22437 - Industrial Vision

Final Project: Simple car license plate reader

Andreu Massanet Felix

Universitat de les Illes Balears

Contents

1 Introduction and objectives

In this final project we implemented a car license plate recognizer using the image-processing techniques covered during the course. The setting is a commission from the Costitx Local Police: a gang of reckless drivers, known as the *Vodka Clan*, is wreaking havoc on the town's streets, and since the Town Council has no budget for a professional license plate detection system, a simple detector is requested that, given the image of a license plate, automatically extracts the registration code.

The license plates handled follow the reduced catalogue of the company So-So-Mobility, which simplifies the problem. Each plate is made up of three parts: white letters on a blue background on the left indicating the country (`GB`, `D` or `PL`), three uppercase letters that can only be `A`, `B` or `C`, and four digits that can only be `0`, `1`, `7` or `8`. The output for each plate has the form `GB-ABC0178`.

The system works both with the ideal images in the `easy` folder and with those in the `noisy` folder, which are corrupted with noise and require a prior filtering stage.

The solution is structured into four files:

- `main.m`: manages the image paths and runs the main loop.
- `preprocess_license_plates.m`: cleans the noise from the images.
- `identify_license_plate.m`: segments the plate and returns the code.
- `detected_plates_to_txt.m`: writes the results and the per-country count to a text file.

Throughout this report we explain the decisions made at each stage and the difficulties encountered, referring to the intermediate figures saved in `output/`.

2 Explanation of the solution and the work carried out

The central idea of the solution is to segment the image by color: the plates have a very large white region with the LLLNNNN characters and a small blue region on the left (the country strip). From these two masks, the two regions are cropped, the connected components are extracted, and each character is analyzed using topological descriptors (`EulerNumber`, which counts the holes) and geometric ones (`Area`, `Eccentricity`, `Extent`). Since the catalogue of possible characters is very small, the number of holes is almost always enough to tell them apart.

During development, several points were encountered where a technical decision had to be made. The most relevant ones are:

- **Choice of the preprocessing filter.** After analyzing the type of noise, several options were tried, among them a custom filter with mask

$$Filter = \frac{1}{5}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{1}$$

  which is a cross-shaped kernel that averages the central pixel with its four cardinal neighbors: it smooths the noise in the horizontal and vertical directions and preserves diagonal edges better than a standard 3×3. However, the 3×3 mean filter gives equally good results with much simpler code, so that is the one that has been kept.

- **Filtering out the stars of the European flag.** In the first versions we tried to remove them with morphological operations (openings and closings), but in the end we decided to filter them by color, tuning the thresholds of the RGB masks so that the stars were discarded from the start.

- **Dynamic crops and thresholds.** In the first versions, when we only worked with the `easy` images, the cut between the country strip and the LLLNNNN region was made with fixed pixel coordinates, and the area thresholds were also hardcoded. When we moved to the `noisy` images this solution stopped working, so it was generalized by computing the crops dynamically with `regionprops` and `BoundingBox`.

- **Telling the 1 from the 7.** The two characters have the same `EulerNumber` (zero holes), so a second descriptor, `Extent`, is added to tell them apart.

3 Explanation of the source code

This section describes how the code designed for reading license plates is structured. It is split into four `.m` files: `main`, `preprocess_license_plates`, `identify_license_plate` and `detected_plates_to_txt`.

3.1 Main

The `main` is basically a double loop: over folders and, within each folder, over images. The images are sorted alphabetically with `sort` on the names returned by `dir`, so that the output listing is in order.

For each image the preprocessing is applied, `identify_license_plate` is called, the resulting code is accumulated in a `matriculas` vector, and the intermediate images are saved in `output/<name>/` with `imwrite`, one folder per plate. At the end, a single call to `detected_plates_to_txt` generates the `.txt` file with the results.

```
addpath("scripts/");

% easy and noisy folders
carpetas = ["input/easy/", "input/noisy/"];
mkdir("output");
matriculas = [];

for c = 1:1:length(carpetas)
    % keep only the images and sort them
    listado = dir(carpetas(c) + "*.png");
    tabla = struct2table(listado);
    nombres = sort(string(tabla.name));

    for i = 1:1:length(nombres)
        img = imread(carpetas(c) + nombres(i));

        % noise removal, needed to identify the plate
        img = preprocess_license_plates(img);

        % identify the plate
        [codigo, pasos] = identify_license_plate(img);
        matriculas = [matriculas; codigo];
        disp(codigo);

        mkdir("output/" + nombres(i));

        % save the intermediate images in each folder
        imwrite(img, "output/" + nombres(i) + "/original.png");
        imwrite(pasos.blanco, "output/" + nombres(i) + "/white_mask.png");
        imwrite(pasos.azul, "output/" + nombres(i) + "/blue_mask.png");
        imwrite(pasos.zona_matricula, "output/" + nombres(i) + "/plate.png");
        imwrite(pasos.zona_pais, "output/" + nombres(i) + "/country.png");
    end
end

% generate the txt
detected_plates_to_txt(matriculas);
```

Code 1: Main loop of `main.m`

The intermediate images are saved directly with `imwrite` instead of displaying them with `figure` and `saveas`: this way no windows are opened during execution and the result on disk is a clean image for each step, with no titles or *subplots*.

3.2 Image preprocessing

Before choosing the filter, we had to determine what type of noise is present in the `noisy` images. Figure 1 shows the same plate in its two versions: in the noisy version a layer of color noise can be seen over the background and over the white regions.

Figure 1: `easy` image (noise-free) and the same image in its `noisy` version. A layer of color noise can be seen over all regions, both dark and light.

The histogram (Figure 2) already shows the difference: in the `easy` image the intensities are concentrated in a few values at the extremes, while in the `noisy` image they spread into bell shapes. This points to Gaussian noise.

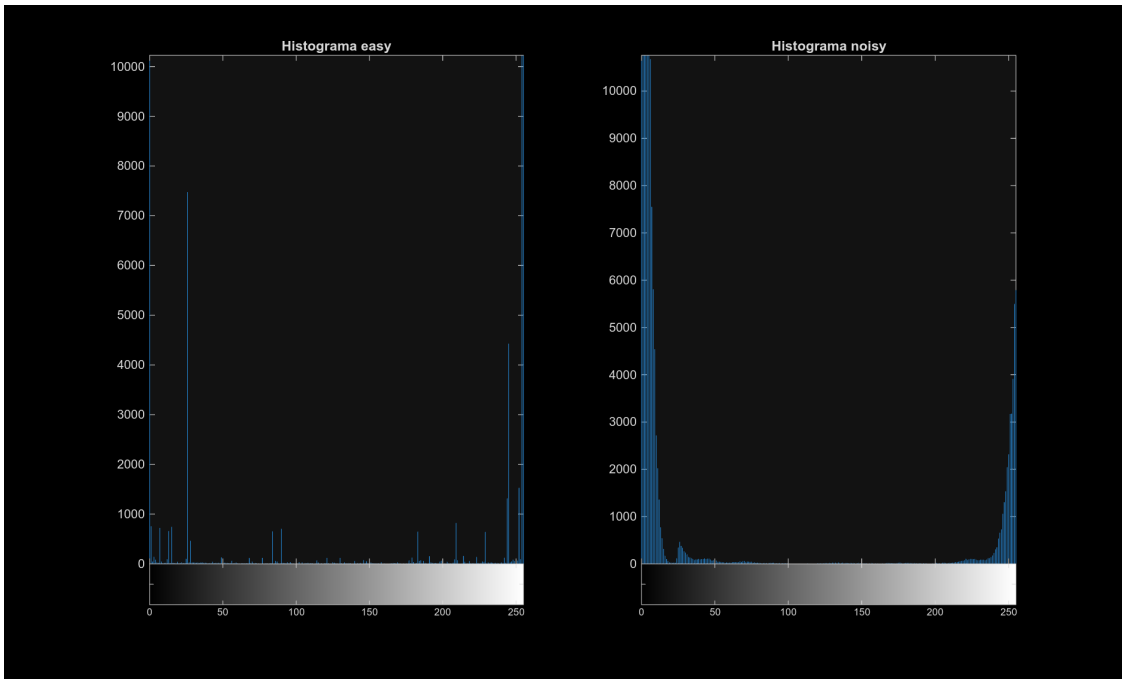

Figure 2: Histogram of the `easy` *image (left) and the* `noisy` *one (right), in grayscale. In the noisy version the peaks at the extremes turn into bell shapes.*

To confirm this, the mean and standard deviation of the gray value were measured in two flat regions, a dark one and a light one. The results are those in Figure 3: $\mu \approx 0{,}011$ and $\sigma \approx 0{,}017$ in the dark region, and $\mu \approx 0{,}989$ and $\sigma \approx 0{,}017$ in the light one. The standard deviation is practically the same in both regions, even though the means are completely opposite. This means that the spread of the noise **does not depend on the pixel value**, i.e. it is additive and zero-mean. Formally, the model is:

$$g(x,y) = f(x,y) + n(x,y), \qquad n(x,y) \sim \mathcal{N}(0,\sigma^2), \tag{2}$$

where f is the ideal image, g is the observed image and n is the noise, which follows a normal distribution with zero mean and constant variance σ^2.

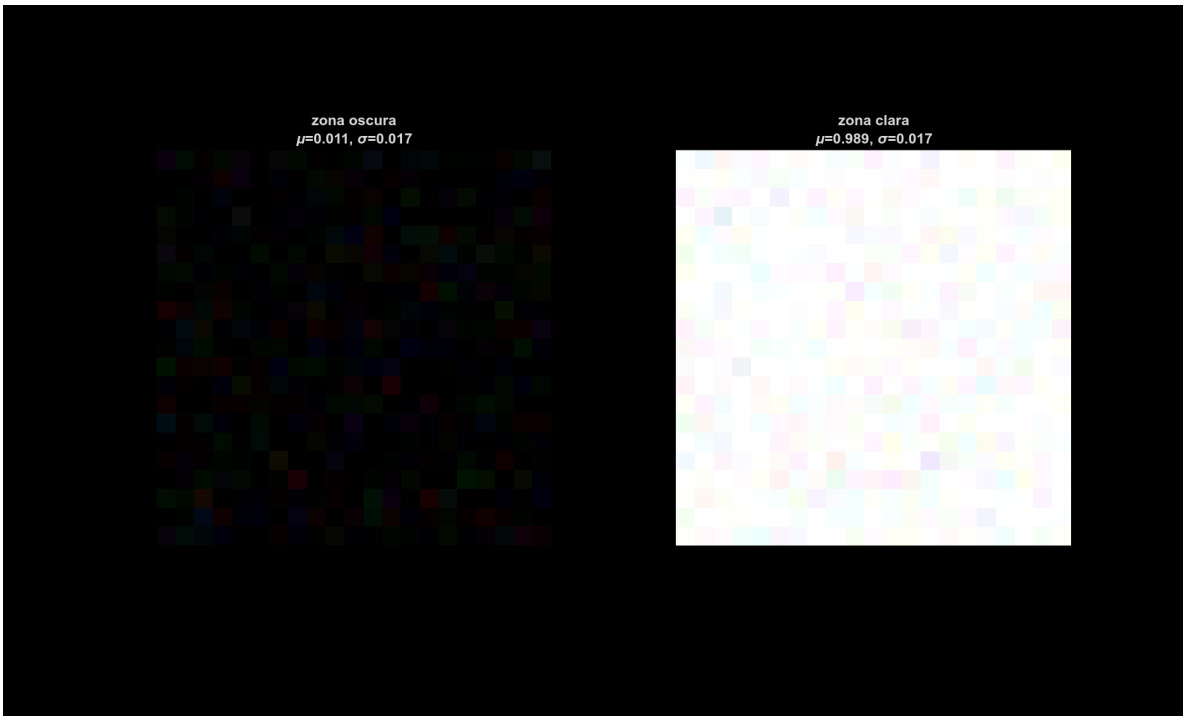

Figure 3: Close-up of the dark region (left) and the light region (right) used to measure the noise. The σ is similar in both despite the means being opposite: the noise is additive and does not depend on intensity.

Once the noise is identified as Gaussian, the appropriate thing is to apply a **linear filter** (mean or Gaussian), as seen in Lab 6. The median filter is discarded because it is designed for salt and pepper and here there are no impulses. A 3×3 mean filter with symmetric *padding* was chosen, whose kernel is:

$$h = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \tag{3}$$

and which is applied to the image. It is the simplest linear filter and sufficient for this case: the noise drops enough for the color masks to work without problems. Figure 4 shows the effect.

Figure 4: Noisy image (left) and result after applying the 3×3 mean filter (right). The noise is noticeably reduced.

The function's code is very compact:

```
% Removes the noise from the images.
% The noise is Gaussian, so we remove it with a 3x3 mean
% filter. We use symmetric padding.
function img = preprocess_license_plates(img)
    filtro = fspecial('average', [3 3]);
    img = imfilter(img, filtro, 'symmetric');
end
```

Code 2: The `preprocess_license_plates.m` function

3.3 Plate identification

This is the central function of the program. It receives the already-filtered image and returns the plate string and a *struct* with the intermediate steps. The process is divided into four phases, explained below, each with its figures and its code.

3.3.1 Generating the white and blue masks from the RGB channels.

The masks are obtained by comparing the three channels separately. For white we use $(R > 0{,}6) \wedge (G > 0{,}6) \wedge (B > 0{,}6)$, and for blue $(B > 0{,}4) \wedge (R < 0{,}4) \wedge (G < 0{,}47)$. The thresholds were tuned by looking at histograms and by trial and error: the blue of the country strip is fairly saturated, so red and green must stay low while blue is high. The bound of 0,47 on green is slightly higher than the one on red because it was observed that the blue of the strip has a somewhat larger green component than red. Figure 5 shows the two resulting masks.

Figure 5: Phase 1: original and color masks (white and blue).

```
img = im2double(img);
R = img(:,:,1);
G = img(:,:,2);
B = img(:,:,3);

% separate the white pixels
blanco = (R > 0.6) & (G > 0.6) & (B > 0.6);
% separate the blue pixels
azul = (B > 0.4) & (R < 0.4) & (G < 0.47);
```

Code 3: Generating the color masks

3.3.2 Segmenting the LLLNNNN region (largest white connected component) and the country region (largest blue connected component).

The white region of the plate is, by far, the largest white component in the image. The country strip is the largest blue component. `regionprops` is used with `Area` and `BoundingBox` on each mask, the component with the maximum area is chosen and its *bounding box* is kept. This way the crop adapts on its own to the size and position of each plate, with no need to fix coordinates by hand.

An important detail: although the *bounding box* of the country is computed on the `azul` mask, the crop itself is always done on the `blanco` mask, both for the plate and for the country. The blue mask is only used to locate the strip. This way, inside the country rectangle, the letters `PL`, `GB` or `D` (which are white in the original image) appear as white on a black background without having to invert anything.

By contrast, inside the plate region the characters are dark on a white background, so that one is indeed inverted before processing it. Figure 6 shows the two crops and the already-inverted plate.

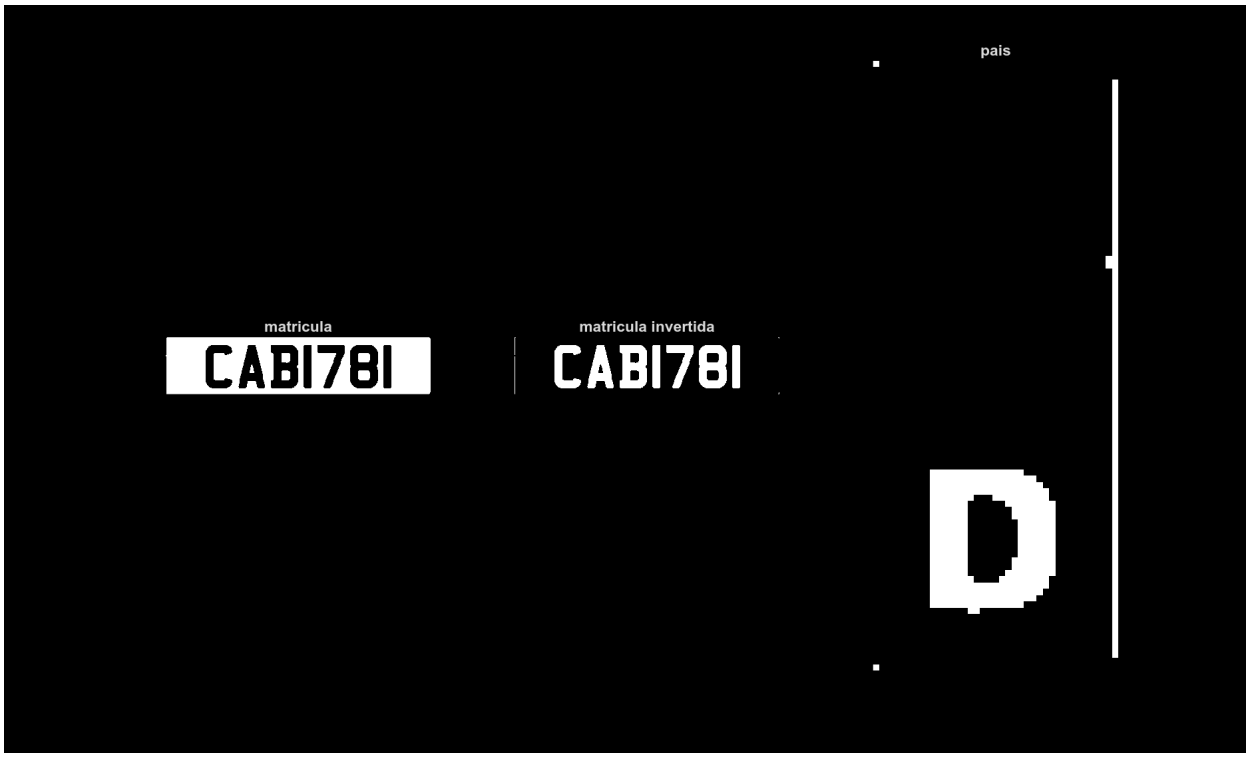

Figure 6: Phase 2: plate, inverted plate and country.

```
% crop the plate (largest white component)
comp = regionprops(blanco, ["Area", "BoundingBox"]);
areas = [comp.Area];
[area_max, mayor] = max(areas);
caja = round(comp(mayor).BoundingBox);
zona_matricula = blanco(caja(2):caja(2)+caja(4)-1, caja(1):caja(1)+caja(3)-1);

% crop the country (largest blue component)
comp = regionprops(azul, ["Area", "BoundingBox"]);
areas = [comp.Area];
[area_max, mayor] = max(areas);
caja = round(comp(mayor).BoundingBox);
zona_pais = blanco(caja(2):caja(2)+caja(4)-1, caja(1):caja(1)+caja(3)-1);

% on the plate the letters are dark: invert them
zona_matricula = 1 - zona_matricula;
zona_matricula = logical(zona_matricula);
```

Code 4: Cropping the two regions and inverting the plate

3.3.3 Identifying the country letters.

The blue strip may contain one letter (`D`) or two (`GB`, `PL`). A first idea was to use a counter and distinguish by the number of detected components, but junk shows up: the masks also detect the white stars of the European flag as small dots, and the vertical frame between the strip and the white region appears as a very long straight strip. Figure 7 shows this clearly.

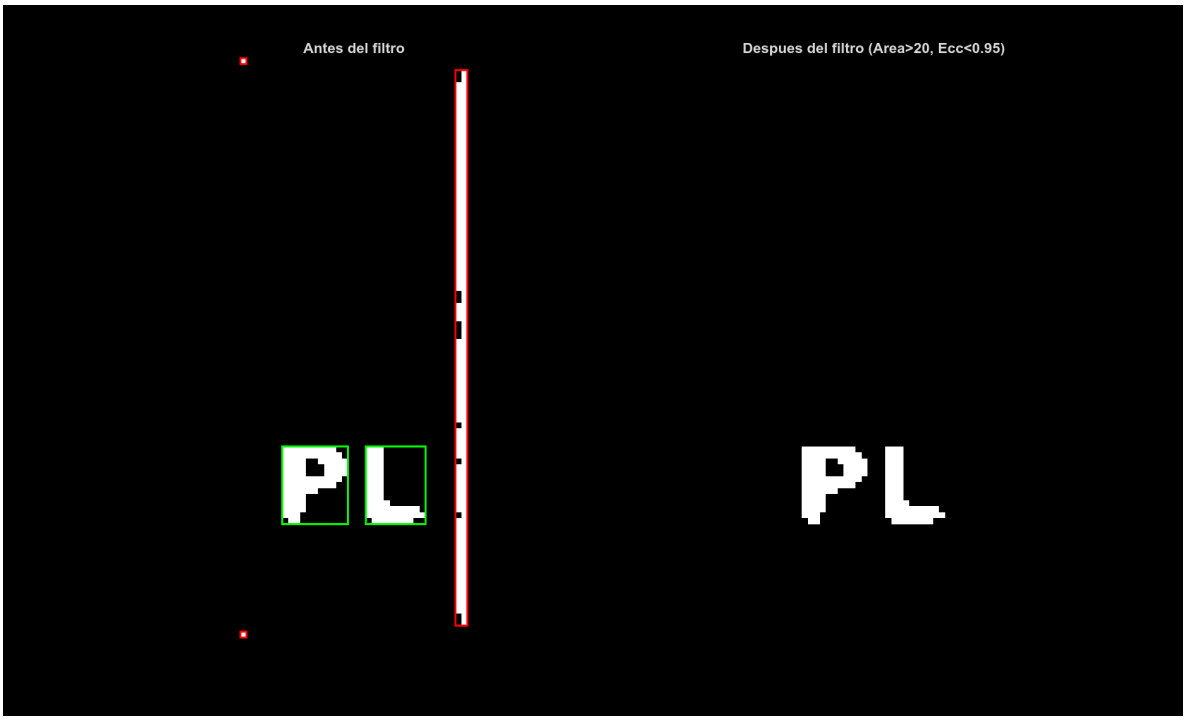

Figure 7: Components of the country region before and after the filter. In green, the valid letters. In red, the dots coming from the stars and the straight strip of the frame, which must be discarded.

Several strategies were tried to filter out this junk. What worked best is combining two descriptors: `Area > 20` discards the tiny dots (stars) and isolated pixels; `Eccentricity < 0,95` discards the straight strips (eccentricity close to 1), keeping the letters (eccentricity below 0,9).

Using morphological operations (the opening and closing from Lab 8) to clean the junk before labeling the components was also considered. They were tried with several structuring elements and radii, but the problem was that the country letters are small and the `noisy` versions already have them a bit eroded; any morphological operation ended up breaking them.

Once the junk is filtered out, the three countries are distinguished using the `EulerNumber` descriptor, which for a single component is:

$$E = 1 - H, \tag{4}$$

where H is the number of holes in the region. If a single letter remains, it is the `D` of Germany; if two remain, the holes of the first are examined: the `P` of Poland has one hole ($E = 0$), while the `G` of Great Britain has no holes ($E = 1$). Schematically:

$$\text{country} = \begin{cases} \text{D} & \text{if 1 letter remains} \\ \text{PL} & \text{if 2 letters remain and } E_1 = 0 \\ \text{GB} & \text{if 2 letters remain and } E_1 = 1 \end{cases}$$

```
comp = regionprops(zona_pais, ["Area", "Eccentricity", "EulerNumber"]);
letras = [];
for k = 1:1:length(comp)
    % discard the dots (by area) and the straight strips (by eccentricity)
    if comp(k).Area > 20 && comp(k).Eccentricity < 0.95
        letras = [letras; comp(k)];
    end
end
% D is a single letter
% PL and GB have two letters: we look at the holes of the first character
```

```
% P has one hole
% G has no holes
if length(letras) == 1
    letras_pais = "D";
elseif letras(1).EulerNumber == 0
    letras_pais = "PL";
else
    letras_pais = "GB";
end
```

Code 5: Junk filtering and country identification

3.3.4 Identifying the plate characters.

Junk components also appear in the plate region, especially in the `noisy` images: residual noise dots and pieces of the frame line that separates the country strip from the rest of the plate. Figure 8 shows them. Here, unlike the country, the letters are large and the junk is very small, so a single `Area > 200` filter is enough to discard it.

Figure 8: Components of the plate region before and after the filter. In green, the seven valid characters. In red, residual noise dots and a piece of the frame line, discarded by the `Area > 200` filter.

Once the components are filtered, the characters are classified by the number of holes, just as in the country case, applying equation (4). The classification is as follows: two holes ($E = -1$) are `B` in the first three or `8` in the last four; one hole ($E = 0$) is `A` or `0`; no holes ($E = 1$) is `C`, `1` or `7`.

Since the first three positions can only be letters and the last four only digits, the `EulerNumber` is enough to discriminate them, except for the `1` and the `7`, which both have zero holes. To separate them the `Extent` descriptor is used: the `1` fills almost all of its *bounding box* (it is a thin rectangle), while the `7` leaves a triangular gap underneath and has a lower `Extent`. The threshold of 0,8 works well to tell them apart.

To avoid nesting the code so much, the processing of the seven characters is split into two loops: one for the first three (letters) and another for the last four (digits). The logic of each is as follows:

$$\text{letter} = \begin{cases} \text{B} & \text{if } E = -1 \\ \text{A} & \text{if } E = 0 \\ \text{C} & \text{if } E = 1 \end{cases} \qquad \text{digit} = \begin{cases} 8 & \text{if } E = -1 \\ 0 & \text{if } E = 0 \\ 1 & \text{if } E = 1 \text{ and } \mathit{Extent} > 0{,}8 \\ 7 & \text{if } E = 1 \text{ and } \mathit{Extent} \leq 0{,}8 \end{cases}$$

```
letras_matricula = "";
% the first 3 are letters
% we count the holes of each character
% B has two holes
% A has one hole
% C has no holes
for k = 1:1:3
    switch caracteres(k).EulerNumber
        case -1
            letras_matricula = letras_matricula + "B";
        case 0
            letras_matricula = letras_matricula + "A";
        case 1
            letras_matricula = letras_matricula + "C";
    end
end
% the last 4 are numbers
% we count the holes of each character
% 8 has two holes
% 0 has one hole
% 1 and 7 have no holes: we tell them apart by the extent
for k = 4:1:7
    switch caracteres(k).EulerNumber
        case -1
            letras_matricula = letras_matricula + "8";
        case 0
            letras_matricula = letras_matricula + "0";
        case 1
            if caracteres(k).Extent > 0.8
                letras_matricula = letras_matricula + "1";
            else
                letras_matricula = letras_matricula + "7";
            end
    end
end

codigo = letras_pais + "-" + letras_matricula;
```

Code 6: Classifying the seven characters

3.4 Conversion to .txt

For the conversion to `.txt`, the `detected_plates_to_txt` function is called, whose input parameter is the list of detected plates and which returns nothing.

The procedure starts with a country counter. A three-element vector called `contadores` is defined, where the first position corresponds to Great Britain, the second to Poland and the third to Germany. To identify the country of each plate it is enough to read the first character: `D` is Germany, `G` is Great Britain and `P` is Poland. Each time one of these countries is detected, the corresponding counter is incremented.

Finally the `matricules.txt` file is written with the list of plates, the `---` line, the word `Totals:` and the three counters. `writematrix` is used because the data is already in a `string` vector.

```
% Saves the list of plates together with the per-country counts
% into a txt file
function detected_plates_to_txt(lista)
    % 1 = GB, 2 = PL, 3 = D
    contadores = zeros(3,1);

    for num_matriculas = 1:1:length(lista)
        mat_temp = lista(num_matriculas);
        mat_temp = char(mat_temp);
        switch mat_temp(1)
            case "G"
                contadores(1) = contadores(1) + 1;
            case "P"
                contadores(2) = contadores(2) + 1;
            case "D"
                contadores(3) = contadores(3) + 1;
        end
    end

    % write to txt
    lista = [lista; "------"];
    lista = [lista; "Totals:"];
    lista = [lista; "GB: " + contadores(1)];
    lista = [lista; "PL: " + contadores(2)];
    lista = [lista; "D: " + contadores(3)];
    writematrix(lista, 'output/matricules.txt');
end
```

Code 7: The `detected_plates_to_txt.m` function

4 Results for each input image

The program correctly processes the **12 plates** (6 from `easy` and 6 from `noisy`). Table 1 shows the plates detected for each input image.

Image	Result
licplate_08.png	D-CAB1781
licplate_12.png	PL-CCA0078
licplate_17.png	PL-BBA1717
licplate_20.png	PL-CBA1708
licplate_24.png	GB-AAC1817
licplate_26.png	D-BBB0101
licplate_08_noise.png	D-CAB1781
licplate_12_noise.png	PL-CCA0078
licplate_17_noise.png	PL-BBA1717
licplate_20_noise.png	PL-CBA1708
licplate_24_noise.png	GB-AAC1817
licplate_26_noise.png	D-BBB0101

Table 1: Plates detected for the 12 input images.

The content of the `matricules.txt` file generated by the program is the one shown in Figure 9.

```
D-CAB1781
PL-CCA0078
PL-BBA1717
PL-CBA1708
GB-AAC1817
D-BBB0101
D-CAB1781
PL-CCA0078
PL-BBA1717
PL-CBA1708
GB-AAC1817
D-BBB0101
------
Totals:
GB: 2
PL: 6
D: 4
```

Figure 9: Content of the `matricules.txt` file generated by the program.

The most interesting thing is that the results for an `easy` image and its `noisy` version come out **identical**. That is, the noise filter cleans enough for the rest of the process to run just as with the ideal images, and the applied thresholds and descriptors hold up well against the remaining noise. The per-country count is consistent with the listed plates.

For a visual check of the segmentation result on the 12 images (binarized country and plate, easy and noisy in columns), see the appendix at the end of the document.

5 Conclusions

A license plate recognizer was implemented that works correctly for the twelve images, both in their ideal version and in their noisy version. The recognizer relies on techniques covered in the labs: threshold segmentation in RGB (Lab 3), linear filtering for Gaussian noise (Lab 6), connected-component analysis and region descriptors such as `EulerNumber`, `Area`, `Eccentricity` and `Extent` (Lab 9).

Where the most back-and-forth happened was in filtering the junk in the country region: in the end, combining `Area` with `Eccentricity` was enough. Morphological operations were also discarded because they broke the small country letters.

As for limitations, the whole recognizer depends on the closed So-So-Mobility catalogue being maintained: if a letter or a digit outside the expected set appears, classification by number of holes no longer works. For something more general it would be necessary to train a classifier (for example a network like those in Lab 10), but for this case this approach has been sufficient.

Appendix: segmentation of the 12 images

Figure 10 shows the segmentation result for the 12 input images (6 `easy` in the left columns and 6 `noisy` in the right columns). For each image the two binarized regions after junk filtering are shown: the country region (`D`, `PL` or `GB`) and the seven plate characters. Above each pair the name of the original file appears. It serves as a quick visual check that the segmenter produces clean components for all country and character combinations in the catalogue.

Figure 10: Segmentation result (binarized country and plate, after junk filtering) for the 12 input images.