**Project Encryption-Decryption and Template Matching**

**Structure of the project**

The project is divided into two parts, encryption and decryption, and matching template. To solve both themes we used more functions that will be explained in how they work, what they return and what they change through the parameters. The order of functions will be in order from top to bottom from how they were defined in the project. The project also uses 2 named data structures "Pixel" and "correlation".

***Structure Pixel* uses 3 variables of the unsigned char type for storing single pixel channel values**

***Struct correlation* retains the first pixel in a window by int "i" and int "j", valuation of the correlation between the window and the template by double "value" and the number of the pattern through the "template"**

**Xorshift32**

This function takes the as parameters the seed from which the numbers will be made generated randomly, the height and width of the image.

This function creates a unidimensional array int in which it puts W \* H \* 2 numbers generated.

The function returns the address of the first element in the array to save its location.

Linearization\_imagine

This function takes as a parameter the path of the image to be linearized. The function opens the file where pixels are to be taken and retains in the "width" and "height" variables the width and height of the image.

A one-dimensional pixel type "l\_i" is created that retains the values of the 3 color channels of each pixel.

Because not all images are padding-free, the function holds in variable padding how many bytes were put at each end of the line. While placing the pixels of the image in the unidimensional array, the function checks if it's at the end of a line. If it's over a line, it uses "fseek (f, padding, SEEK\_CUR)" to pass over padding bytes so that they are not put in the unidimensional array. After putting the pixels, the "switching\_pixels" function is called to invert the pixels and returns the address of the first element in the array.

Switching\_pixels

This function takes as parameters an unidimensional array pixel structure type, the height and the length of the image.

Because when you read a .bmp file, the pixels you get begin at the bottom of the image, this function is used to spin all the pixels in the vector. This is a good thing for not it only helps linearize an image but it also works in a way to inversely return the pixels to the positions on which they are aligned. To clarify this function I will explain its functionality by an example:

Let the image:

(0 1 2)

(3 4 5)

(6 7 8)

If we just read the pixels we get in the picture we would get the vector:

(6, 7, 8, 3, 4, 5, 0, 1, 2)

We completely flip this vector and get:

(2, 1, 0, 5, 4, 3, 8, 7, 6)

At this point we are going to go through the elements of the vector at a rate of width number of elements, in this case the width is 3 so we will we move from 3 to 3 elements and we will reverse them

(0, 1, 2, 3, 4, 5, 6, 7, 8)

So we got the linearized image that is changed through parameter. As I mentioned above, if we put a picture unidimensional already linearized in this function, it will restore to as it was before. Here it will restore it to

(6, 7, 8, 3, 4, 5, 0, 1, 2)

Random\_permutation

This function takes the width and height of the image and the unidimensional array with pseudo-random xorshift32 numbers.

The function uses the Durstenfeld algorithm to create the unidimensional array of unsigned int for the permutation vector.

https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle

Returns the address of the first element in the array.

Pixel\_permutation

This function takes the parameters of the unidimensional array whose pixels

will be permutated, unidimensional array with random permutations, a secondary vector representing the copy of the image before using the vector with permutations, the width and the height of the image.

The function copies the pixels of the image before permutation. After this

we use the formula

"Linear\_image [arr\_of\_permutations [i]]. R = li [i] .R;" to permute pixels of the image, where:

linear\_image is the vector to be permutated

arr\_of\_permutations is the vector with permutations

li is the copy vector

The function returns the pixel vector parameter by parameter

**Change**

This function takes as parameters the pixel vector that must be encrypted, the vector with pseudo-random generated numbers, the original image path, the path of the image where encrypted pixels will be put, and the path of the file text from where the secret key will be taken

The function takes the necessary data to function from the original\_bmp file

and filetxt and passes through each pixel in the linearized vector and encrypts it

by formula:



Where:

* Ck is the encrypted pixel
* SV is the secret key
* P0’, Pk’ are the permutated pixels
* Rw\*h, Rw\*h+k are numbers from the pseudo generated numbers
* The equation represents XOR between the unsigned chars of each elements

Two variables are used to navigate through the bytes of an int

type unsigned char \* p, \* p2 that will pass through each byte of

numbers required for encryption

The function returns the new linearized and encrypted vector

Rechange

This function takes as parameters the vector that will receive the decrypted pixels, the encrypted vector, the pseudo-random generated vector, the secret

key, the width and the height

The function uses the decryption equation:



The terminology remains with the exception that Ck’ will be the decrypted pixel

Also in this case two unsigned char \* p, \* p2 variables are used for

Passing through the bytes of the numbers as well as in "change".

The function returns via the parameter "D" the decrypted image.

Chi\_test

This function takes as parameters a linear vector of pixels, the width and

the height of the image

The function creates 3 vectors of 256 size, one for each channel,

initialized with 0, and passes through the image so that the vector value of

whose position is equal to channel value, increases by 1

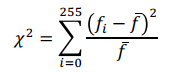
Example: red\_channel [linear\_image [i] .R] ++;

Where red\_channel is the channel with 256 values initialized with 0

Linear\_image is the pixel vector

I is the current pixel

Then uses the equation:



For each color channel, the values on the screen are displayed

Afis

This function takes as parameters the vector with the pixels of which will be put in the image, the width and length of the image, the original image path, the image path where the pixels will be placed

The function puts the necessary data from the original image and the pixels from the vector in to the destination image

Grayscale

This function takes as parameters the vector with pixels, the width, and the height of the image

In function, each pixel is passed and the formula is applied:

0.299 linear\_image [i] .R + 0.587 \* linear\_image [i] .G +0.114\*linear\_image[i] .B;

Then, each channel of the current pixel is assigned that value

The function returns the pixel vector to which it was applied the grayscale algorithm

Color

This function takes as parameters the pixel vector, the position of the first pixel

in the window to be colored by "x" and "y", the width and the length of the image and color represented by a pixel structure variable. The function goes through the first line, the last line, the first column and the last

column of the pixel matrix in the linearized vector to be colored

by the formulas:

D [x \* width + i] .R = D [(x + 15) \* width + i] .R = RGB.R; (lines)

D [(x + i) \* width + y] .R = D [(x + i) \* width + y + 11]. RGB.R(Columns)

And this is done for each channel of each pixel on the edge window.

Temp\_plate

This function takes as parameters the pixel vector, the template vector,

the pixel position of the pixel vector by int i, int j, the minimum value of

correlation, the width and the height of the image, the width and the height of the pattern

The function holds in

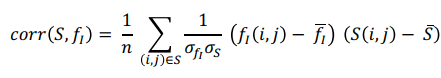
Medie\_template: The average pixels intensities of the template

Medie\_image: The average of pixel intensities in the window

Sigma\_template: The standard deviation of intensities of the template

Sigma\_image: The standard deviation of intensities of the window

Aux\_aux correlation value by formula:



Check that aux\_aux is higher than the minimum value

correlation and if yes then it is returned, otherwise it returns -1

Compare

This function takes as parameters 2 const \* void, makes them type

correlation and checks if the first variable has a higher correlation than

the second variable.

If so, then return 1

If it is smaller, it returns - 1

Else returns 0

Suprapunere

This function takes 2 correlation-type parameters

First, check whether the two have the areas intersected or not

If yes then the difference between the width of a template with the module of

the difference between the two j-s, and the difference between the height of one

with the difference module between the two i-s is made

The area is stored in arie\_ij and the report is made using the formula:

arie\_ij / (arie\_i + arie\_i - arie\_ij)

and returns the date value.

Remove\_correlation

This function takes the correlation vector parameters and the length of it.

Because the vector is ordered downward to each element is

checked the overlay with all the elements after it and if it

finds an overlap of more than 0.2, the element after it is eliminated

The function returns the vector with the overlaps removed by the parameter