UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

GRADO EN ING. SIST. DE TELECOM., SONIDO E IMAGEN

"Aplicación para la detección y reconocimiento de matrículas de vehículos"

TRABAJO FINAL DE GRADO

Autor/a:
Pablo Pallarés Font de Mora

Tutor/a:
Jordi Bataller Mascarell
José Ignacio Herranz Herruzo

GANDIA, 2019

Resumen

El propósito de este TFG es la implementación de un código orientado a la detección y reconocimiento de matrículas de automóviles con el sistema de matriculación español. A este tipo de sistemas de reconocimiento se les conoce generalmente como sistemas ANPR (Automatic Number Plate Recognition) o LPR (License Plate Recognition). La aplicación debe ser capaz de reconocer, en primer lugar, los automóviles existentes en la escena; en segundo lugar, la matrícula de cada automóvil y, por último, reconocer los caracteres presentes en la matrícula. La aplicación basada en algoritmos de procesado de Imagen y Visión Artificial es capaz de trabajar tanto en tiempo real (RT), como con imágenes de archivo.

Palabras Clave

Detección, Reconocimiento, Algoritmo, Procesado de Imagen, Visión Artificial.

Abstract

The purpose of this TFG is the implementation of code aiming to detect and recognize car license plates according to the Spanish registration system. These types of recognition systems are generally referred to as ANPR (Automatic Number Plate Recognition) or LPR (License Plate Recognition) systems. The application must be able to recognize, firstly, the existing cars in the scene; secondly, the license plate of each car and finally, to identify the characters present in the license plate. The application, based on image processing algorithms and Artificial Vision, can work both in real-time (RT), and also with archive images.

Keywords

Detection, Recognition, Algorithm, Image Processing, Computer Vision.

Índice

Tabla de ilustraciones

1. Introducción

1.1. Presentación

La Inteligencia Artificial está cada vez más presente en nuestro día a día, tanto nosotros como las grandes empresas o los gobiernos dependemos cada vez más de esta tecnología [1]. Pese a que la Inteligencia Artificial abarca muchos más campos, en el caso concreto de la Visión Artificial el objetivo principal que se persigue es adquirir, procesar y analizar imágenes del mundo real de forma que se genere una información interpretable por una máquina. Estos sistemas han tenido una gran expansión en el sector industrial en los últimos tiempos. Una de las aplicaciones más extendidas son los sistemas de reconocimiento automático de matrículas.

Los sistemas de reconocimiento automático de matrículas (ANPR, por sus siglas en inglés Automatic Number Plate Recognition) nacen de la conveniencia de digitalizar elementos con identificación numérica y procesar de forma rápida y eficaz el gran volumen de datos que así se genera. El área de la Inteligencia Artificial que estudia y diseña este tipo de sistemas es la de visión artificial, juntamente con reconocimiento de patrones y redes neuronales.

El ANPR tiene numerosas aplicaciones destinadas principalmente a agilizar el manejo de información y reducir la cantidad de recursos humanos necesarios en situaciones, como la recaudación electrónica de peaje en autopistas de pago, control de entradas en los parkings, sanciones electrónicas por mal estacionamiento, radares, etc. Estos sistemas son capaces de escanear las matrículas en vehículos con velocidades de hasta 160 km/h y, además, suelen utilizar iluminación infrarroja para poder tomar fotografías en cualquier momento del día.

El proceso que llevan a cabo estos sistemas se divide en 3 fases:

1) Identificación del vehículo (Automatic vehicle identification, AVI)

2) Localización de la placa (Car plate recognition, CPR)

3) Reconcimiento del número de matrícula (License plate recognition, LPR)

La tecnología ANPR hace uso también del reconocimiento óptico de caracteres OCR, una vez aislada y procesada la matrícula. Es específica para una región, debido a la variación entre matrículas de un lugar a otro. Algunos sistemas más complejos pueden diferenciar variantes internacionales, aunque la mayoría de programas se adaptan a cada país individualmente [2]. En general, una característica importante de estos sistemas debe ser la adaptabilidad a las condiciones cambiantes del entorno de uso, ya que no se puede esperar que una placa de matrícula esté en perfectas condiciones de iluminación, posición, etc. para ser identificada.

En este proyecto se pretende desarrollar un sistema APNR para una región específica, en concreto para el estado español. Se desarrollará un software capaz de reconocer automáticamente matrículas a partir de imágenes fijas, que el propio sistema es capaz de extraer de un fragmento de vídeo en tiempo real. Para ello se utiliza la herramienta MATLAB, haciendo uso de las diferentes funciones relacionadas con el procesado de imágenes en cualquiera de sus fases, ya sea en el preprocesado, segmentación, definición de descriptores, etc.

Para comprobar la efectividad del sistema se llegado incluso al montaje de una maqueta que simula la entrada a un parking, mediante el uso de una webcam, coches en miniatura y matrículas a su escala.

1.2. Motivación y objetivos

Este proyecto tiene como motivación principal, establecer un entorno de trabajo que permita comprender y estudiar el comportamiento, tanto teórico como práctico, de los sistemas ANPR. A continuación, se muestran los objetivos pretendidos en este proyecto:

- Estudiar de los principios de reconocimiento de patrones, y los sistemas donde se aplica.
- Estudiar el funcionamiento de los sistemas APNR, tanto Software como Hardware.
- Estudiar las diferentes técnicas de detección de coches para aislarlos en una imagen digital.
- Conocer de la estructura y características de una matrícula.
- Buscar y estudiar las necesidades de un OCR (Optical Character Recognition).
- Desarrollar y exponer todos los procesos presentes en el sistema APNR para matrículas del Estado español a desarrollar.
- Realizar un diseño de la interfaz de usuario de la aplicación.
- Diseñar la arquitectura interna de la aplicación, los métodos y algoritmos que vamos a utilizar.
- Elaborar una interfaz de usuario gráfica con la herramienta GUI de Matlab.
- Implementar un código eficiente y robusto siguiendo la arquitectura generada para justificar los principios teóricos desarrollados.
- Realizar una fase de pruebas para comprobar los resultados.
- Comprobar los resultados para obtener conclusiones de los algoritmos diseñados y proponer posibles mejoras para corregir los problemas detectados.

1.3. Metodología

El trabajo se organizado en 2 grandes bloques. Primero, el reconocimiento de matrículas a partir de una imagen fija. Y, en segundo lugar, la extracción de imágenes a partir de un fragmento de video y el reconocimiento de las matrículas, todo ello en tiempo real.

La metodología usada como forma de trabajo se basa en recopilar información, diseñar la estructura de la aplicación, implementar dicha estructura y, finalmente, comprobar el correcto funcionamiento.

El primer paso fue obtener información procedente de artículos científicos, temario de asignaturas relacionadas con la inteligencia artificial encontrados en la red, etc. para determinar que función desempeña la visión artificial en los sistemas de reconocimiento automático de matrículas, y cómo se aplica.

Después de recabar esta información se establecieron unos objetivos alcanzables a través de este proyecto. De esta forma se hizo un primer boceto del entorno gráfico, y

se diseñaron una serie funciones y algoritmos para el reconocimiento de matriculas con imágenes fijas.

Tras implementar estos algoritmos y obtener unos resultados satisfactorios, se pasa a diseñar un algoritmo capaz de reconocer imágenes en tiempo real. Al realizar la comprobación de los algoritmos en tiempo real se observó que el software devolvía la misma matrícula repetida, y se decidió mejorar el código añadiendo un filtrado para evitar matrículas repetidas.

Por último, se implementó el diseño de la interfaz gráfica de usuario diseñada para facilitar el uso de la aplicación al usuario. En este entorno gráfico se puede ver cómo funciona el algoritmo mientras se observa el proceso que sigue la imagen.

1.4. Organización la memoria

La estructura de esta memoria es la siguiente.

- **Capítulo 1: Introducción.** En este capítulo se realiza una presentación del proyecto, así como las motivaciones que han llevado ha desarrollar este trabajo y los objetivos a cumplir. También se detalla la estructura que sigue el documento y la metodología empleada.

- **Capítulo 2: Análisis de requerimientos.** Se desarrollan más extensamente los objetivos fijados y se detalla la planificación que se va a seguir con hitos concretos para lograr estos objetivos.

- **Capítulo 3: Diseño de la aplicación.** Durante este capítulo se muestra, por una parte, el entorno gráfico a crear a través de bocetos y, por otra parte, el diseño y la arquitectura interna de la aplicación, las funciones a crear y utilizar y los principales algoritmos. De esta forma se exponen todas las partes y procesos que forman parte del software del sistema APNR.

- **Capítulo 4: Implementación.** Se realiza la parte práctica para justificar los principios teóricos y los algoritmos expuestos. Por otra parte, se concretan todas las herramientas utilizadas en el proyecto.

- **Capítulo 5: Descripción del entorno gráfico.** Se vuelve a describir el entorno gráfico, en esta ocasión ya definitivo, a través de una guía de usuario. Se hace uso de capturas para explicar las posibilidades que ofrece el programa y comprobar su funcionamiento con ejemplos.

- **Capítulo 6: Análisis de resultados.** Incluye el análisis de los resultados finales. Se obtienen conclusiones respecto al código diseñado, y se presentan los principales problemas encontrados.

- **Capitulo 7: Posibles mejoras.** Se plantean ideas con el fin de mejorarlo, además de nuevas e interesantes posibilidades que podrán incorporarse en un futuro.

- **Capítulo 8: Conclusión.** Se exponen las conclusiones alcanzadas realizar el proyecto. También se hace un breve repaso de todos los puntos vistos en la memoria de forma objetiva y concisa.

- **Bibliografía.** Detalla las fuentes de información utilizadas durante la elaboración del proyecto.

- **Declaración de trabajo original.** Se reconoce el trabajo como propio y exento de copia a terceros.

1.5. Principales problemas

Los inconvenientes encontrados en el proyecto vienen dados por la suposición de unas condiciones ideales que no se dan en la realidad. Algunos de los problemas que se han encontrado durante las pruebas han sido:

- Imágenes con mala iluminación.
- Elementos en la imagen con características morfológicas similares a las matrículas.
- Repetición de una matrícula.
- Caracteres indeseados en el OCR.

Durante los siguientes capítulos, principalmente en el capítulo 6, se presenta la solución a algunos de estos problemas para llegar a un algoritmo robusto, y en el capitulo 7 se proponen soluciones para los problemas que han quedado sin resolver.

2. Análisis de requerimientos del programa y desarrollo de objetivos

Para la correcta elaboración de la aplicación se establecen una serie de requerimientos necesarios. En primer lugar, el programa debe resultar un sistema APNR para el reconocimiento de matrículas con el sistema de matriculación español. El código debe de trabajar a gran velocidad poder para obtener resultados en tiempo real, y ser lo más robusto posible. Debe mostrar las diferentes fases del procesado de imagen llevado a cabo para obtener la matricula, de esta forma se puede visualizar que procesos sigue y donde falla, en el caso remoto de que falle. Por último, debe de tener una interfaz gráfica sencilla e intuitiva para el usuario, de este modo se garantiza el correcto uso de la aplicación.

Para lograr este fin se plantean una serie de objetivos e hitos concretos a seguir:

1) **Estudiar de los principios de reconocimiento de patrones, y los sistemas donde se aplica.** Durante esta tarea se realiza un trabajo de investigación. Hitos concretos:

   a) Búsqueda de información relacionada con el reconocimiento de patrones.
   b) Selección de los algoritmos y técnicas de reconocimiento de patrones más apropiados para la aplicación perseguida.
   c) Búsqueda de información sobre los sistemas donde se aplica esta técnica, para determinar que tipo de sistema es el apropiado para el reconocimiento de matrículas.

2) **Estudiar el funcionamiento de los sistemas APNR, tanto Software como Hardware.** Continuando con el trabajo de investigación, pero centrado en el sistema concreto a desarrollar. Hitos concretos:

   a) Investigación a cerca de los principios básicos de los sistemas APNR.
   b) Listado del material y herramientas necesarios para desarrollar un sistema APNR.
   c) Estudio de la viabilidad del proyecto. Una vez conocido el material y herramientas necesarios se divide el proyecto en dos partes: software y Hardware. Inicialmente se centra el foco en desarrollar un software capaz de realizar un reconocimiento automático de matriculas para imágenes estáticas haciendo uso de la herramienta Matlab.

3) **Estudiar las diferentes técnicas de detección de coches para aislarlos en una imagen digital.** Continuando con el trabajo de investigación, se busca la forma de aislar el coche para empezar a trabajar sobre la imagen del coche aislado. Hitos concretos:

   a) Búsqueda de herramientas de Matlab que puedan servir para este fin.
   b) Una vez conocidas las herramientas de Matlab posibles, probar las distintas posibilidades para elegir la mejor opción.
   c) Determinar si hay que realizar algún preprocesado de imagen para introducir la imagen preprocesada a la herramienta elegida.

4) **Conocer de la estructura y características de una matrícula.** Una vez tenemos el coche aislado, el siguiente objetivo es asilar la matricula para poder tratarla de

forma individual y proceder al reconocimiento de los caracteres internos. Hitos concretos:

a) Búsqueda de las características principales de una matrícula con el sistema de matriculación español en el BOE.
b) Elaborar una lista con las medidas y propiedades de la placa
c) Elaborar una lista de características de los caracteres internos: diseño, orden, etc.

5) **Buscar y estudiar las necesidades de un OCR (Optical Character Recognition).** Con el fin de convertir la imagen digital en datos útiles debemos conocer las necesidades de este proceso. Hitos concretos:

a) Establecer que tipo de imagen recibe el OCR.
b) Determinar con diferentes pruebas si la función de Matlab es válida o, por el contrario, hay que generar un nuevo OCR
c) En el caso de que sea válida, leer la documentación a cerca de su funcionamiento para sacarle al mayor provecho posible.

6) **Desarrollar y exponer todos los procesos presentes en el sistema APNR para matrículas del Estado español a desarrollar.** Esta tarea se basa en recoger e interconectar todos los procesos por los que debe pasar la imagen digital para reconocer los caracteres internos de una matrícula. Hitos concretos:

a) Recoger en una lista todos los procesos descritos en las tareas anteriores para, manteniendo el orden, interconectarlos.
b) Determinar el preprocesado de imagen necesario en cada proceso.

7) **Realizar un diseño (Boceto) de la interfaz de usuario de la aplicación.** El objetivo es diseñar un entorno gráfico con botones, imágenes y texto, de forma que se introduzca una imagen y tras el proceso obtengamos los caracteres de la matrícula en forma de texto en alguna parte de nuestro entorno. Hitos concretos:

a) Teniendo en cuenta las características que debe tener nuestra aplicación, dibujar como se distribuyen los diferentes elementos en nuestro entorno gráfico

8) **Diseñar la arquitectura interna de la aplicación, los métodos y algoritmos que vamos a utilizar.** Con la lista de procesos elaborada y las interconexiones entre ellos realizada, se procede a hacer un diseño de la estructura interna de la aplicación. Hitos concretos:

a) Elaborar un esquema de la estructura interna de la aplicación con las relaciones entre procesos y métodos establecidas
b) Diseño de cada método utilizado. ¿Qué recibe y qué devuelve? Ejemplos: recibe una imagen en color y devuelve una en blanco y negro, recibe una imagen y devuelve una secuencia de caracteres, etc.
c) Elaboración de los algoritmos internos de cada método, y funciones generales.

9) **Elaborar una interfaz de usuario gráfica con la herramienta GUI de Matlab.** Elaborar de forma gráfica, sin entrar en la programación interna, una interfaz de usuario gráfica basada en el diseño elaborado en la tarea 7. Hitos concretos:

a) Leer la documentación a cerca de la GUI de Matlab para conocer el funcionamiento de la herramienta.

b) Diseñar gráficamente la interfaz de usuario imitando el boceto de la tarea 7.

10) **Implementar un código eficiente y robusto para justificar los principios teóricos desarrollados.** Haciendo uso de los esquemas de la estructura interna y los algoritmos de los métodos generamos un código robusto. Hitos concretos:

a) Generar un código para los métodos específicos en Matlab, y comprobar su funcionamiento.
b) Generar un código para las funciones generales llamadas por la GUI de Matlab (interfaz de usuario gráfica) generada. Estas funciones generales harán uso de los métodos específicos y, a su vez, serán llamados por los elementos de la GUI.
c) Programar las funciones de los elementos: textos, imágenes, botones, etc. utilizados en la GUI.

11) **Realizar una fase de pruebas para comprobar los resultados.** En esta fase se probará el código implementado ya con la interfaz gráfica definitiva. Hitos concretos:

a) Fase de pruebas del código con imágenes estáticas, y solucionar los problemas detectados
b) Fase de pruebas del código con la cámara e imágenes en tiempo real (RT).

12) **Comprobar los resultados para obtener conclusiones de los algoritmos diseñados y proponer posibles mejoras para corregir los problemas detectados.** Con el proyecto finalizado y el código funcionando correctamente se realiza un análisis de los resultados obtenidos en las pruebas y se extraen conclusiones a cerca de las posibles mejoras a implementar.

3. Diseño de la aplicación

3.1. Diseño de la interfaz gráfica de usuario

Para diseñar la arquitectura interna de la aplicación y determinar los métodos necesarios es imprescindible realizar un boceto inicial del entorno gráfico. Se pretende crear una interfaz de usuario con: dos botones, un visor de imágenes, un texto fijo, y un texto editable.

- **Botón 1:** Tendría la función de abrir el explorador de archivos para cargar una imagen que contenga un coche con matriculación española. Una vez cargada la imagen se inicia el proceso de reconocimiento automático.

- **Botón 2:** Su función sería activar la cámara para iniciar el proceso en tiempo real.

- **Visor de imágenes:** En este cuadro se debería visualizar la imagen cargada desde archivo en el caso de pulsar el botón 1, y en el caso de pulsar el botón 2, se retransmitirían imágenes capturadas por la cámara en tiempo real.

- **Texto estático:** Texto fijo donde se viese escrito “Matrículas encontradas:” para, a continuación, escribir las matriculas detectadas.

- **Texto dinámico:** Este Texto debería estar inicialmente en blanco, y tras iniciar el proceso con la imagen estática o en tiempo real aparecería la/s matricula/s obtenida/s.

Ilustración 1: Boceto Interfaz Gráfica de Usuario (GUI)

3.2. Arquitectura y diseño de funciones

El diseño de esta aplicación para el reconocimiento automático de matriculas se basa en los siguientes algoritmos:

1. **Localización de los vehículos:** Se encarga de encontrar y aislar los vehículos de la imagen general.

2. **Binarización y Segmentación de la imagen:** Binariza la imagen encuentra los distintos objetos presentes en la imagen del coche.

3. **Filtrado de objetos y localización de la matrícula:** Encargado filtrar los elementos encontrados en la imagen haciendo uso de las propiedades de las matriculas con el fin de encontrar y aislar la matrícula en la/s imagen/imágenes de el/los vehículo/s.

4. **Morfología Binaria**: Encargada de realizar operaciones binarias sobre la imagen de la matrícula aislada y binarizada para mejorarla, con la intención de facilitar el cálculo de descriptores, y la labor del OCR.

5. **Reconocimiento óptico de caracteres (OCR)**.

6. **Análisis sintáctico y geométrico:** comprueba que los caracteres encontrados y sus posiciones coincidan con las reglas específicas del BOE para matriculas españolas.

La complejidad de cada una de estas partes del programa determina la exactitud del sistema. Como ya se adelantaba al final del capitulo 1.2. “Motivación y objetivos”, el proyecto se divide en 2 grandes bloques: reconocimiento de matrículas a partir de una imagen almacenada en memoria, y reconocimiento de matrículas en tiempo real con la cámara USB. El proceso debe ser prácticamente idéntico, quizá la diferencia más evidente es la fuente de la imagen. Por lo que se pretende crear una aplicación que, una vez se haga click sobre cualquiera de los dos botones, las funciones y algoritmos utilizados sean idénticos.

Para elaborar la arquitectura interna se definen las dos funciones principales: detectorMatriculasFoto() y detectorMatriculasFrame(), cuya llamada se realizará al pulsar los botones de la GUI: “Cargar Imagen” y “Encender Cámara” respectivamente. Estas dos funciones serán las encargadas de llevar a cabo todo el proceso desde que entra la imagen, o las capturas del video en tiempo real hasta que se obtienen la matricula o lista de matriculas en forma de texto.

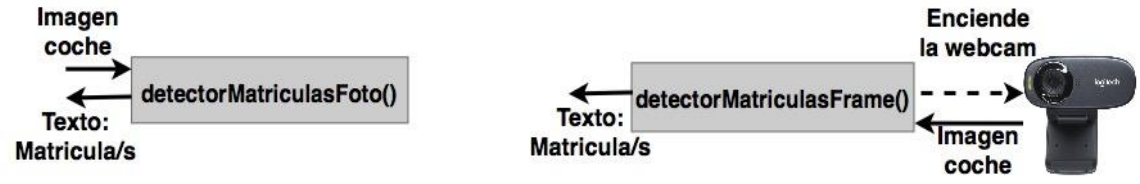


Ilustración 2: Arquitectura de las funciones principales

Estas funciones principales, a su vez, harán uso de otras funciones de menor tamaño con la intención de no acumular toda la carga computacional en una función, y facilitar

la tarea de depuración. Algunas de estas subfunciones serán tomadas de la librería de Matlab y otras serán de creación propia y específica para el proyecto. Véase las más relevantes:

- **vehicleDetectorACF():** Esta función forma parte de la biblioteca de MATLAB que devuelve un detector de vehículos pre-entrenado usando imágenes de los lados delantero, trasero, izquierdo y derecho de los vehículos. No requiere parámetro de entrada.

Ilustración 3: Arquitectura de la función vehicleDetectorACF

También es posible pasarle el modelo de vistas con las que ha sido entrenado el detector, es decir, puedo seleccionar si el detector ha sido preentrenado con imágenes de coches desde todas las perspectivas ***'full-view'*** o únicamente con imágenes de la cara delantera y trasera de los vehículos ***'front-rear-view'*** [3].

- **detect():** Esta función, también de la biblioteca de Matlab, recibe como parámetro de entrada un detector como el que devuelve la función anterior. De esta función se obtiene la posición del posible vehículo (parámetro: "bboxes") y la puntuación que esta función le da, es decir, cuanta más puntuación tenga el objeto detectado más probabilidades tiene de ser un vehículo (parámetro: "scores"). [4]

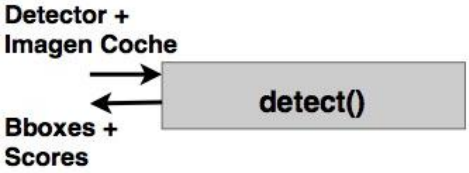


Ilustración 4: Arquitectura de la función detect

- **colorThreshold():** El objetivo de esta función es binarizar la imagen que recibe como parámetro de entrada, es decir, recibe una imagen en color y devuelve una en blanco y negro.

Ilustración 5:Arquitectura de la función colorthreshold

- **reconoce():** Esta función recibe una imagen con la posible matrícula recortada y devuelve como parámetro de salida la matrícula en formato de texto como una cadena de caracteres.

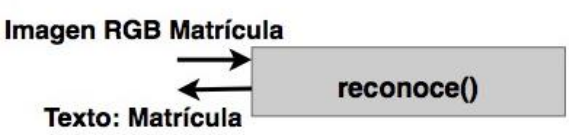


Ilustración 6: Arquitectura de la función reconoce

El algoritmo diseñado para esta función es el siguiente:

1) Transforma la imagen de la posible matrícula en color a escala de grises.

2) Se transforma la imagen en escala de grises a blanco y negro con una umbralización automática.

3) Se realizan una serie de operaciones binarias para facilitar la labor del OCR (Morfología Binaria).

4) OCR para obtener los caracteres de la matrícula.

5) Análisis sintáctico y geométrico para garantizar que los caracteres encontrados y sus posiciones sean las correctas. Este proceso se realizará a través de otra subfunción: *analizaMatricula().*

- **analizaMatricula():** Esta función analiza las posibles matrículas detectadas ya en formato de texto y en el caso de que efectivamente contenga letras y números, y no se trate de un reflejo con características morfológicas similares a una matricula, las ordena siguiendo el formato de matriculación español:

  4 números + espacio + 3 letras mayúsculas, ejemplo: 8996 JMC

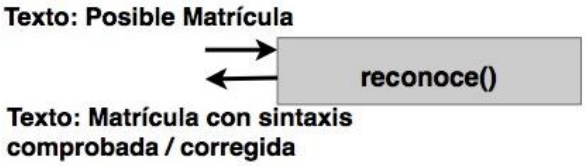


Ilustración 7: Arquitectura de la función analizaMatricula

Una vez definidas las subfunciones pensadas para facilitar la labor a las funciones principales se procede a describir el algoritmo de las funciones principales:

1) Llamada a la función *vehicleDetectorACF()* para obtener un detector preentrenado.

2) Llamada a la función *detect()* utilizando el detector obtenido y la imagen en color de un coche. La función *detect()* devuelve una matriz con la posición de los posibles vehículos y una matriz con las puntuaciones recibidas.

3) Se crea una matriz de imágenes de los posibles vehículos recortados de la imagen principal. Solo se incluyen en esta matriz aquellos posibles vehículos que superen una puntuación determinada (el umbral para decidir la puntuación mínima a partir de numerosas pruebas con imágenes).

4) Se binarizan las imágenes de los posibles vehículos aislados, y se crea una nueva matriz con imágenes en blanco y negro.

5) Se realiza una búsqueda de objetos en las imágenes binarizadas. Estos objetos son sometidos a una serie de filtros: orientación, relación de aspecto, etc. para aislar las posibles matriculas y almacenarlas en una nueva matriz.

6) Cada una de las imágenes de las posibles matriculas se pasa por la función *reconoce()* para ser procesada y obtener los caracteres de la matricula a través del OCR.

7) Se analiza que no se repitan matrículas entre las encontradas y se devuelve una lista con todas las matriculas encontradas en la imagen sin repeticiones, es decir, en el caso de que se encuentre dos veces la misma matrícula una de ellas se eliminará de la lista.

En el caso de la función *detectorMatriculasFrame()* el algoritmo es idéntico. Únicamente requiere añadir un paso anterior para encender la cámara USB, y el proceso se realizará en varios frames. Gracias al filtrado mencionado en el paso 7 se evitará obtener la misma matricula en cada frame en el que aparezca el mismo vehículo.

4. Implementación

Tras plantear toda la estructura interna y los algoritmos de las funciones que compondrán el código final se procede a su programación a través de scripts en la herramienta Matlab. Para ello se han consultado los manuales **"getstart**" y **"matlab_oop"** que se adjuntan en la carpeta de **Documentos_Consultados** contenida en la carpeta de **Ficheros Anexos**. A continuación, se muestran los principales problemas encontrados a la hora de implementar los diseños realizados, y la solución planteada para cada uno.

En primer lugar, véase la implementación de las subfunciones:

1) En el caso de las funciones *vehicleDetectorACF()* y *detect()* al ser funciones de la biblioteca de Matlab no hay nada que implementar. Únicamente hay que buscar en la documentación de MathWorks como realizar la llamada a dichas funciones.

2) La función *colorThreshold()* se crea haciendo uso de la aplicación de MATLAB llamada "ColorThresholder", mediante la cual, definimos una serie de filtros para pasar la imagen a escala de grises y umbralizar de forma que queden distinguidas las partes de la imagen con los bits más cercanos al color blanco, ya que todas las matrículas en los coches son de dicho color. Véase el proceso:

    a. En primer lugar, se clica sobre la opción de cargar imagen y se selecciona una imagen de archivo para comenzar con el proceso:

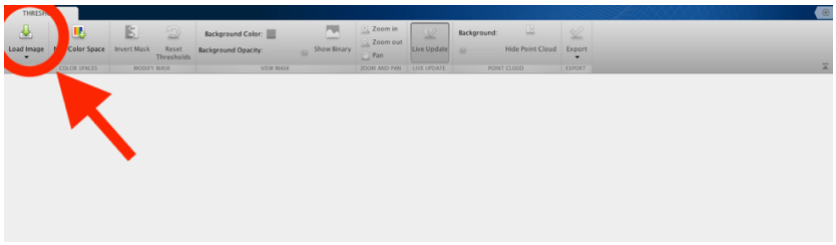


Ilustración 8: Tutorial app colorThresholder. Cargar imagen

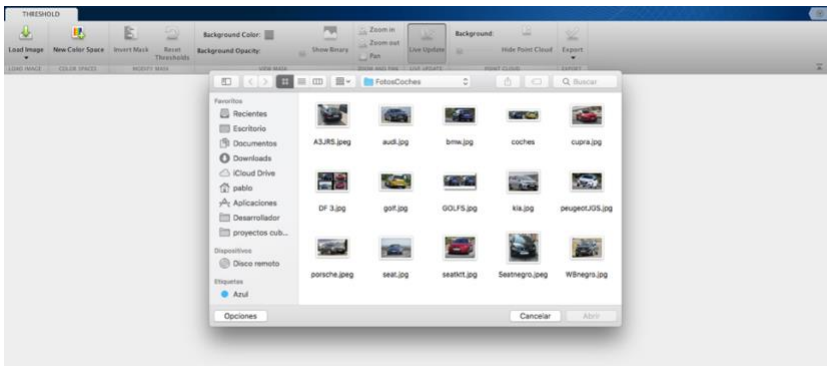


Ilustración 9: Tutorial app colorThresholder. Seleccionar imagen

    b. En segundo lugar, se selecciona el modelo de color para realizar el filtrado. En este caso se selecciona el modelo HSV ya que nos interesa filtrar principalmente la saturación para aislar los blancos.

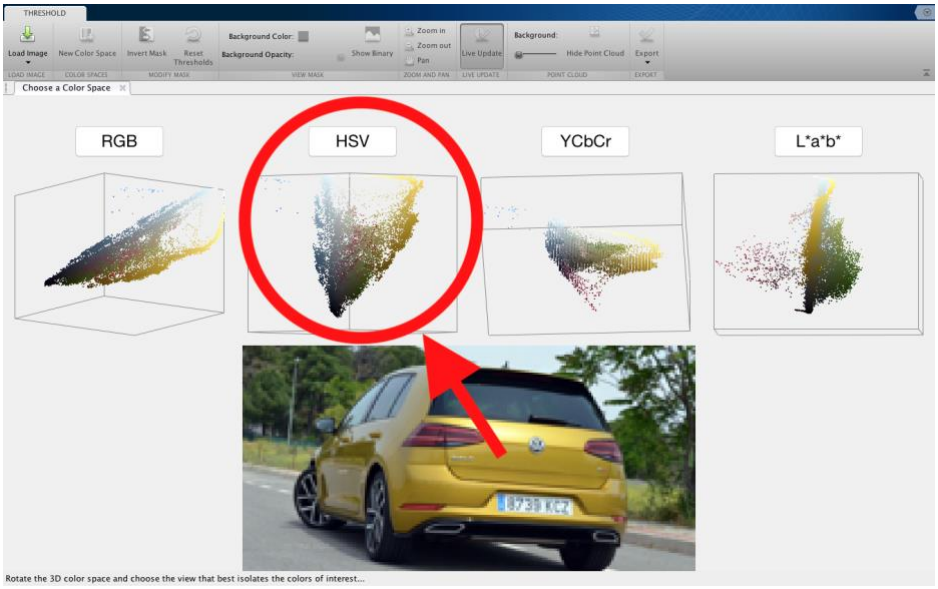


Ilustración 10: Tutorial app colorThresholder. Seleccionar modelo de color

c. Y por ultimo se filtran las 3 componentes principalmente la Saturación para aislar los pixeles con valores cercanos al blanco:

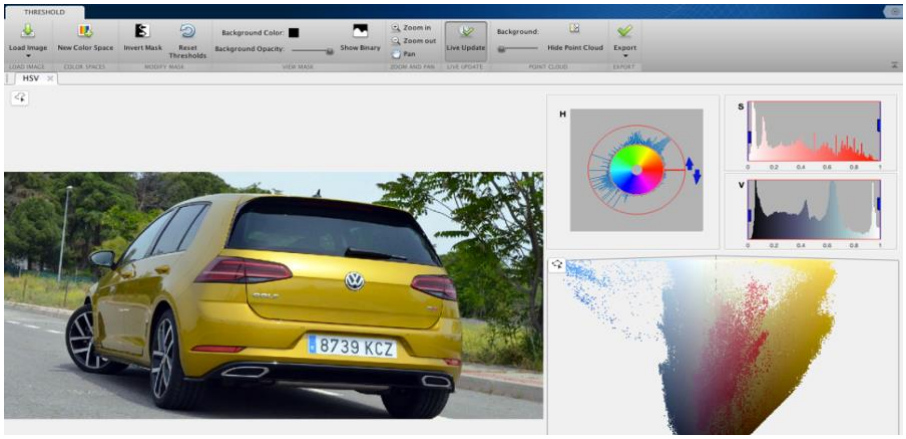


Ilustración 11: Tutorial app colorThresholder. Imagen prefiltrado

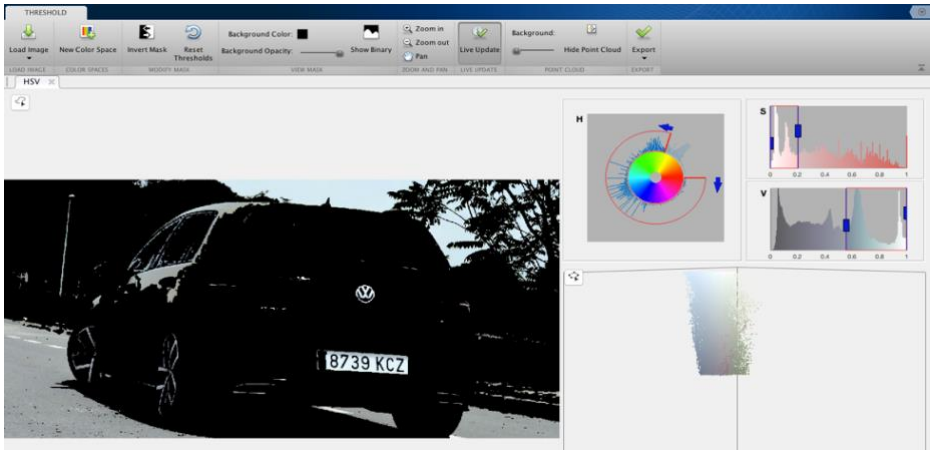


Ilustración 12: Tutorial app colorThresholder. Imagen postfiltrado

Este proceso se repite con diferentes imágenes para estimar unos valores medios para los filtros de las componentes HSV.

d. Una vez se estiman unos valores medios se exporta la función y automáticamente la aplicación crea el código y podemos guardarlo en la carpeta deseada folder:

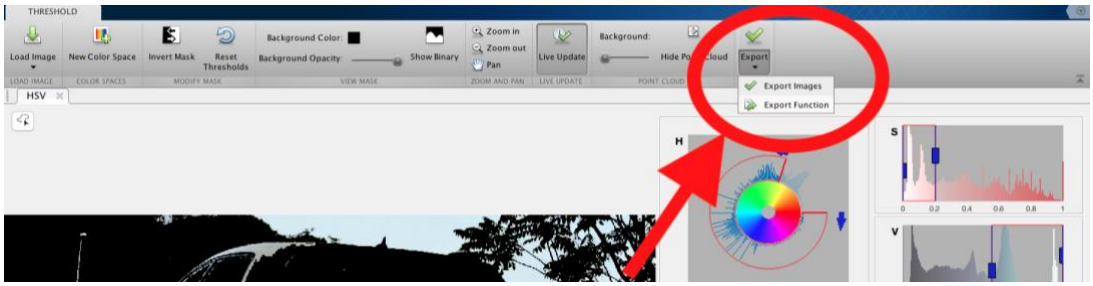


Ilustración 13: Tutorial app colorThresholder. Exportar función

3) Para la función *reconoce()* ha habido que jugar con las distintas operaciones binarias, experimentando con imágenes de matriculas aisladas probando con distintas combinaciones de operaciones . El código final es el siguiente:

```
function [MatriculaTxt] = reconoce(filename)
      %%Llega una imagen de la posible matrícula en color, y devuelve
      una matriz de caracteres que contiene el texto de la matrícula.
```

Ilustración 14: Imagen RGB de la matricula que recibe la función

```
      %Pasa la imagen en color a escala de grises automáticamente con la
      función de biblioteca de Matlab rgb2gray() y umbraliza también
      automáticamente con la función de biblioteca imbinarize() y
      obtiene una imagen binaria.
```

```
A=imbinarize(rgb2gray(filename));
```

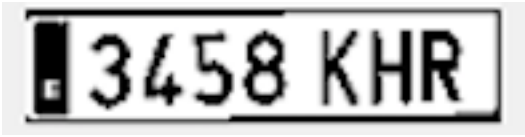


Ilustración 15: Imagen binarizada

```
%un fill para rellenar huecos de la imagen, es decir, los grupos
de pixeles aislados.
B=imfill(A,'holes');
```

Ilustración 16: Imagen con la operación binaria fill

```
%operación binaria XOR entre la imagen binarizada y con el fill
para eliminar los bordes. Se obtiene la imagen complementaria para
que los números queden en negro y la placa en blanco. Ya que de
la imagen anterior obteníamos el resultado contrario.

D=xor(A,B);
D=imcomplement(D);
```

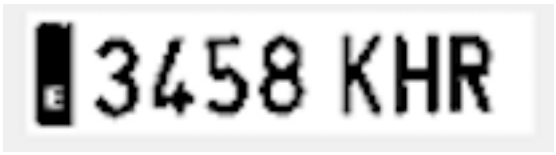


Ilustración 17: Imagen definitiva para el OCR

```
%La siguiente parte de código condiciona el OCR para que en la
primera parte de la matrícula (concretamente el 60% del eje mayor)
solo detecte números y la E de Europa inicial, y en la segunda
parte de la matricula solo detecte consonantes.

propiedad=regionprops(A,'MajorAxisLength','MinorAxisLength','Bou
ndingBox');
bounding=propiedad.BoundingBox;
x=bounding(1);
y=bounding (2);
hsize=bounding(3);
vsize=bounding(4);
[alto,ancho]=size(D);



w1=hsize*0.6;
w2 = hsize - w1;

roi= [x y w1 vsize];
```

```
        OCR1 = ocr(D,roi,'CharacterSet','E0123456789');

        roi = [w1 y w2 vsize];
        OCR2 = ocr(D,roi,'CharacterSet','BCDFGHJKLMNPQRSTVWXYZ');

        %a continuación unifica la cadena de números con la de letras en
        una única cadena de caracteres.

        MatriculaTxt = strcat(OCR1.Text,OCR2.Text);

        %Se pasa la matrícula por analizaMatricula() para realizar un
        análisis sintáctico siguiendo la normativa definida por el BOE.

        MatriculaTxt = analizaMatricula(MatriculaTxt);
End
```

4) En el caso de la función *analizaMatricula()* recibe la matricula obtenida por el OCR y realiza un proceso para ajustar el formato al recogido por el BOE, tal y como se desprende a continuación:

```
function [MatriculaFinal] = analizaMatricula(matriculaTxt)

        % Esta función analiza las posibles matrículas detectadas y en el
        caso de que efectivamente contenga letras y números las ordena en
        función al  formato de matrÌcula de coche español 4 números +
        espacio + 3 letras mayúsculas, ej: 8996 JMC

        % Se crea la variable MatriculaFinal para que en el caso de no
        tratarse de una matricula la función devuelva este parámetro
        vacío.

        MatriculaFinal=' ';

        for a=1:1:length(matriculaTxt)

        % Si el carácter leído es un número o una letra forma parte de la
        matrícula. Esto se hace para revisar que no se cuelen signos
        indeseados, para reforzar el filtrado hecho en reconoce()
              If(matriculaTxt(a)=='0')||(matriculaTxt(a)=='1')||(matricul
              aTxt(a)=='2')||(matriculaTxt(a)=='3')||(matriculaTxt(a)=='4
              ')||(matriculaTxt(a)=='5')||(matriculaTxt(a)=='6')||(matric
              ulaTxt(a)=='7')||(matriculaTxt(a)=='8')||(matriculaTxt(a)==
              '9')||(matriculaTxt(a)=='B')||(matriculaTxt(a)=='C')||(matr
              iculaTxt(a)=='D')||(matriculaTxt(a)=='F')||(matriculaTxt(a)
              =='G')||(matriculaTxt(a)=='H')||(matriculaTxt(a)=='J')||(ma
              triculaTxt(a)=='K')||(matriculaTxt(a)=='L')||(matriculaTxt(
              a)=='M')||(matriculaTxt(a)=='N')||(matriculaTxt(a)=='P')||(
              matriculaTxt(a)=='Q')||(matriculaTxt(a)=='R')||(matriculaTx
              t(a)=='S')||(matriculaTxt(a)=='T')||(matriculaTxt(a)=='V')|
              |(matriculaTxt(a)=='W')||(matriculaTxt(a)=='X')||(matricula
              Txt(a)=='Y')||(matriculaTxt(a)=='Z')


                 MatriculaFinal=strcat(MatriculaFinal, matriculaTxt(a));
              end
        end%%for
```

```
        %En el caso de que efectivamente sea una matricula con 7 caracteres
        4 números + 3 letras se introduce un espaciamiento entre las letra
        y números para cuadrarlo con el formato de matriculación español

        If (length(MatriculaFinal)>6)

              for b=length(MatriculaFinal):-1:5

                    MatriculaFinal(b+1) = MatriculaFinal(b);
              end%for b

              MatriculaFinal(5)= ' ';

        end %%if
end %function
```

4.1. Reconocimiento de matrículas a partir de una imagen

Esta función ha recibido el nombre de **detectorMatriculasFoto()**, pese a que en ella se implementan más fases del procesado de imagen como la segmentación y el reconocimiento. Como ya adelantábamos hace uso de distintas subfunciones ya descritas: vehicleDetectorACF(), detect(), reconoce(), etc.

A continuación, se analiza paso a paso el código implementado en este primer bloque:

- Se inican las variables de los contadores utilizadas mas adelante: **count1, count2**, y **count3**. También se inicializa la variable **MatriculaTxtFinal** que será la variable que se ira modificando para ser la resultante de la función general. Por último, se inicializan las siguientes variables de las cuales más adelante se mostrará su origen y su función: **UMBRAL_PUNTUACION_COCHE, COMPACIDAD_MEDIDAS_BOE, RELACION_ASPECTO_MEDIDAS_BOE, UMBRAL_CARACTERES_MINIMOS, y existe**.

- Se utilizan las funciones de **"vehicleDetectorACF"** y **"detect"** para reconocer los posibles vehículos de la imagen leída y obtener su ubicación en la imagen y la probabilidad de que efectivamente se trate de un vehículo. Se incluyen sobre la imagen original una serie de rectángulos que encierran los posibles vehículos con la puntuación otorgada:

Ilustración 18: Detección de los posibles vehículos

- Se programa un bucle para recorrer todos los posibles vehículos detectados y en el caso de que las probabilidades de tratarse de un vehículo superen un umbral preestablecido en 15% en la variable inicializada al principio del código **"UMBRAL_PUNTUACION_COCHE" …**

- Se recorta la imagen original con el comando "**imcrop**" [5] de la biblioteca de Matlab, haciendo uso del parámetro **"bboxes"** devuelto por la función **detect()** que contiene la información sobre la posición del posible coche a analizar. Tras recortar los vehículos se almacenan en una matriz de coches llamada **coches{v}.** La "v" se refiere al número de iteración del bucle, para analizar todos los posibles coches.

Ilustración 19: Vehículos 1 y 2 recortados

- Se emplea la función **"colorThreshold",** para transformar a escala de grises y umbralizar la imagen de cada coche y convirtiéndola de esta forma en una imagen binaria para poder ejecutar la siguiente instrucción.

- Con la función **"regionprops()"** [6] se obtiene una matriz de valores que obtenidos de los objetos reconocidos en la imagen binaria. Los parámetros elegidos para ser extraídos para cada uno de los elementos detectados son: 'Area', 'Perimeter', 'BoundingBox', 'Orientation', 'MajorAxisLength', 'MinorAxisLength'. Estos parámetros serán utilizados para establecer una serie de condiciones a cumplir y de esta forma discriminar elementos con el objetivo de dar con la matrícula.

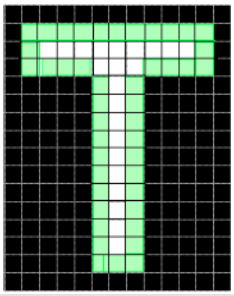
Nombre de la propiedad	Descripción	Soporte de N-D	Soporte de GPU	Generación de código
'Area'	Número real de píxeles en la región, devueltos como escalar. (este valor puede diferir ligeramente del valor devuelto por bwarea, que pondera diferentes patrones de píxeles de forma diferente.) Para encontrar el equivalente al área de un volumen 3D, utilice la propiedad 'Volume' de regionprops3.	Sí	Sí	Sí
'BoundingBox'	Rectángulo más pequeño que contiene la región, devuelto como un vector 1 porQ*2 , donde Q es el número de dimensiones de la imagen. Por ejemplo, en el vector [ul_corner width], ul_corner especifica la esquina superior izquierda del cuadro delimitador en el formulario [x y z ...]. width especifica el ancho del cuadro delimitador a lo largo de cada dimensión en el formulario [x_width y_width ...]. regionprops utiliza ndims para obtener las dimensiones de la matriz de etiquetas o la imagen binaria, ndims(L)y numel para obtener las dimensiones de los componentes conectados, numel(CC.ImageSize).	Sí	Sí	Sí
'MajorAxisLength'	Longitud (en píxeles) del eje principal de la elipse que tiene los mismos momentos centrales normalizados segundos como la región, devuelto como escalar.	2-D solamente	Sí	Sí
'MinorAxisLength'	Longitud (en píxeles) del eje menor de la elipse que tiene los mismos segundos momentos centrales normalizados como la región, devuelto como escalar.	2-D solamente	Sí	Sí
'Orientation'	Ángulo entre el eje xy el eje principal de la elipse que tiene los mismos segundos momentos que la región, devuelto como escalar. El valor es en grados, que van desde-90 grados a 90 grados. Esta figura ilustra los ejes y la orientación de la elipse. El lado izquierdo de la figura muestra una región de imagen y su elipse correspondiente. El lado derecho muestra la misma elipse con las líneas azules sólidas que representan los ejes. Los puntos rojos son los focos. La orientación es el ángulo entre la línea punteada horizontal y el eje principal. 	2-D solamente	Sí	Sí
'Perimeter'	Distancia alrededor del límite de la región. regresó como un escalar. regionprops calcula el perímetro calculando la distancia entre cada par de píxeles adyacentes alrededor del borde de la región. Si la imagen contiene regiones no contiguas, regionprops devuelve resultados inesperados. Esta figura ilustra los píxeles incluidos en el cálculo perimetral de este objeto. 	2-D solamente	Sí	Sí

Ilustración 20: Tabla propiedades regionprops() [6]

- Se crea un nuevo bucle para analizar cada uno de los objetos detectados en las diferentes imágenes de los posibles vehículos. Se realiza un filtrado haciendo uso de las propiedades geométricas de las matrículas. En esta etapa del proceso se emplean las propiedades obtenidas con la función **"regionprops()"** [6], tal y como se desprende a continuación:

  - La primera propiedad a determinar es la compacidad de cada objeto reconocido mediante la siguiente operación:

$$compacidad\{a\} = \frac{(Perímetro\ objeto)^2}{Area\ objeto}$$

    - "a" es el índice del segundo bucle, es la forma de realizar la operación para todos los objetos detectados.

    - Si se realiza esta misma operación con las proporciones reales de una matricula el resultado es el siguiente:

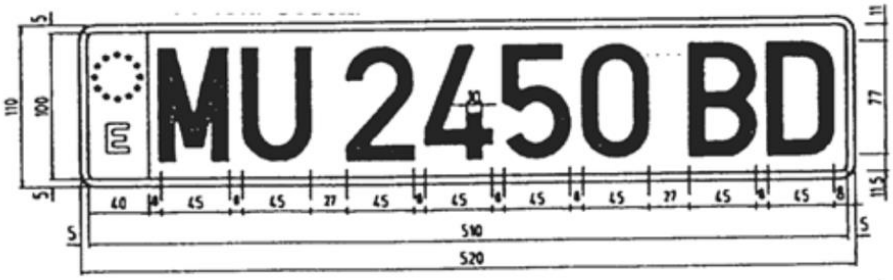


Ilustración 21: Medidas matrícula BOE [7]

Teniendo en cuenta que al umbralizar la imagen nos quedaremos únicamente con la parte blanca de la placa tomamos como base (B) = eje mayor = 520-5-5-40 = 470, y como altura (h) = eje menor = 110-5-5=100.

$$Área\ real = B * h = 470 * 100 = 47000$$

$$Perímetro\ real^2 = (2 * B + 2 * h)^2 = 1140^2$$

$$compacidad\ real = \frac{Perímetro\ real^2}{Area\ real} = \frac{1140^2}{47000} = 27.651$$

Se establece así otra de las variables inicializadas en la primera parte de código **COMPACIDAD_MEDIDAS_BOE =** 27.651.

- La segunda propiedad que se calcula es, la relación de aspecto de cada objeto reconocido, mediante la siguiente operación:

$$relación\ de\ aspecto\{a\} = \frac{Tamaño\ eje\ mayor\ objeto}{Tamaño\ eje\ menor\ objeto}$$

  - Si se realiza esta operación con las proporciones reales de una matrícula se obtiene el siguiente resultado:

$$Eje\ Mayor\ real = Base = 470$$

$$Eje\ Menor\ real = Altura = 100$$

$$relación\ de\ aspecto\ real = \frac{Eje\ mayor\ real}{Eje\ menor\ real} = \frac{470}{100} = 4.7$$

De esta forma se establece otra de las variables nombradas al inicio de la función: **RELACION_ASPECTO_MEDIDAS_BOE = 4,7.**

- Las últimas propiedades utilizadas son la orientación fijada en 0º o 180º, ya que la matricula tiene una posición horizontal o

prácticamente horizontal, y el Área que se ha fijado en un Área mínima de 800 pixeles tras realizar un testeo con una gran cantidad de imágenes.

- Una vez establecidos los cálculos reales y los valores que toman en el caso de tratarse de una matrícula se establecen las siguientes condiciones:

  - **Condición 1:** Si la compacidad{a} es mayor que la variable **COMPACIDAD_MEDIDAS_BOE** * 0.6 y compacidad{a} es menor que **COMPACIDAD_MEDIDAS_BOE** * 1.4 se pasa a la siguiente condición.

  - **Condición 2:** si la orientación del objeto está entre 10º y – 10º o entre 170º y 190º grados se pasa a la tercera comparación. Esta condición tiene el fin de filtrar objetos que a diferencia de las matrículas no se encuentren en posición horizontal

  - **Condición 3:** si relación de aspecto{a} es mayor a **RELACION_ASPECTO_MEDIDAS_BOE** * 0.6 y relación de aspecto{a} es menor a **RELACION_ASPECTO_MEDIDAS_BOE** * 1.4 se prosigue con el algoritmo y se salta a la cuarta y última comparación.

  - **Condición 4:** se realiza un filtrado de Área para eliminar los objetos demasiado pequeños para tratarse de una matrícula, como los reflejos, adhesivos, etc.

- En el caso de que se superen las 4 condiciones se incluye un rectángulo de color rojo sobre la imagen original para indicar la posición de las posibles matriculas. Para localizar y encuadrar las posibles matrículas se utiliza el parámetro **"BoundingBox"** obtenido con la función **regionprops()** [6] que nos devuelve la posición de la caja que envuelve al objeto en cuestión.

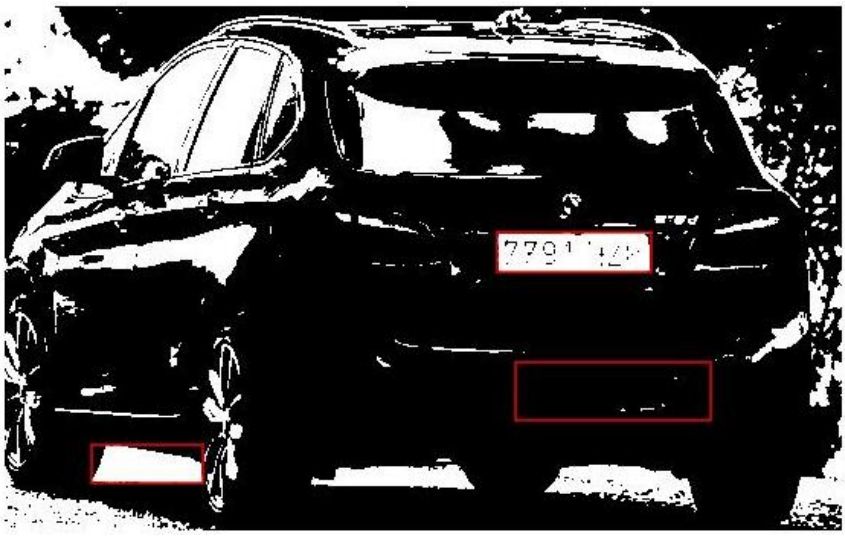


Ilustración 22:Detección de las posibles matrículas coche 1

Ilustración 23:Detección de las posibles matrículas coche 2

- Haciendo uso de este mismo parámetro se recortan mediante la función "**imcrop()"** [5] las posibles matriculas sobre la imagen del posible coche en color, para almacenarlas en una matriz de imágenes de posibles matriculas en color. El orden de esta matriz lo lleva el primero de los contadores **"count1"** inicializado al principio de la función. Este contador incrementa su valor en uno cada vez que un objeto supere las 4 condiciones explicadas anteriormente:

Ilustración 24:Matrícula coche 1 recortada

Ilustración 25:Matrícula coche 2 recortada

- En el paso anterior se obtienen más imágenes, es decir, además de la matricula propiamente detectada se almacenarán el resto de posibles matriculas detectadas. Todas estas posibles matriculas se utilizarán como parámetro de entrada de función **"reconoce()"** que como bien se ha explicado anteriormente realiza una combinación de operaciones binarias para facilitar la función del OCR, el cual consta de dos partes. En el primer 60% del eje mayor busca únicamente la siguiente lista de caracteres: E, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. En el segundo 40% del eje mayor busca los siguientes caracteres: B, C, D, F, G, H, J, K, L, M, N, P, Q, R, S, T, V, W, X, Y, Z:

Ilustración 26: Búsqueda de caracteres

- Finalmente pasará por la función **"analizaMatricula()"** que eliminará los caracteres indeseados como la E de Europa inicial. El resultado es una matriz de caracteres que incluye la posible matrícula detectada:

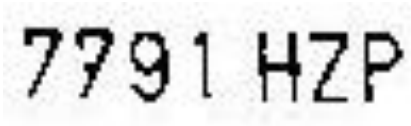


Ilustración 27: Imagen matrícula coche 1 tratada binariamente para ser utilizada por el OCR

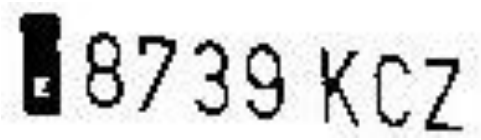


Ilustración 28:Imagen matrícula coche 2 tratada binariamente para ser utilizada por el OCR

- Como ya se adelantaba este proceso se realiza tanto para las matrículas como para los objetos de condiciones similares. Para discriminar definitivamente estos objetos se establece dos nuevos filtros con las siguientes condiciones:

  - **Condición 1:** si la posible matrícula contiene un número de caracteres adecuado para efectivamente tratarse de una matricula se pasa a la siguiente condición. Este número de caracteres viene establecido por la variable inicializada como **UMBRAL_CARACTERES_MINIMOS** = 8 (4 números + espacio + 3 letras).

    En este caso la cuenta de matriculas la llevará el segundo contador **"count2"** inicializado también al inicio de la función **"detectorMatriculasFoto()"** que se incrementará en uno cada vez que se cumpla condición descrita.

  - **Condición 2:** se comprueba que la matrícula leída por el OCR no se repita, es decir, en el caso de que se repita la misma matrícula la descarta:

Ilustración 29: Ejemplo matrícula repetida

Si supera este filtro se añade a un **String** (Cadena de caracteres). Este String recibe el nombre de "**MatriculaTxtFinal"**, inicializada en blanco al principio de la función y utilizada como parámetro de salida de la propia función.

Para este filtrado se utiliza la última de las variables inicializadas al inicio de la función. Se trata de la variable **"existe"** de carácter booleno, es decir, que únicamente puede tomar dos valores: "**true**" y "**false**" o "**1**" y "**0**" respectivamente. Compara la nueva matricula carácter a carácter con cada una de las introducidas anteriormente, en el caso de que haya mas de una. Si la matricula se repite la variable "**existe**" inicializada en "**false**" cambia su valor a "**true**" y se descarta y se vuelve a sustituir el valor de existe a "**false**". En el caso de superar este ultimo filtro se añade al String separada de la anterior por un salto de línea. Este es el aspecto final:

"7791 HZP

8739 KCZ"

Esta cuenta la lleva el último de los contadores inicializado **"count3"** que se incrementará en uno cada vez que se cumpla la ultima condición.

Hay que añadir que a lo largo de todo el código se han incluido varios **"pause()"** [8] para que en la interfaz gráfica se aprecien las fases del proceso completo.

4.2. Reconocimiento de matrículas en tiempo real

Esta función recibe el nombre de **detectorMatriculasFrame()** y realiza un proceso idéntico al explicado en el apartado anterior. En este caso se repite el proceso frame a frame desde que se enciende la cámara hasta que se apaga.

Al llamar a esta función se crea una conexión con la cámara USB conectada al ordenador para realizar este proceso se ha incluido la función de Matlab: **"support package for usb webcams"** [9]. Tras crear la conexión con la cámara, se hace la llamada a la función "**snapshot()"** [10] de la biblioteca de Matlab para obtener los frames que está capturando la cámara en tiempo real .

Se establece un bucle finito para que la cámara esté activa durante un tiempo determinado. El número de iteraciones definido arbitrariamente en este código es de 6 de modo que cuando analice 6 frames se cerrará la conexión con la cámara y devolverá todas las matriculas que haya encontrado mientras la cámara estaba activa. Necesita algo más de tiempo ya que repite 6 veces el proceso completo. Como se indicaba el número 6 es un número de iteraciones arbitrario, se podría haber programado un bucle infinito que cerrara la conexión al indicárselo el usuario.

5. Instalación y Uso del Programa

5.1. Instalación y puesta en marcha

Para instalar la aplicación se requieren los siguientes aspectos:

1) Instalar la versión de Matlab 2018B. Puede que funcione con otras versiones, pero el programa está elaborado en la versión citada.

2) Descargar la carpeta de **Ficheros anexos**. En esta carpeta se encuentran todas las funciones necesarias para el funcionamiento de la aplicación, así como un conjunto de imágenes de vehículos para las pruebas, etc.

3) Descargar el Toolbox de Matlab "support package for usb webcams". A través de esta herramienta se puede establecer una conexión entre la cámara USB y Matlab.

4) Adquirir una cámara USB. Cuanto mayores prestaciones tenga, mejores serán los resultados.

5) Abrir la herramienta Matlab y seleccionar como folder de trabajo la carpeta de ficheros adjuntos descargada

6) Ejecutar en la línea de comandos de Matlab la llamada a la función principal que genera la interfaz gráfica de usuario. Esta función recibe el nombre de **detectorMatriculas()**:

```
>> detectorMatriculas()
```

Al ejecutar este comando se abre la siguiente ventana con el software listo para utilizar:

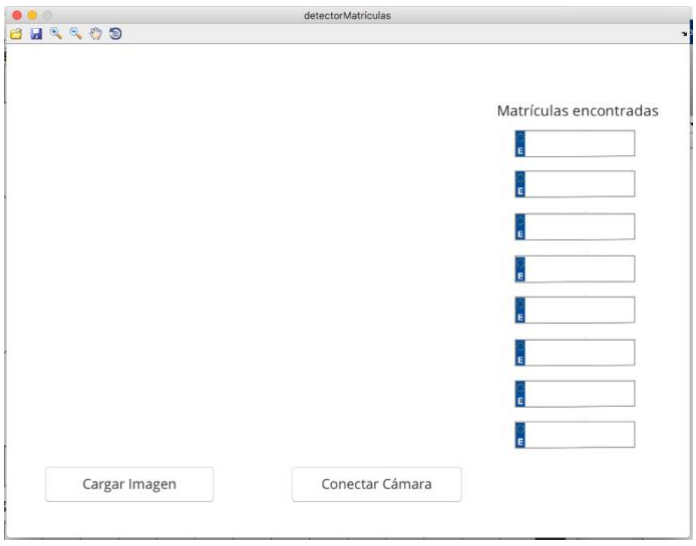


Ilustración 30: Diseño GUI

5.2. Descripción del entorno gráfico y su funcionamiento

En este apartado se describe detalladamente las partes que componen el entorno gráfico definitivo. Para elaborar esta interfaz se ha utilizado el tutorial de GUI de Matlab "**GUI_Tutorial".** Este tutorial esta en la carpeta **Documentos_Consultados** de la carpeta **Ficheros Anexos.**

La interfaz gráfica de usuario, como ya se adelantaba en el capitulo 2, esta creada con la herramienta **"GUI"** de Matlab que permite trabajar de forma interactiva con un editor gráfico que genera automáticamente el código correspondiente. Tal y como se planificaba en el boceto del entorno gráfico cuenta con 2 **Buttons** (Cargar imagen y Conectar Cámara), dos **Static Text** (Static y Dinamic) dos **Axes:** Axes1 y Axes2.Véase la función que desempeña cada uno de los elementos enumerados:

- Cargar Imagen: al clicar sobre él se restean los campos: **Axes1** y **Dinamic** para reiniciar el proceso. A continuación, se abre una ventana para seleccionar una imagen fija de vehículo/s almacenada en archivo.

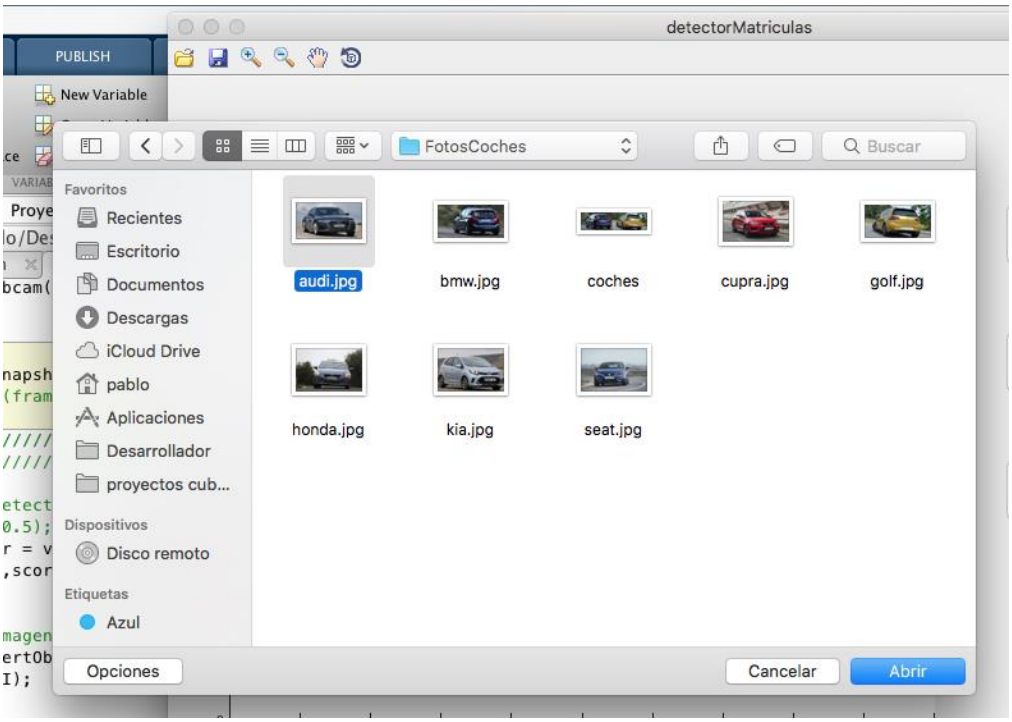


Ilustración 31: Proceso Cargar imagen 1

Una vez la seleccionamos se llama internamente a la función **detectorMatriculasFoto()** y se muestra en **Axes1** las distintas fases del procesado de imagen presentes en esta función. Durante este proceso en la casilla de texto **Dinamic** aparecerá: "cargando…". Finalmente, se muestra en la casilla **Dinamic** la/s matricula/s encontradas en la imagen.

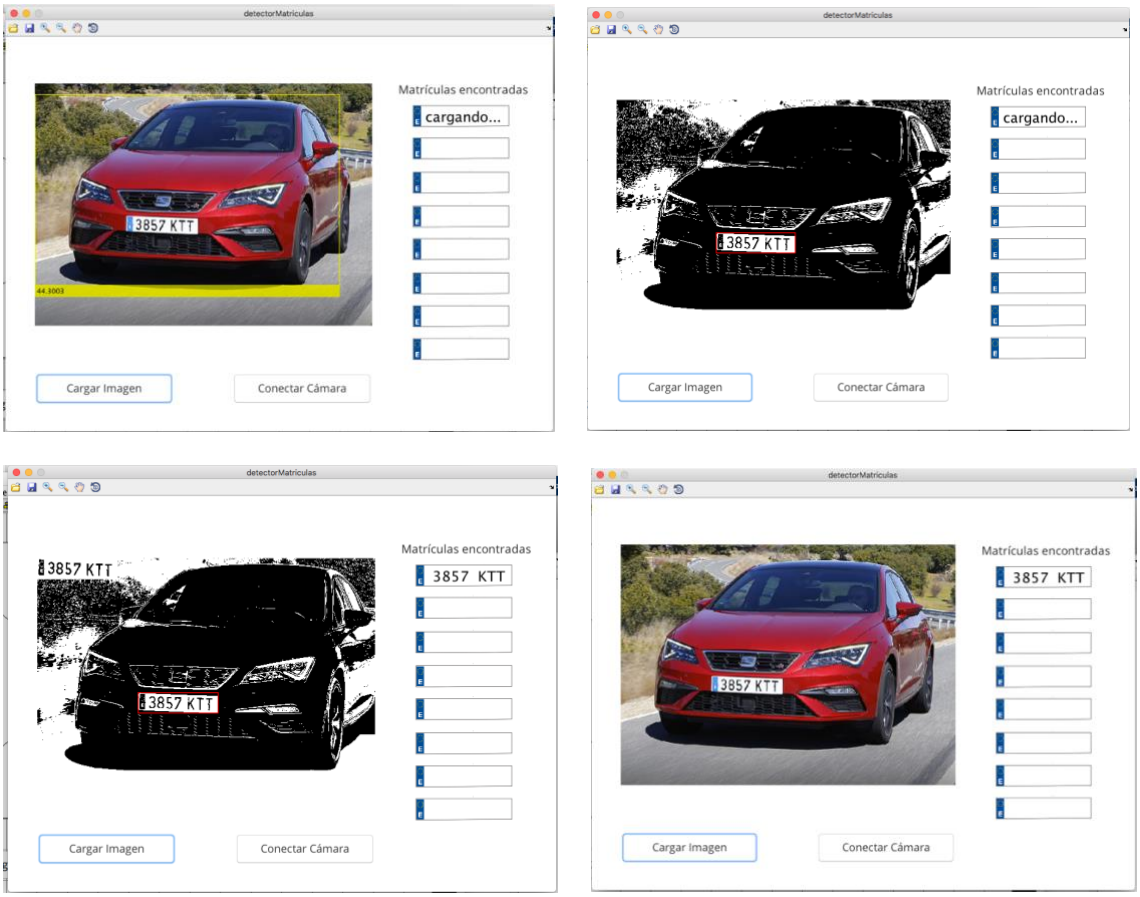


Ilustración 32: Proceso Cargar imagen 2

-botón Conectar Cámara: al clicar sobre él se restean los campos: **Axes1** y **Dinamic** para reiniciar el proceso. A continuación, se llama internamente a la función **detectorMatriculasFoto()** que se encarga de abrir la conexión con la cámara USB conectada a nuestro ordenador, y muestra en **Axes1** las distintas fases del procesado de imagen para cada frame. Durante este proceso en la casilla de texto **Dinamic** aparecerá: "cargando…". Una vez realizado el proceso en todos los frames se cierra la conexión con la cámara y muestra en la casilla **Dinamic** la/s matricula/s encontradas en mientras la cámara ha permanecido encendida.

Ilustración 33: Proceso conectar cámara

-Static: está situado sobre las matriculas reconocidas y contiene el texto “Matrículas encontradas”.

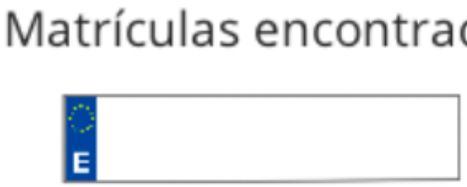


Ilustración 34: Static Text

-Dinamic: tal y como se adelantaba es el encargado de mostrar las matriculas detectadas. Cuando los botones “Cargar Imagen” y “Conectar Cámara” llaman internamente a las funciones **detectorMatriculasFoto() y detectorMatriculasFrame()** respectivamente, este objeto es el encargado de recibir el String con la/s matrícula/s encontradas.

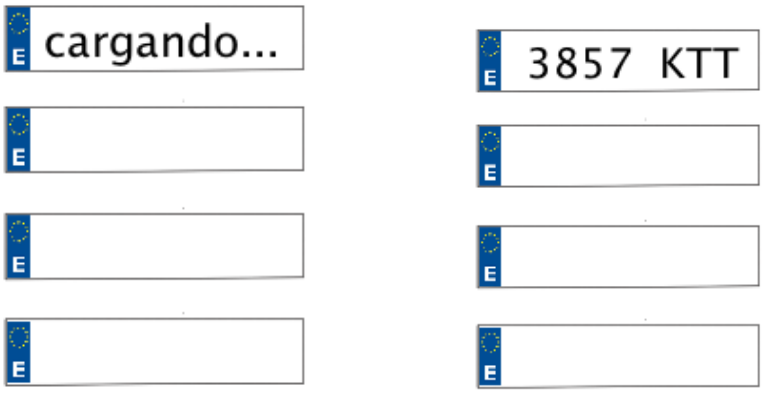


Ilustración 35: Dinamic Text

-Axes1: Como ya se ha visto es el cuadro de imagen encargado de mostrar cada una de las fases del proceso realizado sobre la imagen inicial o las capturas tomadas en tiempo real al llamar a las dos funciones principales: **detectorMatriculasFoto() y detectorMatriculasFrame().**

-Axes2: Es un elemento añadido a posteriori. Al correrse el código de la GUI se inicializa directamente cargando una imagen de matrículas vacías apiladas con canal de transparencia. La imagen solo contiene el borde de las matriculas:

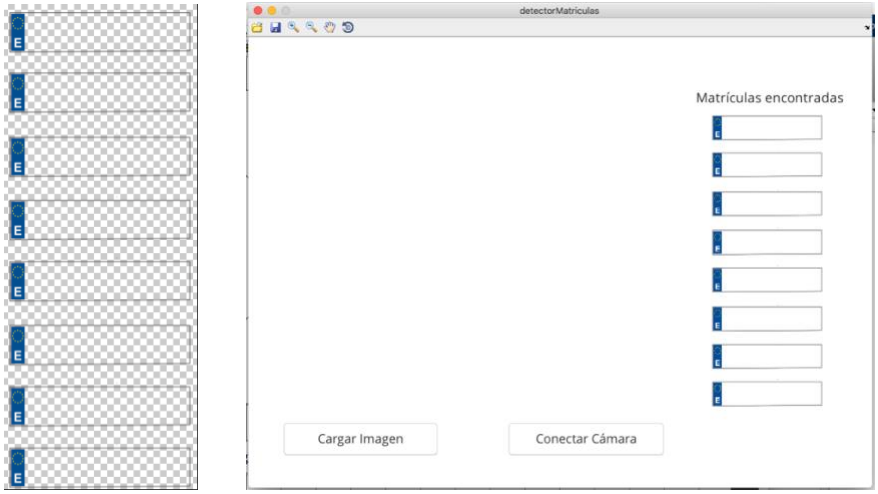


Ilustración 36: Imagen cargada en el Axes2

6. Análisis de resultados y principales problemas encontrados

En este capítulo se analizan los resultados de las diferentes fases de los dos procesos principales. Por lo general los resultados obtenidos son bastante favorables, aunque también se han detectado algunos inconvenientes.

El problema principal es que el resultado depende en gran medida de la iluminación de la imagen, y el color del coche. Como se comentaba en el capítulo de introducción este tipo de sistemas APNR suelen utilizar cámaras infrarrojas, es por este motivo por el cual las imágenes con mala iluminación tomadas por cámaras no-infrarrojas dan problemas. Fruto de la mala iluminación también fallan las funciones de la biblioteca de Matlab como **detect()** y **vehicleDetectorACF().** Véase en la siguiente imagen los problemas del detector, le da una puntuación de 11% a los pilotos traseros del automóvil y, sin embargo, no detecta el vehículo:

Ilustración 37: Vehículo no detectado (mala iluminación)

En la siguiente ilustración la iluminación es mejor y no tiene problema para detectar los distintos vehículos presentes:

Ilustración 38: Vehículos detectados (buena iluminación)

La función **colorThreshold()** encargada de binarizar las imágenes de los coches detectados tampoco funciona correctamente con todas las imágenes. Dan problemas las imágenes con poca iluminación y las imágenes donde el color del coche es muy similar al de la matrícula (blanco, gris muy claro, etc.), porque la placa se confunde con el resto del vehículo al binarizar.

Ilustración 39: Matrícula no detectada (mala iluminación)

Ilustración 40: Matrícula no detectada (vehículo blanco)

Otro problema afrontado es la detección de matrículas suponiendo imágenes con una correcta iluminación, debido a que como ya se anticipaba en los capítulos anteriores al binarizar la imagen para buscar objetos y sus propiedades con la función **regionprops()** se encuentran más objetos con características similares capaces de superar los cuatro filtros morfológicos de compacidad, relación de aspecto, orientación y Área:

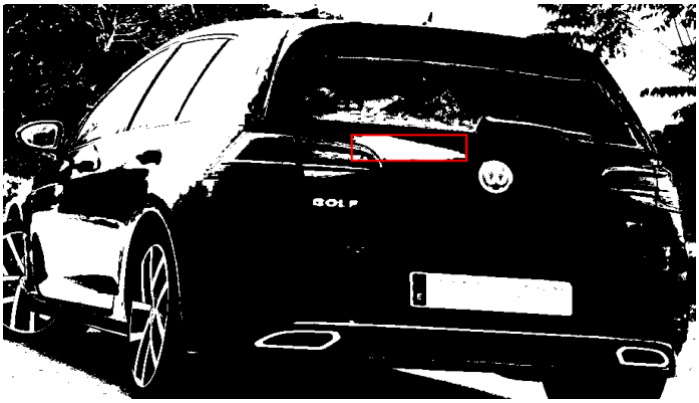


Ilustración 41: Objetos con propiedades similares a las de las matrículas

Por este motivo se establecen 2 filtros de texto. A pesar de no poder eliminar los objetos con morfología similar a la de las matrículas, estos objetos no contienen los caracteres típicos de una matrícula. Normalmente se trata de reflejos, faros, etc. De forma que realizando el filtrado de caracteres descrito al final del capítulo 4.1 eliminamos definitivamente estos objetos.

Otro problema resuelto es determinar la combinación de operaciones binarias más eficaz para preparar la matrícula para el OCR y condicionar al OCR para únicamente buscar números y la E de Europa en la primera parte de la matricula y letras en la segunda, de lo contrario confundía algunos caracteres como: Q y 0, 4 y L, etc. Por ultimo, filtrar la E de Europa para eliminarla.

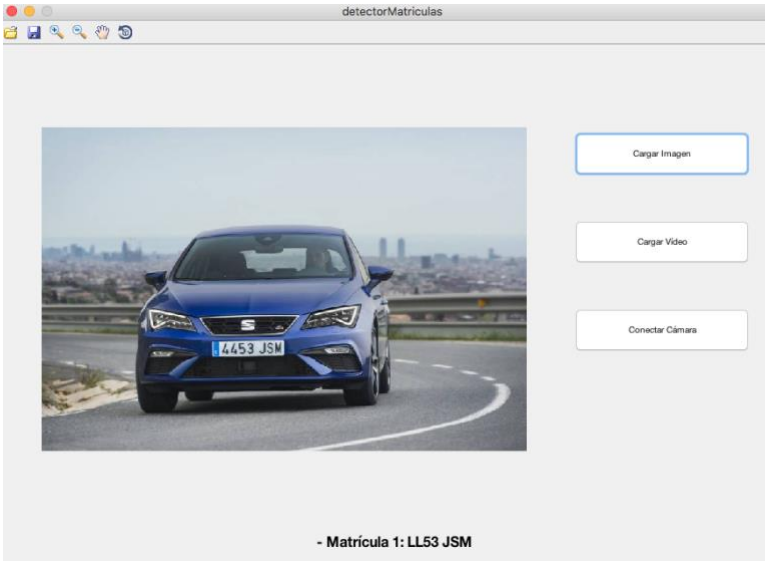


Ilustración 42: Diseño inicia. Caracteres mal detectados

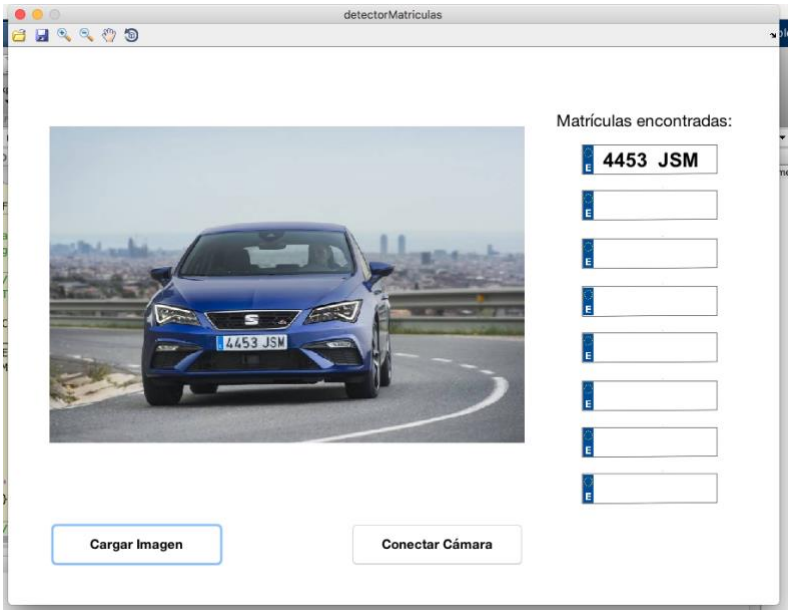


Ilustración 43: Diseño final. Caracteres bien detectados

Como se comentaba los resultados son muy favorables tras corregir gran parte de los problemas encontrados. En la mayoría de casos el resultado es correcto, véase diferentes ejemplos de esto:

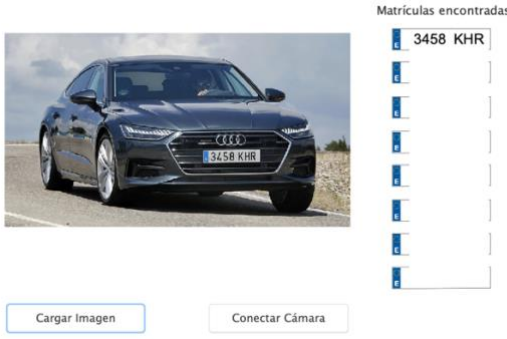


Ilustración 44: Prueba con vehículo con reflejos

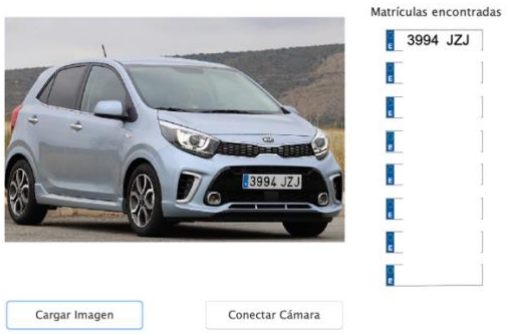


Ilustración 45: Prueba con vehículo con colores claros

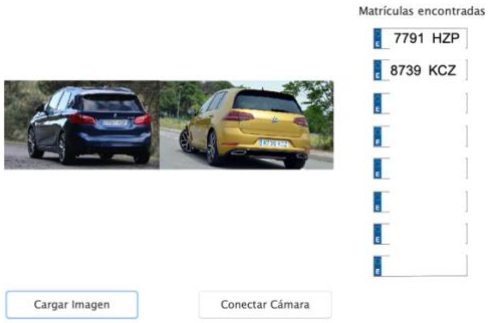


Ilustración 46: Prueba con 2 vehículos

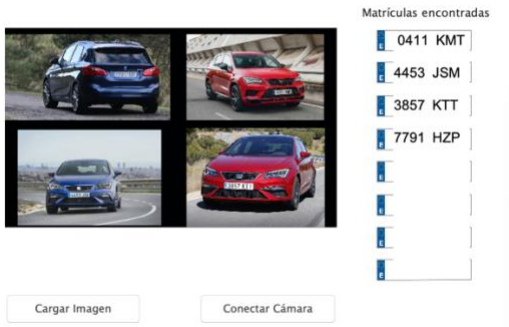


Ilustración 47: Prueba con 4 vehículos

También funciona correctamente con imágenes tomadas con una ligera angulación donde el coche y la matricula no están en horizontal:

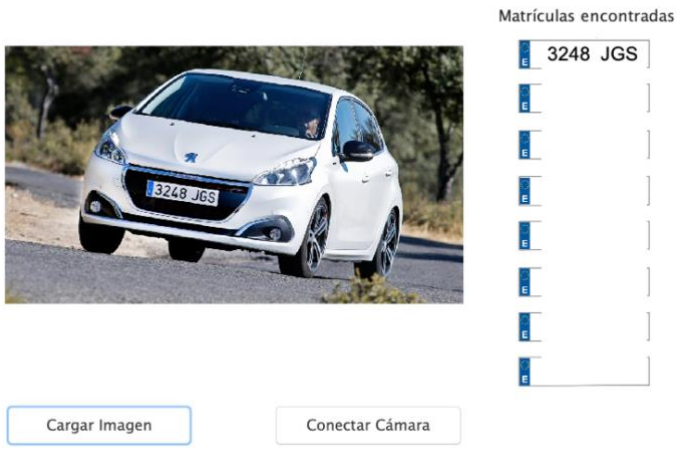


Ilustración 48: Prueba con vehículo y matrícula inclinada

Con imágenes tomadas lateralmente donde las matrículas tienen cierta perspectiva se obtienen resultados óptimos igualmente:

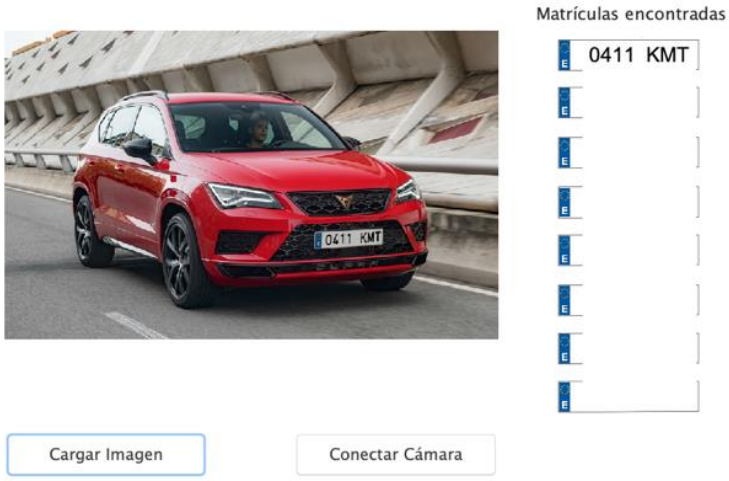


Ilustración 49: Prueba con vehículo y matrícula con perspectiva

Otro factor que funciona sin problema alguno es el filtro para evitar la repetición de matrículas. Por un lado, podemos generar una imagen duplicando un vehículo para hacer la comprobación con la función **detectorMatriculasFoto():**

Ilustración 50: Prueba con vehículo repetido

Las imágenes utilizadas en las pruebas están contenidas en la carpeta **Imagenes_Para_Pruebas** que se encuentra en la carpeta **Ficheros Anexos**. En la carpeta **Resultados,** también contenida en la carpeta de **Ficheros Anexos,** hay una carpeta interna con el nombre **Resultados_Imagenes_Para_Pruebas q**ue contiene capturas de los resultados de todas estas pruebas.

Por otro lado, la función **detectorMatriculasFrame()** repite el proceso 6 veces para frames muy similares o prácticamente idénticos con la intención de garantizar una cierta robustez. De esta forma se analiza 6 veces el mismo vehículo. Al visualizar el vídeo **Video_Desmostracion.MP4** que se encuentra en la carpeta **Resultados** contenida, a su vez, en la carpeta de **Ficheros Anexos,** y la siguiente captura del final del proceso se puede observar que no se repite la matricula pese a obtenerla 6 veces.

Ilustración 51: Vídeo demostración

7. Posibles mejoras

Como se exponía en el capitulo anterior, uno de los principales problemas es la iluminación de la imagen. Para solucionar este problema se plantean dos soluciones. En primer lugar, determinar un punto donde realizar una ecualización automática del histograma para utilizar la única función de umbralización sea útil en la mayoría casos posibles.

En las imágenes digitales, un histograma de color representa el número de píxeles que tienen en cada una de las listas fijas de rangos de colores, que se extienden sobre el espacio de color de la imagen, es decir, el conjunto de todos los posibles colores. [11] La ecualización del histograma se realiza sobre una imagen en escala de grises. Es una transformación que pretende obtener para una imagen un histograma con una distribución uniforme. Es decir, que exista el mismo número de píxels para cada nivel de gris del histograma [12].

Ilustración 52: Ecualización del histograma [12]

Aunque esta primera idea mejoraría el software, quizá la mejor opción es la segunda solución propuesta: sustituir la cámara Logitech de 720px de resolución por una cámara infrarroja con mayor resolución, como la ***DF5200HD-IR-ANPR***, cuyas imágenes están optimizadas para el reconocimiento de matrículas en vehículos de forma automática. De [8] esta forma se soluciona el problema de iluminación y calidad:

Ilustración 53: Cámara infrarroja - Sistema APNR [13]

Otro de los problemas mencionado que ha quedado sin solución es la detección de matriculas en coches claros. Para este problema se presenta la siguiente solución, introducir Machine Learning para que el software sea capaz de encontrar la matricula sin necesidad de binarizar.

El Machine Learning es un campo de la Inteligencia Artificial basado en la búsqueda de algoritmos y heurísticas para convertir muestras de datos en programas de ordenador automáticamente. Los modelos o programas resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio (potencialmente infinito) de datos.

En este caso se basaría en un algoritmo de aprendizaje supervisado, es decir, el algoritmo genera una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. Para realizar este entrenamiento se escogería una gran cantidad de imágenes de vehículos con matrículas españolas. A este grupo de imágenes se le denomina conjunto de entrenamiento. Para cada imagen de vehículo o vehículos del conjunto de entrenamiento se señala la posición y encuadre de la matrícula, de forma que el algoritmo lo tome como parámetro de salida. A partir de este conjunto de entrenamiento se van estableciendo unos descriptores comunes.

Tras realizar el aprendizaje hay una fase de pruebas donde se escoge un grupo de imágenes de vehículos mas pequeño y sin repetir ninguna del conjunto anterior y se pone a prueba el algoritmo. En el caso de que funcione correctamente seguimos adelante con la función generada, si todavía falla cogemos un nuevo conjunto de imágenes no repetidas y continuamos con el aprendizaje hasta obtener una función lo suficientemente fiable [14].

Otro punto a tratar es la realización de un OCR más especifico, entrenado únicamente con caracteres de Matrículas españolas. Pese a que esta propuesta le daría al software un plus de robustez para este sistema APNR concreto, se ha considerado una medida innecesaria debido a los buenos resultados obtenidos y la posibilidad de que la fuente de las matriculas españolas cambie como en el caso de Holanda que en 2002 cambio la fuente para introducir pequeños espacios en algunas letras como la P o la R para hacerlas más distinguibles y por tanto más legibles para este tipo de sistemas [1]:

Ilustración 54: Fuente matrículas holandesas

Por último, se propone añadir un sistema de detección de velocidad del vehículo capturado en Tiempo Real. Para este fin sería conveniente añadir una segunda cámara USB para realizar cálculos de distancias a través de la estereoscopía que ofrece contar con un sistema de dos cámaras calibradas. Es una buena opción para implementarlo en versiones futuras.

8. Conclusiones

Podemos concluir que la detección y el reconocimiento de matrículas mediante el procesado de imagen es una tarea muy compleja. Hay una gran cantidad de factores que pueden interferir negativamente en el proceso: mala iluminación, reflejos, defectos, suciedad, etc. Pese a que no se pueden esperar unas condiciones ideales en el medio una buena adquisición de la imagen facilita en gran medida la tarea del software.

Este software es capaz de detectar e identificar matrículas con el sistema de matriculación español de forma efectiva, y robusta. Es capaz de trabajar a gran velocidad, ya que se le requiere que pueda hacer el reconocimiento en tiempo real con la intención de poder ser aplicado en entornos como en la entrada de un parking, en autopistas para cobrar el peaje electrónicamente y ahorrar tiempo, etc.

Sin embargo, todavía puede se puede optimizar el código para aumentar su rendimiento y efectividad. Como se comentaba en el capitulo anterior existe un amplio abanico de posibilidades para mejorar el código.

Además de las posibilidades comentadas se propone como trabajo futuro generar una aplicación que sirviéndose de este tipo de sistemas APNR, un servidor y una base de datos similar a la de la DGT pueda gestionar: el registro, notificaciones, pago, etc. de multas de forma inmediata. De tal manera que, por una parte, no sería necesario estar pendiente de si llega o no llega una posible multa por exceso de velocidad; en el caso de estar mal estacionado se recibiría una notificación de la multa por mal estacionamiento, dando la oportunidad al usuario/conductor de evitar que retire el coche la grúa, etc.

La realización de este proyecto me ha permitido familiarizarme con los conceptos relacionados con es procesado de imagen, con la herramienta Matlab y ampliar mis conocimientos de programación, principalmente centrada en el procesado de imagen. También he podido comprobar en la práctica los conceptos teóricos adquiridos durante el proyecto, de esta forma he podido comprobar que la práctica no es tan ideal como la teoría y que en ella recae mayor complejidad.

Tras superar diversos problemas que han ido surgiendo, cabe mencionar que la complejidad del proyecto ha supuesto un gran esfuerzo y muchas horas. A pesar de ello es justo afirmar que ha valido la pena y se puede decir que se ha alcanzado con éxito los objetivos iniciales de este TFG.

Bibliografía

[1] M. Merino, «xakata.com,» Xakta, 18 Agosto 2018. [En línea]. Available: https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-inteligencia-artificial-antagonica-como-puede-manipular-a-otras-ias. [Último acceso: 09 Agosto 2019].

[2] Wikipedia, «es.wikipedia.org,» Wikipedia, 04 Julio 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Reconocimiento_automático_de_matr%C3%ADculas. [Último acceso: 13 Agosto 2019].

[3] MathWorks-vehicleDetectorACF(), «Mathworks.com,» [En línea]. Available: https://la.mathworks.com/help/driving/ref/vehicledetectoracf.html?searchHighlight=•%20vehicleDetectorACF()%3A&s_tid=doc_srchtitle.

[4] MathWorks-Detect(), «MathWorks.com,» [En línea]. Available: https://es.mathworks.com/help/vision/ref/acfobjectdetector.detect.html?searchHighlight=detect&s_tid=doc_srchtitle..

[5] Matlab imcrop, «Mathworks.com,» [En línea]. Available: https://www.mathworks.com/help/images/ref/imcrop.html?searchHighlight=imcrop&s_tid=doc_srchtitle.

[6] Matlab regionprops, «Mathworks.com,» [En línea]. Available: https://www.mathworks.com/help/images/ref/regionprops.html?s_tid=doc_ta.

[7] BOE, «BOE.es,» [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2000-16805.

[8] Matlab-pause, «Mathworks.com,» [En línea]. Available: https://es.mathworks.com/help/matlab/ref/pause.html?searchHighlight=pause()&s_tid=doc_srchtitle.

[9] MATLAB Support Package for USB Webcams, «Mathworks.com,» [En línea]. Available: https://www.mathworks.com/matlabcentral/fileexchange/45182-matlab-support-package-for-usb-webcams.

[10] Matlab snapshot, «Mathworks.com,» [En línea]. Available: https://www.mathworks.com/help/supportpkg/ipcamera/ug/snapshot.html?searchHighlight=snapshot&s_tid=doc_srchtitle.

[11] Wikipedia, «Wikipedia/Histograma,» [En línea]. Available: https://es.wikipedia.org/wiki/Histograma.

[12] Wikipedia, «Wikipedia/Ecualización_del_histograma,» [En línea]. Available: https://es.wikipedia.org/wiki/Ecualización_del_histograma.

[13] M. L. Michelone, «UNOCERO,» 17 Octubre 2017. [En línea]. Available: https://www.unocero.com/entretenimiento/programacion-ludica-ecualizacion-del-histograma-de-una-imagen/.

[14] Dallmeier, «Seguridadprofesionalhoy,» [En línea]. Available: https://www.seguridadprofesionalhoy.com/camara-reconocimiento-matriculas/.

[15] V. Roman, «Medium.com,» [En línea]. Available: https://medium.com/datos-y-ciencia/introduccion-al-machine-learning-una-gu%C3%ADa-desde-cero-b696a2ead359.

Declaración de trabajo original

Yo, Pablo Pallarés declaro que soy el autor de este trabajo y que no ha sido copiado o hecho por otras personas.