

Proyecto 2. Procesado de imágenes usando morfología matemática con MATLAB

Máster Universitario en Ingeniería de Telecomunicación

Procesado de señales acústicas e imágenes

Enrique Fernández Sánchez

Link al código: Github: Raniita/image-processing-matlab

22 de diciembre de 2021

Índice

1 Detectar placas de calles	4
1.1 Primera implementación	4
1.2 Segunda implementación	13
1.3 Diez imágenes de la primera implementación	17
1.4 Diez imágenes de la segunda implementación	22
2 Código desarrollado	27
3 Referencias y bibliografía consultada	29

Listado de códigos

1	Lectura de la imagen	4
2	Detectar bordes I	5
3	Dilatar la imagen	6
4	Llenar brechas interiores	7
5	Retirar los objetos no conectados con el borde	8
6	Suavizamos el objeto con elemento estructurante rectangular	9
7	Multiplicación con imagen inicial	10
8	Aplicar OCR e imagen final	11
9	Detección de bordes II	13
10	Código implementado para segmentación y OCR de carteles de calles, utilizando morfología matemática	27

Índice de figuras

1	Lectura de la imagen	4
2	Detección de bordes I	5
3	Dilatar la imagen	6
4	Llenar brechas interiores	7
5	Retirar objetos no conectados con el borde	8
6	Suavizamos el objeto con elemento estructurante rectangular	9
7	Multiplicación con imagen inicial	10
8	Aplicar OCR e imagen final	12
9	Detección de bordes II	13
10	Dilatar imagen	14
11	Llenar brechas interiores	14
12	Retirar objetos no conectados con el borde	15
13	Suavizado del objeto	15
14	Multiplicación con imagen inicial	16
15	Aplicar OCR e imagen final	16
16	[1] Imagen final Plaza Condesa de Peralta	17
17	[1] Imagen final Callejón de la Soledad	17
18	[1] Imagen final Calle del Aire	18
19	[1] Imagen final Calle San Agustín	18
20	[1] Imagen final Calle Jabonerías	19
21	[1] Imagen final Calle García Lorca	19
22	[1] Imagen final Calle Juan Fernández	20
23	[1] Imagen final Calle Tolosa La Tour	20
24	[1] Imagen final Calle Pepe Conesa	21
25	[1] Imagen final Calle Casa de Hierro	21
26	[2] Imagen final Plaza Condesa de Peralta	22
27	[2] Imagen final Callejón de la Soledad	22
28	[2] Imagen final Calle del Aire	23

29	[2] Imagen final Calle San Agustín	23
30	[2] Imagen final Calle Jaboneras	24
31	[2] Imagen final Calle Garcia Lorca	24
32	[2] Imagen final Calle Juan Fernández	25
33	[2] Imagen final Calle Tolosa la Tour	25
34	[2] Imagen final Calle Pepe Conesa	26
35	[2] Imagen final Calle Casa de Hierro	26

1 Detectar placas de calles

En este apartado vamos a comentar la implementación de mi “script” para detectar placas de calles, utilizando morfología matemática con MATLAB.

El código completo de mi implementación lo encontramos en la sección: X.

1.1 Primera implementación

A modo de ejemplo, voy a ir comentando cada uno de los pasos que sigue una imagen hasta llegar a su segmentación final y posterior procesado del OCR.

1. Lectura de la imagen.

Código 1: Lectura de la imagen

```
1 % Leer imagen
2 img = imread(['./photos/' dataset_images{j}]);
3 img_rgb = img;
4 img = im2double(rgb2gray(img));
5 %figure, imagesc(img), axis off image, colormap gray
6
```



Figura 1: Lectura de la imagen

2. **Detectar bordes.** En este primer ejemplo de implementación, he utilizado una función para detectar bordes que no es morfológica. Dicha función es `edge`, que busca los bordes de la imagen en función de la intensidad de esta. Para ello, ajustamos el valor de `threshold` en el que se detectan los bordes.

Código 2: Detectar bordes I

```
1 % Detectar bordes (segmentacion)
2 [x, threshold] = edge(img, 'sobel');
3 % Valor de ajuste para resaltar todavía más los bordes de la placa
4 k = 1.2;
5 mask = edge(img, 'sobel', threshold * k);
6 %figure, imagesc(mask), axis off image, colormap gray
7
```

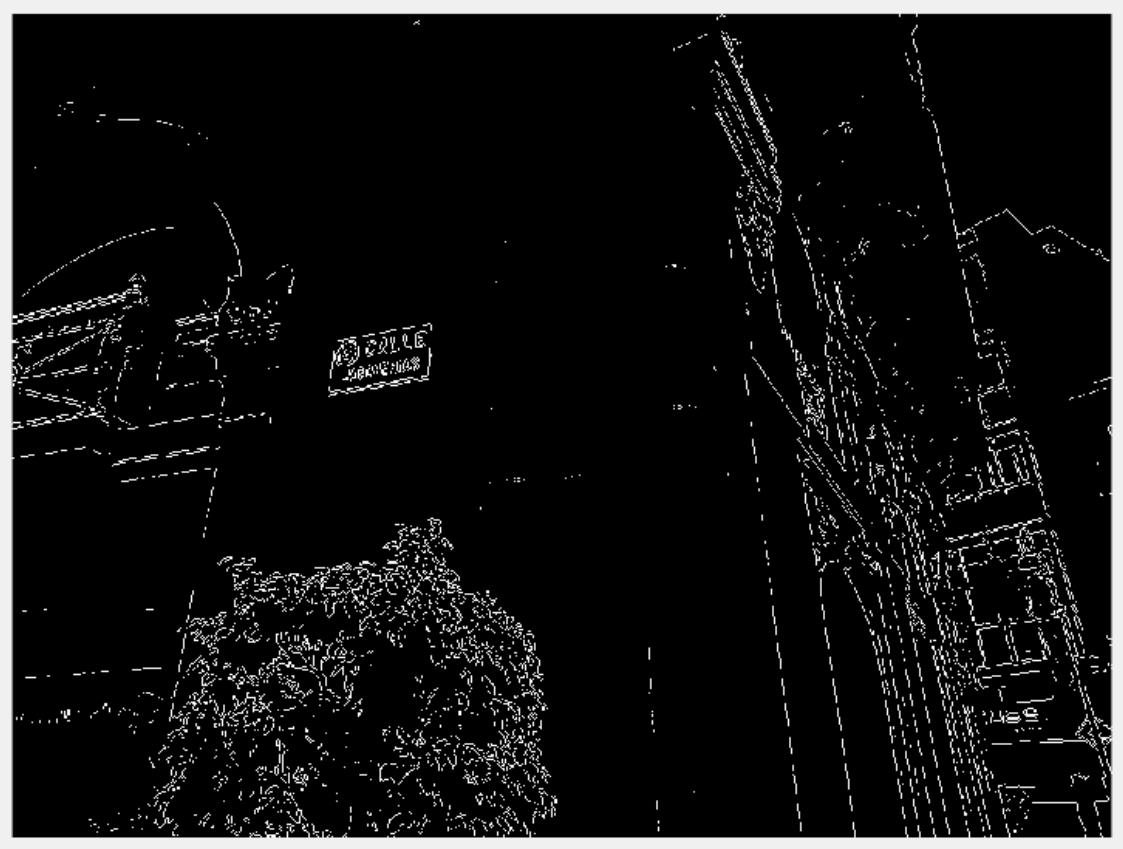


Figura 2: Detección de bordes I

3. **Dilatar imagen.** En este paso procedemos a dilatar la imagen con dos elementos estructurantes, estos elementos son dos lineas del mismo tamaño, situadas cada una a 0 y 90 grados respectivamente. Con esto conseguimos dilatar todavía más los bordes de la calle, además de detectar todas esas formas que tienen más pinta de linea recta en sus bordes.

Código 3: Dilatar la imagen

```
1 % Dilatar la imagen
2 struct_elem90 = strel('line', 12, 90);
3 struct_elem0 = strel('line', 12, 0);
4 mask_dilate = imdilate(mask, [struct_elem90 struct_elem0]);
5 %figure, imagesc(mask_dilate), axis off image, colormap gray
6
```



Figura 3: Dilatar la imagen

4. **Llenar brechas interiores.** En este paso pretendo llenar todos los posibles huecos que puedan encontrarse dentro de una región que ya ha sido dilatada. En principio con el paso anterior ya debería estar suficiente lleno, sin embargo nos vendrá bien aplicar este paso ya que nos mejora los resultados del paso siguiente.

Código 4: Llenar brechas interiores

```
1 % Llenar brechas interiores  
2 mask_fill = imfill(mask_dilate, 'holes');  
3 figure, imagesc(mask_fill), axis off image, colormap gray  
4
```

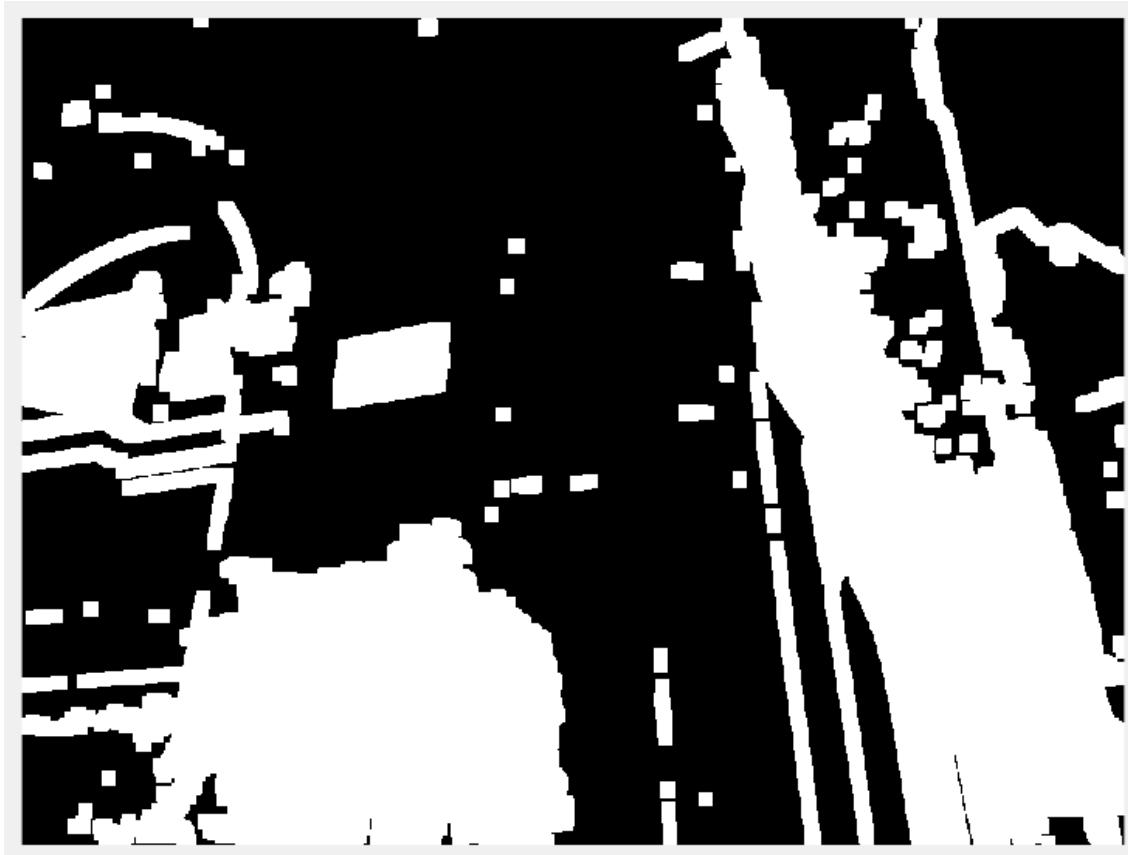


Figura 4: Llenar brechas interiores

5. **Retirar los bordes no conectados con el borde.** Este paso nos permite eliminar todas las estructuras de la imagen que no están conectadas con el borde de la imagen. Esto nos limpia mucho la imagen, dejando prácticamente la placa de la calle ya segmentada. El valor 4 hace referencia a que los pixeles conectados son aquellos en los que los bordes se tocan, también podemos sustituir por 8, que hace referencia a que los bordes o esquinas se tocan.

Código 5: Retirar los objetos no conectados con el borde

```
1 % Retirar los objetos no conectados con el borde  
2 mask_noborder = imclearborder(mask_fill, 4);  
3 %figure, imagesc(mask_noborder), axis off image, colormap gray,  
4
```

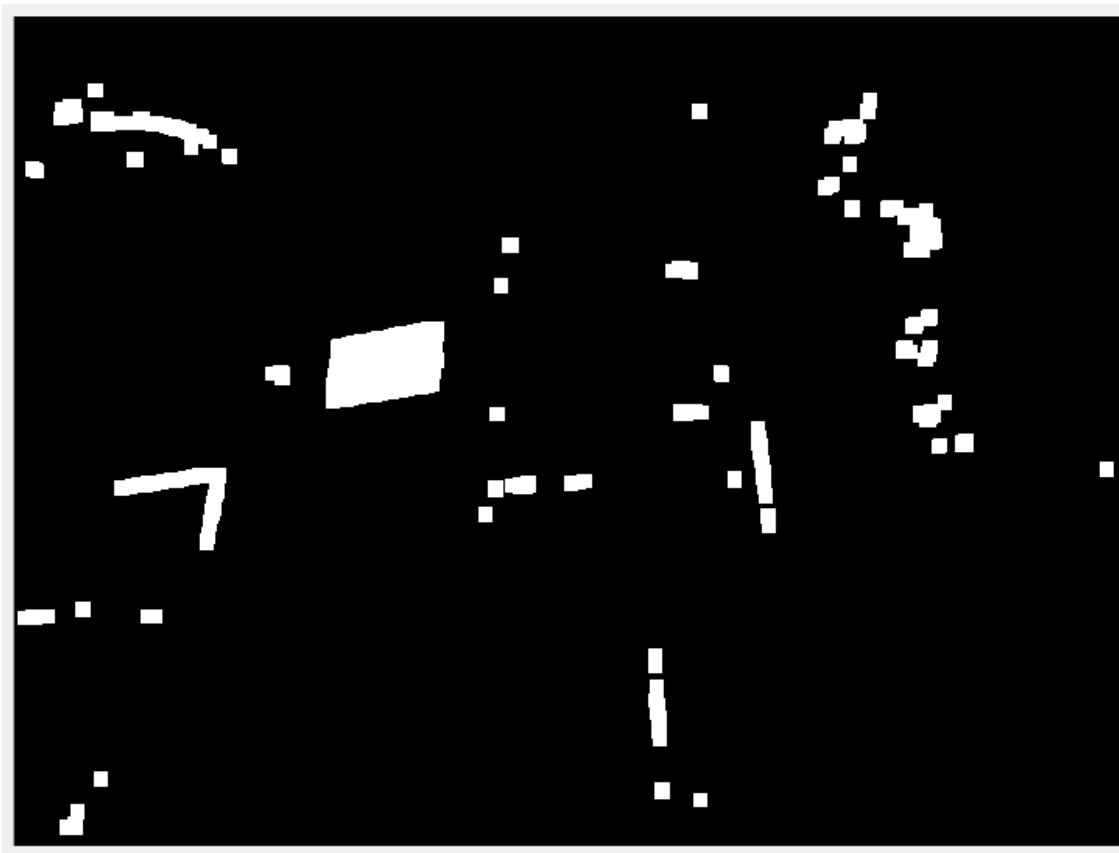


Figura 5: Retirar objetos no conectados con el borde

6. **Suavizado del objeto.** Con el objetivo de eliminar todos esos elementos residuales que quedan en la imagen, aplicamos un suavizado con un elemento estructurante de dimensiones similares a las de la placa a segmentar.

Código 6: Suavizamos el objeto con elemento estructurante rectangular

```
1 % Suavizamos el objeto
2 struct_elemD = strel('rectangle', [12 15]);
3 mask_final = imerode(mask_noborder, struct_elemD);
4 mask_final = imerode(mask_final, struct_elemD);
5 %figure, imagesc(mask_final), axis off image, colormap gray
6
```

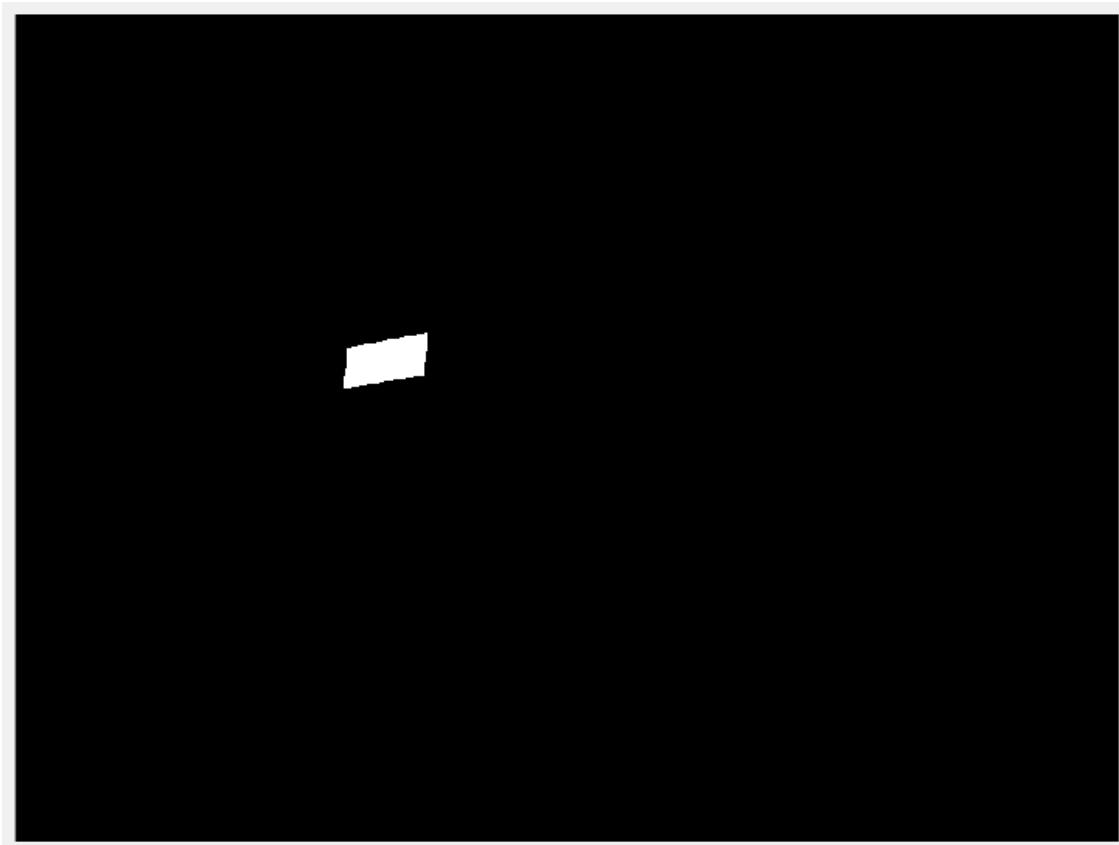


Figura 6: Suavizamos el objeto con elemento estructurante rectangular

7. **Multiplicación con imagen inicial.** En este paso, ya tenemos una máscara binaria con la posición de la placa de la calle, por lo que tendremos que multiplicarlo con la imagen inicial. Además, aplicamos un pequeño filtrado morfológico para aumentar el detalle de las letras de la placa, con el objetivo de facilitar al OCR la detección de los caracteres.

Código 7: Multiplicación con imagen inicial

```
1 % Multiplicar
2 img_mult = immultiply(img, mask_final);
3 se_close = strel('line', 3, 90);
4 img_mult = imclose(img_mult, se_close);
5 %figure, imagesc(img_mult), axis off image, colormap gray
6
```



Figura 7: Multiplicación con imagen inicial

8. Aplicar OCR e imagen final. En este último paso, aplicamos OCR sobre la imagen obtenida en el paso anterior. Para ello, he tenido que descargar un dataset de *tesseract* entrenado para el lenguaje español. La información obtenida por el OCR se ha representado en la figura, además del grado de confianza de la predicción.

Código 8: Aplicar OCR e imagen final

```

1 % OCR
2 % SRC: https://es.mathworks.com/help/vision/ref/ocr.html
3 % SRC: https://es.mathworks.com/help/vision/ug/recognize-text-using-optical-
4 % character-recognition-ocr.html
5 % SRC traineddata: https://github.com/tesseract-ocr/tessdata/blob/074
6 % c37215b01ab8cc47a0e06ff7356383883d775/spa.traineddata
7 results_ocr = ocr(img_mult, 'TextLayout', 'Block', 'Language', {'./tessdata/' %
8 % spa_old_2015.traineddata'});
9 results_ocr = ocr(img_mult, 'TextLayout', 'Block');

10 % Obtenemos el texto del OCR
11 string_calle = '';
12 for i=1:length(results_ocr.Words)
13     if i==1
14         string_calle = results_ocr.Words{i};
15     else
16         string_calle = strcat(string_calle, [' ' results_ocr.Words{i}]);
17     end
18 end
19 disp(['El nombre de la calle es: ' string_calle])
20 % Grado de confianza
21 grade_confidence = mean(results_ocr.WordConfidences);
22 str_confidence = ['Media confianza: ' num2str(grade_confidence)];
23 disp(str_confidence);

24 % Overlay
25 % SRC: https://es.mathworks.com/help/vision/ref/ocr.html
26 figure('Position', get(0, 'Screensize')),
27 imagesc(imfuse(img_rgb, mask_final, 'blend')),
28 axis off image,
29 colormap gray,
30 title(['Detectar calle en imagen: ' strrep(dataset_images{j}, '_', '_') ])
31 % Nombre de la calle
32 text(10, 20, string_calle, 'BackgroundColor', [1 1 1])
33 % Confianza
34 text(10, 60, str_confidence, 'BackgroundColor', [1 1 1])
35

```

Tal y como podemos ver en la figura siguiente, la predicción del OCR no es 100 % exacta para esta imagen, ya que depende de muchos factores. Se podrían aplicar técnicas para mejorar las letras del interior de la placa y ver si el resultado obtenido mejora. Sin embargo, dada la diferencia entre las diferentes fotos escogidas para el trabajo, se hacia realmente difícil que todas las imagen arrojaran una buena predicción en el OCR.



Figura 8: Aplicar OCR e imagen final

1.2 Segunda implementación

Por otro lado, he sustituido la detección de bordes de la implementación anterior con un método de gradiente morfológico, con el objetivo de maximizar los métodos vistos en esta parte de la asignatura. Para ello, vamos a comentar los resultados de la misma imagen utilizada en el apartado anterior.

Procedo a repasar los pasos de la implementación anterior, pero aplicando el cambio mencionado.

1. **Lectura de la imagen.** Equivalente al apartado anterior.
2. **Detectar bordes usando morfología.** Para esta implementación hemos obtenido el gradiente morfológico, y posteriormente transformado la salida a una imagen binarizada, con el objetivo de equiparar ambas implementaciones.

Código 9: Detección de bordes II

```
1 % Detectar bordes (gradiente morfológico)
2 % SRC: https://blogs.mathworks.com/steve/2006/09/25/dilation-erosion-and-the-morphological-gradient/
3 struc_elemGrad = strel('diamond', 2);
4 mask = im2bw(imdilate(img, struc_elemGrad) - imerode(img, struc_elemGrad));
5 %figure, imagesc(mask), axis off image, colormap gray, colorbar
6
```



Figura 9: Detección de bordes II

3. Dilatar imagen.

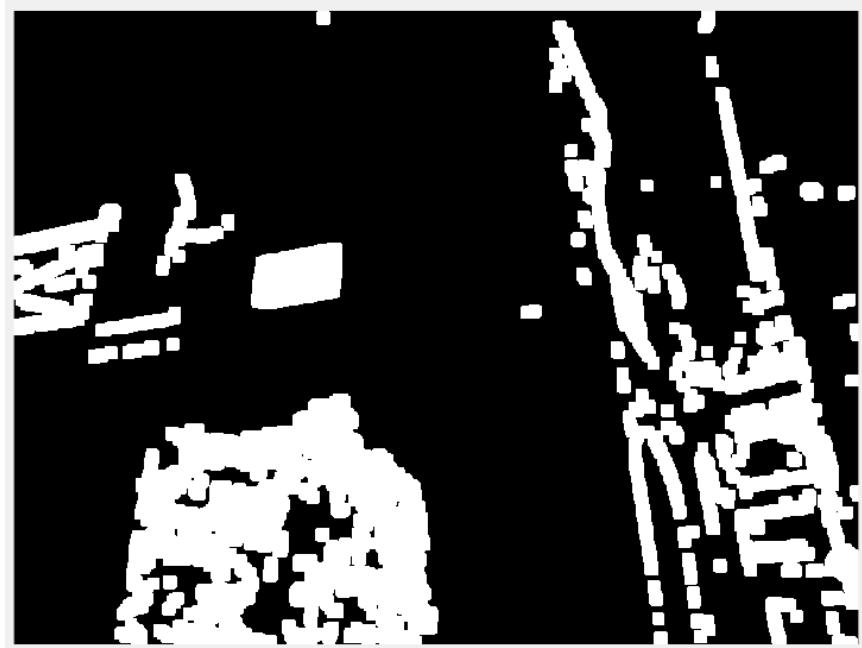


Figura 10: Dilatar imagen

4. Llenar brechas interiores.

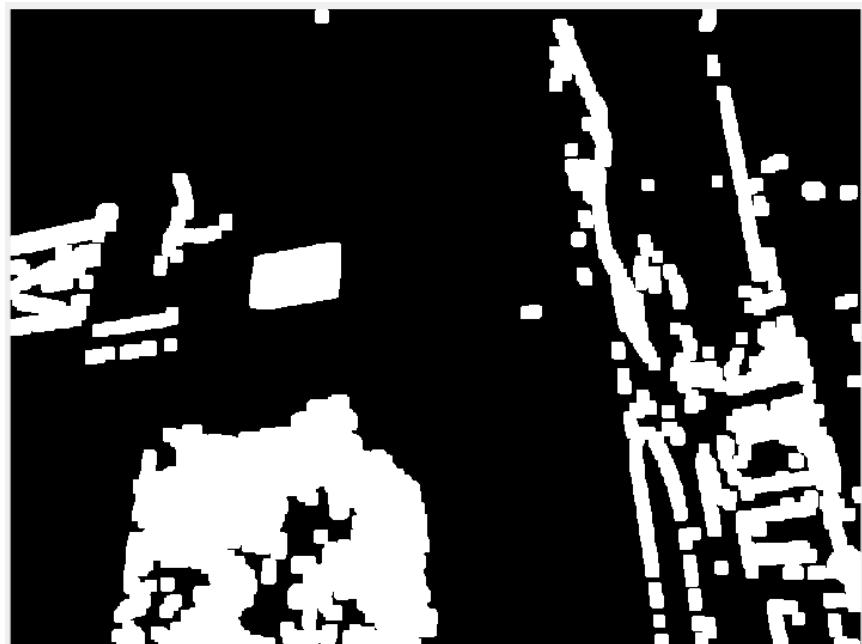


Figura 11: Llenar brechas interiores

5. Retirar los objetos no conectados con el borde.

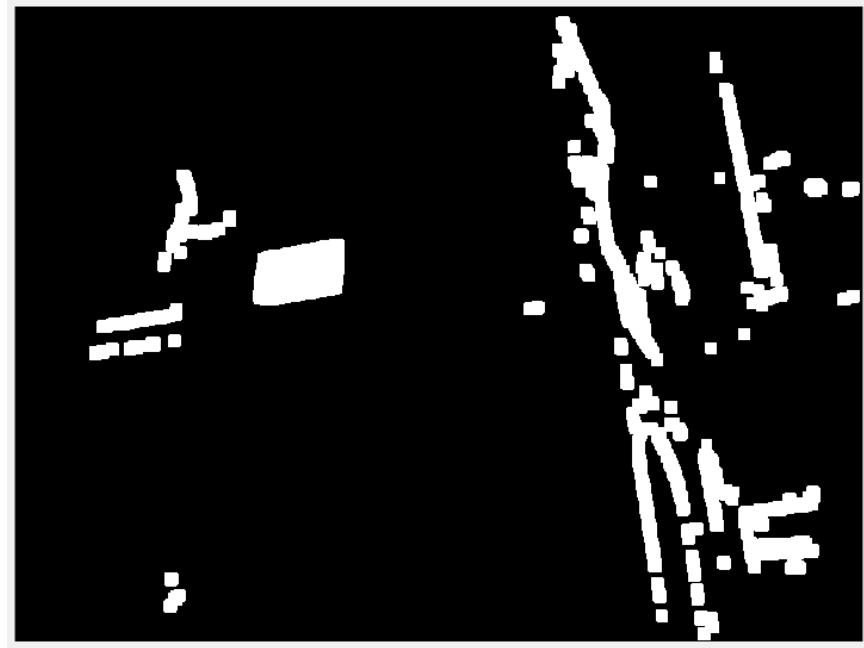


Figura 12: Retirar objetos no conectados con el borde

6. Suavizado del objeto.



Figura 13: Suavizado del objeto

7. Multiplicación con imagen inicial.



Figura 14: Multiplicación con imagen inicial

8. Aplicar OCR e imagen final.



Figura 15: Aplicar OCR e imagen final

1.3 Diez imágenes de la primera implementación



Figura 16: [1] Imagen final Plaza Condesa de Peralta



Figura 17: [1] Imagen final Callejón de la Soledad



Figura 18: [1] Imagen final Calle del Aire



Figura 19: [1] Imagen final Calle San Agustín



Figura 20: [1] Imagen final Calle Jabonerías



Figura 21: [1] Imagen final Calle Garcia Lorca



Figura 22: [1] Imagen final Calle Juan Fernández



Figura 23: [1] Imagen final Calle Tolosa La Tour



Figura 24: [1] Imagen final Calle Pepe Conesa



Figura 25: [1] Imagen final Calle Casa de Hierro

1.4 Diez imágenes de la segunda implementación

En esta implementación hay algunas imágenes que no conseguimos encontrar el cartel de las calles, sin embargo, las he dejado ya que he considerado que sería bastante ilustrativo la diferencia entre ambas implementaciones, además de las diferencias ya existentes en los resultados obtenidos por el OCR.



Figura 26: [2] Imagen final Plaza Condesa de Peralta



Figura 27: [2] Imagen final Callejón de la Soledad



Figura 28: [2] Imagen final Calle del Aire



Figura 29: [2] Imagen final Calle San Agustín



Figura 30: [2] Imagen final Calle Jabonerías



Figura 31: [2] Imagen final Calle Garcia Lorca



Figura 32: [2] Imagen final Calle Juan Fernández



Figura 33: [2] Imagen final Calle Tolosa la Tour



Figura 34: [2] Imagen final Calle Pepe Conesa



Figura 35: [2] Imagen final Calle Casa de Hierro

2 Código desarrollado

En este apartado se encuentra el código integro utilizado para todas las imágenes. Es importante remarcar que todas las imágenes utilizan los mismos elementos estructurantes, por lo que se trata de un algoritmo genérico. Lo único que cambia es la parte de detección de bordes que hemos comentado en el apartado anterior.

Código 10: Código implementado para segmentación y OCR de carteles de calles, utilizando morfología matemática

```
1 %% Detectar placas de calles
2 % SRC: https://es.mathworks.com/help/images/detecting-a-cell-using-image-
3 % segmentation.html
4 %clc,
5 %close all
6
7 dataset_images = {'calle_condesaPeralta.jpg', ...      %1: OK
8   'callejon_soledad_2.jpg', ...      %2: OK
9   'calle_aire.jpg', ...      %3: OK
10  'calle_sanAgustin.jpg', ...      %4: OK
11  'calle_jabonerias.jpg', ...      %5: OK
12  'calle_garciaLorca.jpg', ...      %6: OK
13  'calle_juanFernandez.jpg', ...    %7: OK (sale alguna ventana)
14  'calle_tolosa.jpg', ...      %8: OK
15  'calle_pepeConesa.jpg', ...      %9: OK
16  'calle_casaHierro.jpg'};        %10: OK
17
18 %j = 5;
19 for j=1:length(dataset_images)
20   disp([' num2str(j) '. Ejecutando morfologia para la imagen: ' dataset_images{j}])
21
22 % Leer imagen
23 img = imread('./photos/' dataset_images{j});
24 img_rgb = img;
25 img = im2double(rgb2gray(img));
26 %figure, imagesc(img), axis off image, colormap gray
27
28 % Detectar bordes (segmentacion)
29 [x, threshold] = edge(img, 'sobel');
30 % Valor de ajuste para resaltar todavia mas los bordes de la placa
31 k = 1.2;
32 mask = edge(img, 'sobel', threshold * k);
33 %figure, imagesc(mask), axis off image, colormap gray
34
35 % Detectar bordes (gradiente morfológico)
36 % SRC: https://blogs.mathworks.com/steve/2006/09/25/dilation-erosion-and-the-
37 % morphological-gradient/
38 struc_elemGrad = strel('diamond', 2);
39 mask = im2bw(imdilate(img, struc_elemGrad) - imerode(img, struc_elemGrad));
40 %figure, imagesc(mask), axis off image, colormap gray, colorbar
41
42 % Dilatar la imagen
43 struct_elem90 = strel('line', 12, 90);
44 struct_elem0 = strel('line', 12, 0);
```

```

43 mask_dilate = imdilate(mask, [struct_elem90 struct_elem0]);
44 %figure, imagesc(mask_dilate), axis off image, colormap gray
45
46 % Llenar brechas interiores
47 mask_fill = imfill(mask_dilate, 'holes');
48 %figure, imagesc(mask_fill), axis off image, colormap gray
49
50 % Retirar los objetos no conectados con el borde
51 mask_noborder = imclearborder(mask_fill, 4);
52 %figure, imagesc(mask_noborder), axis off image, colormap gray,
53
54 % Suavizamos el objeto
55 struct_elemD = strel('rectangle', [12 15]);
56 mask_final = imerode(mask_noborder, struct_elemD);
57 mask_final = imerode(mask_final, struct_elemD);
58 %figure, imagesc(mask_final), axis off image, colormap gray
59
60 % Visualizar segmentacion
61 % src: https://es.mathworks.com/help/images/ref/imfuse.html
62 %img_fuse = imfuse(img, BWfinal, 'falsecolor', 'ColorChannels', [1 2 1]);
63 %figure, imagesc(img_fuse), axis off image, colormap gray
64
65 % Multiplicar
66 img_mult = immultiply(img, mask_final);
67 se_close = strel('line', 3, 90);
68 img_mult = imclose(img_mult, se_close);
69 %figure, imagesc(img_mult), axis off image, colormap gray
70
71 % OCR
72 % SRC: https://es.mathworks.com/help/vision/ref/ocr.html
73 % SRC: https://es.mathworks.com/help/vision/ug/recognize-text-using-optical-
74 % character-recognition-ocr.html
75 % SRC traineddata: https://github.com/tesseract-ocr/tessdata/blob/074
76 % c37215b01ab8cc47a0e06ff7356383883d775/spa.traineddata
77 results_ocr = ocr(img_mult, 'TextLayout', 'Block', 'Language', {'./tessdata/
    spa_old_2015.traineddata'});
78 %results_ocr = ocr(img_mult, 'TextLayout', 'Block');
79
80 % Obtenemos el texto del OCR
81 string_calle = '';
82 for i=1:length(results_ocr.Words)
83     if i==1
84         string_calle = results_ocr.Words{i};
85     else
86         string_calle = strcat(string_calle, [' ' results_ocr.Words{i}]);
87     end
88 end
89 disp(['El nombre de la calle es: ' string_calle])
90
91 % Overlay imagen original con recuadro
92 % [Falla cuando hay muchos cuadros]
93 % SRC: append https://es.mathworks.com/help/vision/ug/recognize-text-using-
94 % optical-character-recognition-ocr.html
95 %calle_BBox(1) = results_ocr.WordBoundingBoxes(1,1);

```

```

93 %calle_BBox(2) = results_ocr.WordBoundingBoxes(2,2);
94 %calle_BBox(3) = results_ocr.WordBoundingBoxes(1,3);
95 %calle_BBox(4) = results_ocr.WordBoundingBoxes(2,4);
96 %img_box = insertObjectAnnotation(I, 'rectangle', calle_BBox, string_calle);
97 %figure, imagesc(img_box), axis off image, colormap gray
98
99 % Grado de confianza
100 grade_confidence = mean(results_ocr.WordConfidences);
101 str_confidence = ['Media confianza: ' num2str(grade_confidence)];
102 disp(str_confidence);
103
104 % Overlay
105 % SRC: https://es.mathworks.com/help/vision/ref/ocr.html
106 figure('Position', get(0, 'Screensize')),
107 imagesc(imfuse(img_rgb, mask_final, 'blend')),
108 axis off image,
109 colormap gray,
110 title(['Detectar calle en imagen: ' strrep(dataset_images{j}, '_', '')])
111 % Nombre de la calle
112 text(10, 20, string_calle, 'BackgroundColor', [1 1 1])
113 % Confianza
114 text(10, 60, str_confidence, 'BackgroundColor', [1 1 1])
115 disp('*****')
116 end

```

3 Referencias y bibliografía consultada

- Mathworks: Detectar células mediante detección y morfología de bordes
- Mathworks: Dilatación, erosión y el gradiente morfológico
- Mathworks: imfuse
- Mathworks: imclearborder
- Mathworks: edge
- Mathworks: *Recognize Text Using Optical Character Recognition (OCR)*
- Mathworks: OCR