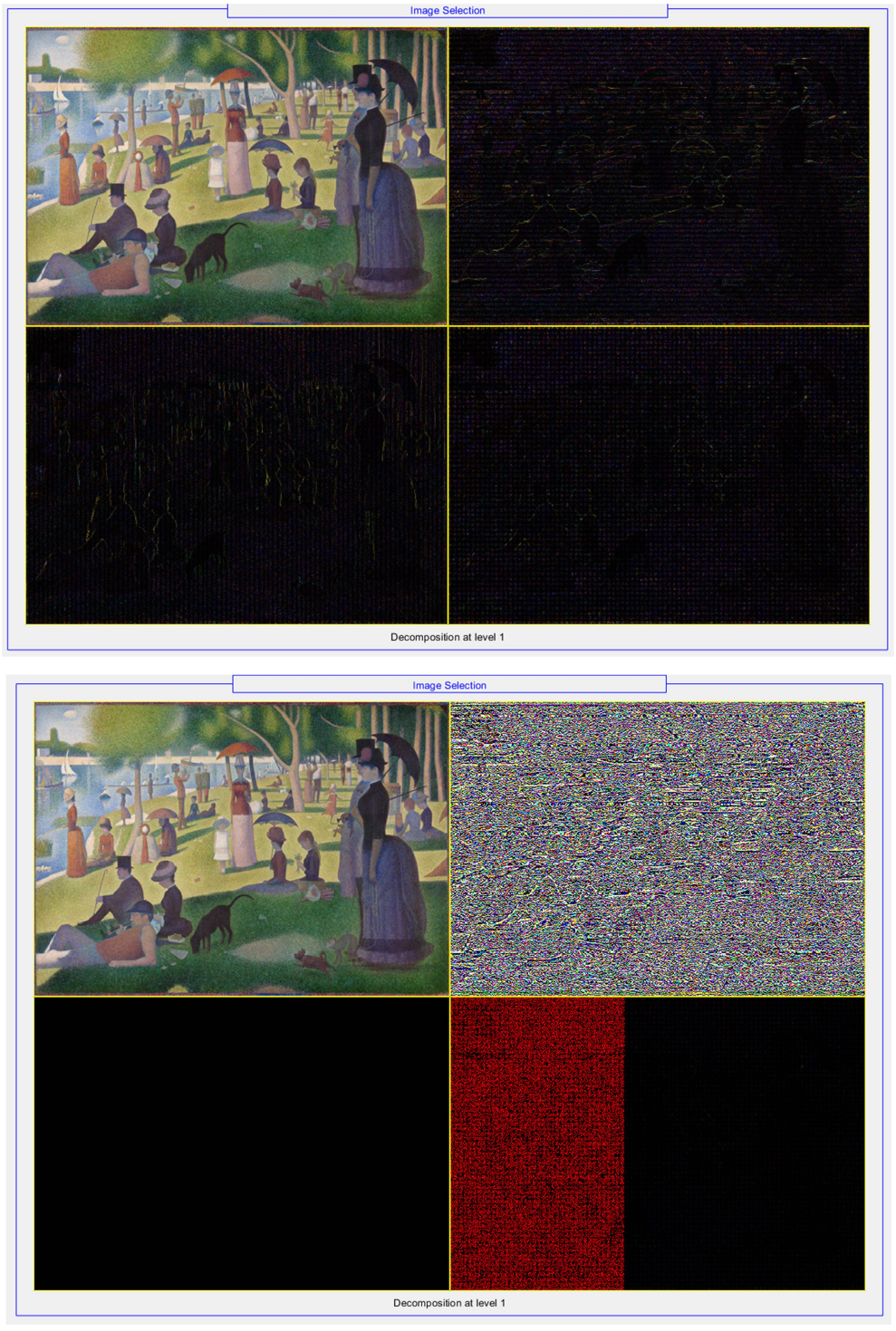

Figure 11: Close-up decompositions of original (above) and audio-embedded (below) images

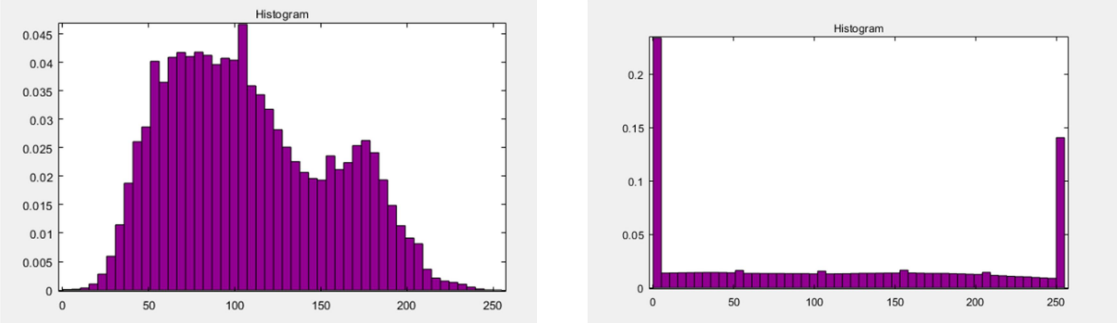

Figure 12: Histogram of Seurat piece initially (left) vs. audio-embedded (right)

The simple image had the greatest visual difference in the large-file-embedding tests. Figure 13 shows the image approximated from those coefficients. Around all of the straight edges, there are light changes in the shades of blocks of color. This creates the illusion of afterimages in this picture. Since there was small, concentrated variation and almost no noise in the original image, the accumulation of small changes in many coefficients was much more apparent.

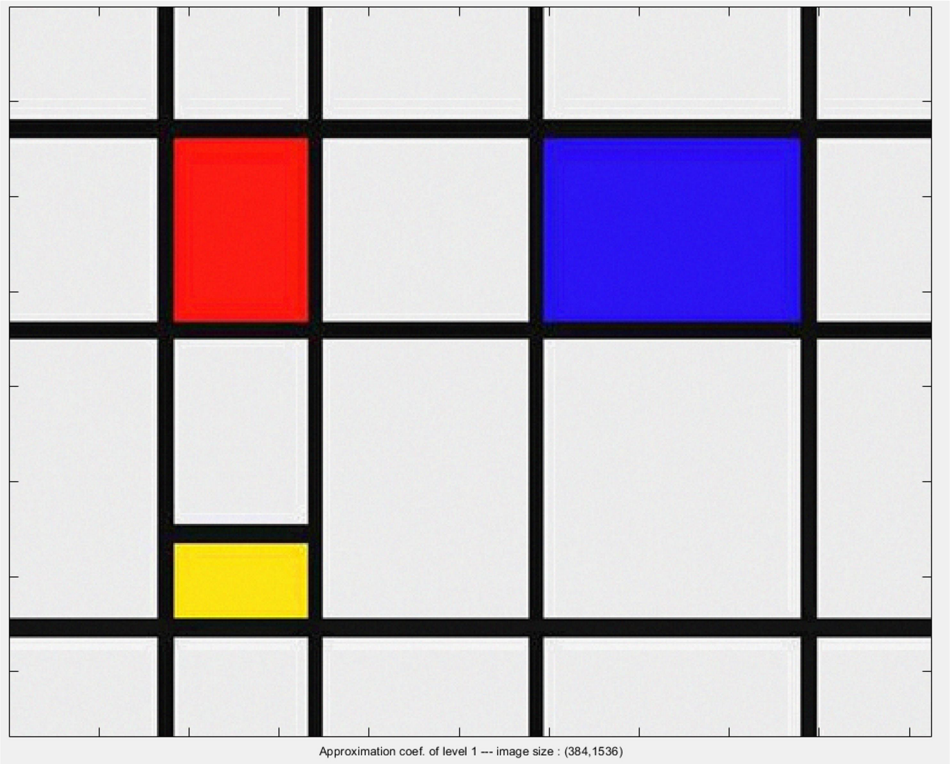

Figure 13: Noise created by the audio file embedded in the coefficients

In this application, there was little difference between the Haar and Daubechie II MRA in the visual output of the synthesized images.

Overall, using wavelet decomposition and reconstruction for simple steganography allows for a very general and fairly robust method of hiding and extracting the binary data of images. As file sizes increase, the differences in the underlying coefficients and visual integrity of the target images become more apparent.

References

"Chapter 6: The Daubechies Wavelets." A First Course in Wavelets with Fourier Analysis, by Albert Boggess and Francis J. Narcowich, Wiley-Blackwell, 2009, pp. 234–249.

Lo, Han-Yang, et al. "Wavelet Based Steganography and Watermarking.", Cornell University, 1998, www.cs.cornell.edu/topiwala/wavelets/report.html.

Seurat, Georges. *A Sunday Afternoon on the Island of La Grande Jatte*. The Art Institute of Chicago, https://www.artic.edu/Artworks/27992/a-Sunday-on-La-Grande-Jatte-1884, 1884, The Art Institute of Chicago, Chicago, IL.

TAMU, ESPN. "Texas A&M University Logo." Twitter.com, 2015, pbs.twimg.com/profile_images/378800000413334813/8a3dbf0c6d9a35f5efcf4c26afa45fff.png.

Code for Update_Coefs.m

```
clc;
fprintf("Welcome to the Steganography Machine!")

while(true)
    fprintf("\nPlease select one of the following options:\n\t1 Load coefficients from
a decomposition\n\t2 Load file to embed in coefficients\n\t3 Hide loaded file in
coefficients\n\t4 Extract file from coefficients\n\t5 Save coefficients\n\t6 Quit\n")
    selection = input("Please enter a number (1-6): ");
    fprintf("Selection: %d\n\n", selection);
    if(isempty(selection))
        selection = 6;
    end
    switch(selection)
        case 1 %Load coefficients
            load_filename = input("Enter filename of the .mat file: ", 's');
            if load_filename == ""
               load_filename = "jatte_original_decomp.mat";
            end

            %Load file
            load(load_filename);
            temp_coefs = coefs;

            fprintf("'%s' loaded.\n", load_filename)

        case 2 %Load file to hide
            hide_filename = input("Enter filename of the file to hide: ", 's');
            if hide_filename == ""
               hide_filename = "hello.txt";
            end

            %Load file
            hide_ID = fopen(hide_filename);
            file_to_hide = fread(hide_ID);

            fprintf("'%s' loadad.\n", hide_filename);

        case 3 %Hide file
            if(~exist('temp_coefs', 'var') || ~exist('file_to_hide', 'var'))
                fprintf("ERROR: Coefficients or file not loaded.\n")
                continue
            end

            fprintf("Size of coefs = %d\n", size(temp_coefs, 2))
            fprintf("Size of data = %d\n", size(file_to_hide, 1))

            temp_coefs = hide_in_coefs(temp_coefs, file_to_hide);
            fprintf("File hidden in coefficients.\n")

        case 4 %Extract file
            if(~exist('temp_coefs', 'var'))
                fprintf("ERROR: Coefficients not loaded.\n")
                continue
            end

            ext = extract_file(temp_coefs, 4);
            save_file = input("Enter a filename to save extracted file. ", 's');
            if save_file == ""
                save_file = "extracted.txt";
            end
```

```
            f_ID = fopen(save_file, "w");
            fwrite(f_ID, ext);
            fprintf("Extracted file saved to %s.\n", save_file)

        case 5 %Save coefficients
            if(~exist('temp_coefs', 'var'))
                fprintf("ERROR: Coefficients not loaded.\n")
                continue
            end

            save_coefs = input("Enter a .mat filename to save coefficients: ", "s");
            S.('coefs') = temp_coefs;
            S.('sizes') = sizes;
            if save_coefs == ""
                save_coefs = "hello.mat";
            end

            save(save_coefs, '-struct', 'S')

        otherwise
            fprintf("Goodbye\n")
            if(exist('f_ID', 'var'))
                fclose(f_ID);
            end

            if(exist('hide_ID', 'var'))
                fclose(hide_ID);
            end
            clear;
            return
    end
end

%--------------------BEGIN FUNCTIONS--------------------%
function new_str = swap_bits(old_str, new_bits)
    %Replaces the last bits of bitstring old_str with the bits of new_str
    old_size = strlength(old_str);
    bits_size = strlength(new_bits);

    new_str = extractBetween(old_str, 1, old_size - bits_size) + new_bits;
end

function altered = hide_in_coefs(target, data)
     %Replaces the last n bits in each coefficient in target with part of %the binary
     code of data


    altered = target;

    %Temp setup: each PAIR of 2 coefficient holds a byte
    %32-bit header to hold number of datapoints to read
    if size(target, 2) < 2 * size(data, 1) + 32
        fprintf("Error: Number of coefficients too small to hold data.\n");
        return
    end

    %Add data to beginning to determine how many values have been stored
    vals_to_read = string(dec2bin(size(data, 1), 32));
    vals_mat = [bin2dec(extractBetween(vals_to_read, 1, 8));
                bin2dec(extractBetween(vals_to_read, 9, 16));
                bin2dec(extractBetween(vals_to_read, 17, 24));
                bin2dec(extractBetween(vals_to_read, 25, 32))];
```

```
    data = [vals_mat; data];

    for i = 1:size(data, 1)
        data_bits = string(dec2bin(data(i), 8));
        left_bits = extractBetween(data_bits, 1, 4);
        right_bits = extractBetween(data_bits, 5, 8);

        target_bits1 = string(dec2bin(altered(1, 2 * i - 1), 8));
        target_bits2 = string(dec2bin(altered(1, 2 * i), 8));

        target_bits1 = swap_bits(target_bits1, left_bits);
        target_bits2 = swap_bits(target_bits2, right_bits);

        altered(1, 2 * i - 1) = bin2dec(target_bits1);
        altered(1, 2 * i) = bin2dec(target_bits2);
    end
end

function extracted = extract_file(target, bits_per_coeff)


    %Determine how many values to read (using 32b tag at beginning of file)
    j = 1;
    num_to_read = "";
    while j * bits_per_coeff <= 32
        temp_coeff = string(dec2bin(target(1, j), 16));
        temp_substr = extractBetween(temp_coeff, 16 - bits_per_coeff + 1, 16);
        num_to_read = num_to_read + temp_substr;
        j = j + 1;
    end

    extracted = zeros(bin2dec(num_to_read), 1);
    for i = 1:bin2dec(num_to_read)
        temp_byte = "";
        k = 1;
        while k * bits_per_coeff <= 8
            temp_coeff = string(dec2bin(target(1, j), 16));
            temp_substr = extractBetween(temp_coeff, 16 - bits_per_coeff + 1, 16);
            temp_byte = temp_byte + temp_substr;
            j = j + 1;
            k = k + 1;
        end
        extracted(i) = bin2dec(temp_byte);
    end
end
```