UNIVERSIDAD DE GRANADA

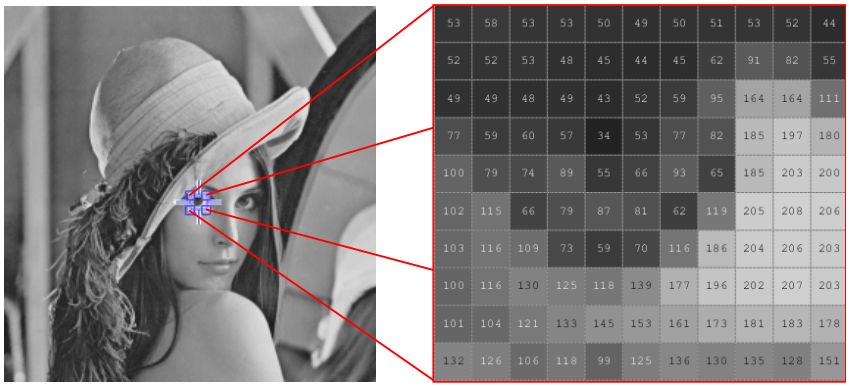
ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2021/2022



Guion de prácticas

Copiar, pegar, binarizar

Febrero de 2022

Índice

1. Objetivos

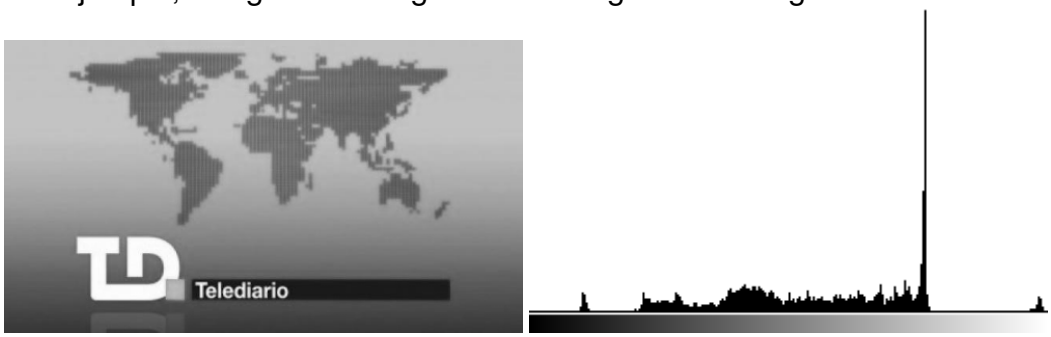
El desarrollo de esta práctica pretende servir a los siguientes objetivos:

- Continuar con otras operaciones basadas en el histograma como la binarización selectiva.
- Explorar la iteración sobre la imagen desde posiciones posiblemente incorrectas sin que esto produzca errores. Para ello se permite copiar un área de la imagen indicando un rectángulo sobre ella. Y pegar esta imagen recortada dentro de otra.

2. Binarización adaptativa de la imagen

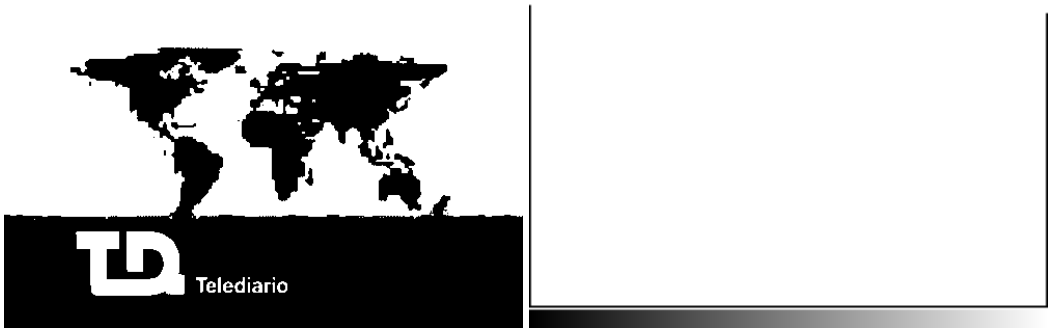
La función de binarizar una imagen a partir de un nivel t permmite transformar en 255 cualquier nivel estrictamente mayor que t y en 0 los demás, lo que produce una imagen binarizada, solo con dos niveles 0 y 1.

Por ejemplo, la siguiente imagen tiene el siguiente histograma



Si la imagen anterior se binariza a 128 se obtiene lo siguiente

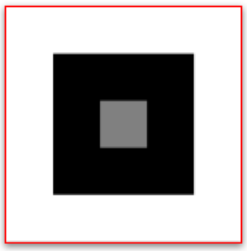
Una imagen también se puede binarizar de forma adaptativa, para ello, se utiliza como t aquél nivel que deja por debajo, o igual a él, al menos a la mitad de la imagen. En este caso, el umbral elegido sería $t = 145$ y produciría este resultado



3. Copiar un área de una imagen

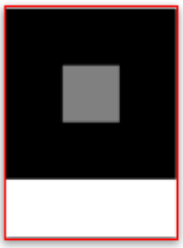
Por ejemplo, dada la siguiente imagen.

255	255	255	255	255
255	0	0	0	255
255	0	128	0	255
255	0	0	0	255
255	255	255	255	255



se podría generar una copia de la misma en el rectángulo (1,1) con 3 de ancho y 4 de alto, produciendo este resultado.

255	255	255	255	255
255	0	0	0	255
255	0	128	0	255
255	0	0	0	255
255	255	255	255	255



Pero también se podrían haber copiado los siguientes rectángulos, que involucran a posiciones fuera de la imagen, y el resultado sería el siguiente, sin provocar ningún error

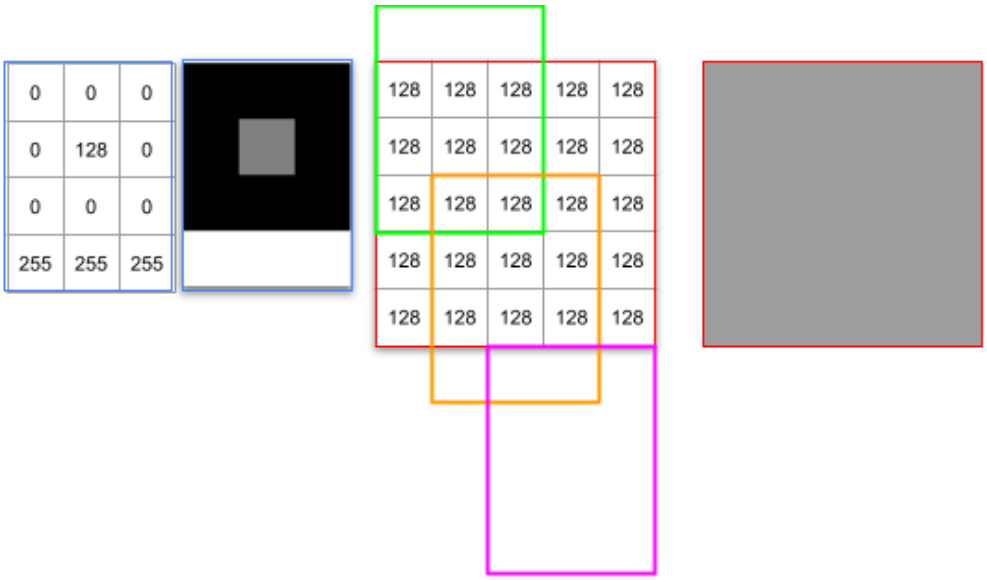
255	255	255	255	255
255	0	0	0	255
255	0	128	0	255
255	0	0	0	255
255	255	255	255	255

255	255	255	255	255
255	0	0	0	255
255	0	128	0	255
255	0	0	0	255
255	255	255	255	255

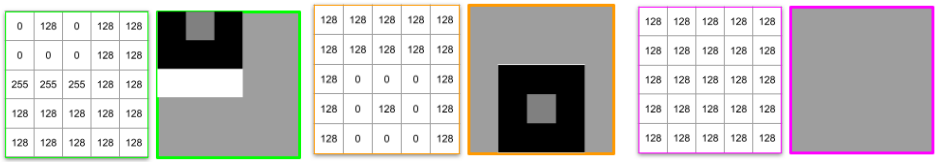
4. Pegado selectivo y gradual en otra imagen

Cualquier imagen, independientemente de su procedencia, se puede pegar dentro de otra, de manera que los píxeles de la primera sustituyen a los píxeles de la segunda, teniendo en cuenta lo comentado anteriormente respecto de la posibilidad de indicar posiciones fuera de la imagen.

Por ejemplo si quisiésemos pegar la imagen pequeña (enmarcada en color azul) en la segunda imagen (rojo) en tres posiciones distintas (verde, naranja y rosa)



Los resultados de cada una de ellas, serían estos



4.1. Pegado selectivo

Cuando se pega una imagen en otra, se pueden elegir qué niveles se consideran transparentes, es decir, no se van a copiar en la imagen de destino. Para ello lo más sencillo es considerar un nivel de gris de la imagen, k, y pegar solo aquellos píxeles cuyo nivel estén por encima de k.

4.2. Pegado gradual

Normalmente, en una operación de pegado, los píxeles de la primera imagenn sustituyen a los de la segunda. En este caso, se puede indicar que se fusionen a nivel $\alpha \in [0, 100]$ de forma que si b_1 es el byte de la primera imagen y b_2 es el byte de la segunda imagen, entonces el byte que realmente se pega en la imagen de destino es

$$b_d = \frac{\alpha * b_1 + (100 - \alpha) * b_2}{100}$$

5. Histogram

```
1  /**
2   * @file Histogram.h
3   * @author MP
4   */
5  #include <istream>
6  #include <fstream>
7  #include "Byte.h"
8  #include "Image.h"
9
10 #ifndef _HISTOGRAM_H_
11 #define _HISTOGRAM_H_
12
13
14
15 #define HISTOGRAM_LEVELS MAX_BYTE ///< Max number of bytes allowd for
16 #define HISTOGRAM_TOLERANCE 0.01
17
18 /**
19 @brief A black and white histogram
20  */
21 class Histogram {
22 private:
23     int _data[HISTOGRAM_LEVELS]; ///< datos de la imagen
24
25 public:
26
27     /**
28      * @brief It builds an empty
29      */
30     Histogram();
31     /**
32      * @brief It returns the number of levels in the histogram
33      * @return The number of levels
34      */
35     int size() const;
36     /**
37      * @brief Sets the whole histogram to 0
38      */
39     void clear();
40     /**
41      * @brief It returns the value associated to the level indicated
42      * @param level The level indicated
43      * @return The value associated to the level
44      */
45     int getLevel(Byte level) const;
46     /**
47      * @brief It sets the value associated to the level
48      * @param level The level
49      * @param npixeles The new value
50      */
51     void setLevel(Byte level, int npixeles);
52     /**
53      * @brief It returns the maximum value stored
54      * @return The max of the levels
55      */
56     int getMaxLevel() const;
57     /**
58      * @brief it returns the average value stored
59      * @return The average level
60      */
61     int getAverageLevel() const;
62
63     /**
64      * It returns a balance level, that is, the level that leaves half of the points
65      * underneath or equal to it.
66      * @return The point of balance of the histogram
67      */
68     int getBalancedLevel() const;
69     /**
70      * @brief It returns a unique hash code for every object so that they might be compared
71      * @return The hash code as an string
72      */
73     std::string inspect() const;
74 };
75 #endif
```

6. Image

```
1  /**
2  @file Image.h
3  @brief Manejo de imÃ¡genes digitales en formato PGM blanco y negro
4  @author MP-DGIM - Grupo A
5   **/
6
7  #ifndef _IMAGE_H_
8  #define _IMAGE_H_
9
10 #include <istream>
11 #include <fstream>
12 #include "Byte.h"
13 #include "Histogram.h"
14
15 #define IMAGE_MAX_SIZE 200000 ///< Max number of bytes allowd for
16 #define IMAGE_DISK_OK 0 ///< Image read/write successful
17 #define IMAGE_ERROR_OPEN 1 ///< Error opening the file
18 #define IMAGE_ERROR_DATA 2 ///< Missing data in the file
19 #define IMAGE_ERROR_FORMAT 3 ///< Unknown image format
20 #define IMAGE_TOO_LARGE 4 ///< The image is too large and does not fit into memory
21 /**
22 @brief A black and white image
23  */
24 class Image {
25 private:
26     Byte _data[IMAGE_MAX_SIZE]; ///< Bytes of the image
27     int _height; ///< number of rows
28     int _width; ///< number of columms
29 public:
30     /**
31      * @brief It builds an empty, image
32      */
33     Image();
34     /**
35      * @brief It builds a fully black image with @a width columns and @a height rows
36      * @param height number of rows
37      * @param width number of columns
38      */
39     Image(int width, int height);
40     /**
41     @brief It gives the number of rows of the image
42     @return number of rows
43      */
44     int height() const;
45     /**
46     @brief It gives the number of columns of the image
47     @return The number of rows
48      */
49     int width() const;
50     /**
51      * @brief It assigns the value @a v to the position(x,y) of the image.It must check that
52      * the values x and y are valid, otherwise, it does not do anythig.
53      * @param x The column
54      * @param y the row
55      * @param v The new value
56      */
57     void setPixel(int x, int y, Byte v);
58     /**
59      * @brief It returns the value of the requested (x,y) position. It must check that
60      * the values x and y are valid, otherwise, it returns a negative value. Please note that
61      * the value returned is a int
62      * @param x The column
63      * @param y the row
64      * @return The value of the pixel in [0-256] or -1 if there is an access error
65      */
66     int getPixel(int x, int y) const;
67     /**
68      * @brief It assigns the value @a v to the linear position i of the image. It must check that
69      * the values i is valid, otherwise, it does not do anythig.
70      * @param i The linear position
71      * @param v The new value
72      */
73     void setPos(int i, Byte v);
74     /**
75      * @brief It returns the value of the requested linear position. It must check that
76      * the value i is valid, otherwise, it returns a negative value. Please note that
77      * the value returned is a int
78      * @param i The linear position
79      * @return The value of the pixel in [0-256] or -1 if there is an access error
80      */
81     int getPos(int i) const;
82     /**
83      * @brief  It sets all pixels of the image to the value given
84      * @param b The value
85      */
86     void flatten(Byte b);
87     /**
88      * @brief It produces a mesh of vertical and horizontal stripes all along the
89      * image. Every prim pixels it is set to 255 anad every sec pixels
90      * it is set to 127
91      * @param prim Gap between primary mesh
92      * @param sec Gap between secondary mesh, Default value is 0
93      */
94     void mesh(int prim, int sec=0);
95
96     /**
```

```
97      * @brief It shows an image in an external window, ready for inspection. It uses
98      * the program display (ImageMagick) to display every image. For an easier identification
99      * process of all images shown are labeled with a title
100     * @param title The title on top of the window
101     */
102    void showInWindow(std::string title);
103    /**
104     * @brief It calculates the hash value of the image and returns it inside a string,
105     * together with its dimension.
106     * @return a string that contains the dimension and the hash value of the image
107     */
108    std::string inspect();
109    /**
110     * @brief It opens a file that contains a PGM Image and reads the data into
111     * a iname in memmory
112     * @param filename Name of the file
113     * @return a code that means the following: 0 - Successful operation.
114     * 1 - Error opening the file
115     * 2 - Error reading the data
116     * 3 - The detected data does not follow the PGM techincal description
117     */
118    int readFromFile(const char filename[]);
119
120    /**
121     * @brief It writes the Image on disk, in PGM ascii format
122     * @param filename The name of the disk file which will contain the image
123     * @return The same code that readFromFile()
124     */
125    int saveToFile(const char filename[]) const;
126    /**
127     * @brief It calculates the histogram of the image, and returns it into an
128     * instance of the class Histogram
129     * @param values
130     */
131    Histogram getHistogram() const;
132   /**
133     * @brief It takes the histogram of the image and depicts a new image with the
134     * visualization of the histogram according to these rules
135     * ~~~~
136     *   +-     +--------------------------------o
137     *   |      |                                | <-- 1 pix white line
138     *   |      |   *                            | -+
139     *   |      |   **                           |  |
140     *   |      |   **                           |
141     *   |      |   ***                    *     |   Normalized histogram
142     *   |      |   ***                   **     |     150 pix
143     *   h=160  |   ****    *             **     |  |
144     *          |   *****  ***           ****    |  |
145     *   |      |  ************************** | -+
146     *   |      |--------------------------------| 1 pix  white line
147     *   |      |--------------------------------| -+
148     *   |      |                                |  8 pix Scale of gray levels
149     *   +-     o--------------------------------+ -+
150     *
151     *          |                                |
152     *          +---------- w  = 256 ------------+
153     * ~~~~
154     * @return
155     */
156    Image depictsHistogram() const;
157
158    /**
159     * @brief It segements the histogram by groups whose value is higher than the admitted tolerance and
        returns
160     * an array of images, each of which corresponds to one of these objects
161     * @param set The array of images
162     * @param nimages The number of images found
163     * @param maximages The max number of images to be found
164     * @param tolerance The tolerance addmitted
165     */
166    void extractObjects(Image set [], int &nimages, int maximages, double tolerance=HISTOGRAM_TOLERANCE)
        const;
167
168
169    /**
170     * It returns a binarization of the original image. All pixels  strictly greater than the value @a t
171     * are set to 11111111 and the others to 00000000.
172     * @param t The threshold. This value must be within [0,255]. In the case
173     * that the threshold is not within these bounds, an automatic threshold is chosen,
174     * that is, the first level that leaves, at least the half of points less that
175     * or equal to it
176     * @return A copy of the original image
177     */
178    Image threshold(int t = -1) const;
179    /**
180     * @brief It returns a subimage of the original image given by the parameters.
181     * If the parameters exceed  the dimensions of the original image, then
182     * the cut image should not exceed those limits.
183     * @param x x-coordinate of the top left corner
184     * @param y y-coordinate of the topt left corner
185     * @param w width of the subimage
186     * @param h height of the subimage
187     * @return The subimage cut
188     */
189    Image copyArea(int x, int y, int w, int h) const;
190
191    /**
192     * @brief It pastes a secondary image into the original image, at the given position,
193     * into the same original image, which results modified.
194     * This operation should maintain the limits of the original image
```

```
195         * @param x x-coordinate of the top left corner
196         * @param y y-coordinate of the topt left corner
197         * @param from The second image
198         */
199        void pasteArea(int x, int y, const Image &from, int toneup=-1, int merge=100);
200
201
202    };
203    #endif
```

7. Práctica a entregar

7.1. Primera parte

Esta parte se entrega junto a la segunda parte en una única entrega. Se ha divido en dos partes para que su implementación sea más gradual.

- Se deben implementar las funciones incluidas en el fichero `Image.h` y en `Histogram.h`
- Leer una imagen de disco, que llamaremos $input$ y otra imagen a copiar, que llamaremos $copyfrom$. Registraremos el ancho w y alto h de $copyfrom$.
- Segmentar la imagen $copyfrom$ en objetos y elegir el primero de la colección de imágenes que resulta, es decir, el de la posición 0, que llamaremos $coleccion$.
- Binarizar $copyfrom$ de forma selectiva, lo que dará otra imagen que llamaremos bin
- Pegar $copyfrom$ en la posición $(0, 0)$ de input. Pegar $coleccion$ en la posición $(w + 5, 0)$ y bin en $(w + 5, h + 5)$.
- Pegar $coleccion$, desde el nivel 64 en adelante, en la posición $(2 * w + 10, 0)$ y bin, desde el nivel 64 en adelante, en $(2w + 10, h + 5)$.
- Pegar $coleccion$, desde el nivel 64 en adelante y $alpha = 50$, en la posición $(3 * w + 15, 0)$ y bin, desde el nivel 64 y $alpha = 50$, en $(3w + 15, h + 5)$.
- Guarda el resultado en una imagen dentro de la carpeta `data/` llamada `new.pgm`.

7.1.1. Ejemplo de ejecución

Si se coge como imagen a copiar la imagen `kfc.pgm`

y como *input* la imagen `telediario.pgm`,

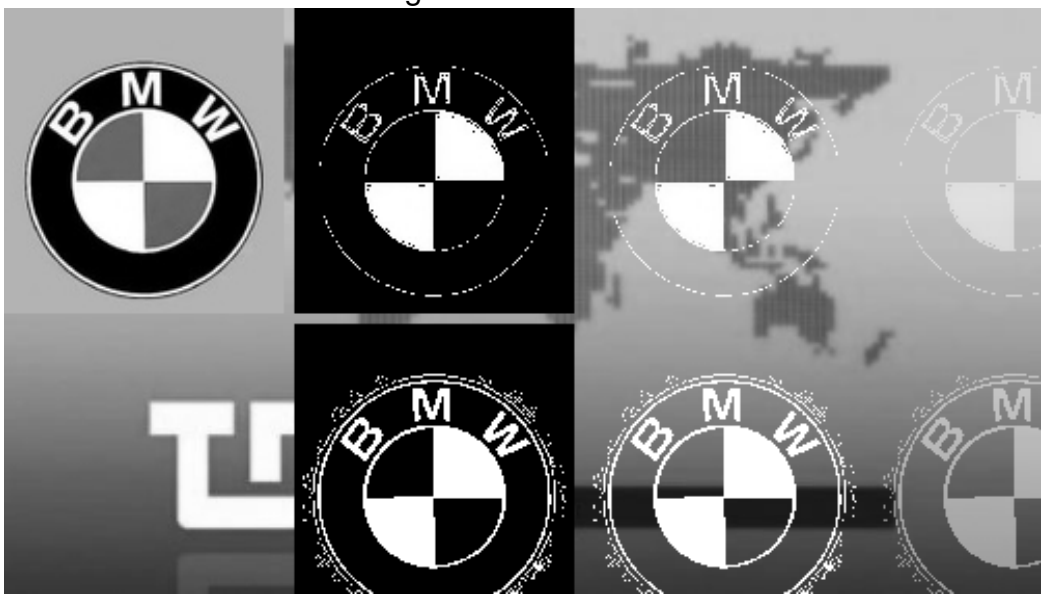
el resultado debería ser el siguiente


```
> dist/Debug/GNU-Linux/practica3

...Reading image from ./data/telediario.pgm
500x282

[im_input] 500x282 10368250849137031550

...Reading image from ./data/kfc.pgm
89x84

[im_copyfrom] 89x84 5043932668275557787
```

```
Thresholding to level 118

[im_bin] 89x84 10058019176305826292
Found object 0 in [249,251]
Found object 1 in [229,230]
Found object 2 in [227,228]

[im_collection[0]] 89x84 10449573973878039676

...Saving image into ./data/new.pgm

[im_output] 500x282 14874412274324249425
Press [RETURN] to continue ...
```

Si, por el contrario, se coge como imagen a copiar la imagen `bmw.pgm`

el resultado debería ser el siguiente



7.2. Segunda parte

Esta es la verdadera práctica a entregar. Hace exactamente lo mismo que la parte anterior, pero recibe los datos de entrada desde la línea de comandos

```
p3b -i <input> [-c <copyfrom> -o <output>]
```

- $-i < input >$ Es un parámetro obligatorio y determina qué imagen se considerará como $input$
- $-o < output >$ Es un parámetro opcional. Si no se indica, el resultado sólo aparece en pantalla. Si se indica, además de en pantalla, el resultado se guarda en disco con el nombre indicado

- $-c < copyfrom >$ Es un parámetro opcional. Si aparece, se utiliza la imagen indicada como $copyfrom$, en otro caso, no se hace ningún cambio a la imagen $input$.
- Todos los parámetros pueden aparecer en cualquier orden.

```
lcv@numenor:Practica3b: dist/Debug/GNU-Linux/practica3b

Error in call:  Missing input file
Please use: -i <input> [-c <copyfrom> -o <output>]

-i <input>
Input image from <input>
-c <copyfrom>
Copy clip from <copyfrom>
-o <output>
Output image to <output>
```

```
lcv@numenor:Practica3b: dist/Debug/GNU-Linux/practica3b -i data/telediario.pgm

...Reading image from data/telediario.pgm
500x282

[im_input] 500x282 10368250849137031550

[im_output] 500x282 10368250849137031550
Press [RETURN] to continue ...
```

```
lcv@numenor:Practica3b: dist/Debug/GNU-Linux/practica3b -o ./data/telebmw.pgm
-i data/telediario.pgm -c data/bmw.pgm

...Reading image from data/telediario.pgm
500x282

[im_input] 500x282 10368250849137031550

...Reading image from data/bmw.pgm
135x147

[im_copyfrom] 135x147 4417026241012456264
Thresholding to level 180

[im_bin] 135x147 5849421183935098771
Found object 0 in [249,255]
Found object 1 in [174,182]
Found object 2 in [98,105]

[im_collection[0]] 135x147 5876811545242803409

...Saving image into ./data/telebmw.pgm

[im_output] 500x282 3239281732513615719
```

7.3. Tests run

```
---
```