Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Visión por computador:

Short Project

Detección y lectura de matrículas

Q1 2022-2023

José Morote García (jose.morote@estudiantat.upc.edu)

Irene Bertolín Rico (irene.bertolin@estudiantat.upc.edu)

Índice

Detección de matrícula

La primera fase del proyecto ha sido la detección de la matrícula y cada uno de sus caracteres.

Se han realizado dos estrategias algo diferentes para maximizar en medida de lo posible la correcta binarización y detección de los elementos necesarios para el reconocimiento de la matrícula.

Estrategia 1

Esta estrategia (*función 15*) se ha enfocado más en la geometría y posicionamiento de la matrícula y sus caracteres (todas las figuras son del mismo ejemplo de muestra).

Preprocesamiento de la imagen

Para el preprocesamiento, primero se ha pasado la imagen a escala de grises y posteriormente se ha suavizado utilizando *filtro mediana* para minimizar el ruido (*función 1)*.

Después se ha binarizado utilizando *moving averages* (*función 2*), puesto que lo más normal en imágenes de coches en la calle es que en el ambiente, al ser callejero, pueda haber mucha o muy poca iluminación, y este método al ser local, es invariante a ella, dando buenos resultados sobre todo con los caracteres.

Por último, dado que el objetivo más adelante es detectar regiones utilizando la función *regionprops,* se tuvo un pequeño problema con el método utilizado para binarizar, y fue que dada la binarización local, en muchos casos no se formaba un objeto cerrado en los bordes de la matrícula (condición necesaria para que el *regionprops* considere la matrícula como una región).
Por tanto para este paso también se decidió conseguir la imagen de bordes, utilizando el método *Canny* (el que mejor resultados tuvo generalmente) (*función 3*), de manera que al ser mucho más probable que los bordes de la matrícula se encuentren cerrados, el *regionprops* la tomará en cuenta.

Los resultados se pueden observar en la siguiente figura:

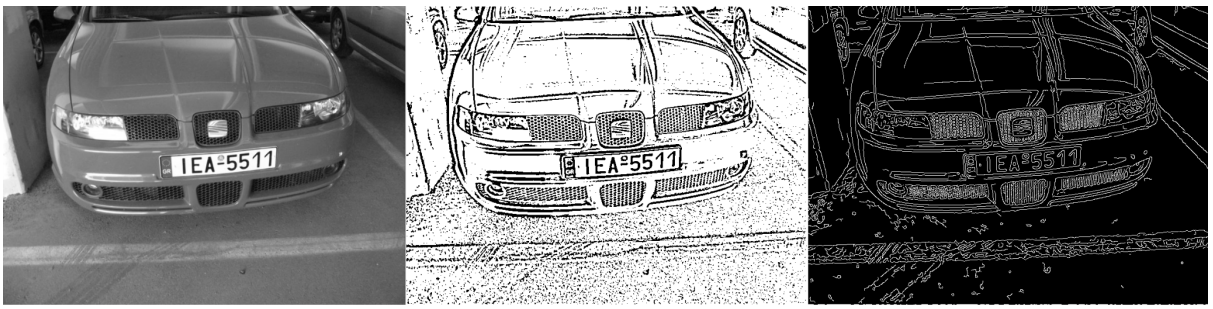

Figura 1: Imagen a escala de grises (izquierda), imagen binarizada (centro) y imagen bordes (derecha)

Filtrado de regiones

El siguiente paso es el filtrado de las regiones extraídas con el *regionprops,* dado que el simple uso de esta función puede generar potencialmente cientos de regiones, de las cuales únicamente se necesitan las asignadas a la matrícula.

Todas estas regiones pasan por cuatro filtros, el resultado final de los cuales deben ser los mejores candidatos a matrícula.

El primer filtro (*función 8*) es dedicado a buscar candidatos a la placa de la matrícula, y las condiciones (o dicho de otra manera las características) que cumplirán las regiones extraídas son las siguientes:

- Tienen una anchura como mínimo el doble que la altura, dado que por la forma de la placa es una característica que deben cumplir todas.
- Tienen un área mínima, para evitar regiones demasiado pequeñas.
- Tienen una anchura como máximo 7 veces más grande que la altura, de manera que evitamos regiones innecesariamente largas, en las cuales es muy improbable que se encuentre la matrícula.

Para finalizar este primer filtro, se aplica un pequeño filtro adicional para quitar las regiones que contienen lo mismo (*función 9*).

Se puede observar un ejemplo del resultado en la siguiente figura:

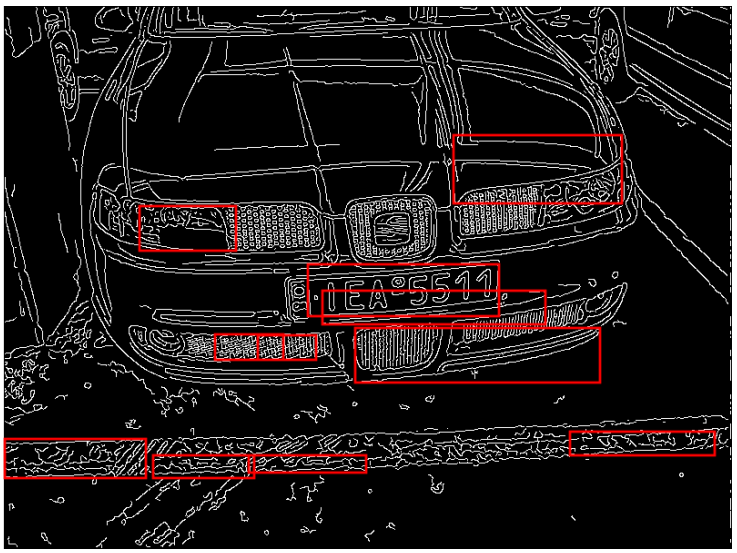

Figura 2: Resultado del filtro de placas de matrícula sobre la imagen de bordes

De este filtro se esperan los mejores candidatos para ser matrícula (como se puede observar en la figura anterior), y se elegirá entre uno u otro candidato según su contenido, por tanto para cada uno de estos candidatos se ha realizado un *crop* de esa misma región pero en la imagen binarizada, y se elegirá uno de ellos según si con los siguientes 3 filtros, enfocados a la detección de caracteres, se consiguen buenos resultados o no.

El segundo filtro es dedicado a filtrar los candidatos a caracteres por relación de aspecto y por área mínima (*función 11*), de cada una de las regiones conseguidas anteriormente, y el resultado de este primer filtrado son regiones con las siguientes características:

- Tienen mayor altura que anchura, ya que toda letra o número tiene mayor altura que anchura
- Cumplen con un área mínima, para eliminar posibles regiones demasiado pequeñas, en las cuales obviamente no cabe un carácter

Una vez aplicado, se pueden ver resultados como los siguientes, donde en algunos hay malos resultados y en uno de ellos están los correctos:

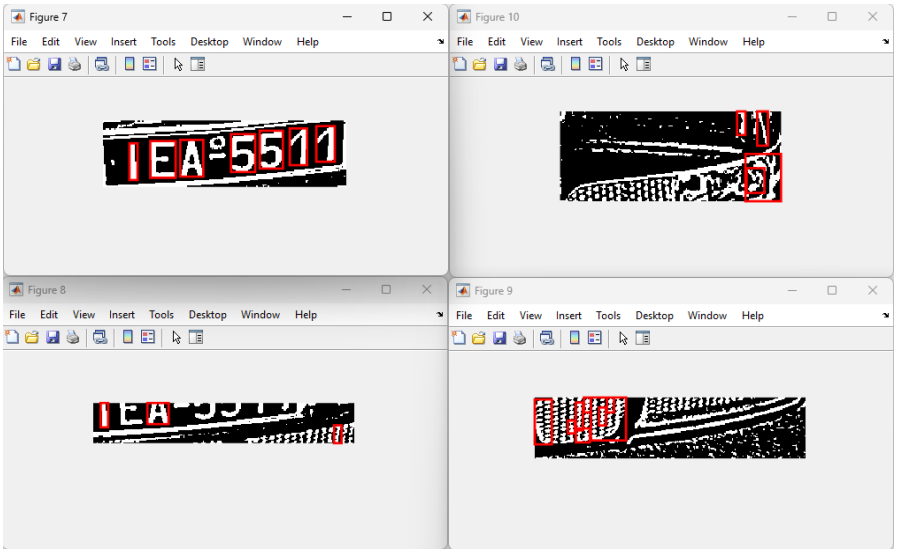

Figura 3: Algunas de las partes *cropeadas* anteriormente con el segundo filtro aplicado

El tercer filtro es un poco más sofisticado, con el objetivo de conseguir conjuntos de caracteres, de manera que, en vez de ir región por región, se recorren todos los pares de regiones posibles (habiéndolos ordenado anteriormente según su posición horizontal, para facilitar los cálculos, *función 13*), y se mira que cumplan los siguientes criterios:

- Que tengan la misma altura (con un margen de error muy pequeño).
- Que tengan una anchura parecida (con un margen un poco mayor a causa de la diferencia de anchura entre una “I” o un “1” y cualquier otro carácter).
- Que no se solapen, mirando en orden que un carácter acaba antes que empieza el siguiente.
- Que no estén demasiado inclinados, mirando el valor absoluto de la pendiente entre cada par de regiones (*función 10*).

Una vez aplicado este filtro los resultados son como los mostrados a continuación:

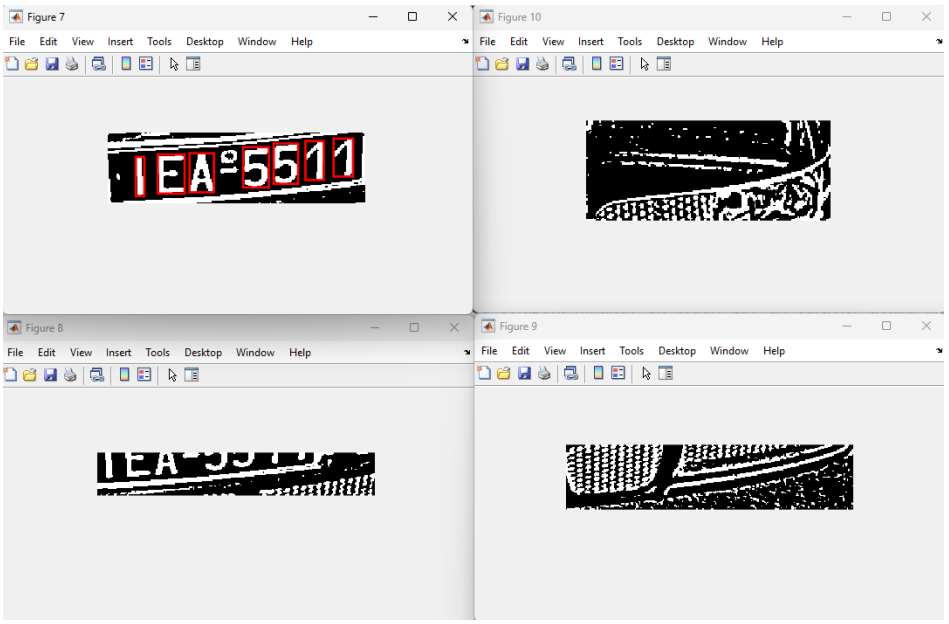

Figura 4: Tercer filtro aplicado a las regiones resultantes del segundo filtro

En este ejemplo anterior, el resultado de la imagen superior izquierda es el mismo que en el mostrado en la figura 3, pero del resto de imágenes ya no hay regiones, dado que no cumplen los requisitos.

Por cuarto y último filtro, a pesar de los buenos resultados del filtro anterior, se ha aplicado para asegurarse de la cantidad correcta de caracteres de la matrícula, y así evitar regiones dispersas, de manera que en este filtro nos guiamos por la alineación de una cierta cantidad de regiones, representando los caracteres en línea de toda matrícula.
Los resultados extraídos de este último filtro (*función 12*) cumplirán los siguientes requisitos:

- La cantidad de regiones con la misma pendiente (cercana a 0, ya que las matrículas pueden estar más o menos inclinadas, pero siempre con un rango máximo dado que lo normal es que estén horizontalmente colocadas) será entre 5 y 7, por si hay algún carácter que no ha sido reconocido por el *regionprops*.

Idóneamente, el resultado final de esta estrategia es el mostrado en la figura siguiente:

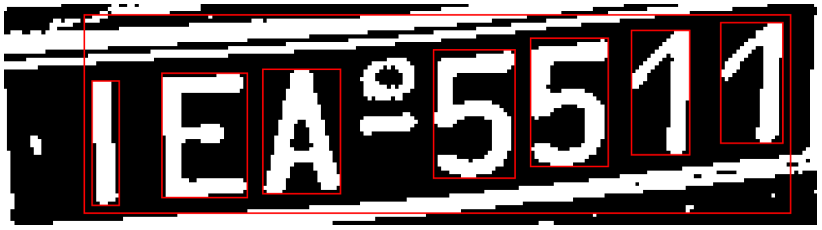

Figura 5: Resultado final idóneo de la estrategia 1

Esta estrategia tiene algunos fallos dado que es muy poco permisiva cuando algún tipo de suciedad une las letras con el borde de la matrícula o cuando las letras están algo borrosas y al binarizar la imagen no se encuentran completas. Para ello, se ha implementado una segunda estrategia para aumentar el número de resultados correctos en caso de que la primera estrategia no haya funcionado correctamente.

Estrategia 2

Esta segunda estrategia (*función 16*) se aplica en caso de que la primera no obtenga todos los caracteres de la matrícula como *regionprops*. Gran parte de la implementación es igual que la primera estrategia, a diferencia de que se introducen nuevas funciones para obtener el resultado deseado.

Preprocesamiento de la imagen

Para el preprocesamiento de la imagen original, se aplica el mismo procedimiento que en la estrategia 1: conversión a escala de grises, suavizado con *filtro mediana*, binarización con *moving averages* y aplicación del método *Canny* (*funciones 1, 2 y 3*). Como esta segunda estrategia dependerá del resultado obtenido del filtrado de regiones, el preprocesado es invariante.

Filtrado de regiones

A la hora de realizar el filtrado de regiones, primero se vuelven a aplicar los mismos cuatro filtros (*funciones 13 y 14*) explicados anteriormente en la estrategia 1. En este caso y como se ha mencionado anteriormente, el resultado obtenido es un conjunto de *regionprops* correspondientes a los carácteres de la matrícula detectada de tamaño diferente a siete, como se muestra a continuación:

Figura 6: Resultado final erróneo de la estrategia 1

Seguidamente, se aplican dos funciones a esta segunda estrategia que permiten conseguir los siete *regionprops* deseados:

En primer lugar, se aplica una transformación geométrica de rotación (*función 6*) a la imagen *cropeada* para que las matrículas que se ven inclinadas debido a la perspectiva de la imagen tomada se muestren rectas (paralelas al eje horizontal).
Para conseguirlo, se realizan los siguientes pasos:

- De las regiones ordenadas de manera creciente según su coordenada *x*, se selecciona el primer y último *BoundingBox* detectado como carácter de los *regionprops*.
- Se obtienen las coordenadas de las esquinas superiores izquierdas de ambos *BoundingBox*, siendo x1 y y1 las coordenadas horizontal y vertical del primero, respectivamente, y lo mismo con x2 y y2 para el último.
- Se calcula la pendiente formada entre estos dos *boundingBox* con la fórmula $slope = (y2 - y1) / (x2 - x1)$.
- Se calcula el ángulo de rotación necesario con la fórmula $angle = rad2deg(\, acos(dot(u,\, v) / (norm(u) * norm(v)))\,)$, siendo $u = [x2 - x1;\, y2 - y1]$, es decir, un vector con las distancias horizontal y vertical entre ambos *BoundingBox* y $v = [x2 - x1;\, y1 - y1]$, un vector con la distancia horizontal entre ambos y 0.

- Se aplica la rotación del ángulo en positivo o negativo según el signo de la pendiente (si la pendiente es negativa, rotación negativa; sino, rotación positiva).

En las dos figuras siguientes se muestra un esquema de la rotación y el resultado de esta transformación:

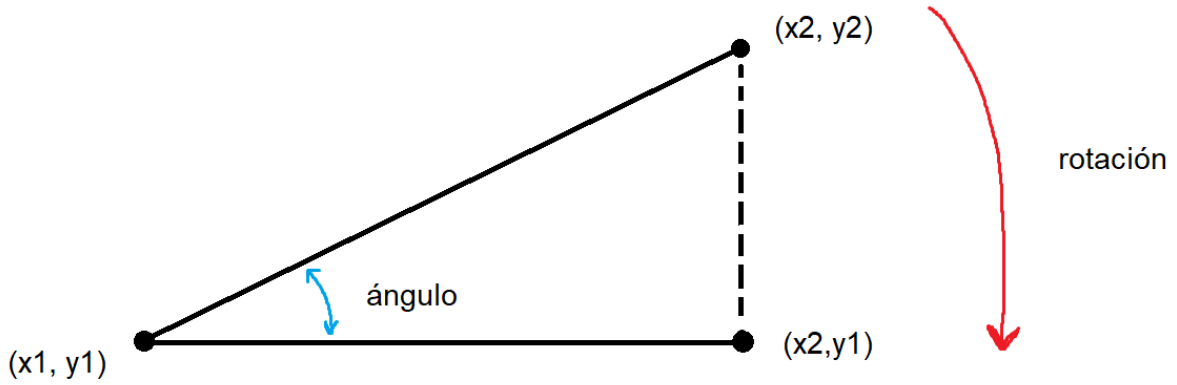

Figura 7: Esquema de la rotación aplicada para tener la matrícula horizontal (se haría de la misma manera si (x2,y2) estuviera por debajo

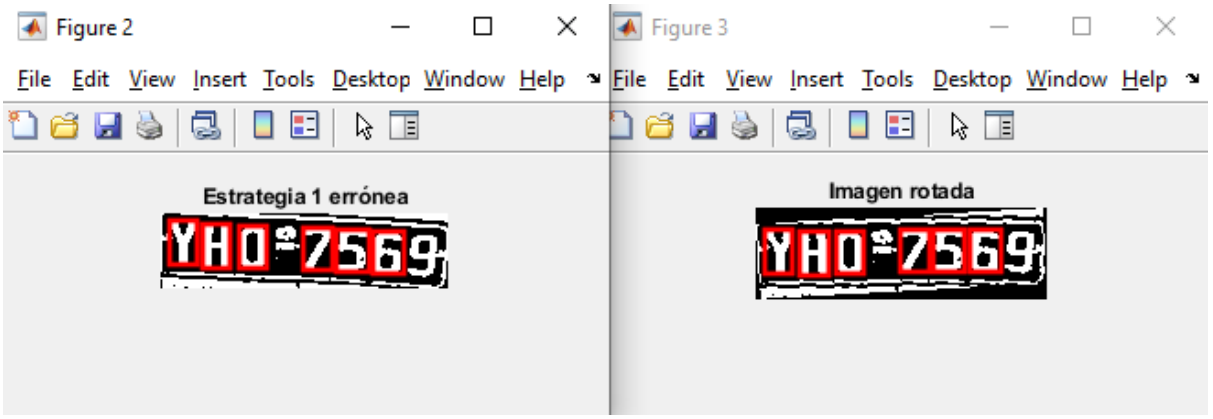

Figura 8: Imagen errónea obtenida de la estrategia 1 antes de aplicar la rotación (izquierda) y después (derecha)

Esta primera transformación se realiza para poder aplicar la segunda función que resolverá el problema de falta de detección de los siete *regionprops* correspondientes a los carácteres de la matrícula. A continuación, la segunda función se encarga de eliminar manualmente los fragmentos blancos que no forman parte de los caracteres. Como los *regionprops* detectados están alineados horizontalmente, es posible hacerlo sin eliminar partes de caracteres. Para ello, primero se obtiene la coordenada superior izquierda del primer *BoundingBox* y se pone valor 0 a todos los píxeles comprendidos entre la fila 1 y la componente *y* de esa coordenada para todas las columnas, es decir, se elimina la parte superior de los *regionprops*. Se hace lo mismo para la parte inferior, la parte izquierda y la parte

derecha, de esta manera se deja en blanco únicamente lo comprendido dentro de los *regionprops*. El resultado se ve así:

Figura 9: Resultado después de eliminar regiones blancas en la estrategia 2

Finalmente, a esta estrategia se le aplican de nuevo los cuatro filtros de la estrategia 1 (*funciones 13 y 14*) porque durante ese proceso es posible que se hayan detectado *regionprops* no correspondientes a caracteres de la matrícula (como se observa en la figura 8 con el símbolo entre letras y números, por ejemplo). Además, también se obtiene un *regionprop* correspondiente a toda la placa (*función 7*), es decir, englobando los siete caracteres, como se muestra a continuación:

Figura 10: Resultado obtenido después de aplicar la estrategia 2.

Esta segunda estrategia, a pesar de que sigue siendo algo estricta ya que depende mucho de que todos los caracteres de la matrícula formen cada uno una región, gracias a las operaciones realizadas anteriormente consigue evitar imperfecciones que impedían a la estrategia 1 funcionar correctamente.

Reconocimiento de caracteres

La segunda fase del proyecto ha sido el reconocimiento de caracteres.
Para ello, se ha utilizado el método de *Classification Using HOG Features,* ya que es muy útil para describir las formas de los números y letras.

Dadas las pocas imágenes de plantilla para poder reconocer los caracteres, no se ha utilizado *training* en sí, sino que para el reconocimiento de una letra o número se ha decidido comparar los histogramas de gradientes orientados con cada una de las posibilidades y clasificar el carácter al que mejor resultado tenga.

El primer paso ha sido tener las plantillas para la comparación de caracteres.
Se ha editado las imágenes *Greek-License-Plate-Font-old.jpg* y *Greek-License-Plate-Font-2004.svg.jpg* para incluir algunas letras (como la letra 'O') que se encontraban en algunas matrículas pero no estaban presentes en las imágenes, y también se han creado manualmente otras dos plantillas (*modelo3.jpg* y *modelo4.jpg)* con otros tipos de formas de carácter para tener más probabilidades de reconocer la letra correctamente por la forma.

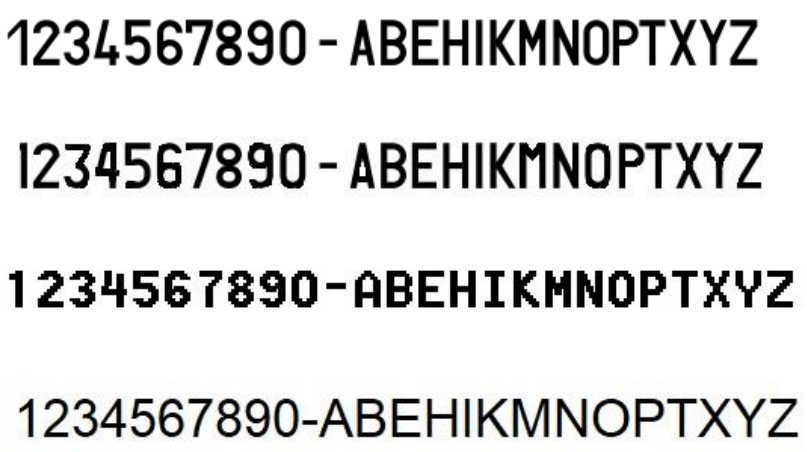

Figura 11: plantillas para la comparación de caracteres

Una vez importadas las imágenes, en cada imagen se ha *cropeado* cada letra con *regionprops* y se han reescalado y aplicado *padding* para que finalmente todas las letras se encuentren dentro de un cuadrado de exactamente 200x200 píxeles (*función 20*). A su vez, se ha hecho un pequeño *imdilate* para suavizar un poco las letras y mejorar en un futuro la mejor clasificación posible.

Después de todas estas operaciones, se ha extraído el *HOG* de cada uno de los caracteres (*función 17*) y el resultado se ha guardado en un *struct* (el cual es diferente para letras y números).

Figura 12: *HOG* de cada uno de los caracteres de *Greek-License-Plate-Font-2004.svg.jpg*

A partir de aquí ya es posible el reconocimiento de caracteres.

Por último, para el reconocimiento de las matrículas, para cada matrícula se extraen los caracteres (resultado de la estrategia 1 o la estrategia 2), y se les aplica el mismo tratado que a las letras plantilla (*función 20*), pero con algunas operaciones adicionales para eliminar pequeñas estructuras que no pertenecen a la letra o pulir más los bordes. Además, por último se hace el mismo *imdilate* que con las letras plantilla como última operación para coincidir lo máximo posible con el mismo perfilado de bordes para la correcta clasificación.

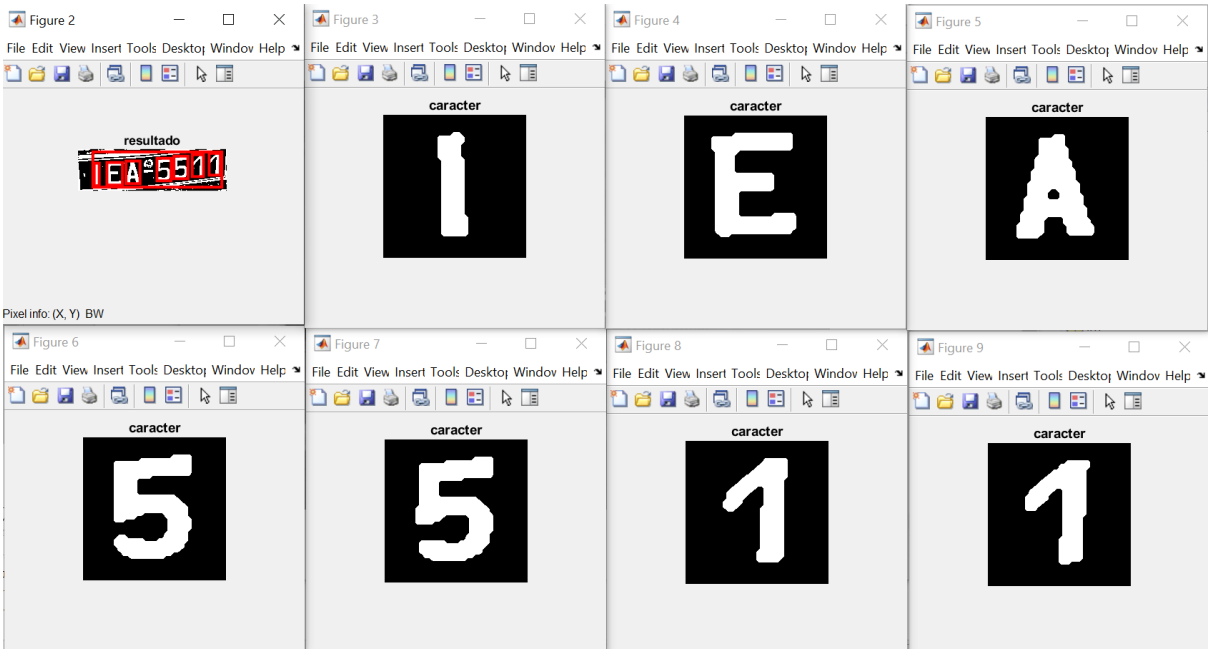

Figura 13: *cropeado* de los caracteres de la matrícula modificados

Una vez extraído el *HOG* de cada uno carácteres de la matrícula, se compara cada uno de ellos con los *HOGs* guardados de cada una de las plantillas de letras mediante el método para calcular distancia entre histogramas *Chi Square* (*función 21*), y la letra o número que menor distancia dé será a dónde se clasifica el carácter que se está evaluando (*función 18*).

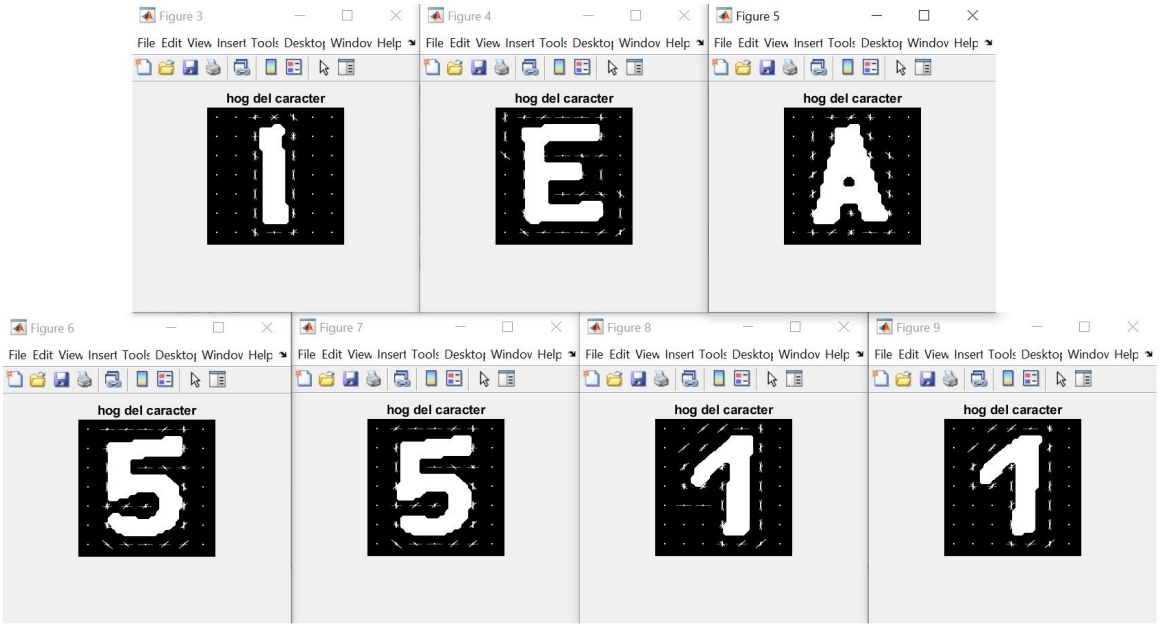

Figura 14: *HOG* de cada carácter de la matrícula

El resultado de reconocer el carácter es guardado en un *string* que contendrá la matrícula entera reconocida una vez finalizado el proceso.

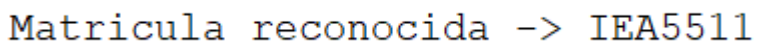

Figura 15: resultado final del reconocimiento de los caracteres de la matrícula

Este método, a pesar de no ser tan efectivo como un buen *training* con muchas imágenes dedicadas a ello y *SVM*, da bastantes buenos resultados con las pocas imágenes disponibles para ello. Por ese mismo motivo las imágenes con histograma de gradientes orientados muy parecidos son de vez en cuando confundidos dado que no es un método tan sofisticado.

Además de este método, se quiso probar otro basado en *Object Detection in a Cluttered Scene Using Point Feature Matching*. Consiste en encontrar un objeto específico, en este caso un carácter, en una escena mediante la búsqueda de correspondencias de puntos entre una imagen de referencia y una imagen de destino. Para ello, se detectan las características de transformación invariantes a escala (*SIFT features*) y se obtener sus 200 puntos más fuertes, tanto de los caracteres de la matrícula como los de la imagen de destino, que es la imagen proporcionada "*Greek-License-Plate-Font-old.jpg*". A partir de aquí, se comparan los puntos obtenidos carácter por carácter y el que más coincidencias tenga es el correcto.
Debido a la complejidad de este algoritmo, los resultados obtenidos fueron mucho peores que los del primer método ya que la imagen de coincidencias de puntos putativos relacionaba características que no correspondían.
La idea inicial para este método fue comparar cada carácter de matrícula con la imagen de destino "*Greek-License-Plate-Font-old.jpg*" pero como salían coincidencias de puntos putativos muy dispersos, se decidió dividir los caracteres de la imagen de destino y compararlos uno a uno. De esta manera, el coste del programa aumentó significativamente comparado con el primer método implementado.
Por estas razones, se decidió usar el método de *Classification Using HOG Features.*

Estadísticas

A partir de los resultados generados por el detector de matrículas tras procesar las 67 imágenes de coches proporcionadas por la asignatura, se han generado varios gráficos para visualizar las estadísticas de los datos obtenidos.

En este primer gráfico se muestra el porcentaje de matrículas que han sido reconocidas correctamente y las que no. Como se puede observar, para más de la mitad del *dataset* el algoritmo ha funcionado como se esperaba pero aún así, sigue existiendo un margen de mejora para aumentar su efectividad.

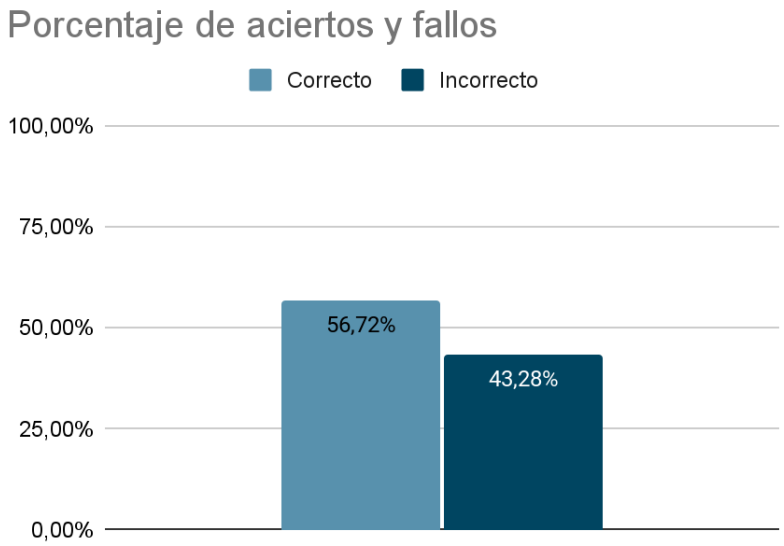

Figura 16: gráfico del porcentaje de aciertos y fallos del detector de matrículas

El siguiente gráfico visualiza el porcentaje de uso de las dos estrategias dentro de las matrículas correctamente reconocidas. Se puede ver que el 58,21% de las matrículas son reconocidas por la estrategia 1 y, por tanto, no requieren de la utilización de la estrategia 2. El resto de las imágenes sí que necesitan ser tratadas por ambas estrategias.

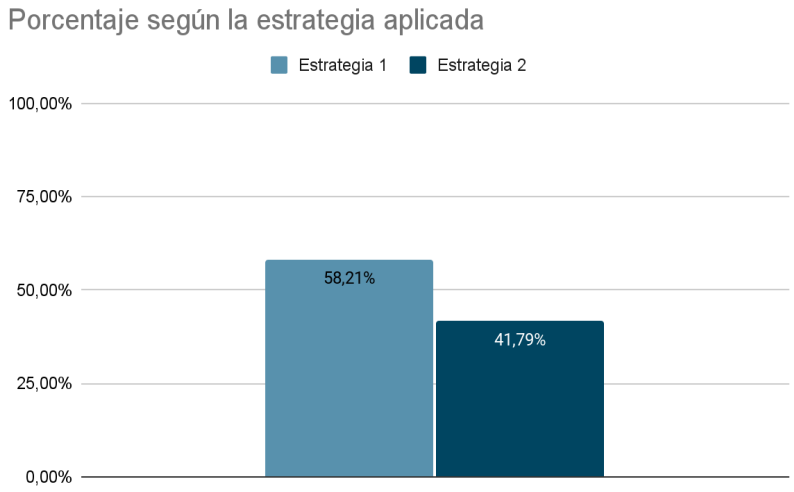

Figura 17: gráfico del porcentaje de uso de las dos estrategias

La estadística del siguiente gráfico muestra la cantidad de falsos positivos y falsos negativos localizados por el detector de matrículas. En este programa, se consideran falsos positivos aquellos resultados donde los caracteres de la matrícula reconocida son incorrectos. Por ejemplo, lo sucedido con la imagen *DSCN0412.jpg*. Por otro lado, se consideran falsos negativos aquellos resultados donde no se ha podido reconocer la ubicación de la matrícula. Por ejemplo, lo que pasa con la imagen *DSCN0413.jpg*.

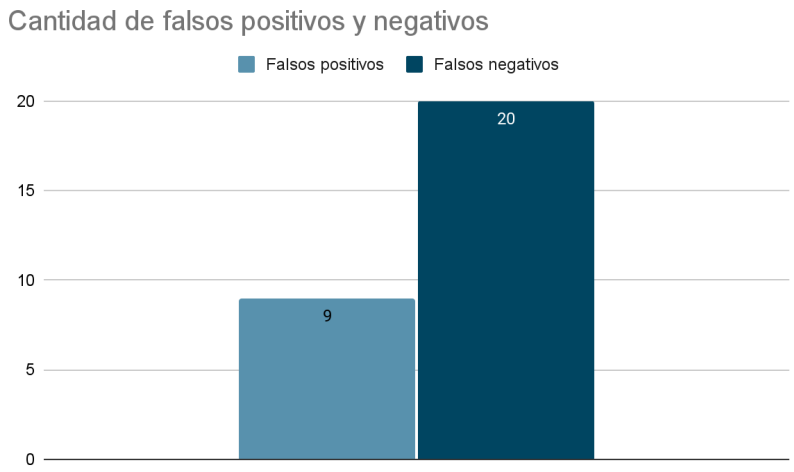

Figura 18: gráfico de la cantidad de falsos positivos y negativos

En adición, se ha comparado el porcentaje de letras reconocidas correctamente por los dos métodos de reconocimiento de caracteres implementados. De esta manera, se comprueba cómo el algoritmo seleccionado produce mucho más acierto que el descartado.

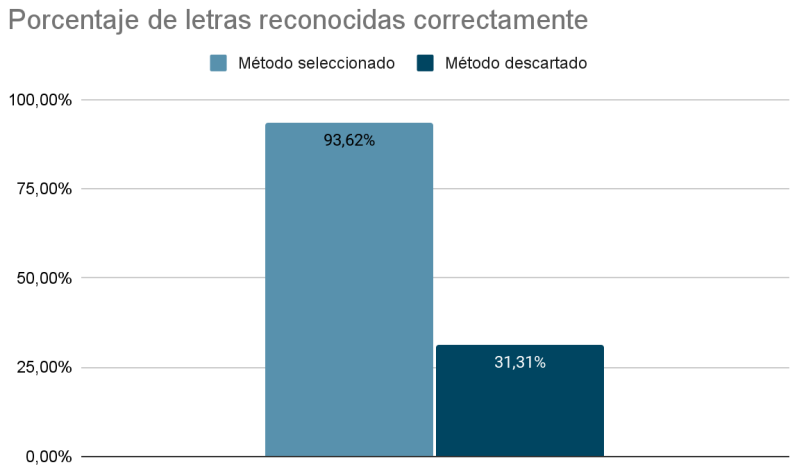

Figura 19: gráfico del porcentaje de letras reconocidas correctamente por los dos métodos

Por último, se ha comparado el tiempo de ejecución de todo el código implementado, comparando las 67 matrículas de la carpeta de imágenes proporcionada, pero cambiado el método de reconocimiento de caracteres. Así se comprueba que, como se ha mencionado anteriormente, el algoritmo seleccionado es mucho tres veces más rápido que el descartado. Destacar que el tiempo mostrado ha sido haciendo output de cada imagen original y el resultado de la detección de la matrícula y sus *regionprops*, como en la entrega del checkpoint, y con un portátil de gama media.

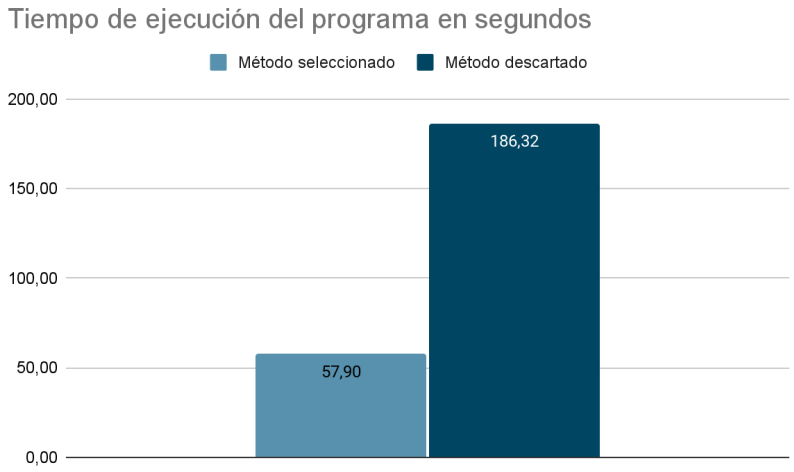

Figura 20: gráfico del tiempo de ejecución del programa

Anexo de las funciones del código

El código del programa está dividido en 3 partes:

- **detector_matriculas.m:** Script principal, donde se "entrenan" las dos estructuras de datos que contienen los *HOGS* de las letras y números plantilla respectivamente. Luego se leen todas las imágenes de la carpeta *day_color(small sample)*, y se procede a aplicar las dos estrategias (dependiendo el acierto de la primera) y a reconocer cada matrícula.

- **detectorFunctionContainer.m:** Contiene todas las funciones necesarias para la detección de la matrícula y sus letras. Sus principales funciones son *strat1* (correspondiente a la estrategia 1) y *strat2* (correspondiente a la estrategia 2).

- **recognizerFunctionContainer.m:** Contiene todas las funciones necesarias para el reconocimiento de cada letra. Sus principales funciones son *training* y *recognize*.

Funciones *detectorFunctionContainer*

1. *smoothedCar = imageSmoothing(im)* : Suaviza la imagen del coche utilizando filtro mediana
2. *binarizedCar = imageBinarization(im)* : Binariza la imagen del coche usando *moving averages*
3. *edgesCar = imageEdges(im)* : Devuelve la imagen de bordes usando el método *Canny*

4. *displayBoundingBoxLP(imagenMatricula, letras, placa)* : Muestra la imagen de la placa con los *BoundingBox* conseguidos
5. *n = nSlopesMinValue(v) :* Devuelve el número de *slopes* con cierto valor mínimo
6. *image = rotateImage(candidatesToLetters, Icropped)* : Rota la imagen *cropeada* de manera que quede horizontal usando el primer y último candidato a letras previamente ordenados crecientemente según su coordenada *x*
7. *plate = boxLetters(stats)* : Devuelve el *BoundingBox* que engloba todas las letras detectadas
8. *candidatesToPlates = filterByLp(imagen_bordes)* : Filtra la imagen para encontrar candidatos a placas de matrícula, devolviendo sus *BoundingBoxes*
9. *detectedPlates = removeUselessPlates(stats)* : Elimina los *BoundingBoxes* de las placas detectadas que tienen más *BoundingBoxes* dentro, ya que un candidato a palca no debe tener candidatos dentro
10. *detectedLetters = filterBySimilarityAndNearLowSlope(stats)* : Filtra una serie de *BoundingBoxes* para encontrar pares de letras con poca pendiente entre ellas, que estén cerca, tengan una altura muy parecida y que no se solapen entre ellas
11. *detectedLetters = filterByMinAreaAndRa(stats)* : Filtra una serie de *BoundingBoxes* para encontrar candidatos a letras mirando que tengan un área mínima y una relación de aspecto de manera que su altura sea mayor o igual que su anchura
12. *detectedLetters = filterByAmountOfSlopes(stats)* : Filtra una serie de *BoundingBoxes* para encontrar candidatos a letras mirando que una cierta cantidad de *BoundingBoxes* (5 a 7) se encuentren más o menos en fila con un pendiente pequeño
13. *sortedStats = sortStats(stats)* : Ordena los *BoundingBoxes* según las coordenadas *x* de su esquina superior izquierda
14. *candidatesToLetters = filtersForStrat1(stats)* : Se junta los filtros 10,11,12 para la estrategia 1
15. *[placa, letras, imagen_crop] = strat1(imagen_coche_bordes, imagen_coche_binarizada)* : Se aplica la estrategia 1, teniendo como output la *BoundingBox* de la placa, los *BoundingBoxes* de las letras, y la imagen en sí de la placa detectada
16. *[placa, letras, imagen_crop] = strat2(imagen_coche_bordes, imagen_coche_binarizada)* : Mismo output que la estrategia 1, pero aplicando los filtros y criterios de la estrategia 2.

Funciones

recognizerFunctionContainer

17. *[trainingDigits, trainingLetters] = training(fonts, digits, letters)* : Consigue los *HOGs* de todas las letras de todas las plantillas pasadas en *fonts* y devuelve dos estructuras de datos *trainingDigits* y *trainingLetters* que contienen para cada letra y número respectivamente 4 *HOGs* pertenecientes a ellos (uno por cada plantilla usada). Estas dos estructuras serán utilizadas para clasificar cada letra en el reconocimiento de la matrícula.
18. *matricula = recognize(imagen, letras, trainingDigits, trainingLetters)* : Devuelve un string con la matrícula reconocida usando las estructuras *trainingDigits* y *trainingLetters* para la clasificación de cada caracter.
19. *sortedStats = sortStats(stats)* : Función equivalente a la función 13 del *detectorFunctionContainer*.
20. *image = squareAndResize(im, desiredHeight)* : Aplica una serie de transformaciones para que tanto los caracteres de las plantillas como los caracteres de las matrículas tengan el mismo tamaño, y sus histogramas de gradientes se puedan comparar, ya que es un requisito que el tamaño de los dos histogramas sea el mismo para poder aplicar la distancia entre ellos.
21. *D = distChiSq(X,Y)* : Calcula la distancia entre dos histogramas usando el método *Chi Square*. Esta es la única función utilizada que inicialmente se utilizó código de otra fuente, pero viendo la facilidad de la fórmula y como la función realmente no era eficiente, se hizo manualmente de manera vectorizada, y lo único que se guardó de la implementación anterior fue la manera de poder dividir cuando el denominador es 0, sumando *eps (=2.2204e-16)* a este denominador (la distancia entre un número cualquiera y el siguiente número en la representación interna en punto flotante, dicho de otra manera, es la mínima diferencia que puede haber entre dos números diferentes). La implementación anterior se puede ver en el siguiente link: http://www.cs.columbia.edu/~mmerler/project/code/pdist2.m

CLASE CON FUNCIONES PARA LA DETEC-CION DE LA POSICION DE LA MATRICULA

```
classdef detectorFunctionContainer < handle
    methods (Static)
        % Suaviza la imagen del coche utilizando filtro mediana
        function smoothedCar = imageSmoothing(im)
            smoothedCar = medfilt2(im, [2,2]);
        end

        % Binariza la imagen del coche usando moving averages
        function binarizedCar = imageBinarization(im)
            h = ones(10)/10/10;
            promedio = imfilter(im, h, 'conv', 'replicate');
            k = 5;
            binarizedCar = im > (promedio-k);
        end

        function edgesCar = imageEdges(im)
            edgesCar = edge(im, 'Canny');
        end

        % Muestra la imagen de la placa con los boundingbox conseguidos
        function displayBoundingBoxLP(imagenMatricula, letras, placa)
            figure, imshow(imagenMatricula), title('resultado')
            hold on
            for i = 1:length(letras)
                rectangle('Position',
 letras(i).BoundingBox, 'EdgeColor', 'r', 'LineWidth', 2);
            end
            rectangle('Position', placa, 'EdgeColor', 'r', 'LineWidth', 2);
            impixelinfo
        end

        % Devuelve el numero de slopes con cierto valor minimo
        function n = nSlopesMinValue(v)
            minVal = 0.2;
            n = 0;
            for i=1 : length(v)
                if v(i) <= minVal
                    n = n + 1;
                end
            end
        end

        function image = rotateImage(candidatesToLetters, Icropped)
            image = Icropped;
            candidatesToLetters =
 detectorFunctionContainer.sortStats(candidatesToLetters);
            region1 = candidatesToLetters(1).BoundingBox;
            region2 = candidatesToLetters(end).BoundingBox;
```

```
            x1 = region1(1); y1 = region1(2);
            x2 = region2(1); y2 = region2(2);

            slope = (y2 - y1) / (x2 - x1);

            u = [x2-x1; y2-y1]; v = [x2-x1; y1-y1];
            angle = rad2deg( acos(dot(u, v) / (norm(u)*norm(v))) );
            if (~isinf(angle) && ~isnan(angle))
                if slope < 0
                    image = imrotate(Icropped, -angle);
                else
                    image = imrotate(Icropped, angle);

                end
            end
        end

        % Devuelve el BoundingBox de todas las letras detectadas
        function plate = boxLetters(stats)
            x_min_left = stats(1).BoundingBox(1); y_min_up =
stats(1).BoundingBox(2);
            x_max_right = stats(1).BoundingBox(1) + stats(1).BoundingBox(3);
            y_max_down = stats(1).BoundingBox(2) + stats(1).BoundingBox(4);
            for i=2 : length(stats)
                region = stats(i).BoundingBox;
                x_left = region(1); y_up = region(2); width = region(3);
height = region(4);
                x_right = x_left + width;
                y_down = y_up + height;
                if (x_left < x_min_left)
                    x_min_left = x_left;
                end
                if (y_up < y_min_up)
                    y_min_up = y_up;
                end
                if (x_right > x_max_right)
                    x_max_right = x_right;
                end
                if (y_down > y_max_down)
                    y_max_down = y_down;
                end
            end
            offset = 2;
            x_left_up = x_min_left - offset;
            y_left_up = y_min_up - offset;
            x_right_low = x_max_right + offset;
            y_right_low = y_max_down + offset;
            height = y_right_low - y_left_up;
            width = x_right_low - x_left_up;
            plate =  [x_left_up, y_left_up, width, height];
        end

        % Filtra la imagen para encontrar placas de matricula, devolviendo
        % los boundingboxes de los candidatos a placas encontrados
```

```
        function candidatesToPlate = filterByLp(imagen_bordes)
            stats = regionprops(imagen_bordes, "BoundingBox", "Area");
            candidatesIndices = [];
            candidatesToPlate = struct;
            for i = 1:length(stats)
                region = stats(i).BoundingBox;
                area = stats(i).Area;
                width = region(3);
                height = region(4);
                area_minima = 200;
                if (width >= 2*height) && (area > area_minima) && (width <=
 7*height)
                    candidatesIndices(end+1) = i;
                end
            end

            for i=1 : length(candidatesIndices)
                region = stats(candidatesIndices(i)).BoundingBox;
                candidatesToPlate(end+1).BoundingBox = region;
            end
            candidatesToPlate(1) = [];
            candidatesToPlate =
 detectorFunctionContainer.removeUselessPlates(candidatesToPlate);
        end

        % Elimina los BoundingBoxes de las placas detectadas que tienen mas
        % BoundingBoxes dentro, ya que un candidato a placa no debe tener
        % candidatos dentro
        function detectedPlates = removeUselessPlates(stats)
            detectedPlatesIndices = [];
            detectedPlates = struct;
            for i=1 : length(stats)
                remove = false;
                region = stats(i).BoundingBox;
                x = region(1); y = region(2); width = region(3); height =
 region(4);
                for j=1 : length(stats)
                    if i ~= j
                        region2 = stats(j).BoundingBox;
                        x2 = region2(1); y2 = region2(2); width2 = region2(3);
 height2 = region2(4);
                        if (x < x2) && (y < y2) && (x+width > x2+width2) && (y
+height > y2+height2)
                            remove = true;
                        end

                        if (x < x2) && (y < y2) && (x2 < x+width) && (y2 < y
+height)
                            widthIntersec = min(x+width, x2+width2)-x2;
                            heightIntersec = min(y+height, y2+height2)-y2;
                            areaIntersec = widthIntersec * heightIntersec;
                            percentageArea = areaIntersec / (width*height);
                            if percentageArea > 0.70
                                remove = true;
```

```
                        end
                    end
                end
            end
            if ~remove
                detectedPlatesIndices(end+1) = i;
            end
        end

        for i=1 : length(detectedPlatesIndices)
            region = stats(detectedPlatesIndices(i)).BoundingBox;
            detectedPlates(end+1).BoundingBox = region;
        end
        detectedPlates(1) = [];
    end

    % Filtra una serie de BoundingBoxes para encontrar pares de letras con
 poca pendiente entre ellas, que
    % esten cerca, tengan una altura muy parecida, y que no se solapen
 entre ellas
    function detectedLetters = filterBySimilarityAndNearLowSlope(stats)
        detectedLettersIndices = [];
        detectedLetters = struct;
        for i = 1:size(stats, 2)
            region = stats(i).BoundingBox;
            x_left_1 = region(1); y_up_1 = region(2); width = region(3);
 height = region(4);
            for j=i+1 : size(stats, 2)
                region2 = stats(j).BoundingBox;
                x_left_2 = region2(1); y_up_2 = region2(2); height2 =
 region2(4);
                slope = (y_up_2 - y_up_1) / (x_left_2 - x_left_1);
                if (x_left_2 < x_left_1+(width*3)) && (height2 <= height
+4) && (height2 >= height-4) && (x_left_2 > x_left_1+width) && (abs(slope) <
 0.2)
                    detectedLettersIndices(end+1) = i;
                    detectedLettersIndices(end+1) = j;
                end

            end
        end
        % eliminamos los repetidos
        while ~isempty(detectedLettersIndices)
            region = stats(detectedLettersIndices(1)).BoundingBox;
            detectedLetters(end+1).BoundingBox = region;
            detectedLettersIndices =
 detectedLettersIndices(detectedLettersIndices~=detectedLettersIndices(1));
        end

        detectedLetters(1) = [];
    end

    % Filtra una serie de BoundingBoxes para encontrar candidatos a
    % letras mirando que tengan una area minima y una relacion de
```

```
        % aspecto de manera que su altura sea mayor o igual que su anchura
        function detectedLetters = filterByMinAreaAndRa(stats)
            detectedLettersIndices = [];
            detectedLetters = struct;
            for i = 1:size(stats)
                region = stats(i).BoundingBox;
                area = stats(i).Area;
                width = region(3); height = region(4);
                area_minima = 50;
                if (height >= width) && (area > area_minima)
                    detectedLettersIndices(end+1) = i;
                end
            end

            for i=1 : length(detectedLettersIndices)
                region = stats(detectedLettersIndices(i)).BoundingBox;
                detectedLetters(end+1).BoundingBox = region;
            end
            detectedLetters(1) = [];
        end

        % Filtra una serie de BoundingBoxes para encontrar candidatos a
        % letras mirando que una cierta cantidad de BoundingBoxes (5 a 7)
        % esten mas o menos en fila con un pendiente pequeño
        function detectedLetters = filterByAmountOfSlopes(stats)
            detectedLettersIndices = [];
            detectedLetters = struct;
            for i=1 : size(stats, 2)
                region = stats(i).BoundingBox;
                x_left_1 = region(1); y_up_1 = region(2);
                slopes = [];
                for j=1 : size(stats, 2)
                    region2 = stats(j).BoundingBox;
                    x_left_2 = region2(1); y_up_2 = region2(2);
                    slope = (y_up_2 - y_up_1) / (x_left_2 - x_left_1);
                    slopes(end+1) = abs(slope);
                end

                slopes = sort(slopes);
                belongs_plate = false;
                if detectorFunctionContainer.nSlopesMinValue(slopes) >= 5 &&
detectorFunctionContainer.nSlopesMinValue(slopes) <= 7
                    belongs_plate = true;
                end

                if belongs_plate
                    detectedLettersIndices(end+1) = i;
                end
            end

            % eliminamos los repetidos
            while size(detectedLettersIndices, 2) ~= 0
                region = stats(detectedLettersIndices(1)).BoundingBox;
                detectedLetters(end+1).BoundingBox = region;
```

```
                detectedLettersIndices =
detectedLettersIndices(detectedLettersIndices~=detectedLettersIndices(1));
            end

            detectedLetters(1) = [];
        end

        function sortedStats = sortStats(stats)
            T = struct2table(stats);
            sortedTable = sortrows(T, 'BoundingBox');
            sortedStats = table2struct(sortedTable);
        end

        % Juntamos los filtros para la estrategia 1
        function candidatesToLetters = filtersForStrat1(stats)
            % ordenamos de menor a mayor coordenada x
            sorted_stats = detectorFunctionContainer.sortStats(stats);

            % filtrado 1
            possibleLetters =
detectorFunctionContainer.filterByMinAreaAndRa(sorted_stats);

            % filtrado 2
            possibleLetters2 =
detectorFunctionContainer.filterBySimilarityAndNearLowSlope(possibleLetters);

            % filtrado 3
            candidatesToLetters =
detectorFunctionContainer.filterByAmountOfSlopes(possibleLetters2);
        end

        % Se aplica la estrategia 1, teniendo como output la BoundingBox de
        % la placa, las BoundingBoxes de las letras, y la imagen en si de
        % la placa detectada
        function [placa, letras, imagen_crop] = strat1(imagen_coche_bordes,
imagen_coche_binarizada)
            possiblePlates =
detectorFunctionContainer.filterByLp(imagen_coche_bordes);
            placa = [];
            letras = struct;
            area = 600*800; % area maxima posible (toda la imagen)
            imagen_crop = imagen_coche_bordes;
            for i=1 : length(possiblePlates)
                region = possiblePlates(i).BoundingBox;
                Icropped = ~imcrop(imagen_coche_binarizada, region);
                stats = regionprops(Icropped, "BoundingBox", "Area");
                if isempty(stats)  % en caso de que no haya detectado ningun
tipo de posible letra, ir a la siguiente iteracion
                    continue;
                end
                letras = detectorFunctionContainer.filtersForStrat1(stats);
                if isempty(letras)  % en caso de que no haya detectado ningun
tipo de posible letra, ir a la siguiente iteracion
                    continue;
```

```
                end

                imagen_crop = Icropped;
                if (length(letras) == 7)
                    area_placa = region(3)*region(4);
                    if (area_placa < area)
                        placa = detectorFunctionContainer.boxLetters(letras);
                        break;
                    end
                end
            end
        end


        function [placa, letras, imagen_crop] = strat2(imagen_coche_bordes,
imagen_coche_binarizada)
            posiblesplacas =
detectorFunctionContainer.filterByLp(imagen_coche_bordes);
            placa = [];
            letras = struct;
            imagen_crop = imagen_coche_binarizada;
            for i=1:length(posiblesplacas)
                placa = posiblesplacas(i).BoundingBox;
                Icropped = ~imcrop(imagen_coche_binarizada, placa);
                posiblesletras = regionprops(Icropped, "BoundingBox", "Area");
                posiblesletras =
detectorFunctionContainer.filtersForStrat1(posiblesletras);
                if isempty(posiblesletras)  % en caso de que no haya detectado
ningun tipo de posible letra, ir a la siguiente iteracion
                   continue;
                end

                Icropped =
detectorFunctionContainer.rotateImage(posiblesletras, Icropped);
                posiblesletras = regionprops(Icropped, "BoundingBox", "Area");
                posiblesletras =
detectorFunctionContainer.filtersForStrat1(posiblesletras);
                if isempty(posiblesletras)  % en caso de que no haya detectado
ningun tipo de posible letra, ir a la siguiente iteracion
                   continue;
                end

                region = posiblesletras(1).BoundingBox;
                x = round(region(1)); y = round(region(2)); height =
region(4);
                Icropped(1:y, :) = 0; Icropped(y+height:end, :) = 0;
Icropped(:, 1:5) = 0; Icropped(:, end-5:end) = 0;
                posiblesletras = regionprops(Icropped, "BoundingBox", "Area");
                if isempty(posiblesletras)  % en caso de que no haya detectado
ningun tipo de posible letra, ir a la siguiente iteracion
                   continue;
                end
                letras =
detectorFunctionContainer.filtersForStrat1(posiblesletras);
```

```
                if isempty(letras)  % en caso de que no haya detectado ningun
 tipo de posible letra, ir a la siguiente iteracion
                   continue;
                end
                imagen_crop = Icropped;
                placa = detectorFunctionContainer.boxLetters(letras);
                % posiblemente aqui si hay varias opciones de
                % resultado, elegir la mejor de alguna manera
                if (length(letras) == 7)    % en caso de que haya detectado ya
 una posible matricula, acabar el proceso
                    break;
                end

            end
        end

    end
end
```

```
ans =

  detectorFunctionContainer with no properties.
```

Published with MATLAB® R2022a

CLASE CON FUNCIONES PARA RECONOCER LOS CARACTERES DE LA MATRICULA

```
classdef recognizerFunctionContainer < handle
    properties

    end
    methods (Static)
        function [trainingDigits, trainingLetters] = training(fonts, digits,
 letters)
            trainingDigits = struct;
            trainingLetters = struct;
            digitsHistograms = [];
            lettersHistograms = [];
            for x=1 : size(fonts, 3)
                im = fonts(:,:,x);
                font = ~imbinarize(im);
                letras_train = regionprops(font, 'BoundingBox');

                for i=1 : length(letras_train)
                    letra = imcrop(font, letras_train(i).BoundingBox);
                    width = letras_train(i).BoundingBox(3); height =
 letras_train(i).BoundingBox(4);
                    desiredHeight = 200;
                    if (width > height)
                        continue;
                    end
                    indexForLetter = i-11; % con las plantillas para
 matriculas griegas, los numeros son los 10 primeros
                    letra =
 recognizerFunctionContainer.squareAndResizeImage(letra, desiredHeight);
                    %figure, imshow(letra)
                    letra = imdilate(letra, strel('disk', 3));
                    hog_30x30 = extractHOGFeatures(letra,'CellSize',[28 28]);
  % el que mejores resultados ha dado es 30 30
                    if i < 11
                        digitsHistograms = [digitsHistograms;hog_30x30];
                    else
                        trainingLetters(indexForLetter).Letter =
 letters(indexForLetter);
                        lettersHistograms = [lettersHistograms;hog_30x30];
                    end
                end
            end
            for i=1 : length(digits)
                % metemos para cada digito tanto el de la primera imagen
                % como el de la segunda
                trainingDigits(i).Digit = digits(i);
                histograms = cat(1, digitsHistograms(i,:), digitsHistograms(i
+length(digits),:), digitsHistograms(i+2*length(digits),:), digitsHistograms(i
+3*length(digits),:));
```

```
                trainingDigits(i).Histogram = histograms;
            end
            for i=1 : length(letters)
                trainingLetters(indexForLetter).Letter = letters(i);
                histograms = cat(1, lettersHistograms(i,:),
 lettersHistograms(i+length(letters),:), lettersHistograms(i
+2*length(letters),:), lettersHistograms(i+3*length(letters),:));
                trainingLetters(i).Histogram = histograms;
            end

        end

        function matricula = recognize(imagen, letras, trainingDigits,
 trainingLetters)
            matricula = "";
            % primero ordenar letras de menor a mayor x
            letras = recognizerFunctionContainer.sortStats(letras);
            for i=1 : length(letras)
                letra = letras(i);
                im_letra = imcrop(imagen, letra.BoundingBox);
                desiredHeight = 200;
                im_letra =
 recognizerFunctionContainer.squareAndResizeImage(im_letra, desiredHeight);

                % operaciones morfologicas para ayudar al reconocimiento
                %im_letra = imclose(im_letra, strel('square', 20));


                ero = imerode(im_letra, strel('rectangle', [20,15]));   %
 conseguimos marca de solo el caracter
                im_letra = imreconstruct(ero, im_letra);                %
 quitamos elementos que sobran que no pertenezcan a la letra
                %para quedarse con solo el objeto mas grande
                im_letra = imclose(im_letra, strel('disk', 5));         %
 cerramos partes pequeñas para suavizar lo maximo posible las letras y numeros
                im_letra = imdilate(im_letra, strel('disk', 5));        %
 agrandamos un poco (de la misma manera que con las imagenes de training)
                %figure, imshow(im_letra)
                hog_30x30 = extractHOGFeatures(im_letra,'CellSize',[28 28]);

                if i < 4
                    min_distance = -1;
                    min_distance_index = -1;
                    for j=1 : length(trainingLetters)
                        for k=1 : size(trainingLetters(j).Histogram, 1)
                            distance =
 recognizerFunctionContainer.distChiSq(hog_30x30,
 trainingLetters(j).Histogram(k,:));
                            % al acabar el loop conseguiremos el que tenga
 distancia menor,
                            % es decir, el que mas se parezca
                            if min_distance == -1
                                min_distance = distance;
                                min_distance_index = j;
```

```
                            else
                                if distance < min_distance
                                    min_distance = distance;
                                    min_distance_index = j;
                                end
                            end
                        end
                    end
                    matricula = matricula +
trainingLetters(min_distance_index).Letter;
                else
                    min_distance = -1;
                    min_distance_index = -1;
                    for j=1 : length(trainingDigits)
                        for k=1 : size(trainingDigits(j).Histogram, 1)
                            distance =
recognizerFunctionContainer.distChiSq(hog_30x30,
trainingDigits(j).Histogram(k,:));
                            % al acabar el loop conseguiremos el que tenga
distancia menor,
                            % es decir, el que mas se parezca
                            if min_distance == -1
                                min_distance = distance;
                                min_distance_index = j;
                            else
                                if distance < min_distance
                                    min_distance = distance;
                                    min_distance_index = j;
                                end
                            end
                        end
                    end
                    matricula = matricula +
trainingDigits(min_distance_index).Digit;
                end

            end
        end

        function sortedStats = sortStats(stats)
            T = struct2table(stats);
            sortedTable = sortrows(T, 'BoundingBox');
            sortedStats = table2struct(sortedTable);
        end

        function image = squareAndResizeImage(im, desiredHeight)
            [height, width]= size(im);

            scaleHeight = (desiredHeight/height);
            image = imresize(im, [100, width*scaleHeight]);

            difference = ceil(scaleHeight*(height-width)/2);
            image = padarray(image, [20 20+difference], 0);
            if size(image,1) < size(image,2)
```

```
                image(end+1,:) = 0;
            end

            image = imresize(image, [desiredHeight, desiredHeight]);
        end

        function D = distChiSq( X, Y )
            A = (X-Y).^2; B = X+Y;
            C = A ./ (B+eps);       % + eps para que no de error cuando se
 divide entre 0 y el resultado sea 0
            D = sum(C);
            D = D/2;
        end

    end
end
```

```
ans =

  recognizerFunctionContainer with no properties.
```

Published with MATLAB® R2022a

PROGRAMA PRINCIPAL

```
detector = detectorFunctionContainer;
recognizer = recognizerFunctionContainer;

Letters =
 ["A"; "B"; "E"; "H"; "I"; "K"; "M"; "N"; "O"; "P"; "T"; "X"; "Y"; "Z"];
Digits = ["1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9"; "0"];
% cargamos las plantillas
font = rgb2gray(imread("Greek-License-Plate-Font-2004.svg.png"));
font2 = rgb2gray(imread("Greek-License-Plate-Font-old.jpg"));
font3 = rgb2gray(imread("modelo3.jpg"));
font4 = rgb2gray(imread("modelo4.jpg"));
fonts = cat(3, font, font2, font3, font4);
% entrenamos las estructuras
[trainingDigits, trainingLetters] = recognizer.training(fonts, Digits,
 Letters);

matriculasReales =
 ["ZMZ9157", "YKM2435", "YEP7236", "YYP4245", "YHP2336", "YHE2993", "ZZH8585", "YZP4923",

  "ZKK8153", "ZZN5726", "IEA6907", "IEA5511", "ZME7027", "YKK3431", "IZB2701", "YAZ2074",

  "ZZA9341", "BIZ1100", "YYO8246", "YPP7390", "YNH8511", "IZI2154", "ZHM1169", "IEO8056",

  "ZYY6708", "ZZA9341", "YKK3876", "ZHX1648", "ZYX5517", "ZYZ8289", "NAN1663", "ZZX8178",

  "IEE2359", "II5658", "IEB2949", "IEP1025", "YBE5094", "YKX1115", "ZKT1403", "IZA6106", "

  "YHK9499", "ZKE5495", "ZYB9024", "YOO5657", "ZZH3597", "YZP2401", "ZKB1454", "IZB2701",

  "IZE4030", "IZE4030", "NZB2491", "ZZA7958", "IBM9201", "ZYX7440", "IBT3672"];

matriculasCorrectas = 0;
letrasCorrectas = 0;
letrasTotales = 0;
cont1 = 0;
cont2 = 0;
falsosNegativos = 0;
falsosPositivos = 0;
a = dir('day_color(small sample)\*.jpg');   % cargamos todas las imagenes de
 matriculas
nf = size(a);
%tic
for i=1 : nf
    filename = horzcat(a(i).folder,'/',a(i).name);
    I = imread(filename);
    matricula = "";
    im = rgb2gray(I);
    %figure, imshow(im)
    imagen_prueba = detector.imageSmoothing(im);
    imagen_edges = detector.imageEdges(im);
```

```
    imagen_bin = detector.imageBinarization(im);

    disp("------- Matricula a reconocer: " + matriculasReales(i) + "
-------");

    strat = 1;
    [placa, letras, imagen_crop] = detector.strat1(imagen_edges, imagen_bin);
    if (length(letras) ~= 7)
        strat = 2;
        [placa, letras, imagen_crop] = detector.strat2(imagen_edges,
imagen_bin);
    end
    if (length(letras) == 7)
        %detector.displayBoundingBoxLP(imagen_crop, letras, placa);
        matricula = recognizer.recognize(imagen_crop, letras, trainingDigits,
trainingLetters);
        disp("Estrategia usada: " + strat);
    end

    matricula = convertStringsToChars(matricula);
    matriculaReal = convertStringsToChars(matriculasReales(i));
    if (~isempty(matricula))
        letrasTotales = letrasTotales + length(matriculaReal);
        if (length(matricula) <= length(matriculaReal))
            for j=1 : length(matricula)
                if (matricula(j) == matriculaReal(j))
                    letrasCorrectas = letrasCorrectas + 1;
                end
            end
        else
            for j=1 : length(matriculaReal)
                if (matricula(j) == matriculaReal(j))
                    letrasCorrectas = letrasCorrectas + 1;
                end
            end
        end
    end

    if (strat == 1)
        cont1 = cont1 + 1;
    else
        cont2 = cont2 + 1;
    end

    disp("Matricula reconocida: " + matricula);
    if (matriculasReales(i) == matricula)
        matriculasCorrectas = matriculasCorrectas + 1;
    else
        if (matricula == "")
            falsosNegativos = falsosNegativos + 1;
        else
            falsosPositivos = falsosPositivos + 1;
        end
        disp("Matricula INCORRECTA: " + a(i).name);
```

```
    end

end
%toc
disp("Numero de matriculas correctas = " + num2str(matriculasCorrectas));
disp("Porcentaje de matriculas correctamente reconocidas = " +
 num2str(matriculasCorrectas*100/nf(1)) + " %");
disp("Porcentaje de letras correctamente reconocidas = " +
 num2str(letrasCorrectas*100/letrasTotales) + " %");
disp("Porcentaje de uso estrategia 1 = " + num2str(cont1*100/nf(1)) + " %");
disp("Porcentaje de uso estrategia 2 = " + num2str(cont2*100/nf(1)) + " %");
disp("Falsos negativos = " + falsosNegativos);
disp("Falsos positivos = " + falsosPositivos);

------- Matricula a reconocer: ZMZ9157 -------
Estrategia usada: 1
Matricula reconocida: ZMZ9157
------- Matricula a reconocer: YKM2435 -------
Estrategia usada: 1
Matricula reconocida: YKM2435
------- Matricula a reconocer: YEP7236 -------
Estrategia usada: 1
Matricula reconocida: YEP7236
------- Matricula a reconocer: YYP4245 -------
Matricula reconocida:
Matricula INCORRECTA: DSCN0413.jpg
------- Matricula a reconocer: YHP2336 -------
Estrategia usada: 1
Matricula reconocida: YHP2336
------- Matricula a reconocer: YHE2993 -------
Estrategia usada: 1
Matricula reconocida: YHE2993
------- Matricula a reconocer: ZZH8585 -------
Matricula reconocida:
Matricula INCORRECTA: DSCN0416.jpg
------- Matricula a reconocer: YZP4923 -------
Estrategia usada: 1
Matricula reconocida: YZP4923
------- Matricula a reconocer: ZME8325 -------
Estrategia usada: 1
Matricula reconocida: AMB8545
Matricula INCORRECTA: IMG_0378.jpg
------- Matricula a reconocer: YHO7569 -------
Estrategia usada: 2
Matricula reconocida: YMO7569
Matricula INCORRECTA: IMG_0379.jpg
------- Matricula a reconocer: ZKK8153 -------
Estrategia usada: 1
Matricula reconocida: ZKK8153
------- Matricula a reconocer: ZZN5726 -------
Estrategia usada: 1
Matricula reconocida: ZZN5726
------- Matricula a reconocer: IEA6907 -------
Estrategia usada: 1
```

```
Matricula reconocida: IEA6907
------- Matricula a reconocer: IEA5511 -------
Estrategia usada: 1
Matricula reconocida: IEA5511
------- Matricula a reconocer: ZME7027 -------
Estrategia usada: 1
Matricula reconocida: ZMB7027
Matricula INCORRECTA: IMG_0384.jpg
------- Matricula a reconocer: YKK3431 -------
Estrategia usada: 1
Matricula reconocida: YKK3431
------- Matricula a reconocer: IZB2701 -------
Estrategia usada: 1
Matricula reconocida: IZB2701
------- Matricula a reconocer: YAZ2074 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0387.jpg
------- Matricula a reconocer: YPE2367 -------
Estrategia usada: 1
Matricula reconocida: YPE2367
------- Matricula a reconocer: NE6027 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0389.jpg
------- Matricula a reconocer: ZZA9341 -------
Estrategia usada: 1
Matricula reconocida: ZZA9341
------- Matricula a reconocer: BIZ1100 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0393.jpg
------- Matricula a reconocer: YYO8246 -------
Estrategia usada: 2
Matricula reconocida: YYB8246
Matricula INCORRECTA: IMG_0394.jpg
------- Matricula a reconocer: YPP7390 -------
Estrategia usada: 1
Matricula reconocida: YPP7390
------- Matricula a reconocer: YNH8511 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0396.jpg
------- Matricula a reconocer: IZI2154 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0414.jpg
------- Matricula a reconocer: ZHM1169 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0415.jpg
------- Matricula a reconocer: IEO8056 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0420.jpg
------- Matricula a reconocer: IEN8393 -------
Estrategia usada: 1
Matricula reconocida: IEN8393
------- Matricula a reconocer: ZYM7983 -------
Estrategia usada: 1
Matricula reconocida: ZYM7983
```

```
------- Matricula a reconocer: ZYY6708 -------
Estrategia usada: 1
Matricula reconocida: ZYY6708
------- Matricula a reconocer: ZZA9341 -------
Estrategia usada: 1
Matricula reconocida: ZZA9341
------- Matricula a reconocer: YKK3876 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0448.jpg
------- Matricula a reconocer: ZHX1648 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0449.jpg
------- Matricula a reconocer: ZYX5517 -------
Estrategia usada: 1
Matricula reconocida: ZYX5517
------- Matricula a reconocer: ZYZ8289 -------
Estrategia usada: 1
Matricula reconocida: ZYZ9209
Matricula INCORRECTA: IMG_0452.jpg
------- Matricula a reconocer: NAN1663 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0453.jpg
------- Matricula a reconocer: ZZX8178 -------
Estrategia usada: 1
Matricula reconocida: ZZX8178
------- Matricula a reconocer: ZMZ2317 -------
Estrategia usada: 1
Matricula reconocida: ZMZ2317
------- Matricula a reconocer: ZZY4066 -------
Estrategia usada: 1
Matricula reconocida: ZZY4066
------- Matricula a reconocer: IEE2359 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0457.jpg
------- Matricula a reconocer: II5658 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0460.jpg
------- Matricula a reconocer: IEB2949 -------
Estrategia usada: 1
Matricula reconocida: IEB2949
------- Matricula a reconocer: IEP1025 -------
Estrategia usada: 1
Matricula reconocida: IEP1025
------- Matricula a reconocer: YBE5094 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0463.jpg
------- Matricula a reconocer: YKX1115 -------
Estrategia usada: 2
Matricula reconocida: YKX1115
------- Matricula a reconocer: ZKT1403 -------
Estrategia usada: 1
Matricula reconocida: ZKT1403
------- Matricula a reconocer: IZA6106 -------
Matricula reconocida:
```

```
Matricula INCORRECTA: IMG_0466.jpg
------- Matricula a reconocer: YXN7980 -------
Estrategia usada: 2
Matricula reconocida: YXN7980
------- Matricula a reconocer: ZKH1671 -------
Estrategia usada: 2
Matricula reconocida: ZKX7677
Matricula INCORRECTA: IMG_0468.jpg
------- Matricula a reconocer: YHK9499 -------
Estrategia usada: 2
Matricula reconocida: YNK9499
Matricula INCORRECTA: IMG_0469.jpg
------- Matricula a reconocer: ZKE5495 -------
Estrategia usada: 1
Matricula reconocida: ZKE5495
------- Matricula a reconocer: ZYB9024 -------
Estrategia usada: 2
Matricula reconocida: ZYB9024
------- Matricula a reconocer: YOO5657 -------
Estrategia usada: 1
Matricula reconocida: YOO5657
------- Matricula a reconocer: ZZH3597 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0473.jpg
------- Matricula a reconocer: YZP2401 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0474.jpg
------- Matricula a reconocer: ZKB1454 -------
Estrategia usada: 1
Matricula reconocida: ZBI4541
Matricula INCORRECTA: IMG_0475.jpg
------- Matricula a reconocer: IZB2701 -------
Estrategia usada: 1
Matricula reconocida: IZB2761
Matricula INCORRECTA: IMG_0476.jpg
------- Matricula a reconocer: IBY2254 -------
Estrategia usada: 1
Matricula reconocida: IBY2254
------- Matricula a reconocer: IET2457 -------
Estrategia usada: 1
Matricula reconocida: IET2457
------- Matricula a reconocer: IZE4030 -------
Estrategia usada: 2
Matricula reconocida: IZE4030
------- Matricula a reconocer: IZE4030 -------
Estrategia usada: 1
Matricula reconocida: IZE4030
------- Matricula a reconocer: NZB2491 -------
Estrategia usada: 1
Matricula reconocida: NZB2491
------- Matricula a reconocer: ZZA7958 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0482.jpg
------- Matricula a reconocer: IBM9201 -------
```

```
Estrategia usada: 1
Matricula reconocida: IBM9201
------- Matricula a reconocer: ZYX7440 -------
Estrategia usada: 1
Matricula reconocida: ZYX7440
------- Matricula a reconocer: IBT3672 -------
Matricula reconocida:
Matricula INCORRECTA: IMG_0485.JPG
Numero de matriculas correctas = 38
Porcentaje de matriculas correctamente reconocidas = 56.7164 %
Porcentaje de letras correctamente reconocidas = 93.921 %
Porcentaje de uso estrategia 1 = 58.209 %
Porcentaje de uso estrategia 2 = 41.791 %
Falsos negativos = 20
Falsos positivos = 9
```

Published with MATLAB® R2022a