Volume 6, Issue 7, July 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Arduino Based Text Steganography in Color Images

Dr. Riyadh Z. Mahmood, Lecturer,
College of Computer Sc. and Mathematics
University of Mosul/Iraq

Dr. Nasseer M. Basheer, Lecturer,
Technical Engineering College/Mosul
Northern Technical University/Iraq

Abstract--This paper incorporates the LSB (Least Significant Bit) steganography for a text in color images in the RGB (Red Green Blue) format. A certain pseudorandom seed (key) is utilized in both sender and receiver sides, for randomly embedding and retrieving the text bits. The Arduino microcontroller is used to manage all the operations needed, with SDRAM shield to store the image and an LCD display to show required messages and control options. The embedding and retrieving principle used may apply to almost any host image size because it were accomplished on each randomly chosen pixel and does not require loading the image to the Arduino kit. The host image showed quite good PSNR as compared to the original, the PSNR was 75.5% for a 10 bytes embedded text, and gets lower as the text bytes increase to be 55.14% for 100 bytes of embedded text. An image of dimensions of 256x256 is used as a host image. The method is simple, versatile, and efficient.

Keywords-- RGB images, LSB Steganography, random embedding, Arduino, and Arduino shields.

I. INTRODUCTION

Hiding a message in different media files was a well-known procedure, and was used for many purposes ranging from secret messages, to keeping owner rights in a media file he established. The media files may include [1] text, sound, image, or video as a host for the text to be embedded. Host used here is the RGB color images with file records each having one byte of data representing one of the three colors for each pixel, additionally the image file has 54 bytes of image data which comes as the header. The embedding is done by writing the text bits in the LSB of each randomly chosen record or byte of the host image. Arduino is an open source microcontroller with quite promising future due to its programming flexibility and wide range of accessories [2]. The type used is the Mega 2560 kit model among many models with different abilities and applications. The accessories used here are the SDRAM and the GLCD (Graphical LCD) available as shields that may be attached to the kit. Theoretically there are no programming limitations for host image size as long as it fits within the used SDRAM capacity, while the text size is roughly limited to the image size divided by eight. The effect of embedding on the host image is measured by calculating the PSNR (Peak Signal to Noise Ratio) for the image before and after embedding. Many researchers had worked in the field of embedding text bits in the LSB of a host image pixels, their work included a variety of details with a lot of means [3]. Some researchers encrypted, or ciphered the message, or even dealt with it according to its numeric bytes values. Others established different methods for choosing in which cover pixels embedding should occur using random, statistical, or other approaches. Some of these methods also utilized analyzing the cover or arranging it into groups according to its numeric values. Nevertheless the method used here is direct and utilized the pseudo random function as a guide for bits embedding.

II. LSB EMBEDDING

This is a well-known embedding procedure that is commonly used when information is hidden in a host image. It incorporates changing the LSB (Least Significant Bit) of certain host elements by writing one bit of the information in each element [4]. For the condition here the elements meant are each of the three components for each pixel in the host image. As known each pixel is formed of three bytes, one for each of the colors (red, green, and blue). Each byte has a numeric value ranging from 0 to 255. The embedded bit may be a zero or a one, as the original bit to be altered. So embedding may change the LSB value or it may not. If the value is changed the numerical order of the LSB is only 1 as known so it will not have such an influence on the host image. Here bytes selection is done on a random basis considering each pixel component as a possible nominate for embedding. A certain seed (pseudo-random key) is used in both embedding and retrieving to control the random choosing of pixels elements to carry the information.

III. ARDUINO MICROCONTROLLERS

These are open source programmed microcontrollers available with many kits versions which differ by capabilities and comes according to user's projects requirements. Also they are available with a wide range of accessories that are designed to be compatible with it. The accessories include sensors, displays, memory extensions..etc. The type used here is the Arduino Mega 2560 microcontroller kit, with specifications can be noted in [5]. Accessories implemented in this

work are the SD RAM module, and the Graphical Liquid Crystal display (GLCD). Both accessories used are essential. The first to keep the image before and after embedding, with the text to be embedded, and also used for keeping supporting files for the embedding process. While the GLCD is used to display the query messages regarding choosing the image file and the text file, also gives the user certain indications about the stages reached during the embedding process. The programming language used with Arduino is a robust subset of ANSI C (ANSI: American National Standards Institute). This subset which may also be called Arduino C can be referenced in [6], and other books.

IV. TEXT EMBEDDING

In this article the text embedding is to be dealt with. There are certain kit initialization steps that are common between text embedding and retrieving procedures, these will be given here only.

4.1) Kit Initialization Procedure: These are the certain steps used for aligning the controller kit with its accessories and with the computer used. It can be summarized by the following:

1- Including outside libraries such as: SPI; the Serial Peripheral Interface, SD; the attached memory unit, GLCD; the Graphical LCD, and font for the GLCD.
2- Defining the files used by giving its names, types, and lengths. 3-Defining the Baud rate (as 9600 bps) for communicating with the computer.

4.2) Ask the user which (.bmp) image stored in the SD card is to be used as the host. Read the image, Calculate its size, then print on GLCD the image size.

4.3) Remove from the SD card any (h.txt-the histogram file) already existing from previous running of the program. Then establish a new (h.txt) file with length equaling to the image file length less 54 bytes which are the image header which is not to be used for embedding. This is a file with all elements are zeros prepared for the embedding procedure. Then show the message "Histogram Completed" on GLCD with its size printed as well.

4.4) Ask the user to select which text file he wants to embed in the image, among text files available in the SD card. Open text file, calculate its size and print the size for the user to note.

4.5) Pre-extend the text file by four bytes for text file length (in bytes), with keeping the first two bytes of them as zeros if not needed. Then define an internal unsigned variable named "temp" as having all the contents of the new text file. Attaching length of the text file is quite important because it decides at what point the retrieving stops after getting back all text bits.

4.6) Rest of embedding steps are:

a- For each byte in the new text file "temp" do the following:
b- Generate a random pixel location in the image for embedding by using "random" function with a fixed seed which is to be used for all embedding and retrieving purposes.
c- Open image file in SD card for reading and get the pixel value whose location was chosen randomly, and put its value in a variable called "data".
d- Put the text bit in "temp" in the LSB location in "data". Now embedding is done to the pixel temporarily brought from the image file. This is done by using [data=(data&0xFE)|temp%2;] and changing the value of "temp" for each bit in each of the bytes by [temp=temp/2; at end of step g].
e- Close image file as opened for reading purpose.
f- Open image file in SD card for writing. Modify the pixel contents of the same randomly chosen pixel by putting "data" contents in it.
g- Close image file for writing (modify "temp" here).
h- Check for embedding of all bits of the current text byte. If not all embedded go to step (b) otherwise go to step (i).
i- Check for embedding of all text bytes. If there are any byte not embedded go to step (a) otherwise go to step (j).
j- Announce end of text embedding on GLCD as "FINISH" message.

Fig. 1 shows a flowchart for embedding procedure.

V. TEXT RETRIEVING

The procedure used for text retrieving is given here. Kit initialization steps are same as those given in 4.1. Other requirements are:

5.1) Ask the user which (.bmp) host image is to be used for text retrieval. Read the specified image, Calculate its size and print the size on the GLCD.

5.2) Since the same SD card is used for both embedding and retrieval, Remove the already existing histogram file. Then generate a new one also with dimensions of the image less 54 bytes, which represents the header length.

5.3) Remove any already existing file for retrieved texts resulted from previous running of the program.

5.4) As here we do not know the embedded text length so first it is essential to get it from the first four bytes of the embedded text. This requires the following:

a- For 32 bits standing for four bytes repeat the following steps.
b- Generate the random pixel location by "random" function with the same seed number.
c- Open host image file for reading and read the pixel chosen.
d- Put the pixel value in the variable called "data".
e- Get the LSB of "data" and arrange it to form the four bytes representing the text length. This is performed by using the following expression:

[text_size=text_size+(data& 1)*p]
Where p is set to 1 at beginning of step (e), and is incremented by 2 after each time this expression is executed.

f- After getting text size, print it on the GLCD, for the user's information.

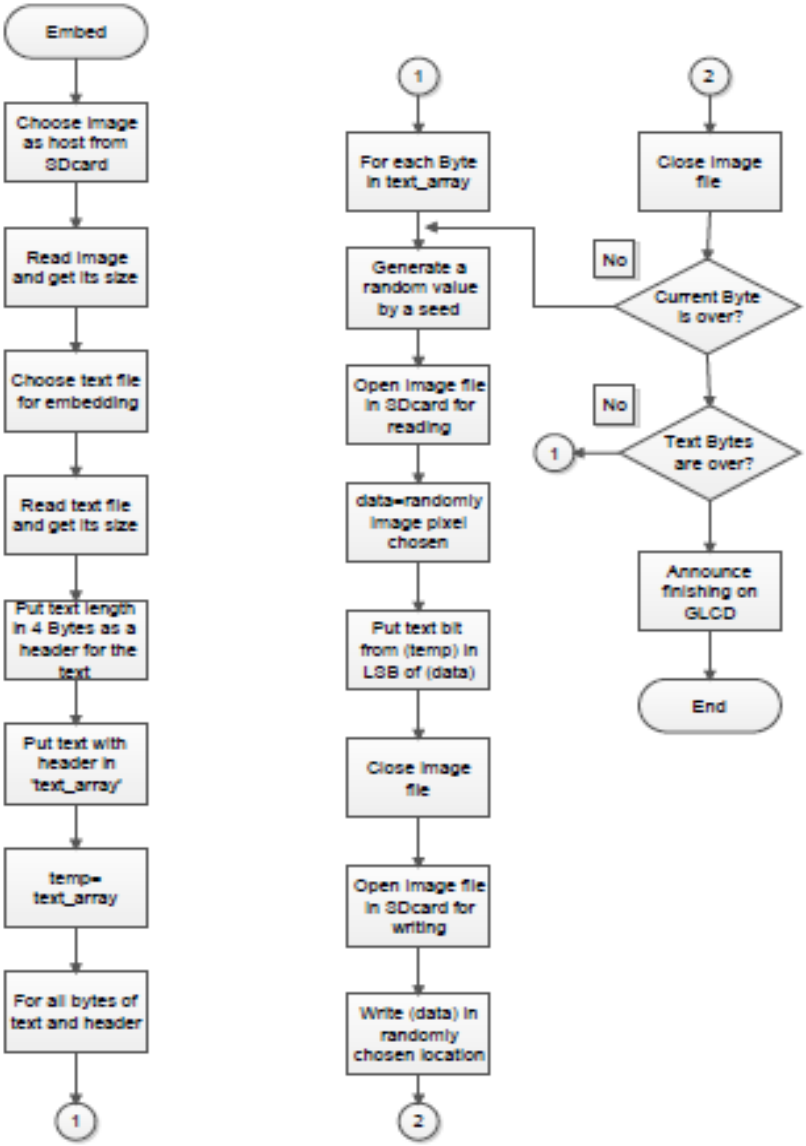


Fig. 1 Text bits embedding flowchart.

5.5) Now text retrieval is performed according to its length in bytes by following the same steps (b to e) given in 5.4 for each byte. With each resulting byte kept in "text_array" variable instead of "text_size" as in 5.4. This is to be repeated for number of bytes as decided by text length got in step 5.4.

5.6) Print the retrieved text on the GLCD display, and print "FINISH" notification.
Fig. 2 shows the flowchart for text retrieval.

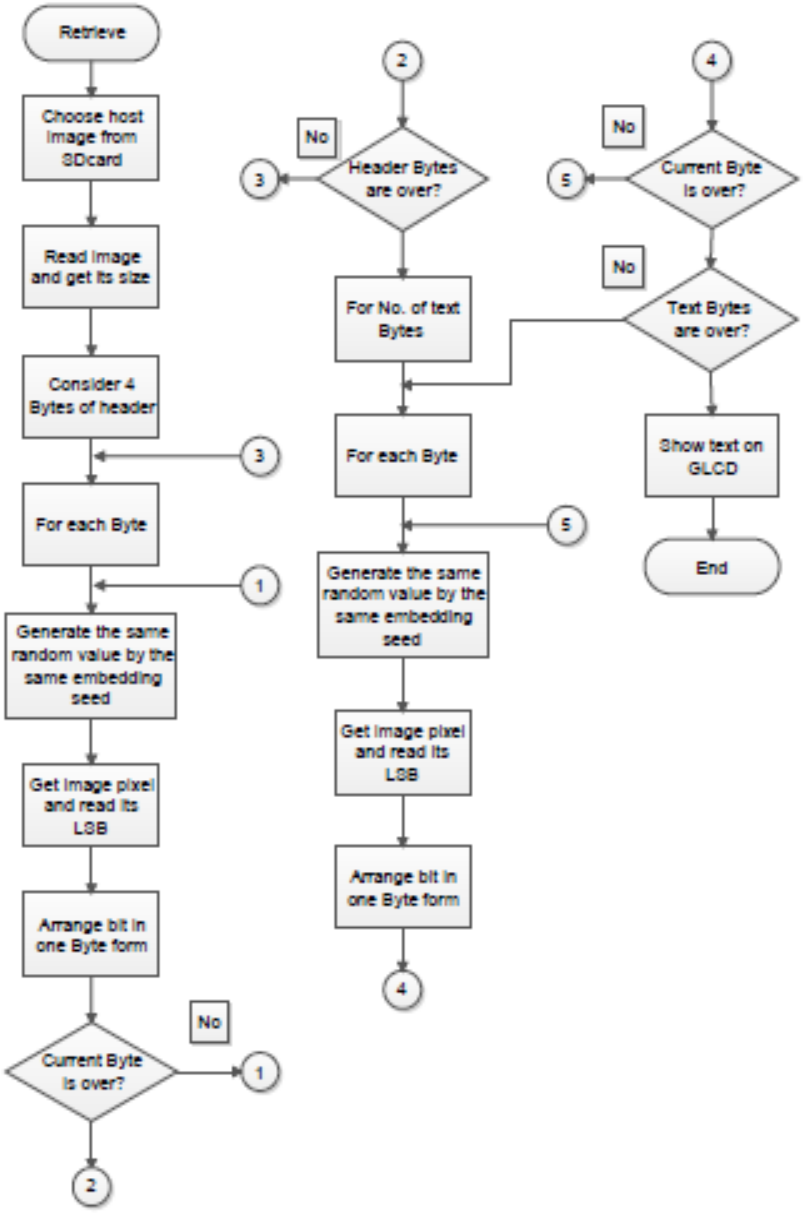


Fig. 2 Text retrieval flowchart.

VI. RESULTS

This article gives the text embedding effect on the host for different text lengths. The standard used for checking host degradation by embedding is the Peak Signal to Noise Ratio (PSNR) for colored images. Which is thought to be quite a good metric for these purposes since it gives the effect of pixels values change due to the embedding procedure. The PSNR for colored images can be given by [7]:

$$PSNR = 10 \log_{10} \frac{(255)^2 \times 3}{MSE_{total}}$$

The MSE_{total} is the summation of the Mean Square Error for the three layers R, G, and B, which is defined as:

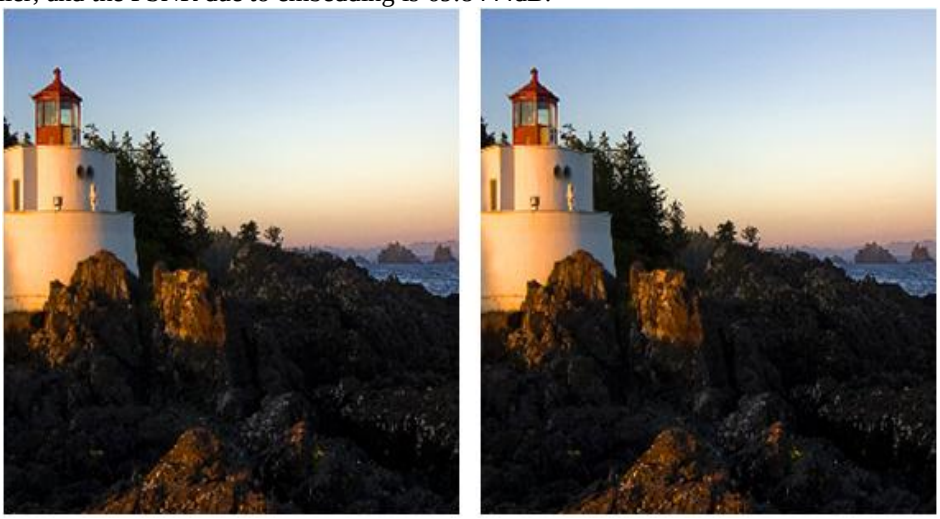
$$MSE_{total} = MSE_R + MSE_G + MSE_B$$

MSE_x: is the Mean Square Error for (x) component which is defined as:

$$MSE_x = \frac{1}{n} \sum_{i=1}^{n} (P_{ix} - Q_{ix})^2$$

Giving summation of all pixels squared differences, between original pixel P_{ix} and retrieved pixel Q_{ix}, divided by the total number of pixels (n) for that component of the image.

Fig. 3 shows the image used as host before and after embedding of a 20 Bytes length text. Image dimensions are 256x256 as mentioned earlier, and the PSNR due to embedding is 69.8444dB.



(b) (a)

Fig. 3 a-Host image before embedding, b-Same image with 20 Bytes message.

The following Fig. 4 gives the PSNR values in (dB) as drawn versus the embedded text length in Bytes ranging from 10 to 100 Bytes, for the same host image.

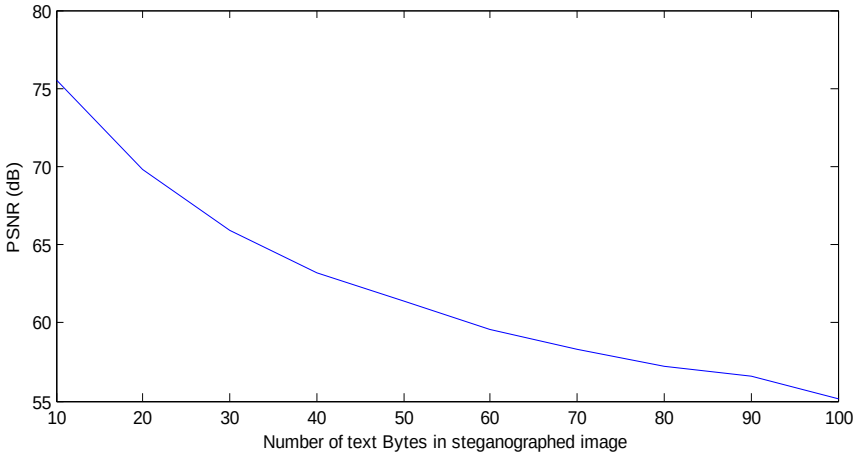


Fig. 4 Degradation of host image quality in Fig. 3, due to variable text message lengths in Bytes, measured as PSNR.

VII. CONCLUSIONS

This paper utilizes the well-Known and effective LSB embedding procedure for hiding a text in a host image. The Arduino microcontroller is used to perform the needed steganography requirements. Host image is an RGB format image with pixel elements arranged as file records, with an 54 byte header which is excluded from embedding efforts. The following points can be noted here:

1) Maximum text length is limited by the number of available bytes with in the host, since in each byte (one element of each pixel) the LSB may embed one bit of the text to be hidden.
2) Host image size has no theoretical limitation as long as it fits in the SDRAM used which should hold the original and modified images. This comes from the fact that the procedure used does not depend on loading the image to the kit which may take a long time and cause memory issues depending on kit hardware. As seen the principle used depends on changing the pixels elements values that were chosen randomly for embedding, also embedding time depends on text length rather than image size.
3) The image quality after embedding is quite good with PSNR ranging from 75.5dB to 55.14dB for embedding of 10 to 100 bytes respectively for a certain 256x256 image . As known the human eye does not recognize any difference in quality for PSNR values of 40dB and above [8]. These values of PSNR vary when the text, host image, and (or) seed value are changed. Also PSNR becomes less for the same text length as the host image dimensions are raised because the effect of the total mean square error will be less.
4) The histogram file (h.txt) is used for assuring that no bit is miss embedded, or retrieved by the pseudo random function. By writing a one in each location representing any chosen pixel element and checking on this basis. This is not given in the flowcharts for simplicity only.
5) This application is an example of the effectiveness of using the Arduino microcontroller for dealing with standard principles already known and used in the field of image processing and data hiding.

REFERENCES

[1] T.Morkel, J.H.P.Eloff, and M.S.Olivier; "An Overview of Image Steganograpy", in HS Venter, JHP Eloff, L Labuschagne and MM Eloff(eds), Proceedings of the fifth Annual Information security South Africa Conference (ISSA2005), Sandton, South Africa, June/July 2005.

[2] Martin Evans, Joshua Noble, and Jordan Hochenbaum;"Arduino in Action", Manning Publishing Co. 2013, ISBN:9781617290244.

[3] Matma Jain, and Saroj Kumar Lenka;"A Review of Digital Image Steganography using LSB and LSB Array", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 11, Number 3 (2016) pp 1820-1824, © Research India Publications. http://www.ripublication.com.

[4] Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom, Jessica Fridrich, and Ton Kalker; "Digital Watermarking and Steganography", second edition, Elsevier Inc., 2008, ISBN: 978-0-12-372585-1

[5] John Boxall;"Arduino Workshop, A Hands-on Introduction with 65 Projects", no starch press, 2013, ISBN-10: 1-59327-448-3, ISBN-1978-1-59327-448-1.

[6] Jack Purdum; "Beginning C for Arduino", Apress, 2012, ISBN-13(pbk):978-1-4302-4776-0, ISBN-13(electronic): 978-1-4302-4777-

[7] Bibhas C. Dhara and Bhabatosh C., " Color image compression based on block truncation coding using pattern fitting principle", the Pattern Recognition Society, 2007.

[8] WALKER, James S., and NGUYEN, Truong Q.;" Wavelet Based Image Compression, the sixth chapter of The Transform and Data Compression Handbook"; edited by RAO, K.R. and YIP, P.C., CRC Press, ISBN-0-8493-3692-9, 2001.

