Instituto Tecnológico de Buenos Aires

22.90 Automación Industrial

Trabajo práctico de visión

Segundo Parcial

Grupo 3:
Lin, Xi 59780
Muscariello, Gianfranco 60361
Figueroa, Alejo 60412
Molina, Facundo 60526

Docentes:
Arias, Rodolfo Enrique
Sofio Avogadro, Federico
Spinelli, Mariano Tomás

Presentado:
Corrección:

Contents

1 Enunciado

Programar un algoritmo que permita identificar todas las letras que pertenecen a un mismo grupo de colores. Hay que tener en cuenta que la posición de las letras, los cuadros pintados, y las letras en sí, cambian. Sin embargo, los 5 colores seleccionados no se modificaran.

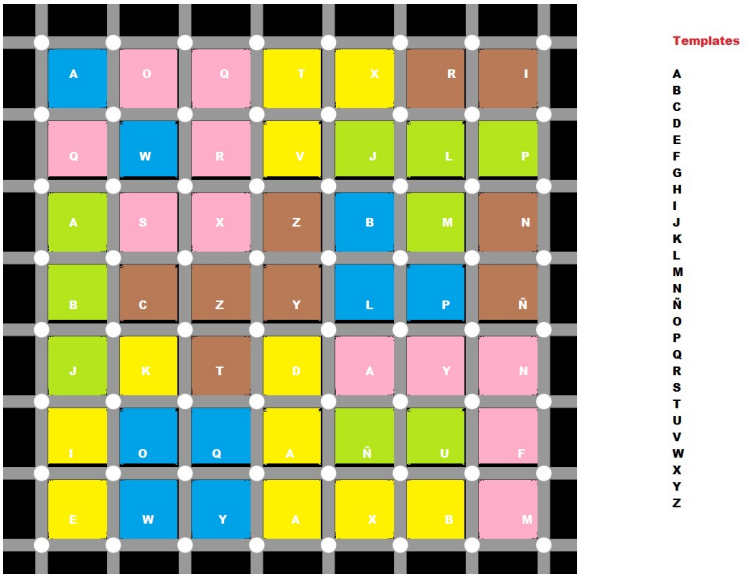


Figure 1: Imagen a evaluar.

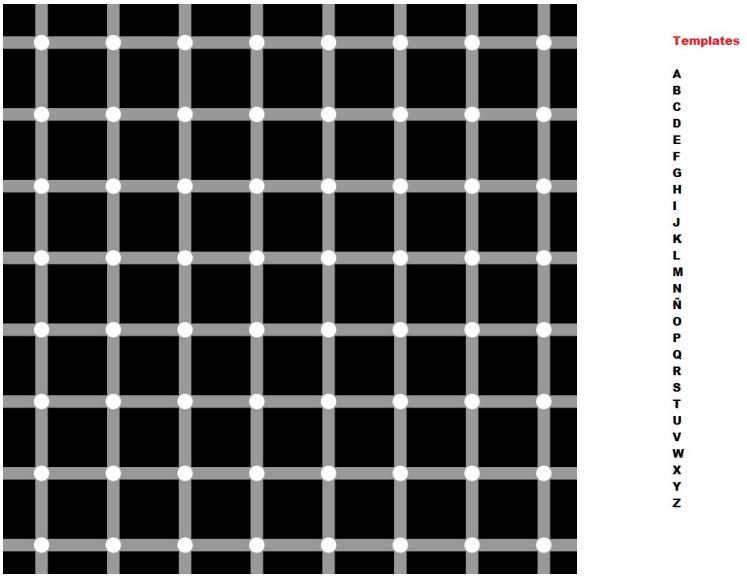


Figure 2: Template de las imagenes a evaluar.

2 Armado de los templates de las letras

Se utilizó el template provisto para extraer cada letra del abecedario para luego poder comparar con las letras ubicadas dentro de cada cuadrado de color. Se le aplicó el complementario a las matrices de cada letra debido a que las letras de template estaban en negro, pero las de los cuadrados en blanco.

```
function templates = get_templates()
% GET_TEMPLATES  Devuelve imagenes limpias con los bloques asociados a cada color
%   templates = get_templates()
%
%   templates: lista con 27 imagenes de 15x15, cada una letra
%   las letras son extraidas del archivo de base.

%Cargamos imagen de muestra
im=iread('template.jpg');
im_d=idouble(im);

imgrey_d=imono(im_d);   %imagen en escala de grises, double

%********** LETRAS **********
% Letra A
templateA = imgrey_d(90:104,932:946);
A = 1 - templateA ;

% Letra B
templateB = imgrey_d(114:128,932:946);
B = 1 - templateB ;

...

% Letra Z
templateZ = imgrey_d(688:702,933:947);
Z = 1 - templateZ ;

templates = [A B C D E F G H I J K L M N NIE O P Q R S T U V W X Y Z];
```

Listing 1: Definición *get_templaates*

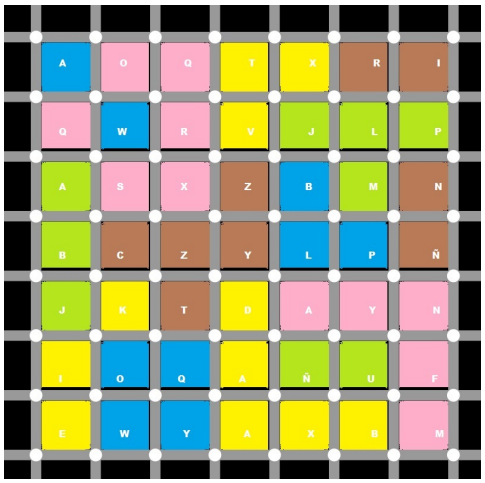
Se obtienen los templates de las letras que se muestran a continuación.

Figure 3: Templates de las letras

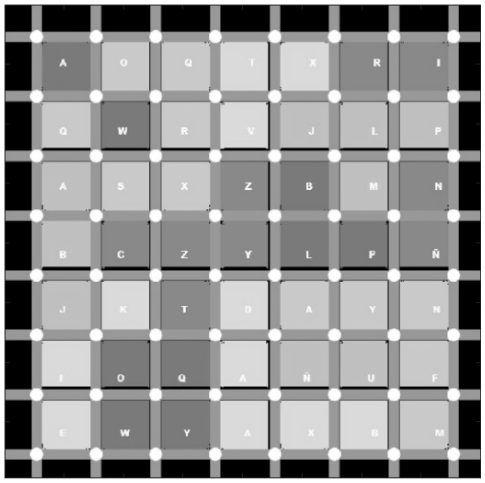
3 Filtrado por colores

3.1 Análisis de los colores

Para empezar, se inspeccionaron tanto la imagen original (Figura 4a) como la imagen en escala de grises (Figura 4b). Se observa a simple vista que incluso en escala de grises se pueden distinguir los diferentes sectores dados por diferentes colores.



(a) Imagen de prueba original.



(b) Imagen en escala de grises.

Figure 4

Luego pasamos a analizar el histograma de la imagen en escala de grises, para encontrar los limites entre 0 y 255 para cada uno de los colores. En la Figura 5 observamos una separación muy marcada entre los picos de cada uno de los colores, así que usaremos thresholds en base a los valores en los que cada color se encuentra completamente contenido (no usamos los picos de cada color, si no un threshold ligeramente superior para englobar pequeños cambios de tono del mismo color).

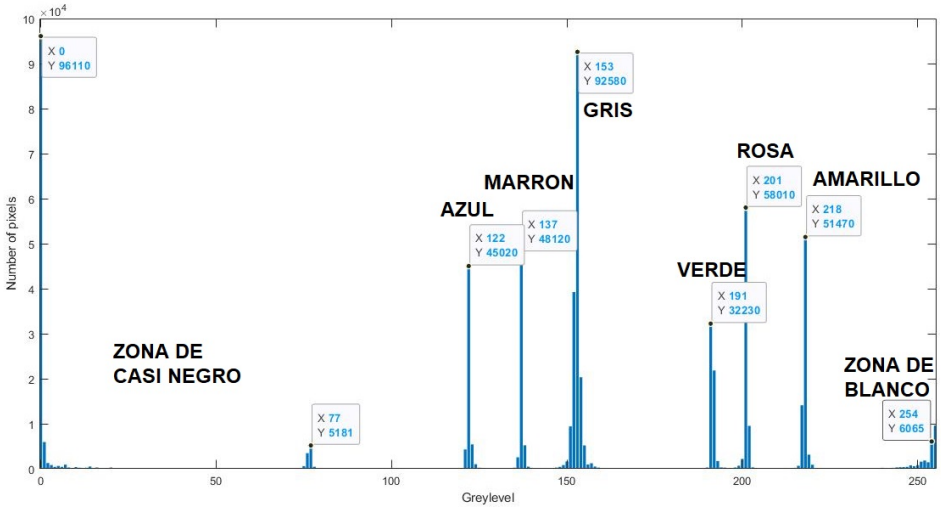


Figure 5: Histograma de la escala de grises de la imagen de prueba original.

3.2 Separación de colores

Finalmente, separamos la imagen en seis imágenes diferentes, cada una con el threshold asociado a su color respectivo, y las fuimos restando uno a uno con su color "anterior" en escala de grises. En la Figura 6 se muestra un ejemplo de este proceso para los bloques de color azul.

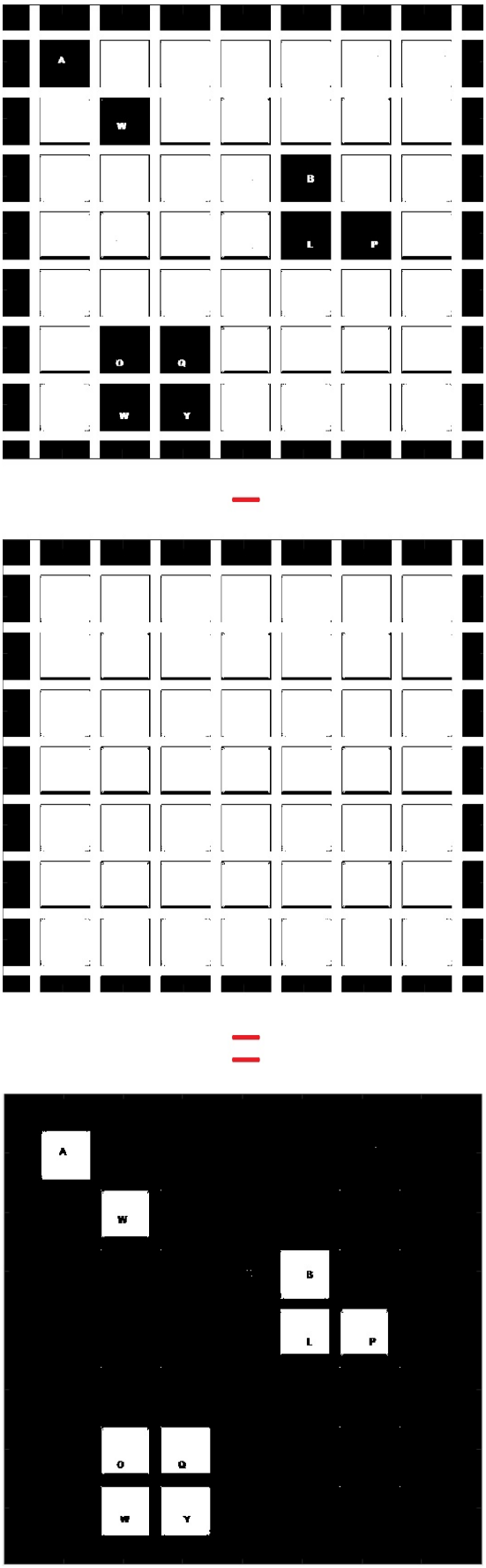


Figure 6: Ejemplo de la resta de imágenes.

Esta operación resultó en la Figura 7, donde vemos que para cada color se pudo extraer la zona de sus bloques, pero con un poco de ruido del tipo *salt*, y con las letras aún en su interior. Esto se arreglará en la siguiente sección.

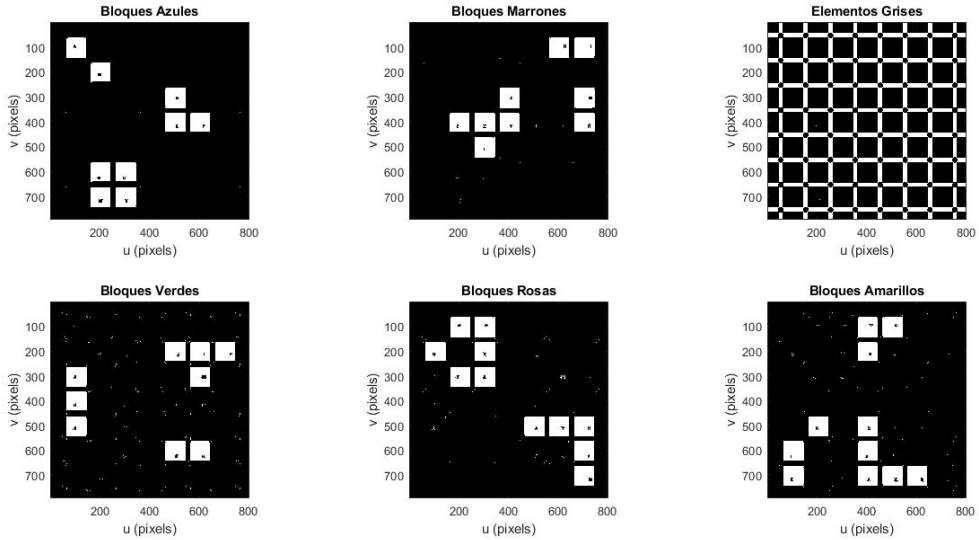


Figure 7: Separación de bloques por colores.

4 Filtro de ruido y letras para bloques

Ahora que tenemos una imagen ruidosa y con letras, pero bien separada para cada color, es sencillo remover el ruido y las letras, al igual que hacer que los bloques sean más similares al rectángulo que pretende ser.

4.1 Operación de cierre

Usamos una operación de cierre para eliminar las letras, utilizando un elemento estructural de tamaño a penas más grande que la letra de mayor tamaño. De esta forma, se limpia internamente a los bloques. El resultado de esta operación se puede observar en la Figura 8.

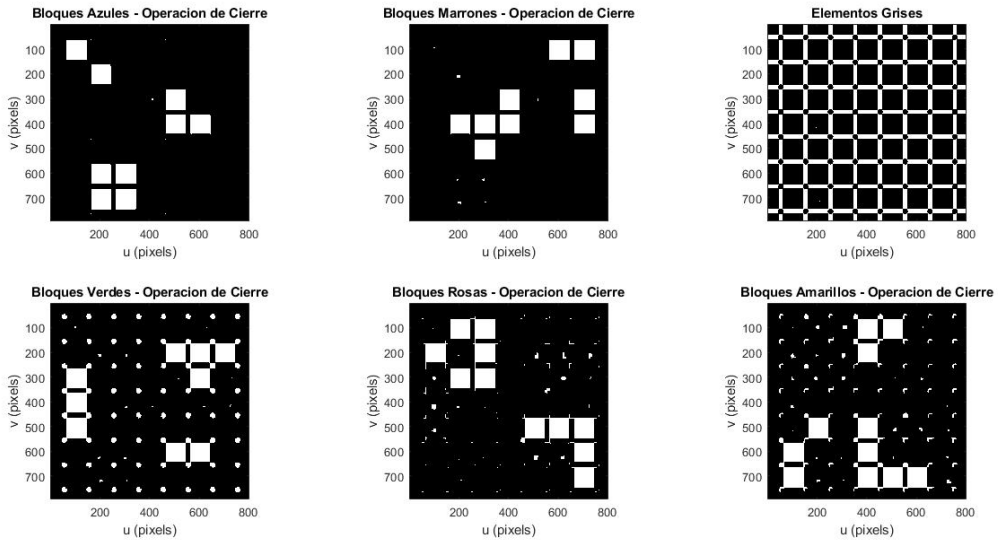


Figure 8: Filtrado de letras con una operación de cierre.

4.2 Operación de apertura

Se pudo ver que a pesar de filtrar las letras, la imagen resultante hizo que el ruido tipo *salt* se haga más grande. Esto puede solucionarse con una operación de apertura con un elemento estructural de tamaño mayor al ruido, pero menor al de un bloque. Usamos un tamaño de aproximadamente 75% del bloque más pequeño. El resultado se observa en la Figura 9, y también se aprecia que los bloques quedaron "muy cuadrados", ideal para la siguiente operación: detección de bloques.

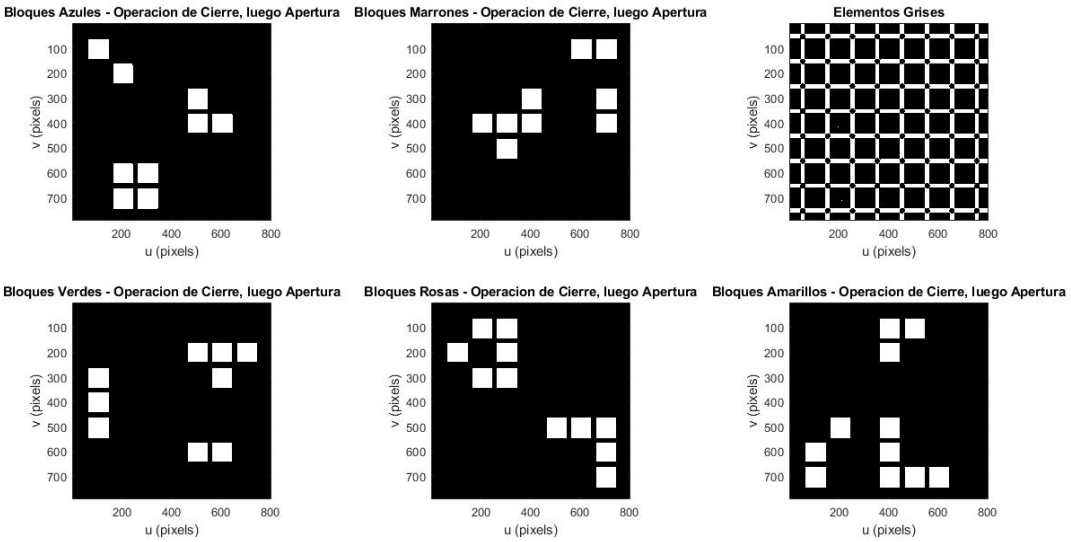


Figure 9: Filtrado de ruido con una operación de apertura.

Tanto la separación de colores, como la limpieza de los bloques se encapsularon en la siguiente función:

```
function color_blocks=get_color_blocks(im_grey)
% GET_COLORED_BLOCKS  Devuelve imagenes limpias con los blocks asociados a cada color
%   color_blocks = color_blocks(im_grey)
%
%   color_blocks: lista con 5 imagenes, cada una con los blocks de cada
%   color, sin letras. En el siguiente orden:
%       AZUL
%       MARRON
%       VERDE
%       ROSA
%       AMARILLO
%   im_grey: imagen original en escala de grises.

%***Rango de los diferentes colores, encontrado analizando el histograma de la imagen de muestra.
range_black = [0 110];
range_blue = [120 130];
range_brown = [135 144];
range_gris = [150 163];
range_green = [190 194];
range_pink = [199 203];
range_yellow = [216 220];

%Genero blocks blue
im_black = im_grey > range_black(2);
im_blue = im_grey > range_blue(2);
blocks_blue = im_black - im_blue;

%Genero blocks brown
im_brown = im_grey > range_brown(2);
blocks_brown = im_blue - im_brown;

%Genero elementos grises
im_gris = im_grey > range_gris(2);
blocks_gris = im_brown - im_gris;

%Genero blocks green
im_green = im_grey > range_green(2);
blocks_green = im_gris - im_green;

%Genero blocks pink
im_pink = im_grey > range_pink(2);
blocks_pink = im_green - im_pink;

%Genero blocks yellow
im_yellow = im_grey > range_yellow(2);
blocks_yellow = im_pink - im_yellow;

%******* Operacion de cierre y apertura **********
%Cierre
%Una letra no es mucho m s que 18*18, tomamos un elemento estructural de ese tama o para
    eliminarlas
S = ones(18,18);

blocks_blue_closed = iclose(blocks_blue, S);
blocks_brown_closed = iclose(blocks_brown, S);
blocks_green_closed = iclose(blocks_green, S);
blocks_pink_closed = iclose(blocks_pink, S);
blocks_yellow_closed = iclose(blocks_yellow, S);

%Apertura
%Un bloque tiene approx 85x85, tomamos un elemento estructural de un poco mas de 75%
S = ones(65,65);

blocks_blue_clean = iopen(blocks_blue_closed, S);
blocks_brown_clean = iopen(blocks_brown_closed, S);
blocks_green_clean = iopen(blocks_green_closed, S);
blocks_pink_clean = iopen(blocks_pink_closed, S);
blocks_yellow_clean = iopen(blocks_yellow_closed, S);

color_blocks = [blocks_blue_clean blocks_brown_clean blocks_green_clean blocks_pink_clean
    blocks_yellow_clean];
```

Listing 2: Definición *get_color_blocks*

5 Identificación de los bordes de los cuadrados de cada color

Se definió la función **get_squares_coord** que permite obtener las coordenadas de los bordes superior izquierdo e inferior derecho de la imagen procesada para cada color. Para lograrlo, se empleó función **iblob** del toolbox de visión de Corke para identificar y agrupar los cuadrados. Se aprovechó el conocimiento de las dimensiones aproximadas en pixeles de los cuadrados para filtrar los blobs reconocidos por área. Dadas las funciones que se emplearon para la comparación y detección de letras, no fue necesario analizar todos los bordes de los cuadrados. Asimismo, la función permite reconocer la cantidad de cuadrados a ser analizados por cada color.

Posteriormente, se utilizaron los métodos de Blobs para construir las coordenadas a ser devueltas en un arreglo de matrices de la forma:

$$\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$$

```
function corners = get_squares_coord(img)
% GET_SQUARES_COORD  Devuelve coordenadas de los vertices superior izquierdo e
%   inferior derecho todos los cuadrados de una imagen.
%   corners = get_squares_coord(img)
%
%   -img: imagen, idealmente con el ruido filtrado, pero un poco de ruido
%   no tiene impedimentos.
%   -corners: matriz con vertices del segmento en formato 2x2 |x1 x2|
%                                                             |y1 y2|

im_thres = imono(img)>0.9;

% Se obtienen los blobs de un area minima en la imagen.
% Se tomo el area promedio de un bloque para no tomar al posible ruido.
blobs = iblobs(im_thres, 'area', [5000,10000]);
n = size(blobs);
n = n(2);

%Se inicializa el vector de matrices de coordenadas.
corners = zeros(2,2,n);

%Para cada uno de los blobs:
for i = 1:n
    %Obtenemos el centroide
    centx = blobs(i).uc;
    centy = blobs(i).vc;
    %Aproximamos al blob como un rectangulo perfecto porque sabemos que, mas
    %alla de deformaciones durante el filtrado, los blobs son rectangulos
    base = sqrt(blobs(i).area)/2;
    corners(:,:,i) = [int32(centx-base) int32(centx+base); int32(centy-base) int32(centy+base)];
end
```

Listing 3: Definición *get_squares_coord*

6 Identificación de la letra dentro de cada cuadrado

Una vez separados los cuadrados por colores e identificadas las posiciones de los cuadrados, se llama a la función *find_letter* para identificar qué letra es la que se encuentra dentro.

La función itera por todos los templates de las letras para identificar el que tiene mayor correlación. No se optó por utilizar el método del punto de threshold porque este difiere para cada letra y puede llevar a múltiples detecciones de falsos positivos o no detectar ninguno. Con el método que aplicamos nos aseguramos de que haya una única letra que es a la que más 'se parece'.

Además en el código se distingue el caso de la N y la Ñ debido a que son muy parecidos, pero el problema que surge es debido al tamaño de la letra Ñ que supera al resto, es por eso que se extrae otra matriz más grande para hacer mejor la comparación.

```
function letter = find_letter(square, templates, imgrey_d)
% FIND_LETTER  La letra mas probable de estar dentro del bloque.
%   letter = find_letter(square, templates, imgrey_d)
%
%   -letter: letra con mayor probabilidad.
%   -square: coordenadas de un bloque, asumimos que hay una sola letra dentro
%   del bloque.
%   -templates: array de imagenes de template de las letras, ordenadas.
%   -imgrey_d: imagen original en donde buscar el bloque y la letra.

%Lista de letras
letras = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
N = size(letras);

square=imono(square);   %imagen en escala de grises, double

letter = '';
maxvalue = 0;

%Se itera comparando contra todas las letras y quedamos con el maximo pico
for i = 1:N(2)
    Detect=isimilarity(templates(1:15,(i-1)*15 +1: (i)*15),square); % Comparacion cuadrado y template de letra(i)
    [max,p] = peak2(Detect, 1, 'npeaks', 1);    % Medimos pico de intensidad contra esa letra
    if max > maxvalue
        letter = letras(i); %En caso de ser la mayor de todas,
        maxvalue = max;
    end
end

%Caso deteccion   : agrandamos el tama o del template para N y    para
%lograr una mejor deteccion. No vale la pena generalizar la    porque
%queremos mantener el tama o del template al minimo para mejorar la
%deteccion.(Fue la soluci n que encontramos. Al agrandar todos los templates
%se producian colisiones).
if letter == 'N'
    im=iread('template.jpg');
    im_d=idouble(im);
    imgrey_d=imono(im_d);

    % Letra N
    templatN = imgrey_d(389:405,931:947);
    Ntemp = 1 - templatN ;

    % Letra
    templatNIE = imgrey_d(409:425,932:948);
    NIEtemp = 1 - templatNIE ;

    DetectN=isimilarity(Ntemp,square);
    [maxN,pN] = peak2(DetectN, 1, 'npeaks', 1);
    DetectNIE=isimilarity(NIEtemp,square);
    [maxNIE,pNIE] = peak2(DetectNIE, 1, 'npeaks', 1);
    if maxN > maxNIE
        letter = 'N';
    else
        letter = '  ';
    end
end
```

Listing 4: Definición *find_letter*

7 Implementación

Se integraron las funciones anteriormente estudiadas en un solo *main.m*, el cual se puede apreciar a continuación:

```
%% Preparaci n del espacio de trabajo.
clear all close all clc

%Carga de imagen.
imgrey_d = iread('PROBAME_G3.jpg', 'grey');  %imagen en escala de grises, double

%Tama o de imagen.
size_img = size(imgrey_d);
fila = size_img(1);
columna = size_img(2);

%% Imagenes de cuadrados por cada color.
%se obtiene una matriz que contiene 5 im genes. Cada una tiene un filtrado
%de los cuadrados de cada color.
%El orden de los colores es en base a su nivel en escala de grises.
color_blocks = get_color_blocks(imgrey_d);

%% Iteracion por cada color identificando letras en cuadrados
%Se obtienen los templates de cada letra, en base a una imagen de base.
templates = get_templates();
colors = ["Blue", "Brown", "Green", "Pink", "Yellow"];  %Array auxiliar

for i = 0:length(colors)-1

    fprintf('%s: ', colors(i+1))    %Color iterado

    %Identificaci n de coordenadas de cuadrados por color
    img = color_blocks(:, i * columna+1:(i+1) * columna);   %Se analiza un solo color
    corners = get_squares_coord(img);   %coordenadas de cuadrados del color
    squares_num = size(corners);        %cantidad de cuadrados por color

    %Identificaci n de letras
    for j = 1: squares_num(3)
        %Se analiza un solo cuadrado del color
        img_square = get_square(imgrey_d, corners(:,:,j));
        %Detecci n de la letra m s probable en el cuadrado
        letter = find_letter(double(img_square), templates, double(imgrey_d));
        fprintf(letter)
    end
    fprintf('\n')
end
```

Listing 5: Implementación *main*

Que, al ejecutarse con la imagen dada por la cátedra, produce la siguiente salida en consola:

```
>> main
Blue: AWBLPOQWY
Brown: RIZNCZYÑT
Green: JLPAMBJÑU
Pink: OQQRSXAYNFM
Yellow: TXVKDIAEAXB
```

Figure 10: Salida en consola al ejecutar la implementación.

Lo cual coincide con lo correspondiente a la imagen original.