



FIB

Facultat d'Informàtica
de Barcelona

Departament d'Enginyeria de Sistemes,
Automàtica i Informàtica Industrial

UNIVERSITAT POLITÈCNICA DE CATALUNYA

VISIÓ PER COMPUTADOR

Exercici 4 de Laboratori

Facultat d'Informàtica de Barcelona

Adrian Cristian Crisan

Filip Gedung Dorm

Pablo Vega Gallego

Barcelona, Octubre de 2021

Índice

Introducción.....	1
Binarización global.....	2
Resultados con otras imágenes.....	11
Binarización local	12
Resultados con otras imágenes.....	12

Introducción

En esta sesión haremos una pequeña introducción a las técnicas de binarización y segmentación de imágenes. En concreto se trabajarán los siguientes conceptos:

- Binarizaciones globales.
- Binarización local mediante la función *colfilt*.
- Segmentación por agrupamiento de píxeles en imágenes binarizadas.

El objetivo de la sesión es hacer una pequeña aplicación para capturar i contrastar documentos usando el móvil, en escenarios donde la iluminación no está del todo controlada.

Las imagen principal y originales (sin aplicarle nada) usada para esta práctica:

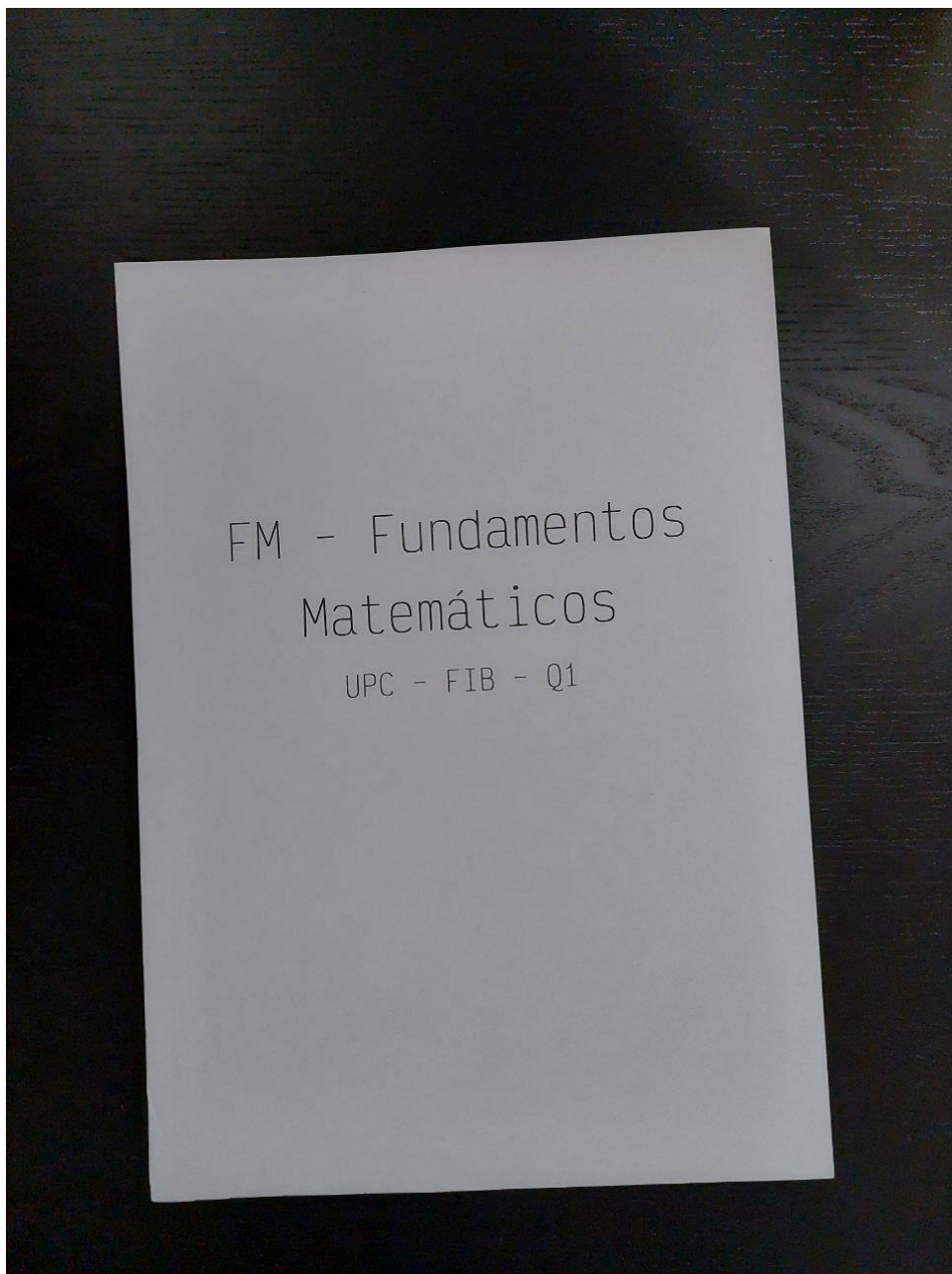


Figura 1 – Imagen 1 (original)

Binarización global

Lo primero que hacemos es leer la imagen i la pasamos a blanco y negro.

```
imagen_original = imread('foto1.jpeg');  
imagen_grises = rgb2gray(imagen_original);
```

Conseguimos el histograma de la imagen a escala de grises

```
histogramas_grises = imhist(imagen_grises);  
plot(histogramas_grises);
```

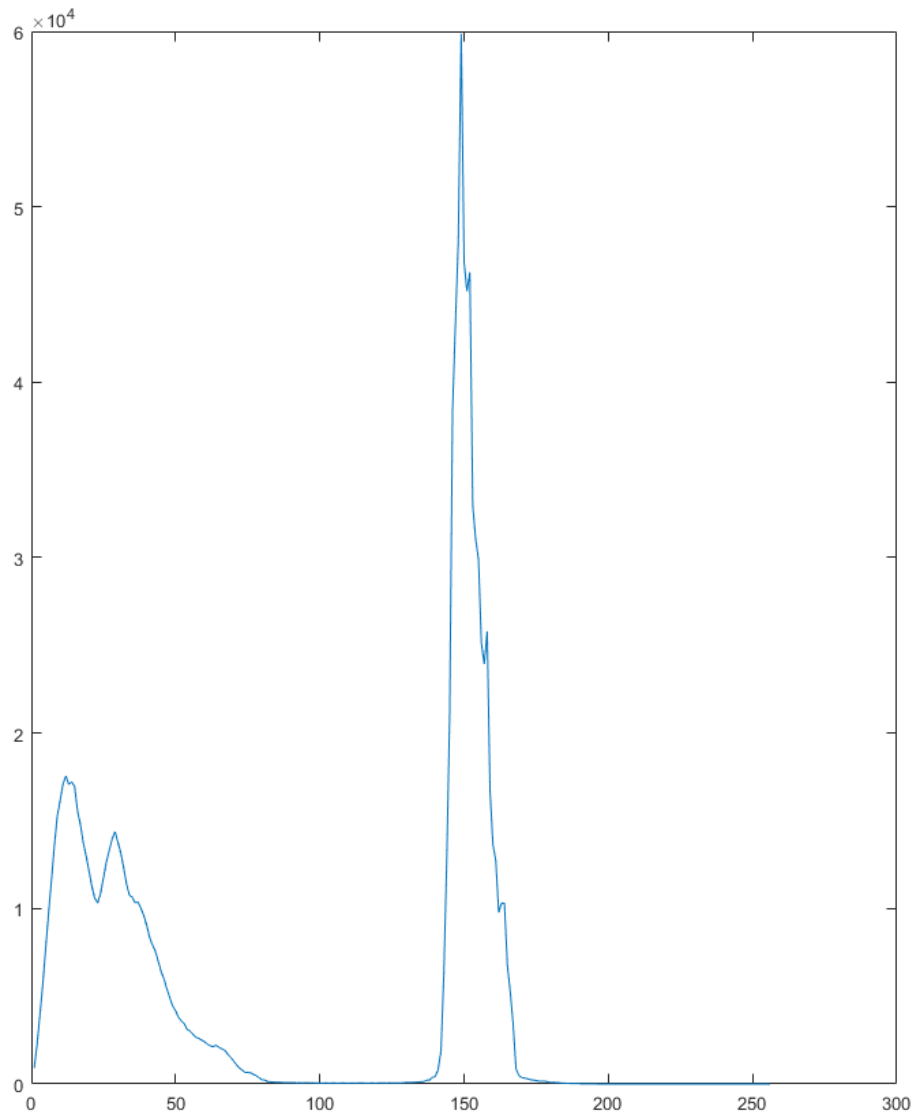


Figura 2.1 – Histograma de la imagen 1 a escala de grises

Calculamos el treshhold global y obtenemos la imagen resultante de quedarnos con los valores de rango superior al umbral.

```
otsu = otsuthresh(histogramas_grises) * 255;  
imagen_blanco_y_negro = imagen_grises > otsu;  
imshow(imagen_blanco_y_negro);
```

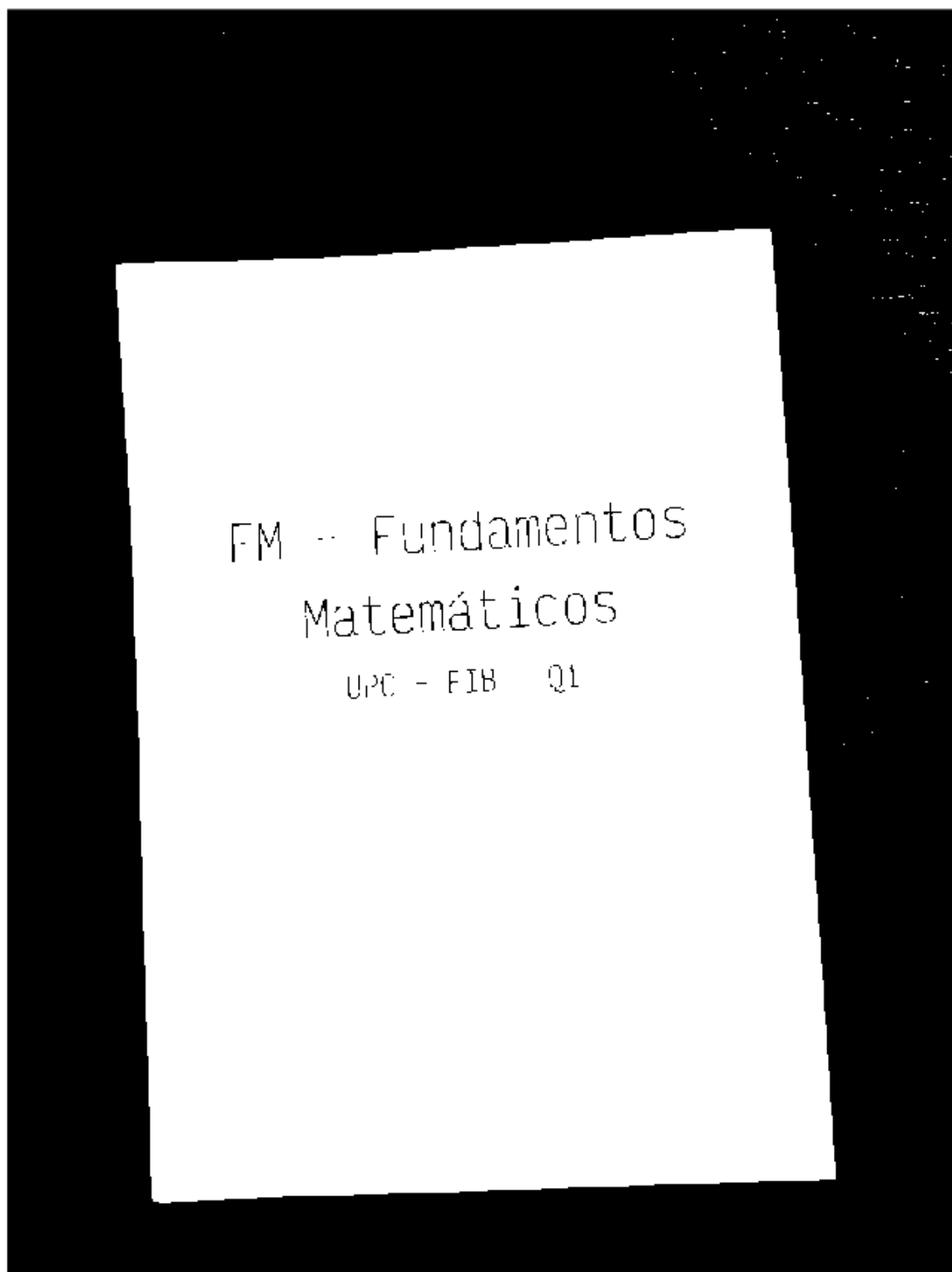


Figura 2.3 – Imagen tras aplicar el treshold

Eliminamos las pequeñas imperfecciones que quedan en la imagen

```
SE = strel('disk', 1);  
imagen_abierta = imopen(imagen_blanco_y_negro, SE);  
imshow(imagen_abierta);
```



Figura 2.4 – Imagen tras eliminar pequeñas imperfecciones

Resaltamos los caracteres para mayor precisión

```
imagen_negada = ~imagen_abierta;  
imagen_negada_dilatada = imdilate(imagen_negada, SE);  
imagen_abierta_dilatada = ~imagen_negada_dilatada;  
imshow(imagen_abierta_dilatada);
```

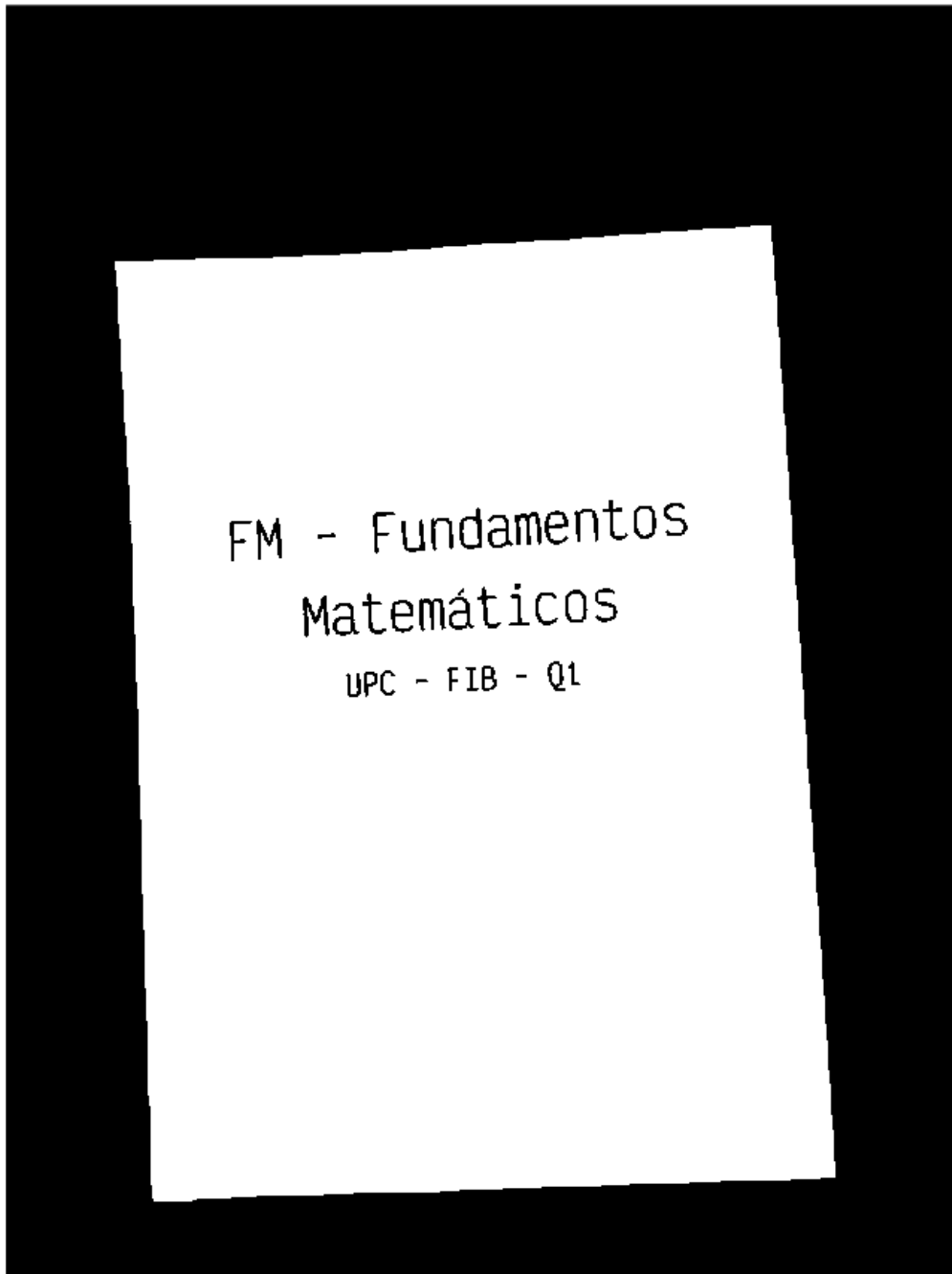


Figura 2.5 – Imagen con caracteres resaltados

Rotamos la imagen para dejar la hoja lo más horizontal posible

```
theta = rotacion(imagen_abierta_dilatada);  
imagen_rotada = imrotate(imagen_abierta_dilatada, theta);  
  
C = corner(imagen_rotada);  
imshow(imagen_rotada)  
hold on  
plot(C(:,1),C(:,2), 'r*');  
hold off
```



Figura 2.6 – Imagen rotada

Función usada para saber el ángulo que hay que rotar:

```
function theta = rotacion(imagen)
    [y, x] = find(imagen);

    nx = x - mean(x);
    ny = y - mean(y);

    nx = nx';
    ny = ny';

    c = cov(nx, ny);
    [evector, evalues] = eig(c);

    [~, ind] = max(diag(evalues));
```



```
theta = -pi/2-atan2(evectors(ind, 2), evectors(ind, 1));  
end
```

Establecemos el área que debe ser recortada

```
xmin = min(transpose(C(:, 1)));  
ymin = min(transpose(C(:, 2)));  
xmax = max(transpose(C(:, 1)));  
ymax = max(transpose(C(:, 2)));  
  
imshow(imagen_rotada)  
hold on  
plot([xmin, xmax], [ymin, ymax], 'r');  
hold off
```

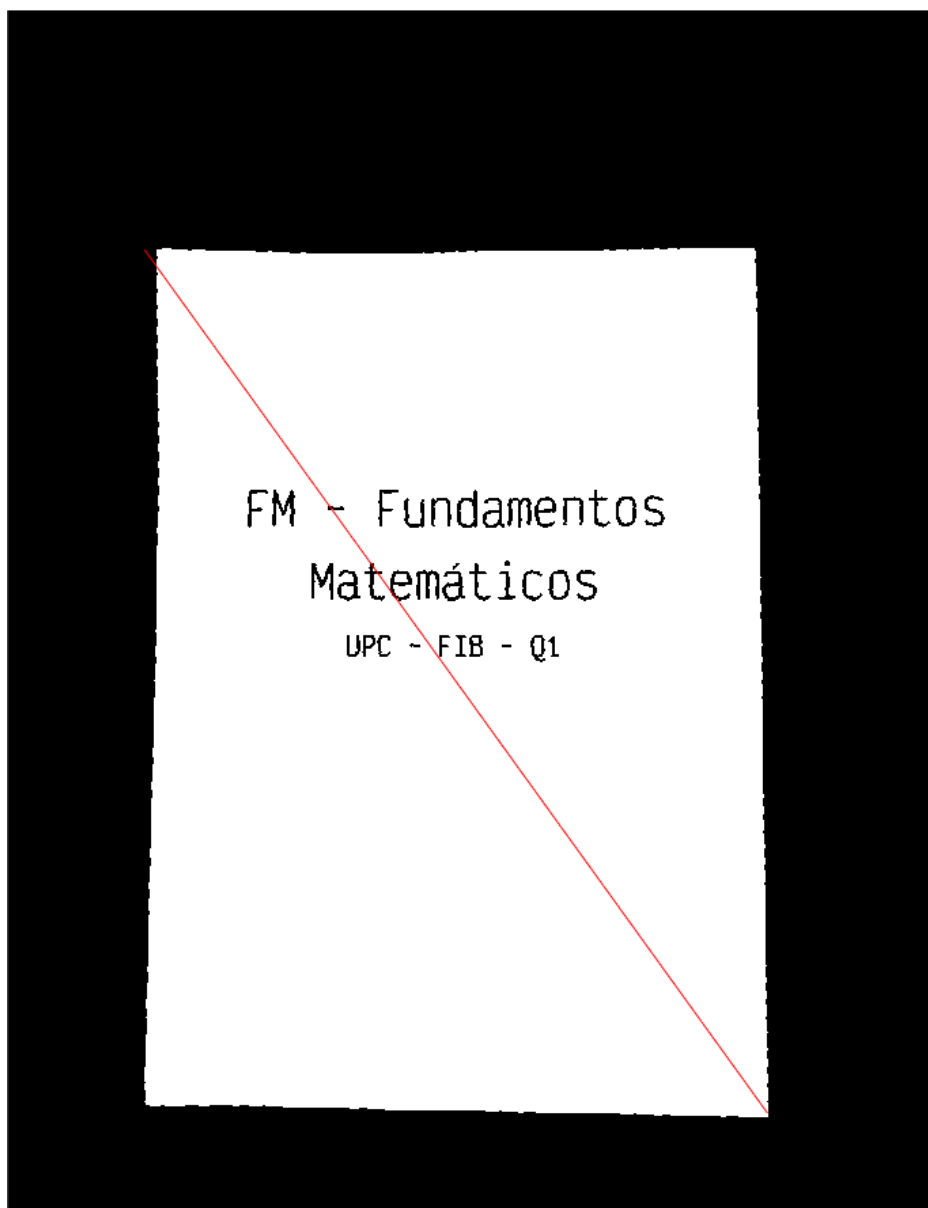


Figura 2.6 – Imagen con la diagonal de corte

Recortamos la imagen

```
imagen_cortada = imcrop(imagen_rotada, [xmin ymin xmax-xmin ymax-ymin]);  
imshow(imagen_cortada);
```



Figura 2.7 – Imagen recortada

Eliminamos los restos del fondo

```
tamanoc = size(imagen_cortada)  
i = 1;  
blanco = false;  
while ~blanco && i < tamanoc(1)  
    j = 1;  
    while ~blanco && j < tamanoc(2)  
        blanco = imagen_cortada(i, j);  
        imagen_cortada(i, j) = 1;  
    end  
    i = i + 1;  
end
```

```

        j = j + 1;
    end
    blanco = false;
    i = i + 1;
end

i = tamanoc(1);
blanco = false;
while ~blanco && i >= 1
    j = tamanoc(2);
    while ~blanco && j >= 1
        blanco = imagen_cortada(i, j);
        imagen_cortada(i, j) = 1;
        j = j - 1;
    end
    blanco = false;
    i = i - 1;
end

j = 1;
blanco = false;
while ~blanco && j < tamanoc(2)
    i = 1;
    while ~blanco && i < tamanoc(1)
        blanco = imagen_cortada(i, j);
        imagen_cortada(i, j) = 1;
        i = i + 1;
    end
    blanco = false;
    j = j + 1;
end

j = tamanoc(2);
blanco = false;
while ~blanco && j >= 1
    i = tamanoc(1);
    while ~blanco && i >= 1
        blanco = imagen_cortada(i, j);
        imagen_cortada(i, j) = 1;
        i = i - 1;
    end
    blanco = false;
    j = j - 1;
end

SE1 = [1, 1; 1, 1];
imagen_cortada = ~imerode(~imagen_cortada, SE1);
imagen_cortada = ~imdilate(~imagen_cortada, SE1);

imshow(imagen_cortada)

```

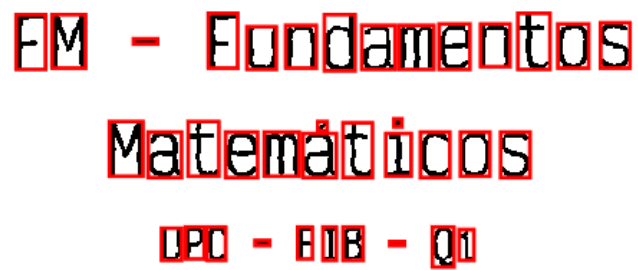
FM - Fundamentos Matemáticos

UPC - FIB - Q1

Figura 2.8 – Imagen preprocesada para leer los símbolos

Encuadramos los símbolos detectados

```
labeledImage = bwconncomp(~imagen_cortada);  
                measurements = regionprops(labeledImage, 'BoundingBox');  
for k = 1 : length(measurements)  
    thisBB = measurements(k).BoundingBox;  
    rectangle('Position', [thisBB(1),thisBB(2),thisBB(3),thisBB(4)],...  
            'EdgeColor','r','LineWidth',2 )  
end
```

The image shows the text 'FM - Fundamentos Matemáticos' and 'LPC - EIB - Q1' where each character is enclosed in a red rectangular box, indicating symbol detection. The text is centered on a white background.

FM - Fundamentos
Matemáticos
LPC - EIB - Q1

Figura 2.9 – Imagen final con los símbolos resaltados

Nos da un resultado de 37 símbolos detectados, realmente el número de símbolos que hay son 35, debido a que el punto de la i y las tildes las detecta cómo símbolos por separado, pero la detección se puede considerar satisfactoria.

[Resultados con otras imágenes](#)

Binarización local

Resultados con otras imágenes