Colour Guided Colour Image Steganography using MATLAB

Report submitted to SASTRA Deemed to be University as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

GOTTUMUKKALA SAI
(Reg. No.: 122004080, B.Tech in ECE)

February 2021

School of Electrical & Electronics Engineering
SASTRA DEEMED TO BE UNIVERSITY
(A University established under section 3 of the UGC Act ,1956)
Tirumalaisamudram Thanjavur-613401

School of Electrical & Electronics Engineering
THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**COLOUR GUIDED COLOUR IMAGE STEGANOGRAPHY USING MATLAB**" submitted as a requirement for the course, **CSE302 : COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri. GOTTUMUKKALA SAI (Reg.No.122004080, B.Tech in ECE**) during the academic year 2020-21, in Electronics & Communication Engineering

Project Based Work *Viva voc*e held on ________________

Examiner 1 **Examiner 2**

Table of Contents

ACKNOWLEDGEMENTS

First of all, I would like to thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr S.Vaidyasubramaniam , Vice-Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr.Thenmozhi.K, Dean,School of EEE** and **R.Chandramouli, Registrar** for their overwhelming support during my course span in SASTRA Deemed University.

I would specially thank and express my profound gratitude to **Prof.Manikandan.C** for providing me an opportunity to do this project.

I also thank all the Teaching and Non-teaching faculty, and all other people who have directly or indirectly helped me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who helped me acquire this interest in the project and aided me in completing it within the deadline without much struggle.

ABSTRACT

Information security has become a cause of concern because of the electronic eavesdropping. Capacity, robustness and invisibility are important parameters in information hiding and are quite difficult to achieve in a single algorithm. This project proposes a novel steganography technique for digital color image which achieves the purported targets.

Of the three colour channels (Red, Green, Blue) in a given colour image, the least two significant bits of any one of the channels of the color image is used to channelize the embedding capacity of the remaining two channels.

In this project we have considered Red as our default guide to channelize the embedding capacity

KEY WORDS : Steganography , Channel , Embedding

List of Figures

INTRODUCTION

In today's info-driven world, with accruement of confidential information, there has been a corresponding increase in the attempts to sabotage the security guards of such information. This has led to an avalanche of pro and anti-security innovations. Steganography is one such pro-security innovation in which secret data is embedded in a cover. The concept of data hiding or steganography was first introduced by Simmons in 1983. Dictionary defines "Steganography" as the art of writing in cipher, or in character, which are not perceivable except to person who has the key. In the field of computers, steganography has evolved as the promising option of hiding a message in a cover, whose presence cannot be discerned by any third party without the knowledge of the key. The cover in which the message is hidden can either be a text, image, audio or video file. Even after hiding the data, the stego image should be imperceptible i.e., the cover image and the stego image should be inert and impregnable.

STEGANOGRAPHY :

Steganography is the practice of concealing a message within another message. That is, We conceal a secret data inside a cover data in such a way that any unintended third person will not notice the presence of secret data.

CHANNEL :

Any colour image is composed of pixels. Every pixel in an image has its own R,G,B values in the range of [0,255]. How many numbers are used to specify the color of each pixel is the number of channels each pixel has. In RGB as described above, an image has three numbers for each pixel that directly correspond to the three R, G and B elements in the computer display. Such RGB images have three channels.

ENCRYPTION :

Encryption is the process where we take the secret data and push it into the image bit by bit. In this project we embed the seecret data into Blue and Green channels depending upon the LSBs of each pixel in Red channel.

METHODOLOGY :

ENCRYPTION ALGORITHM :

1. Convert the given secret data into binary format

2. Split the cover image C into Red, Green and Blue Planes.(R,G and B respectively)

3. For each pixel in R, do the following:
    3.1. Let b[0]=LSB of the current pixel in R
    3.2. Let b[1]=Next LSB of the current pixel in R
    3.3. Let n= (Decimal value of b) + 3
    i.e., (Excess 3 value of b)
    3.4. If (n mod 2 = 0) then,
    Embed (n/2) bits of secret data in current pixels of G and B.
    Else
    Embed [(n-1)/2] bits and [(n+1)/2] bits of secret data in current pixels of G and B
    3.5. If all secret data is embedded, then
    Go to step-4

4. Store the resulting image as Stego Image (S)

RECOVERY ALGORITHM :

Input: Stego Image(S) , Length of secret data (n)
Output: Secret Data (D)

1. Split the stego image S into Red, Green and Blue Planes.(R,G and B respectively)

2. For each pixel in R, do the following:
    2.1. Let b[0]=LSB of the current pixel in R
    2.2. Let b[1]=Next LSB of the current pixel in R
    2.3. Let n= (Decimal value of b) + 3
    i.e., (Excess 3 value of b)
    2.4. If (n mod 2 = 0) then,
    Read (n/2) LSBs of current pixels of G and B and concatenate to D.
    Else
    Read [(n-1)/2] bits and [(n+1)/2] LSBs of current pixels of G and B respectively and concatenate to D.

3. Store the resulting recovered secret data (D).

IMPLEMENTATION :

ENCRYPTION :

1. Converting the given secret data into binary format :

Code :

```
% Message to be embedded
message='hey sastra';

% Length of the message where each character is 8 bits
len = length(message) * 8;

% Get all the ASCII values of the characters of the
message
ascii_value = uint8(message);

% Convert the decimal values to binary values
bin_message = transpose(dec2bin(ascii_value, 8));

% Get all the binary digits in separate rows of matrix
bin_message = bin_message(:);

% Length of the binary message
N = length(bin_message);

% Converting the char array to numeric array
bin_num_message=str2num(bin_message);
```

In this step, we take a secret message that is to be transmitted and convert it into binary format so that we can embed the binary data into the pixels of cover image

2. Split the image into RGB channels

Code :

```
% Split the cover image into R,G,B channels
redChannel = input(:,:,1);
greenChannel = input(:,:,2);
blueChannel = input(:,:,3);
```

Find the LSBs of each pixel in Red channel

Code :

```
% (i,j) pointers of G and B channels to traverse while embedding
ipG = 0; jpG = 0; ipB = 0; jpB = 0;

% Traverse through the image
for irC = 1 : heightrC
    if flag == 0
        for jrC = 1 : widthrC

            % Check if more bits are remaining to embed
            if(embed_counter <= N)

            % Finding the Least Significant Bit of the current pixel
            LSB0 = mod(double(redChannel(irC, jrC)), 2);

            % Finding second LSB
            temp = mod(double(redChannel(irC, jrC)), 4);
            if temp == 2||3
                LSB1 = 1;
            else
                LSB1 = 0;
            end
```

3. Depending upon the LSBs of pixel i,j in red plane, embed certain number of bits into B and G channels as explained in the above algorithm.

Code :

```
 if LSB1 == 0 && LSB0 == 0

                    for i = (ipG+1)
                        for j = (jpG+1) : (jpG+2)
                        % Embed two bits in greenChannel

                            if embed_counter > len
                                break
                            else
                                % LSB based embedding
                                LSB =
mod(double(greenChannel(i,j)), 2);
                                temp = double(xor(LSB,
bin_num_message(embed_counter)));
                                % Embedding
                                fingC(i,j) =
greenChannel(i,j) + temp;
                                embed_counter =
embed_counter + 1;
                                % Shift to next row after
all columns are filled
                                if jpG == 512
                                    ipG = ipG + 1;
                                    jpG=0;
                                else
                                    jpG = jpG + 1;
                                end

                            end

                        end
                    end

                    for i = (ipB+1)
                        for j = (jpB+1) % Embed one bit in
blueChannel

                            if embed_counter > len
                                break
                            else
                                LSB =
mod(double(blueChannel(i,j)), 2);
                                temp = double(xor(LSB,
bin_num_message(embed_counter)));
                                finbC(i,j) =
blueChannel(i,j) + temp;
                                embed_counter =
embed_counter + 1;
                                if jpB == 512
                                    ipB = ipB + 1;
                                    jpB=0;
                                else
                                    jpB= jpB + 1;
                                end
                            end
                        end
                    end
                end
```

Similarly, we embed binary data into the pixels even if the LSBs are 1,0 0,1 1,1 also

4. Now we concatenate these channels after embedding into one image

Code :

```
% Concatenate the channels
rgbImage = cat(3, redChannel, fingC, finbC);

% Write both the input and output images to local storage
% Mention the path to a folder here.
imwrite(input, "D:\CN_Steg\originalImage.png");
imwrite(rgbImage, "D:\CN_Steg\stegoImage.png");
```

In this way we get a stego image and hence the process of encryption is done.

RECOVERY :

1. Split the stego image into R,G,B channels

Code :

```
% Getting the input image
input = imread("D:\CN_Steg\stegoImage.png","png");

% Split the image into R,G,B channels
redChannel = input(:,:,1);
greenChannel = input(:,:,2);
blueChannel = input(:,:,3);
```

Find the LSBs of each pixel in Red channel :

Code :

```
% Traverse through the image
for irC = 1 : heightrC
    if flag == 0
        for jrC = 1 : widthrC

            % Check if more bits are remaining to embed
            if(embed_counter <= N)

            % Finding the Least Significant Bit of the current pixel
            LSB0 = mod(double(redChannel(irC, jrC)), 2);

            % Finding second LSB
            temp = mod(double(redChannel(irC, jrC)), 4);
            if temp == 2||3
                LSB1 = 1;
            else
                LSB1 = 0;
            end
```

2. To recover the hidden data we do exact opposite of what we did during encryption

Code :

```
if LSB1 == 0 && LSB0 == 0

                    % This means there are two bits in
green and one bit in
                    % blue channels resp.

                    for i = (ipG+1)
                        for j = (jpG+1) : (jpG+2)

                            if counter > message_length
                                break
                            else
                                % Store the LSB of the pixel
in extracted_bits
                                extracted_bits(counter, 1) =
mod(double(greenChannel(i, j)), 2);
                                counter = counter + 1;
                                if jpG == 512
                                    ipG = ipG + 1;
                                    jpG=0;
                                else
                                    jpG = jpG + 1;
                                end

                            end

                        end
                    end

                    for i = (ipB+1)
                        for j = (jpB+1)

                            if counter > message_length
                                break
                            else
                                % Store the LSB of the pixel
in extracted_bits
                                extracted_bits(counter, 1) =
mod(double(blueChannel(i, j)), 2);
                                counter = counter + 1;
                                if jpB == 512
                                    ipB = ipB + 1;
                                    jpB=0;
                                else
                                    jpB= jpB + 1;
                                end
                            end
                        end
                    end
                end
```

Convert the extracted binary data into ASCII characters

Code :

```
% Powers of 2 to get the ASCII value from binary
binValues = [ 128 64 32 16 8 4 2 1 ];

% Get all the bits in 8 columned table
% Each row is the bits of the character
% in the hidden text

binMatrix = reshape(extracted_bits, 8,(message_length/8));

% Convert the extracted bits to characters
% by multiplying with powers of 2
binMatrix;
textString = char(binValues*binMatrix);

% Print the hidden text
disp("MESSAGE : ");disp("");
disp(textString);
```

Evaluate the metrics

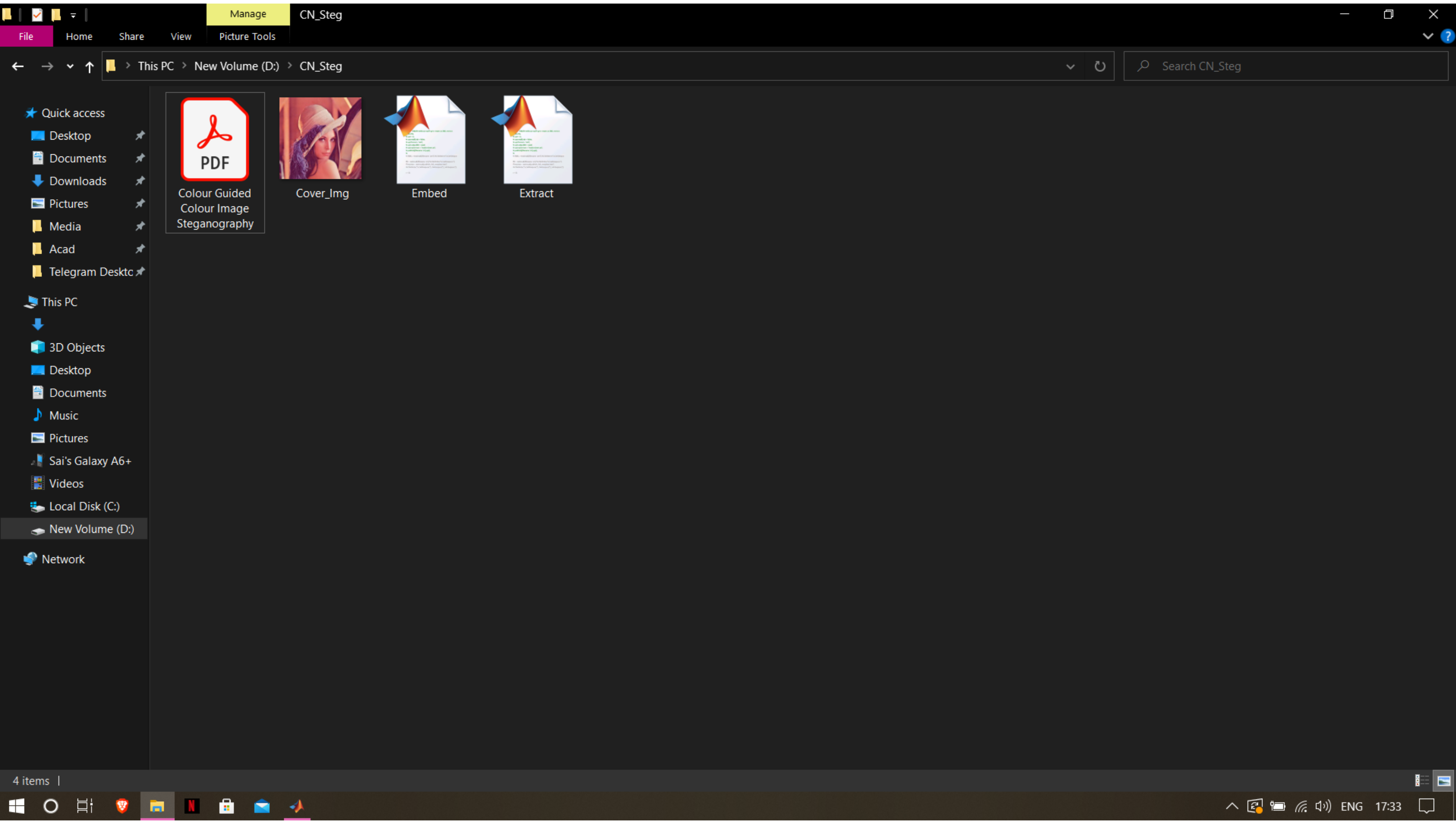
Code :

```
im0=imread("D:\CN_Steg\originalImage.png");
im1=imread("D:\CN_Steg\stegoImage.png");
peaksnr=psnr(im0,im1);
disp("PSNR : ");
disp(peaksnr);
mse=immse(im0,im1);
disp("MSE : ");
disp(mse);

figure(1);
subplot(2,2,1);
histogram(im0);
subplot(2,2,2);
histogram(im1);


subplot(2,2,3);
h1=histogram(im0);
hold on
h2=histogram(im1);
```

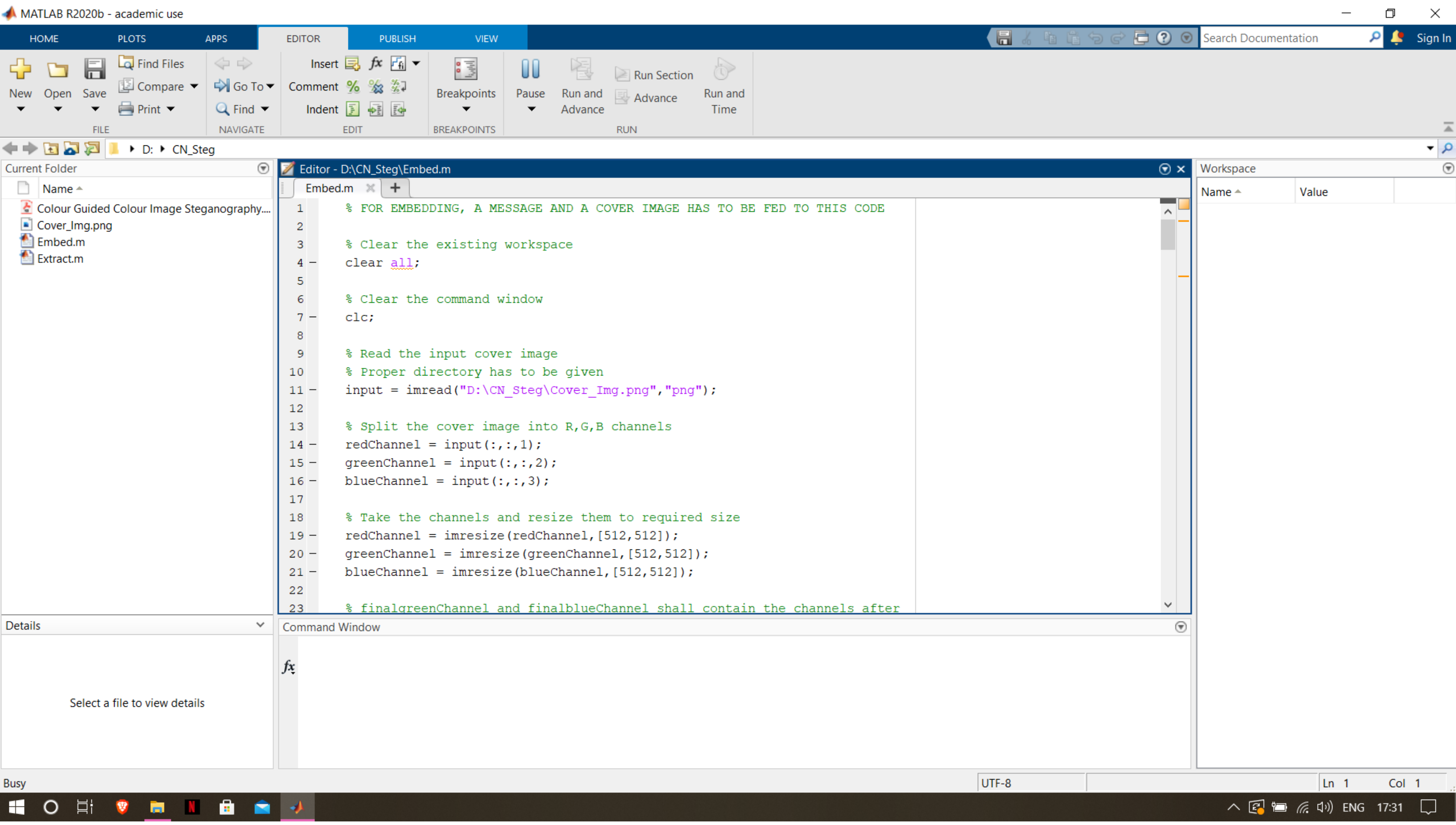
SCREENSHOTS :

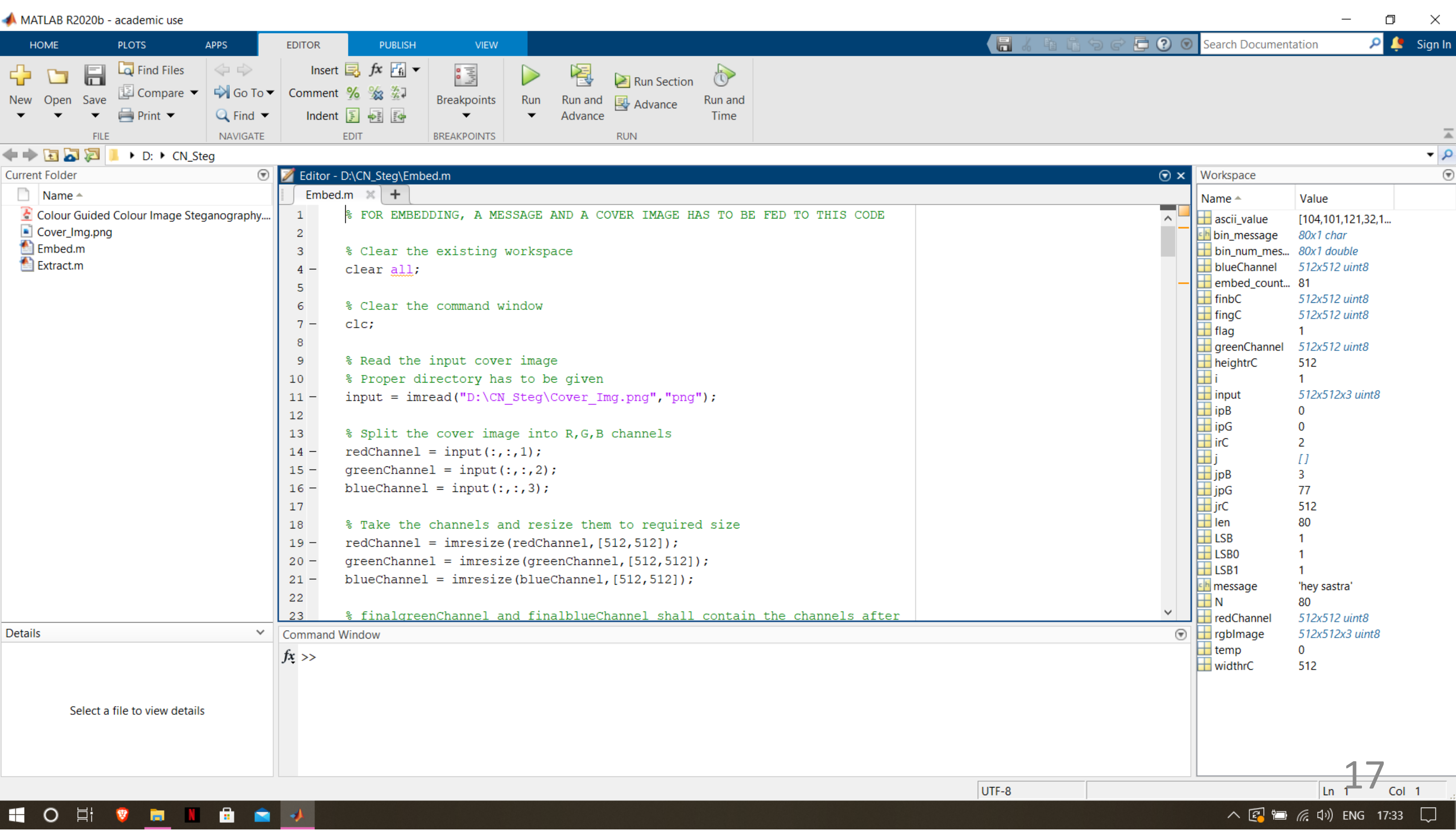
The below captured screenshots illustrate how the code makes use of cover image and embed the secret data into it and creates a stego image in the directory

Directory before running encryption code :

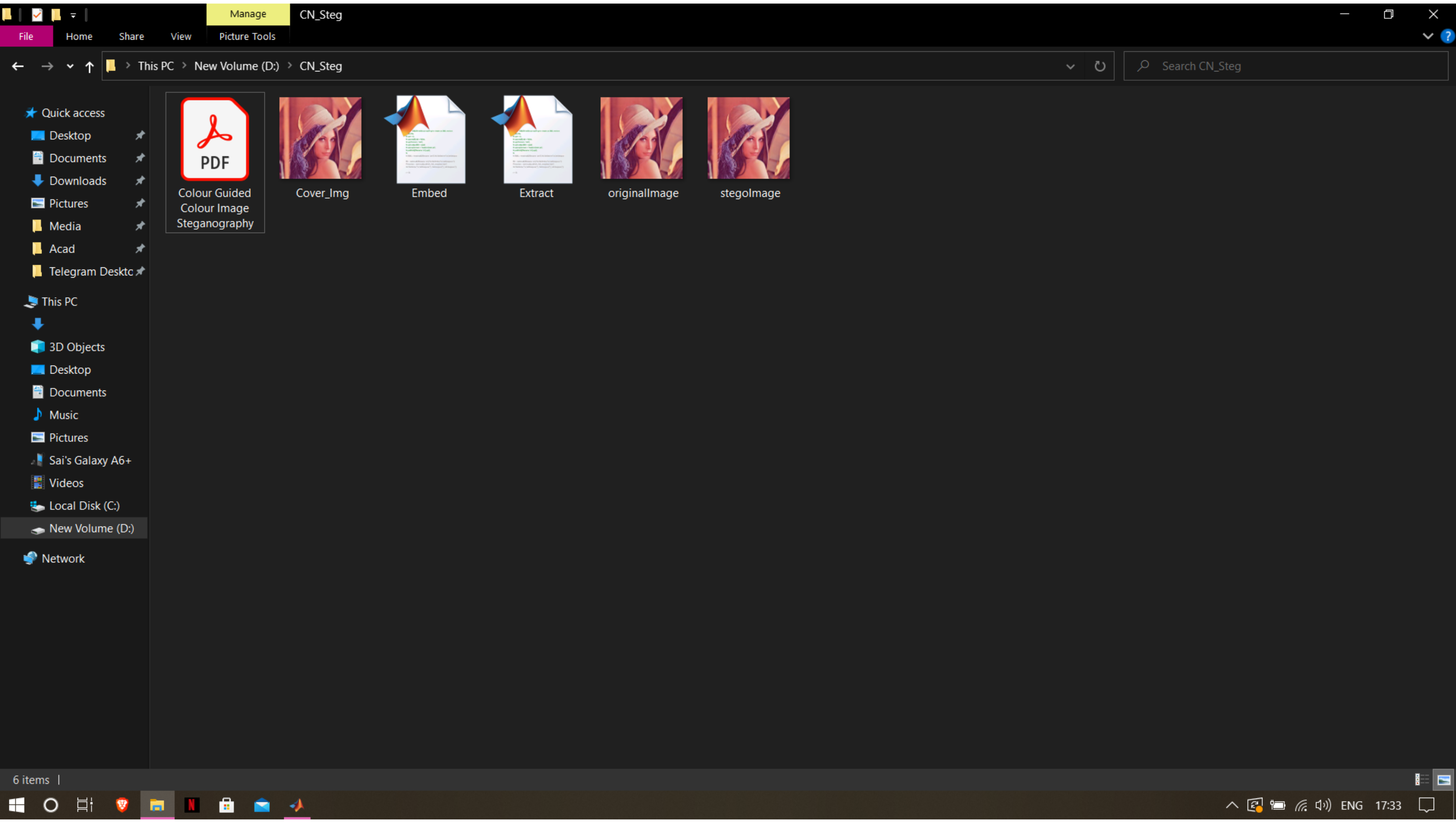


Execution of code :



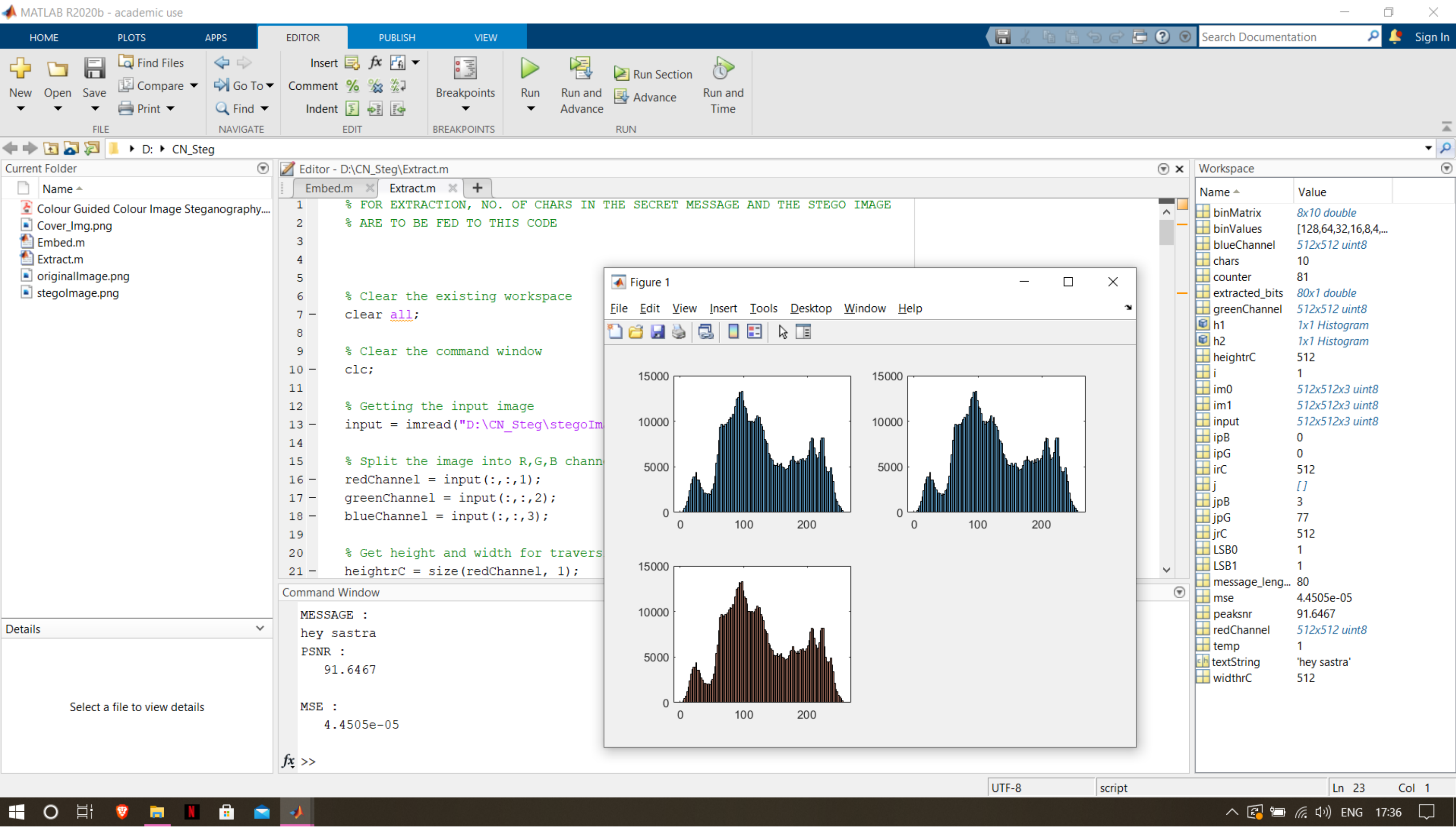


Directory after execution of code :



We can notice the creation of originalImage and stegoImage

Execution of recovery code :



We can notice successful recovery of message. In general this code is executed at the receivers end. Histograms of originalImage,stegoImage and an overlap are plotted and the performance metrics PSNR and MSE are evaluated.

In this way the secret data is recovered at the receivers end

Cover Image used for Steganography

Stego image that is transmitted

It is very evident from the above images that the changes we do by embedding are unnoticeable to naked eye.

METRICS :

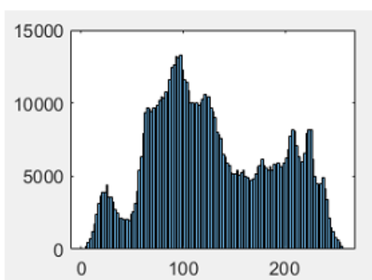
Metrices	Results
MSE	4.4505e-05
PSNR	91.6467

PSNR :

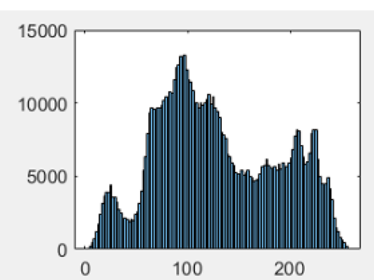
Peak signal-to-noise ratio (**PSNR**) is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

MSE :
Mean Squared Error (MSE) measures the average of the squares of the errors - that is, the average squared difference between the estimated values and the actual value

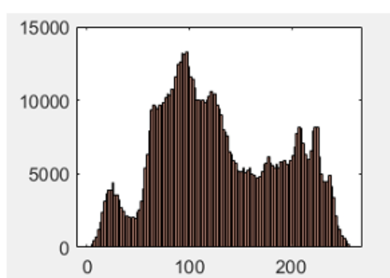
HISTOGRAMS :



Histogram of Lena before



Histogram of Lena after



Histograms overlapped

SOURCE CODE

ENCRYPTION :

```
% FOR EMBEDDING, A MESSAGE AND A COVER IMAGE HAS TO BE FED TO THIS CODE

% Clear the existing workspace
clear all;

% Clear the command window
clc;

% Read the input cover image
% Proper directory has to be given
input = imread("D:\CN_Steg\Cover_Img.png","png");

% Split the cover image into R,G,B channels
redChannel = input(:,:,1);
greenChannel = input(:,:,2);
blueChannel = input(:,:,3);

% Take the channels and resize them to required size
redChannel = imresize(redChannel,[512,512]);
greenChannel = imresize(greenChannel,[512,512]);
blueChannel = imresize(blueChannel,[512,512]);

% finalgreenChannel and finalblueChannel shall contain the channels after
                                    % embedding
fingC = greenChannel;
finbC = blueChannel;

% Message to be embedded
message='hey sastra';

% Length of the message where each character is 8 bits
len = length(message) * 8;

% Get all the ASCII values of the characters of the message
ascii_value = uint8(message);

% Convert the decimal values to binary values
bin_message = transpose(dec2bin(ascii_value, 8));

% Get all the binary digits in separate rows of matrix
bin_message = bin_message(:);

% Length of the binary message
N = length(bin_message);
```

```
% Converting the char array to numeric array
bin_num_message=str2num(bin_message);

% Get height and width for traversing through the red channel of
                    % the image to embed the data
heightrC = size(redChannel, 1);
widthrC = size(redChannel, 2);

% Counter for number of embedded bits for termination after embedding
embed_counter = 1;

% To exit the outer loop once the whole message is embedded
flag=0;

% (i,j) pointers of G and B channels to traverse while embedding
ipG = 0; jpG = 0; ipB = 0; jpB = 0;

% Traverse through the image
for irC = 1 : heightrC
   if flag == 0
      for jrC = 1 : widthrC

         % Check if more bits are remaining to embed
         if(embed_counter <= N)

         % Finding the Least Significant Bit of the current pixel
         LSB0 = mod(double(redChannel(irC, jrC)), 2);

         % Finding second LSB
         temp = mod(double(redChannel(irC, jrC)), 4);
         if temp == 2 || 3
            LSB1 = 1;
         else
            LSB1 = 0;
         end

         % LOGIC AS PER THE REFERENCE

         if LSB1 == 0 && LSB0 == 0

            for i = (ipG+1)
               for j = (jpG+1) : (jpG+2) % Embed two bits in greenChannel

                  if embed_counter > len
                     break
                  else
                     % LSB based embedding
                     LSB = mod(double(greenChannel(i,j)), 2);
                     temp = double(xor(LSB, bin_num_message(embed_counter)));
                     % Embedding
                     fingC(i,j) = greenChannel(i,j) + temp;
                     embed_counter = embed_counter + 1;
                     % Shift to next row after all columns are filled
                     if jpG == 512
                        ipG = ipG + 1;
                        jpG=0;
                     else
                        jpG = jpG + 1;
                     end

                  end

               end
            end

            for i = (ipB+1)
               for j = (jpB+1) % Embed one bit in blueChannel

                  if embed_counter > len
                     break
                  else
                     LSB = mod(double(blueChannel(i,j)), 2);
                     temp = double(xor(LSB, bin_num_message(embed_counter)));
                     finbC(i,j) = blueChannel(i,j) + temp;
                     embed_counter = embed_counter + 1;
                     if jpB == 512
                        ipB = ipB + 1;
                        jpB=0;
                     else
                        jpB= jpB + 1;
```

```
if LSB1 == 1 && LSB0 == 0

        for i = (ipG+1)
            for j = (jpG+1) : (jpG+3) % Embed three bits in greenChannel

                if embed_counter > len
                    break
                else
                    LSB = mod(double(greenChannel(i,j)), 2);
                    temp = double(xor(LSB, bin_num_message(embed_counter)));
                    fingC(i,j) = greenChannel(i,j) + temp;
                    embed_counter = embed_counter + 1;
                    if jpG == 512
                        ipG = ipG + 1;
                        jpG=0;
                    else
                        jpG = jpG + 1;
                    end
                end

            end
        end

        for i = (ipB+1)
            for j = (jpB+1) : (ipB+2) % Embed two bits in blueChannel

                if embed_counter > len
                    break
                else
                    LSB = mod(double(blueChannel(i,j)), 2);
                    temp = double(xor(LSB, bin_num_message(embed_counter)));
                    finbC(i,j) = blueChannel(i,j) + temp;
                    embed_counter = embed_counter + 1;
                    if jpB == 512
                        ipB = ipB + 1;
                        jpB=0;
                    else
                        jpB = jpB + 1;
                    end
                end

            end
        end
    end

    if LSB1 == 0 && LSB0 == 1

        for i = (ipG+1)
            for j = (jpG+1) : (jpG+2) % Embed two bits in greenChannel

                if embed_counter > len
                    break
                else
                    LSB = mod(double(greenChannel(i,j)), 2);
                    temp = double(xor(LSB, bin_num_message(embed_counter)));
                    fingC(i,j) = greenChannel(i,j) + temp;
                    embed_counter = embed_counter + 1;
                    if jpG == 512
                        ipG = ipG + 1;
                        jpG=0;
                    else
                        jpG = jpG + 1;
                    end
                end

            end
        end
        for i = (ipB+1)
            for j = (jpB+1) : (ipB+2) % Embed two bits in blueChannel

                if embed_counter > len
                    break
                else
                    LSB = mod(double(blueChannel(i,j)), 2);
                    temp = double(xor(LSB, bin_num_message(embed_counter)));
                    finbC(i,j) = blueChannel(i,j) + temp;
                    embed_counter = embed_counter + 1;
                    if jpB == 512
                        ipB = ipB + 1;
```

```
 if LSB1 == 1 && LSB0 == 1

          for i = (ipG+1)
             for j = (jpG+1) : (jpG+3) % Embed three bits in greenChannel

                if embed_counter > len
                   break
                else
                   LSB = mod(double(greenChannel(i,j)), 2);
                   temp = double(xor(LSB, bin_num_message(embed_counter)));
                   fingC(i,j) = greenChannel(i,j) + temp;
                   embed_counter = embed_counter + 1;
                   if jpG == 512
                      ipG = ipG + 1;
                      jpG=0;
                   else
                      jpG = jpG + 1;
                   end
                end

             end
          end
          for i = (ipB+1)
             for j = (jpB+1) : (ipB+3)% Embed three bits in blueChannel

                if embed_counter > len
                   break
                else
                   LSB = mod(double(blueChannel(i,j)), 2);
                   temp = double(xor(LSB, bin_num_message(embed_counter)));
                   finbC(i,j) = blueChannel(i,j) + temp;
                   embed_counter = embed_counter + 1;
                   if jpB == 512
                      ipB = ipB + 1;
                      jpB=0;
                   else
                      jpB = jpB + 1;
                   end
                end
             end
          end
       end


      else
         flag = 1;
         continue;
         end
      end
   else
      break;
   end
end

% Concatenate the channels
rgbImage = cat(3, redChannel, fingC, finbC);

% Write both the input and output images to local storage
% Mention the path to a folder here.
imwrite(input, "D:\CN_Steg\originalImage.png");
imwrite(rgbImage, "D:\CN_Steg\stegoImage.png");
```

RECOVERY :

```
% FOR EXTRACTION, NO. OF CHARS IN THE SECRET MESSAGE AND THE STEGO IMAGE
% ARE TO BE FED TO THIS CODE




% Clear the existing workspace
clear all;


% Clear the command window
clc;


% Getting the input image
input = imread("D:\CN_Steg\stegoImage.png","png");


% Split the image into R,G,B channels
redChannel = input(:,:,1);
greenChannel = input(:,:,2);
blueChannel = input(:,:,3);


% Get height and width for traversing through the image to embed the data
heightrC = size(redChannel, 1);
widthrC = size(redChannel, 2);


% Number of characters of the hidden text
chars = 10;


% Number of bits in the message
message_length = chars * 8;


% counter to keep track of number of bits extracted
counter = 1;


ipG = 0; jpG = 0; ipB = 0; jpB = 0;


% Traverse through the image
for irC = 1 : heightrC
  for jrC = 1 : widthrC

    % If more bits remain to be extracted
    if (counter <= message_length)

      % Finding the Least Significant Bit of the current pixel
      LSB0 = mod(double(redChannel(irC, jrC)), 2);

      % Finding second LSB
      temp = mod(double(redChannel(irC, jrC)), 4);

      if temp == 2||3
        LSB1 = 1;
      else
        LSB1 = 0;
      end
```

```
 if LSB1 == 0 && LSB0 == 0

    % This means there are two bits in green and one bit in
    % blue channels resp.

   for i = (ipG+1)
      for j = (jpG+1) : (jpG+2)

         if counter > message_length
            break
         else
            % Store the LSB of the pixel in extracted_bits
            extracted_bits(counter, 1) = mod(double(greenChannel(i, j)), 2);
            counter = counter + 1;
            if jpG == 512
               ipG = ipG + 1;
               jpG=0;
            else
               jpG = jpG + 1;
            end

         end

      end
   end

   for i = (ipB+1)
      for j = (jpB+1)

         if counter > message_length
            break
         else
            % Store the LSB of the pixel in extracted_bits
            extracted_bits(counter, 1) = mod(double(blueChannel(i, j)), 2);
            counter = counter + 1;
            if jpB == 512
               ipB = ipB + 1;
               jpB=0;
            else
               jpB= jpB + 1;
            end
         end
      end
   end
end

if LSB1 == 1 && LSB0 == 0

   % This means there are three bits in green and two bits in
    % blue channels resp.

   for i = (ipG+1)
      for j = (jpG+1) : (jpG+3)

         if counter > message_length
            break
         else
            % Store the LSB of the pixel in extracted_bits
            extracted_bits(counter, 1) = mod(double(greenChannel(i, j)), 2);
            counter = counter + 1;
            if jpG == 512
               ipG = ipG + 1;
               jpG=0;
            else
               jpG = jpG + 1;
            end
         end

      end
   end
```

```
for i = (ipB+1)
        for j = (jpB+1) : (ipB+2)

            if counter > message_length
                break
            else
                % Store the LSB of the pixel in extracted_bits
                extracted_bits(counter, 1) = mod(double(blueChannel(i, j)), 2);
                counter = counter + 1;
                if jpB == 512
                    ipB = ipB + 1;
                    jpB=0;
                else
                    jpB= jpB + 1;
                end
            end

        end
    end
end

if LSB1 == 0 && LSB0 == 1

    % This means there are two bits in green and two bits in
    % blue channels resp.

    for i = (ipG+1)
        for j = (jpG+1) : (jpG+2)

            if counter > message_length
                break
            else
                % Store the LSB of the pixel in extracted_bits
                extracted_bits(counter, 1) = mod(double(greenChannel(i, j)), 2);
                counter = counter + 1;
                if jpG == 512
                    ipG = ipG + 1;
                    jpG=0;
                else
                    jpG = jpG + 1;
                end
            end

        end
    end
    for i = (ipB+1)
        for j = (jpB+1) : (ipB+2)

            if counter > message_length
                break
            else
                % Store the LSB of the pixel in extracted_bits
                extracted_bits(counter, 1) = mod(double(blueChannel(i, j)), 2);
                counter = counter + 1;
                if jpB == 512
                    ipB = ipB + 1;
                    jpB=0;
                else
                    jpB= jpB + 1;
                end
            end
        end
    end
end

if LSB1 == 1 && LSB0 == 1

    % This means there are three bits in green and three bits in
    % blue channels resp.

    for i = (ipG+1)
        for j = (jpG+1) : (jpG+3)

            if counter > message_length
                break
            else
                % Store the LSB of the pixel in extracted_bits
                extracted_bits(counter, 1) = mod(double(greenChannel(i, j)), 2);
                counter = counter + 1;
                if jpG == 512
```

```
 for i = (ipB+1)
          for j = (jpB+1) : (ipB+3)

             if counter > message_length
               break
             else
               % Store the LSB of the pixel in extracted_bits
               extracted_bits(counter, 1)
= mod(double(blueChannel(i, j)), 2);
               counter = counter + 1;
               if jpB == 512
                 ipB = ipB + 1;
                 jpB=0;
               else
                 jpB= jpB + 1;
               end
             end
           end
         end
       end

     end
   end
end

% Powers of 2 to get the ASCII value from binary
binValues = [ 128 64 32 16 8 4 2 1 ];

% Get all the bits in 8 columned table
% Each row is the bits of the character
% in the hidden text

binMatrix = reshape(extracted_bits, 8,(message_length/8));

% Convert the extracted bits to characters
% by multiplying with powers of 2
binMatrix;
textString = char(binValues*binMatrix);

% Print the hidden text
disp("MESSAGE : ");disp("");
disp(textString);




im0=imread("D:\CN_Steg\originalImage.png");
im1=imread("D:\CN_Steg\stegoImage.png");
peaksnr=psnr(im0,im1);
disp("PSNR : ");
disp(peaksnr);
mse=immse(im0,im1);
disp("MSE : ");
disp(mse);

figure(1);
subplot(2,2,1);
histogram(im0);
subplot(2,2,2);
histogram(im1);


subplot(2,2,3);
h1=histogram(im0);
hold on
h2=histogram(im1);
```

CONCLUSION :

In this paper we have proposed a novel and adaptive method to embed the secret data in the cover image with high security and imperceptibility. The receiver does not need the original image to extract the information.

The embedding depends completely on the nature of the pixels which is not predictable. This makes it completely adaptive and random because the nature of the pixels cannot be controlled; it is inherent of an image.

REFERENCES :

[1] Bruice Schneier, Applied Cryptography Protocols, Algorithm and Source Code in C. Second edition. Wiley India edition 2007

[2] S. Katzenbeisser, F.A.P. Petitcolas, Information Hiding Techniques for Steganography and Digital Watermarking, Artech House, Norwood, MA, 2000.

[3] W. Bender, D. Gruhl, N. Morimoto, A. Lu, Techniques for data hiding‖ IBM Syst. J. 35 (3&4) (1996) 313–336.

[4] G. J. Simmons, "The prisoners' problem and the subliminal channel" in Proc. Advances in Cryptology (CRYPTO '83), pp. 51-67

[5] R.Amirtharajan and R.John Bosco Balaguru. Constructive Role of SFC & RGB Fusion versus Destructive Intrusion.International Journal of Computer Applications 1(20):30–36

[6] R.Amirtharajan and Dr. R. John Bosco Balaguru, Tri-Layer Stego for Enhanced Security – A Keyless Random Approach‖ - IEEE Xplore, DOI, 10.1109/IMSAA.2009.5439438.

[8] R.Amirtharajan, R. Akila, P.Deepikachowdavarapu, A Comparative Analysis of Image Steganography. International Journal of Computer Applications 2(3)(2010):41–47.

[7] Abbas Cheddad, Joan Condell, Kevin Curran, Paul Mc Kevitt, Digital image steganography: Survey and analysis of current methods Signal Processing 90 (2010) 727–752 [8] R.Amirtharajan, R. Akila, P.Deepikachowdavarapu, A Comparative Analysis of Image Steganography. International Journal of Computer Applications 2(3)(2010):41–47.

[9] C.K. Chan, L.M. Chen, Hiding data in images by simple LSB substitution, Pattern Recognition 37 (3) (2004) 469–474.

[10] L.M. Marvel, C.G. Boncelet Jr., C.T. Retter, Spread spectrum image steganography, IEEE Trans. Image Process. 8 (8) (1999) 1075-1083.

[11] F.A.P. Petitcolas, R.J. Anderson, M.G. Kuhn, Information hiding—a survey, Proc. IEEE 87 (7) (1999) 1062–1078.

[12] N. Provos and P. Honeyman, Hide and seek: An introduction to steganography, IEEE Security Privacy Mag.,1 (3) (2003) 32–44/

[13] Po-Yueh Chen Hung-Ju Lin, A DWT Based Approach for Image Steganography”, International Journal of Applied Science and Engineering 4(3)(2006): 275-290

[14] R.Amirtharajan, Krishnendra Nathella and J Harish, ―Info Hide – A Cluster Cover Approach International Journal of Computer Applications 3(5)(2010) 11–18.

[15] R.Z. Wang, C.F. Lin, J.C. Lin, Image hiding by optimal LSB substitution and genetic algorithm, Pattern Recognition 34 (3) (2000) 671–683.

[16] C.M. Wang, N.I. Wu, C.S. Tsai, M.S. Hwang, A high quality steganography method with pixel-value differencing and modulus function, J. Syst. Software 81 (1) (2008) 150–158.

[17] Young-Ran Park, Hyun-Ho Kang, Sang-Uk Shin, and Ki-Ryong Kwon, An Image Steganography Using Pixel Characteristics Y. Hao et al. (Eds.): CIS 2005, Part II, Springer-Verlag Berlin Heidelberg LNAI 3802, (2005) 581– 588.