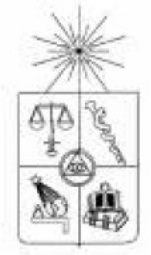

**UNIVERSIDAD DE CHILE**
**FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS**
**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

# DESARROLLO DE UNA INTERFAZ DE PROGRAMACIÓN PARA DETECCIÓN Y SEGUIMIENTO DE RASGOS FACIALES

**MEMORIA PARA OPTAR AL TÍTULO DE**
**INGENIERO CIVIL EN COMPUTACIÓN**

**FRANCISCO JAVIER RODRÍGUEZ ELORZA**

PROFESOR GUÍA:
PATRICIO INOSTROZA

MIEMBROS DE LA COMISIÓN:
ERIC TANTER
JOHAN FABRY

SANTIAGO DE CHILE
ENERO 2008

# Resumen

Las interfaces de usuario permiten la comunicación entre una persona y un computador. La calidad de la interacción depende en gran medida de la comodidad y exactitud de las ordenes que el usuario le entrega a la máquina. Por lo general las personas que tienen alguna discapacidad física en sus manos ven comprometidas la utilización de estas tecnologías, por lo que resulta muy necesario implementar nuevas interfaces que faciliten el uso y también expandan las posibilidades de comunicación.

La detección de la cara y rasgos faciales en tiempo real permite crear una nueva gama de interfaces de usuario, ya que la persona puede entregarle ordenes a la máquina con solo mover su cabeza, ojos o realizando alguna mueca. La enorme cantidad de pequeños músculos que existen en el rostro proveen muchas posibilidades de interacción.

El presente trabajo de título tiene por objetivo crear una API que detecte la ubicación del rostro y los ojos en un video en tiempo real. La que va a permitir la creación de nuevas interfaces de usuario en el futuro.

Para la implementación de la API se utilizaron algoritmos de aprendizaje que recuperan características del rostro y de los ojos, los que son de vital importancia para la detección.

Como resultado la API permite encontrar en tiempo real la ubicación del rostro y los ojos, y mediante una sencilla aplicación se demuestra el potencial que tiene esta herramienta.

# Agradecimientos

Agradezco a Dios por todas las oportunidades que me ha regalado en la vida, a todas las personas que ha puesto en mi camino para llegar al lugar en que estoy. En particular quiero agradecer a mis padres por todas sus preocupaciones con mi persona y a mis hermanos por demostrarme que yo también podía estudiar esta carrera y por todo su apoyo.

Agradezco también a mi profesor guía Patricio Inostroza por darme la posibilidad de realizar este trabajo. A Nic Chile por el financiamiento de esta memoria. A la Escuela de Ingeniería de la Universidad de Chile por abrirme sus puertas, a todos los académicos, secretarias, funcionarios que me entregaron su cariño durante mi permanencia en la escuela.

Por último y muy afectuosamente agradezco a mis amigos por estar siempre ahí conmigo tanto en los momentos buenos como en malos. A la Iglesia Católica por transmitirme la Fe en Dios y a mi comunidad por todas sus oraciones, apoyo y cariño.

# Índice general

# Capítulo 1

# Introducción y objetivos

## 1.1. Introducción

Desde el inicio de la computación se han buscado formas que ayuden a la interacción entre un hombre y su computador. Idealmente se busca que el esfuerzo que requiera la persona que interactúa con la interfaz sea el menor posible ya sea para adaptarse al modo de operar de la interfaz o de sus posibles restricciones que dificultan la manera de trabajar con ella.

Las interfaces de usuario permiten a una persona establecer una comunicación con un computador. A través de éstas, la persona puede dar información al sistema. Los dispositivos más tradicionales y conocidos son el teclado y el mouse. Hay otros tipos de dispositivos menos comunes como el SpaceMouse [1] y el Phantom [2]. Uno de los dispositivos más innovadores y famosos creados en el último tiempo es el Wii Remote [3], el que trae in-

corporado un sensor de movimiento que responde a los movimientos de las manos cambiando el paradigma de uso de los videojuegos, ahora es mucho más simple jugar dado que las instrucciones que genera el usuario se hacen de una manera más intuitiva que sólo al apretar algunos botones.

El desafío de brindar nuevas interfaces tiene como objetivo mejorar la interacción entre una persona y un computador, facilitando principalmente el uso y control del éste. Esto trae consigo el desarrollo de software que explote las funcionalidades provistas por la nueva interfaz, en caso que éstas existan.

Cabe mencionar que las interfaces mencionadas anteriormente traen consigo requisitos que el usuario debe cumplir para poder hacer uso de ellas, por ejemplo para usar un teclado o un mouse la persona debe tener habilidades en sus manos que permitan controlar estos dispositivos, en caso que la persona tuviese problemas en sus manos o por algún accidente las haya perdido, se le hace casi imposible usar este tipo de dispositivos y por ende se complica muchísimo utilizar un computador para éstas personas.

En esta memoria se desarrollará una interfaz de programación (API) la cual detectará y seguirá el rostro y otros rasgos de este, en particular se intentará detectar y seguir el ojo. Para ello se necesitará de algún tipo de cámara que capture las imágenes del rostro, para su posterior procesamiento. La función de esta memoria es permitir el desarrollo de nuevas aplicaciones como por ejemplo el control del puntero del mouse [4], [5], [6], [7] animar expresiones del rostro humano [8] y tantas otras. Estas nuevas aplicaciones

idealmente deben ayudar a personas con discapacidad física o mental a usar el computador.

La adaptación de las personas que utilizan la interfaz es primordial en la utilización de ésta, ya que si el usuario no se logra acostumbrar a manejarla, esta será dejada de lado y por ende va a fallar en su objetivo de ayudar a la comunicación entre el usuario y un computador.

Se busca entonces innovar con interfaces de usuario que necesiten utilizar recursos simples para su adaptación. Para lograr este resultado, se necesita que el sistema computacional sea consciente de la presencia de una persona.

El punto anterior es muy útil en el caso en que los usuarios del sistema tengan problemas de discapacidad física o mental.

Se han desarrollado investigaciones que utilizan procesamiento de imágenes para el seguimiento de los ojos y el control del mouse [4] y también para capturar distintas zonas del rostro y usarlas en una animación 3D [8]. La limitante de estas investigaciones es que las soluciones fueron dependientes de la aplicación, es decir, atacan sólo la problemática descrita y no permite darle nuevos usos como el desarrollo de nuevas aplicaciones.

En el presente trabajo de memoria se busca desarrollar una interfaz de programación (API) donde se interpretará la información captada por una cámara, para luego proveer datos relevantes sobre la posición del rostro y de algunos rasgos de éste tales como la posición de los ojos, la nariz o la boca.

Es de gran importancia la implementación de la API ya que puede servir como base de nuevos proyectos en el corto plazo. Estos proyectos futuros

pueden utilizar la API sin necesidad de entender mayormente como funcionan los algoritmos de detección, sólo deben comprender cuáles son los parámetros de entrada y los resultados entregados por la API.

Otra característica relevante de desarrollar una API para la detección del rostro y rasgos faciales es que puede ser reutilizada o extendida para abarcar objetivos más especificos. También se pueden implementar nuevos algoritmos que tengan mejores resultados o utilicen nuevas técnicas de detección que se descubran en el futuro.

Para la evaluación de este trabajo, se construirá una aplicación prototipo que verifique el potencial de la API y su correcto funcionamiento.

## 1.2. Objetivo General

Desarrollar una interfaz de programación (API) para la detección y seguimiento del rostro y rasgos faciales mediante el procesamiento de imágenes capturadas por una cámara de video.

Se espera que como resultado del procesamiento de las imágenes capturadas, se entregue la posición e imágenes del rostro y de rasgos faciales característicos, como los ojos, la nariz o la boca, lo que permitiría un seguimiento de los mismos. Sin embargo, en este trabajo de memoria centrará su esfuerzo en la detección de la posición del ojo como rasgo facial aparte de la detección de la cara. Como objetivo también se incluye el desarrollo de una aplicación prototipo que demuestre la utilidad de la API para diversos

contextos.

## 1.3. Objetivos Específicos

Los objetivos específicos que se plantean son los siguientes.

1. Realizar una búsqueda bibliográfica sobre trabajos relacionados.

2. Definir los elementos que compondrían una interfaz de programación (API) que apoye la búsqueda y seguimiento de la cara y rasgos faciales.

3. Implementar la API antes definida analizando las imágenes obtenidas mediante una cámara.

4. Optimizar el procesamiento de datos e imágenes para alcanzar respuestas del API en tiempo real.

# Capítulo 2

# Antecedentes

En este capítulo se expondrán diversos métodos para la detección de las pupilas del ojo y también para la detección del rostro.

En memorias orientadas a encontrar la pupila del ojo, se utilizaba una técnica que se basaba en enviar luz infrarroja a la pupila emulando el efecto de ojos rojos [13],[8],[4]. Distintas investigaciones demuestran que se puede detectar la ubicación del ojo y la pupila en un video analizando características de la intensidad de cada píxel [9], [10] que se encuentra dentro de la región de la cara, aprovechando el hecho que el iris y la pupila tienen un alto contraste con la esclera. Además se presenta en este informe una técnica que determina patrones rectangulares (características rectangulares) sobre una muestra de caras y ojos[11],[12] los que deciden si una imagen pequeña (sub-ventana) es una cara un ojo o cualquier otra cosa.

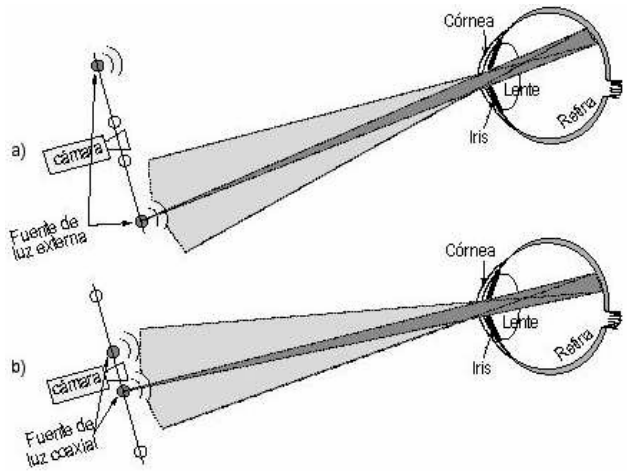


Figura 2.1: Esquema del efecto de la iluminación activa y su refracción.

## 2.1. Detección de la pupila del ojo usando una luz infrarroja

Esta técnica ha sido utilizada en varios trabajos de memoria con resultados satisfactorios [4], [8], [13]. El método consiste en un sistema que ilumina la retina del ojo aprovechando la propiedad reflexiva que tiene. Al iluminar el ojo con una fuente de luz muy cercana al lente de una cámara se produce como resultado el efecto de "ojos rojos". En la figura 2.1 se ejemplifica ésto.

La detección se logra al comparar imágenes sucesivas, una de ellas con la luz infrarroja encendida, Figura 2.1b) para provocar el efecto mencionado anteriormente y otra con esa fuente de luz apagada. Por ende entre ambas imágenes la única diferencia vendría a ser las pupilas de los ojos.

## 2.2. Detección del ojo utilizando la intensidad de los píxeles y bordes

Se han desarrollado diversos algoritmos para detectar y extraer zonas del rostro. En [9], [10] se desarrolló un método que se compone de dos partes, la primera estima la región donde se encuentra el rostro y la segunda donde se extraen las zonas del rostro que se interesan encontrar. En la primera parte, para estimar donde se encuentra el rostro, se define una región pequeña (píxel), la cual va creciendo en la medida que los píxeles adyacentes son similares a éste. El problema con este algoritmo es cuando la imagen donde se encuentra el rostro cuenta con un fondo con tonalidades muy similares a la del rostro. Una mejora sustancial a este algoritmo es la de borrar la parte de la imagen externa al contorno del rostro. Para ello se utiliza el algoritmo de Sobel [17] de detección de bordes antes de aplicar la técnica antes descrita.

Al interior de la región facial se procede a buscar candidatos para las zonas del rostro que estamos buscando, en particular los ojos. Para realizar esto se asignan costos o pesos a cada píxel y dependiendo de la intensidad en comparación con el resto de la imagen que éste píxel y algunos de sus vecinos tengan se considerarán como candidatos para encontrar la zona de los ojos.

Finalmente se divide la imagen en tres zonas, la región del ojo izquierdo, la del ojo derecho y la de la boca. Dentro de estas zonas se escogen los píxeles con mayor costo, determinando de esa forma posición del ojo izquierdo, del ojo derecho y la posición de la boca.

## 2.3. Detección del rostro usando características rectangulares

Para lograr una búsqueda más eficiente surgió una variante a la detección mediante la intensidad de los píxeles. Esta consiste en analizar regiones más grandes que un píxel, en particular regiones rectangulares características de las caras que permiten diferenciarlas de otros objetos. Dicho de otra manera primero se buscan patrones rectangulares sobre una muestra de caras y no caras. Para encontrar las caras se realiza un barrido de sub-ventanas a la imagen calculando para cada una de ellas los patrones. El resultado de los patrones determina sin una sub-ventana es efectivamente cara [11].

Los patrones (características rectangulares) señalados anteriormente corresponde a la suma de la intensidad de los píxeles al interior de las regiones rectangulares. Podría parecer que se estaría trabajando directamente con los píxeles siendo igualmente ineficiente, pero para mejorar ese hecho se creó una estructura muy similar a la de la imagen original que permite calcular la suma de intensidad de píxeles al interior de un rectángulo en tiempo constante sin importar el tamaño ni la posición del rectángulo, ésta estructura tiene el nombre de imagen integral.

Existen diferentes patrones a evaluar dentro de la sub-ventana como las que aparecen en la figura 2.2. El resultado de cada característica rectangular es la diferencia de la suma de píxeles de las áreas al interior del rectángulo.

Para encontrar directamente el(los) rostro(s) se realiza un barrido com-

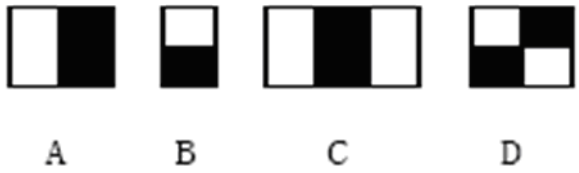


Figura 2.2: Rectángulos característicos utilizados para detectar el rostro.

pleto y a distintas escalas de sub-ventanas en la imagen, en cada posición se evalúan las características rectangulares y se determina si es una cara, ésto depende del resultado que es comparado con un valor umbral.

Se calculan previamente los umbrales y las características rectangulares a usar mediante Adaboost [4.4.3]. Un conjunto de caras y no caras son procesados para determinar el umbral y la característica rectangular. Las características rectangulares pueden ser combinados e incluso formar un algoritmo de decisión en forma de cascada, donde en cada etapa se descartan las imágenes que no correspondan a ser clasificadas como caras.

La imagen integral es una estructura del mismo tamaño que la imagen original, en ella se guardan los valores que después sirven para calcular la suma de píxeles de las distintas zonas rectangulares. Cada elemento de la imagen integral corresponde a la suma de los píxeles ubicados sobre y a la izquierda de la posición del píxel en la imagen original, en la figura 2.3 se muestra más claramente.

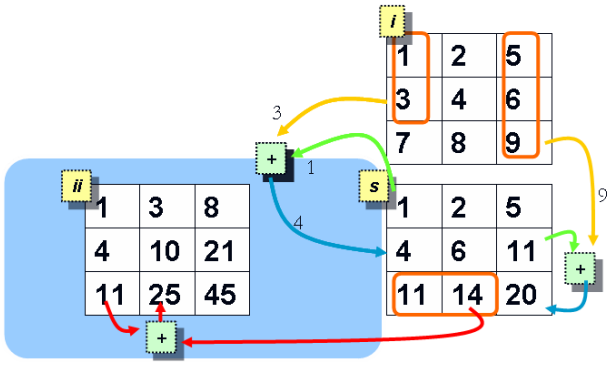


Figura 2.3: Esquema para el cálculo de la integral de imagen, dentro de un rectángulo.

## 2.4. Detección de los ojos utilizando características rectangulares

De manera similar a lo expuesto en el punto anterior se realiza la búsqueda de los ojos en una imagen, la única diferencia vendría a ser las nuevas características rectangulares que se utilizan, estas fueron diseñadas dada la configuración del ojo. Una vez encontrado el ojo se puede utilizar el algoritmo que se basa en la intensidad de los píxeles para encontrar con mayor exactitud la pupila.

## 2.5. Conclusiones

En este trabajo se utilizaron las características rectangulares para la detección del rostros y de los ojos. La principal ventaja que tiene este método con respecto a los otros señalados en éste capítulo es el no depender de una

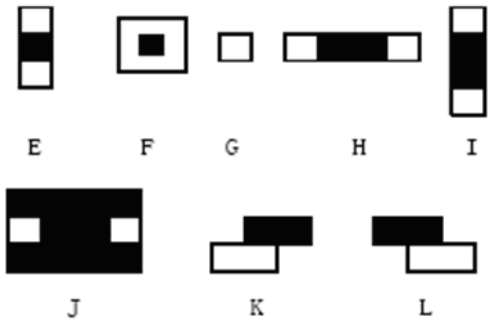


Figura 2.4: Rectángulos característicos utilizados para detectar el ojo.

fuente de luz externa y aprovechar la eficiencia de las características rectangulares.

# Capítulo 3

# Plataforma de trabajo

En este capítulo se señalarán las herramientas utilizadas durante el desarrollo de esta memoria. Se detallarán tanto el hardware como el software que apoyaron este trabajo.

## 3.1. Hardware

El desarrollo de la memoria se realizó en un computador con procesador AMD Athlon 64 X2 3800+ con 2 GB de memoria DDR2. Para la digitalización de las imágenes obtenidas desde la cámara se utilizó la tarjeta Prolink Pixelview PlayTV, la que permite conectar cualquier dispositivo con salida de video analógica.

La cámara utilizada durante la memoria es la SONY EVI-D70, la que posee un zoom óptico importante y además esta montada sobre un mecanismo

que permite mover el lente en distintas direcciones para obtener una toma más apropiada, el mecanismo mecánico y el zoom digital de la cámara puede ser controlada desde la aplicación. La cámara está conectada al PC por un cable de video compuesto directamente a la capturadora y por un cable serial. La cámara es controlable por medio del protocolo VISCA, creado por SONY. La biblioteca libVISCA es usada en la aplicación desarrollada con el fin de realizar algunos ajustes para determinar la dirección hacia donde apuntará la cámara.

## 3.2. Software

El sistema operativo utilizado fue la distribución de Linux Ubuntu 8.04, el que viene con la API video4linux incluida. Esta API permite acceder a los frames provistos por la capturadora de video antes descrita. La API de esta memoria y la aplicación desarrollada fueron programadas en C/C++ por motivos de eficiencia y de fácil integración con video4linux. Además se utilizó brevemente el lenguaje interpretado Python, para realizar algunas pruebas sobre un tipo de imágenes en particular.

En el desarrollo de la aplicación de prueba se usó además la biblioteca OpenGL para la visualización de las imágenes en la pantalla y brindar un atractivo gráfico a la aplicación final. A continuación se explicará con más detalle algunas componentes utilizadas durante el desarrollo de la memoria.

### 3.2.1. Video4linux

Video4linux es una API de captura de video hecha para Linux [19], soporta una amplia variedad de capturadoras de video y webcams. Además viene integrado en el kernel. El método de captura es el siguiente, primero se abre el dispositivo de captura, luego se lee su configuración y se modifica dependiendo de los requerimientos de captura, se pide la memoria necesaria para los buffers donde se van a almacenar los cuadros (frames) de captura, se piden los cuadros uno a uno para posteriormente ser procesados.

A través de la API video4linux es posible capturar video con una fluidez de 30 cuadros por segundo (fps). Para lograr este desempeño se deben realizar sincronizaciones con el buffer de memoria que utiliza el dispositivo.

Los cuadros capturados son posteriormente enviados como entrada a la API desarrollada en esta memoria, la que procesa y determina tanto la posición de los rostros detectados como la de los ojos. Una caractéristica del trabajo realizado es que no depende del método de captura de las imágenes y se podría utilizar cualquier otra forma de capturar video e incluso se podría leer un video almacenado y usarlo como una entrada.

### 3.2.2. OpenGL

OpenGL es una biblioteca gráfica [18], permite realizar aplicaciones que generen gráficos 2D y 3D. Se utilizó ésta biblioteca para desplegar en pantalla las imágenes procesadas.

Figura 3.1: Aplicación 3D desarrollada en OpenGL visualizando un canal de televisión.

Para reflejar los resultados obtenidos por la API presentada en este trabajo, se utilizó OpenGL para mostrar en pantalla las imágenes procesadas con pequeños detalles como por ejemplo encerrar con un cuadrado los rostros y ojos detectados.

Una ventaja de la API desarrollada en esta memoria es que no depende de una biblioteca gráfica para desplegar los resultados obtenidos, se puede utilizar cualquier otra biblioteca gráfica ya que no es necesario de OpenGL para la detección en sí, sólo se utiliza para mostrar en pantalla las imágenes procesadas.

Una pequeña aplicación permite corrobar la eficacia de esta biblioteca gráfica para el despliegue de una imagen digitalizada por la capturadora de video a 30 fps (ver figura 3.1).

# Capítulo 4

# Desarrollo

En las secciones 2.3,2.4 se describieron brevemente los algoritmos de detección tanto para el rostro como para los ojos usando características rectangulares, a continuación se explica con más detalle en que consiste y la forma en que fueron implementados.

El mecanismo que se emplea para detectar la cara y otras partes del rostro es semejante tanto para una imagen fija como para un video, ya que el video se puede analizar como una serie de imágenes o cuadros que son procesados en tiempo real, por lo que la eficiencia del algoritmo a implementar es crucial para el éxito de este trabajo, ya que la finalidad es lograr la detección en un video capturado por una cámara.

A continuación se justificará la utilización de características rectangulares y algunos datos importantes para una comprensión más a fondo.

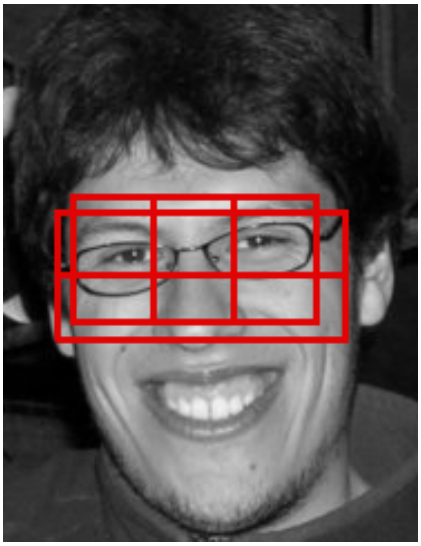

Figura 4.1: Ejemplos de características del rostro, la zona de los ojos es más oscura que la de las mejillas y los ojos son más oscuros que la nariz.

## 4.1. Características rectangulares

Para reconocer algún tipo de objeto es menester saber cuáles son las características visuales que permiten distinguirlo del resto. Por ejemplo una pelota; la pelota es redonda, la mayoría de las veces es de un color claro, y si la queremos buscar en una cancha de fútbol, una característica es el contraste con respecto al color del pasto (excepto cuando se juega con nieve). Lo mismo ocurre con el rostro, se pueden identificar algunas características tales como el contraste que existe entre la zona de los ojos con respecto a las mejillas ya que estas últimas son de un color más claro. Lo mismo ocurre con los ojos que son más oscuros que la zona de la nariz 4.1.

Para la detección de la cara y los ojos se buscaron características rec-

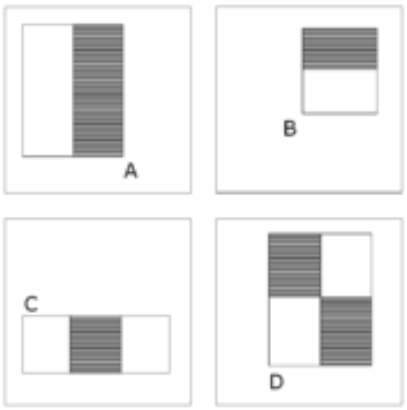


Figura 4.2: Ejemplos de características rectangulares utilizadas para la detección del rostro. A y B son los rectángulos de dos características, C el de tres características y D el de cuatro características.

tangulares principalmente por dos razones. Las características rectangulares actúan como un sistema que recopila conocimiento, cosa que es difícil de hacer usando una cantidad finita de información. Otra razón de peso es la mejor velocidad de detección con respecto al sistema de detección basado en intensidad de píxeles. Otra ventaja de esta técnica es que no necesita la ayuda de una luz infrarroja externa como se utilizó en [13], [8], [4]. En las figuras 4.2,4.3 se ven las características rectangulares utilizadas.

Las características rectangulares son similares a las Haar-like [14], la idea consiste en trabajar con regiones rectangulares y la suma de píxeles al interior de estos, todas las regiones son formadas a partir de rectángulos colindantes. En un comienzo se utilizaron tres tipos de características rectangulares para detectar el rostro. El rectángulo de dos características es formado como la diferencia de la suma de ambas regiones rectangulares, ambas regiones son

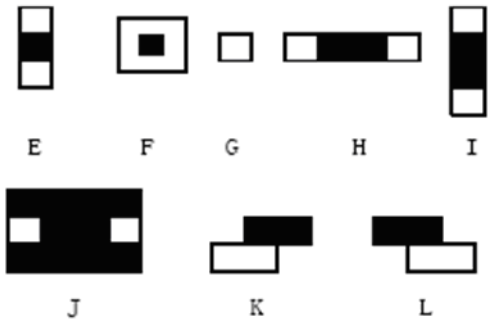


Figura 4.3: Características rectangulares utilizadas para detectar el ojo. Son un poco más complejas que los utilizadas para el rostro pero igualmente efectivas.

del mismo tamaño y pueden estar en posición vertical u horizontal (ver figura 4.2). El rectángulo de tres características es representado como la diferencia entre la suma de los rectángulos exteriores menos la suma del rectángulo del centro. El de cuatro rectángulos es calculado como la diferencia entre los rectángulos de las diagonales.

## 4.2. Imagen integral

La suma de píxeles que es utilizada para calcular las características rectangulares presentadas en la sección anterior pueden calcularse rápidamente usando una representación distinta a la representación de intensidad de píxeles que viene almacenada en la imagen original, esta nueva representación tiene el nombre de imagen integral, la que permite calcular la suma de píxeles para cada rectángulo en tiempo constante independiente del tamaño del rectángulo. El valor de la imagen integral en la posición $(x, y)$ es equivalente

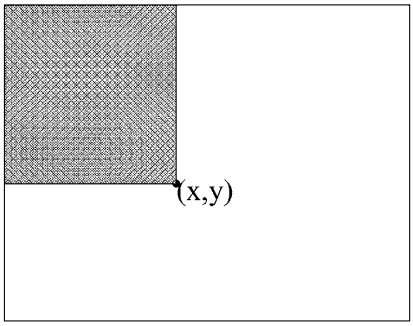


Figura 4.4: Representación gráfica de la imagen integral en la posición $(x, y)$.

a la suma de píxeles de la zona superior izquierda incluyendolo a $(x, y)$:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{4.1}$$

donde $ii(x, y)$ es la imagen integral e $i(x, y)$ es la intensidad del píxel de la imagen original. La fórmula para el cálculo de la imagen integral se plantea a continuación.

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{4.2}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{4.3}$$

Donde $s(x, y)$ representa la suma acumulada sobre la columna, de esta forma en una sola pasada se puede calcular la imagen integral.

Para encontrar la suma de píxeles para cada rectángulo se necesitan sólo cuatro referencias de la imagen integral(ver figura 4.5). Por lo que el cálculo de la suma de píxeles viene dada en la siguiente fórmula.

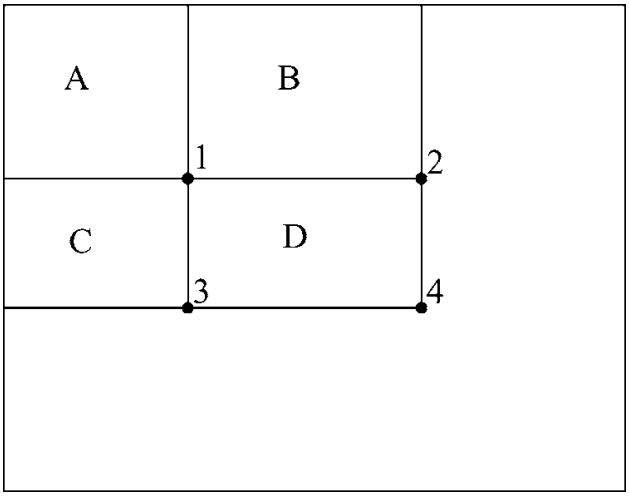


Figura 4.5: Calculando la suma de píxeles usando la imagen integral.

$$D = 1 + 4 - 2 - 3 \tag{4.4}$$

## 4.3. Introducción al algoritmo de detección

Los algoritmos de detección convencionales de la cara y de cualquier tipo de objetos hacen un barrido sobre imagen original, de igual manera ocurre esto con las características rectangulares. Pero los otros algoritmos deben realizar el barrido a distintas escalas de la misma imagen, en cambio las características rectangulares pueden ellas mismas escalarse para detectar caras y ojos de distinto tamaño.

Una sub-ventana recorre la imagen en su totalidad, esta sub-ventana en cada posición debe determinar si lo que está al interior de ella es una cara o no,

o es un ojo o no. Las características rectangulares son calculadas al interior de esta sub-ventana y dependiendo del valor arrojado por las características rectangulares se determina si esa sub-ventana representa a estos objetos.

La ventaja empírica que tienen las características rectangulares es que por construcción permiten un aprendizaje, lo que será detallado más a fondo en las próximas secciones, a eso se le debe sumar como una ventaja la eficiencia que tiene para calcular cada característica rectangular y también el no necesitar escalar a distintos tamaños la imagen original. Estos rasgos marcan una diferencia importante al comparar este método de detección con otros que pudieran parecer más complejos y eficaces que este.

## 4.4. Construcción del detector

En la sección previa se explicó el procedimiento que lleva a cabo el algoritmo de detección junto con una justificación. Se explicó que una sub-ventana recorre la imagen clasificándose como cara en caso de ser así y no cara en caso contrario, el mismo procedimiento se aplica para el ojo. La sub-ventana depende del resultado de las características rectangulares para su clasificación.

En esta sección se detallará como se construyó el detector de caras y rasgos faciales (ojos), la detección de los ojos tiene una implementación muy similar a lo que ocurre con el rostro ya que la única diferencia son las características rectangulares aplicadas. Además se denotará una terminología de los tipos

de errores. Se explicará el algoritmo de aprendizaje de las caras y no caras, que determinan las características rectangulares que se van a utilizar y una optimización sobre ellos usando reglas de decisión. También está incluida en esta sección, la manipulación de las características rectangulares a distintas escalas.

### 4.4.1. Errores en la detección

Existen dos clases de errores al analizar una sub-ventana. *Falso positivo* es el error cuando el detector determina que la sub-ventana es una cara u ojo cuando en verdad no lo es. *Falso negativo* es el error cuando el detector determina que la sub-ventana no es una cara cuando en verdad sí lo es.

Preferentemente se busca minimizar ambos tipos de error en el detector, pero por construcción del detector se optará por tender a cero los *falsos negativos* y a partir de ahí, minimizar los *falsos positivos*.

### 4.4.2. Funciones clasificadoras

Dentro de una sub-ventana de 24x24 píxeles hay alrededor de 160.000 posibles características rectangulares sólo para detectar el rostro, sería imposible calcular semejante cantidad para cada sub-ventana en tiempo real. Por ende se debe buscar una pequeña cantidad de características rectangulares para clasificar correctamente una sub-ventana. El objetivo consiste en encontrar estas características rectangulares.

Antes de explicar como se encuentran las características rectangulares se define un clasificador débil como:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{si } pf(x) < p\theta \\ 0 & \sim \end{cases} \tag{4.5}$$

Donde $x$ corresponde a la sub-ventana, $f$ una característica rectangular, $p$ la polaridad para determinar la desigualdad y $\theta$ es el valor umbral. Esta es la función clasificadora más básica, ya que por si sola, esta función clasificadora no tiene altas tasas de detección y es una primera aproximación al clasificador definitivo o clasificador fuerte que finalmente será una combinación de estas funciones más débiles.

Para encontrar las características rectángulares y su respectivos umbrales se pasa por un proceso de entranamiento, muy similar a Adaboost. Para ello se necesitan un conjunto de imágenes positivas (caras y ojos) y otras negativas (no caras y no ojos). El proceso se explica en la siguiente sección.

### 4.4.3. Adaboost

El algoritmo Adaboost [15] entrena las características rectangulares que se van a utilizar en el detector. Para el entrenamiento se utiliza un conjunto de imágenes etiquetadas positiva y negativamente.

El entrenamiento consiste en buscar una secuencia de características rectangulares con su respectivo umbral, donde en cada etapa de la secuencia se consideran los errores de la etapa anterior, para eso se le asignan pesos a las

imágenes que fueron clasificadas erróneamente por la función clasificadora. Además a cada etapa se le asigna un factor de ponderación dependiendo de que tan buena sea la función clasificadora de la etapa. Los entrenamientos para los ojos y cara se realizan de forma independiente, y para ambos es igual.

El algoritmo de entramiento que encuentra $T$ funciones clasificadoras procede de la siguiente manera:

- Dado un conjunto de imágenes $(x_1, y_1), ..., (x_n, y_n)$ donde $y_i = 0, 1$ se etiqueta la imagen como negativa o positiva respectivamente.
- Se inicializan los pesos asignados a cada imagen $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0, 1$ respectivamente y donde $m$ corresponde al número de imágenes negativas y $l$ el número de imágenes positivas.
- Para $t = 1, ..., T$
  - Se normalizan los pesos asignados a cada imagen de entrenamiento $w_{t,i} \leftarrow \frac{w_{t,i}}{Z_t}$ donde $Z_t = \sum_{j=1}^{n} w_{t,j}$
  - Se busca la característica rectangular y su respectivo umbral que minimice el error de la función considerando la ponderación de los pesos a cada imagen. ($\epsilon_t < 0{,}5$)

$$\epsilon_t = \min_{f,p,\theta} \sum_i |h(x_i, f, p, \theta) - y_i| \tag{4.6}$$

  - Se define $h_t(x) = h(x, f_t, p_t, \theta_t)$ donde $f_t, p_t, \theta_t$ minimizan $\epsilon_t$

- Se actualizan los pesos para cada imagen

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \tag{4.7}$$

donde $e_i = 0$ si la imagen $i$ fue correctamente clasificada, $e_i = 1$ si se equivocó. $\beta_t = \frac{e_t}{1-e_t}$

- Finalmente el clasificador fuerte es:

$$C(x) = \begin{cases} 1 & \text{si } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \sim \end{cases} \tag{4.8}$$

donde $\alpha_t = \log \frac{1}{\beta_t}$

En 4.6 se debe buscar la característica rectangular con el umbral y polaridad que tenga menor error ponderado (con el peso de los ejemplos) durante el entrenamiento.

Se va probando cada combinación posible de característica rectangular que esté al interior del cuadrado de 24x24 píxeles. Para esa característica rectangular se debe encontrar el umbral $\theta$ y polaridad p que minimicen esa característica rectangular.

Para ello primero se ordenan de menor a mayor los valores retornados por la característica rectangular de todas las imágenes de entrenamiento, y en una pasada se puede encontrar el umbral que entrega el mínimo para esa característica rectangular; por cada elemento al ir recorriendo la lista ordenada se evalúan cuatro sumas: $T^+$ como la suma de todos los ejemplos

positivos, $T^-$ la suma de todos los ejemplos negativos, $S^+$ como la suma de las imágenes positivas hasta la posición donde se está recorriendo la lista ordenada y $S^-$ como la suma de imágenes negativas hasta la posición donde se está recorriendo la lista ordenada.

Para cada posición de la pasada descrita en el párrafo anterior el error corresponde a:

$$e = \text{mín}(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \tag{4.9}$$

O dicho de otra forma es la suma entre los *falsos negativos* ya que su valor es menor (mayor) al umbral y los *falsos positivos* dado que su valor es mayor (menor) al umbral en el caso que la polaridad es positiva (negativa).

De esta forma se encuentran las características rectangulares que después son utilizadas para detectar las caras y ojos en la imagen (video).

### 4.4.4. Escalamiento de las características rectangulares

En la sección anterior entrenamos un detector que encuentra caras en una sub-ventana de 24x24 píxeles, lo que limita la detección de caras que tengan un tamaño mayor. Sin embargo existen dos formas de detectar caras de mayor tamaño.

La primera de ellas es ir achicando la imagen, una imagen de 400x300 se tiene que achicar 13 veces para detectar caras de mayor tamaño usando las

| 24 | 30 | 38 | 48 | 60 | 76 | 90 | 114 | 144 | 180 | 228 | 280 | 342 | 432 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Cuadro 4.1: Tamaño de las sub-ventanas, los tres primeros forman el resto de la tabla mediante ponderaciones enteras.

características rectangulares encontradas para sub-ventanas de 24x24, esto aumenta el tiempo de procesamiento del algoritmo de detección. La otra opción es escalar las características rectangulares y realizar el barrido con sub-ventanas más grandes.

En [11] se propone ir escalando las sub-ventanas en un factor de 1,25 a partir de 24x24. La posición de las características rectangulares es escalada en el mismo factor 1,25 que la sub-ventana. El problema al trabajar con píxeles es que hay que redondear las posiciones escaladas, esto afecta al resultado del clasificador pues ya no es la misma característica rectangular calculada en la escala original. Para evitar el redondeo de la posición de los píxeles se propone en éste trabajo tener una factor escalar variable que fluctúa levemente de un tamaño a otro, la fluctuación es alrededor de 1,25. Se trabajó con una base de características rectangulares en tres diferentes tamaños 24x24,30x30,38x38 y así formar las siguientes tamaños como una ponderación entera de las bases señaladas anteriormente, solucionando de esta manera el problema de redondear la posición de los píxeles.

### 4.4.5. Cascada de detectores

En la sección anterior se mostró como calcular una secuencia de características rectangulares (se detectaron alrededor de 100), luego el detector formado con estas características rectangulares procede a barrer con una sub-ventana la imagen. Para cada posición durante el barrido se calcula el valor entregado por todas las características rectangulares.

Para una ventana de 380x380 se deben verificar alrededor de 1,000,000 de sub-ventanas y si para cada sub-ventana se tienen que calcular cerca de 100 características rectangulares en total habría que calcular 100,000,000 de características rectangulares. Es necesario reducir esa cantidad para lograr detectar correctamente en tiempo real el video.

Lo que se propone en [11] es crear un algoritmo de cascada con las características rectangulares para no tener que que calcular todas las características rectangulares para todas las ventanas. La sub-ventana se va evaluando por etapas, en las primeras etapas se calculan las características rectangulares más simples que descarten la mayor cantidad de sub-ventanas, para luego en las etapas posteriores descartar las sub-ventanas más difíciles que requieran más cálculos.

En la figura 4.6 se ve más claramente el algoritmo en cascada.

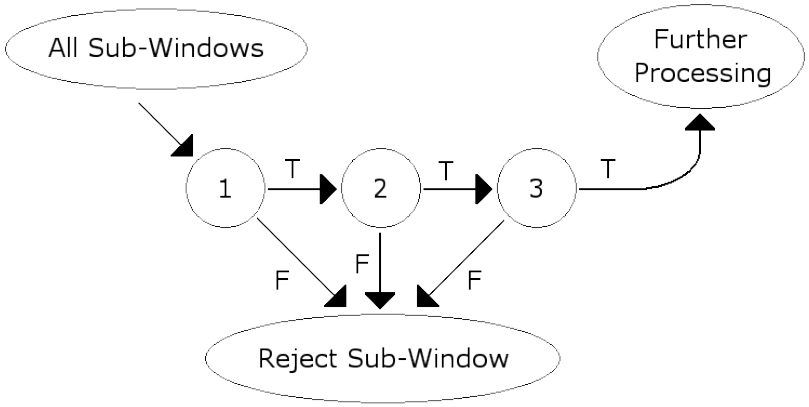


Figura 4.6: Algoritmo de detección en cascada, en cada etapa se van descartando las sub-ventanas que se consideran negativas y las que llegan al final son clasificadas como caras u ojos.

## 4.5. Implementaciones realizadas

En esta sección se explicará el software implementado durante el desarrollo de la memoria.

En una primera parte se trabajó con el procesamiento de video, leyendo los cuadros (frames) enviados desde la capturadora de video mediante la API video4linux, ya que la finalidad de la memoria es la detección en tiempo real de los rostros y ojos de la gente que está al frente de una cámara. Por eso se consideró en un inicio esta tarea junto con la de visualizar en la pantalla las imágenes capturadas.

Además se trabajó mucho con los píxeles en duro, ya que los cuadros obtenidos por la digitalización de la imagen enviada por la capturadora de video los entrega como un arreglo (tira) de bytes. Donde cada byte representa

la intensidad ya sea del píxel o una componente de este.

Se implementaron también las técnicas y los algoritmos vistos en la sección 4.4, tales como la imagen integral, el entrenador de características rectangulares, el clasificador en cascada y el barrido de la imagen con sub-ventana.

Se hicieron pruebas de concepto para probar la eficacia de la imagen integral, en particular la detección de zonas oscuras de video en tiempo real.

Para la construcción del detector se utilizaron bibliotecas que cargaban imágenes en memoria y que también ayudaban a desplegar imágenes en pantalla.

Se intentó programar en Python lo mismo que se implementó en un inicio en C++ como una alternativa de desarrollo las que finalmente no fueron satisfactorias.

Se utilizaron dos bases de datos para el entrenamiento del detector. La primera fueron caras y ojos recopilados directamente desde la Web y la segunda fue la base de datos de imágenes de Viola&Jones utilizada en [11].

### 4.5.1. Capturar video

Luego de investigar sobre la API video4linux probando ejemplos encontrados en internet, se comprendió su funcionamiento. Una de los detalles más importantes sobre video4linux es la utilización de los frames almacenados en la memoria de la capturadora.

Las capturadoras de video para lograr un *frame rate* de 30fps tienen un buffer donde guardan una cierta cantidad de frames. En La API video4linux

se hacen la peticiones de los cuadros asíncronamente, es decir, se hace la petición, la API retorna, pero el arreglo de bytes en memoria no está necesariamente con todo el contenido de la imagen, por ende se realiza otra petición de sincronización, ahí la API espera que el arreglo de bytes este completo y en ese momento retorna. Este proceso descrito logra un *frame rate* de 30 fps aproximadamente, sólo si se emplea de forma correcta.

Bajo el supuesto que el buffer contiene ocho frames, la forma de iterar sobre el buffer es la siguiente:

- Inicialmente se piden todos los frames del buffer
- $i = 0$
    1. Se sincroniza el frame $i$, de lo contrario el frame puede estar inconcluso.
    2. Se procesan los píxeles obtenidos desde el arreglo de bytes entregados por la capturadora de video.
    3. Se pide el frame $i$ (el de la siguiente vuelta sobre el buffer).
    4. $i = (i + 1)\,\%8$ (Avanza al frame siguiente, si $i = 8$ se da vuelta el buffer).
- El ciclo concluye con el fin del programa

Esta es la manera de conseguir una tasa de 30fps para la capturadora de video, de lo contrario se obtienen tasas menores, empeorando la calidad de la imagen y por ende la detección caras.

### 4.5.2. Bibliotecas para cargar imágenes y desplegar video

Para desplegar en pantalla los frames entregados por la API video4linux, se utilizó la biblioteca SDL en particular la clase SDL_Surface que guarda el frame como una textura que, posteriormente, usando OpenGL, puede verse como un video normal.

Para mostrar las imágenes capturadas se debe agregar solamente un paso al algoritmo presentado en 4.5.1, esto corresponde a crear una textura en OpenGL. Entonces el algoritmo quedaría de la siguiente forma.

- Inicialmente se piden todos los frames del buffer
- $i = 0$
    1. Se sincroniza el frame $i$, de lo contrario el frame puede estar inconcluso.
    2. Se procesan los píxeles obtenidos desde el arreglo de bytes entregados por la capturadora de video y se crea la textura en SDL.
    3. Se genera la textura en OpenGL con el método *glTexImage2D*.
    4. Se pide el frame $i$ (el de la siguiente vuelta sobre el buffer).
    5. $i = (i + 1)\,\%8$ (Avanza al frame siguiente, si $i = 8$ se da vuelta el buffer).
- El ciclo concluye con el fin del programa

Figura 4.7: Aplicación 3D desarrollada en OpenGL visualizando el canal 13.

De esta forma se puede visualizar en una pantalla los frames entregados por la capturadora de video, con la sensación de estar viendo la televisión en modo normal.

Un ejemplo de lo anterior es la figura 4.7, lamentablemente es una imagen fija pero al menos observando el cubo se pretende probar la idea de ver un video en 3D. En la esquina de las caras del cubo se ve el logotipo de canal 13.

### 4.5.3. Procesamiento de píxeles

La paleta de colores escogida inicialmente para la captura de video fue RGB. La tira de bytes entregada por la API video4linux consiste en tres bytes consecutivos para definir el color de cada píxel. Los píxeles vienen ordenados por filas partiendo de la fila superior de la imagen.

El trabajo con los píxeles directamente es a bajo nivel como se presume en el párrafo anterior. Para obtener la intensidad de los píxeles que es necesaria para la imagen integral fue necesario en primera instancia promediar las componentes RGB de cada píxel, más adelante en el transcurso de la memoria, se trabajó con el formato YUV donde la primera componente corresponde a la intensidad del píxel, evitando la necesidad de promediar las componentes como era al inicio.

Las sub-ventanas detectadas dentro de cada cuadro se dibujaban también a bajo nivel. Cada píxel (en RGB) correspondiente a las aristas de la subventana se asignaba todo el color de una componente del píxel y las demás se dejaban en cero.

Las imágenes de entrenamiento finalmente están en escala de grises (1 Byte por píxel).

### 4.5.4. Implementación imagen integral

La imagen integral permite calcular la suma de píxeles en tiempo constante para todo tipo de rectángulos. Para ello se hace referencia a cuatro valores de la imagen integral y se realizan un par de operaciones aritméticas sobre ellos.

Cada posición de la imagen integral resume la suma de todos los píxeles de la parte superior izquierda a la posición del píxel.

Hay un pequeño detalle al calcular la suma de píxeles al interior de un rectángulo mediante la imagen integral. El detalle consiste en determinar la

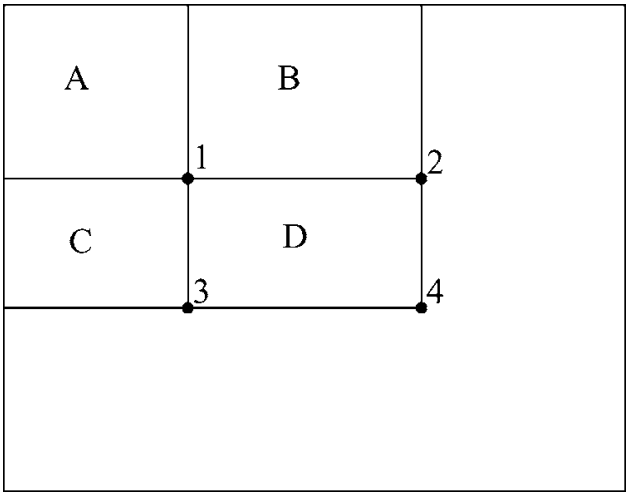


Figura 4.8: Calculando la suma de píxeles usando la imagen integral.

posición de los vértices del rectángulo al que se va a calcular la suma de píxeles. En la figura 4.8 está la idea para calcular la suma de píxeles.

Anteriormente se mostró la formula con la que se va a calcular la suma de píxeles al interior del rectángulo D la que se presenta nuevamente a continuación.

$$D = 4 - 2 - 3 + 1 \tag{4.10}$$

Esto es ya que: $4 = A + B + C + D$, y para encontrar la suma de píxeles en el rectángulo $D$ se le resta a esa ecuación la parte superior, es decir, el valor en la posición 2 equivalente a $A + B$, para quitar C se resta el valor en la posición 3 que es equivalente a $A + C$, finalmente se agrega la suma de píxeles en $A$ ya que se quitó dos veces. Por eso se suma el valor en 1.

El problema nace al escoger los vértices del rectángulo ya que si escogemos

los mismos vértices se pierde la primera fila y la primera columna de píxeles. Para evitar eso, se trasladan la posición 1,2 y 3. La posición 1 se traslada un píxel a la izquierda y otro hacia arriba, la posición 2 se traslada un píxel a arriba y la posición 3 se traslada un píxel a la izquierda. De esta forma se obtiene la suma fidedigna de píxeles al interior del rectángulo D de la figura 4.8.

La estructura de la imagen integral es levemente distinta a la de intensidad de píxeles de la imagen original. La primera diferencia es la precisión numérica ya que la intensidad de píxeles en cada posición de la imagen es de 1 Byte. En cambio para la imagen integral cada posición corresponden a 4 bytes. La otra diferencia es que la imagen integral va a constar de una fila y una columna más de valores ceros lo que permite calcular la suma de píxeles en el borde superior e izquierdo como cero.

### 4.5.5. Detección zonas oscuras en tiempo real

Para testear la imagen integral y la suma de píxeles, se implementó un detector de zonas oscuras. Se buscaba el cuadrado más oscuro de tamaño 40x40 para cada frame.

Para realizar dicho procedimiento, se siguió el mismo procedimiento de captura, procesamiento y despliegue del video capturado que fue explicado en secciones previas. En resumidas cuentas el procedimiento fue:

- Inicialmente se piden todos los frames del buffer

- $i = 0$

    1. Se sincroniza el frame $i$, de lo contrario el frame puede estar inconcluso.
    2. Se procesan los píxeles obtenidos desde el arreglo de bytes entregados por la capturadora de video y se crea la textura en SDL.
    3. Se calcula la imagen integral con respecto al frame capturado.
    4. Se realiza un barrido para cada sub-ventana de tamaño 40x40 y se calcula la suma de píxeles al interior del sub-ventana. De todas las sub-ventanas se escoge la mayor valor.
    5. Se dibuja en la textura un la sub-ventana (cuadrado de tamaño 40x40) en la posición seleccionada en el paso anterior.
    6. Se genera la textura en OpenGL con el método *glTexImage2D*.
    7. Se pide el frame $i$ (el de la siguiente vuelta sobre el buffer).
    8. $i = (i + 1)\,\%8$ (Avanza al frame siguiente, si $i = 8$ se da vuelta el buffer).

- El ciclo concluye con el fin del programa

Las figuras 4.9, 4.10 muestran el correcto funcionamiento de detección de zonas oscuras. Queda comprobado en la práctica la veracidad de la integral de imagen y la suma de píxeles.

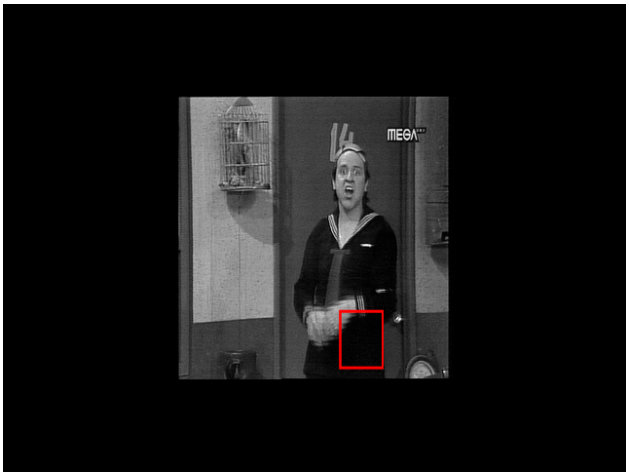


Figura 4.9: Screenshot zona oscura en video (canal mega).

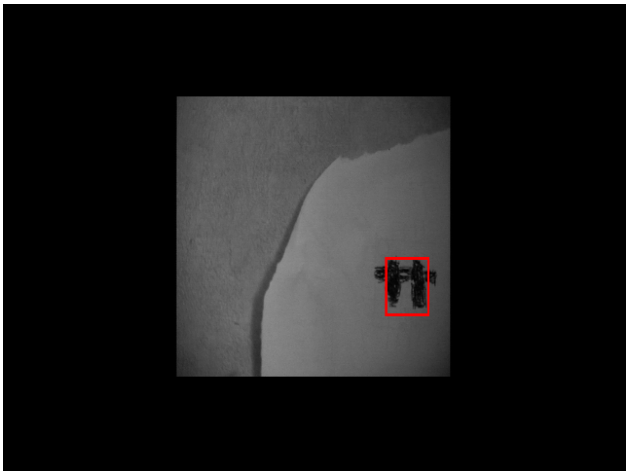

Figura 4.10: Screenshot zona oscura en video (cámara VISCA).

### 4.5.6. Implementación en Python

Se realizó una implementación del algoritmo de detección de caras en el lenguaje de programación Python. En Python usando la biblioteca Pygame es posible cargar imágenes en distintos formatos (PNG, JPG, GIF, PGM, etc.).

El autor tenía conocimiento previo en este lenguaje de programación, por lo que tuvo un corto tiempo de adaptación para implementar los algoritmos de entrenamiento y detección. Otra ventaja era la facilidad para el autor al momento de programar.

Lamentablemente los tiempos de ejecución de los algoritmos de entrenamiento y detección de caras es mucho mayor en Python que en la implementación realizada en C++. El manejo de listas (arreglos) es menos eficiente y al ser un lenguaje interpretado no ayuda en la velocidad de ejecución que es necesaria para llevar a buen puerto este proyecto.

En ese momento el alumno encontró la biblioteca SDL_image que también permite cargar imágenes en distintos formatos. Como consecuencia de ello el alumno decidió continuar el desarrollo en C++, tanto para la búsqueda de características rectangulares como para la detección de las caras y ojos en imágenes.

### 4.5.7. Implementación características rectangulares

A continuación se detallarán las características rectangulares para la detección de la cara y de los ojos.

Cabe destacar la implementación de cuatro características rectangulares para detectar el rostro: (Entre paréntesis se encuentra el nombre abreviado de la característica rectangular):

- Dos rectángulos horizontales (Rf2h)
- Dos rectángulos verticales (Rf2v)
- Tres rectángulos verticales (Rf3v)
- Cuatro rectángulos, como matriz de 2x2 (Rf4)

Se implementaron ocho características rectangulares para la detección de los ojos:

- Tres rectángulos verticales (Rf3v)
- Un cuadrado central (Rfcc)
- Un solo cuadrado (Rf1c)
- Tres rectángulos horizontales con rectángulo interior extendido (Rf3hx)
- Tres rectángulos verticales con rectángulo interior extendido (Rf3vx)
- Un rectángulo con dos rectángulos internos (Rf2i)

- Dos rectángulos no 100 % colindantes a la derecha (Rf2cd)
- Dos rectángulos no 100 % colindantes a la izquierda (Rf2ci)

Cada una de estas características rectangulares contienen las posiciones de los rectángulos que forman la característica rectangular. Destaca el método que retorna el valor de la diferencia de la suma de los rectángulos, que es posible de calcular directamente con las variables almacenadas pasándole como entrada la imagen integral.

Es importante recalcar que todos los rectángulos de las distintas características rectangulares mantienen la misma proporción, por eso conociendo los valores extremos del rectángulo se pueden conocer las posiciones intermedias para formar los rectángulos internos de la característica rectangular.

Estas clases son la base de toda la memoria, pues en una primera etapa se buscan las características rectangulares durante el entrenamiento basado en imágenes positivas y negativas, para luego utilizar esas características rectangulares en la detección del rostro y los ojos durante el barrido de la sub-ventana a la imagen.

### 4.5.8. Entrenamiento

Para el entrenamiento de las características rectangulares en un comienzo se utilizaron alrededor de 500 imágenes de las cuales eran 200 caras y 300 no caras, de tamaño 24x24 píxeles. Las imágenes fueron recopiladas desde internet (google, facebook, etc) y de algunas fotos familiares, lamentablemente el

proceso no detectaba un gran número de características rectangulares ya que al ir variando los pesos de las imágenes llegaba a un punto de estancamiento donde no había más variación. Como mejora se realizó el entrenamiento de las características rectangulares con la base de datos de Viola&Jones [16] que es de alrededor de 5000 caras y 10000 no caras, logrando de esa forma encontrar un mayor número de características rectangulares.

De la misma forma se realizó el entrenamiento de los ojos donde se utilizaron 200 ojos y 400 no ojos para el entrenamiento.

Los entrenamientos se realizaron en forma independiente, ya que se usan diferentes características rectangulares.

El entrenamiento consistió en las siguientes etapas:

1. Se cargan todas las imágenes de entrenamiento como texturas. Etiquetandolas positiva y negativamente para diferenciarlas más adelante.

2. Se calcula la imagen integral para todas las texturas del punto anterior.

3. Para $w1 = 1 \ldots (24 - 8), h1 = 1 \ldots (24 - 8), w2 = w1 \ldots 24, h2 = h1 \ldots 24$, donde $w1, h1, w2, h2$ representan todos los posibles rectángulos de tamaño mayor a 8x8 que entran en una sub-ventana de 24x24 píxeles.

   *a*) Se reinicia una lista donde se almacenarán los valores entregados por la característica rectangular evaluados para las imágenes.

   *b*) Para todas las imágenes.

1) Se instancia la característica rectangular correspondiente, se determinan las posiciones como: $(w1, h1)$,$(w2, h2)$.

2) Se almacena el valor entregado por la característica rectangular en la lista.

*c*) Se calcula el valor umbral y el error correspondiente a los valores almacenados en la lista. Para encontrarlos se ordena de menor a mayor la lista de los valores calculados por la característica rectangular, luego se recorre la lista una lista donde el umbral corresponde a un valor intermedio a la posición donde el error es mínimo.

*d*) Se compara el error obtenido con esta característica rectangular con el mínimo parcial. En caso de ser menor se deja como nuevo mínimo.

4. Se guarda la característica rectangular, el umbral, la polaridad y el error en un archivo de texto, que es utilizado por el clasificador definitivo para generar la secuencia de clasificadores débiles.

De esta forma se obtienen las características rectangulares utilizadas por el detector.

### 4.5.9. Escalamiento características rectangulares

En la sección anterior se detalló el algoritmo para encontrar características rectangulares con imágenes de tamaño 24x24 píxeles. Las sub-ventanas

entrenadas son también de tamaño 24x24. Análogamente se procede para encontrar las características rectangulares de tamaños 30x30 y 38x38 píxeles. Sólo se necesitan escalar las imágenes de entrenamiento. En esta memoria se usó el software ImageMagick para el escalamiento de las imágenes.

Con esas base de tres tamaños se implementará el algoritmo para escalar a todas las escalas las características rectangulares antes mencionadas.

Se justificó previamente la utilización de una base de características rectangulares por el problema de redondeo de decimales al momento de escalar los rectángulos. Ya que originalmente se había trabajado con una escala fija de 1,25 entre un tamaño y el siguiente. Ahora al trabajar con una escala variable pero cercana a 1,25 los tamaños de las sub-ventanas son siempre múltiplos enteros de 24, 30 y 38. De esta forma también los posiciones de los rectángulos serán múltiplos enteros.

El algoritmo para escalar a distintas escalas es el siguiente:

- Todas las imágenes que fueron usadas de entrenamiento tienen el mismo tamaño que la sub-ventana a la que se quieren escalar las características rectangulares.

- Se encuentran los archivos con las características rectangulares de las escalas base.

    1. Se cargan todas las imágenes de entrenamiento como texturas. Etiquetando las caras y no caras para diferenciarlas más adelante.

2. Se calcula la imagen integral para todas las texturas del punto anterior.

3. Se leen todas las características rectangulares obtenidas desde los archivos de texto y se procede a escalar por el valor entero correspondiente.

4. Para cada característica rectangular $f$.

   *a*) Se calcula el valor de la $f$ para todas las imágenes y se almacenan en una lista.

   *b*) Se calcula el valor umbral y el error correspondiente.

   *c*) Se graba el resultado en un archivo de texto.

De esta manera se realiza el escalamiento de las características rectangulares para las sub-ventanas que se necesiten, el requisito básico es el listado de características recatangulares base y que las imágenes se encuentre en el tamaño destino para evaluar las características rectangulares nuevas.

Se resalta a esta altura que ya no es necesario realizar una búsqueda exhaustiva para los nuevos tamaños, con el problema asociado al tiempo de procesamiento que esto traería, ya que serían muchas más combinaciones. Sólo se recalculan los umbrales mediante la evaluación de las nuevas características rectangulares en las imágenes escaladas.


Figura 4.11: Resultado detección de caras para una imagen de ejemplo.

### 4.5.10. Barrido de sub-ventana y primeros resultados

Hasta el momento se ha desarrollado un clasificador de caras y ojos que permite determinar si una sub-ventana es una cara o no y si es un ojo o no. En esta sección se explicará como formar a partir de esto un detector de caras donde la entrada no es una sub-ventana si no que es una imagen completa. El resultado es la misma imagen pero las caras detectadas están enmarcadas con un cuadro rojo. La figura 4.11 muestra el funcionamiento preliminar.

Se detectaron dos caras de cuatro que estaban frente a la cámara, no se detectaron las caras que tenian anteojos de sol ni la cara que estaba de perfil.

El algoritmo de barrido de sub-ventana sobre la imagen es el siguiente:

1. Se leen desde un archivo de texto todas las características rectangu-

lares $f$ con sus respectivos umbrales $\theta$ y polaridades $p$. Esto incluye las características rectangulares en todas las escalas que el detector trabaja.

2. Se calcula la imagen integral a la imagen de entrada al detector.

3. Para todos los tamaños de sub-ventanas a barrer.

   *a*) Se barre la imagen con la sub-ventana actual.

   *b*) En caso favorable se agrega la sub-ventana a un listado de detecciones.

4. Se ajustan las sub-ventanas sobrepuestas.

5. Se dibujan las sub-ventanas en la imagen.

6. Se genera la textura con OpenGL.

En la figura 4.12 se aprecia sobreposición de las sub-ventanas detectadas por el clasificador. Hay varias formas de abordar este problema. Una de ellas es promediando las sub-ventanas que se encuentra cerca. En la figura 4.13 se puede apreciar el resultado de promediar los vértices de las sub-ventanas detectadas. Otra opción es dejar la sub-ventana que las contiene a todas. En la figura 4.14 se puede apreciar el resultado.

En este trabajo se escogió el promediar las sub-ventanas ya que se encontraba un ajuste más apropiado.

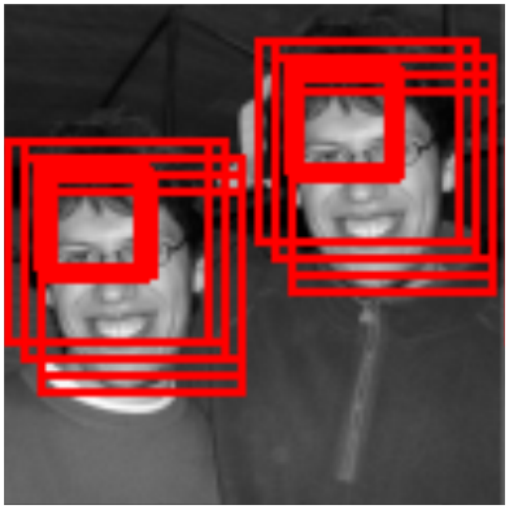

Figura 4.12: Ejemplo sub-ventanas sobrepuestas sin ajuste.

Figura 4.13: Ejemplo sub-ventanas sobrepuestas con ajuste promedio.

Figura 4.14: Ejemplo sub-ventanas sobrepuestas con ajuste de sub-ventana contenedora.

## 4.6. API detección cara y rasgos faciales

En las secciones previas, se hizo incapie en la teoría e implementación del algoritmo de detección del rostro y de los ojos dejando un poco de lado el tema de la interfaz de programación, el que es uno de los ejes de esta memoria.

En trabajos de título anteriores [8], [13], [4] se hicieron valiosos aportes en la detección del rostro y las pupilas de los ojos, creando aplicaciones innovadoras y eficientes, pero lamentablemente no permitían realizar nuevas aplicaciones en base a ese trabajo realizado. Por esa razón esta memoria debía dejar la puerta abierta a nuevos trabajos en el futuro.

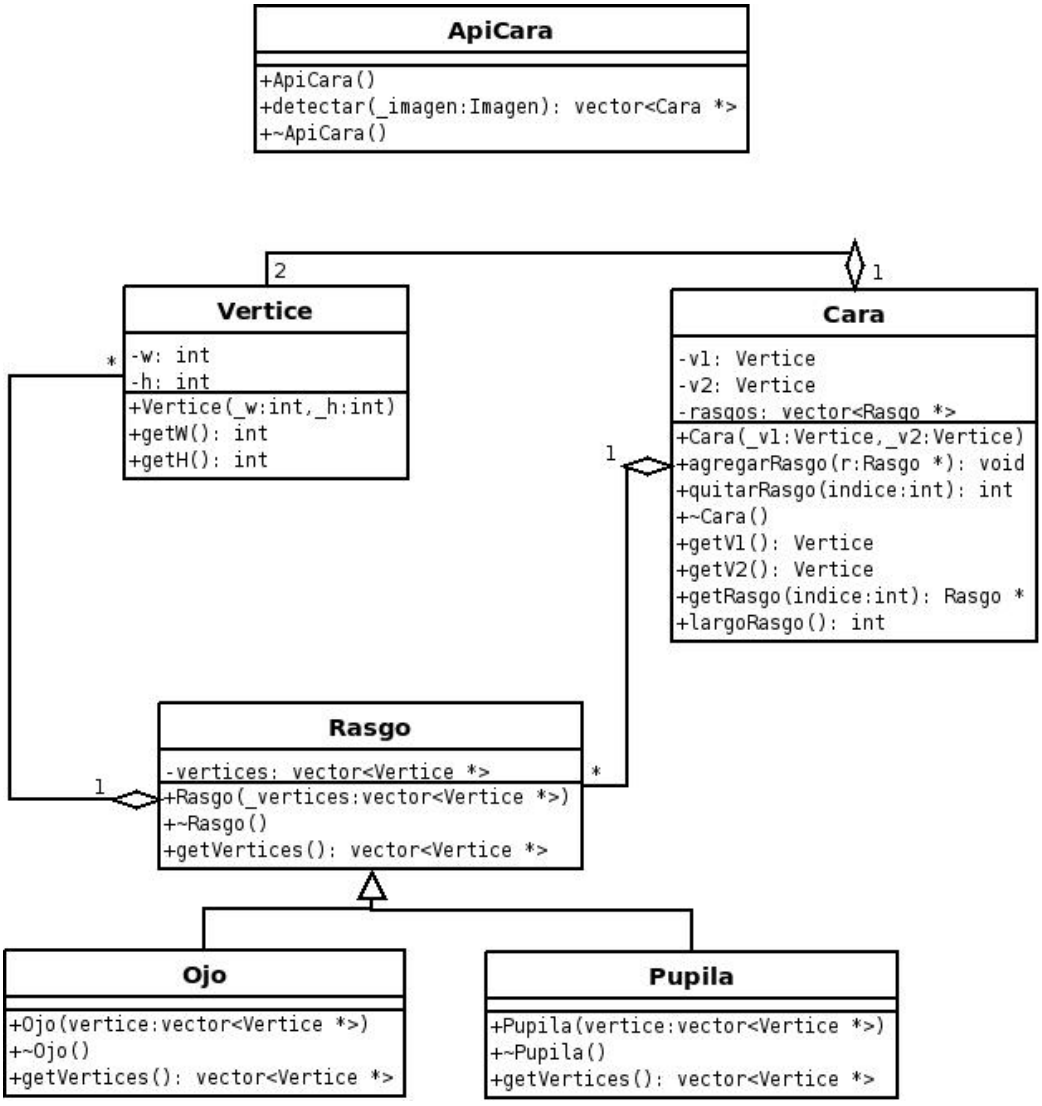


Figura 4.15: Diagrama con las clases visibles de la API desarrollada.

La interfaz de programación que se explica en esta sección permitirá que más adelante se puedan abordar nuevos proyectos que tengan como base la implementación realizada en el presente trabajo.

La API recibe como entrada una imagen y responde con una lista de caras, las que contienen a su vez una lista de rasgos. Se pensó en dejar abierta la posibilidad en el futuro de agregar nuevos rasgos tales como nariz, boca, orejas, etc. Todos los nuevos rasgos pueden ir almacenados en la lista de rasgos de cada rostro.

En la figura 4.15 se distinguen las clases utilizadas, se puede apreciar la extensibilidad del tipo Rasgo, ejemplo de ello es la presencia de la clase pupila, la que queda a modo de ejemplo para una futura implementación. Y da una idea de lo que se puede hacer con la información entregada por la API.

La información de las detecciones queda resumida en la ubicación de los vértices extremos (superior izquierdo e inferior derecho) del rectángulo que rodea el área detectada. Estos vértices son los que darán vida a las nuevas aplicaciones.

Destaco nuevamente la utilidad de esta API, funciona tanto para imágenes fijas como para video en tiempo real. Eso ya que el video puede interpretarse como una secuencia de cuadros (frames). La API desarrollada puede servir para un sinnúmero de nuevas aplicaciones y la limitante es solo la imaginación.

## 4.7. Aplicación de prueba de la API

Para testear la API implementada se desarrolló una aplicación que demostrará el potencial y utilidad de la interfaz de programación.

La aplicación consiste en recortar y almacenar los rostros que aparecen en un video, en particular las caras que son captadas por una cámara. Los recortes son almacenados en el disco duro del equipo, y se guardan ya sea mediante una interrupción (presionar una tecla), o bien cada cierto tiempo determinado por un temporizador ó finalmente se puede realizar el recorte al primer cuadro (frame) donde una cara es detectada, el software luego de realizar la primera detección no realiza más recortes si la detección siguiente sobrepone la detección previa.

Esta aplicación puede ser de gran utilidad para el control de acceso a recintos privados, ya que se puede hacer un seguimiento de las personas que han ingresado a un lugar.

La explicación del algoritmo de recortes de caras es la siguiente:

- Sea $f$ un frame capturado por la cámara.

    1. Se recupera el listado de caras que están en $f$ usando la API.
    2. Se compara con el listado de caras detectadas en el frame anterior, si no está sobrepuesta se almacena en disco. Si está sobrepuesta se descarta para el almacenamiento.

En los otros dos casos se almacenan las caras que se detecten en el mo-

mento de la interrupción.

En esta sección se demostró la utilidad de la API en un caso cotidiano como el control de acceso, dejando claro el potencial que la API tiene para futuras aplicaciones.

## 4.8. Trabajo a futuro

Algunas sugerencias para continuar el desarrollo de este proyecto son:

- Agregar nuevas características rectangulares que detecten otros rasgos de la cara (pupilas, nariz, boca, etc.) o detectar expresiones de la cara como lo que hace Sony con sus nuevas cámaras digitales (smile shutter [20]).
- Rediseñar el detector para hacerlo más extensible a agregar nuevas características rectangulares, y crear una interfaz que permita automatizar el entrenamiento de ellas.
- Lograr una mayor eficiencia en la implementación de barrido de subventanas, priorizando las ubicaciones cercanas a una cara u ojos ya antes detectadas.
- Desarrollar aplicaciones utilizando esta API que permitan la interacción de personas discapacitadas fisicamente con el computador. Por ejemplo el control del mouse usando las pupilas de los ojos.

# Capítulo 5

# Conclusiones

En este trabajo de memoria se realizó una investigación y la posterior implementación del algoritmo de detección de caras y ojos, tanto en imágenes fijas como en un video en tiempo real. Se desarrolló además una API que permite la creación de nuevas aplicaciones. Se demostró finalmente la utilidad de la API en una sencilla aplicación para el control de acceso a recintos.

A continuación un resumen más detallado del trabajo realizado durante la memoria:

- Se utilizaron diversas API tales como OpenGL, video4linux tanto para el despliegue de imágenes y video como para la obtención de video desde la tarjeta capturadora.
- Se implementó el algoritmo Adaboost para encontrar características rectangulares usando una base de datos de imágenes recopiladas desde la web.

- Se demostró la efectividad del algoritmo planteado en [11] por Viola&Jones barriendo la imágen (frame) usando sub-ventanas de distintos tamaños.

- Se implementó una API que permite detectar las caras y ojos en imágenes estáticas y en video en tiempo real.

- Se demostró el funcionamiento de la API durante las pruebas en la detección de caras en tiempo real usando una cámara de video (SONY VISCA) conectada a la capturadora del PC. Para ello se implementó el software de control de acceso.

# Bibliografía


[1] SpaceMouse, http://www.spacemouse.com/products/Classic.htm

[2] Phantom, http://www.sensable.com/haptics/products/phantom.htm

[3] Wii Remote, http://en.wikipedia.org/wiki/Wii_Remote

[4] Víctor González Toro, "Seguimiento visual para el control del mouse"

[5] Yuan-Pin Lin, Yi-Ping Chao, Chung-Chih Lin and Jyh-Horng Chen, "Webcam Mouse Using Face and Eye Tracking in Various Illumination Enviroments"

[6] Jonathon B. Hiley, Andrew H. Redekopp, and Reza Fazel-Rezai, "A Low Cost Human Computer Interface based on Eye Tracking"

[7] Islamic Azad, M.Atyabi, M.S.Khajeh Hosseini, and M.mokhtari, "The Webcam Mouse: Visual 3D Tracking of Body Features to Provide Computer Access for People with Severe Disabilities"

[8] Reynaldo Palma Darrigrande, "Animación de expresiones del rostro humano mediante la captura de textura facial"

[9] Chun-Hung Lin, Ja-Ning Wu, "Automatic Facial Feature Extraction by Genetic Algorithms"

[10] Mohamed Rizon, Tsuyoshi Kawaguchi, "Automatic Eye Detection Using Intensity and Edge Information"

[11] Paul Viola, Michael Jones, "Robust Real-Time Face Detection"

[12] Huchuan Lu, Wei Zhang, Deli Yang, "Eye Detection Based on Rectangle Features and Píxel-Pattern Based Texture Features"

[13] Mohamed Mohamed, "Animación en Tiempo Real del Rostro Humano Usando Dispositivos Móviles"

[14] Haar-like features,http://www.cs.ubc.ca/ pcarbo/viola-traindata.tar.gz `http://en.wikipedia.org/wiki/Haar-like_features`

[15] Adaboost, `http://en.wikipedia.org/wiki/Adaboost`

[16] Base de datos Viola&Jones, `http://www.cs.ubc.ca/~pcarbo/viola-traindata.tar.gz`

[17] Algoritmo Sobel, `http://es.wikipedia.org/wiki/Sobel`

[18] OpenGL, `http://www.opengl.org/`

[19] video4linux `http://es.wikipedia.org/wiki/Video4Linux`

[20] Smile shutter, `http://www.sony-latin.com/deteccion-rostros/`