

Ejercicio 4.

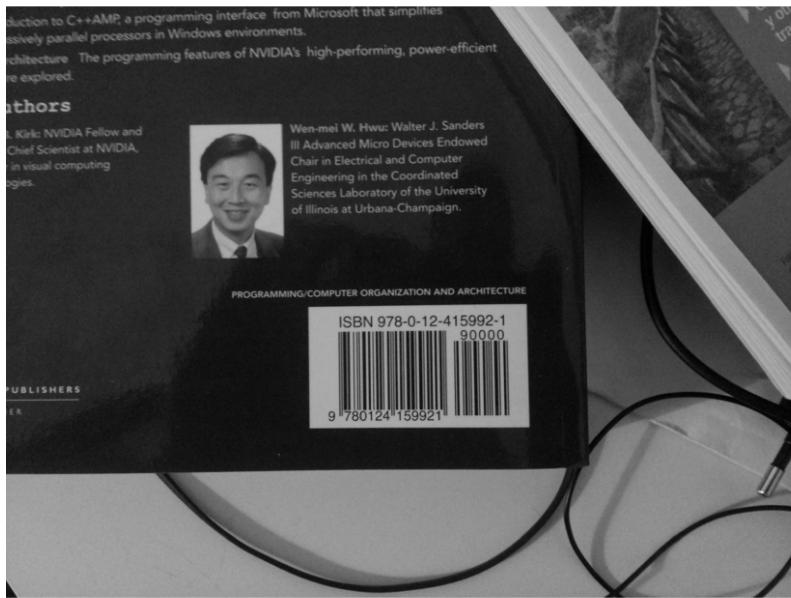
Desarrollar un programa en Matlab/Octave que permita **detectar códigos de barras** utilizando rutinas básicas de procesamiento de imagen. Para probar el algoritmo se proporciona un pequeño conjunto de imágenes junto con el enunciado (sección de recursos de moodle). Igualmente, el alumno podrá capturar sus propias imágenes y comprobar la efectividad del método desarrollado. Los pasos a seguir para tratar de detectar un código de barras se detallan a continuación:

1. **Reducir la resolución** de la imagen a una cuarta parte. No necesitamos imágenes de 8M (como las del dataset) para detectar códigos de barras.

```
clear all
A = imread("barcode3.jpg"); % 2448x3164
B = imresize(A, 0.25); % 612x816
```

2. **Convertir la imagen a escala de grises**

```
I = rgb2gray(B);
imshow(I)
```



3. **Calcular la magnitud del gradiente** para la imagen en escala de grises.

- Usar el operador Scharr para calcular el gradiente en la dirección x e y
- El resultado serán dos imágenes (gradientX, gradientY). Una con el gradiente para la dirección x y otra con el gradiente para la dirección y.
- Calcular la diferencia en valor absoluto de ambas imágenes (gradientX-gradientY)

```
%python code:
```

```
%def from_gradient(gray):
%    fieldx = cv2.Scharr(gray, cv2.CV_32F, 1, 0) / 15.36
%    fieldy = cv2.Scharr(gray, cv2.CV_32F, 0, 1) / 15.36
%
%    return VectorField(fieldx, fieldy)

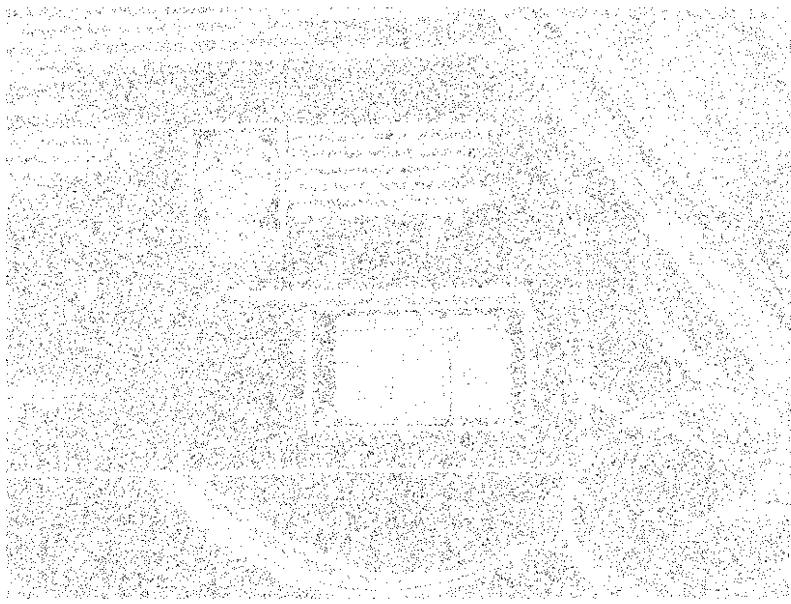
[gradientX,gradientY] = imgradientxy(I);
imshow(gradientX)
```



```
imshow(gradientY)
```



```
diff = abs(gradientX-gradientY);  
imshow(diff)
```



4. Para eliminar ruido en la imagen de gradiente resultante, aplicar un **filtro media** (tamaño de kernel 9x9 por ejemplo)

```
J = medfilt2(diff,[9 9]);  
imshow(J)
```

5. A continuación, **umbralizar el fotograma** resultante. Buscamos mantener la atención únicamente en aquellas zonas con un gradiente elevado. Todo valor de gradiente por debajo de 200 tomará valor cero. Todo valor por encima de 200 tomará valor 1.

```
%T = 255/200; % valor entre [0 1]
%BW = imbinarize(J,T);
BW = J > 200;
imshow(BW)
```



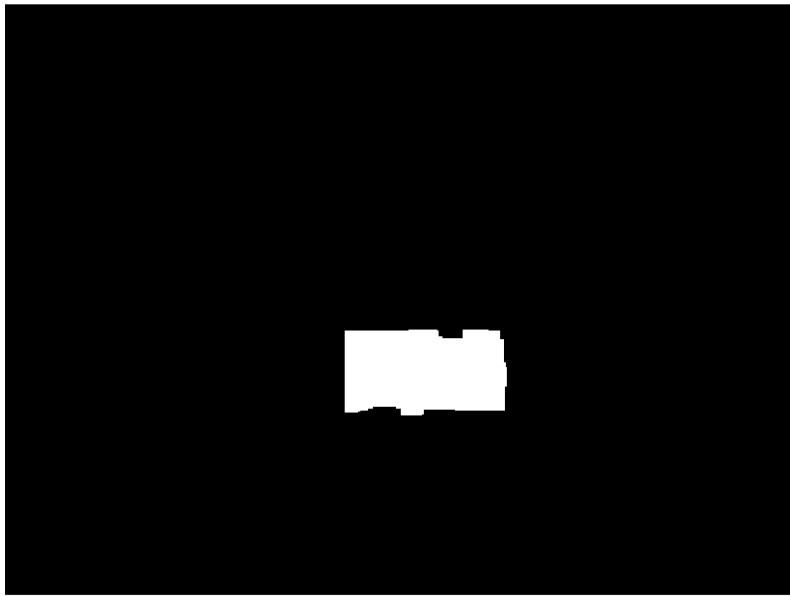
6. Aplicar una **operación de cierre** morfológico (imclose) utilizando un elemento estructurante de 21x7. El objetivo es eliminar las separaciones existentes entre las barras y simplificar así la tarea de extraer el contorno del código de barras.

```
SE = strel('rectangle',[7 21]);
BW = imclose(BW,SE);
imshow(BW)
```



7. Aplicar **cuatro operaciones de erosión y cuatro de dilatación** empleando un elemento estructurante cuadrado de 3x3. El objetivo es eliminar pequeños blobs en la imagen.

```
SE2 = strel('square', 3);
for i=1:4
    BW = imerode(BW,SE2);
end
for j=1:4
    BW = imdilate(BW,SE2);
end
imshow(BW)
```



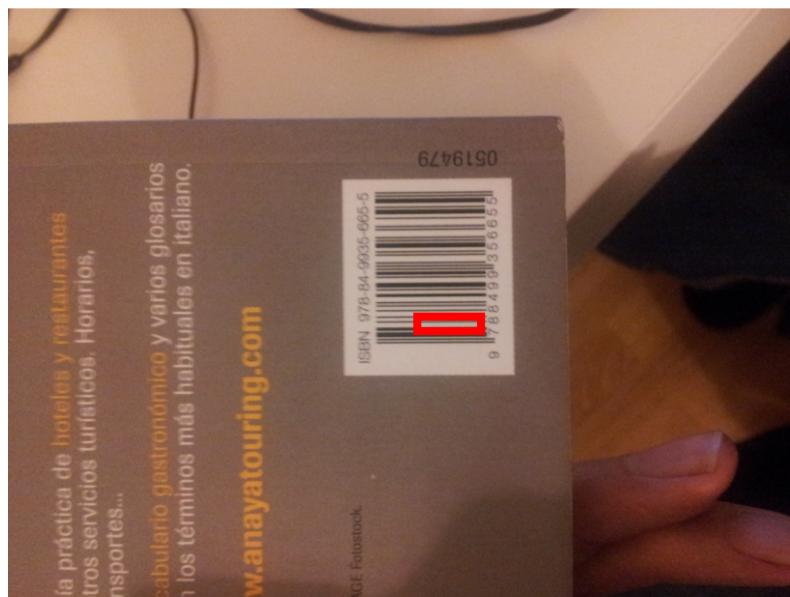
8. **Calcular los contornos** en la imagen resultante tras aplicar las operaciones morfológicas. Seleccionar aquel de mayor tamaño y dibujar el bounding box que lo contiene en la imagen original. El programa desarrollado deberá presentar al menos una función llamada barCodeDetector que reciba como argumento una imagen de entrada y presente en pantalla el resultado de la detección.

```
%contours = imcontour(BW);
%imshow(contours)
st = regionprops(BW, 'BoundingBox', 'Area' );
[maxArea, indexOfMax] = max([st.Area]);
imshow(B)
rectangle('Position',[st(indexOfMax).BoundingBox(1),st(indexOfMax).BoundingBox(2),st(indexOfMax).BoundingBox(3),st(indexOfMax).BoundingBox(4)])
```

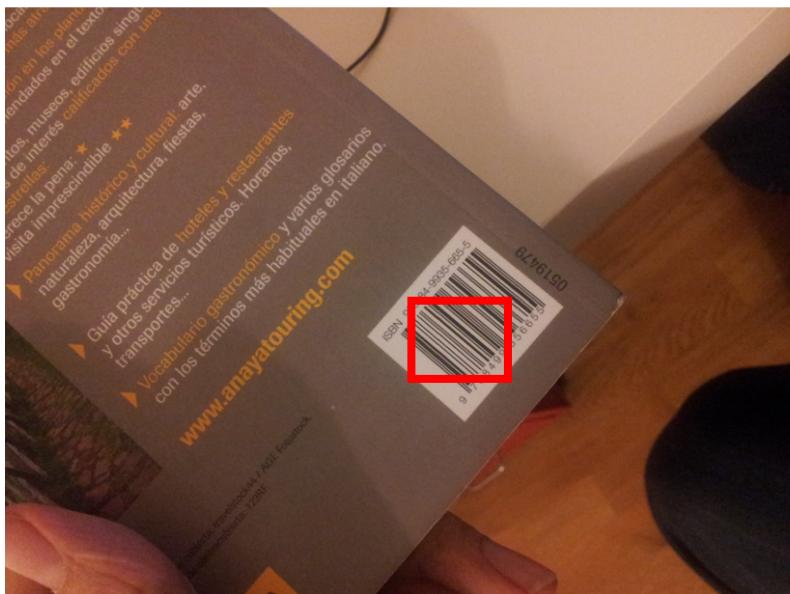


% Testing

```
A1 = imread("barcode1.jpg");
A2 = imread("barcode2.jpg");
A3 = imread("barcode3.jpg");
A4 = imread("barcode4.jpg");
A5 = imread("barcode5.jpg");
A6 = imread("barcode6.jpg");
barCodeDetector(A1)
```



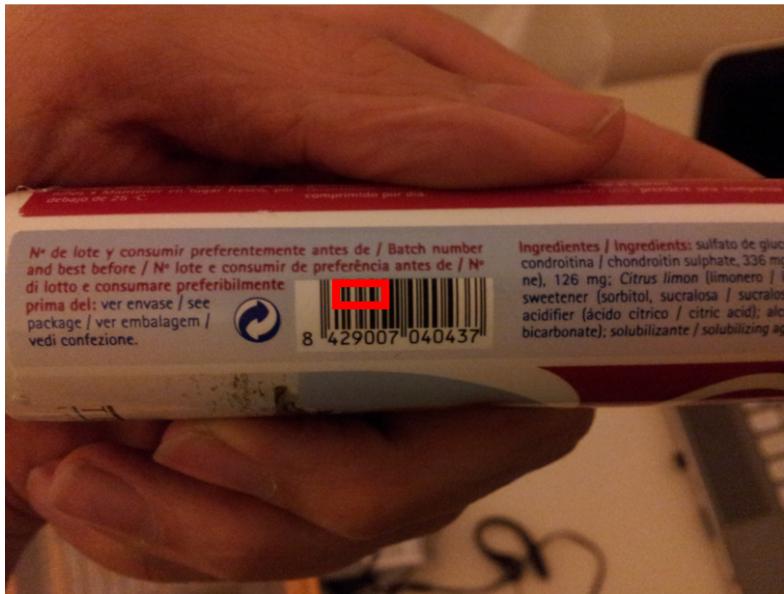
barCodeDetector(A2)



barCodeDetector(A3)



barCodeDetector(A4)



barCodeDetector(A5)



barCodeDetector(A6)

