Escola de Enxeñaría de Telecomunicación

Universidade Vigo

DETECTA LETRAS

Fundamentos de Sonido e Imagen

14 DE MAYO DE 2021

TELECO UVIGO
Samuel López Iglesias
Blas Fernández Álvarez
María Alejandra Álvarez Sepúlveda
Carlos Lagarón Real

Presentación:

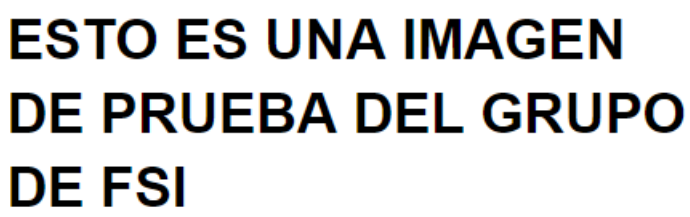
DetectaLetras es una función de MatLab creada por el grupo 16 de FSI como solución al problema propuesto por el profesorado de FSI sobre OCR (*Optical character recognition*).

En este informe se presentarán las tareas propuestas cumplidas, resultados, distintos análisis de errores, posibles soluciones y algunos casos de uso.

Tareas cumplidas:

(Todas las imágenes utilizadas fueron generadas por el grupo, siendo utilizadas en formato uint8 en todo momento)

Para recrear las tareas cumplidas utilizaremos esta imagen de prueba:



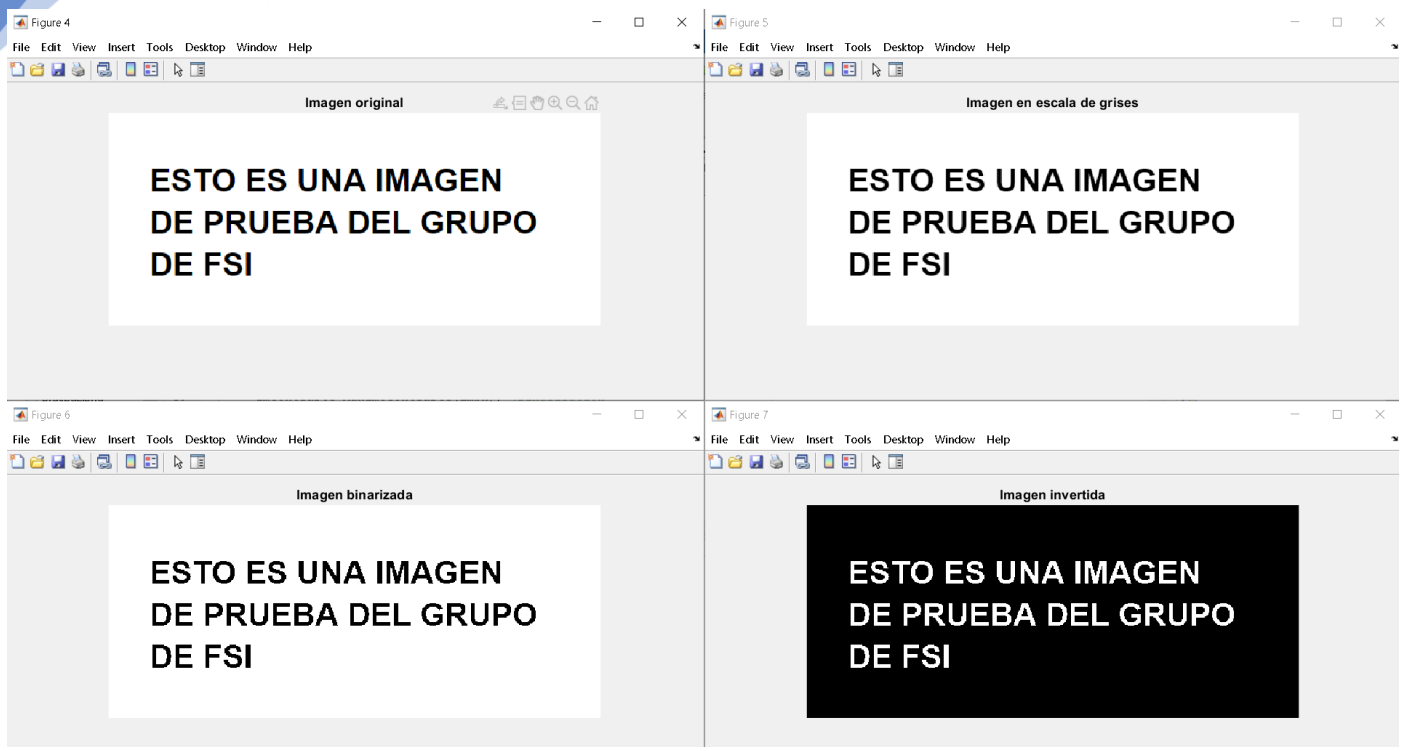
El programa se ejecuta mediante la llamada a la función *DetectaLetras('nombre_archivo.formato')*

1. **TAREA 1:** Binarización

   En este apartado se pedía convertir la imagen en lógica (falso equivale a negro, verdadero equivale a blanco).

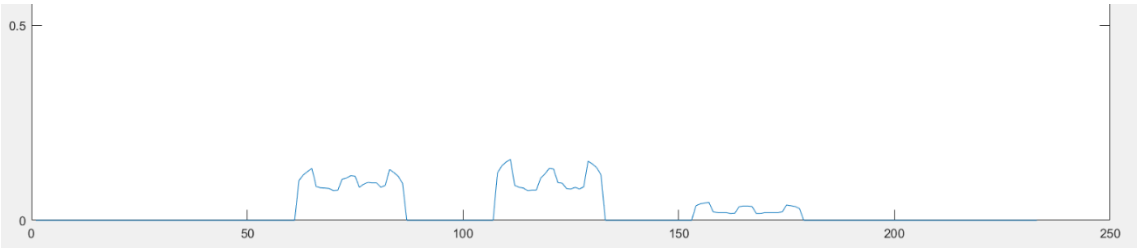
   Para ello cargamos la imagen original, la pasamos a escala de grises, aplicamos la función MaxContraste de la práctica 2 de imagen y la invertimos.

```
im = imread(imc);
figure;imshow(im);title('Imagen original');
R1 = im(:,:,1);
G1 = im(:,:,2);
B1 = im(:,:,3);
imbn = 0.3*R1+0.59*G1+0.11*B1;
figure;imshow(imbn);title('Imagen en escala de grises');
imcontraste=MaximoContraste(imbn); %Binarizacion
figure;imshow(imcontraste);title('Imagen binarizada');
imneg=255-imcontraste;
figure;imshow(imneg);title('Imagen invertida');
```

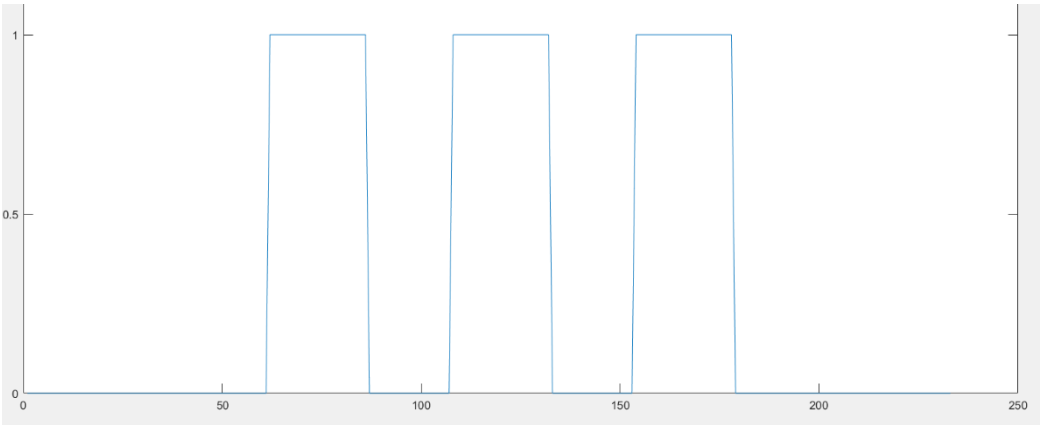


2. **TAREA 2:** Segmentar filas

Sumamos las filas de la matriz y obtenemos un vector que las define claramente.



Utilizamos un umbral para detectar las filas.



Mediante bucles anidados buscamos el principio y fin de los índices de cada fila.

```
sumafilas = sum(imneg,2);
sumafilasnormalizada = (sumafilas/(410e3));
plot(sumafilasnormalizada);ylim([0 1.5]);

for i=1:length(sumafilasnormalizada)
    if sumafilasnormalizada(i,:)>0.01
        sumafilasnormalizada(i,:)=1;
    end
end
plot(sumafilasnormalizada);ylim([0 1.5]);

j=1;
while (j<length(sumafilasnormalizada))

    while (sumafilasnormalizada(j,:) ==0)

        if(j==length(sumafilasnormalizada))

            break
        else
            j=j+1;
        end
    end

    if(j==length(sumafilasnormalizada))
        break
    else
        punto_inicio_fila = j;
    end

    while (sumafilasnormalizada(j,:) ==1)

        j=j+1;

    end

    punto_final_fila = j;

    j = j+3;
imfila = imneg(punto_inicio_fila:punto_final_fila,:);
figure();imshow(imfila);
```

Enseñando por pantalla las filas como imágenes parciales.

3. **TAREA 3**: Segmentar caracteres

De la misma forma que para extraer las filas, extraemos los caracteres dentro de cada fila, con un bucle de la forma del anterior, sumando previamente las columnas.

```
k=1;
while (k<length(sumacolumnasnormalizada))
    espacio_negro=0;

    while (sumacolumnasnormalizada(:,k) <0.1)

        if(k==length(sumacolumnasnormalizada))
            break
        else
            k=k+1;
            espacio_negro = espacio_negro+1;
        end
    end

    if(k==length(sumacolumnasnormalizada))
        break
    else
        punto_inicio_letra = k;
    end

    if(espacio_negro>=7)

         fprintf(" ");
    else

    end
    while (sumacolumnasnormalizada(:,k) >0.1)

        if(k==length(sumacolumnasnormalizada))
            break
        else
            k=k+1;
        end

    end

    punto_final_letra = k;

    k = k+1;


car = imfila(:,punto_inicio_letra:punto_final_letra);
```

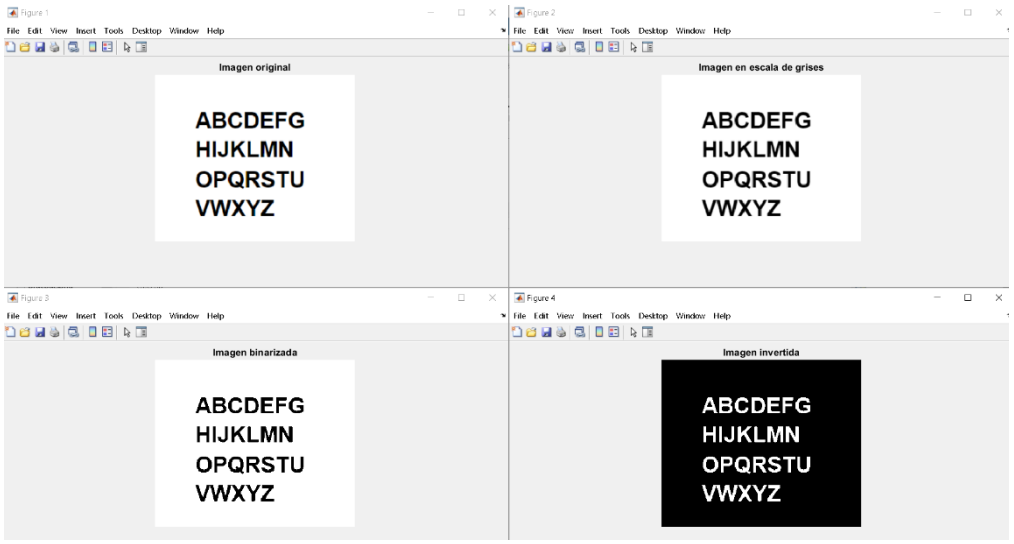
4. **TAREA 4:** Preprocesar caracteres

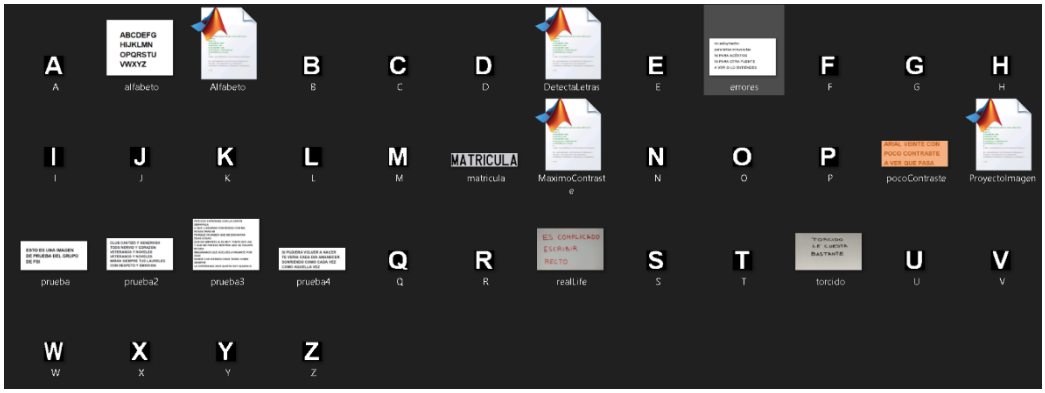
De los caracteres obtenidos en la tarea 3, los reescalamos de manera cuadrada, a un tamaño de (32x32) píxeles.

```
[m,n]=size(car);
carcuadrada = [zeros(m,fix((m-n)/2)) car zeros(m,fix((m-n)/2))];
[~,n1]=size(carcuadrada);
carcuadrada2 = [zeros(1,n1);carcuadrada;zeros(1,n1)];
[m2,~]=size(carcuadrada2);
carcuadrada3 = [zeros(m2,1) carcuadrada2 zeros(m2,1)];
carNN = imresize(carcuadrada3,[32 32]);
```

5. **TAREA 5:** Crear alfabeto

Mediante la función *Alfabeto* externa a *DetectaLetras* cargamos una imagen con todas las letras del alfabeto y generamos una base de datos con una imagen individual para cada caracter, los cuales obtenemos aprovechando el código de las tareas 2, 3 y 4. Estas serán utilizadas para realizar las comparaciones con los caracteres obtenidos en las imágenes de prueba.





Base de datos

6. **TAREA 6:** Función de reconocimiento

En este caso optamos por implementar la función directamente dentro de *DetectaLetas* desde donde en primer lugar llamamos a *Alfabeto* y cargamos todas las letras en forma de vector.

```
Alfabeto;
A=imread('A.png');
B=imread('B.png');
C=imread('C.png');
D=imread('D.png');
E=imread('E.png');
F=imread('F.png');
G=imread('G.png');
H=imread('H.png');
I=imread('I.png');
J=imread('J.png');
K=imread('K.png');
L=imread('L.png');
M=imread('M.png');
N=imread('N.png');
O=imread('O.png');
P=imread('P.png');
Q=imread('Q.png');
R=imread('R.png');
S=imread('S.png');
T=imread('T.png');
U=imread('U.png');
V=imread('V.png');
W=imread('W.png');
X=imread('X.png');
Y=imread('Y.png');
Z=imread('Z.png');
```

El proceso para reconocer el caracter consiste en coger cada uno por separado en cada paso del bucle y crear una matriz de correlación, en la cual el valor de sus posiciones será la correlación del caracter con cada una de las letras del alfabeto.

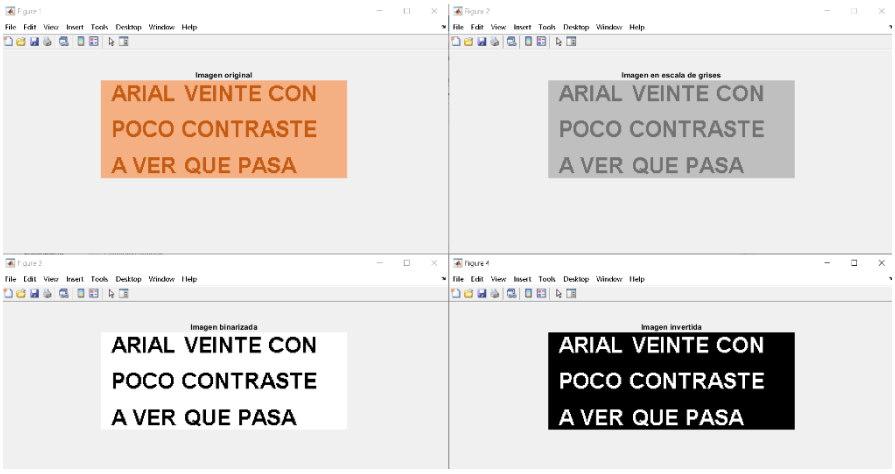
El caracter reconocido será aquel que tenga el valor máximo en la matriz de correlación, el cual se imprime por pantalla.

```
matriz_corr=[corr2(carNN,A), corr2(carNN,B), corr2(carNN,C), corr2(carNN,D), corr2(carNN,E), corr2(carNN,F),●●●

[~,b]=max(matriz_corr);

letra = b;
switch letra
    case 1
        fprintf("A");
    case 2
        fprintf("B");
    case 3
        fprintf("C");
    case 4
        fprintf("D");
    case 5
        fprintf("E");
    case 6
        fprintf("F");
    case 7
        fprintf("G");
    case 8
        fprintf("H");
    case 9
        fprintf("I");
    case 10
        fprintf("J");
    case 11
        fprintf("K");
        ⋮
```

```
>> DetectaLetras('prueba.png')
 ESTO ES UNA IMAGEN
 DE PRUEBA DEL GRUPO
 DE FSI
```

Resultados, análisis de errores y posibles soluciones:

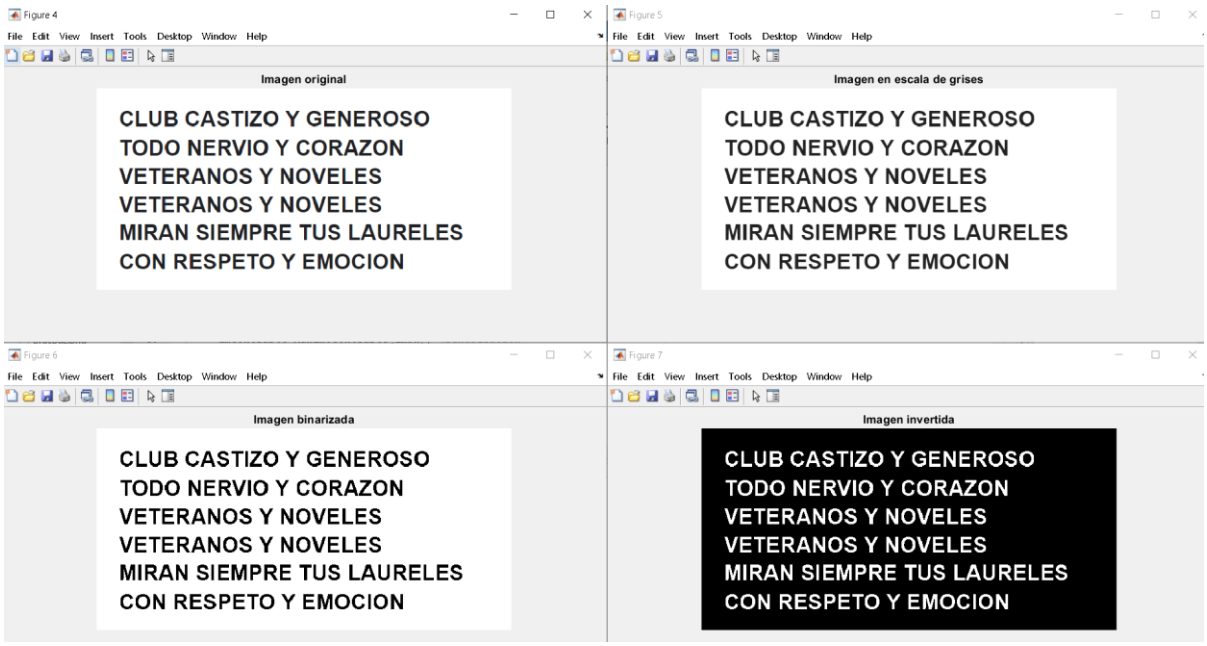
- **Imagen con poco contraste:**




```
>> DetectaLetras('pocoContraste.png')
 ARIAL VEINTE CON
 POCO CONTRASTE
 A VER QUE PASA
```

Vemos que el programa funciona bien con imágenes con poco contraste gracias a la función MaxContraste.

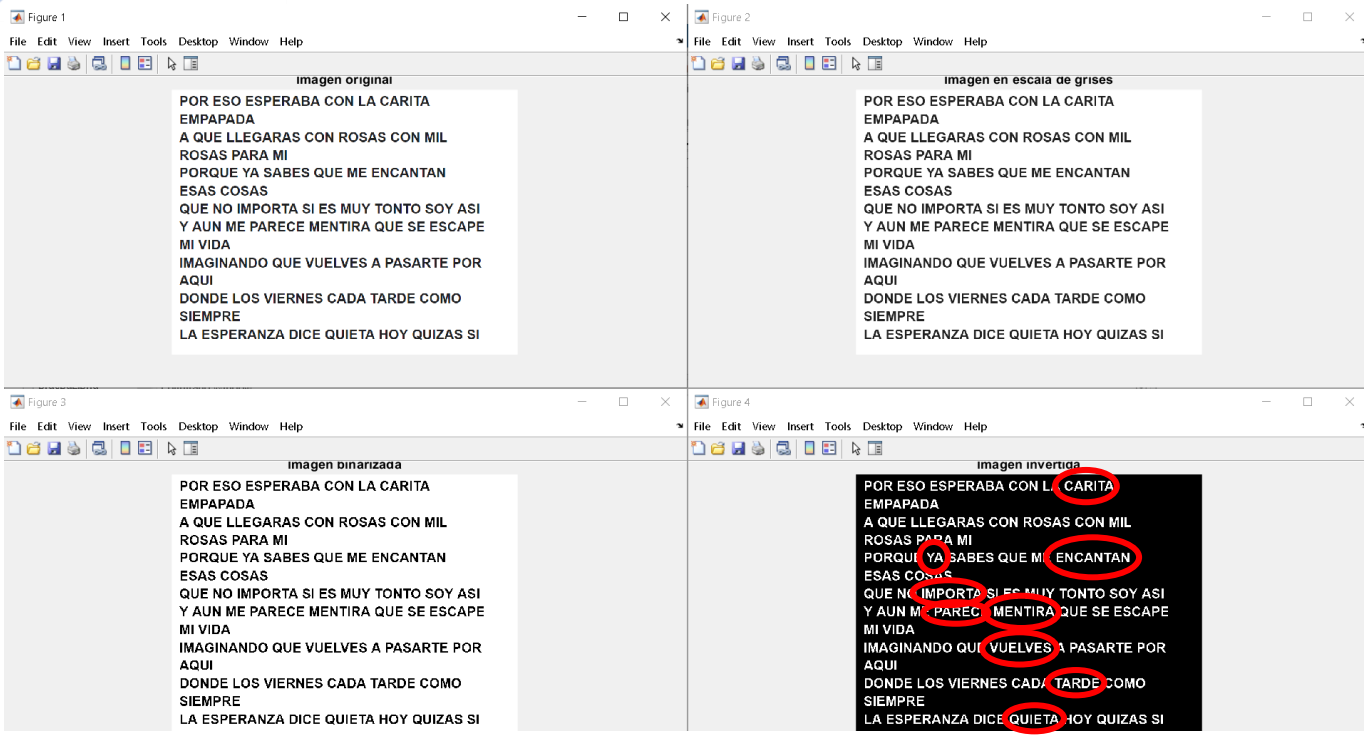
- **Sin superposición de caracteres:**




```
>> DetectaLetras('prueba2.png')
 CLUB CASTIZO Y GENEROSO
 TODO NERVIO Y CORAZON
 VETERANOS Y NOVELES
 VETERANOS Y NOVELES
 MIRAN SIEMPRE TUS LAURELES
 CON RESPETO Y EMOCION
```

Funciona correctamente.

- **Con superposición de caracteres:**



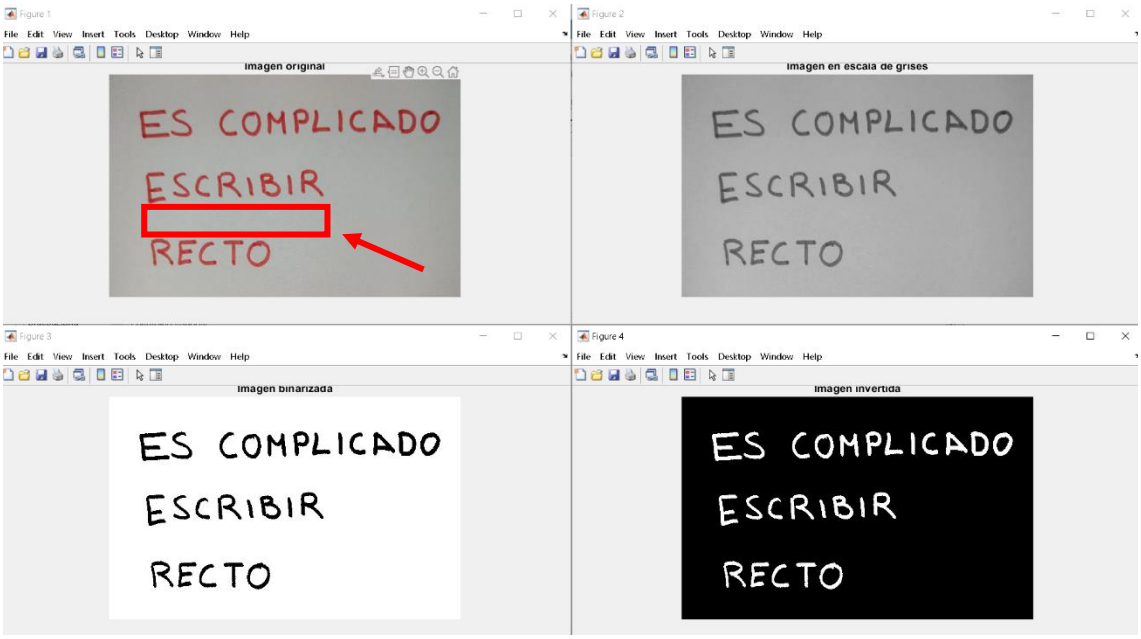

```
>> DetectaLetras('prueba3.png')
 POR ESO ESPERABA CON LA CARID
 EMPAPADA
 A QUE LLEGARAS CON ROSAS CON MIL
 ROSAS PARA MI
 PORQUE N SABES QUE ME ENCANPN
 ESAS COSAS
 QUE NO IMPORD SI ES MUY TONTO SOY ASI
 Y AUN ME MRECE MENTIRA QUE SE ESCAPE
 MI VIDA
 IMAGINANDO QUE VUEWES A PASARTE POR
 AQUI
 DONDE LOS VIERNES CADA PRDE COMO
 SIEMPRE
 LA ESPERANZA DICE QUIEP HOY QUIZAS SI
```

Con superposición de caracteres nos referimos a combinaciones del estilo TA, YA, PA o LV por ejemplo, donde claramente se observa que la A se “mete por debajo” de la T, la Y o la P. O bien la L “se mete por debajo de la V”. Esto provoca que no exista un cero de por medio entre los caracteres, por tanto en la detección tomará ambos caracteres como solo uno, y posteriormente busca la mayor correlación que da como resultado un único caracter, como es de esperar.

POSIBLE SOLUCIÓN: *una solución infalible (pero tediosa y poco eficiente), sería crear una base de datos con todas las palabras del abecedario y comparar palabras con palabras en lugar de letras con letras. De esta forma nos aseguramos que SIEMPRE habrá un valor 0 entre cada palabra.*

Esta solución es viable teniendo en cuenta de que los textos procesados son sintácticamente coherentes.

- **Imagen real:**



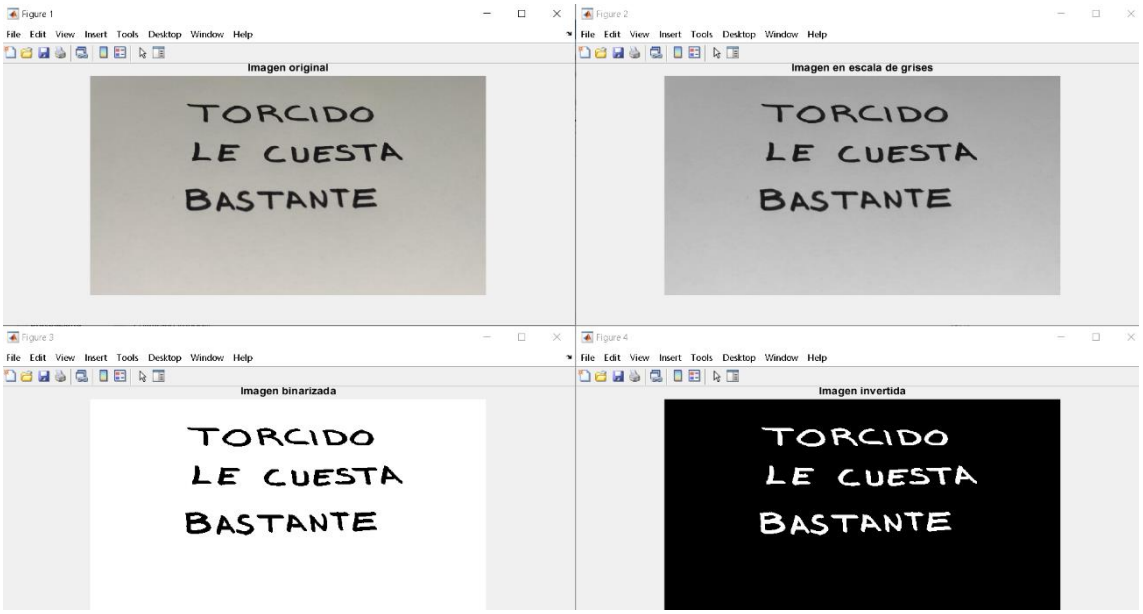

```
>> DetectaLetras('realLife.jpeg')

 ES C C H F A I A P V P

 L X A F I A I F

 R F L T O
```

La traducción no la realiza del todo mal, de hecho, se observa una correcta separación entre filas y algunos de los caracteres son satisfactoriamente traducidos. Pero es obvio que el programa no está optimizado para estos casos.

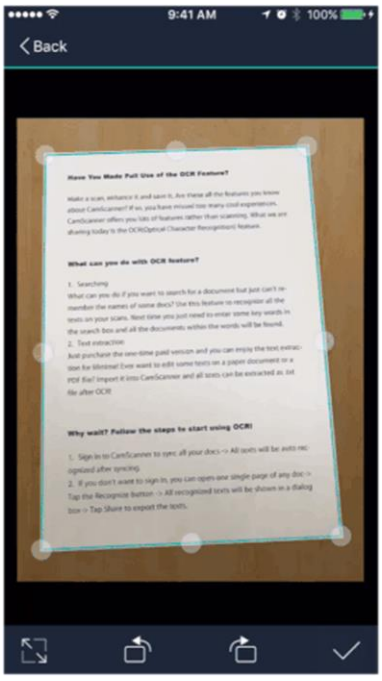
Si la escritura fuese totalmente perfecta, con las separaciones constantes y adecuadas e imitando la fuente Arial, funcionaría sin problema. Pero la escritura humana no es perfecta (y menos la mía).

POSIBLE SOLUCIÓN: se podría crear una inteligencia artificial de reconocimiento de caracteres, que fuese aprendiendo por si sola las infinitas posibilidades de combinaciones de fuente, tamaño y caligrafía de cada prueba.

- **Imagen real torcida:**




```
>> DetectaLetras('torcido.jpeg')

 T O BCI O O

 L E L U BD K

 B AS T K N T R
```

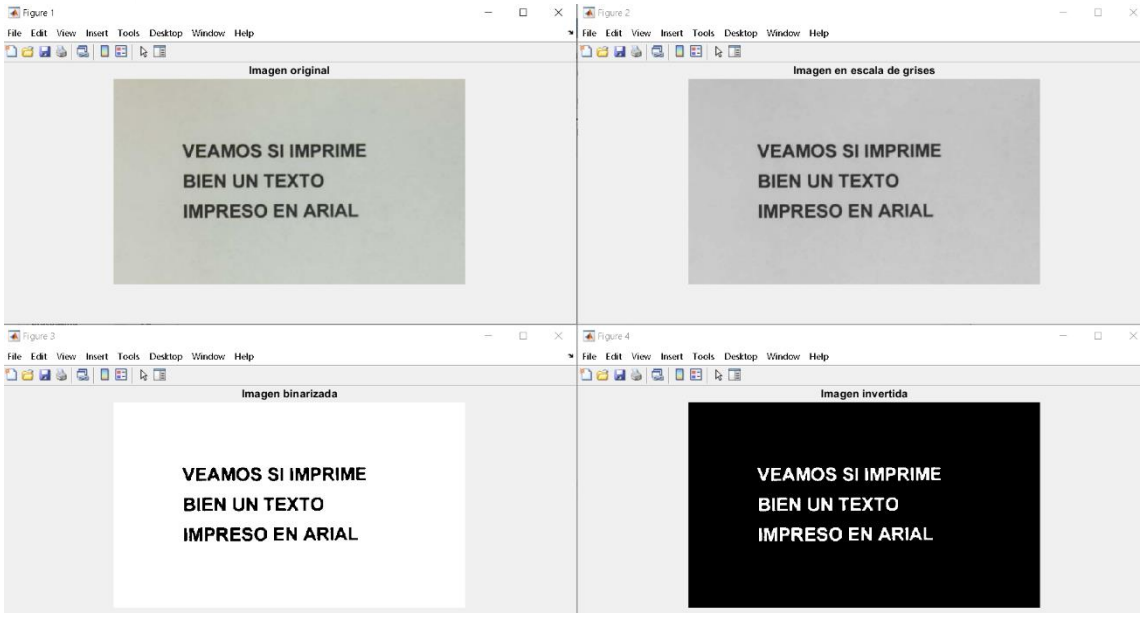


Nos encontramos en un caso parecido al anterior, la escritura no es la idónea, a pesar de ello, el programa es capaz de reconocer ciertos caracteres.

El problema de las imágenes inclinadas es que el programa pierde la perspectiva real, ve las palabras cercanas más grandes y estiradas que las alejadas.

POSIBLE SOLUCIÓN: se podría programar un pre-reescalado y pre-centrado de la imagen antes de procesarla. Es lo que hacen aplicaciones como CamScanner.

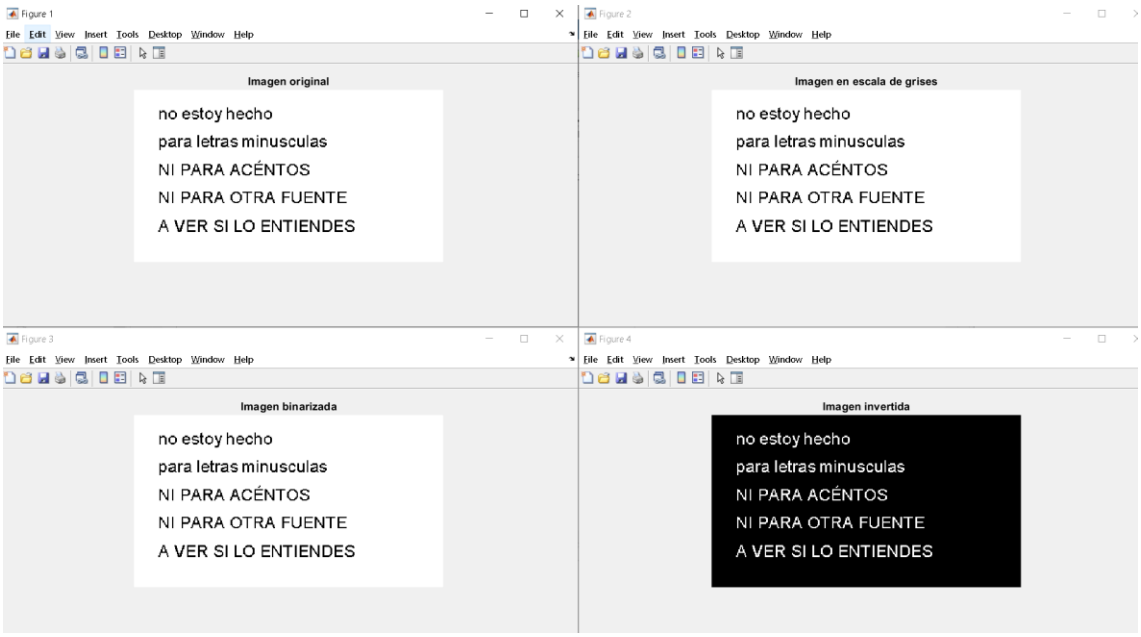
- **Texto impreso en Arial**




```
>> DetectaLetras('impreso.jpeg')

VEAMOS SI IMPRIME
BIEN UN TEXTO
IMPRESO EN ARIAL
```

Como comprobamos, lo traduce de manera correcta.

- **Futuras implementaciones:**



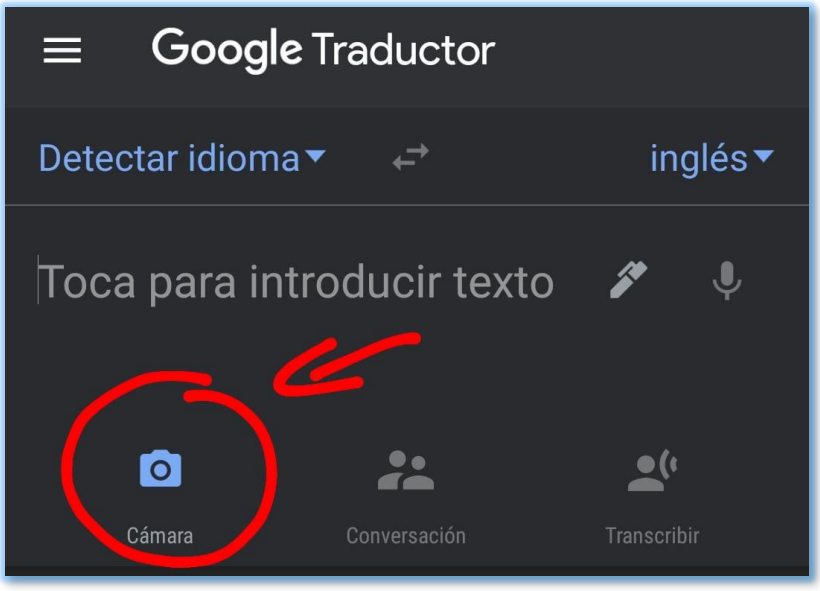
```
>> DetectaLetras('errores.png')

 IIIINVIIIYIINIIIIII



 IIUIU INIIUVIIIIIIIIVIIIIIUV


 NI PARA ACENIOS
 NI PARA OI RA FIIENI E
 A ITER SI I O ENI IENDES
```

En esta prueba utilizamos una fuente distinta a Arial, letras minúsculas y signos de acentuación simplemente para comentar como se podrían implementar estas características, ya que no es lo que se pedía en el proyecto.

POSIBLE SOLUCIÓN: vamos a recurrir a lo que ya comentamos antes, aumentar la base de datos, de tal forma que incluya letras minúsculas, combinaciones de vocales con signos de acentuación, signos ortográficos, y todas las fuentes posibles. Incluso podríamos traducir textos de otros sistemas de escritura como podría ser el chino, ruso, escandinavo o árabe de manera sencilla simplemente **aumentando la base de datos**.

Casos de uso:

El reconocimiento de textos lo vemos diariamente en muchísimas aplicaciones, como puede ser el reconocimiento de matrículas, digitalización de documentos escaneados o incluso en Google Translator donde podemos traducir a otros idiomas textos obtenidos a partir de una imagen.



Pero OCR sigue sujeto a varias limitaciones, como es la calidad de la imagen a transcribir, que debe ser medianamente aceptable, o textos manuscritos donde ya comprobamos su complejidad.

Conclusiones:

No ha sido un proyecto fácil de implementar, sobre todo la parte de obtener los índices de inicio y final de cada fila y caracter.

Pero gracias a ello, nos hacemos una idea de como funcionan hoy en día este tipo de sistemas OCR. Sin duda, cuando entremos en un parking pensaremos en ello.