Stego-tool

ECI-2019

# Introduction

Steganography is the study and application of techniques that allow to hide messages within other objects, establishing a cover channel that prevents the very act of communication from being noticed by external observers.

Images are commonly used as cover objects due to the relatively high amount of data they contain and some other properties that will be detailed later, making them ideal candidates for hiding information without notice. In the present article, the development of an algorithm for injecting text within this type of files and the results obtained from its execution on many examples will be discussed.

The image format chosen corresponds to the BMP file format, which stores digital images as a bitmap. Each pixel is represented by a fixed number of bits (may it be 1, 2, 4, 8, 16, 24, 32 or more). This array of bits is most commonly arranged in RGB format, establishing the beginning bits as the red value of the image, the middle ones as its green value and the last ones as its blue value.

The human eye is not as responsive to blue as to the other two colours, making the blue component of a BMP file a good candidate for hiding information within it. That lead to the decision to implement an algorithm that given a picture in BMP and a text encoded using ASCII, it replaces the least significant bits of the image (which correspond to its blue component) with the bits that represent the text.

## Development

A requisite for the programme developed was for it to be able to receive a maximum number of least significant bits (k for here on) which could be used to hide the message.

The algorithm implemented takes the value of k and checks if it will be enough to hide the message within the pixels of the given message; if not, then k is changed to the minimum value needed. k can have a value from 1 to 8 (a greater number could potentially take more bits that what are given to the blue component). If the text can be properly inserted in the image, then the number k is hidden within its first three pixels, using one bit from each one.

After that, the length in characters of the message is stored as a character in itself, using the same algorithm as for the rest of the message. This allows text sizes of up to 256 characters (as only 8 bits are used to represent them).

The algorithm for hiding a character is pretty simple: the bits of the character are taken in chunks of k bits, which are then stored in the k least significant bits of some consecutive pixels. If the last chunk is of a size less than k, due to an uneven size difference within the size of a character and the k chosen, then only the needed less significant bits of the next pixel will be used.

Before inserting the next character, there is the possibility of a pseudo-random padding being introduced. First, for adding a padding at all the binary exclusive or (xor) operation between the amount of pixels left available for padding (its total is calculated at the beginning of the hiding algorithm) and the last pixel used for steganography must result in a 1 in the least significant bit. If this condition is fulfilled, then the jump is calculated as follows:

% representing the calculation of the remainder of a division between integer numbers and the char size equalling 8, as ASCII is used as the encoding.

The procedure for recovering the hidden message is straightforward: after extracting k and the text size from within the first pixels of the image the reverse algorithm for inserting a character must be applied. Also, the calculation of the padding jumps is executed in the very same manner. This results in that the recipient of the message only needs to know this algorithm for recovering it, as the original image has no more porpurse after the text has been hidden.

Ultimately, for the development of the so-called “stego-tool” the programming language C was chosen because of the low level facilities that it provides. Additional modules were created to provide functionality for extracting and storing the pixels of images in BMP format and for calculating certain measures between an original image and its resulting copy after applying steganography on it, which will be explained in more detail in the next section.

## Procediment

In order to test the tool and get results about the performance, several images were selected and classified into four groups, depending on the theoretical difficulty in hiding messages without notice. These groups are: open nature (landscapes), portraits, still lifes and text.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| ***Figure 1:*** *Open nature (landscape) image examples* | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| ***Figure 2:*** *Portrait image examples* | | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| ***Figure 3:*** *Still life image examples* | | |

|  |  |
| --- | --- |
|  |  |
|  |  |
| ***Figure 4:*** *Text image examples* | |

Each picture was used to hide messages, varying the k parameter and the length of the text to hide. For each time, it was calculated three values that represent the effectiveness of the algorithm hiding the given text. These are: the mean squared error (MSE), the peak signal-to-noise ratio (PSNR) and the structural similarity (SSIM).

## Results

The tables below show the worst values obtained for each image of the four groups taken.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 1:*** *Open nature images worst values* | | | |
| **#** | **MSE** | **PSNR** | **SSIM** |
| 1 | 13329 | 103.25 | 0.999994 |
|
| 2 | 3147 | 109.51 | 0.999999 |
|
| 3 | 3640 | 157.05 | 0.999998 |
|
| 4 | 9629 | 104.66 | 0.999996 |
| 5 | 8949 | 56.81 | 0.999996 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 2:*** *Portrait images worst values* | | | |
| **#** | **MSE** | **PSNR** | **SSIM** |
| 1 | 92615 | 94.79 | 0.999957 |
|
| 2 | 9794 | 104.58 | 0.999995 |
|
| 3 | 5541 | 107.06 | 0.999997 |
|
| 4 | 4604 | 107.26 | 0.999998 |
| 5 | 6529 | 106.34 | 0.999997 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 3:*** *Still life images worst values* | | | |
| **#** | **MSE** | **PSNR** | **SSIM** |
| 1 | 6180 | 106.59 | 0.999997 |
|
| 2 | 5353 | 107.21 | 0.999998 |
|
| 3 | 23310 | 100.14 | 0.999989 |
|
| 4 | 15476 | 102.43 | 0.999993 |
| 5 | 40332 | 98.43 | 0.999981 |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 3:*** *Text images worst values* | | | |
| **#** | **MSE** | **PSNR** | **SSIM** |
| 1 | 3541 | 109.00 | 0.999998 |
|
| 2 | 3 | 139.72 | 0.999999 |
|
| 3 | 14638 | 102.84 | 0.999993 |
|
| 4 | 4 | 138.47 | 0.999999 |

|  |  |
| --- | --- |
|  |  |
| ***Figure 5:*** *Graphics of MSE and PSNR vs length of hidden text in the text image #3* | |

|  |  |
| --- | --- |
|  |  |
| ***Figure 6:*** *Graphics of MSE and PSNR vs length of hidden text in the portrait image #3* | |

# Conclusions

After testing the programme with many different images, variable messages and various values of k, the following conclusions were reached:

* Every message correctly hidden in an image was properly recovered.
* The pseudo-random padding generates localized jumps; in other words, the spaces between adulterated pixels are relatively small. This results in the text been inserted in the lower border of the screen, as the pixels are actually counted starting from the bottom left corner of the picture.
* The best results were unexpectedly obtained from text images. These are usually the worst performing in steganography due to having very little difference in its content. In these cases, it was clearly observed how the MSE increased as k or the length of the text did so. Also, the PSNR decreased when those values raised.
* The MSE of the images of the other categories tends to concentrate around a certain value or be close to zero. This last behaviour could be caused due to the combination of least significant bits of the images being similar to the bits representing the text to hide.
* The values of SSIM obtained all across the board were extremely good. This could be caused due to the small length of the text hidden in comparison to the size of the images used. Also, the previously discussed location of the adulterated pixels could be a factor in this.

A possible modifications to be made in the future is to increment the amount of bits used to store the length of the text from 8 to 16 (2 bytes or characters), which would allow messages of up to 65536 characters.