DIP Project Report

Group Members:

Name	Roll No
Rahul Agarwal	19ucs264
Akshat Chhaparwal	19ucs211
Mayank Raj	19ucs151
Naresh Kumawat	19ucs046

Image Steganography

Objective:

The main objective of our project is to hide the data in the image using LSB based data embedding at the sender side and to decode the data at the receiver side.

Description:

Image Steganography is the method of hiding secret data inside an image. Images are made up of pixels which usually refer to the color of that particular pixel. In a greyscale (black and white) image, these pixel values range from **0-255**, 0 being black and 255 being white.
LSB stands for Least Significant Bit. The idea behind LSB embedding is that if we change the last bit value of a pixel, there won't be much visible change in the color. For example, 0 is black. Changing the value to 1 won't make much of a difference since it is still black, just a lighter shade.

Text Hiding and Extraction

Algorithm for hiding Text:

1. Read the input image
2. Convert the image to a grayscale image.
3. Resize the image if needed.
4. Add character ETX (end of text) at the end of the message.
5. Get all the ASCII values of the characters of the message
6. Convert the secret message to its binary format.
7. Initialize output image same as the input image.
8. Traverse through each pixel of the image and do the following:
    a. Convert the pixel value to binary.
    b. Get the next bit of the message to be embedded.
    c. Create a variable temp.
    d. If the message bit and the LSB of the pixel are the same, set temp = 0.
    e. If the message bit and the LSB of the pixel are different, set temp = 1.
    f. This setting of temp can be done by taking the XOR of the message bit and the LSB of the pixel.
    g. Update the pixel of the output image to input image pixel value + temp.
9. Keep updating the output image till all the bits in the message are embedded.
10. Finally, write the input as well as the output image to the local system.

Example:

Input: message='This is a secret message'

Processing:

Original Image

Grayscale Image

Grayscale Image

'This is a secret message'

Stego Image

Output: Image with the given message embedded

Comparison of images:

Original Image

Stego Image

As we can see in the above images, the input and the output image look exactly the same to the human eye. The output image has the message embedded in it.

Algorithm for extracting Text:

1. Get the input image.
2. Traverse through the image, one pixel at a time.
3. Store the Least Significant Bit (LSB) of each pixel in an array *extracted_bits*.
4. Convert the extracted bits to characters.
5. Remove text after ETX character
6. Print the hidden text.

Example:

Input:

Processing:

Stego Image

→ **'This is a secret message'**

Output: message='This is a secret message'

Image Hiding and Extraction

Algorithm for hiding an image:

1. Get the original image and secret image.
2. Resize the original image such that it is equal to the secret image.
3. Hide the secret image inside the original image
    a. Take bitwise AND of each pixel in the original image with -16. So that last 4 bit of each pixel in the image is replaced with four 0's.
    b. Shift the bits in the hidden image to the right by 4 places. So that ending 4 bits are replaced by starting 4 bits and starting 4 bits becomes 0.
    c. Do bit-wise OR of the result obtained in (a) and (b). So that last 4 bits in (a) which were 0 becomes the starting 4 bit of the hidden image.
4. Get the resulting steganographic image.

$f_1(x,y)$ = 170 (10101010)
Original Image

bit-wise AND
(170,-16)

10101010
11110000
=
10100000

$f_2(x,y)$ = 240 (11110000)
Secret Image

bit-wise RShift
(240,4)

00001111

g(x,y)
Steganographic Image

bit-wise OR
($f_1(x,y)$,$f_2(x,y)$)

10100000
00001111
=
10101111

Original Image **Secret Image** **Steganographic image**

Algorithm for extracting image:

1. Get the steganographic image
2. Shift the bits in the steganographic image by 4 places to the left so that the new image becomes the secret image with the last 4 bits as 0
3. Get the resulting secret image. (This image will have less intensity than the original secret image)

g(x,y) = 10101111
Steganographic Image

bit-wise LShift
(g(x,y),4)

11110000
$f_2'(x,y)$
Secret Image

Steganographic image

Extracted Image

Enhancing the extracted image

Algorithm for sharpening the image using unsharp masking:

1. Blur the original image
2. Subtract the blurred image from the original
3. Add the mask to the original

In Matlab, image sharpening using unsharp masking can directly be done using the **imsharpen(img)** function.

Extracted Image

Enhanced Image

Preprocessing of Cover Image

If we look closely at our new image of Mario, we can see a very light circle around him. So, we have to preprocess our cover image so that it contains a variety of colors rather than an image with a lot of white space.

1. **Isolate the background**
2. **Modifying the background**

Algorithm for isolating the background:

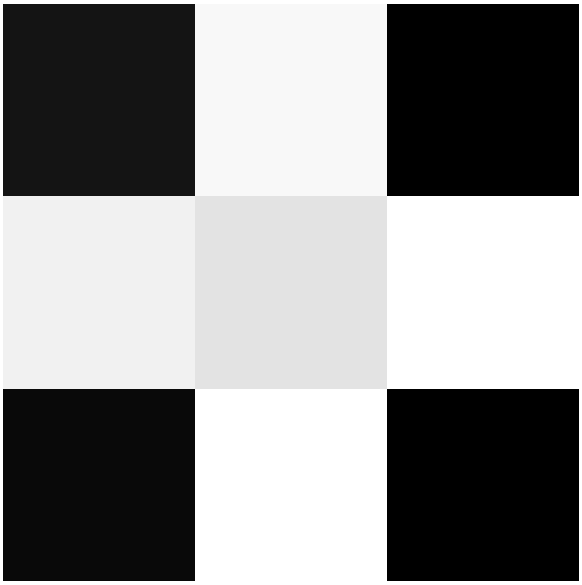
1. Read the cover image
2. Convert the image to the grayscale image
3. Take a disk-shaped structuring element with radius 1 and 4 line structuring elements which is used to approximate the disk shape
4. Dilate the grayscale image obtained in step 2 with the above structuring element
5. Erode the grayscale image with the structuring element
6. Subtract each element in the image obtained in step 5 from the corresponding element in the image obtained in step 4
7. Convert the image obtained in step 6 to binary image, by replacing all pixels in the input image with luminance greater than 0.03 with the value 1 (white) and replacing all other pixels with the value 0 (black).
8. Take a new disk-shaped structuring element with radius 3 and 4 line structuring elements
9. Perform morphological closing on the mask obtained in step 7 with the above structuring element. The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations.
10. Fill holes in the above binary image
11. Extract the largest connected components (objects) from the above binary image i.e. filter image, retaining only 1 object with the largest area.
12. Take logical NOT of above mask
13. Extract all connected components (objects) from binary image obtained above whose value of area is in the range [-Inf, 5000 - eps(5000)]
14. Take the logical OR of the image obtained in step 11 and step 13.

Example:

img = imread('sample.jpg');

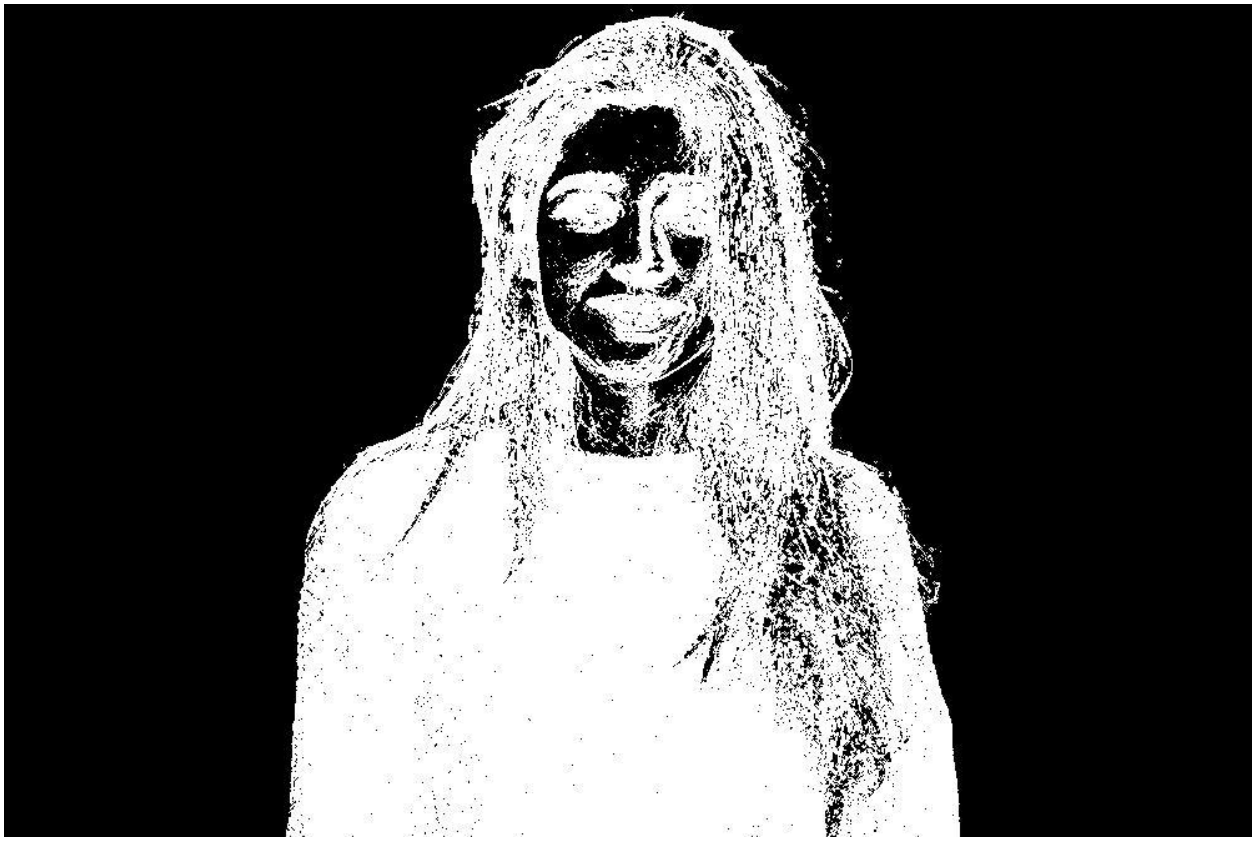
gray = rgb2gray(img);

SE = strel('Disk',1,4);



morphologicalGradient = imsubtract(imdilate(gray, SE),imerode(gray, SE));

mask = im2bw(morphologicalGradient,0.03); SE = strel('Disk',3,4);



mask = imclose(mask, SE);

mask = imfill(mask,'holes');

mask = bwareafilt(mask,1);

notMask = ~mask;

mask = mask | bwpropfilt(notMask,'Area',[-Inf, 5000 - eps(5000)]);

showMaskAsOverlay(0.5,mask,'r');

Algorithm for Modifying the background

1. Read a new background image
2. Resize the background image to the original image
3. Break down the planes of original cover image into r, g, b component
4. Change intensity of every pixel value of all 3 components of the original image where the mask is not applied to the pixel values of the background image.
5. Reconstruct the RGB image

Example:

background = imread('background.jpg');
[x, y, z] = size(img);
background = imresize(background, [x y]);

r = img(:,:,1);

g = img(:,:,2);

b = img(:,:,3);

rb = background(:,:,1);

gb = background(:,:,2);

bb = background(:,:,3);

r(~mask) = rb(~mask);

g(~mask) = gb(~mask);

b(~mask) = bb(~mask);

img = cat(3,r,g,b);

Cover Image

Background

Modified Image