Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Ingeniería Técnica en Informática de Gestión

Curso Académico 2009/2010

Reconocimiento de modelo y marca de automóviles

Proyecto Fin de Carrera

Autor: Roberto Valle Fernández
Tutor: José Miguel Buenaposada Biencinto

19 de enero de 2010

Agradecimientos

Me gustaría dar las gracias a las siguientes personas que me han ayudado en el desarrollo del proyecto fin de carrera (PFC):

- A mi tutor José Miguel Buenaposada Biencinto por cada uno de los consejos que me ha dado, porque siempre ha tenido una respuesta para las muchas preguntas que le he hecho, y sobre todo porque siempre confió en mí, incluso en esos días en los que la desesperación me impedía ver el final del proyecto. 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 y la confianza que mostraron en mi trabajo. Sobre todo a tí Sergio Báez, por todo ese tiempo en el que trabajamos juntos desde el instituto, y porque seguramente si no fuese por tu ayuda, nunca hubiese llegado aquí.

- Gracias de corazón a todos mis amigos de siempre por hacerme feliz, a todos los que me quieren tal y como soy, porque eso me hace más fuerte. Siempre os llevaré en mi corazón pase lo que pase.

- Y por último, quiero dedicarle este trabajo y el título de ingeniero técnico en informática a mi familia, por todos los ánimos que me dieron. 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.

Resumen

En los últimos años, se han empezado a desarrollar sistemas cada vez más sofisticados para el reconocimiento automático de vehículos a través de secuencias de vídeo o imágenes. Este tipo de aplicaciones resultan de gran utilidad para aquellas empresas que están buscando, por ejemplo la manera de localizar la matrícula de un vehículo para recuperar sus caracteres, o que necesitan reconocer la marca y el modelo del automóvil en cuestión.

Con la realización de este proyecto fin de carrera (PFC) buscamos **construir una aplicación informática que consiga identificar la marca y el modelo de un vehículo a través de una imagen frontal del mismo**. La aplicación servirá por ejemplo para evitar los robos en zonas como parkings que tienen sistemas de reconocimiento de matrículas, de tal forma que, aún intercambiando las matrículas entre coches de modelos distintos, el programa nos detecte que la matrícula del automóvil que sale no coincide con la del coche que clasificamos al entrar. El sistema también podría servir para clasificar el vehículo de un conductor que ha cometido una infracción, y ha sustituido la matrícula para que no pueda ser localizado (la policía actuaría en consecuencia atrapando al conductor del automóvil).

Tras realizar una serie de experimentos para obtener la mayor tasa de aciertos posible, la solución propuesta hará uso de: una base de datos que construiremos con al menos 50 modelos distintos de coches, en total serán unas 300 imágenes; un método para rectificar a una dimensión de 256 x 92 píxeles las imágenes que se usen; el descriptor SURF para extraer las características de las imágenes; y un clasificador que desarrollaremos basándonos en las distancias que hay entre imágenes, para encontrar la clase a la que pertenece la imagen que queramos identificar.

La tasa de acierto de marcas reconocidas alcanzada por nuestro sistema es de un 83 % si se quiere clasificar un automóvil con el modelo de la base de datos que más se parece. Esta **tasa de aciertos llega a un 94 %** cuando tenemos en cuenta los cinco modelos a los que más se parece el automóvil a identificar (se considera como acierto si la marca y el modelo del automóvil coincide con la de alguno de los cinco modelos que menos distancia tienen). La aplicación se ha desarrollado bajo un entorno de trabajo Windows, con el lenguaje C++, apoyándonos en las bibliotecas: OpenSURF, ANN y OpenCV.

Índice general

1. Introducción **1**
- 1.1. Objetivos . . . 2
- 1.2. Posibles problemas . . . 2
- 1.3. Estructura del proyecto fin de carrera . . . 4

2. Estado del arte **5**
- 2.1. Descripción de los trabajos anteriores . . . 5
  - 2.1.1. Tesis de máster de Michal Conos [1] . . . 5
  - 2.1.2. Tesis de máster de Louka Dlagnekov [2] . . . 6

3. Algoritmos utilizados **7**
- 3.1. Localizar la matrícula . . . 8
- 3.2. Rectificar la imagen . . . 9
- 3.3. Extracción de las características . . . 10
- 3.4. Clasificador de marca y modelo . . . 12
- 3.5. Verificación de modelos de automóviles . . . 14

4. Resultados experimentales **17**
- 4.1. Construcción de la base de datos . . . 17
- 4.2. Metodología de los experimentos . . . 18
- 4.3. Tamaño de la rejilla . . . 19
  - 4.3.1. Rejillas sin solapamiento . . . 20
  - 4.3.2. Rejillas con solapamiento . . . 22
- 4.4. Desplazar las celdas de la rejilla . . . 23
- 4.5. Los N siguientes modelos más parecidos . . . 26
- 4.6. Distancias a otros modelos de automóviles . . . 27
- 4.7. Técnica de verificación . . . 28

5. Descripción informática **31**
- 5.1. Metodología y plan de trabajo . . . 31
- 5.2. Captura de requisitos . . . 32
  - 5.2.1. Requisitos funcionales . . . 32
  - 5.2.2. Requisitos no funcionales . . . 33
- 5.3. Análisis y diseño . . . 33
- 5.4. Implementación . . . 35
- 5.5. Bibliotecas . . . 37
  - 5.5.1. OpenSURF . . . 37
  - 5.5.2. ANN . . . 38
  - 5.5.3. OpenCV . . . 38

6. Conclusiones **41**
6.1. Discusión de los resultados 41
6.2. Conclusiones 42
6.3. Trabajos futuros 43

Índice de figuras

1.1. Ejemplo de utilización de ANPR. 1
1.2. Imágenes a evitar en el sistema que diseñemos. 3

3.1. Esquema que resume el método de entrenamiento utilizado. 7
3.2. Esquema que resume el método de clasificación utilizado. 8
3.3. Localización de la matrícula en la imagen de entrada. 8
3.4. Método usado para colocar la matrícula en la imagen rectificada. 9
3.5. Ejemplo de imagen en la que se corrige la orientación del vehículo. 9
3.6. Ejemplos de imágenes rectificadas: a), c) y e) en b), d) y f) respectivamente 10
3.7. Imágenes rectificadas con las matrículas en blanco. 10
3.8. Extracción de las características de una imagen. 11
3.9. Funcionamiento y aplicaciones de OpenSURF. 12
3.10. Extracción de características usando una rejilla. 12
3.11. Esquema que muestra como se separan los modelos en distintas clases. . . . 14
3.12. Esquema que muestra la estructura de la imagen a identificar. 14
3.13. Esquema que resume el método de verificación de modelos entre imágenes. 15

4.1. Imágenes de las rejillas sin solapamiento. 20
4.2. Combinación de las rejillas de 46 x 46 y 23 x 23. 21
4.3. Imágenes de las rejillas con solapamiento. 22
4.4. Distintos desplazamientos entre celdas para una rejilla de 46 x 46. 23
4.5. Imágenes de las rejillas con el desplazamiento de las celdas. 24
4.6. Distancias entre imagen y clases: (a) Mismo modelo (b) Modelos distintos. 27
4.7. Histograma de frecuencias. (Distancias a otros modelos). 28
4.8. Distancias entre imágenes: (a) Mismo modelo (b) Modelos distintos. 29
4.9. Histograma de frecuencias. (Técnica de verificación). 29

5.1. Diseño de la aplicación. 34
5.2. Diagrama de secuencia del sistema. 35
5.3. Árbol con los directorios de la aplicación. 36

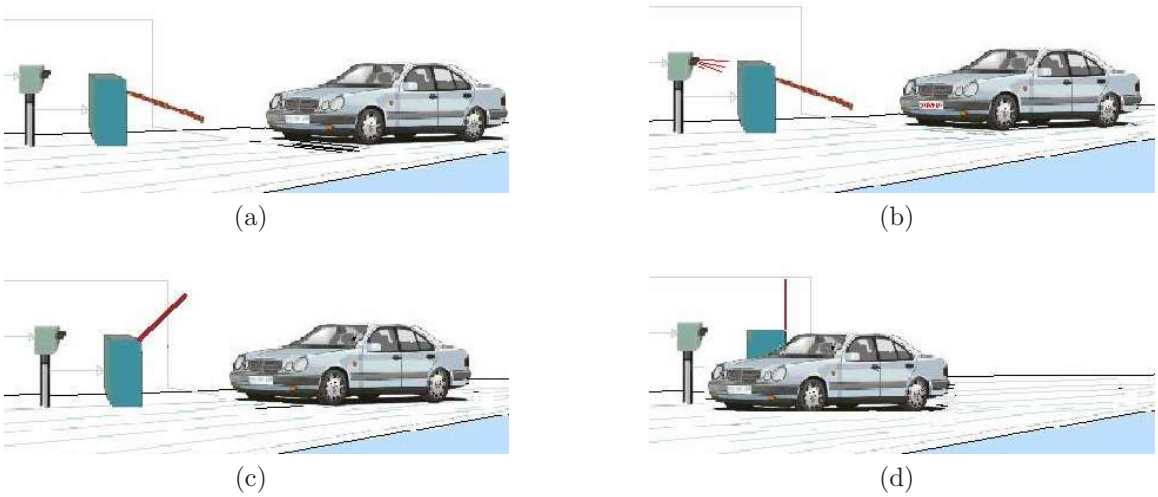
6.1. Método propuesto para la clasificación de imágenes. 41

Capítulo 1

Introducción

En estos últimos años la importancia de los métodos de autentificación y reconocimiento de vehículos a aumentado considerablemente. Cada vez son más los países que utilizan este tipo de sistemas para solucionar problemas como: el robo de vehículos, el control de acceso de coches a una zona, detectar 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. Si alguna de las cámaras de vigilancia que tienen repartidas por la ciudad (CCTV Closed-Circuit Television [11]) identifica un vehiculo del que no consta que pague dicha tarifa, el propietario del vehículo será localizado y amonestado.

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 números que contenga. A este tipo de aplicaciones se las conoce como ANPR (Automatic Number Plate Recognition) y suelen conseguir resultados bastante buenos. Como se ha comentado anteriormente, este tipo de sistemas de reconocimiento se utilizan en multitud de situaciones. En la Figura 1.1 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 1.1: Ejemplo de utilización de ANPR.

Llegados a este punto, nos podemos dar cuenta de que realmente este tipo de aplicaciones no clasifican el vehículo por la clase de coche, sino por lo que pone en su matrícula (necesitaremos consultar una base de datos que almacene la marca y el modelo que tiene cada vehículo asociándolo a su matrícula). Normalmente con identificar la matrícula se podrá clasificar el automóvil, salvo en los casos en lo que se hayan intercambiado las matrículas entre coches. Además, hay aplicaciones que ni siquiera serán capaces de acceder a las bases de datos de tráfico, y por tanto no podrán consultar el vehículo que hay asociado a una matrícula determinada.

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 nos devuelva automáticamente la marca y el modelo de un vehículo a partir de una imagen frontal del mismo. La aplicación le permitirá al usuario situar la matrícula en la imagen para que el sistema pueda conocer la ubicación del coche. Para llegar a este objetivo final, es necesario alcanzar los siguientes objetivos parciales:

- *Construcción de la base de datos.* Es imprescindible crear una base de datos amplia con vehículos de varios modelos distintos para poder alcanzar los siguientes objetivos. El clasificador que construyamos se probará mejor cuantos más modelos consigamos formar. Estas mismas imágenes también las podremos utilizar para realizar los experimentos en el desarrollo del PFC.

- *Desarrollar un clasificador de imágenes de automóviles en base a su marca y modelo.* La idea es diseñar un algoritmo para el reconocimiento de patrones que me permita clasificar, a partir de la imagen frontal de un automóvil, la marca y el modelo del mismo.

- *Evaluación del rendimiento del clasificador.* Tras realizar los experimentos que se muestran en el Capítulo 4 debemos evaluar los resultados obtenidos.

- *Implementación en C++ del clasificador.* El objetivo es trasladar a C++ [16] los resultados que se obtienen en las fases anteriores del PFC. El usuario debe ser capaz de poder cargar una imagen frontal de un coche en la aplicación, y que esta nos clasifique la marca del automóvil.

1.2. Posibles problemas

Los seres humanos en general somos mejores recordando la apariencia o la marca de los vehículos, que memorizando el contenido de sus matrículas. El sistema de reconocimiento que proponemos en este PFC se basa en esta idea de identificar los vehículos extrayendo las características de la vista frontal del automóvil. Esto implicará que debamos tener especial cuidado con las imágenes que usemos durante el desarrollo de este sistema. Enumeramos algunas de estas situaciones de peligro:

- *Oclusiones que tapen el vehículo.* En los experimentos que se realizan en el PFC se ha intentado que el vehículo a identificar se vea lo más claramente posible en la imagen.

Aunque en este proyecto no se considera, en la vida real ocurren este tipo de circunstancias (oclusiones, niebla, etc) por lo que un sistema comercial debería funcionar correctamente en estas situaciones.

- *País de los modelos.* Algunas marcas sacan versiones distintas del mismo modelo dependiendo del país donde se vaya a utilizar. Tenemos que controlar que las imágenes que coleccionemos para la base de datos no sean muy diferentes para un mismo modelo. En los ANPR, el problema que supone que cambien las matrículas dependiendo del país, es mucho mayor pues normalmente los formatos son distintos. En este PFC se han elegido modelos comercializados en España.

- *Número de vehículos en la imagen.* Es posible que en la imagen que le pasemos a la aplicación haya más de un vehículo que puede ser clasificado. Se ha de valorar que debería de ocurrir en este tipo de situaciones. En este proyecto consideramos que los automóviles que se quieran clasificar estarán localizados manualmente en la imagen.

- *Cambios en la luminosidad.* Como se ha indicado anteriormente, lo mas importante es que el vehículo que queramos identificar se vea claramente, por eso siempre que podamos hemos de conseguir fotos de buena calidad (sin grandes contrastes de luz). De cualquier manera, en es PFC no se realizarán experimentos con imágenes que presenten estos problemas.

- *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.

En la Figura 1.2 se observan varios ejemplos de imágenes problemáticas para la aplicación. En el Capítulo 6 detallaremos el comportamiento del sistema para cada uno de estos casos que presentan problemas. Por último, no debemos olvidar que a la hora de implementar la aplicación debemos optimizarla lo máximo posible, de forma que, identificar la marca de un vehículo a partir de una imagen frontal del mismo, no nos lleve más de 10 segundos en tiempo de ejecución (este tiempo dependerá de la cantidad de imágenes que haya en la base de datos, cuyas características habrá que leer en el clasificador).

(a) (b) (c)

Figura 1.2: Imágenes a evitar en el sistema que diseñemos.

1.3. Estructura del proyecto fin de carrera

El *Capítulo 2* ofrece un estudio del nivel actual de desarrollo que han alcanzado los sistemas de reconocimiento de vehículos. Repasamos además el funcionamiento de los sistemas que se han usado como base en la aplicación diseñada en este PFC.

El *Capítulo 3* detalla el funcionamiento de los procedimientos que se usan a lo largo del proyecto. Describe los métodos propuestos para rectificar imágenes, extraer las características de estas y clasificarlas por la marca y el modelo.

En el *Capítulo 4* se realizan los experimentos del proyecto, con el objetivo de obtener una buena tasa de reconocimiento de vehículos.

En el *Capítulo 5* se especifica el plan de trabajo y la metodología que se ha seguido en la elaboración del sistema. Usaremos además UML para detallar algunos diagramas que faciliten la comprensión de cómo se debe usar la aplicación.

El *Capítulo 6* presenta las conclusiones del PFC.

Capítulo 2

Estado del arte

Basándonos en los artículos y libros que se han consultado para la realización del PFC, lo que buscamos en este apartado es detallar las técnicas de reconocimiento de marcas y modelos de vehículos que han seguido sus autores.

2.1. Descripción de los trabajos anteriores

2.1.1. Tesis de máster de Michal Conos [1]

Esta tesis trata de clasificar la marca y el modelo de un vehículo, a través de su vista frontal. El proceso de reconocimiento que propone se detalla en los siguientes apartados:

- *Normalización de las imágenes.* En este sistema se utilizan dos bases de datos distintas de vehículos. Una de ellas contiene 100 imágenes frontales de coches de distintos modelos (usa 10 modelos en total: Opel, Skoda, Fiat, etc); la otra base de datos contiene 500 imágenes frontales de camiones (7 modelos de camiones: Iveco, Mercedes, etc). Todas las imágenes de ambas bases de datos se rectifican a un tamaño de 256 x 92 píxeles. Con respecto a la locacalización del vehículo en la imagen, el sistema confía en la detección de la matrícula previamente.

- *Extracción de las características.* La características de la imagen normalizada se extraen utilizando el descriptor local SIFT. Durante el capítulo de experimentos realizan una serie de pruebas con distintas representaciones de SIFT: SIFT-Ori, SIFT-Grad y SIFT-GradWei (analizadas las 3 sin solapamientos y con solapamientos). Para definir los punto de interés se coloca una rejilla encima de la imagen (las celdas actuarán como regiones de interés).

- *Construcción del clasificador.* En este último apartado realizan prubas con dos métodos de clasificación distintos: k-NN (k Nearest Neighbour), con el que obtienen los mejores resultados, y FLD (Fisher's Linear Discriminant Analysis).

En el mejor de los casos, se consigue un 93.75 % de aciertos en la clasificación de automóviles, utilizando SIFT-GradWei y el clasificador k-NN. La rejilla que se coloca para conseguir este resultado es de 19 x 13 celdas. Con respecto a la clasificación de camiones, la tasa de aciertos es de un 89.13 % en el mejor de los casos, utilizando para ello una rejilla de 15 x 9 celdas.

Por último se realizan una serie de experimentos con imágenes en las que tenemos diferentes problemas: se comprueba la sensibilidad del sistema con imágenes borrosas, se estudia como influye una reducción de imágenes a la hora de construir el clasificador, sensibilidad añadiendo ruido a las imágenes, etc. En todos los casos problemáticos la tasa de aciertos se ve disminuida.

2.1.2. Tesis de máster de Louka Dlagnekov [2]

Esta otra tesis de máster, a diferencia de la anterior, va a presentar un sistema que combina tanto la localización y la identificación de la matrícula, como la clasificación del vehículo con su marca y modelo.

Con respecto a la parte de localización y reconocimiento de la matrícula, comentar que utiliza un sistema que trabaja en tiempo real. En el proyecto que estamos desarrollando en este PFC, aunque si utilizamos la matrícula para localizar el automóvil en la imagen, no necesitaremos reconocer sus caracteres, por lo que no vamos a entrar a detallar más esta parte de la tesis de máster.

Se propone el método que desarrollamos en los siguientes apartados, para la parte de reconocimiento de marca y modelo de un vehículo:

- *Normalización de las imágenes.* Utiliza una base de datos con 1140 imágenes frontales de automóviles en las cuales tenemos 38 modelos distintos. En total utiliza 30 imágenes para construir cada modelo. El tamaño de las imágenes rectificadas que se utiliza es de 400 x 200 píxeles, con la matrícula correctamente situada en el medio de la imagen.

- *Extracción de las características y clasificación.* A la hora de extraer las características de las imágenes normalizadas, se realizan una serie de experimentos con distintos tipos de descriptores:

    1. Método basado en la apariencia del automóvil. Trata la imagen entera M x N como un vector de características. Utiliza PCA (Principal Component Analysis) para reducir la dimensión del vector M x N que se genera. Utilizando esta técnica se obtiene una tasa de aciertos del 23.7 % .
    2. Método basado en las características del automóvil. Funciona localizando primero las características más importantes del vehículo en la imagen, para después describir cada uno de estos puntos de interés. Utiliza el detector y el descriptor de imágenes que proporciona SIFT. Tras realizar los experimentos, obtienen una tasa de aciertos del 89.5 %.

Capítulo 3

Algoritmos utilizados

Para hacer funcionar nuestro sistema se han usado una serie de algoritmos cuyo funcionamiento se entrará a detallar en cada una de las secciones de este capítulo, y cuyos resultados se reflejan en el Capítulo 4.

Antes se va a analizar el método de entrenamiento que se ha seguido durante el desarrollo del proyecto para construir el clasificador. Se analiza la etapa que denominamos **entrenamiento de las imágenes del sistema**, y para entender su funcionamiento se ha realizado el esquema de la Figura 3.1 que detalla los algoritmos que necesita para funcionar.

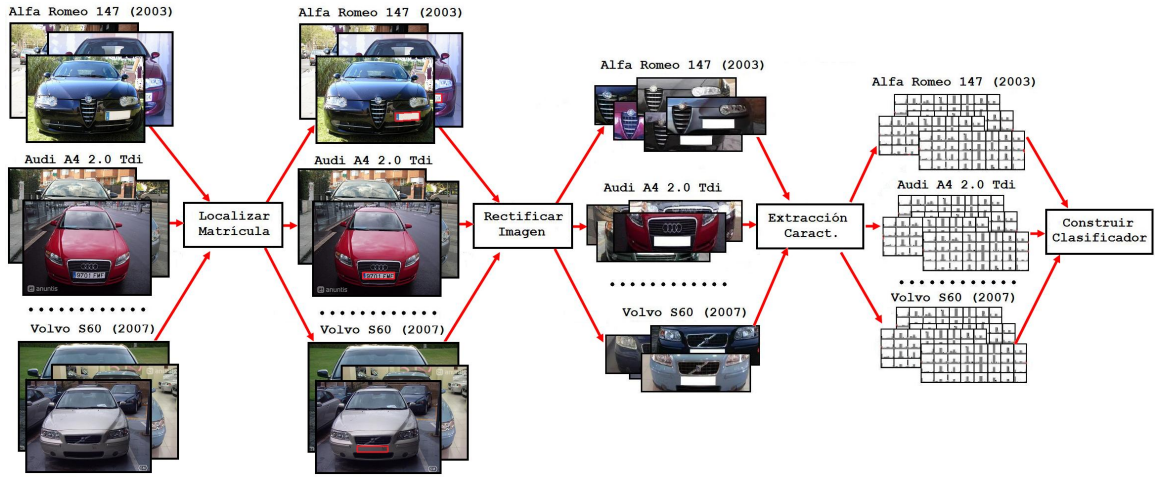


Figura 3.1: Esquema que resume el método de entrenamiento utilizado.

La aplicación actuará conociendo la situación del automóvil a partir de la localización de la matrícula (tenemos apuntadas las coordenadas de la matrícula en un fichero de texto aparte), y devolverá como resultado las imágenes frontales del vehículo rectificadas. En esta imagen normalizada, la matrícula se colocará en una posición que establecemos siguiendo los criterios que se detallan en [1] (en la Sección 3.1 se entra más en detalle).

Tras separar dichas imágenes en distintas clases dependiendo del modelo del automóvil (Alfa Romeo, Audi, ...), se pasará a la etapa en la que se describen estas imágenes corregidas, con el objetivo de obtener un vector de características de 64 componentes para cada clase. Estos vectores serán los que utilizemos para construir el clasificador.

Una vez construido el clasificador que se utilizará en la aplicación, entramos en la etapa de **clasificación de un automóvil**. El objetivo es que el sistema permita identificar la marca y el modelo de un automóvil dada una imagen frontal del mismo. Una vez se consigan extraer las características de la imagen, usaremos el clasificador que hemos creado en la etapa anterior para determinar su marca y modelo. En este método de clasificación que detallamos no podremos tener anotadas las coordenadas de las matrículas de los coches, por lo que el diseño del algoritmo para la localización de la matrícula será distinto. En la Figura 3.2 observamos las etapas genéricas para la clasificación de un automóvil.

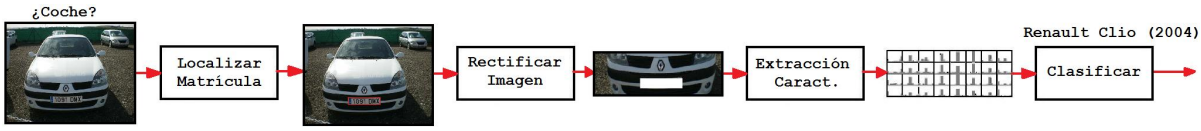


Figura 3.2: Esquema que resume el método de clasificación utilizado.

Llegados a este punto, vamos a detallar en las siguientes secciones los algoritmos seguidos en las etapas de entrenamiento y clasificación.

3.1. Localizar la matrícula

Tanto en la etapa de entrenamiento y construcción del clasificador, como en la etapa de clasificación del vehículo, se necesitará de un sistema que reconozca las esquinas de la matrícula para poder localizar el automóvil en la imagen. Las esquinas de la matrícula se marcarán tras cargar la imagen que se vaya a identificar. Para que no haya ambigüedades con respecto al orden en el que se pueden seleccionar las esquinas, le permitiremos al usuario dibujar un rectángulo alrededor de la matrícula para posteriormente, en la implementación del algoritmo, obtener las coordenadas de las esquinas de este rectángulo que se ha dibujado. En el ejemplo de la Figura 3.3 se puede observar el orden que seguimos para señalar las esquinas de la matrícula durante el desarrollo del PFC. Una vez tenemos localizado el automóvil en la imagen de entrada podemos pasar a normalizar la imagen.

Figura 3.3: Localización de la matrícula en la imagen de entrada.

3.2. Rectificar la imagen

Lo primero que hay que explicar es el objetivo que queremos alcanzar normalizando las imágenes, qué método se ha seguido para conseguirlo y qué características tendrá la imagen que obtengamos como resultado. Para ello, es necesario saber que las imágenes de los coches que se utilizan en la aplicación pueden tener distintos tamaños y distintas resoluciones (tanto las imágenes que usamos para construir la base de datos, como las que el usuario de la aplicación quiera clasificar).

Necesitamos establecer un mismo tamaño para todas las imágenes rectificadas, de tal forma que al aplicar el descriptor sobre estas, tengamos colocado siempre el automóvil en la misma posición. Para ello es necesario determinar la posición el la que colocaremos la matrícula del coche en la imagen normalizada. Siguiendo la estructura de los experimentos que hacían los autores de [1] se elige un tamaño de 256 x 92 para las imágenes rectificadas, y se centrará la matrícula en la posición definida por las coordenadas [92.48,54.86], [175.06,54.86], [92.48,75.96] y [175.06,75.96]. Podemos calcular estos puntos siguiendo la fórmula que muestra el dibujo de la Figura 3.4.

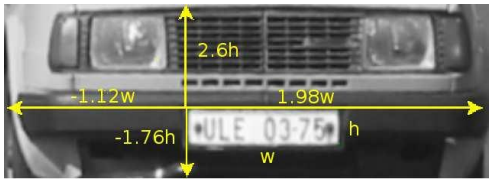


Figura 3.4: Método usado para colocar la matrícula en la imagen rectificada.

Es muy probable que el automóvil que vayamos a identificar no ocupe la imagen de entrada en su totalidad, por lo que necesitaremos recortar la imagen frontal de ese coche. Con el sistema de corrección de imágenes que se va a construir, trataremos además de evitar los posibles problemas de orientación que pueda presentar el vehículo que se quiere clasificar. Para el caso de que la orientación del automóvil no sea frontal a la cámara que captura la imagen, aplicaremos una transfomación geométrica (basada en la homografía [22] y [23]) que nos ayude a acercar y orientar la imagen (en la Figura 3.5 se muestra un ejemplo).

(a) (b)

Figura 3.5: Ejemplo de imagen en la que se corrige la orientación del vehículo.

Tenemos algunos ejemplos en la Figura 3.6 con imágenes normalizadas que utilizamos en la base de datos de la aplicación (es especialmente importante darse cuenta de como las matrículas se sitúan en el mismo lugar en los tres casos).

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

Figura 3.6: Ejemplos de imágenes rectificadas: a), c) y e) en b), d) y f) respectivamente

El último cambio que hemos realizado en nuestras imágenes es el de poner en blanco la matrícula en las imágenes ya recortadas para que su contenido no afecte en la clasificación del automóvil. De esta manera cuando el usuario necesite identificar la foto de un coche, el sistema no se verá influido por el color o el contenido de la matrícula (la mayoría de las imágenes tienen la matrícula borrosa porque el dueño no quiere que se vea, y otras tantas tienen carteles encima por ejemplo para anunciar su venta). Las imágenes que se mostraban anteriormente podemos observarlas de nuevo en la Figura 3.7, ahora sí con la matrícula en blanco.

(a) (b) (c)

Figura 3.7: Imágenes rectificadas con las matrículas en blanco.

3.3. Extracción de las características

El objetivo de esta sección es buscar la forma de describir cada una de las imágenes rectificadas con las que trabaja la aplicación para que se puedan diferenciar los distintos modelos de automóviles. Primero vamos a analizar el funcionamiento de los dos descriptores locales más importantes aparecidos en al literatura:

- *Scale-Invariant Feature Transform (SIFT).* Es un algoritmo desarrollado por David Lowe en 1999 [3], que se usa para detectar y describir características de una imagen (por ejemplo la imagen de la Figura 3.8). A diferencia de otros detectores de puntos de interés, el detector de SIFT es muy poco sensible frente a cambios en el tamaño de la imagen, posibles rotaciones de la misma y grandes cambios en la ilumniación. El principal problema que plantea SIFT es que el algoritmo esta patentado y por tanto no se puede utilizar en aplicaciones comerciales.

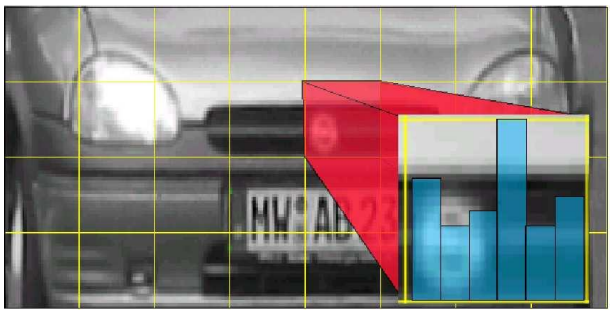


Figura 3.8: Extracción de las características de una imagen.

El proceso de extracción de las características de SIFT consta de tres pasos: detectar los puntos de interés de una imagen, asignar la orientación del gradiente a cada región de interés detectada y describir cada una de estas regiones de interés. La idea del algoritmo es dividir en 16 partes la región que usa para describir, y luego asignar a cada una de estas subregiones las orientaciones de los gradientes para cada casilla del octante que se crea. El resultado que obtenemos es un vector de 128 componentes (16 x 8 subregiones) que caracterizan esa región de interés.

- *Speeded Up Robust Features (SURF).* Este será el descriptor por el que nos decantaremos en este proyecto. Presentado por Herbert Bay, Tinne Tuytelaars y Luc Van Gool en 2006 [4], SURF presenta numerosas ventajas con respecto a SIFT en su implementación citeSu06. La característica más importante que tiene SURF con respecto a SIFT es su rapidez en ejecución. SURF es bastante más rápido que SIFT, sin pérdida de rendimiento. Para la aplicación que se diseñará, se va a utilizar la biblioteca de OpenSURF [6] que podemos obtener libremente a través de Internet. Si miramos la Figura 3.9, podemos observar dos imágenes que nos ayudarán a entender mejor el funcionamiento de OpenSURF y las posibilidades que ofrece a nuestro sistema.

La Figura 3.9 (a) muestra gráficamente qué significa el descriptor de un punto de interés en una orientación concreta. OpenSURF divide la región a describir en 16 subregiones, y cada una de estas subregiones la subdivide a su vez en otras 4 subsubregiones más pequeñas (si hacemos un cálculo del número de celdas que tiene ahora el descriptor veremos que tiene 16 x 4 = 64 subsubregiones). Para cada casilla del descriptor, calcula la orientación del gradiente basándose en las propiedades de la imagen cuyas características se están extrayendo, y finalmente devuelve un vector de 64 componentes con las características.

La imagen de la Figura 3.9 (b) nos muestra el funcionamiento del detector de puntos de interés que proporciona OpenSURF en su implementación. Probamos el detector en una imagen con la vista frontal de un coche para observar que regiones detecta el sistema como importantes. Mirando a la imagen, vemos rodeados con círculos los puntos de interés, y con líneas de color verde la orientación del gradiente asignada al descriptor. Vemos como hay varias zonas del coche que el detector no considera de interés por lo que, durante el desarrollo de los experimentos en el Capítulo 4 sólo usaremos el descriptor que proporciona OpenSURF.

(a) (b)

Figura 3.9: Funcionamiento y aplicaciones de OpenSURF.

Finalmente, se ha decidido utilizar SURF para extraer las características de las imágenes normalizadas por las ventajas que demuestra frente a SIFT. Para detectar las regiones de interés de las imágenes, vamos a experimentar aplicando una rejilla cuyas celdas actúen como regiones de interés (rejilla como la que tenemos en la imagen de la Figura 3.10). De esta forma, somos nosotros quienes le indicamos al descriptor de que zonas debe extraer las características. Utilizaremos tantos descriptores SURF como celdas tenga la rejilla que apliquemos a la imagen.

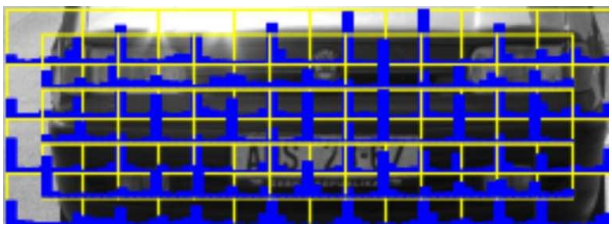


Figura 3.10: Extracción de características usando una rejilla.

3.4. Clasificador de marca y modelo

Tras obtener las características de las imágenes rectificadas, llega el momento de almacenar estos vectores en una estructura de datos, separándolos según sean de un modelo u otro, con el objetivo de que más tarde podamos clasificar un automóvil por su marca y modelo.

Esto es lo denominaremos construcción del clasificador, y es una de las fases necesarias en la etapa de entrenamiento. A la hora de implementar el clasificador de marca y modelo de automóviles seguiremos el algoritmo que se desarrolla en [5]. El funcionamiento es muy sencillo, se calculan 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 que asignaremos a la imagen sin clasificar.

Recordar que los descriptores "q_d" que devuelve SURF son secuencias de 64 componentes por cada región de interés que describe, es decir que tendremos tantas secuencias de 64 componentes como celdas tenga la rejilla que le pasemos. Para calcular la distancia que existe entre la imagen a identificar y los modelos de la base de datos usaremos la Ecuación (3.1). 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 que estamos procesando:

$$dist[m,i] = \sum_{d=1}^{n} ||q_d - NN_{m,i}(q_d)||^2 \tag{3.1}$$

El funcionamiento de algoritmo que usamos para clasificar la marca y el modelo de un vehículo será:

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

2. Para cada modelo "m" que tengamos en la base de datos calculamos 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 que hay en la clase "m" para cada vector de características "q_d" ($NN_{m,i}(q_1)$, $NN_{m,i}(q_2)$, ..., $NN_{m,i}(q_n)$).

   2.2. Calculamos las distancias que hay entre los vectores q_1, q_2, ..., q_n y sus correspondientes vecinos $NN_{m,i}(q_1)$, $NN_{m,i}(q_2)$, ..., $NN_{m,i}(q_n)$.

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

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

Para implementar eficientemente $NN_{m,i}(q_d)$ utilizaremos las estructuras de datos y los algoritmos de búsqueda que la biblioteca ANN (Approximate Nearest Neighbor) [14] nos proporciona. Usamos el paquete ann-octave de [20] en la etapa de entrenamiento, y la implementación de kNN en OpenCV [21] al diseñar la interfaz gráfica en C++.

En la Figura 3.11 se muestra un esquema de como quedarían separados los datos de los modelos en el caso de tener 6 imágenes de automóviles para cada modelo y usar una rejilla con 52 celdas. El objetivo es separar, con ayuda de la estructura de datos kdTree, los modelos de los vehículos en variables diferentes. En cada kdTree es muy importante no mezclar la información de los vectores de 64 componentes (diferenciados con distintos colores: rojo, naranja, amarillo, etc), para que posteriormente al clasificar la imagen de entrada podamos obtener el vecino más cercano a un descriptor $NN_{m,i}(q_d)$.

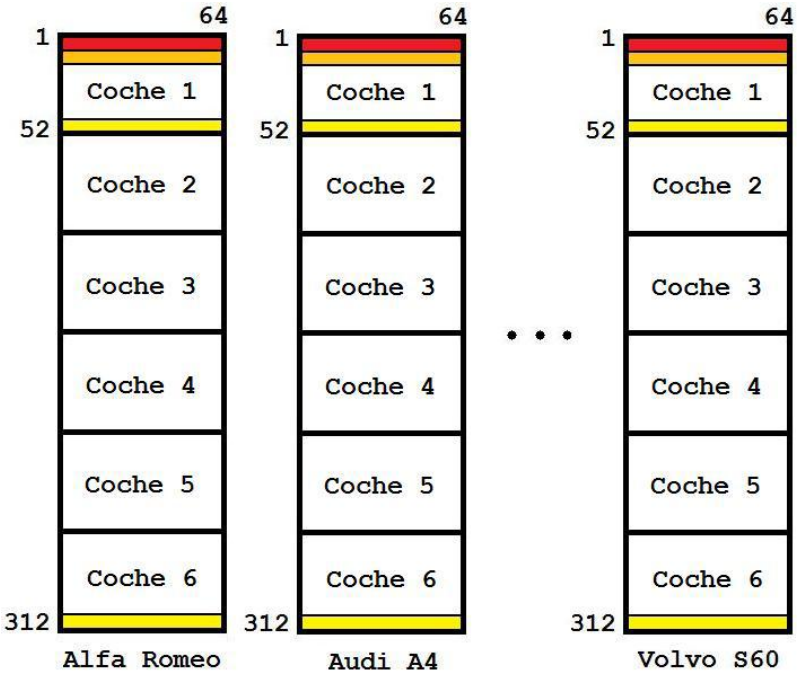


Figura 3.11: Esquema que muestra como se separan los modelos en distintas clases.

A la hora de encontrar cual es la clase más cercana a cada uno de los descriptores SURF extraídos de la imagen a clasificar, usaremos alguno de los algoritmos de búsqueda (annkPriSearch) que funcionan con la estructura de datos (kdTree) donde guardamos las imágenes de cada modelo. Como observamos en la Figura 3.12 tenemos que encontrar el vecino más cercano para cada uno de los vectores de 64 componentes que tenemos (los diferenciamos con los colores: rojo, naranja, amarillo, etc).

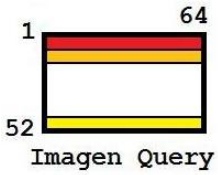


Figura 3.12: Esquema que muestra la estructura de la imagen a identificar.

Llegados a este punto se va a destacar que, aunque nos apoyemos en la implementación de kNN para desarrollar el procedimiento de clasificación de marca y modelo, el método de clasificación usado en este PFC será el de la Ecuación (3.1).

3.5. Verificación de modelos de automóviles

Para el caso de que no quisiéramos identificar la marca y el modelo de una imagen sino simplemente verificar si dadas dos imágenes frontales de coches distintos, ambos son del mismo modelo, el algoritmo que tendríamos que utilizar cambiaría con respecto al de clasificación de marca y modelo que hemos comentado antes. Para entender el algoritmo de verificación que queremos estudiar en el PFC usaremos la Figura 3.13.

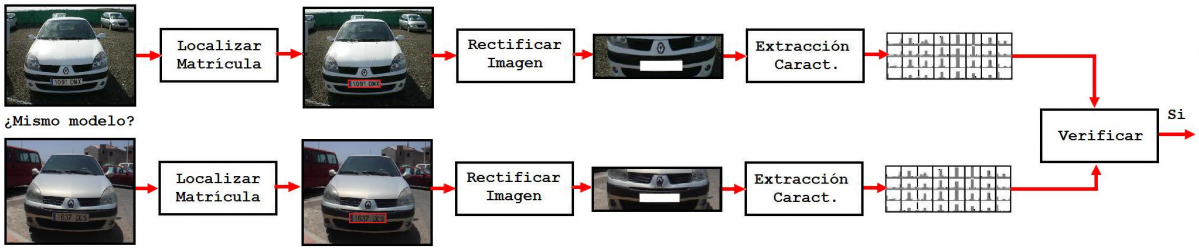


Figura 3.13: Esquema que resume el método de verificación de modelos entre imágenes.

En este caso, el algoritmo es mucho más sencillo pues lo único que tendríamos que hacer es comparar los descriptores SURF de ambas imágenes, tantas veces como celdas tenga la rejilla que usamos para extraer las características. Comparamos los descriptores que genera la rejilla que aplicamos en una de las imágenes $f1_d$, con los descriptores que genera la rejilla en otra imagen $f2_d$. Haremos tantas comparaciones como celdas tenga la rejilla que estemos aplicando "n". El sumatorio de todas esas comparaciones entre los descriptores nos dará la distancia que hay entre las dos imágenes del coche. La fórmula sería:

$$dist = \sum_{d=1}^{n} ||f1_d - f2_d||^2 \tag{3.2}$$

Posteriormente en el Capítulo 4 veremos que posibles aplicaciones podrían usar este sistema y calcularemos cual es la distancia máxima que debemos obtener para considerar que los dos coches son de la misma marca y modelo.

Capítulo 4

Resultados experimentales

En este capítulo se realizan por fin los experimentos que nos permitirán alcanzar el objetivo del PFC, que es clasificar correctamente la marca y el modelo de un automóvil a través de la imagen frontal del mismo. En cada uno de los siguientes apartados se analizan las distintas situaciones que determinan los resultados obtenidos (como pueden ser el tamaño de la base de datos que construyamos, las dimensiones de la rejilla que utilizamos para extraer características, etc). Finalmente se realizará un experimento de verificación de coches entre modelos, en el que se tratará de comprobar si el sistema es capaz de reconocer si dos automóviles son o no del mismo modelo.

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 que vamos a usar, tanto para trabajar en los experimentos, como para construir la aplicación en C++.

Primero se entra a valorar la cantidad de imágenes que conviene tener para cada modelo. A la hora de realizar los experimentos siempre utilizaremos una de las imágenes de cada clase como imagen a clasificar, y el resto de imágenes como datos del modelo en cuestión. Crearemos por tanto una base de datos en la que tendremos 6 imágenes frontales por cada modelo de automóvil, utilizando una como imagen a identificar, y las otras 5 como imágenes conocidas para cada modelo (estas 5 imágenes serán las que se usen para hacer el clasificador). Para la fase de diseño de la interfaz gráfica en C++ se usan las 6 imágenes conocidas para construir el clasificador.

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 de los vehículos compartirán marca), sabiendo además que una misma marca de automóviles saca cada dos o tres años un nuevo modelo de coche. Si hacemos cuentas, en total conseguimos crear una base de datos con 6 x 50 modelos = **300 imágenes distintas.**

En el Cuadro 4.1 se ha construido una tabla con las marcas y los modelos de los 50 automóviles que vamos a utilizar durante el desarrollo del PFC.

Id	Modelo de Coche	Id	Modelo de Coche
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 4.1: Tabla con los modelos utilizados en el PFC.

4.2. Metodología de los experimentos

En este apartado se van a explicar los mecanismos que se han usado para realizar los experimentos. Como ya se ha comentado en el apartado anterior, vamos a dividir la base de datos que se ha construido en, imágenes que vayamos identificar, y imágenes que servirán de base para construir el clasificador de marca y modelo. Vamos a usar lo que se conoce como **técnica de validación cruzada** para determinar que imagen va a ser la que usemos como imagen a clasificar. El funcionamiento del algoritmo consiste en:

1. Inicializamos la variable que usamos para cambiar de imagen a clasificar. Llamaremos a la variable CONT = 1.

2. Hasta que la variable CONT no sea mayor que 6 seguimos los siguientes pasos:

   2.1. Para cada uno de los 50 modelos que utilizamos al crear la base de datos (m_1, m_2, ..., m_{50}):

2.1.1. De las 6 imágenes que tenga el modelo en cuestión elegimos la que indique la variable CONT ($imag_1$, $imag_2$, ..., $imag_6$).

2.1.2. Usaremos $imag_{CONT}$ como imagen a identificar. El resto de imágenes son las que utilizamos para construir el clasificador.

2.2. Tenemos 50 imágenes que trataremos de clasificar y 250 imágenes que usamos para construir el clasificador. Es el momento de realizar las pruebas y los experimentos.

2.3. Cambiamos de imágenes a identificar, es decir que CONT = CONT +1.

3. Llegados a este punto se habrán realizado los 6 experimentos posibles. Si calculamos la media de las 6 tasas de acierto obtenidas, conseguiremos la media de aciertos total del sistema.

A la hora de realizar el experimento que se menciona en el paso 2.2 del algoritmo anterior, habrá que explicar que resultados queremos obtener y como los hemos conseguido. El objetivo del PFC es construir una aplicación que me permita clasificar la marca y el modelo de un automóvil por lo que, el resultado más importante que debemos obtener será la tasa de aciertos de nuestro sistema. Para ello se va a usar la fórmula de la Ecuación (4.1) que vemos a continuación.

$$Tasa\ aciertos\,\% = \frac{Num.\ de\ aciertos}{Num.\ total\ de\ clasificaciones} * 100 \tag{4.1}$$

Será en el paso 3 del algoritmo desarrollado anteriormente, cuando podamos calcular la media de aciertos total del sistema (media de las tasas de acierto de los 6 experimentos).

Además de la media de aciertos puede resultar interesante calcular también la desviación típica del sistema. De esta manera conseguiremos saber si las tasas de acierto en los 6 experimentos han sido semejantes o no (si la desviación típica es muy elevada el sistema no terminará de ser bueno para todas las imágenes). La fórmula de la desviación típica que utilizamos es la de la Ecuación (4.2).

$$\sqrt{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{4.2}$$

Siendo "n" el número de experimentos que se realizan en el sistema, o lo que es lo mismo, el número de imágenes que tenemos para cada modelo de la base de datos. En nuestro caso tendremos que n = 6. Para representar las tasa de acierto que calculamos en cada iteración usamos x_i. Para la media de aciertos total del sistema $\bar{x}$.

4.3. Tamaño de la rejilla

En este apartado el objetivo es que, dada una imagen frontal rectificada de un coche con la matrícula en blanco (como las que hemos generado antes), el sistema nos genere una serie de vectores que caractericen en conjunto la imagen. Para conseguir esto se va a usar el descriptor que nos proporciona SURF, pero para poder utilizarlo es necesario indicarle antes ciertas regiones de interés en la imagen para que pueda hacer la descripción (SURF proporciona un detector de puntos de interés que no vamos a usar en este proyecto).

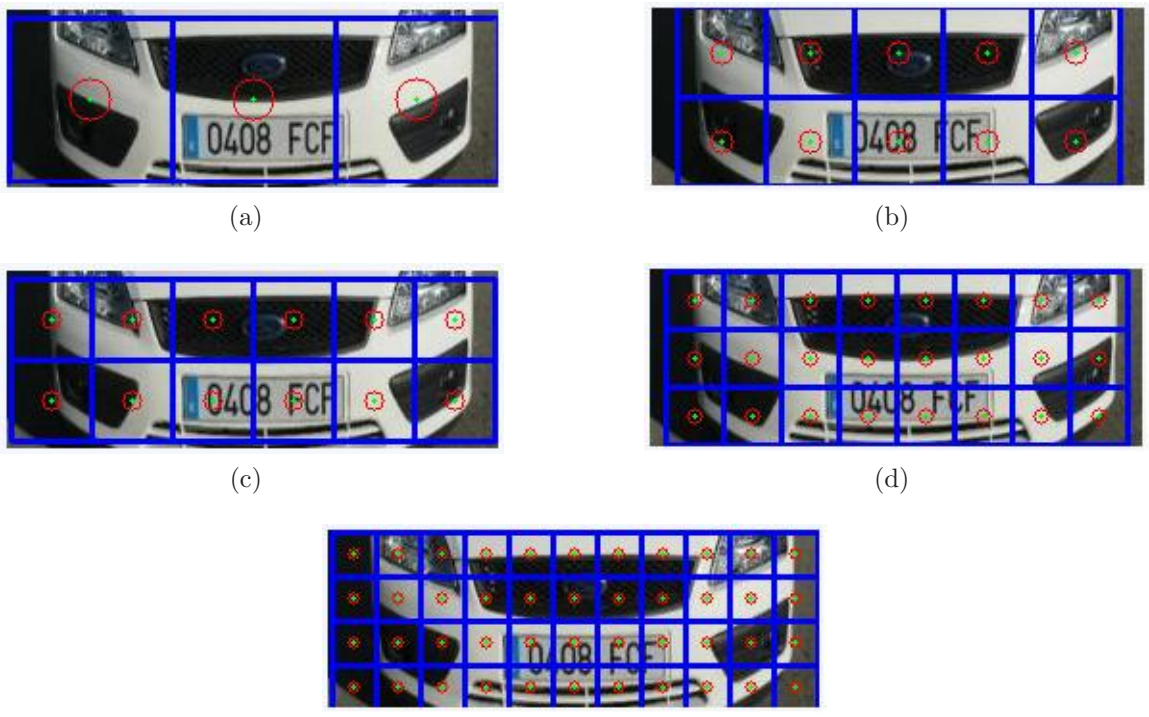
Es aquí donde entra en juego la rejilla, que no es ni más ni menos que un conjunto de celdas que actuarán como regiones de interés para SURF. El objetivo será encontrar la rejilla más adecuada para nuestro tamaño de imagen, que será la que nos proporcione una mayor tasa de acierto de modelos identificados. Para ello primeramente vamos a probar con las 5 rejillas sin solapamiento más básicas, que cuadren en nuestro tamaño de 256 x 92 perfectamente. Posteriormente podremos aplicar algún solapamiento entre celdas para comprobar si mejora la tasa de acierto (lo normal es que mejore al incluir más elementos que describen la imagen).

Por último probaremos aplicando un pequeño desplazamiento a las celdas las rejillas que tienen mejor tasa de acierto, haciendo que haya más solapamiento entre ellas y dejando espacio para que podamos meter algunas celdas más.

4.3.1. Rejillas sin solapamiento

Empezamos probando con las 5 rejillas sin solapamiento que podemos observar en la Figura 4.1. Cada una de estas rejillas tiene un número de celdas cuadradas que usaremos como regiones de interés para SURF.

Lógicamente a la hora de crear las rejillas se ha de tener en cuenta que cuanto más grandes sean las celdas menos de estas va a caber en la imagen rectificada de 256 x 92. Si nos fijamos en la Figura 4.1, en (a) tenemos una rejilla con tan sólo 3 celdas de 85 píxeles de tamaño (tenemos pocos descriptores porque el tamaño de la celda es muy grande). El caso opuesto sería el de la rejilla (e) en la que caben 44 celdas pero con un tamaño de 23 píxeles.



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

Figura 4.1: Imágenes de las rejillas sin solapamiento.

En el Cuadro 4.2 se puede ver la tasa de acierto que obtenemos con cada una de estas rejillas, y como se observa la mejor tasa de acierto la obtenemos con la que usa el tamaño para cada celda de 46 x 46 píxeles (2 x 5 celdas). Seguramente se debe a que las que tienen un tamaño de celda menor, aunque generan más descriptores, pierden precisión al tener una región de interés más pequeña. De momento parece que para imágenes con un tamaño de 256 x 92 las celdas de 46 píxeles son muy superiores a las demás.

Número de celdas	Tamaño de las celdas	Media de aciertos (%)	Desviación Típica
3	85 x 85 píxeles	59.667	5.9889
10	46 x 46 píxeles	75.667	7.6333
12	42 x 42 píxeles	68.667	10.172
24	30 x 30 píxeles	66.333	4.8028
44	23 x 23 píxeles	52.333	2.6583

Cuadro 4.2: Resultados con las rejilla sin solapes.

Una vez llegados a este punto, antes de probar con los solapamientos de celdas en estas rejillas, sería conveniente analizar cuanto mejoraría la tasa de aciertos si combinamos varias rejillas sin solapes unas encima de las otras. Con esta técnica lo que se pretende es capturar los detalles que deja escapar la rejilla con las celdas más grandes.

Para ello vamos a probar por ejemplo colocando sobre la imagen la mejor rejilla que teníamos de 46 x 46 píxeles, y justo encima aplicando una segunda rejilla mas pequeña de 23 x 23 píxeles para que mejore los detalles de la imagen. Teniendo en cuenta que hemos juntado 2 rejillas sin solapes a la vez esperamos que la tasa de aciertos sea considerablemente mayor que la que ya teníamos con la rejilla de 46 x 46, pues el número de descriptores si que ha aumentado al juntar ambas rejillas (ahora tenemos tantos descriptores como celdas tenían la rejilla de 46 x 46 y la de 23 x 23, que son 10 + 44 = 54 celdas en total). En la Figura 4.2 tenemos la combinación de dichos tamaños de rejilla en la imagen rectificada del coche.

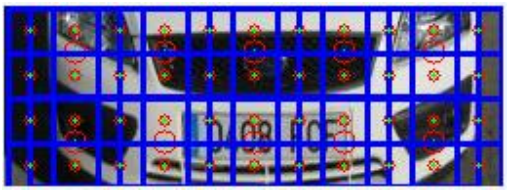


Figura 4.2: Combinación de las rejillas de 46 x 46 y 23 x 23.

Los resultados obtenidos como vemos en el Cuadro 4.3 no son nada prometedores pues conseguimos una media de aciertos del 68.667 % con una desviación típica de 6.4083 (resultados inferiores a los que obteníamos usando únicamente la rejilla de 46 x 46). Este resultado nos indica que combinar varias rejillas a la vez no es el camino a seguir ya que no llegamos a las expectativas que teníamos previstas. Probaremos, ahora sí, con el solapamiento de celdas a partir de las rejillas anteriores.

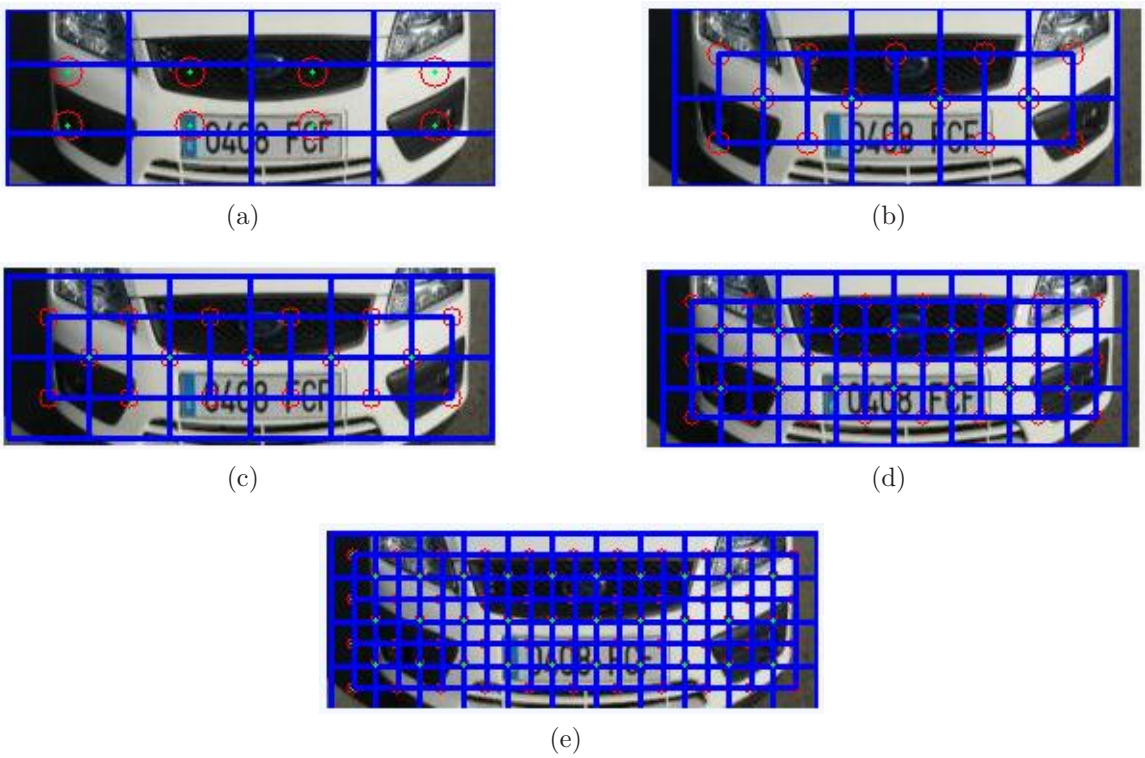
Número de celdas	Tamaño de las celdas	Media de aciertos (%)	Desviación Típica
54	46 x 46 y 23 x 23	68.667	6.4083

Cuadro 4.3: Resultado de combinar 2 rejillas sin solapes de 46 x 46 y 23 x 23.

4.3.2. Rejillas con solapamiento

Ahora vamos a probar la técnica del solapamiento de celdas con las rejillas analizadas anteriormente. Esta técnica trata de volver a describir regiones que ya han sido examinadas mediante nuevas celdas que colocamos en las intersecciones que se crean con las rejillas sin solapamiento, aumentando así el número de celdas de cada rejilla. Se van a analizar las 5 rejillas con solapes que tenemos en la Figura 4.3.

Si nos fijamos en la Figura 4.3 vemos que el solapamiento en (a) se prueba con una nueva rejilla de 8 celdas de 64 que solapa 4 celdas de arriba y 4 de abajo. Para el resto de rejillas que analizamos el solape se hace añadiendo celdas en las intersecciones. Por ejemplo en (b) pasamos de tener 10 celdas a 14 porque hay 4 intersecciones en el medio de la imagen.



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

Figura 4.3: Imágenes de las rejillas con solapamiento.

En el Cuadro 4.4 podemos ver la tasa de acierto que obtenemos con cada una de las rejillas con solapes. Como ocurría con las rejillas sin solapamiento, la mejor es la de tamaño 46 x 46, pero ahora además vemos como ha aumentado la tasa de acierto llegando ya casi al 80 % de modelos de coches reconocidos.

Número de celdas	Tamaño de las celdas	Media de aciertos (%)	Desviación Típica
8	64 x 64 píxeles	61.333	9.2664
14	46 x 46 píxeles	79.667	5.8538
17	42 x 42 píxeles	74	7.3756
38	30 x 30 píxeles	77.333	7.6594
74	23 x 23 píxeles	62.333	5.5737

Cuadro 4.4: Resultados de la rejilla con solapamiento.

4.4. Desplazar las celdas de la rejilla

En esta sección vamos a probar la última técnica que nos permitirá aumentar la tasa de aciertos de modelos reconocidos. Este experimento que se va a realizar consiste en desplazar N píxeles cada celda de la rejilla con respecto a la anterior, de forma que cada celda este más cerca una de otra, lo que permite meter más celdas descriptoras a la vez que conseguimos un solapamiento mayor entre ellas.

A la hora de aplicar la técnica del desplazamiento es importante saber cuantos 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 el espacio que dejemos nos permita incluir alguna celda más.

En la Figura 4.4 encontramos un ejemplo con 4 rejillas distintas que usan celdas del mismo tamaño (en el ejemplo son de 46 x 46), pero cuyas distancias entre celdas son diferentes. En (a) observamos la imagen de la rejilla sin solapes, en (b) tenemos esa misma rejilla pero con un desplazamiento de 23 píxeles en cada celda, en (c) el desplazamiento pasa a ser de 30.66 píxeles y por último en (d) es de 34.5 píxeles.

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

Figura 4.4: Distintos desplazamientos entre celdas para una rejilla de 46 x 46.

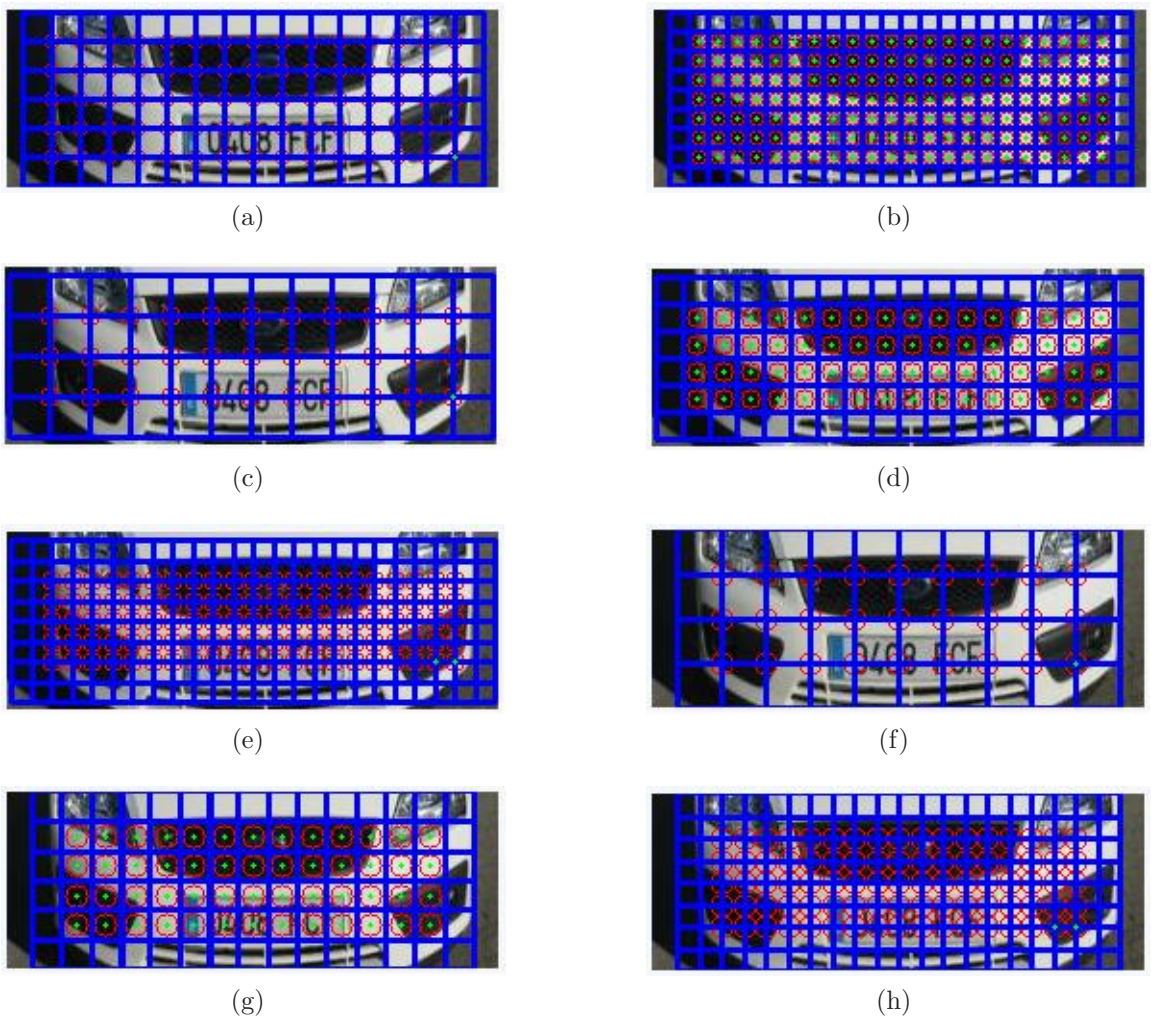
En el tabla de resultados que tenemos en el Cuadro 4.5 se puede observar como aumenta el número de celdas de la rejilla a medida que la distancia entre celdas adyacentes

es menor (el número total de celdas aumenta cuando aplicamos un desplazamiento mayor a las celdas).

Tamaño de las celdas	Solapamiento de las celdas	Número total de celdas
46 x 46 píxeles	sin solapados	10
46 x 46 píxeles	1/2 solapado	27
46 x 46 píxeles	2/3 solapados	52
46 x 46 píxeles	3/4 solapados	85

Cuadro 4.5: Resultados de los distintos desplazamientos probados en la celda de 46 x 46.

Una vez mostradas con este ejemplo (rejilla con celdas de 46 x 46) las distancias que usaremos para probar la técnica, llega la hora de elegir de entre las primeras rejillas que analizamos sin solapamientos las que mejores resultados nos dieron que serán las de tamaño 30 x 30, 42 x 42 y 46 x 46 píxeles. En la Figura 4.5 tenemos la imagen de cada una de las rejillas con desplazamiento que vamos a usar.



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

Figura 4.5: Imágenes de las rejillas con el desplazamiento de las celdas.

En el Cuadro 4.6 presentamos los resultados de cada una de las rejillas que tenemos en la Figura 4.5.

Solapamiento de las celdas	Número de celdas	Tamaño de las celdas	Media de aciertos (%)	Desviación Típica
1/2 solapado	75	30 x 30 píxeles	77	6.5422
2/3 solapados	154	30 x 30 píxeles	80.667	8.3586
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 4.6: Resultados con el desplazamiento de las celdas.

Finalmente podemos observar como la rejilla de 52 celdas es la que ofrece mejores resultados, y por tanto es la que vamos a analizar más en detalle. Lo primero será estudiar el cálculo de la media de aciertos y la desviación típica del sistema con el que experimentamos. Una vez se han realizado los 6 experimentos posibles con la base de datos del PFC (según se explica en la Sección 4.2), obtenemos las 6 tasas de aciertos que calculamos en cada iteración mediante la Ecuación 4.1. Los resultados son los siguientes: $x_1 = 80\,\%$, $x_2 = 90\,\%$, $x_3 = 76\,\%$, $x_4 = 80\,\%$, $x_5 = 86\,\%$ y $x_6 = 86\,\%$.

Para calcular la media de aciertos total del sistema seguiremos la fórmula que se recoge en la Ecuación (4.3).

$$\bar{x} = \frac{x_1 + x_2 + x_3 + x_4 + x_5 + x_6}{6} = 83\,\% \tag{4.3}$$

Con respecto al calculo de la desviación típica utilizaremos la fórmula que definimos en la Ecuación 4.2. Si aplicamos los resultados que se han calculado en los 6 experimentos de la rejilla con 52 celdas llegamos al resultado que se muestra en en la Ecuación (4.4) que desarrollamos.

$$\sqrt{\sigma} = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + ... + (x_6 - \bar{x})^2}{5}} = 5{,}1769 \tag{4.4}$$

Resulta interesante verificar cuales son los fallos que se obtienen con la rejilla de 52 celdas en esta base de datos, para ver las posibles confusiones que puede haber entre los distintos modelos de coches. En la tabla del Cuadro 4.7 podemos observar como muchos de los errores que se cometen en los experimentos se deben a que el aotumóvil que queremos clasificar se confunde con otro de la misma marca pero de un modelo distinto, que se parece mucho al que buscamos (por ejemplo es fácil que confunda el Seat Altea con el Seat Altea XL, o el Seat Córdoba de 2006 y el Seat Córdoba de 2002, etc).

Para cada uno de los experimentos x_1, x_1, ..., x_6 que comprobemos se va a especificar, en caso de fallo en la identificación del automóvil, el índice del modelo con el que se ha clasificado erróneamente, y en caso de acierto, el símbolo $\surd$ que indicará que la clasificación es correcta.

Id	Modelo de Coche	x_1	x_2	x_3	x_4	x_5	x_6	Id	Modelo de Coche	x_1	x_2	x_3	x_4	x_5	x_6
1	Alfa Romeo	√	√	√	√	√	√	26	Opel Corsa	√	√	√	√	√	√
2	Audi A4	√	√	√	√	√	√	27	Opel Vectra	46	√	√	√	√	√
3	Audi A6	√	√	√	√	√	√	28	Opel Zafira	√	√	27	√	27	√
4	BMW Serie 1	21	√	√	√	√	√	29	Peugeot 106	16	√	√	√	15	√
5	BMW Serie 3	√	√	√	√	√	√	30	Peugeot 206	√	√	√	√	7	√
6	Citroen C3	√	√	√	√	√	√	31	Peugeot 307	√	√	√	√	√	√
7	Citroen C4	30	√	√	√	√	√	32	Peugeot 407	√	√	7	√	√	√-
8	Citroen C5	√	√	√	√	√	√	33	Renault Clio	√	√	√	√	√	√
9	Citroen Xantia	√	√	√	42	√	√	34	Renault Laguna	√	√	45	√	√	√
10	Citroen Xsara	9	49	28	√	√	√	35	Renault Megane	√	√	√	√	√	√
11	Citroen Xsara Picasso	√	20	20	√	√	√	36	Renault Scenic	41	√	√	√	4	√
12	Daewoo Nubira	√	√	√	√	√	√	37	Seat Altea	38	√	√	38	√	38
13	Fiat Punto 2002	√	√	√	√	√	√	38	Seat Altea XL	√	√	37	√	√	√
14	Fiat Punto 2006	√	√	√	15	√	√	39	Seat Cordoba 2000	√	√	43	45	√	√
15	Fiat Stilo	√	√	√	√	√	18	40	Seat Cordoba 2002	√	√	√	43	√	√
16	Ford Escort	√	9	√	√	32	√	41	Seat Cordoba 2006	40	√	√	√	√	40
17	Ford Fiesta	√	√	√	25	20	19	42	Seat Ibiza 1994	24	√	√	√	√	10
18	Ford Focus	√	√	√	√	√	√	43	Seat Ibiza 2006	√	√	41	41	√	√
19	Ford Focus C-Max	47	√	18	√	√	√	44	Seat Leon	√	√	√	46	√	√
20	Ford Ka	√	√	√	√	√	√	45	Seat Toledo 2000	√	√	√	41	√	√
21	Hyundai Coupe 1997	√	43	√	√	√	29	46	Toyota Avensis	√	√	39	√	√	√
22	Hyundai Coupe 2000	√	√	√	√	√	√	47	Toyota Corolla	√	√	√	√	√	√
23	Kia Carnival	√	√	√	√	√	25	48	Volkswagen Passat	√	√	√	12	√	√
24	Mercedes Benz	√	√	√	27	9	√	49	Volkswagen Polo	√	19	46	√	√	√
25	Opel Astra	√	√	√	√	√	√	50	Volvo S60	√	√	√	√	√	√

Cuadro 4.7: Tabla que muestra con que modelo se confunde cada automóvil al realizar los experimentos de clasificación.

4.5. Los N siguientes modelos más parecidos

En las tasas de acierto del apartado anterior se tenía en cuenta una clasificación como acierto si el modelo de la imagen de entrada coincidía con el modelo que menos distancia devolvía. En este caso, también nos sería de utilidad para la aplicación si el modelo esperado fuese alguno de los N siguientes automóviles más parecidos. Por tanto tendremos mejores tasas de acierto dependiendo de cuantos vecinos consideremos como acierto en los experimentos. En el Cuadro 4.8 podemos ratificar lo que ya suponíamos inicialmente, que es que clasificamos el modelo correcto de la imagen a identificar siempre entre los 5 modelos más parecidos en el 94 % de las ocasiones. Este resultado habla muy bien de la medida de clasificación que usamos (distancias que calculamos con la Ecuación 3.1).

Número de modelos a tener en cuenta	Media de aciertos (%)	Desviación Típica
2	87.667	3.8816
3	92	2.1909
4	93.333	3.0111
5	94	2.8284

Cuadro 4.8: Tasas de acierto considerando más de un modelo.

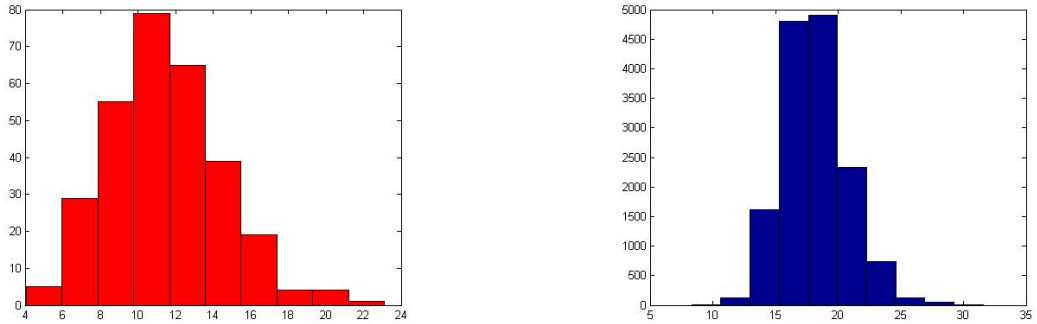
4.6. Distancias a otros modelos de automóviles

En este apartado analizamos qué es lo que ocurriría si se presenta en el sistema, como imagen de entrada, un vehículo que no hemos utilizado en la base de datos. En este caso, lo que es seguro es que el sistema clasificará el automóvil con un modelo equivocado (tal vez la marca sí la clasifique correctamente, si el modelo se parece mucho a uno que sí hemos utilizado).

Basándonos en la Ecuación 3.1 que se ha usado como método de clasificación, vamos a analizar cómo son las distancias que utiliza el sistema cuando clasifica una imagen. Primeramente vamos a ver que distancias existen entre las imágenes de un modelo "m" con respecto a la clase (kdTree) de ese modelo, y después vamos a verificar las distancias que hay entre las imágenes de ese modelo "m" con el resto de clases (modelos distintos). Si verificamos que las distancias entre imágenes del mismo modelo, son muy inferiores a las distancias con modelos distintos, podríamos hacer que el propio sistema descartara la clasificación de un automóvil que no tenemos en nuestra base de datos.

Para hacer las pruebas usaremos las 300 imágenes de la base de datos, comparando cada imagen con los 50 modelos que tenemos. De las 50 distancias que se obtienen por cada imagen, una será la distancia de la imagen con la clase del modelo, y las otras 49 distancias serán con clases de modelos distintos. Si hacemos cálculos se verá que realizamos los experimentos con **300 distancias entre imagen y clases del mismo modelo y 14700 distancias entre imagen y clases de modelos distintos.**

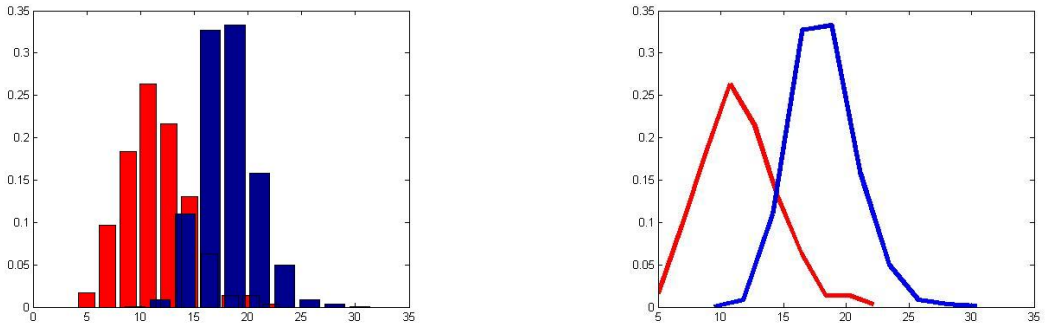
En la Figura 4.6 realizamos dos histogramas con las distancias. Tenemos, por un lado las distancias entre imagen y clases del mismo modelo (a), y por otro las distancias entre imagen y clases de modelos distintos (b).



(a) (b)

Figura 4.6: Distancias entre imagen y clases: (a) Mismo modelo (b) Modelos distintos.

Efectivamente existe una diferencia de distancias dependiendo de si la imagen se clasifica con su modelo correcto o no. Las distancias, si se clasifica correctamente oscilan entre 10 y 12, mientras que clasificando mal oscilan entre 15 y 20. En la Figura 4.7 normalizamos las gráficas (dividiendo cada histograma por el número de distancias que se han usado en cada uno), con el objetivo de juntar los histogramas en la misma gráfica, normalizando los valores entre [0,1].



(a) (b)

Figura 4.7: Histograma de frecuencias. (Distancias a otros modelos).

Usando la fórmula de la Ecuación 3.1, no obtenemos una diferencia en las distancias lo suficientemente grande como para asegurar que no se va a clasificar un modelo de automóvil que no tenemos en la base de datos. Es cierto que por ejemplo, para todos los casos en los que la comparación devuelve una distancia menor que 10, la clasificación es correcta; y para los casos en lo que devuelve una distancia mayor a 22.5, la clasificación del automóvil es errónea. No obstante, no vamos a considerar este estudio en el desarrollo de la aplicación.

4.7. Técnica de verificación

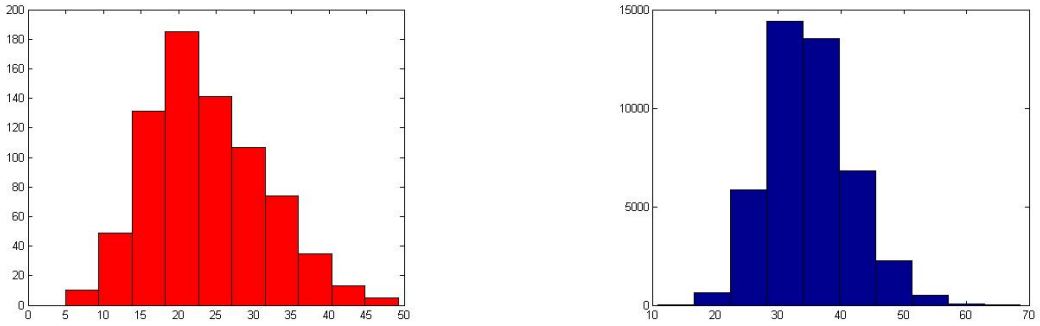
En este último apartado vamos a probar la técnica de verificación que hemos introducido en la Sección 3.5 del proyecto. A día de hoy todavía existen muchos problemas en lugares como parkings, donde se necesita reconocer cada automóvil, de manera que no pueda salir del garaje un coche distinto del esperado. Si funciona nuestro sistema de verificación como esperamos no podría salir un automóvil distinto del que entró, ayudando a evitar de esta manera los robos.

Los experimentos que realizamos tratan de probar que la aplicación es capaz de verificar, a partir de dos imágenes frontales de automóviles, que los dos coches son de la misma marca y modelo. Para extraer las características de las imágenes vamos a usar la rejilla de 52 celdas de tamaño 46 que nos ofrecía los mejores resultados. El algoritmo que usamos para verificar que los dos vehículos son del mismo modelo, será el que definimos en la Ecuación 3.2 (comparar cada uno de los descriptores de las imágenes y sumar estos resultados para obtener la distancia que separa a ambas imágenes). Teóricamente para que nuestro sistema de verificación funcione correctamente debería de pasar que la distancia entre coches de la misma marca y modelo siempre sea más pequeña que entre modelos distintos.

Para hacer las pruebas usaremos las 300 imágenes de la base de datos, comparando cada imagen con el resto (5 imágenes siempre serán de su mismo modelo y las otras 294 serán de modelos distintos). Con esta base de datos finalmente generaremos una matriz con 90000 distancias de las que, 88200 serán distancias entre modelos distintos, y las otras 1800 distancias entre imágenes del mismo modelo (300 de estas últimas son distancias comparando la misma imagen, distancias de valor cero).

Por último descartaremos la mitad de las distancias, sabiendo que entre la $imag_1$ y la $imag_2$ tenemos la misma distancia que entre la $imag_2$ y la $imag_1$. Realizamos los experimentos con **750 distancias de imágenes del mismo modelo y 44100 distancias para imágenes de modelos distintos.**

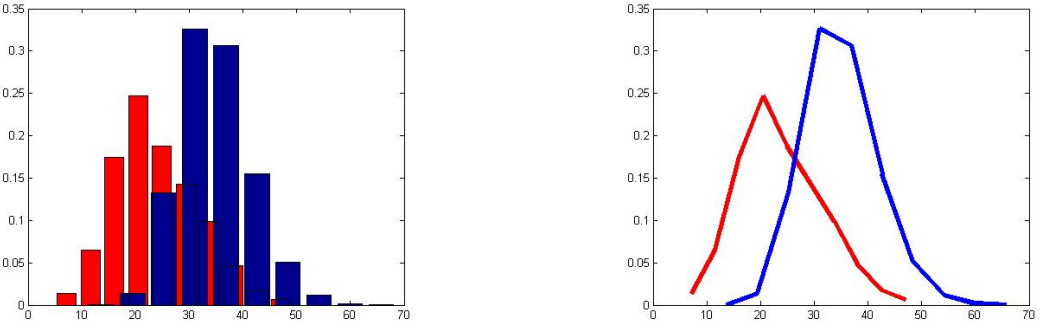
Lo más importante es verificar que la distancia entre coches del mismo modelo es mucho más pequeña que entre coches de modelos distintos (el problema puede estar en que, aunque sea un modelo distinto, tenga la misma marca y esta se parezca bastante a la que buscamos). Para ello, podemos visualizar la Figura 4.8 donde tenemos, por un lado las distancias entre coches del mismo modelo en (a), y por otro las distancias entre coches de modelos distintos en (b).



(a) (b)

Figura 4.8: Distancias entre imágenes: (a) Mismo modelo (b) Modelos distintos.

Observando las gráficas, podemos llegar a la conclusión de que efectivamente hay una diferencia de distancias dependiendo de si las imágenes de coches son del mismo modelo o no. La distancia para el mismo modelo se acerca a 20, mientras que entre modelos distintos oscila entre 30 y 40. En la Figura 4.9 normalizamos las gráficas, dividiendo cada histograma por el número de distancias que se han usado en cada uno. El objetivo es poder juntar los histogramas en la misma gráfica normalizando los valores entre [0,1].



(a) (b)

Figura 4.9: Histograma de frecuencias. (Técnica de verificación).

Mediante la Figura 4.9 podemos determinar que no está claro que la distancia que utilizamos en la Ecuación 3.2 sea buena para realizar este experimento de verificación.

Lo mismo ocurre con la mejor rejilla que se ha obtenido para la extracción de las características de las imágenes; no sabemos si hay alguna rejilla que funcione mejor al realizar este experimento. Quedará como trabajo para el futuro el sustituir la rejilla y/o la distancia, para establecer una mejoría en la verificación de modelos de automóviles.

Capítulo 5

Descripción informática

Este capítulo detalla la construcción de la aplicación gráfica en C++ que nos permite utilizar todo lo que hemos probado en el Capítulo 4. Seguidamente se analizarán en detalle algunas etapas en el desarrollo del software, como pueden ser la captura de los requisitos o la realización en análisis del sistema. También se entra a valorar el plan de trabajo que se ha seguido en la elaboración del PFC y las bibliotecas que se usan para su implementación.

5.1. Metodología y plan de trabajo

En este primer apartado se van a entrar a detallar las pautas que se han seguido para hacer el PFC que se presenta. Para ello se indican tanto la metodología y el modelo que se han usado a la hora de construir la aplicación, como el orden que se ha seguido en el plan de trabajo a lo largo de la realización del proyecto. Por último detallamos al final de cada etapa una estimación del tiempo que se ha necesitado para realizar las tareas que se proponen.

Un punto fundamental para que el desarrollo de un PFC transcurra de manera satisfactoria es la selección de un ciclo de vida adecuado al principio del proyecto. Para este PFC en concreto primeramente se intentó usar un desarrollo en cascada, pero resulta difícil de aplicar en un proyecto con unos requisitos tan flexibles como este (durante gran parte del PFC no tendríamos nada tangible o ejecutable). De esta manera, se consideró usar algún ciclo de vida que generase resultados intermedios como el desarrollo en espiral, de manera que pudiésemos ver algo funcionando periódicamente.

Por tanto, una vez **elegido el modelo en espiral como herramienta para desarrollar el software del PFC**, hemos de detallar cuales han sido las tareas del plan de trabajo que actúan como iteraciones del modelo.

- *Fase de documentación.* Lo primero que necesitamos es introducirnos en el mundo del reconocimiento de imágenes con el ordenador. En el Capítulo 2 detallamos algunos de los artículos relacionados con la identificación de vehículos que hemos estudiado para adentrarnos de lleno en el tema de el proyecto. Tiempo estimado: 1 mes.

- *Creación de la base de datos.* En este segundo ciclo se tratarán de conseguir imágenes de automóviles vistos de frente que nos ayudarán a probar el funcionamiento de los ciclos posteriores a este.

Sabiendo que nuestro PFC tiene como objetivo el poder clasificar un coche con su marca y modelo, es necesario que las imágenes de coches que se coleccionen sean de marcas y modelos distintos, creando así la mayor cantidad de modelos de coches posibles. Tiempo estimado: 2 meses.

- *Transformar imagen.* El objetivo en este ciclo es que dada una imagen frontal de un coche y indicándole de alguna manera al sistema las coordenadas de la matrícula, se obtenga una imagen normalizada del coche. Para ello usaremos algunas técnicas de tratamiento de imágenes (basadas en la homografía) que nos ayudarán además a corregir la imagen de posibles desplazamientos (puede que la imagen del coche no sea totalmente frontal). Tiempo estimado: 1 mes.

- *Extraer características.* En este ciclo trabajaremos sobre la imagen rectificada con el descriptor SURF para obtener las características de esta. El objetivo es obtener una ristra de vectores de características de 64 componentes que nos permitan posteriormente identificar un automóvil. Tiempo estimado: 1 mes.

- *Aplicación del clasificador.* Una vez necesitemos por fin clasificar nuestra imagen, es necesario construir el clasificador. Usando las estructuras de datos y los métodos que nos proporciona la biblioteca de programación de ANN conseguiremos saber qué clase es la más cercana a cada uno de los vectores de 64 componentes de la imagen a identificar. Posteriormente desarrollaremos un algoritmo que nos calcule a qué modelo de los que tenemos en la base de datos se parece más y así se lo hará saber al usuario. Tiempo estimado: 2 meses.

- *Interfaz que muestre los resultados visualmente.* Este último ciclo permitirá probar de forma global todo el trabajo que hemos realizado. Se creará una interfaz gráfica usando C++ y las bibliotecas de OpenCV para presentar los resultados. Tiempo estimado: 2 meses.

5.2. Captura de requisitos

En este apartado se detallan todas las condiciones que tiene que cumplir la aplicación, para que su funcionamiento pueda considerarse correcto. Llamaremos requisitos funcionales a las acciones que deberá ser capaz de desempeñar el producto deseado y requisitos no funcionales a otras posibles propiedades del producto en sí.

5.2.1. Requisitos funcionales

Los requisitos funcionales son aquellos que indican qué debe hacer o cómo debe reaccionar la aplicación. Los requisitos que tiene que cumplir este sistema son:

RF1. El sistema dará la opción al usuario de cargar la imagen de un coche para poder identificar la marca y modelo del mismo.

RF2. El sistema permitirá al usuario crear un rectángulo sobre la imagen con el objetivo de que este encuadre la matrícula y así poder obtener las coordenadas de las esquinas. Una vez se hayan seleccionado las esquinas se puede normalizar la imagen a un tamaño de 256 x 92 píxeles.

RF3. Si el usuario se ha equivocado al marcar la matrícula podrá rehacer la selección cuantas veces quiera, siempre que no se haya procedido a la etapa de extracción de las características.

RF4. También es importante que a la hora de encuadrar con el rectángulo la matrícula, empecemos desde la esquina superior izquierda.

RF5. Es necesario que la imagen que se cargue sea la vista frontal del coche para obtener una correcta clasificación del coche.

RF6. El sistema presentará al usuario por defecto las imágenes de los 5 modelos que más parecido tienen con la imagen que queríamos identificar.

5.2.2. Requisitos no funcionales

Los requisitos no funcionales, son todos aquellos que no definen la funcionalidad de la aplicación, pero que son necesarios para su correcto funcionamiento. Los requisitos no funcionales detectados durante el desarrollo de la aplicación son:

RNF1. Desarrollo de la aplicación bajo un entorno Windows con OpenCV, que caracteriza la interfaz por su facilidad de uso (para poder utilizar este sistema se debe tener instalada la biblioteca OpenCV en el ordenador).

RNF2. Implementación de la aplicación usando el lenguaje C++ [16].

5.3. Análisis y diseño

Una vez conocida la relación que existe entre la aplicación y el usuario, en este apartado se va a tratar de exponer el funcionamiento del sistema que se va a desarrollar. El principal objetivo de esta sección es conseguir modularizar la aplicación (bajo acoplamiento entre los módulos), para no tener problemas en caso de realizar algún cambio en la implementación de los algoritmos de clasificación.

Lo primero que haremos será describir la estructura del sistema, mostrando sus clases, atributos y las relaciones entre estos. Usamos el diagrama de clases de la Figura 5.1 para crear el diseño conceptual de la información que se manejará en el sistema, y para explicar el funcionamiento básico de la aplicación.

El módulo *Programa Principal* es el que va a funcionar como elemento de comunicación con el usuario, lo que implicará usar los métodos que se han definido en cada uno de los submódulos *Clasificador KNN*, *Descriptor SURF* y *GUI*, para hacer funcionar la aplicación. El submódulo *Base de Datos* lo utilizará el sistema de clasificación para leer de un fichero de texto los vectores de características que hemos extraído con SURF.

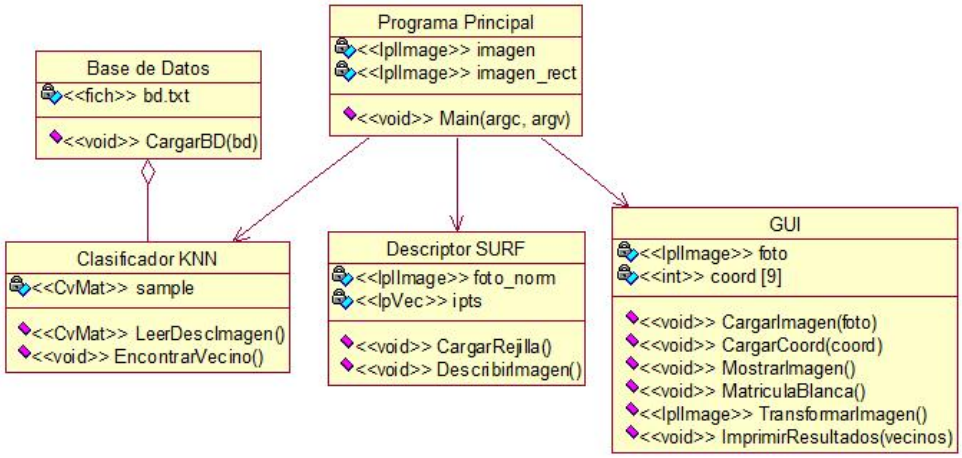


Figura 5.1: Diseño de la aplicación.

A continuación se entra a detallar el funcionamiento de los métodos de cada clase, sus correspondientes parámetros y su conexión con el resto de módulos:

- *Main(**int** argc, **char**** argv).* Programa principal en el que crearemos los objetos de tipo GUI, SURF y KNN que nos permitirán utilizar los métodos que implementan. Recibe los parámetros que introduce el usuario al ejecutar la aplicación.

- *CargarImagen(**IplImage*** foto).* Con esta función cargamos en el módulo para la interfaz GUI, la imagen que le hemos pasado al Main con el parámetro de entrada.

- *CargarCoord(**int** coord[9]).* Método que usamos para cargar las coordenadas de las esquinas de la matrícula.

- *MostrarImagen().* Función que muestra por pantalla la imagen con la que estamos trabajando.

- *MatriculaBlanca().* Método que rellena de color blanco la región que se delimita con las coordenadas marcadas (se deberá marcar la matrícula del automóvil).

- *TransformarImagen().* Realiza las trasformaciones geométricas sobre la imagen. Como resultado obtenemos la imagen rectificada que vamos a usar en el módulo SURF.

- *ImprimirResultados(**int** vecinos).* Función que utilizamos para mostrar por pantalla los resultados que obtenemos del clasificador. El número de vecinos indica cuantos modelos usamos para reconocer la imagen a identificar.

- *CargarRejilla().* Carga de un fichero de texto la rejilla que utilizamos para extraer las características de la imagen normalizada. Las coordenadas de la rejilla las guardamos en el atributo privado ipts de tipo **IpVec**.

- *DescribirImagen().* Método que usamos para pasarle a la imagen rectificada el descriptor SURF. Los resultados los guardaremos en un fichero de texto que leeremos en el módulo de clasificación.

- *CargarBD(**fich** bd).* Carga en el clasificador KNN los vectores con las características que se han extraído de las imágenes de la base de datos.

- *LeerDescImagen().* Función utilizada para cargar las características extraídas con SURF de la imagen de entrada. Los datos los guardaremos en la matriz que definimos con el atributo privado sample de tipo **CvMat**.

- *EncontrarVecino().* Método que utilizamos para crear el algoritmo de clasificación del sistema. Los resultamos se guardan en un fichero de texto que consultamos con la llamada al método ImprimirResultados(5).

Para terminar esta parte vamos a crear el diagrama de secuencia que tenemos en la Figura 5.2. El diagrama de secuencia es uno de los diagramas más efectivos para modelar la interacción entre los objetos de un sistema a traveś del tiempo. Se muestran los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.

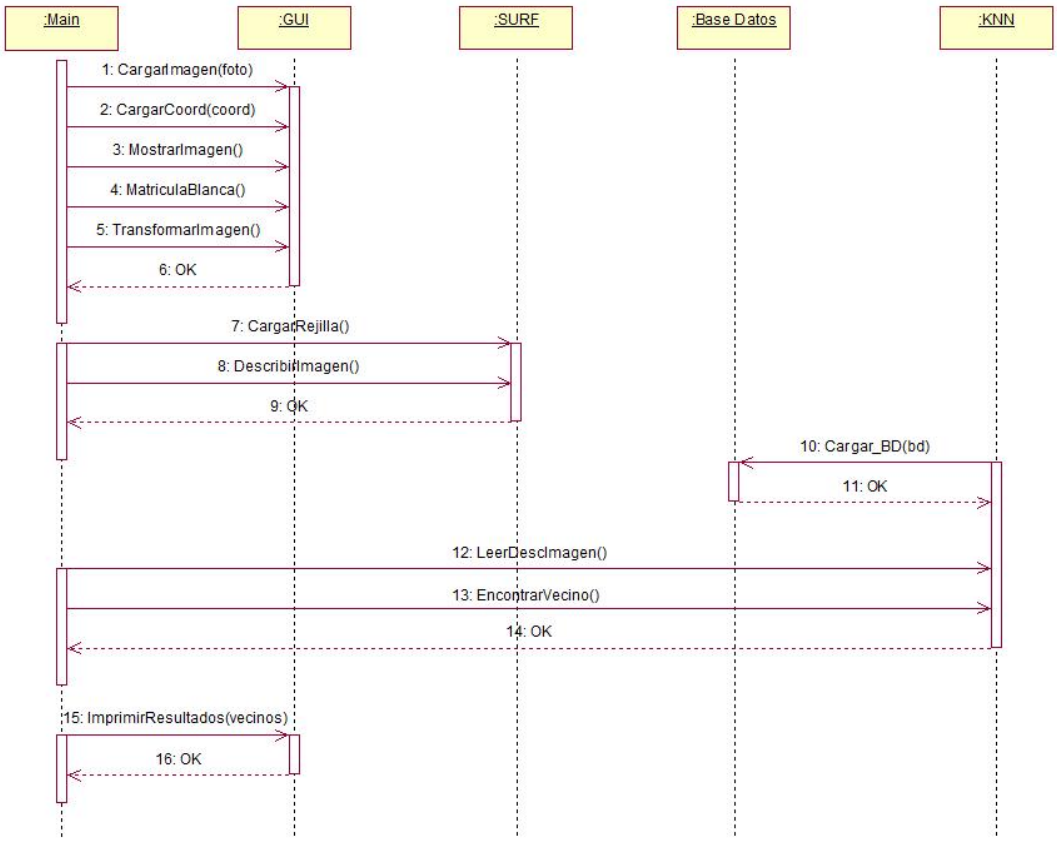


Figura 5.2: Diagrama de secuencia del sistema.

5.4. Implementación

En esta sección se detalla la estructura jerárquica que va a tener el sistema que desarrollamos en el PFC, así como el número de ficheros que va a utilizar, y el uso que le daremos a cada uno. En la Figura 5.3 se ha dibujado un esquema con los directorios de las carpetas que vamos a utilizar.

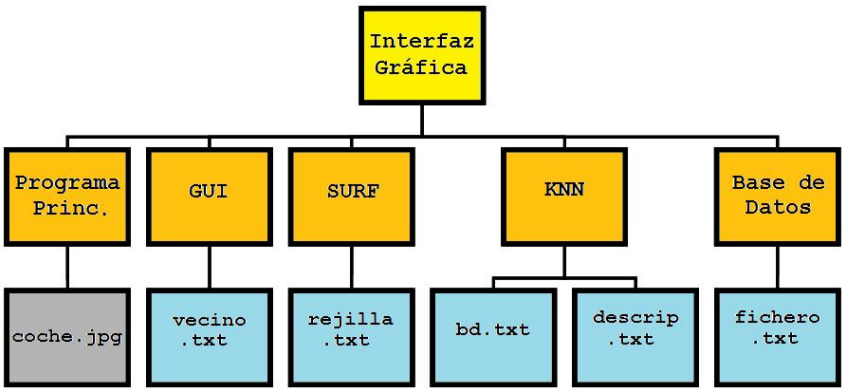


Figura 5.3: Árbol con los directorios de la aplicación.

Ahora entraremos en cada uno de los directorios para analizar en detalle el contenido de los mismos:

- El directorio raíz será *Interfaz Gráfica* donde guardaremos las 5 subcarpetas con las que modularizamos el sistema.

- En la carpeta *Programa Principal* guardaremos los ficheros con los que realizaremos las pruebas en la aplicación. Usando un entorno de desarrollo como Dev-C++ crearemos un proyecto **Main.dev** para poder enlazar los ficheros que se vayan a usar (necesitaremos enlazar todos los ficheros .hpp y .cpp que tengamos en GUI, SURF y KNN). El fichero principal de este proyecto Main.dev será **main.cpp**, en el que tendremos que añadir las siguientes líneas de código para poder hacer uso de los métodos que se definan en estos módulos.

  ```
  #include "../GUI/GUI.hpp"
  #include "../SURF/MiSURF.hpp"
  #include "../KNN/KNN.hpp"
  ```

  Por último vamos a guardar en esta carpeta la imagen del coche que queramos identificar que llamamos **coche.jpg**. Una vez compilamos los ficheros con éxito, se generará en esta carpeta un fichero ejecutable **Main.exe** que podremos utilizar para probar la aplicación. Para ello solo necesitamos pasarle la imagen desde línea de comandos:

  ```
  > Main.exe coche.jpg
  ```

- La interfaz gráfica que observamos cuando ejecutamos la aplicación, la tenemos en la carpeta *GUI*. Con los métodos de este módulo debemos ser capaces de mostrar la imagen que hemos cargado desde línea de comandos, permitir al usuario seleccionar la matrícula y obtener una imagen normalizada con la matrícula en blanco.

  En este directorio se ha creado un proyecto **GUI.dev** que utilizamos como enlace para los ficheros con los que va a trabajar el módulo. El primer archivo del proyecto será la interfaz **GUI.hpp**, en la que definimos los métodos de la clase GUI que implementaremos en el otro fichero del proyecto **GUI.cpp**. El módulo GUI además hace uso de un archivo de texto **vecino.txt** que nos genera el módulo KNN al usar el clasificador. En este fichero tendremos anotados los índices de los vecinos que mostraremos por pantalla (clases que más se parecen a la imagen de entrada).

- La carpeta *SURF* nos permitirá describir una imagen rectificada con vectores de 64 componentes. Crearemos un proyecto **RejillaSURF.dev** en el que añadiremos los archivos .hpp y .cpp que necesitamos para compilar la biblioteca de OpenSURF con Dev-C++. A este mismo proyecto le vamos a enlazar los ficheros cabecera **MiSURF.hpp** y implementación **MiSURF.cpp** para crear la clase SURF.

  El fichero de texto **rejilla.txt** contiene los datos que utiliza el módulo para crear las celdas de la rejilla (regiones de interés para extraer las características), siguiendo el siguiente formato:

celda	x	y	ancho	alto
0	13	0	46	46
1	28.33	0	46	46
2	43.66	0	46	46
...	...	...	...	...

- En la carpeta *KNN* creamos el proyecto **KNN.dev** que usamos para encontrar los vecinos más próximos. Para compilar el módulo necesitaremos añadir, junto a los ficheros **KNN.hpp** y **KNN.cpp**, los archivos de la biblioteca "ml"de OpenCV.

  A la hora de construir el clasificador usaremos los datos que tenemos guardados en el fichero **bd.txt** con las características de las imágenes de la base de datos, y el fichero **descripcion.txt** con los vectores de la imagen rectificada que vamos a identificar.

- El último módulo a analizar será la *Base de Datos* donde guardaremos las imágenes que se usan para hacer la clasificación. Junto a las 300 imágenes que utilicemos vamos a crear un fichero de texto que llamaremos **fichero.txt** en el que guardaremos toda la información relativa a los coches. El formato del archivo es el siguiente:

imagen	marca	(x1,y1)	(x2,y1)	(x1,y2)	(x2,y2)
0000.jpg	Alfa_Romeo_147_2003	325,254	418,249	325,283	416,278
0001.jpg	Alfa_Romeo_147_2003	319,252	425,248	319,286	423,281
...	...	...	...	...	...
0006.jpg	Audi_A4_2.0Tdi	199,284	363,285	198,315	363,317
...	...	...	...	...	...

5.5. Bibliotecas

En este último apartado se van a analizar con detalle cada una de las bibliotecas que se usan en el PFC, así como los métodos que se utilizan para diseñar la aplicación.

5.5.1. OpenSURF

OpenSURF es una biblioteca escrita en C++ con OpenCV que utiliza como descriptor de características SURF (Speeded Up Robust Features). La principal característica de SURF frente a otros descriptores es que, además de ser robusto frente a cambios de iluminación, es invariante a las deformaciones y a la rotación.

Los autores de SURF aseguran también que SURF es ligeramente más rápido que el descriptor SIFT en el que se basa [6]. Podemos descargarnos la biblioteca de SURF desde la pagina web del autor [18].

Con respecto a los métodos de OpenSURF que vamos a utilizar durante el desarrollo del proyecto:

- *surfDetDes.* Esta función se encarga primero de detectar regiones de interés en una imagen, y después de describirlas para extraer las características con vectores de 64 componentes. Como en este PFC hemos decidido marcarle nosotros las regiones de interés a la imagen a través de una rejilla, solo usaremos este procedimiento en la fase de pruebas, por ejemplo en la Figura 3.9 (b).
- *surfDes.* Este otro método es el que se encarga de describir la región de interés que le pasamos como parámetro. Basta con leer las coordenadas de cada celda de la rejilla y almacenarlas en el tipo de datos Ipoint.
- *drawIpoints.* Esta función permite dibujar en la imagen un punto en cada región de interés con el objetivo de que el usuario conozca la zona que se va a describir.
- *drawWindows.* Permite visualizar el tamaño de la ventana del descriptor que se va a aplicar a un punto de interés.

5.5.2. ANN

ANN es una biblioteca escrita en C++ que nos proporciona los tipos de datos y las funciones necesarias a la hora de buscar el vecino más cercano a una clase [14]. Usaremos la implementación de ann que obtenemos con los paquetes extras de octave (octave-forge), y que podemos obtener de [20].

- *annkPriSearch.* Es el método que nos devolverá los índices de los K vecinos aproximados. Primeramente deberemos construir una estructura de datos kdTree para cada clase o modelo de la base de datos, y posteriormente aplicaremos la función annkPriSearch en cada kdTree para obtener el vecino más cercano a cada clase (pasándole como argumentos el vector de características de 64 componentes y K=1).

5.5.3. 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 bajo Mac OS X, Windows o Linux [13]. A continuación se detallan todas las funciones de OpenCV utilizadas en el proyecto:

- *cvWaitKey.* Espera la pulsación de una tecla durante los milisegundos indicados en el parámetro que se le pasa (la espera es infinita si el parámetro es 0).
- *cvNamedWindow.* Crea una ventana con el nombre y las dimensiones indicadas.
- *cvMoveWindow.* Mueve la ventana que se ha creado a las coordenadas que le indicamos por parámetro.
- *cvLoadImage.* Carga una imagen de disco en una variable con el tipo IplImage.

- *cvSaveImage.* Guarda en el directorio indicado la imagen asignada a una variable de tipo IplImage.
- *cvShowImage.* Muestra la imagen que tenemos en el segundo parámetro que se le pasa a la función, en la ventana del primer parámetro. Tiene que ser una ventana creada con cvNamedWindow.
- *cvReleaseImage.* Libera el espacio reservado para trabajar con la imagen.
- *cvFillPoly.* Colorea con un color indicado por parámetro, el rectángulo que le indicamos con las cuatro coordenadas de sus esquinas.
- *cvGetPerspectiveTransform.* Calcula la transformación perspectiva entre puntos (homografía) que guardamos en variables del tipo CvPoint2D32f. La matriz en la que almacenamos el resultado será del tipo CvMat.
- *cvWarpPerspective.* Aplica una transformación geométrica a una imagen.
- *cvCreateMat.* Crea una matriz de tipo CvMat con las filas y columnas que se indican a través de los parámetros.
- *cvmSet.* Asigna un valor a un elemento de la matriz CvMat.
- *cvReleaseMat.* Libera el espacio reservado para la variable de tipo CvMat.

Capítulo 6

Conclusiones

En este último capítulo primero se va a presentar un resumen de los resultados obtenidos, y a continuación se van a enumerar algunas aplicaciones que se podrían realizar a partir de este PFC. Finalmente, para seguir avanzando en el campo del reconocimiento de vehículos, se sugieren posibles mejoras y algunas ideas que podrían perfeccionar este sistema.

6.1. Discusión de los resultados

A través de los experimentos que hemos ido viendo a lo largo del PFC, se han ido estableciendo las bases que iba a tener la aplicación. En la fase de entrenamiento construimos el clasificador que usaremos para identificar los automóviles (la tasa de aciertos en la clasificación de los vehículos es de un 83 %, que aumenta a un 94 % si calculamos los cinco modelos que más se parecen). En la fase de clasificación simplemente comprobamos que el sistema diseñado funciona correctamente, para poder pasar a la construcción de la interfaz gráfica.

Para realizar los experimentos en ambas fases, se han usado distintos programas dependiendo de la fase del proyecto en la que nos encontrásemos, por ejemplo, en la fase de tratamiento de imágenes se ha usado Octave/Matlab para los experimentos [15], y OpenCV en el diseño de la interfaz gráfica. Para construir el clasificador ocurre lo mismo, utilizamos la versión de ANN disponible en [20] para lo experimentos, y para la construcción de la aplicación usamos las funciones que implementa OpenCV como KNN.

En la Figura 6.1 se observa un esquema que muestra el funcionamiento de la aplicación que se va a diseñar.

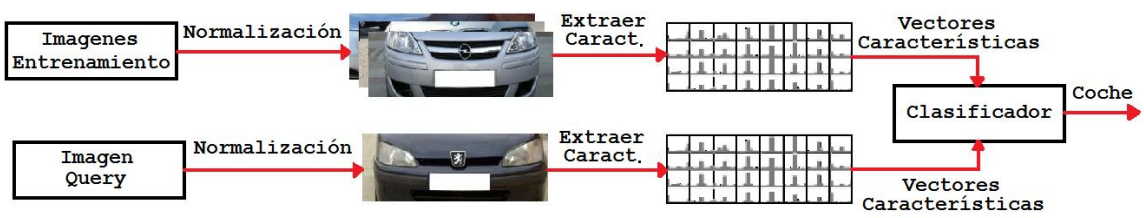


Figura 6.1: Método propuesto para la clasificación de imágenes.

Este tipo de sistemas como el que se propone en el PFC, se podrían utilizar en cualquier empresa que necesite identificar vehículos, sin embargo presentan el problema de la localización de la matrícula en la imagen. Se necesitará desarrollar un procedimiento para la localización automática de automóviles, o utilizar un sistema ya desarrollado de localización de matrículas.

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

- *Oclusiones que tapen el automóvil.* Si tenemos en cuenta que las imágenes de la base de datos que usan en la aplicación son todas sin oclusiones (se puede ver perfectamente el vehículo), cuando en la imagen que se quiera identificar el coche esté parcialmente tapado por otro objeto, la probabilidad de que clasifique la marca y el modelo correctamente se verá disminuida.

- *País de los modelos.* De igual manera, los vehículos que se han usado en los ejemplos son todos de uso en España. De todas formas, querer identificar por ejemplo un coche americano no supondría ningún problema, siempre y cuando se encuentre el modelo en la base de datos.

- *Número de vehículos en la imagen.* Si la imagen capturada es lo suficientemente grande como para tener al menos dos vehículos vistos de frente y queremos identificarlos, primero se le indicará al sistema las coordenadas de la matrícula de uno de ellos y después la del otro vehículo. La aplicación no es capaz de identificar más de un automóvil a la vez.

- *Cambios en la luminosidad.* 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 porcentaje de acierto del modelo del mismo. Hay que evitar siempre que se pueda dos cosas: la oscuridad de la noche y los destellos de luz por el día.

- *Posible vehículo dañado.* Por último debemos evitar también que el vehículo a identificar este gravemente dañado, de tal forma que la estructura del mismo se vea visiblemente afectada.

6.2. Conclusiones

Para verificar que el proyecto se ha realizado de acuerdo a lo esperado, basta con comprobar que se han alcanzado con éxito los objetivos y subobjetivos que se marcaron en el Capítulo 1:

- *Construcción de la base de datos del sistema.* Construimos una base de datos con 50 modelos distintos de automóviles, y 6 imágenes frontales por cada clase. Si hacemos las cuentas vemos que la base de datos del PFC consta en total de 300 imágenes distintas, que nos permitirán hacer experimentos que devuelvan resultados significativos.

- *Desarrollar un clasificador de imágenes de automóviles en base a su marca y modelo.* Usamos como método de clasificación el que se detalla en el Capítulo 3 de algoritmos. La idea es localizar el automóvil que se vaya a identificar en la imagen de entrada, extraer sus características con SURF y realizar la clasificación usando el algoritmo que diseñamos en la Sección 3.4.

- *Evaluación del rendimiento del clasificador.* Es en el Capítulo 4 de experimentos donde evaluamos el rendimiento del sistema que vamos a construir para este PFC. Los mejores resultados que se consiguen son, una tasa de aciertos del 83 % clasificando el automóvil correctamente con un único modelo, y una tasa del 94 % si se identifica correctamente el vehículo entre los 5 modelos que más parecido tienen con la imagen.

- *Implementación en C++ del clasificador.* Construimos una interfaz gráfica usando C++ y la biblioteca OpenCV. El usuario de la aplicación podrá cargar una imagen frontal de un automóvil, y el sistema clasificará el vehículo por su marca y modelo (será necesario indicarle la localización del vehículo seleccionando las esquinas de la matrícula en la imagen).

6.3. Trabajos futuros

El trabajo realizado en este PFC (reconocimiento de vehículos por medio de las características que podemos extraer de una imagen) es solamente el comienzo de un proyecto con un gran margen de mejora. Hay muchos aspectos en los que poder trabajar en un futuro para complementar esta aplicación. En esta sección vamos a tratar de enumerar algunas de estas mejoras:

- *Localización automática del vehículo.* Uno de los mayores problemas que encontrábamos al usar la aplicación, era que necesitábamos las coordenadas de las esquinas de la matrícula para localizar el vehículo a identificar. Para resolver este problema necesitaríamos, o un sistema de localización de placas de matrícula de coches como [9], o un sistema que reconozca el frontal del automóvil en su conjunto como [10]. Combinar el PFC con alguno de estos sistemas supondría una gran mejora pues nos ayudaría a localizar el automóvil evitando la localización de la matrícula en nuestra aplicación.

- *Realizar pruebas con nuevos descriptores.* Una de las piezas básicas que ayudan al buen funcionamiento del sistema diseñado es la utilización de SURF para extraer las características de las imágenes normalizadas. No obstante podríamos haber experimentado con otros descriptores de imágenes como DCT, para ver si así se obtenían mejores resultados. Otra opción es probar el detector de puntos de interés que trae SURF en su implementación y ver si los resultados así son mejores que con la mejor rejilla de los experimentos.

- *Vista trasera de los vehículos.* Otra posible mejora en nuestro sistema sería crear una nueva base de datos con las vistas traseras de los vehículos. Puede ser interesante probar que resultados se obtienen, repitiendo los experimentos del proyecto con imágenes traseras de los coches (tal vez las características de estas imágenes traseras sean más descriptivas que las frontales).

- *Sustituir la imagen frontal por vídeo.* Consiste simplemente en cambiar la imagen frontal del vehículo que queremos clasificar, por un vídeo del mismo [7]. En ese vídeo debe aparecer el automóvil orientado de frente a la cámara de vídeo, de tal forma que, una vez realizada la grabación se pueda descomponer el vídeo en una secuencia de imágenes. La mejora vendría al descomponer en diferentes imágenes el vídeo, para poder clasificar el automóvil desde distintas perspectivas.

- *Implementar la técnica de verificación.* La última mejora que recomendamos será la de diseñar el sistema de verificación con el que hemos trabajado en el Capítulo 4 de experimentos. Igual que hemos hecho una aplicación para clasificar la marca y el modelo de un coche, lo suyo sería hacer otra interfaz distinta que nos permita cargar dos imágenes frontales de un automóvil, de manera que el sistema indique si son o no de la misma marca y modelo.

Bibliografía

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

http://vellum.cz/-mikc/oss-projects/CarRecognition/doc/dp.pdf

[2] Louka Dlagnekov *Video-based Car Surveillance: License Plate, Make, and Model Recogntion*, 4 Oct. 2005. Department of Computer Science and Engineering, University of California, San Diego.

http://cseweb.ucsd.edu/classes/fa05/cse252c/cse252c-lpr-mmr.pdf

[3] David G. Lowe *Object Recognition from Local Scale-Invariant Features*, September 1999. International Conference on Computer Vision, Greece, Page:1150-1157.

http://www.cs.ubc.ca/-lowe/papers/iccv99.pdf

[4] Herbert Bay, Tinne Tuytelaars y Luc Van Gool *SURF: Speeded Up Robust Features*, Computer Vision and Image Understanding (CVIU), Vol. 110, 2008, Page:346-359,

ftp://ftp.vision.ee.ethz.ch/publications/articles/eth-biwi-00517.pdf

[5] Oren B., Eli S. y Michal Irani *In Defense of Nearest-Neighbor Based Image Classification*, June 2008. Computer Vision and Pattern Recognition. IEEE Conference, Page:1-8

www.wisdom.weizmann.ac.il/-irani/InDefenceOfNN-CVPR08.pdf

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

http://www.cs.bris.ac.uk/Publications/Papers/2000970.pdf

[7] Hazim K. Ekenel y Johannes Stallkamp *Video-based Face Recognition on Real-World Data*, Octubre 2007. Page:1-8. IEEE 11th International Conference

[8] T. Natarajan y K. R. Rao *Discrete Cosine Transform*, 1974. IEEE Transactions on Computers. C-32, Enero 1974. Page:90-93

[9] Miguel Ángel Fernández Díaz *Detección de matrículas en imágenes*, 2008. Proyecto Fin de Carrera, Ingeniería Técnica en Informática de Sistemas. Escuela Superior de Ingeniería Informática URJC.

[10] Paul Viola y Michael Jones *Robust Real-time Object Detection*, ICCV 2001: 747.

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

[12] Josef Sivic y Andrew Zisserman *Video Google: A Text Retrieval Approach to Object Matching in Videos*, 2003. International Conference on Computer Vision.

[13] Gady Agam *Introduction to programming with OpenCV*, Enero 2006. Department of Computer Science. Illinois Institute of Technology.

[14] David M. Mount *ANN Programming Manual*, 2006. Department of Computer Science and Institute for Advanced Computer Studies. University of Maryland.

[15] J. García, J. I. Rodríguez y J. Vidal *Aprenda Matlab 7.0*, Diciembre 2005. Escuela Técnica Superior de Ingenieros Industriales UPM.

[16] Bruce Eckel *Thinking in C++*, Enero 2000. Prentice Hall, New Jersey.

ISBN: 0-13-979809-9.

[17] Tobias Oetiker, H. Partl y I. Hyna *The Not So Short Introduction to LaTeX 2*, September 2008.

http://tobi.oetiker.ch/lshort/lshort.pdf

[18] Christopher Evans, 2009. OpenSURF library. University of Bristol.

http://code.google.com/p/opensurf1/

[19] Colin Laplace, H. Lai y M. Berg, 2009. Download Dev-C++ 4.

http://www.bloodshed.net/dev/devcpp.html

[20] Xavier Delacour, 2008. Paquete ANN para trabajar desde octave.

http://octave-swig.sourceforge.net/octave-ann.html

[21] OpenCV 2.0 (K Nearest Neighbors), 2009. Manual de KNN en OpenCV 2.0.
opencv.willowgarage.com/documentation/k-nearest-neighbors.html

[22] Wikipedia, 2010. Homografía.
http://en.wikipedia.org/wiki/Homography

[23] Wikipedia, 2010. Warping.
http://en.wikipedia.org/wiki/Image-warping