Universidad Zaragoza

Proyecto Fin de Carrera

Sistema de vigilancia y control de personas basado en la detección y reconocimiento de caras mediante visión por computador

Autor

Martín Gracia Valero

Director

Carlos Sagüés Blázquiz

Universidad de Zaragoza / Escuela de Ingeniería y Arquitectura

2012

Agradecimientos

Gracias de todo corazón a mis padres, mi hermano, mi mujer y mi hija, por hacerme sentir afortunado y con fuerzas para terminar este proyecto.

Sistema de vigilancia y control de personas basado en la detección y reconocimiento de caras mediante visión por computador

RESUMEN

La seguridad se ha convertido en un tema prioritario en la sociedad actual. Los sistemas de video vigilancia han proliferado de manera clara para cubrir esta necesidad. Los grandes lastres de estos sistemas son tanto el desembolso económico que requieren en algunas ocasiones como, sobre todo, el gran consumo de espacio en disco que necesitan para el almacenamiento de video.

Con un escenario como este, se propone desarrollar un sistema básico de vigilancia y control de personas que consiga ahorro en espacio de almacenamiento y ofrezca bajo coste. Este sistema estará basado en la detección de personas a través de sus caras utilizando visión por computador. La detección se realizará sobre la imagen proporcionada por una cámara fija de vigilancia o similar.

Con el propósito de obtener el necesario ahorro en almacenamiento, en lugar de almacenar todo el video, el sistema guardará una parte esencial de la información proporcionada: la cara de la persona detectada. Esta información podrá ser posteriormente post-procesada en caso de necesidad. El reconocimiento de la persona detectada (quien es esa persona), que es uno de los objetivos de este proyecto, forma parte de este posible post-procesado.

Dentro del amplio abanico de sistemas de detección de caras, el método desarrollado por Paul Viola y Michael Jones en el año 2001 [1] constituyó todo un logro en el campo ya que permitió que sistemas de detección de objetos pudieran ser usados en entornos de tiempo real (video) manteniendo e incluso mejorando la eficacia de la detección de los sistemas sobre imagen fija (foto). La detección de caras que se realiza en este proyecto tiene como base este método. En este proyecto se hace un cuidado análisis del trabajo de Viola-Jones.

Para el reconocimiento de las caras, se va a usar una técnica denominada *Eigenfaces* [2], basada en el análisis de componentes principales (ACP). Es un método de reconocimiento en 2D muy popular y que ofrece resultados más que aceptables.

Se estudiará asimismo la posibilidad de incluir el etiquetado de caras para flexibilizar las búsquedas de personas detectadas, inspirado en lo que ofrecen aplicaciones como *Picasa* de *Google* o *iPhoto* de *Apple*.

Para obtener a su vez un bajo coste económico se va a hacer uso de la librería de código libre *Open Source Computer Vision Library* (*OpenCV*) de *Intel*, referente en el desarrollo de aplicaciones libres de visión por computador, así como del entorno de desarrollo integrado de *Microsoft Visual C++* en su versión *Express*.

Tabla de contenidos

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

La seguridad se ha convertido en un tema prioritario en la sociedad actual. Los sistemas de video vigilancia han proliferado de manera clara para cubrir esta necesidad. Sin embargo, estos sistemas tienen grandes lastres, como el desembolso económico que requieren en algunas ocasiones y sobre todo el gran consumo de espacio en disco que necesitan para el almacenamiento de video. Si bien es verdad que desde hace unos años la capacidad de almacenamiento ha dejado de ser un asunto problemático y caro, cada vez los sistemas ofrecen mayor capacidad a un menor precio y una de las claves de todo sistema de vigilancia es su posibilidad de compresión para minimizar ese coste de almacenamiento.

Con un escenario como este, el presente proyecto ofrece un sistema básico de vigilancia y control de personas que consigue un gran ahorro en espacio de almacenamiento y ofrece muy bajo coste económico. El sistema está basado en la detección de personas a través de sus caras utilizando visión por computador. La detección se realizará sobre la imagen proporcionada por una cámara fija de vigilancia o similar.

Con el propósito de obtener el necesario ahorro en almacenamiento, en lugar de almacenar todo el video, el sistema guarda una esencial parte de la información proporcionada: la cara de la persona detectada. Esta información puede ser procesada posteriormente en caso de necesidad para, por ejemplo, reconocer quién es esa persona detectada.

El pilar básico en el que se apoya este sistema de vigilancia es la detección de caras. Dentro del amplio abanico de sistemas de detección de objetos y en concreto de caras, el método desarrollado por Paul Viola y Michael Jones en el año 2001 [1], robusto a la vez que muy rápido, constituyó todo un logro en el campo, ya que permitió que sistemas de detección de objetos pudieran ser usados en entornos de tiempo real (video) manteniendo e incluso mejorando la eficacia de la detección de los sistemas sobre imagen fija (foto). La detección de caras que se realiza en este proyecto tiene como base este método. Un amplio análisis del trabajo de Viola-Jones se expone en uno de los capítulos de esta memoria.

Como se ha mencionado previamente, el sistema permite el post-procesado de la información recogida por el detector. Una de las posibles tareas de post-procesado, implementada en este sistema, es el reconocimiento de la persona detectada, es decir, saber quién es la persona de interés. Para el reconocimiento de las caras, se hace uso de una técnica denominada *Eigenfaces* [2], basada en el Análisis de Componentes Principales (ACP). Es un método de reconocimiento 2D muy popular y que ofrece resultados más que aceptables. Se ha decidido utilizar este método debido a que encaja perfectamente en el ámbito de este proyecto por su sencillez y sus resultados.

1.1 Contexto

Este trabajo se encuadra en el contexto del campo de visión por computador, (*computer vision)*. Dentro de ese amplio campo, la detección de objetos es un tema de gran interés y en el que está basado este proyecto.

1.1.1 **Visión por computador**

La visión por computador, también conocida como visión artificial o visión por ordenador[1], es un sub-campo de la inteligencia artificial. El propósito de la visión por computador es conseguir que un ordenador sea capaz de extraer información relevante para "interpretar" y "entender" la escena. Los datos de la imagen pueden adoptar muchas formas, como por ejemplo secuencias de vídeo (*stream*), cámaras en tiempo real (cámara web) o datos multidimensionales como los de un escáner médico.

El reconocimiento es considerado como el problema fundamental en la visión artificial. Se trata de determinar si los datos de una imagen contienen un objeto, característica o actividad específica. Esta tarea, que normalmente puede ser resuelta con firmeza y sin esfuerzo por un humano, aún no está resuelta de manera satisfactoria en la visión artificial para el caso más general: objetos arbitrarios en situaciones o imágenes arbitrarias. Los métodos actuales para hacer frente a este problema sirven para resolverlo sólo para determinados objetos, como objetos geométricos simples (por ejemplo, los poliedros), rostros humanos, caracteres escritos a mano o impresos y vehículos. Además estos han de estar en situaciones específicas de buena iluminación y fondo contrastado.

Una versión ampliada de esta explicación sobre la visión por computador se puede consultar en el Anexo A.

1.1.2 **Detección de objetos. Detección de caras**

La detección de objetos es una tecnología informática relacionada con la visión por ordenador y el procesamiento de imágenes. Se trata de detectar instancias de objetos de una clase determinada, como por ejemplo personas, edificios o coches, en imágenes digitales o vídeos. La detección de caras puede considerarse como un caso concreto de la detección de objetos.

Para definir de una manera sencilla lo que es la detección de caras bastaría decir que se trata de la tecnología que sirve para determinar la ubicación y los tamaños de rostros humanos en imágenes digitales. Detecta rasgos faciales y no considera cualquier otra cosa como edificios, árboles o vehículos. Siendo más precisos, dada una imagen, el objetivo de la detección de caras es identificar todas las regiones de la imagen que contienen alguna cara, independientemente de su posición tridimensional, su orientación y las condiciones de iluminación. Ese problema es difícil porque se enfrenta al hecho de que las caras tienen un alto grado de variabilidad en el tamaño, forma, color y textura.

En los últimos años, se han desarrollado numerosas técnicas para la detección de caras en una sola imagen. Los primeros algoritmos de detección se centraban en la detección de rostros humanos frontales, mientras que los nuevos y más complejos algoritmos tratan de resolver el problema más general y complicado de la detección de caras en cualquier posición, es decir, caras que pueden aparecer rotadas en el eje horizontal, en el vertical o en ambos.

Para este proyecto se decidió seguir el método de Paul Viola y Michael Jones, un sistema rápido y robusto de detección de objetos que garantiza un alto nivel de acierto en muy poco tiempo, lo que le permite ser utilizado en aplicaciones de tiempo real. La elección de ese trabajo como uno de los pilares de este proyecto se corresponde tanto a la sugerencia del director como al hecho de que sea uno de los trabajos más citados por la comunidad científica en cuanto a visión por computador se refiere.

[1] A lo largo de este proyecto se hará mención al concepto general de visión por computador a través de tres términos equivalentes: visión artificial, visión por ordenador o visión por computador.

1.1.3 **Reconocimiento de caras**

El reconocimiento de caras puede considerarse un área muy activa en el campo de visión por computador. Es un proceso que se lleva estudiando activamente desde hace varias décadas, produciendo diferentes aplicaciones para sistemas de seguridad, robótica, interfaz hombre-máquina, cámaras digitales, juegos y un largo etcétera.

El reconocimiento de caras requiere de dos fases:

- Detección de caras: obtener cualquier cara de una imagen para facilitar su posterior procesado.

- Reconocimiento de caras: la cara detectada y procesada se compara con una base de datos de caras conocidas, buscando encontrar la mejor coincidencia.

Aunque la detección de caras se realiza hoy en día con un alto nivel de efectividad (90%-95%), el reconocimiento rápido y simple sigue algo lejos de esa precisión. Existen multitud de técnicas de reconocimiento, entre las que destacan:

- *Eigenfaces*, o análisis de componentes principales (*Principal Component Analysis method* - PCA) [2].

- *Fisherfaces* o análisis de discriminantes lineales *(Linear Discriminant Analysis method* - LDA) [26].

- Modelo Oculto de Markov (*Hidden Markov Models* - HMM) [27].

- Modelo de Apariencia Activa (*Active Appearance Models* AAM) [28].

En este proyecto se hace uso del método de *Eigenfaces*, basado en el análisis de componentes principales. Este es un método popular de reconocimiento de caras que permite obtener resultados aceptables en poco tiempo.

1.2 **Objetivos**

Los objetivos concretos del presente proyecto de fin de carrera son:

- Estudiar el método de Paul Viola y Michael Jones para la detección rápida de objetos.

- Detectar caras en fotos, video y en tiempo real sobre la imagen de una cámara web. La Figura 1 muestra un ejemplo de lo que se considera detección de caras en una foto.

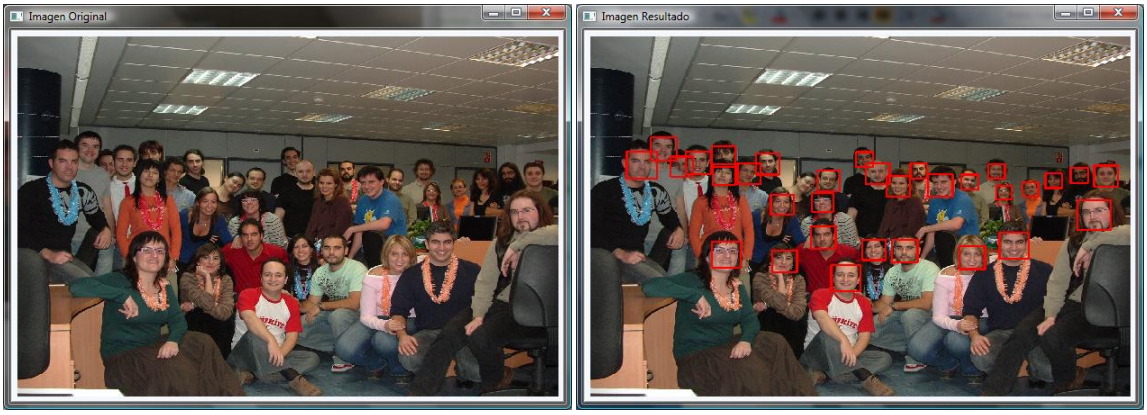


Figura 1. A partir de la foto original (izda.), se detectan las regiones de la misma que contienen alguna cara (dcha.).

- Reconocer caras. A partir de una base de datos de caras conocidas, encontrar de entre todas ellas la mejor coincidencia con respecto a la persona de interés.

- Haciendo uso de la detección y el reconocimiento, implementar un sistema completo de vigilancia y control que detecte y sea capaz de reconocer personas, minimizando el espacio de almacenamiento a utilizar con respecto a un sistema de video más convencional.

- Estudiar la viabilidad de añadir un etiquetado automático de personas.

- Minimizar el coste económico haciendo uso de software libre.

Como se puede observar a lo largo de este documento, cada uno de esos objetivos ha estado presente en el desarrollo del proyecto.

1.3 Entorno de desarrollo

Para la realización de este proyecto se ha hecho uso principalmente de dos herramientas software: *OpenCV* y *Microsoft Visual C++*.

Para todas las tareas relacionadas con el tratamiento de imágenes se ha usado la biblioteca *OpenCV* (*Open Source Computer Vision*), una biblioteca de código abierto que incluye actualmente más de 2500 algoritmos para el procesamiento de imágenes, visión por ordenador y otros algoritmos numéricos de propósito general.

Por otro lado, la implementación del sistema se ha realizado en el entorno de desarrollo integrado de *Microsoft Visual C++* en su versión gratuita (*Express Edition*).

Más detalles sobre estas dos herramientas se encuentran en el Anexo B de esta memoria.

1.4 Trabajos previos consultados

Muchos son los trabajos existentes que abordan el tema de la detección de caras dentro de la bibliografía actual relacionada con la visión por ordenador. Para la realización de este proyecto se han consultado varios de estos trabajos, estudiando en profundidad el presentado por Paul Viola y Michael Jones en las conferencias sobre visión por computador y reconocimiento de patrones en los años 2001 [1] y 2004 [9]. Un listado más completo de

artículos consultados acerca de la detección de objetos y caras se encuentra disponible en el Anexo C.

En el ámbito del reconocimiento, el artículo de la revista *ServoMagazine* [11], publicado en Abril de 2007, acerca de un reconocedor de caras basado en el método de Análisis de Componentes Principales (ACP), cuya implementación se apoya en la librería de Intel *OpenCV* (*Open Source Computer Vision Library*), ha sido clave en la posterior implementación del reconocedor en este proyecto.

Para concluir esta sección, cabe destacar los trabajos realizados por Modesto Fernando Castrillón Santana [29], profesor de la Universidad de Las Palmas de Gran Canaria, que dejan patentes todas las posibilidades que este campo de la detección de caras ofrece en el presente y puede ofrecer en el futuro.

1.5 Resumen de capítulos

Se muestra a continuación una breve descripción del contenido del resto de secciones de la memoria, separado por capítulos.

- Capitulo 2: Detección de objetos y caras. Estudio en profundidad del detector de Viola-Jones.

- Capitulo 3: Este capítulo detalla el diseño e implementación del detector de caras realizado en este proyecto.

- Capítulo 4: Capítulo donde se expone el reconocedor de caras implementado.

- Capítulo 5: Detalla el sistema de vigilancia obtenido con sus características y funcionalidades, incluyendo el interfaz grafico de la aplicación final. Este capítulo contiene el estudio realizado acerca de la posibilidad de incluir un etiquetado automático de caras.

- Capítulo 6: Conclusiones del proyecto.

El resto del contenido de la memoria lo forman los anexos con información adicional para referencia del lector.

Capítulo 2 - Estudio y evaluación de un esquema rápido y robusto para la detección de objetos y caras: el detector de Viola-Jones

El detector de caras es el pilar fundamental del sistema desarrollado en este proyecto debido a que el resto de características y funcionalidades de la aplicación dependen de la precisión, robustez y rapidez del detector.

Este capítulo se centra en el detector de objetos desarrollado por Paul Viola y Michael Jones [1] [9], conocido como detector de Viola-Jones.

Sin embargo, antes de analizar dicho trabajo, se amplía lo comentado en la introducción acerca de la detección de caras y objetos con el fin de contextualizar de manera más precisa lo expuesto a continuación.

2.1 Detección de objetos y caras

La detección de objetos consiste en la obtención de instancias de objetos de una determinada clase en imágenes digitales o vídeos. Estos objetos a detectar pueden ser cualquier cosa, desde una carretera o un coche, hasta objetos más pequeños como un lápiz o un ojo. Entre todos ellos, el objeto que más estudios ha generado es el rostro humano.

La tecnología de detección de caras se entiende como un método para obtener la presencia, ubicación y tamaño de rostros humanos en imágenes digitales aleatorias. Para ser más precisos, intenta detectar aquellas zonas de una imagen que contienen una cara, dejando a un lado todo lo demás.

Figura 2. Ejemplo de uso de la tecnología de detección de caras sobre imagen fija (foto). De la imagen original (izquierda) se detectan aquellas zonas que contienen una cara (derecha).

El caso óptimo de detección sería aquel que consiguiera que la cara se detectase independientemente de su posición, su orientación y otras condiciones propias de la imagen o el entorno como puede ser la iluminación. Sin embargo, este detector óptimo es muy difícil de lograr por muchas razones, siendo la razón fundamental el hecho de que los rostros humanos tienen un alto grado de variabilidad en el color, tamaño, forma e incluso textura.

Son muchos los algoritmos que implementan la tarea de detección de caras como una tarea de clasificación de patrones binarios. Es decir, el contenido de una determinada parte de una imagen se transforma en funciones y un clasificador entrenado con ejemplos de lo que se debe reconocer como cara decide si esa región de la imagen es una cara o no. A la hora de su

implementación, existe un gran desafío al utilizar como método una clasificación de patrones cuando se habla de la detección de objetos, y en concreto de caras. Una imagen natural contiene a menudo muchos más patrones en el fondo de la imagen que patrones representativos de una cara. De hecho, el número de patrones de fondo puede ser de 1000 a 100000 veces mayor que el número de los patrones de una cara. Esto significa que si uno desea una alta tasa de detección combinada con un bajo número de falsas detecciones se necesitan clasificadores muy específicos. Algunos de los algoritmos para la detección de caras publicados obtienen buenos resultados, llegando a sobrepasar el 90% de aciertos con un mínimo nivel de falsos positivos.

Últimamente, muchos de los esfuerzos en investigación acerca del procesamiento de imágenes incluyen la detección o reconocimiento de caras e incluso el reconocimiento de la expresión de la cara. Sin embargo, no son pocos los métodos que se basan en que las caras en una imagen o una secuencia de imágenes han sido previamente identificadas y localizadas. Para construir sistemas totalmente automatizados que analicen la información presente en imágenes que contengan caras, el uso de algoritmos de detección de caras robustos y eficientes es obligatorio. Este hecho es justamente el que hace tan interesante el trabajo de Viola-Jones, analizado ampliamente en este proyecto, ya que su método de detección está considerado como uno de los más eficaces (máximo nivel de acierto y mínimo de fallo) y eficientes (rápido procesamiento de imágenes).

Para terminar esta pequeña introducción, cabe mencionar varias aplicaciones en las que se suele hacer uso de las técnicas de detección de caras:

- En la biometría, a menudo como parte de (o junto a) un sistema de reconocimiento facial.
- En la vigilancia por vídeo.
- En los sistemas con interacción hombre-máquina.

Así mismo, últimamente todas las cámaras digitales incluyen la detección de caras para el ajuste automático del enfoque (*autofocus*).

2.2 Detector de Viola-Jones

Muchos son los artículos que se encuadran en el campo de visión por computador y en concreto en el tema de detección de objetos. Sin duda, entre los artículos más influyentes podemos encontrar los presentados por Paul Viola y Michael Jones, en los que describen un detector de objetos (en este caso caras) rápido, robusto y aplicable en sistemas de tiempo real.

Paul Viola y Michael Jones, en el año 2001, describieron a través de algoritmos e ideas nuevas la posibilidad de obtener un detector de objetos capaz de procesar imágenes rápidamente y alcanzar altas tasas de detección.

Las ideas y algoritmos desarrollados fueron motivados en gran medida por la tarea de detección de rostros. Con este fin, Paul Viola y Michael Jones construyeron un sistema que detecta caras más rápidamente que cualquier otro detector en su momento, pero además logrando unas tasas de falsos positivos que son equivalentes a los mejores resultados publicados hasta la fecha.

En otro tipo de sistemas de detección, información adicional como la diferenciación de imágenes en una secuencia de video o la distinción de colores se utiliza para obtener también una alta tasa de aciertos y pocos falsos positivos. Sin embargo, en el detector expuesto por Viola

y Jones se obtienen iguales e incluso mejores resultados sólo con la información presente en imágenes en blanco y negro.

Como bien remarcan los autores de este trabajo, la obtención de un detector de caras rápido tiene multitud de aplicaciones prácticas, como los sistemas que trabajen con imágenes en tiempo real, por lo que se adapta perfectamente a lo que se busca en este proyecto. Además, en aplicaciones donde altas tasas de detección no son estrictamente necesarias, este detector permite dedicar más tiempo y recursos a procesamientos y análisis adicionales.

En los siguientes apartados de esta sección se detalla el método de Viola-Jones, describiendo las características utilizadas para la detección, una representación especial de las imágenes denominada imagen integral, la creación de clasificadores fuertes a partir de otros más débiles y la obtención de una cascada de esos clasificadores.

2.2.1 **Características simples**

El procedimiento de detección aquí descrito clasifica imágenes basándose tan solo en el valor de características simples. El principal motivo para el uso de características simples en vez de usar píxeles es que, a través de determinadas características, se pueden adquirir unos conocimientos que son difíciles de aprender usando una cantidad de datos finita como son los píxeles. Pero en este sistema también hay una segunda motivación fundamental: el sistema basado en características funciona mucho más rápido que un sistema basado en píxeles.

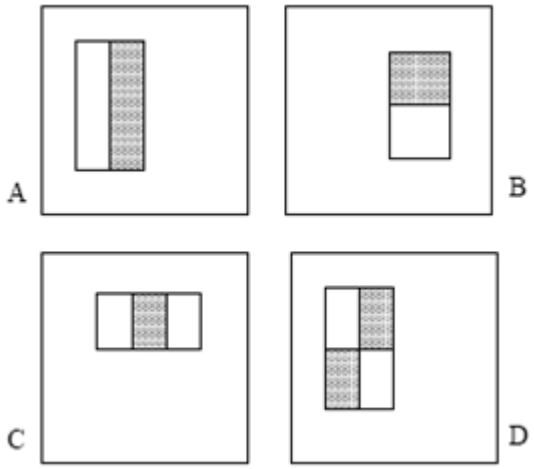


Figura 3. Ejemplos de plantillas de características en forma de rectángulos. Todas ellas son relativas a la ventana de detección sobre la que se aplican. La suma de los píxeles dentro de los límites de los rectángulos blancos se resta de la suma de los píxeles en los rectángulos de color gris. Características de dos rectángulos se muestran en (A) y (B). En (C) se muestra una característica de tres rectángulos y en (D) una de cuatro rectángulos.

Las características utilizadas se basan en la transformada de Haar que fue utilizada por Papageorgiou et al y presentada en su artículo de 1998 [4]. Más concretamente, se hace uso de tres tipos de funciones en forma de rectángulos. El valor de una característica de dos rectángulos es la diferencia entre la suma de los píxeles dentro de dos regiones rectangulares. Las regiones tienen el mismo tamaño y forma y son horizontal o verticalmente adyacentes (Figura 3). En una característica de tres rectángulos se calcula la suma de los píxeles de los dos rectángulos de fuera y ese valor se resta de la suma de píxeles obtenida en el rectángulo del centro. Por último, en características de cuatro rectángulos, el valor se obtiene como la diferencia entre pares diagonales de rectángulos.

2.2.2 **Imagen integral**

Como se ha mencionado, las características usadas en este detector se representan en forma de rectángulos. Con el fin de evitar cálculos redundantes, los autores determinaron que estas características pueden ser calculadas rápidamente usando una representación intermedia de la imagen que los autores denominan imagen integral. La imagen integral en el punto *x,y* contiene la suma de los píxeles que están tanto a la izquierda como encima de *x,y* inclusive (Figura 4):

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y')$$

Donde *ii(x,y)* es la imagen integral e *i(x,y)* es la imagen original.

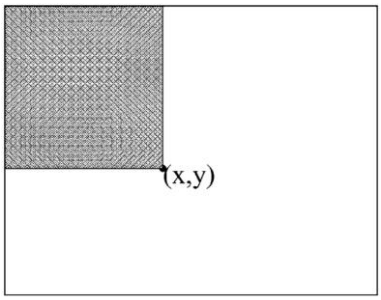


Figura 4. El valor de la imagen integral en el punto (x,y) es la suma de los píxeles situados arriba y a la izquierda del punto.

Usando el siguiente par de recurrencias,

s(x,y) = s(x,y - 1) + i(x,y) (1)
ii(x,y) = ii(x - 1,y) + s(x,y) (2)

donde *s(x,y)* es la suma acumulada de la fila, *s(x, -1) = 0*, y *ii(-1,y) = 0*, la imagen integral puede ser calculada en una sola pasada sobre la imagen original.

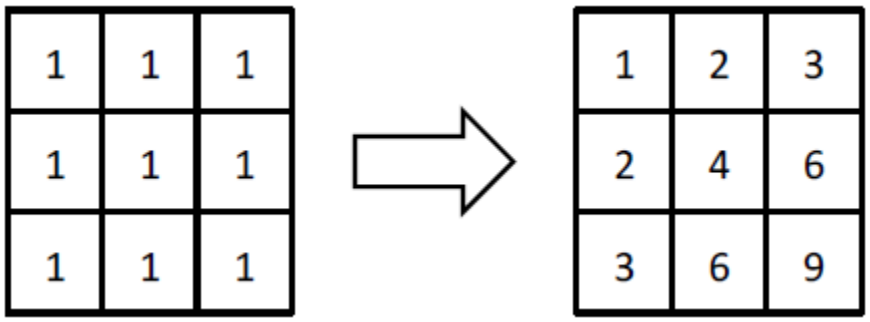


Figura 5. Ejemplo de una imagen de 3x3 (izquierda) junto a su correspondiente imagen integral (derecha).

Usando la imagen integral, cualquier suma de un rectángulo puede ser calculada con cuatro referencias de un vector de valores.

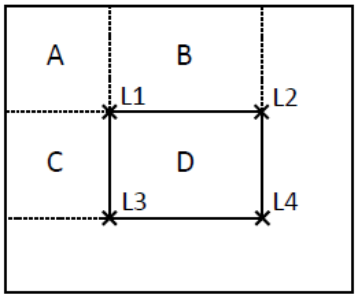


Figura 6. La suma de los píxeles dentro del rectángulo D puede ser calculada con cuatro referencias de un vector. El valor de la imagen integral en el punto L1 es la suma de los píxeles del rectángulo A. El valor en el punto L2 es A + B, en el punto L3 es A + C y en el punto L4 es A + B + C + D. La suma dentro de D puede ser calculada como L4 + L1 - L2 - L3.

Como se aprecia en la Figura 6, la diferencia de sumas de dos rectángulos puede ser calculada con ocho referencias. Debido a que las características de rectángulos definidas anteriormente implican que los rectángulos sean adyacentes, dos de esas ocho referencias son la misma por lo que la diferencia se puede calcular con solamente seis referencias. Del mismo modo, características de tres rectángulos pueden ser calculadas con ocho referencias mientras que con nueve se obtiene la diferencia en el caso de características de cuatro rectángulos.

Aunque los rectángulos como base de las características son primitivos comparados con otras posibles alternativas, los autores sostienen que su uso junto con la imagen integral proporciona una eficiencia que compensa ampliamente su limitada flexibilidad.

2.2.3 **Clasificadores fuertes**

Debido al enorme número de características de rectángulos asociadas con cada sub-ventana de la imagen, aunque cada función se pueda calcular de manera muy eficiente, realizar el cómputo del conjunto completo se antoja impensable. Sin embargo, un número muy pequeño de estas características se pueden combinar para formar un clasificador eficaz. El principal reto es encontrar estas características.

En este sistema, una variante del algoritmo *AdaBoost* se utiliza para impulsar el rendimiento de otro algoritmo de aprendizaje más simple, a veces denominado *weak*, débil, convirtiendo al algoritmo resultante en uno más fuerte.

Con este objetivo, el algoritmo de aprendizaje débil está diseñado para seleccionar aquellas características de rectángulos que mejor separen los ejemplos positivos de los negativos, es decir, que mejor separen las fotos donde aparecen caras de aquellas en las que no aparecen caras. Para cada característica, el clasificador débil determina el umbral óptimo de clasificación, de tal forma que el mínimo número de imágenes son clasificadas de manera errónea. Por lo tanto, un clasificador débil $h(x)$ está compuesto por una característica f, un determinado umbral θ y una función p que determina la dirección del signo de desigualdad:

$$h(x) = \begin{cases} 1 & \text{si } p\, f(x) < p\, \theta \\ 0 & \text{de lo contrario} \end{cases}$$

Donde x es una sub-ventana de 24x24 píxeles de una imagen de ejemplo. El algoritmo *AdaBoost*, en cada ejecución, selecciona una característica de entre todas las potenciales características.

En la Figura 7 y en la Figura 8 se muestran gráficamente un clasificador débil y un clasificador fuerte respectivamente.

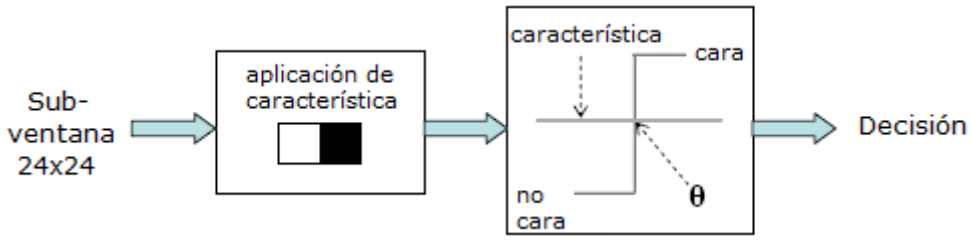


Figura 7. Diagrama de un clasificador débil obtenido a partir de una característica de dos rectángulos.

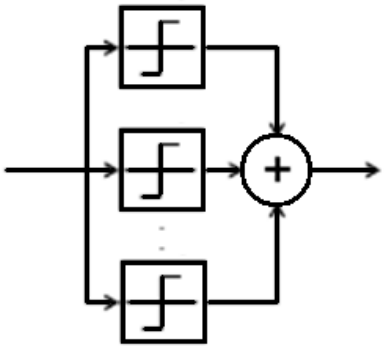


Figura 8. Diagrama de un clasificador fuerte obtenido a partir de varios clasificadores débiles aplicando el algoritmo de AdaBoost.

Como referencia para el lector, se puede consultar el Anexo E para obtener más información sobre el algoritmo de *AdaBoost*.

Prestando atención a las características propiamente dichas, los rectángulos iniciales seleccionados por *AdaBoost* para la tarea de detección de rostros son muy fáciles de interpretar y se presentan en la Figura 9.

Figura 9. Primera y segunda características seleccionada por AdaBoost. Las dos características se muestran solas en la fila superior y superpuestas sobre una cara de ejemplo en la fila inferior.

La primera característica seleccionada se centra en el hecho de que la región de los ojos es a menudo más oscura que la región de la nariz y las mejillas, mientras que la segunda característica seleccionada se basa en la propiedad de que la zona de los ojos es a menudo más oscura que el puente de la nariz.

2.2.4 **Cascada de clasificadores**

En esta sección se describe un algoritmo para la construcción de una cascada de clasificadores que logre una mayor tasa de detección y al mismo tiempo reduzca el tiempo de cálculo todo lo posible.

La principal idea se basa en el hecho de que se pueden obtener clasificadores muy pequeños, y por tanto muy eficientes, que rechazan muchas de las entradas (sub-ventanas de la imagen) negativas mientras que detectan casi todas las entradas positivas. Los clasificadores más simples se usan de esta forma para rechazar la mayoría de las sub-ventanas negativas antes de que clasificadores más complejos se usen para lograr bajas tasas de falsos positivos.

La forma general del proceso de detección es la de un árbol de decisiones, llamado por los autores como “cascada" [1] (ver Figura 10). Un resultado positivo de la primera etapa activa la evaluación del clasificador de la segunda, que es más complejo pero también se ha ajustado para alcanzar tasas de detección mayores. Un resultado positivo de esta segunda desencadena la evaluación del clasificador de una tercera, y así sucesivamente. Un resultado negativo en cualquier punto lleva inmediatamente a rechazar la sub-ventana de la imagen analizada a la que no se le aplica ningún clasificador más.

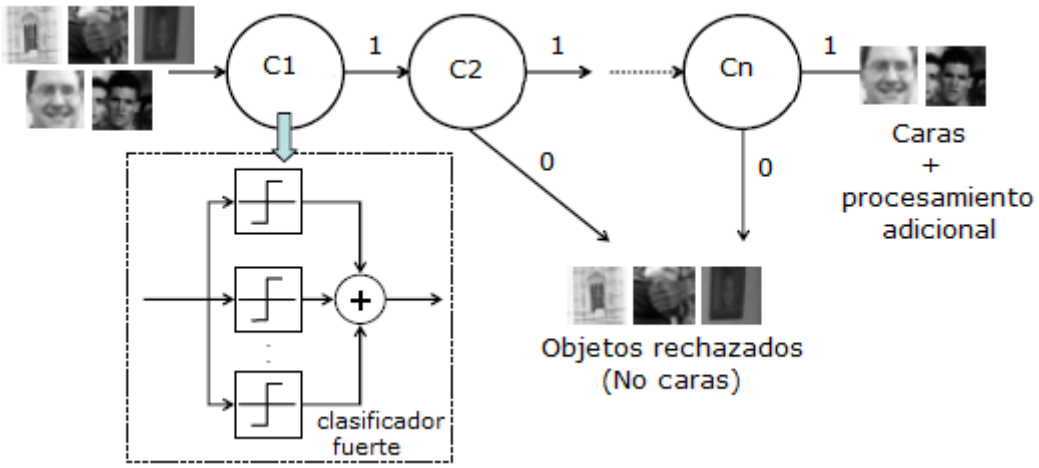


Figura 10. Representación esquemática de la cascada de clasificadores usada para la detección de caras. Una serie de clasificadores fuertes (C1, C2, … Cn) se aplican a todas las sub-ventanas de la imagen a analizar. El clasificador inicial elimina un gran número de ejemplos negativos con muy poco tiempo de cálculo y procesado. Las etapas siguientes eliminan más ejemplos negativos pero requieren cálculos adicionales. Tras unas cuantas etapas de procesado, el número de sub-ventanas de la imagen que contienen caras se reduce de manera drástica. Además, para un posterior procesamiento pueden usarse nuevas etapas en cascada (como se usa en este sistema) o bien cualquier otro sistema de detección alternativo.

Las diferentes etapas de la cascada de clasificadores están constituidas por clasificadores fuertes obtenidos usando *AdaBoost* y ajustando posteriormente el umbral que minimice los falsos negativos. En general, un umbral más bajo proporciona un rendimiento superior en

cuanto a la tasa de detección de ejemplos positivos pero conlleva también una alta tasa de falsos positivos.

Para demostrar las virtudes que una estructura en cascada proporciona a este sistema de detección, los autores realizaron un experimento muy simple en el que comparaban un clasificador fuerte de 200 características con un clasificador de 10 etapas de clasificadores de 20 características. Aunque las diferencias en precisión entre ambos clasificadores eran mínimas, el clasificador en cascada era casi diez veces más rápido.

El rendimiento de la estructura de la cascada refleja el hecho de que dentro de una única imagen existen una gran cantidad de sub-ventanas que no contienen ninguna cara y por lo tanto son sub-ventanas a rechazar. De esta forma, la cascada rechaza el mayor número de ejemplos negativos como sea posible en la fase más temprana. Aunque un ejemplo positivo provoca la evaluación del siguiente clasificador de la cascada más complejo, esto ocurre con relativa poca frecuencia.

2.2.5 **Pruebas y resultados**

Para las pruebas de este sistema, Viola y Jones usaron un clasificador de 38 etapas entrenado para detectar caras frontales orientadas verticalmente. El número de características en las primeras cinco etapas del detector es 1, 10, 25, 25 y 50 respectivamente. El resto de etapas contiene cada vez más características. El número total de características en todas las capas es 6061.

Cada clasificador de la cascada fue entrenado con 4916 imágenes con caras (así como sus imágenes simétricas para un total de 9832) y aproximadamente 10000 sub-ventanas sin caras, todas de tamaño 24x24 píxeles.

Una vez entrenado, el detector escanea la imagen de entrada a través de múltiples escalas y posiciones. Las diferentes escalas se obtienen modificando el detector en vez de la imagen. El detector también escanea la imagen en diferentes posiciones simplemente desplazando la ventana de detección unos cuantos píxeles (Δ). El valor de Δ se ve afectado por la escala del detector de tal manera que si la escala es s la ventana es desplazada $[s\Delta]$, donde $[]$ es una operación de redondeo. La elección de Δ afecta tanto a la velocidad del detector como a la precisión. A mayor valor, menor precisión y viceversa.

Debido a que una imagen se procesa a diferentes escalas y en diferentes posiciones, ocurre con relativa frecuencia que una misma zona de la imagen da como resultado varias detecciones positivas. Para que el sistema devuelva como resultado una sola detección por cara, todas detecciones se agrupan en subconjuntos formados por detecciones que se superponen en alguno de sus bordes. El resultado final se obtiene hallando la media de las esquinas de las detecciones dentro de cada subconjunto.

Para demostrar el rendimiento de este detector, Viola y Jones hicieron uso del conjunto de imágenes del test MIT+CMU [30]. Este test, que combina imágenes obtenidas en el Instituto Tecnológico de Massachusetts (*Massachusetts Institute of Technology*, MIT) y en la Universidad de Carnegie Mellon (*Carnegie Mellon University*, CMU), consta de 130 imágenes que contienen un total de 507 caras. Probado con este test, el detector de Viola-Jones evalúa de media tan solo 10 características del total de 6061 por cada sub-ventana analizada. Esto es posible gracias a que la gran mayoría de sub-ventanas son rechazadas en la primera y en la segunda etapa de la cascada.

Los resultados realizados plasmaron que el detector de Viola-Jones es aproximadamente 15 veces más rápido que el detector de Rowley-Baluja-Kanade, que usa una combinación de

redes neuronales [5], y 600 veces más rápido que el de Schneiderman-Kanade, sistema en el que se utiliza un conjunto de modelos estadísticos para capturar la variación en la apariencia de las caras [6].

En la Tabla 1 aparece una comparativa de los resultados en cuanto a las tasas de detección y el número de falsos positivos entre el detector de Viola Jones y los otros detectores nombrados. Todos han sido evaluados con el test MIT+CMU.

Detector \ Falsos positivos	**10**	**31**	**50**	**65**	**78**	**95**	**167**
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%
Rowley-Baluja-Kanade	83.2%	86.0%	—	—	—	89.2%	90.1%
Schneiderman-Kanade	—	—	—	94.4%	—	—	—

Tabla 1: Tasa de detección y número de falsos positivos de varios detectores probados con el test MIT+CMU. Para facilitar la comparación entre sistemas, se presenta el número de falsos positivos en vez de la tasa de falsos positivos. Para calcular la tasa de falsos positivos simplemente hay que dividir el valor mostrado entre el número de sub-ventanas analizadas. En los experimentos realizados, el número de sub-ventanas analizadas es de 75.081.800.

2.3 Conclusiones

El sistema de detección de caras de Viola-Jones analizado en este capítulo proporciona resultados altamente satisfactorios con un bajo coste computacional. Esto lo convierte en un sistema muy eficaz para usar en aplicaciones de tiempo real.

Todos los algoritmos, nuevas técnicas y representaciones que aquí se presentan son bastante genéricos y pueden extenderse de manera sencilla a la detección de cualquier tipo de objetos, así como a otras tareas de visión por computador y procesamiento de imágenes.

Cabe mencionar que en los artículos de Viola y Jones se presentan un conjunto de experimentos que demuestran las virtudes de este detector en un entorno complejo, ampliamente estudiado en el ámbito de la detección de caras. Del mismo modo, el estudio sobre rendimiento de detectores de caras realizado por el profesor Modesto Castrillón *et ál*. en 2008 [8], avala de manera clara el gran rendimiento del detector de Viola-Jones.

Existen otros trabajos sobre la evaluación de distintos algoritmos de detección de caras. Puede consultarse por ejemplo el publicado por Ming-Hsuan Yang *et ál*. [10] en el año 2002, en el cual se categorizan y evalúan las técnicas desarrolladas para detectar caras en una imagen.

Capítulo 3. Detección de caras (*face detection*)

La detección de caras se puede entender como la base sobre la que se sustenta este proyecto. Si bien es cierto que el sistema final obtenido presenta múltiples funcionalidades más avanzadas que la simple detección, sin un correcto y eficiente método de detección de caras, casi ninguna de estas funcionalidades sería posible.

Como se ha explicado en el capítulo anterior, uno de los métodos de detección de objetos, y en concreto de caras, que más relevancia ha tenido en este ámbito ha sido el de Viola-Jones. Este detector, adaptado y mejorado por Rainer Lienhart, es el que se encuentra disponible en la biblioteca de *OpenCV* [17] y el que se ha usado en el desarrollo de este proyecto.

3.1 Entrenamiento (*haartraining*)

Para usar el detector de objetos de *OpenCV*, lo primero que hay que obtener es un buen clasificador[2] que sea capaz de diferenciar rápidamente los objetos que se pretenden detectar.

La instalación por defecto de *OpenCV* proporciona al desarrollador un conjunto bastante aceptable de clasificadores. En la versión 2.3, la lista incluye, entre otros, clasificadores para detectar:

- Caras
- Pares de ojos (con o sin gafas)
- Ojos sueltos (derecho o izquierdo)
- Cuerpo (entero, parte de arriba o parte de abajo)
- Nariz
- Boca

Todos ellos se presentan en formato XML listos para ser usados directamente por cualquier aplicación que se desarrolle haciendo uso de *OpenCV*, sin necesidad de nada más que referenciarlos en el código en los lugares adecuados. En el Anexo F se muestra un listado más detallado con algunos clasificadores disponibles con la instalación por defecto de *OpenCV* para referencia del lector.

Por otro lado, *OpenCV* proporciona además ciertas herramientas para poder crear detectores bajo demanda. En ese caso, es posible crear clasificadores para cualquier tipo de objetos, siempre y cuando se siga un proceso determinado de entrenamiento y se utilicen un número suficiente de muestras positivas (contienen el objeto a detectar) y negativas (no contienen el objeto a detectar). Ese número puede oscilar entre 5000 y 10000, dependiendo de la calidad del clasificador que se quiera generar.

Por describirlo brevemente, el mecanismo de entrenamiento es similar al entrenamiento de un árbol de decisiones. Cada etapa del clasificador es entrenada solamente con las imágenes (muestras) que hayan pasado todas las etapas anteriores. De esta forma, se consigue dejar para etapas posteriores tareas de clasificación más costosas ya que serán evaluadas pocas veces.

La instalación de *OpenCV* incluye junto con las herramientas propias de entrenamiento (todas cuyo código está disponible en *<directorio de instalación de OpenCV>\modules\haartraining*) un documento explicativo que describe el proceso que hay

[2] Aunque existe una ligera diferencia técnica entre los conceptos de clasificador y detector, a lo largo de este documento pueden aparecer intercambiados ya que básicamente cumplen la misma función.

que seguir para obtener el detector deseado. Este documento, llamado **haartraining.htm**, se encuentra en la ruta de instalación de *OpenCV*.

Los principales inconvenientes de generar clasificadores propios son, por un lado, el gran número de muestras positivas y negativas que es preciso recopilar, y por otro lado, el tiempo que se tarda en general el clasificador, que puede variar de algunos días a varias semanas dependiendo de la capacidad y recursos del sistema donde se entrena.

Existen en la red referencias a múltiples experimentos sobre la generación de detectores bajo demanda usando las herramientas de *OpenCV*. Uno bastante completo y bien documentado, creado por un desarrollador japonés llamado Naotoshi Seo, se encuentra disponible en su página web y sirve de ejemplo de cómo se debe seguir el proceso de entrenamiento de *OpenCV* para obtener un clasificador de caras[3]. Otro ejemplo interesante es el artículo de Florian Adolf donde realiza un entrenamiento para generar un detector de tazones (*bowls*)[4].

El hecho de que el objeto que se busca detectar en este proyecto es el rostro humano hace que existan clasificadores muy fiables ya disponibles con *OpenCV*, sin que aparezca la necesidad de crear nuevos. La cara, como se ha mencionado en el capítulo sobre el detector de Viola-Jones, ofrece unos rasgos muy identificativos difícilmente localizables en otras partes de la anatomía humana o incluso en otro tipo de objetos. Por tanto la cara es un objeto muy estudiado y que además ofrece resultados altamente satisfactorios en su detección.

El análisis comparativo de los detectores disponibles y compartidos por la comunidad de *OpenCV* llevado a cabo por el profesor de la Universidad de Las Palmas de Gran Canaria M. Castrillón Santana junto a varios colegas [8], condujo al autor de este proyecto a usar para la parte de detección de caras del sistema el clasificador **haarcascade_frontalface_alt2.xml** presente con la instalación de *OpenCV*. Este clasificador, muy rápido y efectivo, detecta caras en posición frontal o ligeramente rotadas y fue creado por Rainer Lienhart basado en imágenes de caras de 20x20 píxeles. Los clasificadores que R. Lienhart implementó para *OpenCV* difieren ligeramente de los descritos en el método de Viola-Jones, ya que usan un árbol de clasificadores [16] en vez de una cascada de clasificadores, pero la esencia es la misma.

Una vez seleccionado el clasificador a utilizar, parece preciso saber cuál es su funcionamiento junto con las funciones de *OpenCV* relacionadas con la detección de objetos.

3.2 Detección

Con la biblioteca de *OpenCV* es muy sencillo utilizar un clasificador para escanear una imagen en busca un cierto objeto. Simplemente hay que hacer uso de la función **cvHaarDetectObjects()**.

La función **cvHaarDetectObjects()** se encarga de realizar una comprobación completa de la imagen a analizar desplazando ventanas escaneo que recorren toda la imagen en busca del objeto con el que el clasificador fue entrenado. Internamente, esta función ejecuta otra función llamada **RunHaarClassifierCascade**() que, en el caso de que la ventana de búsqueda encuentre una cara (la región analizada no ha sido descartada por ninguna de las etapas del detector), devuelve 1, devolviendo 0 en caso contrario. Cuando toda imagen ha sido analizada, **cvHaarDetectObjects()**devuelve una secuencia de rectángulos que señalan las detecciones encontradas.

[3] http://note.sonots.com/SciSoftware/haartraining.html
[4] http://es.scribd.com/doc/6993383/OpenCV-Object-Detection-HowTo

La implementación de este detector sobre *OpenCV* permite que la ventana de búsqueda pueda modificar su tamaño para analizar la imagen a diferentes escalas, aspecto que incide directamente en la eficiencia del detector ya que es mucho más eficiente redimensionar la ventana de búsqueda que la imagen. Este trabajo lo realiza otra función de nombre **cvSetImagesForHaarClassifierCascade()** que es llamada por **cvHaarDetectObjects()**.

Analizando un poco más la función **cvHaarDetectObjects()** cabe mencionar que, además de la imagen y el clasificador, es preciso proveerla de la información adicional que se detalla a continuación:

- Tamaño de ventana mínima: en número de píxeles de lado, es el tamaño mínimo que tendrá la ventana de búsqueda. Por defecto, la función de detección se configura de tal manera que coincide con el tamaño de las muestras con las que el clasificador fue entrenado.

- Factor de escala: es el factor por el que la ventana de búsqueda aumenta de tamaño entre un escaneo y otro. Por ejemplo, un valor de 1.1 significa un aumento del 10% en el siguiente escaneo.

- Tamaño de ventana máximo: en número de píxeles de lado, es el tamaño máximo al que llegará la ventana de búsqueda. Cuando se alcanza el límite, se detiene el escaneo. Por defecto se establece el tamaño de la imagen como tamaño máximo de ventana.

- Número mínimo de vecinos: número mínimo de detecciones (menos 1 pues se habla de vecinos) que hacen positiva la detección. El hecho de que la imagen se escanee a diferentes escalas puede hacer que el mismo objeto sea detectado más de una vez. Este parámetro hace que se puedan descartar como detecciones positivas grupos de objetos que no lleguen a un valor mínimo. Poniéndolo a 0 se puede usar para que el detector muestre todas las detecciones y permitan al desarrollador modificar el método de agrupamiento que la función ofrece por defecto.

- Modo: modo de operación. Según la guía de referencia de *OpenCV* [17], el único modo soportado es el de *Canny Prunning*, con el que el detector descarta regiones de la imagen que contienen demasiados o muy pocos bordes por lo que no pueden contener el objeto buscado. En el caso de la detección de caras, el modo de *Canny Prunning* agiliza ligeramente el proceso.

La configuración de todos estos valores incide directamente en la eficacia del detector. Por ejemplo, los parámetros por defecto (factor de escala 1.1, número mínimo de vecinos 3 y modo 0) están diseñados para una efectividad de detección alta aunque lenta. Para operaciones con videos o imágenes en tiempo real, los mejores parámetros son 1.2 para el factor de escala, 2 para el número mínimo de vecinos y activar el modo de *Canny Prunning*. Configurar el tamaño mínimo de ventana de manera adecuada puede ser otro aspecto que incida en la velocidad de detección ya que, cuanto mayor sea su valor, menos escaneos son necesarios.

3.3 Primeras pruebas

Para comprobar el correcto funcionamiento del clasificador seleccionado (**haarcascade_frontalface_alt2.xml**), se realizaron diferentes pruebas iniciales que se han documentado y se encuentran disponibles en el Anexo G.

En esas pruebas, varias fotos de tamaño aproximado de 640x480 píxeles se procesaron con el propósito de comprobar el acierto del detector, así como su velocidad en la detección. Todas las fotos contenían al menos una cara detectable.

Se intentó abarcar con las pruebas un amplio espectro de posibles detecciones, incluyendo personas con diferentes tonos de piel, personas con la cara parcialmente tapada, personas a diferentes distancias, etc.

Los resultados de las pruebas realizadas para medir la eficacia del detector destacan que:

- El detector tiene una tasa de acierto superior al 95% con una baja tasa de falsos positivos (comparándolos con el número de sub-ventanas analizadas, no con el número de aciertos).

- Existe una limitación en cuanto a la alineación horizontal de las caras. Hay que tener en cuenta que la capacidad del detector depende claramente de la base de datos usada en el entrenamiento. En este caso, las caras deben de estar posicionadas de tal manera que la zona de los ojos, una de las características más diferenciables del rostro humano, se encuentre alineada horizontalmente.

- La detección se realiza rápidamente sin perder eficacia.

Las conclusiones extraídas de las pruebas fueron altamente satisfactorias. Las limitaciones encontradas están directamente relacionadas con el conjunto de muestras usadas en el entrenamiento y, aunque limitaciones fácilmente corregibles en caso necesario, no afectan al rendimiento del sistema final en el contexto de este proyecto.

3.4 Uso del detector para el sistema de control

Una vez obtenidos los resultados de las primeras pruebas que verifican el gran comportamiento del detector de *OpenCV*, había que pensar en la mejor manera de usarlo para cumplir los objetivos marcados en este proyecto, que en relación con la detección de caras se pueden resumir en:

- Detección de todas las caras presentes en una imagen fija (foto) o móvil (video).

- Detección de todas las caras presentes en imágenes en tiempo real (cámara web).

- Almacenamiento de información esencial sobre cada cara detectada para un posible post-procesado.

- Coste mínimo de espacio de almacenamiento.

Estos objetivos se consiguen realizando tres pasos que se detallan a continuación:

1. Obtener imagen donde buscar caras según la fuente:
    a. Foto, directamente, de una en una
    b. Video, fotograma a fotograma hasta el final del video
    c. Cámara web, fotograma a fotograma hasta que se detenga la transmisión

2. Procesar la foto o fotograma por regiones (ventanas de búsqueda) según los parámetros elegidos de:
    a. Tamaño de ventana mínima
    b. Factor de escala
    c. Tamaño de ventana máximo
    d. Número mínimo de vecinos
    e. Modo

3. Para cada cara detectada guardar:
    a. Imagen individual conteniendo la cara con el menor ruido[5] de fondo
    b. Instante de la detección

Este proceso genera un almacén de imágenes (en el contexto de este documento, ese almacén se referencia también como base de datos de caras) con las caras de todas las personas detectadas junto con el instante de su detección.

Como resultado de este simple proceso, en el sentido estricto de la información recopilada, que son las caras junto con el instante temporal, se consigue tener disponible para posible post-procesado la misma información básica que puede ofrecer un sistema que almacena video, pero minimizando drásticamente el coste de almacenamiento, como se muestra a continuación.

3.5 Comparativa almacenamiento

Para la realización de la tabla que se muestra al final de esta sección donde se compara el espacio en disco necesario en un sistema de vigilancia a través de video y el necesario para el sistema desarrollado en este proyecto, se han establecido los siguientes datos:

- Espacio requerido para 48 horas de video: 25GB. Esto equivale a 520MB cada hora, 8680KB por minuto. Dato basado en el sistema de vigilancia con *Vitamin D 4.1*[6], versión *Starter* (gratis).

- Tamaño medio de una cara (100x100 píxeles) almacenada con la aplicación de este proyecto: 7KB.

Por otro lado, se han estimado los siguientes escenarios para comparar ambos sistemas en la misma situación de vigilancia y control de personas:

- Peor caso para el detector de caras. Mucha afluencia de gente. 100 personas por minuto. Ejemplo: entrada a concierto o evento deportivo.

- Caso medio. Afluencia normal de gente. 20 personas por minuto. Ejemplo: calle.

- Mejor caso. Poca afluencia de gente. 5 personas por minuto. Ejemplo: entrada a gimnasio.

La comparativa acerca del espacio requerido en cada escenario para 1 minuto de vigilancia se presenta a continuación en la Tabla 2:

Sistema / Escenario	Concierto	Calle	Gimnasio
Video vigilancia	8680KB	8680KB	8680KB
Detector de caras	700KB	140KB	35KB

Tabla 2: Espacio en disco necesario para 1 minuto de vigilancia con los sistemas de video vigilancia y de detección de caras.

[5] Ruido en este contexto se entiende como información no útil que puede distorsionar un resultado.
[6] http://www.vitamindinc.com

Se puede observar que en el peor de los casos para el detector de caras, este necesita unas 12 veces menos de tamaño que el sistema de vigilancia con vídeo. En el mejor de los casos para el detector, la diferencia se establece en 248 veces menos de espacio necesario.

Gráficamente, la diferencia es aun más apreciable:

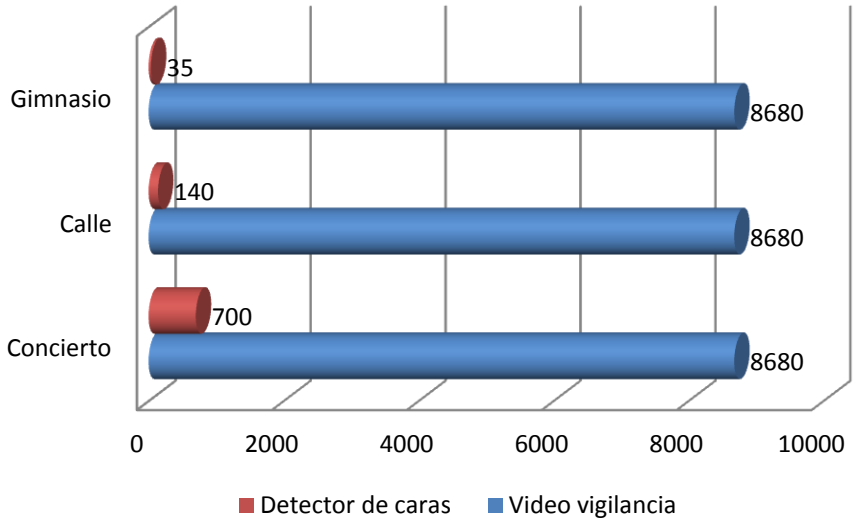


Figura 11: Gráfica comparativa del espacio en disco necesario para 1 minuto de vigilancia con ambos sistemas. Medido en KB.

Debido a la linealidad de los datos, la diferencia se mantiene proporcional con otros valores temporales, como por ejemplo, con 1 hora de vigilancia:

Sistema / Escenario	Concierto	Calle	Gimnasio
Video vigilancia	520 MB	520 MB	520 MB
Detector de caras	42 MB	8,4 MB	2,1 MB

Tabla 3: Espacio en disco necesario para 1 hora de vigilancia con los sistemas de video vigilancia y de detección de caras.

Los datos aquí mostrados satisfacen plenamente el objetivo establecido en cuanto a ahorro de espacio que requería esta aplicación.

Capítulo 4. Reconocimiento de caras (*face recognition*)

El reconocimiento de caras puede considerarse también un área muy activa en el campo de visión por computador. Es un proceso que se lleva estudiando desde hace más de 20 años, produciendo diferentes aplicaciones para sistemas de seguridad, robótica, interfaz hombre-máquina, cámaras digitales, juegos y un largo etcétera.

El reconocimiento de caras requiere de dos fases bien diferenciadas:

- Detección de caras: obtiene cualquier cara de una imagen para facilitar su posterior procesado.

- Reconocimiento de cara: la cara detectada y procesada se compara con una base de datos de caras conocidas, buscando encontrar la coincidencia y decidir quién es esa persona.

Aunque la detección de caras se realiza hoy en día con un alto nivel de efectividad (el método comentado anteriormente en este proyecto roza el 94% de tasa de aciertos), el reconocimiento rápido y simple sigue algo lejos de esa precisión.

Para la tarea de reconocimiento, en este proyecto se hace uso del método de *Eigenfaces* [2] que está basado en el Análisis de Componentes Principales (ACP) [18]. Aunque existen otras técnicas como el Análisis de Discriminantes Lineales (ADL) [31], se decidió que el método de *Eigenfaces* encajaba perfectamente en el ámbito de este proyecto debido a los resultados aceptables que permite obtener de una manera simple.

4.1 Método *Eigenfaces*

El método *Eigenfaces*, al igual que otros métodos de reconocimiento o de detección, requiere de una fase previa de entrenamiento. En esta fase, el sistema se entrena con imágenes de las personas que se desean reconocer. Estas imágenes se denominan imágenes de entrenamiento (*training images*). En la fase de reconocimiento, ante la pregunta de "quien es esta persona" cuando se le muestra al sistema una cara a comprobar, *Eigenfaces* proporciona como respuesta la persona más parecida de entre todas con las que fue entrenado.

Eigenfaces usa las imágenes de entrenamiento para aprender el modelo de cara. Para la obtención de este modelo se aplica la técnica llamada Análisis de Componentes Principales (ACP). El ACP es una técnica utilizada para reducir la dimensión de un conjunto de datos. Simplificando al máximo su aplicación en el caso concreto de las caras, con el ACP se obtiene un sub-espacio de patrones con las variaciones más significativas entre las imágenes faciales conocidas, reduciendo la dimensión del conjunto de datos de los N píxeles que forman cada imagen (10000 en una imagen de 100x100 píxeles) a ese número de patrones. Esos patrones significativos o componentes principales del conjunto de caras de entrenamiento se llaman en este contexto *eigenfaces*[7] y dan nombre al método de reconocimiento en su conjunto.

Una vez obtenido el sub-espacio de patrones o *eigenfaces*, el último paso de la fase de entrenamiento consiste en proyectar las caras de entrenamiento sobre él (proyectar sobre un sub-espacio significa encontrar el punto más cercano al objeto de entrada en ese sub-espacio) para

[7] Se denominan *eigenfaces* ya que el ACP usado para el reconocimiento de caras se basa en la descomposición en vectores propios (*eigenvectors* en inglés) de la matriz de covarianza.

obtener una representación de cada una de las caras en ese mismo sub-espacio de reducida dimensión.

En la fase de reconocimiento, el método *Eigenfaces* de nuevo ejecuta la misma tarea de proyección sobre el sub-espacio obtenido en la fase de entrenamiento, pero esta vez de la cara a comprobar, reduciendo de nuevo la dimensión de la imagen de entrada a la deseada. Una vez proyectada, el resultado de la proyección se compara con cada una de las proyecciones de las caras de entrenamiento para obtener la cara más cercana en el sub-espacio. Esa cara pertenece a la persona que *Eigenfaces* ha reconocido como la más parecida.

En el Anexo H se explican características del método de *Eigenfaces* con más detalle para referencia del lector.

4.2 Análisis del estado del arte del método *Eigenfaces*

Al igual que en la fase de detección de caras, se realizó un estudio del arte del método *Eigenfaces* para confirmar lo adecuado de su utilización para la fase de reconocimiento del proyecto.

Cuando se comenzó este estudio y pese a la popularidad del método *Eigenfaces*, no existían muchos módulos o programas sencillos para probar su capacidad. Sin embargo, tanto la documentación de *OpenCV* [19] como una de las páginas web más relevantes en el campo como *Face Recognition Homepage* [20] citaban el trabajo de un desarrollador llamado Shervin Emami [21]. Este desarrollador ofrece un programa llamado **OnlineFaceRec.exe** que es el que se usó para este análisis[8]. Todas las pruebas realizadas con esta aplicación aparecen documentadas en el Anexo I.

De las pruebas realizadas se concluyen dos aspectos importantes:

- El reconocedor de caras con *Eigenfaces* ofrece, como se preveía, resultados bastante aceptables en poco tiempo y es un buen candidato para realizar las tareas de reconocimiento en el proyecto.

- El valor de confianza que genera la herramienta utilizada en las pruebas (**OnlineFaceRec.exe**) no es fiable cuando el número de caras de entrenamiento es elevado, ya que existen errores cuyo reconocimiento ofrece un valor de confianza que excede el 97% (consultar el Anexo I para más detalle).

La primera conclusión confirma la buena elección del uso de *Eigenfaces* en el contexto de este proyecto. La segunda de las conclusiones debe motivar un estudio de alternativas para mostrar la confianza de un reconocimiento en caso de que sea necesario.

En las siguientes secciones se explica a grandes rasgos la implementación de la parte del reconocedor para la aplicación desarrollada en este proyecto.

[8] Con el lanzamiento de la versión 2.4 de *OpenCV* en Mayo de 2012, *OpenCV* ofrece ahora una clase específica para reconocimiento de caras con programas de prueba y ejemplos, similar a lo que ofrece para detección de caras. Este nuevo módulo es además el que referencia la página *Face Recognition Homepage* reemplazando al trabajo de Shervin Emami.

4.3 Entrenamiento

La fase de entrenamiento de esta herramienta difiere de las demás consultadas en cuanto a la metodología del entrenamiento se refiere, no en cuanto a la técnica.

La diferencia fundamental es que en los trabajos consultados la información para el entrenamiento se suministra de manera totalmente manual generando un fichero de texto al uso, donde se deben escribir a mano las referencias de las localizaciones de cada una de las imágenes de entrenamiento. En el caso de la herramienta de este proyecto, aunque en esencia la base del entrenamiento es la misma, los datos propios de dicho entrenamiento se los proporciona la misma herramienta con las detecciones llevadas a cabo. Es, por decirlo así, un sistema de entrenamiento autoalimentado.

Por otro lado, la herramienta sí que necesita información adicional en cuanto al reconocimiento se refiere. Está claro que la herramienta no es capaz de saber el nombre de la persona que aparece en la foto N sin más información que la recogida de la cámara web, el video o la foto. Pero sí que debería ser capaz de decir que la persona de la foto M se parece a la de la foto N con cierto nivel de precisión. Si en un momento dado el sistema es alimentado con información adicional suficiente, por ejemplo la persona de la foto N se llama Emma, entonces debería ser capaz de decir que la persona de la foto M se parece a Emma, con ese mencionado nivel de precisión.

Esta herramienta ofrece la posibilidad de modificar los nombres de las personas detectadas tanto en la fase de entrenamiento como posteriormente.

4.3.1 Entrenamiento con fotos almacenadas (*offline*)

La forma más sencilla de entrenar al sistema es tomando como imágenes de entrenamiento las que la aplicación tiene almacenadas en su base de datos de caras. Este tipo de entrenamiento se ha denominado *offline*.

El proceso que se sigue es el siguiente:

- Si ya existe un entrenador, se ofrece la opción de añadir las imágenes de la base de datos de caras a ese entrenador o crear un entrenador nuevo.

- Las imágenes de entrenamiento se pre-procesan (escala de grises, conversión de tamaño si lo precisa y ecualización) y se copian en una carpeta temporal de entrenamiento.

- A partir de las fotos procesadas se crea un fichero de texto con la información esencial de cada imagen, como puede ser el nombre de la persona y la ruta a la imagen. La identificación con nombre debe ser hecha a mano con el interfaz mencionado anteriormente. Si no se hace, se autogenera un nombre por cada una de las fotos.

El fichero de texto generado, que es en esencia el fichero del entrenamiento, es la base del reconocedor.

4.3.2 Entrenamiento a través de la cámara web (*online*)

La otra manera de entrenar al sistema es mediante imágenes obtenidas de la cámara web en tiempo real. En contraposición al anterior método, este se ha denominado entrenamiento *online*.

En el entrenamiento *online* todas las personas se incluyen en el actual reconocedor. No se pregunta si se quiere reiniciar el entrenamiento porque la mayoría de las veces no se querrá reiniciar pero aun así, si se desea, la herramienta ofrece la opción de reiniciar el reconocedor en el momento en que sea preciso.

Este método de entrenamiento *online* sigue el proceso descrito a continuación:

- Se recoge el nombre de la persona a detectar.
- Se inicializa la cámara.
- Se selecciona uno de cada *n* fotogramas para analizar, de acuerdo a la frecuencia de escaneo establecida.
- Cada fotograma seleccionado se escanea en busca de caras.
- Cada cara detectada se recorta y se almacena de la siguiente manera:
    - Original en la base de datos
    - Imagen procesada en base de datos de entrenamiento
- Se añade una línea por cada cara en el fichero de texto del entrenamiento.

Cabe mencionar que el fichero de texto base del reconocedor es compartido por ambos métodos de entrenamiento, lo que dota de gran flexibilidad a este proceso en su conjunto.

4.3.3 Generación del reconocedor

La parte del entrenamiento directamente relacionada con la obtención del reconocedor siguiendo el método de *Eigenfaces* se separa de los dos métodos de entrenamiento para poder ser utilizada tanto con el entrenamiento *offline* como con el *online*.

El proceso para la generación del reconocedor sigue los pasos siguientes:

- Se cargan las imágenes referenciadas en el fichero de texto del entrenamiento.
- Se generan los diferentes almacenes de datos, como por ejemplo el concerniente a los nombres de persona, así como otras estructuras necesarias para los cálculos relacionados con el método de ACP.
- Se hace uso de la función de *OpenCV* **cvCalcEigenObjects()** para calcular el sub-espacio de caras propias (*eigenfaces*).
- Cada imagen de entrenamiento se proyecta mediante la función **cvEigenDecomposite()** sobre el sub-espacio obtenido anteriormente.

Todos los datos generados (número y dimensión de las imágenes de entrenamiento, número de personas, nombre de las personas, valores de las *eigenfaces*, etc.) se almacenan en un fichero XML de nombre **facedata.xml**. Este archivo es el reconocedor final obtenido cuando concluye la fase de entrenamiento.

4.4 Uso del reconocedor para el sistema de control

El reconocedor obtenido se usa para realizar comprobaciones automáticas acerca de si una persona ha sido previamente detectada por el sistema o no. Sin lugar a dudas y basando la afirmación en el estudio previo realizado, es la funcionalidad que hace a este sistema destacar por encima de otras aplicaciones de vigilancia y control consultadas, ya que en ningún caso ofrecen posibilidad de reconocimiento facial de personas a coste cero. Si se requiere reconocimiento, el desembolso económico resulta elevado.

Similarmente al entrenamiento, existen dos métodos de comprobación o reconocimiento de personas: *offline* y *online*.

El reconocimiento *online* es aquel realizado automáticamente por el sistema a través de la cámara web. Cada cara que se detecta en la imagen de la cámara web se evalúa con el reconocedor previamente obtenido para comprobar si es una persona conocida o no. El resultado se muestra en el cuadro de mensajes de la ventana principal de la aplicación.

En el método *offline*, las imágenes a comprobar se le suministran al sistema a través de un simple interfaz para cargar imágenes. El reconocimiento *offline* se presenta a su vez en dos modalidades: para una persona o para varias. El de una persona muestra el resultado de la evaluación con una ventana informativa, mientras que en el de múltiples personas, al sistema se le indica un almacén de imágenes a analizar y este, tras las evaluaciones, guarda resultados en un fichero de formato CSV pero no los muestra al usuario. Es lo que se ha denominado modo *debug*.

4.4.1 Proceso del reconocimiento *offline*

El reconocimiento *offline* sigue el proceso descrito a continuación:

- Por cada foto a reconocer se realiza el mismo pre-procesado que con las imágenes de entrenamiento.
- Se carga toda la información necesaria desde el archivo del reconocedor **facedata.xml**.
- Cada imagen a analizar se proyecta sobre el sub-espacio ACP obtenido en la fase de entrenamiento.
- Se busca la imagen de entrenamiento cuya proyección en ese sub-espacio sea la más cercana.
- La distancia entre ambas proyecciones se usa para determinar un valor de confianza/aproximación.

Todos los resultados se almacenan para análisis. Si la comprobación es de una persona, se informa además al usuario del nivel de confianza y se le muestra la cara comprobada y la más cercana encontrada.

4.4.2 Proceso del reconocimiento *online*

El reconocimiento *online* sigue el mismo proceso descrito para el método *offline* pero esta vez las caras a comprobar son suministradas automáticamente por el sistema con cada detección

a través de la cámara web. Los resultados, además de ser almacenados para análisis, se muestran al usuario a través del cuadro de mensajes de la ventana principal de la aplicación.

4.5 Resultados, pruebas y cambios

Debido a la relevancia de la funcionalidad de reconocimiento de la aplicación desarrollada, el autor de este proyecto realizó exhaustivas pruebas sobre el rendimiento del mismo, el por qué de los fallos, como facilitar al usuario el entrenamiento, y sobre todo como mejorar los resultados buscando diferentes alternativas que pudiesen minimizar las limitaciones que presenta el método *Eigenfaces*.

Por no extender en demasía la parte principal de la memoria, estas pruebas, resultados, cambios y conclusiones se detallan en el Anexo J.

Las conclusiones más relevantes de las pruebas fueron:

- Es recomendable hacer una correcta selección de imágenes de entrenamiento eliminando aquellas que pudiesen confundir al sistema (por ejemplo los falsos positivos) o que ofrezcan poca variación con respecto a otras.

- A mayor número de caras de entrenamiento, mayor tiempo de generación del reconocedor y mayor el tiempo en comprobar una persona.

- Para el entrenamiento *online*, almacenar una imagen por segundo es suficiente.

En resumen, el sistema desarrolla un nivel de acierto por encima del 85% en casi cualquier caso, llegando al 90% con relativa facilidad únicamente con 6 muestras por sujeto para el entrenamiento. En las tareas más costosas en tiempo, no tarda más de 2 segundos en generar un reconocedor a partir de 42 muestras.

4.6 Inclusión de un nivel de confianza

El método seguido para el reconocimiento ofrece como resultado la cara de la persona más parecida a la comprobada de entre todas con las que se entrenó al sistema. Si bien es un resultado muy aceptable en el contexto de este proyecto (control de personas), en la fase final de la implementación se decidió añadir al resultado de los reconocimientos un valor que permitiera saber con qué confianza el sistema es capaz de reconocer a una persona. No es lo mismo decir que una persona conocida es la más cercana a la probada que decir lo mismo pero con un nivel de confianza del 90%. De esta manera se lograba dotar al sistema de mejores prestaciones en las tareas de reconocimiento.

Como se ha mencionado anteriormente, la aplicación usada para el análisis del estado del arte de *Eigenfaces* (**OnlineFaceRec.exe**) ofrecía un nivel de confianza pero de ninguna manera aceptable en la mayoría de los casos, ya que personas que no están en la base de datos aparecían reconocidas con un 97% de confianza en algunos casos. Por lo tanto el método de cálculo usado en esa aplicación fue desechado.

Los datos que se tienen para establecer un valor de confianza del reconocimiento son básicamente los que ofrece la distancia entre la cara de entrenamiento y la cara a reconocer.

Para determinar esa separación entre la cara de test y cada una de las de entrenamiento en el sub-espacio de *eigenvectors* obtenido, inicialmente se calculaba una simulación de la distancia Euclídea (sin aplicar la raíz cuadrada). En concreto, para cada cara de entrenamiento se realizaba el siguiente cálculo:

$$Distancia = \sum_{i=1}^{n} (\text{ProyeccionTest}_{Ei} - \text{Proyeccion}Train_{Ei})^2$$

Donde:

- n es el número de vectores propios en el sub-espacio ACP calculado previamente.
- $\text{ProyeccionTest}_{Ei}$ es el valor de la proyección de la cara a comprobar en el *eigenvector* E_i.
- $\text{Proyeccion}Train_{Ei}$ es el valor de la proyección de la cara de entrenamiento en el *eigenvector* E_i.

El problema que presenta este cálculo es que al no haber normalización, el dato final de distancia cuando el número de caras de entrenamiento es elevado se convierte en un valor poco manejable y difícil que convertir en algún tipo de nivel de confianza entre 0 y 1.

Para normalizar el valor de la distancia, se probó primeramente con una simulación de la prueba de Pearson [23] que establece que se puede comparar la discrepancia entre una distribución observada y otra teórica usando:

$$Distancia = \sum_{i=1}^{n} \frac{(\text{ProyeccionTest}_{Ei} - \text{Proyeccion}Train_{Ei})^2}{\text{Proyeccion}Train_{Ei}}$$

Donde n, $\text{ProyeccionTest}_{Ei}$ y $\text{Proyeccion}Train_{Ei}$ hacen mención a lo mismo que en la anterior fórmula.

Aplicando esta manera de obtener la distancia entre las proyecciones teórica (*train*) y observada (test), resultan valores más manejables pero de nuevo no se encuentra la manera de convertir esos valores a límites de confianza.

Finalmente, si se calcula la varianza para cada uno de los vectores propios (*eigenvectors*), la distancia obtenida en cada uno de ellos se puede normalizar a esa varianza, de forma que el dato obtenido se distribuye como una Chi-cuadrado con grados de libertad igual al número de *eigenvectors*.

El nuevo cálculo se realiza de la siguiente manera:

$$Distancia = \sum_{i=1}^{n} \frac{(\text{ProyeccionTest}_{Ei} - \text{Proyeccion}Train_{Ei})^2}{\text{Varianza}_{Ei}}$$

Donde n, $\text{ProyeccionTest}_{Ei}$ y $\text{Proyeccion}Train_{Ei}$ hacen mención a lo mismo que en las anteriores fórmula y Varianza_{Ei} indica la varianza del vector propio o *eigenvector* E_i.

Este cálculo de separación de caras, implementado en la herramienta final, permite obtener un valor de confianza para cada reconocimiento realizado.

Sin embargo, debido a que los grados de libertad de la tabla Chi-cuadrado están relacionados con el número de caras de entrenamiento y este número de caras es decidido en cada momento por el usuario, solamente se ha incluido el cálculo de un nivel de confianza hasta un grado de libertad determinado, ya que es preciso introducir manualmente los datos de una tabla Chi-cuadrado en el código de la aplicación.

El resultado final que ofrece la herramienta incluye el nivel de confianza del reconocimiento si el número de caras de entrenamiento usadas no supera cierto límite, pero en cualquier caso siempre ofrece la cara más cercana según el último cálculo realizado.

Capítulo 5. Sistema final de vigilancia y control

La herramienta final desarrollada en este proyecto es un sistema de vigilancia y control que ofrece un sencillo interfaz (ver Figura 12) para permitir al usuario su utilización.

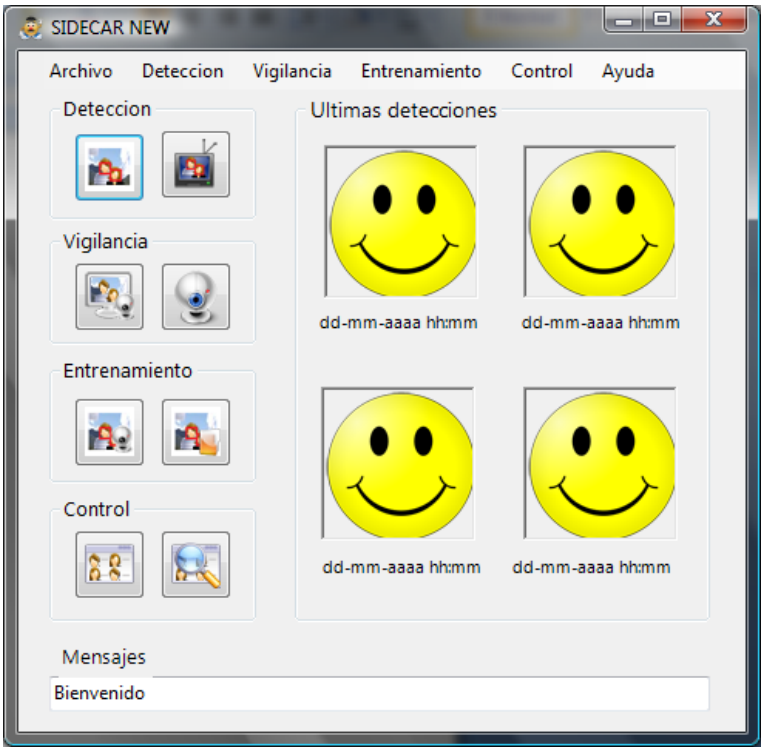


Figura 12. Ventana principal de la aplicación de vigilancia y control final desarrollada.

La base de este sistema es la detección rápida de rostros sobre todo tipo de imágenes (foto, video, cámara web) y sus principales características son el mínimo espacio de almacenamiento que requiere para almacenar las detecciones así como la posibilidad de realizar reconocimiento de personas.

A lo largo de este capítulo se va a describir de manera concisa todo lo que ofrece el sistema, dejando los detalles e información adicional de cada funcionalidad para el Anexo M relativo al manual de usuario.

5.1 Opciones de configuración

Una de las características principales de esta aplicación es la flexibilidad que ofrece para permitir al usuario configurar varios aspectos de la misma.

Las opciones de configuración se presentan en una ventana adicional, separadas en dos pestañas: opciones generales y opciones del detector.

En la primera pestaña se presentan las opciones más generales de la aplicación, que son:

- Cambiar base de datos de caras: se permite al usuario establecer el directorio donde almacenar las caras detectadas. Esta ruta es la que la aplicación utilizará para obtener las caras con las que realizar diferentes acciones, como por ejemplo el entrenamiento *offline* y la muestra de las detecciones.

- Cambiar directorio para entrenamiento: debido a que las imágenes a usar en el entrenamiento requieren un procesado (convertir a blanco y negro, ecualizar, etc.), existe un almacén adicional de caras procesadas. La ruta a este almacén se establece a través de esta opción.

- Modificar frecuencia de detección sobre cámara web: este parámetro determina cada cuantos fotogramas se escanea la imagen de la cámara web en busca de caras. Cada segundo se procesan aproximadamente 30 fotogramas, por lo tanto, un valor de 30 establece que se analice un fotograma cada segundo. Si el valor de la frecuencia es 0, la aplicación escanea todos los fotogramas. Este valor debe ajustarse para obtener un equilibrio acertado entre la capacidad de la máquina donde se ejecuta la aplicación y lo exhaustiva que se requiera la detección.

- Modificar frecuencia de detección sobre video: igual que en el caso de la cámara web, esta frecuencia determina cada cuantos fotogramas se escanea la imagen de video en busca de caras.

- Factor de recorte: usado durante la fase de pruebas del reconocedor, este parámetro indica el número de píxeles a recortar de cada imagen de entrenamiento para intentar eliminar el posible ruido en forma de fondo, pelo, orejas, etc. Solo se permite su uso en tareas de depuración ya que requiere habilitar trozos de código fuente.

La segunda pestaña, con las opciones del detector, permite la modificación de todos los parámetros configurables del detector mencionados en el capítulo 3, que son:

- Factor de escala.

- Numero de vecinos.

- Modo.

- Tamaño de ventana mínima de escaneo (en píxeles).

- Tamaño de ventana máxima de escaneo (en píxeles).

Es recomendable que el usuario de la aplicación realice pruebas con distintos valores de estos parámetros del detector para conseguir un detector que se ajuste de la mejor manera al entorno donde se utilice y poder aprovechar al máximo el potencial del sistema.

Una vez explicado todo lo que se puede modificar a través de la ventana de opciones de la aplicación, se presentan en los siguientes apartados todas las demás funcionalidades del sistema, agrupándolas en tres grupos: detección, reconocimiento y otras funcionalidades.

5.2 **Funcionalidades de detección**

Las funcionalidades que ofrece la aplicación relacionadas con la detección de objetos se separan en detección sobre foto, sobre video y sobre imagen de cámara web.

Para la detección de caras en imagen fija (fotos) esta aplicación ofrece:

- Cargar foto: necesario para indicar al sistema sobre que foto se desea realizar la detección. Deben ser archivos JPG. A pesar de que *OpenCV* soporta múltiples formatos de imagen (BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF y TIF), se decidió establecer JPG como formato principal debido a su alto nivel de compresión y a su extendido uso.

- Mostrar foto cargada: muestra la foto seleccionada en una ventana aparte.

- Detectar en foto: ejecuta la detección de rostros sobre la foto seleccionada. Se muestra el resultado de la detección en un mensaje informativo junto con la imagen original y la imagen resultado.

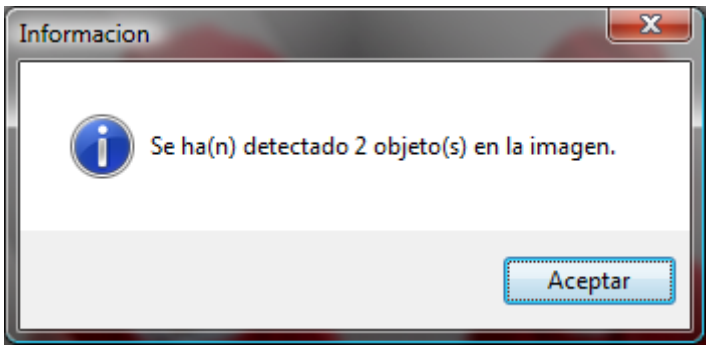


Figura 13. Mensaje informativo de una detección sobre imagen fija (foto).El resultado muestra 2 objetos detectados.

Figura 14. Imagen original e imagen resultado de una detección sobre imagen fija (foto)

Disponible en la ventana principal a través del botón  .

- Detectar en foto rotada varias veces: debido a la limitación del detector relacionada con la alineación horizontal de las caras, se incluyó esta funcionalidad para minimizar su impacto en la detección en fotos.

- Detectar en foto almacenando resultados sobre la acción (modo *debug*): útil para la fase de pruebas durante el desarrollo de la aplicación. Se incluye en el sistema final para ser usada en la depuración de la configuración de los parámetros del detector. La información, que incluye la foto analizada, los parámetros del detector utilizado y el resultado, se almacena en el archivo **resultados_deteccion.csv**.

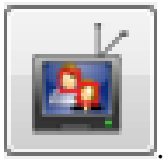
Para la detección de caras en video, las funcionalidades son:

- Cargar video: necesario para indicar al sistema sobre que archivo de video se desea realizar la detección. Los archivos deben ser del tipo AVI, MP4 o FLV.

- Mostrar video: reproduce el video seleccionado e informa al usuario sobre algunas de sus características (velocidad y resolución) en una ventana informativa.

- Detectar en video: ejecuta la detección de rostros sobre el video seleccionado.

Figura 15. Instante de una detección de cara sobre imagen de video. El objeto detectado se recuadra.

Cada cara detectada es automáticamente almacenada en la base de datos de caras. Esta funcionalidad se encuentra disponible en la ventana principal de la aplicación a través del botón  .

Para la detección de caras en imagen de cámara web, las funcionalidades son:

- Mostrar imagen de cámara web: permite al usuario comprobar el correcto funcionamiento de la cámara web a usar.

- Detectar sobre imagen de cámara web: modo vigilancia. Este es el modo principal de operación de la aplicación. Se inicializa la cámara web cuya imagen se muestra en una ventana aparte y cada cara detectada es almacenada automáticamente en la base de datos de caras junto con el instante temporal de su detección.

Se encuentra disponible a través del botón  .

Como referencia para el lector, el Anexo K lista las compatibilidades de este modo de funcionamiento con respecto al resto de funcionalidades de la aplicación.

La última de las funcionalidades relacionadas con la detección es la posibilidad de cargar diferentes tipos de detectores. Los detectores son ficheros XML que definen que objeto es el que se va a buscar en la imagen. Algunos de estos ficheros, disponibles tanto en el paquete de instalación de *OpenCV* como en la red, ofrecen la posibilidad de detectar diferentes partes del cuerpo como:

- Cara Frontal → *seleccionado por defecto*
- Cuerpo
- Perfil
- Nariz
- Boca
- Ojos
- Otros

Si se selecciona la opción Otros, se permite cargar cualquier otro detector, siempre archivos de tipo XML.

5.3 Funcionalidades de reconocimiento

Las siguientes funcionalidades están relacionadas con el reconocimiento:

- Entrenamiento *offline*: se obtienen los datos para el reconocedor a partir de las fotos almacenadas en la base de datos de caras.

  Se encuentra disponible a través del botón  .

- Entrenamiento *online*: se obtienen los datos para el reconocedor a partir de las caras que se vayan reconociendo a través de la imagen de la cámara web. Esta opción permite a su vez establecer un nombre de la persona con la que se va a entrenar al sistema.

  Se encuentra disponible a través del botón  .

- Generación del reconocedor: el sistema, a partir de los datos recogidos durante el entrenamiento (*offline*, *online* o una mezcla de ambos), genera el fichero XML del reconocedor que será usado a partir de ese momento por la aplicación.

- Reinicio de reconocedor: si por cualquier circunstancia es preciso crear un nuevo reconocedor desde cero, esta opción permite eliminar de manera automática los datos del anterior reconocedor.

- Comprobar persona: se pide al usuario que seleccione una imagen de una cara y se ejecuta la tarea de reconocimiento sobre esa cara haciendo uso del reconocedor previamente generado. Es una de las principales características de esta aplicación ya

que permite conocer con resultados bastante notables si una persona determinada fue detectada en algún momento por esta aplicación.

Se encuentra disponible a través del botón  .

El resultado de la ejecución, mensaje informativo previo e imágenes de prueba y más cercana, se muestran como indican las Figuras 16 y 17.

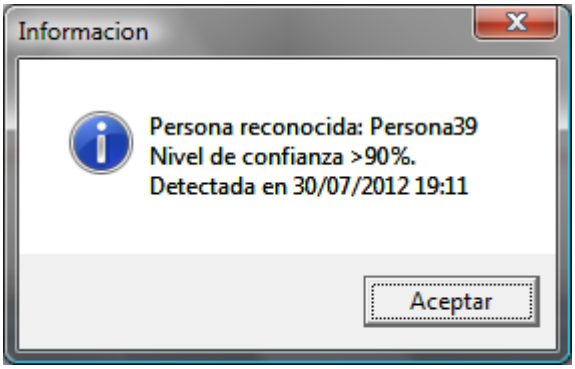


Figura 16. Mensaje informativo acerca del reconocimiento de la persona. Se muestra el nombre de la persona reconocida, el nivel de confianza y la fecha de la detección.

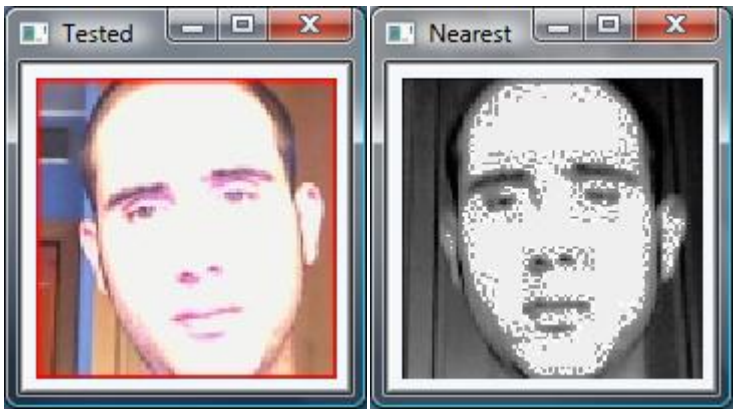


Figura 17. Resultado de un reconocimiento. A la izquierda, la cara a comprobar. A la derecha, la cara reconocida como la más parecida entre todas las usadas en el entrenamiento.

Como se puede apreciar en la figura anterior, la imagen más cercana (*Nearest*) se muestra ya pre-procesada, lo que, en caso de error, puede indicar la razón del fallo (por ejemplo algunas imágenes con mucha intensidad de luz o zona de sombras pierden información al ser ecualizadas).

- Reconocimiento *online* con cámara web: se ejecuta la tarea de reconocimiento sobre cada una de las caras detectadas en la imagen de la cámara web. El resultado de la detección se muestra en el cuadro de mensajes de la ventana principal, como muestra la Figura 18.

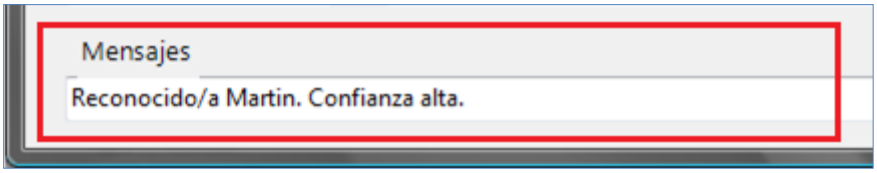


Figura 18. El cuadro de mensajes de la ventana principal muestra en tiempo real el resultado de los reconocimientos.

Se encuentra disponible a través del botón  .

- Comprobar múltiples personas (modo *debug*): se ejecuta la tarea de reconocimiento a todas las fotos presentes en la capeta seleccionada por el usuario y almacena resultados en el archivo de formato CSV llamado **resultados_reconocimiento.csv**. Los resultados almacenados son la ruta de la carpeta, el nombre de cada imagen comprobada, el número de persona reconocida, su nombre y el nivel de confianza de la detección.

- Modificar nombres: permite al usuario establecer los nombres de las personas presentes en la base de datos de la aplicación para el entrenamiento. En la fase de entrenamiento, si el usuario no introduce como información el nombre de cada persona, el sistema le otorga a cada cara un nombre de persona incremental (Persona1, Persona2,… PersonaN). La opción de modificar nombres permite establecer estos nombres después de concluida esa fase de entrenamiento. El interfaz que se presenta al usuario se muestra en la Figura 19.

Figura 19. Interfaz de la opción de modificación de nombres para entrenamiento. Compatible con cualquier método de entrenamiento.

Debido a lo tedioso que puede resultar introducir nombres si el número de personas en la base de datos es muy elevado, el sistema permite saltar fotos y finalizar con la modificación en cualquier instante.

Cuando la aplicación es provista de información acerca de los nombres de las personas de entrenamiento, los resultados de los reconocimientos muestran el nombre de la persona reconocida (Figura 20).

Figura 20. Interfaz de la opción de modificación de nombres para entrenamiento. En este ejemplo, el nombre de la persona previamente introducido en el sistema es mostrado.

5.4 Otras funcionalidades

El resto de funcionalidades que presenta la aplicación, muchas de las cuales permiten la realización de diferentes tareas de control, se muestran a continuación:

- Mostrar últimas detecciones: muestra las últimas ocho detecciones realizadas por la aplicación.

  Se encuentra disponible a través del botón  .

- Mostrar todas detecciones: una de las principales funciones de control de personas es saber que personas han sido detectadas y cuando. Con esta funcionalidad, el usuario es capaz de ver con un interfaz muy sencillo y manejable todas las detecciones almacenadas por el sistema en la base de datos de caras de la aplicación. La Figura 21 ilustra el interfaz de la muestra de todas detecciones.

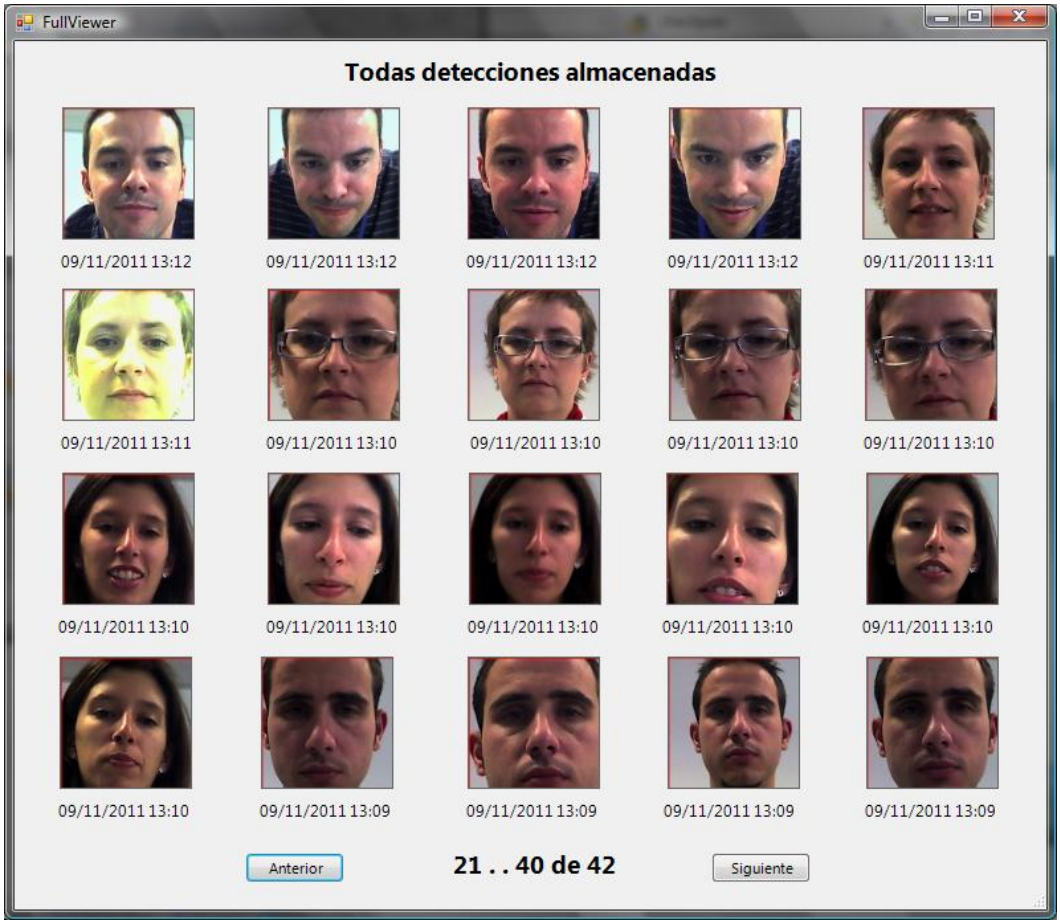


Figura 21. Detecciones 21 a 40 de las 42 presentes en la base de datos de caras en el momento de la ejecución de la opción para mostrar todas detecciones.

- Conversión de imagen a JPG: como se ha mencionado anteriormente, el formato de archivos de imagen establecido para esta aplicación es JPG por varias razones. Sin embargo, para dotar de mayor flexibilidad al sistema, se añadieron funcionalidades para la conversión de varios tipos de formatos a JPG. Cada conversión crea una copia en JPG de la imagen de origen seleccionada por el usuario y la almacena en la base de datos de caras para que pueda ser usada por la aplicación.

- Conversión de múltiples imágenes a JPG: en este caso, al sistema se le indica una carpeta de imágenes y todas las imágenes de formatos soportados son convertidas a JPG y almacenadas en la base de datos de caras.

- Acceso a ficheros de registro de eventos (TXT) de:

    - Detecciones: el sistema registra todas las detecciones realizadas en el archivo **DetectionLog.txt** para revisión en caso necesario. A través de esta opción, se accede directamente al contenido de ese registro.

    - Reconocimientos: del mismo modo que con las detecciones, el sistema registra todos los reconocimientos llevados a cabo en el archivo **RecognitionLog.txt**. Este registro se accede desde la herramienta a través de esta opción.

- Acceso a ficheros de resultado (CSV) de:
    - Detecciones: acceso directo al archivo de resultados de detecciones **resultados_deteccion.csv**.
    - Reconocimientos: acceso directo al archivo de resultados de reconocimientos **resultados_reconocimiento.csv**.
- Edición de detecciones: esta opción permite un acceso directo a la base de datos de caras para la edición de las imágenes a usar en el entrenamiento si se precisa. Como se comenta en el capítulo anterior, es altamente recomendable revisar las imágenes que se van a usar para el entrenamiento, pues cuanto mejor sea el entrenamiento, más preciso y robusto se va a generar el reconocedor. Por lo tanto, imágenes que contengan falsos positivos u otras que añadan poca información al entrenamiento pueden ser fácilmente eliminadas de la base de datos.

Por último, la opción de ayuda permite acceder directamente al manual de usuario de la aplicación.

5.5 Estadísticas de la implementación

El sistema final desarrollado sigue la estructura de ficheros establecida por el asistente de aplicaciones de tipo formulario que ofrece *Microsoft Visual Studio*.

Se han creado 6 archivos principales, de extensión .h, que contienen la implementación de una clase formulario cada uno, que son:

- **Form.h**: formulario principal de la aplicación: 4173 líneas.
- **Propiedades Generales.h**: formulario de opciones: 885 líneas.
- **Full Vierwer.h**: formulario para mostrar todas detecciones: 888 líneas.
- **Image Viewer.h**: formulario para mostrar las últimas detecciones: 350 líneas.
- **Recoger Nombre.h**: formulario para recoger nombres de personas: 125 líneas.
- **Modificar Nombres.h**: formulario para modificar los nombres de las personas usadas para entrenamiento: 245 líneas.

En total, estos archivos principales de la aplicación en su conjunto constituyen cerca de 6700 líneas de código, incluyendo, no solo la implementación de tareas sino también la codificación relativa a todos los componentes presentes en los formularios.

La planificación realizada previa al desarrollo del proyecto aparece detallada en el Anexo D, que incluye un diagrama de Gantt de dicha planificación. De esa estimación y el posterior trabajo se concluye que las horas invertidas en la realización de este proyecto es aproximadamente de 470 horas más restrasos.

5.6 Estudio sobre etiquetado automático de caras

En este apartado final se muestra el estudio que se ha llevado a cabo en este proyecto acerca de la posibilidad de añadir un etiquetado de caras automático.

Un etiquetado automático de caras en el que a un grupo de caras se le asigna un nombre de persona, implementado haciendo uso de tareas de reconocimiento, permitiría a la aplicación desarrollada ofrecer un control más profundo de todas las personas detectadas y facilitaría la asignación de nombres a las mismas. Aunque la aplicación ofrece un etiquetado de caras manual tanto en el entrenamiento *offline* como en el entrenamiento *online*, no existe una manera automática de agrupar caras. *Picasa* de *Google* [24] es un organizador de imágenes digitales con un sistema de reconocimiento integrado que permite asociar grupos de caras detectadas en las fotos presentes en el ordenador con un nombre, de tal manera que las búsquedas de una persona da como resultado todas aquellas imágenes de esa persona. Similarmente, *iPhoto* de *Apple* [25], distribuido con la *suite* de aplicaciones *iLife*, incluye un sistema para etiquetar a las personas detectadas en las fotos, personas que luego pueden buscarse a través de la aplicación *Spotlight*.

Existen otros sistemas de tratamiento de imágenes que incluyen reconocimiento facial, como el *Picture Motion Browser* (PMB) de *Sony* o el *Windows Live Photo Gallery* de *Microsoft*, pero la idea estudiada para este proyecto se inspira en lo que ofrecen los dos primeros, y sobre todo *Picasa*, que es la aplicación que el proyectando ha podido probar.

Lo primero que se necesita para poder etiquetar caras es poder establecer que se considera que dos caras son la misma persona o no.

Como se ha explicado en el capítulo anterior, en la parte final de la implementación se decidió incluir un nivel de confianza en la tarea de reconocimiento para poder ofrecer un resultado más robusto. Con este nivel de confianza, el sistema es capaz de decir no solo qué persona es la más parecida de entre todas con las que fue entrenado, sino que ofrece, con ciertas limitaciones, la confianza que tiene en ese reconocimiento. Estableciendo un nivel de confianza mínimo (por ejemplo del 90%) para los reconocimientos positivos, la aplicación desarrollada podrá determinar automáticamente si dos personas son la misma o no.

Si se extiende la idea de comparación de dos personas al número total de caras en la base de datos, se podrían establecer grupos de imágenes que, según la aplicación, pertenecerían a la misma persona.

El proceso para llevarlo a cabo sería el siguiente:

1. Seleccionar la primera foto de la base de datos y crear el primer grupo de caras.

2. Comprobarla con todas las demás.

3. Aquellos reconocimientos positivos que superen el nivel de confianza mínimo, se consideran la misma persona que la de la foto a comprobar, se añaden a ese grupo y se marcan para no ser evaluados de nuevo.

4. Se continúa con la siguiente foto de la base de datos que no haya sido ya marcada como perteneciente a un grupo y se vuelve al paso 1.

5. Cuando todas las fotos son asociadas a los grupos, se presentan al usuario para que incluya los nombres.

En este proceso, se hace imprescindible seleccionar correctamente el nivel de confianza mínimo que discierne una comprobación positiva de una negativa. Si el nivel de confianza

mínimo es alto, pueden aparecer muchos grupos de pocas imágenes, dejando sin agrupar posiblemente caras de la misma persona. Por otro lado, si el nivel es bajo, puede ocurrir lo contrario e imágenes de diferentes personas aparecen en el mismo grupo. Un ejemplo del primer tipo de errores, en este caso en *Picasa*, se ilustra en la Figura 22, lo que deja patente la dificultad de esta tarea.

Figura 22. Captura de pantalla de la aplicación Picasa. A pesar de que hay un grupo de imágenes de Martin, entre las no seleccionadas en ningún grupo aparecen fotos de Martin.

La solución para evitar estos problemas pasa por tener un nivel de confianza muy granulado y evaluar todas las fotos para asociarlas solamente al grupo con el que el nivel de confianza sea mayor. Una foto que resulte reconocida al 90%, pero que luego resulte reconocida al 93%, y más adelante al 97%, debería asociarse al último grupo de los tres.

La implementación del nivel de confianza basada en la comparación del error mínimo con una distribución Chi-cuadrado de N grados de libertad, siendo N el número de caras de entrenamiento menos uno, tenía un par de limitaciones. Una de ellas era la necesidad de introducir manualmente los datos de la tabla Chi-cuadrado para realizar las comparaciones. La necesidad de tener un alto nivel de granularidad para el nivel de confianza no hace sino incrementar esta limitación, ya que es mayor el número de datos necesarios a mayor granularidad requerida. Además, debido a que es el usuario en cada instante el que establece el número de caras de entrenamiento, imposibilita el uso de ese nivel en cualquier caso, y solo permite su uso hasta un número determinado de caras de entrenamiento.

La decisión tomada tras este estudio es que el etiquetado de caras automático es posible pero no se ha implementado debido a las limitaciones encontradas en el cálculo del nivel de confianza de los reconocimientos.

Capítulo 6. Conclusiones

La aplicación presentada en este documento, además de cumplir con los objetivos establecidos, supone mejoras con respecto a otras aplicaciones de vigilancia y control consultadas durante el estudio previo. De entre esas mejoras, dejando a un lado las incluidas en los objetivos principales del proyecto acerca del ahorro en espacio y en coste económico, cabe destacar las siguientes:

- Se ofrecen opciones de procesado y detección totalmente configurables. Muchas de las aplicaciones consultadas no facilitan la modificación de los parámetros de detección o incluso lo hacen imposible. La aplicación de este proyecto muestra con una interfaz sencilla los parámetros de detección (tamaño de ventana, escala, etc.) y otras opciones generales (almacén de caras, frecuencia de escaneo, etc.) para que el usuario pueda configurar de manera fácil el comportamiento de la herramienta.

- Existe la posibilidad de detección automática en video (detección *offline*) y no solamente a través de una cámara web.

- Se ofrecen distintos tipos de detectores al alcance de un clic (de ojos, de boca, de nariz, etc.) pero también es posible cargar cualquier otro tipo de detector.

- Código libre y simple, adaptable a cualquier plataforma.

El resultado final es una aplicación de control y vigilancia de personas sencilla pero con múltiples funcionalidades muy fáciles de aprovechar por parte del usuario.

En los siguientes apartados de este capítulo se mencionan de manera breve los principales hitos conseguidos, los problemas más relevantes a los que se ha hecho frente y las posibles ampliaciones de este proyecto, para finalizar con la opinión personal del autor.

6.1 Hitos conseguidos

Se listan a continuación de manera breve los principales hitos conseguidos con la realización de este proyecto:

- Haber desarrollado una aplicación que cumple los requisitos y objetivos que se plantearon a comienzo del proyecto.

- Conseguir que una aplicación implementada con herramientas software de código libre incluya mejoras con respecto a otras que se encuentran ya en el mercado y que en la mayoría de los casos suponen un desembolso económico.

- Entender y poder explicar la base teórica del método de detección de objetos implementado en la biblioteca de software *OpenCV*, método además muy famoso e importante en el ámbito de la detección de objetos y caras.

- Crear una aplicación de usuario de Windows con interfaz agradable y multitud de funcionalidades relacionadas con la detección de caras y otros objetos fácilmente accesibles para el usuario.

Cada uno de estos hitos otorgan un gran valor y sentido al esfuerzo empleado por el autor de este proyecto en su realización.

Por último parece interesante destacar que otros sistemas de vigilancia analizados que proporcionan muchas de las funcionalidades de la aplicación desarrollada en este proyecto (detección y almacenamiento de caras, reconocimiento, etc.), como por ejemplo *VeriLook Surveillance SKD* de *Neurotechnology* [22], ofrece unos resultados similares en cuanto a espacio utilizado (almacena usando una base de datos relacional hasta 10.000 plantillas de caras, cada una conteniendo una cara, en aproximadamente 30MB de disco duro, que equivale a 3 KB por cara), pero no se han tenido en cuenta a la hora de realizar comparativas debido a su elevado coste (790 dólares por licencia en el caso de *VeriLook*), lo que descartaba este sistema en el contexto del presente proyecto.

6.2 Principales problemas encontrados

Si bien es cierto que tanto la biblioteca *OpenCV* como los entornos de desarrollo *Visual Studio* de *Microsoft* son ampliamente utilizados y abunda la documentación sobre ellos, diferentes problemas tuvieron que ser abordados y corregidos por el autor de este proyecto a lo largo de su realización.

El primero a destacar es la necesidad de configuración adicional que el hecho de usar dos herramientas de software diferentes, en este caso *OpenCV* y *Visual C++*, implica a la hora de desarrollar una aplicación.

Relacionado con el punto anterior cabe mencionar que *OpenCV* al actualizarse a la versión 2.0 cambió radicalmente la manera de instalación, dejando obsoleta la configuración previa añadida en *Visual C++*.

Por otro lado, en cuanto a problemas de tipo técnico, el principal escollo a solventar fue el hecho de que *OpenCV* requiera muchas veces tipos no manejados (*unmanaged*) como argumentos para sus funciones. Esto, unido a que el IDE (*Integrated Development Environment*) de *Visual C++* ofrece una gran variedad de métodos para manejo de *Strings*, que es un tipo manejado (*managed*), complica sobremanera el intercambio de datos entre funciones de *OpenCV* y las de sistema.

En el Anexo L aparecen detallados estos contratiempos así como los errores más relevantes relacionados con el código que se tuvieron que investigar y corregir.

6.3 Posibles ampliaciones

Las posibles continuaciones de este proyecto son muchas, destacando entre ellas la implementación del etiquetado de caras automático analizado en este mismo proyecto y que no se ha implementado. Con un etiquetado de caras que lograse tener catalogadas todas las caras almacenadas, el sistema podría ser capaz de ofrecer información adicional difícil de encontrar en otras aplicaciones similares, como por ejemplo un resumen de las veces que una misma persona ha sido detectada, es decir, ha llegado al lugar donde se sitúa la cámara de vigilancia y control.

El hecho de que *OpenCV* sea multiplataforma ofrece a su vez proyectos a priori simples en el concepto pero probablemente complejos en cuanto a su implementación. Migrar lo aquí implementado a otras plataformas, como por ejemplo al sistema operativo *Android* para dispositivos móviles, posibilitaría su uso en teléfonos móviles y/o tabletas.

Otra de las ampliaciones que se podrían añadir es un entrenamiento para detección de otro tipo de objetos, no solo caras, o caras en otras posiciones. Si bien esta posibilidad aparece referenciada en la memoria, no se llevó a cabo porque la preparación y las pruebas necesarias

requieren mucho tiempo y el detector de caras disponible ofrecía resultados suficientemente buenos .

Para finalizar, simplemente comentar que debido a que *OpenCV* a partir de su versión 2.4 (Mayo 2012) incluye en el paquete de funciones el reconocimiento de caras[9] realizado a partir de varios métodos (*Eigenface*, *Ficherface*, etc.), una posible continuación de este proyecto sería evaluar qué método funciona mejor en el ámbito de esta aplicación (imágenes para entrenamiento tomadas de cámara web, video o foto) e implementarlo.

[9] http://docs.OpenCV.org/trunk/modules/contrib/doc/facerec/

Anexo A. Visión por computador

La visión por computador, también conocida como visión artificial o visión por ordenador, es un sub-campo de la inteligencia artificial. El propósito de la visión por computador es conseguir que un ordenador sea capaz de "entender" o "ver" una escena o imagen.

La visión por computador tiene que ver con la teoría para la construcción de sistemas artificiales que obtengan información de las imágenes. Los datos de la imagen pueden adoptar muchas formas, como pueden ser secuencias de vídeo (*stream*), cámaras en tiempo real (por ejemplo una *cámara web*) o datos multidimensionales como los de un escáner médico.

Ejemplos de aplicaciones de sistemas de visión por ordenador son:

- Detección/Reconocimiento de eventos u objetos (por ejemplo para sistemas de vigilancia y/o rescate).
- Interacción hombre-máquina.
- Organización de información (por ejemplo la indexación de bases de datos de imágenes).
- Modelado de objetos o entornos (por ejemplo, análisis de imágenes médicas o topográficas).
- Control de procesos (por ejemplo un robot industrial o un vehículo autónomo).

La visión por ordenador también puede ser descrita como un complemento de la visión biológica. En la visión biológica, la percepción visual de los seres humanos y de distintos animales se encuentra en continuo estudio, lo que genera como resultados modelos acerca del funcionamiento de estos sistemas en términos de procesos fisiológicos. La visión por ordenador, por otro lado, estudia y describe sistemas de visión artificial implementados en software y/o hardware. El intercambio de conocimientos entre la visión biológica y la visión por ordenador ha demostrado ser cada vez más fructífera para ambos campos.

Al igual que la visión por ordenador se considera una parte o sub-campo de la inteligencia artificial, también se puede entender que la visión por computador se divide en diferentes subdominios que se corresponden con tareas que se consideran típicas de visión por computador. Ejemplos de estas tareas típicas son reconstrucción de escenas, cálculo de movimiento, detección de eventos y objetos, seguimiento, reconocimiento, aprendizaje o restauración de imágenes.

Actualmente, el campo de visión por computador puede ser caracterizado como inmaduro y diverso. Aunque existen trabajos anteriores, no fue hasta finales de 1970 cuando un estudio más profundo de la materia se inició al conseguir que los ordenadores pudieran manejar grandes conjuntos de datos como son las imágenes. Sin embargo, al proceder estos estudios de otros campos no existe una formulación estándar del problema de la visión por ordenador. Del mismo modo, no hay ninguna formulación estándar acerca de cómo los problemas de visión por ordenador deben ser solucionados. En lugar de ello, existe una gran cantidad de métodos para la solución de diversas tareas típicas de visión por ordenador, métodos que al ser muy específicos rara vez pueden ser generalizados para una amplia gama de aplicaciones. Muchos de los métodos y aplicaciones están aún en el estado de la investigación básica, pero cada vez más los métodos han encontrado su camino en productos comerciales, en los que a menudo constituyen una parte de un sistema más amplio que puede resolver tareas complejas (por ejemplo, en el

ámbito de la imágenes médicas, el control de calidad o mediciones en los procesos industriales). En la mayoría de las aplicaciones prácticas de visión por ordenador, los ordenadores son pre-programados para resolver una tarea. Sin embargo, métodos basados en el aprendizaje son cada vez más comunes.

Uno de los campos de aplicación más importantes de la visión por computador es la visión o el procesamiento de imágenes médicas. Esta área se caracteriza por la extracción de información de datos de la imagen con el fin de establecer un diagnóstico médico de un paciente. En general, los datos de la imagen se presentan en forma de imágenes microscópicas, imágenes de rayos X, angiografías, imágenes de ultrasonido y tomografías. Un ejemplo de información que se puede extraer de esos datos de la imagen es la detección de tumores, arteriosclerosis u otros cambios en el tejido que puedan ser malignos. Asimismo, también puede extraerse información para la medición de las dimensiones de órganos, la circulación de la sangre, etc. Esta área de aplicación también apoya la investigación médica, proporcionando nueva información como por ejemplo acerca de la estructura del cerebro o acerca de la calidad de los tratamientos médicos.

Una segunda área de aplicación de la visión artificial es la industria. En este caso, se extrae la información con el fin de apoyar un proceso de fabricación. Un ejemplo es el control de la calidad donde productos finales o incluso pequeños detalles en el proceso se están automáticamente inspeccionando con el fin de encontrar defectos y posteriormente poder ser corregidos. Otro ejemplo es la medición de la posición y orientación de un determinado objeto para que pueda ser recogido por un brazo del robot.

Las aplicaciones militares son probablemente uno de los mayores ámbitos de la visión artificial. Los ejemplos evidentes son la detección de los soldados o vehículos enemigos y misiles guiados. Los sistemas de orientación y guía de misiles más avanzados envían el misil a una zona en vez de a un objetivo concreto. Este objetivo se elige cuando el misil alcanza la zona determinada y se adquieren imágenes más precisas del lugar en cuestión.

Una de las nuevas áreas de aplicación son los vehículos autónomos, que incluyen los sumergibles, vehículos terrestres (pequeños robots con ruedas, coches o camiones), vehículos aéreos y vehículos aéreos no tripulados UAV (*unmanned aerial vehicles*). El nivel de autonomía varía entre totalmente autónomos (no tripulados) a los vehículos donde sistemas basados en visión por ordenador dan apoyo a un conductor o un piloto en diversas situaciones. Los vehículos totalmente autónomos suelen utilizar la visión artificial para la navegación, es decir, para saber dónde están, para la elaboración de un mapa de su entorno y/o para la detección de obstáculos. El sistema de visión también se puede utilizar para la detección de ciertos eventos, como por ejemplo implementar un sistema que sea capaz de detectar incendios forestales. En cuanto a vehículos donde la visión artificial sirve de apoyo, ejemplos son los sistemas de alerta de obstáculos en los automóviles y los sistemas autónomos de aterrizaje de los aviones. Varios fabricantes de automóviles han mostrado sistemas de conducción autónomos de los coches, pero esta tecnología aún no ha llegado a un nivel donde se pueda llevar al mercado. En este sentido, existen abundantes ejemplos de vehículos militares autónomos que van desde avanzados misiles guiados a vehículos aéreos no tripulados para misiones de reconocimiento. La exploración del espacio ya está siendo realizada con vehículos autónomos utilizando visión por ordenador, por ejemplo con el vehículo de la NASA llamado *Mars Exploration Rover*.

Otras áreas donde se aplica cada vez más la visión por computador son en el apoyo a la creación de efectos visuales para el cine y en el campo de la vigilancia.

Cada una de las áreas de aplicación descritas anteriormente hace uso de un conjunto de tareas de visión por ordenador que se podrían considerar típicas de este campo. Un claro ejemplo es la tarea de reconocimiento, la cual debe ser planteada y solucionada a la hora de detectar tanto un tumor como un obstáculo en la carretera.

El reconocimiento de puede considerar como el problema clásico en la visión artificial. Se trata de determinar si los datos de una imagen contienen un objeto, característica o actividad específica. Esta tarea, que normalmente puede ser resuelta con firmeza y sin esfuerzo por un humano, aún no está resuelta de manera satisfactoria en la visión artificial para el caso más general: objetos arbitrarios en situaciones o imágenes arbitrarias. Los actuales métodos para hacer frente a este problema sirven para resolverlo sólo para determinados objetos, como objetos geométricos simples (por ejemplo, los poliedros), rostros humanos, caracteres escritos a mano o impresos, vehículos, y en situaciones específicas de buena iluminación y fondo contrastado, la situación y orientación de un objeto en relación a la cámara con la que se ha tomado la imagen. Distintas variaciones del problema del reconocimiento aparecen en la bibliografía del campo de visión por computador. De entre las más relevantes cabe destacar las siguientes:

- Detección: los datos de una imagen se escanean con el fin de hallar una condición específica. Por ejemplo, la detección de posibles anomalías en las células o los tejidos a través de imágenes médicas o la detección de un vehículo en un sistema automático de peaje. Normalmente, la detección se basa en aspectos relativamente sencillos y rápidos de calcular por lo que se suele utilizar para encontrar regiones de interés que sean más pequeñas que la imagen inicial. A estas zonas de la imagen con posibilidades de ofrecer resultados positivos se les aplican técnicas de cálculos más exigentes para intentar producir una interpretación lo más precisa posible.

- Identificación: una instancia o un objeto específico es reconocido. El ejemplo más sencillo es la identificación de una persona en concreto a partir de la cara o de la huella digital, pero también se puede aplicar a la identificación de un determinado vehículo

Además del reconocimiento, el cálculo de movimiento (una secuencia de imágenes es procesada para producir una estimación de la velocidad) y la reconstrucción de escenas (dadas una o más imágenes de una escena o un vídeo, calcular un modelo 3D de la escena) son tareas que se consideran típicas del campo de visión por ordenador.

Anexo B. Entorno de desarrollo

En este anexo se ofrece un resumen de la biblioteca de Intel para visión por computador llamada *OpenCV*, abreviatura de su nombre completo en inglés *Open Source Computer Vision Library*, así como del entorno desarrollo integrado para lenguaje C++ de *Microsoft*, *Visual C++*. Ambas tecnologías han sido claves en el desarrollo de este proyecto.

B.1 *Open Source Computer Vision Library*

OpenCV, abreviatura de *Open Source Computer Vision*, es una biblioteca de código abierto que incluye actualmente más de 2500 algoritmos para el procesamiento de imágenes, visión por ordenador y otros algoritmos numéricos de propósito general. Su licencia BSD (*Berkeley Software Distribution*[10]) permite que sea usada tanto con propósitos comerciales como de investigación.

La biblioteca es multiplataforma, y puede ser usada en sistemas clásicos como *Mac OS X*, *Windows* o *Linux*, así como en sistemas operativos más recientes como *Android*.

OpenCV se ofrece con interfaces para los lenguajes de programación C, C++, *Phyton* y *Java* (solo para *Android*), siendo los de C y C++ los más completos.

Este proyecto de *Intel* que fue lanzado en 1999, cuenta con más de 46000 personas en el grupo de usuarios y ha tenido más de 3.5 millones de descargas. El hecho de que sea de código abierto hace que cada día más y más personas lo utilicen y a su vez contribuyan a su expansión y robustez, ya sea corrigiendo errores (*bugs*) o añadiendo nuevas funcionalidades.

B.1.1 Motivación para su creación y futuro

En el campo de la visión por ordenador, vasto y en continuo desarrollo, no existía una API (*Application Programming Interface*) estándar como podían ser *OpenGL* (*Open Graphics Library*) y *DirectX* en gráficos. La mayoría del software para visión por ordenador se podía encasillar en alguno de los siguientes tipos:

- Código de investigación: inestable, independiente e incompatible.
- Comercial: caro.
- Soluciones especializadas integradas en hardware: aparatos de vigilancia, equipos médicos, etc.

En 1999, *Intel* decidió que una librería estándar simplificaría el desarrollo de cualquier nueva aplicación es este campo y lanzó el proyecto de *OpenCV*, cuyos objetivos primordiales fueron:

- Crear un conjunto de funciones para tareas de visión por ordenador con código libre, portable y optimizado para esas tareas.
- Expandir el conocimiento sobre la visión por ordenador.

[10] Hace referencia a la licencia de software otorgada principalmente para los sistemas BSD. Es una licencia de software libre permisiva.

La versión alfa del proyecto fue lanzada en el año 2000 durante la conferencia sobre visión por ordenador y patrones de reconocimiento del Instituto de Ingenieros Eléctricos y Electrónicos (*IEEE Conference on Computer Vision and Pattern Recognition*). La primera versión final no salió a la luz hasta el año 2006. En el año 2009, *OpenCV* fue lanzada en su versión 2.0, que incluía cambios sustanciales en el interfaz de C++, haciéndolo más sencillo, más seguro, más eficiente y más completo.

Como metas, el proyecto pretende proveer un *Tool-Kit* o marco de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado realizando su programación en código optimizado, aprovechando además las capacidades que proveen los procesadores multi-núcleo. *OpenCV* puede adicionalmente utilizar las Primitivas de Rendimiento Integradas de Intel (IPP, *Integrated Performance Primitives*), que es un conjunto de rutinas de bajo nivel específicas para procesadores *Intel*.

B.1.2 ¿Por qué *OpenCV* para este proyecto?

El estudio previo de artículos relacionados con la detección de caras así como la sugerencia del director llevó al estudio de esta biblioteca como alternativa para el desarrollo de este proyecto.

El hecho de que *OpenCV* sea un conjunto de funciones donde el procesamiento de imágenes en tiempo real ha sido siempre un objetivo para los desarrolladores de la biblioteca, añade otro dato a favor de su elección ya que encaja perfectamente con el contexto de este proyecto.

A su vez, su publicación da bajo licencia BSD que permite que sea usada libremente para propósitos de investigación, como es el presente proyecto, concuerda plenamente con el objetivo de mínimo coste previamente establecido.

B.2 *Microsoft Visual C++ Express Edition*

La implementación del sistema se ha realizado en el entorno de desarrollo integrado de *Microsoft Visual C++* en su versión gratuita (*Express Edition*).

Visual C++ Express es parte de la familia *Visual Studio Express*, un conjunto gratuito de herramientas de software que desarrolladores sobre plataformas *Windows* pueden utilizar para crear aplicaciones personalizadas, tanto a nivel básico como experto. *Visual C++* está diseñado para ofrecer al desarrollador un control profundo y detallado cuando se generan bien aplicaciones *Windows* nativas (COM+[11]) o bien sobre .NET *Framework*[12], como la de este proyecto.

Cuando se habla de la versión *Express* de *Visual Studio*, se hace referencia a la versión gratuita del producto. Estas versiones *Express* proporcionan un subconjunto de la funcionalidad que está disponible en otras ediciones de *Visual Studio* (*Ultimate*, *Premium*, *Profesional*) y por lo tanto, algunos de los componentes, bibliotecas o características disponibles en esas ediciones no están disponibles en la versión *Express* del producto.

[11] http://es.wikipedia.org/wiki/Component_Object_Model
[12] http://msdn.Microsoft.com/es-es/netframework/default.aspx

B.2.1 Estructura básica de un proyecto en *Visual C++*

La gran ventaja de usar un entorno de desarrollo integrado es que la estructura básica de ficheros y las relaciones o referencias necesarias para su posterior compilación y ejecución se pueden obtener automáticamente a través de un asistente para aplicaciones o *wizard*.

Este proyecto se ha desarrollado a partir de los ficheros estándar principales generados mediante un asistente para aplicaciones de tipo formulario, que son aplicaciones con una interfaz de usuario de *Windows*. Estos ficheros son:

- Archivo de solución (.**vcproj**): contiene información acerca de las plataformas, configuraciones y características del proyecto seleccionadas en el asistente para aplicaciones.

- Archivo de código fuente de C++ (.**cpp**): contiene el código para mostrar el formulario.

- Archivo principal de encabezado (**Form1.h**): contiene la implementación de la clase de formulario.

- Archivo de inclusión **StdAfx.h**: es el archivo principal de inclusión. En él se deben especificar todos los archivos de inclusión estándar del sistema o archivos de inclusión específicos de un proyecto utilizados frecuentemente pero rara vez modificados.

En el caso del archivo de solución (**vcproj**) y del archivo principal de exclusión (**StdAfx.h**), son archivos únicos. Por el contrario, el proyecto final consta de múltiples archivos de encabezado y de código fuente, que se corresponden con cada uno de los formularios que ha sido preciso desarrollar.

B.2.2 ¿Por qué *Visual C++ Express* para este proyecto?

Al comienzo de este proyecto, la biblioteca de *OpenCV* se ofrecía con interfaces muy completos solamente para C y C++. A la hora de buscar un entorno de desarrollo integrado para ese tipo de lenguajes que a su vez ofreciese la posibilidad de crear aplicaciones con una interfaz de usuario de *Windows*, apareció la posibilidad de usar *Visual C++* de *Microsoft*. Debido a que el presente proyecto se iba a desarrollar en plataformas *Windows* de *Microsoft*, parecía buena herramienta para trabajar.

Otro dato que ayudó a la elección de esta herramienta fue el hecho de que exista en la red infinidad de documentación sobre la interacción entre *OpenCV* y *Visual C++*, lo que iba a facilitar en gran medida el desarrollo del proyecto.

Del mismo modo, el hecho de que *Microsoft* ofrezca una versión totalmente gratis del producto para desarrolladores, concordaba con el objetivo de minimizar el coste del proyecto, haciendo uso de software libre.

B.3 Uso de *OpenCV* con *Visual C++ Express*

Para hacer funcionar de manera correcta la biblioteca *OpenCV* con el entorno de desarrollo integrado *Visual C++*, son necesarios varios pasos adicionales al simple hecho de instalar cada aplicación por separado.

En concreto, hay que indicar que se van a usar funciones de una librería de la que no tiene constancia. Los ajustes pertinentes se realizan en el propio *Visual C++* a través de los menús de opciones de proyectos y soluciones. Las modificaciones necesarias son:

- Archivos de inclusión: Ruta de acceso que se utilizará al buscar archivos de inclusión durante la generación de un proyecto de VC++. Se corresponde con la variable de entorno INCLUDE.

- Archivos de biblioteca: Ruta de acceso que se utilizará al buscar archivos de bibliotecas durante la generación de un proyecto de VC++. Se corresponde con la variable de entorno LIB.

- Archivos de código fuente: Ruta de acceso que se utilizará al buscar archivos de código fuente para *IntelliSense*.

El paso final necesario para completar la configuración debe realizarse para cada proyecto o solución que se vaya a desarrollar en este entorno. Consiste en añadir como dependencias adicionales para la entrada del vinculador de C++ las bibliotecas de *OpenCV*.

Una vez completados los pasos anteriores, un proyecto desarrollado en *Visual C++* que use llamadas a funciones de *OpenCV* ya está listo para compilarse y poder generar el ejecutable.

A continuación, se detalla cómo realizar estas configuraciones. Primero se describen los ajustes generales para a continuación detallar los ajustes de proyecto, todo dentro del entorno de *Visual C++*.

B.3.1 Propiedades generales

Los ajustes generales se realizan a través de:

- Herramientas
- Opciones

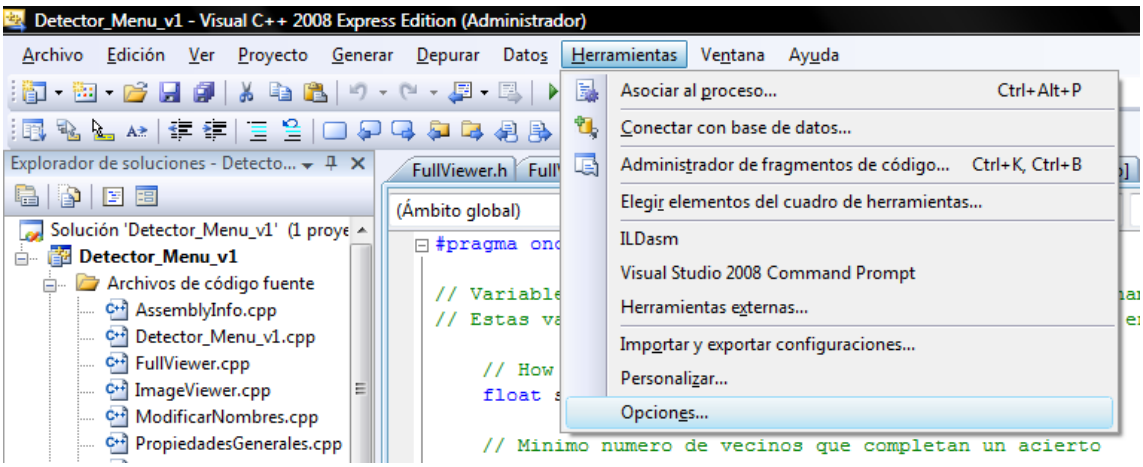


Figura B1. Captura de pantalla del menú Herramientas dentro de Visual C++ 2008 Express Edition.

A continuación, se selecciona:

- Proyectos y soluciones

- Directorios de VC++

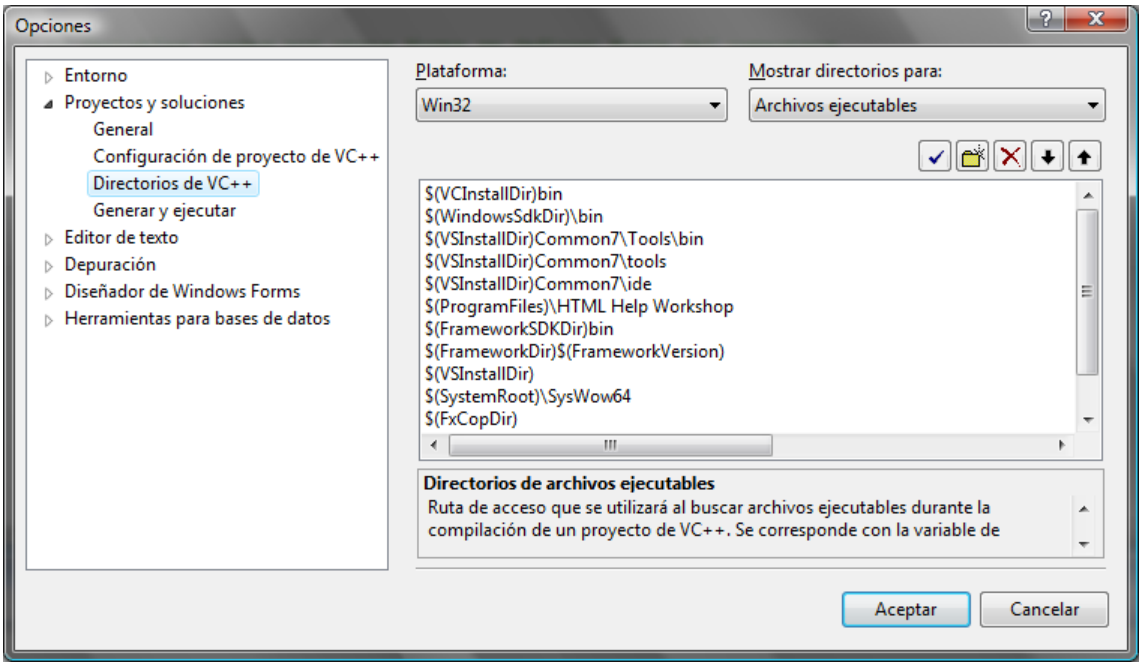


Figura B2. Captura de pantalla del menú Opciones, Directorios de VC++, Archivos ejecutables.

Es necesario modificar los directorios de archivos de inclusión, archivos de biblioteca y archivos de código fuente. En cada caso, se selecciona el tipo deseado con el menú desplegable a tal efecto (parte superior derecha).

- Archivos de inclusión: Ruta de acceso que se utilizará al buscar archivos de inclusión durante la generación de un proyecto de VC++. Se corresponde con la variable de entorno INCLUDE. Se deben incluir las rutas a los directorios INCLUDE de *OpenCV*.

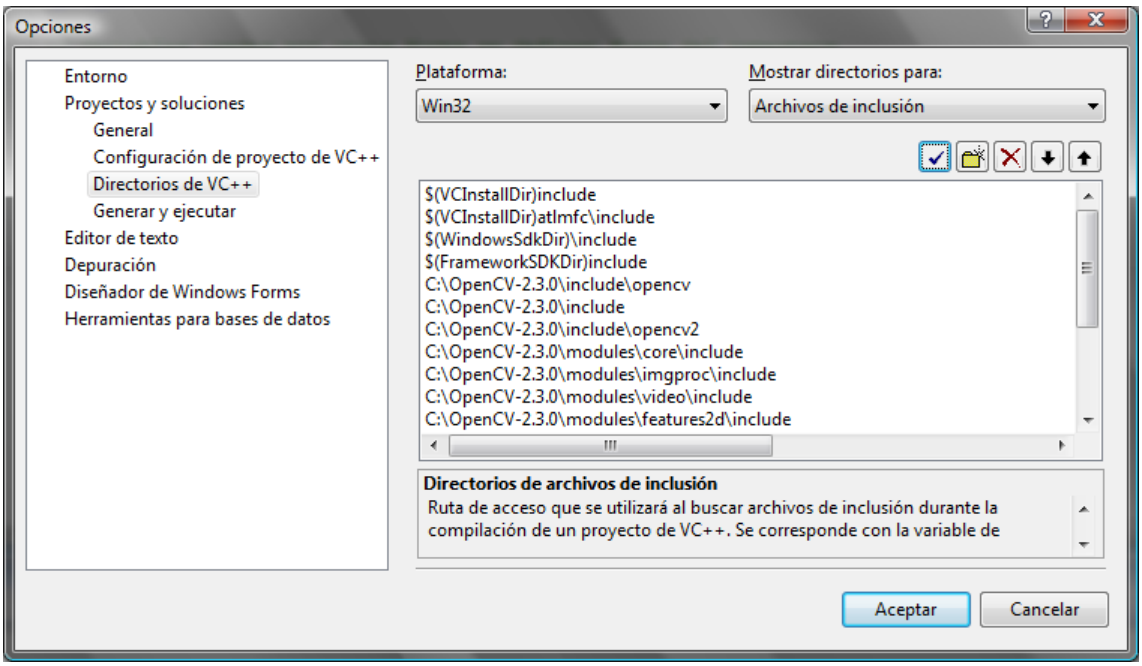


Figura B3. Captura de pantalla del menú Opciones, Directorios de VC++, Archivos de inclusión.

- Archivos de biblioteca: Ruta de acceso que se utilizará al buscar archivos de bibliotecas durante la generación de un proyecto de VC++. Se corresponde con la variable de entorno LIB. Se deben incluir las rutas al directorio LIB de *OpenCV*.

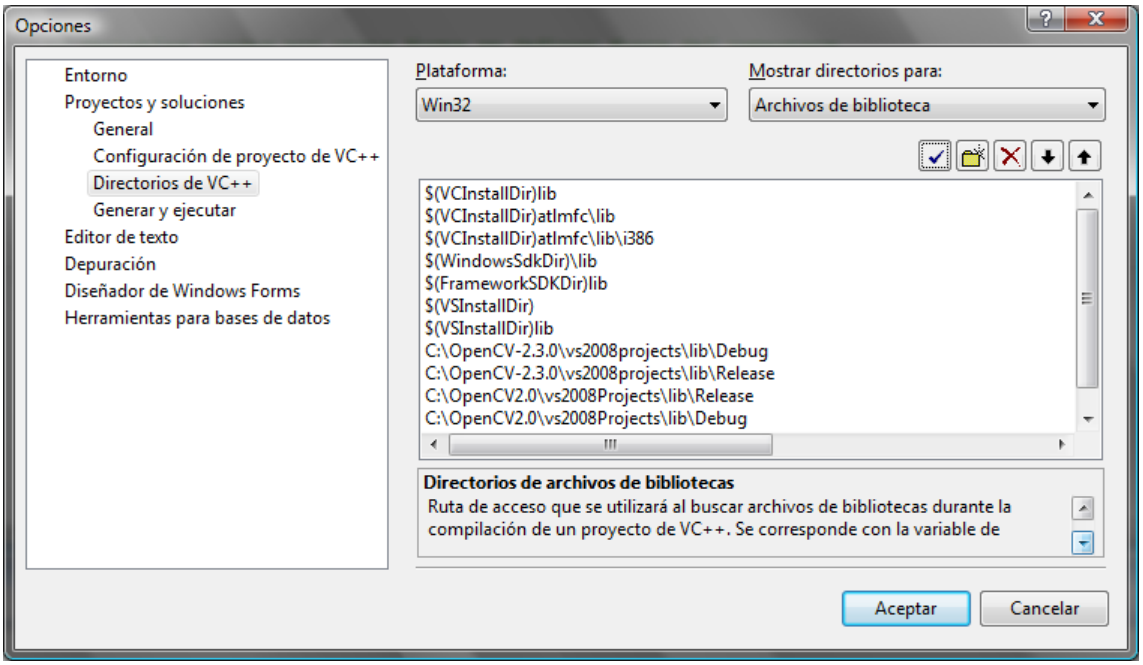


Figura B4. Captura de pantalla del menú Opciones, Directorios de VC++, Archivos de biblioteca.

- Archivos de código fuente: Ruta de acceso que se utilizará al buscar archivos de código fuente para *IntelliSense*. Se deben incluir las rutas a los directorios con archivos de código fuente de *OpenCV*.

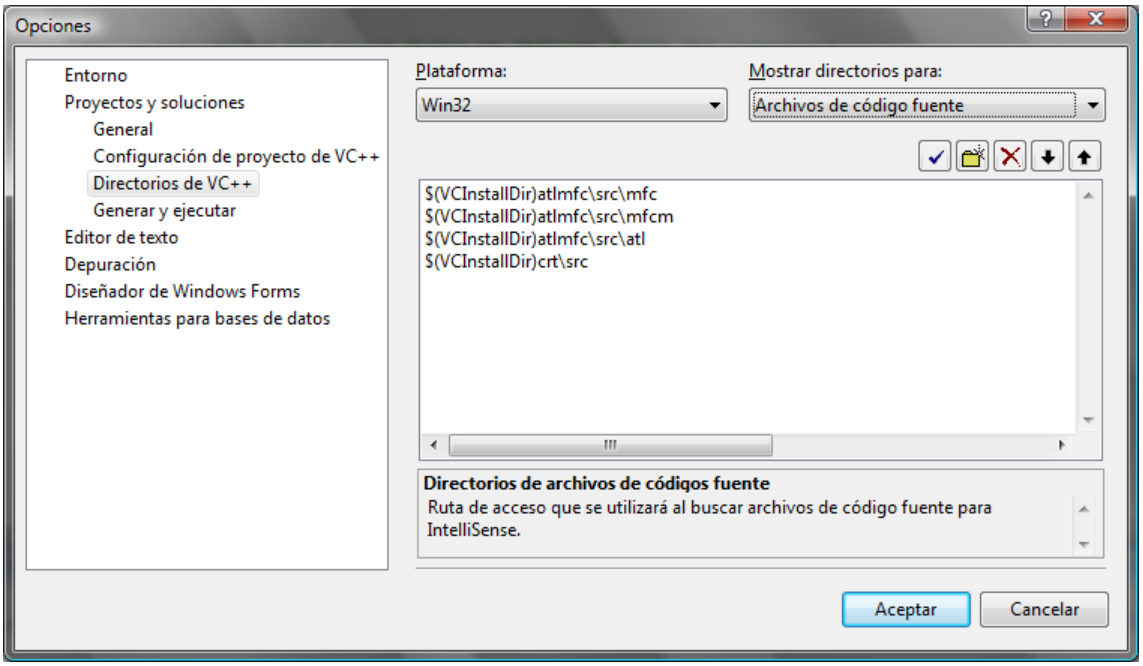


Figura B5. Captura de pantalla del menú Opciones, Directorios de VC++, Archivos de código fuente.

B.3.2 Propiedades del proyecto

Los ajustes del proyecto se realizan a través de:

- Proyecto
- Propiedades

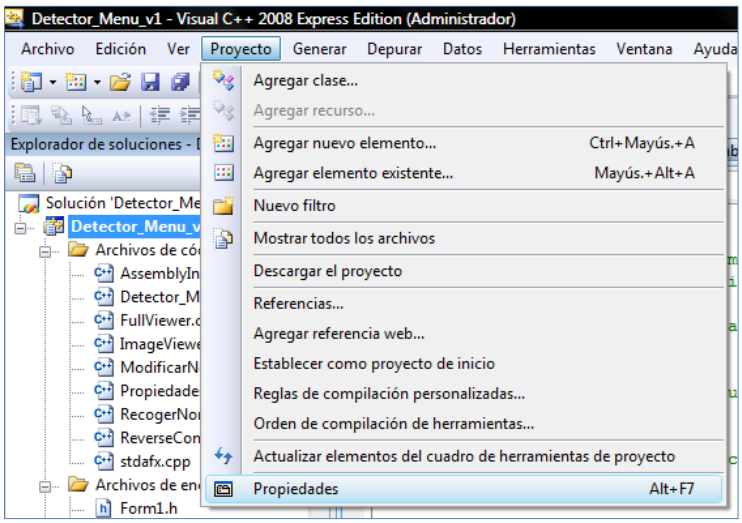


Figura B6. Captura de pantalla del menú Proyecto dentro de Visual C++ 2008 Express.

A continuación, hay que acceder a:

- Propiedades de configuración
- Vinculador
- Entrada

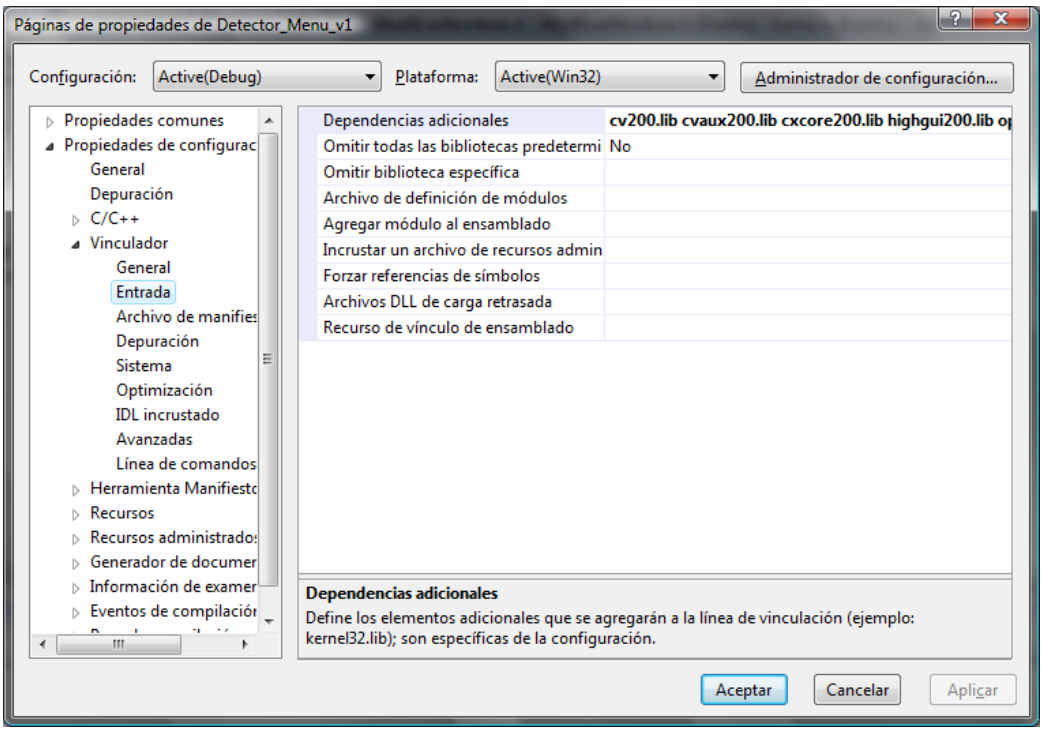


Figura B7. Captura de pantalla de la página de propiedades del proyecto, Propiedades de configuración, Vinculador, Entrada.

En la sección de dependencias adicionales, hay que añadir todas las bibliotecas necesarias de *OpenCV*.

B.3.3 Instalación de *OpenCV* versión 2.0 para uso con *VS Express C++* 2008

Por último, este último sub-apartado incluye una explicación completa acerca de los pasos adicionales que es preciso realizar para que la instalación de la versión 2.0 de *OpenCV* pueda ser usada junto con *Visual C++*. Esta explicación resulta de gran utilidad ya que la versión 2.0 de *OpenCV* requiere una construcción manual de los archivos de biblioteca necesarios para desarrollar aplicaciones (.LIB). Esto no era necesario para la versión 1.0 y es por eso que se incluye aquí la explicación como referencia.

Debido a infinidad de problemas técnicos, el paquete de instalación de la versión 2.0 de *OpenCV* no incluye librerías pre-compiladas de *OpenCV* para su uso en entornos de desarrollo integrados. Aunque ofrece librerías construidas con *MinGW* 4.3.3[13] suficientemente buenas para ejecutar ejemplos y pruebas proporcionados con la instalación, no son suficientes para el desarrollo de aplicaciones basadas en *OpenCV* en entornos como *Microsoft Visual Studio*, *Borland IDE*, etc. Para estos entornos de desarrollo así como para otras versiones de *MinGW*, es necesario construir las librerías con el compilador de la propia aplicación de desarrollo y usando también *CMake*[14]. En el caso de *Visual C++* 2008 *Express*, entorno usado en el desarrollo de este proyecto, la información detallada de cómo generar estas librerías se encuentra a continuación.

- Obtener *OpenCV* 2.0.
- Instalar en ruta sin espacios (por una limitación o *bug* en los scripts, no se admite las rutas con espacios).
- Obtener *CMake* (los binarios).
- Instalar *CMake*.
- Ejecutar *CMake* GUI (*Graphical User Interface*).
- En *CMake* GUI, "*Where is the source code*", seleccionar *C:\OpenCV2.0*.
- Crear un directorio llamado vs2008 en *C:\OpenCV2.0*.
- En *CMake* GUI, "*Where to build the binaries*", seleccionar el directorio creado antes (*C:\OpenCV2.0\vs2008*).
- Ir a Configurar y seleccionar "*Visual Studio 9 2008*".
- Una vez terminado, seleccionar las opciones que se deseen teniendo en cuenta quitar la opción de ENABLE_OPENMP ya que las versiones *Express* de *Visual Studio* no soportan *Open MP*[15].

[13] *Minimalist GNU for Windows*, implementación de los compiladores GNU para la plataforma Win32.
[14] Herramienta multiplataforma de generación o automatización de código.
[15] *OpenMP* es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas.

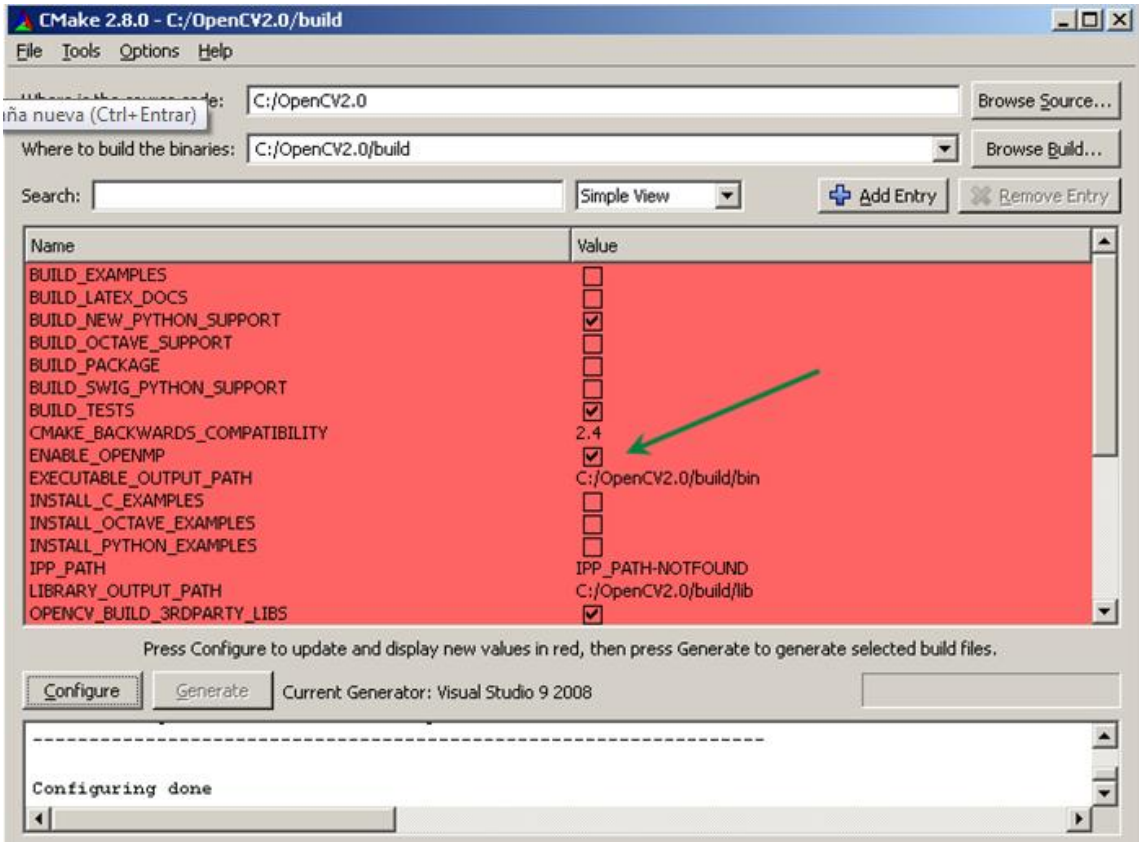


Figura B8. Captura de pantalla de la aplicación CMake indicando la opción de ENABLE OPENMP que se recomienda no seleccionar

- Clicar en Configurar.
- Clicar en Generar.

A partir de aqui, se continua en *Visual Express*.

- Abrir proyecto seleccionando *C:\OpenCV2.0\vs2008\OpenCV.sln*.
- Construir el proyecto en modo *Debug*.
- Construir el proyecto en modo *Release*.
- Añadir al *System Path C:\OpenCV2.0\vs2008\bin\Debug* y *C:\OpenCV2.0\vs2008\bin\Release*.

Lo último a configurar es lo relacionado con la interacción entre *OpenCV* y *Visual Express*, detallado en secciones anteriores.

Con la última versión de *OpenCV* utilizada en este proyecto, versión 2.3, existen unas consideraciones de configuración adicionales que se presentan a continuación.

Es necesario poner como referencias todos los archivos nuevos, pero sin eliminar las referencias a los de versiones anteriores (2.0). Por ejemplo los DLL a añadir son:

- OpenCV_core230d.lib

- OpenCV_highgui230d.lib
- OpenCV_video230d.lib
- OpenCV_ml230d.lib
- OpenCV_legacy230d.lib
- OpenCV_imgproc230d.lib

Asimismo, hay que poner en archivos de inclusión todos los módulos referenciados obsoletos. En este caso, los nuevos *include* están en C:\OpenCV-2.3.0\modules. Se deben añadir:

- C:\OpenCV-2.3.0\modules\imgproc\include
- C:\OpenCV-2.3.0\modules\video\include
- C:\OpenCV-2.3.0\modules\features2d\include
- C:\OpenCV-2.3.0\modules\legacy\include
- C:\OpenCV-2.3.0\modules\objdetect\include
- C:\OpenCV-2.3.0\modules\calib3d\include
- C:\OpenCV-2.3.0\modules\flann\include
- C:\OpenCV-2.3.0\modules\highgui\include

Por último, en las propiedades del proyecto, además de las referencias existentes, es preciso poner como entrada del vinculador C++ todos los ficheros de biblioteca siguientes, ya que en esta versión 2.3 las librerías se encuentran separadas:

- OpenCV_core230d.lib
- OpenCV_highgui230d.lib
- OpenCV_video230d.lib
- OpenCV_ml230d.lib
- OpenCV_legacy230d.lib
- OpenCV_imgproc230d.lib
- OpenCV_calib3d230d.lib
- OpenCV_contrib230d.lib
- OpenCV_features2d230d.lib
- OpenCV_flann230d.lib
- OpenCV_gpu230d.lib
- OpenCV_objdetect230d.lib
- OpenCV_ts230d.lib

Anexo C. Análisis y diseño

En este anexo se presentan diferentes actividades relacionadas con fases previas a la implementación de la aplicación (desarrollo de código) que se realizaron a lo largo del proyecto y que se pueden englobar como pertenecientes a fases de análisis y diseño.

Se incluyen los requisitos establecidos, diagramas básicos de actividad de las funcionalidades principales del sistema, un estudio sobre aplicaciones presentes en el mercado en el ámbito de este proyecto, el listado de artículos académicos consultados por el autor del proyecto, y por último, los modelos de la interfaz de usuario diseñadas para el sistema completo en la parte final del proyecto.

C.1 Requisitos del proyecto

Aunque los objetivos del proyecto aparecen expuestos en la memoria principal, este apartado pretende dar simplemente una descripción más detallada de las características que debe cumplir el sistema a desarrollar.

Para comenzar, se listan los requisitos funcionales que definen las acciones fundamentales que debe realizar la aplicación al recibir información, procesarla y producir resultados:

- La comunicación del usuario con el sistema deberá ser a través de una aplicación de ventanas (estilo *Windows*).
- El sistema deberá permitir cargar imágenes fijas (fotos), vídeos e imagen de cámara web.
- El sistema deberá mostrar en sus correspondientes ventanas las opciones principales.
- Se deberá informar de errores al introducir datos incorrectos por parte del usuario.
- Se deberá informar al usuario de resultados relevantes: caras detectadas, persona reconocida, etc.
- El sistema deberá almacenar información con resultados para estadísticas y análisis.

Por otro lado, se detallan los requisitos no funcionales del sistema, es decir, aquellos que no forman parte directamente del funcionamiento (entradas, procesamiento, salidas) del mismo. En el caso de este proyecto, se reducen a los siguientes:

- Todas las herramientas de software usadas para el desarrollo de la aplicación no deben suponer coste económico alguno.
- En el almacenamiento de cualquier tipo de información debe primar el ahorro en coste de espacio en disco.
- Debe generarse una guía de usuario completa y entendible.

C.2 Diagramas de estado

En este apartado se muestran los diagramas de estado de las tres funcionalidades más relevantes del sistema: la detección de caras a través de la cámara web, el entrenamiento del reconocedor con caras obtenidas de la cámara web y el reconocimiento de personas también a través de la cámara web.

Estas funcionalidades, que sirvieron al autor del proyecto para plasmar de manera gráfica lo que se pretendía conseguir con la implementación de cada una de ellas, no fueron obviamente las primeras en ser diseñadas, pero se recogen en esta memoria ya que plasman de manera muy sencilla lo más interesante que ofrece este sistema.

El primer diagrama, el más sencillo de los tres, mostrado en la Figura C1, ilustra la detección de caras a través del a cámara web, que se podría considerar como la acción de vigilancia.

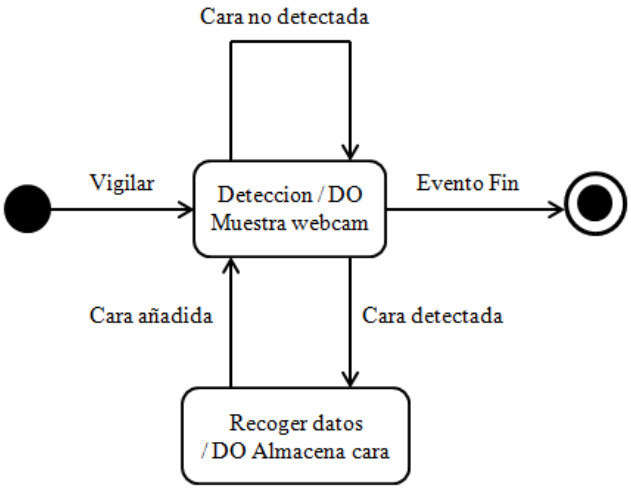


Figura C1. Diagrama de estados de la detección de caras a través de la imagen de la cámara web.

En él se aprecia que todas caras detectadas en la imagen suministrada (y mostrada) por la cámara web son almacenadas mientras no se interrumpa el proceso debido a la llegada del evento de finalización (por ejemplo que el usuario pulse la tecla Escape).

El diagrama de la Figura C2, que detalla la acción relacionada con el entrenamiento del reconocedor con imágenes recogidas a través de la cámara web, indica que el proceso comienza cuando se invoca a la acción entrenar. En ese momento, es necesaria la recogida del nombre del sujeto con el que se va a entrenar al sistema a través de un formulario básico. Una vez confirmado el nombre, el sistema muestra la imagen de la cámara web y comienza la tarea de detección. Cada cara detectada implica su almacenamiento y la actualización del entrenador. Mientras no se detecten caras y no se detecte el evento de finalización, el sistema continúa en el estado detección indefinidamente.

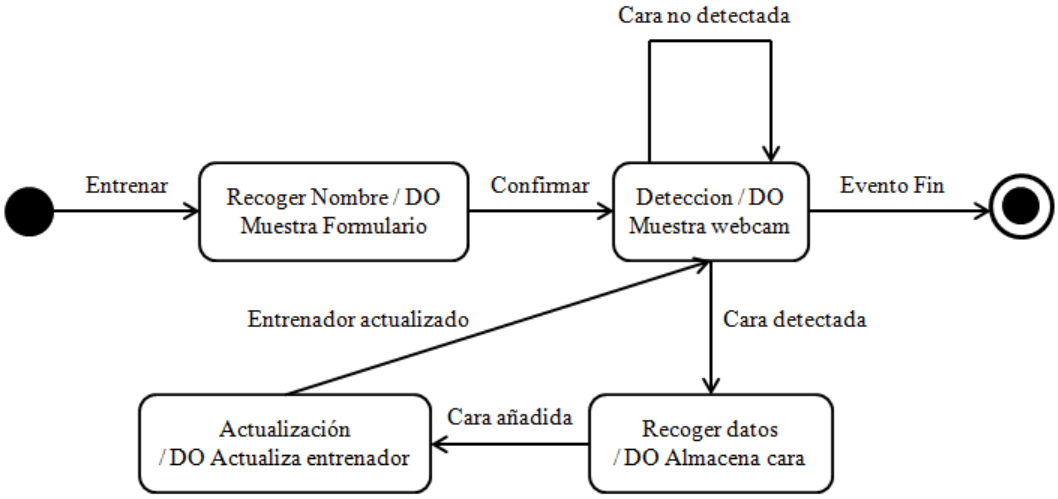


Figura C2. Diagrama de estados del entrenamiento a través de la cámara web para tareas de reconocimiento.

Para finalizar, el diagrama de la Figura C3 muestra la acción relacionada con el proceso de reconocimiento de personas a través de la cámara web.

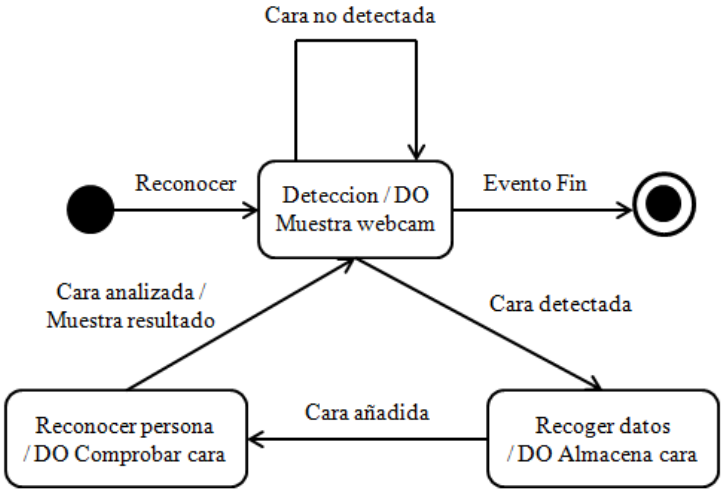


Figura C3. Diagrama de estados del reconocimiento a través de la imagen de la cámara web.

Similarmente a lo que ocurre con la detección y el entrenamiento con cámara web, el sistema se mantiene en el estado de detección de caras mientras no se detecte niguna y no se reciba el evento de finalización. Cuando una cara es detectada, se almacena y se intenta reconocer a esa persona, informando del resultado de la comprobación.

C.3 Estudio previo sobre aplicaciones similares en el mercado

Se muestra en la siguiente tabla un listado de herramientas relacionadas con la función de vigilancia y provistas de funcionalidades parecidas a las que se pensaba ofrecer en este proyecto, todas ellas consultadas por parte del autor como estudio previo antes del diseño del sistema a desarrollar.

Producto	Precio más bajo	Características destacables	Proveedor
EyeLine Video Surveillance	$35 para una cámara	- Diseñado para profesionales - Reconoce movimiento para capturar pantalla	http://www.nchsoftware.com/surveillance/index.html
Magic Video Capture/Convert/Burn Studio (Captura de video; no vigilancia)	$24.95	- Demasiado simple - Creación directa de DVD	http://www.magic-video-software.com/magic_video_studio/
uWatchIt - Network Video Monitoring	Gratis	- No soporta muchas cámaras - Reconoce movimiento para capturar pantalla - S.O.Win2000/XP/2003	http://www.sybu.co.za/prod_watchit.php
Move Action	Gratis	- Los archivos de configuración se modifican editándolos - Reconoce movimiento para capturar pantalla - S.O.Win2000/XP	Desarrollado por Wuul
Vitamin D 4.1 Starter	Gratis para una cámara (versión Starter; existen Basic y Pro)	- Detección de movimiento - Detección según reglas definidas por el usuario (región de interés sobre todo) - Motor muy sensible y no admite cambios - Requiere bastante espacio en disco (Starter Ed. Requiere 25GB para 48 horas de video)	http://www.vitamindinc.com
Cámara web Video Diary	Gratis	- Graba videos para clasificarlos con etiquetas de texto - No muy apto para vigilancia	Desarrollado por Mouser
Easy Free Web Cam	Gratis	- Detección de movimiento - Funciona con móviles o PDAs soportadas	Desarrollado por UK Software
Video Surveillance Monitor	$29.95	- Detección de movimiento - Vigilancia remota	http://www.athtek.com/cámara web-surveillance-monitor.html
Blue Iris	$29.95	- Detección de movimiento - Envío de avisos	http://blueirissoftware.com

Producto	Precio más bajo	Características destacables	Proveedor
Cerberus	Gratis	- Detección de movimiento - Monitorización de hasta 10 cámaras	http://www.freshney.org/cerberus/index.htm
Cam Wizard	21.89€	- Detección de movimiento - Envío de avisos - Número ilimitado de fuentes de video, cámaras web, …	http://www.ledset.com/camwiz/index.htm
TeboCam	Gratis	- Detección de movimiento - Envío de avisos - Calibrado de la detección a mano	http://teboweb.com/Cámara web.html
cámara webXP	Gratis para una cámara	- Detección de movimiento - Envío de avisos	http://www.cámara webxp.com/home.aspx
Look Surveillance SDK[16]	$790 licencia SDK + precio por unidad donde se instala la aplicación desarrollada	Este modulo completo de vigilancia tiene como principal característica que se ofrece como un SDK completo que soporta múltiples plataformas y lenguajes de programación.	http://www.neurotechnology.com/verilook-surveillance.html

Tabla 4. Listado de herramientas de vigilancia consultadas durante el estudio previo del mercado.

Las conclusiones que se extrajeron fueron que los sistemas menos completos son los más baratos pero los más limitados, teniendo como característica principal la detección de movimiento (sea cual sea) para realizar capturas de pantalla, pero sin ofrecer mucho más que eso. Por otro lado y como es lógico, sistemas más complejos y que ofrecen en principio un gran catálogo de funcionalidades requieren un desembolso que puede variar de los 20€ (los más sencillos) hasta los 700€ para sistemas completos como el *SDK Look Surveillance de Neurotechnology*.

Este pequeño estudio saca a la luz a su vez que, de entre todas las herramientas consultadas, no existe ninguna que ofrezca de manera simple y directa la detección y reconocimiento de personas, conjugando además esas funcionalidades con el siempre buscado ahorro en espacio y por supuesto con un coste mínimo de dinero, objetivos todos ellos del presente proyecto.

[16] *Software Development Kit.*

C.4 Literatura consultada

Se muestra en la siguiente tabla la lista con los artículos y trabajos académicos más destacados consultados durante la fase inicial de este proyecto.

Título	Autores	Año
Face Recognition - A Literature Survey	R. Chellappa, P. J. Phillips and A. Rosenfeld	2003[17]
An Extended Set of Haar-like Features for Rapid Object Detection	Rainer Lienhart and Jochen Maydt	2002
Rapid Object Detection using a Boosted Cascade of Simple Features	Paul Viola and Michael Jones	2001
Robust Real-time Object Detection	Paul Viola and Michael Jones	2001
Detecting Faces in Images - A Survey	Ming-Hsuan Yang, David J. Kriegman and Narendra Ahuja	2002

Tabla 5. Listado de artículos relacionados con el contexto del proyecto consultados por el autor del proyecto.

Todos los artículos de la lista anterior se encuentran disponibles en la red para libre consulta.

Para reflejar la relevancia de alguno de estos artículos, se muestra en la siguiente tabla el número de citaciones según dos motores de búsqueda e indexación de artículos académicos: *CiteSeer* [12] y *Google* Académico (*Google Scholar*) [13]:

Título	CiteSeer	Google Scholar
Face Recognition - A Literature Survey	518	3585
An Extended Set of Haar-like Features for Rapid Object Detection	224	1399
Rapid Object Detection using a Boosted Cascade of Simple Features	1208	6028
Robust Real-time Object Detection	506	48[18]
Detecting Faces in Images - A Survey	415	2821

Tabla 6. Número de citaciones de varios artículos consultados, según datos de CiteSeer y Google Scholar.

[17] Reedición. Original publicado en el año 2000.

[18] El artículo de los mismos autores con título *Robust real-time face detection* puede confundir al motor de búsqueda. Ese artículo aparece con 5454 citaciones.

Como referencia adicional cabe destacar que con fecha de Septiembre de 2011, el artículo de Paul Viola and Michael Jones *Rapid Object Detection using a Boosted Cascade of Simple Features* se encontraba entre los 100 artículos relacionados con la Ingeniería Informática (*computer science*) más citados desde que CiteSeer almacena esta información (1990).

C.5 Diseño de la interfaz del sistema y resultado final

En esta primera imagen se muestra el primer diseño pensado para el sistema final de detección, donde la imagen real de la cámara web, video o foto se presenta en un cuadro principal de la ventana de la aplicación. La herramienta *Visual Studio* ofrece la posibilidad de obtener este tipo de diseños o *layouts* sin necesidad de introducir código.

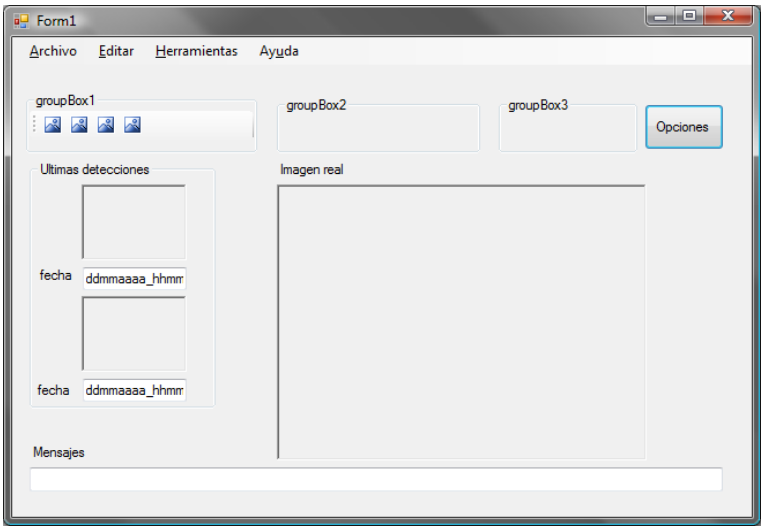


Figura C4. Primer diseño de la ventana principal de la aplicación final. La parte más destacada correspondería con la imagen de la cámara web.

En cuando a la modificación de opciones que se deben mostrar al usuario, se pensó en presentarlas en una ventana aparte (modo *pop-up*), separando los diferentes tipos de configuraciones en pestañas:

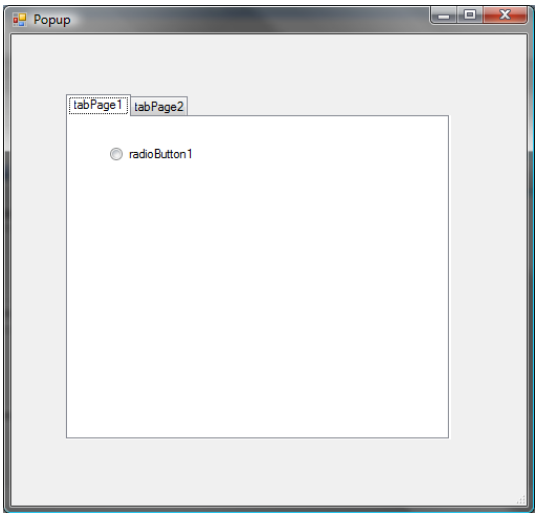


Figura C5. Primer diseño de la ventana de opciones de la aplicación final, accedida desde la ventana principal.

C.5.1 Modificaciones sobre el primer diseño

El gran beneficio que genera el uso de la librería *OpenCV* es que ofrece multitud de funciones y procedimientos que encapsulan tareas muy complejas de tratamiento de imagen. Entre las más utilizadas y llamativas por su sencillez son las relacionadas con el interfaz de usuario. Por ejemplo, mostrar una imagen en una ventana estilo *Windows* requiere de simplemente dos líneas:

```
cvNamedWindow()   // para crear la ventana
cvShowImage()     // para mostrar la imagen
```

Sin embargo, hay que ceñirse a los requisitos de estas funciones. Por ejemplo, no se permite mostrar una imagen con **cvShowImage()** dentro de un control estándar como puede ser un *PictureBox* de un IDE convencional como *Visual C++ Express*.

Estas restricciones implican una ligera modificación de la ventana principal de la aplicación. En vez de tener la imagen en tiempo real de la cámara web en una parte integrada de la ventana principal, esta se mostrará en una ventana adicional.

Se obtendría algo como la imagen siguiente:

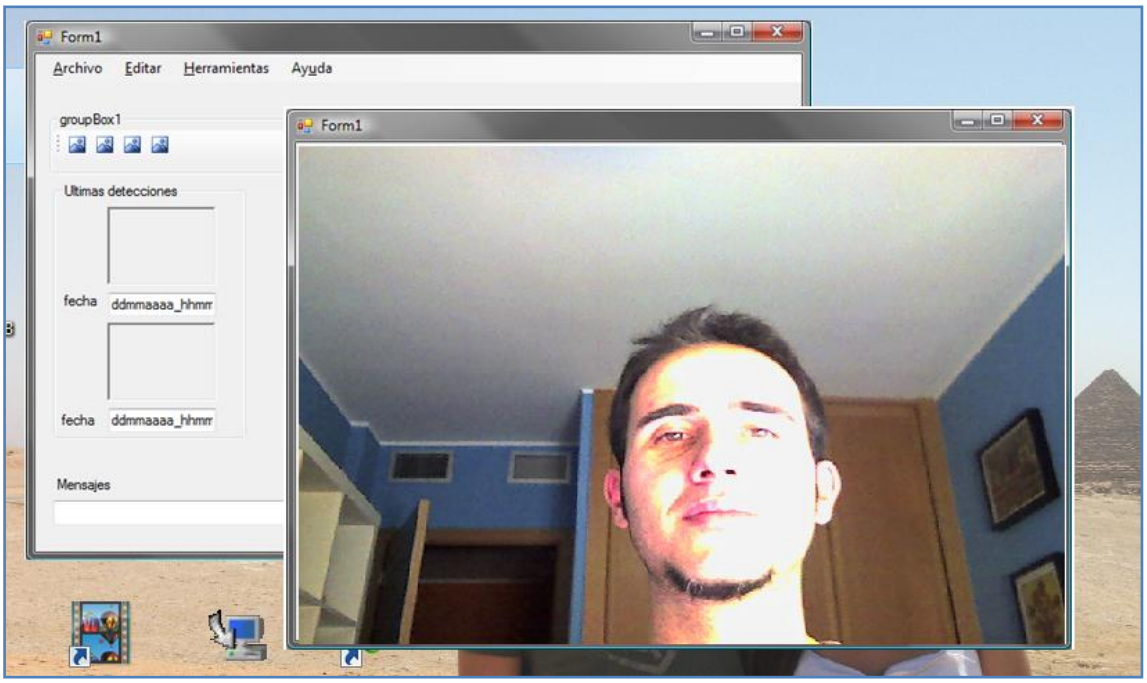


Figura C6. Ventana con imagen de cámara web. En el fondo, ventana principal de la aplicación.

Con este nuevo diseño el usuario consigue:

- Libertad para colocar en el lugar deseado la ventana de la imagen de cámara web sin afectar la ventana principal de la aplicación. Hay que tener en cuenta que la mayor parte del tiempo la cámara web va a estar en funcionamiento pero no resultará imprescindible estar observándola, pues la detección es automática.

- Posibilidad de redimensionar esa ventana, o incluso minimizar en la barra de tareas, sin que se vea afectada la eficacia de la aplicación.

Este tipo de diseño donde de una misma aplicación se obtienen varias ventanas que se pueden modificar libremente (mover, redimensionar, etc.) está ampliamente extendido y aceptado. Aplicaciones como *Winamp* o *Photoshop* son claros ejemplos de esta tendencia.

C.5.2 Primera versión no productiva

En la siguiente imagen se observa el resultado de lo que se diseñó como la primera versión completa del sistema de vigilancia y reconocimiento:

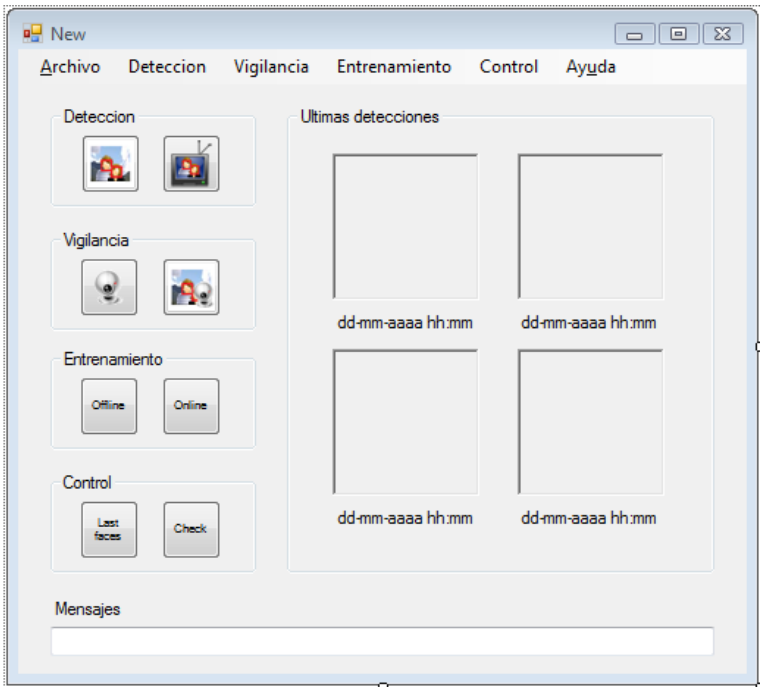


Figura C7. Ventana principal de la primera versión de la aplicación final implementada.

Las diferentes funcionalidades de la aplicación se muestran accesibles desde la barra de menú. Sin embargo, para mayor comodidad, las funcionalidades principales se ofrecen también a través de botones.

C.5.3 Primera versión productiva

La primera version de la aplicación se muestra en las siguientes imágenes. Este resultado definitivo, aunque más ligado a la fase de implementación, se incluye aquí simplemente como referencia.

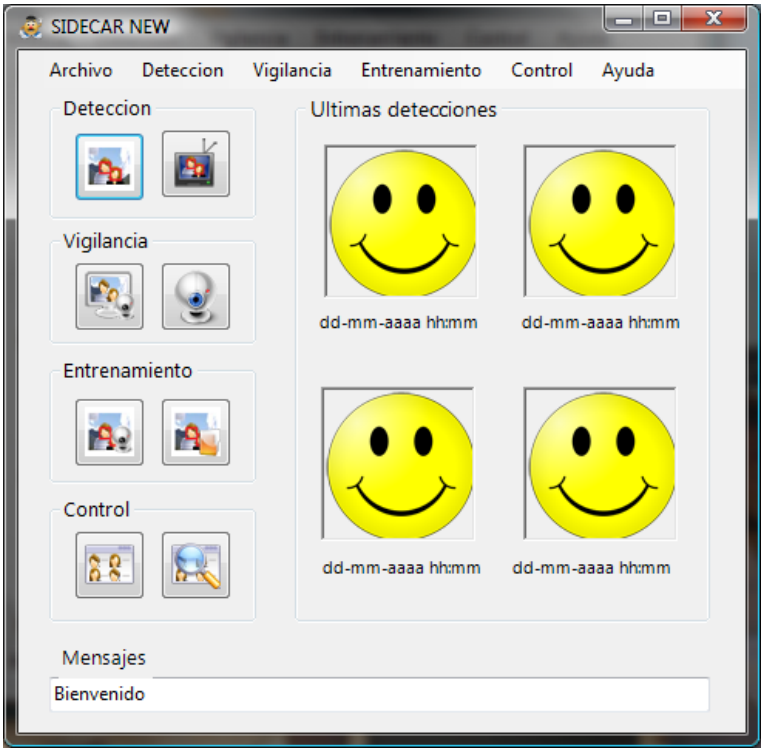


Figura C8. Ventana principal de la primera versión productiva implementada de la aplicación final.

Como se puede observar, el resultado final es fiel reflejo de lo desarrollado en la fase de diseño. También de acuerdo al diseño pensado, las opciones para configurar la aplicación aparecen como una ventana adicional con dos pestañas fácilmente accesibles por el usuario:

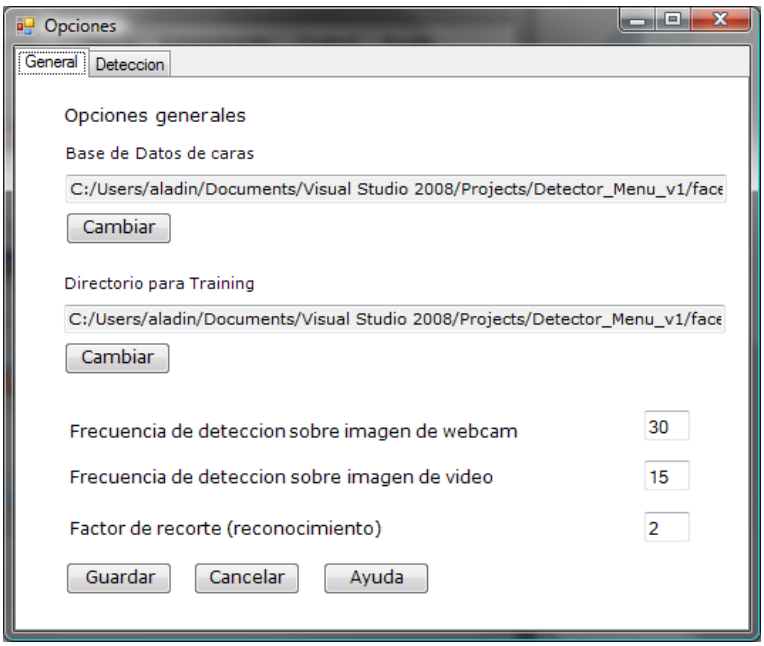


Figura C9. Ventana con las opciones generales de la aplicación. La segunda pestaña muestra las opciones propias del detector.

De esta manera el usuario es capaz de modificar parámetros del detector y de la aplicación en general de una manera muy intuitiva y simple.

Anexo D. Planificación

En este anexo se muestra primeramente la estructura del proyecto descompuesta en tareas, basado en el modelo de gestión de proyectos EDT (acrónimo de Estructura de Descomposición del Trabajo[19]) seguido una vez establecidos los objetivos, para después mostrar el diagrama de Gantt con la estimación de tiempos de cada una de esas tareas y los resultados de dicha estimación.

D.1 Estructura de Descomposición del Trabajo

Una EDT es simplemente una presentación gráfica bien estructurada de un proyecto que incluye el trabajo requerido (en forma de tareas y sub-tareas) para cumplir con los objetivos de éste. Cada nivel descendente de la EDT añade más detalle acerca del trabajo del proyecto.

Los objetivos para diseñar correctamente la EDT deben centrarse en productos del trabajo o entregables, más que en esfuerzos. Basándose en los objetivos de este proyecto, la estructura general de gestión del primer nivel de profundidad de la EDT de este proyecto se estableció como muestra la Figura D1:

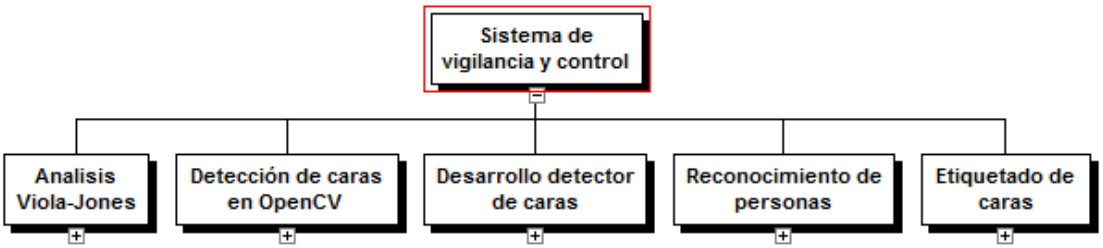


Figura D1: Primer nivel de profundidad de la EDT para obtener un Sistema de vigilancia y control

Las siguientes figuras (Figura D2 hasta Figura D6) ofrecen una visión del resto de niveles de profundidad de cada una de las tareas del primer nivel, descendiendo hasta los elementos finales, también llamados "Paquetes de Trabajo".

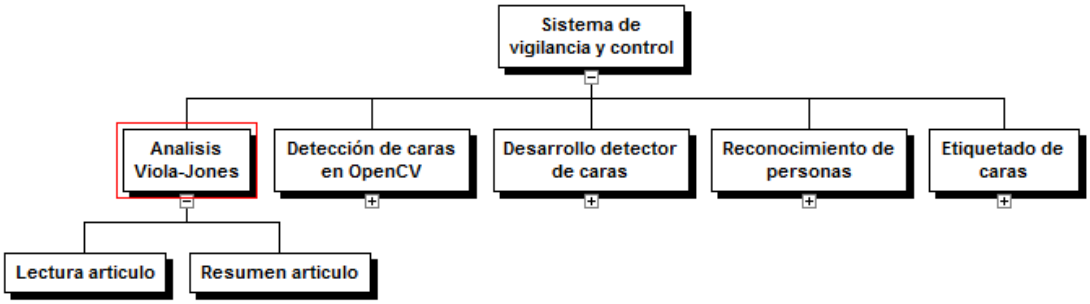


Figura D2: Diagrama EDT de la tarea "Análisis de Viola-Jones" desde la raíz hasta los elementos finales.

[19] También conocido por su nombre en inglés *Work Breakdown Structure* o WBS.

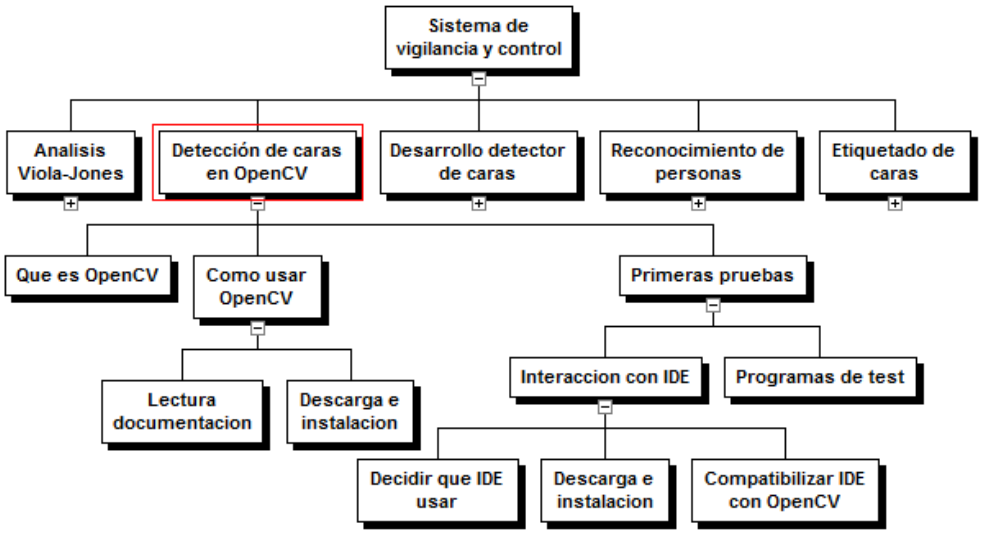


Figura D3: Diagrama EDT de tarea "Detección de caras en OpenCV" desde la raíz hasta los elementos finales.

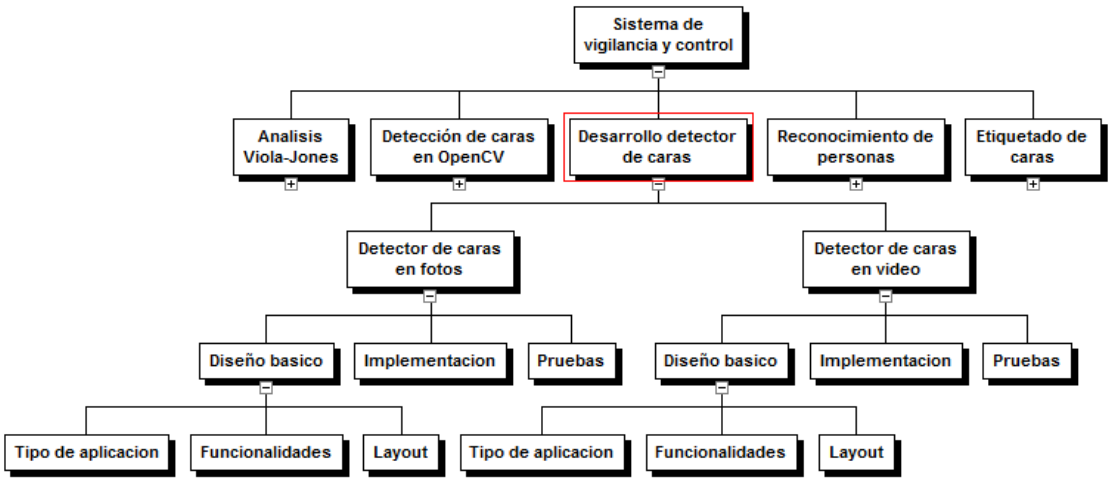


Figura D4: Diagrama EDT de tarea "Desarrollo detector de caras" desde la raíz hasta los elementos finales.

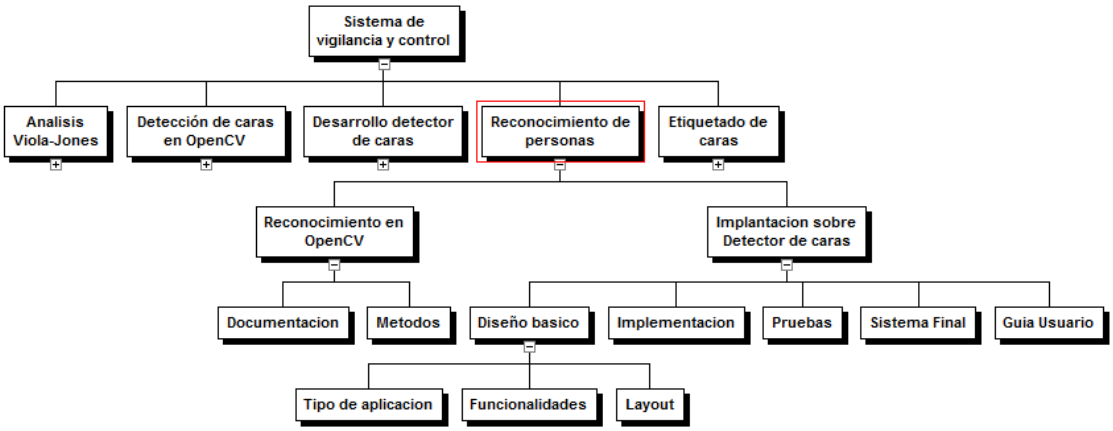


Figura D5: Diagrama EDT de tarea "Reconocimiento de personas" desde la raíz hasta los elementos finales.

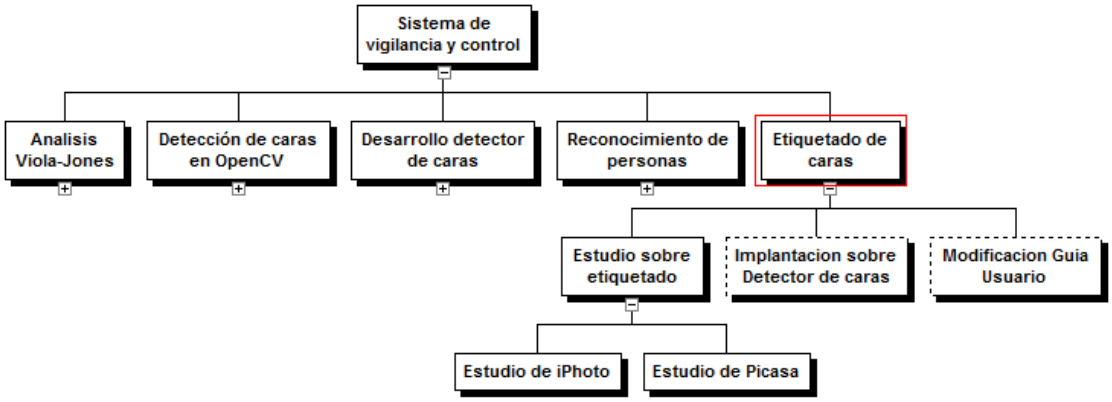


Figura D6: Diagrama EDT de tarea "Etiquetado de caras" desde la raíz hasta los elementos finales. Se marca con borde punteado las tareas opcionales[20]

Asignando una estimación de tiempo a cada una de los elementos finales se obtiene una planificación temporal del proyecto, que se puede representar como un diagrama de Gantt.

D.2 Diagrama de Gantt

El diagrama o gráfica de Gantt tiene como objetivo mostrar de manera simple e intuitiva el tiempo de dedicación previsto para diferentes tareas de un proyecto. Así mismo, también ofrece una manera de indicar relaciones y dependencias existentes entre varias tareas.

A partir de los elementos finales de la EDT del proyecto se obtuvo el diagrama de Gantt de la Figura D7.

[20] Decisión de diseño del proyectando ya que las EDT no definen tareas opcionales según la documentación consultada.

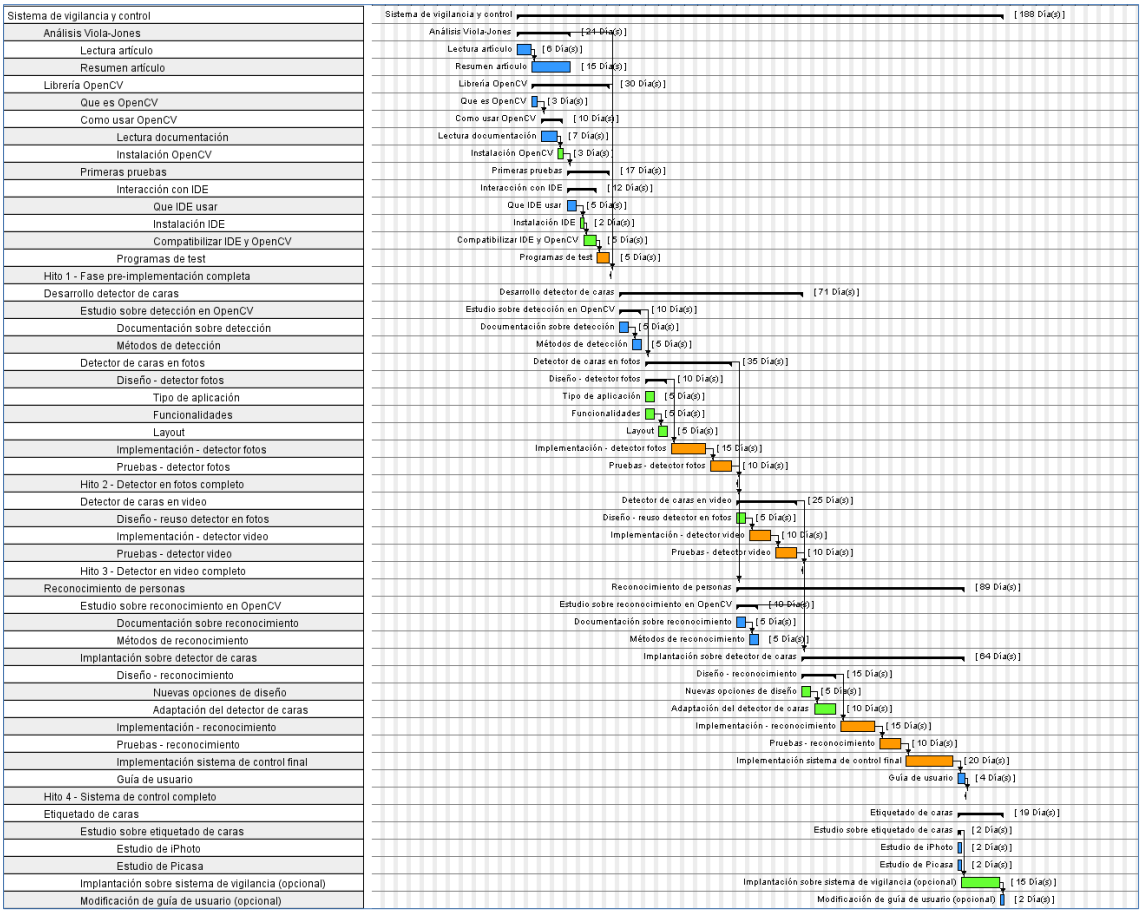


Figura D7: Diagrama de Gantt del sistema de vigilancia y control completo obtenido a partir de la EDT.

Como se observa en la Figura D7, se estimó un tiempo total para la finalización del proyecto de 188 días, calculando unas 2,5 horas de trabajo efectivo por cada día, haciendo un total de 470 horas estimadas.

Cabe destacar que se ha omitido en el diagrama cualquier referencia temporal exacta (mes, año) ya que se pretende reflejar únicamente el tiempo efectivo de trabajo estimado para la ejecución de cada tarea en particular y en consecuencia del proyecto en conjunto. Esto permite evitar que aparezcan en este tipo de diagramas desviaciones temporales que puedan ser debidas a causas totalmente ajenas al desarrollo del proyecto (vacaciones, enfermedad, otros motivos personales, etc.).

D.3 Resultados de la planificación

Si bien es cierto que la descomposición en tareas elementales del proyecto global ayudó al proyectando a seguir unas pautas establecidas a la hora de llevarlas a cabo, no se pudo completar el proyecto en el tiempo total establecido para ello.

Una de las causas del retraso fue que el autor del proyecto decidió sobre la marcha añadir al sistema final diferentes funcionalidades que no se habían pensado a la hora de la planificación, como pueden ser todo lo relacionado con opciones de *debug* que se ofrecen, incluyendo generación y acceso a registros (*logs*). Estas adiciones fueron pensadas para dotar al sistema final de más riqueza y se determinó que compensaba su inclusión a la penalización en tiempo que podían acarrear.

Por otro lado, el proyectando se encontró con varios contratiempos técnicos relacionados con la implementación de la aplicación difíciles de prever, algunos de los cuales aparecen detallados en el Anexo L de este mismo documento.

Según estimaciones del autor, el retraso acumulado al final se aproxima al 10% del total, lo que equivale a unas 45-50 horas más. Como se comenta, es simplemente una estimación porque no se tomó nota de estos retrasos cuando las fases se iban completando, lo que hubiese dado un dato real a la conclusión del proyecto.

Una vez dicho lo anterior, tanto el tiempo extra dedicado en añadir nuevas funcionalidades a la aplicación como el hecho de resolver problemas encontrados en la fase de implementación no han hecho más que contribuir al enriquecimiento personal del proyectando, hecho este que compensa el retraso en la consecución del proyecto final.

Anexo E. *AdaBoost*

AdaBoost, abreviatura de *Adaptive Boost*, mejora adaptable en castellano, es un algoritmo de aprendizaje formulado por Yoav Freund y Robert Schapire. *AdaBoost* es adaptable en el sentido de que los clasificadores obtenidos sucesivamente son mejorados para favorecer los casos clasificados erróneamente por los clasificadores anteriores, es decir, el aprendizaje se concentrará en los ejemplos difíciles de clasificar.

AdaBoost llama a un clasificador débil en varias ocasiones en una serie de rondas. En cada llamada, una distribución de pesos es actualizada, indicando la importancia de los ejemplos en el conjunto de datos para la clasificación. En cada ronda, los pesos de cada ejemplo incorrectamente clasificado se incrementan (o alternativamente, los pesos de cada ejemplo correctamente clasificado se reduce), de modo que el nuevo clasificador se centra más en esos ejemplos erróneos.

A continuación se muestra como referencia un resumen del algoritmo *AdaBoost* para aprendizaje de clasificadores:

- Dadas imágenes de ejemplo $(x_i, y_i),\ldots,(x_n, y_n)$, donde $y_i = 0, 1$ para ejemplos negativos y positivos respectivamente
- Se inicializan pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$, para donde m y l son el número de ejemplos positivos y negativos respectivamente
- Para $t = 1,\ldots,T$:
    1. Normalizar los pesos: $w_{t,i} \leftarrow \frac{wt,i}{\sum_{j=1}^{n} wt,j}$
    2. Para cada característica, j, entrenar un clasificador h_j. El error es evaluado con respecto a w_t, $\epsilon_j = \sum_i wi \mid h_j(x_i) - y_i \mid$
    3. Escoger un clasificador débil, h_t, con el menor error ϵ_t
    4. Actualizar los pesos: $w_{t+1,i} = w_{t,i}\beta_t^{1-ei}$ donde $e_i = 0$ si el ejemplo x_i es clasificado correctamente, $e_i = 1$ en caso contrario, y $\beta_t = \frac{\epsilon t}{1-\epsilon t}$

El clasificador fuerte obtenido es:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t \ h_t\,(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{de lo contrario} \end{cases}$$

Donde $\alpha_t = \log \frac{1}{\beta t}$

Cada ronda del proceso de aprendizaje selecciona una característica de entre 180.000 posibilidades.

Toda la información relativa al proceso general de *boosting* y acerca del algoritmo *AdaBoost* se puede encontrar en la recopilación que hizo la Universidad de Princeton sobre el tema, disponible en:

http://www.cs.princeton.edu/~schapire/boost.html

En esta recopilación se incluyen varios de los artículos presentados por los autores del algoritmo, Yoav Freund y Robert E. Schapire, destacando *A Short Introduction to Boosting* [7], presentado en el *Journal of Japanese Society for Artificial Intelligence* en 1999.

Anexo F. Clasificadores / Detectores

En este anexo se muestra una lista con algunos de los clasificadores disponibles tanto con la instalación por defecto de *OpenCV* como generados por desarrolladores independientes, que ofrecen sus clasificadores para libre uso o bajo licencia de *copyright*.

Descripción	Autor	Tamaño imagen base[21]
Cara frontal (varios)	Rainer Lienhart	24x24 o 20x20
Cara perfil	David Bradley	20x20
Cuerpo humano entero	David Bradley	14x28
Cuerpo humano, parte de arriba	David Bradley	22x18
Cuerpo humano, parte de abajo	David Bradley	19x23
Par de ojos, de frente	Yusuf Bediz	35x16
Ojo derecho	Modesto Castrillón	18x12
Ojo izquierdo	Modesto Castrillón	18x12
Par de ojos, de frente[22]	Modesto Castrillón	22x5
Par de ojos, de frente[23]	Modesto Castrillón	45x11
Cabeza y hombros	Modesto Castrillón	22x20
Boca	Modesto Castrillón	25x15
Nariz	Modesto Castrillón	25x15
Ojo (izquierdo o derecho)	Ting Shan	24x12
Reloj de pared	Celal C. Elgün	30x30

Tabla 7. Clasificadores presentes en la instalación por defecto de la biblioteca OpenCV.

Como se puede observar, el profesor de la Universidad de Las Palmas de Gran Canaria, Modesto Castrillón Santana, es uno de los grandes contribuyentes de *OpenCV* en cuanto a clasificadores de refiere. La versión 2.3 de *OpenCV* proporciona por defecto 7 clasificadores desarrollados por el profesor Castrillón. Debido al uso de sus clasificadores en la aplicación final, se referencia en la sección de bibliografía uno de sus trabajos junto a Hannes Kruppa y Bernt Schiele [15].

[21] En píxeles.
[22] Computado con 700 muestras positivas.
[23] Computado con 7000 muestras positivas.

Anexo G. Pruebas iniciales con *facedetect.exe*

OpenCV ofrece al usuario la posibilidad de probar multitud de aplicaciones ya desarrolladas al incluir en su instalación una gran variedad de archivos ejecutables, entre los que se encuentra un detector de caras llamado **facedetect.exe**.

Existen muchas versiones de código de un detector de caras haciendo uso de *OpenCV*. Cada uno de los códigos es muy similar a los demás debido a que las funciones principales para detectar caras se encuentran disponibles en *OpenCV* en forma de funciones listas para ser llamadas desde cualquier programa implementado. Un ejemplo de código de un detector de caras usando funciones de *OpenCV* se encuentra disponible en la propia página oficial de *OpenCV*, accesible en http://OpenCV.willowgarage.com/wiki/FaceDetection, que incluye además una breve explicación de su funcionamiento.

En el caso de estas primeras pruebas, se ha desarrollado por parte del proyectando otro código ligeramente diferente, intentado que fuese lo más simple y a la vez robusto posible. La simplicidad se ha buscado porque el objetivo principal es demostrar que lo que ofrece *OpenCV*, en concreto para la detección de caras, se puede llevar a cabo de una manera poco compleja.

OpenCV ofrece distintos clasificadores para la detección de diferentes características. Todos ellos están disponibles en el directorio de instalación de *OpenCV* en la ruta *<directorio instalación de OpenCV>\data\haarcascades*.

Hay que mencionar que el clasificador que mejor funciona es el que detecta caras de frente. Esto se debe sencillamente a que una cara de frente tiene dos patrones muy claros fácilmente expresables en forma de rectángulos, que son:

- Plano horizontal: zona de ojos y cejas más oscura que las mejillas.
- Plano vertical: zona de ojos y cejas más oscura que el puente de la nariz.

Figura G1. Patrones en forma de dos o tres rectángulos respectivamente reconocibles para un rostro humano

Estas dos características tan determinantes no existen en ningún otro aspecto de nuestra anatomía y dificultan sobremanera la detección de esas otras partes. Por este motivo estas pruebas se han realizado utilizando únicamente el clasificador de detección de caras de frente.

La ejecución del programa se debe realizar parametrizando la llamada con el archivo que contiene el clasificador deseado:

- `facedetect --cascade="C:/Archivos de Programa/OpenCV/data/haarcascades/haarcascade_frontalface_alt2.xml"`

Si además se precisa analizar una foto en concreto y no la que referencia por defecto el programa, también hay que pasar en la llamada dicha foto, incluyendo si es necesario la ruta completa o relativa. Por ejemplo, si la foto a analizar se encuentra en el directorio *C:\Archivos de programa\OpenCV\data\fotos* con el nombre de **foto.jpg**, la llamada al programa debe hacerse así:

- `facedetect --cascade="C:/Archivos de Programa/OpenCV/data/haarcascades/haarcascade_frontalface_alt2.xml" foto.jpg`

Si la foto a analizar no se encuentra en el mismo directorio que la foto, la llamada al programa debe hacerse así:

- `facedetect --cascade="C:/Archivos de Programa/OpenCV/data/haarcascades/haarcascade_frontalface_alt2.xml" "C:/Archivos de Programa/OpenCV/data/fotos/foto.jpg"`

G.1 Muestras

Para las pruebas se han usado diferentes imágenes propias de tamaño aproximado de 640x480. El formato usado es JPG. Se ha usado el programa *The GIMP Portable* para la conversión de fotos a distintos tamaños según requería la prueba.

La primera parte de las pruebas (fotos 1 a 5) la componen fotos más o menos sencillas, sin añadir más dificultad que la propia de la posición de la persona y la distancia a la cámara.

La segunda parte contiene fotos en principio más exigentes para el detector, como puede ser una foto con personas de piel oscura, donde los rasgos diferenciales del rostro humano mencionados anteriormente (puente de la nariz, mejillas) no son tan contrastados.

G.2 Pruebas de eficacia

La primera parte de las pruebas se centró en el estudio de la eficacia del detector, es decir, la capacidad del detector de encontrar todas las caras presentes en una imagen.

Para estas pruebas, se usaron los parámetros del detector siguientes:

- Factor de escala: 1.1.
- Número de vecinos: 2.
- Modo: ninguno.
- Tamaño inicial ventana de escaneo: 20x20 píxeles.

Los resultados, mostrados a continuación, dejan patente el buen funcionamiento del detector en casi todas las situaciones, sin dejar de plasmar las carencias propias de un entrenamiento concreto. Al pie de cada foto aparece una nota breve acerca del resultado obtenido.

Foto 1 - Resultado

Figura G2. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 1.

En esta foto se detectan perfectamente las dos caras de las niñas. Primer test de todos satisfactorio. Sin embargo, el fondo claro favorece la detección.

Foto 2 - Resultado

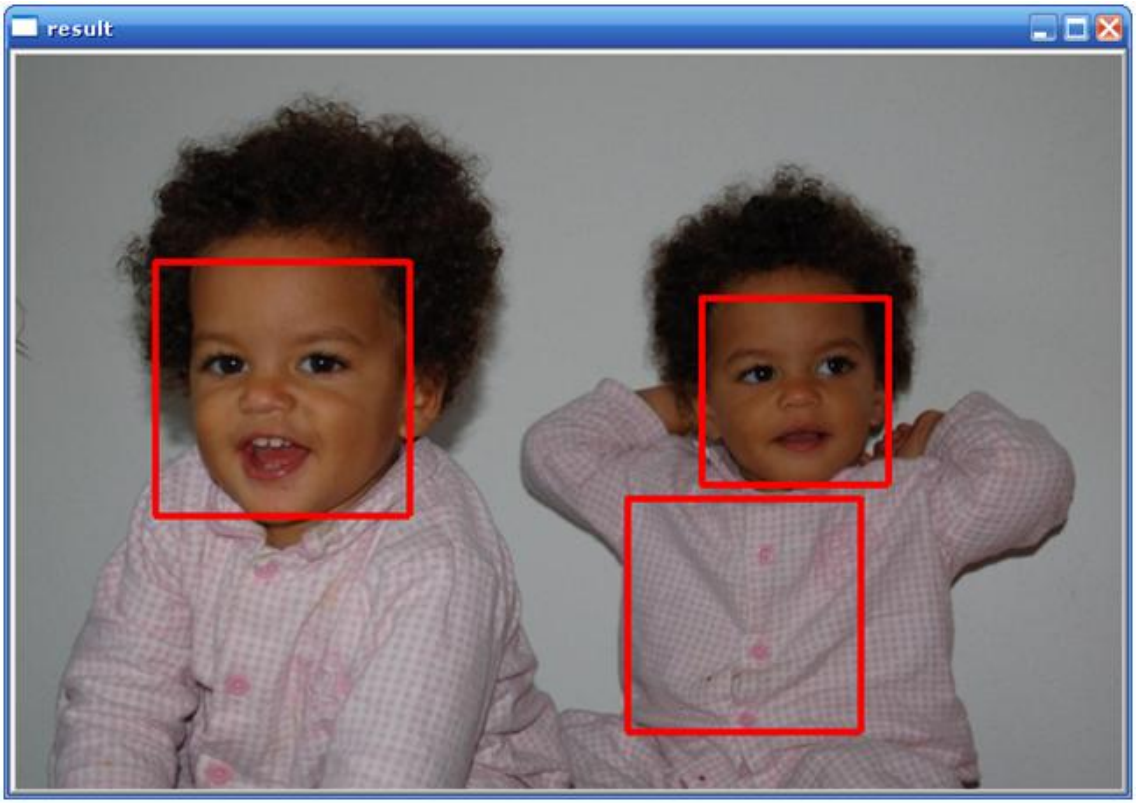


Figura G3. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 2.

Aparece un falso positivo que comprende la zona del torso de la niña de la derecha. Para solucionar esta detección incorrecta, habría que añadir al set de negativos el trozo que ha dado positivo y volver a entrenar al sistema. De esa forma, en el siguiente análisis, la zona marcada ahora como positiva no debería aparecer.

Por otro lado, la misma imagen evaluada con el mismo detector y cambiando solamente el parámetro de escala de 1.1 a 1.2 no detecta este falso positivo.

Foto 3 - Resultado

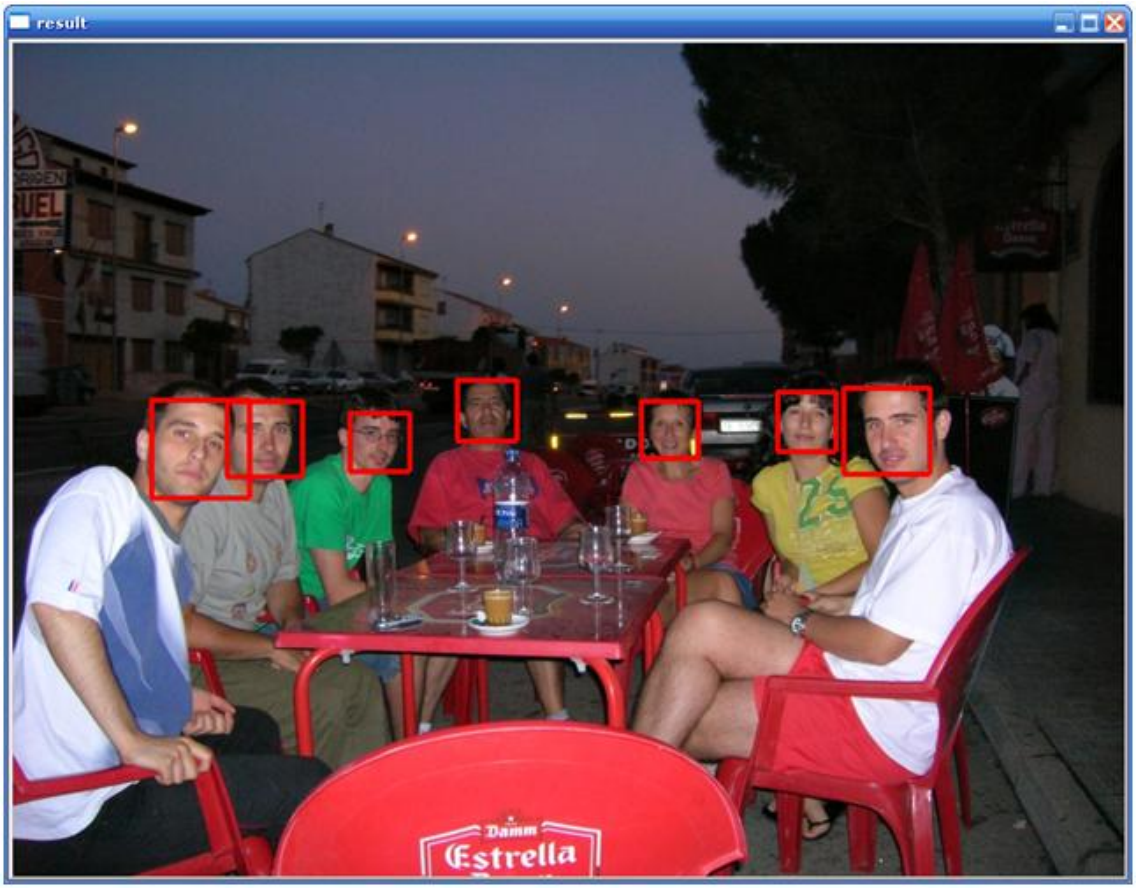


Figura G4. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 3.

Todas las caras son detectadas perfectamente incluso estando a diferentes distancias. No aparecen falsos positivos.

Foto 4 - Resultado

Figura G5. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 4.

En esta foto aparecen dos falsos negativos. Estos se deben a que una de las caras (la de la izquierda) solamente aparece de perfil, mientras que el error en la otra es debido a la no alineación de las caras horizontalmente, ya que el conjunto de muestras positivas del entrenamiento está formado únicamente por caras alineadas horizontalmente.

Foto 4b - Resultado

Figura G6. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 4b.

La foto de esta prueba, llamada 4b, es igual a la anterior pero rotada unos pocos grados de manera que la cara de la derecha aparezca alineada horizontalmente. De esta forma, el detector sí que encuentra la cara que antes no había detectado debido a la no alineación horizontal de los ojos.

Foto 5 - Resultado

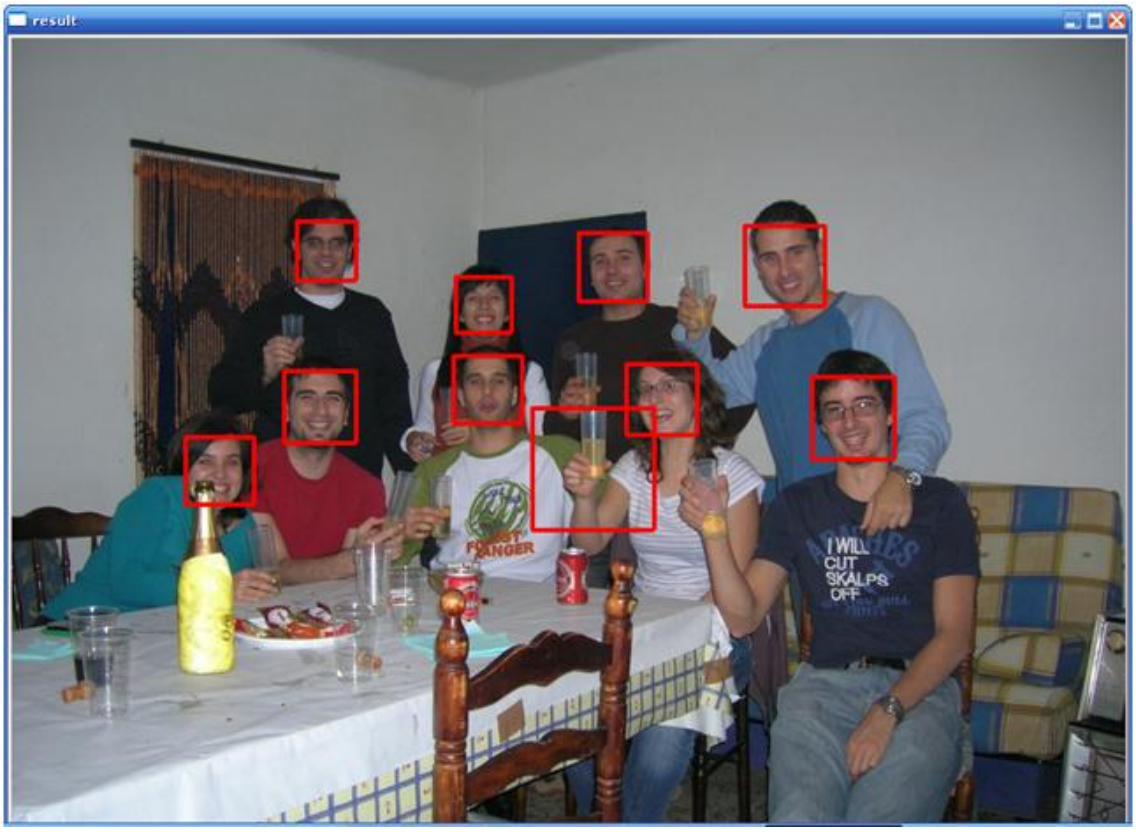


Figura G7. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 5.

En esta foto aparece un falso positivo. Una vez más, para solucionar este fallo habría que añadir al set de negativos la parte de la foto que ha dado positivo y volver a entrenar al sistema.

Foto 6 - Resultado

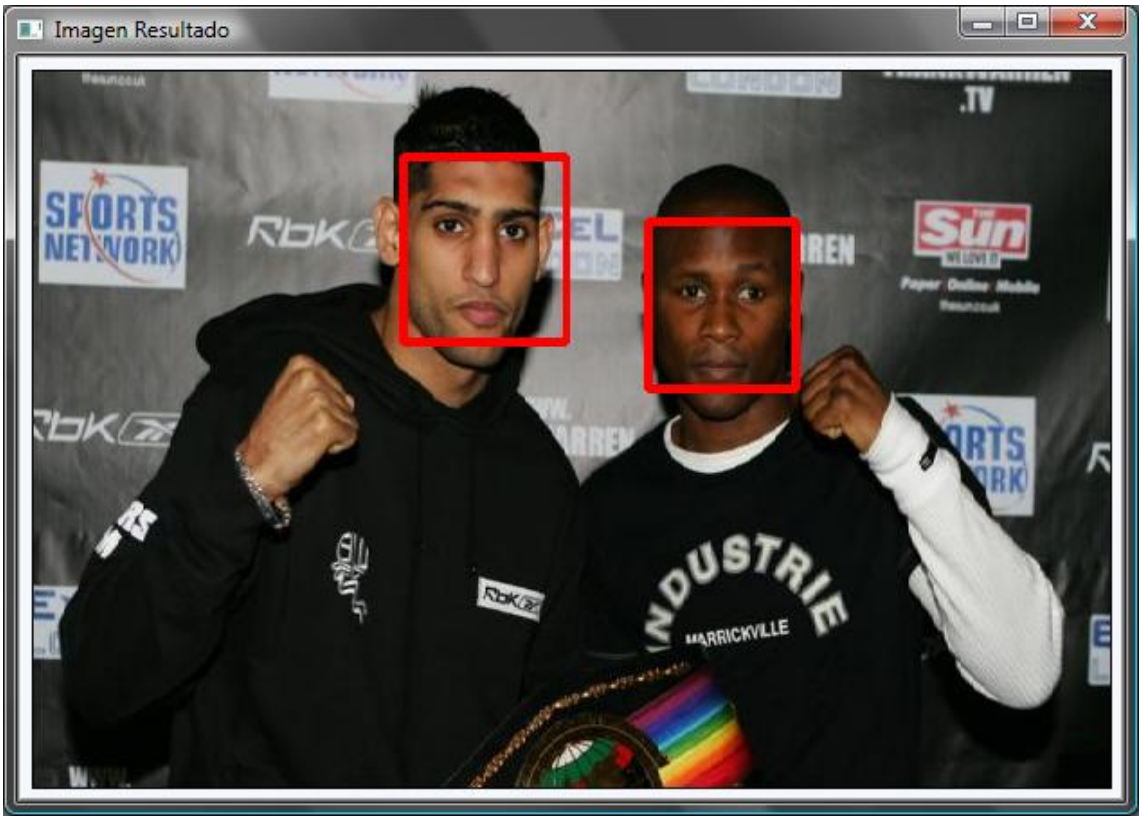


Figura G8. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 6.

A pesar del fondo oscuro y del tono de piel de ambas personas, el detector funciona correctamente.

Foto 7 - Resultado

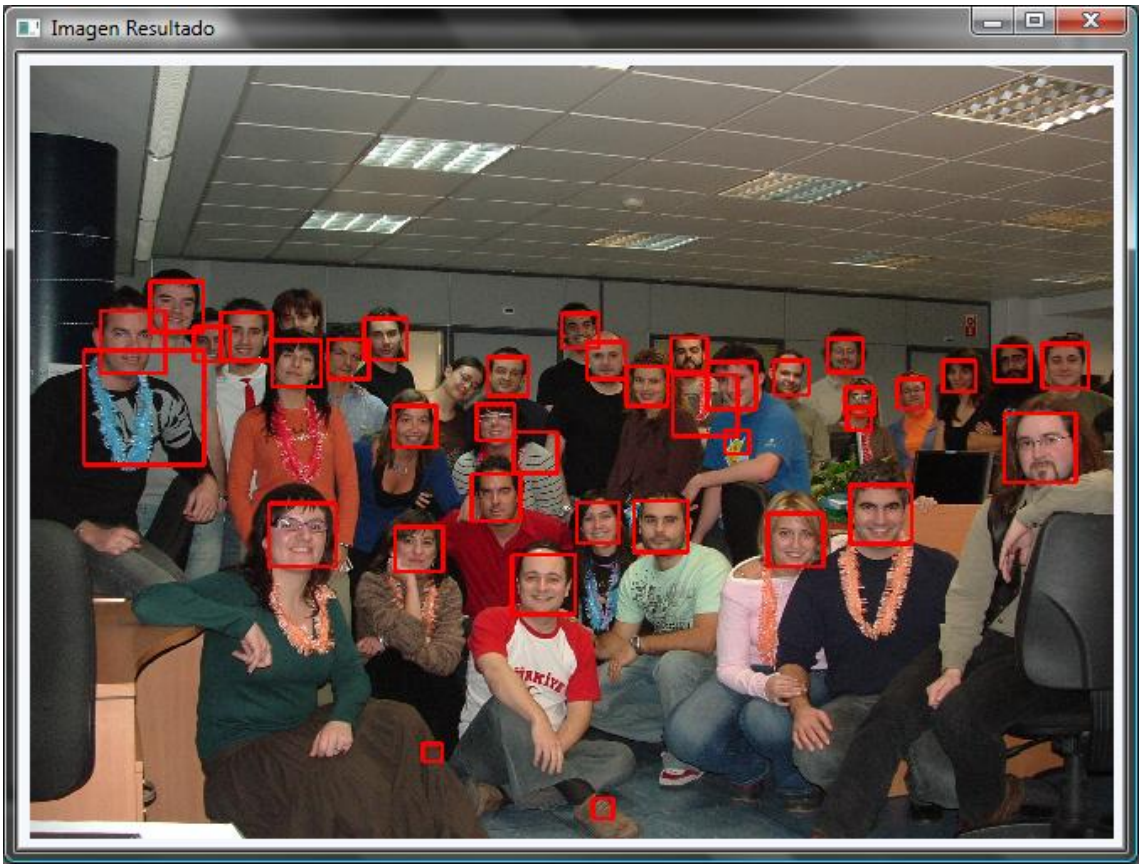


Figura G9. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 7.

Un número alto de personas no supone un problema para el detector salvo por el hecho de que la variedad de colores eleva el número de falsos positivos y, con los parámetros de prueba del detector, aparece un falso negativo no esperado (el falso negativo de la cara de perfil era esperado).

Esta misma foto, procesada con ligeras variaciones en las propiedades del detector no presenta falsos positivos y solamente deja de reconocer la persona de perfil (ver Figura 1, página 20 de este mismo documento).

Foto 8 - Resultado

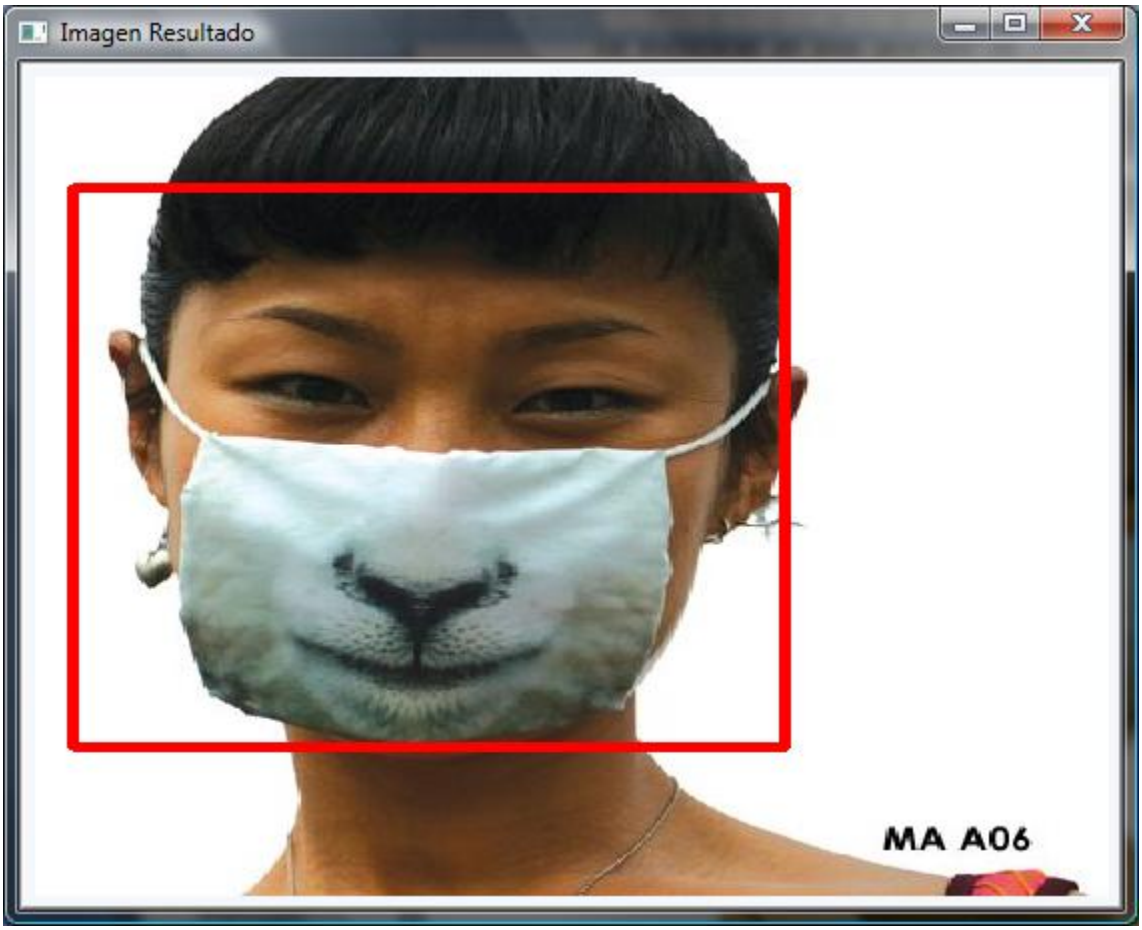


Figura G10. Captura de pantalla con la imagen resultado de la prueba de detección de caras sobre la foto 8.

Una cara humana con la parte de arriba bien diferenciada es reconocida a la perfección a pesar de que toda la parte de abajo, desde la mitad de la nariz hasta el cuello, esté escondida tras una máscara.

Pruebas de rendimiento (velocidad de detección)

Además de la eficacia del detector, otro factor determinante a tener en cuenta en el ámbito de este proyecto es la velocidad de las detecciones: un detector muy bueno pero no lo suficientemente rápido no era útil.

Para medir velocidad del detector, el autor de este proyecto tuvo que desarrollar una pequeña aplicación propia en la que, haciendo uso del mismo detector que en las pruebas anteriores sobre eficacia, se determinara el tiempo que se tarda en comprobar las fotos y obtener las caras.

En estas pruebas los parámetros de detección se ajustaron a los valores recomendados para realizar una detección rápida, que son:

- Factor de escala: 1.2.

- Número de vecinos: 2.
- Modo: *Canny Prunning*.
- Tamaño inicial ventana de escaneo: 30x30 píxeles.

Los resultados de las pruebas, realizadas en un ordenador con procesador AMD *Dual-Core* a 2GHz y almacenados automáticamente en un fichero CSV, se presentan a continuación:

Foto	Tamaño[24]	Número de caras detectadas	Tiempo[25]
1	860x572	2	733
2	640x425	2	421
3	840x630	7	749
4	640x480	0	374
4b	640x480	1	405
5	640x480	10	375
6	682x400	2	468
7	1280x960	35	1794
8	433x420	1	140

Tabla 8. Resultados de las pruebas realizadas con los tiempos de ejecución para cada una de las fotos.

Como se puede observar, el tiempo de total de escaneo de una imagen no supera en ningún caso el medio segundo cuando el tamaño de la imagen es de 640x480 píxeles, que es la medida de un formato de vídeo o cámara web estándar. Este dato revela que el detector sería capaz de escanear imágenes de video o cámara web aproximadamente 2 veces por segundo de manera precisa ya que, si comparamos el número de caras detectadas en cada caso con los resultados sobre eficacia mostrados anteriormente, el detector sigue siendo capaz de detectar prácticamente todas las caras que detectaba anteriormente. Además, solamente en el caso de la imagen de 1280x960, donde se detectan nada menos que 35 caras, el detector tarda más de 1 segundo en analizar la imagen.

[24] En píxeles.
[25] En milisegundos.

Anexo H. Método *Eigenfaces*

En este anexo se profundiza un poco más en la parte teórica del método de *Eigenfaces* utilizado en este proyecto para las tareas de reconocimiento de personas. Este método, como se comenta en la parte principal de la memoria, se basa en el Análisis de Componentes Principales también llamado ACP o PCA por su nombre en inglés (*Principal Component Analysis*).

H.1 ¿Por qué ACP o reducción de dimensión?

En un conjunto de datos, cada uno con unas determinadas características medibles, la información de interés tiene a menudo una dimensión mucho menor que el número total de mediciones. En el caso de una imagen, el valor de cada píxel puede considerarse como una medición. Es decir, por cada imagen de 100x100 píxeles se tienen 10000 mediciones. Esto lleva a pensar que lo más probable es que de alguna manera se puedan distinguir las caras de diferentes personas con un número de mediciones menor que 10000. De antemano no se pretende conocer ese número, pero es muy probable que sea mucho menor que el número de píxeles.

Si esta suposición es correcta, al comparar imágenes entre sí píxel a píxel para poder diferenciarlas, la suma de todas las diferencias de píxeles crearía un ruido (información no útil) que es extremadamente alto en comparación con la información realmente útil. Si se elimina ese ruido, se pueden conseguir resultados similares con mucho menos esfuerzo.

Para reducir la dimensión de los datos, se sigue la técnica denominada Análisis de Componentes Principales, ya mencionada en la parte principal de esta memoria.

H.2 Número máximo de componentes principales

Una vez aceptado el supuesto de que se pueden encontrar un número de mediciones menor que el inicial para diferenciar datos entre sí (denominados a partir de ahora componentes principales debido a la técnica utilizada para su obtención), parece preciso encontrar cual puede ser ese número.

El número de componentes principales a encontrar está limitado por el número de puntos de datos. Si se piensa en un conjunto de datos que consta de un solo punto, no existe un componente principal que establezca la separación máxima con otro ya que no hay nada que separar. En un conjunto de datos con sólo dos puntos, la línea que conecta estos dos puntos es el primer componente principal. Pero no hay segundo componente principal, porque no hay nada más que separar: los dos puntos pertenecen a la línea. Extendiendo esta fórmula, tres puntos definen un plano, que es un objeto de dos dimensiones, por lo que un conjunto de datos con tres puntos de datos no puede tener más de dos componentes principales. Y así sucesivamente.

En el método de *Eigenfaces*, cada cara de 100x100 se trata como dato simple (en un espacio de dimensión 10000), por lo que el número de componentes principales a encontrar nunca será mayor que el número de imágenes de caras del conjunto menos uno.

H.3 Componentes principales como vectores propios (*eigenvectors*)

Al obtener los componentes principales, realmente se obtienen los vectores propios (o *eigenvectors* en inglés) de la matriz de covarianzas asociada al conjunto de imágenes de entrenamiento. Aunque los vectores propios son un concepto de álgebra lineal, en el contexto de

este método y por consiguiente de este proyecto, simplemente se entienden como un nombre alternativo para los componentes principales.

En un espacio vectorial de dimensión 10000 (imágenes de 100x100 píxeles), los vectores propios se pueden representar en forma de imágenes haciendo que sus componentes sean la luminosidad de cada píxel, es decir, directamente su valor. De esta forma, cada vector propio aparece representado como una imagen de una cara (más o menos borrosa) que se denominan *eigenfaces*, término que da nombre al método.

La Figura H1 sirve para ilustrar el aspecto que tienen las *eigenfaces*.

Figura H1. Ejemplo de 8 eigenfaces obtenidas con la aplicación tras la fase de entrenamiento.

H.4 Limitaciones

Como cualquier metodología, *Eigenfaces* presenta varias limitaciones:

- Depende de la posición de las caras en la imagen: si las caras se desplazan, puede que no reconozca a la persona.
- Depende de la escala de las caras: si las caras se escalan a diferentes tamaños, puede que no reconozca a la persona.
- Depende del fondo de la imagen tras las caras: diferente fondo puede confundir al reconocedor.
- Depende de la rotación de las caras: distinta rotación de las caras puede confundir al reconocedor.
- Depende de la luminosidad de la imagen: diferente brillo o intensidad en la imagen hacen que el reconocedor no funcione correctamente.

Es relevante entender que todas estas limitaciones vienen dadas por el hecho de que la base de este algoritmo es la comparación directa del valor (en escala de grises) de cada pixel de la imagen de prueba con su correspondiente pixel (situado exactamente en la misma posición) en cada una de las imágenes de entrenamiento.

Cada una de las limitaciones se puede compensar para minimizar su impacto en el reconocedor. Por ejemplo, para evitar que el fondo de la imagen incida negativamente en el resultado, se puede recortar (*crop* en inglés) la imagen de la cara para dejar solamente lo más reconocible de ella, dejando incluso fuera del análisis el pelo y las orejas. Así mismo, el hecho

de que el detector de caras usado en la captura de caras fuese entrenado con personas de frente y su cara alineada horizontalmente, minimiza el error por posibles rotaciones. En cuanto al problema derivado de la diferente intensidad de la luz en la imagen, la ecualización del histograma (maximizar el contraste de una imagen sin perder información estructural) puede en algunos casos minimizar su impacto y mejorar el rendimiento del reconocedor.

En la aplicación desarrollada en este proyecto, se realiza ecualización del histograma de cada imagen de entrenamiento y de comprobación. Además, la posibilidad de añadir recorte adicional a las caras detectadas se estudió exhaustivamente aunque al final se decidió no incluirla en la versión definitiva.

Anexo I. Pruebas de *Eigenfaces* con *OnlineFaceRec.exe*

El presente anexo detalla las pruebas realizadas por el autor del proyecto para evaluar la validez de un reconocedor de caras basado en el método de *Eigenfaces*. Se usó la aplicación llamada **OnlineFaceRec.exe** desarrollada por Shervin Emami, distribuida de manera gratuita por el autor.

I.1 Entorno

El entorno de pruebas tenía las siguientes características:

- HP Pavilion dv5.
- CPU AMD Turion X2 Dual-Core 2GHz.
- 4GB RAM.
- Windows Vista Home Premium SP2 32 bits.
- OpenCV 2.3.
- Visual Express C++ 2008.

I.2 Prueba de reconocimiento *offline*

Aunque **OnlineFaceRec.exe** ofrece la posibilidad de realizar tareas de reconocimiento directamente a través de la cámara web, el uso de esta funcionalidad requería modificaciones y nueva compilación que se consideraron innecesarias para las pruebas a realizar. Por lo tanto, las pruebas se llevaron a cabo haciendo uso del reconocimiento denominado *offline* (fotos en vez de imagen de cámara web).

El archivo ejecutable **OnlineFaceRec.exe** se puede usar a través de la línea de comandos de MS-DOS. Uso:

- OnlineFaceRec [<command>] // sin argumentos se ejecuta la detección online a través de cámara web

Los comandos válidos son:

- train <train_file>
- test <test_file>

Para las pruebas se usan las fotos de la base de datos de caras de ORL (*Olivetti Research Lab's*) descargable desde http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html[26].

I.2.1 Prueba 1

Entrenamiento: 10 fotos de 2 de las personas de la base de datos. Fichero ***train-10x2.txt***.

[26] Cortesía de *AT&T Laboratories Cambridge.*

Test: 8 fotos de las 20 usadas en el entrenamiento.
Resultado:

```
8 test faces loaded
Training data loaded (20 training images of 2 people):
People: <Person1>, <Person2>.
nearest = 1, Truth = 1 (Correct). Confidence = 1.000000
nearest = 1, Truth = 1 (Correct). Confidence = 1.000000
nearest = 1, Truth = 1 (Correct). Confidence = 1.000000
nearest = 1, Truth = 1 (Correct). Confidence = 1.000000
nearest = 2, Truth = 2 (Correct). Confidence = 1.000000
nearest = 2, Truth = 2 (Correct). Confidence = 1.000000
nearest = 2, Truth = 2 (Correct). Confidence = 1.000000
nearest = 2, Truth = 2 (Correct). Confidence = 1.000000
TOTAL ACCURACY: 100% out of 8 tests.
TOTAL TIME: 4.2ms average.
```

Conclusión: resultado esperado. El 100% de precisión se debe a que las fotos del test forman parte del entrenamiento.

I.2.2 Prueba 2

Entrenamiento: 1 foto de cada una de las 40 personas en la base de datos. Fichero ***train-1x40.txt***.
Test: 4 fotos de las personas 1 y 2. Ninguna de las 8 fotos se ha usado en el entrenamiento.
Resultado:

```
8 test faces loaded
Training data loaded (40 training images of 40 people):
People: <Person1>, <Person2>, …, <Person40>.
nearest = 18, Truth = 1 (WRONG!). Confidence = 0.674366
nearest = 1, Truth = 1 (Correct). Confidence = 0.657570
nearest = 1, Truth = 1 (Correct). Confidence = 0.662632
nearest = 18, Truth = 1 (WRONG!). Confidence = 0.652984
nearest = 2, Truth = 2 (Correct). Confidence = 0.817309
nearest = 2, Truth = 2 (Correct). Confidence = 0.713989
nearest = 2, Truth = 2 (Correct). Confidence = 0.783370
nearest = 2, Truth = 2 (Correct). Confidence = 0.778284
TOTAL ACCURACY: 75% out of 8 tests.
TOTAL TIME: 6.5ms average.
```

Conclusión: resultado mejor de lo esperado. Con solamente una foto por persona, es capaz de acertar el 75% de las veces. Otra conclusión que se extrae es que la foto del entrenamiento de la persona 18 (foto 1) se debe parecer a la persona 1, fotos 5 y 9 de la base de datos que han sido los dos errores del test. Esto se comprueba con las fotos reales de la base de datos.

Figura I1. De izquierda a derecha, persona 18 foto 1, persona 1 fotos 5 y 9, e imagen de entrenamiento de la persona 1.

I.2.3 Prueba 3

Entrenamiento: 5 fotos de cada una de las 40 personas en la base de datos. Fichero ***train-5x40.txt***. El tiempo del entrenamiento deja de ser inmediato (20' aprox.).
Test: 4 fotos de las personas 1 y 2. Ninguna de las 8 fotos se ha usado en el entrenamiento.
Resultado:

```
8 test faces loaded
Training data loaded (200 training images of 40 people):
People: <Person1>, <Person2>, …, <Person40>.
nearest = 1, Truth = 1 (Correct). Confidence = 0.939434
nearest = 1, Truth = 1 (Correct). Confidence = 0.941012
nearest = 1, Truth = 1 (Correct). Confidence = 0.928038
nearest = 1, Truth = 1 (Correct). Confidence = 0.936846
nearest = 2, Truth = 2 (Correct). Confidence = 0.955264
nearest = 2, Truth = 2 (Correct). Confidence = 0.959605
nearest = 2, Truth = 2 (Correct). Confidence = 0.961705
nearest = 2, Truth = 2 (Correct). Confidence = 0.961310
TOTAL ACCURACY: 100% out of 8 tests.
TOTAL TIME: 26.5ms average.
```

Conclusión: Resultados de 100% en acierto y rondando el 95% en cuanto a precisión aunque cabe destacar que el autor de la herramienta advierte que el cálculo de la confianza o efectividad no tiene ninguna base científica, no es fiable y puede llevar a confusiones.

I.2.4 Prueba 4

Entrenamiento: 9 fotos de cada una de las 40 personas en la base de datos. Fichero ***train-5x40.txt***. Se deja una sin usar por individuo para poder probar su resultado. El tiempo del entrenamiento es de 1 minuto aproximadamente.
Test: 1 fotos de cada una de las 40 personas. Ninguna foto se ha usado en el entrenamiento.
Resultado:

```
40 test faces loaded
Training data loaded (360 training images of 40 people):
People: <Person1>, <Person2>, …, <Person40>.
nearest = 1, Truth = 1 (Correct). Confidence = 0.962164
nearest = 2, Truth = 2 (Correct). Confidence = 0.978516
nearest = 3, Truth = 3 (Correct). Confidence = 0.980417
nearest = 4, Truth = 4 (Correct). Confidence = 0.976992
nearest = 40, Truth = 5 (WRONG!). Confidence = 0.973394
nearest = 6, Truth = 6 (Correct). Confidence = 0.985300
nearest = 7, Truth = 7 (Correct). Confidence = 0.970698
nearest = 8, Truth = 8 (Correct). Confidence = 0.977743
nearest = 9, Truth = 9 (Correct). Confidence = 0.972168
nearest = 38, Truth = 10 (WRONG!). Confidence = 0.960090
nearest = 11, Truth = 11 (Correct). Confidence = 0.972743
nearest = 12, Truth = 12 (Correct). Confidence = 0.979667
nearest = 13, Truth = 13 (Correct). Confidence = 0.984157
nearest = 14, Truth = 14 (Correct). Confidence = 0.963569
nearest = 15, Truth = 15 (Correct). Confidence = 0.981401
nearest = 16, Truth = 16 (Correct). Confidence = 0.973172
nearest = 17, Truth = 17 (Correct). Confidence = 0.979135
nearest = 18, Truth = 18 (Correct). Confidence = 0.974709
nearest = 19, Truth = 19 (Correct). Confidence = 0.979579
nearest = 20, Truth = 20 (Correct). Confidence = 0.979816
nearest = 21, Truth = 21 (Correct). Confidence = 0.979469
nearest = 22, Truth = 22 (Correct). Confidence = 0.978380
nearest = 23, Truth = 23 (Correct). Confidence = 0.979187
nearest = 24, Truth = 24 (Correct). Confidence = 0.977105
```

```
nearest = 25, Truth = 25 (Correct). Confidence = 0.980347
nearest = 26, Truth = 26 (Correct). Confidence = 0.986203
nearest = 27, Truth = 27 (Correct). Confidence = 0.972211
nearest = 28, Truth = 28 (Correct). Confidence = 0.969525
nearest = 29, Truth = 29 (Correct). Confidence = 0.976103
nearest = 30, Truth = 30 (Correct). Confidence = 0.981944
nearest = 31, Truth = 31 (Correct). Confidence = 0.974777
nearest = 32, Truth = 32 (Correct). Confidence = 0.975986
nearest = 33, Truth = 33 (Correct). Confidence = 0.991578
nearest = 34, Truth = 34 (Correct). Confidence = 0.984067
nearest = 35, Truth = 35 (Correct). Confidence = 0.969399
nearest = 36, Truth = 36 (Correct). Confidence = 0.964676
nearest = 37, Truth = 37 (Correct). Confidence = 0.978118
nearest = 38, Truth = 38 (Correct). Confidence = 0.982136
nearest = 39, Truth = 39 (Correct). Confidence = 0.972386
nearest = 5, Truth = 40 (WRONG!). Confidence = 0.972871
TOTAL ACCURACY: 92% out of 40 tests.
TOTAL TIME: 50.1ms average.
```

Conclusión: Resultados de 92% en acierto. Los fallos en el primer y último caso (confusión persona 5 con 40) se deben al parecido relativo de las personas en cuestión.

Figura I2. Arriba, persona 40 fotos 1 a 5. Abajo, persona 5 fotos 1 a 5.

El otro fallo es menos explicable, salvo por el hecho de que la persona 10 en la foto 10 mira de frente con la cabeza alta y las del entrenamiento suyas tienen la mirada un poco más agachada.

I.3 Conclusión final

De las pruebas realizadas se concluyen dos aspectos importantes. Por un lado, el reconocedor de caras con *Eigenfaces* ofrece, como se preveía, resultados bastante aceptables en poco tiempo y es un buen candidato para realizar las tareas de reconocimiento en el proyecto.

Por otro lado y mucho menos relevante, cabe destacar que el valor de confianza que genera la herramienta utilizada no es del todo fiable, ya que por ejemplo en la prueba 4 hay errores cuyo valor de confianza excede el 97%. Esto debería motivar un estudio de alternativas para mostrar la confianza de un reconocimiento en caso de que sea necesario.

Anexo J. Reconocimiento: pruebas, resultados y cambios

En este anexo se detallan todas las pruebas, resultados y modificaciones que se llevaron a cabo en la fase de depuración del reconocedor implementado en este sistema.

J.1 Primeras pruebas

Para la primera prueba, la foto a comprobar formaba parte de la base de datos usada en el entrenamiento del reconocedor que consistía en solamente 6 caras de 3 personas diferentes (2 fotos por persona):

- Foto: 2011-08-26_13-21-50-169.jpg
- Resultado: Persona reconocida correctamente.

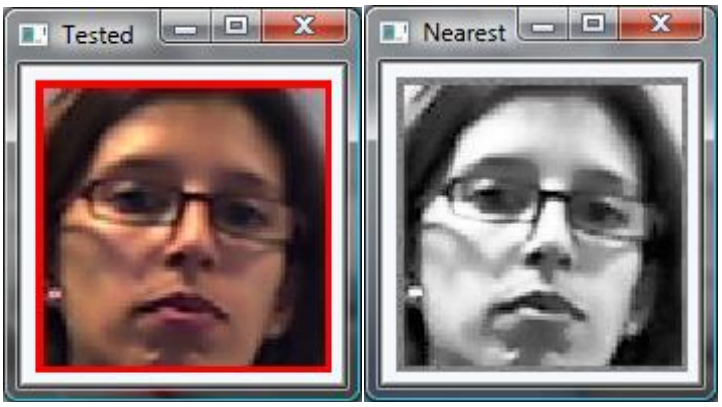


Figura J1: Imagen comprobada (izquierda) y resultado (derecha) de la primera prueba. Como era de esperar, el reconocedor no falla.

Para la siguiente prueba, se usó una base de datos con 63 caras donde las caras del entrenamiento habían sido obtenidas por el propio sistema. La foto a comprobar no pertenecía a esa colección de fotos.

- Foto: 2011-06-15_21-14-48-727.jpg
- Resultado:

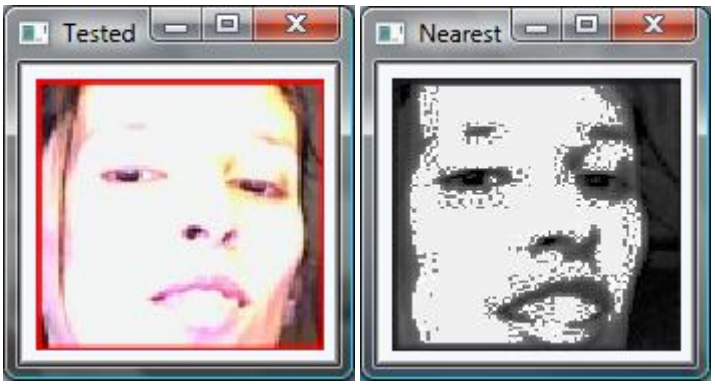


Figura J2: Imagen comprobada (izquierda) y resultado (derecha) de la segunda prueba.

De nuevo el reconocedor acierta. Sin embargo, el hecho de que se observase bastante ruido en la parte derecha de la cara de entrenamiento llevó a pensar en que este tipo de situaciones podía acabar afectando negativamente al resultado.

Por ello se pensaron posibles mejoras a añadir que pudiesen minimizar el posible efecto de la presencia de ruido en las imágenes de entrenamiento.

La idea pasaba por aplicar un nuevo post-procesado a las caras detectadas para intentar eliminar en la mayor medida posible este ruido que no ofrece información y que además dificulta el reconocimiento. Es decir, recortar la cara (*crop* en inglés) de la imagen. Si se logra dejar solamente el rostro de las personas sin ni siquiera el pelo (ya que este puede variar mucho en la misma persona), el nivel de robustez del reconocedor debería ser más elevado.

J.2 Pruebas con factor de recorte (*crop*)

Con una base de datos de 63 caras para el entrenador y teniendo en cuenta que las caras del entrenamiento habían sido obtenidas por el propio sistema, se probaron 8 caras de las cuales 2 pertenecían al conjunto usado en el entrenador. Los resultados obtenidos dependiendo del factor de recorte (en píxeles) fueron:

- Factor de recorte 0: ofrece 7 resultados acertados de 8.
- Factor de recorte 5 y factor de recorte 10: 100% de acierto.

Como no parecían muy concluyentes, se opto por probar más exhaustivamente.

La siguiente prueba se realizó con la misma base de datos para el entrenamiento pero comprobando 149 fotos. Los resultados según el factor de recorte fueron:

Resultados con *CROP* = 0%:

- Las caras usadas en el *train*ing (63 + 1 duplicada) se reconocen correctamente.
- De las restantes, hay 71 aciertos de 85.
- Los 14 fallos son debido a que las fotos son muy de cerca o de lejos. 2 de estos fallos son entendibles ya que se trata de una hermana gemela a la del entrenamiento.

En total, hay 135 aciertos y 14 fallos → 90% de acierto. Si se descuentan los 64 aciertos obvios, se obtiene un 83,5% de acierto.

Resultados con *CROP* = 5%:

- Las caras usadas en el *train*ing (63 + 1 duplicada) se reconocen correctamente.
- De las restantes, hay 72 aciertos de 85.
- Errores solucionados respecto a *CROP* = 0:
    1. 2011-08-15_19-09-16-934.jpg (Otro reconocido como Martin)
    2. 2011-08-26_13-19-43-078.jpg (Martin muy cerca)
    3. 2011-08-26_13-20-39-533.jpg (Martin muy cerca)

4. 2011-08-26_13-22-24-417.jpg (Martin de lejos)

- Errores nuevos respecto a *CROP* = 0:
    1. 2011-06-15_21-14-48-727.jpg (Daniela de cerca)
    2. 2011-08-26_13-22-11-824.jpg (Daniela de cerca)
    3. 2011-08-26_13-22-12-422.jpg (Daniela de cerca)

- Los 13 fallos son debido a que las fotos son muy de cerca o de lejos. 2 de estos fallos son entendibles ya que se trata de una hermana gemela a la del entrenamiento.

En total, hay 136 aciertos y 13 fallos → 91% de acierto. Si descontamos los 64 aciertos obvios, obtenemos un 84,7% de acierto.

Resultados con *CROP* = 10%:

- Las caras usadas en el *train*ing (63 + 1 duplicada) se reconocen correctamente.

- De las restantes, hay 73 aciertos de 85.

- Errores solucionados respecto a *CROP* = 0:
    1. 2011-08-15_19-09-16-934.jpg (Otro reconocido como Martin)
    2. 2011-08-26_13-19-43-078.jpg (Martin muy cerca)
    3. 2011-08-26_13-20-39-533.jpg (Martin muy cerca)
    4. 2011-08-26_13-22-24-417.jpg (Martin de lejos)

- Errores solucionados respecto a *CROP* = 5:
    1. 2011-08-26_13-22-23-895.jpg (Martin de lejos)
    2. 2011-08-26_13-22-25-007.jpg (Martin de lejos)

- Errores nuevos respecto a *CROP* = 0:
    1. 2011-06-15_21-14-48-727.jpg (Daniela de cerca)
    2. 2011-08-26_13-22-11-824.jpg (Daniela de cerca)
    3. 2011-08-26_13-22-12-422.jpg (Daniela de cerca)

- Errores nuevos respecto a *CROP* = 5 (tampoco estaba en *CROP* = 0):
    1. 2011-08-26_13-20-29-812.jpg (Martin de muy cerca)

- Los 12 fallos son debido a que las fotos son de cerca y al hacer el recorte se pierde información. Además, como en los otros casos, 2 de estos fallos son entendibles ya que se trata de una hermana gemela a la del entrenamiento.

En total, hay 137 aciertos y 12 fallos → 92% de acierto. Si descontamos los 64 aciertos obvios, obtenemos un 85,8% de acierto.

Tras estas pruebas se concluyó que los resultados eran aceptables pero mejorables ya que se tenían aproximadamente 20 caras de entrenamiento por cada persona a reconocer. Había 3 personas en la base de datos pero sin variedad en cuanto a distancia. Parecía preciso entrenar al sistema con algunas fotos de más cerca o más lejos de cada uno de los individuos, por ejemplo realizando un entrenamiento a conciencia haciendo uso del entrenamiento *online* con la cámara web. Existía también la opción de hacer uso del recorte pero de una manera algo más compleja

y computacionalmente costosa pero que a priori debería dar mejores resultados. Se basaría en crear automáticamente varios ejemplos de las fotos de entrenamiento, cada una de las cuales con un factor de recorte diferente. Por ejemplo, con la foto1.jpg crear: foto1_0.jpg con recorte = 0, foto1_5.jpg con recorte = 5 y foto1_10.jpg con recorte = 10. De esta forma se conseguirían muy buenos ejemplos para el entrenador con recorte 5 o 10 en fotos de entrenamiento donde la persona aparezca de lejos con bastante ruido alrededor de la cara (pelo, fondo, etc.), mientras que fotos de cerca aparecerán bien recogidas en las fotos sin recorte.

Otra forma de generar más fotos de entrenamiento interesantes sería generando fotos espejo o rotadas. Así se obtendrían muestras de las mismas personas sin necesidad de más interacción por su parte y se lograría generar un entrenador mucho más completo.

En resumen, parecía interesante crear un entrenador más robusto (reiterando el hecho de que los resultados son bastante aceptables a pesar de todo) obteniendo más muestras de las personas de la base de datos, haciendo uso de técnicas como:

- Entrenamiento *online* – interacción con el usuario: con una pequeña guía de posturas, en cuestión de segundos se pueden obtener muestras suficientes.

- Incremento automático de muestras:
    1. Simulando cercanía o lejanía con el factor de recorte
    2. Rotando
    3. Imagen espejo

Ambas técnicas no son excluyentes ya que al entrenamiento *online* (recogida de muestras a través de cámara web) se le pueden aplicar los métodos de generación automática de muestras para incrementar el potencial del entrenador. Teniendo en cuenta que el tamaño medio de las fotos almacenadas no supera los 5-8KB, el coste de almacenamiento triple o cuádruple por cada foto de *train*ing es totalmente asumible. Sin embargo habría que prestar atención al tiempo de creación de un entrenador con muchas fotos de prueba, ya que el aumento del tamaño de las estructuras de datos necesarias ralentiza el proceso.

J.3 Pruebas con incremento automático de caras para entrenamiento

Para estas pruebas, por cada imagen de origen se generan:

1. Imagen convertida a escala de grises.
2. Imagen con recorte simple (*crop* = 5).
3. Imagen con recorte doble (*crop* = 10).
4. Imagen invertida en el eje vertical (*horizontal flip* o *mirror*).

Resultados de la prueba con una base de datos de 63 caras para el entrenador (se obtienen 252 muestras). Las caras del entrenamiento han sido obtenidas por el propio sistema:

- El entrenamiento dura aproximadamente unos 15-20 segundos.

- Tamaño de *face*data.xml = 40902KB[27].

[27] Por comparar, el tamaño con una base de datos de entrenamiento de 32 caras (8 originales x 4) es de 5090KB y el tiempo en su creación es de 1-2 segundos. Esto es 8 veces menos tanto en número de fotos original, como en tamaño de las estructuras de datos como en tiempo de obtención del reconocedor. Por lo tanto existe un incremento lineal.

- Tiempo de test simple: 4 segundos por foto.
- Tiempo de test con resultados (sin mostrar imágenes resultado): 1 segundo.
- Probadas 8 caras: 100% de acierto.
- Probadas 149 fotos: 86,1% de acierto.

Resultados de la prueba con una base de datos de 175 caras para el entrenador (se obtienen 600 muestras). De nuevo, todas las caras del entrenamiento han sido obtenidas por el propio sistema:

- El entrenamiento dura aproximadamente unos 2 minutos y 40 segundos.
- Tamaño de facedata.xml = 116555KB.
- Tiempo de test simple (mostrando ventanas con foto resultado) = 12-13 segundos por foto.
- Tiempo de test con resultados (sin mostrar imágenes resultado): 1-2 segundos
- Probadas 8 caras. 100% de acierto.
- Probadas 149 fotos: 86,8% de acierto.

Los resultados revelan que el acierto del sistema se incrementa, pero a costa de un aumento más que considerable en el tiempo de generación del reconocedor. La decisión tomada fue que esta opción no se implementaría en la versión final.

Sin embargo, la idea subyacente de generación automática de imágenes podría seguir siendo aprovechable. Únicamente habría que "darle la vuelta a la tortilla": en vez de incrementar las muestras de entrenamiento, se incrementan las muestras de prueba. De esta manera se podría intentar mejorar, en caso necesario, los resultados del reconocedor de manera automática sin intervención del usuario pero además obteniendo sin afectar al tiempo de generación del reconocedor y espacio en la base de datos (de entrenamiento).

J.4 Pruebas con fotos externas

Se realizó a continuación una prueba adicional para ver el comportamiento del sistema cuando es alimentado con fotos para el entrenador ajenas al sistema.

Se utilizaron caras de la ORL *database*[28]. Debido al formato de las fotos de esa base de datos, formato PGM y tamaño 92x112, era necesario convertir las imágenes a JPG de 100x100 para homogeneizar las pruebas. Aprovechando las funcionalidades de *OpenCV*, se añadieron dos funciones nuevas a la aplicación: conversión de una sola imagen de cualquier formato reconocido por *OpenCV* a JPG o de todas imágenes en una carpeta. De esta forma, cualquier foto de una cara que se pueda tener puede automáticamente pasar a formar parte de la base de datos de caras de la aplicación, sin que esta haya tenido que detectarla.

[28] Cortesía de *AT&T Laboratories Cambridge*.

Los primeros problemas detectados fueron que la imagen PGM convertida a JPG de 100x100 no ofrecía los mismos resultados que las originales JPG obtenidas desde cámara web, video o foto con la propia aplicación. Hay que considerar los múltiples aspectos de una foto al almacenarla como *IplImage* con *OpenCV*. Por ejemplo, la profundidad en bits o el número de canales. Si la imagen PGM tiene mucha más definición en algunos aspectos, esto hay que corregirlo. Por lo tanto, si la profundidad en la imagen convertida se estandariza a 8 bits en vez de a la propia profundidad de la imagen origen, existe mejoría. En todo caso, no se consigue la precisión que ofrece el reconocedor entrenado con fotos propias, aunque es una nueva funcionalidad añadida (permitir que fotos externas formen parte del entrenamiento).

J.5 Pruebas con entrenamiento *online*

En las pruebas con imágenes obtenidas del entrenamiento con la cámara web, debido a que la frecuencia de escaneo era muy elevada, el reconocedor era capaz de procesar hasta 4 caras por segundo.

Se recogieron muestras de 8 personas en un entorno sin grandes cambios de luz. Uno de los sujetos era simplemente una foto en un calendario. De cada sujeto se tomaron aproximadamente 50 fotos.

El primer reconocedor se genero a partir de 20 imágenes aleatorias de cada persona. Los resultados no fueron nada alentadores: el acierto no llegó al 78%. Seleccionando fotos más acertadas para el entrenador de cada persona (en varias distancias, eliminando algún falso positivo detectado, etc.) se logró llegar al 80% de acierto. Eliminando del entrenador el sujeto cuyas imágenes fueron obtenidas de una foto, el acierto subió por encima del 83%. Cada incremento en el acierto generó conclusiones ya en este punto:

- Es recomendable hacer una pequeña selección de imágenes de entrenamiento eliminando aquellas que pudiesen confundir al sistema o que ofrezcan poca variación con respecto a otras.

- Es preciso, en la medida de lo posible, eliminar los falsos positivos del entrenamiento.

Para la siguiente prueba se creó un reconocedor usando todas las imágenes disponibles. Los datos concretos de la prueba fueron:

- Fotos totales originales: 346.

- Set de entrenamiento: 1384 fotos.

- Tiempo de ejecución:
    - Entrenamiento *offline*: 16 segundos
    - Generar reconocedor: 16 minutos

- Tamaño del XML reconocedor: 244660 KB

- Tiempo de reconocimiento de una persona: 30 segundos

Los resultados de estas pruebas ofrecieron resultados aún más pobres que la anterior: el acierto no llegó al 75%.

La conclusión tras esta segunda prueba fue que parecía necesario reducir el número de imágenes de entrenamiento tomadas con la cámara web, entendiendo que no siempre a mayor numero de caras de entrenamiento mejores resultados ya que lo que se puede conseguir es

confundir al reconocedor. Por lo tanto, la configuración por defecto de la herramienta establece la captura de imágenes y su procesado una vez por segundo aproximadamente (cada 30 fotogramas).

J.6 Pruebas definitivas con pocas muestras

Las pruebas finales se realizaron creando reconocedores a partir de un número pequeño de muestras de cada sujeto de entrenamiento, como se había concluido en las pruebas anteriores. Además, se llevaron a cabo las otras conclusiones obtenidas hasta el momento acerca de seleccionar un buen conjunto de muestras de entrenamiento para depurar el reconocedor y obtener mejores resultados.

Las pruebas indicaron que a medida que el número de caras de entrenamiento avanzaba, el acierto también se elevaba. En la tabla 9 se muestra la evolución en el caso de tres sujetos, Ana, Daniela y Blanca, probando 75 caras en el caso de Ana, 43 en el caso de Blanca y 38 en el caso de Daniela:

Sujeto de test	2 muestras[29]	3 muestras	4 muestras	5 muestras	6 muestras
Ana	100%	100%	99%	100%	100%
Blanca	79%	84%	93%	72%	79%
Daniela	47%	50%	50%	97%	100%
Media	**75%**	**78%**	**81%**	**90%**	**93%**

Tabla 9. Porcentajes de acierto del reconocedor a partir del número de muestras de entrenamiento.

Cabe destacar que el paso de 50% a 97% de acierto en el caso de Daniela se debió a una elección acertada de la nueva cara a usar en el entrenamiento. Esa misma cara hacía que el nivel de acierto en el caso de Blanca se viese reducido, ya que la posición de la cara nueva de Daniela era muy similar a varias muestras de Blanca, lo que confundía al reconocedor.

Más adelante, se probó a añadir gente a la base de datos actual, dejando las mismas muestras de las personas anteriores e incluyendo 6 muestras de una nueva persona, en este caso David. El resultado obtenido fue muy prometedor ya que los porcentajes de Ana, Blanca y Daniela se mantuvieron, y el de David se colocó directamente en el 100%.

Al añadir a 2 personas más, Martin y Víctor, los resultados quedaron de la siguiente manera:

- Ana: 89% (desciende debido a la confusión con muestras nuevas con posiciones corporales parecidas).
- Blanca: 79% (los mismos fallos se mantienen).
- Daniela: 100%.
- David: 90% (posición de una muestra de Víctor muy parecida a 4 fotos seguidas de David).
- Martin: 94%.

[29] Numero de caras de cada persona tomadas como muestra en la fase de entrenamiento. El total de muestras del entrenamiento es ese número multiplicado por el número de personas (3).

- Víctor: 84% (5 fallos en fotos seguidas muy parecidas es lo más destacado).

Media obtenida de 90% aproximadamente, teniendo 36 fotos de muestra para el entrenamiento (6 por persona) y comprobadas 280 fotos.

Al añadir por ultimo a otro sujeto, Javi, se obtuvieron los resultados siguientes:

Sujeto de test	Acierto	Comentario
Ana	88%	Un fallo adicional
Blanca	79%	Los mismos fallos
Daniela	100%	Perfecto
David	75%	Muchas de las imágenes de David y de Javi son bastante parecidas, confundiendo al reconocedor en 7 donde antes no fallaba
Javi	84%	Un fallo adicional
Martin	90%	Un fallo adicional
Víctor	82%	Fallos relacionados con los de David
Media	**85%**	

Tabla 10. Porcentajes de aciertos de las pruebas con 7 sujetos (42 muestras) y probadas 329 fotos.

De nuevo, y a pesar de los nuevos errores encontrados al tener a Javi con muestras parecidas a las de David, la media de acierto que ofrece el sistema es bastante aceptable. Modificando ligeramente el entrenador para que las imágenes de David y Javi no se confundan, el sistema volvió a ofrecer resultados del 90% e incluso por encima.

J.7 Conclusiones

Una de las principales conclusiones obtenidas es que a mayor número de caras mejor parece el resultado global, pero teniendo muy en cuenta que un número demasiado elevado de caras tiene los siguientes inconvenientes:

- Puede confundir al reconocedor si se añaden muestras de personas muy parecidas a las ya presentes.

- Ralentiza todo el proceso de entrenamiento y reconocimiento.

Por otro lado, es recomendable hacer una correcta selección de imágenes de entrenamiento eliminando aquellas que pudiesen confundir al sistema (por ejemplo los falsos positivos) o que ofrezcan poca variación con respecto a otras.

En resumen, el sistema desarrolla un nivel de acierto por encima del 85% en casi cualquier caso y llega al 90% con relativa facilidad, únicamente con 6 muestras por sujeto para el entrenamiento. En cuanto a tiempo de ejecución, su comportamiento también es la generación de un reconocedor con 42 muestras no tarda más de 2 segundos y el tiempo de reconocer y almacenar resultados en el caso de mayor número de caras de prueba (75) de un solo sujeto, no tarda más de 3 segundos.

Anexo K. Modo vigilancia. Compatibilidades

Es interesante observar las compatibilidades o incompatibilidades del sistema cuando se encuentra en modo de vigilancia. La lista siguiente muestra las tareas que se pueden ejecutar en modo vigilancia y las que no. Las marcadas como N/A se consideran no aplicables ya que requieren de la misma imagen de cámara web.

Funcionalidad	Funciona en Modo vigilancia
Cargar foto	SÍ
Mostrar foto	SÍ
Detectar en foto	SÍ – Requiere reactivación de la vigilancia
Detectar en foto rotada varias veces	SÍ – Requiere reactivación de la vigilancia
Detectar en foto almacenando resultados sobre la acción (modo *debug*)	SÍ – Requiere reactivación de la vigilancia
Cargar video	SÍ
Mostrar video	SÍ
Detectar en video	SÍ
Mostrar imagen de cámara web	N/A
Detectar sobre imagen de cámara web	N/A
Entrenamiento *offline*	SÍ
Entrenamiento *online*	N/A
Generación del reconocedor	SÍ
Reinicio de reconocedor	SÍ
Conversión de imagen a JPG	SÍ
Conversión de múltiples imágenes a JPG	SÍ
Mostrar ultimas detecciones	SÍ
Mostrar todas detecciones	SÍ
Comprobar persona	SÍ
Comprobar múltiples personas (modo *debug*)	SÍ
Cambiar base de datos de caras	SÍ
Cambiar directorio para entrenamiento	SÍ
Modificar frecuencia de detección sobre cámara web	NO RECOMENDABLE – puede dejar de detectar
Modificar frecuencia de detección sobre video	SÍ
Modificar factor de recorte (modo *debug*)	SÍ
Modificar factor de escala	SÍ
Modificar número de vecinos	SÍ
Modificar modo	SÍ
Modificar tamaño de ventana mínima de escaneo	SÍ
Modificar tamaño de ventana máxima de escaneo	SÍ
Cargar detector XML	NO tiene efecto

Tabla 11. Compatibilidades o incompatibilidades del sistema cuando se encuentra en modo de vigilancia.

La tabla anterior indica que salvo la modificación del tipo de detector utilizado que no tiene efecto, y la modificación de la frecuencia de detección sobre la cámara web que puede hacer que el sistema deje de detectar, todas las demás funcionalidades están disponibles aunque el sistema se encuentre en modo de vigilancia, lo que ofrece gran versatilidad al sistema.

Asimismo, hay que recalcar que todas estas opciones tienen preferencia sobre la vigilancia, por lo que cuando se están ejecutando, el sistema detiene la detección.

Anexo L. Errores y contratiempos

En este apartado se muestran diferentes errores técnicos y otros contratiempos encontrados durante el desarrollo del proyecto y su solución.

L.1 Errores técnicos

Uno de los primeros errores de compilación encontrados estaba relacionado con la configuración del proyecto en el entorno de desarrollo integrado (IDE) de *Visual C++*. En concreto, con la no compatibilidad de algunas funciones no administradas con el *Common Language Runtime*[30] (CLR) puro:

```
1> ------ Operación Generar iniciada: proyecto: Detector_Menu_v1,
configuración: Debug Win32 ------
1> Compilando...
1> stdafx.cpp
[…]
1> c:\archivos de programa\OpenCV\cxcore\include\cxtypes.h(205):
   error C3862: 'cvRound': no se puede compilar una función no
   administrada con /clr:pure o /clr:safe
1> La función utiliza un ensamblador en línea o una función
   intrínseca no admitida
1> c:\archivos de programa\OpenCV\cxcore\include\cxtypes.h(205):
   error C3645: 'cvRound' : no se puede utilizar __clrcall en
   funciones compiladas para código nativo
[…]
== Generar: 0 correctos, 1 incorrectos, 0 actualizados, 0 omitidos
```

La solución consistió en establecer la compatibilidad del código con CLR estándar desde el propio IDE:

- Proyecto.
- Propiedades del proyecto.
- Propiedades de configuración.
- General.
- Compatibilidad con *Common Language Runtime*.

También relacionado con la configuración del proyecto en *Visual C++* se encuentra el siguiente error sufrido, en este caso en el momento de vinculación:

```
C:\Documents and Settings\Administrador\Mis documentos\Visual
Studio 2005\Projects\Detector_Menu_v1\Debug\Detector_Menu_v1.exe:
fatal error LNK1120: 36 externos sin resolver
```

Como muestra el error, el sistema es incapaz de vincular diferentes símbolos externos si no se configura el proyecto correctamente. La solución pasa por indicar al vinculador donde debe buscar todos esos símbolos, que no es otro sitio que librerías de *OpenCV*. De nuevo, a

[30] El *Common Language Runtime* o CLR es un entorno de ejecución para los códigos de los programas que corren sobre la plataforma *Microsoft* .NET.

través de las propiedades del proyecto se soluciona este error. Los pasos a seguir de encuentran detallados en el Anexo B de este documento relacionado con la configuración de *OpenCV* y *Visual C++*.

Otro de los errores encontrados tiene que ver con el fichero **windows.h**. Si el proceso de generación de la solución (*build*) muestra el error de que el archivo **windows.h** no se encuentra, es necesario instalar la última versión disponible de *Microsoft Windows SDK*. Adicionalmente, se deben configurar las propiedades generales de *Visual C++* para el perfecto acoplamiento con el SDK instalado.

En esta explicación se toma *Microsoft Platform SDK for Windows Server 2003 R2* como versión del *Microsoft Windows SDK*. Ruta de instalación:

C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2

En esta explicación se omiten las capturas de pantalla debido a su similitud con las presentadas en el Anexo B.

Los ajustes se realizan a través de:

- Herramientas
- Opciones
- Proyectos y soluciones
- Directorios de VC++
- Mostrar directorios para:
    - Archivos ejecutables: Se añade la ruta al directorio BIN del SDK. En este caso *C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Bin*.
    - Archivos de inclusión: Se añade la ruta al directorio INCLUDE del SDK. En este caso *C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Include*.
    - Archivos de biblioteca: Se añade la ruta al directorio LIB del SDK. En este caso *C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Lib*.

El último de los errores que se mencionan aquí apareció en tiempo de ejecución, en forma de una excepción no controlada.

El detalle del error es el siguiente:

```
Excepción del tipo 'System.MissingMethodException' en
Detector_Menu_v1.exe
Excepción no controlada del tipo 'System.MissingMethodException'
en Detector_Menu_v1.exe

Información adicional: Método no encontrado: 'Int64
System.IO.Directory.Seek(Int64, System.IO.SeekOrigin)'.
```

Haciendo uso de la opcion de ejecución en modo *debug* (depuración), se descubrió que aparecía al primer instante de la ejecución de la aplicación, al ejecutar la llamada de creación de la ventana principal:

```
Application::Run(gcnew Form1());
```

El hecho de que este error apareciese sin haber existido modificación alguna del código, hace pensar que una de las múltiples actualizaciones automáticas de *Windows* es la posible causa.

La solución encontrada resultó ser muy sencilla afortunadamente. Simplemente hubo que reiniciar desde cero la compilación y generación del ejecutable de la solución del proyecto con la herramienta para tal uso que ofrece *Visual Studio* (Generar | Limpiar solución). Esta acción elimina archivos intermedios y de resultados para el proyecto actual. Tras una nueva compilación y vinculación de bibliotecas, la excepción no controlada dejó de aparecer.

L.2 Problema con las caras a tomar para el entrenamiento en el método *online*

Al principio, se pensó en hacer uso de la imagen en blanco y negro (B&N) y ecualizada sobre la que se aplica el detector de caras. A la vez que se seleccionaba la cara de la imagen original para almacenarla en la base de datos de caras, se seleccionaba de la foto en B&N la cara para almacenarla en la base de datos de entrenamiento. Sin embargo, los primeros resultados no eran nada satisfactorios: las imágenes de entrenamiento obtenían una confianza del reconocedor del 60% aproximadamente. Si el entrenamiento *offline* ofrecía resultados del 99,9% con las mismas imágenes, este 60% obviamente no era aceptable.

Debido a que la ecualización de una cara dentro del conjunto de la foto general podría ser diferente de la ecualización de la cara recortada, se decidió coger la imagen de la base de datos de caras (original en color) y aplicarle la ecualización y el B&N. Este era un proceso sumamente sencillo ya que la imagen estaba accesible en una variable. Así que solamente convirtiendo esa imagen era suficiente. Los resultados mejoraron pero desafortunadamente, la confianza no llegaba al 90%. De nuevo, inaceptable si el entrenamiento *offline* ofrecía esos resultados cercanos al 100%.

El hecho de que el entrenamiento *offline* funcionase a la perfección forzó a que se implementara en el *online* el mismo proceso, esto es, cargar imagen de la cara para luego convertirla y almacenarla. De esta manera, se asegura que se realiza exactamente los mismo pasos que en el entrenamiento *offline* y, lo que es más importante, lo mismo que en los procesos de comprobación. El resultado, óptimo: caras de la base de datos de entrenamiento ofrecen 99,9% de nivel de confianza.

L.3 Otros contratiempos

Sin duda alguna, uno de los aspectos más problemáticos en cuanto a generación de código en este proyecto ha sido el manejo de cadenas de caracteres.

El hecho de que *OpenCV* requiera muchas veces tipos no manejados (*unmanaged*) como argumentos para sus funciones, unido a que el IDE de *Visual C++* ofrece una gran variedad de métodos para manejo de *Strings*, que es un tipo manejado (*managed*), complica sobremanera el intercambio de datos entre funciones de *OpenCV* y las de sistema.

Un ejemplo muy sencillo para entender este problema es el caso de obtener del usuario el nombre del archivo a analizar a través de un dialogo de abrir archivo (**OpenFileDialog**). Este diálogo devuelve un dato de tipo System::String, que es manejado. Si necesitamos pasar ese dato a la funcion de *OpenCV* de cargar una imagen (**cvLoadImage**) debemos convertir el *String* a un array de caracteres (tipo no manejado*)*, ya que la funcion de *OpenCV* tiene como parámetro un array de caracteres, no un String.

Para resolver este tipo de inconvenientes, el proyectando tuvo que implementar funciones de conversion tanto en una direccion como en la otra. Su nombre, indica el tipo de conversion que realizan:

- `MiStringtoChar`: convierte un dato de tipo *String* a otro de tipo *Char* *
- `MiChartoString`: convierte un dato de tipo *Char* * a otro de tipo *String*

Para la funcion **MiStringtoChar** se usó lo referenciado en la página de soporte de *Microsoft* MSDN (*Microsoft Developer Network*) acerca de la conversion entre varios tipos de cadenas de caracteres:
http://msdn.Microsoft.com/en-us/library/ms235631(v=vs.90).aspx

Los siguientes contratiempos, presentados en este anexo debido a su relevancia, se detallan brevemente a continuación:

- Nuevas versiones de *OpenCV* han ido apareciendo durante el tiempo que ha durado el proyecto fin de carrera: Se empezó con la versión 1.0. Esta versión ha sido ampliamente utilizada en la implementación del detector de caras en fotos. Incluye librerías completas DLL. A la hora de empezar con la implementación del detector de caras en video y sobre todo la parte de reconocimiento, se decidió actualizar *OpenCV* a la versión 2.0. Esta nueva versión 2.0 tiene el inconveniente de que los archivos de biblioteca .LIB necesarios para el desarrollo de proyectos en entornos de desarrollo integrados no están generados. El cómo obtenerlos aparece explicado en el Anexo B de esta memoria.

- Diferentes versiones de *Visual Studio C++ Express* usados a lo largo del proyecto requirieron algo de reescritura del código de la aplicación: Comienzo con *Visual C++ 2005 Express* debido a que era el más usado entonces con aplicaciones de *OpenCV*. Paso a *Visual C++ 2008 Express Edition*.

- Diferentes sistemas operativos usados a lo largo del proyecto: Comienzo con un *Windows XP*. Debido al cambio de ordenador del *Windows XP* a un *Windows Vista*, se decidió usar la aplicación *VMWare* para seguir desarrollando código sobre un *Windows XP*. Ultima parte desarrollada en un *Windows Vista*. Esto ha servido para comprobar que con pequeños cambios de configuración, las aplicaciones desarrolladas funcionan bien sobre diferentes sistemas operativos *Windows*.

Como ocurre con cualquier dificultad superada, estos contratiempos, una vez superados, sirvieron de enriquecimiento personal al autor del proyecto.

Anexo M. Manual de usuario

En este manual de usuario se presentan todas las funcionalidades de la aplicación desarrollada, especificando claramente como acceder a cada una de ellas y su propósito.

M.1 Introducción

La herramienta SIDECAR (SImple DEtector de CARas) permite realizar tareas de vigilancia y control de personas a través de la imagen proporcionada por una cámara web.

La tarea básica de vigilancia consiste en monitorizar en todo momento la imagen de la cámara web y detectar cualquier persona que aparezca en ella. Cada detección implica el almacenamiento de la cara de la persona detectada y el instante temporal de dicha detección. Con esta información esencial el sistema es equiparable a un sistema de circuito cerrado de video en cuanto a vigilancia y control de personas se refiere pero ahorrando a su vez una inmensa cantidad de espacio de almacenamiento.

Para el control de personas, SIDECAR ofrece tanto funcionalidades de consulta de personas detectadas e instante de la detección como tareas básicas de reconocimiento de personas. Estas tareas de reconocimiento permiten al sistema ser capaz de informar al usuario automáticamente si cierta persona ha sido previamente detectada y, en caso de poseer esa información, proveer el nombre de esa persona reconocida.

SIDECAR ofrece un sencillo interfaz para permitir al usuario el fácil acceso a todo su potencial (Figura M1).

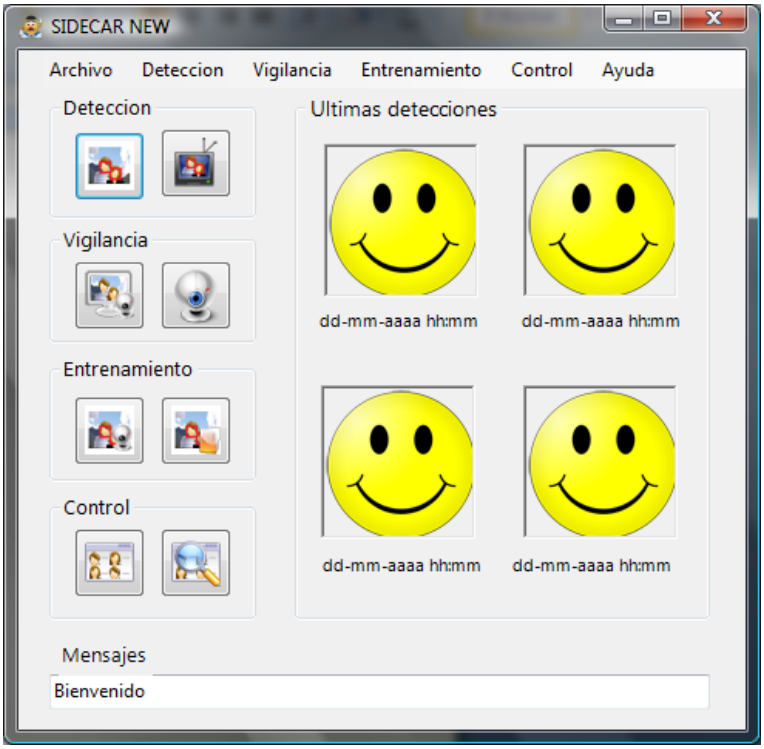


Figura M1: Ventana principal de la aplicación SIDECAR para vigilancia y control de personas.

Esta ventana principal presenta la barra de menú para acceder a todas las funcionalidades y accesos directos a:

- Detección:
  - en foto
  - en video

- Vigilancia:
  - iniciar vigilancia
  - vigilar y reconocer

- Entrenamiento (para reconocedor):
  - con fotos
  - con imágenes de cámara web en tiempo real

- Control:
  - mostrar detecciones
  - comprobar persona

Así mismo, muestra las ultimas detecciones junto con el instante de la detección y un cuadro de mensajes para mantener al usuario informado de las tareas que se están ejecutando y varios resultados según corresponda.

A continuación se presentan en diferentes apartados todas las funcionalidades de SIDECAR para guiar al usuario en el uso de esta herramienta.

M.2 Opciones de configuración

Las opciones de configuración de SIDECAR permiten al usuario de la herramienta establecer los parámetros que mejor se ajusten al entorno donde va a ser usada para obtener el máximo rendimiento.

Las opciones de configuración se presentan en una ventana adicional, accedidas a través de **Archivo | Opciones** y separadas en dos pestañas: opciones generales y opciones del detector.

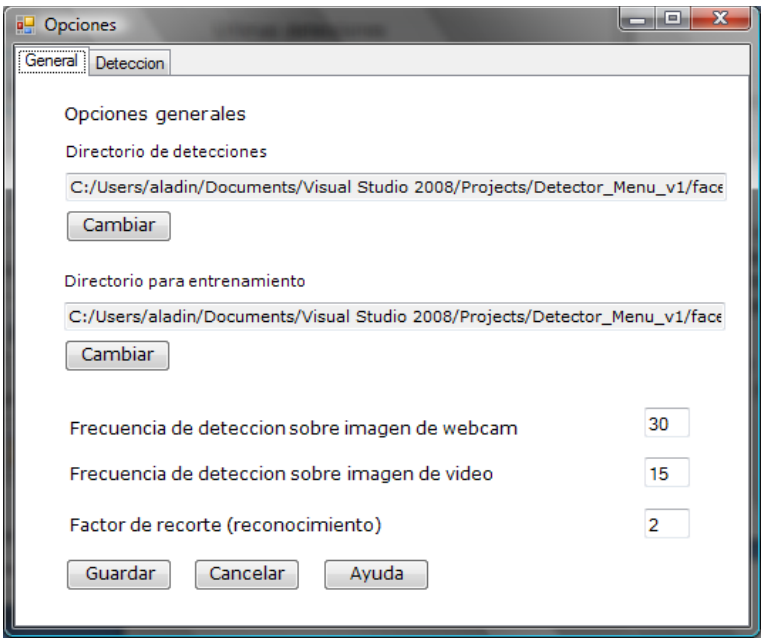


Figura M2. Ventana de opciones generales de SIDECAR. Se muestran los valores actuales.

En la primera pestaña (ver Figura M2) se presentan las opciones más generales de la aplicación, que son:

- Directorio de detecciones: se permite al usuario establecer el directorio donde almacenar las caras detectadas. Esta ruta es la que la aplicación utilizará para obtener las caras con las que realizar diferentes acciones, como por ejemplo el entrenamiento *offline* y la muestra de las detecciones.

- Directorio para entrenamiento: debido a que las imágenes a usar en el entrenamiento requieren un procesado (convertir a blanco y negro, ecualizar, etc.), existe un almacén adicional de caras procesadas. La ruta a este almacén se establece a través de esta opción.

- Modificar frecuencia de detección sobre imagen de cámara web: este parámetro determina cada cuantos fotogramas se escanea la imagen de la cámara web en busca de caras. Cada segundo se procesan aproximadamente 30 fotogramas, por lo tanto, un valor de 30 establece que se analice un fotograma cada segundo. Si el valor de la frecuencia es 0, la aplicación escanea todos los fotogramas. Este valor debe ajustarse para obtener un equilibrio acertado entre la capacidad de la máquina donde la aplicación se ejecuta y lo exhaustivo que se requiera la detección.

- Modificar frecuencia de detección sobre imagen de video: igual que en el caso de la cámara web, esta frecuencia determina cada cuantos fotogramas se escanea la imagen de video en busca de caras.

- Factor de recorte: usado durante la fase de pruebas del reconocedor, este parámetro indica el número de píxeles a recortar de cada imagen de entrenamiento para intentar eliminar el posible ruido en forma de fondo, pelo, orejas, etc. Solo se permite su uso en tareas de depuración ya que requiere habilitar trozos de código fuente.

El botón de ayuda (Figura M3) permite acceder a una breve descripción de cada parámetro para referencia del usuario.

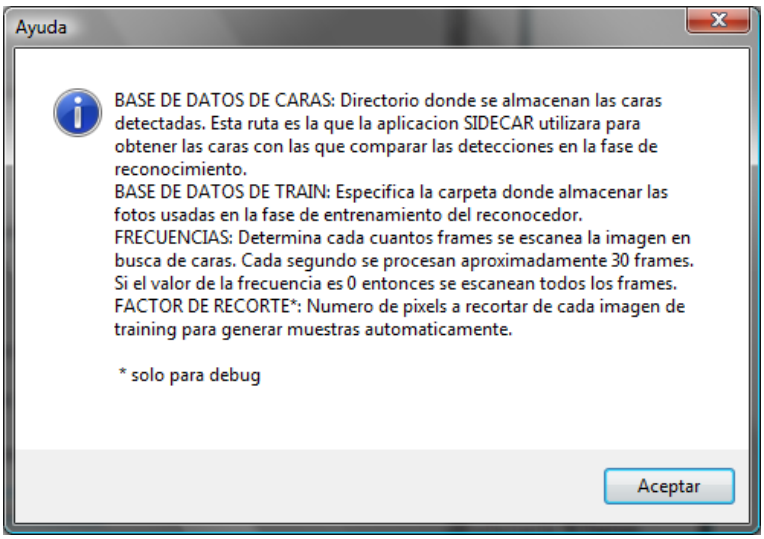


Figura M3. Ventana de ayuda integrada con información de las opciones generales.

La segunda pestaña (Figura M4), con las opciones del detector, permite la modificación de todos los parámetros configurables del detector mencionados en el capítulo 3, que son:

- Factor de escala: establece el factor por el que la ventana de búsqueda se escala en cada subsiguiente escaneo. En número entero. Un valor de 10 corresponder a un incremento de la ventana de un 10%.

- Numero de vecinos: mínimo número de detecciones vecinas (solapadas) que completan el objeto buscado. Con número de vecinos igual a N, todos los grupos de N-1 detecciones son rechazados. Si N es 0, no se rechaza ninguna detección lo que puede aumentar el número de falsos positivos.

- Modo: modo de búsqueda. Solamente soportado el modo 1 (*Canny Prunning*) en el que se rechazan regiones con muy pocos o demasiados bordes.

- Tamaño de ventana mínima de escaneo (en píxeles): tamaño inicial de la ventana de búsqueda.

- Tamaño de ventana máxima de escaneo (en píxeles): tamaño final de la ventana de búsqueda. Al llegar a ese valor, la búsqueda concluye.

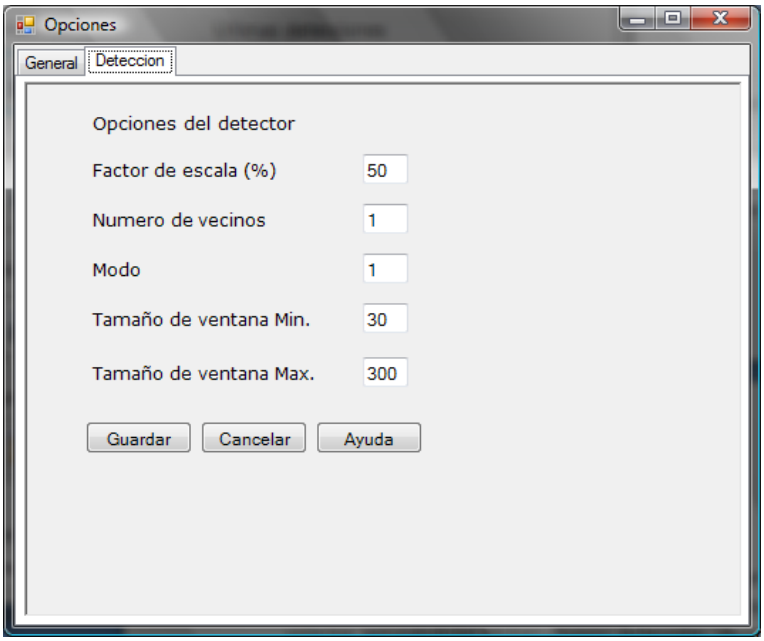


Figura M4. Ventana de opciones de detección de SIDECAR. Se muestran los valores actuales.

El botón de ayuda integrada (Figura M5) permite acceder a una breve descripción de cada parámetro para referencia del usuario.

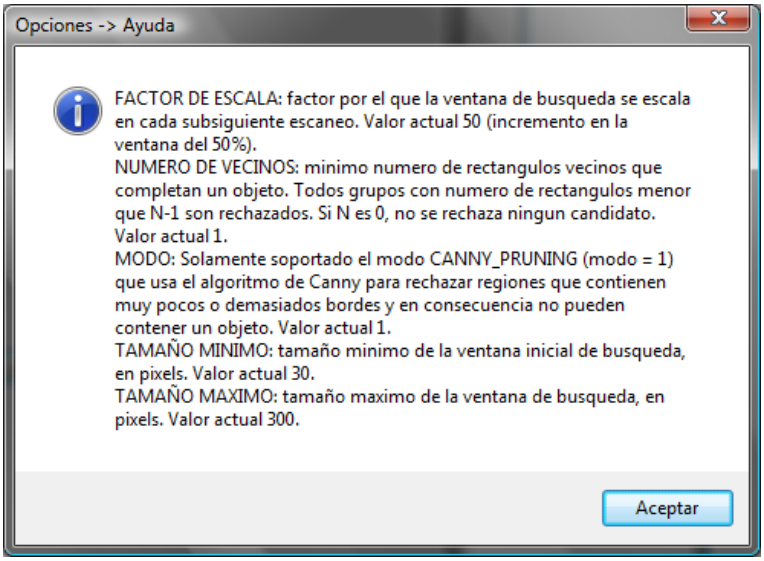


Figura M5. Ventana de ayuda integrada con información de las opciones generales.

Es altamente recomendable que el usuario de la aplicación realice pruebas con distintos valores de estos parámetros del detector para conseguir un detector que se ajuste de la mejor manera al entorno donde se utilice y poder aprovechar al máximo el potencial del sistema.

M.3 Detección

Las funcionalidades que ofrece la aplicación a través del menú de **Detección** son:

- **Foto | Cargar foto**: necesario para indicar al sistema sobre que foto se desea realizar la detección. Deben ser archivos JPG. A pesar de que *OpenCV* soporta múltiples formatos de imagen (BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF y TIF), se decidió establecer JPG como formato principal debido a su alto nivel de compresión y a su extendido uso.

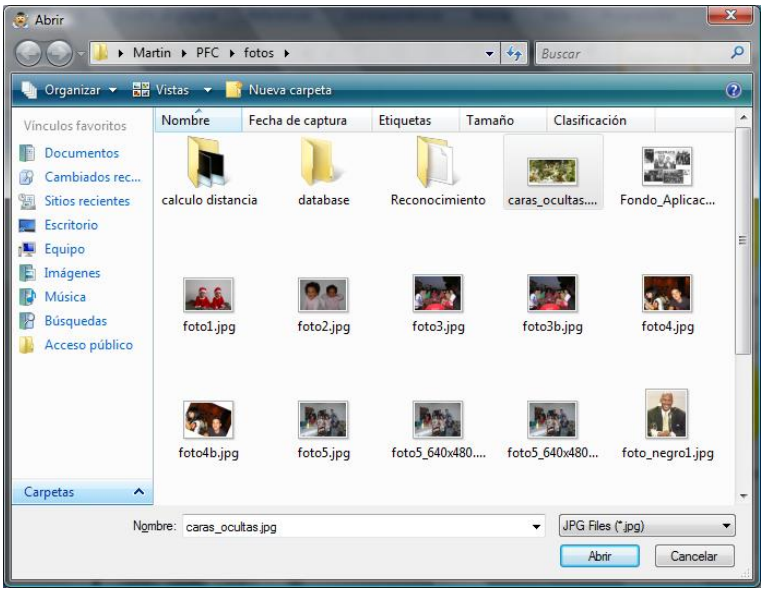


Figura M6. Diálogo para cargar una foto accedido a través de Detección | Foto | Cargar foto.

- **Foto | Mostrar foto**: muestra la foto seleccionada en una ventana aparte.

- **Foto | Detectar caras**: ejecuta la detección de rostros sobre la foto seleccionada. Se muestra el resultado de la detección en un mensaje informativo junto con la imagen original y la imagen resultado.

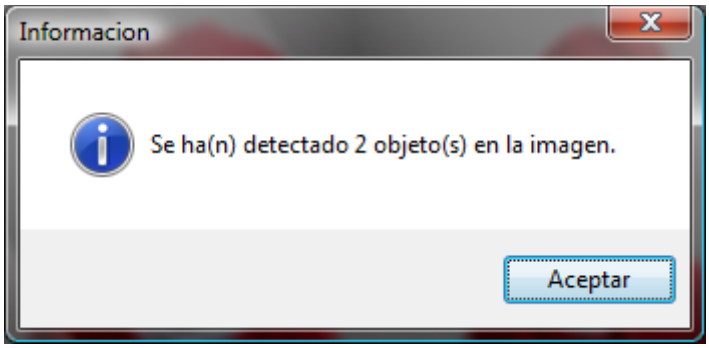


Figura M7. Mensaje informativo de una detección sobre imagen fija (foto).El resultado muestra 2 objetos detectados.

Figura M8. Imagen original e imagen resultado de una detección sobre imagen fija (foto)

- **Foto | Rotar y Detectar**: debido a la limitación del detector relacionada con la alineación horizontal de las caras, se incluyó esta funcionalidad para minimizar su impacto en la detección en fotos.

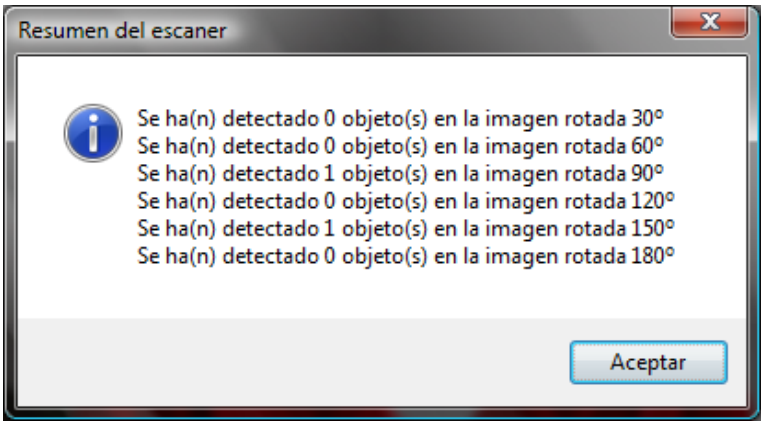


Figura M9. Ventana informativa del resultado de la acción de rotar y detectar.

- **Foto | Detectar (*debug)***: útil para la fase de pruebas durante el desarrollo de la aplicación, se incluye en el sistema final para usarse en la depuración de la configuración de los parámetros del detector. La información, que incluye la foto

analizada, todos los parámetros del detector utilizado y el resultado, se almacena en el archivo **resultados_deteccion.csv**.

- **Video | Cargar video**: indica al sistema sobre que archivo de video se desea realizar la detección. Los archivos deben ser del tipo AVI, MP4 o FLV.

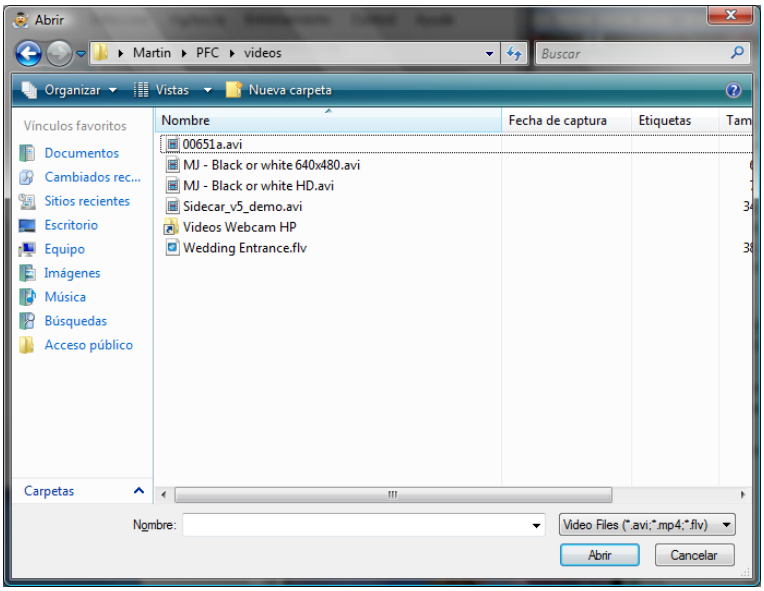


Figura M10. Diálogo para cargar un vídeo accedido a través de Detección | Video | Cargar video.

- **Video | Mostrar video**: reproduce el video seleccionado e informa al usuario sobre algunas de sus características (velocidad y resolución) en una ventana informativa.

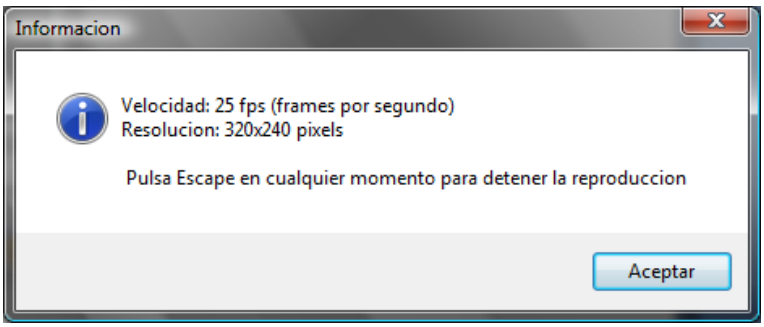


Figura M11. Ventana informativa con características del video a mostrar.

***Nota**:*
En SIDECAR, la reproducción tanto de los vídeos como de la imagen de cámara web puede ser cancelada en cualquier momento pulsando la tecla de escape o cerrando la ventana de reproducción.

- **Video | Detectar en video**: ejecuta la detección de rostros sobre el video seleccionado. Cada cara detectada es automáticamente almacenada en la base de datos de caras.

Figura M12. Instante de una detección de cara sobre imagen de video. El objeto detectado se recuadra.

La última de las funcionalidades relacionadas con la detección es la posibilidad de cargar diferentes tipos de detectores (**Detección | Detectores XML**). Los detectores son ficheros XML que definen que objeto es el que se va a buscar en la imagen. Algunos de estos ficheros, disponibles tanto en el paquete de instalación de *OpenCV* como en la red, ofrecen la posibilidad de detectar diferentes partes del cuerpo como:

- Cara Frontal → *seleccionado por defecto*
- Cuerpo
- Perfil
- Nariz
- Boca
- Ojos
- Otros

Si se selecciona la opción Otros, se permite cargar cualquier otro detector, siempre archivos de tipo XML.

M.4 Vigilancia

Las funcionalidades que ofrece la aplicación a través del menú de **Vigilancia** son:

- **Mostrar cámara web**: permite al usuario comprobar el correcto funcionamiento de la cámara web a usar. Este modo no lleva a cabo ninguna detección.

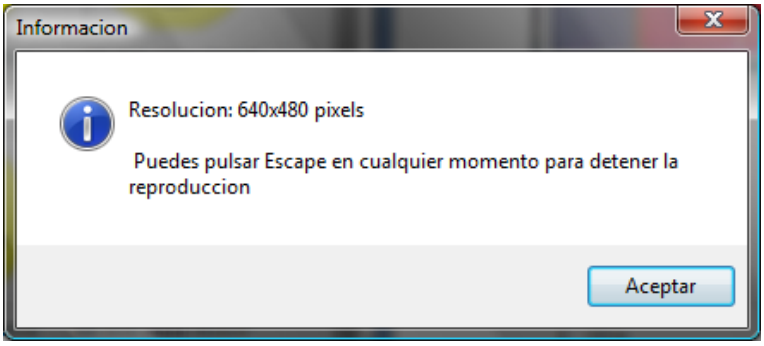


Figura M13. Ventana informativa con características de la imagen de cámara web a mostrar.

- **Iniciar vigilancia**: Este es el modo principal de operación de la aplicación. Se inicializa la cámara web cuya imagen es mostrada en una ventana aparte y cada cara detectada es almacenada automáticamente en la base de datos de caras junto con el instante temporal de su detección.

Figura M14. En el modo vigilancia, la imagen proporcionada por la cámara web se muestra a la vez que es escaneada en busca de caras.

- **Vigilar y reconocer**: Reconocimiento *online* con cámara web: se ejecuta la tarea de reconocimiento sobre cada una de las caras detectadas en la imagen de la cámara web. El resultado de la detección se muestra en el cuadro de mensajes de la ventana principal, como muestra la Figura M15.

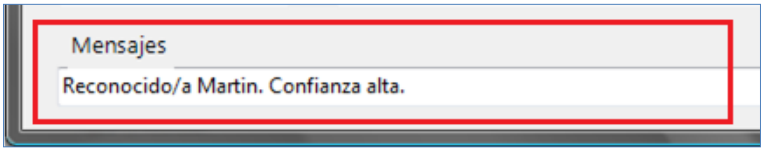


Figura M15. El cuadro de mensajes de la ventana principal muestra en tiempo real el resultado de los reconocimientos.

M.5 Entrenamiento

Las funcionalidades que ofrece la aplicación a través del menú de **Entrenamiento** son:

- ***Online***: se obtienen los datos para el reconocedor a partir de las caras que se vayan reconociendo a través de la imagen de la cámara web. Esta opción permite a su vez establecer el nombre de la persona con la que se va a entrenar al sistema.

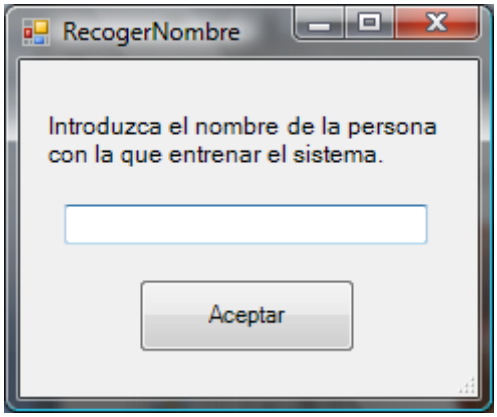


Figura M16. Ventana para la introducción del nombre de la persona con que entrenar al sistema.

Al finalizar el entrenamiento, se muestra una ventana informativa.

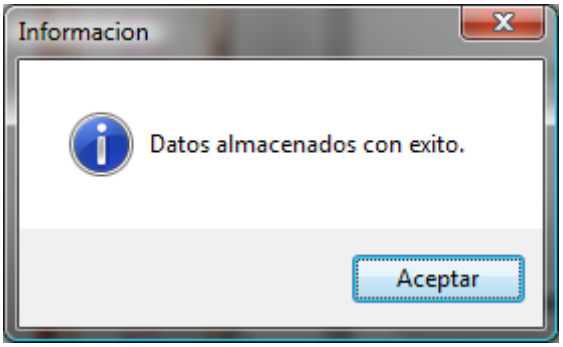


Figura M17. Ventana informativa al término del entrenamiento online.

- ***Offline***: los datos para el reconocedor se obtienen a partir de las fotos almacenadas en la base de datos de caras. Se le ofrece además al usuario la posibilidad de establecer el nombre de las personas con las que se va a entrenar al sistema.

Figura M18. Ventana para la introducción de nombres asociados a las caras en la base de datos usadas en el entrenamiento offline.

Si no se desea introducir nombres, el sistema asigna unos por defecto. Al finalizar el entrenamiento, se muestra un mensaje informativo.

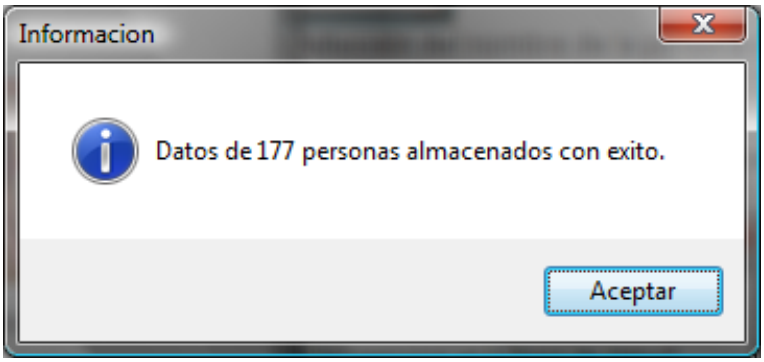


Figura M19. Mensaje informativo con el resultado del entrenamiento realizado.

- **Generar reconocedor**: el sistema, a partir de los datos recogidos durante el entrenamiento (*offline*, *online* o una mezcla de ambos), genera el fichero XML del reconocedor que será usado a partir de ese momento por la aplicación. Este proceso puede durar varios minutos dependiendo del número de caras de entrenamiento usadas por lo que antes de su inicio se pide confirmación al usuario.

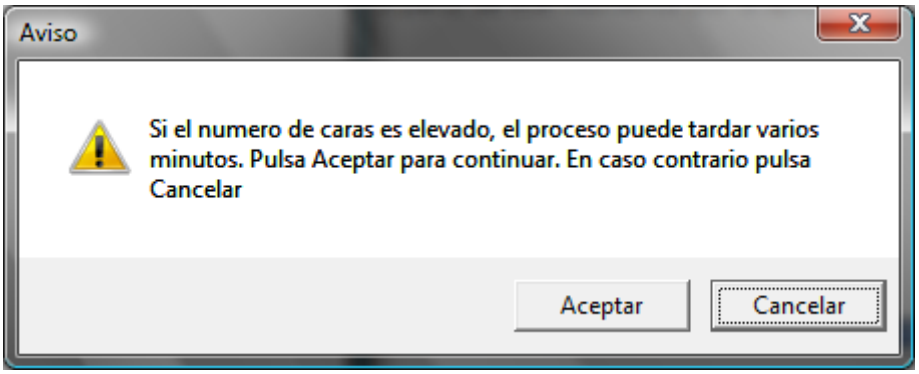


Figura M20. Ventana de confirmación de comienzo de generación del reconocedor.

La aplicación muestra una ventana informativa al finalizar el proceso.

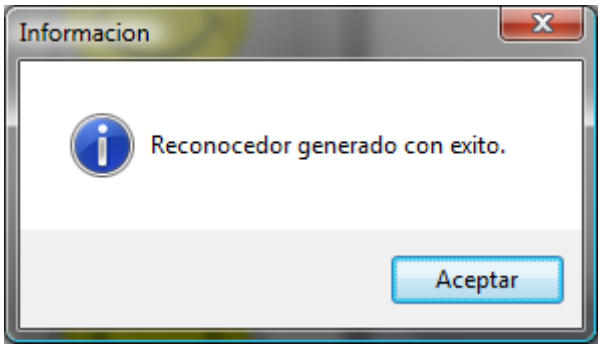


Figura M21. Ventana informativa al concluir el proceso de generación del reconocedor.

- **Reinicio reconocedor**: si por cualquier circunstancia es preciso crear un nuevo reconocedor desde cero, esta opción permite eliminar de manera automática los datos del anterior reconocedor.

- **Convertir | Imagen a JPG**: como se ha mencionado anteriormente, el formato de archivos de imagen establecido para esta aplicación es JPG por varias razones. Sin embargo, para dotar de mayor flexibilidad al sistema, se permite la conversión de varios tipos de formatos a JPG. Cada conversión crea una copia en JPG de la imagen de origen seleccionada por el usuario y la almacena en la base de datos de caras para que pueda ser usada por la aplicación.

- **Convertir | Imágenes a JPG**: en este caso, al sistema se le indica una carpeta de imágenes y todas las imágenes de formatos soportados son convertidas a JPG y almacenadas en la base de datos de caras.

M.6 Control

Las funcionalidades que ofrece la aplicación a través del menú de **Control** son:

- **Comprobar | Persona**: se pide al usuario que seleccione una imagen de una cara y se ejecuta la tarea de reconocimiento sobre esa cara haciendo uso del reconocedor previamente generado. **Es una de las principales características de esta aplicación** ya que permite conocer con resultados bastante notables si una persona determinada fue detectada en algún momento o no.

  El resultado de la ejecución, mensaje informativo previo e imágenes de prueba y más cercana, se muestran como indican las Figuras M22 y M23.

Figura M22. Mensaje informativo acerca del reconocimiento de la persona. Se muestra el nombre de la persona reconocida, el nivel de confianza y la fecha de la detección.

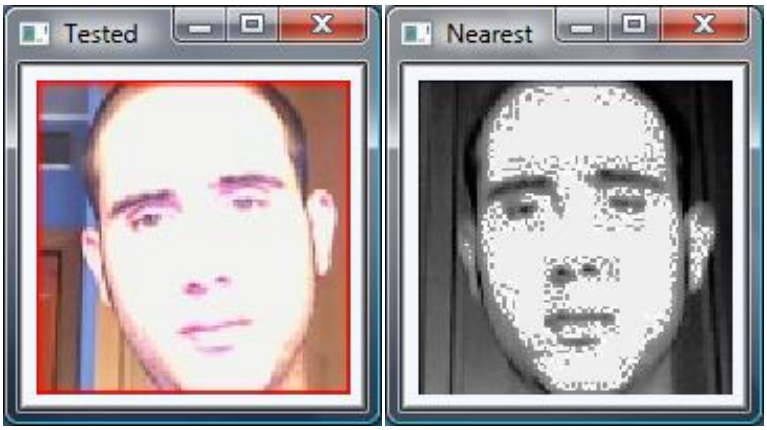


Figura M23. Resultado de un reconocimiento. A la izquierda, la cara a comprobar. A la derecha, la cara reconocida como la más parecida entre todas las usadas en el entrenamiento.

- **Comprobar | Personas (*debug*)**: se ejecuta la tarea de reconocimiento a todas fotos presentes en la capeta seleccionada por el usuario y almacena resultados en el archivo de formato CSV llamado **resultados_reconocimiento.csv**. Los resultados almacenados son la ruta de la carpeta, el nombre de cada imagen comprobada, el número de persona reconocida, su nombre y el nivel de confianza de la detección.

- **Últimas detecciones**: al igual que la ventana principal muestra las últimas cuatro detecciones realizadas, información que se va refrescando en tiempo real en el caso de que la cámara web se encuentre activa y detectando, las últimas ocho se muestran directamente con esta opción.

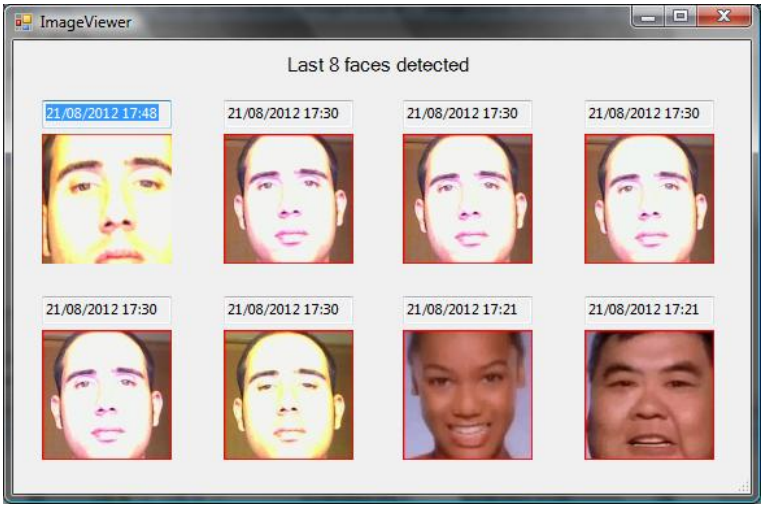


Figura M24. Últimas ocho caras detectadas por la aplicación junto con el instante de su detección.

- **Todas detecciones**: una de las principales funciones de control de personas es saber que personas han sido detectadas y cuando. Con esta funcionalidad, el usuario es capaz de ver con in interfaz muy sencillo y manejable todas las detecciones almacenadas por el sistema en la base de datos de caras de la aplicación. La Figura M25 ilustra el interfaz de la muestra de todas detecciones.

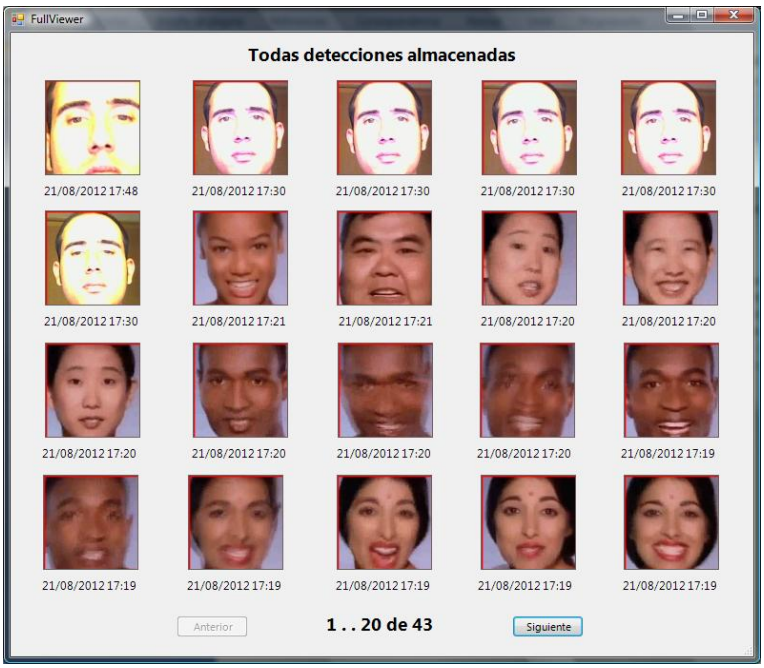


Figura M25. Detecciones 1 a 20 de las 43 presentes en la base de datos de caras en el momento de la ejecución de la opción para mostrar todas detecciones.

- **Modificar nombres**: permite al usuario establecer los nombres de las personas presentes en la base de datos de la aplicación para el entrenamiento. Esta opción es compatible con cualquier método de entrenamiento. El interfaz que se presenta al usuario es idéntico al de introducción de nombre en el entrenamiento *offline*.

- **Editar detecciones**: esta opción permite un acceso directo a la base de datos de caras para la edición de las imágenes a usar en el entrenamiento si se precisa. Es recomendable revisar las imágenes que se van a usar para el entrenamiento ya que cuanto mejor sea el entrenamiento más preciso y robusto se va a generar el reconocedor. Por lo tanto se recomienda eliminar de la base de datos imágenes que contengan falsos positivos o que añadan poca información al entrenamiento.

- **Ver registros | Registro detecciones**: el sistema registra todas las detecciones realizadas en el archivo **DetectionLog.txt** para revisión en caso necesario. A través de esta opción se accede directamente al contenido de ese registro.

- **Ver registros | Registro reconocimientos:** del mismo modo que con las detecciones, el sistema registra todos los reconocimientos llevados a cabo en el archivo **RecognitionLog.txt**. A través de esta opción se accede directamente al contenido de ese registro.

- **Ver registros | Resultados detecciones**: acceso directo al archivo de resultados de detecciones **resultados_deteccion.csv**. Este archivo almacena información de las detecciones cuando se usa la opción de detectar en modo *Debug*. Entre la información registrada se encuentra la ruta de la imagen analizada, el detector usado, los parámetros del detector así como el resultado obtenido y el tiempo de ejecución.

- **Ver registros | Resultados reconocimientos**: acceso directo al archivo de resultados de reconocimientos **resultados_reconocimiento.csv**. Este archivo almacena información de los reconocimientos cuando se usa la opción de comprobar un conjunto de imágenes. Entre la información registrada se encuentra la ruta de la carpeta analizada y el resultado obtenido de cada reconocimiento.

M.7 Ayuda

Las funcionalidades que ofrece la aplicación a través del menú de **Ayuda** son:

- **Manual de usuario**: acceso directo al este documento en formato HTML.

- **Acerca de…**: información relativa a la aplicación: versión, autor, etc.

Bibliografía

[1] P. Viola y M. Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Presentado en *Computer Vision and Pattern Recognition*, 2001.

[2] Mathew A. Turk y Alex P. Pentland. *Face Recognition Using Eigenfaces*. Presentado en *IEEE Conference on Computer Vision and Pattern Recognition*, 1991.

[3] Y. Freund y R. Schapire. *A decision-theoretic generalization of on-line learning and an application to boosting*. Presentado en *Computational Learning Theory: Eurocolt '95*, 1995.

[4] C. Papageorgiou, M. Oren, y T. Poggio. *A general framework for object detection*. Presentado en *International Conference on Computer Vision*, 1998.

[5] H. Rowley, S. Baluja y T. Kanade. *Neural network-based face detection*. Presentado en *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.

[6] H. Schneiderman y T. Kanade. *A statistical method for 3D object detection applied to faces and cars*. Presentado en *International Conference on Computer Vision*, 2000.

[7] Y. Freund y R. Schapire. *A short introduction to boosting*. Presentado en *Journal of Japanese Society for Artificial Intelligence*, 1999.

[8] M. Castrillón, O. Déniz, L. Antón y J. Lorenzo. *Face and facial feature detection evaluation*. Presentado en *International Conference on Computer Vision Theory and Applications (VISAPP)*, 2008.

[9] P. Viola y M. Jones. *Robust Real-Time Face Detection*. Presentado en *International Journal of Computer Vision*, 2004.

[10] M. Yang, D. Kriegman y N. Ahuja. *Detecting Faces in Images: A Survey*. Presentado en *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[11] Servo Magacine.
http://www.servomagazine.com/

[12] CiteSeer^{X}.
http://citeseerx.ist.psu.edu/index

[13] *Google* Académico.
http://scholar.*Google*.es/

[14] Gary Bradski y Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Septiembre 2008.

[15] Hannes Kruppa, Modesto Castrillón-Santana y Bernt Schiele. *Fast and Robust Face Finding via Local Context*. Presentado en *IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, 2003.

[16] Lienhart, R., Liang, L. y Kuranov, A. *A detector tree of boosted classifiers for real-time object detection and tracking*, 2003.

[17] *OpenCV* 2.1 C *Reference*.
http://OpenCV.willowgarage.com/documentation/c/index.html

[18] Análisis de Componentes Principales.
http://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales

[19] *Face Recognition* using *OpenCV*.
http://OpenCV.willowgarage.com/wiki/FaceRecognition

[20] *Face Recognition* Homepage.
http://www.face-rec.org/

[21] Página web de Shervin Emami.
http://www.shervinemami.info/

[22] *VeriLook Surveillance SDK*. *Face identification for video surveillance systems*
http://www.neurotechnology.com/verilook-surveillance.html

[23] Prueba χ^2 de Pearson.
http://es.wikipedia.org/wiki/Prueba_%CF%87%C2%B2_de_Pearson

[24] *Picasa* de *Google*.
http://Picasa.Google.com/

[25] *Apple – iPhoto*.
http://www.Apple.com/es/ilife/iPhoto/

[26] K. Etemad y R. Chellappa, *Discriminant Analysis for Recognition of Human Face Images*. Presentado en el *Journal of the Optical Society of America*, 1997.

[27] A.V. Nefian y M.H. Hayes III. *Hidden Markov Models for Face Recognition*. Presentado en *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 1998.

[28] T.F. Cootes, K. Walker y C.J. Taylor. *View-Based Active Appearance Models*. Presentado en *IEEE International Conference on Automatic Face and Gesture Recognition*, 2000.

[29] Página web del profesor ModestoFernando Castrillón Santana.
http://mozart.dis.ulpgc.es/Gias/modesto.html

[30] *Face Recognition databases*
http://www.face-rec.org/databases/

[31] Análisis de Discriminantes Lineales (inglés)
http://en.wikipedia.org/wiki/Linear_discriminant_analysis