Universitat Politècnica de Catalunya

Visió per Computador
Grau en Enginyeria Informàtica

EXERCICI 6 DE LABORATORI

IMAGE BINARIZATION

Autor:
Daniel DONATE

Professor:
Manel FRIGOLA
Q2 Curs 2020-2021

6 d'abril del 2021

*Nota per al professor: Les imatges d'aquest document s'han escalat per a què s'ajustin al format del mateix, de manera que han perdut una notable resolució. Si vols comprovar que els resultats, efectivament, són correctes, te les puc enviar per correu quan vulguis. Gràcies!

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Enunciat de l'exercici

En aquesta sessió treballarem amb tècniques de binarització i segmentació d'imatges. L'objectiu serà elaborar una petita aplicació de VC que ens permetrà contrastar, retallar i comptar (de forma aproximada) el nombre de caràcters d'una imatge d'un document captada amb el mòbil, en un entorn on la il·luminació no està gaire controlada.

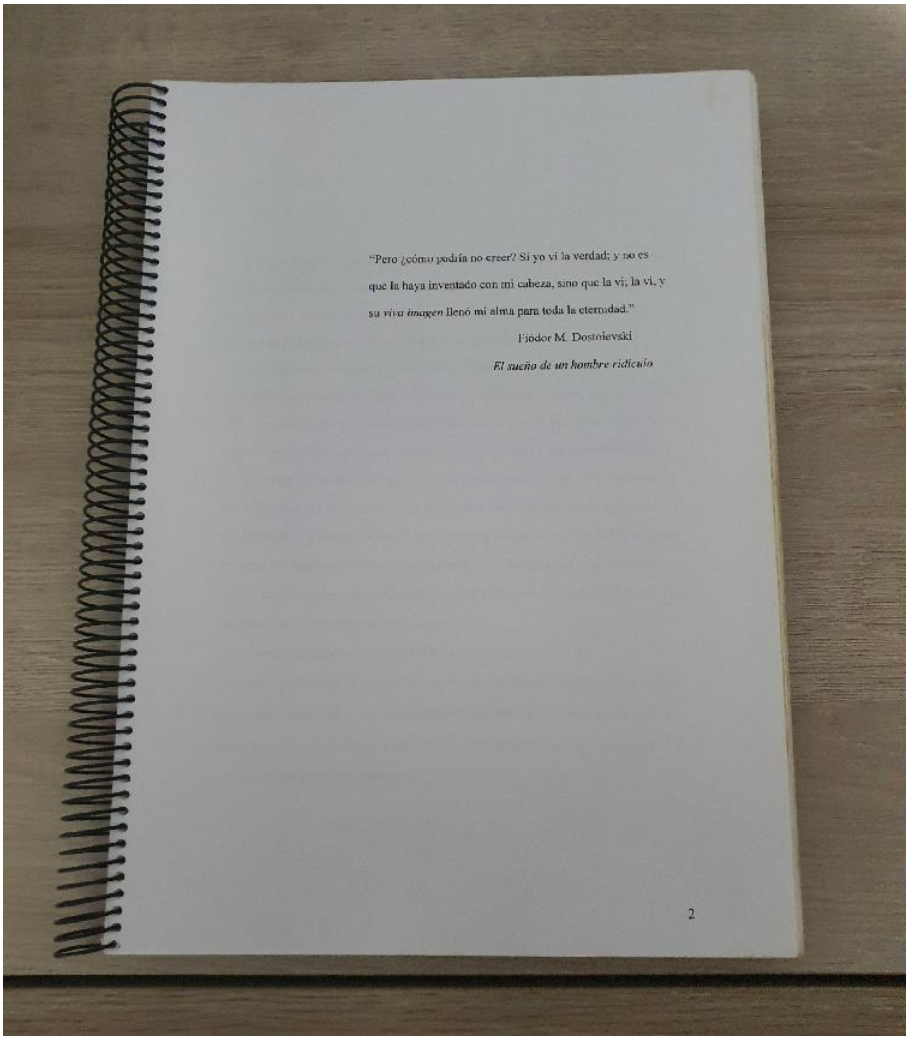

Figura 1: Imatge RGB original que farem servir al llarg de la pràctica

El que veurem serà, primer, una forma d'aplicar una binarització global de la imatge. Després, intentarem realitzar la binarització de forma local, mitjançant la funció *colfilt*, de MATLAB. I, finalment, segmentarem la imatge per extreure i fer un recompte dels caràcters del text.

Binarització global de la imatge

Notem que la idea d'aquest enfocament és trobar un llindar d'intensitat en la imatge original en escala de grisos, de tal manera que el full quedi binaritzat a blanc i tota la resta a negre. El procediment que s'ha fet servir per trobar aquest *threshold* ha estat el Mètode de Otsu, que explicarem a continuació, per una qüestió de completesa.

El mètode de Otsu, com altres, escull el *threshold* examinant l'histograma de la imatge. La idea bàsica (en la seva versió més simple, que és la que ens ocupa) és buscar dos pics— que

representen valors de píxels de primer pla i de fons, respectivament—i escollir un punt entre aquests dos pics com a *threshold* de la binarització. Seguint la notació del *paper* original, observem que els píxels de la imatge poden prendre els valors $i = 1, \dots, L$. El recompte de l'histograma per als píxels amb valor i és n_i, i la *freqüència* associada és $p_i = n_i/N$, on N és el nombre de píxels de la imatge. La cerca d'un *threshold* es pot formular com el problema de dividir els píxels de la imatge en dues *classes*: C_0 i C_1, on C_0 és el conjunt de píxels amb valors d'intensitat $[1, \dots, k]$, i C_1 és el conjunt de píxels amb valors d'intensitat $[k+1, \dots, L]$.

Les freqüències totals de cada classe, C_0 i C_1 són, respectivament, ω_0 i ω_1:

$$\omega_0 = \sum_{i=1}^{k} p_i = \omega(k) \qquad \omega_1 = \sum_{i=k+1}^{L} p_i = 1 - \omega_0(k)$$

Les mitjanes de les classes, μ_0 i μ_1 són el valor mitjà dels píxels en C_0 i C_1 i venen donades per:

$$\mu_0 = \sum_{i=1}^{k} i \cdot p_i / \omega_0 = \mu(k)/\omega(k) \qquad \mu_1 = \sum_{i=k+1}^{L} i \cdot p_i / \omega_1 = \frac{\mu_T - \mu(k)}{1 - \omega(k)},$$

on $\mu(k) = \sum_{i=1}^{k} i \cdot p_i$ i μ_T, el valor de píxel mitjà de tota la imatge, és: $\mu_T = \sum_{i=1}^{L} i \cdot p_i$.

Anàlogament, les variàncies de les classes, σ_0^2 i σ_1^2 són, respectivament:

$$\sigma_0^2 = \sum_{i=1}^{k} (i - \mu_0)^2 p_i / \omega_0 \qquad \sigma_1^2 = \sum_{i=k+1}^{L} (i - \mu_1)^2 p_i / \omega_1$$

Otsu parla de tres mesures de "bona separabilitat de classes": variància dintre de la classe (λ), variància entre classes (κ) i variància total (η), que venen donades per aquestes tres expressions:

$$\lambda = \sigma_B^2 \qquad \kappa = \sigma_T^2/\sigma_W^2 \qquad \eta = \sigma_B^2/\sigma_T^2$$

on $\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2$ i $\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2$

Otsu observa que maximitzar qualsevol d'aquests criteris equival a maximitzar els demès. En particular, maximitzar η equival a maximitzar σ_B^2, que es pot re-escriure en termes del *threshold* seleccionat, k, de la següent manera:

$$\sigma_B^2(k) = \frac{[\mu_T\omega(k) - \mu(k)]^2}{\omega(k)[1-\omega(k)]}.$$

Aquesta darrera equació és el nucli de l'algoritme. σ_B^2 es calcula per a tots els possibles valors de llindar i escollim com a *threshold* el valor que maximitzi aquest criteri. Amb tot, el mètode de Otsu pot implementar-se de la següent manera, per tal de trobar el llindar k de la nostra imatge original (*Figura 1*) convertida a escala de grissos: I[1]:

```
h = imhist(I);
L = length(h);
p = h / sum(h);
omega = cumsum(p);
mu = cumsum(p .* (1:L)');
mu_T = mu(end);

sigma_b_squared = (mu_T * omega - mu).^2 ./ (omega .* (1 - omega));
```

A partir d'aquí, podem trobar el *threshold*, k, calculant el màxim d'aquest vector de 256 elements: $sigma\_b\_squared$. La següent figura mostra l'histograma de la imatge I (en blau),

[1] També podem utilitzar, directament, la funció *otsuthresh*, de MATLAB, però he pensat que podria ser interessant implementar manualment l'algoritme per tal d'entendre'l una mica millor

junt amb la corba σ_B^2 (en taronja) i, en vertical, el valor de *threshold* trobat que, com veiem, coincideix amb el punt de la corba on σ_B^2 és màxim.

```
yyaxis left
imhist(I)
ylabel('Histogram')

yyaxis right
plot(sigma_b_squared)
ylabel('\sigma_B^2')

xlim([1 256])

[~,k] = max(sigma_b_squared);
hold on
plot([k k],ylim,'LineWidth',5)
hold off
```

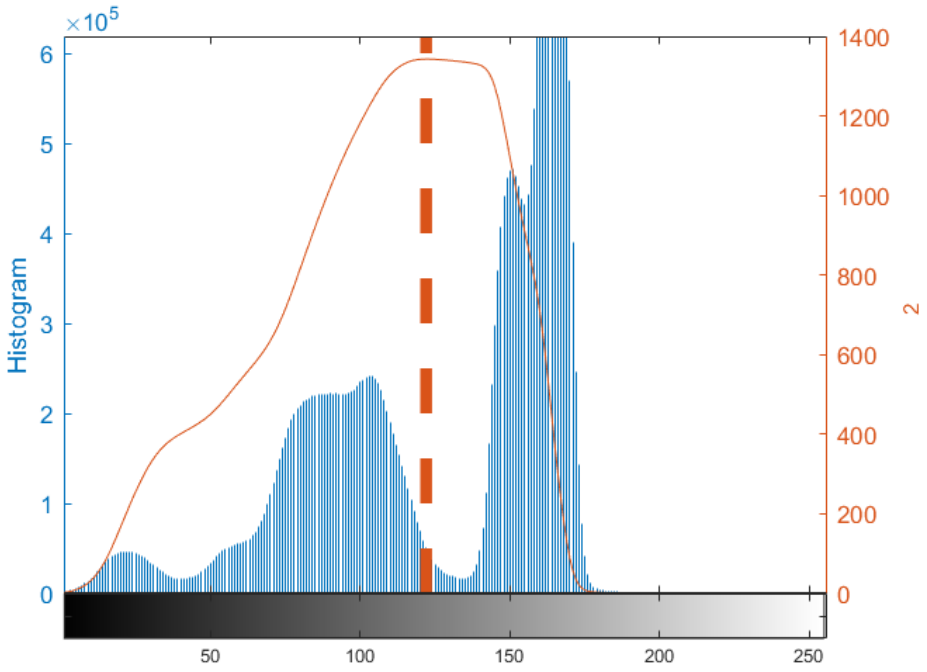

Figura 2: Threshold obtingut mitjançant el mètode de Otsu (k=122)

El resultat de binaritzar la imatge original amb aquest valor que hem trobat pel *threshold* es mostra a la *Figura 3*. Com s'observa, el resultat és prou bo, ja que, com desitjàvem, tot el full ha quedat en blanc (tret del text, lògicament) i el fons de la imatge (tret d'alguns punts aïllats on la il·luminació era diferent) ha quedat en negre.

Per retallar la imatge i quedar-nos només amb la part on apareix el document, podem valdre'ns de la funció *imcrop*, que ens permet generar una nova imatge (retallada) especificant la regió de la imatge original amb la que volem quedar-nos, mitjançant un rectangle. Podem trobar la frontera *real* del document de la imatge amb la funció *bwtraceboundary* mitjançant les següents comandes. El problema és que, com s'aprecia a la *Figura 4*, els *borders* del document no són rectes respecte els eixos X i Y de la imatge (ja que aquesta s'ha pres amb perspectiva), de manera que ens haurem de conformar amb generar la nova imatge a partir de la *Bounding Box* de la component connexa més gran de l'escena, que és, lògicament, la del full blanc (*Figura 5*).

```
dim = size(BW);
col = round(dim(2)/2)-90; row = min(find(BW(:,col)));
boundary = bwtraceboundary(BW,[row, col],'N');
imshow(BW)
hold on; plot(boundary(:,2),boundary(:,1),'g','LineWidth',3); hold off;
```

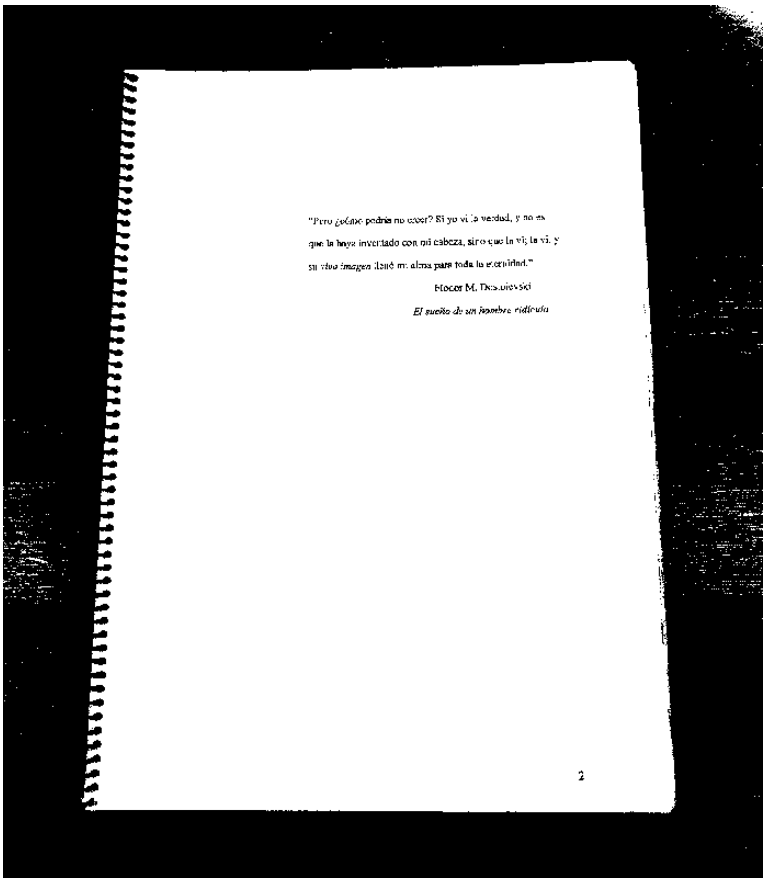

Figura 3: Imatge binaritzada de forma global, BW, amb el mètode de Otsu

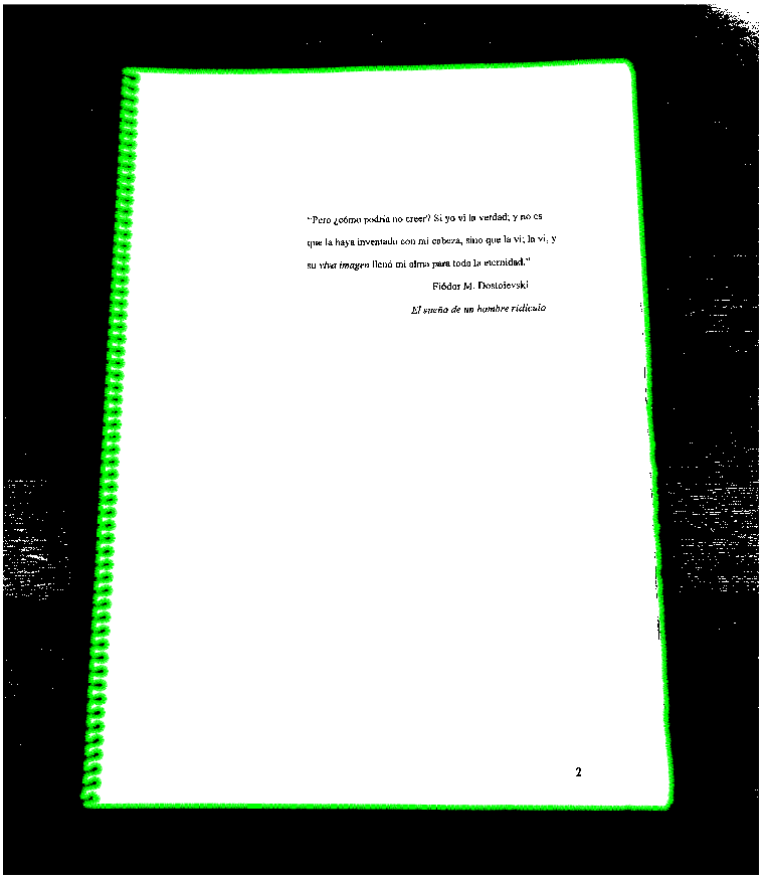

Figura 4: Frontera del full blanc de la imatge original (en verd)

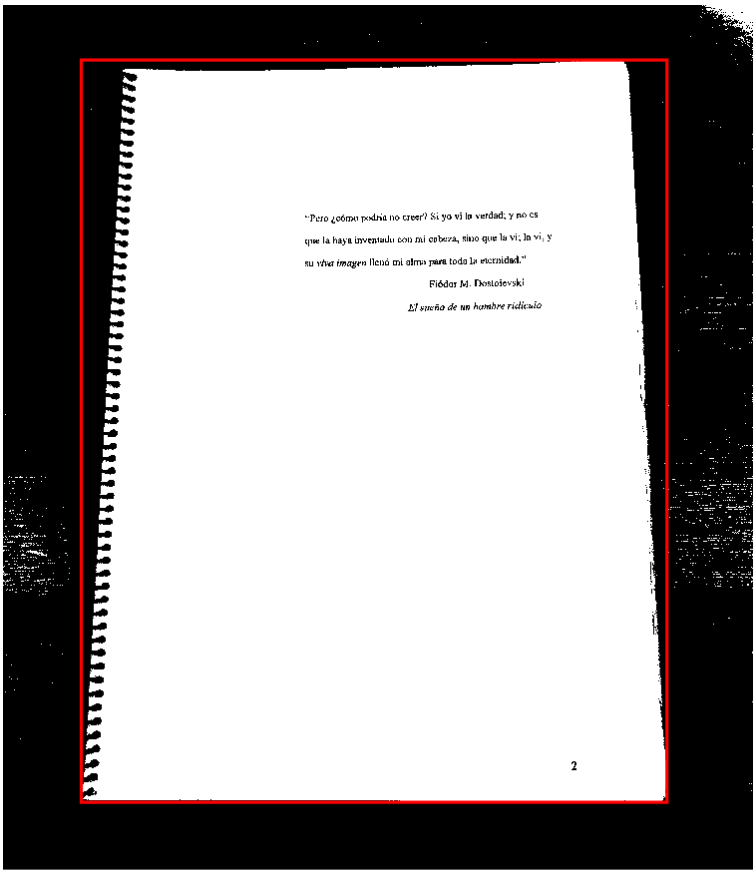

Figura 5: Bounding Box de la component connexa més gran de la imatge

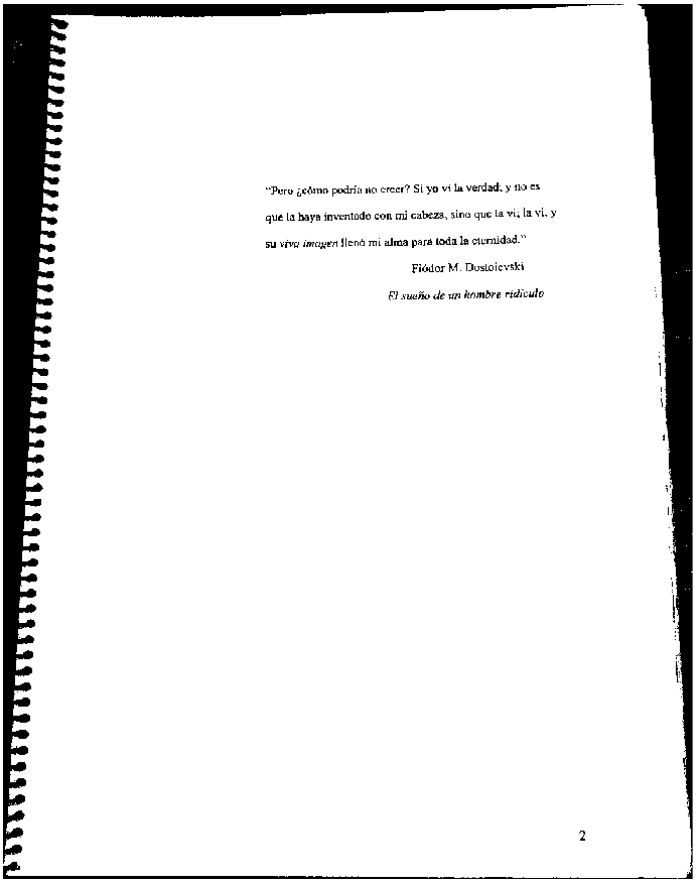

Figura 6: Imatge retallada a partir de la Bounding Box de la Figura 5

Per trobar els components connexos de la imatge podem fer servir la funció *bwconncomp*. A continuació, extraurem les propietats *BoundingBox* i *Area* a través de la funció *regionprops*, que ens permetrà obtenir la caixa contenidora de cada component connexa, junt amb la seva àrea. La *BoundingBox* que nosaltres busquem és, lògicament, la que té l'àrea màxima. Així, podem construir un rectangle que passarem a *imcrop* per generar una nova imatge retallada (*Figura 6*).

```
labeledImage = bwconncomp(BW);
R = regionprops(labeledImage,'BoundingBox','Area');
for i=1:length(R)
    Area(i) = R(i).Area;
end
[~,index] = max(Area)
bb = R(index).BoundingBox;
r = rectangle('Position',bb,'EdgeColor','r','LineWidth',2);
BW = imcrop(BW,r.Position);
imshow(BW);
```

Binarització local de la imatge

Ara veurem una manera d'efectuar una binarització local de la imatge original, mitjançant la funció *colfilt*. La idea és molt simple: utilitzarem una finestra lliscant $[M\ N]$ que binaritzi a blanc els píxels que són K nivells de gris superiors que el promig de la finestra. L'objectiu és intentar ajustar la mida d'aquesta finestra: M i N a la mida d'un caràcter i una línia del text del document, respectivament. Un bon resultat l'he obtingut utilitzant una finestra lliscant $[15x15]$ amb $K = 125$ nivells de gris (altres combinacions de paràmetres són possibles i són, segurament, millors). La crida MATLAB que s'ha de fer servir per realitzar la binarització és la següent:

```
BW2 = colfilt(I,[15 15],'sliding',@myfunction,125);
```

La funció *myfunction*, que té el comportament que acabem de descriure, té aquesta forma:

```
function [y] = myfunction(I,K)
    [f,c] = size(I);
    y = I(ceil(f/2),:);
    for j = 1:c
        sum = 0;
        for i = 1:f
            sum = sum + I(i,j);
        end
        average = sum/f;
        y(j) = ((y(j)-average) > K)*255;
    end
end
```

El resultat obtingut aplicant aquesta binarització local es mostra a la *Figura 7* (sense retallar).

Segmentació i recompte de caràcters

Per acabar, intentarem comptar els caràcters del text del full de la imatge. Per fer això tornarem a fer servir les funcions *bwconncomp* i *regionprops*, que ja havíem fet servir per trobar la *Bounding Box* de la component connexa més gran. La idea en realitat és la mateixa. Enlloc de buscar l'ària màxima, volem trobar les components connexes que representen cadascuna de les lletres del text (notem que caldrà trobar les components a partir de la imatge binària que teníem abans, però *negada*, ja que ara volem detectar caràcters, en negre, no el full, en blanc). El procediment, doncs, és el que es mostra a continuació. Notem que estem considerant només components que tinguin un àrea en un rang determinat (que s'ha estimat a partir de la imatge

original, de forma aproximada), per intentar evitar considerar com a caràcters elements com els accents, els punts de les is, o els punts aïllats que han pogut quedar sense binaritzar-se bé.

```
labeledImage = bwconncomp(~BW);
R = regionprops(labeledImage,'BoundingBox','Area');
numberOfComponents = length(R);
imshow(BW);
characters = 0;
for component = 1:numberOfComponents
    if R(component).Area > 75 && R(component).Area <= 733
        characters = characters + 1;
        bb = R(component).BoundingBox;
        rectangle('Position',bb,'EdgeColor','r','LineWidth',2);
        hold on;
    end
end
```

A la *Figura 8* es mostra un detall del resultat d'aplicar aquest procés de segmentació. Com observem, la segmentació és força acceptable. No tenim encerclats signes ortogràfics ni elements d'accentuació sintàctica, com desitjàvem, i el nombre de caràcters comptabilitzat és bastant proper al real (hi ha una diferència de 10 caràcters).

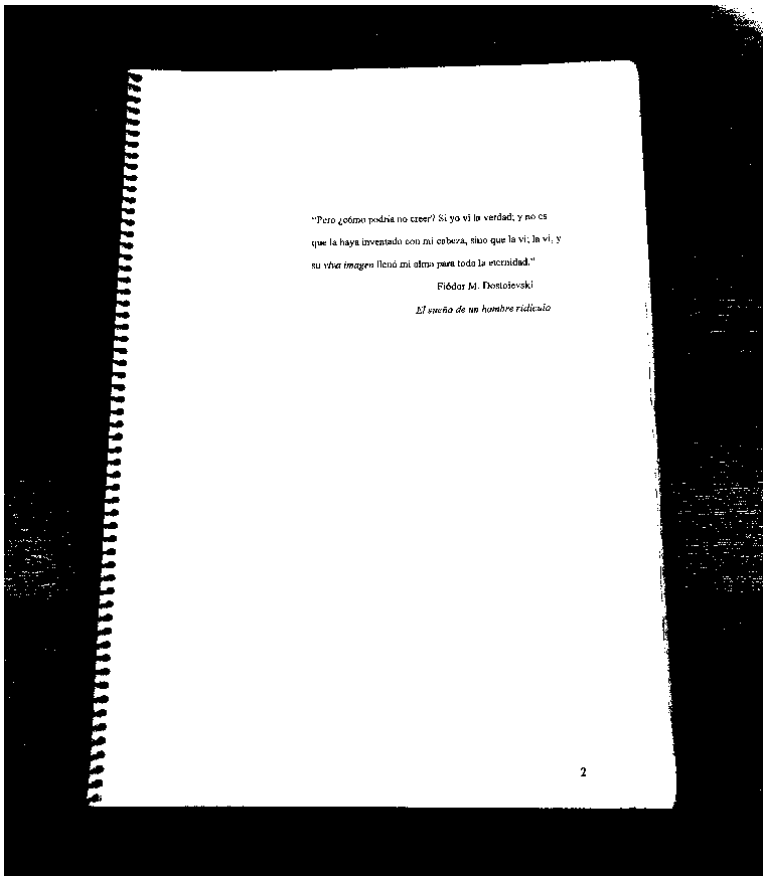

Figura 7: Binarització local de la imatge (sense fer el cropping)

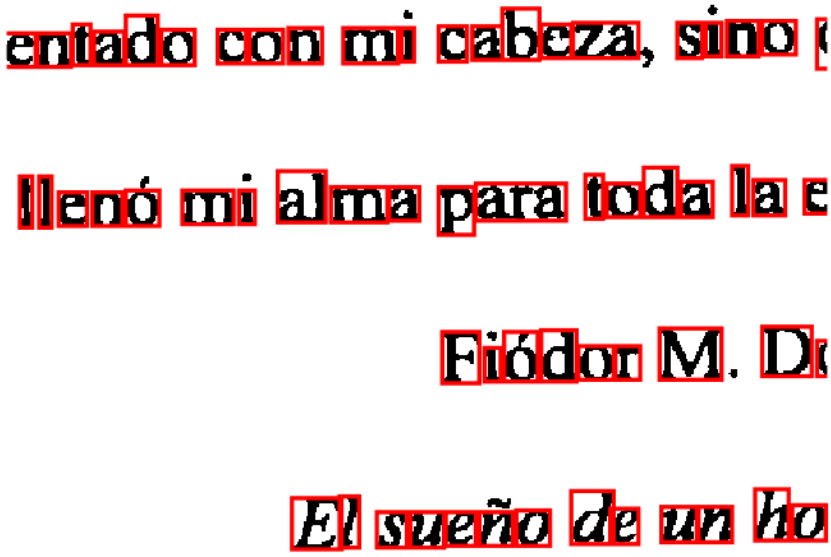

Figura 8: Detall de la segmentació dels caràcters de la imatge binària. El nombre de caràcters compats és 120

Limitacions de la binarització global i millores

La binarització global que hem construït funciona prou bé quan la il·luminació és més o menys *uniforme*, però presenta grans febleses quan tenim ombres dures en la imatge (p.ex: *Figura 9*).

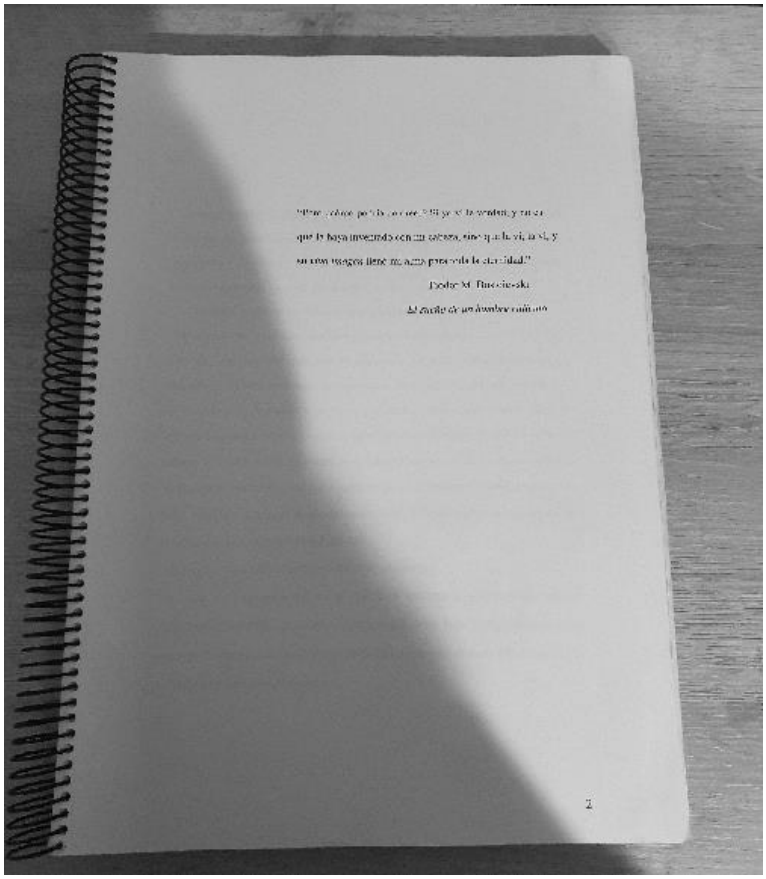

Figura 9: Imatge del mateix text de la Fig 1 (a escala de grisos), amb hard shadows

Els canvis d'il·luminació en els píxels de les zones sota l'ombra són tan forts que la nostra binarització *considera* que gran part del full en blanc és, en realitat, fosc, produint resultats generalment dolents aplicant binarització global, però no amb la binarització local (*Figura 10)*.

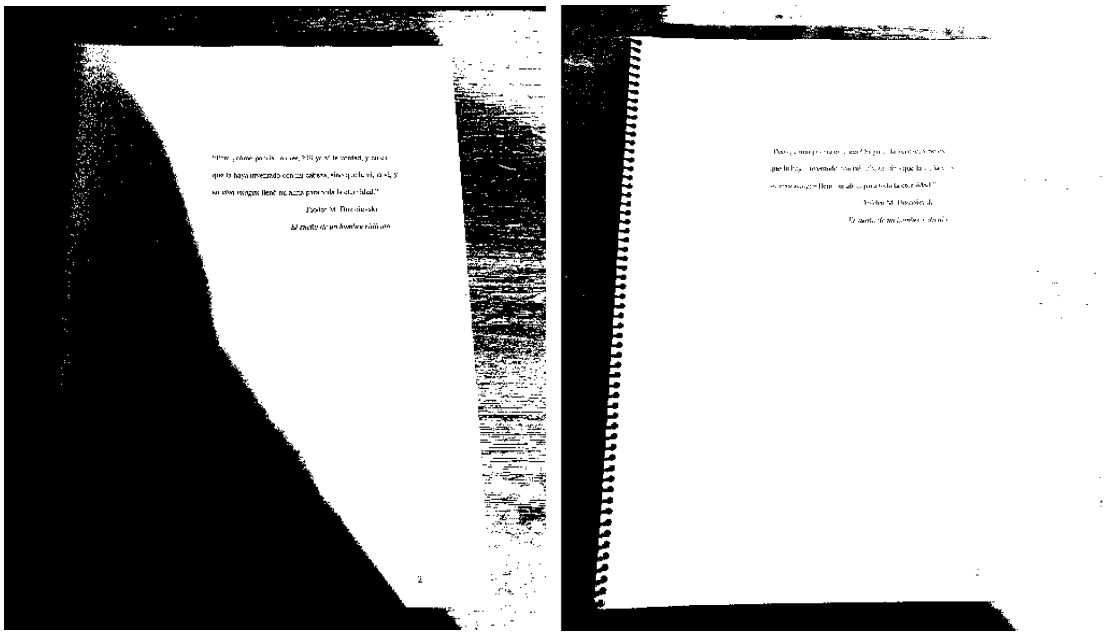

Figura 10: Binarització global de la Figura 9 (esquerra) i binarització local de la mateixa imatge (dreta)

El resultat podria millorar notablement si utilitzem un llindar adaptatiu. Podem fer servir, per exemple, la comanda *bwconncomp(I,'adaptive',...)* —tot indicant que els píxels de primer pla (els del text) són més foscos que els píxels de fons (els del full blanc) — que utilitza un algoritme conegut com a Mètode de Bradley[2] per trobar la imatge del llindar adaptatiu de la imatge d'entrada (veure *Figura 11, esquerra*) i, a partir d'aquesta, produir la imatge binaritzada (*Figura 11, dreta*). Podríem fer servir la funció *adaptthresh* per modificar el càlcul del *threshold* adaptatiu i obtenir un resultat força millor que el que mostrem aquí.

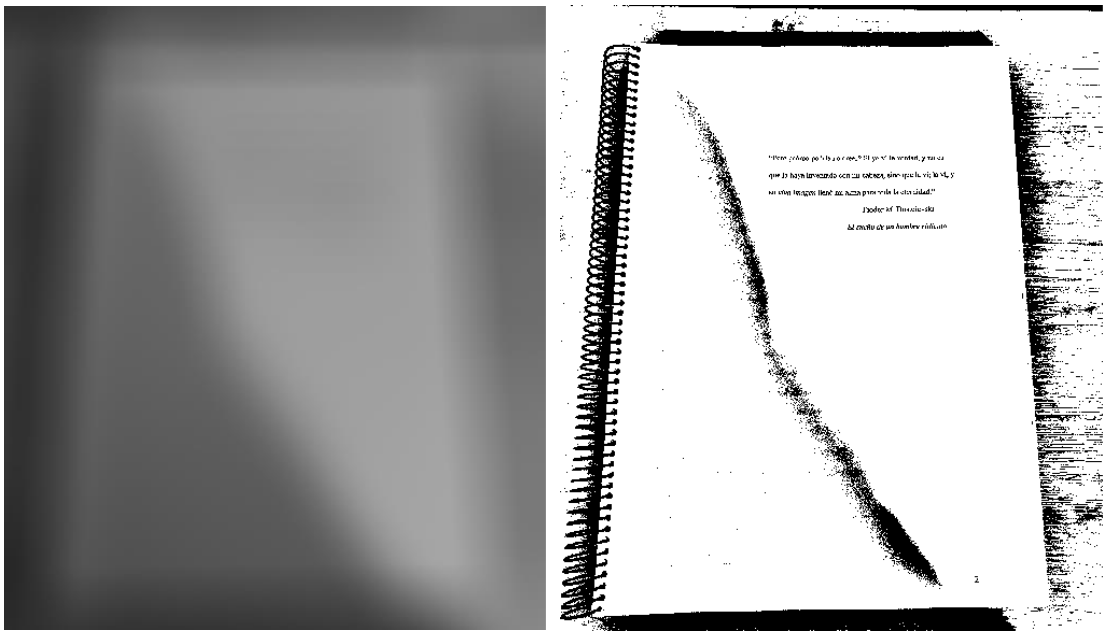

Figura 11: Umbral adaptatiu de la Figura 9 (esquerra) i binarització adaptativa de la mateixa imatge (dreta)

[2] D.Bradley, G.Roth: "Adaptive Thresholding Using Integral Image," Journal of Graphics Tools, vol. 12, pp. 13-21, (2007)

Per últim, podem millorar el *cropping* de la imatge si apliquem una transformació geomètrica de la imatge d'entrada. La idea simplement és detectar els quatre vèrtexs de les cantonades que delimiten el full de la imatge i utilitzar la funció *imwarp* per eliminar l'efecte de la perspectiva de la fotografia i aconseguir que les cantonades estiguin completament alineades amb els eixos X i Y de la imatge (*Figura 12*). D'aquesta forma, podrem fer retallar la imatge binaritzada sense que ens quedin regions negres del fons de la imatge, com passava, per exemple, a la *Figura 6*.

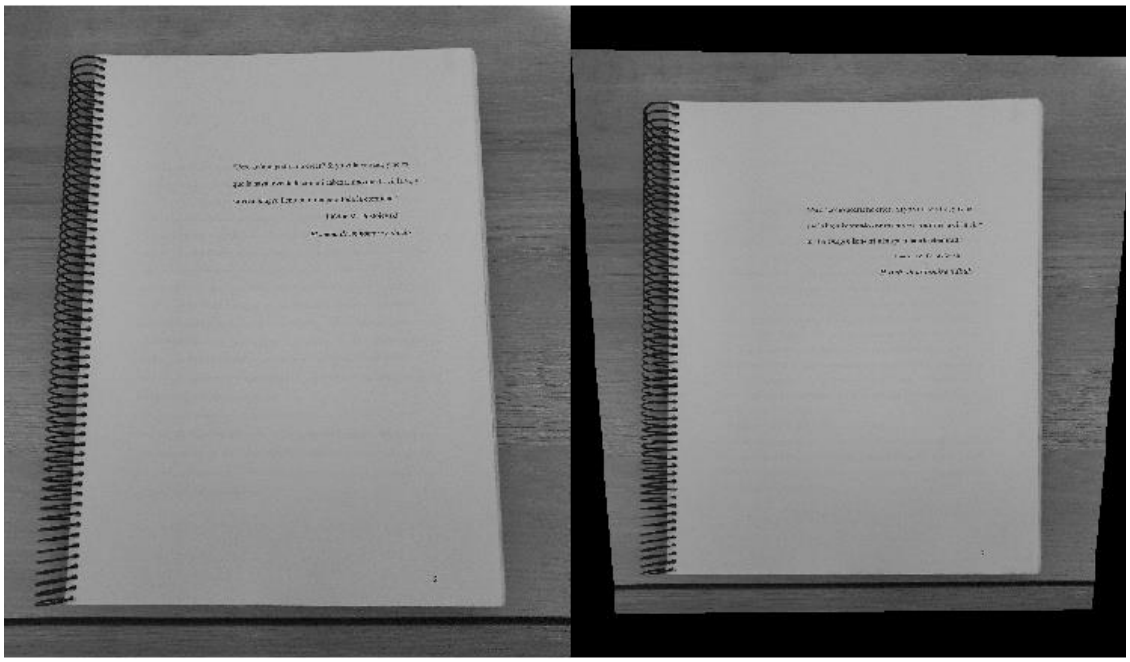

Figura 12: 'Montage' de la imatge original (esquerra) i de la imatge transformada (dreta)