EE 424 Final Project Report

Name: Jake Hafele and Jonah Frosch
Date: 5/2/2024
Section: 1

I. Least Significant Bit Theory

The overall thought behind Least Significant Bits (LSB) is that images have a color resolution much higher than can be seen by the naked eye. Whether the last bit, or last two bits, or last 3, etc. change, it could have a minimal to no noticeable effect on the image when viewed by a user, since a 1/255 change in red value is minimal. You can take advantage of this trait by forcing all LSB's to 0, and using the LSBs as a form of data storage, hiding data within an image, known as steganography.

By utilizing the structural similarity index function **ssim** in MATLAB, we can compare the difference in the original and modified image which contains a hidden piece of media. This will allow us to compare the impact of hiding multiple bits on different media sizes and the readability of the modified image.

Our project aims to replicate and display varying functions of this process using three major steps:
Hiding Process:

1. Load Original Image to hide data within
2. Load Data to be hidden
3. Convert hidden data to 1D array of bits
4. Clear N least significant bits of every Original Image pixel
5. Set N least significant bits of Original Image pixel to N bits of hidden data

Recovery Process

1. Iterate over every pixel of Modified Image with Hidden Data, reading N hidden bits per pixel in order
2. Convert 1D array of read bits back into original Hidden Data file type (Text, Image, etc)

Analysis

1. Display Original Image and Modified Image with Hidden Data side by side
2. Compare structural similarity index (ssim) of Modified Image with Hidden Data against Original Image

Our project will focus on three different tasks for different types of media to hide within a 2D image, including:

1. Text within an Image
2. Image within an Image
3. Any Media within an Image

Each of these three tasks will ramp up the complexity of the code, while allowing for more possibilities with the least significant bit implementation of Steganography.

II. Hiding Text Within Images

We first began with the goal of embedding a text file into an image of our friend, Caden Kraft. We figured embedding a text image would be a strong start to targeting the goal of embedding an image within another image with the LSB steganography technique.

The code follows a similar structure as outlined in Section 1. Both the original image and target text file to hide are read into our Matlab code. We then convert our matlab code from a String into a 1D array of bits. We had to zero pad by a certain amount of bits so the overall 1D array of the hidden text message would be a multiple of N, with N being the number of bits to encrypt per pixel of the original image.

We utilized a 3D for loop to hide and decrypt the hidden text message from the original image. This enabled us to hide N least significant bits of text data within each color of each pixel of the original image. Once the end of the hidden message was hit, or the last pixel of the original image was met, the hiding for loop would end. The same process would occur in reverse, in which each pixel of the three level nested for loop will read each pixel color to recover N bits of hidden text data per loop iteration.

After the text is recovered, it is rebuilt in reverse from the 1D array of bits, back into a String array. This message is printed to the display in MATLAB, and both the original and modified image are displayed in MATLAB for comparison. Alongside this, the ssim function is utilized, which analyzes the structural similarity of two images to compare the quality of the modified image, which contains a hidden message.

Figure 1 and **Figure 2** demonstrate the full text script of Jerry Seinfeld's The Bee Movie, embedded into a picture of our friend Caden Kraft. The modified image is displayed with multiple N bits encrypted per pixel (N=1, 4, 7), in which it can be seen that with a higher N value each hidden pixel becomes much more corrupted. Based on the structure of our for loop, the pixels starting in the top left corner are altered first, until the hidden string ends, which leads to one part of the image becoming extremely distorted, while other parts are the same as the original image.

Developed code is listed under Appendix A.

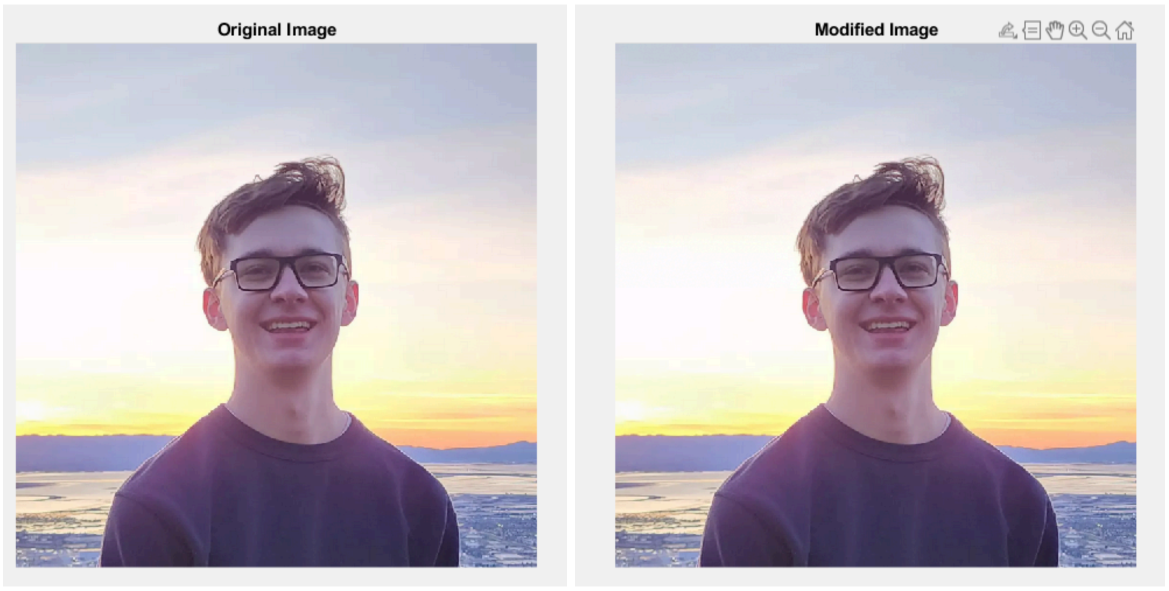

Figure 1. Original Caden Image (Left), Bee Movie Text Embedded into Caden N=1 (Right)

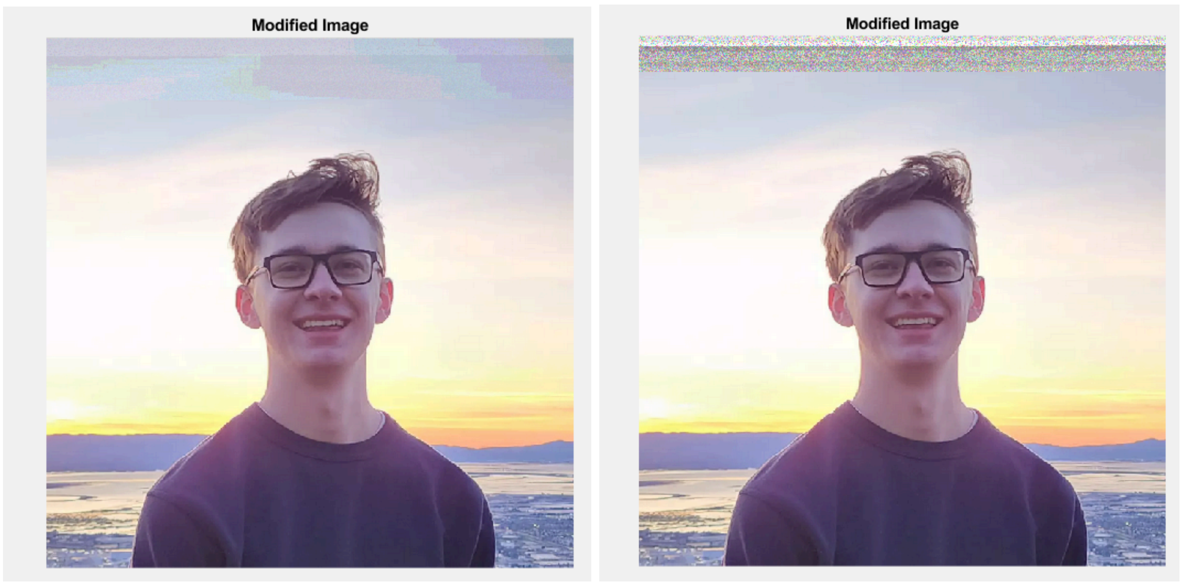

Figure 2. Bee Movie Text Embedded into Caden, N=4 (Left), N=7 (Right)

Table 1 demonstrates the ssim comparison for the Bee Movie text within the image of Caden. The full script was able to be embedded with N=1 bits encrypted into each pixel, which is ideal because the color becomes less corrupted for each pixel. This is backed up by the fact that the resulting ssim value degrades from 1 as the number of N bits increases. As N increases, the hidden data becomes much more compact, but corrupts the target section of the image to a higher degree, leading to a lower ssim value, as seen with N=7 compared to N=1.

N Bits Hidden	1	2	3	4	5	6	7
SSIM	0.99855	0.99672	0.9937	0.98353	0.95263	0.94069	0.93373

Table 1. Structural Similarity Index for Bee Movie Script into Caden

Figure 3 demonstrates the same text file of the Bee Movie reproduced 20 times, to further show the severity of high data with high number of bits encrypted. There is such a large amount of data to be encrypted (793KB) into a smaller image of Caden (442KB). We discovered that Windows uses a form of PNG compression to reduce the file size, which revealed a larger dimension size of the original image to hide a message into. This target image had a dimension of 673x670x3 uint8 values, which was due to lossless compression. The smallest N value to encrypt this data was N=5, which had a severe amount of corruption. When N=7, the top of the image was severely damaged as well, while the bottom remained intact since the entire String text file had been hidden.

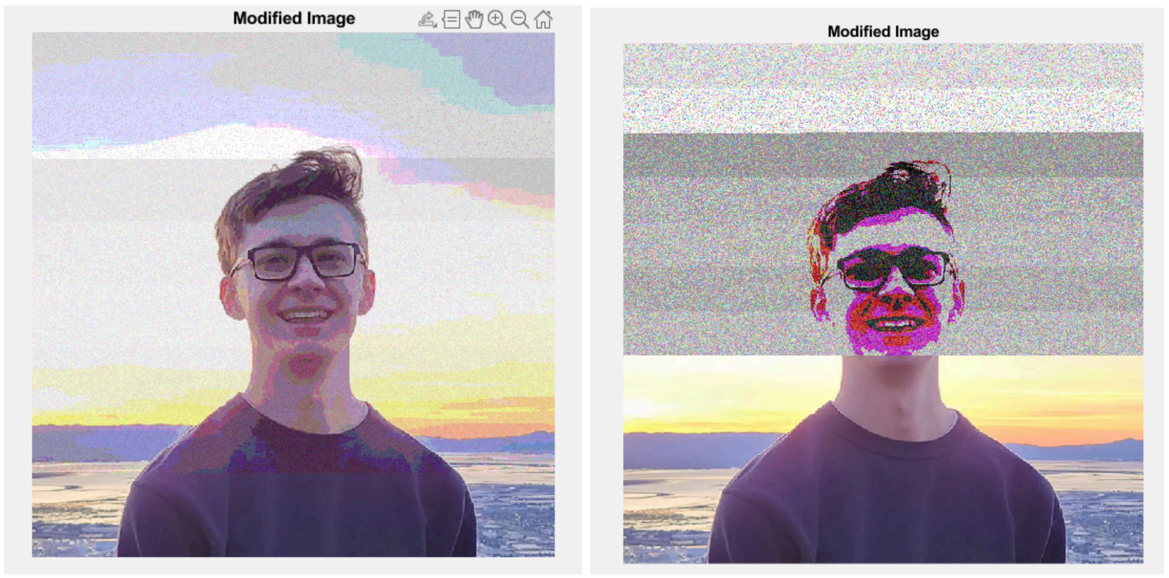

Figure 3. Bee Movie x9 Text Embedded into Caden, N=5 (Left), N=7 (Right)

Table 2 represents the ssim results for embedding the Bee Movie script 20 times within the image of our friend. As mentioned before, the hidden message could not be fully hidden within the image with an N value of less than 5. Even for the larger N values, the ssim value was much lower than before due to having to encrypt 9 times the amount of data.

N Bits Hidden	1	2	3	4	5	6	7
SSIM	Too Small	Too Small	Too Small	Too Small	0.65244	0.46648	0.44219

Table 2. Structural Similarity Index for Bee Movie Script X9 into Caden

III. Hiding Images Within Images

Hiding images into other images had the same process for hiding bits within another 2D image, except it had additional steps for processing the hidden message and recreating it from the modified image. This code can be seen below in Appendix A.

The same original image of our friend Caden was used, as seen in **Figure 4**, alongside a beautiful picture of the Cliffs of Moher from Ireland to hide. The goal was to use an image which would require a larger than N=1 compression to compare the recovered image.

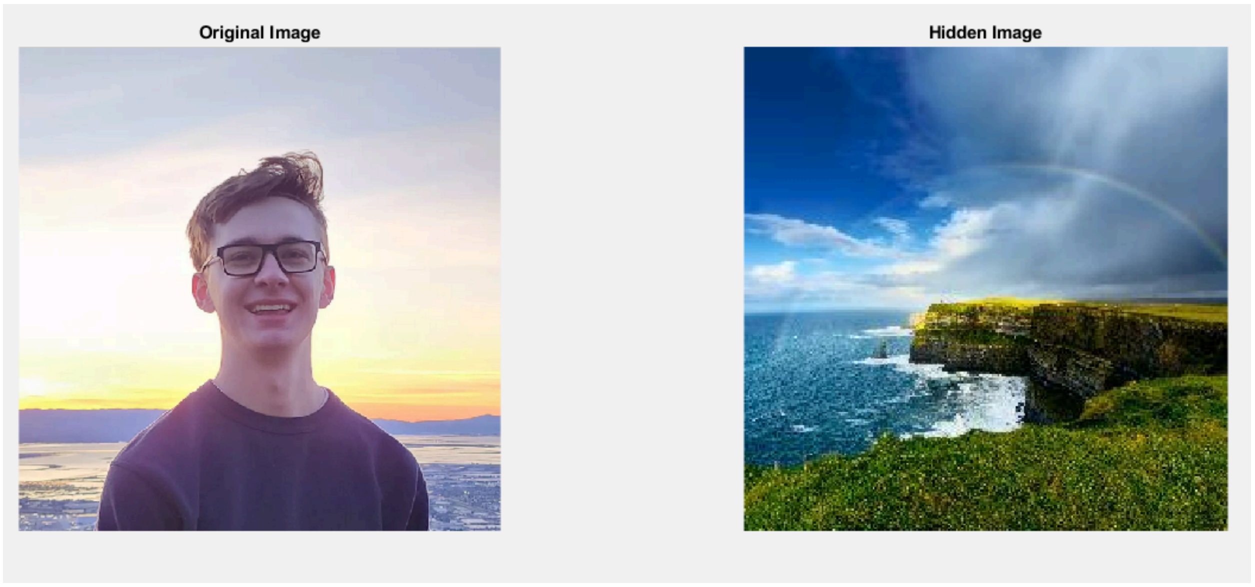

Figure 4. Original Caden with Hidden Image Cliffs

Figures 5, 6, and 7 show the resulting modified image which has a hidden image inside it, alongside the recovered Cliffs image. **Figure 5** displays that the image is not fully recoverable with N=1, but is when we double the amount of potential data encrypted with N=2 (**Figure 6**). After this point, any additional N bits hidden will lead to more corruption at the top of the image, as discussed before, and seen in **Figure 7.**

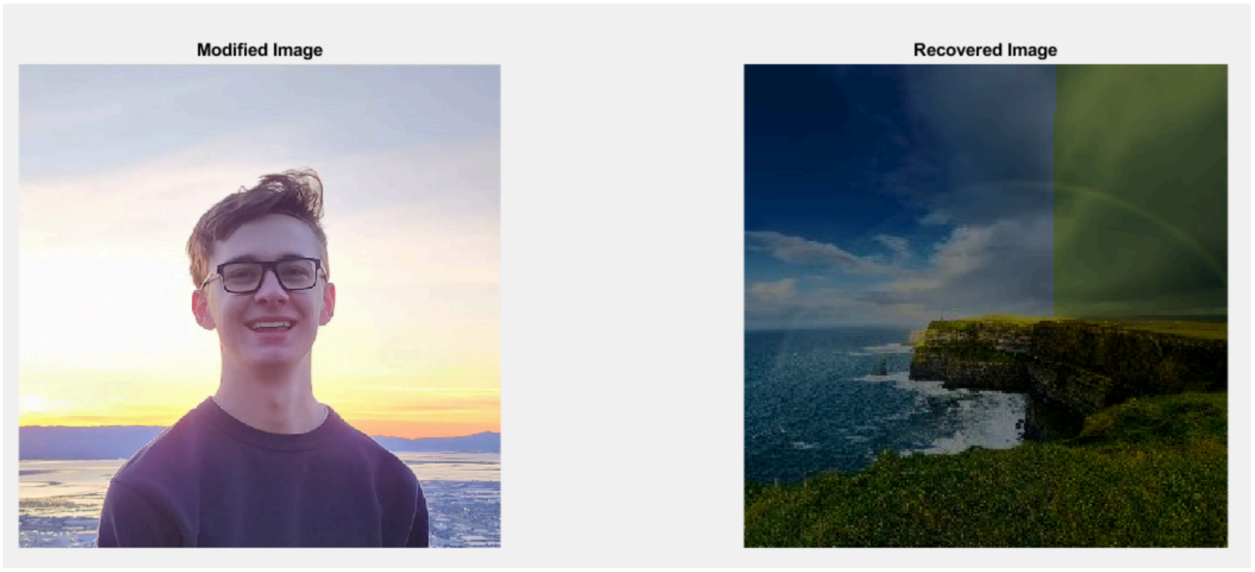

Figure 5. Cliffs Image Embedded into Caden, N=1

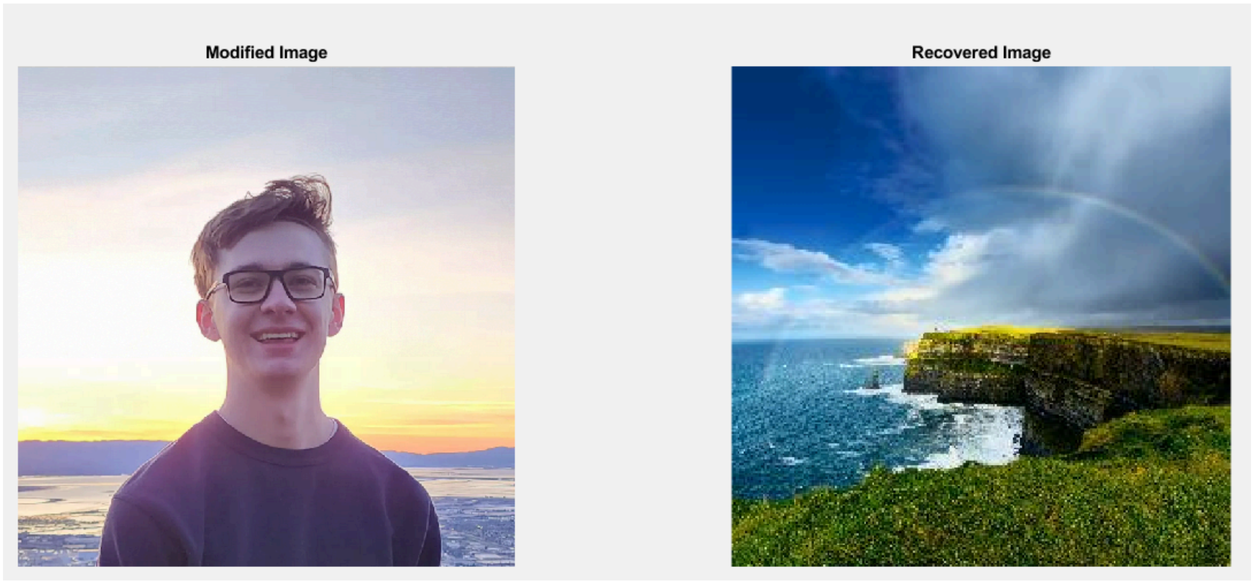

Figure 6. Cliffs Image Embedded into Caden, N=2

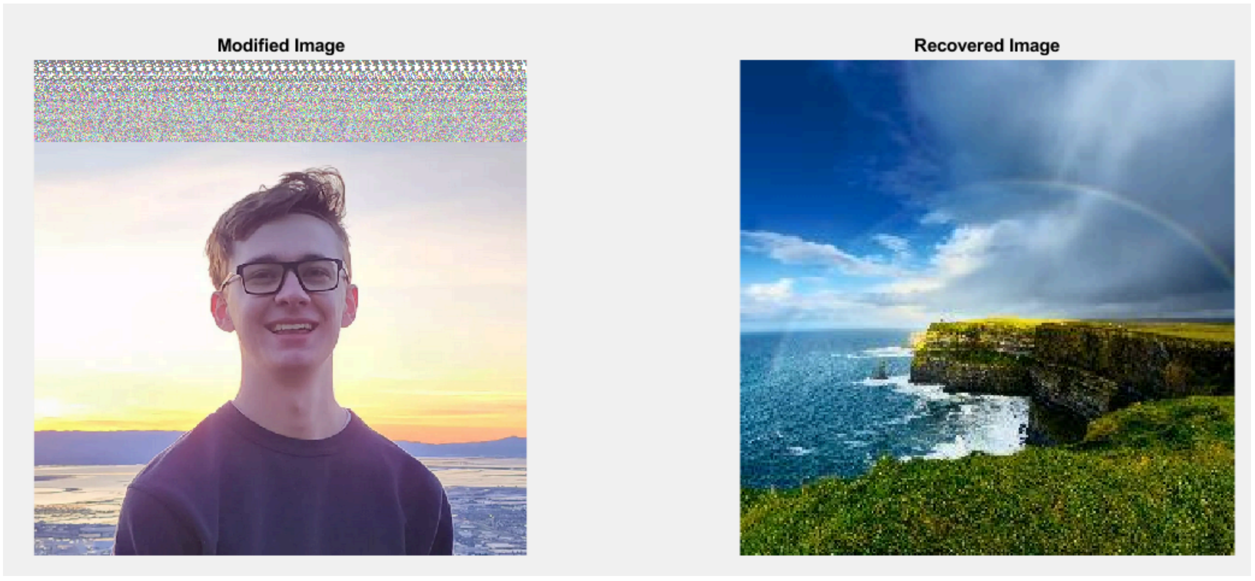

Figure 7. Cliffs Image Embedded into Caden, N=7

The structural similarity index can now be calculated for both the image hiding data AND the target image to be hidden. This is beneficial to compare the recovered image, since if the structural similarity index is not 1, then the image has not been fully recovered. **Table 3** backs up our inspection of the above figures, showing that N=2 is the smallest number of bits encrypted in which we can fully recover the image. As the N value increases above this, the structural similarity index decreases, and the resulting modified image is corrupted more than needed.

N Bits Hidden	1	2	3	4	5	6	7
SSIM Original Image	0.99782	0.98949	0.97253	0.93363	0.86951	0.83697	0.83531
SSIM Recovered Image	0.48562	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

Table 3. Structural Similarity Index for Cliffs Image into Caden

Next, we attempted to use a higher resolution image, which we called Landscape, as seen in **Figure 8**. As before, we used the same baseline image to hide data in our friend Caden.

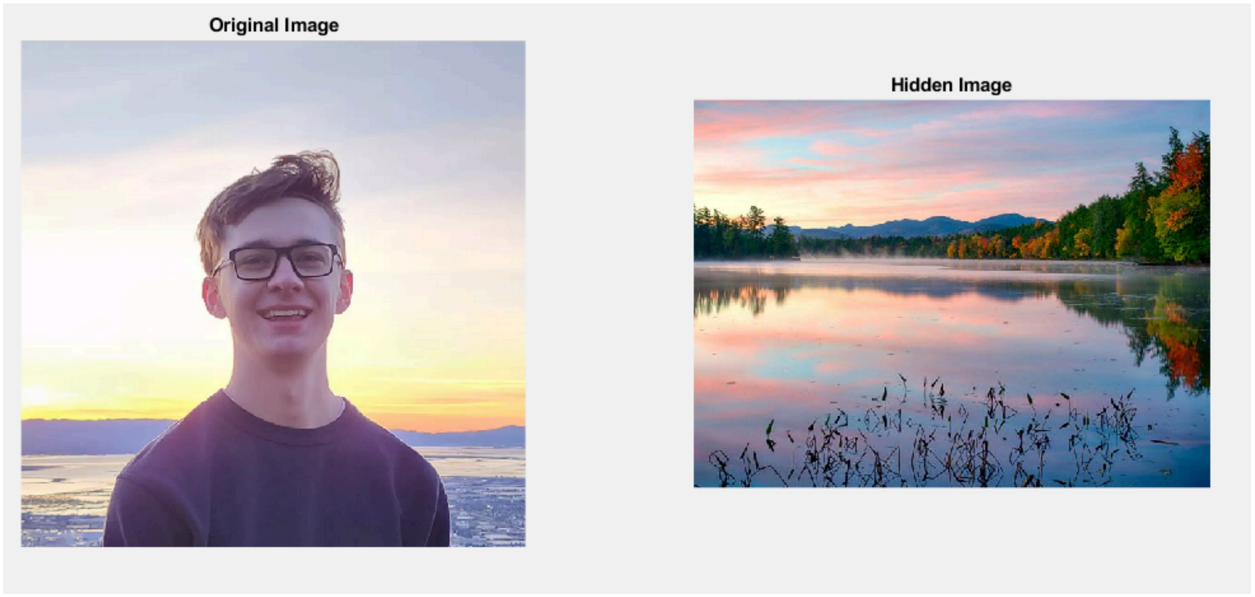

Figure 8. Original Caden with Hidden Landscape

Like before, this image is a high enough resolution that it cannot be hidden with N=1 bits hidden in every pixel, as seen in **Figure 9**. Since this image is larger in size than the cliffs image, it requires an N value of at least 6 bits hidden to fully recover the image, as seen in **Figure 11**. Between these stages, certain colors will be hidden, such with N=4 in which the overall image appears the same with a red tint in **Figure 10**. This is due to how we order hiding the pixels, in which a for loop will iterate and first hide all R, G, and then B values. In each modified image with hidden data, it can be seen that corruption becomes much more severe with higher resolutions and higher N values, as seen before.

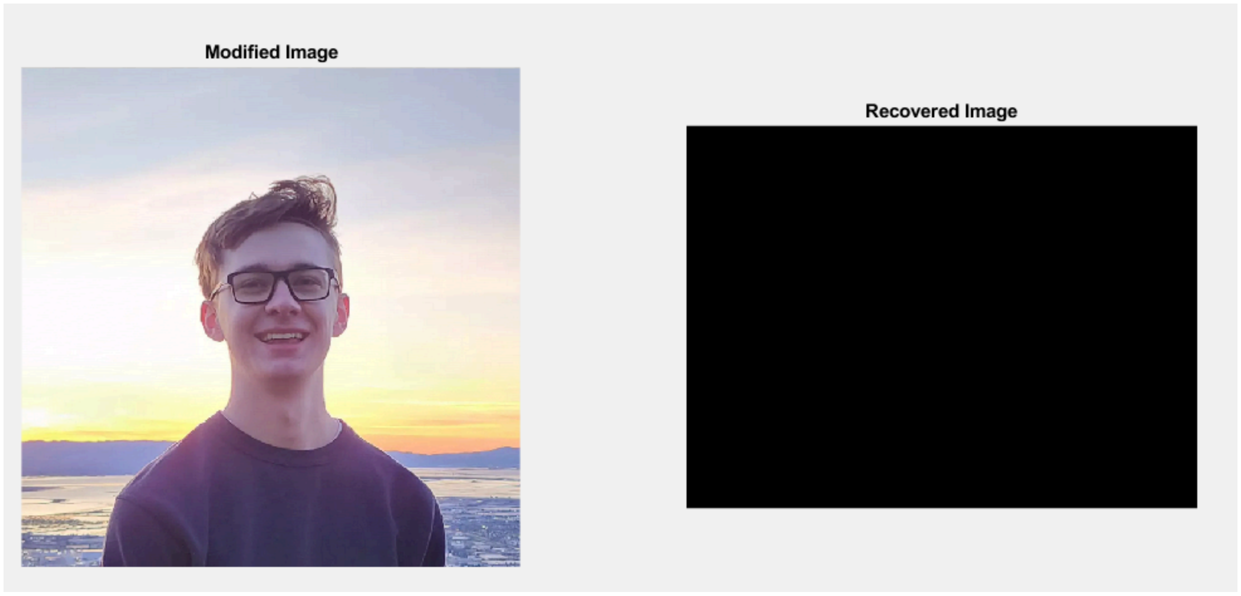

Figure 9. Landscape Image Embedded into Caden, N=1

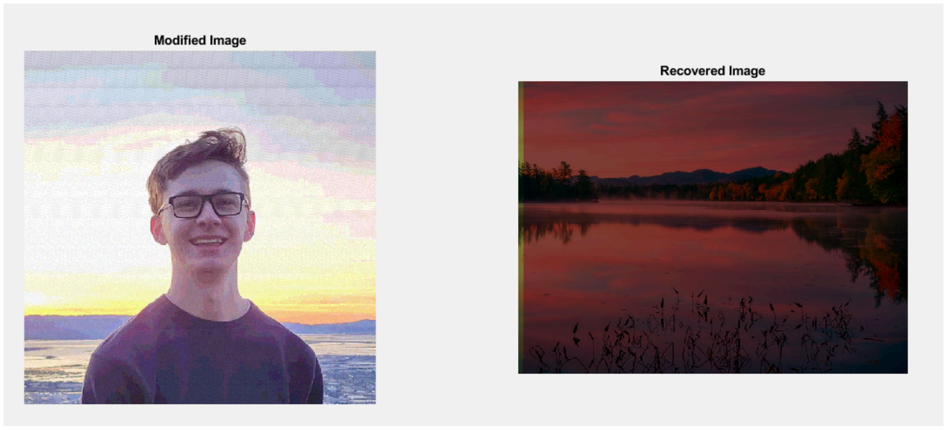

Figure 10. Landscape Image Embedded into Caden, N=4

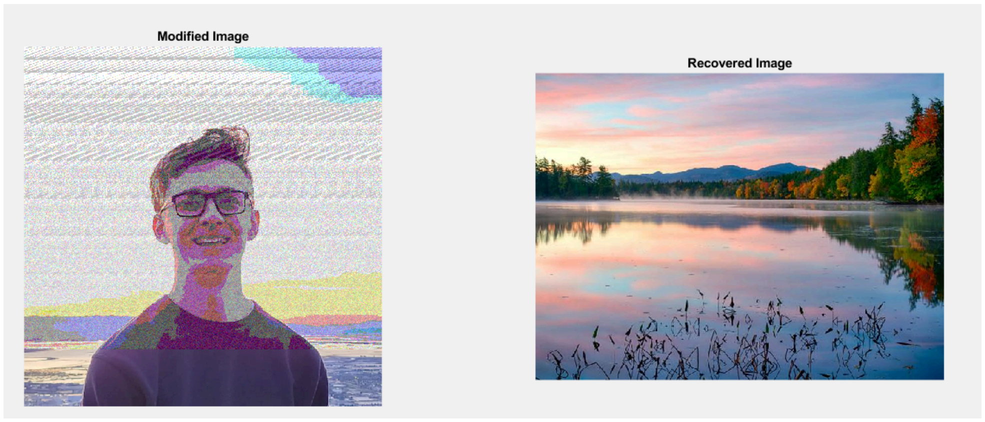

Figure 11. Landscape Image Embedded into Caden, N=6

The same structural similarity indices were recovered for the Landscape hidden image as the Cliffs image. The image can be fully recovered with at least N=6 bits of hidden data, which leads to a severe drop in similarity for the modified image, in which it is very obvious that the image is corrupted. This shows the limitations of this design with smaller images to hide data into.

N Bits Hidden	1	2	3	4	5	6	7
SSIM Original Image	0.99804	0.98622	0.94642	0.8089	0.54971	0.36106	0.31908
SSIM Recovered Image	0.00397	0.020648	0.061384	0.25223	0.86167	1.00000	1.00000

Table 4. Structural Similarity Index for Landscape Image into Caden

IV. Hiding Bitstream Within Images

Taking this whole process one step further, entire files of any kind could be placed within the LSB's of images using a similar process as the text files, by reading files out as a raw stream of binary and placing bits in one at a time. The major transition to this form of steganography was actually reading out files as binary within matlab, and generating "keys" for identifying how to pull files back out.

Generating a binary stream was a fairly straightforward process, involving opening a selected file and reading it off to an array using the data type "ubit1", which would read off a list of bits from the file. This would be as a list of integers, so some conversions would need to be run before the data would be ready to be placed into a file.

As for the "keys" spoken of, we needed standards for how a matlab script would know how to pull files back out. The set standard would be 3 keys total:

- An always 3 bit LSB key on the very first pixel that would determine the number of LSB's used in the rest of the image, so that the correct bits would be pulled out and reassembled. By using the first pixel and only 3 bits, there would be minimal quality loss, and on the very corner of the image, so it would be virtually invisible.
- An always 40 bit following stream that would convert to an integer representative of the number of bits stored in the image. This was a large enough set that it could successfully use files over 100 GB (much larger than the scope of this project) and would be used so that the program would know when to halt pulling bits
- A final key that would read off bits as 7 bit ASCII characters until a "?" character was read. This would reveal the file name for the data to be written as, and since a "?" cannot be used when writing a file name, it acts as a perfect mark as a stopping point to transition from reading the filename to reading data.

A separate script used to reveal the hidden file would implement these known keys and could then correctly extract any kind of file from any image, so long as the image contained a high enough pixel count.

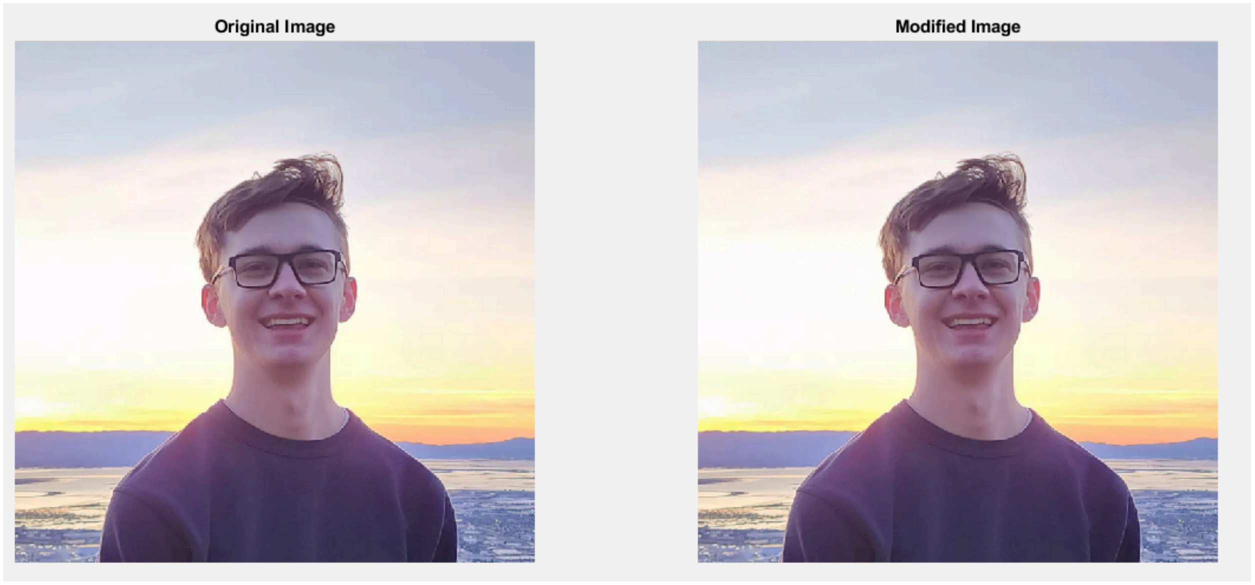

Figure 12. Bee Movie Embedded into Caden, N=1, ssim=0.99828

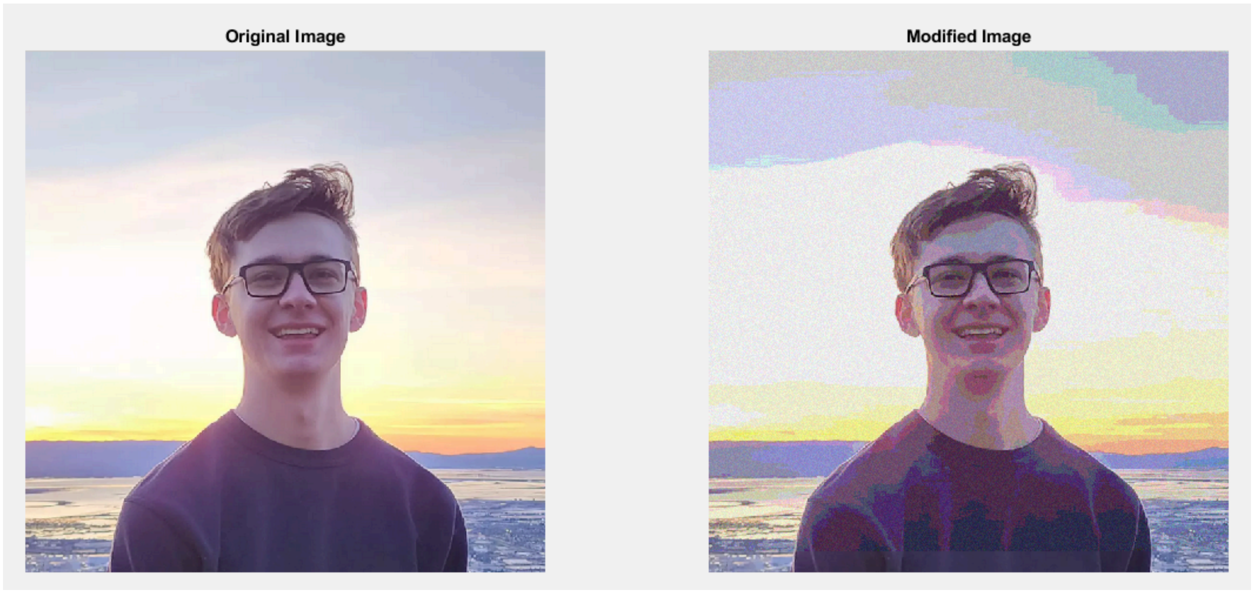

Figure 13. Bee Movie x20 Embedded into Caden, N=5, ssim=0.62883

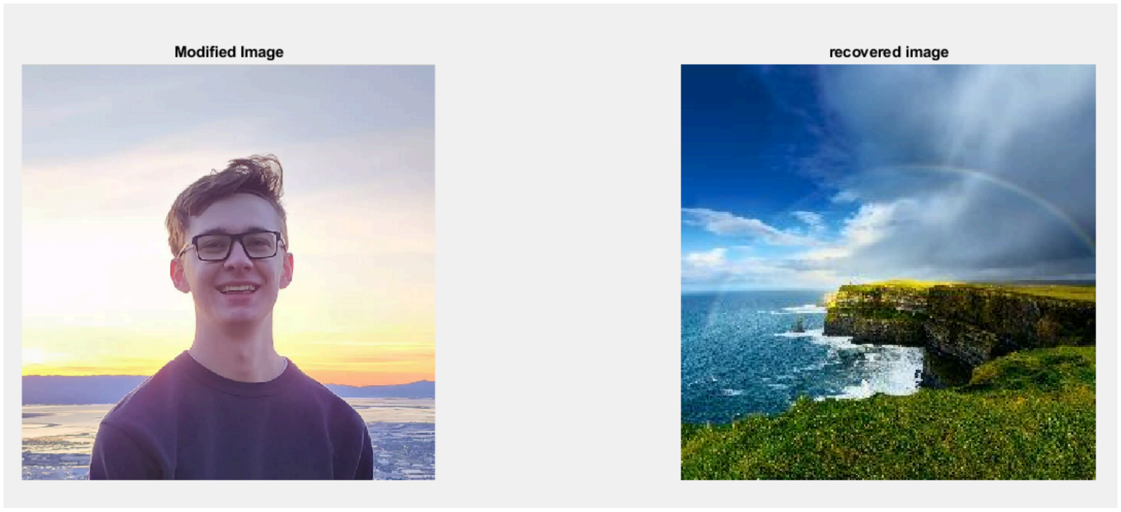

Figure 14. Landscape Embedded into Caden, N=1, ssim=0.99981

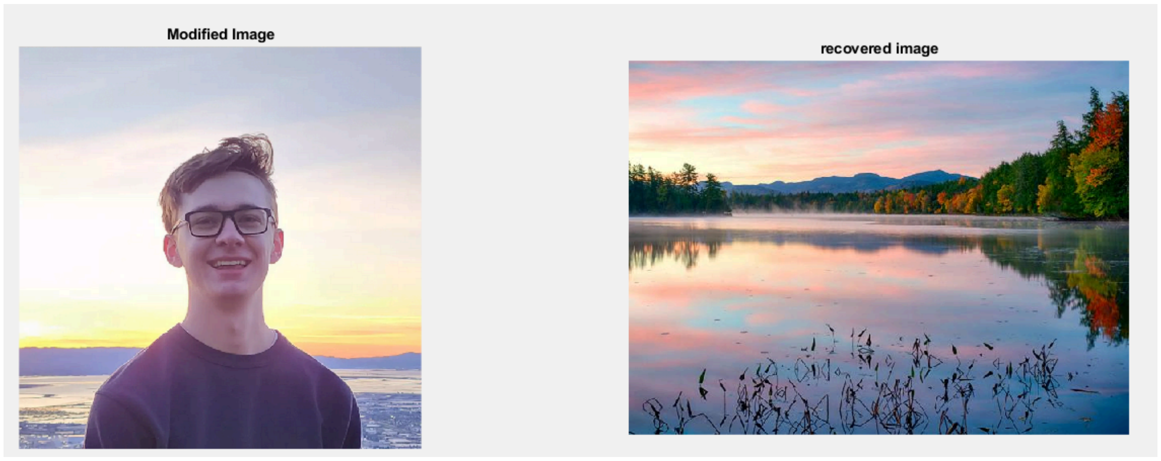

Figure 15. Landscape Embedded into Caden, N=1, ssim=0.99916

File	**Bee Movie**	**Bee Movie x9**	**Cliffs**	**Landscape**
Original N	1	5	2	6
Bitstream N	1	5	1	1
Original ssim	0.99855	0.65244	0.98949	0.36106
Bitstream ssim	0.99828	0.62883	0.99981	0.99916
SSIM Improvement %	99.973%	96.381%	101.042%	276.729%

Table 5. Bitstream Improvement

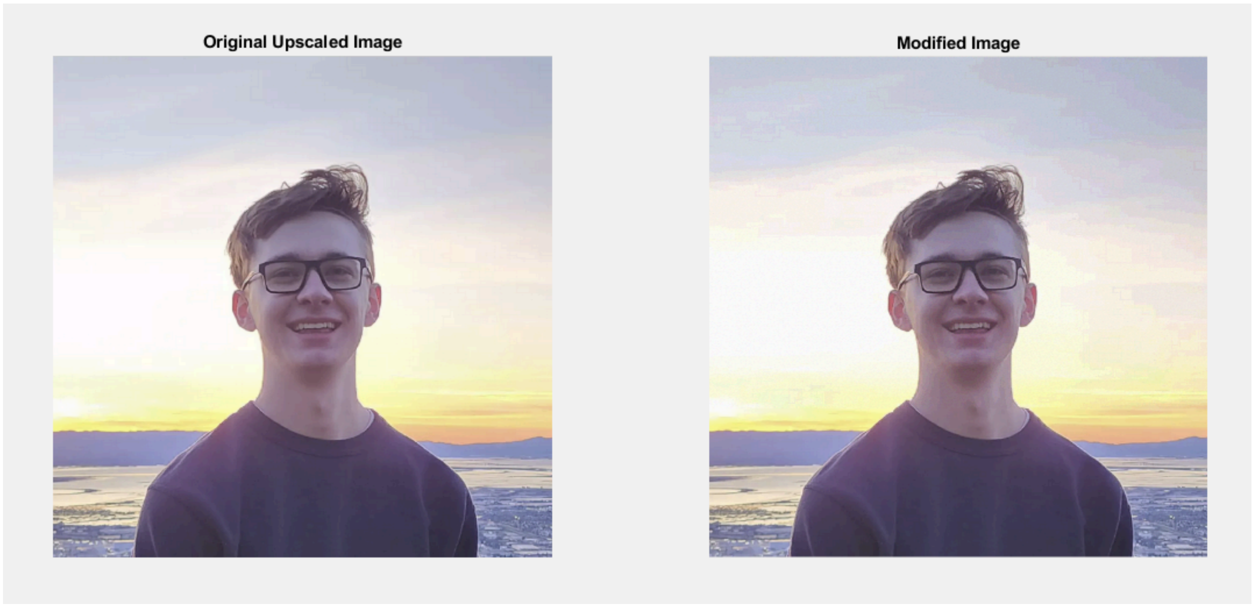

Figure 16. Bee movie film embedded in upscaled Caden, N=2, ssim=0.9621

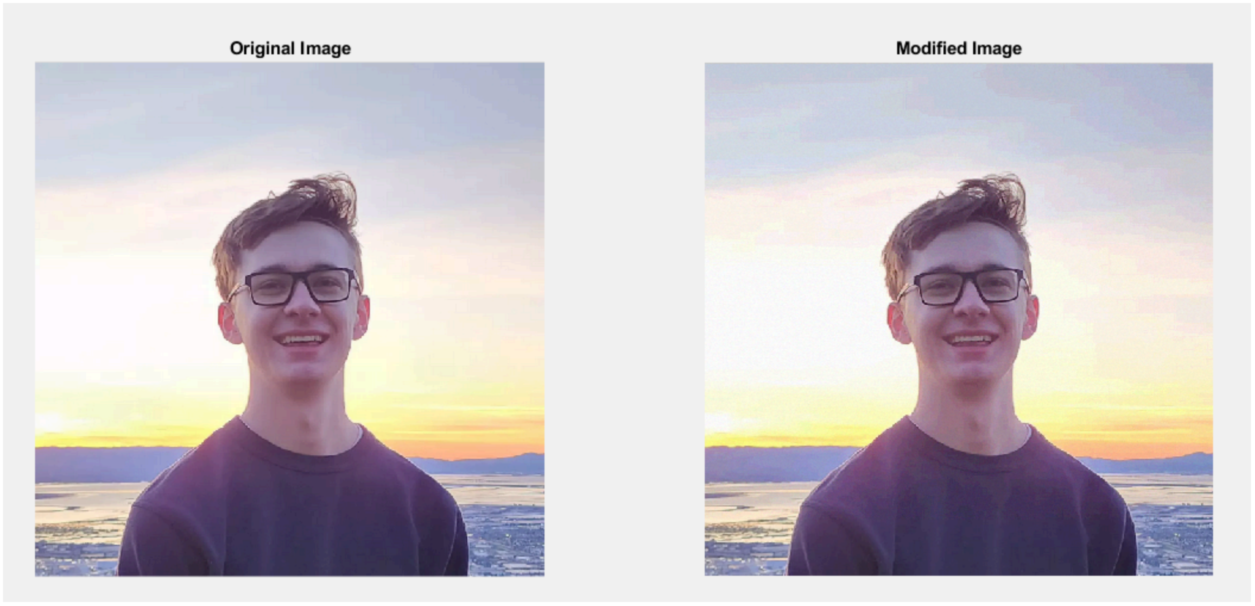

Figure 17. Caden image hidden within itself, N=3, ssim=0.95966

V. Conclusion

In conclusion, we have successfully designed a program to hide text, an image, and any form of media within another 2D image. We also have included code for error analysis by displaying each modified and recovered piece of media, as well as automatically determining the required number of bits to hide the encrypted media. With large resolution images, it is possible to hide large amounts of media due to Windows data compression, which is lossless.

VI. References

D. Neeta, K. Snehal and D. Jacobs, "Implementation of LSB Steganography and Its Evaluation for Various Bits," *2006 1st International Conference on Digital Information Management*, Bangalore, India, 2007, pp. 173-178, doi: 10.1109/ICDIM.2007.369349.

VII. Appendix - Code

Hide Text Code

```
N = 7; %Number of bits to clear/hide


% Load Original Image

original_filename = "caden.png";

original_image = imread(original_filename);


% Create Hidden Message

hidden_filename = "hello.txt";

FID = fopen(hidden_filename);

hidden_char = fscanf(FID,'%c');

fclose(FID);

hidden_double = double(hidden_char);

hidden_binary = dec2bin(hidden_double);

hidden_reshape_nofill = reshape(hidden_binary, [], 1);

num_pad = N - rem(length(hidden_reshape_nofill), N);

zero_append = char(zeros(num_pad, 1));

hidden_reshape = [hidden_reshape_nofill; zero_append];


% Hide hidden message in original image

int_clear = 2^8 - 2^N;

modified_image = original_image;

[x y z] = size(original_image);

message_index = 1;
```

```
for a = 1:x
        for b = 1:y
    for c = 1:z %every 3 is R, G, B
      if message_index > length(hidden_reshape) %FIXME, bad solution
        break;
      end
      modified_image(a, b, c) = bitand(original_image(a, b, c), int_clear);
      int_fill = (hidden_reshape(message_index : message_index + N - 1));
      int_fill_dec = bin2dec(int_fill.');
      message_index = message_index + N;
      modified_image(a, b, c) = bitor(modified_image(a, b, c), int_fill_dec);
    end
        end
end

% Display modified image with hidden message
figure
subplot(1,2,1);
imshow(original_image);
title("Original Image");
subplot(1,2,2);
imshow(modified_image);
title("Modified Image");
imwrite(modified_image, "BeeMovieCaden.png");

% Recover hidden message from modified image
message_index = 1;
recovered_binary(1:length(hidden_reshape)) = zeros();
int_recover = 2^N - 1;
```

```
for a = 1:x

        for b = 1:y

    for c = 1:z %every 3 is R, G, B

      if message_index > length(hidden_reshape) %FIXME, bad solution

        break;

      end

      recovered_int = bitand(modified_image(a, b, c), int_recover);

      binary_loop = dec2bin(recovered_int, N);

      recovered_binary(message_index : message_index + N - 1) = binary_loop;

      message_index = message_index + N;

    end

        end

end


% Convert recovered Message

reovered_binary_no_pad = recovered_binary(1:length(hidden_reshape) - num_pad);

recovered_reshape = reshape(reovered_binary_no_pad, [], 7);

recovered_char = char(recovered_reshape);

recovered_string = char(bin2dec(recovered_char));

recovered_string = convertCharsToStrings(recovered_string);

disp("Recovered message: " + recovered_string);


% Error Calculation

ssim_err = ssim(modified_image, original_image);

disp("ssim error: " + ssim_err); %Value closer to 1 represents better image quality
```

Hide Image Code (Modified from Hide Text)

```
% Create Hidden Image

hidden_filename = "caden.png";

hidden_image = imread(hidden_filename); %original uint8

[hidden_x hidden_y hidden_z] = size(hidden_image);


hidden_binary = dec2bin(hidden_image);

hidden_reshape_nofill = reshape(hidden_binary, [], 1);

num_pad = N - rem(length(hidden_reshape_nofill), N);

zero_append = char(zeros(num_pad, 1));

hidden_reshape = [hidden_reshape_nofill; zero_append];


% Convert recovered Message

reovered_binary_no_pad = recovered_binary(1:length(hidden_reshape) - num_pad);

recovered_reshape = reshape(reovered_binary_no_pad, [], 8);

recovered_char = char(recovered_reshape);

recovered_binary = bin2dec(recovered_char);

recovered_image = uint8(reshape(recovered_binary, hidden_x, hidden_y, hidden_z));


% Error Calculation

rec_err = ssim(recovered_image, hidden_image);

disp("ssim recovered image error: " + rec_err); %Value closer to 1 represents better image quality
```

Hide Bitstream Code (Modified from Hide Image)

```
% Load Original Image

original_filename = "caden.png";

original_image = imread(original_filename);

[x y z] = size(original_image);


% Create Hidden Message

hidden_filename = convertCharsToStrings(uigetfile({'*.*'}, 'File Selector'));

hidden_filename_2 = hidden_filename + '?';

hidden_filename_chars = convertStringsToChars(hidden_filename_2);


FID = fopen(hidden_filename);

filebits = fread(FID, '*ubit1', 'ieee-le');

fclose(FID);

bitlength = length(filebits);


bits = [zeros(43 + 7 * (length(hidden_filename_chars)), 1); filebits];

bytecount = x * y * z;

N = 1; %Number of bits to clear/hide

while bytecount * N < length(bits)

        N = N+1;

end

H = N;

if H >= 4

  bits(1) = 1;

        H = H - 4;

end

if H >= 2

  bits(2) = 1;
```

```
        H = H - 2;
end
if H >= 1
  bits(3) = 1;
end
total_length = length(bits);
mod_length = total_length;
for i = 1:40
        if mod_length >= 2^(40 - i)
    bits(3 + i) = 1;
    mod_length = mod_length - 2^(40-i);
        end
end
textvar = double(hidden_filename_chars);
textbin = [];
for i = 1:length(textvar)
        for k = 1:7
    textbin((i - 1) * 7 + (k)) = 0;
    if textvar(i) >= 2 ^ (7-k)
      textbin((i - 1) * 7 + (k)) = 1;
      textvar(i) = textvar(i) - 2 ^ (7-k);
    end
        end
end
bits(44:43 + 7 * length(hidden_filename_chars)) = textbin;


% for i = 1:bitlength/8
%     bytes(i) = 0;
%     for p = 1:8
%         bytes(i) = bytes(i) + bits((i-1)*8 + p) * 2 ^ (p-1);

%     end

% end

% fileID = fopen('bytes.png','w');

% fwrite(fileID, bytes);

% fclose(fileID);


hidden_double = double(bits);


hidden_binary = dec2bin(hidden_double);

hidden_reshape_nofill = reshape(hidden_binary, [], 1);


num_pad = N - rem(length(hidden_reshape_nofill), N);

zero_append = char(zeros(num_pad, 1));


hidden_reshape = [hidden_reshape_nofill; zero_append];


% Hide hidden message in original image


int_clear = 2^8 - 2^N;

N_clear = 2^8 - 2^3;

modified_image = original_image;


message_index = 1;


first = 0;

for a = 1:x

        for b = 1:y

    for c = 1:z %every 3 is R, G, B
```

```
if first == 0;

   if message_index > length(hidden_reshape) %FIXME, bad solution
               break;
   end

   modified_image(a, b, c) = bitand(original_image(a, b, c), N_clear);

   int_fill = (hidden_reshape(message_index : message_index + 3 - 1));
   int_fill_dec = bin2dec(int_fill.');

   message_index = message_index + 3;

   modified_image(a, b, c) = bitor(modified_image(a, b, c), int_fill_dec);
   first = 1;
else
   if message_index >= length(bits) %FIXME, bad solution
               break;
   end
   while message_index + N > length(bits)
               N = N -1;
   end
   modified_image(a, b, c) = bitand(original_image(a, b, c), int_clear);

   int_fill = (hidden_reshape(message_index: message_index + N - 1));
   int_fill_dec = bin2dec(int_fill.');

   message_index = message_index + N;
```

```
                modified_image(a, b, c) = bitor(modified_image(a, b, c), int_fill_dec);

            end

        end

                end
    disp(message_index/bitlength)
end

% Display modified image with hidden message

figure
subplot(1,2,1);
imshow(original_image);
title("Original Image");

subplot(1,2,2);
imshow(modified_image);
title("Modified Image");

imwrite(modified_image, "ModifiedImage.png");

ssim_err = ssim(modified_image, original_image);

disp("ssim error: " + ssim_err); %Value closer to 1 represents better image quality
disp("N: " + N);
```

Reveal Bitstream Code (Modified from Hide Image)

```
% Recover hidden message from modified image

modified_image = imread("ModifiedImage.png");

[x y z] = size(modified_image);

message_index = 1;

recovered_binary = [];


int_recover = 2^3 - 1;

bitlength = 0;

name_discovered = 0;

charbit = 0;

filename = [];

name_ready = 0;

charvar = 0;

first = 0;

for a = 1:x

        for b = 1:y

    for c = 1:z %every 3 is R, G, B

      if first == 0;


        recovered_int = bitand(modified_image(a, b, c), int_recover);


        binary_loop1 = dec2bin(recovered_int, 3);


        recovered_binary(message_index : message_index + 3 - 1) = binary_loop1;


        message_index = message_index + 3;

        N = (recovered_binary(1)-48) * 4 + (recovered_binary(2)-48) * 2 + (recovered_binary(3)-48);

        int_recover = 2^N - 1;
        first = 1;

      elseif message_index < 44

        recovered_int = bitand(modified_image(a, b, c), int_recover);

        binary_loop = dec2bin(recovered_int, N);

        recovered_binary(message_index : message_index + N - 1) = binary_loop;

        message_index = message_index + N;
      end
      if message_index >= 44 && name_ready == 0;
        for i = 0:39
                    bitlength = bitlength + (recovered_binary(43 - i) - 48) * 2 ^ (i);
        end
      end
      if message_index > 43 && name_ready == 1 && name_discovered == 0;

        if message_index > bitlength %FIXME, bad solution
                    break;
        end

        recovered_int = bitand(modified_image(a, b, c), int_recover);

        binary_loop = dec2bin(recovered_int, N);

        recovered_binary(message_index : message_index + N - 1) = binary_loop;
        charbit = charbit + N;

        if charbit >= 7

                charvar = 0;

                charbit = charbit - 7;

                for i = 0:6

                charvar = charvar + (recovered_binary(44 + 7 * length(filename) + i) - 48) * 2 ^ (6-i);

                end

                filename = [filename charvar];

        end


        message_index = message_index + N;

      end

      if name_discovered == 1


        if message_index > bitlength && name_discovered == 1; %FIXME, bad solution

                break;

        end


        recovered_int = bitand(modified_image(a, b, c), int_recover);


        binary_loop = dec2bin(recovered_int, N);


        recovered_binary(message_index : message_index + N - 1) = binary_loop;


        message_index = message_index + N;

      end

      if message_index > 43

        name_ready = 1;

      end
```

```
            if charvar == 63
                name_discovered = 1;
            end
            if message_index > bitlength && name_discovered == 1; %FIXME, bad solution
                break;
            end
        end
        if message_index > bitlength && name_discovered == 1; %FIXME, bad solution
            break;
        end
                end
end


% Convert recovered Message
databitlength = bitlength - (43+7*length(filename));
bits = recovered_binary(43+7*length(filename):bitlength);
for i = 1:databitlength/8
    bytes(i) = 0;
                for p = 1:8
        bytes(i) = bytes(i) + (bits((i-1)*8+p + 1) - 48) * 2 ^ (p-1);
                end
end
filestring = ("recovered " + convertCharsToStrings(char(filename(1:length(filename)-1))));
fileID = fopen(filestring,'w');
fwrite(fileID, bytes);
fclose(fileID);
```