Czech Technical University in Prague

Faculty of Electrical Engineering

Semester project

Steganographic tool for Matlab

Tomáš Dlask

This project was done during the A0B17MTB course at FEE CTU as a semester work to practice programming in Matlab.

June 2015

Contents

Introduction

Steganography[1] in general is a method of hiding information from third parties. A comparison with cryptography is inevitable. Even though both methods might seem similar, the principle of security differs. In cryptography, third parties may know that an encrypted message is being sent; therefore it is based on the impossibility (or difficulty) of decoding. On the other hand, steganography is focused on the undetectability. The third party should not know that there is a hidden message. The "opposite side" of steganography is called steganalysis, which targets the detection.

Digital media, for instance images or video files, can be utilised to store hidden information. This work will be mainly focused on storing data in an image and trying to avoid its detection.

Storing general data in an image

A regular picture taken with present-day digital camera consists of three colour layers – red, green and blue. Each pixel is described by a triad of whole numbers in the range from 0 to 255, which corresponds to 8 bits. Therefore, $2^{3 \cdot 8} \approx 16$ million colours can be distinguished in each pixel.

Human vision, unable to differentiate such amount of colours, enables us to replace the least significant bit (LSb) to store a hidden message. The principle is very straightforward, as is shown in the table below. For the sake of simplicity, the image structure is simplified to a sequence of 8-bit numbers.

Original data (decimal)	125	84	222	185
Original data (binary)	01111101	01010100	11011110	10111001
Message to be hidden	1	0	1	0
Data with hidden message (binary)	01111101	01010100	11011111	10111000
Data with hidden message (decimal)	125	84	223	184

Storing the message in the LSb alters the parity of the original data and reduces the overall number of colours to $2^{21} \approx 2$ million, which is still enough. Even when using two LSbs and comparing the original with the altered image, the difference cannot be seen by humans without the assistance of graphical software.

Reducing the colour scale smartly

A method of storing a general "message" in an image was introduced and will be used to store one bit of information per 8 bits (corresponds to one colour layer of a pixel) of the covering image. Now, possibilities of storing an image will be discussed. The easiest way is to replace the LSb of the covering image with the most significant bit (MSb) of the hidden image. But, the quality of the hidden image will deteriorate to a considerable extent.

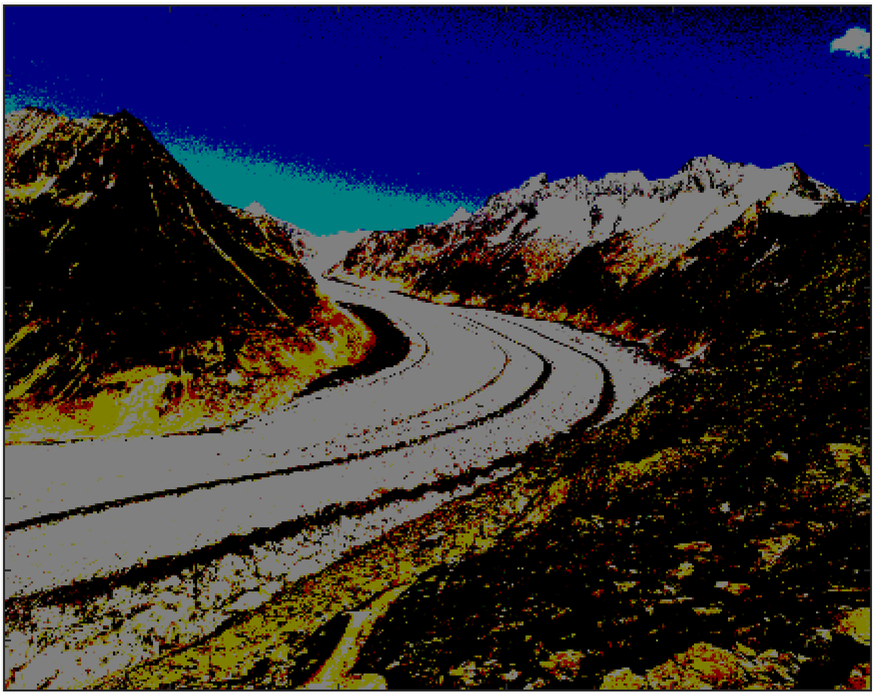


Image 1: The MSb of an image

Fortunately, there is a better way to utilise the LSb that is based on another imperfection of human vision. Imagine a picture where only white and black pixels alternate half-and-half. If this picture is seen from a far enough distance, it would seem grey. If the picture contained three times more white pixels than black ones, it would appear brighter. Thus, still using only one bit, more colour shades can be perceived. The process of creating such image is called dithering.

This method of altering an image is based on "distribution of error". Simply, if one pixel had the value of 100 (40% grey) and was assigned 0 (black), the neighbouring pixels receive this information and will tend to become brighter to compensate this error. Dithering is already implemented in Matlab, thus there is no need to describe the algorithm precisely.

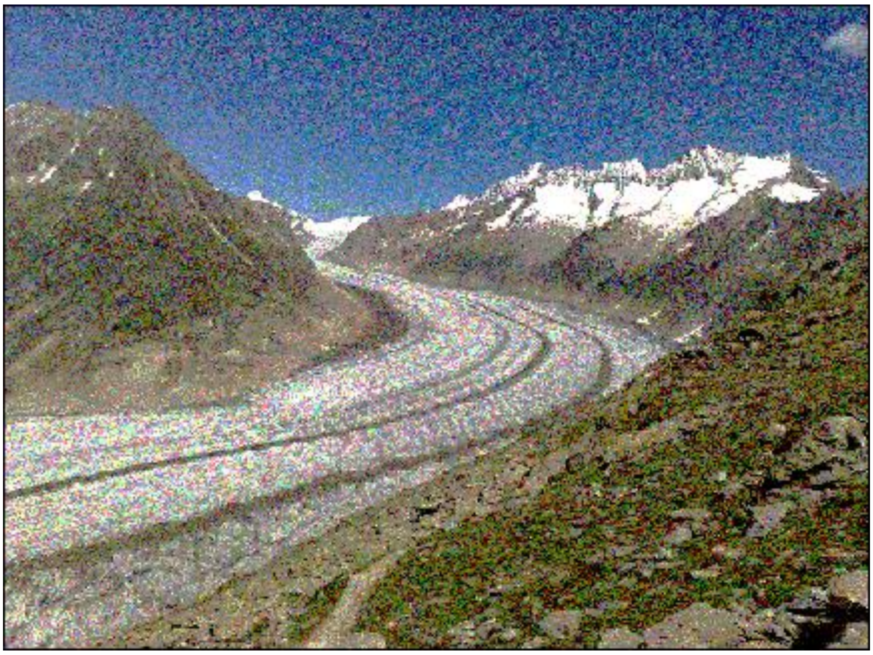


Image 2: Dithered coloured image creates the illusion of more colours.

The effects of dithering on black and white image are presented below. Dithering is substantially better than simple extraction of the MSb. Right-upper corner is dithered image, left-lower is MSb, right-lower is similar to MSb, but the threshold between black and white is not set to 128, but 224 to feature more black pixels.

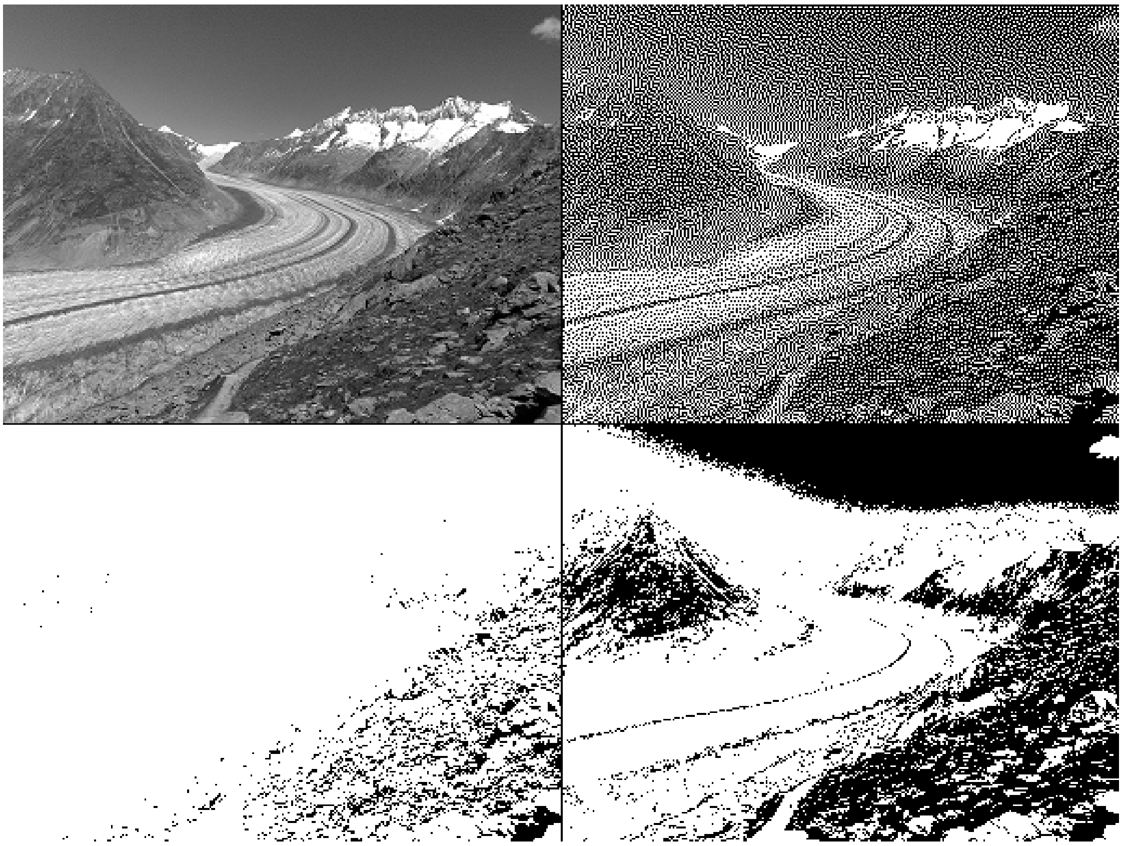
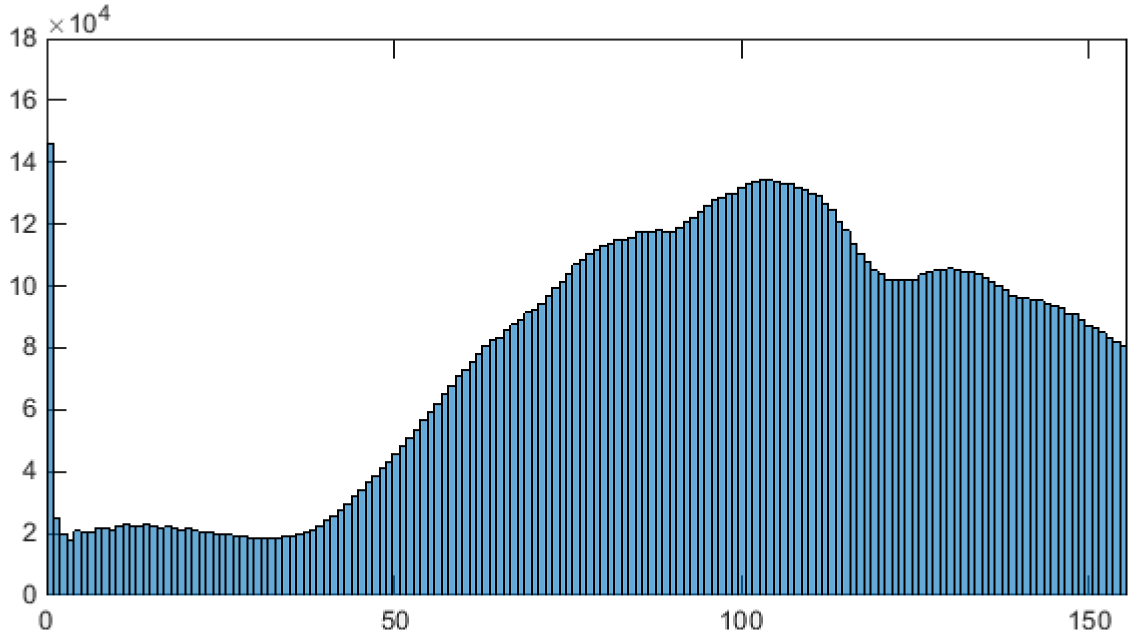


Image 3: Comparison between methods of converting a greyscale image (left-upper) to black and white.

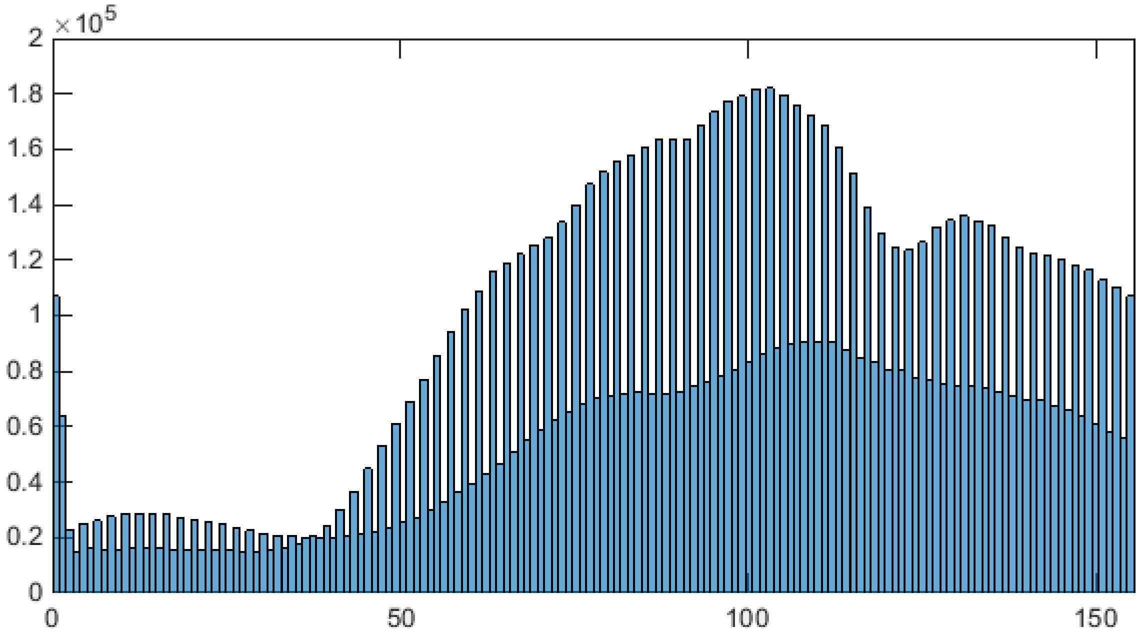
The basic version

After this introduction, it is possible to store an okay-looking image in the LSb of another image. First of all, it would not be safe at all to store the image directly, because these methods are well-known and the information is not obscured in any way. Secondly, the detection is very easy just by looking at the histogram of the image.

To demonstrate this phenomenon, parts of histograms of a typical image and the same one with hidden message are provided.



Graph 1:Part of histogram of a typical image.



Graph 2: Part of histogram of an image with hidden information in the LSb.

The first histogram is continuous as it should be, but the second one reveals that something is suspicious about the image. The explanation is logical: the hidden image was not balanced, 1 occurs in the LSb more often than 0, which leads to these deviations in the histogram. Also, if the histogram was divided into two graphs - one with the frequencies of odd numbers and the other with frequencies of even numbers, then even after linear scaling, the graphs would not overlap, because the brightness of particular parts of the hidden image does not match the corresponding parts of the covering image. A more elaborated process must be applied.

Balancing the histogram and encoding the image

According to the findings in the previous paragraph, the image cannot be stored directly and must be altered in order to balance the histogram. In my implementation, magic matrix was utilised for this task. Generating a large magic matrix is fast and deterministic in Matlab[2]. In addition, odd and even numbers occur half-and-half in the matrix, allowing the “mixing” to be efficient.

To demonstrate the principle, let S be a square matrix containing zeros and ones representing a black and white image that needs to be stored. Next, M will be the magic matrix of the same size. Note that both matrices are square, but that is allowable, as the process can be easily generalised by using reshape function (as implemented) or enlarging the image along shorter dimension to form a square. Finally, M_2 matrix is defined as M mod 2 (element-wise remainder after division by 2). Then, following equations hold:

$$E = (M_2 + S) \bmod 2 \text{ … the encoded image}$$

$$S = (M_2 + E) \bmod 2 \text{ … decoding the original image}$$

Also, this process can be easily generalised to 2 bits if the numbers in S are also 2-bit.

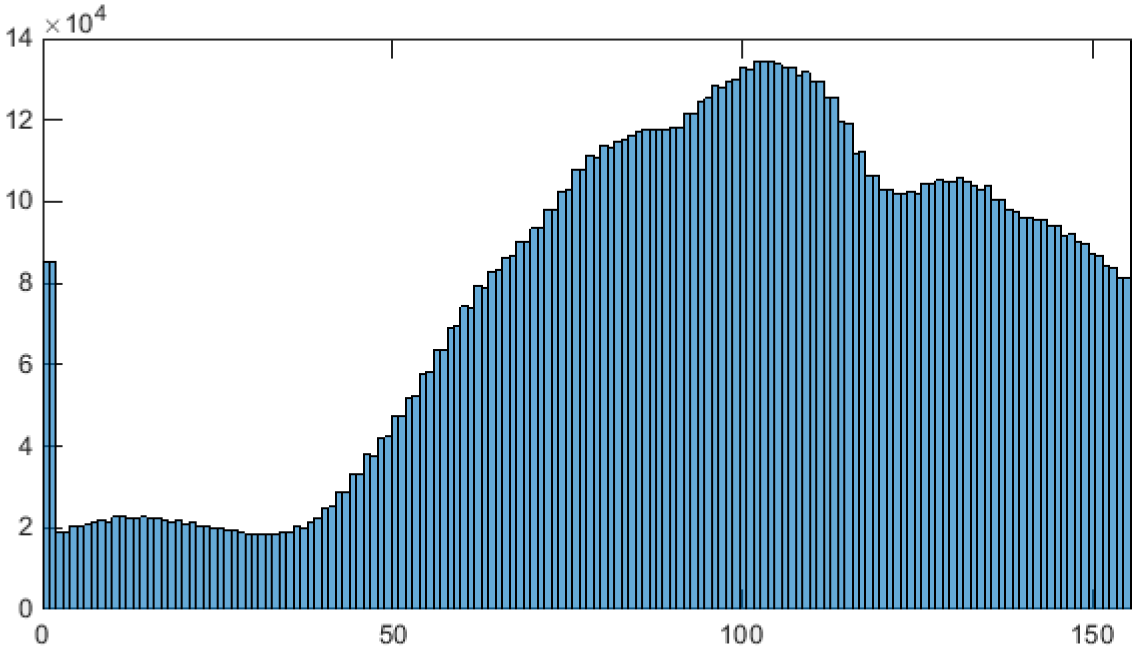
$$M_4 = M \bmod 4$$

$$E = (M_4 + S) \bmod 4$$

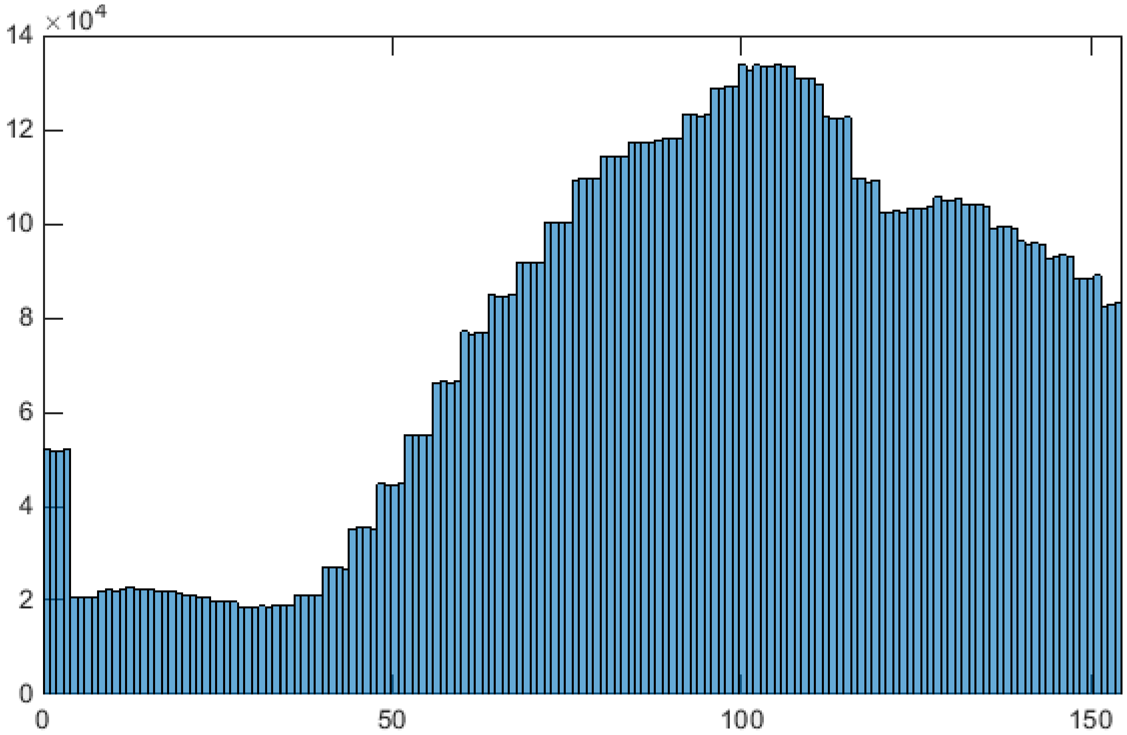
$$S = (3 \cdot M_4 + E) \bmod 4$$

These formulas are easy to comprehend: using the M_N matrix, the values of S are circularly shifted. And to decode it back, it is necessary to complete the whole circular shift.

This process may serve as a basic encoding that varies with the size of the image, yet another improvement could be employed – password protection. If the image *S* (or encoded *E*) is viewed as a sequence of numbers (e.g. using linear indexing) and another (usually significantly shorter) sequence of numbers (password) is given, Vigenère cipher[3] can be employed. The previously mentioned encoding equations also represent a special case of this cipher. Now, after encoding, the histogram of the image with hidden information is more continuous.



Graph 3: Part of histogram of an image with hidden information in the LSb after encoding



Graph 4: Part of histogram of an image with hidden information in the 2 LSbs after encoding

After inspecting the graphs, the balancing procedure was a success. The corresponding pairs of odd and even numbers that differ only in the LSb have roughly the same cumulative frequencies. The same result is seen in the case of replacing last two LSbs, where foursomes of numbers share the same frequencies.

But another relevant feature can be observed, the groups of “same frequencies” created “stairs” in the graph. This feature is not as apparent as the discontinuous graph in the previous case; nevertheless it is still detectable with ease. For example, the Pearson’s χ^2-test could confirm that frequencies of even and odd numbers share the same probability distribution. The same could be applied on the histogram of the image with 2 LSbs replaced, where four distributions would be tested.

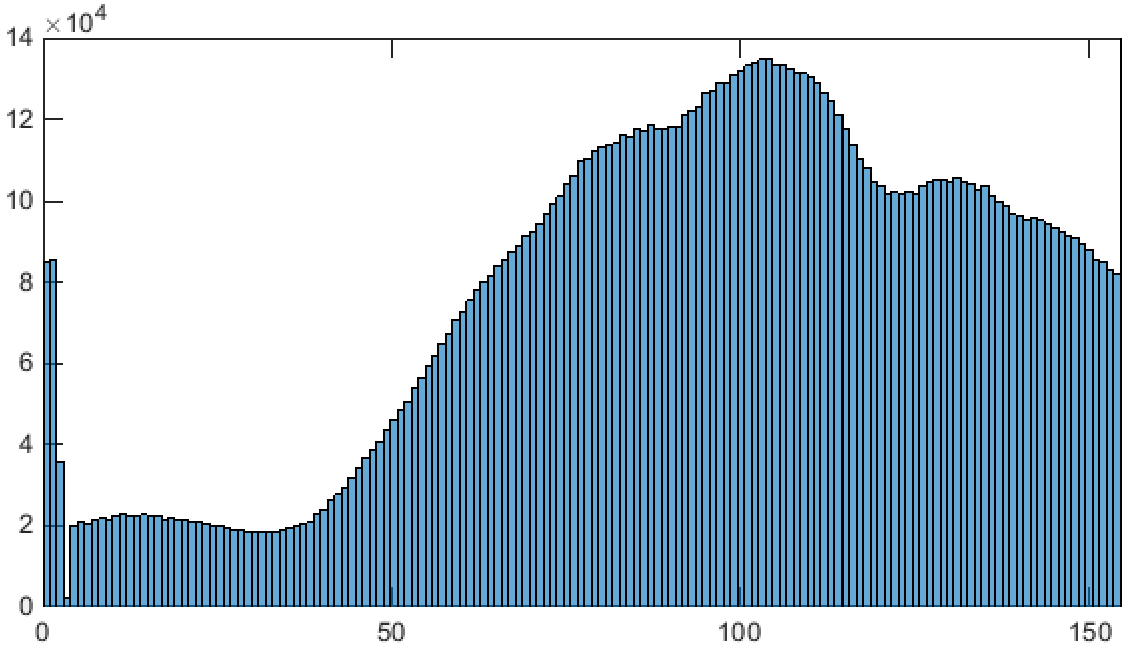
Smoothening the “stairs”

Developing a better “mixing” matrix than magic matrix could lead to the goal, but its values would have to depend on the corresponding 7 MSbs and respect the individual parts of histogram. This approach would be beyond the extent of this assignment.

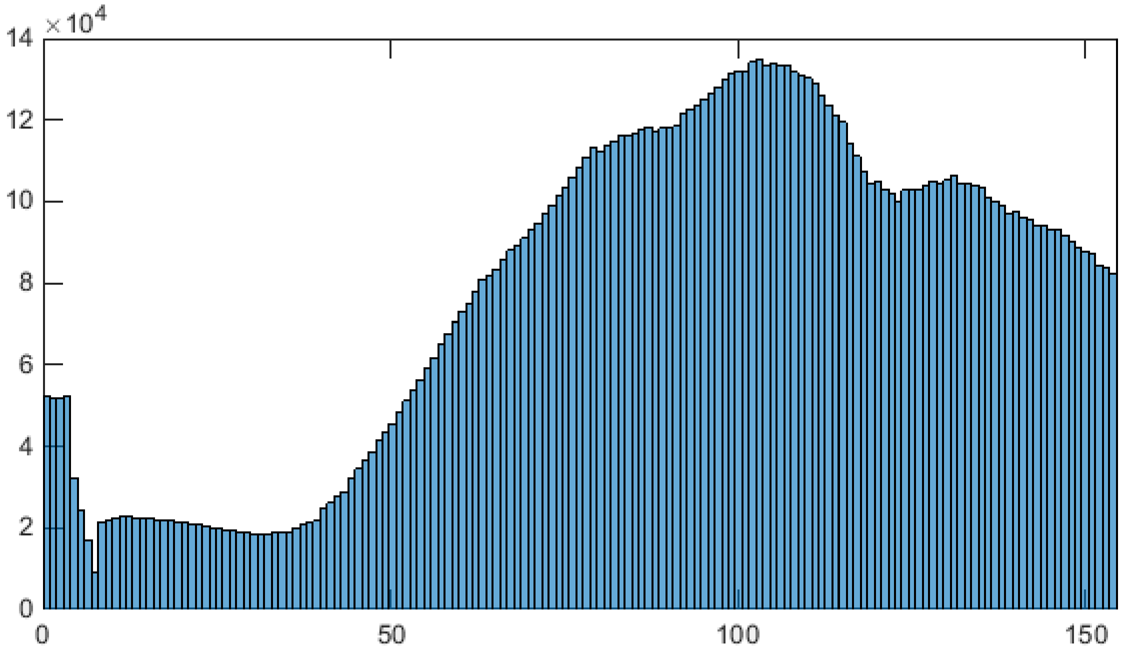
An easier method is simply replacing some “odd colours” with “even colours” and vice versa to smoothen the histogram. This idea might sound unreasonable at first, because the hidden information is stored in the parity of the colour; therefore the information itself is irreversibly destroyed. But after further analysis, the differences that will be done are usually negligible and affect the stored image only slightly.

Smoothening is applied to those colour levels that are found in the increasing or decreasing parts of histogram. Local maxima and minima are left as they are. If a pair is in the increasing part, some even values will be replaced with odd values to adapt to the trend. On the other hand, in decreasing parts, odd values will be replaced with even values. In the case of 2-bit storing, the changes are distributed through whole foursomes.

Note that until now; what was saved in the LSb was also extracted unchanged. If smoothening is applied, this statement no longer holds and it is recommendable to check how smoothening alters given images to decide whether it is applicable in particular cases.



Graph 5: Smoothened graph 3



Graph 6: Smoothened graph 4

It is obvious that smoothening was successful in removing the stairs; the only problems are at the edges of spectrum, where the values would have to be changed considerably.

Examples of the quality of stored images

The results of dithering and the changes in quality caused by smoothening are depicted on the following images. These images consist of 8 colours only and were stored in the LSb.

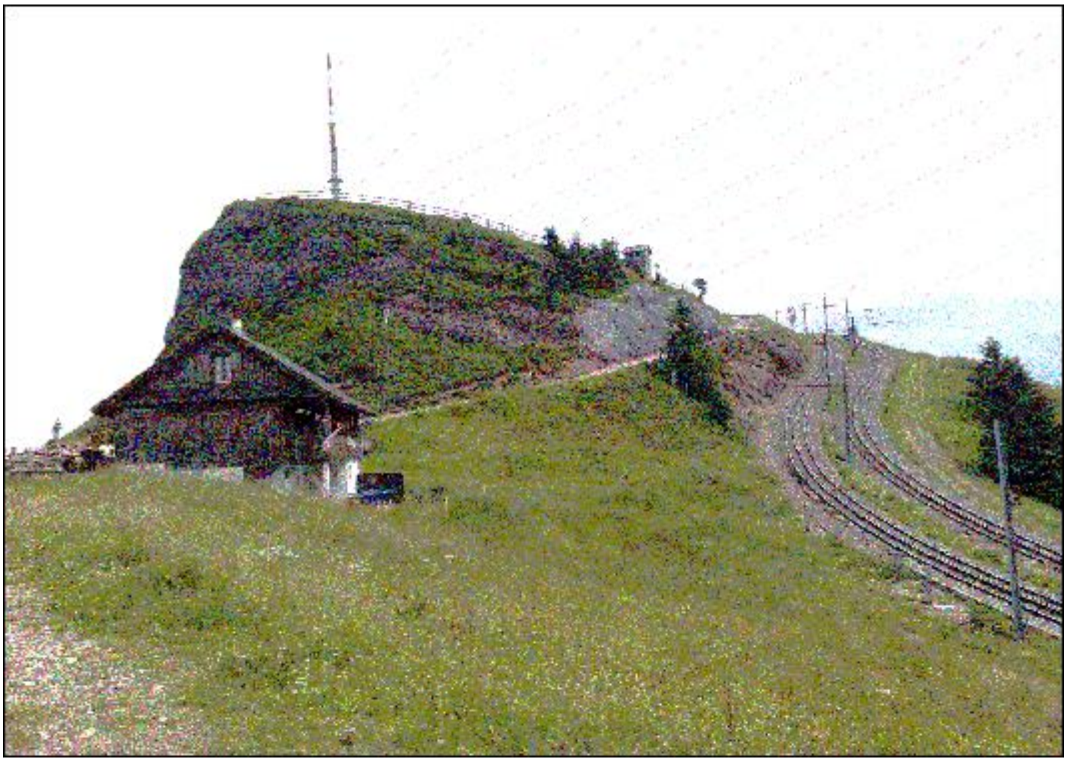


Image 4: Extracted image (stored without smoothening)

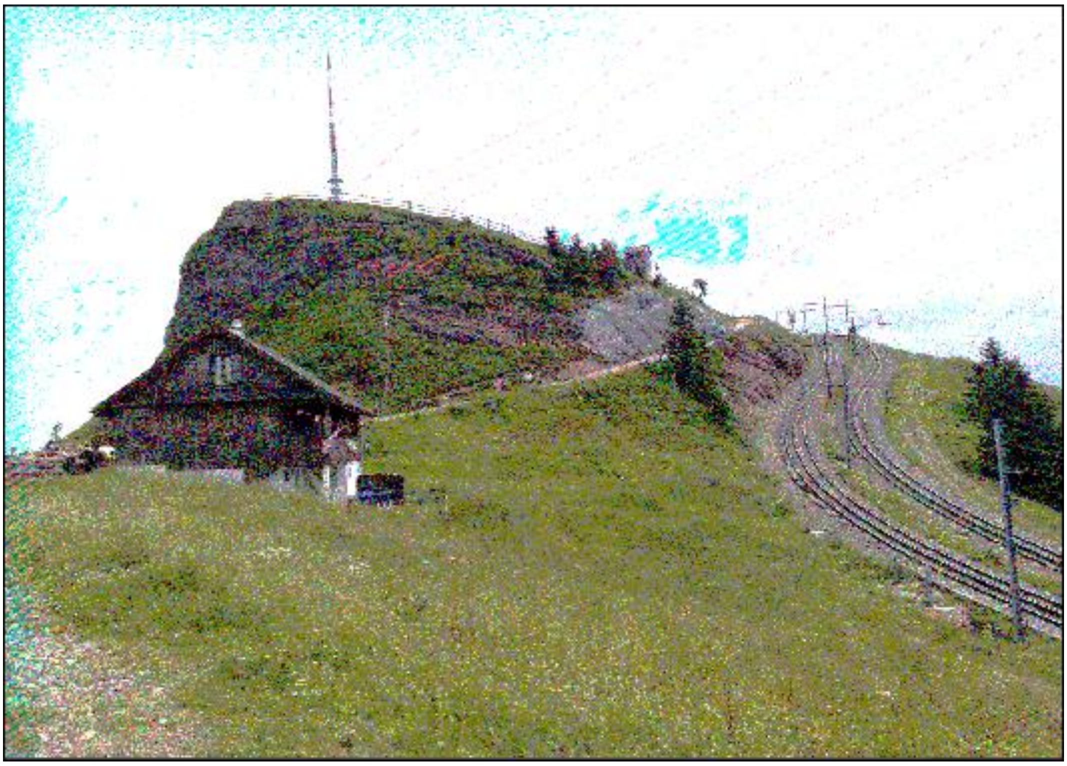


Image 5: Extracted image (stored with smoothening)

These images were stored in two LSbs (64 colours).

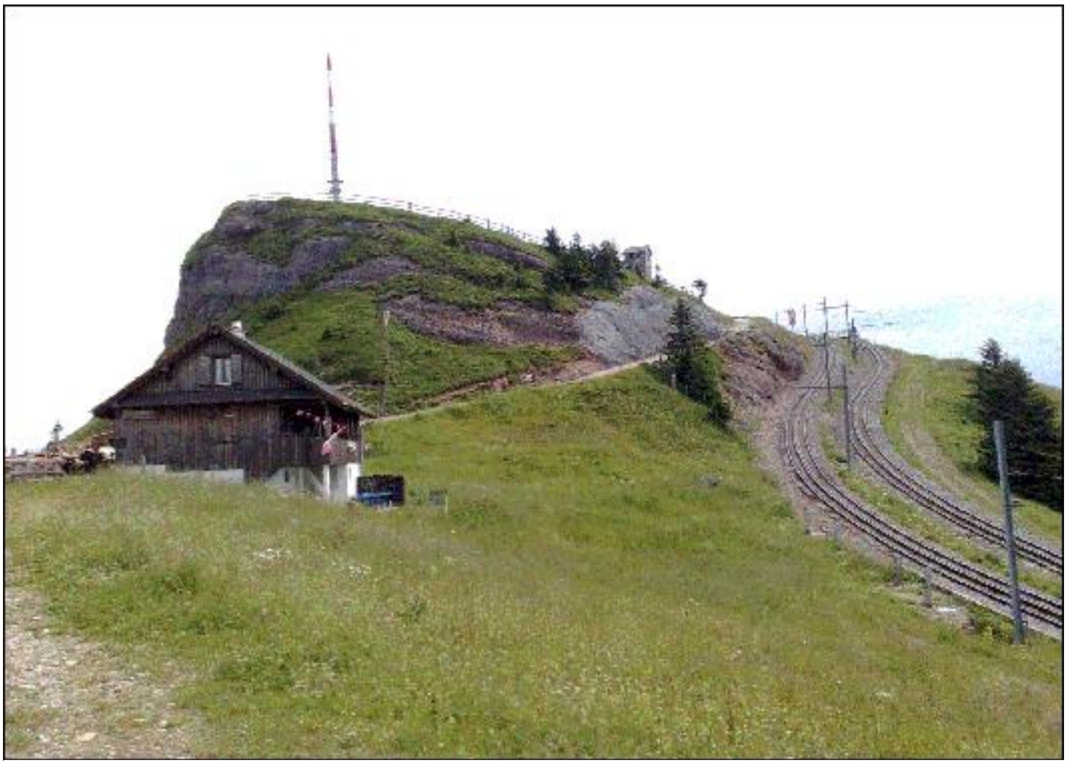


Image 6: Extracted 2-bit image (stored without smoothening)

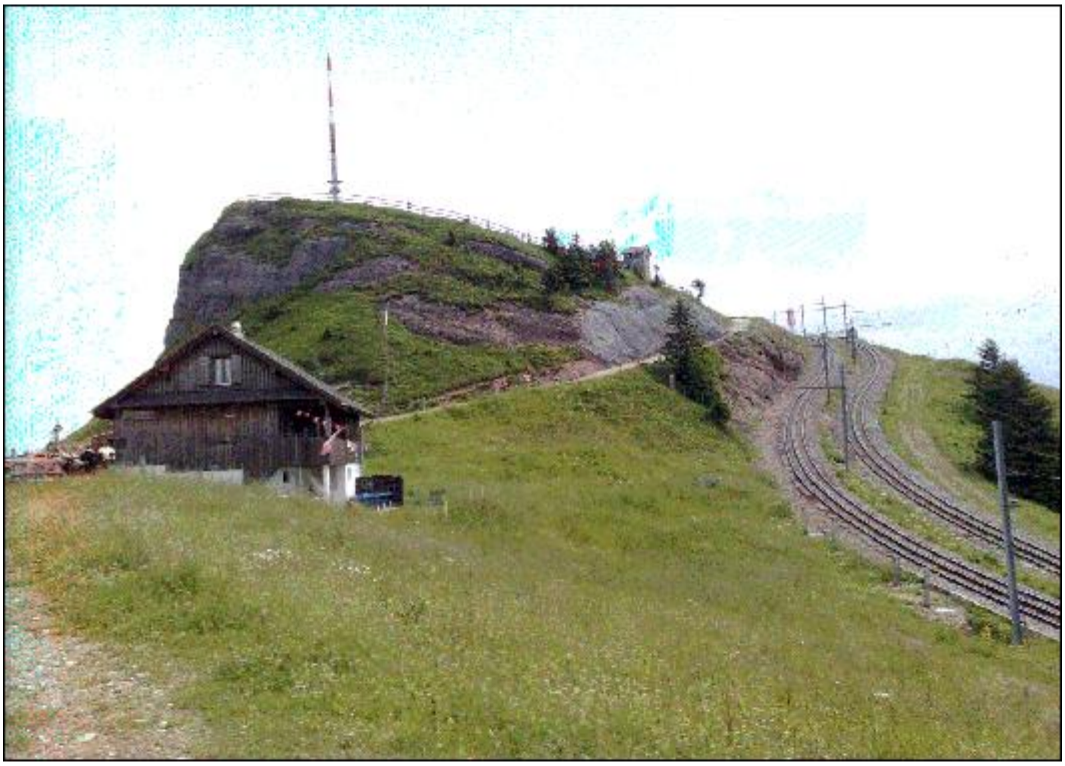


Image 7: Extracted 2-bit image (stored with smoothening)

Finally, these images consist of 2 colours only, one cover (carrier) image could contain up to three images of this quality hidden in each of its colour layers.

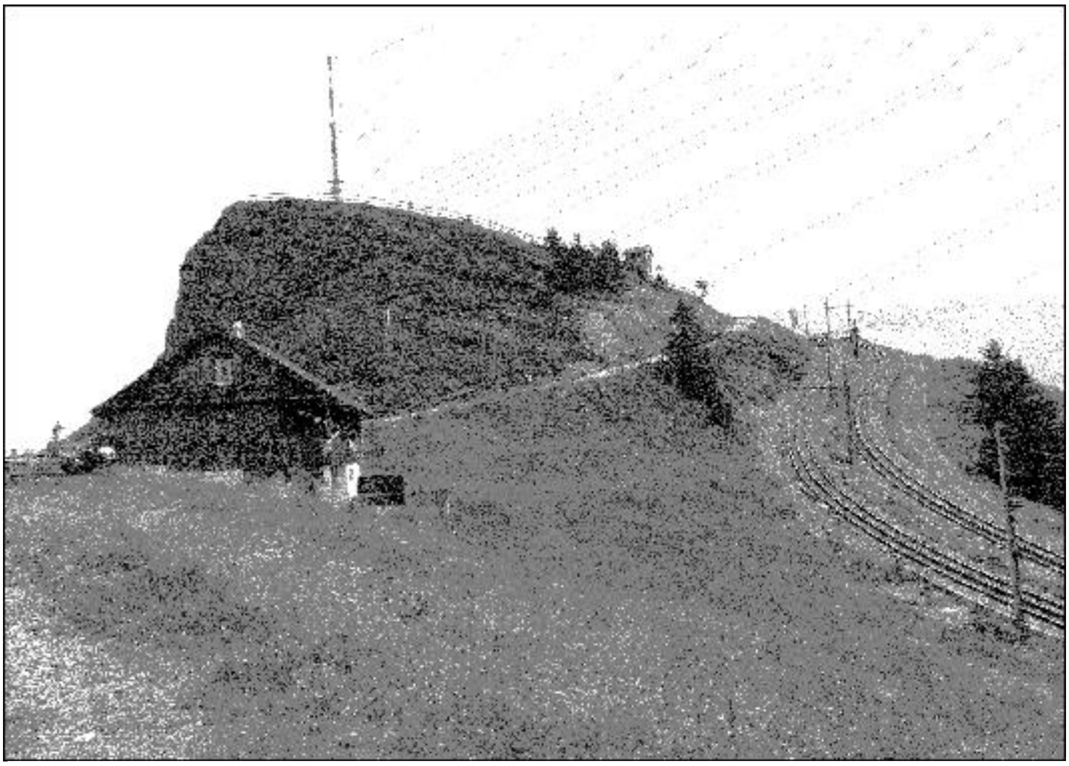


Image 8: Extracted black and white image (stored without smoothening)

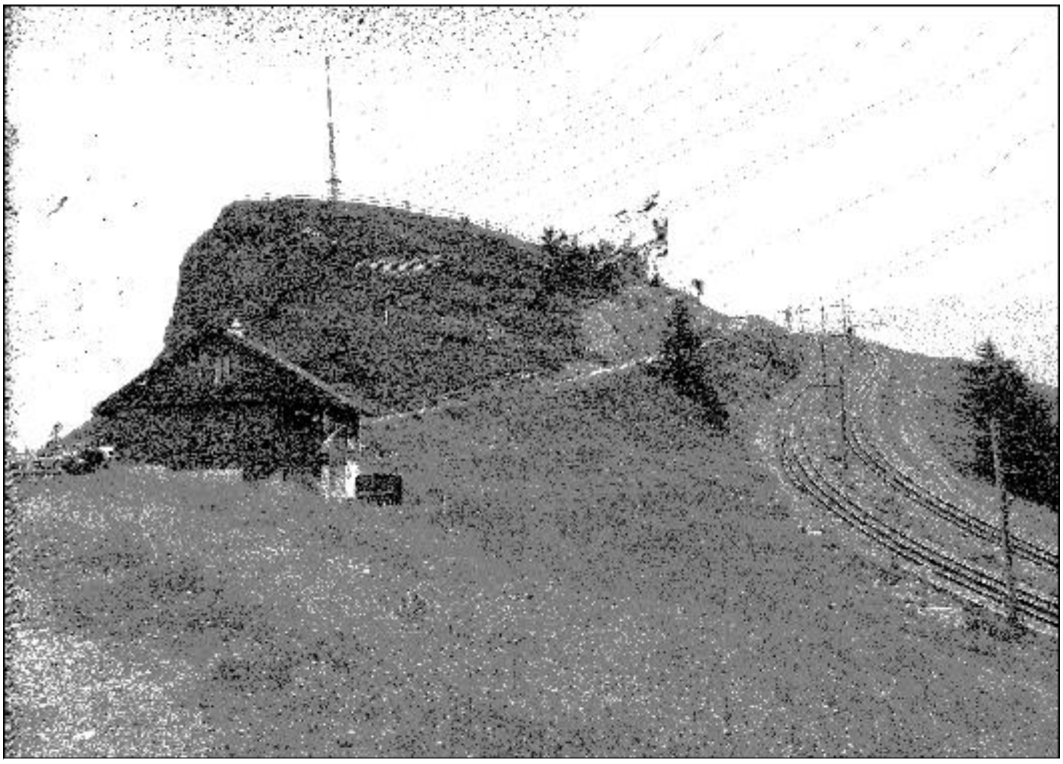
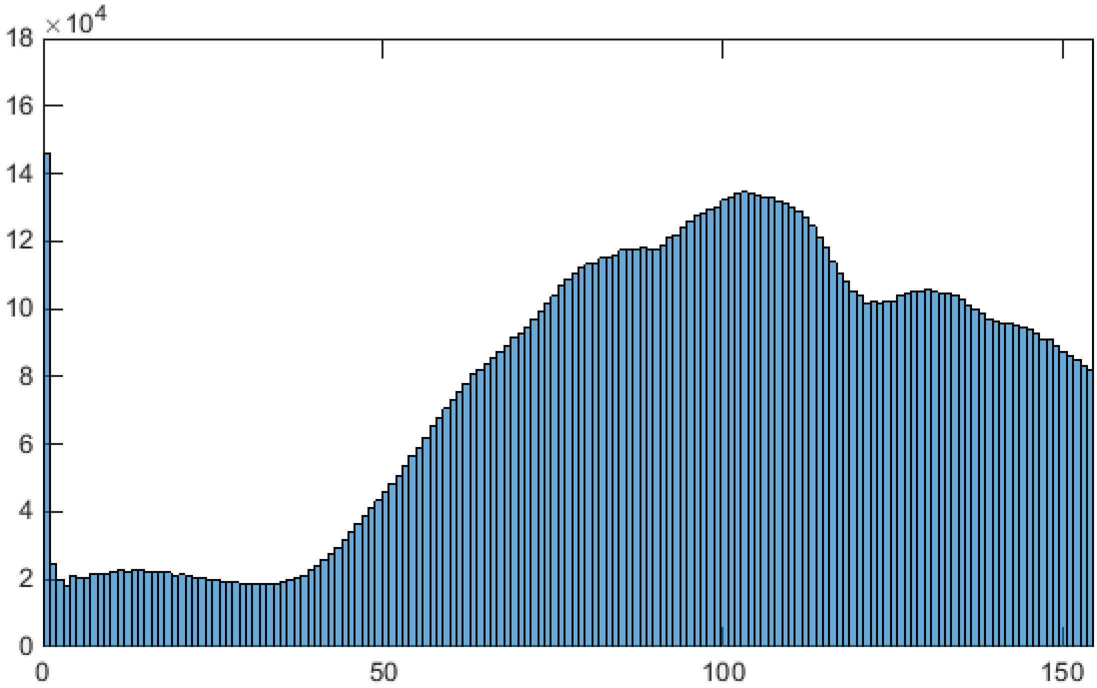


Image 9: Extracted black and white image (stored with smoothening)

Storing text

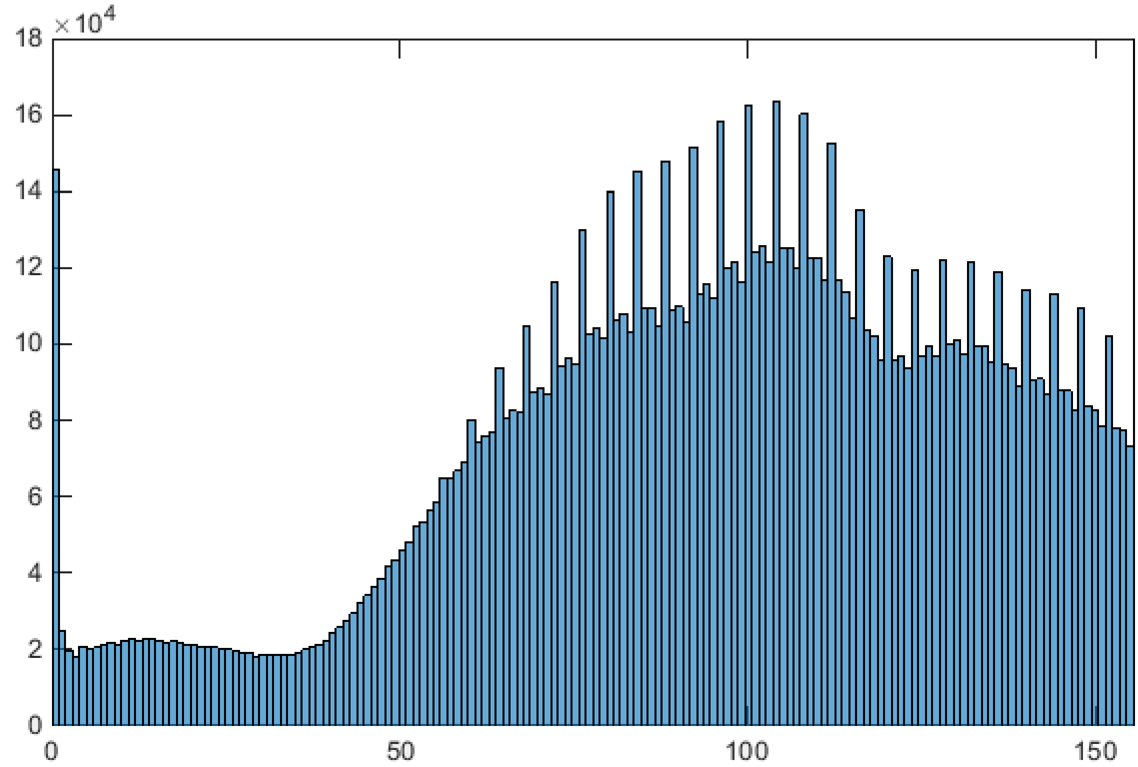
Functions for storing text in an image are a marginal part of this work. Text in Matlab is represented by a (usually row) matrix of characters. Each character takes 2 bytes (16 bits) to be identified, therefore if 2 LSbs were used to store this information, less than 3 pixels would be affected (2 bits per colour layer equals 6 bits per pixel). The text is stored in a determined order and terminated with a special unused character that signals the end.

In this part, no special features prevent detection or apply obscuring procedures; the content is stored directly. Standard 5 MPx picture could store roughly 1 million characters. That's why short texts would definitely not influence the histogram. To prove this hypothesis, 720 characters were stored in a 5 MPx image, the histogram is practically unchanged.

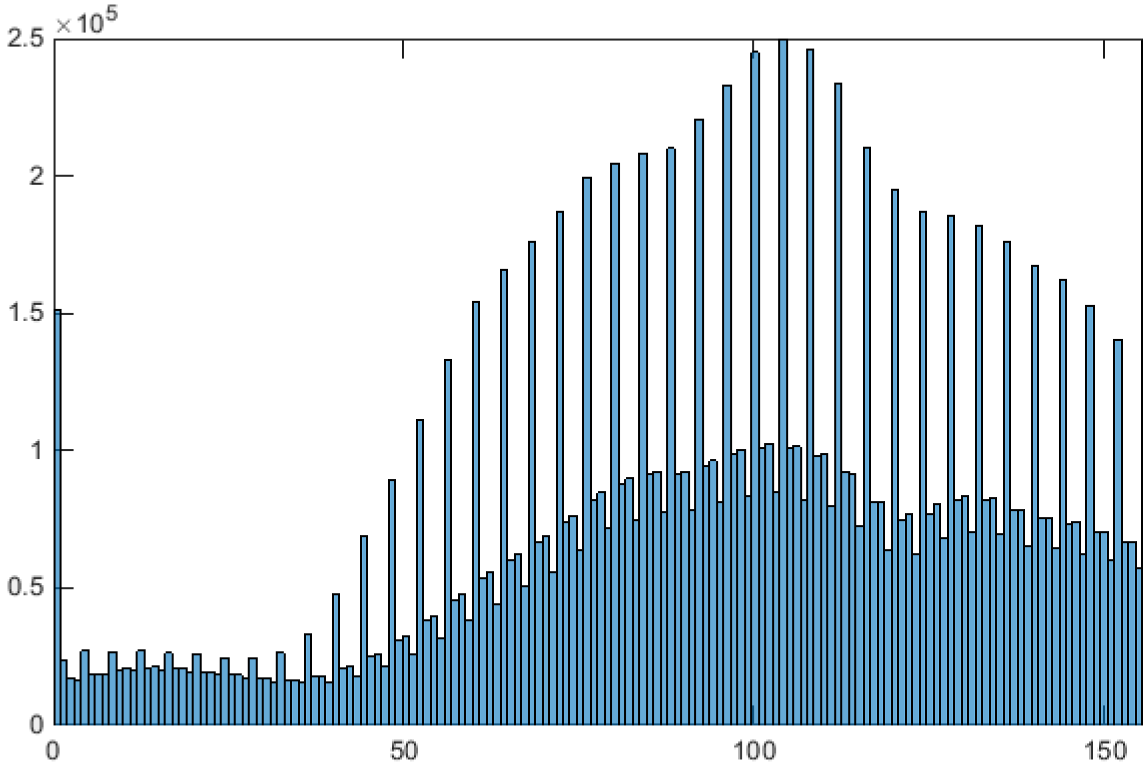


Graph 7: A part of histogram of an image containing a short message (720 characters).

On the other hand, if a long message is stored, the histogram changes significantly. Again, tests were run on a 5 MPx image that was utilised to store the whole text of a novel. Obviously, English text consists mostly of letters a-z and A-Z, therefore some values of LSbs appear more often than other, but the carrier image still cannot be distinguished from a "normal" image by human perception.



Graph 7: A part of the histogram of an image containing *The Old Man and the Sea* (stored as 200.000 characters, approximately using 20 % of the image capacity).



Graph 7: A part of the histogram of an image containing the novel *1984* (stored as 840.000 characters, approximately using 84 % of the image capacity).

Literary aficionados may observe that the amount of stored characters is higher than the actual length of the novel in both cases. The reason for this is the way Matlab handles inputs for text. The text can be divided into paragraphs, but to keep it in the form of a matrix, shorter lines are completed with spaces, so every line has the same amount of characters. This happens when the GUI is used.

On the other hand, the function for storing text can also be used without the GUI. This way, no additional characters are added for the price of disruption of the flow of the text, namely elimination of paragraphs.

The function for storing text is not fully compatible with the GUI (unlike the functions for storing images that are compatible). If the GUI is used, the user can divide the text to paragraphs and it is needed to store their length to be able to reconstruct the structure of the message. The length of the rows is stored as a 2 byte number (the same size as a character) and is added to the text as the first value. The “raw” function does not implement this feature and allows only unstructured text. The general advice is to settle with using only one of the offered methods - either the GUI or the function in a code.

Using the programmed functions

Four functions allow Matlab users to run described processes. For each one, the parameters will be described and functionality shown on an example.

encodeImage

EncodeImage takes five parameters. The first one is the covering image (will become the carrier of the message); the second one is the image that is being hidden. The third argument determines the quality of the hidden image, there are five quality options:

`'C->2bit'`	Hidden image is coloured, uses 2 LSbs of each colour in the covering image (best quality).
`'C->1bit'`	Hidden image is coloured, uses 1 LSb of each colour in the covering image.
`'BW->R'`	Hidden image is black and white and is stored in the LSb of the red layer in the covering image.
`'BW->G'`	Hidden image is black and white and is stored in the LSb of the green layer in the covering image.

`'BW->B'`	Hidden image is black and white and is stored in the LSb of the blue layer in the covering image.

If the last three options are combined, it is possible to store up to three black and white images in one covering image.

The fourth argument is the password, a string of characters with length at least 4. The last argument is a logical value that determines whether smoothing of the histogram is applied or not. The function returns the carrier image with an image hidden within.

If the image being hidden is smaller in any dimension than the cover, a warning is produced and the hidden image is cut in order to fit into the smaller cover. In addition, the carrier image must be stored in a lossless format (e.g. PNG). If it was stored in a loss-making compressed format (e.g. JPG), the stored information would not withstand the losses during compression.

decodeImage

DecodeImage takes three arguments; first one is the carrier image that contains a hidden image, second one is the quality (same as in the table above, determines the source) and third is the password. The function returns the decoded image that was hidden.

Here follows an example of use of encodeImage and decodeImage in Matlab:

```
cover = imread('cover.jpg');
 % load the cover image
hide = imread('secretImage.jpg');
 % load the image to be hidden
result = encodeImage(cover,hide,'C->1bit','geheim',0);
 % encode, now the result can be shown or saved as a file
imwrite(result,'carrier.png');
% save the carrier image

decypher = decodeImage(result,'C->1bit','geheim');
 % decoding
image(decypher);
 % show the hidden image
imwrite(decypher,'decoded.png');
 % save it
```

encodeText

EncodeText function takes 2 arguments, first one is the cover image and the second one is the text to be hidden. Function returns the carrier image containing the hidden message.

decodeText

DecodeText function only needs the image, the hidden message is returned.

```
imageCover = imread('cover.jpg');
 % load the cover image
result = encodeText(imageCover,'You are gonna need a bigger boat.');
 % store message
imwrite(result,'imtext.png');
 % save the carrier image

imageCarrier = imread('imtext.png');
 % load the carrier image
disp(decodeText(imageCarrier));
 % show the message
```

Using GUI

The GUI can be started by running the `gui_start` file and is only a comfortable interface that does not provide any special features, except for storing text (see the “Storing text” chapter for details).

Sources

[1] Steganography. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-06-08]. Available online: http://en.wikipedia.org/wiki/Steganography

[2] Magic square. *MathWorks documentation*. Available online: http://www.mathworks.com/help/matlab/ref/magic.html

[3] Vigenère cipher. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-06-08]. Available online: http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

The testing images (depicting *Aletschgletscher* and *Rigi Kulm*) were taken by the author of this paper and are enclosed to this document.