



# ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

LOCALIZACIÓN DE REGIONES FACIALES EN VÍDEO EN  
TIEMPO REAL

Álvaro Olazarán Santesteban

Miguel Pagola Barrio

Pamplona, 28 julio 2010

## Contenido

INTRODUCCIÓN.....	4
VISION ARTIFICIAL .....	4
SISTEMAS DE DETECCIÓN BIOMÉTRICOS.....	5
SISTEMAS DE DETECCIÓN FACIAL .....	6
OPENCV .....	7
OBJETIVOS.....	8
DESCRIPCIÓN DEL PROYECTO .....	9
ALGORITMO DE EXTRACCIÓN DE MEDIDAS BIOMÉTRICAS FACIALES ...	9
CAPTURA DE FOTOGRAMAS.....	10
FASE DE SUAVIZADO.....	12
FILTRADO GABOR .....	12
UMBRALIZADO.....	17
DILATACIÓN .....	18
FILTRADO DE REGIONES .....	19
LOCALIZACIÓN DE ZONAS FINALES .....	20
EL MÓDULO DE FILTRADO DE PIEL.....	25
PRUEBAS REALIZADAS .....	27
CONCLUSIONES.....	30
LÍNEAS FUTURAS.....	31
APÉNDICE .....	32
MANUAL DEL USUARIO .....	32
INTERFAZ DE USUARIO.....	32
FUNCIONES Y CLASES IMPLEMENTADAS.....	35
PUESTA EN MARCHA DEL PROYECTO EN MICROSOFT VISUAL STUDIO	
2008 .....	39
BIBLIOGRAFÍA .....	49



# INTRODUCCIÓN

## *VISIÓN ARTIFICIAL*

La visión artificial o visión por computador es una disciplina cuyo objetivo es la extracción automática de información del mundo físico a partir de imágenes. Dentro de la visión artificial tiene un significativo papel el seguimiento automático de objetos en movimiento, ya sea en temas de control de calidad en industrias o de seguridad para identificación de terroristas, por citar solo algunos ejemplos representativos.

Se trata de un campo de estudio relativamente nuevo que comenzó a emerger aproximadamente a finales de la década de 1970. Cubre un amplio rango de temas que pertenecen también a otras disciplinas y no existe un problema estándar que defina qué debe resolver la visión artificial y cómo debe hacerlo. Por tanto, existen gran cantidad de maneras de resolver problemas tipo y estos suelen ser específicos a la tarea que realizan. Muchos de estos métodos están en fase de investigación pero cada vez más encuentran aplicaciones en productos comerciales que junto a otros sistemas forman entes capaces de resolver tareas más o menos complejas en campos como el de la medicina o el control de calidad en procesos industriales.

A lo largo de los últimos años se han desarrollado una gran variedad de técnicas y algoritmos que permiten resolver con mayor o menor grado de acierto diversos problemas de visión artificial, pero ¿qué es concretamente la visión artificial?

La visión por computador es una rama del extenso campo de la inteligencia artificial que toma una representación de la realidad (en forma de fotografías en 2D por ejemplo) con cierta información como las tonalidades de gris de una imagen, su contraste o los objetos que la forman para extraer datos de interés que puedan ser usados en aplicaciones automáticas programadas en un ordenador o implementadas en hardware. También la física es un campo cercano a la visión por computador en cuanto a que se aplica la teoría de la óptica para la captación y calibración de la realidad mediante cámaras con el proceso de calibración de estas que conlleva. La neurobiología también tiene relación con el campo que nos ocupa porque de ella se intenta tomar el modelo biológico de visión y el funcionamiento conjunto de ojos, neuronas y estructuras cerebrales para el procesamiento de la imagen en humanos y animales. Aparte de estas disciplinas, también el procesamiento de señales, la estadística, optimización o geometría tienen aspectos en común con la visión artificial.

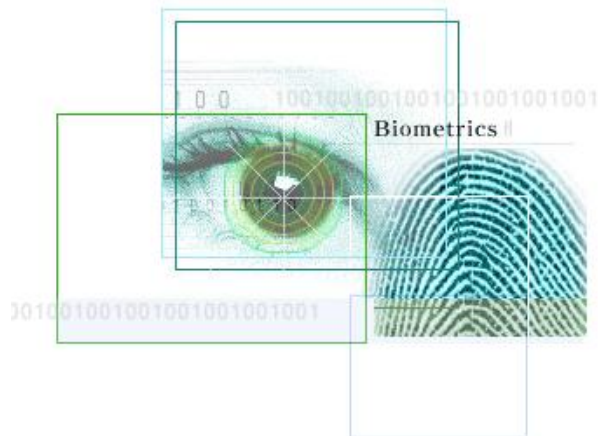
Existen diferentes líneas de investigación dentro del campo de la visión artificial, como son la localización, detección, segmentación y reconocimiento de objetos en imágenes y su evaluación; registrar diferentes imágenes de una misma escena o patrón haciendo concordar el objeto en todas ellas; seguimiento de objetos en una secuencia de imágenes (por ejemplo en vídeo en tiempo real); búsquedas de objetos en bases de datos de imágenes o mapeos de escenas con el fin de obtener modelos tridimensionales de estas. Estas y otras líneas de investigación podrían englobarse en: procesamiento de imagen o métodos de transformación de una imagen en otra realizando procesados intermedios como aumento de contraste, extracción de bordes, reducción de ruido o

transformaciones geométricas con imágenes; visión estéreo o reconstrucción u obtención de información de escenas 3D a partir de varias imágenes; visión máquina o procesamiento de imagen en tiempo real mediante hardware o software para control de robots o sistemas de medición; reconocimiento de patrones u obtención de información a partir de señales de múltiples tipos.

La disciplina de la visión por computador tiene una gran cantidad de aplicaciones. Una de las que más está emergiendo en los últimos años es la aplicación al campo de la medicina para procesamiento médico de imágenes de ayuda en diagnósticos. Normalmente se trata de imágenes de microscopio, de rayos X, ultrasónicas o tomografías y se pretende detectar tumores, arteriosclerosis u otras patologías. Como ya se ha comentado, también la visión por computador tiene un papel protagonista en la industria, sobre todo en procesos de fabricación o montaje en los que se busca un control de calidad automático. Asimismo, se usa para ayudar a los robots a encontrar y detectar la orientación de los objetos con los que tienen que tratar. También tiene cabida en el campo militar y se usa en la detección de enemigos o en la guía de misiles. Una de las más modernas aplicaciones de la visión por computador es su uso para dirigir vehículos de manera automática. También ha encontrado múltiples aplicaciones en el campo de la biometría para identificación de individuos.

## *SISTEMAS DE DETECCIÓN BIOMÉTRICOS*

En la actualidad es necesaria la identificación de individuos constantemente. Se realiza cuando se compra con tarjeta, cuando se viaja, cuando se accede a instalaciones, al sacar dinero del banco, al navegar por la web, etc. Desde el nacimiento las personas poseen rasgos identificativos que las diferencian del resto. La biometría estudia estos rasgos característicos para identificar a los individuos. Así, se puede diferenciar a una persona por su voz capturando muestras de sus voces, extrayendo las características inherentes a cada una y, siguiendo un método de aprendizaje por ejemplo, ser capaz de determinar para una muestra nueva a qué individuo pertenece. Existe una gran cantidad de métodos de reconocimiento biométrico. Por citar algunos se pueden destacar el reconocimiento de iris, el de retina, reconocimiento vascular (según el tramado de las venas de la palma de la mano a través del reflejo de ondas de frecuencia corta), reconocimiento de huella dactilar, de ADN, de uña, de manejo del ratón, del pulso sanguíneo, de radiografías dentales, de marcas de mordida y otras muchas tan variadas como variopintas.



El objetivo de este proyecto es implementar un sistema de detección biométrico que pueda servir posteriormente para diferenciar personas, una vez que se han detectado sus rasgos identificativos. Para ello no se va a usar ninguna de las técnicas

mencionadas, sino que se basa en la que los humanos usan para reconocerse, es decir, en la forma y rasgos de la cara. Aun así, los rasgos de la cara de cada individuo son muchos y complejos de estudiar en profundidad. Lo que se va a hacer es llevar a cabo la detección de la posición de los ojos, nariz y boca en un rostro humano.

## SISTEMAS DE DETECCIÓN FACIAL

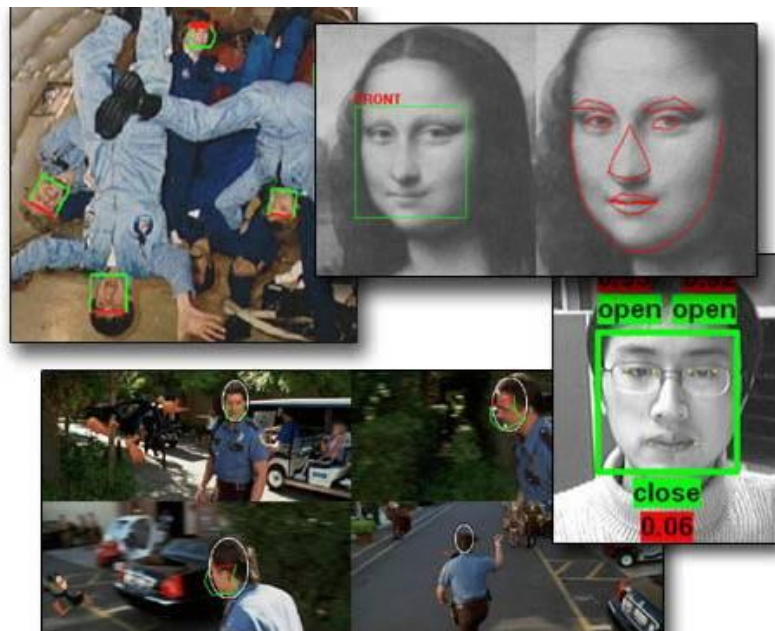
Un sistema de detección facial es una aplicación para localizar caras humanas en imágenes digitales. Su objetivo es determinar qué parte o región de una fotografía es una cara y distinguirla mediante algún método de marcado, del resto de la imagen. Se puede englobar en la disciplina de detección de clases de objetos, en la que se buscan localizaciones y tamaños de objetos que pertenecen a una clase dada, como por ejemplo señales de tráfico, personas, códigos, figuras geométricas, etc.

En sus orígenes, la detección facial se centraba en hallar regiones que pertenecieran a caras con una orientación frontal, mientras que algoritmos más modernos permiten o buscan el poder detectar regiones faciales con una determinada rotación tanto del eje vertical como del horizontal.

La detección de regiones faciales está muy relacionada con la biometría, en cuanto que esta última utiliza detectores de caras para posteriormente extraer, procesar sus rasgos y obtener un resultado concreto como averiguar el nombre y apellidos de la persona que se halla situada enfrente de una cámara.

Hay varios factores que pueden afectar a los resultados de un sistema de detección de caras si este no es lo suficientemente sofisticado. Uno de ellos es la posición de esta respecto del plano de grabación ya que puede presentarse una cara muy cercana al borde o parcialmente oculta. Se dificulta asimismo el proceso de localización si se “viste” la cara con

objetos que la tapan parcialmente como sombreros, pañuelos, gafas, barba, etc. También las condiciones de iluminación o las características de la cámara, e incluso el cómo está configurada, pueden afectar notablemente a los resultados si no se han estudiado con la suficiente profundidad.



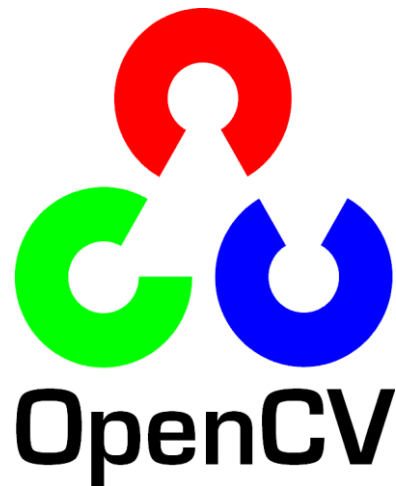
En los últimos años ha habido un gran desarrollo de los sistemas de detección facial para integrarlos en el software o hardware de cámaras fotográficas de manera que casi todas las marcas distribuyen ya sistemas de localización facial para ayudar al enfoque y al encuadre y por lo tanto para producir mejores fotografías.

## *OPENCV*

Para llevar a cabo este proyecto se ha dado uso a la librería OpenCV. El nombre OpenCV proviene de Open Source Computer Vision y es una librería de software libre originalmente desarrollada por Intel en 1999 que provee al programador de funciones y algoritmos relacionados con el área de la visión por computador y por tanto es útil para el procesamiento de imágenes en tiempo real, como en el caso que nos ocupa. OpenCV es muy eficiente debido al uso que hace de las Primitivas de Rendimiento Integradas de Intel, conjunto de rutinas de bajo nivel diseñadas para procesadores Intel. Se distribuye bajo una licencia BSD y se puede usar tanto para fines académicos como comerciales. OpenCV contiene más de 500 funciones optimizadas que, al estar programadas en C, C++ o Python, da la posibilidad de usarlas en múltiples plataformas.

La primera versión de OpenCV (alfa) fue lanzada al público en 2000 en la IEEE Conference on Computer Vision and Pattern Recognition. Entre 2001 y 2005 se liberaron cinco versiones beta hasta que en 2006 nació la versión 1.0. La versión que se ha usado en este proyecto es la 1.1 pre-release, lanzada en octubre de 2008.

Las áreas de aplicación de OpenCV son, entre otras, herramientas de manejo de imágenes 2D o 3D, sistemas de reconocimiento, interfaces hombre-máquina, movimiento de robots, estudio de trayectorias, identificación de objetos, segmentación, visión estéreo, seguimiento de objetos, aprendizaje, etc.



La librería está escrita en su gran mayoría en lenguaje C, lo que la hace portable a otras plataformas. Aun así, se han desarrollado también “conversores” para lenguajes como C#, Python, Ruby o Java. OpenCV tiene versiones para FreeBSD, Linux, Mac OS y Windows, siendo esta última la usada en este proyecto.



## OBJETIVOS

El objetivo de este proyecto de fin de carrera es llevar a cabo la implementación en un lenguaje eficiente de un programa software capaz de detectar zonas faciales de personas en vídeos a color tomados en tiempo real de una cámara.

Para ello se estudiará el API de OpenCV, una librería de visión por computador optimizada para procesadores Intel que es la que se va a usar para tratar las imágenes del vídeo de salida que el usuario visualizará así como la captura de imágenes de la videocámara y alguna parte del procesamiento. Por tanto la realización del proyecto requiere de un estudio previo del API y por lo tanto de cómo usar las funciones que OpenCV provee y cómo aprovechar sus características de procesamiento de imagen. El proyecto estará implementado en lenguaje C++ aunque muchas de las funciones que OpenCV provee están escritas en C. Se ha escogido este lenguaje de programación por su mayor eficiencia en comparación con otros lenguajes, al ser esta característica una de las que más influyen cuando se trata de un programa de procesamiento en tiempo real.

La primera fase del proyecto trata de la implementación de un software capaz de determinar ciertas características biométricas como son las posiciones de ojos, nariz y boca de caras humanas. Para llevar a cabo esta parte se ha estudiado un proyecto previo llamado “Detección de la posición de ojos, nariz y labios dentro de la imagen de una cara” [1], cuyo objetivo era el mismo pero basado en fotografías en vez de en vídeos e implementado en lenguaje Matlab en vez de en C++.

La segunda fase del proyecto es la integración del módulo de detección biométrico con un módulo de detección de regiones de piel realizado en otro proyecto fin de carrera previo [3].

El proyecto deberá localizar en la medida que sea posible una región facial tomada de vídeo en tiempo real y realizará un seguimiento de esta, de forma que cuando se mueva el programa sea capaz de seguirla mostrando al usuario un recuadro que identifique la zona. Aparte de recuadrar la zona facial, el programa identificará y recuadrará también las regiones de los ojos, la base de la nariz y la boca.

Asimismo, se proveerá al usuario de una utilidad en forma de GUI para calibrar ciertos parámetros del programa. Téngase en cuenta que cada cámara está configurada de diferente manera y el lugar en el que esta se sitúa y sus condiciones ambientales de iluminación también varían mucho. Por ello, se podrán ajustar determinadas variables que hagan de la localización facial un proceso con mejores resultados.

La implementación, como se ha dicho, se hará en lenguaje C y C++ mediante el software de desarrollo Microsoft Visual Studio 2008.

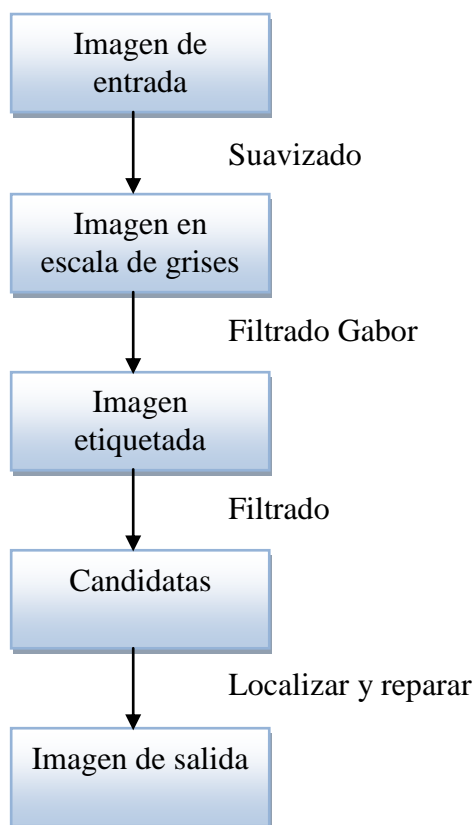


## DESCRIPCIÓN DEL PROYECTO

Como se ha mencionado en la sección objetivos, este proyecto se fundamenta en [1], un PFC llamado “Detección de la posición de ojos, nariz y labios dentro de la imagen de una cara” de Javier Lapieza Fernández, que a su vez se basa en [2], el artículo “Automatic extraction of head and face boundaries and facial features” de Frank Y. Shih y Chao-Fa Chuang. En este proyecto se ha reimplementado en C++ el programa previamente realizado en Matlab para mejorar su eficiencia y se ha cambiado el tratamiento de fotografías por el de vídeos. La eficiencia es un aspecto indispensable ya que se pretende que se procesen la mayor cantidad de fotogramas por segundo posibles y porque además de este algoritmo de reconocimiento biométrico de los rasgos faciales, se ha usado otro algoritmo previamente reconocido de zonas de piel humana. A continuación se va a describir el funcionamiento del método de Frank Y. Shih y Chao-Fa Chuang, pero adaptado a la detección de regiones de ojos, nariz y boca en vídeos a color, lo que es el fundamento del proyecto.

### *ALGORITMO DE EXTRACCIÓN DE MEDIDAS BIOMÉTRICAS FACIALES*

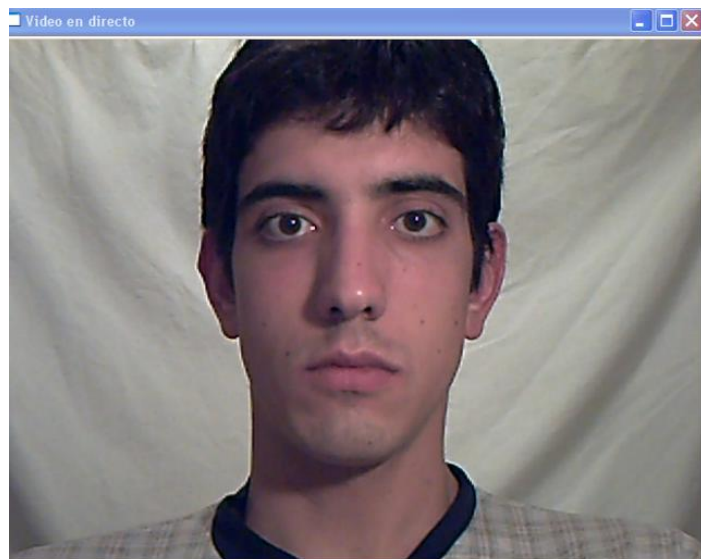
A grandes rasgos, el esquema que sigue este proyecto es el del algoritmo que se propone en [1] y que se muestra a continuación:



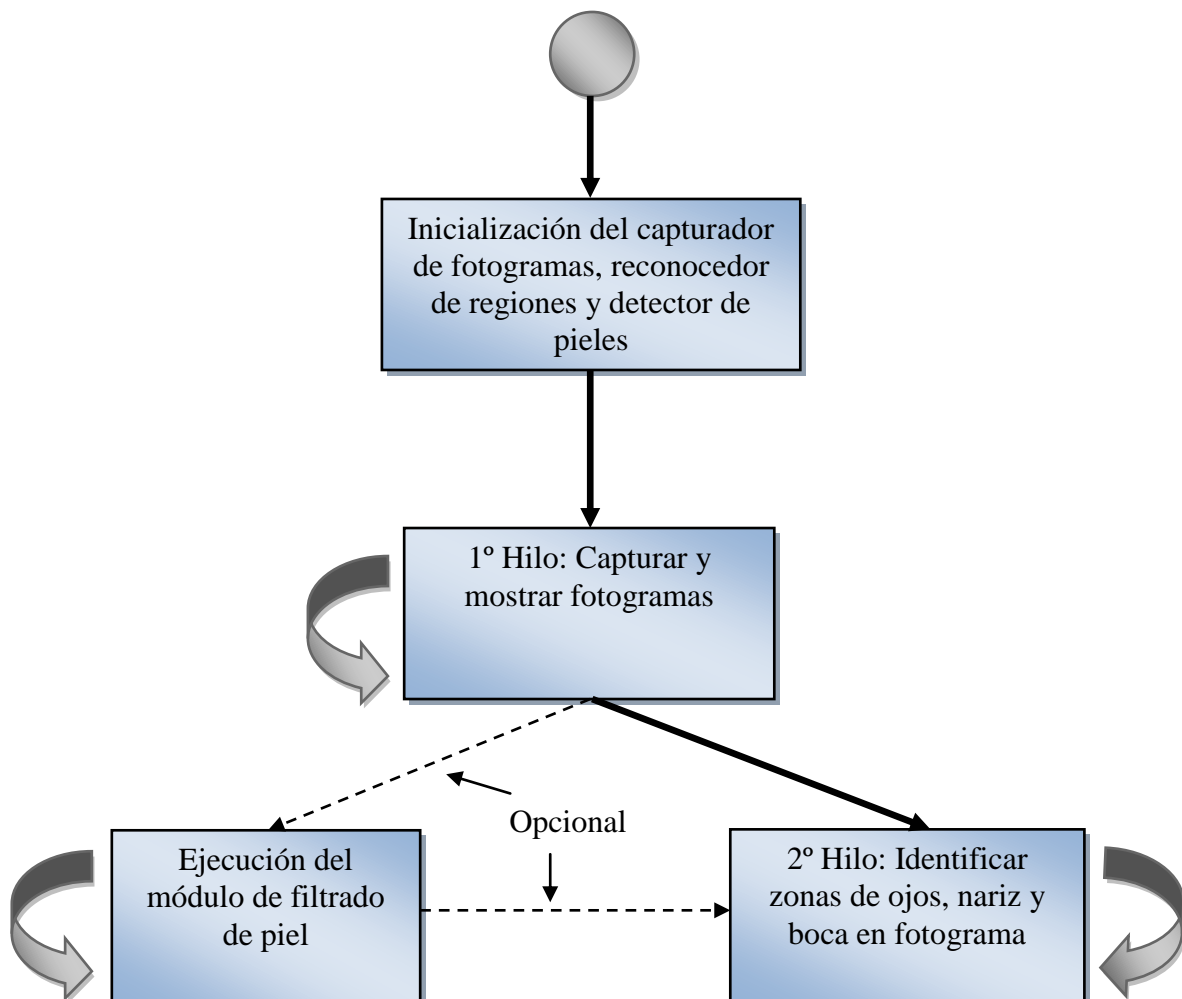
Aunque en este proyecto se tiene de entrada un vídeo y no una fotografía, la idea y la aplicación del algoritmo de [1] no varían. Lo que se hace es ir tomando cada fotograma aplicándole posteriormente el algoritmo, para acto seguido aplicárselo al fotograma actual del vídeo. En las siguientes secciones se explica con detalle cada una de las fases por las que pasa cada uno de los fotogramas del vídeo de entrada a la aplicación. A grandes rasgos, el algoritmo se compone de cuatro fases por las que cada fotograma debe pasar. La primera de ellas aplica un filtro de suavizado en la imagen para “disimular” el posible ruido que exista en esta y que de este modo no afecte en fases posteriores. Una vez se tiene la imagen filtrada y en escala de grises se aplica el filtro de Gabor. Con esta fase se pretenden resaltar las zonas horizontales de la imagen, que serán tomadas como posibles ojos, nariz o boca. Sin embargo, pueden existir multitud de zonas horizontales que por su tamaño o posición no deberían ser candidatas a zonas faciales. De este filtrado de regiones se encarga la tercera fase, para, en la última de ellas, y una vez se tienen las zonas candidatas, realizar el algoritmo de detección y reparación. Su función es la de obtener las zonas que, comparadas con unos patrones de medidas faciales estándar, den la mayor probabilidad de ser las zonas faciales reales. Además, se han añadido algunas etapas que ayudan a mejorar los resultados del algoritmo, como una fase de dilatación de regiones que más adelante se comentará.

## CAPTURA DE FOTOGRAMAS

La velocidad de procesamiento de los fotogramas, por supuesto, depende del hardware en el que se ejecute el programa, entre otros factores. En este proyecto hay que diferenciar entre el número de fotogramas capturados por segundo y el número de fotogramas procesados por segundo. Por un lado, el programa captura y muestra fotogramas de la videocámara casi a la misma velocidad a la que esta los graba. Por otro lado, el programa procesará el mayor número de fotogramas que pueda siendo esta una tarea secuencial, es decir, cuando se haya procesado un fotograma se pasará a procesar el último que haya capturado, no el siguiente al que se acaba de procesar. Es por este motivo que los rectángulos que marcan las zonas faciales se refrescan de manera más lenta que el vídeo mostrado. Para llevar a cabo más de una tarea de forma simultánea, como la captura en tiempo real por un lado y el procesamiento por otro, se han usado hilos POSIX (Pthreads) para Win32. De esta manera, al ejecutar el programa se crea primeramente un hilo encargado de obtener fotogramas de la videocámara y mostrarlos en tiempo real en



una ventana. Seguidamente se crea un segundo hilo que corre paralelo al primero y que se encarga de ejecutar sobre el fotograma actual el algoritmo de localización de ojos, nariz y boca. Por supuesto, el código del programa sigue aunque se hayan creado dos hilos para dos tareas diferentes, por lo que además de estos, se ejecutan las instrucciones del algoritmo de filtrado de piel si se requiere. En este caso, se toman las imágenes de la cámara, se filtra la piel y este fotograma se envía al hilo de detección de zonas. Al estar estructurado el programa en distintas partes que se ejecutan sobre distintos hilos permite dar una sensación al espectador de reconocimiento en tiempo real. Uno de los primeros problemas que se tuvieron tenía relación con este aspecto, ya que en pantalla la velocidad de visualización de los frames que la cámara grababa era la que permitía el algoritmo de reconocimiento de zonas. Es decir, se mostraban tantos fotogramas por segundo como veces por segundo se ejecutaba el algoritmo. En un ordenador potente como en el que se ha programado el proyecto no daba demasiada sensación de “parones” en el vídeo que grababa al usuario y mostraba los recuadros. Sin embargo, en un ordenador estándar la sensación no era de tiempo real. De esta manera se decidió dividir el programa en hilos, y uno de ellos es el encargado de mostrar las imágenes que la cámara graba, prácticamente en tiempo real mientras en otro hilo y a distinta velocidad, se generan los rectángulos que recuadran las zonas detectadas.



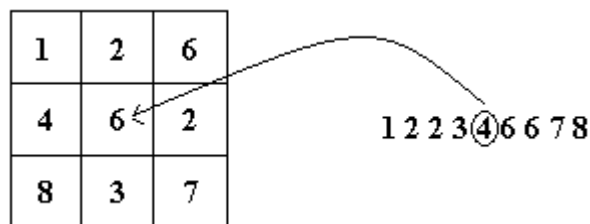
La fase de captura en este proyecto se ha realizado usando una cámara Toshiba Camileo conectada al puerto USB de un PC a modo de cámara web. Este modelo graba vídeo a 30 fps (fotogramas por segundo) con un tamaño de fotograma de 640 x 480 píxeles. Sin embargo, como es obvio, el algoritmo no se ha implementado teniendo en mente únicamente este tipo de cámara sino que puede funcionar con cualquier tamaño de fotograma.

## FASE DE SUAVIZADO

El primer paso que realiza el algoritmo sobre el fotograma del vídeo es la conversión a 256 niveles de gris ya que se está grabando en un formato de imagen a color como es RGB. Para realizar este paso, la librería de visión por computador OpenCV dispone de la función `cvCvtColor()`.

Una vez que se tiene el fotograma en escala de grises se puede proseguir con la fase de suavizado, que consta de un filtrado de la mediana 5x5. Este filtro de la mediana tiene un efecto de suavizado en la imagen y sirve para eliminar el ruido que pudiera haber y que en ocasiones pudiera confundir al programa. Lo que hace el filtro es cambiar cada uno de los píxeles de la imagen por la mediana de los que le rodean y que están a una distancia determinada del primero. En el caso del filtro de la mediana 3x3, se toma una ventana cuadrada de 3 píxeles de lado que se va desplazando a lo largo de todo el fotograma realizando los cálculos e intercambiando los píxeles correspondientes. Para ello se sustituye el nivel de gris del píxel del centro por la mediana de los niveles de gris de los píxeles de toda la ventana. La mediana de un conjunto de valores es aquel valor que cumple que la mitad de los elementos del conjunto son mayores o iguales que él y la otra mitad menores o iguales.

Ejemplo:

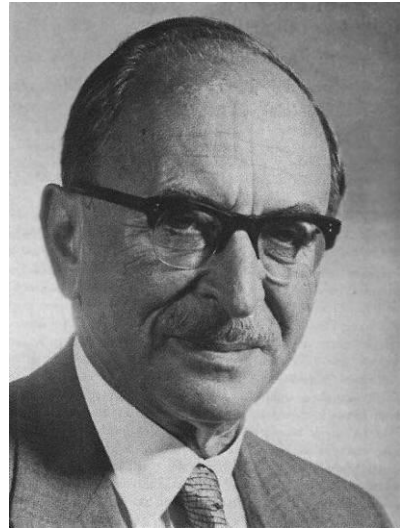


## FILTRADO GABOR

La fase de la aplicación del filtro Gabor es crítica y su buen uso y configuración se traducirán en malos o buenos resultados finales. Esta fase tiene como fin el resaltar las zonas con dirección horizontal en la imagen. Como se sabe, el objetivo de esta parte

del proyecto es localizar tanto los ojos como la nariz y boca de personas que sean grabadas por una cámara. Los ojos y la boca son zonas con tendencia horizontal y la base de la nariz lo es también. Evidentemente, si el individuo que está siendo grabado cambia lo suficiente la orientación de su cabeza, esta técnica de filtrado Gabor para destacar zonas horizontales dejará de funcionar con corrección. Ya que el uso de este filtro es vital en este proyecto, a continuación se estudiará y se mencionarán sus características y utilidades. Comencemos por un breve recordatorio de la vida y méritos del creador del filtro que nos ocupa, el famoso físico Dennis Gabor.

El filtro de Gabor toma el nombre de su creador, Dennis Gabor (Budapest, 5 de junio de 1900 – Londres, 9 de febrero de 1979), físico húngaro y premio Nobel de Física en 1971. Dennis Gabor fue el inventor de la holografía, desarrolló el filtro de Gabor y fue el padre de numerosos trabajos científicos relacionados con la teoría de la comunicación, la óptica física o la televisión en color, entre otros. Además, publicó varios artículos y ensayos sobre la influencia de la tecnología en la sociedad moderna. Por su excepcional carrera recibió además la Medalla de Honor IEEE en el año 1970. Como se ha dicho, un año después recibió el premio Nobel de Física por la invención y desarrollo de la holografía. También ha publicado libros como *La invención del futuro* (1963), *Innovaciones: científicas, tecnológicas, sociales* (1970) o *La sociedad madura* (1972).



El filtro 2D de Gabor es un filtro lineal principalmente usado en procesamiento de imagen como en el caso que nos ocupa, que principalmente consta de una función sinusoidal multiplicada por una gaussiana definida por la fórmula que sigue a continuación:

$$g_{\lambda, \theta, \psi, \sigma, \gamma}(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

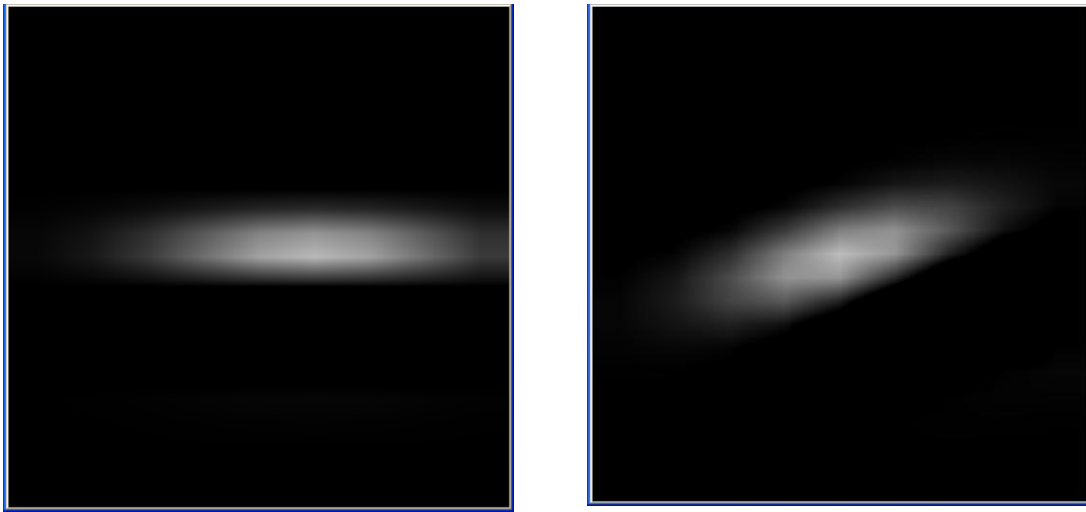
donde

$$\begin{aligned} x' &= x \cos \theta + y \sin \theta \\ y' &= -x \sin \theta + y \cos \theta \end{aligned}$$

El filtro de Gabor se puede clasificar como un filtro de tipo paso banda, esto es, aquel que permite pasar un determinado rango de frecuencias o valores y atenúa el paso del resto. En nuestro caso, lo que deseamos mantener son las formas o regiones horizontales de la imagen, filtrando aquellas formas cuya dirección se aleje de la horizontal. Para ello, como se ha dicho, se aplica a la imagen un filtro obtenido del producto de una función gaussiana con una sinusoidal. Esta función sinusoidal afecta al dominio frecuencial pero abarca todo el dominio espacial por lo que Gabor introdujo la envolvente gaussiana para localizar su función tanto en el dominio espacial como en el frecuencial.

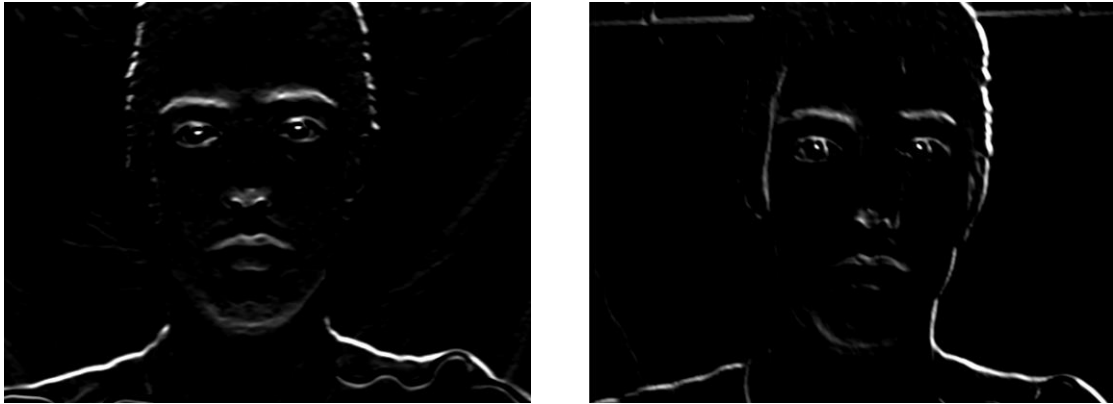
Para llevar a cabo esta fase, se ha implementado un filtro de Gabor configurable y visible en una ventana en la que se puede ajustar según lo deseado. Las múltiples personas y entornos en los que el programa se puede ejecutar requieren que este sea configurable, al menos para la primera vez que se ponga en funcionamiento. No es lo mismo un entorno con múltiples objetos detrás del individuo que un entorno homogéneo, ni que el programa se use para detectar caras cercanas o más lejanas. Es por ello que el usuario tiene libertad de configurar el filtro Gabor como se desee y de este modo sacar el mejor partido al programa para obtener los resultados óptimos.

Así pues, se puede modificar la orientación del filtro, es decir, el parámetro  $\theta$ , de manera que el ángulo de este con la horizontal cambie. Se especifica en grados y toma valores entre 0 y 360. Veamos algún ejemplo:



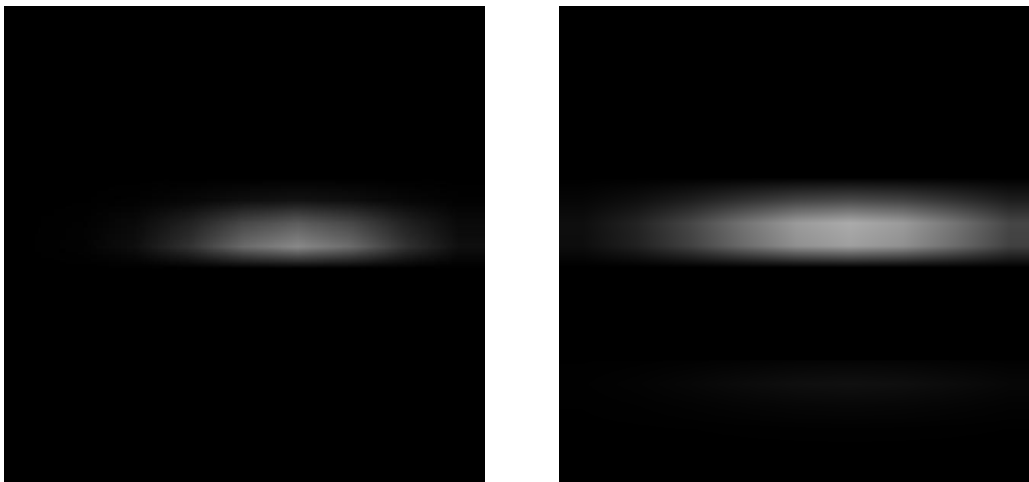
En las imágenes podemos ver un mismo filtro Gabor con distinto parámetro  $\theta$ . En el filtro de la izquierda toma el valor  $0^\circ$  y en el de la derecha  $45^\circ$ . Como se puede intuir, el primer filtro resaltará las zonas horizontales y el segundo las diagonales ascendentes. Veamos su efecto en la práctica:





En los cuadros de encima se muestran cuatro imágenes filtradas con un filtro Gabor. Por supuesto, cada una de ellas con el filtrado configurado de diferente forma. La imagen superior izquierda está procesada con un filtro Gabor cuyo parámetro  $\theta$  (que como se sabe indica su orientación) está ajustado a  $0^\circ$ . Esto da como resultado el resaltado de zonas verticales como el perfil de la cabeza y el cuello, los laterales del tabique nasal y la eliminación de zonas horizontales como la boca, las cejas o la línea de los hombros. La segunda imagen (superior derecha) tiene aplicado un filtro con una orientación de  $45^\circ$ . Esto hace que se remarque el lateral izquierdo de la cabeza y en menor medida otras facciones. La imagen inferior derecha es la opuesta a esta última ya que tiene un filtrado a  $135^\circ$  lo que provoca la situación contraria. Por último, la imagen inferior izquierda tiene un filtrado a  $90^\circ$  que resalta las zonas horizontales como las cejas, ojos, base de la nariz, boca o la línea de los hombros. Este último caso es el que se ha usado en el proyecto para resaltar las zonas horizontales y desechar, en la medida de lo posible, el resto.

Otro de los parámetros configurables del filtro Gabor es su longitud de onda o parámetro  $\lambda$ , que modifica el tamaño de este. Veamos dos ejemplos del mismo filtro Gabor con distinto tamaño y los resultados de su aplicación a dos fotografías.

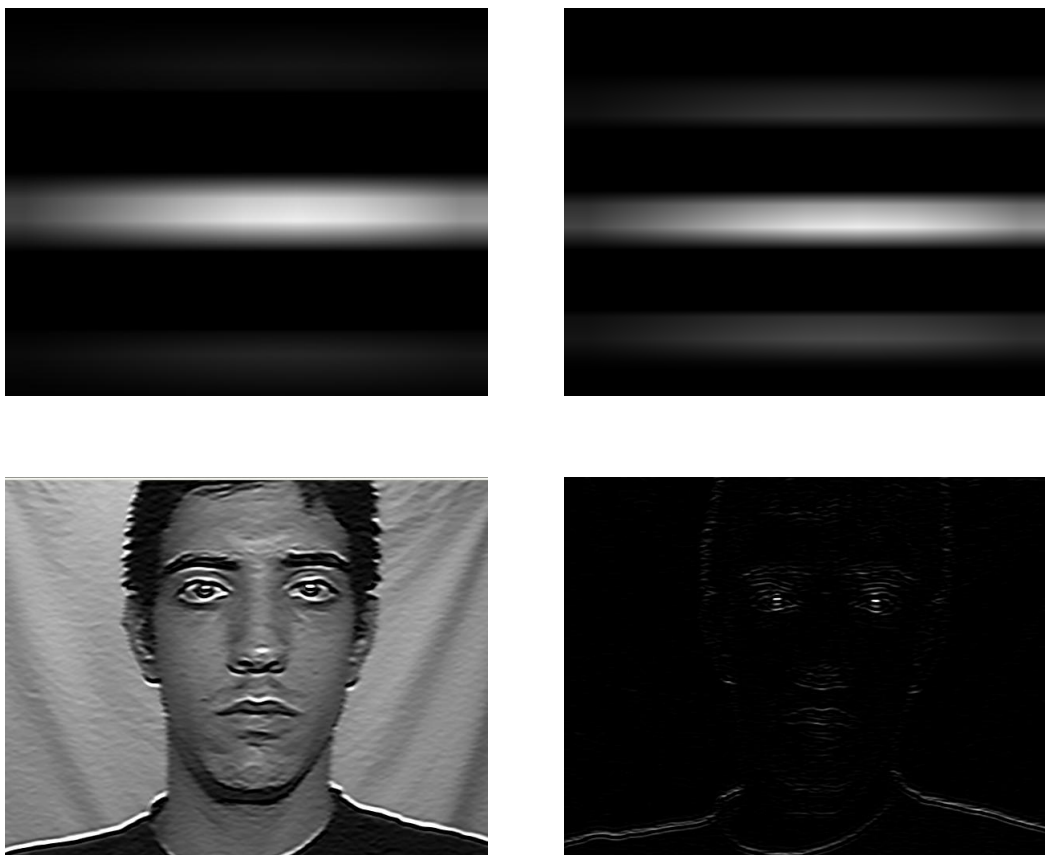




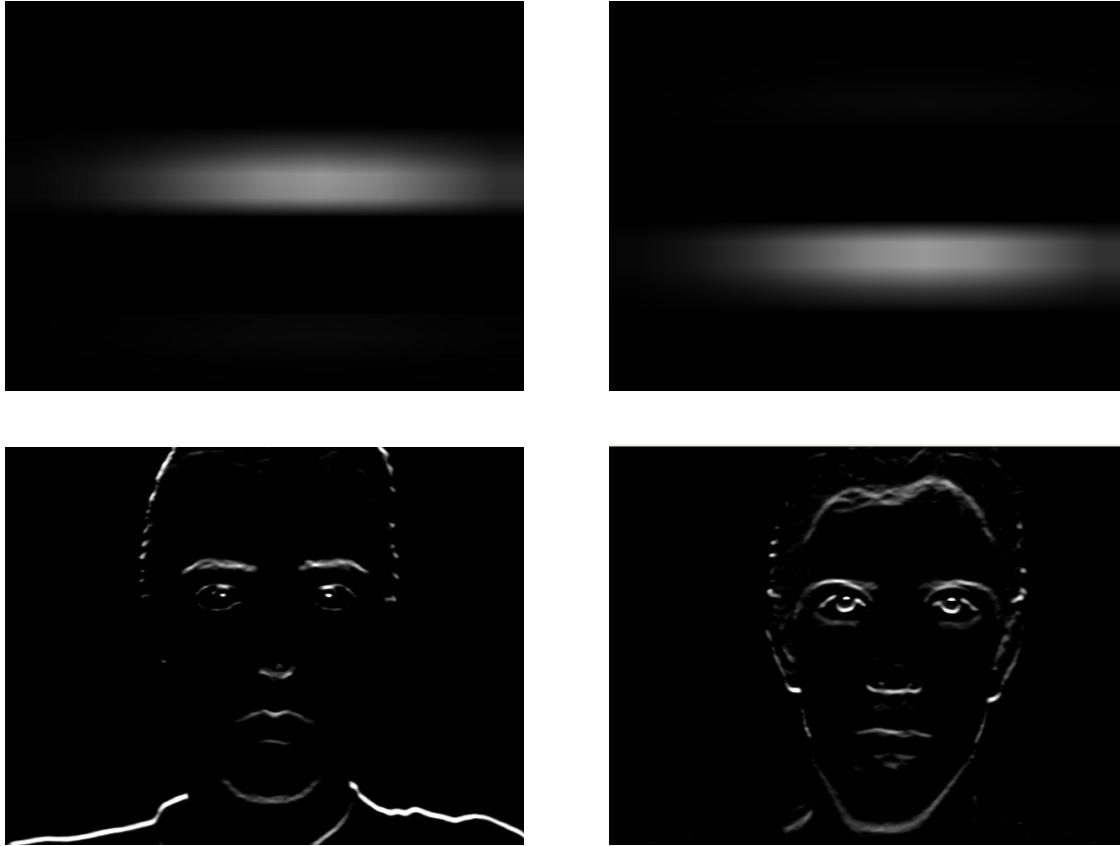
En ellas se aprecia cómo el filtro de la izquierda es más pequeño que el de la derecha. Esto provoca, como se ve a continuación, que las imágenes resultantes del filtrado tengan las zonas horizontales distinto grosor e intensidad.



También es modificable en el filtro Gabor su número de “crestas” y “valles”. Veamos un ejemplo debajo.



Por último, el software permite variar el parámetro  $\psi$  que afecta a la función coseno y que provoca un “desplazamiento” en las ondas del filtro de Gabor.

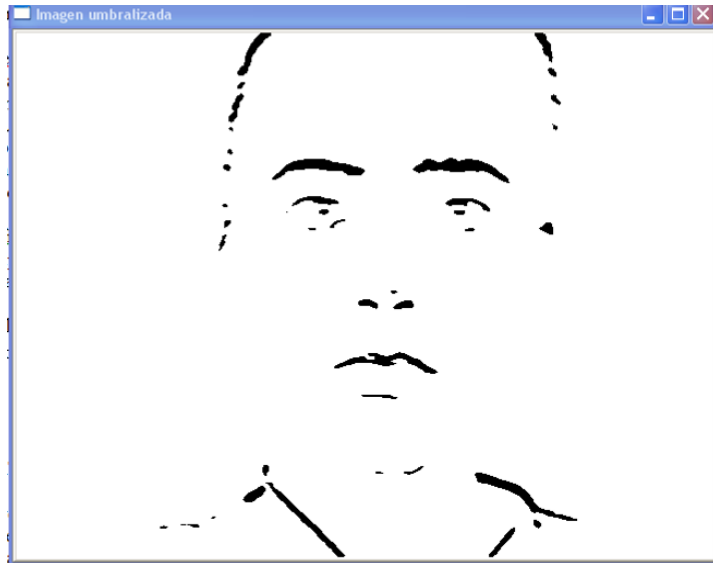


Como se puede ver, el filtro de la izquierda, que se ha desplazado hacia arriba, resalta bordes externos y el filtro de la derecha, desplazado hacia abajo, resalta los bordes internos de las zonas horizontales de la imagen. Como se ha dicho, mediante la ventana de configuración del filtrado Gabor se pueden calibrar los cuatro parámetros mencionados para adecuar el filtro al entorno o al tipo de objetos a filtrar. La varianza define el tamaño del filtro, la pulsación el número de “crestas” y “valles”, la fase define el ángulo o la orientación del filtro (en nuestro caso es necesario configurarlo a  $90^\circ$  como se ven en la imagen) y el parámetro psi define la posición del filtro verticalmente.

## UMBRALIZADO

La fase de umbralizado busca obtener una imagen en blanco y negro a partir de la que se tiene en 256 niveles de gris. No es una tarea fácil, ya que es necesario el obtener un nivel de gris dentro de la escala de 0 a 255, que marque el límite entre blanco o negro

en la imagen resultado. Es decir, la fase de umbralizado contiene un número entre 0 y 255, tal que todos los píxeles con nivel de gris inferior se conviertan en negros (0) y el resto pasen a ser blancos (255). De esta manera, el fotograma pasa a estar compuesto por zonas de colores blanco o negro. Debido a las múltiples características de los diferentes entornos en los que se ejecute el programa, se ha dejado al usuario la opción de seleccionar el umbral deseado. No obstante, previamente a esta fase, a continuación del filtrado Gabor, se ajustan los niveles de gris de la imagen resultante de manera que estos se extiendan desde el 0 hasta 255 mediante un reescalado en el cual el valor más bajo pasa a tener el valor 0 y el más alto se convierte en 255. Los valores intermedios se interpolan según la nueva escala. De esta manera, la elección según el entorno de la cifra de umbral no es tan significativa como en un principio pudiera parecer y normalmente será suficiente usar la cifra propuesta inicialmente en la pantalla de control, es decir, 173.



## DILATACIÓN

Una vez que se tiene la imagen filtrada mediante el filtro Gabor adecuado, se pasa a la fase de dilatación. La fotografía muestra en este punto regiones en tonos de gris pero muchas de ellas son pequeñas o formadas por varias zonas que, si en vez de esta fase se escogieran directamente las diferentes regiones, daría lugar a muchas y a veces pequeñas zonas como son las áreas de los ojos. Es por ello que el aplicar una dilatación en este punto mejora los resultados del proceso. De esta manera se forman regiones más concretas y se disminuye normalmente su número. También ayuda a agrandar las zonas finales como la boca o base de la nariz, que, de otra forma, podrían encuadrarse como líneas horizontales estrechas sin tener en cuenta por ejemplo los dos labios de la boca. De tal modo que se tiende a agrupar zonas próximas en esta fase y a eliminar huecos interiores. Expuesto de manera formal, la dilatación se puede expresar como:

$$A \oplus C = \{x \mid (C)_x \cap A \neq \emptyset\}$$

Es decir, si A y C son dos conjuntos (o en nuestro caso imágenes), la dilatación de la imagen A por el elemento estructural C devuelve un conjunto formado por elementos x tales que, si C se centra en x, al menos uno de los elementos de C es igual a uno de los de A. O, dicho de otra manera y en relación a imágenes, la dilatación es la imagen

resultante de reemplazar cada píxel de la imagen original por una réplica del elemento estructural. Este elemento estructural puede tener la forma que se desee. Obviamente, según su forma, la dilatación variará y la imagen resultante “tenderá” hacia esa forma. Además, se pueden realizar las iteraciones que se deseen con el fin de dilatar las zonas tanto como se quiera.

Como se ha mencionado, el proceso de dilatación ampliará el tamaño de las zonas obtenidas mediante el proceso de umbralizado. Esto dará lugar a regiones más concretas con las que trabajar. Para el proceso de dilatación se ha usado un elemento estructural de 10 píxeles de ancho y 10 píxeles de alto. Además, se le ha dotado de una forma de elipse para obtener zonas más redondeadas, al contrario que si se hubiera usado una forma rectangular, que proporcionaría regiones con sus bordes en dientes de sierra y parecidas a cajas.



## FILTRADO DE REGIONES

En la fase posterior al umbralizado, la de filtrado, lo que se hace es eliminar las regiones que por algún motivo no son buenas candidatas a ojos, nariz o boca. Como se ha visto, en la fase previa se obtienen un conjunto de zonas candidatas en la imagen, aquellas zonas que el filtro Gabor había resaltado. Ahora es preciso acotar el número de estas zonas para acelerar la fase posterior de búsqueda en árbol y para evitar posibles falsos positivos. En el algoritmo original se proponían límites de la cantidad de píxeles de las regiones y otras medidas absolutas. Sin embargo, se ha visto la necesidad de usar algún tipo de medida relativa al tamaño del fotograma e incluso relativa al tamaño de las regiones encontradas.

Así pues, el primero de los filtros de regiones que se aplica, excluye todas las regiones candidatas cuya largura exceda de la tercera parte del ancho del fotograma. En la imagen mostrada, este filtro elimina las dos grandes regiones del fondo que en este caso han aparecido porque se ha aplicado un filtro Gabor con un parámetro  $\psi$  distinto a la imagen de la sección de dilatación, por ejemplo. Este primer paso eliminaría largas zonas horizontales



detectadas como por ejemplo estanterías o armarios en el fondo. A continuación se realiza el cálculo del tamaño medio del área de las regiones detectadas en la imagen. De este valor se obtienen otros dos, al multiplicarlo y dividirlo por 3, que van a definir los límites inferior y superior del área de las regiones candidatas. Así, todas las regiones cuyo área sea mayor que el triple de la media, o menor que un medio de la media, serán descartadas. Por último, se eliminan todas aquellas regiones cuya relación de aspecto entre el alto y el ancho sea menor que 1.2. Es decir, se descartan todas las regiones cuyo alto sea mayor que 1.2 veces su ancho, entendiéndose por alto y ancho el del recuadro que delimita cada una de las regiones. Lo que nos interesa es obtener regiones candidatas con orientación horizontal. Este último filtro elimina por tanto todas aquellas zonas verticales. En la aplicación, si se visualiza la ventana “Regiones candidatas”, se muestran las zonas candidatas en rojo y las restantes en negro.

Una vez que se han eliminado las zonas que, bien por largura, área u orientación, no eran propicias para el buen funcionamiento del proceso, se prosigue con un menor número de estas a la siguiente fase, cuyo objetivo será decidir finalmente cuales de las candidatas obtenidas de esta fase serán ojos, nariz y boca.

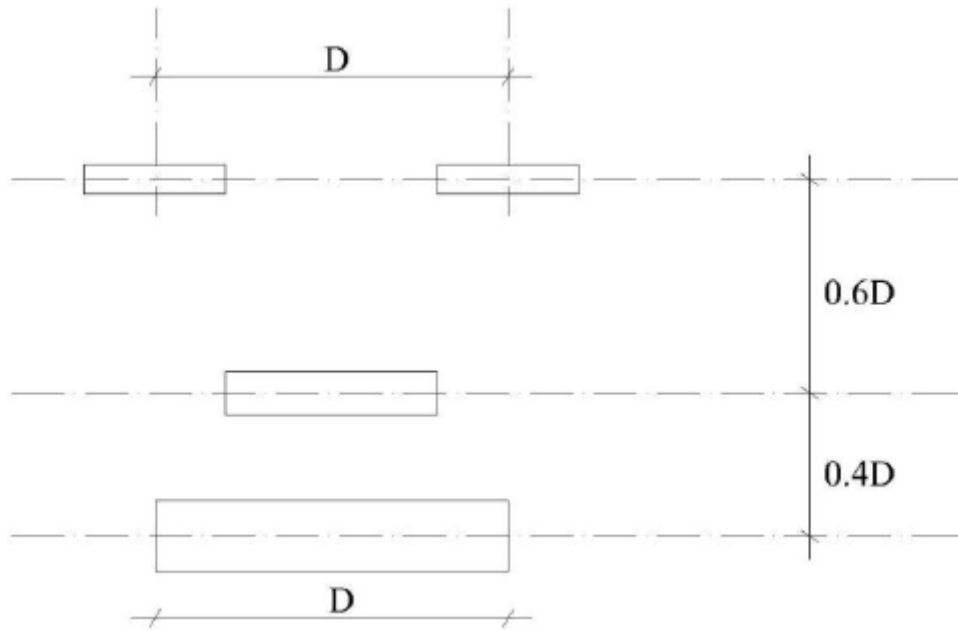
## LOCALIZACIÓN DE ZONAS FINALES

La fase de localización de zonas finales consiste en obtener cuatro zonas correspondientes a dos ojos, nariz y boca, a partir del conjunto de zonas que se tienen tras la fase de filtrado. Para ello, se tiene un vector de siete componentes que experimentalmente define las medidas de los rasgos y sus distancias de una cara estándar o patrón. Se muestra a continuación:

Vector patrón [d1, d2, d3, d4, d5, d6, d7] donde:

- d1 = distancia de la boca a la nariz / D
- d2 = distancia de la nariz al eje de los ojos / D
- d3 = distancia entre los centros de los ojos / D
- d4 = distancia vertical entre los centros de los ojos / D
- d5 = anchura de la nariz
- d6 = anchura del ojo izquierdo / D
- d7 = anchura del ojo derecho / D

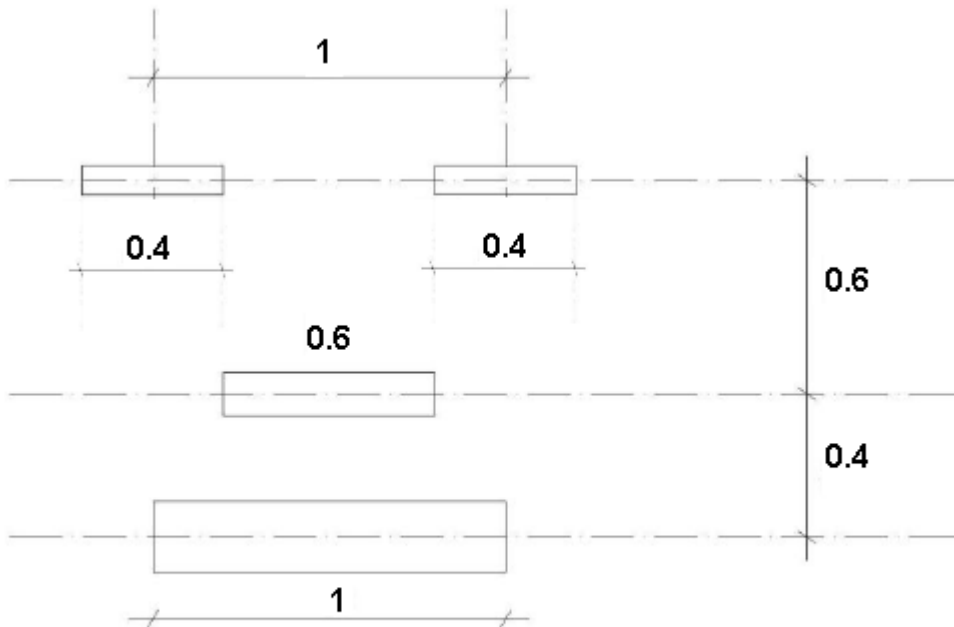
Donde D es la medida de la distancia entre los centros de los ojos, o la anchura de la boca como se aprecia en el boceto siguiente:



Si tomamos  $D = 1$ , las medidas definidas en el vector patrón resultan ser las siguientes:

$$\text{Vector patrón} = [0.4, 0.6, 1, 0, 0.6, 0.4, 0.4]$$

Que corresponden a las medidas mostradas en el siguiente esquema:

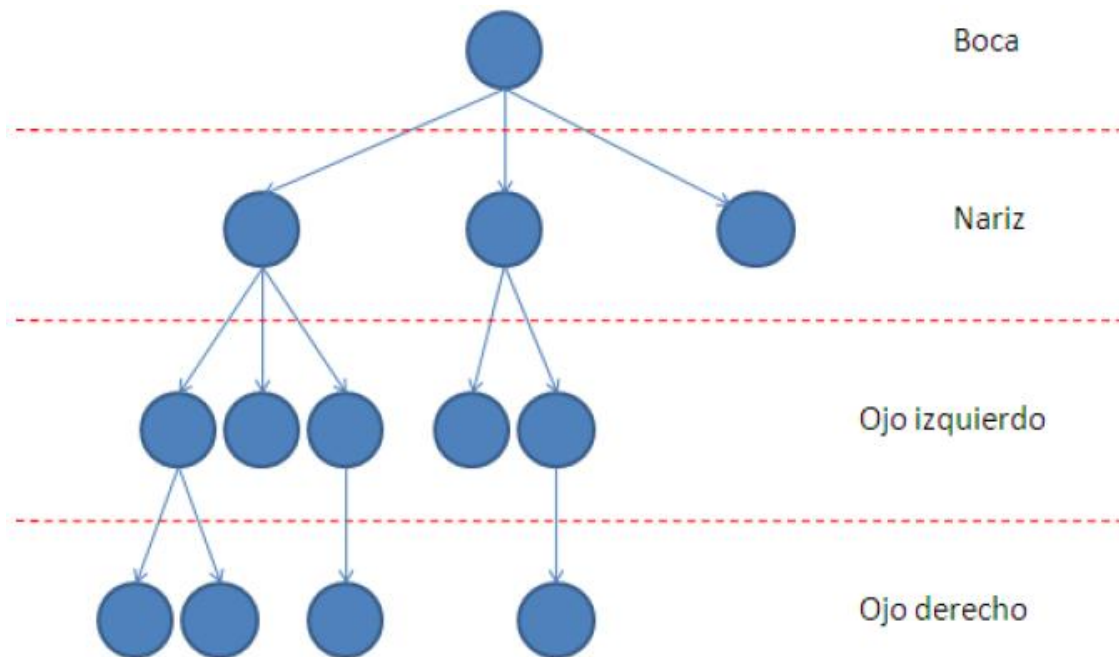


Las medidas y distancias del vector patrón se tomarán como las medidas que una cara idealmente debería tener. Pudiera haber caras cuyos rasgos se diferenciara bastante de los mencionados pero serían casos muy poco comunes. Como se ha dicho,

en esta fase se tiene un conjunto de zonas candidatas a zonas faciales y se desea obtener, teniendo como referencia el vector explicado, las que supuestamente pertenecen a los ojos, nariz y boca, por similitud en sus posiciones y medidas con los esquemas arriba propuestos. Para ello, se aplica un algoritmo de recorrido que va procesando las distintas configuraciones posibles de cuatro elementos y se van comparando todas ellas con el vector patrón. El fin de la fase lo determina la elección de la configuración de zonas cuyas medidas sean más próximas a las presentadas en el vector patrón. Para conocer la “proximidad” de las medidas de la muestra de zonas tomada con las medidas del vector patrón, se usa la fórmula siguiente:

$$DISTANCIA = \frac{1}{7} \sum_{i=1}^7 |d_i - vectorPatrón_i|$$

Así pues, la distancia entre una posible configuración de regiones y el vector patrón es la media de las distancias de cada una de las siete medidas respecto a las medidas correspondientes del vector patrón. Ahora bien, sabemos que para cada conjunto de zonas hay que obtener una distancia, pero, ¿cómo recorrer este conjunto? Para llevar a cabo esta tarea se ha usado un algoritmo de recorrido que construye un grafo en árbol como el que puede verse en la siguiente imagen.

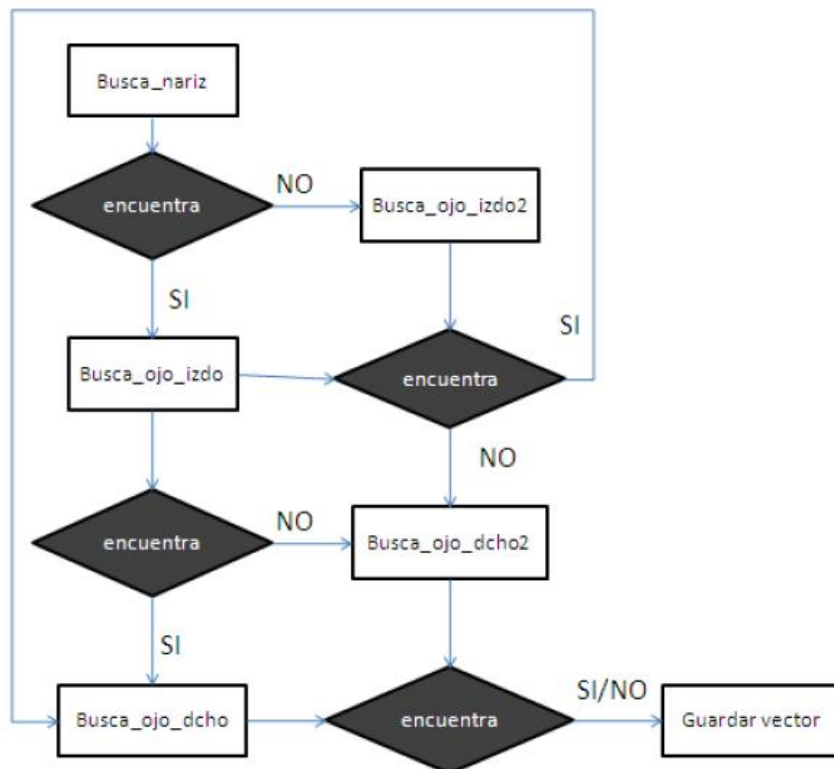


El algoritmo funciona de la siguiente manera. Se toma una zona inicial del conjunto de zonas que se tiene, que pasaría a ser la raíz del grafo. Este nodo se va a considerar como la boca y se va a tratar de encontrar nodos tales que su distribución en el espacio respecto a este nodo raíz siga, dentro de unos límites, las proporciones de distancia que podrían representar una cara. Así, lo siguiente sería buscar la nariz, como un nodo que representa una zona más elevada que la boca y más o menos alineada verticalmente con esta. Para ello, se recorren todas las zonas candidatas a nariz y para las válidas se computa su distancia respecto de la boca y se almacena en el vector. Una vez se tiene una zona candidata a nariz, se procedería a buscar de manera similar el ojo



izquierdo, recorriendo todas las zonas restantes y almacenando aquellas que pudieran tratarse por su tamaño y distancia a la nariz, de un ojo izquierdo. El algoritmo de recorrido termina con la búsqueda del ojo derecho de la cara, como una zona a una distancia similar a  $D$  del ojo izquierdo y con un tamaño de unas 0.4 veces esta distancia  $D$ .

Si el algoritmo no detecta ninguna zona como candidata al nodo de la fase en la que se está (por ej. se tiene la boca pero ninguna zona cumple las medidas de una posible nariz), se marca la distancia de ese elemento en el vector con la misma medida que tiene el vector patrón y se sigue la búsqueda hacia el elemento siguiente, aunque los vectores así construidos optarán con menos posibilidades a ser vector resultante del algoritmo. De esta manera se evita la interrupción de la búsqueda y la eliminación de posibles zonas candidatas por el hecho de no haberse detectado correctamente una de las cuatro en etapas anteriores. El diagrama de flujo del algoritmo de recorrido y búsqueda de zonas faciales de entre todas las candidatas es el siguiente:

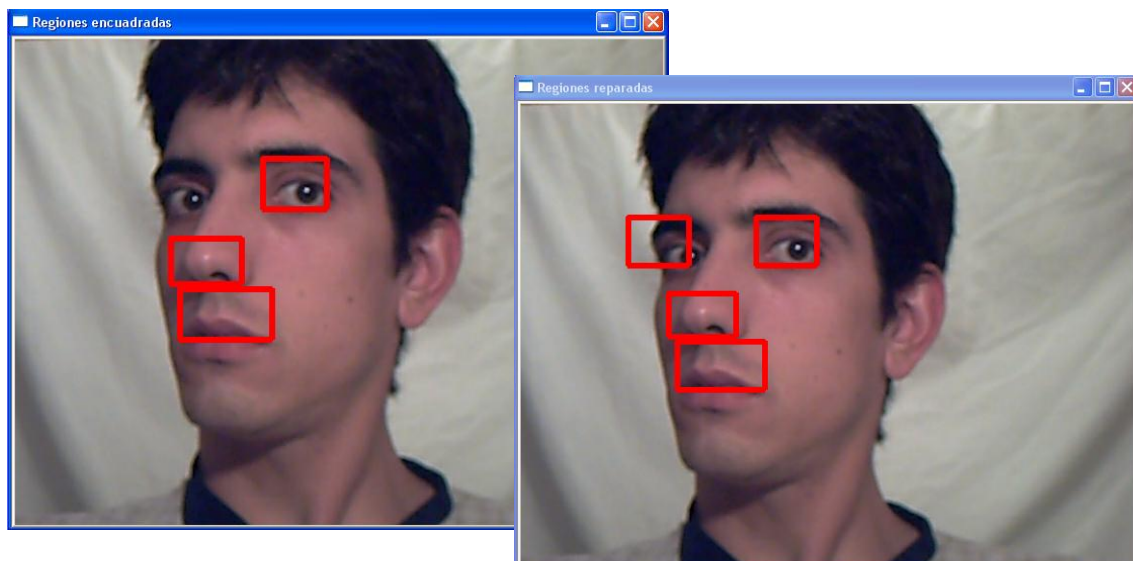


Así pues, se van tomando de forma secuencial las distintas zonas como boca. Una vez se tiene una boca elegida, se pasa a buscar la nariz. Si se encuentra una posible nariz, se pasa a buscar el ojo izquierdo a partir de esta. Si no se encuentra nariz, se busca igualmente un ojo izquierdo pero basándose en las distancias con respecto a la boca. Si se encuentra ojo izquierdo se procede a buscar el ojo derecho, al igual que si no se encuentra ojo izquierdo, solo que en este último caso las distancias a comprobar variarán. Tanto si se encuentra ojo derecho como si no, se guardan las medidas en un vector que se comparará posteriormente con el vector patrón.

Una vez que se tienen los distintos vectores con las medidas de todas las configuraciones candidatas se busca aquel cuya distancia con el vector patrón es menor.

Si todas las posibles configuraciones encontradas contienen las zonas de ojos, nariz y boca, se elegirá como definitiva la que tenga una distancia menor con el vector patrón. Si existen configuraciones con cuatro zonas encontradas y otras en las que no se han encontrado todas (por ej. falta la nariz, o el ojo derecho), se elegirá como bueno el vector con menor distancia de entre los que tengan las cuatro zonas encontradas, desechándose el resto. Si, por último, no se ha encontrado ninguna configuración de zonas completa (el algoritmo de recorrido nunca ha obtenido boca, nariz y dos ojos por faltar en todos los casos algún elemento), se tomará como vector definitivo el que contenga más zonas sin tener en cuenta sus medidas con el vector patrón, o, si existen varios con mayor número de zonas, aquel cuya distancia con el mencionado patrón sea menor. Es decir, siempre se penaliza al vector que no ha encontrado todas las zonas y siempre prevalece un vector con más zonas que uno con menos, aunque las medidas del último sean más cercanas al vector patrón. Esto último puede ocurrir ya que si una zona no se encuentra, en el vector correspondiente las distancias se toman iguales a las del patrón para ese elemento.

Hay ocasiones en las que no es posible encontrar ninguna configuración completa de una cara porque por ejemplo no se ha detectado el ojo izquierdo, como se ve en la imagen de debajo. En estos casos se ejecuta, posteriormente al algoritmo de recorrido en árbol, un algoritmo de reparación de regiones, en el cual se “intuye” la posición de los elementos que faltan y se añade a la imagen el recuadro de la posible posición de esta o estas zonas. El algoritmo de reparación tiene en cuenta las medidas y distancias de los elementos de la cara a reparar y en base a estos y según los datos de la cara patrón se obtienen las regiones aproximadas. Debajo vemos un ejemplo.



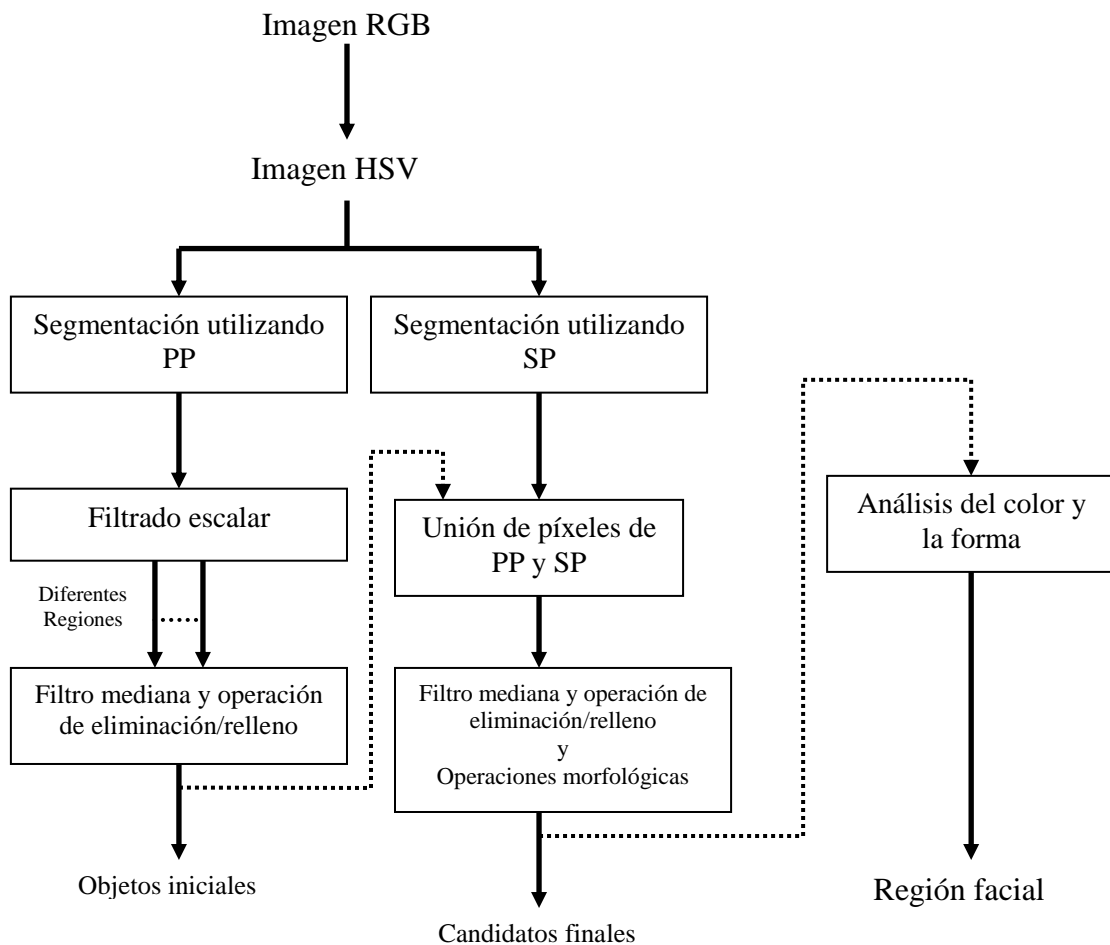
En la imagen de la izquierda se puede ver un rostro que, lejos de mirar de frente a la cámara, lo hace en un ángulo de unos 45°. Esto provoca que la nariz se haya desplazado de la vertical hacia la izquierda. Sin embargo, el algoritmo de búsqueda ha detectado la posición de la nariz como dentro de los límites de la cara patrón. El ojo derecho (que está recuadrado en rojo), también lo ha detectado correctamente en base a la nariz y boca. Sin embargo, el ojo izquierdo, por encontrarse en una zona en sombra y muy cerca del borde de la cara, no ha sido detectado. Como no se había encontrado una configuración de regiones mejor, el vector escogido fue el mostrado. Al ser una detección de tan solo tres elementos, el algoritmo de reparación se ejecuta para, a partir

de la posición y medidas de estos y las medidas del vector patrón, recuadrar una zona en la que previsiblemente se hallará el ojo izquierdo. En la imagen de la derecha se puede ver cómo el algoritmo ha tomado para el ojo izquierdo las mismas medidas que tiene el derecho y para su colocación ha tenido en cuenta que se halle a la misma altura que este y a la misma distancia que este tiene con la boca.

## EL MÓDULO DE FILTRADO DE PIEL

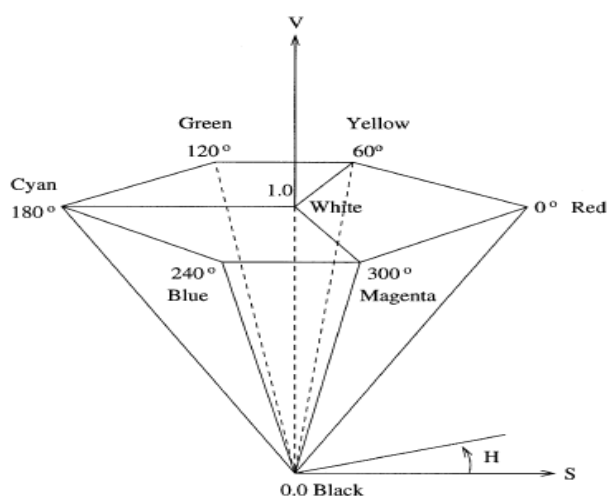
El núcleo de este proyecto, como se ha visto hasta ahora, es la detección de las regiones de ojos, nariz y boca en un rostro grabado por una cámara. No obstante, este proyecto tiene también otro objetivo, y es el de hacer funcionar conjuntamente el módulo de detección de ojos, nariz y boca con un módulo realizado por un compañero capaz de identificar las regiones de piel en una imagen. Para cumplir esta parte, hubo que trabajar conjuntamente con el autor del módulo, cuyo proyecto queda reflejado y se puede consultar en [3].

El esquema del funcionamiento de esta parte del proyecto, realizada por Ioritz Cia Ulacia y no por el autor de este PFC, consiste en lo siguiente [4]:



El método empleado en este módulo está basado en el color de la piel para hacer la segmentación. Generalmente las imágenes están representadas en el espacio de color RGB. Esto quiere decir que el color que representa cada píxel de la imagen depende del valor de la matriz R para la posición de ese píxel, el valor en la matriz G y el valor en la matriz B. Este espacio de color indica la cantidad de rojo, de azul y de verde que contiene cada píxel. Existe más modelos de representación del color como el HSI, el CIE  $L^*u^*v^*$  y el Karhunen-Loeve. Sin embargo el que se emplea en este módulo es el HSV, que es muy parecido al HSI y en el que los valores que afectan a cada píxel son la tonalidad (H), la saturación (S) y el brillo relativo (V). Como se ha dicho anteriormente la segmentación estará basada en el color, así que este modelo HSV viene muy bien para poder realizar esa segmentación, por lo que el primer paso que hay que hacer es la conversión de RGB a este espacio.

La información del color de HSV se representa de forma muy amplia utilizando el conjunto de coordenadas de RGB. Por otro lado, el modelo de color de HSV (tonalidad, saturación, valor) se corresponde más a la percepción humana del color. La representación gráfica de este espacio de color consiste en un prisma hexagonal.



La tonalidad (H), que toma valores entre 0 y 360, comienza en rojo (0). Esto nos da la medida de la composición espectral de un color. La saturación (S) es un cociente que se extiende entre 0 y 1. Este componente muestra la proporción de luz pura de la longitud de onda dominante e indica como de lejos está de un gris del mismo brillo. El valor (V) también se extiende entre 0 y 1 y es una medida del brillo relativo. El origen,  $V=0$ , corresponde a negro. En este valor particular, H y S son indefinidos y sin sentido. Cuando vamos hacia el valor 1 percibimos diferentes cortinas de gris hasta que se alcanza el punto final (donde  $V=1$  y  $S=0$ ) que se considera blanco.

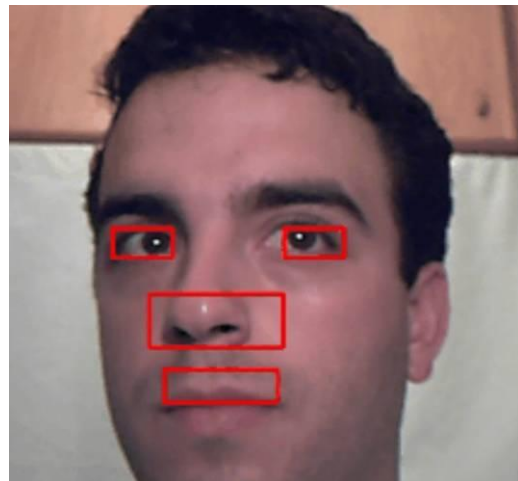
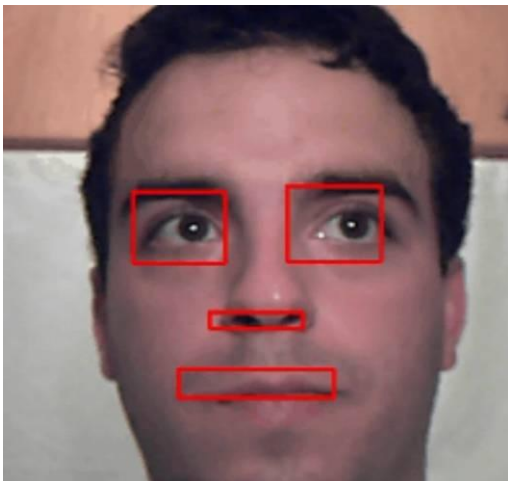
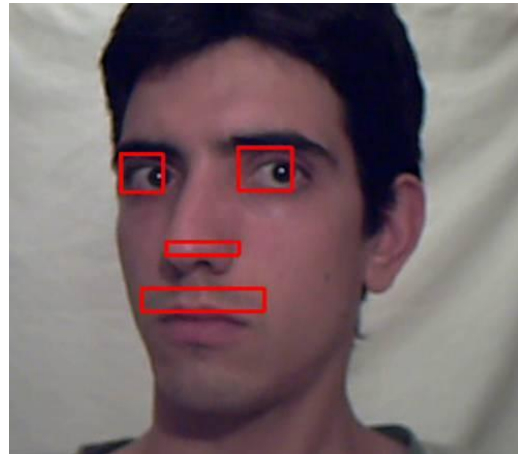
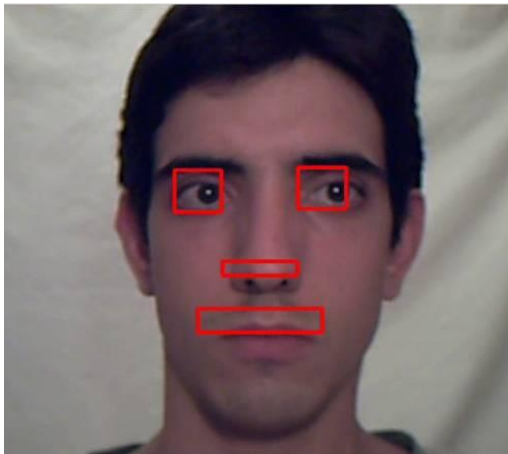
Después de haber segmentado utilizando estos valores lo que se hace es decidir si el trozo segmentado corresponde a una región facial o no. Para ello habría que tener en cuenta diferentes factores dependiendo de la situación.

Como se ha visto en el esquema general del proyecto de la sección “Captura de fotogramas”, opcionalmente se puede hacer funcionar el software junto con este módulo. Se ha decidido que la ejecución de este módulo sea opcional (el usuario puede decidir con el panel de control si ejecutarlo o no) ya que ralentiza considerablemente el programa en su conjunto. En este sentido, podría considerarse una mejora en el proyecto encaminada a realizar detecciones de piel cada cierto tiempo para, en el resto del tiempo, intuir la trayectoria de la zona de piel mediante un algoritmo como el filtro de Kalman. A su vez, el módulo detector de pieles es complejo y no se ha estudiado en profundidad, lo que unido a que en el entorno de pruebas no funcionara correctamente, ha llevado a descartarlo para las pruebas que en este proyecto se muestran, tanto

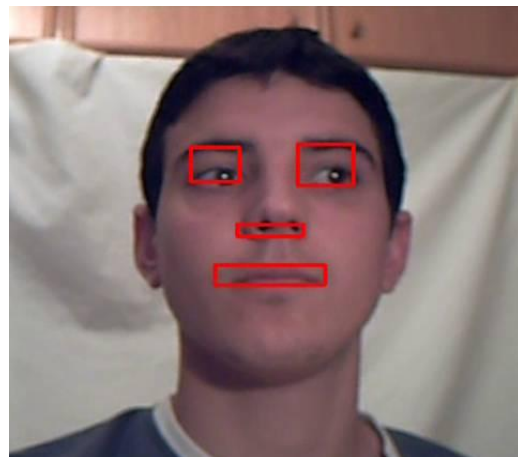
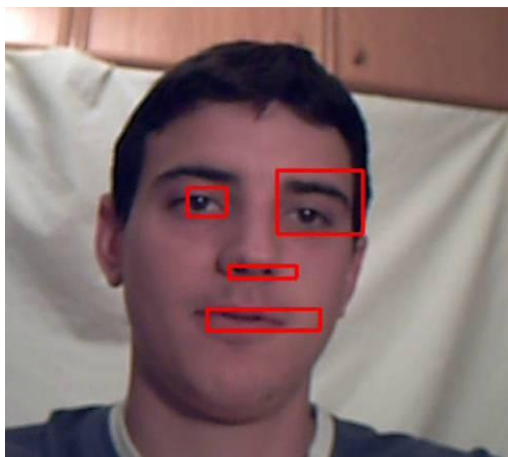
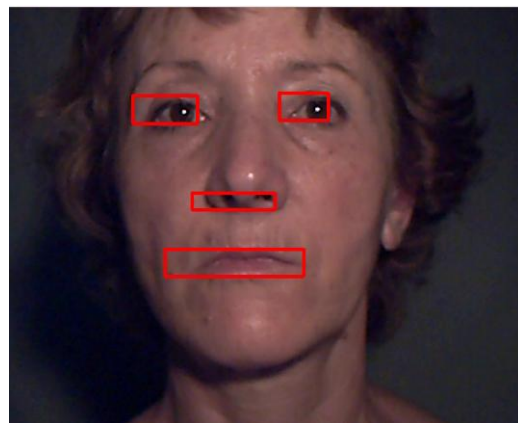
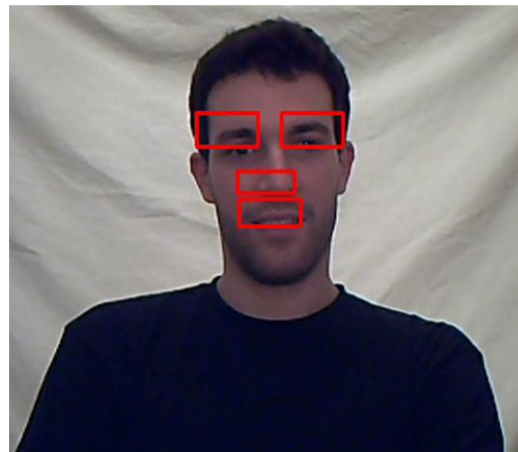
fotografías como el vídeo demostrativo que se proporciona. Si el software se ejecuta en un entorno propicio para la detección de pieles, los resultados serán aún mejores que los vistos en este proyecto (exceptuando la reducción de la velocidad), ya que la zona de búsqueda de regiones horizontales se reducirá considerablemente a la región facial del individuo con la mejora que conlleva en cuanto a objetos del fondo que empeoran actualmente los resultados.

## PRUEBAS REALIZADAS

En esta sección se van a mostrar los resultados visuales de pruebas realizadas a diferentes personas. Obviamente se trata de fotografías. Para visualizar diferentes pruebas en vídeos a tiempo real se adjunta un fichero de vídeo en el cual los resultados, buenos o malos, quedan patentes de forma más clara que con simples fotogramas. Este fichero se ha creado a partir de diferentes pruebas realizadas en un PC con un procesador Intel Quad Core Q9550, esto es, un procesador de cuatro núcleos con una frecuencia de reloj cada uno de 2.83 GHz. La placa base es una Asus P5Q-Pro con 3 GB de memoria RAM a 800 MHz. La cámara, como ya se ha mencionado al comienzo de esta memoria, es una Toshiba Camileo que graba vídeo con una resolución de 640 x 480 píxeles a 30 fotogramas por segundo.











## CONCLUSIONES

Como se ha podido ver, se ha implementado un sistema software capaz de realizar un seguimiento sobre regiones faciales como son los ojos, nariz y boca y hacerlo funcionar junto a otro sistema detector de piel. Uno de los mayores retos a la hora de empezar el desarrollo fue la premisa de que fuera “lo más a tiempo real posible”. Con esto quiere decirse que el vídeo que se muestre dé la menor sensación posible de que se trata de una secuencia de fotogramas mostrados a baja velocidad y que exista una interacción con el usuario. Pues bien, esta premisa se ha cumplido relativamente bien. Si aislamos el módulo detector de regiones de ojos, nariz y boca del detector de pieles, el software funciona con relativa rapidez, eso sí, en un PC potente aunque ninguno que hoy en día no se pueda encontrar con relativa facilidad con un coste medio. Y en buena parte es debido al uso de C y C++, lenguajes de programación muy eficientes y al uso asimismo de la librería OpenCV que contiene algoritmos optimizados para procesadores Intel gracias a las Primitivas de Rendimiento Integradas de Intel, un conjunto de rutinas de bajo nivel. La necesidad de usar C++ me ha aportado conocimientos sobre este lenguaje de programación que no había tenido la oportunidad de usar hasta ahora. También me ha dado la oportunidad de poner en práctica conocimientos de visión por computador que recientemente había estudiado en último curso de carrera, sin olvidar el aprendizaje de nuevos temas aplicables a este campo como el filtro de Gabor.

Los resultados prácticos obtenidos son en general buenos aunque para ello haya que calibrar el programa en cada caso para optimizar la detección. El método implementado, aunque ha arrojado buenos resultados, no es suficientemente bueno como cabría esperar, por ejemplo, de un sistema implantado en la banca o en aeropuertos, por ejemplo. Sí que podría ser muy útil como una de las partes en las que estuviera fundamentado un sistema fiable y competente comercialmente, o podría servir para corroborar las regiones detectadas por otro sistema más complejo. Algunos problemas surgen cuando el fondo contiene variados elementos o cuando el sujeto presenta objetos o rasgos que oculten parcialmente su cara como gafas, barba, arrugas acentuadas en la piel, flequillo... Asimismo, no sería fiable un sistema de aprendizaje que se entrenara para identificar personas en base a las medidas de sus rasgos y las distancias entre ellas, ya que suelen variar con facilidad. Sin embargo, es un sistema relativamente sencillo, rápido y con buenos resultados que podría usarse en aplicaciones no críticas como detección de regiones faciales en cámaras de fotos o web para el mundo del ocio, etc.

Este proyecto me ha brindado la posibilidad de poner en práctica todo un conjunto de conocimientos adquiridos a lo largo de la carrera, que, junto a mi anterior PFC también dedicado al campo de la inteligencia artificial (aprendizaje de rostros humanos mediante redes neuronales artificiales) han hecho que mi paso por este interesante mundo aún prácticamente en sus inicios sea muy satisfactorio.

## LÍNEAS FUTURAS

Por supuesto, este proyecto puede ser mejorado. En esta sección se exponen algunas líneas futuras que podrían ser de interés a la hora de profundizar en la mejora del programa.

Una de las posibles mejoras a realizar en un futuro sería la de permitir al software el tratamiento de caras que estén rotadas en el plano vertical. Como se sabe, el filtro de Gabor incorporado se encarga de resaltar zonas con orientación horizontal. Sin embargo, esta configuración del filtro puede presentar problemas en cuanto la cara sea rotada en el plano vertical. Puede que no sea un punto de gran interés porque raramente un individuo rotará su cara en este plano al mirar a una cámara. Sin embargo, el tener en cuenta este factor podría ser muy beneficioso en muchos casos. Para ello, se podría tratar de averiguar la dirección de la cara en la fase de filtrado de la piel. Teniendo la zona seleccionada como candidata a cara, se aplicaría un algoritmo que determinara la dirección de esta zona. Habiendo obtenido esta dirección, se podría usar para modificar la configuración del filtro Gabor (su parámetro de orientación) para que resalte zonas perpendiculares a la dirección vertical de la cara. No obstante, si el algoritmo de obtención de dirección fallara por no haber obtenido una zona lo suficientemente definida o con una orientación clara, las fases siguientes del programa no trabajarían correctamente y se obtendrían situaciones indeseadas.

Otra de las posibles mejoras podría ser el uso de la nueva versión 2 de OpenCV, que salió al público general cuando ya prácticamente estaba construido el sistema. Según los desarrolladores, la versión 2 aprovecha el rendimiento de los procesadores multinúcleo, lo que, en términos de eficiencia y rapidez, mejoraría previsiblemente el sistema.

Centrándonos ahora en la comodidad de uso y en su interfaz, cabría sugerir el desarrollo de una GUI más completa y atractiva para el usuario. La interfaz existente se ha programado mediante las funciones que OpenCV provee para ello. Sin embargo, esta librería tiene el tema de las GUI muy limitado. Apenas pueden crearse más elementos que ventanas, barras de desplazamiento y regiones con imágenes por lo que podría construirse una interfaz mediante, por ejemplo, Visual C++ que aportara sencillez y comodidad al software.

También podría mejorarse el programa de detección de ojos, nariz y boca, preparándolo para ser capaz de detectar a varias personas en la misma imagen o fotograma. Una posible idea sería la de buscar regiones de elementos mediante otro algoritmo de recorrido que tuviera en cuenta varios vectores de distancias posibles, en vez de uno solo. Estas regiones tendrían la característica de contener cuatro zonas distribuidas de forma similar al vector patrón más arriba expuesto.

Por último, podría intentarse fusionar este módulo de detección de regiones con la aplicación VFace, software de tratamiento de rostros creado y mejorado en el departamento por varios proyectistas.

## APÉNDICE

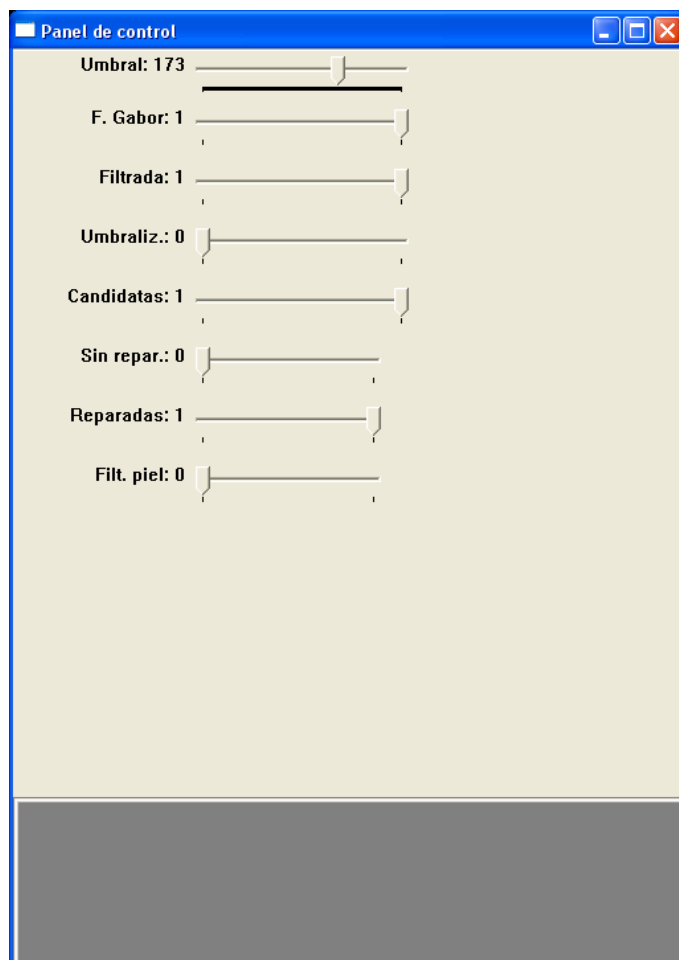
### MANUAL DEL USUARIO

En esta sección se proporcionará al lector o futuro usuario de la aplicación una visión del funcionamiento y estructura en cuanto a clases y funciones del proyecto.

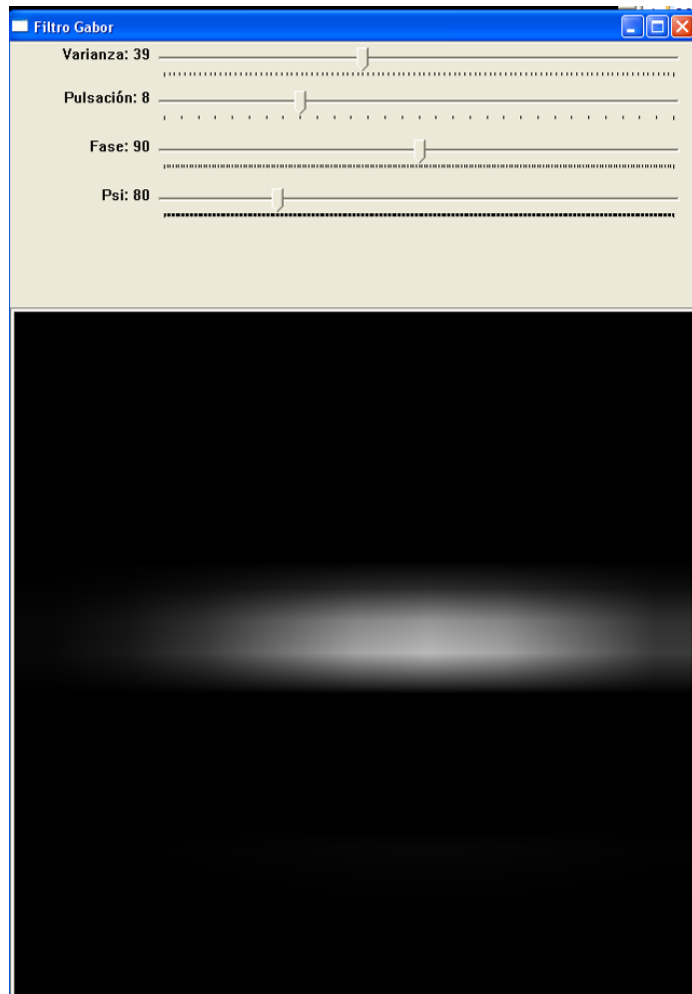
### INTERFAZ DE USUARIO

El interfaz del usuario no es complejo y se limita a dos ventanas que controlan diferentes parámetros y comportamientos de la aplicación, y otras ventanas que muestran los resultados de las distintas fases por los que pasa cada fotograma. Al arrancar la aplicación aparece una ventana llamada Panel de Control, que permite ajustar el umbral de la imagen y abrir y cerrar las diferentes ventanas de que consta el proyecto.

Como se puede ver en la imagen de la derecha, se tiene un primer control que permite ajustar el nivel de umbralizado de la imagen. Aunque por defecto la aplicación usa un valor, el usuario, dependiendo del entorno y la iluminación tiene la opción de cambiarlo y elegir aquel que mejores resultados proporcione. A continuación se ven siete controles de desplazamiento con valores 0 o 1. Corresponden a las siete ventanas que es posible visualizar para seguir el procesado de las imágenes. Obviamente, cuantas más ventanas se abran, más lento será el procesado ya que se requiere más trabajo de CPU. Si se colocan en 0 se cierran las ventanas y se abren seleccionando el 1. Posteriormente se verá la utilidad de cada una de ellas.

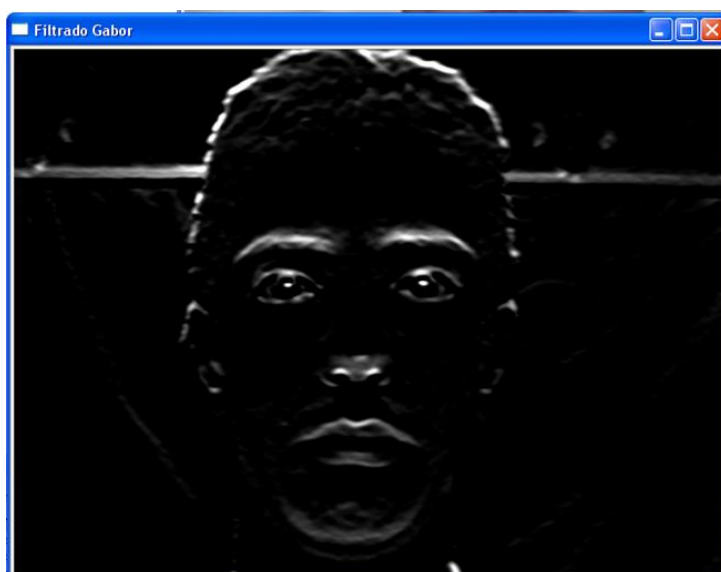


La segunda ventana que permite al usuario configurar el software es Filtro Gabor. En ella se muestran cuatro barras de desplazamiento que permiten al usuario controlar la forma del filtro Gabor que se aplicará a los fotogramas capturados por la cámara. Por orden de arriba a abajo se tienen la varianza, pulsación, fase y psi, los cuatro parámetros que ya se han visto cómo afectan a la forma del filtro Gabor en la sección de arriba “Filtrado Gabor”. Debajo de los controles de usuario se muestra el filtro Gabor según se haya configurado.

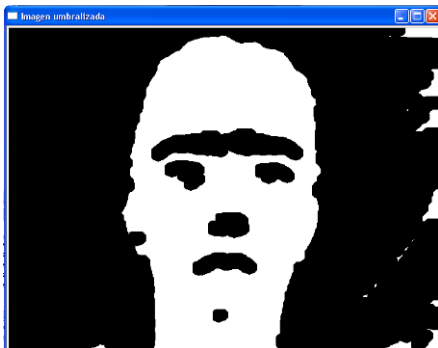


Como ya se ha mencionado, se dispone además de seis ventanas que permiten visualizar el proceso por el que se transforman los fotogramas hasta obtener las regiones recuadradas.

La primera permite visualizar el resultado del filtrado Gabor. Cambiando los parámetros del filtro mediante la pantalla de configuración de arriba, la imagen por consiguiente variará también. Debería ajustarse el filtro hasta que se obtengan las zonas a buscar muy blancas y el resto negras.

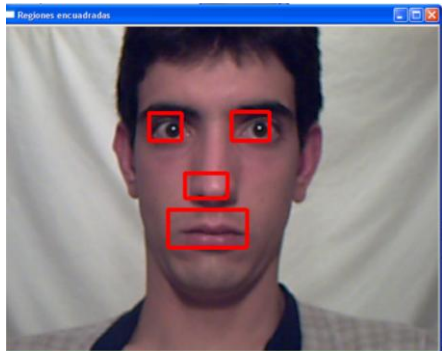
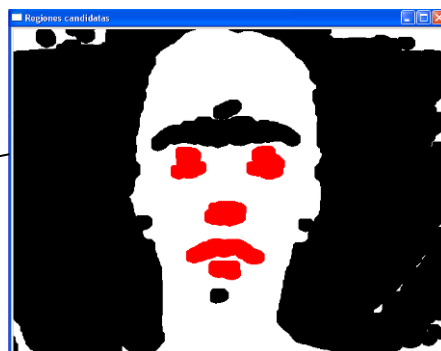


Las cuatro siguientes ventanas que se pueden abrir para visualizar el procesado son las que siguen:



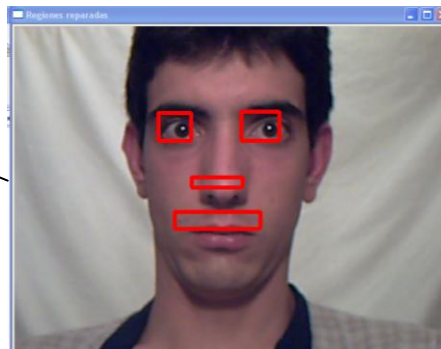
Ventana de umbralización: En ella se puede ver la imagen binarizada después de haber aplicado el límite de umbral definido en el panel de control.

Ventana de regiones candidatas: En ella se pueden ver las distintas regiones que se han tomado como posibles regiones candidatas a ojos, nariz y boca coloreadas de rojo. Las zonas negras se han desechado.



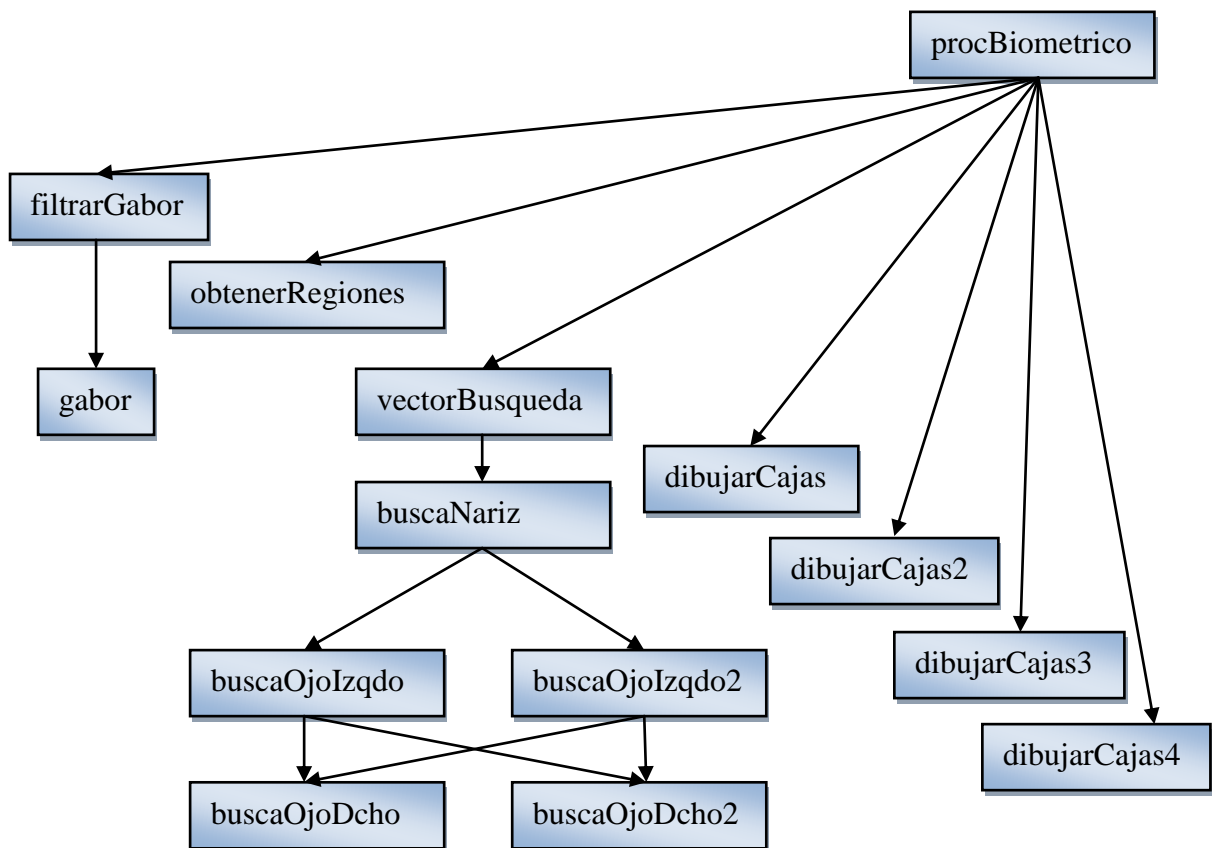
Ventana de regiones encuadradas: En ella se han encuadrado ya las regiones de ojos, nariz y boca después del algoritmo de recorrido en árbol. No tienen porqué haberse encuadrado todas las zonas.

Ventana de regiones reparadas: En ella se ha procesado el algoritmo de reparación de zonas y se dibujan los rectángulos que, definitivamente, marcan las cuatro zonas encontradas.



## FUNCIONES Y CLASES IMPLEMENTADAS

El programa arranca al ejecutar el método `main()` del fichero `Principal.cpp`. Este método no requiere de ningún parámetro de entrada. Se encarga de instanciar las clases `recuadraZonas` y `LocalizacionCaras` y de crear los hilos POSIX que llevarán a cabo la captura de imágenes y el procesado de estas mediante la ejecución de los métodos `Procesamiento()` y `VideoEnDirecto()` de la clase `recuadraZonas`. Aquí se excluyen las funciones pertenecientes al módulo de localización de piel y se invita al lector a leer [3] para más información sobre el tema. El módulo de detección de regiones faciales se inicia al llamar al método `procBiometrico()` de la clase `recuadraZonas`. A continuación se muestra una imagen que resume en un solo esquema el conjunto de funciones y llamadas a estas que se realizan durante la ejecución del módulo de detección de ojos, nariz y boca, y que han sido programadas para este proyecto:



*procBiometrico*: Se encarga de ejecutar el módulo de localización de las zonas de ojos, nariz y boca. Para ello, secuencialmente va llamando a las siguientes funciones.

*filtrarGabor(imagen, test)*: Obtiene a la entrada una imagen a la cual se desea aplicar un filtro Gabor y una variable para indicar si se desea mostrar en pantalla la ventana de configuración del filtro. Se encarga de llamar a la función `gabor` en el caso de que haya

que modificarlo por petición del usuario, o simplemente aplica a la imagen el filtro Gabor almacenado actualmente.

*gabor(tamaño, varianza, pulsación, fase, psi)*: Crea y almacena en memoria un filtro Gabor configurado a partir de los parámetros introducidos a la entrada. Para más información sobre estos, se remite al lector a la sección “Filtrado Gabor” de más arriba.

*obtenerRegiones(imagen)*: Dada una imagen binarizada, localiza las zonas de píxeles contiguos con la misma intensidad formando una lista de regiones candidatas a zonas faciales.

*vectorBusqueda(regiones, vector zonas encontradas)*: A partir de un conjunto de regiones obtenidas mediante la función *obtenerRegiones*, ejecuta el algoritmo de recorrido en árbol y obtiene un vector de cuatro posiciones. Cada una de ellas contiene el número de zona perteneciente a la boca, nariz, ojo izquierdo y ojo derecho.

*buscaNariz(vectores de zonas y otros parámetros no significativos)*: Busca regiones situadas por encima de la actual zona escogida como boca y que estén a una distancia razonable de esta. Si las encuentra llama a la función *buscaOjoIzqdo* y si no existe ninguna que concuerde, llama a *buscaOjoIzqdo2*.

*buscaOjoIzqdo(vectores de zonas y otros parámetros no significativos)*: Busca regiones por encima de la zona de la nariz y a una distancia horizontal y vertical razonables. Para cada región encontrada, lanza la función *buscaOjoDcho*. Si no se ha encontrado ninguna zona candidata a ojo izquierdo, se llama a la función *buscaOjoDcho2*.

*buscaOjoIzqdo2(vectores de zonas y otros parámetros no significativos)*: Busca regiones candidatas a ojo izquierdo pero de distinta forma a la función *buscaOjoIzqdo* ya que, mientras esta lo busca a partir de la nariz, aquella la busca a partir de la posición y medidas de la boca.

*buscaOjoDcho(vectores de zonas y otros parámetros no significativos)*: Busca zonas candidatas a ojo derecho a partir de la posición y medidas del ojo izquierdo. Devuelve el vector formado con las medidas y distancias de las regiones escogidas.

*buscaOjoDcho2(vectores de zonas y otros parámetros no significativos)*: Similar en su comportamiento a *buscaOjoIzqdo2*, localiza las posibles regiones candidatas a ojo derecho a partir de las medidas de la boca o nariz.

*dibujarCajas(imagen, zonas, índice, número de zona)*: A partir de una imagen, un conjunto de zonas, un índice y un número de zona, recuadra con un rectángulo rojo la zona del conjunto identificada mediante el índice.

*dibujarCajas2(imagen, distancia D, coordenadas, número de zona)*: Recuadra con un rectángulo rojo las zonas de nariz o boca a partir de las zonas de los ojos. Se usa en el algoritmo de reparación.

*dibujarCajas3(imagen, zonas, índice, boca, número de zona)*: Se encarga de recuadrar en la imagen de entrada la zona correspondiente al ojo izquierdo en el procesamiento del algoritmo de reparación.





*dibujarCajas4(imagen, zonas, índice, boca, número de zona)*: Recuadra en la imagen de entrada la zona correspondiente al ojo derecho. Usada en el procesamiento del algoritmo de reparación de regiones.

Aparte de las funciones programadas para este proyecto, se han usado algunas de las muchas funciones optimizadas que provee la librería OpenCV y que se listan a continuación:

*cvCaptureFromCAM*: Reserva memoria e inicializa la estructura en la que se guardarán los fotogramas grabados de la cámara.

*cvQueryFrame*: Graba un fotograma proveniente del dispositivo de vídeo, lo descomprime y lo devuelve.

*cvReleaseCapture*: Libera los recursos reservados para la estructura capturadora de vídeo.

*cvCreateMat*: Crea una estructura de tipo matriz.

*cvReleaseMat*: Libera los recursos utilizados por estructuras de tipo matriz.

*cvSet2D*: Asigna un valor a un elemento de una matriz de dos dimensiones.

*cvGetReal2D*: Obtiene el valor de un elemento determinado de una matriz 2D.

*cvGetRow*: Obtiene una fila determinada de una matriz.

*cvCountNonZero*: Cuenta los elementos no nulos de una matriz dada.

*cvCreateImage*: Crea estructuras para almacenar imágenes.

*cvReleaseImage*: Libera la cabecera y los datos de la imagen almacenada en la estructura *IplImage* pasada como parámetro.

*cvMerge*: Fusiona tres imágenes en una sola con tres canales de color.

*cvGetSize*: Obtiene el tamaño en ancho por alto de una imagen dada.

*cvResize*: Cambia el tamaño de una estructura de imagen.

*cvConvertScale*: Copia una matriz en otra con posibilidad de realizar una transformación lineal de sus elementos.

*cvScalar*: Crea tuplas de 1, 2, 3 o 4 elementos de tipo double.

*cvFilter2D*: Aplica un filtro lineal a una imagen dada.

*cvNormalize*: Normaliza un array de entrada a un rango de valores dado.



*cvCvtColor*: Convierte una imagen de un espacio de colores a otro.

*cvSmooth*: Aplica un filtro de suavizado a la imagen dada.

*cvSum*: Suma el valor de los elementos de una matriz.

*cvMul*: Multiplica uno a uno los elementos de dos matrices dadas.

*cvCreateStructuringElementEx*: Crea un elemento estructural del tamaño y forma deseado para el proceso de dilatación.

*cvDilate*: Realiza un proceso de dilatación en una imagen las iteraciones deseadas y con la estructura dada.

*cvThreshold*: Aplica un proceso de umbralizado a la imagen dada.

*cvCloneImage*: Clona una estructura imagen en otra.

*cvCopy*: Copia una estructura de imagen en otra variable pero sin duplicar sus datos.

*cvPoint*: Crea una estructura de tipo punto.

*cvNamedWindow*: Crea una ventana en la que poder alojar imágenes y controles de usuario.

*cvShowImage*: Muestra una imagen en una ventana de usuario.

*cvCreateTrackbar*: Crea una barra de desplazamiento en una ventana de usuario para modificar el estado de un parámetro.

*cvDestroyWindow*: Cierra una ventana de usuario abierta mediante *cvNamedWindow*.

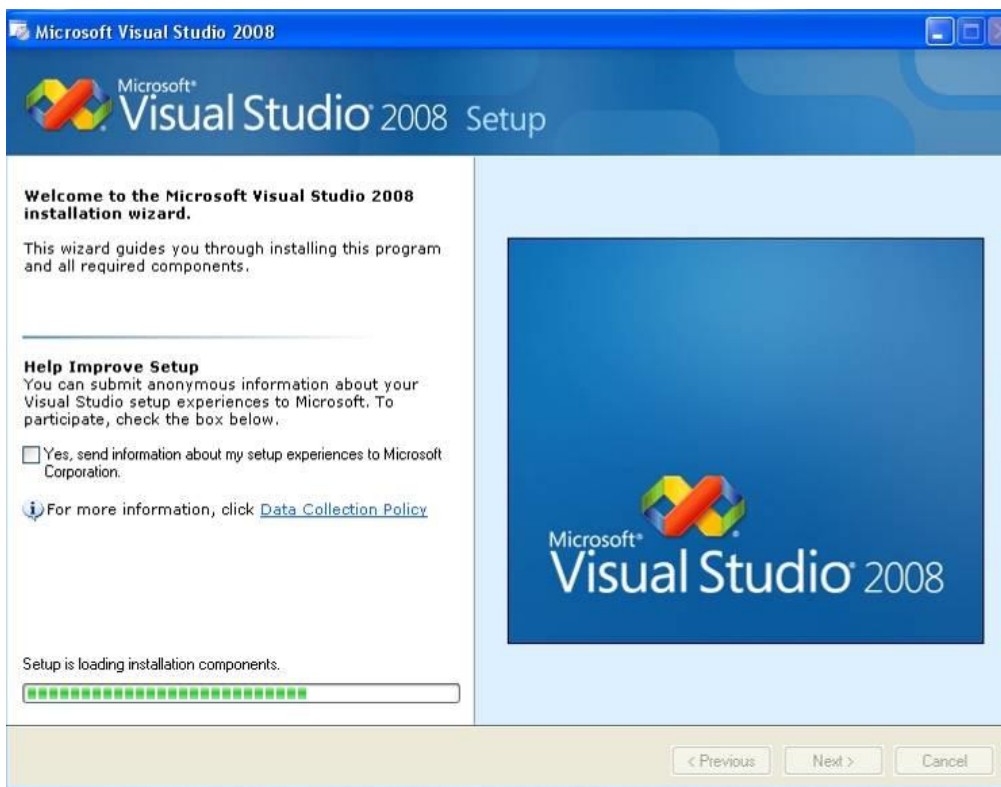
*cvWaitKey*: Espera a que el usuario presione una tecla o el tiempo indicado en milisegundos. Hace posible la interacción del usuario con las ventanas.

*cvRectangle*: Dibuja un polígono rectangular en una imagen con el color y grosor deseados.

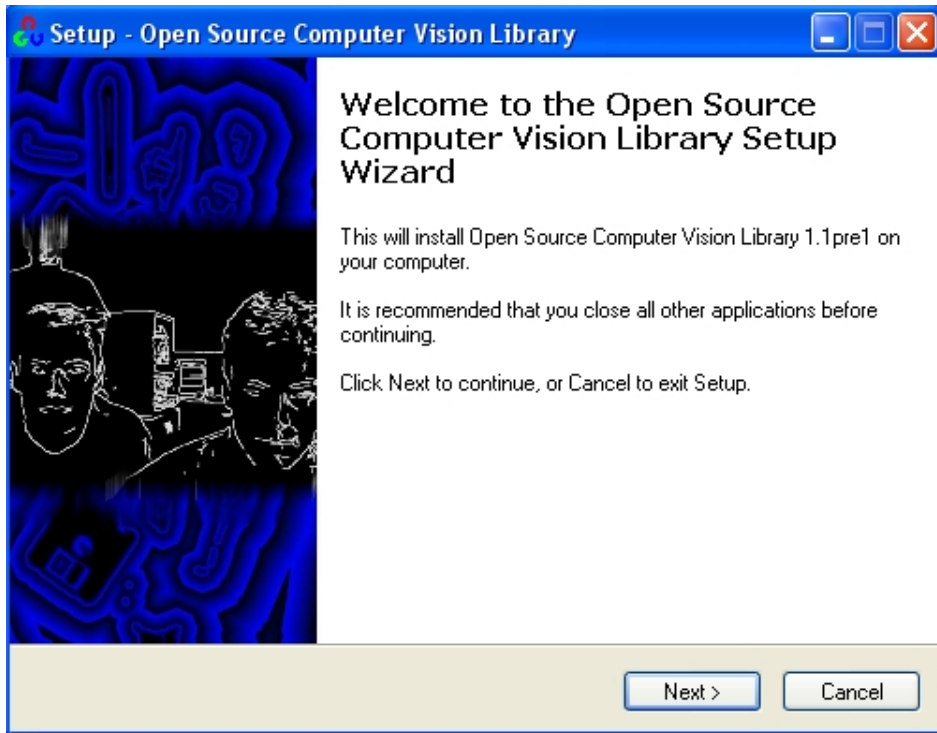
## PUESTA EN MARCHA DEL PROYECTO EN MICROSOFT VISUAL STUDIO 2008

No es tarea sencilla la puesta en marcha del proyecto en el entorno de desarrollo Microsoft Visual Studio 2008, software con el que se desarrolló, para su posible modificación futura. Hacen falta cargar las librerías de hilos pthread y por supuesto OpenCV y pueden surgir problemas de enlazado al no encontrar ficheros fuente o de biblioteca. Por ello, y por experiencia propia, se ha creado esta sección, que se espera ayude a su puesta en marcha a futuros proyectistas que pudieran proseguir con el proyecto mejorándolo o añadiendo funcionalidades adicionales. La puesta a punto del entorno de programación fue uno de los temas que más problemas dio y esta parte permite seguir paso a paso los puntos necesarios para ello.

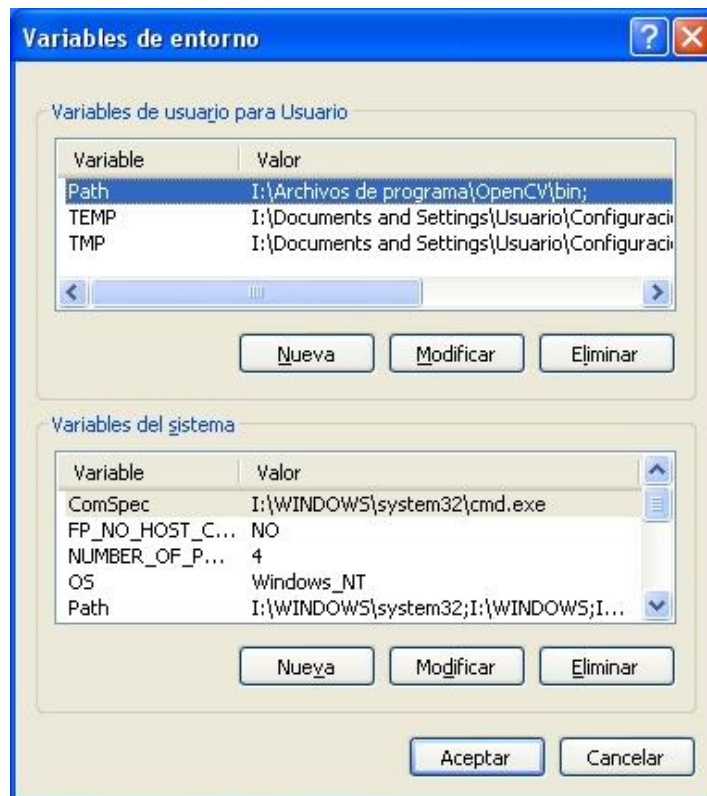
Para la realización de este proyecto se usó la versión 2008 del software Microsoft Visual Studio por lo que un primer paso es la instalación de este programa, que permitirá introducir cambios en el código fuente del proyecto o ver su funcionamiento interno. Se puede realizar una instalación estándar ya que no se requieren funcionalidades avanzadas que provee esta herramienta.



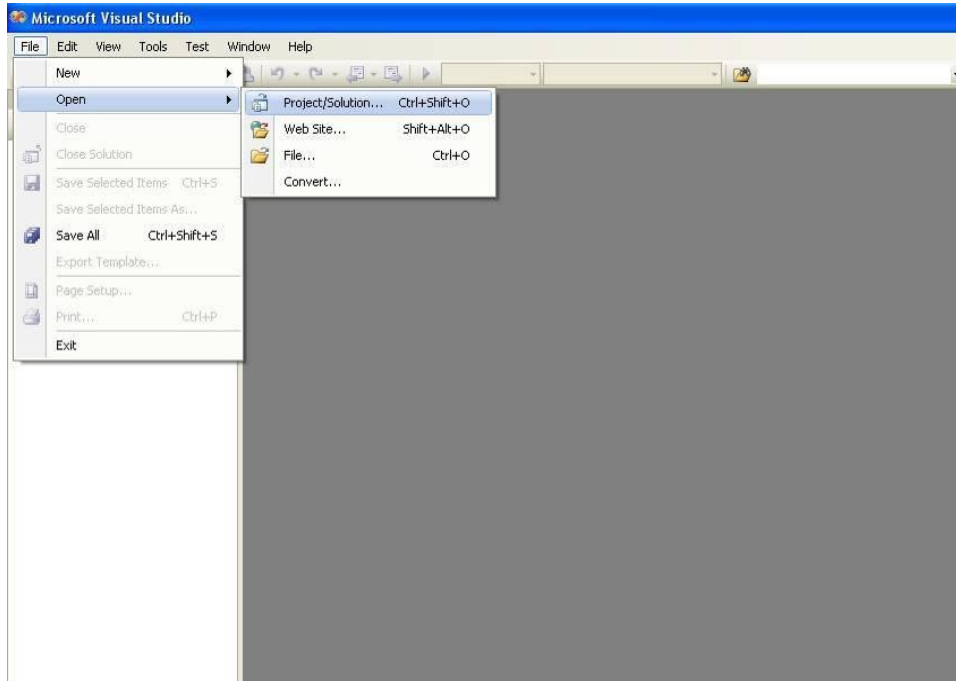
El segundo paso es la instalación de la librería de visión por computador OpenCV que como se sabe, se ha usado para dotar al proyecto de funcionalidades de visión artificial.



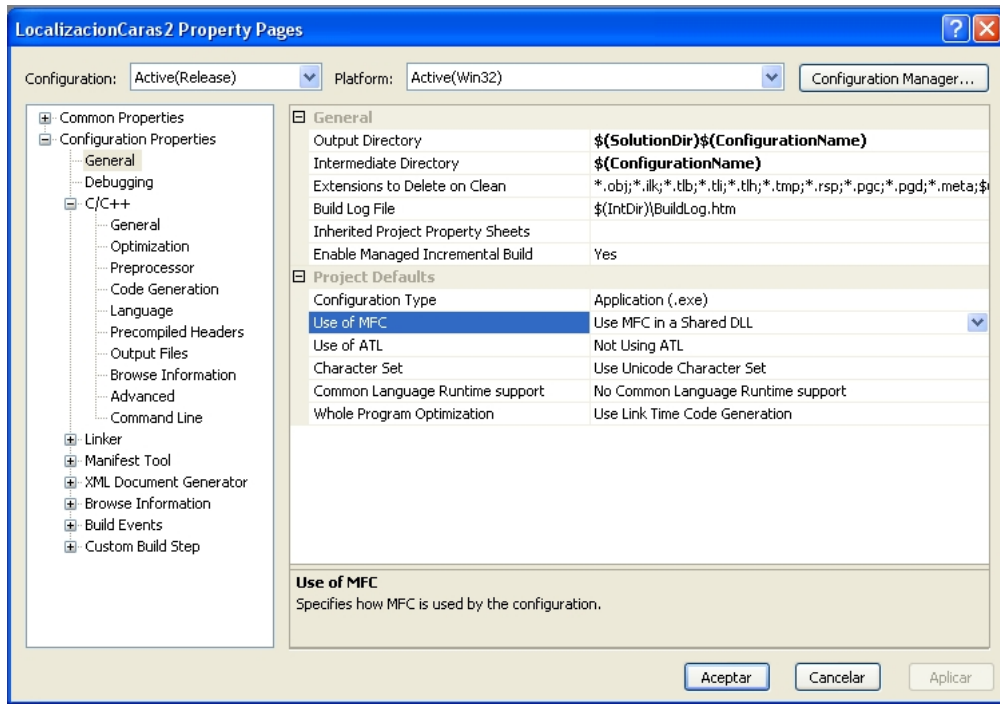
Una vez instalada, y aunque se añade si en la instalación no se desmarca la opción, se debería comprobar que existe una nueva variable de usuario en el sistema operativo que contiene la ruta de instalación de OpenCV y que servirá para indicar al proyecto dónde buscar los archivos de la librería de visión por computador.



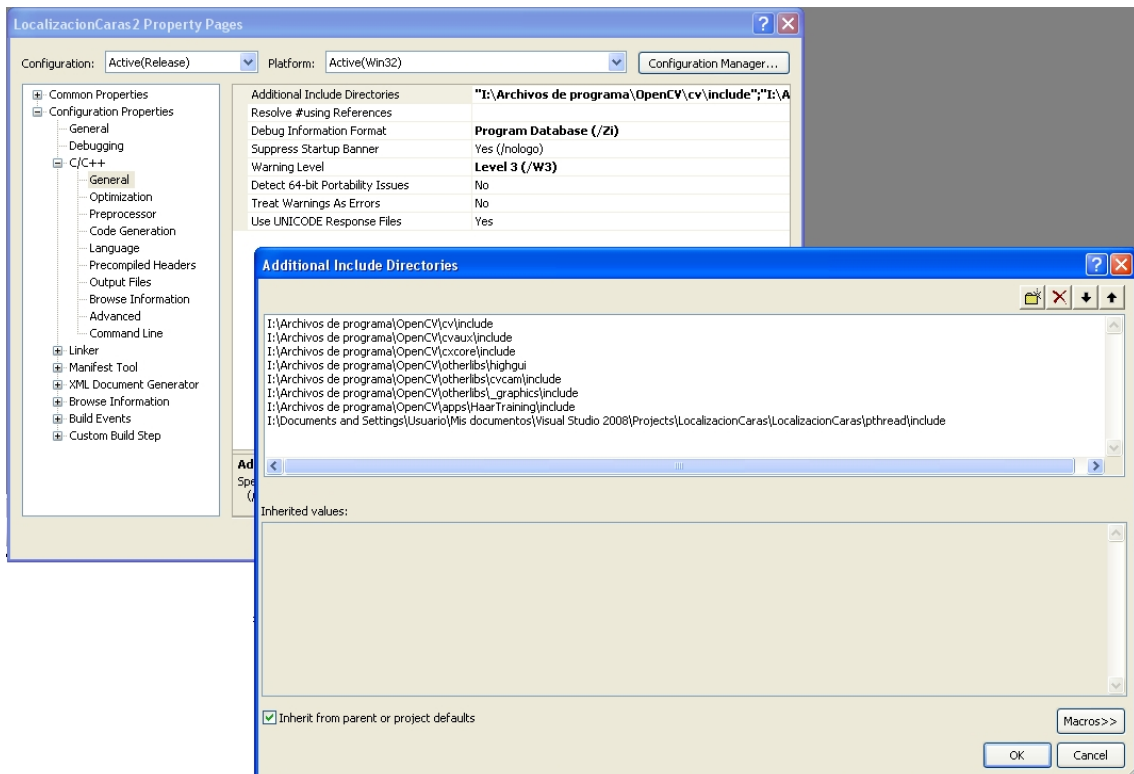
Ahora que ya está instalado tanto Microsoft Visual Studio 2008 como OpenCV se puede ejecutar este primero para abrir el proyecto de reconocimiento facial. Si se usa una versión anterior de Visual Studio es posible que se reconvierta el proyecto a la versión actual, proceso que no asegura que el programa funcione correctamente. Para abrir el proyecto se pulsará en *Proyecto / Solución* dentro del apartado Abrir en la sección *Archivo* del menú y se buscará el proyecto en la ruta que se tenga dentro del disco duro o CD.



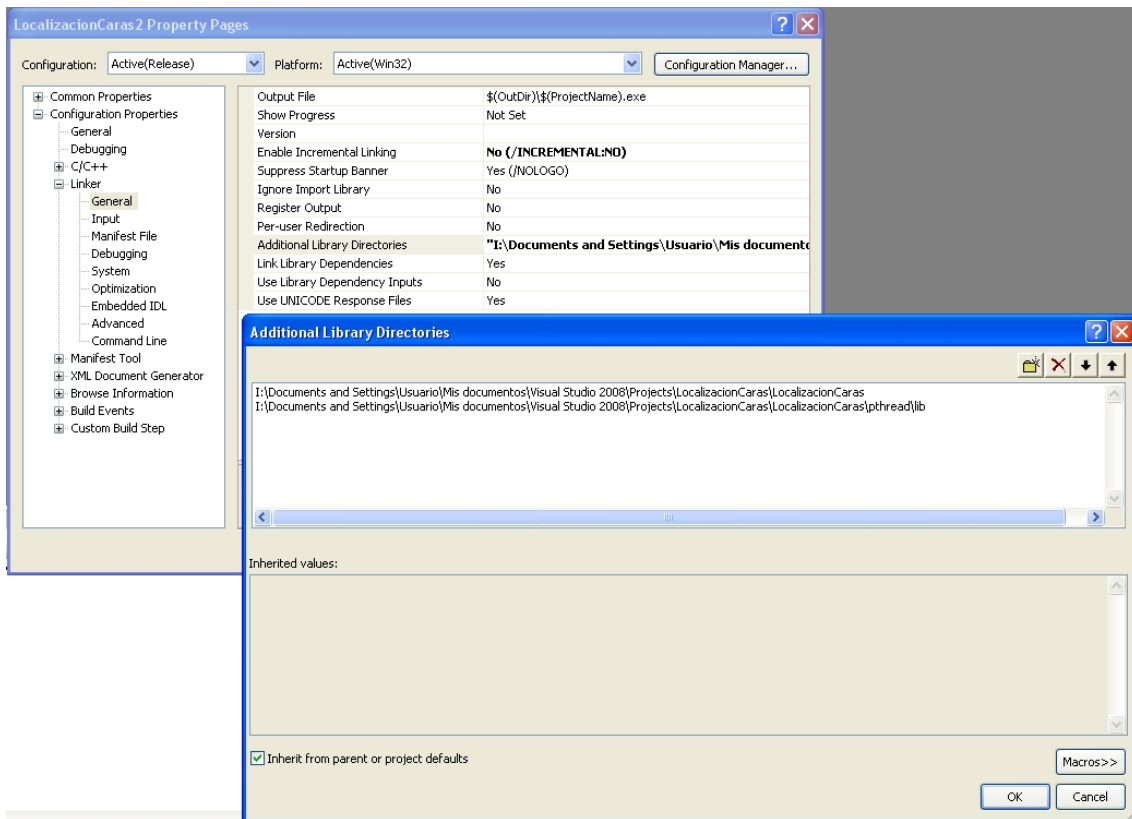
Para que la librería de hilos Pthread funcione correctamente es necesario cambiar la opción que esté seleccionada en el apartado *Use of MFC*, por *Use MFC in a Shared DLL*. Para visualizar esta ventana de configuración hay que pulsar el botón derecho del ratón en el nombre del proyecto en la pestaña *Solution Explorer* y pulsar en *Propiedades*.



El siguiente paso es comprobar que se tienen cargadas correctamente las localizaciones en las que se hallan los archivos de cabecera .h de OpenCV. Para ello hay que acceder a la ventana de propiedades del proyecto y pulsar en la sección General dentro de C/C++ del apartado Configuration Properties. En la línea Additional Include Directories deben estar correctamente incluidas las rutas a las carpetas mencionadas y que se pueden ver en la imagen de debajo.

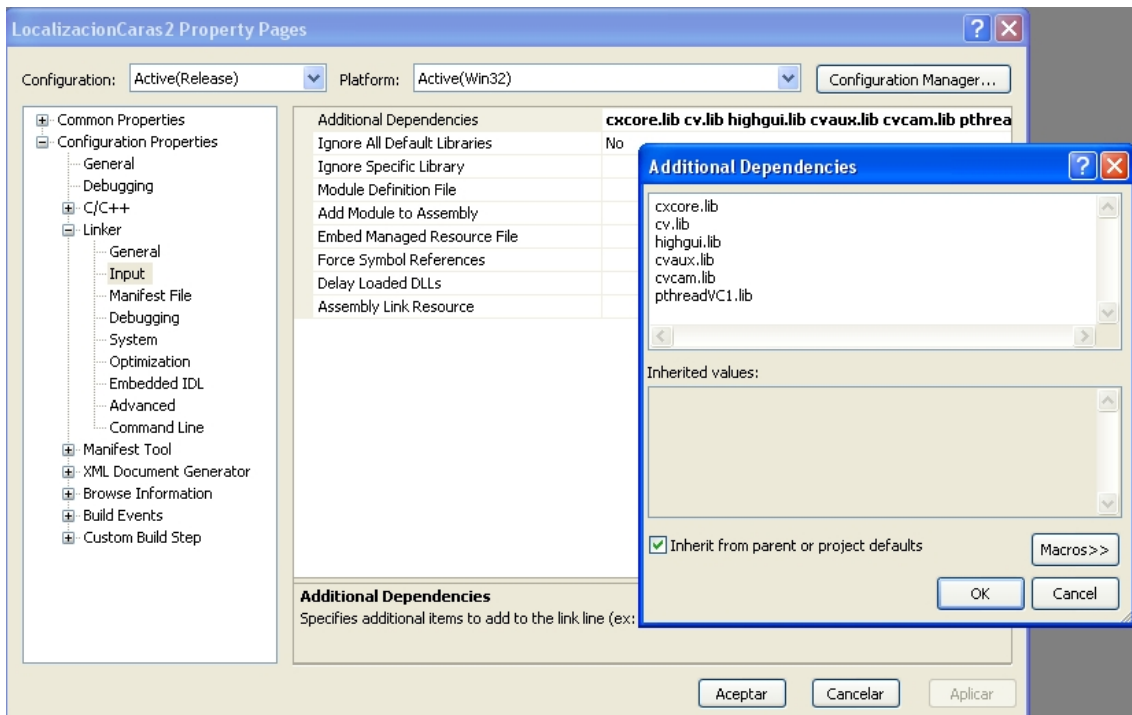


Para que el enlazador encuentre alguna librería extra como la de hilos, es necesario introducir la ruta de la carpeta en la que se localizan. Para ello, dentro de la sección *Linker* y en *General*, es necesario comprobar que la ruta existe en la línea *Additional Library Directories* y es correcta.

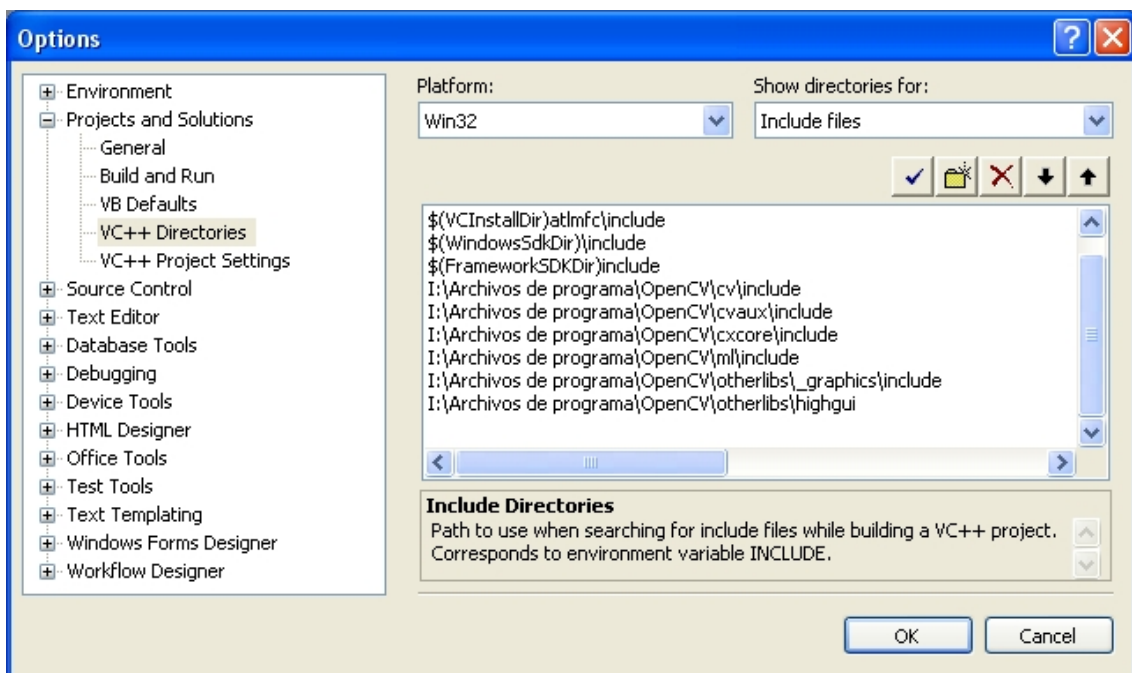


Asimismo, es necesario comprobar que se han añadido todas las librerías que el proyecto va a usar. Para ello, se debe revisar la línea *Additional Dependencies* de la sección *Input* del *Linker*. Las librerías requeridas se muestran en la imagen de debajo.

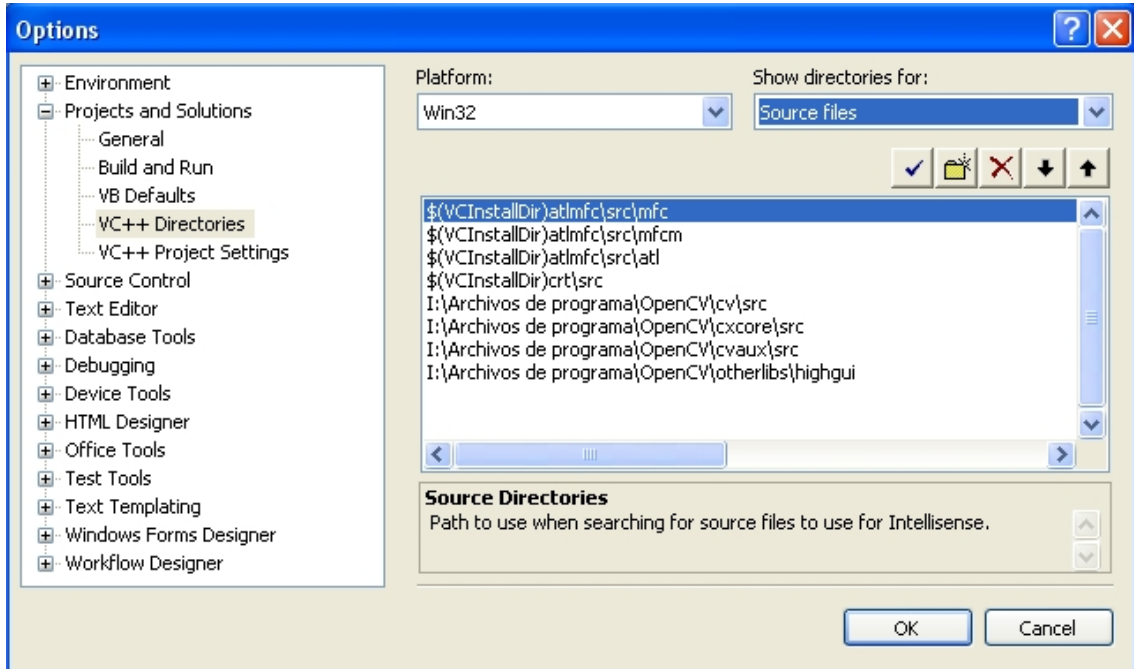




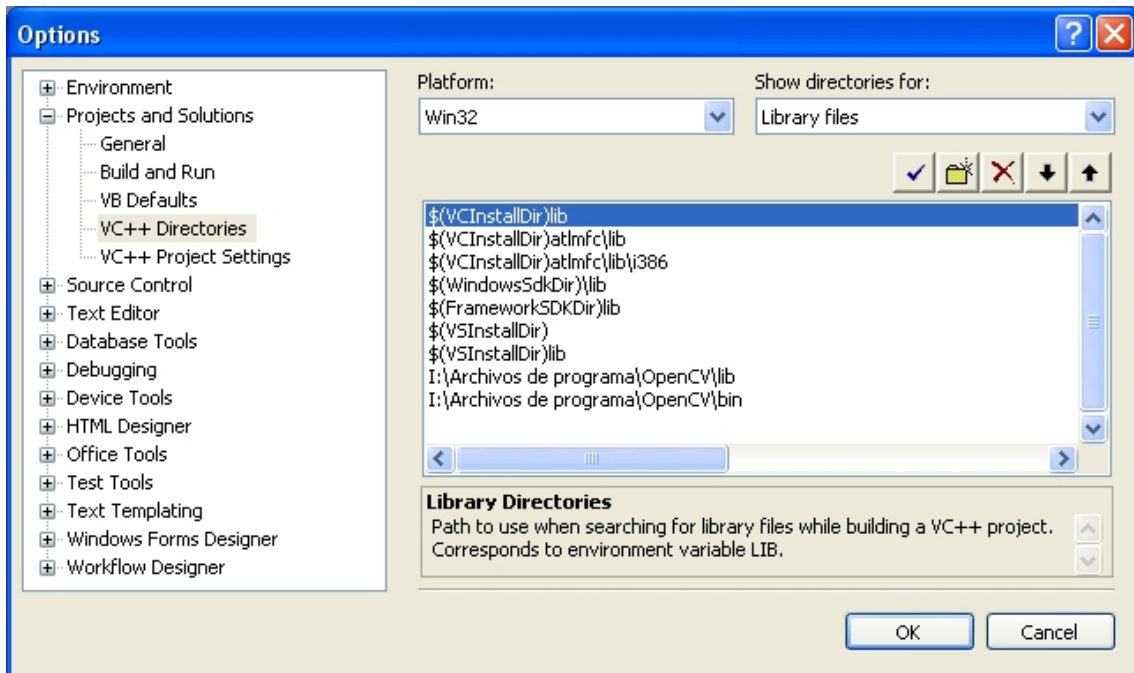
Además de configurar las propiedades del proyecto, es necesario añadir las rutas de librerías y código fuente en las opciones de Microsoft Visual Studio para que se busquen los archivos include en tiempo de compilación. Para ello hay que acceder a la sección Options dentro del menú Tools. Como se ha mencionado, esta configuración es relativa al software Visual Studio y no al proyecto de visión artificial, con lo que cada vez que este se abra en un Visual Studio diferente habrá que configurar esta parte. Para añadir los directorios requeridos hay que acceder a la sección Projects and Solutions y pulsar en VC++ Directories. Aquí hay que mostrar las carpetas de Include mediante el desplegable superior derecha y añadir las que debajo se muestran.



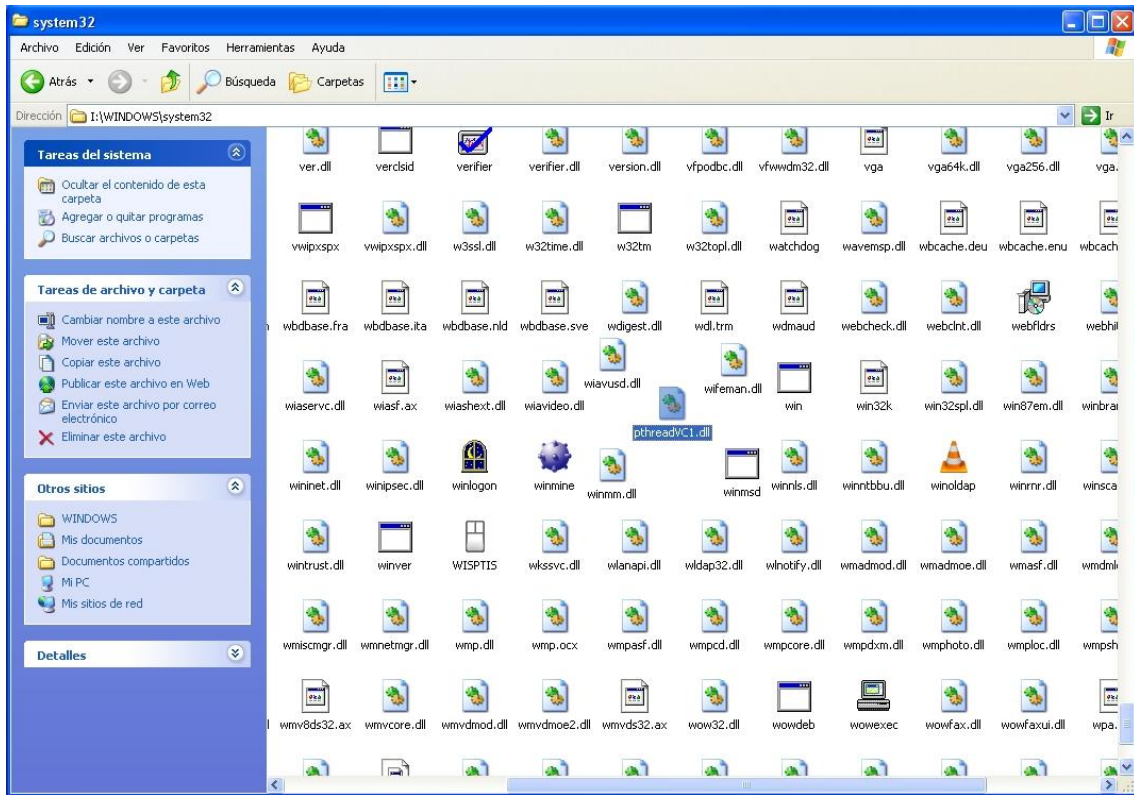
El mismo proceso es necesario realizar para que el compilador encuentre los ficheros fuente de OpenCV. Para ello se deben mostrar los ficheros fuente mediante el mismo desplegable que el anterior y añadir los citados directorios en la imagen de debajo.



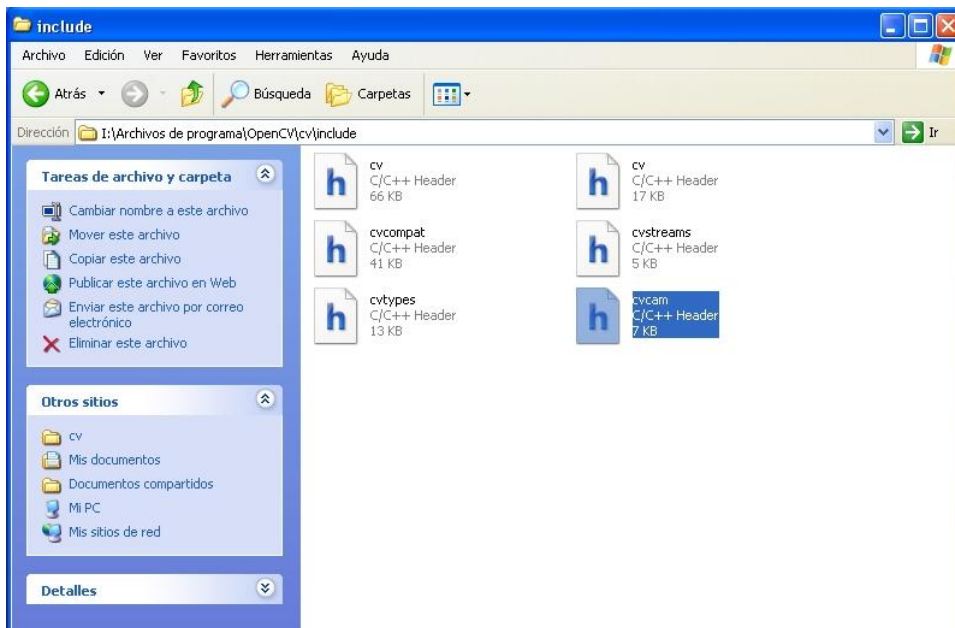
Por último, hay que hacer lo mismo con los directorios de librería que en este caso son dos.



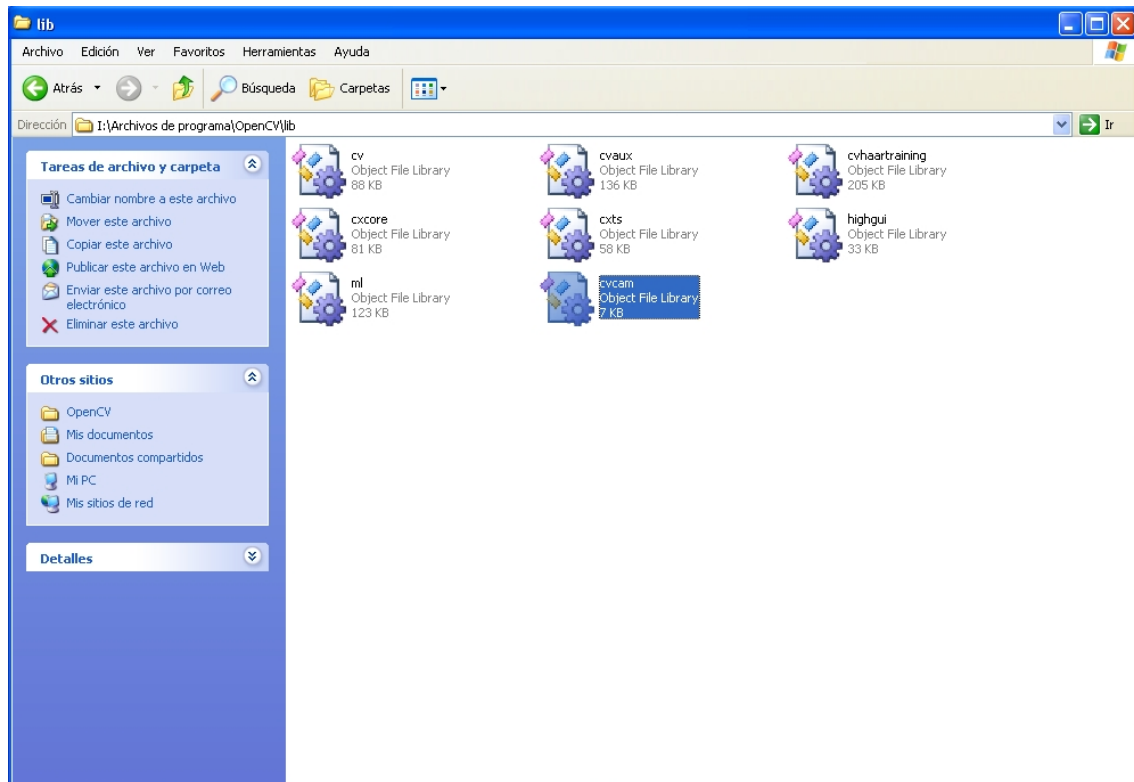
Para que la librería de hilos Pthread funcione, necesita que se aloje el archivo pthreadVC1.dll en la carpeta System32 dentro del directorio de instalación del sistema operativo, WINDOWS.



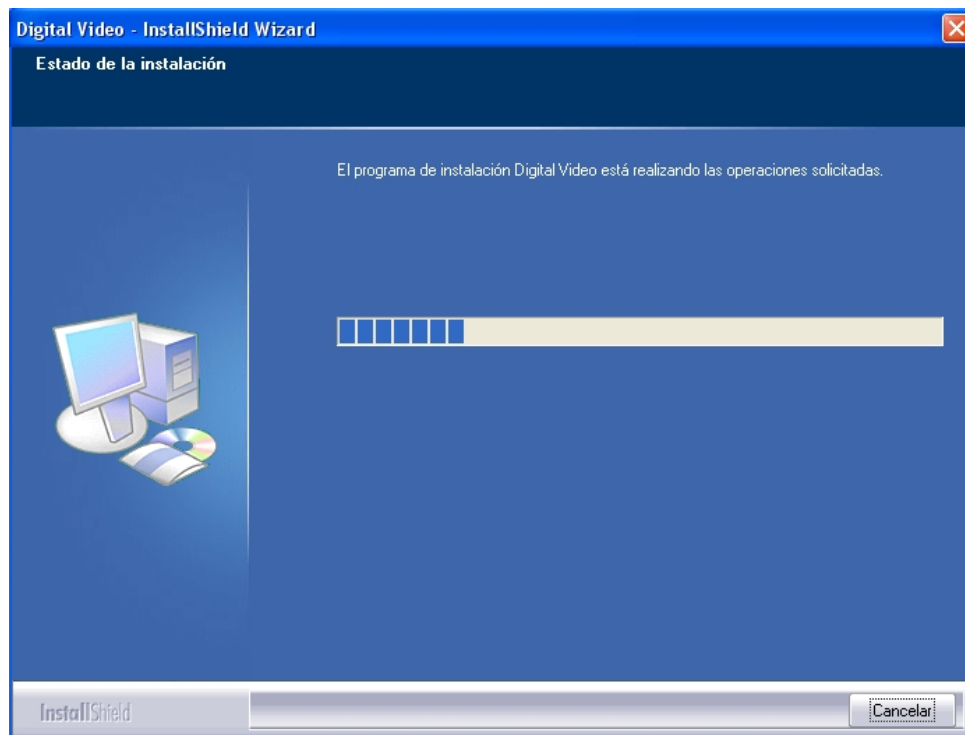
Si la versión de OpenCV instalada no contiene la librería cvcam, es necesario añadirla al directorio de instalación de OpenCV. Para ello es necesario pegar el archivo cabecera cvcam.h dentro de la carpeta cv\include.



Asimismo, hay que pegar el archivo `cvcam.lib` en la carpeta `lib` dentro de la ruta de instalación de OpenCV.



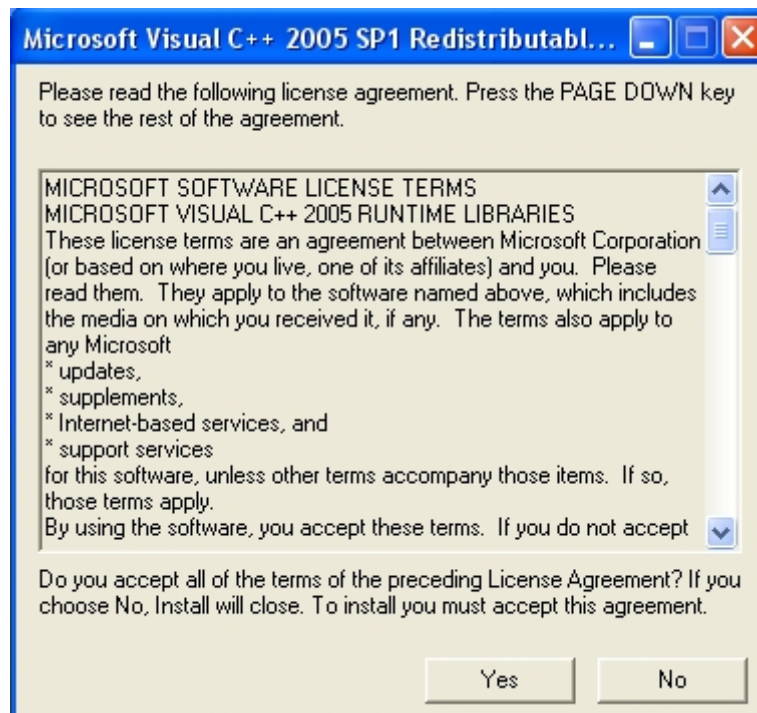
Puede que sea necesario instalar el driver de la cámara que se va a conectar al ordenador si el sistema operativo no reconoce correctamente el nuevo dispositivo.



Es posible que al intentar arrancar la aplicación desde la plataforma de desarrollo Visual Studio se muestre el siguiente error y no permita que esta se inicie.



En este caso, es muy probable que haya que instalar las librerías Microsoft Visual C++ 2005 SP1 Redistributable. Desde la web de Microsoft es posible descargar el paquete y una vez instalado la aplicación debería arrancar correctamente.



## BIBLIOGRAFÍA

- [1] DETECCIÓN DE LA POSICIÓN DE OJOS, NARIZ Y LABIOS DENTRO DE LA IMAGEN DE UNA CARA. UPNA - Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación. PFC 29-09-2008. Javier Lapieza Fernández
- [2] AUTOMATIC EXTRACTION OF HEAD AND FACE BOUNDARIES AND FACIAL FEATURES. Frank Y. Shih. Chao Fa Chuang. Ed. Elsevier
- [3] IMPLEMENTACIÓN DE UN SISTEMA AUTOMÁTICO DE LOCALIZACIÓN DE REGIONES FACIALES EN VÍDEO POR WEBCAM. UPNA- Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación. PFC 15-02-2010. Ioritz Cia Ulacia
- [4] LOCALIZACIÓN AUTOMÁTICA DE REGIONES FACIALES EN SECUENCIAS DE VIDEO A COLOR. UPNA - Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación. PFC 24-07-2008. Marcos Luqui Arilla
- [5] AUTOMATIC LOCATION AND TRACKING OF THE FACIAL REGION IN COLOR VIDEO SEQUENCES. N. Herodotou, K.N. Plataniotis, A.N. Venetsanopoulos. Signal Processing: Image Communication 14 1999. Pp. 359-388.
- [6] LEARNING OpenCV. COMPUTER VISION WITH THE OpenCV LIBRARY. Gary Bradski, Adrian Kaehler. O'Reilly. 2008
- [7] OPEN SOURCE COMPUTER VISION LIBRARY. Manual de referencia.
- [8] THE C PROGRAMMING LANGUAGE. Brian W. Kernighan, Dennis M. Ritchie. Prentice Hall 1988.