Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Ingeniería Informática

Curso Académico 2011/2012

Localización de automóviles, lectura de matrícula y reconocimiento de marca y modelo

Proyecto Fin de Carrera

Autor: Roberto Valle Fernández
Tutor: José Miguel Buenaposada Biencinto

19 de diciembre de 2011

Agradecimientos

En el plano personal, me gustaría dar las gracias a las siguientes personas por la ayuda ofrecida durante el desarrollo de este proyecto fin de carrera (PFC):

- En primer lugar, quiero agradecer a mi tutor José Miguel Buenaposada Biencinto la oportunidad que me ha brindado para realizar este proyecto. Porque siempre ha tenido una respuesta para las muchas preguntas que le he hecho, y sobre todo por la confianza que depositó en mí desde el principio. Por esos casi estos 12 meses de trabajo contigo, muchas gracias.

- También me gustaría dar las gracias a todos mis compañeros de la universidad por el apoyo que siempre me dieron. En especial quiero recordar a David Marchante, Miguel Jiménez, Miguel París y Moisés Vázquez con los cuales he compartido mis dos últimos años de carrera.

- Gracias de corazón a mis amigos de toda la vida por hacerme feliz, a todos los que me quieren tal y como soy. Siempre os llevaré en mi corazón pase lo que pase. No cambiéis nunca.

- Y finalmente, quiero dedicarle el título de ingeniero superior en informática a mi familia, por todos los ánimos que me han dado. Particularmente esto va para mi padre, mi madre y mi hermano, por haberme educado así y por creer en mis posibilidades en todo momento. Como primer miembro de la familia que llega a la universidad quería reservar un hueco de mi memoria para vosotros.

Muchísimas Gracias.

Resumen

Durante los últimos años, se han empezado a desarrollar aplicaciones cada vez más sofisticadas para el reconocimiento automático de ciertos vehículos a partir de una serie de imágenes o secuencias de vídeo generadas por una videocámara. Este tipo de sistemas resultan de gran utilidad para aquellas empresas que están buscando la manera de extraer información característica de los automóviles para clasificarlos (recuperando los caracteres de sus matrículas o reconociendo la marca y el modelo de los vehículos en cuestión) con la idea de evitar robos o controlar el acceso de dichos automóviles a diferentes lugares.

Con la realización de este proyecto fin de carrera (PFC) la idea es **construir una aplicación informática que consiga localizar los automóviles vistos de frente en una escena introducida por imagen o vídeo para tratar de reconocer cada automóvil por su matrícula, marca y modelo asociados**. El objetivo es construir un sistema que permita detectar la posición de todos los automóviles de la escena para, por ejemplo, asegurar que no se comenten infracciones de tráfico. La aplicación se encargará tanto de reconocer el contenido de las matrículas como de verificar la marca y modelo de cada vehículo (la policía actuaría en consecuencia atrapando al conductor del automóvil).

Llegados a este punto, se deben realizar una serie de experimentos para obtener las mejores tasas de acierto posibles para cada uno de los clasificadores generados. La aplicación propuesta hará uso de diferentes bases de datos que formarán un total de 13.000 imágenes con automóviles, matrículas, caracteres, etc. Respecto al rendimiento alcanzado por los clasificadores, la tasa de aciertos llega al 94 % de aciertos en los clasificadores encargados de localizar los objetos de interés (automóviles y las matrículas). Para el clasificador que reconoce los caracteres de la matrícula la tasa de aciertos es del 92 %. Finalmente, para encontrar la marca y modelo al que pertenece el automóvil a identificar se va a diseñar un clasificador que alcanza una tasa de acierto del 83 % al clasificar el automóvil con el modelo de la base de datos que más se le parece (llega a un 92 % cuando el modelo correcto está entre los 3 más parecidos). La aplicación se ha desarrollado bajo un entorno Windows, con el lenguaje de programación C++ y con el apoyo de las bibliotecas OpenSURF y OpenCV.

Índice general

Índice de figuras

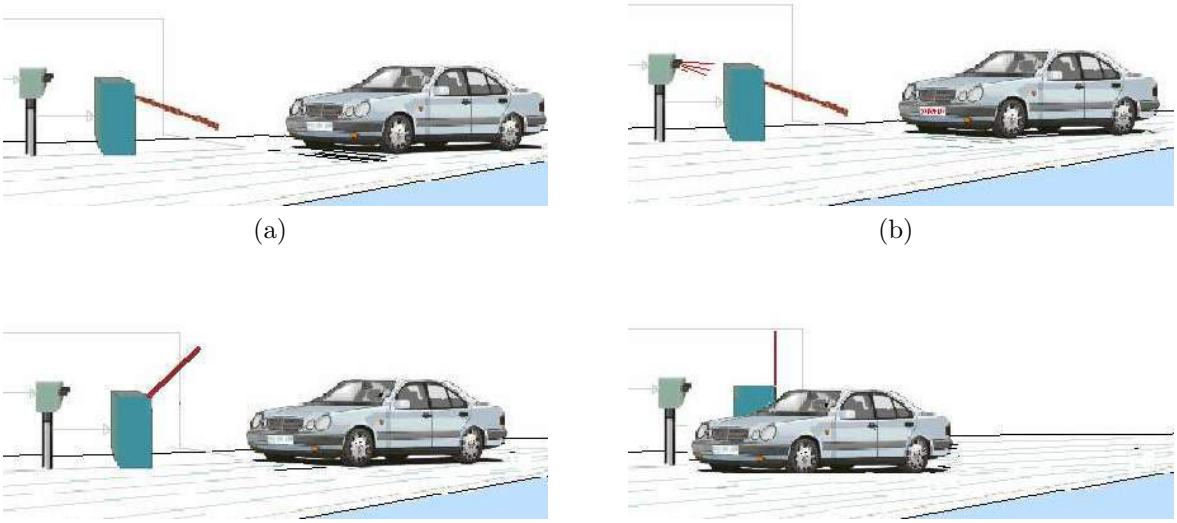
Capítulo 1

Introducción

En estos últimos años la importancia de los métodos de autentificación y reconocimiento de vehículos ha aumentado considerablemente. Cada vez son más los países que utilizan este tipo de sistemas para: evitar el robo de los vehículos estacionados en ciertos aparcamientos, controlar el acceso de los automóviles a determinadas zonas, detectar los vehículos que cometen infracciones de tráfico en las carreteras, etc. Por ejemplo, en el centro de Londres, en un esfuerzo por reducir el tráfico que se forma, los conductores de los vehículos se ven obligados a pagar una tarifa diaria para circular por las calles. Si alguna de las cámaras de vigilancia que tienen repartidas por la ciudad (Figura 1.1) identifica un vehículo del que no consta que pague dicha tarifa, se procederá a multar al propietario del vehículo en cuestión (CCTV Closed-Circuit Television [1]).

Figura 1.1: Cámaras de vídeo vigilancia distribuidas por Londres.

Generalmente los sistemas de identificación de vehículos que existen en el mercado basan su funcionamiento en el reconocimiento automático de la matrícula del mismo, es decir, tratan de localizar primero la matrícula en la imagen para reconocer después los caracteres y los dígitos que contenga. A este tipo de aplicaciones se las conoce como ANPR (Automatic Number Plate Recognition) y suelen conseguir resultados bastante buenos. Estos sistemas a menudo utilizan iluminación infrarroja para hacer posible que la cámara pueda tomar fotografías en cualquier momento del día. En la Figura 1.2 se muestra el típico control de acceso a una zona que bloquea una barra, que nos dará paso tras identificar el vehículo como válido.



(a) (b) (c) (d)

Figura 1.2: Ejemplo de utilización de ANPR.

Llegados a este punto, nos podemos dar cuenta de que realmente este tipo de aplicaciones clasifican el vehículo por el contenido de su matrícula, consultando en una base de datos la marca y el modelo del vehículo que se asocia a dicha matrícula. La idea de este tipo de sistemas es reconocer los caracteres de la matrícula cuando el automóvil se coloca en una determinada posición delante de la cámara. Normalmente con identificar la matrícula se podrá clasificar el vehículo, salvo en aquellos casos en lo que por ejemplo se hayan intercambiado previamente las matrículas entre automóviles.

1.1. Objetivos

El principal objetivo que se persigue con la realización de este proyecto fin de carrera (PFC) es la creación de un sistema que permita localizar automáticamente los automóviles vistos de frente en cualquier posición de la escena, para después identificar los caracteres de cada matrícula asociada y reconocer la marca y el modelo de los vehículos respectivamente. La aplicación desarrollada debe recibir una imagen o vídeo de entrada para procesar su contenido en busca de los automóviles. Para llegar a este objetivo final, es necesario alcanzar los siguientes objetivos parciales:

- *Construcción de las bases de datos.* Durante la realización del PFC es imprescindible crear las diferentes bases de datos que nos permitan generar los clasificadores. La idea es seleccionar imágenes donde observamos **automóviles** vistos de frente para construir los clasificadores encargados de localizar los vehículos y reconocer la marca y el modelo de los mismos. Con respecto al clasificador de matrículas son de interés todas las imágenes de **matrículas** de vehículos (coches, autobuses, camiones, ...) y por último, para generar el clasificador encargado de reconocer las letras y los dígitos se va a formar una base de datos con los **caracteres** extraídos de las matrículas anteriores. Estas mismas imágenes también permitirán realizar los experimentos durante el desarrollo del proyecto.

- *Desarrollar los algoritmos de clasificación oportunos.* En el Capítulo 2 se detallan los algoritmos implementados para detectar los objetos de interés y identificar los caracteres de la matrícula. Para localizar todos los automóviles y cada matrícula en la escena se generan dos clasificadores binarios capaces de determinar la presencia de estos objetos de interés (objeto / no objeto). Finalmente se debe construir un clasificador multiclase encargado de reconocer cada uno de los posibles caracteres que podrían aparecer en las matrículas de los automóviles.

- *Integrar el clasificador de marca y modelo.* En el PFC presentado en [12] se implementó un algoritmo para reconocer la marca y modelo de un automóvil conocida la posición del mismo en la escena (se marcan las esquinas de la matrícula en la imagen de entrada). La idea es intentar ajustar al proyecto actual, el clasificador desarrollado anteriormente, teniendo en cuenta que ahora la matrícula se detecta automáticamente.

- *Evaluación del rendimiento de los clasificadores.* Tras realizar los experimentos que se muestran en el Capítulo 3 debemos evaluar los resultados obtenidos. En principio, para considerar que el trabajo realizado ha sido bueno, esperamos alcanzar una tasa de aciertos cercana al 90 % para cada uno de los clasificadores por separado.

- *Implementación en C++ de la interfaz.* El objetivo es trasladar al lenguaje C++ los resultados que se obtienen en las fases anteriores del PFC. El usuario debería ser capaz de cargar una imagen o vídeo como entrada para la aplicación, que el sistema se encargará de procesar para extraer la información pertinente de los automóviles vistos de frente. La idea es visualizar los resultados que se irán generando en la misma pantalla en la que se van procesando los vehículos, a poder ser en tiempo real (aproximadamente 25 imágenes por segundo).

1.2. Posibles problemas

El sistema de reconocimiento que proponemos en este PFC se basa en esta idea de identificar los vehículos (matrícula y modelo) extrayendo las características de la vista frontal del automóvil. Esto implicará que debamos tener especial cuidado con las imágenes configuradas como entrada para el sistema. A continuación, se van a enumerar algunas de estas situaciones a evitar para reconocer el automóvil correctamente:

- *Oclusiones que dificultan la visualización del vehículo.* A la hora de localizar los automóviles es imprescindible que los vehículos se vean lo más claramente posible en la imagen. Los clasificadores encargados de detectar los objetos de interés están entrenados para reconocer los objetos en situaciones habituales, de forma que bajo determinadas circunstancias (objetos tapando el automóvil, niebla, oscuridad, ...) el sistema puede dejar de funcionar correctamente.

- *País de los automóviles.* Algunas marcas sacan versiones distintas del mismo modelo dependiendo del país donde se vaya a utilizar. Tenemos que controlar que las imágenes coleccionadas para la base de datos no sean muy diferentes para un mismo modelo. Respecto al sistema OCR (Optical Character Recognition), el problema que supone que cambien las matrículas dependiendo del país es mucho mayor pues normalmente los formatos son distintos. Para este PFC se han elegido modelos de automóviles comercializados en España.

- *Posible vehículo dañado.* También se ha de entrar a valorar lo que pasará si el vehículo está dañado de tal forma que se dificulte su reconocimiento cuando lo vemos frontalmente. Sería interesante que cuando los daños no sean muy grandes la aplicación funcionara con normalidad. Tampoco se realizarán experimentos en el PFC con imágenes en esta situación.

- *Orientación de los automóviles.* Para localizar adecuadamente los vehículos en la escena es imprescindible que la posición de los mismos sea la adecuada (automóvil colocado en horizontal). Hay que tener en cuenta que si el clasificador no localiza el frontal del automóvil, no podrán aplicarse el resto de funcionalidades del sistema.

- *Cambios en la iluminación.* Como se ha indicado anteriormente, lo más importante es que el vehículo que queramos identificar se vea claramente, por eso siempre que sea posible será interesante conseguir imágenes de buena calidad (sin grandes contrastes de luz).

En la Figura 1.3 se presentan algunos ejemplos de imágenes problemáticas para la aplicación a desarrollar.

(a) (b) (c)

Figura 1.3: Imágenes que a priori se deberían evitar en el sistema.

Finalmente, al analizar los resultados del sistema en el Apartado 5.1 se detallará el comportamiento de la aplicación desarrollada para cada uno de estos casos que presentan problemas.

1.3. Estructura del proyecto fin de carrera

El *Capítulo 2* detalla el conjunto de técnicas y algoritmos que describen el funcionamiento del sistema encargado de extraer la información de los automóviles vistos de frente en la escena.

En el *Capítulo 3* se realizan los experimentos correspondientes que determinan el rendimiento de los clasificadores generados a lo largo del proyecto. La idea es describir con exactitud las herramientas necesarias para implementar los algoritmos especificados.

En el *Capítulo 4* se especifica el plan de trabajo y la metodología que se ha seguido en la elaboración del sistema. UML permite detallar algunos diagramas que facilitan la comprensión del diseño y la arquitectura del sistema desarrollado.

El *Capítulo 5* presenta las conclusiones del PFC con la idea de determinar si se han cumplido o no los objetivos propuestos en un principio. Por último se propone un listado con algunos trabajos futuros para mejorar la aplicación desarrollada.

Capítulo 2

Algoritmos utilizados

El objetivo de la aplicación es detectar a partir de una imagen o vídeo de entrada, cada uno de los automóviles vistos de frente en la escena, para posteriormente reconocer la marca y modelo de cada uno de los vehículos, así como identificar los caracteres de la matrícula asociada a los mismos. Para ello se van a especificar en cada apartado del capítulo, un conjunto de técnicas y algoritmos que describan el funcionamiento del sistema de la Figura 2.1. Primeramente se implementará un sistema capaz de localizar objetos en tiempo real, a partir de un clasificador entrenado previamente para detectar frontales de automóviles y matrículas respectivamente. Para identificar los caracteres de la matrícula se construirá otro clasificador distinto que reconozca dígitos y letras. Finalmente para reconocer la marca de cada vehículo, se extraerán los detalles del modelo con un descriptor de características capaz de determinar el resultado.

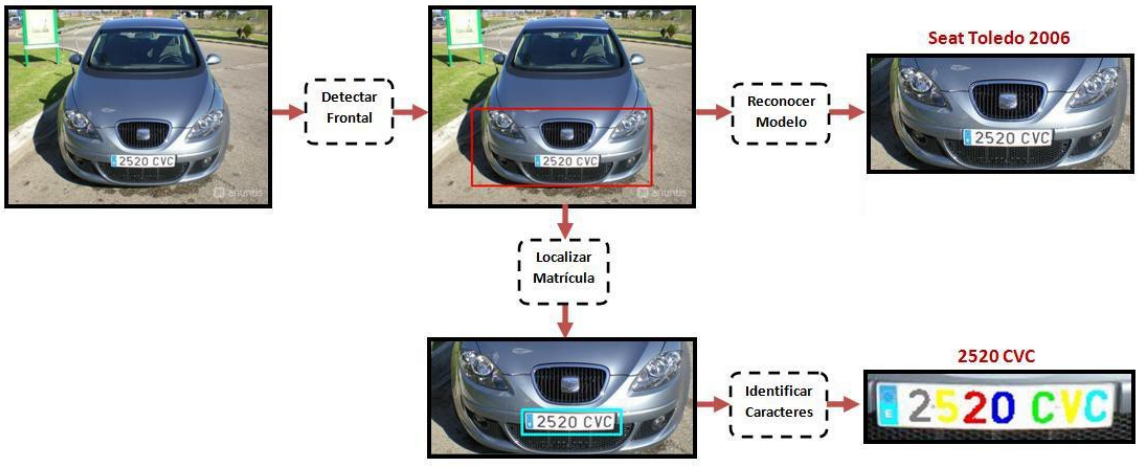


Figura 2.1: Esquema que detalla el funcionamiento del sistema a desarrollar.

2.1. Detectar frontales de automóviles y matrículas

El primer algoritmo a implementar permitirá localizar la posición exacta de un tipo de objeto determinado en una escena introducida por vídeo o imagen. El objetivo será generar un par de clasificadores que se encarguen tanto de localizar los automóviles vistos de frente como las matrículas respectivamente. La idea es entrenar previamente estos clasificadores binarios (objeto / no objeto) para después aplicar los resultados al sistema propuesto.

En cuanto al detector de objetos, cabe destacar la publicación de Paul Viola y Michael Jones [2], donde se presenta la técnica de clasificación seleccionada: **Cascade of Boosted Classifiers Based on Haar-like Features**. La idea del algoritmo será entrenar cada clasificador binario con un conjunto de imágenes donde se observa el objeto a localizar (imágenes positivas) y otras tantas imágenes que no contienen dicho objeto (imágenes negativas). Llegados a este punto, con el clasificador entrenado correctamente, el sistema será capaz de realizar la búsqueda del objeto en cuestión por toda la escena mediante una ventana que recorrerá la imagen aplicando dicho clasificador a diferentes escalas.

A la hora de desarrollar el resto de funcionalidades del sistema es evidente la importancia de esta etapa ya que, tanto para identificar los caracteres de la matrícula como para reconocer el modelo del automóvil, será necesario haber localizado correctamente el frontal del automóvil en la escena. Además, es importante que los clasificadores sean lo suficientemente robustos como para no perder información relevante para las etapas posteriores (como por ejemplo los caracteres de la matrícula).

2.1.1. Algoritmo de ventana deslizante

Independientemente de cuál sea el objeto a localizar, el algoritmo de búsqueda debe recorrer la región de la imagen por donde es posible que aparezca el objeto. Esta técnica recibe el nombre de "ventana deslizante" pues realmente lo que hace es iterar sobre la imagen con una ventana que se desliza a diferentes escalas sobre una determinada región de interés. Para la etapa de detección de automóviles vistos de frente será imprescindible recorrer la imagen entera con la ventana para aplicar el clasificador coche / no coche (ya que no se conoce la posición exacta del automóvil en la imagen). Sin embargo, para la etapa de localización de la matrícula, la región de interés a recorrer por la ventana deslizante se reduce al frontal del automóvil detectado anteriormente, ya que sólo son de interés para la aplicación las matrículas ligadas a los vehículos (así se evita recorrer la imagen entera).

A priori no se conoce el tamaño de los objetos de interés en la imagen por lo que la ventana deslizante deberá ser capaz de cambiar de tamaño para tratar de clasificar todas las posiciones de la imagen a distintas escalas. El clasificador está diseñado para que al ajustar la ventana sobre el objeto de interés, el resultado de clasificación sea positivo. La Figura 2.2 esquematiza el funcionamiento de la ventana deslizante al recorrer una imagen concreta con un automóvil en la escena. El clasificador previamente entrenado se encargará de evaluar cada región para determinar la posible presencia del objeto de interés en la imagen.

A continuación se detalla el pseudocódigo en MATLAB del algoritmo encargado de recorrer la imagen con la técnica de la ventana deslizante. La idea es: establecer un tamaño inicial para la ventana deslizante (*detection_width*, *detection_height*) y recorrer la imagen para encontrar el objeto de interés. Este procedimiento se repetirá ya sea aumentando el tamaño de la ventana deslizante o bien disminuyendo el tamaño de la imagen de entrada en cada iteración. Lo más eficiente será establecer un factor de escala (*detection_scale_step*) que oscile entorno al 10 % y el 20 % para aumentar la ventana deslizante o redimensionar la imagen original (el pseudocódigo presentado implementa esta segunda opción).

```
% Tamaño de la ventana deslizante
DET_WIDTH = 35;
DET_HEIGHT = 15;
% Disminuir la imagen un 20% cada iteración
DET_SCALE_STEP = 1.2;
% Escala que se aplica a la imagen de entrada
factor = 1;
% Movemos a la izquierda o hacia abajo siempre 2 píxeles
step = 2;
% Tamaño de la imagen de entrada
img_width = size(I, 2);
img_height = size(I, 1);
% Tamaño de la imagen de resolución reducida para factor=1
img_red_width = img_width;
img_red_height = img_height;

% Bucle para el cambio de escala
while ((img_red_width >= DET_WIDTH) && (img_red_height >= DET_HEIGHT))
    % Reducir el tamaño de la imagen de entrada.
    Ired = imresize(I, [img_red_height, img_red_width], 'bilinear');
    % Paramos la ventana cuando se alcanzan los píxeles del borde
    stop_height = round((img_red_height - DET_HEIGHT)/step)-1;
    stop_width = round((img_red_width - DET_WIDTH)/step)-1;
    % Recorrer con la ventana deslizante
    for row_index = 1:stop_height
        row = round(row_index * step);
        for col_index=1:stop_width
            col = round(col_index * step);
            Iventana = Ired(row:row+DET_HEIGHT-1, col:col+DET_WIDTH-1);
            % Procesar la región con el clasificador. ¿Está el objeto?
            if (CLASIFICAR_VENTANA(Iventana))
                % Almacenar la posición del objeto y el factor de escala
                Objetos = [Objetos; [row col factor]];
            end
        end
    end
    % Actualizamos el factor de escala
    factor = factor * DET_SCALE_STEP;
    % Actualizamos el nuevo tamaño de la imagen reducida
    img_red_width = round(img_width/factor);
    img_red_height = round(img_height/factor);
end; % while
```

En imágenes estándar de 600 x 450 píxeles, al aplicar una ventana deslizante típica de 20 x 20 píxeles, con un factor de escala del 10 % y con un desplazamiento horizontal y vertical de dos píxeles, se llegan a generar en torno a 1.000.000 de subventanas (imágenes de entrada para el clasificador). Por tanto, hay que tener en cuenta que estamos trabajando con un algoritmo de fuerza bruta con un requerimiento de cálculo muy elevado.

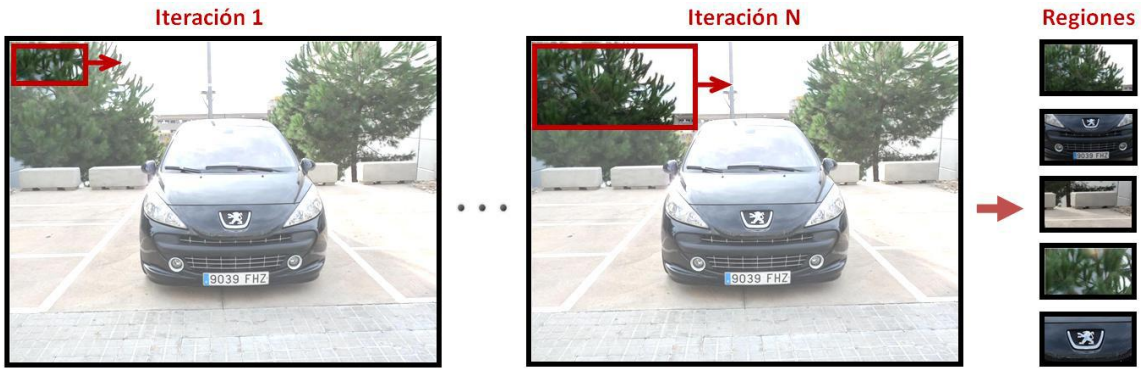


Figura 2.2: Ventana deslizante que recorre la imagen en busca del automóvil.

2.1.2. Extracción de las características

Históricamente, los algoritmos de reconocimiento de objetos dentro de una escena se han caracterizado por su elevado coste computacional debido a que los únicos datos disponibles para localizar un objeto en una imagen eran los valores asociados a la intensidad del color de los píxeles individualmente. Para caracterizar estos objetos, Paul Viola [2] propone utilizar una serie de filtros rectangulares parecidos a los wavelets de Haar para extraer las características más relevantes de cada región de interés generada por la ventana deslizante. La idea de estos **filtros Haar** es determinar mediante un conjunto de máscaras, la existencia de determinadas estructuras de los objetos en la imagen (dependiendo del tipo de filtro Haar es posible analizar estructuras horizontales, verticales, diagonales, etc).

La Figura 2.3 (a) detalla el proceso de extracción de características que se aplica a una región de interés generada. El filtro Haar de la imagen permite comprobar si existe o no un cambio brusco de intensidad entre las regiones que quedan por debajo de la máscara. De esta forma, al calcular la suma de los niveles de gris en cada área del filtro (área blanca y área negra) se puede determinar la existencia de un borde horizontal en el objeto si la diferencia de valores entre ambas regiones es notable.

Las características Haar tienen la ventaja de ser invariantes a la iluminación y a la escala de los objetos de interés, además de ser bastante robustas frente al posible ruido presente en la imagen. Con respecto al tiempo de procesamiento requerido para realizar el cálculo de la suma de los píxeles de cada región del filtro Haar, es imprescindible que sea realmente eficiente para que tenga sentido usar esta técnica. Hay que tener en cuenta que las características Haar se aplican al conjunto de subventanas que genera la ventana deslizante (en torno a 1.000.000), por lo que su procesamiento debe ser muy rápido.

En este punto, se introduce la técnica conocida como **Imagen Integral**, que será la encargada de calcular a partir de cuatro accesos a memoria, la suma de los niveles de gris de cada región definida en el filtro Haar. En la Ecuación 2.1 se detalla el funcionamiento del algoritmo basado en la imagen integral, de forma que para cada pixel de la imagen original (x,y) se calcula el área determinada por la suma de los píxeles que hay encerrados entre el pixel de inicio de la imagen (0,0) y el punto en cuestión (x,y).

$$I(x,y) = \sum_{i=0}^{x}\sum_{j=0}^{y} I(i,j) \tag{2.1}$$

La Figura 2.3 (b) presenta cómo se deben calcular los valores de la imagen integral que nos va a permitir procesar de manera eficiente las regiones blanca y negra que componen un filtro Haar. Para determinar la intensidad de los píxeles de la región $A\widehat{BC}D$, será necesario en primer lugar acceder a cuatro posiciones en la imagen integral "I" para después realizar una serie de operaciones aritméticas básicas sobre las mismas: $(I_A + I_D) - (I_B + I_C)$.

(a) (b)

Figura 2.3: Evaluación de un filtro Haar mediante la imagen integral.

Al realizar el entrenamiento de cada uno de los clasificadores binarios encargados de localizar los automóviles y las matrículas, se generan una serie de resultados que detallan cuáles son las características más discriminantes a la hora de describir los objetos de interés. El objetivo es determinar la estructura de los objetos a través de los filtros Haar correspondientes. Finalmente, cabe destacar el hecho de que el clasificador de automóviles especifica como detalles más importantes los faros y la matrícula de cada vehículo posiblemente por su contraste con el frontal, mientras que el clasificador de matrículas lo que trata es de localizar una ristra de caracteres que destacan sobre el fondo.

2.1.3. Cascada de clasificadores con AdaBoost

El enfoque "Cascade of Classifiers" hace referencia a la estructura del clasificador que propone Viola - Jones [3]. La técnica trata de diseñar un clasificador binario que permita decidir si un objeto cualquiera se encuentra o no en las regiones de interés que genera la ventana deslizante del Apartado 2.1.1. Para ello, el objetivo del algoritmo es construir un clasificador robusto a partir de varios clasificadores denominados "fuertes" colocados secuencialmente para formar las etapas de la cascada (*num_stages*).

En la Figura 2.4 se esquematiza el algoritmo asociado al sistema de localización de objetos que se pretende desarrollar. La idea del algoritmo es ejecutar cada clasificador entrenado previamente, con las regiones que genera la ventana deslizante, para determinar si se trata o no de un objeto de interés. La primera etapa comienza recibiendo el conjunto entero de subventanas generado y analiza cuánto se parecen a las características extraídas de los objetos durante el entrenamiento de la cascada. Las ventanas en las que se pueda apreciar que no está contenido el objeto, dejarán de formar parte del clasificador en las primeras iteraciones. De esta forma, a medida que avancemos secuencialmente por las etapas, únicamente permanecerán en la cascada de clasificadores aquellas regiones clasificadas como positivas en las etapas previas (cada etapa de la cascada aumenta su complejidad a medida que se avanza en la misma).

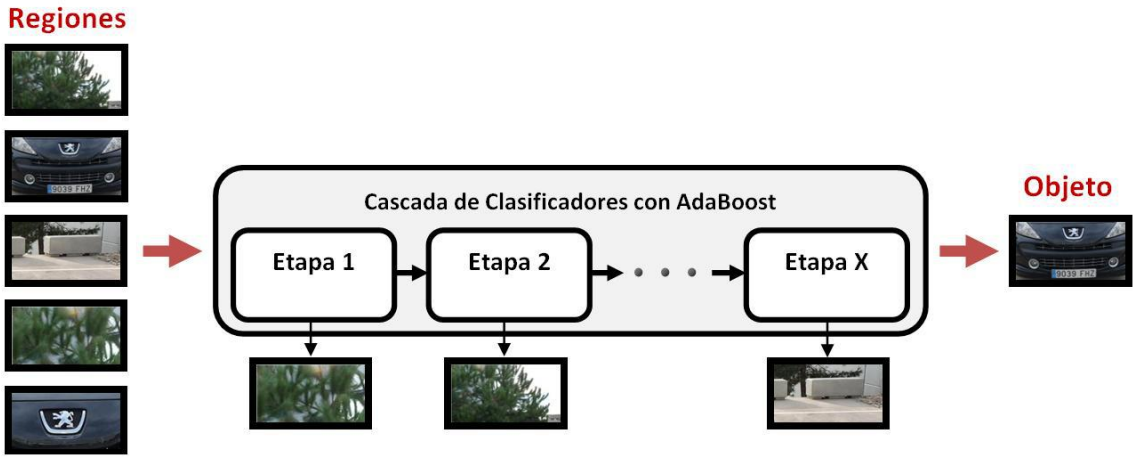


Figura 2.4: Procesamiento de las subventanas mediante la cascada de clasificadores.

La gran ventaja de la técnica que supone la cascada de clasificadores es la reducción del tiempo de procesamiento que se emplea para estudiar las características extraídas de las subventanas. Esto es debido a que la inmensa mayoría de ventanas se descartan en las primeras iteraciones de la cascada (la mayoría son verdaderos negativos) y sólo las imágenes realmente parecidas al objeto permanecen hasta las últimas etapas (permanecen los falsos positivos).

Con respecto a los clasificadores de cada una de las etapas, la idea es construir un clasificador fuerte como combinación lineal de una secuencia de clasificadores de carácter más débil (clasificadores con una tasa de acierto ligeramente superior al 50 % que proporciona un clasificador completamente aleatorio). Además, esta estructura de los clasificadores en cascada permite que se pueda emplear la técnica conocida como **AdaBoost** (Adaptative Boosting) propuesta por Y. Freund [4] en 1996. Cada etapa de la cascada de clasificadores será un clasificador fuerte entrenado mediante AdaBoost. Esto quiere decir que, cada clasificador va a aprender a clasificar con los casos clasificados erróneamente en etapas anteriores (aumentará la complejidad de los clasificadores al avanzar en la cascada).

La Figura 2.5 presenta el esquema que detalla el cálculo de una frontera de decisión para los clasificadores entrenados mediante Boosting que forman cada etapa. La idea del ejemplo es ilustrar cómo es posible distinguir entre una serie de logotipos y frutas con forma de manzana, a partir de clasificadores débiles (líneas paralelas a los ejes) combinados para formar el clasificador binario (logotipo / fruta) deseado. Experimentalmente se demuestra como al combinar las hipótesis generadas por los clasificadores débiles, la frontera de decisión construida separa con éxito las clases en cuestión.

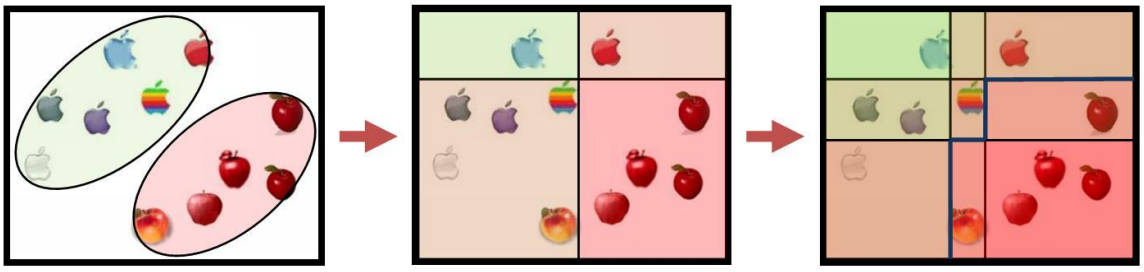


Figura 2.5: Cálculo de la frontera de decisión con las hipótesis de AdaBoost.

Para implementar cada clasificador débil $h_i(x)$ se va a seleccionar, como método de aprendizaje, un modelo basado en los árboles de decisión propuestos por J. R. Quinlan [5] en 1985. Los árboles de decisión establecidos serán los encargados de procesar las características extraídas de la imagen mediante los filtros Haar. De esta forma, para entrenar los clasificadores AdaBoost de cada etapa se van a asignar a todos los clasificadores débiles que la forman $h_i(x)$, unos determinados pesos α_i dependiendo de la tasa de error ϵ_i asociada (Ecuación 2.2).

$$\alpha_i = \log(\frac{1-\epsilon_i}{\epsilon_i}) \tag{2.2}$$

La función $H_T(x)$ de la Ecuación 2.3 permite garantizar la posible presencia de un objeto de interés en la región a procesar. La idea para determinar el resultado del clasificador AdaBoost en una etapa de la cascada concreta, será evaluar los resultados que generan cada uno de los T clasificadores débiles $h_i(x)$ de forma que, si más de la mitad predicen la presencia del objeto el resultado final del clasificador será positivo (localiza el automóvil o la matrícula). Si por el contrario la mayoría determina que no es de interés, el resultado será negativo.

$$H_T(x) = \begin{cases} \textit{positiva} & \text{si } \sum_{i=1}^{T} \alpha_i h_i(x) \geq \frac{1}{2} \sum_{i=1}^{T} \alpha_i \\ \textit{negativa} & \text{en otro caso} \end{cases} \tag{2.3}$$

Finalmente, las subventanas que se hayan clasificado como positivas durante el entrenamiento de esta etapa T (posibles objetos de interés) pasarán a clasificarse nuevamente en la siguiente etapa de la cascada T+1 (así hasta completar todas las etapas de la cascada de clasificadores en caso de que los resultados sean positivos).

2.2. Lectura de los caracteres de la matrícula

Desde hace unos años, los países de la Unión Europea han establecido un formato en común para facilitar el reconocimiento de todas las matrículas de los automóviles que circulan por cada país. A la izquierda se debe detallar con una banda vertical azul el país donde se ha matriculado el vehículo y en el resto de la placa se incluirán los caracteres correspondientes en negro sobre un fondo de color blanco para facilitar su lectura. Desde el año 2000 en España, las matrículas constan de cuatro cifras y tres letras consonantes (exceptuando Ñ y Q) que forman un total de siete caracteres "0000-BBB". Anteriormente, las matrículas españolas se formaban a partir de una o dos letras que indicaban la provincia del vehículo, cuatro cifras más de identificación y finalmente otras dos letras (exceptuando Ñ, R y Q) que hacían un total de hasta ocho caracteres "XX-0000-AA".

Llegado este momento, el sistema ya es capaz de localizar la posición de las matrículas que están asociadas a ciertos automóviles colocados de frente en una escena introducida por imagen o vídeo. El objetivo ahora es construir otro clasificador que se encargue de identificar cada uno de los caracteres que aparecen en las matrículas de los vehículos detectados (10 números y 20 letras posibles). Para reconocer los caracteres, en primer lugar se van a extraer por separado los caracteres de la matrícula mediante la técnica de "Regiones Extremas Máximamente Estables". Acto seguido se clasificará cada caracter respectivamente con la ayuda de la técnica "Random Forest" que vamos a detallar a continuación.

2.2.1. Maximally Stable Extremal Regions (MSER)

Para extraer las características de los caracteres, la idea fundamental a tener en cuenta es que los caracteres negros de la matrícula son elementos altamente distinguibles por su contraste con el fondo blanco de la placa. Esta es la idea de la técnica **MSER** propuesta por J. Matas en [6]. Para comprender mejor el concepto de región extrema máximamente estable propuesto, se van a adjuntar una serie de imágenes umbralizadas en la Figura 2.6 donde se puede observar cómo cambia la escena a medida que se aumenta un umbral que se va aplicando sobre la imagen original (umbralizado creciente).

Figura 2.6: Reconocer MSER aplicando diferentes umbrales en una imagen.

El algoritmo MSER analiza la imagen original realizando una serie de umbralizados con todos los umbrales posibles (de 0 a 255 niveles de gris) de forma que genera un conjunto de imágenes donde se colorean: de color negro los píxeles por debajo del umbral y de color blanco los de por encima. La región extrema máximamente estable hace referencia a cada zona o vecindad de la imagen que permanece sin cambios al aplicar una serie de umbrales consecutivos. Para el caso anterior, partimos de la imagen inicial totalmente blanca (umbral = 0), de forma que al aumentar el valor del umbral se empezará a percibir la silueta del caracter sobre el fondo negro. Esta región con el caracter permanecerá aparentemente estable hasta que el umbral alcance el nivel de gris de los píxeles del caracter y la imagen se perciba totalmente negra (umbral = 255).

Una vez identificados los caracteres de la matrícula, la idea es crear un vector de características con cada región MSER extraída para poder clasificar correctamente cada caracter por separado en un futuro. En la Figura 2.7 se analiza cómo funciona el procesamiento de las matrículas con MSER, cuyo resultado desemboca en las imágenes binarizadas con los caracteres a reconocer. Para generar estas imágenes, es imprescindible realizar una serie de transformaciones geométricas previas sobre las regiones que mejorarán las tasas de reconocimiento del clasificador a construir.

- *Ampliar la región.* En primer lugar, la idea es ajustar mediante un rectángulo el conjunto de píxeles que definen el caracter a identificar (bounding box). Este rectángulo ajustado al contorno del caracter marca el contorno de la imagen que forma el vector de características. No obstante, aunque esta región contiene el caracter completo, en ocasiones es difícil reconocer de qué caracter se trata al estar tan pegado al borde de la imagen. Al aumentar ligeramente el alto y el ancho del rectángulo el clasificador mejorará la tasa de reconocimiento (aumentar un 15 % el ancho y el alto respectivamente).

- *Tamaño estándar.* Es importante establecer un ancho y alto común para normalizar todas las imágenes con los caracteres a reconocer. Este tamaño debe ser exactamente el mismo tanto para la etapa de entrenamiento del clasificador como para las

pruebas. De esta forma, se ha decidido que 12 píxeles de ancho y 20 de alto son suficientes para determinar con precisión de que caracter se trata.

- *Binarización.* El último aspecto a tratar es la ecualización del histograma y posterior umbralización de la imagen. El objetivo es seleccionar un umbral relativamente bajo, de forma que solo los pixeles con tonalidad realmente oscura formen parte del caracter (de los 256 umbrales posibles se establece el umbral a 100).

Figura 2.7: Extracción de los caracteres de la matrícula usando MSER.

2.2.2. Clasificador Random Forest

El objetivo es entrenar un clasificador que permita reconocer los caracteres extraídos de la matrícula anteriormente. La aplicación a desarrollar debe ser capaz de identificar todos los dígitos y letras que se manejan en el sistema de matriculación vigente en España (números del 0 al 9 y letras consonantes excepto Ñ y Q).

En este punto se introduce el clasificador Random Forest (RF) desarrollado por L. Breiman [7], que trata de construir un clasificador multiclase consistente como combinación de varios árboles de decisión especiales conocidos como **Random Trees** (RT), similares a los mencionados en el Apartado 2.1.3 con la cascada de clasificadores. El resultado final del clasificador de caracteres se calcula a partir de las predicciones realizadas por el conjunto de árboles de decisión.

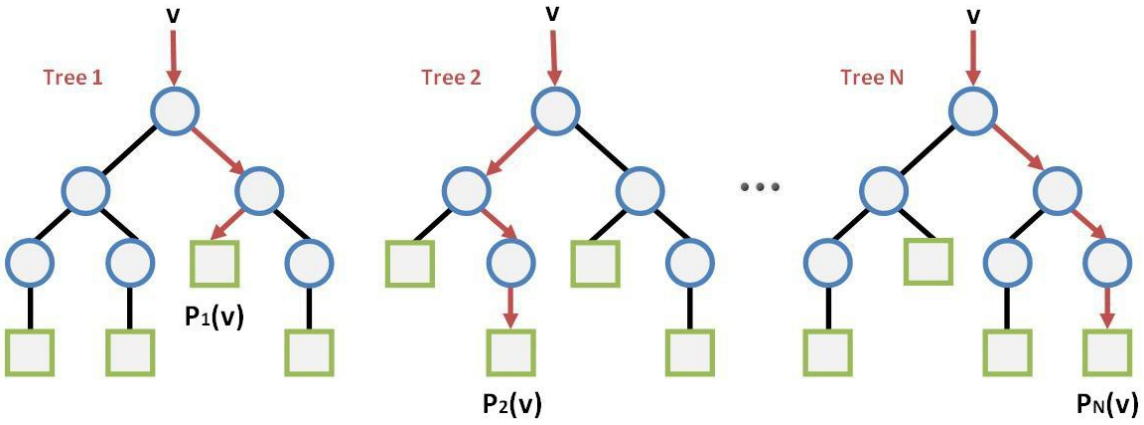


Figura 2.8: Predicción de los RT generados a partir del vector de características "v".

La principal característica de los Random Trees se encuentra en el factor aleatorio que introducen estos árboles de decisión a la hora de seleccionar las muestras de entrenamiento de cada RT. El algoritmo permite a cada árbol seleccionar aleatoriamente y con reemplazamiento su propio subconjunto de entrenamiento de entre todos los parámetros disponibles (todos los dígitos y letras) de forma que cada árbol reconocerá los caracteres de una manera diferente. Además se observa como la estructura de los árboles que componen el Random Forest varía en función de las clases seleccionadas al azar.

Con respecto al funcionamiento del clasificador el algoritmo es bastante simple. El sistema recibirá en primer lugar el vector de características "v" asociado a la imagen con el caracter a reconocer. Este vector pasará a clasificarse en cada uno de los árboles que forman el RF. De esta forma cada árbol de decisión generará una predicción $P_i(v)$ que especificará a qué clase pertenece el vector de características. Por último, la clase que obtenga más votos de entre todos los árboles, determinará el resultado final del clasificador $P(v)$. Para calcular cuál es la clase que más votos recibe se calcula el promedio de las predicciones con la Ecuación 2.4.

La idea del algoritmo es calcular las probabilidades generadas por los N árboles del sistema y verificar cuál de las clases obtiene un mayor porcentaje final. Por ejemplo, con un clasificador de dos árboles y las siguientes predicciones: $P_1(v) = [\mathbf{0} = 0{,}1; \mathbf{8} = 0{,}7; \mathbf{B} = 0{,}2]$ y $P_2(v) = [\mathbf{0} = 0{,}3; \mathbf{8} = 0{,}4; \mathbf{B} = 0{,}3]$ el resultado del clasificador sería el dígito "8", ya que es el valor más votado de los posibles $P(v) = [\mathbf{0} = 0{,}2; \mathbf{8} = 0{,}55; \mathbf{B} = 0{,}25]$.

$$P(v) = \frac{\sum_{i=1}^{N} P_i(v)}{N} \tag{2.4}$$

La técnica de remuestreo del Random Forest está basada en el algoritmo conocido como **Bootstrap** propuesto por Bradley Efron en [8]. Esta herramienta aporta a los RT una serie de ventajas que determinan el éxito del clasificador:

- A la hora de entrenar los árboles de decisión partimos de la muestra con todos los parámetros (dígitos y letras), pero sin embargo cada RT genera su propio subconjunto de entrenamiento para construir su clasificador particular.
- Cada submuestra de entrenamiento debe tener el mismo tamaño de la muestra inicial, aunque los datos se seleccionan aleatoriamente con remplazamiento. Esto quiere decir que algunas clases se podrían repetir y otras ni siquiera aparecer en el subconjunto de ese árbol determinado.
- La idea fundamental es que se puede estimar correctamente el caracter en cuestión a partir de los resultados de los RT, sabiendo que no todos los árboles están entrenados para reconocer todos los caracteres (habrá árboles que no entrenen la clase en cuestión, y otros árboles que sin embargo repitan la clase varias veces en el subconjunto de entrenamiento).
- La gran ventaja que proporciona el algoritmo Bootstrap es poder disminuir la varianza de la estimación mientras se mantiene el sesgo estadístico anterior.
- Rich Caruana realizó un estudio en 2008 [9] para evaluar una serie de algoritmos de aprendizaje supervisados que demuestran la efectividad de la técnica basada en RT.

2.3. Reconocer la marca y el modelo del vehículo

La última funcionalidad que se va a aplicar al sistema trata de reconocer la marca y el modelo de los automóviles localizados en la escena previamente. La idea es construir un nuevo clasificador para identificar la marca y el modelo de los vehículos a partir de las características que se pueden extraer de la región con el frontal del automóvil: Alfa Romeo, Audi A4, Audi A6, BMW Serie 1, ...

En el PFC presentado en [12] se detallan en profundidad los algoritmos de reconocimiento de automóviles que se analizan en este apartado. El objetivo es ajustar los mecanismos empleados en el proyecto que se está desarrollando, sabiendo que ya no es necesario marcar la posición de las esquinas de la matrícula porque el sistema la localiza automáticamente. Para corregir las imágenes se establece un tamaño estándar de 256 x 92 píxeles para extraer el mismo número de características de todas las imágenes (de cada punto de interés se extraen vectores de características de 64 componentes). A continuación se analiza el funcionamiento del descriptor SURF implementado así como el algoritmo desarrollado para construir el clasificador de marca y modelo.

2.3.1. Rectificar la imagen

Es necesario establecer un mismo tamaño para la región de interés que se ajusta al frontal del automóvil localizado, de tal forma que al aplicar el descriptor sobre estas, se coloque siempre el automóvil en la misma posición. Siguiendo la estructura de los experimentos propuestos en [14] se selecciona un tamaño de 256 x 92 para centrar la matrícula. Para evitar además los posibles problemas de orientación del vehículo, realizamos una transformación geométrica que permita acercar y orientar la imagen de los frontales similares a los de la Figura 2.9. Finalmente eliminamos el contenido de las matrículas para que no afecte a la clasificación del automóvil en cuestión.

(a) (b)

Figura 2.9: Ejemplo de imagen rectificada con la matrícula en blanco.

2.3.2. Speeded Up Robust Features (SURF)

Los descriptores son las herramientas que permiten analizar las características de cada una de las imágenes rectificadas a 256 x 92 con las que trabaja la aplicación, para poder diferenciar los distintos modelos de los automóviles a partir de las características del frontal. En 1999, David Lowe desarrolló un potente detector y descriptor de características conocido como *Scale Invariant Feature Transform* (SIFT) [15]. A diferencia de otros detectores de puntos de interés, SIFT es muy poco sensible a los cambios en la iluminación y el tamaño de la imagen así como a posibles rotaciones de la misma. El principal problema que plantea es que el algoritmo está patentado y por tanto no se puede utilizar sin permiso en aplicaciones comerciales.

En 2006, H. Bay, T. Tuytelaars y L. Van Gool presentaron un descriptor equivalente conocido como *Speeded Up Robust Features* (SURF) [16]. La principal ventaja de SURF es que además de no estar patentado, se ejecuta más rápido que SIFT sin pérdida de rendimiento aparente. Llegados a este punto, podemos detallar el funcionamiento del algoritmo que permite extraer las características de la imagen con el frontal del automóvil:

- *Localizar los puntos de interés de la imagen.* SURF proporciona un detector de puntos de interés en su implementación. Sin embargo al realizar los experimentos del sistema, los resultados no son buenos debido a que se pierden varias zonas del frontal que el detector no considera de interés. Para detectar características de todo el frontal, se va a aplicar una rejilla sobre la imagen similar a la de la Figura 2.10, de forma que cada celda actuará como región de interés para el sistema. Así le indicamos al descriptor manualmente de que zonas debe extraer las características.

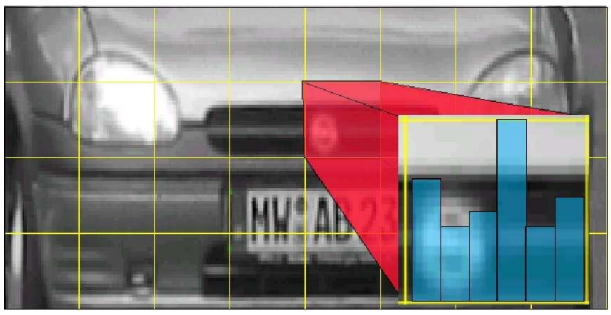


Figura 2.10: Extracción de características de una imagen.

- *Descriptor de características.* SURF divide cada celda de la rejilla en 16 subregiones que a su vez se subdividen en otras 4 subregiones más pequeñas formando un total de 16 x 4 = 64 componentes por cada región de interés. Cada casilla del descriptor calcula las orientaciones de los gradientes en cada componente basándose en las propiedades de la imagen a analizar. El resultado obtenido es un vector de 64 componentes por cada región de interés de la rejilla que caracteriza el modelo del automóvil.

2.3.3. Clasificador de marca y modelo

Tras obtener las características de las imágenes rectificadas, llega el momento de almacenar los vectores en una estructura de datos separándolos según sean de un modelo u otro, con el objetivo de clasificar los automóviles más tarde por su marca y modelo (etapa de entrenamiento del clasificador). La idea del algoritmo propuesto por M. Irani [17] es calcular las distancias entre la imagen a identificar y los modelos de la base de datos, de tal forma que la clase que devuelva la menor distancia de todas será la asignada a la imagen.

Como se especifica en el Apartado 2.3.2, los descriptores "q_d" que devuelve SURF son secuencias de 64 componentes para cada región de interés que se describe, es decir que para la imagen con el frontal del automóvil, se generan tantas secuencias de 64 componentes como celdas tenga la rejilla que se ajusta sobre la imagen. La Ecuación 2.5 permite calcular la distancia que existe entre la imagen a identificar y los modelos de la base de datos, siendo "n" el número de celdas de la rejilla que le pasamos a la imagen sin clasificar "i", y siendo "m" el modelo de automóvil procesado.

$$dist[m,i] = \sum_{d=1}^{n} ||q_d - NN_{m,i}(q_d)||^2 \tag{2.5}$$

A continuación se detalla el funcionamiento de algoritmo que permite clasificar la marca y el modelo de un vehículo:

1. Extraer los vectores de 64 componentes con las características (q_1, q_2, ..., q_n) de cada celda de la rejilla colocada sobre la imagen de entrada "i".

2. Para cada modelo "m" de la base de datos se calculan las distancias que hay entre los descriptores de la imagen sin clasificar y los del modelo en cuestión. Realizamos los siguientes pasos:

   2.1. Calcular el vecino más próximo $NN_{m,i}(q_d)$ que hay dentro de este modelo "m" para cada vector de características q_1, q_2, ..., q_n.

   2.2. Calcular las distancias que hay entre los vectores q_1, q_2, ..., q_n de la imagen a reconocer y sus correspondientes vecinos $NN_{m,i}(q_1)$, $NN_{m,i}(q_2)$, ..., $NN_{m,i}(q_n)$.

   2.3. Realizar el sumatorio de todas estas distancias y guardar el resultado total en la variable $dist[m, i]$, separando los resultados por cada clase "m".

3. Recorrer las variables con las distancias totales generadas para cada modelo de la base de datos ($dist[\text{“}Opel'', i]$, $dist[\text{“}Renault'', i]$, ...,$dist[\text{“}BMW'', i]$). La variable que tenga asignada la menor distancia será la que usaremos para clasificar la imagen de entrada.

Capítulo 3

Resultados experimentales

En este capítulo se realizarán los experimentos que nos permitirán establecer si se alcanza o no el objetivo del PFC, que no es otro que localizar a partir de una imagen o vídeo de entrada, cada uno de los automóviles vistos de frente en la escena, para identificar los caracteres asociados a la matrícula de estos vehículos así como su marca y modelo. En cada uno de los siguientes apartados se analizan los diferentes parámetros que determinan los resultados obtenidos en los experimentos: construcción de la base de datos, tamaño de las imágenes, atributos del clasificador, etc. La idea es describir con exactitud las herramientas que se necesiten para implementar los algoritmos detallados en el Capítulo 2.

3.1. Detectar el frontal del automóvil

La aplicación debe ser capaz de localizar la posición exacta de los frontales de los automóviles en la escena para posteriormente poder localizar cada matrícula y reconocer el modelo de los vehículos. Para construir el clasificador biclase (coche / no coche) se van a realizar una serie de experimentos similares a los que propone Naotoshi Seo [10] en su sistema de localización de caras de personas en tiempo real. La idea es utilizar las mismas herramientas que proporciona OpenCV 2.2 [21] para conseguir unos resultados de clasificación similares.

3.1.1. Construcción de la base de datos

- *Imágenes Positivas.* Hace referencia a todas las imágenes en las que se visualiza al menos un automóvil de frente. Son imágenes necesarias, tanto a la hora de entrenar el clasificador (training), como a la hora de realizar las pruebas que determinen la tasa de acierto del sistema (testing).

  Con respecto a la fase de entrenamiento, se han coleccionado 1371 imágenes de automóviles (en su mayoría de modelos reconocidos en España). Además se han seleccionado de forma que cada una de las imágenes positivas contiene máximo un vehículo en la escena. No obstante, como no parecían bastantes imágenes (siempre en comparación con el proyecto de Naotoshi Seo [10]), se ha tomado la decisión de aplicar una serie de transformaciones lineales sobre las imágenes (rotación positiva, negativa y efecto espejo) de forma que se consigue cuadruplicar la base de datos, alcanzando la cifra de **5484 imágenes positivas distintas para el entrenamiento**.

Para probar el clasificador coche / no coche se van a emplear un total de 510 imágenes, distintas a las que se han utilizado para entrenar el clasificador, aunque sin embargo ahora sí se pueden encontrar algunas imágenes con más de un automóvil en la escena. Disponemos de **540 automóviles para realizar las pruebas**.

- *Imágenes Negativas.* Término utilizado para aquellas imágenes que no contienen el objeto que se está intentando localizar. Son ímágenes en las que se debe evitar que aparezca algún automóvil visto de frente en la escena, ya que estas serán las imágenes que le indicarán al clasificador durante el entrenamiento que objetos no son de interés. El objetivo por tanto, es coleccionar imágenes de carreteras vacías, señales de tráfico y paisajes arbitrarios por los que puedan circular vehículos, así como ciertos objetos que puedan confundise con el frontal de un coche, con la idea de que el clasificador pueda diferenciar el vehículo rápidamente durante el entrenamiento. En la Figura 3.1 se pueden encontrar algunos ejemplos de imágenes negativas de la base de datos.

A la hora de evaluar cuántas imágenes negativas se necesitan para generar un clasificador robusto, se van a realizar una serie de pruebas en el Apartado 3.1.2 para determinar una cantidad concreta. De esta forma, se puede observar como **306 imágenes negativas distintas** resultan suficientes para conseguir los resultados esperados. Hay que tener en cuenta que el tamaño de las imágenes negativas ronda los 800 x 600 píxeles de media, por lo que se pueden obtener multitud de subventanas para entrenar la cascada de clasificadores.

(a) (b)

Figura 3.1: Imágenes negativas asociadas a la detección de automóviles.

3.1.2. Experimentos de clasificación

Una vez construida la base de datos de imágenes llega la hora de comenzar con los experimentos para generar el clasificador. A continuación se detalla el funcionamiento de las herramientas que proporciona la biblioteca OpenCV 2.2 para construir el clasificador binario y evaluar su rendimiento.

- *Createsamples.* Es el programa encargado de **construir el fichero vec** que contendrá la información de los automóviles que se visualicen en las imágenes positivas

(ubicación de los vehículos en la imagen). La herramienta puede ejecutarse directamente sobre un terminal configurando previamente los parámetros que se detallan a continuación.

```
opencv_createsamples -info positiv.dat -vec positiv.vec -w 35 -h 15
```

En primer lugar, se debe construir un fichero con extensión dat donde se especifique ordenadamente la información relacionada con los vehículos de las imágenes positivas. Este fichero almacenará la ruta de acceso relativa de cada imagen, el número de automóviles de frente, y las coordenadas que determinan la posición de los mismos.

```
Positivas/aromeo/img0000.jpg 1 91 218 474 161
Positivas/aromeo/img0001.jpg 1 15 167 483 191
Positivas/aromeo/img0002.jpg 1 72 191 405 137
                    ...
```

El segundo parámetro establece el nombre del fichero vec que la herramienta debe generar, y los dos últimos parámetros de entrada especifican el tamaño mínimo de los automóviles a localizar. Es interesante asignar un valor relativamente bajo, sabiendo que los vehículos de menor tamaño no serán reconocidos por el clasificador. Hay que tener en cuenta que los valores de ancho y alto seleccionados, establecen el tamaño inicial de la ventana deslizante (mencionada en el Apartado 2.1.1) que recorrerá las imágenes en busca de automóviles. La relación de aspecto establecida para la ventana es proporcional al ancho y alto que existe en los frontales de los automóviles, es decir, imágenes generalmente más anchas que altas.

- *Haartraining.* Herramienta que permite **generar el clasificador haarcascade** en formato xml. Realiza el entrenamiento del clasificador a partir de las imágenes positivas y negativas de la base de datos. Posee una serie de argumentos que determinan la tasa de acierto que alcanzará en el sistema.

```
opencv_haartraining -data haarcascade -vec positiv.vec -bg neg.dat
-nstages 31 -nsplits 4 -minhitrate 0.999 -maxfalsealarm 0.5
-npos 5484 -nneg 306 -w 35 -h 15 -nonsym -mode ALL -mem 1024
```

Los dos primeros parámetros establecen la ruta de salida del programa y el fichero vec con los datos de entrada para el entrenamiento del sistema. El siguiente paso es crear otro fichero dat para las 306 imágenes negativas. No obstante, únicamente será necesario especificar la ruta relativa de acceso a cada imagen.

```
Negativas/neg0000.jpg
Negativas/neg0001.jpg
         ...
```

El resto de parámetros determinan el rendimiento del clasificador: *nstages* especifica el número de etapas que forma la cascada de clasificadores implementada en el Apartado 2.1.3; *nsplits* permite configurar la estructura de los nodos en los árboles de decisión de cada clasificador débil de la cascada; *minhitrate* establece la tasa mínima de acierto exigida para los X clasificadores de la cascada, de forma que la

tasa de acierto mínima exigida es de $(0{,}999)^X \approx 0{,}95$; análogamente *maxfalsealarm* especifica la tasa de falsos positivos máxima que es de $(0{,}5)^X \approx 10^{-8}$.

Por último, *npos* y *nneg* establecen el número de imágenes positivas y negativas para entrenar; *nonsym* es la etiqueta que indica que no tiene por qué existir simetría en el objeto; *mode* selecciona todos los tipos de filtros Haar para extraer características (Apartado 2.1.2) y *mem* permite reservar memoria para realizar el cómputo.

- *Performance.* Función que permite **realizar las pruebas** que determinarán el rendimiento del clasificador generado.

  ```
  opencv_performance -data haarcascade -w 35 -h 15 -info test.dat
  ```

  Respecto a los parámetros, la herramienta debe recibir el clasificador haarcascade generado así como el tamaño mínimo de los automóviles a localizar. Además, para acceder a las imágenes de testing se especifican las rutas relativas de las imágenes en un nuevo fichero dat con el formato habitual.

  ```
  Testing/test0000.jpg 1 46 187 442 163
  Testing/test0001.jpg 1 8 167 495 187
                   ...
  ```

  De esta forma la herramienta "performance" contabilizará un acierto cuando la posición detectada por el clasificador coincida con la ubicación apuntada en el fichero. Para calcular la tasa de aciertos se va a usar la fórmula de la Ecuación 3.1.

$$Tasa\ aciertos\,\% = \frac{Num.\ de\ aciertos}{Num.\ total\ de\ casos} * 100 \tag{3.1}$$

A la hora de generar el clasificador biclase es importante controlar el número de etapas seleccionadas para la cascada, así como la cantidad de imágenes de entrenamiento y el tamaño de las mismas. Al seleccionar valores altos para estos parámetros se dispara el tiempo de procesamiento empleado para generar el clasificador, llegando a alargarse a más de 10 horas. Para encontrar los valores más adecuados se van a realizar una serie de experimentos, que permitirán ajustar los parámetros para obtener los mejores resultados posibles.

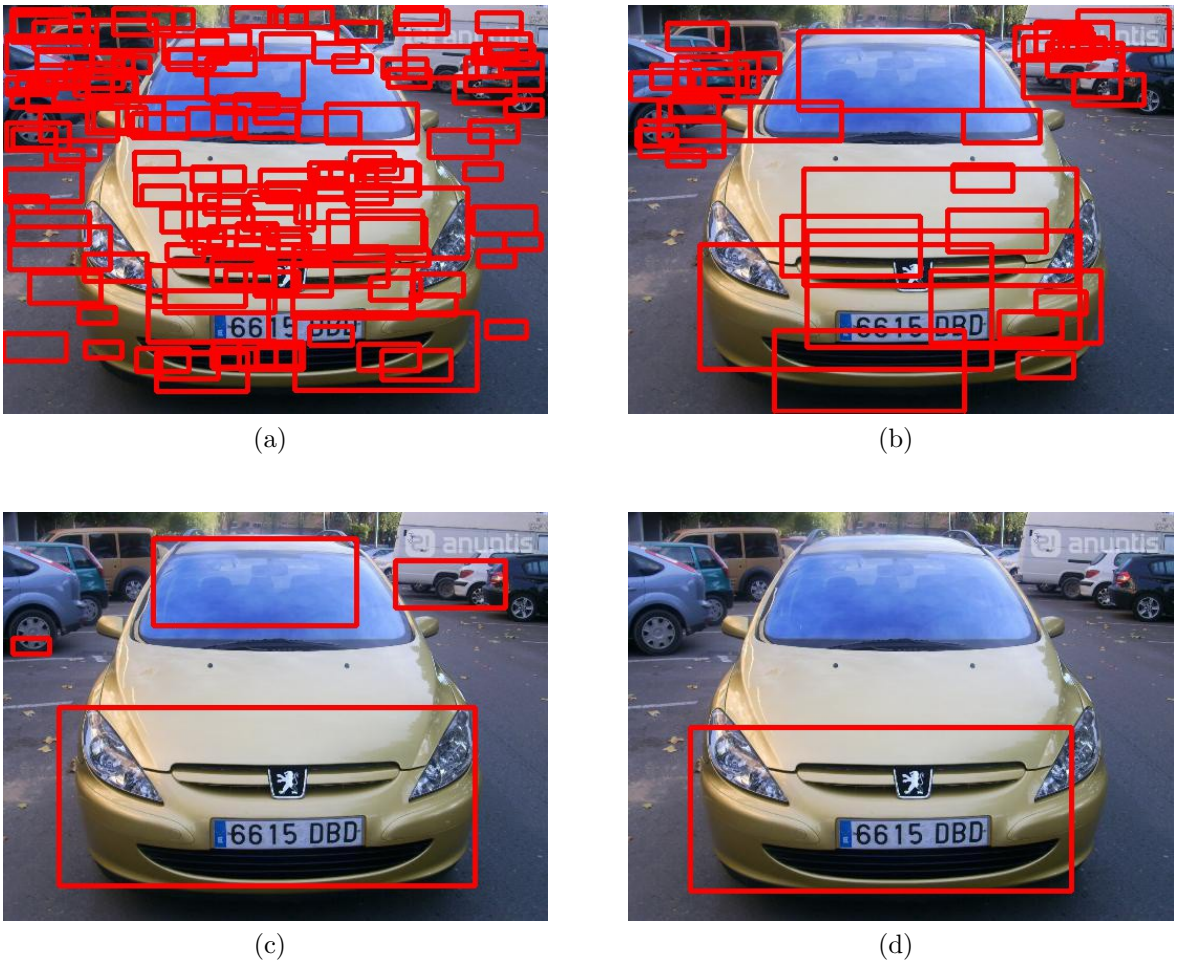
En el Cuadro 3.1 se detalla cómo afecta por ejemplo el parámetro *nneg* (número de imágenes negativas seleccionadas) al rendimiento del clasificador. Como era de esperar, seleccionando todas las imágenes negativas disponibles se obtiene la mejor tasa de acierto. El clasificador de frontales de automóviles construido, es capaz de localizar correctamente los automóviles en 507 de los 540 casos disponibles (**94 % aciertos**).

Imág. Negativas	**Detectados**	**No detectados**	**Falsos Positivos**
50	84	456	66409
100	434	106	16973
200	504	36	1336
300	505	35	658
306	507	33	271

Cuadro 3.1: Resultados del clasificador de frontales de automóviles.

Por último es importante tener en cuenta también el número de detecciones incorrectas del clasificador (falsos positivos). El objetivo es evitar los sistemas que reconocen erróneamente cualquier objeto de la escena como un frontal de un vehículo.

La Figura 3.2 adjunta algunas imágenes donde se puede observar cómo disminuye el número de falsos positivos detectados en la escena según se aumenta el número de imágenes negativas en el entrenamiento. Hay que tener en cuenta que en imágenes con 600 x 450 píxeles, al aplicar una ventana deslizante de 35 x 15 píxeles que aumenta un 20 % cada iteración, se llegan a generar en torno a 750.000 regiones de interés a clasificar. Experimentalmente llegamos a reducir el número de falsos positivos a 271 para las 510 imágenes de test, es decir que cada imagen produce menos de un falso positivo de media (aproximadamente un falso positivo cada 1.500.000 subventanas).



(a) (b)

(c) (d)

Figura 3.2: Resultados de clasificación en función del número de imágenes usadas.

3.2. Localizar la matrícula

La aplicación debe ser capaz de detectar la matrícula que tienen asociada los automóviles localizados en la escena anteriormente. El objetivo ahora es construir un clasificador binario (matrícula / no matrícula) que determine la ubicación de la matrícula en el frontal del vehículo sin perder de antemano ninguno de los caracteres de los extremos. A la hora

de localizar la matrícula, el algoritmo recorrerá únicamente la región de interés generada en el Apartado 3.1, de tal forma que se evita tener que recorrer de nuevo la imagen entera con la ventana deslizante (evitar perder tiempo procesando zonas que no son de interés para la aplicación).

3.2.1. Construcción de la base de datos

- *Imágenes Positivas.* Imágenes que contienen al menos una matrícula en la escena. En su mayoría, las matrículas pertenecen a automóviles españoles, de forma que el formato de las mismas no suele cambiar: matrículas de fondo blanco con caracteres de color negro mate, de 52 centímetros de ancho y 11 centímetros de alto. Las imágenes positivas son imprescindibles tanto a la hora de entrenar el clasificador (training), como para realizar las pruebas que determinan el rendimiento (testing).

  A la hora de evaluar el clasificador la idea es coleccionar imágenes con matrículas de todo tipo de vehículos (autobuses, camiones, parte trasera de automóviles, ...) así como matrículas sueltas, para evitar que el sistema relacione la posición de la matrícula siempre con el frontal de un automóvil (entrenar un clasificador capaz de reconocer matrículas en cualquier posición de la escena).

  Con respecto a la etapa de entrenamiento, se han coleccionado 758 imágenes en las que se incluye al menos una matrícula de interés para la aplicación. Concretamente, hay 7 imágenes con dos matrículas en su escena de forma que realmente disponemos de 765 matrículas para entrenar el clasificador. La idea ahora es realizar una transformación geométrica sobre la imagen (efecto espejo de la Figura 3.3) para duplicar el tamaño de la base de datos hasta llegar a las **1530 matrículas distintas para el entrenamiento**.

(a) (b)

Figura 3.3: Transformación geométrica asociada a las imágenes positivas.

Para la etapa de testing se van a emplear un total de 142 imágenes positivas completamente diferentes a las utilizadas durante el entrenamiento (1 imagen contiene dos matrículas en su escena). Para comprobar la potencia del clasificador en situaciones en las que el vehículo puede aparecer ladeado, se han seleccionado algunas imágenes en las que el automóvil aparece ligeramente rotado. En definitiva, disponemos de **143 matrículas para realizar las pruebas de localización**.

- *Imágenes Negativas.* Hace referencia a aquellas imágenes donde se incluyen elementos que suelen aparecer en la escena de las matrículas pero que hay que evitar que se confundan con estas. El objetivo es coleccionar todo tipo de imágenes relacionadas con los vehículos como (carreteras, retrovisores, neumáticos, etc).

  Como ya ocurría en el Apartado 3.1.1 se deben realizar una serie de experimentos para determinar el número de imágenes negativas a utilizar. De esta forma, a partir de **152 imágenes negativas distintas** se genera una cantidad de subventanas suficiente para conseguir el rendimiento esperado en el clasificador.

3.2.2. Experimentos de clasificación

Construida la base de datos, el objetivo ahora es generar el clasificador biclase de matrículas con las herramientas de OpenCV 2.2 detalladas anteriormente. En este apartado se establecen los valores de los parámetros que mejores resultados proporcionan.

- *Createsamples.* Es el programa encargado de **construir el fichero vec** que contiene la información de la ubicación de las matrículas en las imágenes.

  ```
  opencv_createsamples -info positiv.dat -vec positiv.vec -w 48 -h 12
  ```

  La configuración de los parámetros es similar a la detallada para el clasificador de frontales de automóviles. El fichero dat mantiene el formato: ruta relativa a la imagen, número de matrículas en la escena, y coordenadas que determinan su posición.

  ```
  Positivas/mat0000.jpg 1 283 352 194 54
  Positivas/mat0001.jpg 1 232 256 130 34
  Positivas/mat0002.jpg 1 187 292 140 36
                    ...
  ```

  No obstante, es importante modificar los valores de alto y ancho seleccionados para la ventana deslizante que recorre la imagen en busca de matrículas (tamaño mínimo de las matrículas a localizar). Para ello, la idea es analizar la relación de aspecto que existe en las matrículas reflejadas en el fichero dat y asignar un valor proporcional relativamente bajo, de forma que el ancho sea al menos cuatro veces mayor que el alto de la matrícula.

- *Haartraining.* Programa encargado de **generar el clasificador haarcascade** capaz de localizar las matrículas. Es necesario configurar los parámetros de acuerdo a las necesidades específicas del sistema.

  ```
  opencv_haartraining -data haarcascade -vec positiv.vec -bg neg.dat
  -nstages 32 -nsplits 4 -minhitrate 0.999 -maxfalsealarm 0.5
  -npos 1530 -nneg 152 -w 48 -h 12 -nonsym -mode ALL -mem 1024
  ```

  Los valores asignados a los atributos de haartraining son similares a los especificados en el Apartado 3.1.2. De hecho únicamente cambia el número de imágenes positivas y negativas que se emplean (*npos* y *nneg*), así como la relación de aspecto asociada a las matrículas.

- *Performance.* Herramienta que permite realizar las **pruebas de clasificación** que determinan la tasa de acierto del sistema a partir de las imágenes positivas de test.

```
opencv_performance -data haarcascade -w 48 -h 12 -info test.dat
```

Es importante ajustar correctamente el valor de estos parámetros para generar el clasificador de matrículas en un tiempo razonable. La idea principal es que los valores asignados estén realmente equilibrados entre sí, de forma que el número de subventanas positivas y negativas sea proporcional.

Finalmente, en el Cuadro 3.2 se estudia cómo influye el número de etapas seleccionadas para la cascada de clasificadores (*nstages*) a la tasa de acierto del sistema. Como era de esperar, a medida que se aumentan el número de etapas, el número de falsos positivos mejora proporcionalmente mientras la tasa de aciertos se mantiene alta, localizando correctamente 135 matrículas de los 143 casos disponibles (**94 % aciertos**). Un detalle interesante a tener en cuenta además es que las matrículas empleadas durante los experimentos no tienen una ubicación determinada en las imágenes (pueden no estar ligadas al automóvil), mientras que al ejecutar el clasificador en nuestra aplicación, solo serán de interés las matrículas ubicadas dentro del frontal de los automóviles localizados previamente, lo cual permitirá mejorar aún más el rendimiento del sistema.

Núm. de etapas	**Detectados**	**No detectados**	**Falsos Positivos**
15	140	3	1367
20	138	5	417
25	137	6	196
30	136	7	102
32	135	8	82

Cuadro 3.2: Resultados del clasificador de matrículas.

No obstante, el objetivo no solo es generar un clasificador capaz de localizar las matrículas siempre, sino además evitar el reconocimiento erróneo de cualquier otro objeto de la escena como si de una matrícula se tratase. En la Figura 3.4 se puede apreciar el resultado de disminuir el número de falsos positivos en las imágenes. Simplemente con ajustar bien el número de imágenes negativas y el número de etapas de la cascada podemos ver como mejora el resultado del clasificador (desaparecen los 7 falsos positivos de la imagen). Además, hay que tener en cuenta que posteriormente el sistema tratará de identificar los caracteres que contiene cada matrícula, por lo que cuantas menos detecciones incorrectas generamos, más tiempo evitamos perder procesando falsos positivos.

En imágenes de 600 x 450 píxeles, al aplicar una ventana deslizante de 48 x 12 píxeles que aumente un 10 % cada iteración, se llegan a generar en torno a 1.250.000 regiones de interés a clasificar en el peor de los casos. Experimentalmente se llega a reducir el número de falsos positivos a 82 para las 142 imágenes de test, es decir que cada imagen produce menos de un falso positivo de media (aproximadamente un falso positivo cada 2.500.000 subventanas). Llegados a este punto, la aplicación sería capaz de localizar en una escena los automóviles vistos de frente y las matrículas asociadas a los mismos.

(a) (b)

Figura 3.4: Resultados de clasificación en función del número de etapas.

3.3. Lectura de los caracteres de la matrícula

El objetivo es identificar los caracteres de las matrículas detectadas previamente, mediante un clasificador multiclase que permita reconocer cualquiera de los 30 caracteres que maneja la aplicación (10 dígitos y 20 letras). Para construir el clasificador la idea es primero generar la base de datos para el entrenamiento y después realizar una serie de experimentos con OpenCV 2.2 similares a los que propone Toby Breckon [11] para determinar el rendimiento de la aplicación.

3.3.1. Construcción de la base de datos

Para generar las imágenes que forman la base de datos extraeremos cada caracter de la matrícula por separado mediante el algoritmo MSER propuesto en el Apartado 2.2.1. Posteriormente se procesarán las imágenes de cada uno de los caracteres para extraer sus características. Las imágenes de la base de datos de matrículas facilitan la tarea de encontrar hasta 150 imágenes para cada caracter, es decir 4500 imágenes en total para las 30 clases de la Figura 3.5.

Figura 3.5: Listado con las 30 clases de caracteres disponibles.

La biblioteca OpenCV 2.2 proporciona un método sencillo para extraer la región MSER de cada caracter de la matrícula del automóvil. Para que funcione correctamente se deben configurar una serie de parámetros: *img* en la imagen de entrada que contiene la matrícula localizada anteriormente; *mask* es un parámetro que permite aplicar una máscara determinada sobre la imagen; *contours* es la variable de salida que almacena los MSER extraídos de la imagen; *storage* reserva memoria para procesar la imagen y *params* es el atributo que determina el rendimiento del clasificador.

```
cvExtractMSER(img, NULL, contours, storage, params);
```

La clase CvMSERParams permite configurar las características exigidas a todos MSER de las matrículas. Independientemente de la iluminación y el tamaño de la matrícula, la aplicación debe ser capaz de extraer todos los caracteres posibles de la matrícula, ya que si perdemos un caracter en esta etapa es imposible recuperarlo más adelante.

```
int min_area = cvRound(0.01 * img->width * img->height);
int max_area = cvRound(0.03 * img->width * img->height);
CvMSERParams params = cvMSERParams(delta, min_area, max_area,
                                   max_variation, min_diversity);
```

A continuación se detalla el valor asignado a cada una de las variables que determinan la precisión del MSER: *delta* establece el número de umbrales consecutivos que debe soportar sin cambios una región para ser considerada MSER, como se especifica en el Apartado 2.2.1. La idea es seleccionar un valor relativamente bajo (delta = 10) para no perder ningún caracter de inicio; *min_area* y *max_area* establecen el tamaño mínimo y máximo que deben tener los caracteres en relación al tamaño de la imagen (evitar extraer regiones del contorno de la matrícula); *max_variation* especifica el grado de cambio máximo permitido para los MSER en el intervalo delta, al aplicar la secuencia de umbrales sobre la imagen (max_variation = 0.5f) y *min_diversity* rechaza los MSER con una diversidad menor a la establecida (min_diversity = 0.2f).

Al seleccionar estos parámetros y realizar los experimentos se generan 4500 imágenes positivas, de las cuales el 80 % se destinan a training y el otro 20 % para testing. De esta forma se genera una base de datos que cuenta con **3600 imágenes para entrenar** y **900 imágenes para realizar las pruebas**.

Llegado este momento es interesante analizar el comportamiento del algoritmo MSER al aplicar una imagen o vídeo como entrada al sistema. Aún seleccionando los parámetros que mejor se ajustan al problema, a la hora de extraer los caracteres se producen ciertos problemas que hay que tratar:

- *Repetición.* Al determinar los MSER asociados a los caracteres es común extraer varios contornos de una misma letra o número cuando se asigna un valor *delta* relativamente bajo. Como en una matrícula pueden aparecer varias ocurrencias del mismo caracter es imprescindible almacenar la posición de cada caracter. De esta forma, al clasificar el mismo caracter varias veces, el sistema establecerá una votación para determinar de qué caracter se trata.

- *Falsos MSER.* El otro problema del algoritmo es reconocer equivocadamente una región de la imagen. El caso más habitual es identificar la banda vertical azul de la izquierda de la matrícula como un caracter. Para evitar este problema la idea propuesta es crear una clase nueva en el clasificador multiclase que permita reconocer este tipo de símbolos.

A la hora de presentar los caracteres en pantalla, el objetivo es imprimir los resultados generados en cada posición de la matrícula donde se reconoce un MSER. Para una misma posición con varios MSER extraídos se devuelve el caracter que reciba más votos. Para el caso de la banda vertical azul no se muestra nada por pantalla.

3.3.2. Experimentos de clasificación

Construida la base de datos, llega el momento de entrenar el clasificador multiclase capaz de identificar correctamente los caracteres extraídos de las matrículas. Como se especifica en el Apartado 2.2.2, los Random Trees se presentan como una buena solución para este problema. Además, la biblioteca OpenCV 2.2 permite implementar el algoritmo ajustando una serie de parámetros con la clase CvRTParams.

```
CvRTParams params = CvRTParams(max_depth, 5, 0, false, 15,
                               priors, false, 4, 100, 0.01f,
                               CV_TERMCRIT_ITER | CV_TERMCRIT_EPS);
```

A continuación se detallan ordenadamente los parámetros seleccionados para entrenar el clasificador basado en Random Trees: *max_depth* establece la profundidad máxima que pueden tener los árboles de decisión (max_depth = 25); *min_sample_count* especifica el número mínimo de muestras aleatorias con remplazamiento generadas para entrenar los árboles; *regression_accuracy* solo actúa en clasificadores con árboles de regresión; *use_surrogates* es la etiqueta que determina el uso de nodos suplentes en los árboles para evitar que se estanque el entrenamiento cuando ciertas variables generan la misma probabilidad en un nodo; *max_categories* permite agrupar las 30 clases de caracteres en un número menor para reducir el tiempo de cómputo al calcular los conjuntos de entrenamiento (max_categories = 15); *priors* es el vector de probabilidades que permite asignar a determinadas clases mayor probabilidad de aparición a priori. De esta manera, para el formato "0000-BBB" se asignará un porcentaje de 4/7 para los dígitos y 3/7 para las letras.

Por otra parte, *calc_var_importance* permite calcular el error estimado en cada árbol al elegir determinadas variables para formar cada nodo (calc_var_importance = false); *nactive_vars* establece el número de variables elegidas aleatoriamente del conjunto de entrenamiento para proporcionar a cada nodo el conocimiento necesario que separa las clases; *max_tree_count* especifica el número de árboles máximo para el Random Forest (max_tree_count = 100); *forest_accuracy* determina la precisión del clasificador y *term_crit* establece los criterios de terminación del entrenamiento.

Los resultados obtenidos en los experimentos cumplen con las expectativas planteadas en un principio. De los 900 casos de prueba, se han clasificado correctamente 826 imágenes (**92 % aciertos**). Como resultado, la aplicación en estos momentos es capaz de localizar los automóviles vistos de frente en una escena y reconocer los caracteres de las matrículas asociadas a los mismos.

3.4. Reconocer la marca y el modelo del vehículo

El último objetivo del sistema será clasificar correctamente la marca y el modelo de los automóviles ubicados en la escena a procesar, mediante un clasificador que permita reconocer hasta 50 modelos distintos. Para construir el clasificador, la idea es en primer lugar generar la base de datos de entrenamiento con las imágenes de los automóviles vistos de frente, para por último realizar los experimentos que determinen el rendimiento del clasificador (estructura de la rejilla y descriptor de características implementado).

3.4.1. Construcción de la base de datos

En este apartado se detallan las decisiones que se han tomado para construir la base de datos de los experimentos. Primero se entra a valorar la cantidad de imágenes que conviene tener para cada modelo. La idea es usar la técnica conocida como validación cruzada para garantizar que los resultados obtenidos son independientes de la partición entrenamiento - pruebas realizada (de esta forma, de las 6 imágenes de automóviles por cada modelo, 5 imágenes se emplean para entrenar y 1 para realizar el test). El otro aspecto importante a tener en cuenta en la construcción de la base de datos es el de cuántos modelos de automóviles utilizar. Para obtener resultados significativos serán suficientes 50 modelos diferentes (aunque muchos vehículos comparten marca, como se especifica en el Cuadro 3.3). De esta forma, se genera una base de datos que cuenta con 6 x 50 modelos = **300 imágenes distintas para entrenar y realizar las pruebas**.

Id	Modelo del Automóvil	Id	Modelo del Automóvil
1	Alfa Romeo	26	Opel Corsa
2	Audi A4	27	Opel Vectra
3	Audi A6	28	Opel Zafira
4	BMW Serie 1	29	Peugeot 106
5	BMW Serie 3	30	Peugeot 206
6	Citroen C3	31	Peugeot 307
7	Citroen C4	32	Peugeot 407
8	Citroen C5	33	Renault Clio
9	Citroen Xantia	34	Renault Laguna
10	Citroen Xsara	35	Renault Megane
11	Citroen Xsara Picasso	36	Renault Scenic
12	Daewoo Nubira	37	Seat Altea
13	Fiat Punto 2002	38	Seat Altea XL
14	Fiat Punto 2006	39	Seat Córdoba 2000
15	Fiat Stilo	40	Seat Córdoba 2002
16	Ford Escort	41	Seat Córdoba 2006
17	Ford Fiesta	42	Seat Ibiza 1994
18	Ford Focus	43	Seat Ibiza 2006
19	Ford Focus C-Max	44	Seat León
20	Ford Ka	45	Seat Toledo 2000
21	Hyundai Coupe 1997	46	Toyota Avensis
22	Hyundai Coupe 2000	47	Toyota Corolla
23	Kia Carnival	48	Volkswagen Passat
24	Mercedes Benz	49	Volkswagen Polo
25	Opel Astra	50	Volvo S60

Cuadro 3.3: Tabla con los 50 modelos de automóviles.

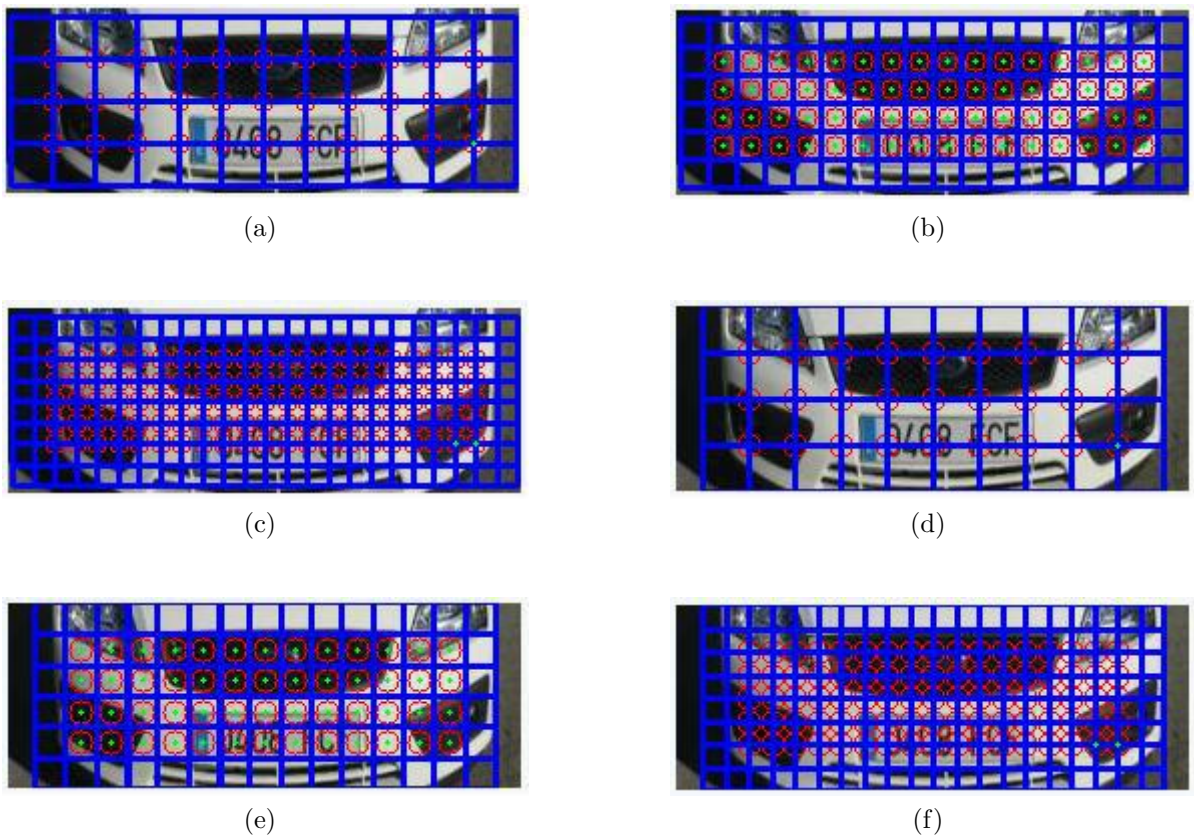
3.4.2. Experimentos de clasificación

Como se especifica en el Apartado 2.3.2, mediante el descriptor SURF se pueden extraer una serie de vectores de características de 64 componentes a partir de unos deter-

minados puntos considerados de interés para cada frontal de automóvil. La idea es ajustar una rejilla a la imagen del frontal para indicarle al descriptor manualmente las regiones consideradas de interés (cada celda de la rejilla establece una región de interés para el descriptor SURF).

El PFC presentado en [12] estudia diferentes tipos de rejillas para seleccionar la que proporcione una mayor tasa de acierto de modelos reconocidos correctamente. El objetivo es analizar las rejillas más sencillas que se ajustan a las imágenes rectificadas a 256 x 92, para después intentar aumentar el número de celdas en aquellas rejillas que ofrecen los mejores resultados. La idea es desplazar N píxeles cada celda de la rejilla con respecto a la anterior, de forma que cada celda termine una más cerca de otra. Esto permitirá colocar más celdas descriptoras en la imagen, mejorando la media de aciertos a medida que aumentamos el solapamiento entre las celdas de la rejilla.

En la Figura 3.6 se detallan algunos ejemplos con las rejillas que mejores resultados presentan que son las que tienen celdas de 42 x 42 y 46 x 46 píxeles de tamaño, a las que se las aplican diferentes desplazamientos entre celdas adyacentes. A la hora de aplicar la técnica del desplazamiento es importante saber cuántos píxeles desplazar las celdas, porque no todas las distancias valen. Estas distancias de separación entre celdas deben de ser de por lo menos la mitad del tamaño de las celdas de la rejilla para que se pueda incluir alguna celda más en el espacio libre.



(a) (b) (c) (d) (e) (f)

Figura 3.6: Imágenes de rejillas con diferentes desplazamientos de celdas.

Finalmente, en el Cuadro 3.4 se presentan los resultados de las rejillas estudiadas, donde se observa como la rejilla con celdas de 46 píxeles y 2/3 solapados ofrece el mejor

rendimiento. A la hora de realizar los experimentos, la técnica de validación cruzada permite estudiar los resultados de los 6 experimentos posibles: $x_1 = 80\,\%$, $x_2 = 90\,\%$, $x_3 = 76\,\%$, $x_4 = 80\,\%$, $x_5 = 86\,\%$ y $x_6 = 86\,\%$.

Solapamiento de las celdas	Núm. de celdas	Tamaño de celda	Tasa de acierto	Desv. Típica
1/2 solapado	33	42 x 42 píxeles	74.333	8.0416
2/3 solapados	64	42 x 42 píxeles	81.667	6.5013
3/4 solapados	105	42 x 42 píxeles	77.333	8.0664
1/2 solapado	27	46 x 46 píxeles	82.333	6.8605
2/3 solapados	52	46 x 46 píxeles	83	5.1769
3/4 solapados	85	46 x 46 píxeles	82	6.8118

Cuadro 3.4: Rendimiento del clasificador entrenado con las mejores rejillas.

Con respecto a la biblioteca para extraer las características, la idea es mantener la biblioteca libre OpenSURF desarrollada bajo C++ en el PFC presentado en [12]. No obstante, la herramienta OpenCV 2.2 ya implementa el detector de puntos de interés y el descriptor de características para los algoritmos SIFT y SURF detallados.

```
surfDes(img, vectorIpts, true);
```

A la hora de extraer las características con OpenSURF han de ajustarse correctamente los parámetros del método "surfDes". Respectivamente: *img* contiene la imagen de entrada con el frontal del automóvil a procesar; *vectorIpts* es el vector que almacena la posición de los puntos de interés asociados a la imagen (celdas de la rejilla) y *upright* la etiqueta que establece invarianza a las rotaciones en las características extraídas del automóvil.

Por último, para implementar eficientemente el algoritmo de la Ecuación 2.5 es imprescindible calcular el vecino más cercano $NN_{m,i}(q_d)$ para cada modelo de la base de datos. La biblioteca ANN (Approximate Nearest Neighbor) [18] proporciona las estructuras de datos y los algoritmos necesarios para entrenar el clasificador y evaluar su rendimiento. No obstante, para la aplicación C++ en desarrollo, la biblioteca OpenCV 2.2 incorpora la estructura de datos CvKNearest para almacenar las características de cada modelo de automóvil y un algoritmo de búsqueda "find_nearest" para encontrar los K vecinos más cercanos a los descriptores SURF extraídos de la imagen.

```
CvKNearest(trainData, trainClasses, 0, false, K);
find_nearest(matFrontal, K, 0, 0, responses, dist);
```

Los parámetros *trainData* y *trainClasses* incluyen la información de los 6 automóviles de cada modelo así como los identificadores de cada celda de la rejilla (6 x 52 = 312 componentes). La idea es generar un CvKNearest por cada uno de los 50 modelos y almacenar las estructuras en una lista, de forma que el método find_nearest pueda calcular la distancia que hay entre un automóvil a clasificar y cada modelo del sistema, recorriendo la lista con un iterador. El atributo K establece el número de vecinos considerados de interés para clasificar como acierto (K = 1) y *matFrontal* almacena las características de la imagen de entrada. Como salida del algoritmo se establecen los vectores *neighbor_responses* y *dist* que devuelven tanto el índice del vecino más cercano como la distancia a esa clase respectivamente.

El objetivo es implementar con el KNN que proporciona OpenCV 2.2 el algoritmo de clasificación de marca y modelo del Apartado 2.3.3 que permite calcular la distancia total (suma de distancias para las 312 componentes) que hay entre la imagen a clasificar y los 50 modelos de la base de datos. Finalmente, la clase que tenga asignada la menor distancia determinará el modelo del automóvil en cuestión, de forma que a la hora de analizar los resultados, se clasificará como acierto si el modelo de la imagen de entrada coincide con la clase que menor distancia devuelve.

Respecto a los resultados de los experimentos, aunque a priori la media de aciertos es del 83 %, seleccionando las 3 clases más cercanas a la imagen se alcanza una **tasa de aciertos del 92 %** (el modelo correcto está entre los 3 modelos más parecidos en el 92 % de las ocasiones). Llegado este momento, la aplicación sería capaz de localizar en una escena los automóviles vistos de frente y las matrículas asociadas a los mismos, para reconocer el modelo de cada vehículo y leer el contenido de las matrículas.

Capítulo 4

Descripción informática

El objetivo de este capítulo es aplicar los algoritmos analizados en el Capítulo 2 para construir una aplicación en C++ que sea capaz de extraer información de los automóviles vistos de frente en una escena. Realizar la descripción informática es simplemente analizar en detalle las etapas del ciclo de desarrollo del software seleccionado: captura de requisitos, fase de análisis, diseño y implementación del sistema.

4.1. Metodología y plan de trabajo

En este apartado se van a entrar a detallar las pautas que se han seguido para desarrollar la aplicación presentada en el PFC. Para ello se indican tanto la metodología y el modelo utilizado a la hora de construir la aplicación, como el orden establecido en el plan de trabajo durante la realización del proyecto. Por último, se va a realizar una estimación del tiempo empleado para realizar las tareas propuestas en cada una de las etapas del plan de trabajo.

Un punto fundamental para que el desarrollo del PFC trascurra de manera satisfactoria es la selección de un ciclo de vida adecuado al principio del proyecto. En principio, un desarrollo en cascada resultaría difícil de aplicar en un proyecto de caracter iterativo como este, ya que no podríamos analizar los resultados de las primeras etapas hasta que se implementara todo el sistema. De esta manera, se consideró usar algún ciclo de vida capaz de generar resultados intermedios que permitan verificar que los resultados de detección de las primeras etapas son adecuados, con la idea de evitar tener que rediseñar completamente la aplicación al final.

El desarrollo en espiral es un modelo de ciclo de vida del software definido por B. Boehn en 1988. Las tareas de este modelo se conforman en una espiral que representa el conjunto de actividades que genera cada una de las etapas del proyecto. De esta manera, en cada iteración del modelo en espiral la idea es: determinar los objetivos de la iteración, analizar los posibles riesgos de cada etapa, desarrollar el algoritmo correspondiente (realizar las pruebas que determinan el rendimiento de cada clasificador detallado en el Capítulo 3) y planificar la siguiente iteración de la espiral. Por tanto, una vez **elegido el modelo en espiral como herramienta para desarrollar el software del PFC**, el objetivo es detallar cuáles han sido las tareas del plan de trabajo que actúan como iteraciones del modelo.

- *Documentación previa.* Introducción al problema de localización de objetos en imágenes y reconocimiento de caracteres por ordenador. La idea es obtener información de artículos relacionados con el tema del proyecto para facilitar la comprensión del resto de etapas. Además es importante configurar el framework para desarrollar la aplicación (Microsoft Visual Studio 2010 + OpenCV 2.2) en el que poder comprobar el funcionamiento del sistema periódicamente. Tiempo estimado: **1 mes**.

- *Localización de automóviles vistos de frente.* Construcción del clasificador binario (automóvil / no automóvil) para localizar los vehículos en la escena. Primeramente hay que generar una base de datos de automóviles completa para poder realizar los experimentos oportunos y ajustar los parámetros que determinan el rendimiento del clasificador. Tiempo estimado: **3 meses**.

- *Construir el clasificador de matrículas.* Al localizar todos los automóviles en la escena, el objetivo es construir un nuevo clasificador capaz de detectar la posición de cada matrícula dentro de la región de interés con el frontal. Aunque la técnica para construir el clasificador es similar a la de la etapa anterior, la base de datos de matrículas es completamente nueva. Por último se realizan los experimentos que establecen la configuración adecuada de los respectivos parámetros. Tiempo estimado: **2 meses**.

- *Lectura de los caracteres de la matrícula.* La idea es extraer los caracteres asociados a la matrícula localizada en la iteración anterior y entrenar un clasificador multiclase que permita reconocer cada letra o dígito. Mediante el algoritmo MSER pueden extraerse los contornos de los caracteres para generar una nueva base de datos de caracteres con la que realizar los experimentos. Tiempo estimado: **3 meses**.

- *Integrar el clasificador de marca y modelo.* El objetivo de esta etapa es ajustar el clasificador de marca y modelo presentando en [12] al sistema en desarrollo. Las iteraciones anteriores ayudan a evitar tener que marcar la posición de las esquinas de la matrícula del vehículo (ahora conocemos la ubicación de la matrícula dentro de la imagen con el frontal). Tiempo estimado: **1 mes**.

- *Aplicación con interfaz C++.* La última iteración trata de combinar el trabajo realizado en las etapas anteriores para crear un sistema en C++ con la funcionalidad planteada en un principio. A la hora de generar el modelo es imprescindible el uso de patrones de diseño para dotar a la aplicación de ciertas características (abstracción de objetos, fácil mantenimiento, etc.) imprescindibles a la hora de desarrollar un software de calidad. Tiempo estimado: **2 meses**.

4.2. Captura de requisitos

Para que el funcionamiento del sistema pueda considerarse correcto, se deberán cumplir una serie de condiciones denominadas requisitos. La idea de este apartado es enumerar todas las reglas que establecen qué tipo de acciones debería realizar el producto deseado así como los requisitos más técnicos asociados a las herramientas que implementan el software.

4.2.1. Requisitos funcionales

Los requisitos funcionales son aquellos que indican qué debe hacer o cómo debería reaccionar la aplicación en determinadas situaciones.

RF1. El sistema permitirá al usuario cargar una imagen o un vídeo como parámetro de entrada para procesar su contenido en busca de automóviles vistos de frente.

RF2. La aplicación deberá cargar automáticamente cada uno de los clasificadores (automóviles, matrículas, caracteres y modelos) al iniciar la ejecución del programa.

RF3. El usuario deberá estar informado en todo momento del estado de la aplicación a medida que se cargan los clasificadores. En caso de producirse algún error es necesario especificar con claridad cuál ha sido el problema.

RF4. El sistema se encargará de reconocer todos los automóviles vistos de frente en la escena correspondiente. Respecto al tiempo de procesamiento el objetivo es localizar los vehículos en tiempo real (25 frames por segundo) y evaluar sus características en el menor tiempo posible.

RF5. La interfaz gráfica deberá señalar las regiones de interés (el frontal del automóvil y la matrícula asociada al mismo) con rectángulos de distintos colores que se ajusten sobre el objeto.

RF6. La aplicación presentará en la propia interfaz tanto el modelo de automóvil como la lectura realizada de la matrícula. La idea es colocar las predicciones encima de los rectángulos que marcan la posición de los vehículos y las matrículas.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que, aunque no definen la funcionalidad mínima exigida de la aplicación, son necesarios para su correcto funcionamiento. A continuación se enumeran algunos de estos requisitos técnicos que detallan las herramientas empleadas para desarrollar el software.

RNF1. El sistema se desarrolla bajo un entorno Windows por lo que es imprescindible trabajar con este sistema operativo para lanzar el ejecutable generado.

RNF2. La aplicación se apoya en las bibliotecas de OpenCV 2.2 y OpenSURF. Para poder lanzar la aplicación se deben instalar previamente dichas bibliotecas. No obstante, se van a proporcionar las dll necesarias para ejecutar el software directamente en Windows.

RNF3. El software se compilará mediante el framework Microsoft Visual Studio 2010 (proyecto desarrollado en lenguaje C++).

4.3. Análisis y diseño

Realizar la instalación de un sistema determinado sin una planificación previa adecuada puede conducir a grandes frustraciones cuando al implementar el software percibamos que no cumple con las expectativas planteadas en un principio. El proceso de análisis y

diseño de sistemas trata de aplicar un conjunto de técnicas y principios con el propósito de definir un sistema con el suficiente detalle como para determinar su realización física. La idea es implementar todos los requisitos establecidos y proporcionar una idea completa del software mediante un diagrama de clases general.

El objetivo de este apartado es construir un sistema de calidad aplicando algunos principios básicos de diseño de la ingeniería del software que se enumeran a continuación.

- *Descomposición y modularidad.* Aplicar el principio de descomposición orientado a objetos para limitar el efecto de cualquier decisión de diseño en el resto del sistema. Dividir el software en diferentes módulos o componentes identificables y tratables por separado.

- *Abstracción.* Permite comprender la esencia de los subsistemas sin conocer detalles innecesarios de la clase. Fomentar el uso de interfaces y variables privadas para ocultar la información contenida en cada módulo.

- *Cohesión y acoplamiento.* Alcanzar un compromiso entre la fuerte cohesión y el bajo acoplamiento. Es interesante tanto mantener unidas las clases relacionadas entre sí para favorecer la comprensión del sistema (cohesión), como eliminar dependencias innecesarias entre clases para mantener el sistema bien dividido (acoplamiento).

Llegado este momento, se va a modelar el **diagrama de clases general** asociado a la aplicación que se detalla en la Figura 4.1. En primer lugar, se analiza la parte del sistema encargada de localizar los objetos de interés en la escena (tanto automóviles como matrículas). Para ello se va a diseñar una interfaz *VehiclesTracker* de la cual heredará la clase *AllVehiclesTracker* que será la encargada de encontrar todos los vehículos vistos de frente en la escena. Cada automóvil localizado se irá almacenando en la clase *Vehicle* donde se guardará la posición del frontal y la matrícula asociada a cada automóvil que se encarga de detectar la clase *PlateDetector*.

Finalmente, para procesar cada una de las regiones de interés localizadas anteriormente (lectura de matrícula y reconocimiento del modelo) se han implementado una serie de clases *PlateRecognition* y *MakeModel* que heredan de la interfaz *VehicleProccesor*. De esta forma, cada automóvil localizado en la escena tendrá asignado un objeto que almacenará todo el grupo de procesadores definidos. La clase *VehiclesAnalyzer* actuará de "director de orquesta" en nuestro sistema para asociar el detector de objetos de interés *VehiclesTracker* y los procesadores de *VehicleProccesor*.

Los **patrones de diseño** propuestos en [19] ayudan a evitar futuros rediseños al asegurar que un sistema pueda evolucionar adecuadamente cuando se vayan a introducir nuevos requisitos al sistema.

- *Prototype.* Patrón de diseño creacional que tiene como objetivo crear nuevos objetos duplicándolos, clonando una instancia creada previamente (prototipo). La idea es proporcionar a la clase interfaz *VehicleProcessor* de un método **clone()** que permita solicitar una copia del prototipo de procesador básico que implementan el lector de matrículas *PlateRecognition* y el clasificador de marca y modelo *MakeModel* respectivamente.

- *Composite.* Patrón estructural que permite construir una clase compleja a partir de otras de carácter más simple que poseen una interfaz común. La clase *Vehicle* deberá tener asociado un procesador de tipo *VehicleProcessor* implementado con la clase colectiva *VehicleProcessorGroup* que almacenará el lector de matrículas *PlateRecognition* y el clasificador de marca y modelo *MakeModel* respectivamente.

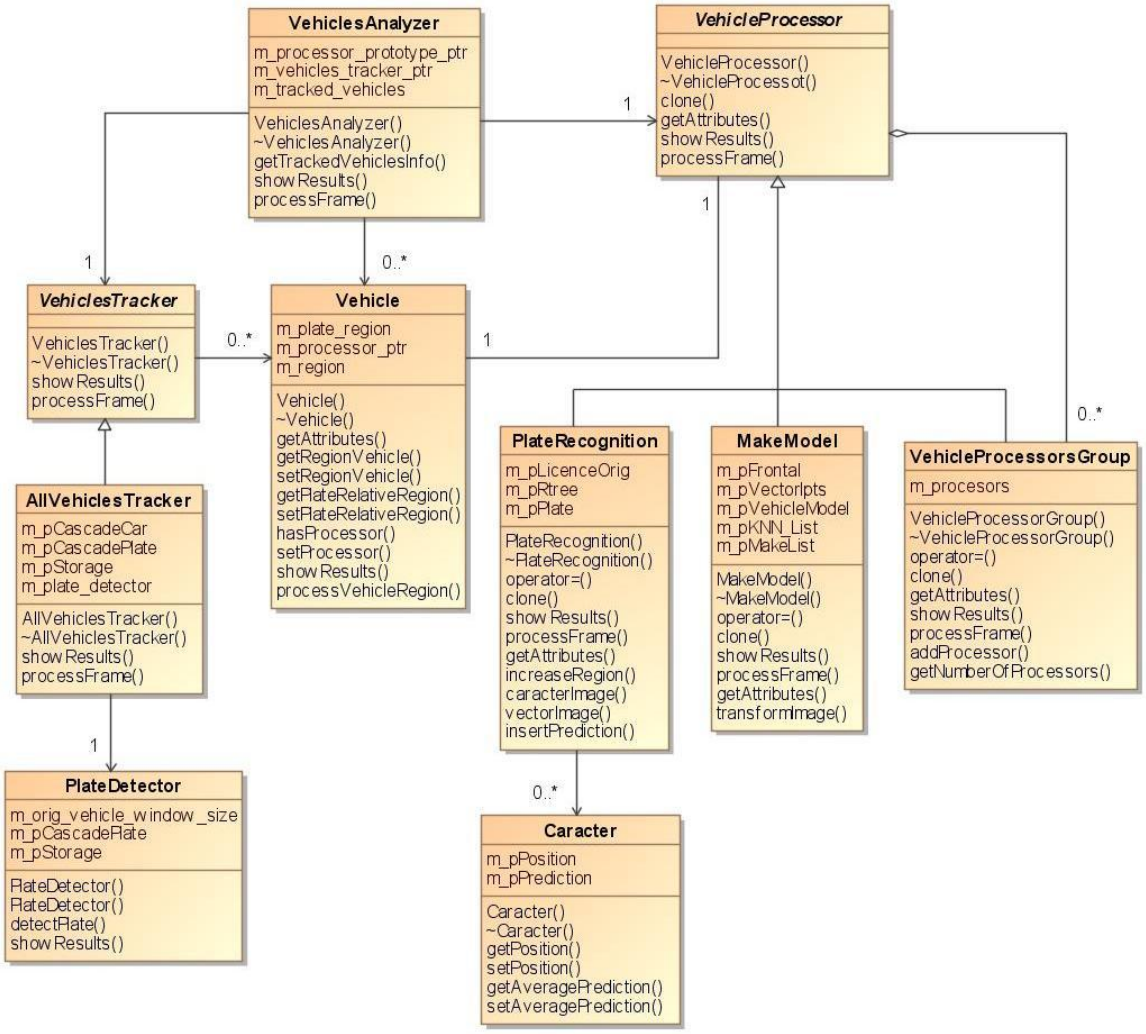


Figura 4.1: Diagrama de clases de diseño UML de la aplicación desarrollada.

Con respecto a las ventajas que ofrece el modelo, destaca la facilidad que proporcionan las clases abstractas *VehiclesTracker* y *VehicleProcessor* a la hora de ampliar los requisitos de la aplicación. De esta forma, para localizar todos los automóviles vistos de frente en la escena, el sistema implementa la clase *AllVehiclesTracker*. A su vez esta clase almacena un objeto de la clase *PlateDetector* que permite localizar las matrículas asociadas a cada uno de los vehículos anteriores. Si en un futuro la aplicación necesitara incluir un nuevo requisito que tratara de localizar los autobuses en la escena, la idea sería simplemente crear una nueva subclase que heredara de *VehiclesTracker*.

Por otra parte, la clase abstracta *VehicleProcessor* permite crear nuevas subclases para procesar y extraer información de los automóviles localizados. De esta forma, la clase *PlateRecognition* se encarga de leer los caracteres de cada matrícula detectada, mientras

que *MakeModel* es el módulo que permite reconocer la marca y modelo a partir de la región de interés con el frontal de los vehículos. Si posteriormente resultara interesante incluir en el sistema un nuevo requisito para determinar en qué casos lleva el automóvil las luces encendidas, la idea del diseño es que simplemente bastase con crear una nueva subclase de *VehicleProcessor*.

El principal problema que plantea el diseño es la dependencia que presenta el modelo entre los objetos de interés a localizar por la clase *VehiclesTracker* y las características que procesa el módulo *VehicleProcessor* de estos objetos. Por ejemplo, es imprescindible localizar la matrícula del vehículo para reconocer después los caracteres correctamente.

Es interesante exponer el funcionamiento inicial del sistema para que el usuario pueda hacerse una idea de cómo se relacionan las clases entre sí durante la implementación del software. Para ello, detallamos una serie de diagramas de secuencia UML que permitirán modelar la interacción entre los objetos del sistema a través del tiempo. Un diagrama de secuencia muestra las instancias de las clases que intervienen en un escenario determinado con líneas discontinuas verticales, y los mensajes pasados entre los objetos con flechas horizontales que representan las invocaciones de los métodos de la clase en orden cronológico.

El diagrama de secuencia de la Figura 4.2 establece la relación que existe entre la clase *VehiclesTracker* encargada de seguir los objetos de interés en la escena y el procesador que permite extraer las características de estos objetos *VehicleProcessor*. El módulo principal *Test_Framework* será el encargado de lanzar el programa y recibir la imagen o vídeo de entrada. En primer lugar llamará al constructor de *AllVehiclesTracker* para localizar todos los áutomóviles de la escena, y a continuación creará la clase *VehicleProcessorGroup* para añadir los respectivos procesadores de lectura de matrícula y reconocimiento del modelo del automóvil. Finalmente, la clase *VehiclesAnalyzer* se encargará de relacionar el tracker encargado de localizar los vehículos y las matrículas con el grupo de procesadores previamente definido.

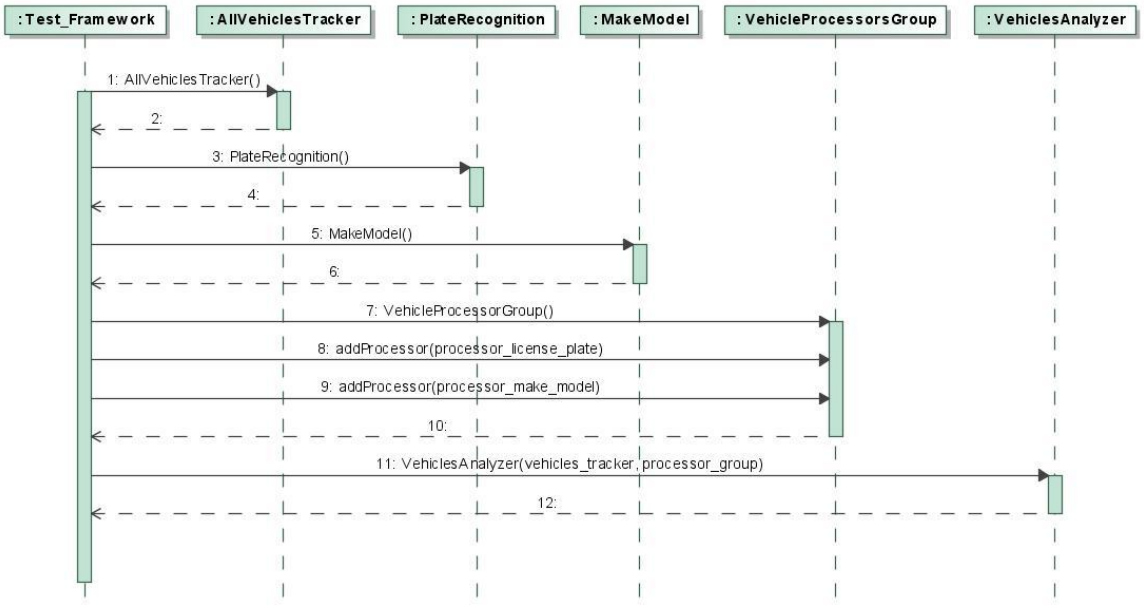


Figura 4.2: Diagrama de secuencia que diseña un analizador de automóviles.

Respecto a la clase *Vehicle* será la encargada de almacenar los valores que determinan la posición del frontal de cada automóvil en la escena, así como la ubicación de las matrículas asociadas a los mismos. La idea es crear una clase *ImageRegion* que permita guardar las coordenadas de la región que se ajusta sobre el objeto de interés localizado. Las subclases que hereden de *VehiclesTracker* deben ser capaces de acceder a *ImageRegion* para asignar con el método **setParams()** los valores de la región que determina la posición del automóvil y la matrícula: coordenada de la esquina izquierda, alto y ancho del rectángulo respectivamente. Llegados a este punto, serán las subclases de la interfaz *VehicleProcessor* las encargadas de acceder a estos campos con el método **getParams()** para procesar la región de la imagen delimitada por esta región de interes.

La clase abstracta *VehiclesProcessor* encargada de procesar los objetos de interés deberá almacenar los caracteres de la matrícula y la marca del automóvil en alguna clase accesible desde el módulo principal del sistema *VehiclesAnalyzer* (proveer un mecanismo para mandar hacia arriba los atributos estimados). El objetivo es construir una clase *VehicleAttribute* que permita almacenar cualquier característica extraída de los objetos de interés de la imagen junto con el nombre y el formato asignado a la variable en cuestión (integer, string, etc).

```
name=[lic_plate], value=[7669CLD], confidence=[1]
name=[make_model], value=[Ford_Fiesta_2004], confidence=[1]
```

Por último, la clase *Viewer* permitirá crear la ventana que actuará de interfaz para el software desarrollado. La idea es que las clases que necesiten dibujar por pantalla ciertas características como imágenes, texto, rectángulos o similares, puedan hacerlo a través del método **showResults()** que se implementa en la mayoría de las clases. A continuación, en la Figura 4.3, establecemos el formato de las clases encargadas de almacenar y mostrar las características que el sistema va procesando.

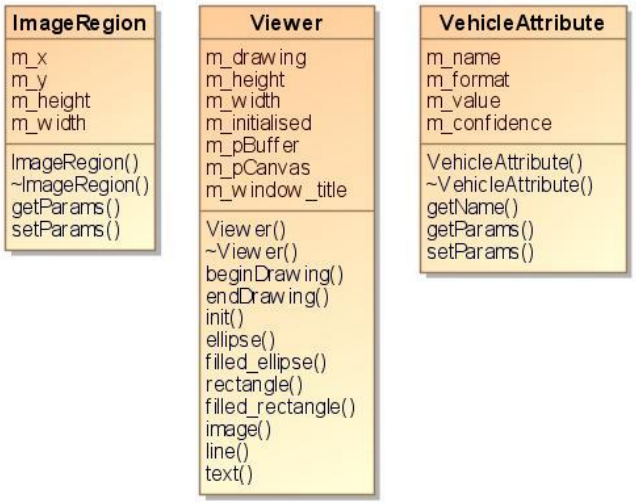


Figura 4.3: Clases que permiten representar las características procesadas.

En el diagrama de secuencia de la Figura 4.4, el módulo principal *Test_Framework* se encarga tanto de recibir la imagen o vídeo de entrada, como de generar la interfaz y procesar cada frame con la idea de mostrar los resultados obtenidos por pantalla. La clase *VehiclesAnalyzer* será la encargada de llamar al método **processFrame()** de cada una

de las subclases de *VehiclesTracker* y *VehicleProcessor* para extraer ciertas características de los objetos de interés. Llegado este momento, será posible detallar la posición del automóvil y la matrícula con dos rectángulos, así como el contenido de la matrícula y la marca y modelo del vehículo.

Como curiosidad, destaca el hecho de anotar con la instrucción **text()** la velocidad a la que procesa las imágenes el software en cada instante (número de frames por segundo). Finalmente, una vez definida la interfaz gráfica y generados los resultados, el sistema es capaz de obtener la información de los vehículos almacenada en las clases *ImageRegion* y *VehicleAttribute* para mostrar además los resultados en la misma consola de Windows o utilizarlos en otro módulo de un sistema más grande.

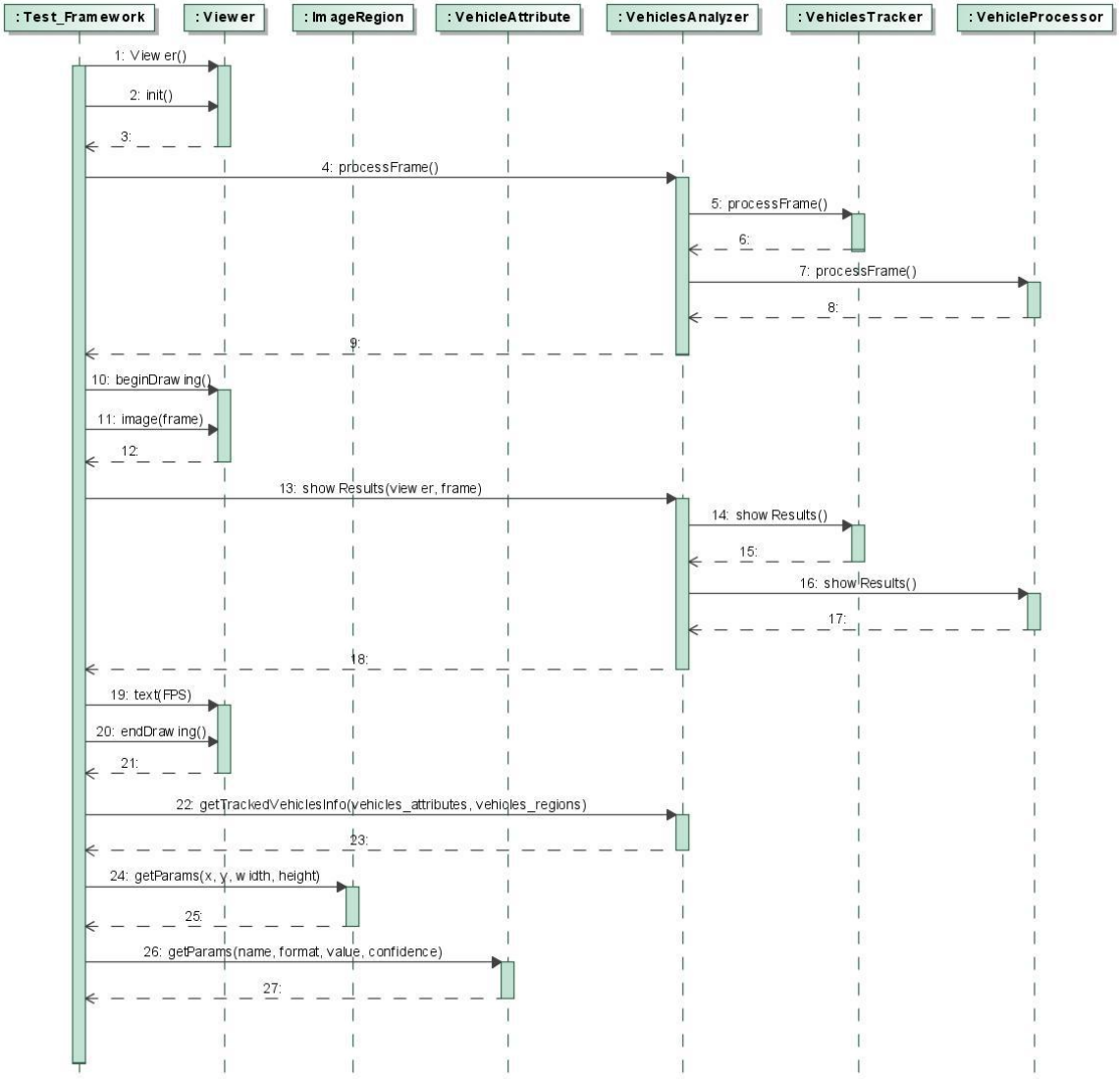


Figura 4.4: Diagrama de secuencia que muestra los resultados en la interfaz.

A continuación se entra a detallar el funcionamiento de los métodos de cada clase, así como sus correspondientes parámetros:

`Clase: VehiclesAnalyzer`

- **processFrame(IplImage *pFrame, Vehicle &v);** Método encargado de procesar una imagen de tipo *IplImage* para extraer ciertas características de determinados

automóviles de la clase *Vehicle*. Dependiendo de la implementación de cada clase, ciertas clases deberán hacer uso de algunas propiedades asociadas previamente a los automóviles. Por ejemplo, para reconocer los caracteres es imprescindible conocer de antemano la ubicación de la matrícula asociada al vehículo "v".

- **showResults(Viewer *pViewer, IplImage *pImage, Vehicle &v);** Función que muestra los resultados obtenidos con la interfaz que define la clase *Viewer*. La idea es que dependiendo de la implementación de cada clase, el método muestre en la escena introducida por la clase *IplImage*, cualquier característica asociada al vehículo "v" (dibujar el rectángulo que se ajusta a la posición del frontal del automóvil y la matrícula, mostrar los caracteres de la matrícula y detallar el modelo del vehículo respectivamente).
- **getTrackedVehiclesInfo(vector <VehicleAttributes> &attributes, vector <ImageRegion> &regions);** Procedimiento que recorre la lista de vehículos localizados en la imagen para obtener tanto los atributos de los automóviles (modelo y lectura de matrícula) como la ubicación de los mismos en la imagen.

`Clase: PlateDetector`

- **detectPlate(IplImage *pFrame, ImageRegion vehicle_region, ImageRegion &plate_region);** Método encargado de recorrer una imagen *IplImage* para localizar las matrículas asociadas a los vehículos vistos de frente en la escena. La posición de la matrícula se almacena en un objeto de la clase *ImageRegion* como ocurre con la posición del frontal que se introduce por parámetro.

`Clase: VehicleProccesor`

- **clone();** Función asociada al patrón de diseño Prototype que devuelve una copia del procesador *VehicleProcessor* instanciado.
- **getAttributes(VehicleAttributes &attributes);** Método que proporciona las características que se han ido almacenando para cada uno de los vehículos durante el procesamiento de la imagen. Tanto el lector de matrículas *PlateRecognition* como el clasificador de marca y modelo *MakeModel* almacenarán sus resultados en la clase *VehicleAttributes*.
- **operator=(VehicleProcessor &vp);** Función encargada de redefinir el operador de asignación para las subclases de *VehicleProcessor*. La idea es asignar a las variables privadas de la clase correspondiente los valores establecidos en la instancia que llega como parámetro.

`Clase: PlateRecognition`

- **increaseRegion(CvPoint2D32f *corner, int width_max, int height_max);** Procedimiento que se encarga de incrementar ligeramente el rectángulo definido por los parámetros de entrada (esquina izquierda, ancho y alto del rectángulo que contiene un caracter de la matrícula).
- **caracterImage(IplImage *pLicence, CvRect pRegionInc);** Función que realiza una serie de transformaciones (ecualización del histograma, binarización, ...) sobre la región de interés *CvRect* que se ajusta sobre un caracter concreto de la matrícula.

- **vectorImage(IplImage *pCaract);** Procedimiento que se encarga de transformar la imagen rectificada de 20 x 12 a un vector fila de 1 x 240 para facilitar la clasificación del caracter.
- **insertPrediction(CvRect pRegionInc, double result);** Método encargado de generar la cadena de caracteres que forman la matrícula del automóvil. La idea es ordenar correctamente los caracteres de la matrícula a partir de la posición que establece el centro de masas del rectángulo que se ajusta sobre cada caracter.

`Clase: Caracter`

- **getPosition();** Función que devuelve la posición que ocupa un caracter en la imagen (permite ubicar correctamente cada caracter en la matrícula).
- **setPosition(CvRect pRegionInc);** Método encargado de calcular el centro de masas del rectángulo que se ajusta sobre un caracter en cuestión.
- **getAveragePrediction();** Función que establece una votación para determinar que caracter colocar en una determinada posición de la matrícula (para el caso de que se hayan asignado diferentes caracteres en una misma posición).
- **setAveragePrediction(double prediction);** Procedimiento que asigna un caracter concreto a una determinada posición de la matrícula.

`Clase: MakeModel`

- **transformImage(Vehicle &v);** Proceso encargado de rectificar las imágenes con el frontal de los automóviles. El objetivo es aplicar las transformaciones detalladas en el Apartado 2.3.1 para acercar, orientar y eliminar el contenido de cada matrícula (mejora la clasificación de marca y modelo).

`Clase: VehicleProcessorsGroup`

- **addProcessor(VehicleProcessorPtr processor_ptr);** Método que añade una instancia de la clase *VehicleProcessor* a la lista que almacena los procesadores que se van a aplicar a los automóviles y las matrículas.
- **getNumberOfProcessors();** Función que devuelve el número de procesadores que se han definido en el grupo.

4.4. Implementación

En este apartado, el objetivo es detallar la estructura jerárquica que va a tener el proyecto desarrollado con la herramienta Microsoft Visual Studio 2010. A la hora de implementar el sistema diseñado anteriormente, son imprescindibles una serie de ficheros que colocaremos en la carpeta generada automáticamente al crear el proyecto. La Figura 4.5 muestra el esquema con los directorios de la aplicación que se va a analizar.

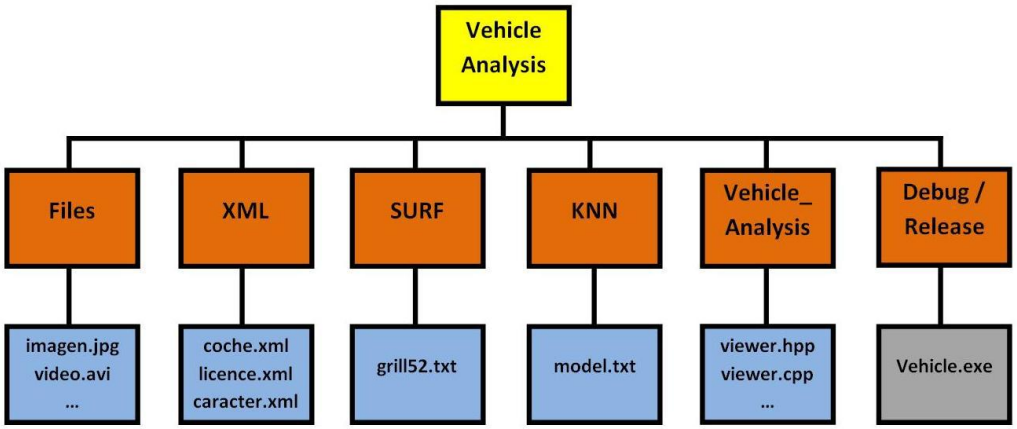


Figura 4.5: Árbol con los directorios de la aplicación.

- *Files.* Es la carpeta que almacena las imágenes y vídeos de prueba para el sistema. La idea es acceder a este directorio para seleccionar alguno de los archivos como entrada para la aplicación generada. Respecto a los formatos soportados por OpenCV, los experimentos se han realizado sin problemas con imágenes en formato jpeg, bmp y vídeos en formato avi o wmv respectivamente.

- *XML.* Directorio en el que encontramos los clasificadores en formato xml que se han ido generando mediante los experimentos del Capítulo 3. El clasificador **cascade_coche.xml** es el encargado de localizar en la escena los automóviles vistos de frente, mientras que **cascade_licence.xml** permite localizar las matrículas de los vehículos. Respecto a los ficheros que genera la herramienta haarcascade de OpenCV 2.2, resulta interesante analizar su estructura para descubrir cuáles son las características Haar más determinantes y dónde se colocan (Apartado 2.1.2).

```
<?xml version="1.0"?>
<opencv_storage>
<haarcascade type_id="opencv-haar-classifier">
  <size>35 15</size>
  <stages><trees><feature>
    <rects>
    <_>3 3 20 8 -1.</_>
    <_>3 3 10 4 2.</_>
    <_>13 7 10 4 2.</_>
    </rects>
    ...
```

  Por último, el clasificador multiclase **rtree_caracters.xml** será el encargado de reconocer cada una de las letras y dígitos que forman la matrícula localizada en la etapa anterior del sistema.

- *SURF.* Es la carpeta donde está establecido el fichero de texto **grill52.txt** que contiene los datos de cada celda de la rejilla que utiliza el descriptor SURF del software como región de interés para extraer las características del modelo del automóvil (Apartado 2.3.2).

celda	x	y	ancho	alto
0	13	0	46	46
1	28.33	0	46	46
...	...	...	...	...
51	197	46	46	46

- *KNN.* Directorio que contiene el fichero **bd_make_model.txt** donde se almacenan las características extraídas de las 300 imágenes que forman la base de datos de marca y modelo de automóviles. El número de características que definen cada vehículo depende de la rejilla empleada para extraer la información, de forma que con una rejilla de 52 celdas y 64 componentes por región de interés, alcanzamos las 3328 características para cada automóvil.

coche	características
0	-0.00311563 0.0672401 ...
...	...
300	0.00175978 0.129043 ...

- *Vehicle_Analysis.* Al crear el proyecto C++ se generará una subcarpeta con el fichero **Vehicle_Analysis.vcxproj** asociado al entorno de desarrollo Visual Studio. Dentro de este directorio se irán almacenando los ficheros cabecera .hpp y fuente .cpp que definen cada una de las clases del diagrama de clases de diseño (all_vehicles_tracker, caracter, make_model, etc.) así como los archivos asociados a la biblioteca OpenSURF (fasthessian, surflib, etc.).

- *Debug / Release.* El framework Visual Studio permite configurar la manera de compilar los ficheros fuente con las clases del sistema. La opción "Debug" compila sin optimizar pero con toda la información de interés para el proceso de depuración (permite corregir errores en tiempo de ejecución). La opción "Release" compila el programa sin generar la información de depuración, por lo que está totalmente optimizada. Independientemente de la configuración establecida, al compilar el software se genera un ejecutable **Vehicle_Analysis.exe** preparado para funcionar. Para ello solo necesitamos pasarle la imagen desde la línea de comandos:

```
> Vehicle_Analysis.exe ../Files/image.jpg
```

La Figura 4.6 muestra el resultado obtenido al aplicar el software que se ha desarrollado sobre una imagen en la que se puede observar un automóvil de frente en la escena. El sistema localizará el vehículo y la matrícula del mismo para reconocer tanto los caracteres de la matrícula como la marca y el modelo del automóvil.

Figura 4.6: Sistema "Vehicle Analysis" aplicado sobre una imagen.

4.5. Bibliotecas

En este último apartado se van a analizar en detalle cada una de las bibliotecas referenciadas a lo largo del PFC, así como los métodos más importantes para el desarrollo de la aplicación.

4.5.1. OpenSURF

OpenSURF es una biblioteca libre implementada sobre C++ que permite generar el detector y el descriptor de características SURF (Speeded Up Robust Features) detallados en el Apartado 2.3.2. La idea en este proyecto es utilizar simplemente el descriptor de características ajustando una rejilla sobre la imagen para indicarle al programa las regiones consideradas de interés. La biblioteca que proporciona C. Evans [20] establece una serie de métodos interesantes para desarrollar la aplicación:

- **surfDes(IplImage *img, vector <Ipoint> &ipts, bool upright);** Función que se encarga de extraer de una imagen las características definidas en el vector los puntos de interés que establece la clase Ipoint. Por último, es posible configurar los descriptores SURF para hacerlos invariantes a posibles rotaciones en el objeto a clasificar.

- **drawIpoints(IplImage *img, vector <Ipoint> &ipts);** Método que dibuja en la imagen una circunferencia en cada región de interés con el objetivo de que el usuario visualice las características que extrae.

- **drawWindows(IplImage *img, vector <Ipoint> &ipts);** Procedimiento que permite visualizar la rejilla que se ajusta sobre la imagen con el frontal del automóvil que se está analizando.

4.5.2. OpenCV

OpenCV es una biblioteca libre de visión artificial desarrollada por Intel. Su licencia BSD (Berkeley Software Distribution) permite que sea usada libremente para propósitos comerciales, funcionando en cualquier plataforma Mac OS X, Windows, Linux e incluso Android a partir de su versión 2.2 (2011). Esta biblioteca proporciona un conjunto de algoritmos realmente interesantes relacionados con el área de la visión por computador que tratamos. A continuación se detallan algunas de las clases de OpenCV utilizadas en el proyecto así como los métodos más relevantes:

`High-level GUI and Media I/O (highgui)`

- *cvCaptureFromCAM.* La entrada de la aplicación es el vídeo que genera la webcam por defecto del ordenador.
- *cvCaptureFromAVI.* El sistema recibe la imagen o vídeo de entrada para el sistema.
- *cvGrabFrame.* Carga la imagen o un frame del vídeo para comenzar el procesamiento.
- *cvRetrieveFrame.* Permite extraer la imagen de tipo IplImage asociada a un frame concreto de la entrada al sistema.
- *cvReleaseCapture.* Libera la memoria que almacena el frame a procesar.
- *cvConvertImage.* Permite convertir las imágenes de RGB a escala de grises y viceversa.
- *cvNamedWindow.* Crea la ventana que actúa de interfaz en la aplicación.
- *cvShowImage.* Guarda la imagen correspondiente en el directorio indicado.
- *cvWaitKey.* Espera la pulsación de cualquier tecla un determinado tiempo.
- *cvDestroyWindow.* Libera el espacio reservado para la interfaz del sistema.

`The Core Functionality (cxcore)`

- *cvMinAreaRect2.* Encuentra el rectángulo de área mínima (Bounding Box) que mejor se ajusta a un conjunto de puntos dado.
- *cvBoxPoints.* Devuelve los vértices del rectángulo anterior.
- *cvDrawContours.* Colorea los contornos indicados (caracteres de la matrícula) con un determinado color.
- *cvFillPoly.* Función que rellena el área que determina un contorno con agujeros o intersecciones en su estructura.
- *cvPutText.* Dibuja una cadena de texto en la interfaz (frames por segundo).
- *cvRectangle.* Función que dibuja un rectángulo a partir de las coordenadas de 2 de sus esquinas opuestas.
- *cvLine.* Traza una línea en la interfaz definida por los 2 puntos de entrada.

- *cvPolyLine.* Dibuja una serie de curvas poligonales en la imagen.
- *cvSetImageROI.* Establece una región de interés (ROI) para una determinada imagen (procesar la ROI con el frontal de automóvil localizado).
- *cvCopy.* Permite realizar la copia de una imagen.
- *cvResetImageROI.* Libera la ROI establecida para trabajar nuevamente con la imagen completa como antes.
- *cvMemStorage.* Reservar espacio en memoria para realizar cualquier operación.
- *cvReleaseMemStorage.* Liberar el espacio de memoria reservado.

`Image Processing and Computer Vision (cv)`

- *CvHaarClassifierCascade.* Permite cargar los clasificadores xml encargados de localizar los objetos de interés en la imagen.
- *cvReleaseHaarClassifierCascade.* Liberar el espacio reservado para almacenar los clasificadores binarios.
- *cvEqualizeHist.* Realiza un proceso de ecualización del histograma sobre la imagen.
- *cvThreshold.* Umbralización de una imagen. Transformar los caracteres de la matrícula a binario.
- *cvGetPerspectiveTransform.* Transformación perspectiva entre puntos (homografía). Posiciona el automóvil para extraer sus características.
- *cvWarpPerspective.* Aplica la transformación geométrica sobre la imagen.
- *cvCvtColor.* Convierte la imagen de un espacio de color a otro.

`Feature Detection and Descriptor Extraction (features2d)`

- *cvMSERParams.* Configuración de los parámetros asociados a la técnica MSER.
- *cvExtractMSER.* Extraer los contornos con los caracteres de la matrícula (fuerte contraste entre los caracteres y el fondo blanco de la matrícula).

`Machine Learning (ml)`

- *CvRTParams.* Establece los parámetros que determinan el comportamiento del clasificador Random Forest.
- *CvRTrees.* Clase que permite realizar el entrenamiento y las pruebas para la lectura de la matrícula a partir de los Random Trees.
- *CvKNearest.* Clase que implementa el modelo KNN (K Nearest Neighbors) que determina el modelo que más se parece al automóvil a clasificar.

Capítulo 5

Conclusiones

En este último capítulo se presenta un resumen de los resultados obtenidos durante el desarrollo del PFC. A continuación se detalla cómo funciona el sistema para cada uno de los problemas planteados en el Capítulo 1. Finalmente, para seguir avanzando en el campo del reconocimiento de vehículos, se sugieren posibles mejoras y algunas ideas que podrían perfeccionar este sistema.

5.1. Discusión de los resultados

A través de los experimentos realizados a lo largo del PFC se han ido configurando las bases de la aplicación a desarrollar. Respecto a la etapa encargada de construir los clasificadores que permiten localizar los automóviles y las matrículas respectivamente, los resultados son realmente buenos (94 % de aciertos en ambos casos). Seleccionando los valores adecuados, somos capaces de procesar en tiempo real la imagen entera con una ventana deslizante para localizar los objetos de interés en la escena. Evidentemente el número de imágenes por segundo que caracteriza el procesamiento disminuye en proporción al número de automóviles que se van a reconocer (no es lo mismo procesar un vídeo con un sólo vehículo en la escena a tener que tratar varios a la vez).

Llegados a la fase de lectura de los caracteres de las matrículas, los resultados que se obtienen a partir de los experimentos cumplen los objetivos planteados en un principio (92 % de aciertos). No obstante, es interesante tener en cuenta que las imágenes de caracteres seleccionadas para formar la base de datos de entrenamiento son realmente sencillas y reflejan con claridad el caracter en cuestión, y sin embargo para las imágenes extraídas de la matrícula en un vídeo puede perderse bastante precisión (hay frames del vídeo en los que el movimiento de la cámara provoca que los carateres puedan aparecer borrosos). En estos casos la aplicación suele cometer errores al predecir algunos caracteres por lo que el sistema da la impresión de ser demasiado sensible al movimiento.

Por último, para la etapa de reconocimiento de marca y modelo el objetivo era mantener los resultados del PFC presentado en [12] (83 % de aciertos de clasificación). Sin embargo, hay que tener en cuenta que ahora para detectar el modelo es imprescindible haber localizado correctamente el automóvil y la matrícula del mismo, mientras que antes se marcaban a mano las esquinas de la matrícula directamente. Además, al localizar automáticamente la matrícula, puede disminuir la precisión del clasificador encargado de extraer las características del frontal.

Volviendo de nuevo a lo comentado en el Capítulo 1, donde se hacía referencia a los posibles problemas que se podían encontrar en la realización del proyecto, se va a describir cómo funcionará el sistema que hemos desarrollado para aquellas situaciones problemáticas que interesa controlar:

- *Oclusiones tapando el vehículo.* A la hora de construir el clasificador encargado de localizar automóviles vistos de frente, las imágenes de la base de datos que usa la aplicación son todas sin oclusiones, es decir que se puede ver perfectamente el vehículo. De esta forma, cuando en la imagen que se quiera identificar el coche esté parcialmente tapado por otro objeto, la probabilidad de que se localice correctamente el automóvil se verá disminuida (además si no se conoce la posición del frontal no podemos ni extraer las características del modelo ni identificar los caracteres de la matrícula).

- *País de los automóviles.* El sistema está entrenado para reconocer automóviles matriculados en España de forma que a priori el sistema detectará la posición del coche aunque probablemente se van a cometer errores durante el reconocimiento del modelo y la lectura de la matrícula (no podríamos identificar las vocales de la matrícula en un modelo extranjero).

- *Posible vehículo dañado.* Si la estructura del automóvil ha sufrido un accidente y está gravemente dañada evidentemente el sistema tampoco será capaz de localizar el frontal del vehículo. Hay que tener en cuenta que si la vista frontal del automóvil está visiblemente afectada, en caso de detectar el coche, el reconocimiento del modelo se complicaría demasiado.

- *Orientación de los automóviles.* Otro aspecto a destacar es la colocación del automóvil en la escena. Las imágenes de la base de datos de entrenamiento utilizan automóviles colocados todos en horizontal a una distancia relativa de la cámara. Al realizar las pruebas test encontramos problemas para detectar el frontal del automóvil cuando el vehículo se encuentra relativamente inclinado o la cámara está prácticamente pegada al capó del vehículo.

- *Cambios en la iluminación.* Es importante controlar los cambios en la luminosidad a la hora de realizar las fotos que queramos pasarle a la aplicación. No vale cualquier tipo de imagen en la que se vea un coche de frente. Cuanto mayor sea la visibilidad del coche, mejor será el rendimiento de los clasificadores encargados de extraer la información de los automóviles (evitar la oscuridad de la noche y los destellos de luz por el día).

Es interesante recordar mediante la Figura 1.3 qué tipo de imágenes podían presentar problemas a la hora de probar el funcionamiento del sistema. En principio, una vez desarrollado el software, es posible determinar que la aplicacíón es capaz de reconocer perfectamente automóviles similares a los presentados en la Figura 5.1 (a medida que el automóvil se vaya alejando de la cámara, irá disminuyendo la precisión de los clasificadores encargados de la lectura de la matrícula y el reconocimiento del modelo).

(a) (b) (c)

Figura 5.1: Imágenes habituales a la hora de reconocer los automóviles.

5.2. Conclusiones

El objetivo es verificar que el proyecto se ha realizado de acuerdo a lo esperado en el Capítulo 1, y para ello bastará con comprobar si han alcanzado con éxito los objetivos y subobjetivos que se marcaron:

- *Construcción de las bases de datos.* Para generar los diferentes clasificadores del sistema se van a seleccionar en torno a 13.000 imágenes en total para construir las diferentes bases de datos (entrenamiento y pruebas). La distribución de las mismas será: 6330 imágenes para construir el clasificador encargado de localizar cada automóvil visto de frente, 1825 imágenes para el clasificador que permite ubicar la matrícula dentro del frontal, 4500 imágenes para generar el clasificador de caracteres de la matrícula y por último 300 imágenes para el clasificador de marca y modelo.

- *Desarrollar los algoritmos de clasificación oportunos.* A lo largo del Capítulo 2 se han especificado en detalle el funcionamiento de los algoritmos empleados durante el desarrollo del sistema. Para localizar los objetos de interés de la aplicación (auotmóviles y matrículas) la idea es emplear la técnica Haarcascade que porporciona OpenCV (Ventana deslizante + Filtros Haar + Cascada de clasificadores con AdaBoost). Respecto a clasificador encargado de leer la matrícula el objetivo es extraer los caracteres con la técnica MSER para reconocer posteriormente con un clasificador basado en Random Forest la letra o dígito en cuestión.

- *Integrar el clasificador de marca y modelo.* La idea es aplicar con éxito el clasificador de marca y modelo presentado en el PFC anterior [12]. Como el proyecto está desarrollado sobre la misma versión de OpenCV no se han encontrado dificultades para acoplar las clases al sistema.

- *Evaluación del rendimiento de los clasificadores.* En el Capítulo 3 de experimentos evaluamos el rendimiento de cada uno de los clasificadores generados para comprobar que los resultados satisfacen los objetivos planteados en un principio.

- *Implementación en C++ de la interfaz.* Construimos una interfaz gráfica usando el lenguaje C++ y la biblioteca OpenCV. El usuario de la aplicación podrá cargar una imagen o vídeo con el frontal de al menos un automóvil, y el sistema clasificará la matrícula y el modelo de cada vehículo sin necesidad de indicarle a la aplicación donde se encuentran estos automóviles en cuestión.

5.3. Trabajos futuros

El trabajo realizado en este PFC asociado al reconocimiento automático de automóviles introducidos por vídeo (lectura de la matrícula y identificación de la marca y modelo) es solamente el comienzo de un proyecto que tiene un gran margen de mejora. Hay muchos aspectos en los que poder trabajar en un futuro para complementar esta aplicación. En este apartado concretamente se van a tratar de enumerar algunas de estas posibles mejoras:

- *Vista lateral y trasera de los automóviles.* La idea es crear nuevas bases de datos con diferentes vistas de los automóviles para entrenar un clasificador que permita reconocer la colocación en la que se está visualizando el vehículo. Clasificar el automóvil desde distintas perspectivas permitiría además extraer nuevas características, quizás más discriminantes para reconocer la marca y modelo que las obtenidas de las imágenes frontales.

- *Realizar más pruebas con diferentes tipos de clasificadores.* Una de las piezas fundamentales que ayudan al buen funcionamiento del sistema desarrollado es la de utilizar los clasificadores detallados en el Capítulo 3. No obstante, sería interesante probar con nuevas herramientas para analizar si se obtienen mejores resultados. Por ejemplo, el PFC desarrollado por Miguel Ángel Fernández en [13] propone el algoritmo RANSAC (Random Sample Consensus) para detectar las líneas de texto de las matrículas sin necesidad de que la placa esté colocada horizontalmente.

- *Reconocer el tipo de vehículo a procesar.* Otro aspecto interesante sería localizar cualquier tipo de vehículo en la escena para clasificarlo posteriormente como automóvil, camión, furgoneta, autobús, etc. La idea sería generar nuevas bases de datos con los vehículos que queremos buscar para aumentar el dominio del sistema. Respecto a la lectura de las matrículas en estos casos, el rendimiento del clasificador debería ser similar principalmente porque el algoritmo implementado no depende de las características del frontal del vehículo.

- *Analizar otras características de interés del frontal.* Llegado este momento el sistema es capaz de extraer características del frontal del automóvil para leer los caracteres de su matrícula y reconocer su marca y modelo correspondiente. No obstante, pueden incorporarse nuevas funcionalidades al sistema (como verificar si el vehículo lleva las luces encendidas o si el conductor lleva puesto el cinturón de seguridad, ...) para mejorar la aplicación.

- *Independencia de la plataforma y Android.* La última mejora que proponemos sería intentar reimplementar el proyecto que se ha desarrollado sobre OpenCV en un entorno Windows, para poder ejecutar el sistema en cualquier dispositivo. El objetivo sería intentar apoyarnos de alguna manera en el lenguaje de programación Java para independizar el proyecto de la plataforma. Otra posibilidad es llevar el desarrollo de la aplicación al ámbito de los dispositivos móviles por medio del sistema operativo Android (OpenCV incorpora desde su versión 2.2 la posibilidad de programar aplicaciones asociadas a la visión artificial en Android).

Bibliografía

[1] Michael McCahill y Clive Norris *CCTV in London*, Centre for Criminology and Criminal Justice, University of Hull, 2002.

[2] Paul Viola y M. Jones, *Rapid object detection using a boosted cascade of simple features*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2001, Pag: 511-518.

[3] Paul Viola y M. Jones, *Robust real-time object detection*, International Journal of Computer Vision (IJCV), July 2004, Pag: 137-154.

[4] Yoav Freund y Robert E. Schapire, *Experiments with a New Boosting Algorithm*, International Conference on Machine Learning (ICML), 1996.

[5] John Ross Quinlan, *Induction of Decision Trees*, Informe Técnico, 1986, Pag: 81-106

[6] J. Matas, O. Chum, M. Urban, y T. Pajdla, *Robust wide baseline stereo from maximally stable extremal regions*, In British Machine Vision Conference (BMVC), 2002, Pag: 384-393.

[7] Leo Breiman y Adele Cutler *Random Forest*, Informe Técnico, University of California, 2001, Pag: 5-32.

[8] Bradley Efron y Robert Tibshirani *An Introduction to the Bootstrap*, Informe Técnico, Stanford University, 1994.

[9] Rich Caruana y N. Karampatziakis *An empirical evaluation of supervised learning in high dimensions*, International Conference on Machine Learning (ICML), 2008, Pag: 96-103.

[10] Naotoshi Seo, *Rapid object detection with a cascade of boosted classifiers based on haar-like features*, 2008.

http://note.sonots.com/SciSoftware/haartraining.html

[11] Toby Breckon, *Machine learning: Optical digits example*, 2011.

http://public.cranfield.ac.uk/c5354/teaching/ml/

[12] Roberto Valle, *Reconocimiento de modelo y marca de automóviles*, Proyecto Fin de Carrera (PFC), Ingeniería Técnica Informática de Gestión (URJC), Tutor: José Miguel Buenaposada, 2010.

[13] Miguel Ángel Fernández, *Detección de matrículas en imágenes*, Proyecto Fin de Carrera (PFC), Ingeniería Técnica Informática de Sistemas (URJC), Tutor: José Miguel Buenaposada, 2009.

[14] Michal Conos, *Recognition of vehicle make from a frontal view*, 2007.

[15] David G. Lowe *Object Recognition from Local Scale-Invariant Features*, International Conference on Computer Vision (ICCV), 1999, Pag:1150-1157.

[16] Herbert Bay, Tinne Tuytelaars y Luc Van Gool *SURF: Speeded Up Robust Features*, Computer Vision and Image Understanding (CVIU), 2008, Page:346-359.

[17] Oren B., Eli S. y M. Irani *In Defense of Nearest-Neighbor Based Image Classification*, Computer Vision and Pattern Recognition (CVPR), 2008, Page:1-8.

[18] David M. Mount *ANN Programming Manual*, University of Maryland, 1998.

[19] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

[20] Christopher Evans *Notes on the OpenSURF Library*, University of Bristol, 2009.

[21] Gady Agam *Introduction to programming with OpenCV*, Department of Computer Science, 2006.

http://www.cs.iit.edu/ãgam/cs512/lect-notes/opencv-intro/opencv-intro.html