**TABLA DE CONTENIDOS**

# 1. Introducción

En esta memoria se recogen las decisiones y planteamientos tomados durante el transcurso del desarrollo de mi proyecto final de carrera.

La memoria está estructurada de la siguiente manera:

Capítulo 1: Presentación y descripción del proyecto. Además, se explican cuáles han sido los motivos de plantearse el proyecto y los objetivos finales propuestos.

Capítulo 2: Se evalúan las tareas concretas que se deben realizar en el transcurso del proyecto.

Capítulo 3: Se hace un estudio exhaustivo de las instrumentaciones y conocimientos previos que se deben realizar para ubicar el entorno de trabajo y llevar a cabo el proyecto. Se hace un análisis previo al diseño, centrado en los métodos que se utilizarán en el componente de detección de la cara y ojos.

Capítulo 4: En el diseño, se hace una explicación centrada con más detalles en cada uno de los componentes que envuelve el proyecto.

Capítulo 5: En la implementación se plasma todo el diseño de las partes que componen el sistema del proyecto.

Capítulo 6: Muestra los resultados de los tests diseñados para comprobar el comportamiento de los componentes y su posible interpretación.

Capítulo 7: Presenta las conclusiones finales del proyecto.

## 1.1. Definición del proyecto

El principal objetivo del proyecto es plantear una interfaz gráfica que permita cambiar el punto de vista de la escena de forma interactiva y natural mediante la detección de la cara utilizando algoritmos de visión por computadora. Se desea también mejorar la fiabilidad y la velocidad de procesamiento de detección de caras humanas mediante el uso de una combinación de diversos métodos de detección facial.

La idea está orientada hacia los sistemas interactivos, es decir, hacia las aplicaciones donde interactúan la persona y la computadora. El sistema se basa en el uso de una cámara y de un módulo “inteligente” para reconocer la imagen. Se utiliza la tecnología de visión por computadora para percibir la posición del usuario. Este tipo de aplicaciones tienen tendencia a crecer en un futuro próximo, donde

cada vez hay más demanda de su uso, y por esto mismo se requiere una mejora en la fiabilidad y la velocidad de procesamiento de este módulo inteligente para reconocer la imagen.

En este proyecto se implementa un sistema “EyeTrack” que permite la interacción del usuario con modelos virtuales tridimensionales. Este proyecto se considera como prototipo inicial de un sistema interactivo que permite la interacción en proyecciones o pantallas grandes al público en general sin necesidad de dispositivos especiales y de forma sencilla e intuitiva. Mediante el movimiento de cabeza se cambia el punto de vista de la escena, simulando las variaciones producidas en la perspectiva por giros y traslaciones de la cabeza.

Actualmente, entre todos los métodos, cada uno tiene sus ventajas e inconvenientes, o bien en fiabilidad o bien en velocidad. Para mejorar la fiabilidad y la velocidad de detección, se ha propuesto la idea de utilizar una combinación de diferentes métodos.

## 1.2. Motivación del proyecto

La motivación principal del proyecto es intentar mejorar la fiabilidad y la velocidad de la detección de caras.

Éste es un proyecto interesante y cautivador por muchos motivos: principalmente, destacaría la importancia de la influencia, hoy en día, de los sistemas persona-computadora.

La tecnología de detección de caras es una de las necesidades de desarrollo más importantes de la investigación interactiva. Desde el punto de vista de la tendencia de la creciente popularidad de las computadoras, la naturalización de la interfaz persona-computadora es una de las orientaciones importantes de desarrollo en el futuro. El humano propone el concepto de los sistemas inteligentes de interfaz hombre-máquina, esperando que las máquinas tengan un nivel de inteligencia igual o similar a un humano. La comunicación entre humano y máquina debería ser en el futuro igual de fácil como entre humano y humano.

Es un desafío encarar un proyecto de este calibre, donde intervienen todos los estamentos de un sistema interactivo. Es también un reto personal aprender y entender el funcionamiento global de las técnicas que componen el proyecto. Cada una de las partes significa un enfrentamiento individual para conocer en profundidad los detalles, para posteriormente extraerlo y adaptarlo en el proyecto.

## 1.3. Objetivos del proyecto

Los objetivos se organizan de la siguiente manera, formando partes diferenciadas:

Primero de todo, el proyecto consiste en implementar los módulos de detección facial. En este caso es implementar el método Haar, el método de tracking-color y el método combinado para localizar la posición de los ojos del usuario de este sistema. La implementación de estos 3 métodos es para, posteriormente, comprobar las mejoras en fiabilidad y velocidad de procesamiento entre ellos.

La segunda parte del proyecto se resume en la implementación del módulo de detección de ojos, pupilas y boca.

La última parte del proyecto se resume en la implementación del módulo de seguimiento de la cara. Una vez realizados los procesos de la primera y de la segunda parte, se procederá a extraer las coordenadas de los ojos y de la cara de la imagen de entrada. Con estas coordenadas, el módulo de seguimiento tiene que calcular la posición y orientación real de la cara humana con más o menos exactitud.

Además se implementará una escena 3D, en este caso será una cámara enfocando a un castillo para simular la vista real de un usuario al monitor. Paralelamente a la detección y seguimiento, el sistema debe modificar interactivamente el punto de vista de esta escena 3D acorde con la localización de los ojos.

Aparte de los anteriores objetivos, en el proyecto se debe diseñar un sistema que sea amigable, comprensible y visual.

## 1.4. Entorno de trabajo

Para poder llevar a cabo el proyecto, se ha tenido que escoger con qué plataforma realizar el proyecto y algunas herramientas concretas para desarrollar el trabajo.

La implantación del proyecto se realizará en Microsoft Visual Studio C++ 2005 para la plataforma Windows XP.

Para la gestión de las ventanas y el sistema de ficheros se utilizará la librería MFC.

Para la implementación de los métodos de detección facial se utilizará la librería de visión por computadora – OpenCV[1].

Para implementar la representación tridimensional se utilizará la librería OpenGL[2].

# 2. Planificación

Una de las partes más comprometidas en todo proyecto es su planificación. Es una parte importante del estudio del proyecto que nos permitirá llevar de forma exhaustiva cada tarea en un tiempo razonable.

El punto de inicio de la planificación son las tareas a realizar durante el transcurso del proyecto.

## 2.1. Tareas a realizar

A continuación se muestra un cuadro de tareas a realizar a lo largo del proyecto junto con su duración estimada (en horas y días). Se ha realizado con la dedicación de 8 horas diarias de trabajo, 5 días a la semana.

| Descripción de tareas | Duración estimada | | Fecha inicio | Fecha fin |
|---|---|---|---|---|
| Estudio de los componentes del proyecto | 10 días (80 horas) | | 25/02/09 | 10/03/09 |
| Estudio de los métodos faciales | 5 días | 40h | | |
| Estudio de la visualización interactiva | 2 días | 16h | | |
| Estudio de la visualización realista | 3 días | 24h | | |
| Análisis y diseño | 19 días (152 horas) | | 11/03/09 | 06/04/09 |
| Adquisición | 1 día | 8h | | |
| Detección de la cara | 3 días | 24h | | |
| Localización de los ojos | 5 días | 40h | | |
| Localización de la boca | 3 días | 18h | | |
| Seguimiento de la cara | 4 días | 32h | | |
| Interfaz 3D | 3 días | 24h | | |

| Implementación | 22 días (176 horas) | | 07/04/09 | 06/05/09 |
|---|---|---|---|---|
| Adquisición | 1 día | 8h | | |
| Detección de la cara | 4 días | 120h | | |
| Localización de los ojos | 6 días | 80h | | |
| Localización de la boca | 4 días | 24h | | |
| Seguimiento de la cara | 4 días | 48h | | |
| Interfaz 3D | 3 días | 48h | | |
| Testing y optimización | 7 días (56 horas) | | 07/05/09 | 15/05/09 |
| Documentación | 13 días (104 horas) | | 18/05/09 | 03/06/09 |
| Total | 71 días (568 horas) | | | |

Tabla 2.1 Estimación inicial detallada

## 2.2. Análisis económico global

En el análisis económico global debemos tener en cuenta las diferentes herramientas utilizadas para el desarrollo del proyecto. Por un lado, debemos cuantificar los costes del personal, teniendo en cuenta principalmente los tiempos estimados de realización del proyecto. Por otro lado, los costes del material *software* y *hardware* utilizados.

Finalmente se completarán con los gastos generales por materiales utilizados y por último se establecerá el coste total del proyecto.

Concepto de recursos humanos

El personal que ha trabajo en la realización de este proyecto se reduce únicamente a un solo trabajador, aunando los perfiles de analista y programador.

| | |
|---|---|
| (568-80 de estudio = 488) horas de analista-programador (35 euros / hora) | 33.040 euros |
| | **17.080 euros** |

Concepto de recursos hardware

La máquina utilizada para el desarrollo del proyecto se considera como nuevas adquisiciones. Se dividen en dos grupos: la máquina monoprocesador sobre la que se ha desarrollado todo el proyecto y la cámara.

| | |
|---|---|
| Monoprocesador Intel | 1.500 euros |
| Cámara | Gratuito |
| | **1.500 euros*** |

Concepto de recursos software

| | |
|---|---|
| Monoprocesador Intel | |
| Microsoft Windows XP SP2 Professional Edition | 300 euros |
| Microsoft Office 2003 | 450 euros |
| Visual Studio 2005 Professional Edition | 469 euros |
| Librería OpenCV | Gratuito |
| Librería OpenGL | Gratuito |
| | **1.219 euros*** |

| | |
|---|---|
| *Imputar el proyecto el coste asociado a la amortización a 3 años | |
| (1500 euros + 1219 euros) x 4 meses / 36 meses | 303 euros |
| | **303 euros** |

| | |
|---|---|
| Total | **17.383 euros** |

# 3. Conocimientos previos y análisis

## 3.1. Sistema interactivo

En estos últimos años el uso de la tecnología interactiva cada vez atrae más la atención de la sociedad. Todos los productos relacionados con esta nueva tecnología, como los juegos interactivos, la proyección interactiva o la tecnología multimedia interactiva, poco a poco van entrado en nuestra vida.

La interactividad se podría definir como la reacción rápida y coherente de un sistema en base a los movimientos voluntarios o intencionados del usuario. El usuario que quiere acceder a cierta información utilizará primero la vista. La mayoría de las veces lo hará mediante los ojos.

Un sistema interactivo genérico mediante la visión por computadora incluiría los siguientes equipos y módulos:

- Cámara: el equivalente a la vista. Puede ser una simple cámara de luz visible o luz infrarroja.
- Módulo de reconocimiento: Se utiliza para analizar y reconocer las imágenes de video que está capturando la cámara con el fin de obtener una información específica. De entre todas las herramientas de desarrollo, el OpenCV es una buena plataforma para realizar dicha actividad.
- Módulo de entrada de la aplicación: Se envían los datos obtenidos del módulo anterior a una aplicación con un protocolo predefinido.
- Aplicación: en nuestro caso se genera una escena 3D o 2D, de acuerdo con la información proporcionada por el módulo anterior. Cambiaría el punto de vista o realizaría cierta acción.

Unos ejemplos de aplicación interactiva podrían ser:

- Juegos interactivos: el uso de cámaras para capturar todos los movimientos y las expresiones del cuerpo humano. El objetivo es llegar a sustituir los datos obtenidos mediante una interfaz hombre-máquina tradicional -como el ratón y el teclado- por los obtenidos mediante la captura de movimientos.
- Visualización interactiva: utilizar una cámara para identificar y obtener la ubicación de las manos, dedos, extremidades u otros elementos sobre la pantalla de proyección, para conseguir diversas funciones como elegir, abrir ventana, cambio de imagen, consultas etc.

- Pizarra: utilizar la cámara para identificar el movimiento de la pluma o dedo de la mano. Junto con el uso de un proyector, sustituir la pizarra y la tiza por una pantalla electrónica más un lápiz electrónico o un dedo. El objetivo es lograr el control de la aplicación con movimientos de mano como entrada.
- Etc.

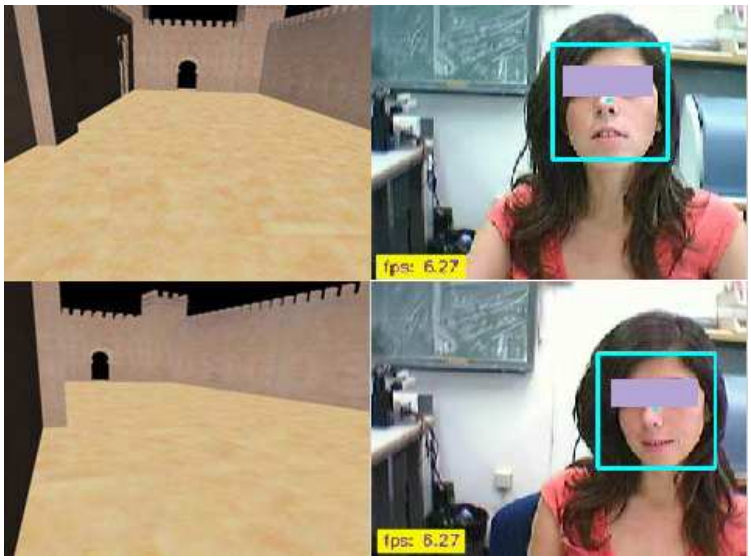


Figura 3.1 Sistema interactivo con localización de la cara

## 3.2. Estructura del ojo humano

El ojo ha sido denominado "el órgano más complejo del cuerpo humano". Brevemente, la parte posterior del ojo está alineada con una capa de la retina que actúa, en gran medida, como la película de una cámara. La retina es una membrana que contiene las células nerviosas que sirven como foto-receptor.

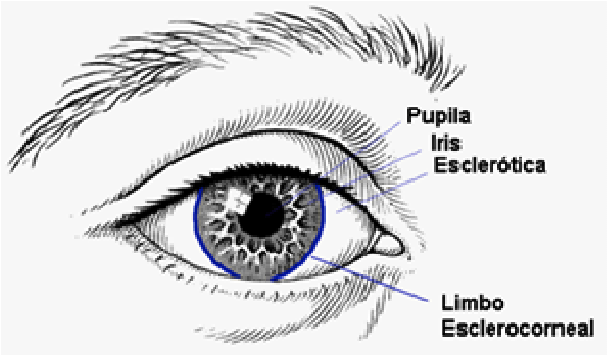


Figura 3.2 Vista frontal del ojo humano

Las células nerviosas tipo foto-receptor de la retina cambian los rayos de luz que entran a través de la pupila en impulsos eléctricos y los transmiten a través de los nervios ópticos a la parte visual del cerebro donde una imagen es percibida.

La posición del ojo en la cabeza decide el campo de visión. Cada ojo percibe una imagen y el cerebro las junta y procesa como una (campo binocular). Cuando los ojos se encuentran en ambos lados de la cabeza, el punto de vista es casi panorámico, pero hay una pérdida de percepción de profundidad estereoscópica. Los seres humanos tienen un campo de visión total de entre 160 a 208 grados, unos 140 grados más o menos. Cada ojo tiene un campo de visión regular entre 120 a 180 grados, mientras que un perro tiene un campo de visión total cerca de 280 grado con 180 a 190 grados para cada ojo y 90 grados de campo binocular.

La propiedad más importante de nuestro sistema de movimiento de los ojos es que puede mover el ojo desde un punto de la mirada a una nueva ubicación de la mirada con gran rapidez. Estos movimientos oculares llamados sacádicos son más rápidos que los movimientos del cuerpo que puede hacer. Se pueden girar los ojos más de 500 grados/segundo, y se hacen más de cien mil de estos movimientos sacádicos durante el día. Estos rápidos movimientos oculares se realizan por una serie de seis músculos adjuntos a la parte exterior de cada ojo. Éstos están dispuestos en tres pares del tipo agonista-antagonista: un par de rotación del ojo horizontal (izquierda - derecha), otros dos giran el ojo en vertical (arriba - abajo) y los terceros permiten 'ciclooxigenasa torsión', o la rotación sobre la línea de visión.

Normalmente un sistema de seguimiento de los ojos debería indicar la posición de iris y pupila en la zona del ojo con las propiedades de escala y rotación invariable bajo las siguientes condiciones:

- Movimientos de ojos sacádicos, mientras la cara está parada.
- Los ojos están parados, pero la cara se está moviendo.
- Se está moviendo la cara y los ojos están haciendo movimientos sacádicos.

## 3.3. Espacio de color

El color está formado por dos componentes: crominancia y luminancia. El valor de crominancia de un objeto identifica la propiedad del color de ese objeto; la luminancia, por otra parte, es una propiedad del entorno del objeto. Toda imagen en color tiene unos valores de crominancia y luminancia.

Un espacio de color es un modelo matemático que describe la forma del color.

Pueden ser representados en forma de números, por lo general formados por tres o cuatro componentes. El primer paso en esta tesis es detectar los ojos dentro de la cara humana en una imagen mediante el color de la piel de las caras. La detección de un objeto utilizando su color puede causar problemas con cambios de luz ambiental o con la aparición de objetos de este color. Las imágenes en color suelen estar representadas en el espacio de color RGB –que son el rojo, el verde y el azul-. Estos tres componentes residen en tres matrices. Cada una de ellas tiene las mismas dimensiones que la imagen original RGB y cada posición contiene la cantidad de color de cada píxel. La combinación de las tres componentes dan el color RGB original. Sin embargo, cualquier variación de la luz ambiental puede cambiar mucho los valores RGB de los píxeles. Por lo tanto, se debe utilizar un espacio de color más robusto que pueda funcionar en ambientes con luz variable.

En este proyecto se ha utilizado el espacio de color HSV para la detección de la piel y de los ojos. Antes de proceder a la parte de métodos que describen los algoritmos para la detección de los ojos, se hará un resumen con información acerca de los diferentes espacios de color.

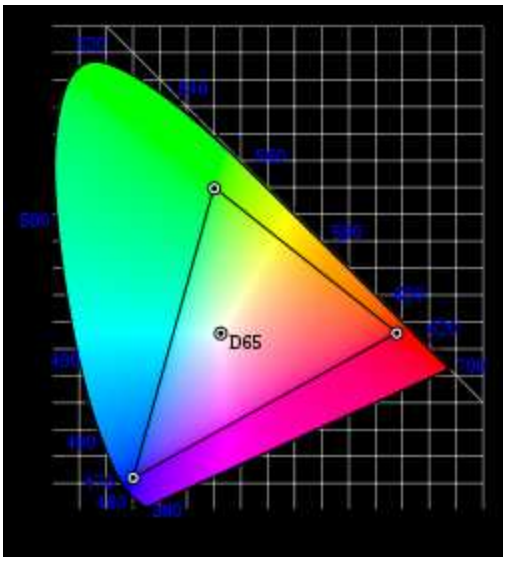


Figura 3.3 Visualización del espacio RGB

### 3.3.1. Espacio de color GRAY

Una imagen en escala de grises es simplemente una en la que los colores son sólo los tonos de gris. La razón para la conversión de imágenes en color a escala de grises es que se requiere menos información para cada píxel. De hecho, una de color gris es aquella en la que el rojo, verde y azul tienen la misma intensidad en el espacio RGB, por lo que sólo es necesario especificar un valor único de intensidad

para cada píxel, mientras que tres valores son necesarios para cada píxel en una imagen a color. Una fórmula común de conversión de RGB a Gray es:

$$[Gray] = [0.212 \quad 0.715 \quad 0.072] \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

A menudo, la intensidad en escala de grises se almacena como un número entero de 8 bits que son 256 posibles diferentes tonos de grises de negro al blanco. Si los niveles son uniformemente espaciados la diferencia entre los sucesivos niveles de grises es significativamente mejor que la resolución de niveles de grises del ojo humano.

Hoy en día las imágenes en escala de grises son muy comunes en la tecnología y en algoritmos de procesamiento de imágenes. Éstos son totalmente suficientes para muchas tareas donde no hay necesidad de utilizar la imagen en color, que es más complicada y difícil de procesar.

### 3.3.2. Espacio de color HSV

El sistema de coordenadas HSV es cilíndrico y el modelo se define como un cono. Define la posición vertical como brillo, la posición angular como HUE[1] y la posición radial como saturación. HUE puede oscilar entre 0 y 360, pero en algunas aplicaciones estos valores están normalizados para 0-100%. El rango de saturación es de 0 a 100% y especifica la posición relativa desde el eje vertical al lado del cono. El valor es el brillo del color y su rango es de 0 a 100%.

Los artistas normalmente prefieren utilizar el modelo de color HSV alternativo a los clásicos espacios de color como RGB, porque es similar a la manera con que los seres humanos tienden a reconocer el color. Por ejemplo, podríamos querer cambiar el color de un automóvil de color amarillo brillante a azul, pero queremos dejar el resto de la escena como estaba -sombras en el coche, etc-. Esto sería una tarea difícil en RGB, pero es relativamente simple en HSV. Debido a que los píxeles de color amarillo del coche tienen un rango especifico de tono, con independencia de la intensidad o saturación, los píxeles pueden ser fácilmente aislados y su componente de tono modificado, lo que permite a un automóvil darle diferentes

[1] HUE (matiz): En teoría del color, una tonalidad es un color "puro", o caracterizado por una sola longitud de onda en el interior del espectro visible (o espectro óptico) de la luz.

colores. Como la mayoría de los procesamientos digitales de imágenes en funcionamiento de los sistemas son en RGB, la operación descrita anteriormente se podría realizar en tres etapas:

- Convertir la imagen RGB a HSV
- Modificar el valor de tono
- Por último, convertir la imagen de vuelta a RGB

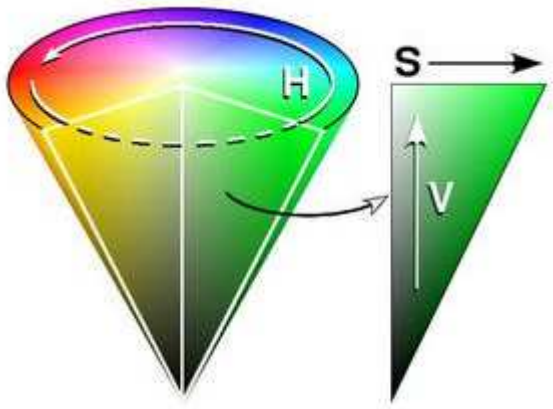


Figura 3.4 Visualización del espacio de color HSV

Hasta el momento, muchos algoritmos de detección de caras se basan en la detección del color de la piel primero. Un error común es que los diferentes modelos de color son necesarios para diferentes razas humanas. En realidad esto no es cierto porque casi todos los seres humanos tienen aproximadamente el mismo tono [4]. Pero el color de la piel humana puede abarcar casi todos los valores de S para un rango limitado de valor HUE tal como se describe en [3]. Como resultado el valor HUE de las imágenes se pueden utilizar en los algoritmos de detección de caras.

### 3.3.3. Espacio de color XYZ

El espacio XYZ de colores permite que éste se exprese como una mezcla de los tres triestímulos: los valores X, Y y Z. El término triestímulo viene del hecho de que los resultados de la percepción del color que vienen de la retina del ojo responden a tres tipos de estímulos. Después de la experimentación, el CIE[2] estableció una

[2] CIE: Establecida en 1913 y con sede en Viena, Austria, la Comisión Internacional de Iluminación es la autoridad en luz, iluminación, color y espacios de colores.

hipotética serie de conjuntos primarios, XYZ, que corresponden a la forma con que la retina del ojo se comporta. La CIE define los conjuntos primarios de manera que todos los mapas en luz visible son una mezcla de X, Y y Z. En general, las mezclas de X, Y, Z y los componentes utilizados para describir un color se expresan en porcentajes de entre el 0 por ciento hasta, en algunos casos, poco más de 100 por ciento.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412411 & 0.357585 & 0.180454 \\ 0.212649 & 0.715169 & 0.072182 \\ 0.019332 & 0.119195 & 0.950390 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$

Durante los experimentos que se han realizado, se ha observado que el uso de la componente Z del espacio de color XYZ de una imagen de la cara da mejores resultados en lugar de su versión en escala de grises.

### 3.3.4. Espacio de color Lab

Lab, a veces denominado CIELAB, es el modelo de color más completo utilizado convencionalmente para describir todos los colores visibles al ojo humano. Los tres parámetros del modelo son: L es la luminancia, a es la posición entre el rojo y el verde, b es la posición entre el amarillo y el azul. La formula de conversión de espacio de color RGB al espacio de color Lab es:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.433910 & 0.376220 & 0.189860 \\ 0.212649 & 0.715169 & 0.072182 \\ 0.017756 & 0.109478 & 0.872915 \end{bmatrix} \times \begin{bmatrix} R/255 \\ G/255 \\ B/255 \end{bmatrix}$$

$$L = 116 \times Y^{1/3} \; for \; Y > 0.008856$$

$$L = 903.3 \times Y \; for \; Y <= 0.008856$$

$$a = 500 \times (f(x) - f(y))$$

$$b = 200 \times (f(y) - f(z))$$

$$where$$

$$f(t) = t^{1/3} \; for \; t > 0.008856$$

$$f(t) = 7.787 \times t + 16/116 \; for \; t <= 0.08856 \tag{3.3}$$

### 3.3.5. Espacio de color YCbCr

El espacio de color YCbCr se basa en la luminancia y crominancia. Se deriva de RGB

utilizando la representación:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & 0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (3.4)$$

Donde Y es la luminancia o componente brightness y Cb y Cr son componentes de color azul y rojo respectivamente.

## 3.4. Image Thresholding

*Thresholding* usa el método del valor umbral. En muchas aplicaciones de visión por computadora es útil para poder separar las regiones de la imagen correspondientes a los objetos en los que estamos interesados de las que corresponden a la imagen de fondo. El umbral a menudo proporciona una forma fácil y conveniente para realizar esta segmentación sobre la base de las diferentes intensidades en el primer plano y las regiones de una imagen. Generalmente el umbral se puede dividir en dos categorías: umbral global y umbral adaptable.

### 3.4.1. Umbral global

Umbral global es un método para convertir una imagen en escala de grises en imagen binaria, que es un tipo especial de escala de grises que sólo tiene dos valores de píxel: blanco y negro, utilizando el valor umbral para toda la imagen. Unos únicos o múltiples niveles de umbral pueden ser determinados. Dado un único umbral, cada píxel de la imagen se compara con el umbral y si la intensidad del píxel es superior al umbral, el píxel se establece como blanco (o negro) en la salida; del mismo modo, si es inferior al umbral, se establece como negro (o blanco). Para varios umbrales, se establecen bandas de valores de intensidades que se establecen en color blanco, mientras que las regiones de la imagen fuera de estas bandas se establecen en negro. Umbral global con múltiples niveles podría estar representado como:

$$O(i,j) = \begin{cases} 0 \; if \;\; I(i,j) \in Z \\ 1 \quad otherwise \end{cases} \quad (3.4)$$

Donde Z es un conjunto de valor intensidad.

Figura 3.5 Imagen original e imagen con umbral global de valor 80

### 3.4.2. Umbral adaptable

Umbral global utiliza un mínimo fijado para todos los píxeles de la imagen y, por tanto, funciona sólo si la intensidad del histograma[3] de la imagen de entrada contiene claramente separados los picos correspondientes al objeto y al fondo. Por lo tanto, no puede hacer frente a las imágenes que contienen, por ejemplo, un fuerte gradiente de iluminación. Por el contrario, umbral adaptable selecciona un umbral individual para cada píxel sobre la base del rango de valores de intensidad de su vecindario local. Esto permite al umbral de una imagen cuyo histograma global de intensidad no contiene picos diferenciados poder superar el problema de la evolución de las condiciones de iluminación en la imagen.

Existen diferentes enfoques para el cálculo del umbral de valor local:

$$Threshold = mean \;\; or \;\; Threshold = median \;\; or \;\; Threshold = \frac{\min + \max}{2} \tag{3.5}$$

El punto crítico en el umbral adaptable es elegir correctamente el tamaño del vecindario para el píxel. El tamaño del barrio tiene que ser lo suficientemente grande como para cubrir un número suficiente de píxeles del objeto y del fondo, de otro modo será un mal umbral. Por otra parte, la elección de las regiones que son demasiado grandes pueden violar la presunción de que la iluminación debe ser más o menos uniforme. Umbral adaptable teniendo en cuenta los valores del histograma de intensidad, está fuera del alcance de este trabajo.

[3] Histograma: En estadística, un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. En el eje vertical se representan las frecuencias, y en el eje horizontal los valores de las variables, normalmente señalando las marcas de clase, es decir, la mitad del intervalo en el que están agrupados los datos.

Figura 3.6 Imagen original e imagen con umbral adaptable

En esta tesis se han utilizado los dos modelos: tanto el umbral global como el adaptable para filtrar la entrada a otras operaciones tales como la detección del círculo para detectar las pupilas, la detección de bordes y se ha observado que, sin umbral, estas operaciones darían un mal resultado.

## 3.5. Detección de caras en el área de visión por computadora

En este proyecto estamos interesados en la detección de los ojos de una sola persona en un vídeo. Por lo tanto, aunque el término "detección" se utiliza para encontrar la ubicación de varios objetos en una imagen, en este proyecto se utilizará la expresión "detección de caras" para encontrar la cara de una única persona en una secuencia de imágenes.

El primer problema para nuestro sistema de seguimiento automático de los ojos es la detección de la cara humana en una secuencia de imágenes. Después de la detección de la cara en la imagen, los ojos se pueden buscar y extraer por sus propias características. Como nuestro objetivo principal en la detección de caras es encontrar un área de búsqueda para la detección de los ojos, necesitamos un método rápido de detección de caras independiente de los componentes estructurales de la cara, como la barba o el bigote. Aunque se podría haber escogido algún método simple de detección de caras, se ha preferido hacer un estudio detallado.

El sistema de visión humano puede detectar y reconocer caras en las imágenes. Las prestaciones de los sistemas de visión humanos son tan altas que también se pueden encontrar múltiples caras en la misma escena con diferentes poses, expresiones faciales, escalas, etc. Una visión parcial de una cara para los seres humanos es suficiente para detectarla en las imágenes. Hoy en día, lamentablemente en la tecnología de visión por ordenador todavía no se dispone de ningún sistema que pueda alcanzar este rendimiento. Sus operaciones dependen de ciertas condiciones controladas.

Hay alrededor de 150 técnicas diferentes de detección de caras en imágenes. Muchas de ellas comparten algunos métodos comunes a través de sus formas. Un estudio detallado acerca de los distintos métodos de detección de caras se da en la encuesta de Yang [6] que los clasifica en cuatro categorías. También en [5] se da otro estudio sobre los métodos de detección de caras.

En el siguiente apartado se explican y analizan métodos de detección de caras y ojos relacionados con este proyecto.

### 3.5.1. Método basado en el conocimiento

Las reglas del método están basadas en el conocimiento humano de lo que constituye una típica cara humana. Por lo general, las reglas capturan las relaciones entre los rasgos faciales.

Estos métodos están diseñados principalmente para la localización de cara. Los métodos utilizan reglas simples para describir las características de una cara, tales como que una cara normalmente aparece en una imagen con dos ojos que son simétricos entre sí, una nariz y una boca. Las relaciones entre las características se pueden representar por las distancias y posiciones relativas.

El problema con este enfoque está en que es difícil traducir el conocimiento humano en una buena definición de reglas. Si estas reglas son demasiado estrictas puede ocurrir que no se detecte la cara por no pasar todas las reglas. Pero, por otro lado, si las reglas son demasiado generales puede haber muchas falsas detecciones.

Un trabajo acerca de este planteamiento fue realizado por Yang y Huang [7]. Ellos utilizaron una jerarquía basada en el conocimiento para la detección de caras. Su sistema consta de tres niveles de reglas. Al más alto nivel, todos los candidatos posibles son encontrados desde una imagen de entrada mediante la aplicación de un conjunto de reglas a cada localización. Las reglas de alto nivel son una descripción general de lo que parece una cara, mientras que las reglas en los niveles inferiores se basan en la detección de los rasgos faciales.

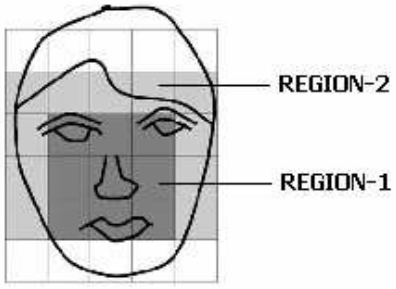


Figura 3.7 Una típica cara que utiliza el método de conocimiento arriba a abajo

Algunos ejemplos de reglas codificadas que se basan en las características de la cara humana utilizados para localizar candidatos en la resolución más baja pueden ser:

- “la parte central de la cara (región-1) tiene cuatro células con intensidad básicamente uniforme”
- “la parte superior de la cara (región-2) tiene una intensidad básicamente uniforme”
- “la diferencia entre el promedio de los valores de color gris de la parte central y la parte superior es significativa”

En la imagen en resolución más baja (nivel 1) se buscan candidatos para ser caras y éstos serán buscados posteriormente en resoluciones mejores. En el nivel 2, se usan histogramas en las caras detectadas en el nivel 1 y seguidamente se detectan los bordes. Las regiones candidatas supervivientes se examinan en el nivel 3 con otro conjunto de reglas que responden a características faciales como los ojos o la boca.

### 3.5.2. Método basado en las características invariables

Las características invariables son las características faciales como los ojos, la nariz, la boca, etc. Este método es diferente al método basado en el conocimiento a priori porque este método reconoce desde abajo hacia arriba. Primero utiliza diversos medios para encontrar estas características, y luego integra todas estas características encontradas para determinar si la región es una cara humana. Un problema común de los métodos basados en características está en las condiciones ambientales como la iluminación, las sombras, etc. que podrían corromper los rasgos faciales.

### 3.5.3. Método basado en plantilla

El método basado en plantilla se puede dividir en dos tipos: plantilla programada y plantilla deformable.

La plantilla programada, en primer lugar elabora una plantilla de modelo estándar. A continuación, calcula los valores relacionados de la plantilla con la región de detección. Cuando los valores están de acuerdo con las directrices establecidas, se decide que la región es una cara humana.

La plantilla deformable, en primer lugar hay que definir los parámetros de la plantilla. A continuación se modificarán estos parámetros de la plantilla según los datos de la región de detección hasta que las dos convergen. Así se conseguirá el objetivo de detectar la cara humana.

### 3.5.4. Métodos basados en color

El color de la piel humana es un medio eficaz para detectar rostros. Diferentes espacios de colores han sido utilizados para la etiqueta de la piel incluyendo píxeles RGB [15], RGB normalizado [8], VHS [4], [9], [10], [11], YCbCr [12], YIQ [13] o XYZ [14]. En este proyecto se ha utilizado una distribución de probabilidad del color para realizar un seguimiento de las caras. Sin embargo, la detección de píxeles de la piel en la región por sí sola no es suficiente para localizar la cara. Para detectar el contorno de la cara, se deben utilizar algoritmos como los propuestos en [4]. Más información detallada se dará en la sección 3.7.

### 3.5.5. Métodos estadísticos

La detección de caras basada en la estadística es utilizar análisis estadísticos y métodos de máquinas de aprendizaje para encontrar las características de las muestras de la cara y de las muestras que no son caras. Después hay que utilizar estas características para construir los clasificadores y utilizar estos clasificadores para completar la detección de la cara.

#### 3.5.5.1. Método de sub-espacio (PCA y LDA)

El método de sub-espacio incluye dos tipos: análisis de componentes principales (PCA) y análisis discriminante lineal (LDA).

**Análisis de componentes principales (PCA)** también conocido como KL (KARHUNEN-loeve). Es una técnica tradicional de proyección sobre un subespacio para el reconocimiento de caras. Es probablemente la más utilizada también. Se tiene un set de imágenes de entrenamiento I. Como primer paso se computa la imagen promedio de I y se le resta a cada una de las imágenes de entrenamiento, obteniendo el set de datos:

$$i_1, i_2, \ldots, i_n \in I - \bar{I} \tag{3.6}$$

Luego se compone una matriz X tal que cada columna es una imagen de muestra. XXT es la matriz de covarianza de las muestras de entrenamiento y las componentes principales de la matriz de covarianza se computan resolviendo.

$$R^T(XX^T)R = \Lambda \tag{3.7}$$

Donde $\Lambda$ es la matriz diagonal de valores propios y R es la matriz de vectores propios orto normales.

Se puede ver geométricamente que R es una matriz de cambio de base que rota los antiguos ejes a los ejes propios, donde el vector propio con valor propio más grande se corresponde con el eje de máxima varianza, el asociado en el segundo más grande se corresponde con la segunda mayor varianza y es ortogonal con el anterior y así sucesivamente.

Finalmente, nos quedamos con los *n* vectores propios de mayor valor propio asociado. El parámetro de compresión en este caso es precisamente n dado que indica la dimensión del vector de características que va a representar a la imagen original. Cada coeficiente de la representación de la imagen se obtiene proyectándola sobre cada uno de los vectores de la base PCA.

**Análisis discriminante lineal (LDA)** es una técnica de aprendizaje supervisado para clasificar datos. La idea central de LDA es obtener una proyección de los datos en un espacio de menor (o incluso igual) dimensión que los datos entrantes, con el fin de que la separabilidad de las clases sea la mayor posible. Es una técnica supervisada ya que para poder buscar esa proyección se debe entrenar el sistema con patrones etiquetado. Es importante aclarar que LDA no busca en ningún momento minimizar el error de representación cometido, como sí lo hacía PCA.

Existen varias implementaciones de LDA, entre ellas se encuentra Fisher-LDA [15]. Para explicarlo vamos a considerar la versión más simple del problema:

Encontrar el vector *w* de proyección, que proyecte los datos a un espació uni-dimensional de manera que se obtenga la mayor separabilidad entre sus clase.

Formalizando, tenemos $x_1 \ldots x_n$ patrones d-dimensionales etiquetados en c clases. Cada clase cuenta con $N_c$ patrones. Se busca w, para obtener $y_i = w^T x_i$ proyecciones uni-dimensionales de los patrones.

Lo que busca Fisher-LDA es maximizar la siguiente función objetivo:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{3.8}$$

Donde $S_B$ es la matriz de dispersión inter-clase y $S_W$ es la matriz de dispersión intra-

clase. Siendo más precisos:

$$S_B = \sum_c N_c(\mu_c - \mu)(\mu_c - \mu)^T \tag{3.9}$$

$$S_W = \sum_c \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T \tag{3.10}$$

Siendo $\mu_c$ la media de cada clase, $\mu$ la media de todos los datos, $N_c$ la cantidad de patrones de la clase *c*.

Fisher-LDA busca encontrar el vector *w* de proyección que maximice el "cociente" entre la matriz de dispersión inter-clase y la matriz de dispersión intra-clase.

Operando se puede ver que el *w* que maximiza la función objetivo debe cumplir:

$$S_B w = \lambda S_W w \tag{3.11}$$

Si $S_W$ es no singular podemos resolver el clásico problema de valores propios para la matriz $S_W^{-1} S_B$:

$$S_W^{-1} S_B w = \lambda w \tag{3.12}$$

Si ahora sustituimos la solución en (3.8) obtenemos lo siguiente:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} = \lambda_k \frac{w_k^T S_B w_k}{w_k^T S_W w_k} = \lambda_k \text{ con } k = 1..d \tag{3.13}$$

Siendo $w_k$ vector propio k de valor propio $\lambda_k$.

En consecuencia, para maximizar la solución debemos considerar el vector propio con mayor valor propio asociado.

Claro está que este desarrollo vale para el caso en que queremos proyectar los datos sobre un espacio unidimensional. Se puede ver sin mayor esfuerzo [16] que para el caso de querer proyectar sobre un espacio m-dimensional, se debe resolver el mismo problema y elegir los m vectores propios con valores propios asociados más grandes.

En nuestro caso particular, donde se trabajó con imágenes (datos de alta dimensión) se aplicó una primera etapa de PCA para reducir la dimensionalidad de los datos. Los datos fueron reducidos a dimensión 100. Cabe recordar que existen formas

directas de aplicar LDA (D-LDA) que no fueron objeto de estudio en este proyecto. Al igual que en los casos anteriores, para la clasificación se utilizó el algoritmo *k-nn*.

### 3.5.5.2. Método basado en redes neuronales

El método basado en redes neuronales tiene la ventaja de poderse construir fácilmente un sistema de redes neuronales como clasificador. Se entrena el sistema con muestras de cara y de no cara. El sistema puede aprender automáticamente las complejidades de densidad de estos dos tipos de muestras.

Por ejemplo:

Rotar la imagen en diferentes ángulos para aumentar la exactitud de detección de la cara en la imagen cuando la cara está rotada, pero así se disminuirá la velocidad de detección. Para esto, Rowley añade un nivel a la red neuronal para detectar el ángulo de rotación de cara. Después de rotar la imagen original en la dirección contraria, se envía al detector de caras [17], como se muestra en la Figura 3.8. El clasificador de cara frontal utiliza dos redes neuronales de múltiples capas para estudiar los modelos (como valor de escala de grises, espacio entre píxeles) de las muestras de cara y de no cara. Su detector de cara frontal está formado por dos partes: una serie de redes neuronales forman un detector frontal y hay un detector para la toma de decisiones.

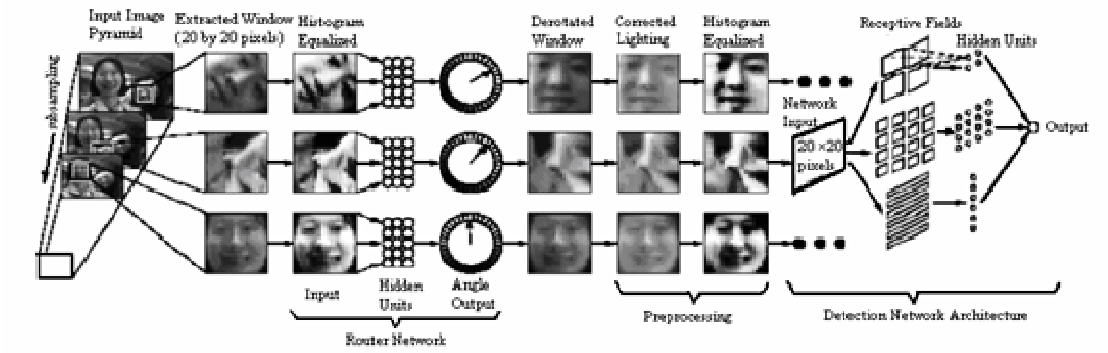


Figura 3.8 Descripción del algoritmo [18]. (Cortesía de H. Rowley)

Para detectar la cara rotada se utiliza otra red neuronal. Se verifica cada imagen que entra en la subventana, y se rotará la ventana con el ángulo de rotación de la cara. A continuación, se envía esta ventana al detector de cara frontal para la detección. Si se utiliza este método se será 17,6 veces más rápido [17].

### 3.5.5.3. Modelo oculto de Markov

HMM (Hidden Markov Model, HMM) es un mecanismo donde coexisten dos procesos aleatorios. Uno es la cadena de Markov con estados finitos, y la segunda es la secuencia de observaciones. Ya que no se puede obtener directamente el estado de la cadena de Markov -sólo se puede a través de la observación del valor- éste se ha convertido en la cadena oculta de Markov.

### 3.5.5.4. Método Boosting

Boosting es un método de aprendizaje estadístico que combina los clasificadores débiles para conseguir un clasificador fuerte. La idea básica es aumentar el peso de las muestras de aprendizaje erróneo. A continuación el algoritmo se centra más en el aprendizaje de las muestras difíciles. Al final el algoritmo seleccionará una serie de clasificadores débiles para formar un clasificador fuerte.

El método conocido como **Adaboost** y **Cascada** es muy popular y tal vez el más importante históricamente. Fue presentado originalmente por Viola y Jones [19] y es una extensión de un clasificador genérico al problema de la detección de objetos en imágenes. Éste método demuestra como a partir de características locales basadas en el cambio de intensidad se podía desarrollar un detector de caras muy robusto. La idea básica es la siguiente: se determinan una serie de características basadas en las sumas y restas de los niveles de intensidad en la imagen. Para ello se utilizan filtros de **Haar** de un cierto tamaño y calculados para las posiciones concretas de la sub-imagen que se quiere clasificar. Dichas características son evaluadas por un clasificador débil para decidir si la sub-imagen corresponde a una cara (aceptada) o no (rechazada) tal y como muestra la Figura 3.9. Si el valor de la característica está por encima de un cierto umbral θ, entonces la ventana se clasificará como cara. Este tipo de clasificadores débiles suelen conseguir unos resultados muy pobres. No obstante, combinando varios módulos como el de la Figura 3.8, se pueden generar clasificadores más robustos (clasificador fuerte) cuya tasa de detección crece exponencialmente. El rectángulo punteado de la Figura 3.10 corresponde a un clasificador fuerte.

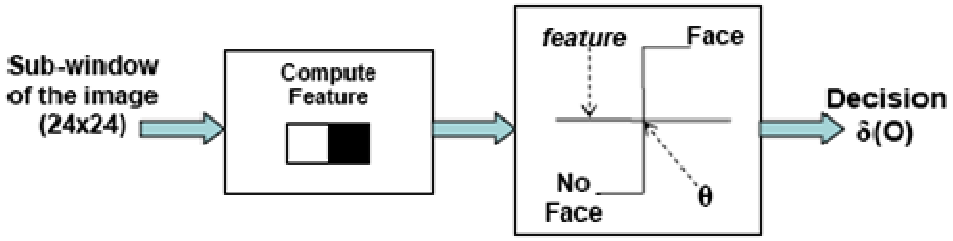


Figura 3.9 Adaboost: clasificador débil

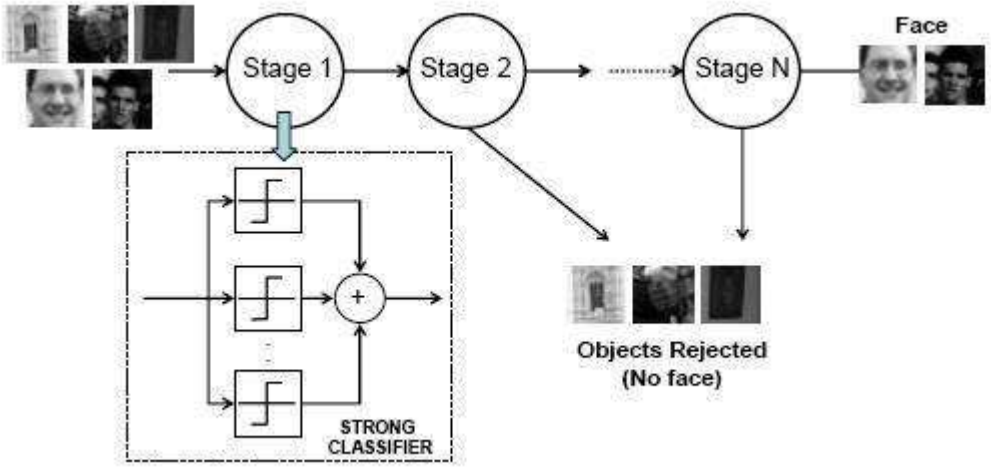


Figura 3.10 Detector de cara Adaboost

De todas maneras, a pesar de que la tasa de detección de caras de un clasificador fuerte puede llegar hasta el 99%, presenta la desventaja de que la tasa de falsas detecciones (aceptar una sub-imagen como cara cuando no lo es) es todavía inaceptable al estar por encima del 30%. Por este motivo, Viola y Jones propusieron un esquema basado en una cascada de clasificadores fuertes como el representado en la Figura 3.9. Cada etapa corresponde a un clasificador fuerte y está entrenada con todos los ejemplos que la etapa anterior no ha podido clasificar correctamente más algunos nuevos. Por tanto, en la etapa de entrenamiento, cada etapa se entrena con un conjunto óptimo de características capaces de detectar cada vez ejemplos más complicados; es decir, las primeras etapas se encargan de descartar sub-imágenes que son muy diferentes de una cara, mientras que las últimas etapas pueden rechazar ejemplos mucho más complicados como pueden ser pelotas, globos, dibujos, etc.

**Características Haar**

El proceso que se va a realizar para el reconocimiento de caras se puede dividir en dos fases: una primera de entrenamiento y una segunda de detección. En la primera fase se caracteriza y afina el sistema de decisión mediante el uso de

cientos de ejemplos de imágenes que se ajustan al objeto buscado (caras en nuestro caso) escalados todos al mismo tamaño. Éstos son los llamados ejemplos positivos. A su vez también exponemos el sistema a ejemplos negativos (también en la misma escala). De esta forma se consigue componer un modelo de las características que debe presentar el objeto buscado. Las características se describen como una serie de plantillas formadas por rectángulos blancos y negros, como los de la Fig. 3.11, así como la escala y la posición que ocupan dentro de la ventana de búsqueda. Estas plantillas se aplican sobre la ventana de búsqueda, sumándose los píxeles de la parte blanca, por un lado, y los de la parte en negro de la plantilla por otro. El valor obtenido es la resta de la suma de los píxeles de ambas regiones, y tras aplicar este proceso a todos los ejemplos ya sean positivos como negativos es el que se va a utilizar para decidir si esta característica es válida para discernir entre la imagen de una cara y otra que no lo es. Cada una de estas características consideradas válidas no son por sí solas capaces de detectar una cara, razón por lo que se les llama clasificadores "débiles", pero pueden suponer el indicio de su existencia. Por ejemplo, una determinada característica puede detectar la diferencia de intensidad entre la zona de los ojos y la parte de las cejas tal y como se muestra en la Fig. 3.12. Para crear clasificadores más robustos se realiza la unión o suma de clasificadores débiles que son ponderados con sus respectivos pesos, obtenidos también gracias al entrenamiento de cientos de ejemplos.

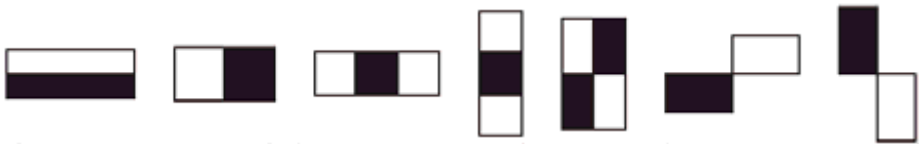

Figura 3.11 Ejemplos de plantillas de características de Haar

Figura 3.12 Característica aplicada a una imagen

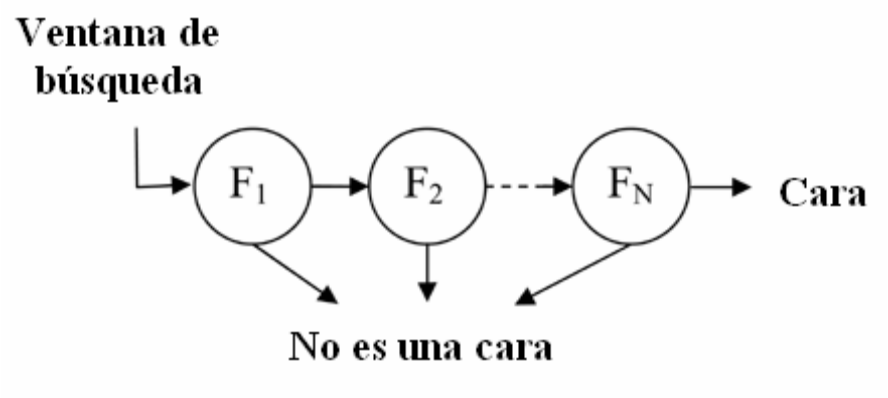


Figura 3.13 Esquema del clasificador en “cascada”

Se dice que este algoritmo utiliza un sistema de decisión en “cascada” porque usa un conjunto de fases de decisión fuerte colocadas una detrás de otra de menor a mayor complejidad. De esta forma, cuando una región es descartada en una de las etapas, por no presentar las características típicas de una cara, ya no es conmutada por el resto de las etapas con el ahorro de cómputo y el aumento de velocidad en el algoritmo que ello supone. El esquema de este sistema se encuentra en la Fig. 3.13, donde cada Fi es una etapa de decisión.

OpenCV emplea 38 etapas de decisión en su clasificador. Los datos provenientes del entrenamiento se guardan en archivos XML, y para el caso de las caras ya existe un entrenamiento predeterminado establecido.

## 3.6. Detección de ojos en el área de visión por computadora

Se han hecho muchos estudios sobre la detección y seguimiento de los ojos. Los estudios sobre los ojos se pueden clasificar en dos categorías principales:

- Detección de los ojos
- Extracción de características de los ojos.

**Detección de los ojos**: Dada una imagen de cara arbitraria, la meta de la detección de los ojos es determinar la ubicación de los ojos. Simplemente, en la detección de los ojos, en la zona donde se hallan ambos ojos, se deben encontrar los dos ojos individualmente. Como resultado del proceso, habitualmente el área donde se encuentra el ojo se indica mediante un rectángulo.

**Extracción de características de los ojos**: Por otra parte, el objetivo de esta

categoría es dar información detallada, como por ejemplo el contorno de la región visible del globo ocular, la zona circular formada por el iris y la pupila, la ubicación de la pupila en la zona visible del ojo, el estado de los ojos (por ejemplo, si parpadean o no parpadean). Este tipo de trabajo es más difícil en el área de visión por computadora. La detección en tiempo real del seguimiento de los pequeños detalles varía mucho dependiendo de las condiciones ambientales y el resultado puede fácilmente fallar.

A pesar de que se han dado muchos detalles acerca de las técnicas de medición de los movimientos oculares, el trabajo existente para conocer con detalle los ojos y sus características se puede clasificar en dos categorías: enfoques basados en infrarrojos y enfoques basados en imagen tradicional.

### 3.6.1. Enfoques basados en infrarrojos

Este enfoque se basa en las propiedades espectrales de las pupilas. El seguimiento de los ojos se realiza mediante un seguimiento de la luz de las pupilas vista a través de infrarrojos. Se ha trabajado mucho en esta técnica y hay algunos sistemas comerciales eye-tracking tales como los sistemas producidos por Iscan Incorporated, LC y Tecnologías Laboratorios de Ciencias Aplicadas (ASL). La detección del ojo basado en infrarrojos está fuera del alcance de este estudio.

### 3.6.2. Enfoques basados en imagen

Kothari y Mitchell [20] usan información espacial y temporal para detectar la ubicación de los ojos. Su proceso se inicia mediante la selección de un grupo de candidatos utilizando campos gradientes. El gradiente en el iris indica siempre el centro (pupila oscura). El centro del iris puede ser calculado mediante la selección del valor más alto. Reglas heurísticas y una gran cantidad de tiempo son necesarios para filtrar candidatos erróneos. Estudios sobre la detección facial utilizando plantillas deformables son presentados en [21] y [22].

#### 3.6.2.1. Plantilla de ojo

Las plantillas deformables proponen una solución a las limitaciones de las plantillas clásicas, permitiendo la deformación de la geometría de la plantilla y garantizando una invariación relativa a las condiciones luminosas. Una plantilla deformable consiste en tres elementos básicos [23]:

- Modelo geométrico: Este modelo geométrico parametrizado define la geometría de la plantilla e introduce restricciones a la deformación de la geometría.
- Modelo de la imagen: Específica cómo una plantilla deformable de una geometría específica se relaciona con unos valores específicos de intensidad en una imagen dada.
- Algoritmo de "matching": Un algoritmo que usa los modelos geométricos y de la imagen para encontrar una plantilla para la imagen.

## 3.7. Adaptive Mean Shift Algorithm (CAMSHIFT)

Los algoritmos de visión por computadora destinados a formar parte de una interfaz de usuario deben ser rápidos y eficientes. Como el seguimiento de la cara es el primer paso en el problema de seguimiento de los ojos, el seguimiento de la cara debe ser en tiempo real y no debe absorver demasiado tiempo de cálculo. Es por eso que se ha escogido seguimiento por color de la piel. Los trabajos [24] y [25] estaban basados en seguimiento por color, pero consumían demasiados recursos ya que usaban correlación de colores, filtro de Kalman, predicción del contorno y otras herramientas.

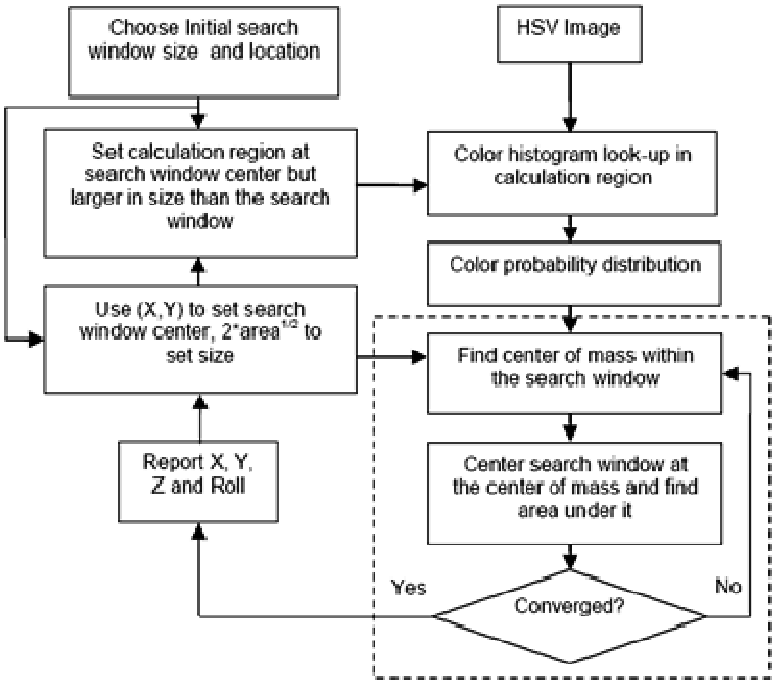


Figura 3.13 Diagrama de seguimiento del objeto color [4]

Se ha utilizado el algoritmo CAMSHIFT [28, 29] (Continously Adaptive Mean Shift) para realizar el seguimiento de una cara humana en una secuencia de imágenes. Este algoritmo es una adaptación del algoritmo Mean Shift [30] para poder tratar las distribuciones de probabilidad dinámicas (con cambios en su tamaño y su posición) que representan los objetos en movimiento. Todas las imágenes serán pasadas del modelo RGB al modelo HSV ya que se utilizará la componente de color (canal H del modelo HSV) para segmentar los objetos en el algoritmo CAMSHIFT. El diagrama de bloques del algoritmo se detalla en la figura 3.14.

El algoritmo empleado se puede resumir en los siguientes pasos:

1) El usuario fija una ventana de búsqueda inicial, seleccionando el objeto a seguir en la primera imagen.

2) Calcular el histograma de la componente de matiz sobre la ventana de búsqueda de la primera imagen. Se omiten los píxeles con valores bajos (S < 30) de saturación o luminancia (V < 10) ya que los valores de matiz correspondientes no son representativos. Los valores del histograma h(x) se escalarán (Ec. (3.14)) para que se encuentren en el rango [0, 255]. De este modo, el histograma representará la distribución de probabilidad del color del objeto a seguir; es decir, la distribución de probabilidad objetivo.

3) Para cada nueva imagen, calcular la retroproyección (*back-projection*) del histograma del paso 2. Esta operación consiste en generar una imagen en escala de grises donde cada píxel tendrá como intensidad el valor del histograma correspondiente al matiz de dicho pixel en la imagen procesada. Así, el valor de cada píxel de esta imagen identificará la probabilidad de que dicho píxel en la imagen procesada pertenezca al objeto.

$$h(x) = h(x)\frac{255}{\max_{x=0}^{255}\left(h(x)\right)} \qquad \forall x \in [0, 255] \tag{3.14}$$

4) Calcular el centroide de la imagen de retroproyección mediante el algoritmo Mean Shift. Se seguirán los siguientes pasos:

a. Calcular el centroide ( $x_c$,$y_c$ ) c c x y en la ventana de búsqueda actual, utilizando el momento de orden 0 ( $M_{00}$ ) y los momentos de orden 1 ( $M_{10}$ , $M_{01}$ ):

$$x_c = \frac{M_{20}}{M_{00}} = \frac{\sum_x \sum_y xI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.15}$$

$$y_c = \frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y yI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.16}$$

b. Establecer el centroide obtenido como nuevo centro de la ventana de búsqueda.

c. Repetir los pasos a y b hasta la convergencia. Es decir, hasta que el centro de la ventana se mueva menos de una cierta cota predeterminada o bien un número fijo máximo de iteraciones.

5) Establecer una nueva ventana de búsqueda para las siguientes imágenes; utilizando como centro el centroide obtenido en el paso 4 y como tamaño, una función del momento de orden 0 [4].

6) Repetir los pasos 3, 4 y 5 para las nuevas imágenes.

Figura 3.14 Una imagen de vídeo y la distribución de probabilidad de la piel

# 4. Diseño

En este capítulo, se presentará en detalle la aplicación del sistema de seguimiento del ojo llamado "EyeTRACK" que se ha realizado. Las principales capacidades del sistema son:

- Detección y seguimiento de la cara.
- Localización de ambos ojos por separado.
- Localización de la boca.
- Seguimiento de los movimientos oculares y medición de la posición del ojo y pupila.

También se ha implementado:

- Una representación de una escena 3D. El punto de vista de ésta puede ser modificado interactivamente acorde con la localización de los ojos. Este tipo de información puede llegar a ser muy útil en interfaces de usuario que se desarrollen en el futuro.

El sistema se basa en tres componentes:

- Componente de detección de la cara.
- Componente de detección de ojos y boca.
- Componente de seguimiento de los ojos y representación de una escena 3D.

El rendimiento de cada paso depende de los pasos anteriores, porque los resultados obtenidos en cada uno de estos componentes se pasan como entrada al siguiente componente.

Por ejemplo, el área de la cara y sus variables asociadas (ancho y alto de la cara) se obtienen durante la detección de la cara y se utilizan para definir las zonas de búsqueda y localización de los ojos. Las reglas basadas en las características geométricas de la cara y la detección de contornos se utilizan en el método de detección de los ojos y boca, además también se utiliza para la detección de la pupila. Para tener este sistema de seguimiento de ojos resulta que todos los componentes tienen que funcionar colectivamente y correctamente.

La estructura de sistema completo está mostrada en la Figura 4.1.

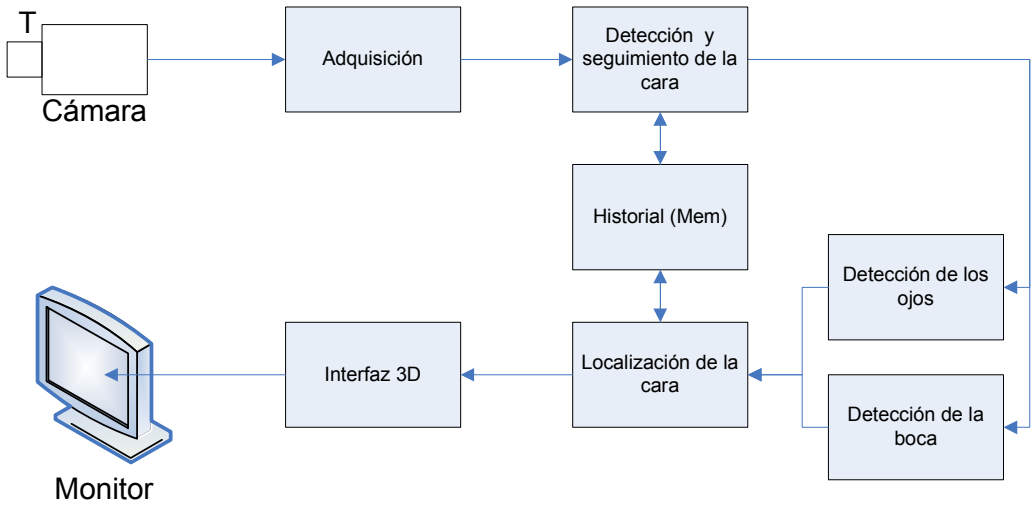


Figura 4.1 Esquema de módulos

## 4.1. Adquisición

Este componente se refiere a la cámara o fuente que ofrece la secuencia de video como entrada del sistema de seguimiento. En el sistema se deja a los usuarios elegir la fuente de entrada. Se permite utilizar una cámara que capture la imagen del usuario o un video creado que contenga las imágenes de la cara del usurario y esté guardado en un aparato de almacenamiento, como por ejemplo un fichero de tipo multimedia .AVI.

Para cumplir este requisito, en el sistema se ha implementado un módulo de elección de fuente de entrada en la interfaz gráfica.

## 4.2. Detección y seguimiento de la cara

El componente de seguimiento o tracking de la cara utiliza el algoritmo CAMSHIFT. Como el algoritmo CAMSHIFT es un algoritmo de seguimiento por color, para que funcione necesita una muestra de piel de la cara. Entonces el sistema también tiene que tener una función que sea capaz de coger la muestra de la piel. Por lo tanto, este módulo se divide en dos submódulos: uno que coja la muestra de piel y otro que realice el seguimiento con CAMSHIFT. Los pasos a seguir son:

1) Obtener la muestra de color de la cara. Generalmente el sistema pide que el usuario deje la cara en el centro o que la acerque a una zona de captura - por ejemplo, puede haber un cuadro en pantalla para indicar la zona; este cuadrado debe ser más pequeño que el área de la cara del usuario.

En este proyecto se ha querido mejorar este paso. Se ha utilizado el algoritmo de detección mediante características Haar y Adaboost. Así, cada vez que se empieza el proceso, el sistema automáticamente detecta la cara del usuario. La imagen de la cara que se ha obtenido será la muestra de color para analizarla en el siguiente proceso.

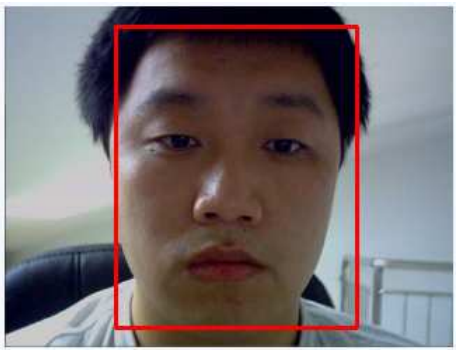

Figura 2. Resultados de la detección mediante Haar

2) CAMSHIFT (sección 3.7). Este punto está formado principalmente por los procesos de análisis de color, análisis del histograma, binarización de la imagen y seguimiento de la cara.

- Análisis de color:

  Consiste en convertir la imagen de la cara en color HUE (RBG a HSV) píxel por píxel, y clasificar los valores por color HUE. Crear el histograma de color HUE basado en estos datos obtenidos.

- Análisis del histograma:

  Analizar el histograma de color HUE. Encontrar un rango de color HUE de la piel del usuario para que el siguiente paso sea binarizar la imagen de la cara.

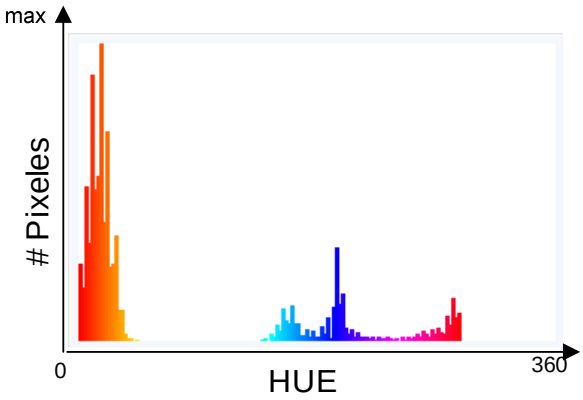


Figura 3. Histograma según HUE (eje horizontal)

- Binarización de la imagen y encontrar el área de la cara:

  El objetivo es obtener una imagen que sólo sea representada por dos tonos de color, en general blanco y negro. La idea para realizar este trabajo es sencilla: sólo se debe decidir qué tono de color dar a cada píxel que sea mayor de un determinado umbral (valor límite), el resto de píxeles tendrán por defecto el otro tono de color. Así se facilita la localización de los ojos.

Figura 4.2 Imagen binarizada

## 4.3. Detección de ojos y boca

Este módulo está formado por dos partes: la detección de los ojos y la detección de la boca. La detección de los ojos se basa en procesos de localización de ojos y pupila para obtener la posición de los ojos sobre la imagen con cierta exactitud.

- Localización de ojos:

  Localizar los ojos en una imagen en color binario a través de las siguientes reglas:

  1) Buscar los ojos por encima del 50% de la región y por debajo del 10% de la región de imagen.
  2) La distancia entre el centro del ojo izquierdo y el ojo derecho tiene que ser dentro de un cierto rango (según el ancho de la cara).
  3) El ángulo que forma la línea entre el centro del ojo izquierdo y el ojo derecho con el eje X tiene que estar dentro de un cierto rango.
  4) Para distinguir las cejas de los ojos, se calcula según la escala de grises (el contraste).

Figura 4.3 Imagen de muestra de detección de ojos

- Localización de las pupilas:

  La localización de las pupilas se hace mediante el umbral de color y detección de contornos. Se puede ver en la Figura 4.4.

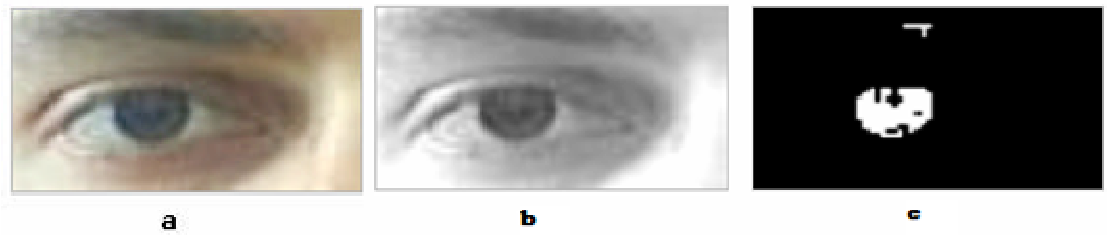


Figura 4.4 Preparación de la imagen del ojo para la detección de pupilas

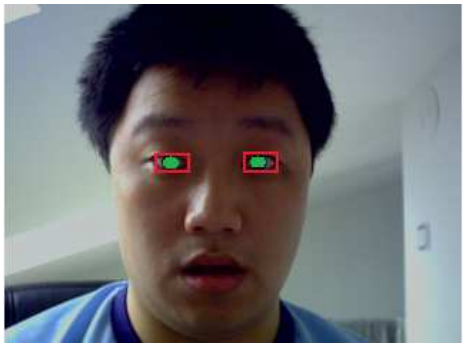

Figura 4.5 Imagen de muestra de detección de pupilas

- Localización de la boca:

Es un poco diferente que la localización de los ojos porque hay que analizar el color del labio en el espacio de color HSV, ya que el color HUE del labio es un poco más rojo que la piel de la cara. Cuando se tiene la imagen binarizada, ya se puede localizar la boca con métodos basados en reglas y detección de contornos. Esto es así porque en la imagen binarizada, el área de la boca será una zona conectada de color negro. Para localizar la boca en una imagen en color binario se buscará por encima del 90 o 100% de la región y por debajo del 50% de la región de la imagen.

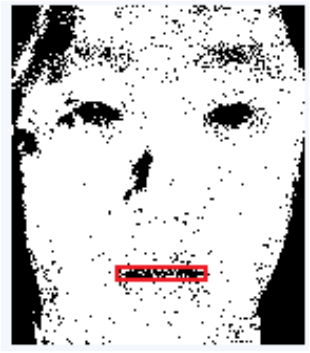

Figura 4.6 Imagen de muestra de detección de la boca

## 4.4. Localización de la cara

Una vez se han localizado la posición de los ojos y de la boca en coordenadas, este módulo se ocupa de hacer los cálculos de la posición de la cara humana en 3D. El movimiento de la cara se comprueba con sus coordenadas del frame anterior. El dato anterior se guarda en un módulo de memoria. Después de realizar el cálculo se sustituirá este módulo de memoria con el nuevo dato.

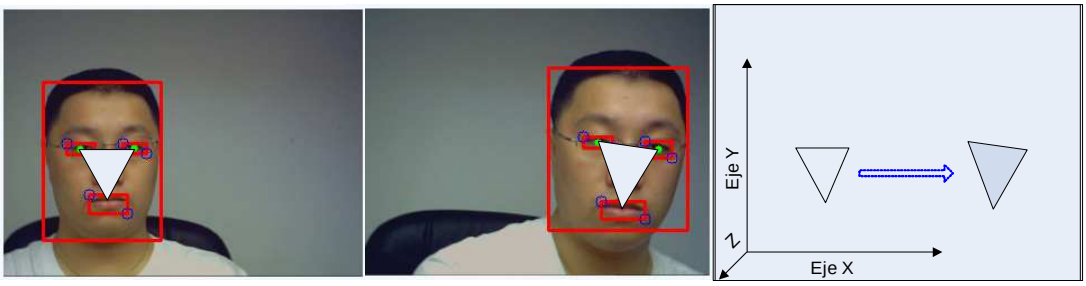


Figura 4.7 a) 1ª frame b) 2ª frame 3) movimiento horizontal de la cara

En la figura 4.7 podemos observar la diferencia entre dos frames. La cara se ha movido en los ejes X y Z (hacia a la derecha y se ha acercado a la cámara). Si se sustituyen las diferencias de las coordenadas de los ojos y de la boca de los dos frames en las fórmulas geométricas podremos calcular casi todos los movimientos en tres dimensiones con una cierta exactitud.

## 4.5. Interfaz 3D

En este modulo se ha implementado una escena 3D. Se pinta un castillo y se crea

una cámara que enfoque a este castillo. Toda la implementación está programada en C++ con OpenGL. Para que la escena se pueda interactuar con la posición de la cara del usuario, también se tiene que diseñar una interfaz que pueda recibir las instrucciones enviadas desde el módulo de seguimiento de la cara para rotar el modelo 3D del edificio. En este proyecto estas operaciones son mover y rotar la cámara de la escena.

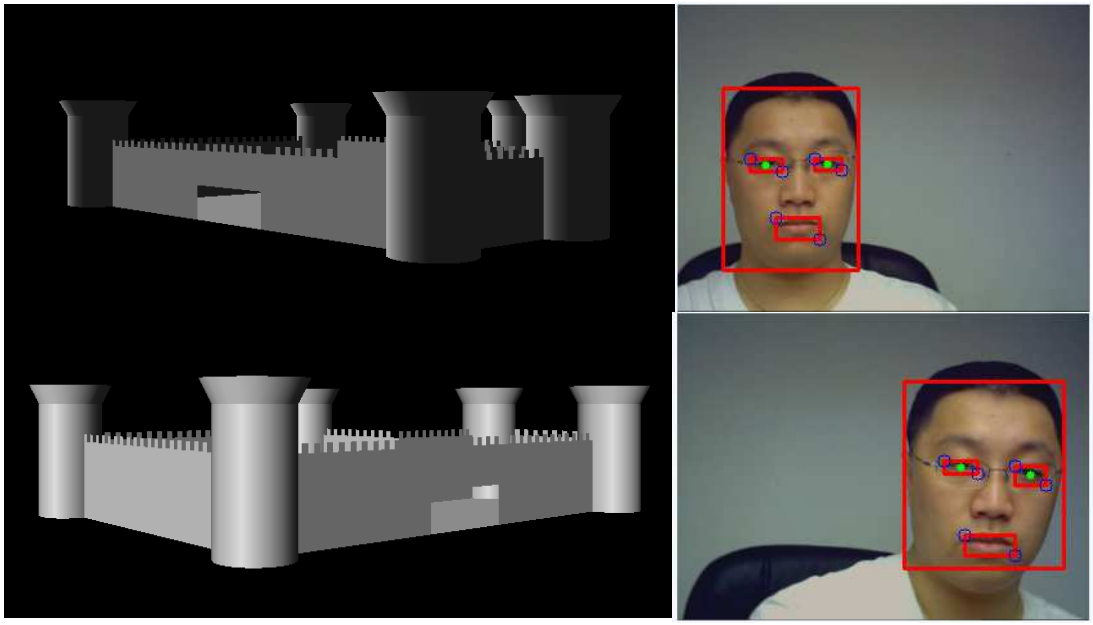

Figura 4.8 Representación de una escena 3D

Una vez realizados los pasos anteriores, las nuevas coordenadas de la cabeza del usuario se utilizan como entrada de la interfaz interactiva 3D y se modifica interactivamente el punto de vista de una escena 3D (Figura 4.8).

## 4.6. Esquema de flujo

Todos los flujos del sistema se pueden ver en la Figura 4.9. Los detalles más precisos pueden verse en la sección 5.1 – 5.3.

El primer paso en el sistema de seguimiento del ojo tiene que ser la localización de la cara porque los ojos sólo ocupan un área pequeña en comparación con la imagen y si se detectan los ojos sin localizar la cara primero, probablemente, se producirán muchas falsas alarmas. En este proyecto se ha implementado un algoritmo de detección de caras basado en características para este propósito.

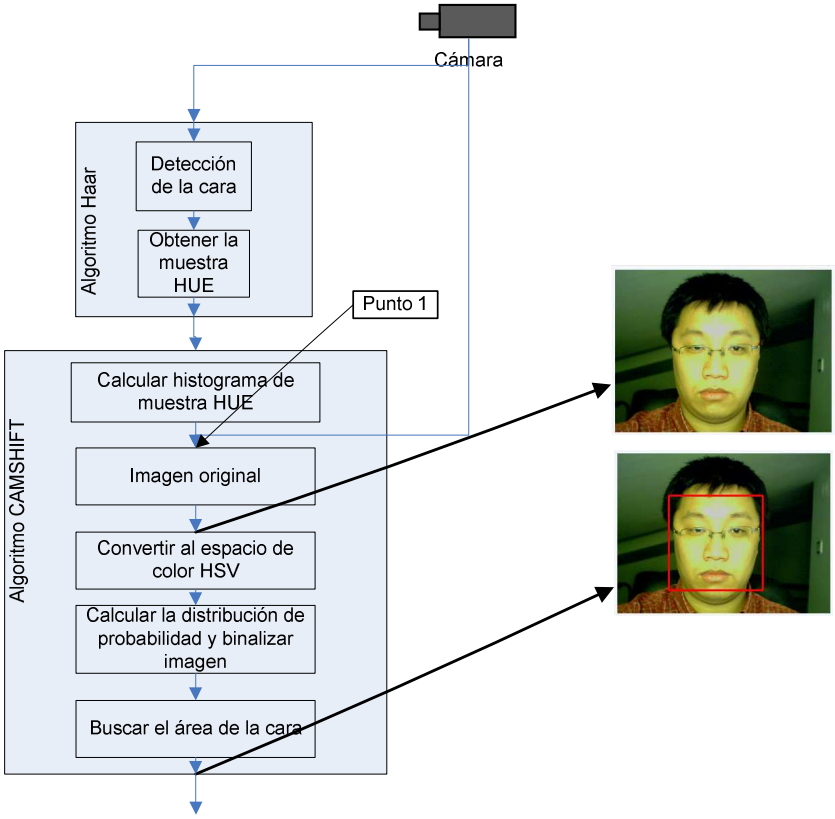


Figura 4.9 Diagrama de flujo de detección y seguimiento de la cara

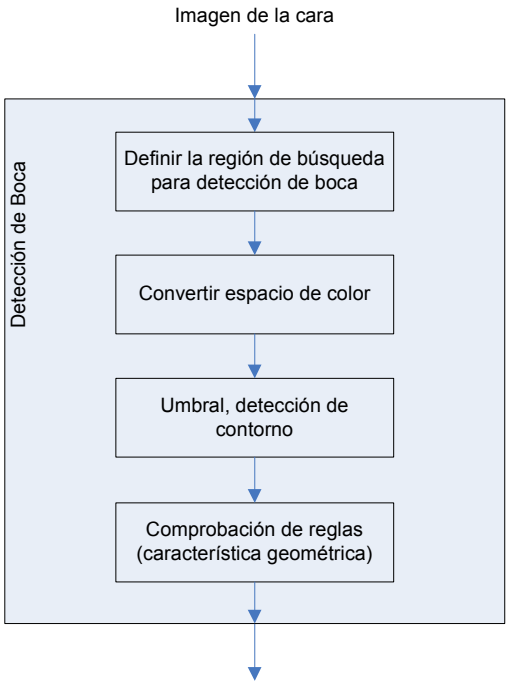


Figura 4.10 Diagrama de flujo de localización de la boca

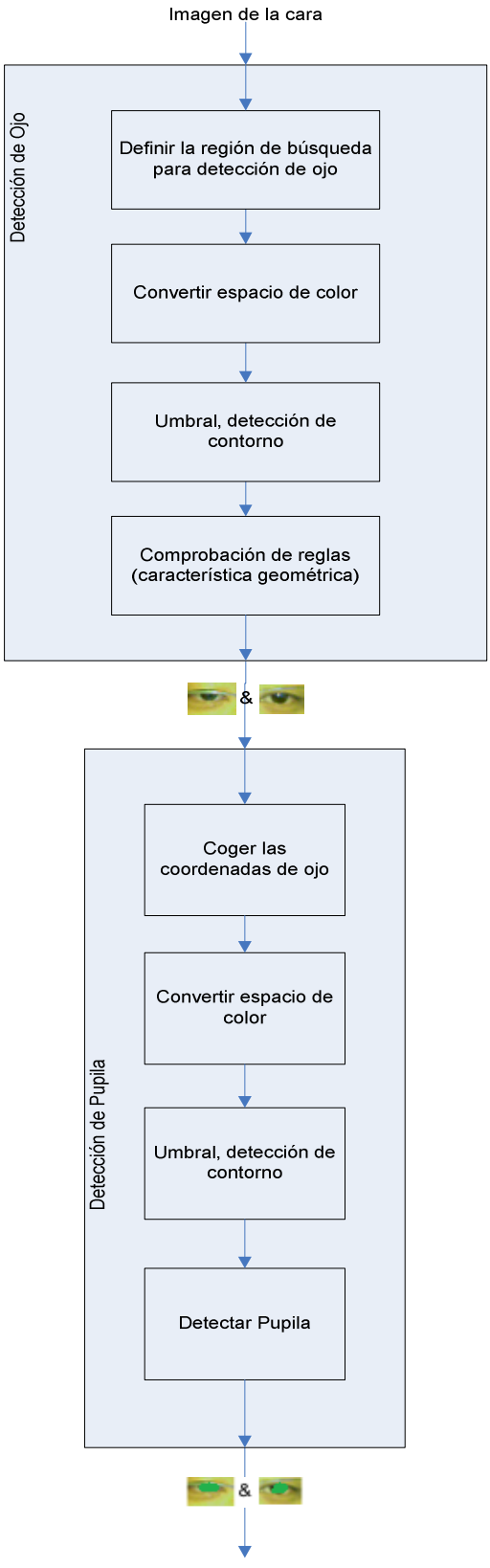


Figura 4.101 Diagrama de flujo de localización de los ojos y las pupilas

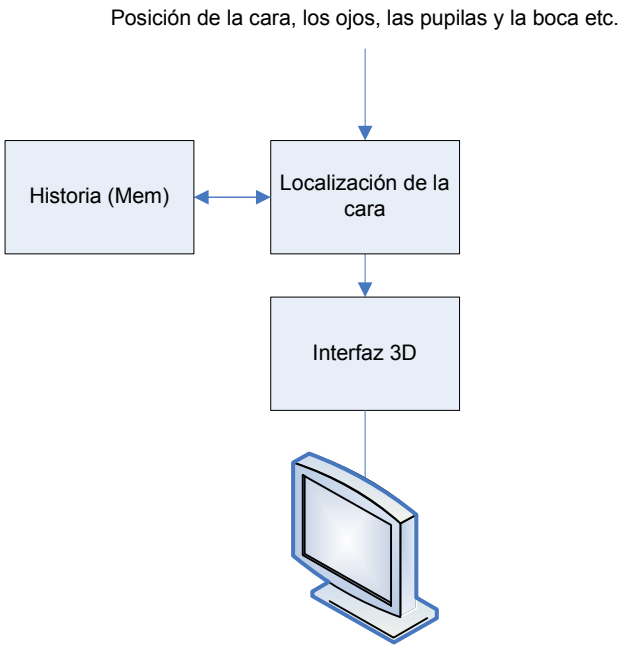


Figura 4.11 Diagrama de flujo de localización de la cara e Interfaz 3D

# 5. Implementación

## 5.1. Detección y seguimiento de la cara

El algoritmo de seguimiento de la cara implementado en este proyecto es un método basado en el color de la piel [4]. Esto fue discutido en detalle en la sección 3.7.

Como se señaló anteriormente, el algoritmo CAMSHIFT es un método robusto no paramétrico para encontrar el pico de la distribución de probabilidad con la ventana de búsqueda con tamaño adaptable. El algoritmo CAMSHIFT trata eficazmente con problemas de la imagen, movimiento de objetos irregulares, ruido de la imagen, distractores y oclusión facial [4]. Una desventaja del algoritmo CAMSHIFT es que se realiza el seguimiento utilizando el color de la muestra del objeto por lo que cualquier solapamiento con el objeto puede llevar el algoritmo a fracasar. Pero como nuestro objetivo es el seguimiento de los ojos, se supone que habrá una persona sentada delante de la cámara sin ningún distractor.

Los detalles de la implementación del algoritmo se detallan a continuación:

1) Para que funcione CAMSHIFT se debe conocer el color de la piel basado en un histograma de distribución de color HUE para hacer la comparación con la próxima imagen. Por lo tanto, al comienzo del proceso de seguimiento, el usuario sólo debe centrar su cara en la imagen. El sistema la detectará automáticamente mediante las características HAAR (ver sección 3.5.5.4 y Anexo 9.1.1). Si existe una cara humana dentro de este área, el sistema capturará la imagen de la cara para caracterizar el color de piel de su cara.

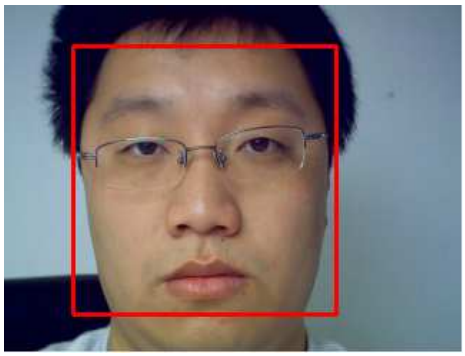

Figura 5.1 Ejemplo de definición de la distribución de probabilidad

2) Después de conseguir la imagen de muestra, se convierte dicha imagen de espacio de color RGB al espacio de color HSV. El HUE es el único componente interesante en CAMSHIFT. Por lo tanto, se extrae el componente HUE y se almacena como un histograma de dimensión 1. Éste

se utilizará más adelante en la construcción de la probabilidad de distribución.

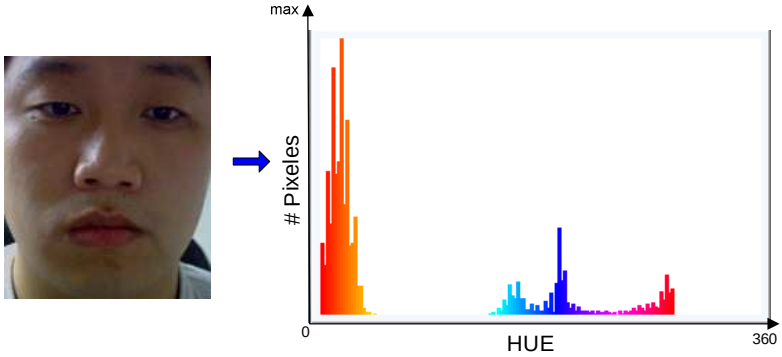


Figura 5.2 a) muestra de la región HUE, b) Histograma de la distribución de probabilidad del componente HUE (eje horizontal)

3) Se obtiene una nueva imagen a través de la cámara. Esta se convierte en el espacio de color HSV.

4) Se inicia el cálculo de la ventana de la imagen.

5) Se calcula la imagen de distribución de probabilidad y la de proyección trasera. Estas imágenes son una forma de indicar la distribución de probabilidad. Se realiza la comparación de cada pixel de la imagen HUE (Figura 5.3a) con el histograma de HUE que está construido en el paso 2.


Figura 5.3 a) Componente HUE de la imagen b) Distribución de probabilidad del componente HUE

6) Modo de la distribución de probabilidad y tamaño de la ventana de búsqueda que indica la zona de la cara a calcular.

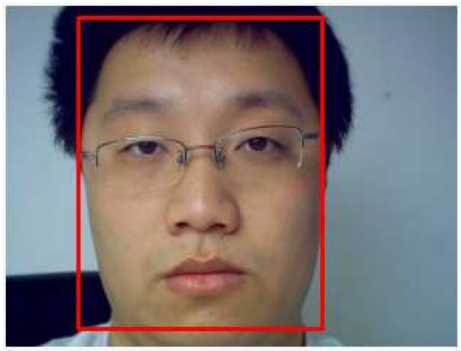

Figura 5.4 Resultado del módulo de seguimiento

7) La ventana calculada anteriormente sirve de guía para detectar la cara en el próximo frame para ahorrar tiempo de cálculo.

En este proyecto el seguimiento de la cara no sirve exclusivamente para hacer un seguimiento de la cara. También sirve como una fuente importante de información para el componente de detección de ojos basado en métodos de reglas y borde (contorno). Proporciona dos informaciones importantes:

- Localización del área de la cara.
- Centro de la imagen de la cara, por lo tanto podemos calcular la anchura de la cara.

Las informaciones de la cara detectada como son la anchura, la altura, el centro o el tamaño están calculados mediante la imagen de proyección trasera del momento de orden 0 ($M_{00}$) y la de los momentos de primer orden (ver sección 3.7). Las informaciones como la anchura de la cara localizada y las coordenadas del área de la cara juegan un importante rol en el resultado de detección de los ojos. Unos malos resultados pueden causar fallos en la detección de los ojos.

Aunque el algoritmo CAMSHIFT para el seguimiento de la cara es bastante robusto, se han observado algunos inconvenientes -tales como el efecto de la luz ambiental y la imagen de baja calidad- con el uso de cámaras.

Los resultados detallados utilizando diferentes condiciones, las ventajas e inconvenientes que se han observado durante los ensayos y los comentarios sobre el algoritmo CAMSHIFT se presentan en la sección 6.2.

## 5.2. Detección de ojos y boca

Los algoritmos de detección de boca y ojos implementados en este proyecto son

algoritmos que funcionan mediante reglas basadas en los conocimientos geométricos de la boca y los ojos de la cara humana. Ésto está explicado con más detalle en la sección 3.5.1. El método de detección con reglas es fácil de entender e implementar y su velocidad de procesamiento es más rápida comparada con otros métodos. También tiene sus inconvenientes, como que éste depende mucho del resultado del proceso anterior, para que en esta etapa se obtengan buenos resultados. Estos factores dependen de las condiciones del lugar de donde se tomen las capturas, influyendo la luz ambiental y otros factores. Hay muchas situaciones difíciles de tratar, por lo tanto el algoritmo no puede definir unas reglas generales para todos los casos porque se reduciría la fiabilidad.

Para explicar el proceso de detección tanto de la boca como de los ojos, se supondrá que no se han producido errores en el proceso anterior y que la imagen de entrada tendrá una calidad lo suficientemente buena y que estará en una condición ideal con un brillo más o menos uniforme.

### 5.2.1. Detección de boca

Los detalles de la implementación del algoritmo se detallan a continuación:

Antes de empezar a detectar la boca, se tiene que definir la zona de búsqueda de la imagen con los datos del proceso anterior, como la coordenada del centro de la cara, anchura de la cara, altura de la cara, etc.

Figura 5.5 Área de la cara

Después de conseguir la imagen para la búsqueda, hay que convertir la imagen de espacio de color RGB al espacio GRAY (escala de grises). A continuación, hacer el umbral con dicha imagen para filtrar los píxeles de color gris bajo.

Figura 5.6 a) Imagen de distribución de color b) Imagen binarizada con umbral

Para encontrar la boca hay que buscar en la zona inferir a partir de la mitad de la imagen.

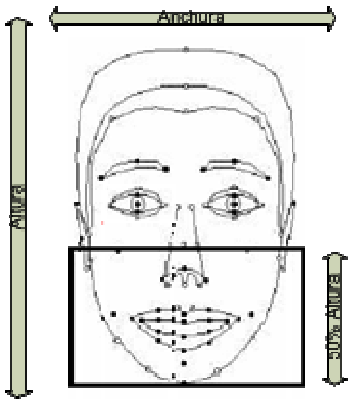


Figura 5.7 El área de búsqueda de la boca

Se buscan todos los contornos conectados en color negro sobre el fondo blanco. Estos contornos indican que no son del color de piel de la cara.

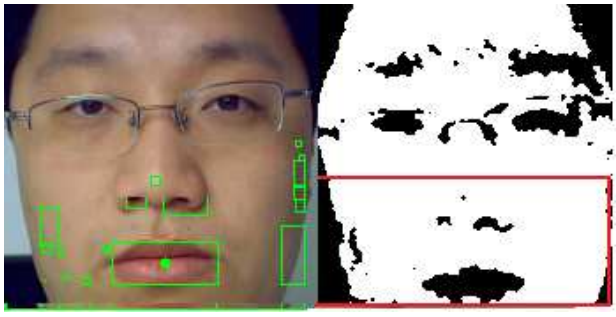

Figura 5.8 a) Imagen marcada de todos los candidatos de boca b) Imagen binarizada

Hay que elegir un único resultado de entre los candidatos que se han encontrado. Se hace en base a las siguientes reglas:

- El área del contorno de la boca es más grande que el del resto.

- La relación entre la anchura y la altura (anchura / altura) tiene que ser > 1.

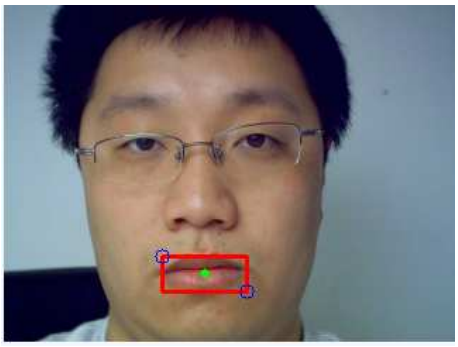

Figura 5.9 Resultado de la localización de boca

### 5.2.2. Detección de ojos

La detección de ojos se trata en dos pasos: uno es la detección aproximada de los ojos, y el otro es, tras detectar los ojos, buscar en el área de cada ojo para encontrar la pupila y así mejorar la exactitud del resultado. Una de las funciones principales del seguimiento de ojos es saber si la cara se ha movido en el eje Z. Esto quiere decir que el usuario se ha acercado a la cámara o se ha alejado. Para saber esto se necesita saber la distancia entre los dos ojos.

#### 5.2.2.1. Detección aproximada

Los detalles de la implementación del algoritmo se detallan a continuación:

Antes de empezar a detectar los ojos, hay que definir la zona de búsqueda de la imagen con los datos del proceso anterior, como la coordenada del centro de la cara, anchura de la cara, altura de la cara, ángulo de giro en el eje vertical, etc.

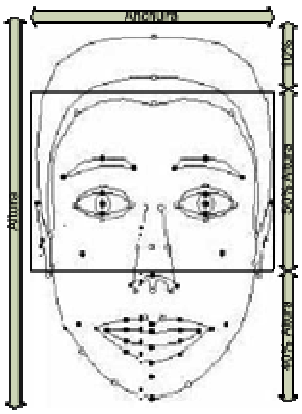


Figura 5.10 Área de la cara

Para encontrar los ojos, se tienen que buscar en el área superior de la mitad de la imagen. El área ideal está marcada en la Figura 5.10

Se buscan todos los contornos conectados en color negro sobre el fondo blanco. Estos contornos indican que no son piel de la cara.

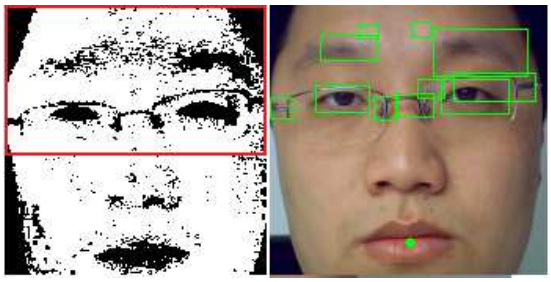

Figura 5.11 a) Imagen binarizada b) Imagen marcada con todos los candidatos de ojos

Se elige el resultado más adecuado entre los candidatos que se han encontrado. Se comprueba con las siguientes reglas:

1) Separar todos los candidatos en dos listas. Una es la lista de ojos izquierdos, la otra es la lista de ojos derechos.

   La anchura del área del ojo izquierdo es entre el 0% y el 50% de la anchura de la cara y la altura del área del ojo izquierdo es entre el 10% y el 60% de la altura de la cara. Todos los rectángulos encontrados, si su punto central pertenece a esta área, serán los ojos izquierdos.

   La anchura del área del ojo derecho es entre el 51% y el 100% de la anchura de la cara y la altura del área del ojo derecho es entre el 10% y el 60% de la altura de la cara. Todos los rectángulos encontrados, si su punto central pertenece a esta área, serán los ojos derechos. Ver la Figura 5.12.

   *ojoIzquierdo*$X_{min}$ = 0

   *ojoIzquierdo*$X_{max}$ = *anchura de la cara* x 50%

   *ojoIzquierdo*$Y_{min}$ = *altura de la cara x 10%*

   *ojoIzquierdo*$Y_{max}$ = *altura de la cara* x 60%

   *ojoDerecho*$X_{min}$ = *anchura de la cara* x 51%

   *ojoDerecho*$X_{max}$ = *anchura de la cara* x 100%

   *ojoDerecho*$Y_{min}$ = *altura de la cara* x 10%

   *ojoDerecho*$Y_{max}$ = *altura de la cara* x 60%

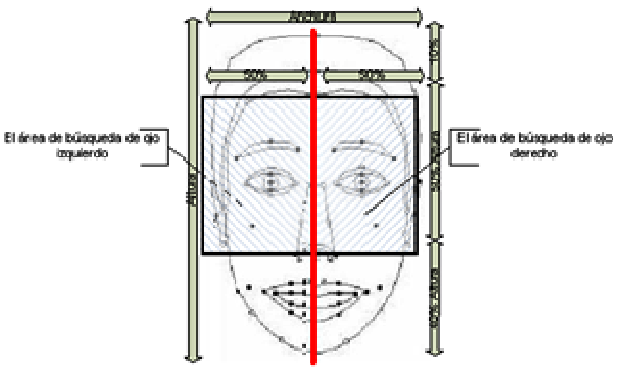


Figura 5.12 Las áreas de búsqueda para el ojo izquierdo y derecho

2) La relación entre la anchura de la cara y la distancia entre el centro del ojo izquierdo y el del ojo derecho (distancia / anchura de la cara) tiene que ser entre 26% y 60%.

   Para obtener las coordenadas del centro del ojo sobre de la imagen se ha utilizado la fórmula (5.1).

$$X = rect.x + rect.anchura / 2.0$$
$$Y = rect.y + rect.altura / 2.0 \quad (5.1)$$

   *rect* es el rectángulo que marca el área del ojo que se ha encontrado sobre la imagen binarizada.

3) El ángulo que forma la línea entre el centro del ojo izquierdo y el del ojo derecho con el eje X tiene que estar entre 10 grados y -10 grados.

4) La escala de la anchura con la altura debe ser mayor o igual que 1.

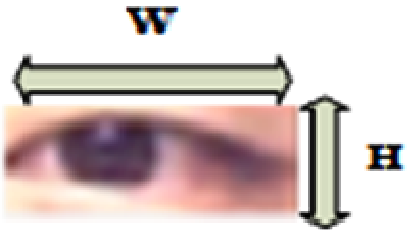


Figura 5.13 La relación entre la anchura y la altura del ojo

5) Después de hacer las comprobaciones anteriores ya se pueden seleccionar las parejas de ojos y cejas. Para la selección se pueden encontrar dos situaciones: a) se han encontrado dos parejas, una de ojos y otra de cejas, b) se han encontrado una pareja de ojos junto con cejas.

El ojo humano tiene generalmente una zona oscura y también una zona clara. Entre estas dos zonas hay un cambio radical de nivel de gris. Mediante esta característica, se puede utilizar el cálculo del contraste de escala grises para localizar los ojos.

Suponiendo que los contornos que se han encontrado por algoritmos tienen un total de m filas y n columnas, entonces el contraste de la imagen del contorno COM(k) se puede calcular mediante la fórmula (5.2) definida:

$$Com(k) = \sum_{i=1}^{n} \sum_{j=1}^{m-1} \left| B_{i,j+1} - B_{i,j} \right| \cdot \min(j, m-j) \tag{5.2}$$

B(i,j) es el valor de gris del píxel fila i y columna j.

Después de calcular el contraste de escala grises de cada contorno, se cogen los dos contornos que tienen máximo valor.

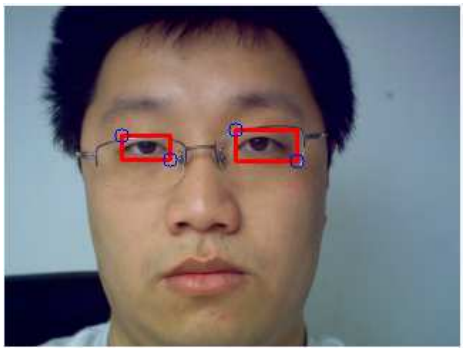

Figura 5.14 Resultado de la detección de ojos sin pupilas

### 5.2.2.2. Detección precisa (Pupilas)

El algoritmo de detección de pupilas se puede dividir en 3 pasos:

1) Conversión del espacio de color

Para facilitar la detección del contorno, hay que hacer la conversión de la imagen del espacio de color RGB a escala de grises. En este proyecto se han estudiado diferentes espacios de color y se ha llegado a la conclusión de que el mejor de ellos para esta situación es el espacio de color XYZ, siendo útil para el proyecto la componente Z.

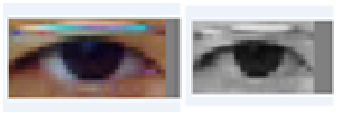

Figura 5.15 a) Área del ojo original b) Componente Z del espacio de color XYZ

2) Umbral

Entre los valores de escala de grises de la pupila y la esclerótica hay una gran diferencia. Utilizar el algoritmo umbral puede eliminar las zonas que no son pupilas.

Figura 5.16 a) Ojo de espacio de color convertido b) Imagen binaria con valor umbral 100

3) Detección del contorno

Como se muestra en la Figura 5.15 b, en la imagen del área del ojo hay un contorno cuya área es mayor que el resto. Ésta será la pupila del ojo.

Figura 5.17 Resultado de detección de la pupila

## 5.3. Localización de la cara e Interfaz 3D

La función de este módulo es, una vez detectada la cara, extraer la posición que ésta ocupa dentro de la imagen. Con esto datos se lleva a cabo una rotación del modelo 3D del edificio en cuestión para que se ajuste a la nueva posición de la cara del usuario.

### 5.3.1. Localización de la cara

En el módulo de seguimiento de la cara hay que detectar el movimiento de la cara del usuario en los ejes X, Y y Z mediante los cálculos de la diferencia de la posición de la cara.

**Movimiento en los ejes X y Y:**

Para la detección del movimiento de la cara del usuario sobre los ejes X y Y se mide la posición del punto central de la cara y se compara con la misma coordenada del frame anterior.

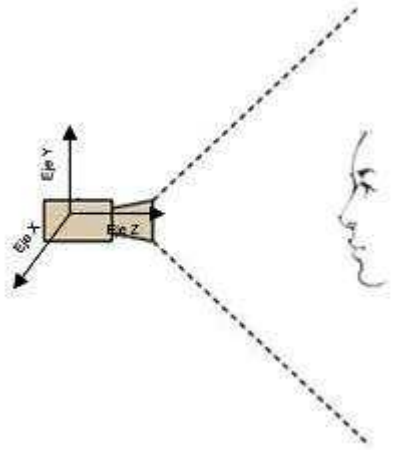


Figura 5.18 Vista lateral

Para el cálculo del movimiento sobre los ejes X y Y, se realizan los siguientes cálculos:

*movimientoCara_eje*X = *puntoCentralAnterior*X – *puntCentralActual*X (5.3)

*movimientoCara_eje*Y = *puntoCentralAnterior*Y – *puntCentralActual*Y (5.4)

Si el resultado de la operación (5.3) es de valor negativo significa que el movimiento de la cara ha sido hacia la izquierda. En caso contrario, se ha desplazado a la derecha.

Si el resultado de la operación (5.4) es de valor negativo significa que el movimiento de la cara ha sido hacia arriba. En caso contrario, se ha desplazado hacia abajo.

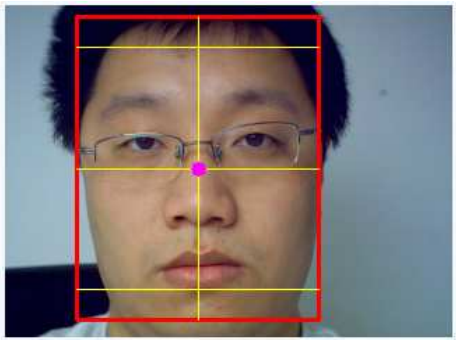

Figura 5.19 Vista frontal

**Movimiento en el eje Z:**

Para conocer el desplazamiento sobre el eje Z se debe calcular la distancia entre los dos ojos (centro de la pupila). Si ésta disminuye significa que el usuario se ha alejado y si ésta aumenta significa que el usuario se ha acercado.

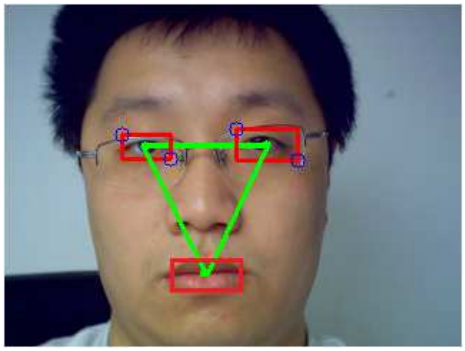

Figura 5.20 Vista frontal

*movimientoCara_ejeZ=distanciaEntreOjosAnterior–distanciaEntreOjosActual* (5.5)

Si el resultado de la operación (5.5) es de valor negativo significa que el movimiento de la cara ha sido hacia atrás. En caso contrario, se ha desplazado hacia delante.

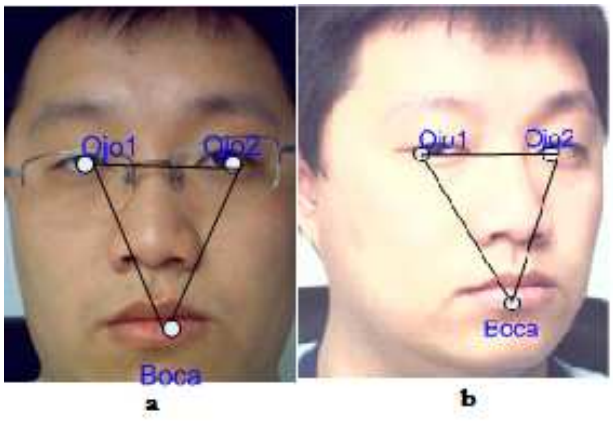


Figura 5.21 Vista frontal a) Vista frontal con la cara girada hacia la derecha

La distancia entre el ojo izquierdo y el derecho no sólo cambia en el caso de que el usuario mueva la cara en profundidad (eje Z), sinó también en el caso de girar la cara en el eje Y (izquierda o derecha), ver la Figura 5.17. En este caso se tiene que saber el ángulo de giro de la cara para calcular la distancia real entre los dos ojos. Los datos estadísticos sobre la cara humana dicen que la distancia entre los dos ojos es de entre 58 y 64 mm y la altura entre los ojos y la boca es de entre 65 y 80 mm para los adultos.

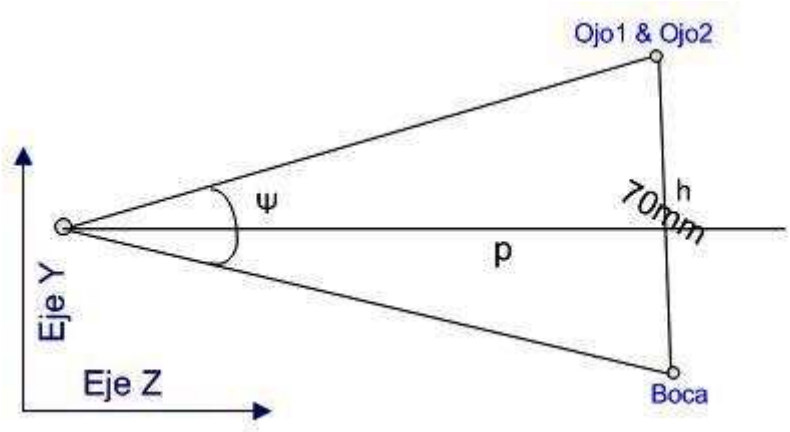


Figura 5.22Vista lateral

Figura 5.18, *h* es la altura entre los ojos y la boca, *Ψ* es el ángulo vertical de la cámara formado por la altura *h* y la distancia *p* entre la cámara y la cara del usuario. El ángulo del campo de visión de la cámara varía según el modelo. La cámara que se ha utilizado en este proyecto tiene unos 43.5 grados de ángulo de campo de visión vertical y unos 58 grados de horizontal (es una escala típica 4:3). La resolución de la cámara es de 640x480. Para mejorar la velocidad de procesamiento se ha bajado la resolución a 320x240.

Con la altura, el ángulo vertical del campo de visión y la altura entre los ojos y la boca en píxeles ya podemos calcular el ángulo *Ψ (5.6).*

$$\psi = \gamma * \frac{h}{H} \tag{5.6}$$

$\gamma$: El ángulo vertical del campo de visión de la cámara, igual a 43.5 grados.

*H*: La altura del campo de visión de la cámara, igual a 240 pixeles.

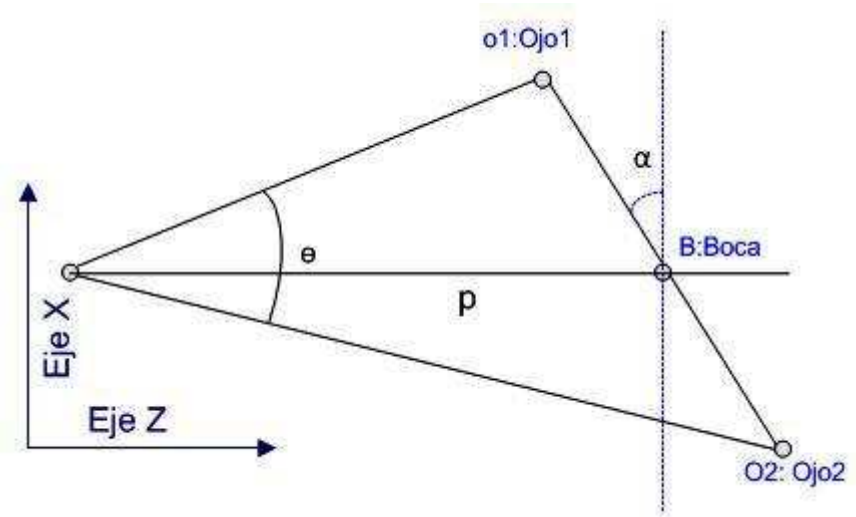


Figura 5.23 Vista vertical

Para calcular el ángulo horizontal *Θ* se puede utilizar la fórmula (5.7).

$$\theta = \varphi * \frac{l}{W} \tag{5.7}$$

$\varphi$: El ángulo horizontal del campo de visión de la cámara, igual a 58 grados.

*W*: La anchura del campo de visión de la cámara, igual a 320 pixeles.

El valor de $\gamma$, $\varphi$, H y W depende de cada cámara y sus configuraciones.

Normalmente las cámaras tienen 53-58 grados de ángulo de visión horizontal y un [4:3](320x240-640x480) de escala de la anchura con la altura.

Una vez calculados los ángulos *Θ* y *Ψ,* se puede calcular el ángulo de giro de la cara α mediante la fórmula (5.8) con la escala de la altura y la anchura entre boca y ojos.

$$\partial \cong artg\left(\frac{\Psi - \theta\frac{70}{60}}{\theta}\right) \tag{5.8}$$

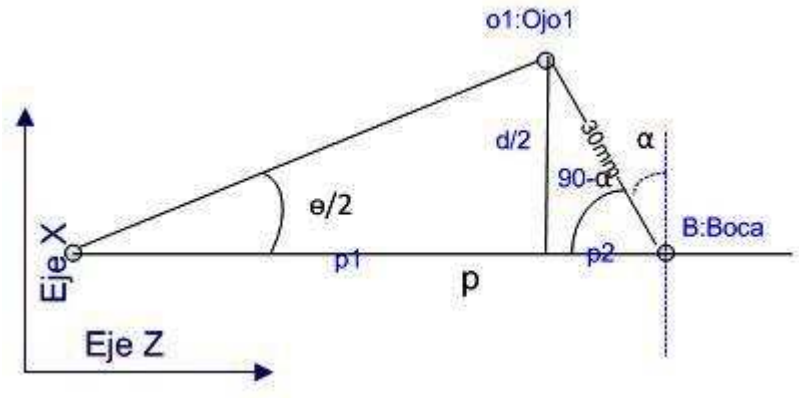


Figura 5.24Vista lateral

Después de calcular el ángulo α, se tiene que calcular la profundidad *P* entre la cámara y la cara. *P* es suma de *p1* y *p2, ver la Figura 5.20*.

P1 = (d/2) / tan (Θ/2);

P2 = (d/2) / tan (90- α);

P = p1 + p2;

## 5.3.2. Interfaz 3D

En este módulo se tiene que implementar una representación de una escena 3D para que se pueda interactuar con el movimiento de la cara del usuario de este sistema.

Se ha implementado esta escena 3D con la librería gráfica OpenGL. El programa tiene todos los componentes de una típica aplicación de tipo OpenGL. OpenGL es una librería grafica que permite la utilización de más de 200 órdenes para generar aplicaciones interactivas 3D. La librería está diseñada para ser independiente de la plataforma y del sistema operativo. Por éllo, no contiene órdenes de gestión de ventanas ni de dispositivos de entrada. Para realizar estas operaciones se necesitan otras librerías como la GLUT.

Se ha estructurado el programa OpenGL en los siguientes componentes:

1) Inicialización del sistema y de las ventanas.

2) Dibujo de la escena.

   Para realizar el dibujo de la escena es necesario indicar al sistema qué rutina se debe ejecutar. Esto se realiza mediante la orden gluDisplayFunc (void(*func) dibujar).

3) Bucle de gestión de eventos. Si el evento cambia el dibujo entonces redibujar.

4) Por último, se debe enviar a OpenGL la orden glutMainLoop(void) para visualizar la ventana creada. La aplicación entra en un bucle y comienza el proceso de los eventos y la ejecución de rutinas de visualización. La visualización sobre la ventana será efectiva.

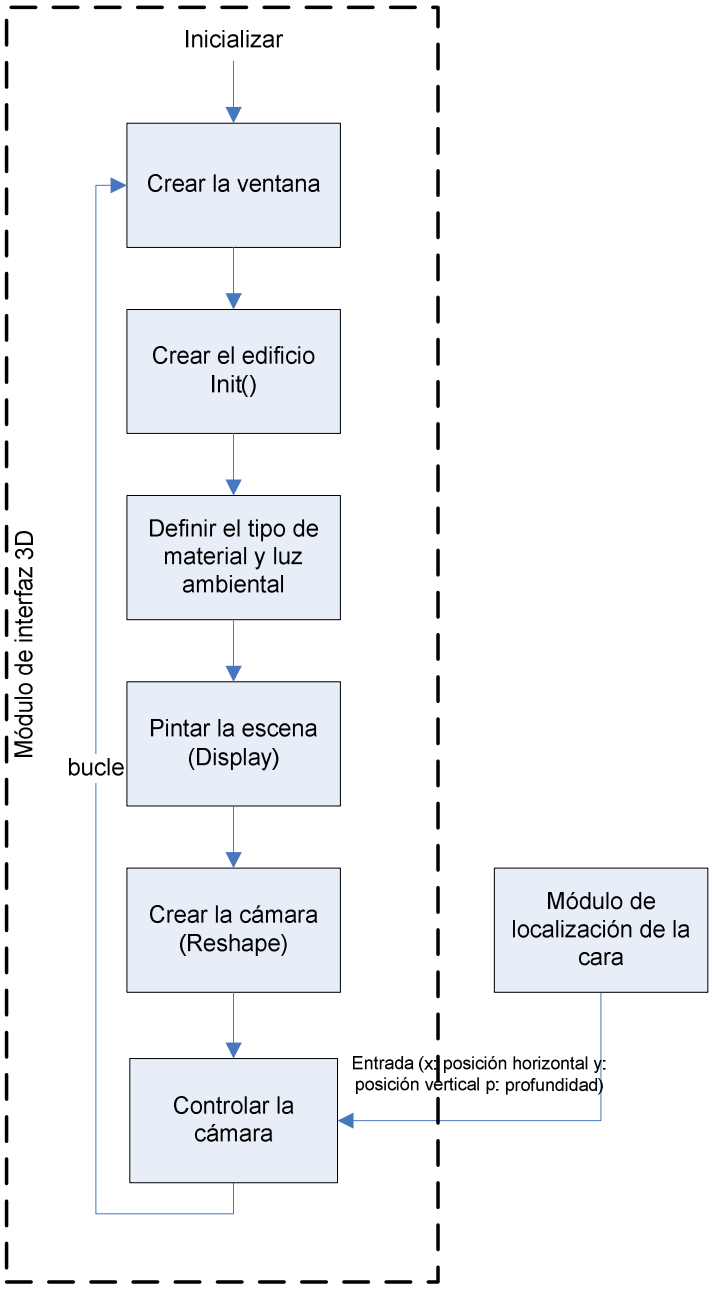


Figura 5.25 Diagrama de flujo de la interfaz 3D

Los detalles de cada rutina y el programa principal pueden encontrarse en el Anexo 9.2.

Display: La rutina que configura la iluminación de la escena y la pinta, etc.

Reshape: La rutina que actualiza la forma y el tamaño de la ventana, configuración de la cámara, etc.

Init: La rutina que crea todas las matrices de un castillo en puntos y líneas.

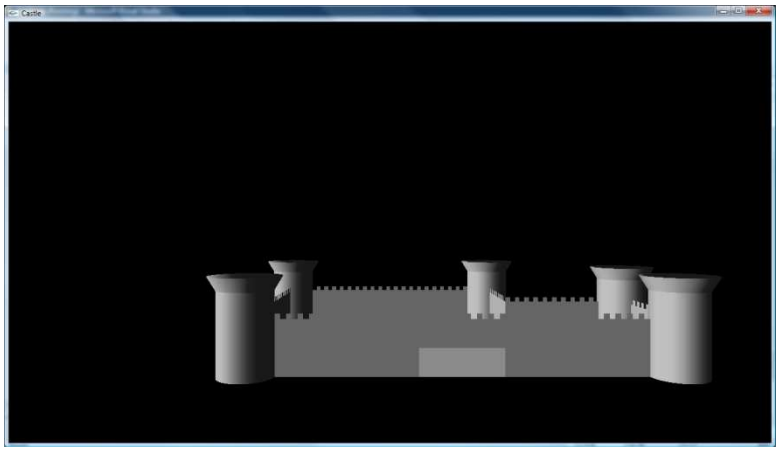

Figura 5.26 Visualización de la escena 3D

Para interactuar la escena con los movimientos de la cara, se ha implementado un procedimiento con parámetros de entrada como las coordenadas de la cara del usuario, la posición horizontal de la cara, la posición vertical de la cara y la distancia (profundidad) de la cara con la cámara. Con estos parámetros este procedimiento ya puede realizar los cambios necesarios de la cámara de la escena, como la nueva posición de la cámara y sus ángulos de giro en dirección vertical y horizontal para que se enfoque siempre al castillo.

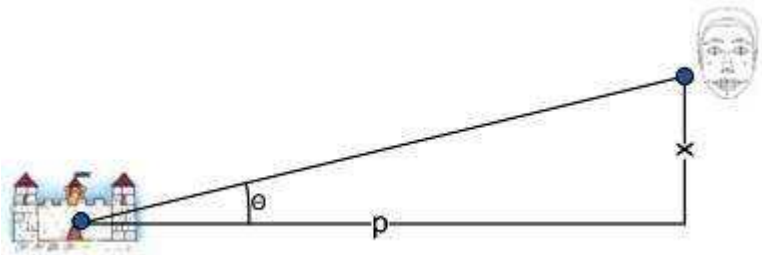


Figura 5.27 Vista vertical

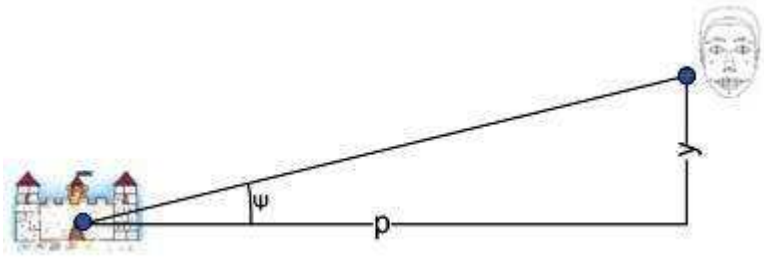


Figura 5.28 Vista lateral

**Procedure** Actualizar_Cámara (*x*: posición horizontal, *y*: posición vertical, *p*: profundidad):

- Camara_mover_ejeX(x);
- Camara_mover_en_ejeY(y);
- Θ = artg(x/p)/2, Camara_rotarY(Θ);
- Ψ= artg(y/p)/2, Camara_rotarX(Ψ)

# 6. Resultado de las pruebas

En este capítulo se presenta el rendimiento del sistema. En primer lugar, se realizaron pruebas para evaluar las propiedades del sistema. Luego el rendimiento de cada etapa. Por último, se explicó el funcionamiento del sistema. El rendimiento de todo el sistema depende de cada paso y el rendimiento de cada paso se efectúa por los resultados obtenidos en los pasos anteriores.

Por ejemplo, para realizar el seguimiento de la cara para interactuar con la interfaz 3D, el sistema debe localizar correctamente la cara en primer lugar, después de la localización de la cara se deben buscar los ojos en el área de la cara. Finalmente, después de la localización de los ojos, se realizan los cálculos de geometría para obtener la posición de la cara en la escena real.

## 6.1. Propiedades de la prueba de instalación

El funcionamiento del sistema ha sido con la videocámara FAMETECH USB20 PC Camera, proporcionando imágenes de resolución 640x480, con un PC estándar con procesador Intel Centrino Duo a 1.83 GHz y 2048 MB de memoria RAM. El rendimiento de cada paso está probado bajo distintas condiciones, tales como distractores alrededor de las áreas de interés de la imagen, diferentes tamaños de imagen, etc.

## 6.2. Resultados del sistema de seguimiento

CAMSHIFT es un seguidor de la cara computacionalmente eficiente. Aunque su sencillez causa limitaciones, la ejecución del seguimiento de la cara fue lo suficientemente buena como para utilizarla en este proyecto. Los resultados de las pruebas son los siguientes.

### 6.2.1. Seguimiento con presencia de distracciones

La ventana de búsqueda de CAMSHIFT converge para abarcar al color de piel dominante conectado según la distribución de probabilidad y cercanía. Es decir, sólo realizará el seguimiento de un área conectada que contenga las propiedades de color definidas en la etapa de calibración. La superposición de regiones de la cara y otros objetos del mismo color que la piel son parcialmente ignorados por CAMSHIFT.

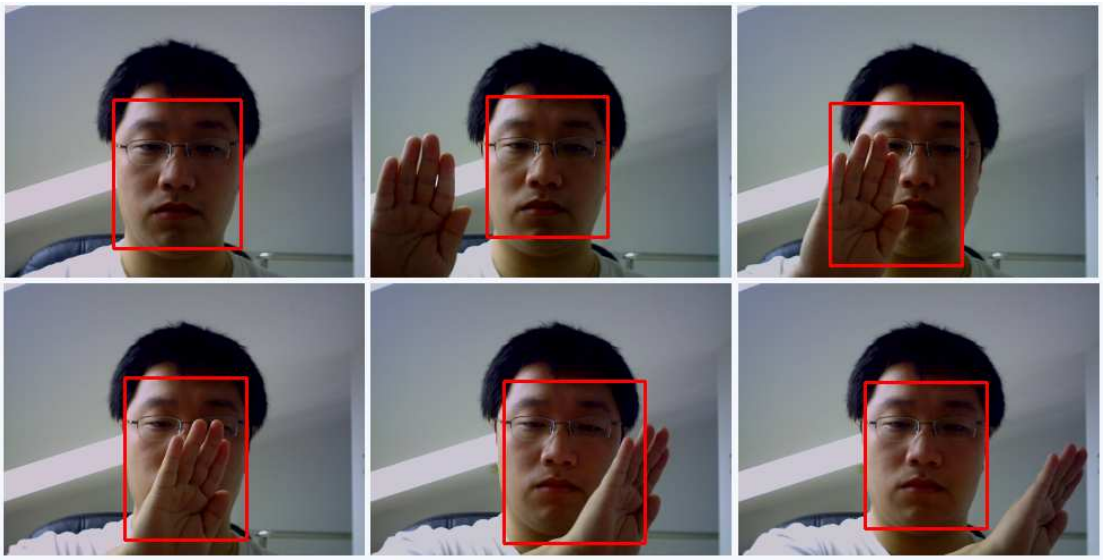

Figura 6.1 El seguimiento de una cara con la presencia de una mano que pasa por delante

El seguidor de la cara implementado ignora las anomalías y esto le permite ser robusto contra las distracciones. Una vez la cara está bajo seguimiento, se tienden a ignorar las anomalías cercanas que tienen una distribución de color no conectada. Como resultado, la presencia de otras caras o movimientos de la mano en la escena no causarán al algoritmo original perder la cara original a no ser que existan grandes ocultaciones.

En este trabajo se ha observado que realizar el seguimiento con fondos complejos es la principal desventaja del algoritmo implementado. Con movimientos de la cara, los objetos del fondo crearán una alta producción de distribuciones de probabilidad y harán crecer el área de tracking de la cara erróneamente. La detección de los ojos de más de una persona y su uso con fondos complejos durante el seguimiento no es posible para la interfaz humano-ordenador implementada. Por la propiedad anterior, nuestro algoritmo de seguimiento de la cara es adecuado para el sistema de detección de los ojos en tiempo real, teniendo una sola cara y haciendo ésta movimientos simples. Por otro lado, el rendimiento de la localización de los ojos depende de la anchura y la altura del rectángulo de la cara. Si se producen errores en el tamaño, puede provocar errores en la detección de los ojos.

### 6.2.2. Seguimiento en ambientes de luz variable

El algoritmo de seguimiento de la cara implementado utiliza solamente el componente HUE del espacio de color HSV. El brillo alto o bajo provoca errores en color porque esto causará falsas detecciones.

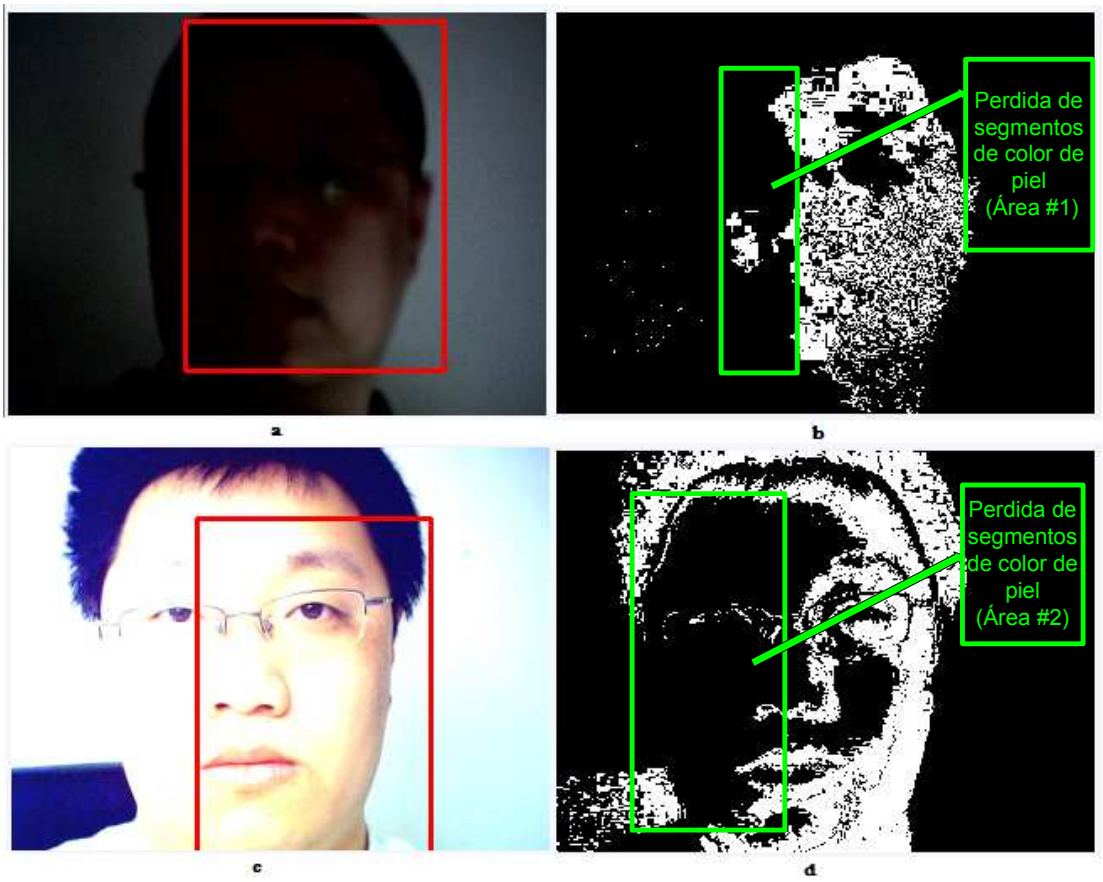


Figura 6.2 El seguimiento de una cara bajo diferentes condiciones de iluminación

El algoritmo implementado se ha aplicado en diferentes condiciones de iluminación. En la Figura 6.2 se resumen los resultados que se han obtenido en nuestros experimentos. Figura 6.2 – b es la distribución de probabilidad de color de la Figura 6.2 – a y similarmente Figura 6.2 – d es la distribución de probabilidad de color de la Figura 6.2 – c. como se observa en la Figura 6.2 – b la zona oscura de la cara (Región # 1) está ignorada por el algoritmos provocando que el rectángulo de la cara sea menos ancho. Del mismo modo, como se ve en la Figura 6.2 – d, la parte izquierda donde se aplica una alta luminosidad es ignorada por el algoritmo. Otro efecto de la alta iluminación es el error en HUE como se muestra en el área 3. Esto hace que el rectángulo de la cara sea más largo en altura.

En este trabajo se ha observado que variaciones de la iluminación son una desventaja del algoritmo implementado. Los píxeles con brillo alto o bajo son ignorados por CAMSHIFT. Haciendo caso omiso de las partes de la cara con iluminación alta o baja hace que la cara sea parcialmente detectada y esto causa fallos de detección de los ojos. Si se obtiene una mala información de la cara, se detectarán mal los ojos.

La calibración de la cámara es un paso importante en el seguimiento del objeto de

color antes de empezar el proceso de seguimiento. Por lo tanto, en este sistema, los cambios repentinos de color deben ser evitados necesariamente mediante ajustes y el brillo de la imagen no debe ser ni demasiado tenue ni demasiado saturado.

### 6.2.3. Detección de ojos y boca con diferentes posturas

El algoritmo de detección de ojos y boca funciona mediante reglas de las características faciales. Se comprueba la posición de los ojos en la imagen de la cara, la distancia y el ángulo entre los dos ojos.

Se han realizado pruebas de detección de ojos y pupilas con imágenes de una cara con diferentes posturas. Estas imágenes ya están procesadas para mejorar los efectos de la luz ambiental.

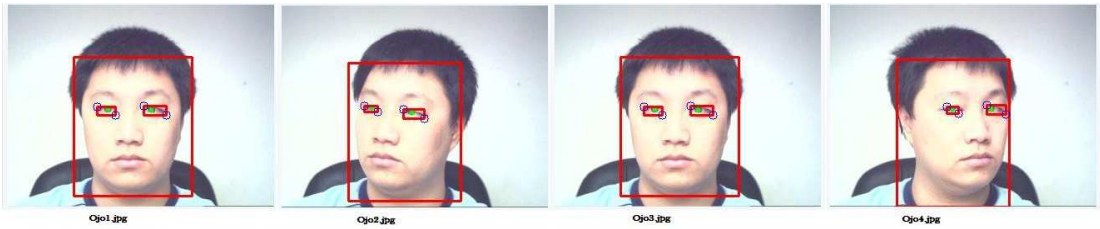


Figura 6.3 Imagen de seguimiento de ojos en 3 posturas


Figura 6.4 Imagen de seguimiento de los ojos en 6 posturas

La Figura 6.3 muestra los resultados de detección en 3 posturas diferentes: cara frontal, giro de la cara a la izquierda y giro de la cara a la derecha. Se han detectado

los ojos y las pupilas correctamente.

La Figura 6.4 muestra los resultados de detección de ojos y pupilas en 6 posturas diferentes: giro de la cara a la izquierda, giro de la cara a la derecha, movimiento de la cara en el eje X, movimiento de la cara en el eje Y, giro de la cara en el eje Z. Se puede observar que la mayoría de las detecciones se han cumplido correctamente, excepto la imagen número 11 que ha fallado.

Figura 6.5 Imágenes de detección de las pupilas

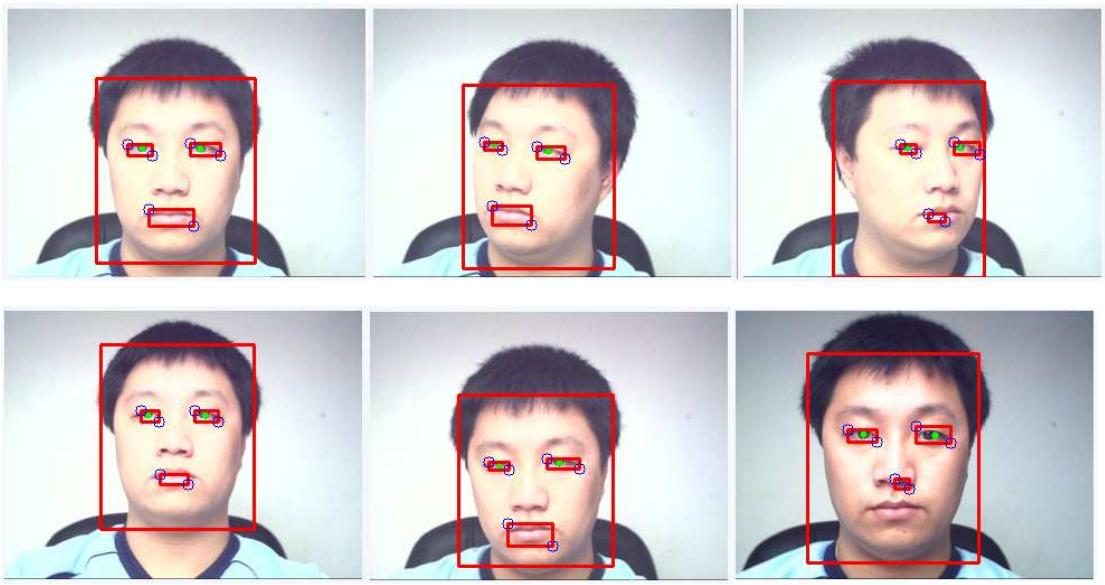

Figura 6.6 Imagen de detección de la boca en 3 posturas

Como se ha demostrado con las figuras anteriores, si se detectan correctamente los ojos, la detección de las pupilas se conseguirá en casi todos los casos. En la Figura 6.5 se muestra todo el proceso de detección de pupilas en imágenes separadas.

La figura 6.6 muestra los resultados de detección de la boca en diferentes posturas. Ha ocurrido el mismo problema que en la detección de los ojos: su proporción de detección ha sido influida por los resultados de la detección de la cara y por las condiciones de luz.

# 7. Conclusión

## 7.1. Objetivos cubiertos

El desarrollo del proyecto finalizó cumpliendo casi todas las partes que componen los objetivos del proyecto.

En este proyecto se ha desarrollado un sistema capaz de hacer el seguimiento de la cara y los ojos en tiempo real. El algoritmo de seguimiento de la cara del sistema está basado en el algoritmo CAMSHIFT desarrollado por [4] que es un algoritmo de seguimiento de la cara con muestra para la búsqueda del color de la piel humana en las imágenes.

La detección de los ojos y pupilas se hace buscándolos sobre la región de la cara que se ha detectado en el paso anterior. El sistema busca todos los contornos conectados en la imagen y después elimina los que no tienen posibilidad de ser ojos mediante unas comprobaciones de reglas basadas en las características faciales. La detección de la boca es similar a la de los ojos, tan sólo hay que añadir un paso más: analizar el color del labio. Durante el desarrollo de este componente, hubo problemas con la definición de reglas y la programación, debido a que no se daban los resultados esperados. Finalmente se volvió a diseñar y a programar de nuevo. Ahora el sistema ya tiene un resultado aceptable.

El último paso es el seguimiento de la cara. Este paso es más sencillo que otros componentes anteriores porque simplemente se calcula la diferencia entre la posición anterior y la nueva de la cara basándose en la geometría proyectiva. La interfaz 3D ha sido implementada con funciones de OpenGL. Este parte ha funcionado como se esperaba, ya que la precisión del resultado depende de los resultados de los componentes anteriores.

El sistema funciona correctamente con cámaras económicas y no requiere de la calibración de la lente de la cámara. Como el sistema es lo suficientemente automático, es muy fácil de utilizar gracias a la ayuda de una interfaz de usuario amigable.

## 7.2. Limitaciones y trabajos futuros

En principio se quiso desarrollar un sistema que fuera completamente automático, pero no hemos previsto usar el sistema junto con una cámara de video de baja calidad. Este sistema no ofrece un módulo de configuración de la cámara, luego no

hay manera de autoajustar los brillos y la saturación de la imagen mediante la cámara ni por software. Esto directamente afecta al resultado de detección de la cara. En el futuro habría que añadir este módulo de configuración tanto para la cámara como para los algoritmos que se han utilizado para la detección.

El rendimiento de todo el sistema depende mucho de la detección de la cara con el algoritmo CAMSHIFT. Las desventajas principales del seguimiento de la cara en este paso son que el sistema es muy afectado por la iluminación y por distracciones de alrededor de la cara humana. Especialmente cuando se intenta hacer el seguimiento con un fondo complejo o con una cámara de video de baja calidad. Esta es la desventaja principal del algoritmo que se ha implementado. Así pues, unos mejores enfoques en la eliminación de la iluminación y distractores podrían ser utilizados en una implementación futura.

La detección de los ojos y boca basada en el método de características faciales es un camino rápido de detección. Pero la fiabilidad no es muy alta. Por lo tanto, en el futuro se podría intentar hacer la implementación de la detección de ojos con el método de la plantilla deformable o con el método PCA.

Con este proyecto es posible calcular la posición de la cara y los ojos en una imagen y determinar los movimientos de ojos que indican la dirección de la mirada. En un trabajo futuro se podría mejorar el diseño de la interfaz de usuario con una interfaz 3D completamente basada en el seguimiento de los ojos.

## 7.3. Estimación inicial vs. Real

Los objetivos iníciales del proyecto han variado respecto a las metas desarrolladas finalmente. Es de prever que la estimación del tiempo de los objetivos iníciales analizados en un principio varíe a lo largo del desarrollo del mismo, por ello radica esta dificultad de concordancia del tiempo real dedicado en cada una de las partes del proyecto.

| Descripción de tareas | Duración estimada | Fecha inicio | Fecha fin |
|---|---|---|---|
| Estudio de los componentes del proyecto | 10 días (80 horas) | 25/02/09 | 10/03/09 |
| Análisis y diseño | 19 días (152 horas) | 11/03/09 | 06/04/09 |

| | | | |
|---|---|---|---|
| Implementación | 22 días (176 horas) | 07/04/09 | 06/05/09 |
| Testing y optimización | 7 días (56 horas) | 07/05/09 | 15/05/09 |
| Documentación | 13 días (104 horas) | 18/05/09 | 03/06/09 |
| Total | 71 días (568 horas) | | |

Tabla 7.1 Resumen de la estimación inicial

| Descripción de tareas | Duración estimada | Fecha inicio | Fecha fin | Desviaciones |
|---|---|---|---|---|
| Estudio de los componentes del proyecto | 10 días (80 horas) | 25/02/09 | 10/03/09 | 0 |
| Análisis y diseño | 20 días (248 horas) | 11/03/09 | 07/04/09 | +1 días |
| Implementación | 24 días (376 horas) | 08/04/09 | 11/05/09 | +2 días |
| Testing y optimización | 7 días (72 horas) | 12/05/09 | 20/06/09 | 0 días |
| Documentación | 18 días (136 horas) | 21/06/09 | 15/06/09 | +5 días |
| Total | 79 días (632 horas) | | | +16 días |

Tabla 7.2 Resumen de la estimación real

Como se puede observar en la tabla de arriba, las desviaciones que se han producido a lo largo del desarrollo del proyecto han sido negativas (en color naranja).

En definitiva, el tiempo real de duración del proyecto ha sido de 632 horas. Con lo cual, la desviación producida respecto a la planificación inicial es de 64 horas.

## 7.4. Áreas de aplicación

El seguimiento y análisis del movimiento de los ojos puede resultar especialmente interesante para las aplicaciones que dependan del diálogo del usuario y del ordenador, ya que estos representan medidas que pueden proporcionar información valiosa acerca de la atención visual y aspectos de los humanos.

Jacob [27] ha presentado muchas áreas de aplicación del seguimiento de los ojos en diseños de interfaces avanzadas. Las aplicaciones que se están utilizando en este trabajo son un punto de partida que se pueden utilizar en una variedad de áreas como: Ciencias de la Computación, Psicología, Ingeniería Industrial o Factores humanos. Se puede utilizar como una herramienta de posicionamiento de alta velocidad. Es también un método eficaz cuando el usuario tiene movilidad reducida.

Algunas aplicaciones más específicas pueden ser:

- Aviación: El análisis de los movimientos de los ojos se ilustra en la importancia de la pantalla principal de vuelo en una cabina moderna de cristal, ya que son la fuente principal de información durante un vuelo. Así, el análisis de los movimientos de los ojos del piloto puede utilizarse en las pruebas de rendimiento del piloto y para formación de pilotos novatos. Los movimientos de los ojos también pueden ser utilizados para evaluar la usabilidad de los nuevos instrumentos en la aeronave, como la selección del objetivo suministro, sistema de armas y mapas electrónicos en movimiento (EMM), en la que el piloto navega todo el mapa con movimientos de ojos.

- Sistemas de vigilancia del conductor: La falta de atención visual juega un papel importante en los accidentes de tráfico. Por lo tanto, el seguimiento del estado del ojo y la dirección de la mirada pueden ser utilizados como un sistema de contramedidas de accidentes.

- Sistemas de información interactiva: En el sistema de visualización de información basada en movimientos de ojos, el usuario utiliza sus ojos como un dispositivo señalador -como un puntero de ratón- para navegar visualmente entre los elementos seleccionables.

- Se puede utilizar para ayudar a personas discapacitadas -como tetrapléjicos- haciendo que los discapacitados puedan mover los ojos y con éstos controlar el sistema, cuando ellos no podrían manipular ningún otro dispositivo de entrada.

- Sistemas de seguridad basados en el reconocimiento del iris.

- Clínica de diagnóstico.
- Juegos por computadora.
- Etc.

## 7.5. Valoración personal

A lo largo del desarrollo del proyecto he adquirido muchos nuevos conocimientos, además de poner en práctica algunos aprendidos durante la carrera. El tiempo de progreso del proyecto ha sido una gran experiencia para mí en todos los sentidos, tanto profesionalmente de cara al mundo laboral que se avecina, como de forma académica, donde se concluye la finalización de los estudios de la carrera de Ingeniería Superior en Informática.

## 7.6. Agradecimientos

Han sido muchos años relacionados con la universidad, por ello es difícil nombrar a todas las personas que me han acompañado en esta etapa. Me gustaría agradecer su paciencia a todos los que sufrían mis cambios de humor, nervios previos a épocas de exámenes y el estrés provocado por la exigencia. A todos los compañeros y amigos por compartir sus conocimientos. Por supuesto, me gustaría dar las gracias a mi tutor Manel Frigola por la paciencia y dedicación para instruirme, a mi amigo Alberto Vila Vecilla por haberme ayudado a corregir en castellano este proyecto, a mi hermana pequeña por confiar en mí y sobre todo a mis padres por darme una educación y la oportunidad de escoger.

# 8. Bibliografía

[1] Open Source Computer Vision

http://opencv.willowgarage.com/wiki/

[2] The Industry's Foundation for High Performance Graphics

http://www.opengl.org/

[3] J.C. Terrillon, M. David, S. Akamatsu, (1998). “Automatic Detection of Human Faces in Naturel Scene Images by use of a Skin Color Model and of Invariant Moments”. Proc. Int. Conf. On Automatic Face and Gesture Recognition, pp. 112-117.

[4] Gary R. Bradski, “Computer Visison Face Tracking for Use in a Preceptual User Interface”. Microcomputer Research Lab, Santa Clara, CA, Intel

[5] E. Hjelmas and B. K. Low, (2001). “Face detection: A survey”. Computer

Vision and Image Understanding, vol. 83, pp. 236-274.90

[6] M. H. Yang, N. Ahuja, (2002). “Detecting Faces in Images: A Survey”. IEEE

Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No.1.

[7] G. Yang and T. S. Huang, (1994). “Human Face Detection in Complex Background”. Pattern Recognition, vol. 27, no. 1, pp. 53-63.

[8] Y. Miyake, H. Saitoh, H. Yaguchi, and N. Tsukada, (1990). “Facial Pattern Detection and Color Correction from Television Picture for Newspaper Printing”. J. Imaging Technology, vol. 16, no. 5, pp. 165-169.

[9] D. Saxe and R. Foulds, (1996). “Toward Robust Skin Identification in Video Images”. Proc. Second Int’l Conf. Automatic Face and Gesture Recognition, pp. 379-384.

[10] R. Kjeldsen and J. Kender, (1996). “Finding Skin in Color Images”. Proc. Second Int’l Conf. Automatic Face and Gesture Recognition, pp. 312-317.

[11] K. Sobottka and I. Pitas, (1996). “Face Localization and Feature Extraction Based on Shape and Color Information”, Proc. IEEE Int’l Conf. Image Processing, pp. 483-486.

[12] H. Wangand S.-F. Chang, (1997). “A Highly Efficient System for Automatic Face Region Detection in MPEG Video”. IEEE Trans. Circuits and Systems for Video Technology, vol. 7, no. 4, pp. 615-628.

[13] Y. Dai and Y. Nakano, (1995). “Extraction for Facial Images from Complex Background Using Color Information and SGLD Matrices”. Proc. First Int’l Workshop Automatic Face and Gesture Recognition, pp. 238-242.

[14] Q. Chen, H. Wu, and M. Yachida, (1995). “Face Detection by Fuzzy Matching”. Proc. Fifth IEEE Int’l Conf. Computer Vision, pp. 591-596.

[15] M.Welling, \Fisher Linear Discriminant Analysis" Department of Computer Science, University of Toronto

[16] Duda, Hart, \Pattern Classi_cation (2nd Edition)" Wiley 0471056693 (2000)

[17] Rowley HA. Neural Network-Based Face Detection. PhD thesis, Carnegie Mellon Univ., 1999.

[18] Rowley H, Baluja S, Kanade T. Rotation Invariant Neural Network-Based Face Detection. Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 38~44, 1998.

[19] Viola P., Jones M.: Rapid Object Detection using a Boosted Cascade of Simple Features, Computer Vision and Pattern Recognition, 2001

[20] R. Kothari and J.L. Mitchell (1996). “Detection of eye locations in unconstrained visual images”. Int Proc. International Conference on Image Processing, volume I, pages 519–522, Lausanne, Switzerland.

[21] A. Yuille, C. D.S., and H. P.W., (1989). “Feature extraction from faces using de formable templates". Int. Proc. CVPR, pp. 104-109.

[22] A. Blake and A. Yuille, (1992). “Active Vision”. Massachusetts Institute of Technology.

[23] A. Yuille, P. Hallinan, and D. Cohen, (1992). “Feature Extraction from Faces Using Deformable Templates”. Int’l J. Computer Vision, vol. 8, no. 2, pp. 99-111.

[24] P. Fieguth and D. Terzopoulos, (1997). “Color-based tracking of heads and other mobile objects at video frame rates”. In Proc. of IEEE CVPR, pp. 21-27.

[25] C. Wren, A. Azarbayejani, T. Darrell, A.Pentland, (1995). “Pfinder: Real- Time Tracking of the Human Body”, SPIE Vol. 2615.

[26] K. Fukunaga, (1990). “Introduction to Statistical Pattern Recognition”.Academic Press, Boston.

[27] R.J.K. Jacob, (1991). “The use of eye movements in human-computer interaction techniques”. ACM Transactions on Information Systems, 9(3):152–169.

[28] Allen, J.G., R.Y.D. Xu y J.S. Jin, (2004)"Object Tracking using CamShift Algorithm and Multiple Quantized Feature Spaces", En: *Pan-Sydney Area Workshop on Visual*

*information processing*, pp. 3-7.

[29] Bradski, G.R., (1998) "Computer Vision FaceTracking for Use in a Perceptual UserInterface", *Intel Technology Journal*, vol. 2, nº2, pp. 13-27.

[30] Comaniciu, D. y P. Meer, (2002) "Mean Shift: A Robust Approach Toward Feature Space Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, nº5, pp. 603-619.

# 9. Anexo

## 9.1. Librería INTEL OPENCV para el reconocimiento de objetos

### 9.1.1. Característica haar

#### cvHaarDetectObjects

Detects objects in the image

```
typedef struct CvAvgComp
{
    CvRect rect; /* bounding rectangle for the object (average
rectangle of a group) */
    int neighbors; /* number of neighbor rectangles in the group
*/
}
CvAvgComp;
```

```
CvSeq* cvHaarDetectObjects( const CvArr* image,
CvHaarClassifierCascade* cascade, CvMemStorage* storage, double
scale_factor=1.1, int min_neighbors=3, int flags=0, CvSize
min_size=cvSize(0,0) );
```

`Image:` Image to detect objects in.

`Cascade:` Haar classifier cascade in internal representation.

`Storage:` Memory storage to store the resultant sequence of the object candidate rectangles.

`scale_factor:` The factor by which the search window is scaled between the subsequent scans, for example, 1.1 means increasing window by 10%.

`min_neighbors:` Minimum number (minus 1) of neighbor rectangles that makes up an object. All the groups of a smaller number of rectangles than `min_neighbors`-1 are rejected. If `min_neighbors` is 0, the function does not any grouping at all and returns all the detected candidate rectangles, which may be

useful if the user wants to apply a customized grouping procedure.

`Flags:` Mode of operation. Currently the only flag that may be specified is `CV_HAAR_DO_CANNY_PRUNING`. If it is set, the function uses Canny edge detector to reject some image regions that contain too few or too much edges and thus can not contain the searched object. The particular threshold values are tuned for face detection and in this case the pruning speeds up the processing.

`min_size:` Minimum window size. By default, it is set to the size of samples the classifier has been trained on (~20×20 for face detection).

Example. Using cascade of Haar classifiers to find objects (e.g. faces).

```
#include "cv.h"
#include "highgui.h"
CvHaarClassifierCascade* load_object_detector( const char*
cascade_path )
{
    return (CvHaarClassifierCascade*)cvLoad( cascade_path );
}

void detect_and_draw_objects( IplImage* image,
                              CvHaarClassifierCascade* cascade,
                              int do_pyramids )
{
    IplImage* small_image = image;
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq* faces;
    int i, scale = 1;
    /* if the flag is specified, down-scale the input image to get
a performance boost w/o loosing quality (perhaps) */
    if( do_pyramids )
    {
        small_image = cvCreateImage( cvSize(image->width/2,image-
>height/2), IPL_DEPTH_8U, 3 );
        cvPyrDown( image, small_image, CV_GAUSSIAN_5x5 );
        scale = 2;
    }
/* use the fastest variant */
    faces = cvHaarDetectObjects( small_image, cascade, storage,
1.2, 2, CV_HAAR_DO_CANNY_PRUNING );
```

```
/* draw all the rectangles */
    for( i = 0; i < faces->total; i++ )
    {
/* extract the rectanlges only */
        CvRect face_rect = *(CvRect*)cvGetSeqElem( faces, i, 0 );
        cvRectangle( image,
cvPoint(face_rect.x*scale,face_rect.y*scale),
                     cvPoint((face_rect.x+face_rect.width)*scale,
                            (face_rect.y+face_rect.height)*scale),
                     CV_RGB(255,0,0), 3 );
    }

    if( small_image != image )
        cvReleaseImage( &small_image );
    cvReleaseMemStorage( &storage );
}

/* takes image filename and cascade path from the command line */
int main( int argc, char** argv )
{
    IplImage* image;
    if( argc==3 && (image = cvLoadImage( argv[1], 1 )) != 0 )
    {
        CvHaarClassifierCascade* cascade =
load_object_detector(argv[2]);
        detect_and_draw_objects( image, cascade, 1 );
        cvNamedWindow( "test", 0 );
        cvShowImage( "test", image );
        cvWaitKey(0);
        cvReleaseHaarClassifierCascade( &cascade );
        cvReleaseImage( &image );
    }
/* takes image filename and cascade path from the command line */
int main( int argc, char** argv )
{
    IplImage* image;
    if( argc==3 && (image = cvLoadImage( argv[1], 1 )) != 0 )
    {
```

```
        CvHaarClassifierCascade* cascade =
load_object_detector(argv[2]);
        detect_and_draw_objects( image, cascade, 1 );
        cvNamedWindow( "test", 0 );
        cvShowImage( "test", image );
        cvWaitKey(0);
        cvReleaseHaarClassifierCascade( &cascade );
        cvReleaseImage( &image );
    }

    return 0;

}
```

## 9.1.2. Contours

**FindContours**

Finds contours in binary image

```
int cvFindContours( CvArr* image, CvMemStorage* storage,
CvSeq** first_contour, int header_size=sizeof(CvContour),
int mode=CV_RETR_LIST, int method=CV_CHAIN_APPROX_SIMPLE,
CvPoint offset=cvPoint(0,0) );
```

`Image:` The source 8-bit single channel image. Non-zero pixels are treated as 1's, zero pixels remain 0's - that is image treated as `binary`. To get such a binary image from grayscale, one may use cvThreshold, cvAdaptiveThreshold or cvCanny. The function modifies the source image content.

`Storage:` Container of the retrieved contours.

`first_contour:` Output parameter, will contain the pointer to the first outer contour.

`header_size:` Size of the sequence header, >=sizeof(CvChain) if `method`=CV_CHAIN_CODE, and >=sizeof(CvContour) otherwise.

`Mode:` Retrieval mode.

- CV_RETR_EXTERNAL - retrive only the extreme outer contours
- CV_RETR_LIST - retrieve all the contours and puts them in the list
- CV_RETR_CCOMP - retrieve all the contours and organizes them into two-level hierarchy: top level are external boundaries of the components, second level are bounda boundaries of the holes
- CV_RETR_TREE - retrieve all the contours and reconstructs the full hierarchy of nested contours

Method: Approximation method (for all the modes, except CV_RETR_RUNS, which uses built-in approximation).

- CV_CHAIN_CODE - output contours in the Freeman chain code. All other methods output polygons (sequences of vertices).
- CV_CHAIN_APPROX_NONE - translate all the points from the chain code into points;
- CV_CHAIN_APPROX_SIMPLE - compress horizontal, vertical, and diagonal segments, that is, the function leaves only their ending points;
- CV_CHAIN_APPROX_TC89_L1,

  CV_CHAIN_APPROX_TC89_KCOS - apply one of the flavors of Teh-Chin chain approximation algorithm.

- CV_LINK_RUNS - use completely different contour retrieval algorithm via linking of horizontal segments of 1's. Only CV_RETR_LIST retrieval mode can be used with this method.

Offset: Offset, by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

The function cvFindContours retrieves contours from the binary image and returns the number of retrieved contours. The pointer first_contour is filled by the function. It will contain pointer to the first most outer contour or NULL if no contours is detected (if the image is completely black). Other contours may be reached from first_contour using h_next and v_next links. The sample in cvDrawContours discussion shows how to use contours for connected component detection. Contours can be also used for shape analysis and object recognition - see squares.c in OpenCV sample directory.

## 9.2. Rutinas de OpenGL

**Display:**

```
void Display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        Camera.Render();
        glLightfv(GL_LIGHT0,GL_POSITION,LightPos);
        glTranslatef(-5.0,0.0,0.0);
        glRotatef(90.0,0.0,1.0,0.0);
        glBegin(GL_POLYGON);
                glNormal3f(0.0,1.0,0.0);
                glVertex3f(0.0,0.0,0.0);
                glVertex3f(10.0,0.0,0.0);
                glVertex3f(10.0,0.0,10.0);
                glVertex3f(5.0,0.0,15.0);
                glVertex3f(0.0,0.0,15.0);
                glVertex3f(0.0,0.0,0.0);
        glEnd();
    //Turn two sided lighting on for the walls
        glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
        glCallList(WallsListNum);
    //Disable it again for the towers:
        glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
        glCallList(TowerListNum);
        glTranslatef(10.0,0.0,0.0);
        glCallList(TowerListNum);
        glTranslatef(0.0,0.0,10.0);
        glCallList(TowerListNum);
        glTranslatef(-5.0,0.0,5.0);
        glCallList(TowerListNum);
        glTranslatef(-5.0,0.0,0.0);
        glCallList(TowerListNum);
        glFlush();
    //Finish rendering
        glutSwapBuffers();
    //Swap the buffers ->make the result of rendering visible

}
```

**Reshape:**

```
void Reshape(int x, int y)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //Angle of view:40 degrees
        //Near clipping plane distance: 0.5
        //Far clipping plane distance: 20.0
        gluPerspective(40.0,(GLdouble)x/(GLdouble)y,1.0,200.0);
        glMatrixMode(GL_MODELVIEW);
        glViewport(0,0,x,y);
}
```

**Init:**

```
void Init(void) //used to create the display lists
{
        TowerListNum = glGenLists(1);
        GLfloat x,z;
        GLfloat NVectY;  //y component for the NVects of the higher part
        glNewList(TowerListNum, GL_COMPILE);
                glBegin(GL_QUADS);
                //Create the lower part of the tower:
                int i;
                for (i = 0; i < NumOfEdges-1; i++)
                {
                        x = cos((float)i/(float)NumOfEdges * PI * 2.0);
                        z = sin((float)i/(float)NumOfEdges * PI * 2.0);
                        glNormal3f(x,0.0,z);
                        glVertex3f(x,LowerHeight,z);
                        //same x,z and NVect:
                        glVertex3f(x,0.0,z);
        //Create the higher part:
        //The y component is the same for all NVects, so we can calculate it here:
    NVectY = (HR-1.0) / (LowerHeight - HigherHeight) * (HR-1.0);
                for (i = 0; i < NumOfEdges-1; i++)
                {
                        x = cos((float)i/(float)NumOfEdges * PI * 2.0);
                        z = sin((float)i/(float)NumOfEdges * PI * 2.0);
                        glNormal3f(x,NVectY,z);
                        glVertex3f(x*HR,HigherHeight,z*HR);
```

```
        //same x,z and NVect:
                        glVertex3f(x,LowerHeight,z);
                        x = cos((float)(i+1)/(float)NumOfEdges * PI * 2.0);
                        z = sin((float)(i+1)/(float)NumOfEdges * PI * 2.0);
                        glNormal3f(x,NVectY,z);
                        glVertex3f(x,LowerHeight,z);
                        //same x,z and NVect:
                        glVertex3f(x*HR,HigherHeight,z*HR);
                }

          x = cos((float)i/(float)NumOfEdges * PI * 2.0);
                z = sin((float)i/(float)NumOfEdges * PI * 2.0);
                glNormal3f(x,NVectY,z);
                glVertex3f(x*HR,HigherHeight,z*HR);

//same x,z and NVect:
                glVertex3f(x,LowerHeight,z);
          x = cos(1.0/(float)NumOfEdges * PI * 2.0);
                z = sin(1.0/(float)NumOfEdges * PI * 2.0);
                glNormal3f(x,NVectY,z);
                glVertex3f(x,LowerHeight,z);
                //same x,z and NVect:
                glVertex3f(x*HR,HigherHeight,z*HR);
                glEnd();

        glEndList();

////////////////////////////////////////////////////////
        //WallList
////////////////////////////////////////////////////////

        WallsListNum = glGenLists(1);
        glNewList(WallsListNum, GL_COMPILE);
        DrawWall(10.0);
        glPushMatrix();
        glTranslatef(10.0,0.0,0.0);
        glPushMatrix();
                glRotatef(270.0,0.0,1.0,0.0);
                DrawWall(10.0);
        glPopMatrix();
        glTranslatef(0.0,0.0,10.0);
        glPushMatrix();
```

```
    glRotatef(180.0,0.0,1.0,0.0);
              DrawWall(5.0);
              glRotatef(90.0,0.0,1.0,0.0);
              glTranslatef(0.0,0.0,5.0);
              DrawWall(5.0);
       glPopMatrix();
    glTranslatef(-5.0,0.0,5.0);
       glPushMatrix();
              glRotatef(180.0,0.0,1.0,0.0);
              DrawWall(5.0);
       glPopMatrix();



        //Last and longest piece:
              glPushMatrix();
              glRotatef(90.0,0.0,1.0,0.0);
              glTranslatef(0.0,0.0,-5.0);
              DrawWall(6.0);

            //the "door"
                      glTranslatef(6.0,0.0,0.0);
                      glBegin(GL_QUADS);
                              glNormal3f(0.0,0.0,-1.0);
                              glVertex3f(0.0,WallHeight / 2.0,0.0);
                              glVertex3f(0.0,WallHeight,0.0);
                              glVertex3f(3.0,WallHeight,0.0);
                              glVertex3f(3.0,WallHeight / 2.0,0.0);

                      glEnd();
                      i = (int)(3.0 / WallElementSize / 2);
                      if (i * WallElementSize > 3.0) i--;
                              glPushMatrix();
                              glTranslatef(0.0,WallHeight,0.0);
                              DrawHigherWallPart(i);
                      glPopMatrix();
              glTranslatef(3.0,0.0,0.0);
              DrawWall(6.0);
       glPopMatrix();
       glPopMatrix();
       glEndList();
}
```

**Programa principal:**

```
//Iniciar el sistema y crear la ventana
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(300,300);
    glutCreateWindow("Castillo");
    glEnable(GL_DEPTH_TEST);
    glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
    glClearColor(0.0,0.0,0.0,0.0);
    //Dibujar la escena
    glutDisplayFunc(Display);
    //Crear la cámara
    glutReshapeFunc(Reshape);
    //Definir el tipo de material
    glMaterialf(GvL_FRONT_AND_BACK, GL_SPECULAR, MatSpec);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, MatShininess);
    //Definir luz
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ModelAmb);
    Init();
    //comienza el proceso, ejecutar las rutinas de visualización
    glutMainLoop();
```

## 9.3. LISTA DE LAS TABLAS

## 9.4. LISTA DE LAS FIGURAS