Assignment # 2, Part 1

This assignment is due by 11:59 p.m. on Friday, Feb 19. All reports should be submitted as PDFs. Only one report must be submitted per team, but all team member's names must appear on the report.

Part 1: Least Significant Bit Data Hiding

In class, we discussed a method to embed a hidden signal in a digital image known as least significant bit (LSB) data hiding. This method operates by replacing a bit plane (typically the least significant) in an image with a hidden binary message. Often, this hidden message is a binary representation of another image or contains text.

- Write a Matlab function that extracts and displays a user specified bit plane from an image. Use this function to examine each of the bit planes in the **peppers.tif** and **baboon.tif** images. As you examine progressively lower bit planes, you should notice that each bit plane increasingly resembles noise.

  What is the highest bit plane for each image that no longer resembles image content and instead appears to be noise? Are these bit planes the same or different for these two images? If they are different, why would this be the case?

- Use your Matlab function from the previous part to examine the different bit planes of the following images: **LSBwmk1.tiff**, **LSBwmk2.tiff**, **LSBwmk3.tiff**. Do any of these images contain hidden content? If any of these images contain hidden information, please make note of the image and include the hidden content in your report.

- Write a Matlab function that replaces the N least significant bit planes from one image with the N most significant bit planes from another image. This function should accept both images along with the parameter N as inputs and return the modified image containing hidden data in unsigned 8-bit integer format as an output. Please comment your code and append it to your report.

  Use this function to hide the top N layers of the **Barbara.bmp** image in both the **peppers.tif** and **baboon.tif** images. How many bit planes of **Barbara.bmp** can you embed in each image before you start to notice distortion in the host image? How many bit planes of **Barbara.bmp** can you embed in each image before you can begin to see the hidden content? Are the number of bit planes that you can embed the same or different for each image? If they are different, why would this be the case?

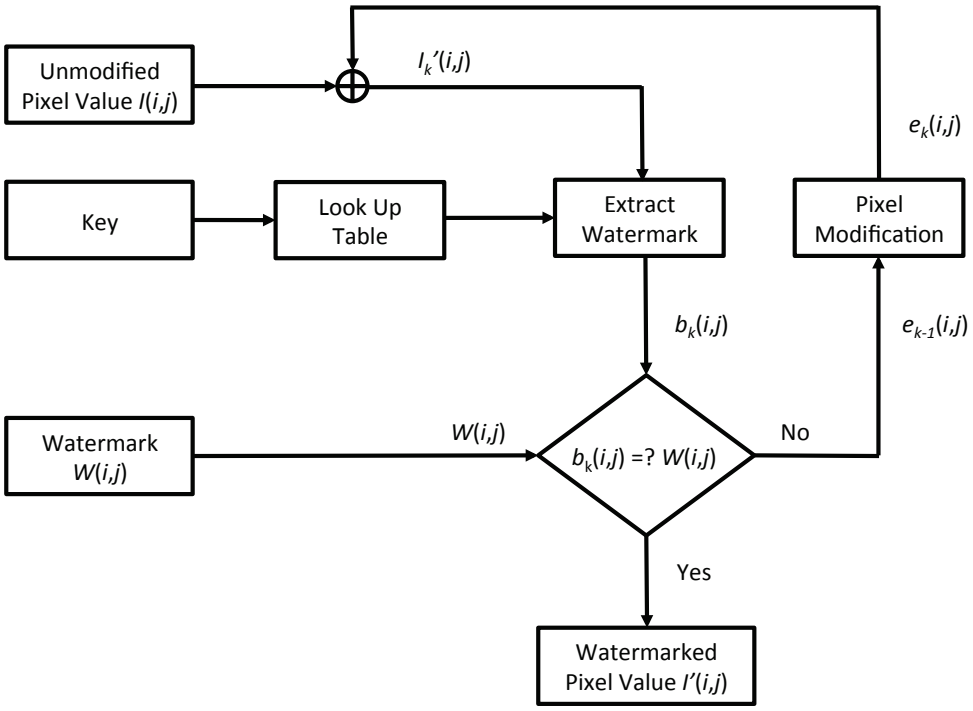
Part 2: Yeung–Mintzer Watermarking

While the LSB data hiding technique can be used to embed a fragile watermark in a digital image, it has several weaknesses. For example, data hidden using LSB can often be detected through visual inspection of each bit plane.

To address these weaknesses, Yeung and Mintzer proposed an improved fragile watermarking scheme that uses a look up table to embed a watermark in a digital image. A flowchart of the Yeung-Mintzer watermarking algorithm is shown in the figure below.

- Write a Matlab function that uses the Yeung–Mintzer altorithm to embed a binary watermark in an image. This function should accept both the image to be watermarked, a binary watermark (with the same number of rows and columns as the image), along with a key to be used to generate the look up table as inputs and return the modified image containing the watermark in unsigned 8-bit integer format as an output. Please comment your code and append it to your report.

  To generate the look up table, use the *rand* command in Matlab to generate an array of 256 uniformly distributed random numbers, then compare them to a threshold of 0.5. This will create an array of 256



uniformly distributed 1's and 0's. The first entry in this array will be the extracted watermark bit corresponding to the pixel value '0', the second entry will be the extracted watermark bit corresponding to the pixel value '1', and so on.

You should seed Matlab's pseudorandom number generator using the watermark key using the `rng` command. This will ensure that the same look up table is generated every time the same key is used.

Your code to generate the look up table should resemble the code below:

```
% seed the random number generator with the user specified key
rng(key)
% generate look up table values
LUTvals= rand(1,256) > .5;
```

Note: You do not need to generate the error diffusion portion of the Yeung–Mintzer algorithm described in their papers.

- Use your Yeung–Mintzer embedding function to hide the most significant bit plane of the **Barbara.bmp** image in both the **peppers.tif** and **baboon.tif** images (i.e. use the most significant bit plane of **Barbara.bmp** as your watermark).

  Use the function that you wrote in Part 1 to examine the lower bit planes of each of these images. Is the hidden watermark visually detectable? Include an image of the least significant bit plane of each watermarked image in your report. What is the PSNR between the original version of each image and their Yeung-Mintzer watermarked versions.

  Now use the LSB embedding function that you wrote in Part 1 to hide the most significant bit plane of the **Barbara.bmp** image in both the **peppers.tif** and **baboon.tif** images. What is the PSNR between the original version of each image and their LSB watermarked versions. Are these PSNR values higher or lower than the PSNR values obtained for the versions watermarked using the Yeung-Mintzer algorithm? Why (please explain which algorithm you would expect to introduce greater distortion into an image)?

- Write a Matlab function that extracts a watermark from an image that has been watermarked using the Yeung–Mintzer algorithm to embed a binary watermark in an image. This function should accept both the watermarked image to be watermarked and the key used to generate the look up table as inputs. It should return the watermark as an output. Please comment your code and append it to your report.

  Use this function to extract the watermark embedded in the image **YMwmkedKey435.tiff**. When extracting the watermark, use the number '435' as the key to generate the look up table. Include a picture of the extracted watermark in your report.

- One of the weaknesses of fragile watermarking using the LSB embedding technique is its susceptibility to attacks. Replace the top half of the watermarked versions (both LSB and Yeung–Mintzer) of the **baboon.tif** image with the bottom half of the **peppers.tif** image. Extract the watermark from each of these images and include them in your report. Now try to design an attack where you replace the top half of the **baboon.tif** image with the bottom half of the peppers image without destroying the watermark. How can you do this for the LSB watermarked image? Is this possible for the Yeung-Mintzer watermarked image (assuming the attacker does not know the key or look up table)?

Part 3: Spread Spectrum Watermarking

In Part 1 of this assignment, we examined the LSB and Yeung–Mintzer fragile watermarking schemes. These watermarks are easily destroyed if the watermarked image undergoes even minor processing. In some cases, a content owner may wish to embed a robust watermark image that will be difficult to destroy.

One very successful robust watermarking technique discussed in class is the spread spectrum watermarking algorithm developed by Cox et al. This algorithm embeds a watermark in the discrete cosine transform (DCT) coefficients of a digital image according to the following embedding procedure:

1. Compute the two dimensional discrete cosine transform (DCT) of the entire image.
2. Identify the N largest DCT coefficients excluding the DC coefficient. These coefficients will be used for embedding.
3. Create the watermark w as a sequence of N random numbers independently drawn from a Gaussian distribution with zero mean and unit variance.
4. Embed the watermark into the N largest DCT coefficients according to the equation

$$v_i' = v_i + \alpha w_i v_i = v_i(1 + \alpha w_i) \tag{1}$$

   where v_i is the i^th largest DCT coefficient of the unmodified image, where v_i' is the i^th largest DCT coefficient of the watermarked image, w_i is the i^th watermark value, and α is the embedding strength. The remaining DCT coefficients are not modified.
5. Compute the inverse DCT to obtain the watermarked image.

To recover the watermark from an image, the content owner must retain an original copy of the unwatermarked image. The watermark is extracted from an image using the following procedure:

1. Compute the two dimensional DCT of the watermarked image and the reference image.
2. Identify the N largest DCT coefficients of the reference image, excluding the DC coefficient.
3. Recover the watermark $\hat{w}$ according to the equation:

$$\hat{w_i} = \frac{1}{\alpha}\left(\frac{v_i'}{v_i} - 1\right) \tag{2}$$

where v_i is the i^th largest DCT coefficient of the reference image, where v_i' is the i^th largest DCT coefficient of the watermarked image, $\hat{w}_i$ is the i^th extracted watermark value, and α is the embedding strength.

The extracted watermark $\hat{w}$ may not exactly match the embedded watermark $\hat{w}$ due to image processing or attacks against the watermarking algorithm. Nonetheless, the embedded watermark can still be identified with a high level of confidence. The watermark $\hat{w}$ extracted from an image can be compared to the watermark w of an individual user using the similarity $\text{sim}(\hat{w}, w)$ measure defined as:

$$\begin{aligned} \text{sim}(\hat{w}, w) &= \frac{\hat{w}^T w}{\hat{w}^T \hat{w}}, \\ &= \sum_{i=1}^{N} \frac{\hat{w}_i w_i}{\hat{w}_i^2}. \end{aligned} \tag{3}$$

Note: This formulation of the similarity metric assumes that the both the embedded and extracted watermarks are column vectors.

- Write a Matlab function that embeds a spread spectrum watermark in an image. This function should accept both the image to be watermarked, a key to generate the watermark, the watermark length N, and the embedding strength α as inputs and return the watermarked image in unsigned 8-bit integer format as an output.

  When generating the watermark, the key should be used as a seed to Matlab's Gaussian pseudorandom number generator *randn*. Your code should resemble the following set of commands:

  ```
  % seed the random number generator with the user specified key
  rng(key)
  % generate i.i.d.  Gaussian watermark with mean = 0 and variance = 1
  wmk= randn(N,1);
  ```

  Please fully comment your code and append it to your report.

- Write a Matlab function that extracts a spread spectrum watermark from an image and measures the similarity between the extracted watermark and a reference watermark. This function should accept as inputs both the watermarked image, an unwatermarked version of the watermarked image, the watermark length N, the embedding strength α, and a key to generate the reference watermark. It should output the similarity score between the watermark extracted from the watermarked image and the reference watermark corresponding to the user specified key. Please fully comment your code and append it to your report.

- Use your embedding function to embed a spread spectrum watermark in the **baboon.tif** image. Set your watermark length to be $N = 1000$ and your embedding strength to be $\alpha = 0.1$. You may choose any integer as the key to generate the watermark (make note of the watermark key you choose in your report). What is the PSNR between the watermarked image and the unwatermarked version of **baboon.tif**? Now extract the watermark from the image. What is the similarity measure between the extracted watermark and the embedded watermark? Typically, for a Gaussian watermark of length $N = 1000$, the watermark is detected if the similarity measure is greater than 7. Is the similarity measure you obtained greater than this detection threshold? Now try measuring the similarity between the extracted watermark and 5 other watermarks chosen at random (i.e. randomly choose 5 different watermark keys that do not match your embedding watermark key). What are the similarity measures you obtain? Are these greater than the detection threshold?

- Now examine the robustness of the spread spectrum watermarking to several different signal processing operations and attacks. Modify your watermarked version of the **baboon.tif** using the following operations and attacks:

- Use Matlab's *imwrite* command to JPEG compress your watermarked image with quality factors of 90, 75, and 50.
- Smooth the image using a 3×3 averaging filter and median filter.
- Use Matlab's *imnoise* command to add Gaussian noise with zero mean and variances of $\sigma^2 = 1, 5, \text{and} 10$ to the image.
- Remove the bottom half of the watermarked image and replace it with the bottom half of the original (unwatermarked) image.

What is the similarity measure between the extracted and embedded watermark for each of these modified images? What is the similarity measure between the extracted watermark and each of the 5 randomly chosen watermarks? Are you able to detect the correct watermark for each of these modified images? Do any of the randomly chosen watermarks yield false matches (i.e. similarity measures > 7)?

- The image **streamSSWmked.tiff** has been embedded with a watermark whose key is an integer somewhere between 1 and 100. Determine the key of the watermark that was embedded into this image. Do this by writing a Matlab script that measures the similarity between the watermark in **streamSSWmked.tiff** and the watermark corresponding to each key between 1 and 100. Use **stream.tiff** as an unwatermarked version of the image when extracting the watermark.

  Hint: It is likely easiest to write a loop that cycles through each key and runs the watermark extraction and similarity measure function that you wrote earlier to determine the corresponding similarity measure.