Matlab Coding For Text Steganography System By Using LSB Insertion Method With Key

Mohammed J. Khami, Lemya G. Shehab and Zeynab M. Jawar*

Abstract

A stenographic system of least significant bit (LSB) insertion method with key is accomplished and designed by Matlab programming language. The proposed system generally differs from other implementation techniques which use the LSB insertion method by the following points:

First point concerns the implemented secret message character set. It automatically detects secret message character set and determines if the message is written with Latin (with only 8 bits per character), or Latin/Arabic (with 16 bits per character), character set. Determining the character set of secret message text comes out with the benefit of the economic usage of the redundant bits of the covering image from one side and from the other side, allows handling of hiding Arabic text message, that very rarely dealt with by other researchers (they usually use English text messages).

The second point which differentiates our approach from others, deals with the way of selecting the location of where the message's bits are going to be embedded among the many pixels of the cover image. By which the given key string is used not just once, as in other techniques, but twice for two consecutive random selection operations, each with different number sequence outcomes.

While the third point is related to the way of equally distribute the hidden message bits on the image sub areas and this makes it more difficult to notice the existence of the hidden message.

The proposed technique is accomplished with the aide of specially written Matlab coded functions and two main algorithms for hiding and extracting secret text messages respectively.

Keywords: Steganography, LSB insertion method, peak signal to noise ratio (PSNR).
Introduction

Steganography is a technique to modulate a message inside a medium of misinterpretation such that the existence of the message is both hidden and difficult to recover when discovered. Many algorithms and procedures, such as Least Significant Bit (LSB), have been written to hide text in an image [1-3]. The goal is to make communication unintelligible to those who do not possess the right keys [4].

LSB method start by passing both secret message and cover image into the encoder. Then inside the encoder, one or several protocols will be implemented to embed the secret information into the cover medium to produce another look like copy of the original covering medium which it will be called stegoimage.

A key is needed in the embedding process. Key can be used to reduce the chance of third party attackers getting hold of the stegoimage and decoding it to find out the secret information [5,6].

In this paper, The hiding system mainly differs from others in three points. First point concerns the implemented character set. Determining the character set of secret message text comes out with the benefit of the economic usage of the redundant bits of the covering image.

![Fig.(1): Steganography system.](image)

The second differentiating point deals with the way of selecting the location of where the message's bits are going to be embedded among the many pixels of the cover image. By which the given key string is used not just once, as in other techniques, but twice for two consecutive random selection operations, each with different number sequence outcomes. While the third point is related to the way of equal distribution of the hidden message bits on the image sub areas and this makes it more difficult to notice the existence of the hidden object.
Proposed LSB insertion technique

Generally, in LSB insertion method with key, a random number generator is used to randomly distribute and hide bits of secret message into the least significant bit of cover image pixels. To do this, in common approach, single least significant bit from one of the whole available cover image pixels is randomly selected and then used in hiding one of the message text bits. This means that the randomizing operation uses the key string only once over the whole image pixels. And since the ordinary color image has lots of pixels, in fact massive quantity of pixels, thus the random number generator consume long time to complete the operation beside that the generation of large vector of random unique integers is consider difficult job from the programming point view.

This paper aims to hide secret text message in one color plane of a given colored cover image. The image whole scenes area will be logically subdivided or partitioned into equal square sub-areas. Pixels of each sub-area are represented by smaller data matrix that will be called as an image block. Image blocks are sequenced in column wise direction, as shown in Fig.(2). This is done by calling specially written Matlab function 'i_j_calc.m' of the following code:

![Image Blocks Sequence (After Partitioning Operation)](image1)

![Image blocks new sequence (After Changing Their Location Randomly)](image2)

Fig.(2): Cover Image partitioning and blocks re-sequencing.
function

 t=j_i_calc(ra,ca,ImBlkHieght)
 % To find upper left element address
 % of each block in logically partitioned
 % array [ra ca] with square blocks of
 % side ImBlkHieght.
 % The [ra ca] array will be partitioned
 % into blocks with columns wise direction.
 % input ra=total no. of rows of the
 % partitioned array.
 % ca=total no. of columns of
 % the partitioned array.
 % ImBlkHieght=required
 % block size.
 % output: t is addresses array of upper
 % left block's element. Where
 i=t{block sequence,1}
 j=t{block sequence,2}
 ie=fix
 (ra/ImBlkHieght)*ImBlkHieght;% No.
 of blocks in vertical direction.
 je=fix(ca/ImBlkHieght)*ImBlkHieght
 ;% No. of blocks in horizontal
 direction.
 t=[ ];
 bc=0;
 for j=1:ImBlkHieght:ie
 for i=1:ImBlkHieght:je
 bc=bc+1;
 t(bc,1)=i;
 t(bc,2)=j;
 end % j
 end % i
end % Function i_j_calc.m

Block size is determined according to the width, in bits, of the
implemented current message character set. The software programs
are written to deal with two different character sets, namely the Latin (8-bit
wide), and Latin/Arabic (16-bit wide). Hence, image data matrix is logically
partitioned into blocks of pixels of size either [8 x 8] or [16 x 16] pixels. Also,
message text has to be chunked into pieces (or text blocks), as depicted in
Fig.(3), each with equal number of characters. Maximum number of
characters in any text block should be such that their total bits number, if
converting them into binary format, is less or equal to total pixels in any
image block.

The returned benefits, of doing so, will be in the efficient use of all
image pixels and making the jobs of English/Arabic text hiding and
recovering more easily.

Order of blocks, at processing
time, is not accomplished according to
their true sequence that comes out
from the partition step. Instead, blocks
are re-ordered depending on element
values of a vector of unique and
randomly generated integer numbers.
Distribution and range of this vector
integer numbers, depend on the key
string (as a random number generator
seed), and total number of partitioned
blocks (as the required total integer
number). This vector can be obtained
by calling ‘RandArrayIndexing.m’
function of the following code:

function idx=RandArrayIndexing
(Key,NumArrayElem)
 % RandArrayIndexing function returns
 a row vector idx, containing ReqNum of
 % unique integers selected randomly
 from 1 to NumArrayElem inclusive, using
 % Key as seed to the used random
 number generator.
 Key = max(cumsum(double(Key))) ;
s=rng(Key); % Set the random number generator seed.
% The matlab 'randperm(n,k)' function returns a row vector containing k
% unique integers selected randomly from 1 to NumArrayElem inclusive.
idx=randperm(NumArrayElem);
end % Function RandArrayIndexing.m

The new block sequence is shown in Fig.(2). At this stage, the key string is used for the first time. The main purpose of this processing step is to control the random manipulation of image pixels on bases of whole image blocks or image blocks level. The second time of using the key string will be at the pixel selection within each single block, where each pixel sequence may be obtained in relative to another vector of unique integers (another key), derived from the from the same original key string but with total number of integers equal to total pixels in one block. This step is illustrated in Fig.(4).

One important step in any steganography system is the one concerning the encoding or the hiding function. The following function Matlab code 'hide_msg_in_im_encoder.m' does the encoding job.

Fig.(3): Message text partitioning.
function Gray_cover_im =
hide_msg_in_im_encoder(Gray_cover_im,key,msg,ImBlkHieght,BitInOneChar)
% This function is used to hide a message of text (msg), into a gray image
% matrix (Gray_cover_im) using LSB method with key string (Key) for
% randomizing the hiding elements locations in image array.
%
% Inputs:  Gray_cover_im =
Gray cover image array.
%  key = any character string that may be used as a seed for
% the random number generator.
%  msg = Text secret message.
%  ImBlkHieght= image rows.
%  BitInOneChar= message character set width in bits.
% Output:  Gray_cover_im=
encoded (or modified) cover image array.
%
% Get the size of the given image matrix.
NumArrayElem=size(Gray_cover_im,1)*size(Gray_cover_im,2);
% Convert message into binary format.
msgmat = dec2bin(msg)-48;
% Create a vector of random unique integer numbers from 1 to NumArrayElem.
idx=RandArrayIndexing(key,NumArrayElem);
% Encoder function is designed to work with text message with characters
% set of BitInOneChar, (8 or 16)
% bit wide character set. And if the character
% set is not equal to BitInOneChar then make it so by add
% leading zeros to it.
if size(msgmat,2)<BitInOneChar
    sizemsgmat=size(msgmat,2);
    msgmat_temp=msgmat;
    for i=1:size(msgmat,2)-size(msgmat,2)
        msgmat_temp=[msgmat_temp,0];
    end
    msgmat=msgmat_temp;
    BitInOneChar=size(msgmat,2);
end
% Encode the message.
msgmat(:,1:BitInOneChar-sizemsgmat)=0;
msgmat(:,BitInOneChar-sizemsgmat + 1 :)
BitInOneChar)= msgmat_temp
(: , 1:sizemsgmat );
end
%
% Start hiding the message bits
in LSB of the image pixel values.
for ROW = 1:ImBlkHieght
  for COL = 1:BitInOneChar
    CR=COL+BitInOneChar*(ROW-1);
    COLROW=idx(CR);
    if msgmat(ROW,COL)==1;
      if rem(Gray_cover_im(COLROW),
        2)==0
        Gray_cover_im(COLROW) =
        Gray_cover_im(COLROW)+1;
      end
    elseif rem(Gray_cover_im(COLROW),
        2)==1
        Gray_cover_im(COLROW) =
        Gray_cover_im(COLROW)-1;
    end
  end % for COL
end % for ROW
%
End % End of encoding function.

Also the system requires,
another function to extract the hidden
message text back. This is done by
'get_msg_from_im_decoder.m'
function of the following Matlab code.

function
msg=get_msg_from_im_decoder

er(Gray_cover_im,key,
ImBlkHieght,BitInOneChar)
% This function is to decode or
extract a text hidden into a gray
image % matrix.
% Inputs:
% Gray_cover_im = Gray cover
image array.
% key = any character string
that may be used as a seed for the
random % number generator.
%ImBlkHieght= image rows.
% BitInOneChar= message
color set width in bits.
% Output:
%  msg = text secret message.
%
% Get the size of the given
image matrix.
NumArrayElem=size(Gray_cove
r_im,1)*size(Gray_cover_im,2);
% Get the addresses of where
message text bits are hidden in
the image.
idx=RandArrayIndexing(key,Nu
mArrayElem);
% Start the decoding
msgmat =
zeros(ImBlkHieght,BitInOneCha
r);
for ROW = 1:ImBlkHieght
  for COL = 1:BitInOneChar
    if
      rem(Gray_cover_im(idx(COL+B
itInOneChar*(ROW-1))),2)==1
        msgmat(ROW,COL) = 1;
      end
    end
  end % for COL
end % for ROW
%
End %End of decoding function.
Two more Matlab codes are written as main programs. One code for hiding the secret message text in a given image file to produce another encoded (or modified) image, the stego image, which has the same size as the original image and looks as same as the original used cover image. While the other code is for extracting the hidden text from the stego image. Both codes need to use the same key string to accomplish their jobs. These codes work according to the following two algorithms respectively.

### Hiding text in image main algorithm:

1. Input secret message 'msg', text and hiding key string 'key'.

2. Calculate maximum bit's number 'BitInOneChar' used by secret message character set.

3. Get path 'pth' and filename 'filen' of the RGB colored cover image. And then read its RGB color's data matrix 'Cover_RGB_Im'.

4. Select from the image data matrix 'Cover_RGB_Im', only the blue color channel 'clr=3', data matrix and assign it to 'Gray_cover_im'.

5. Find rows 'ImageRow', columns 'ImageCol', and image size 'ImageSize', of the original cover image matrix 'Cover_RGB_Im'.

6. By using 'j_i_calc' function, partition the 'Gray_cover_im' data matrix into square blocks of side equal to 'BitInOneChar' data elements. And then find total number of image blocks 'TotImBlk'.

7. Calculate one image block character's hiding capacity 'CharInOneImBlk'.

8. Partition secret message into integer number of text blocks 'TotTxtBlk', each with 'CharInOneImBlk' characters.

9. By using 'hide_msg_in_im_encoder' function, hide 'TotTxtBlk', 'CharInOneImBlk', and 'BitInOneChar' values in last block of the 'Gray_cover_im' data matrix after it has been repartitioned into [8x8] blocks by implementing 'j_i_calc' function again.

10. Hide all secret message block's content into image blocks obtained from step (6) as in following:

   A. By using the key string 'key', and 'RandArrayIndexing' function, create a vector 'idx' with random integer values between 1 and ('TotImBlk' – 1).

   B. Loop for 'TotTxtBlk' times, and for each episode 'txtblknum' do the following:
      - Get current text block number 'txtblknum', and its content 'CurTxtBlk' from those secret message blocks which are not being hidden yet.
      - From the 'TotImBlk' blocks of 'Gray_cover_im', select one image block 'CurImBlk' with sequence number equal to...
the element value at location 'txtblknum' of the vector 'idx'.

• By using 'hide_msg_in_im_encoder' function, hide text block 'CurTxtBlk' in image block 'CurImBlk' to get new modified image block 'NewCurImBlk'.

• Put back the new modified image block 'NewCurImBlk' into its original place in the gray cover image array 'Gray_cover_im'.

C. Assign the original data of the cover image 'Cover_RGB_Im' to a new variable 'New_Cover_RGB_Im'.

1. Replace only blue color channel data matrix of 'New_Cover_RGB_Im' with the modified blue color plane data matrix 'Gray_cover_im'.

12. Save the modified cover image matrix 'New_Cover_RGB_Im' in a bit-map (.bmp) format file with any chosen path and filename.

Extracting secret message from stego image main algorithm:

1. Input path 'pth' and filename 'filen' of the sego image. And then read its RGB color's data matrix 'Cover_RGB_Im'.

2. Input key string 'key' value.

3. Select from the image data matrix 'Cover_RGB_Im' only the blue channel 'clr=3' data matrix and assign it to 'Gray_cover_im'.

4. Find numbers of rows 'ImageRow', and columns 'ImageCol' of the stegoimage matrix 'Cover_RGB_Im'.

5. By using 'j_i_calc' function, partition 'Gray_cover_im' data matrix into square blocks of side equal to [8 x 8] data elements.

6. By using 'get_msg_from_im_decoder' function, extract from the last block the values of total hidden text blocks 'TotTxtBlk', number of character hidden in one image block 'CharInOneImBlk', and maximum bit's number 'BitInOneChar' that is used by character set of the secret message at hiding stage.

7. Recover secret message text from the encoded cover image as follow:

A. By using 'j_i_calc' function, partition the 'Gray_cover_im' data matrix into square blocks of side equal to 'BitInOneChar' data elements.

B. By using the key string 'key', and 'RandArrayIndexing' function, create a vector 'idx' with random integer values between 1 and 'TotImBlk'.

C. Loop for 'TotTxtBlk' times and for each episode 'imblknum' do the following:

• From the total 'TotImBlk' blocks of 'Gray_cover_im', select one image block 'CurImBlk' with sequence number equal to the element value at sequence
'imblknum' of the vector 'idx'.

- By using 'get_msg_from_im_decoder' function, extract from 'CurImBlk' block the part of the secret message 'NewCurTxtBlk'.

- Concatenate the recovered secret message parts 'NewCurTxtBlk' all together to get the full text 'msg' of the hidden secret message.

8. Display the recovered message 'msg'.

**System test**

Our steganography system main algorithms are tested on different color cover images of different sizes and contents. Also the test has been performed with English only, and English/Arabic text messages. The used text messages are with different lengths (any text from 1 characters long to 1/8 of the cover image size). In this paper, the hiding technique uses only one color channel, the blue channel, of the three color channels (RGB), of the image, and of course, the main hiding algorithm can be modified easily to deal with the all three color channels to handle longer text messages.

The **first test** is to compare the quality of origin cover image with stego image. Using the same set of tests images, different image contents can be compared systematically to identify whether a particular image type and contents produce better results. Hiding text in an image has the effect of changing the visual quality of a digital image and since image quality measure may vary from person to person, therefore we use the quantitative/empirical measures of the peak signal to noise ratio (PSNR), to compare the effects of image enhancement algorithms on image quality.

With images, PSNR can be calculated as the ratio between the square of maximum possible pixel value of an image (255 in our case), and the mean square error of distorting noise that affects the quality of its representation [6]. It is usually expressed in terms of the logarithmic decibel scale and performed on images of equal sizes. The mathematical representation of the PSNR for grayscale image is as follows:

\[
PSNR = 20 \log \left( \frac{\text{max}(\text{max} \text{cover})}{(\text{MSE})^{0.5}} \right) \]

Where:

- \( MSE \) (Mean Squared Error) is:
  \[
  MSE = \frac{1}{(r \times c)} \sum \sum (\text{cover} - \text{stego})^2 
  \]

- \( \text{cover} \) : is the matrix data of the original cover image.
- \( \text{stego} \) : is the matrix data of the stego image.
- \( r \) : is the numbers of rows of pixels of the images.
- \( c \) : is the number of columns of pixels of the cover image.
- \( \text{max} \text{cover} \) : is the maximum pixel value that exists in original cover image.

From equation (1) above one can notice that the higher the PSNR, the better the match between origin and stego images and this is because the \( MSE \) of equation (2) will be zero for identical images. And to apply equations (1) for color images, the
\( \text{MSE} \) of equation (2) has to be taken over all pixels values of each individual color channel and is averaged with the number of color channels. The following Matlab code 'psnr_color.m' is written to calculate the PSNR value of a given pair of images.

```matlab
function PSNR_Value = psnr_color(CoverIm, StegoIm)
% psnr_color function is to computes the Peak Signal
% to Noise Ratio for two RGB images with same sizes.
% Inputs: CoverIm image data matrix.
%         StegoIm image data matrix
% Output: PSNR_value in decibel (dB).
%
% Make sure the two images of equal sizes.
if size(CoverIm) ~= size(StegoIm)
    error('The images must have the same size');
PSNR_Value = [ ];
    return
end
% Read the dimensions of the cover image.
[rows columns ~] = size(CoverIm);
% Calculate mean square error of color channels R, G, and B.
mseRcolor = (double(CoverIm(:,:,1)) - double(StegoIm(:,:,1))).^2;
mseGcolor = (double(CoverIm(:,:,2)) - double(StegoIm(:,:,2))).^2;
mseBcolor = (double(CoverIm(:,:,3)) - double(StegoIm(:,:,3))).^2;
mseR = sum(sum(mseRcolor)) / (rows * columns);
mseG = sum(sum(mseGcolor)) / (rows * columns);
mseB = sum(sum(mseBcolor)) / (rows * columns);
% Average mean square error of the three channels:
mse = (mseR + mseG + mseB)/3;
% Calculate PSNR (Peak Signal to noise ratio).
PSNR_Value = 10 * log10((255^2 / mse));
End % End of function
```

**Conclusions**

A stenographic system of LSB insertion method with key is accomplished and designed by Matlab programming language. In testing the system English and English/Arabic secret text messages of different lengths are used on many color cover images of different image types, contents, and sizes. By performing visual test (by people looking for any distortion on the modified images), as depicted in Fig.(5) and systematical measures (by using peak signal to noise ratio, PSNR), of Fig.(6), on system output stego images, the tests prove very good stego images equality in comparable to the original cover images.

Fig.(6,a) shows that computer processing time does not depend on used cover image size but it depends mainly on how many blocks that the secret text message has been chunk to, and it is clear, as in the figure, that the
processing time is directly proportional to the total secret text message blocks.

Fig.(6,b) depicts that the quality measure of the PSNR with the use of the same secret text message and images of different sizes. The figure shows that as cover image size increases so will the PSNR value. The line draw of the relation of Fig.(6,b) is not a straight line draw. This is due to the differences of the used cover images contents.

In Fig.(6,c), the same cover image is used but with secret text messages of different lengths. It clarifies the natural expected reverse proportional relationship between stego image quality (PSNR), and the secret message.

As a general conclusion, the designed system come out with the advantages of user handling simplicity, auto text character set (English or English/Arabic character sets) detection, good distribution of secret text message bits on overall pixels of the cover image, and it has a very good response from time manipulation point of view. The main disadvantage of the designed steganography system is the fact of having a weak robustness to any stego image manipulation and thus any modification or alteration on the stego image may cause a serious message recovery problem.

Fig.(5): Visual test samples: original images in (a) and (b) looks exactly as the modified images of (c) and (d) respectively.
Fig. (6): Relationships between :- (a) Processing time and secret message blocks (using different secret messages and images with different sizes), (b) PSNR and image size (using same secret message and images with different sizes), (c) PSNR and secret message length (using same image with secret messages with different lengths).

References


Steganography Technique Based on LSB Insertion and Encryption”, Advances in Recent Technologies in Communication and Computing, ARTCom ’09. International Conference on.,

لمستخدم Matlab لنظام إخفاء معلومات نصفي صورة باستخدام طريقة الإدراج مع LSB

النتائج

تذكر من إخفاء معلومات نصفي صورة باستخدام طريقة الإدراج مع LSB وتصميمه وانجازه باستخدام لغة البرمجة الماتلاب الاصلية

تختلف النظام المقترح عن تطبيقات التنفيذ الشبيهة والتي تعتمد طريقة LSB في النقطة التالية:

تتعلق نقطة الاختلاف الأولى بالانتشار الأولي لكون مجموعة الأحرف المستخدمة بالرسالة السرية المطلوب اخفاؤها وهل أن الرسالة المستخدمة مكتوبة فقط باللغة اللاتينية (التي تستخدم ثنائيات في تمثيل كل حرف أو رمز فيها)، أو أنها تستخدم مجموعة الأحرف اللاتينية/العربية (التي تستخدم ستة عشر ثنائية في تمثيل كل حرف أو رمز). يستفيد من تحديد ومعرفة نوع مجموعة الأحرف المستخدمة في كتابة الرسالة السرية في الاستعمال الاقتصادي لثنائيات بيانات صورة الغلاف التي تعتمد كوسط في إخفاء ثنائيات الرسالة السرية، من جهة ومن جهة أخرى، فإنها تسمح بالتعامل مع الرسائل المكتوبة بالعربية والتي نادراً ما يتم التعامل معها بقية الباحثين الذين يستخدمون الرسائل ذات النص الإنجليزي في إلغاء الأحيان.

والنقطة الثانية التي تميز نهجنا من الآخرين، تخص الأطراف الاختبار لموقع ثنائيات الرسالة السرية وفي أي من ثنائيات صورة الغلاف الكثير يجب إخفاء كل منها. كما مع أساليب البحث الأخرى، يتم استخدام مفهوم باتني في اختيار موقع الثنائية المناسبة من بين ثنائيات صورة الغلاف ولكن الاختلاف يتمثل باستخدام قيم سلسلة رقمت ذلك المفهوم ليس لمرة واحدة فقط كما هو في حالة إلغاء البحث الأخرى ولكن لمترين متتاليتين من عمليات الاختيار العشوائي إذ تنتج كل عملية منها تسلسل رقمي مختلف يعتمد في اختيار مواقع الثنائية المناسبة للإخفاء ضمن صورة الغلاف.

في حين ترتبط نقطة الاختلاف الثالثة بطرقية التوزيع المتساوي لثنائيات الرسالة السرية على مساحة صورة الغلاف مما يجعل الأمر أكثر صعوبة في ملاحظة وجود رسالة خفية فيها.

تم تنفيذ النظام المقترح عن طريق إعداد وكتابة بعض النصوص الخاصة بالبحث باستخدام لغة برمجة الماتلاب مع عرض خوارزمياته الأولي للكشف والثانية لاسترجاع نص الرسالة السرية.