UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

"Trabajo de grado previo a la obtención del Título de Ingeniero Electrónico y Telecomunicaciones"

TRABAJO DE GRADUACIÓN

Título del Proyecto:

"SISTEMA DE RECONOCIMIENTO DE EXPRESIONES FACIALES APLICADAS A LA INTERACCIÓN HUMANO-ROBOT"

AUTOR:

JUAN CARLOS TOAQUIZA AGUAGALLO

DIRECTORA:

ING. PAULINA VÉLEZ

Riobamba – Ecuador

AÑO 2016

Los miembros de Tribunal de Graduación del proyecto de investigación con el título: **SISTEMA DE RECONOCIMIENTO DE EXPRESIONES FACIALES APLICADAS A LA INTERACCIÓN HUMANO-ROBOT** presentado por: **Juan Carlos Toaquiza Aguagallo** y dirigida por: **Ingeniera Paulina Vélez.**

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite el presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

Ing. Aníbal Llanga
Presidente del Tribunal **Firma**

Ing. Paulina Vélez
Director del Proyecto **Firma**

Ing. Giovanny Cuzco
Miembro de Tribunal **Firma**

CERTIFICACIÓN DEL TUTOR

Certifico que el presente trabajo de investigación previo a la obtención del grado de Ingeniero en ELECTRONICA Y TELECOMUNICACIONES. Con el tema: **"SISTEMA DE RECONOCIMIENTO DE EXPRESIONES FACIALES APLICADAS A LA INTERACCIÓN HUMANO-ROBOT"** ha sido elaborado por el estudiante **Juan Carlos Toaquiza Aguagallo**, el mismo que ha sido revisado y analizado en un cien por ciento con el asesoramiento permanente de mi persona en calidad de Tutor por lo que se encuentran aptos para su presentación y defensa respectiva.

Es todo cuanto puedo informar en honor de la verdad

Ing. Paulina Vélez

AUTORÍA DE INVESTIGACIÓN

La responsabilidad del contenido de este Proyecto de Graduación, corresponde exclusivamente a **Juan Carlos Toaquiza Aguagallo** e **Ingeniera Paulina Vélez;** y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

Juan Carlos Toaquiza Aguagallo

C.I 060388206-9

AGRADECIMIENTO

Principalmente quiero agradecer la ayuda recibida por parte de mi familia. Sin ellos no hubiera sido posible realizar esta etapa de mi vida

Quiero agradecérselo también a mi Directora de proyecto Paulina Vélez que me ha ayudado continuamente en todo el desarrollo del mismo aportando nuevas ideas y distintos puntos de vista. Sobre todo la disponibilidad que ha tenido para resolverme cada una de las dudas que me iban surgiendo.

Por último, a toda la gente que estuvo cerca de mí en las buenas y en las malas, un enorme agradecimiento a ellos

Juan C Toaquiza A

DEDICATORIA

A mis padres y a mi familia que me ayudaron en todo el proceso, los quiero, gracias por la paciencia y el incondicional apoyo. Este triunfo es también de ustedes

Juan C Toaquiza A

INDICE GENERAL

.

INDICE DE FIGURAS

INDICE DE TABLAS

Resumen

En este proyecto se ha implementado un programa de reconocimiento de expresiones faciales para su uso en cualquier campo que requiera de la visión por computador. Se trabaja en la detección de las principales partes del rostro, buscando el camino más apropiado para el mismo; explicando de manera adecuada las líneas de código que intervienen en este proceso, tratando de no ser repetitivo en las mismas. En la detección de expresiones faciales se utilizó experimentación para lograr llegar a este objetivo.

Todo el programa está desarrollado con software libre, comenzando por el sistema operativo hasta las librerías para la visión por computador. Dichas librerías poseen la base de datos en archivos especiales del tipo xml que vienen en sus instaladores. El uso de las mismas en una programación se le conoce como HaarCascade, por lo cual este proyecto es desarrollado en este método.

Para las respectivas pruebas del sistema se utiliza la cámara web que tiene el computador, pudiendo funcionar sin ningún problema con una cámara externa conectada a este. Mencionadas pruebas son realizadas con el rostro del programador en distintos ángulos, con la apropiada distancia a la cámara y una adecuada influencia de la luz, no mucha, ni muy poca. Dependiendo de los resultados se sacaran las respectivas conclusiones y de esta manera catalogar a este proyecto como recomendable para su uso inmediato, o a su vez abre un campo de estudio para mayor desarrollo.

UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERIA

CENTRO DE IDIOMAS

Lic. Eduardo Heredia

08 de Agosto del 2016

SUMMARY

In this project, a program of recognition of facial expressions has been implemented for its use in any field that requires computer vision. It is working on the detection of the main parts of the face, searching the most appropriate way for thereof; explaining properly the lines of code that intervene in this process, trying not to be repetitive in thereof. Experimentation was utilized in detecting facial expressions in order to reach this goal.

The whole program is developed with free software, starting with the operating system until libraries for computer vision. These libraries have the database in special files of type XML that come in their installers. The use of thereof in a programming is known as HaarCascade, whereby this project is developed in this method.

The webcam of the computer was used for the respective tests of the system, being able to operate without any problem with an external camera connected to this. The aforementioned tests are done with the face of the programmer at different angles, with the proper distance to the camera and an adequate influence of light, neither too much nor too little. The respective conclusions will be drawn depending on the results, and thus classify this project as recommendable for its immediate use.

CENTRO DE IDIOMAS INSTITUCIONAL

UNIVERSIDAD NACIONAL DE CHIMBORAZO

RIOBAMBA - ECUADOR

DIRECCION

INTRODUCCIÓN

Al igual que los ojos y cerebro sirven al humano para entender el mundo que nos rodea. La visión por computador es un avance tecnológico que sigue métodos para adquirir, procesar, analizar y comprender las imágenes para producir información o datos que puedan ser tratados por un computador. Los datos pueden conseguirse a través de una secuencia de imágenes vista desde una cámara de video.

El ser humano captura la luz a través de los ojos, esta información viaja a través del nervio óptico hasta el cerebro donde se procesa. El primer paso del proceso es encontrar los elementos más simples para descomponer la imagen, después el cerebro interpreta la escena. La visión por computadora trata de emular este comportamiento, definiendo de esta manera cuatro fases. La primera fase consiste en la captura o adquisición de las imágenes digitales mediante algún tipo de sensor o medio, que podría ser una cámara de video. La segunda etapa consiste en el tratamiento digital de las imágenes, con objeto de facilitar la etapa anterior. Aquí es donde, se eliminan partes indeseables de la imagen o se realzan partes importantes de la misma. La siguiente fase es la segmentación, y consiste en aislar los elementos que interesan de una escena para comprenderla. Por último la etapa de reconocimiento o clasificación, donde se pretende distinguir los objetos segmentados, gracias al análisis de ciertas características que se establecen previamente para diferenciarlos (Vélez J. et al, 2003).

En la actualidad la tecnología se ha convertido en una herramienta imprescindible dentro de la vida cotidiana y al que las personas tienen acceso cada vez más. Desde un pequeño juguete hasta un gran sistema de comunicación, y por supuesto el campo de la salud es uno de los más beneficiados cuando al hablar de visión por computadora se refiere.

Según el “Informe Mundial sobre la Discapacidad, 2011” de la Organización Mundial de la Salud (OMS), menciona que más de mil millones de personas en el mundo tienen alguna forma de discapacidad; de ellas, casi 200 millones experimentan dificultades considerables en su funcionamiento y/o autosuficiencia.

La cifra aumentara, por diferentes motivos, a medida que transcurra los años (Organización Mundial de la Salud , 2011).

Este grupo de personas necesita de algún tipo de cuidado, sobre todo en su estado anímico, ya que esta influye en su salud, pudiendo empeorar o mejorar su situación. Tener una persona para que se encargue de dicha tarea es posible, pero muy pesado, así que se puede hacer uso de los avances en visión por computadora, para poder descifrar en su rostro, el estado de ánimo en el que se encuentra la persona, y de esta manera ejecutar alguna acción según la disposición emocional que presente.

El prestigioso Psicólogo Paul Ekman, es el pionero del estudio de las expresiones faciales en relación con las emociones que sentimos. Para determinar ciertas expresiones de las emociones como universales, realizó un estudio, en el que recoge las expresiones faciales que muestran diversas culturas ante las mismas emociones. Los resultados ofrecieron hasta siete expresiones faciales ante determinadas emociones que se pueden considerar universales, en común para todas las culturas: alegría, tristeza, miedo, sorpresa, repugnancia, ira y desprecio o desdén (Aragón, 2011)

El presente proyecto se enmarca dentro de la línea de Visión por Computadora, lo que conduce a investigar sobre las técnicas y herramientas utilizadas para el reconocimiento de patrones, debiendo seleccionar el método adecuado de visión por computadora, desarrollar el algoritmo, y finalmente el reconocimiento de expresiones faciales.

CAPITULO I

1. FUNDAMENTACIÓN TEÓRICA

El Sistema de Reconocimiento de Expresiones Faciales requiere para su desarrollo de la correcta elección del sistema operativo, porque de esta manera se evitaran futuras complicaciones durante la elaboración del programa principal.

Se eligió el sistema operativo Linux debido a la correcta adaptación con C++, que es el lenguaje de programación en el cual se va a desarrollar el reconocimiento de Expresiones Faciales, siendo la base fundamental en el diseño y elaboración del proyecto de investigación.

1.1.LINUX

Linux es un sistema operativo gratuito y de libre distribución inspirado en el sistema Unix. Una de las mayores ventajas es que es fácilmente portable a diferentes tipos de ordenadores, por lo que existen versiones para casi todos los tipos de ordenadores, desde PC y Mac hasta estaciones de trabajo y superordenadores. No está pensado para ser fácil de emplear, sino para ser sumamente flexible.

Figura 1. Logo Linux

Fuente. http://www.redusers.com/noticias/kernel-linux-3-2-ya-esta-disponible/

Existen dos características que lo diferencian del resto de sistemas existentes en el mercado. La primera, es libre, esto significa que por su uso no se debe pagar algún tipo de licencia a ninguna casa desarrolladora de software. La segunda, es que el sistema operativo viene acompañado del código fuente.

Está formado por el núcleo del sistema (conocido como kernel) más un gran número de programas y bibliotecas, lo que hacen posible su utilización. Gran parte de estos programas y bibliotecas se deben al proyecto GNU, por lo que, muchos conocen a Linux como GNU/Linux, esto resalta que el sistema esté formado por el núcleo y una gran parte del software producido por el proyecto GNU.

Figura 2. Compañías relacionadas con Linux

Fuente. http://logodatabases.com/ibm-logo.html/ibm-logo

Programadores de todo el mundo han diseñado y programado el sistema, el cual sigue en continuo desarrollo bajo la coordinación de Linus Torvalds, la persona de la que partió la idea de este proyecto, a principios de la década de los noventa. En la actualidad, grandes compañías, como IBM, SUN, HP, Novell y RedHat, y otras, aportan a Linux grandes ayudas tanto económicas como de código.

Cada vez más programas y aplicaciones están disponibles para este sistema, aumentando su calidad versión a versión. De estos, la gran mayoría vienen acompañados del código fuente y se distribuyen bajo los términos de licencia de la GNU General Public License.

Actualmente muchas casas de software comercial distribuyen sus productos para Linux, aumentando la presencia del mismo en empresas por la excelente relación calidad-precio que se consigue con este sistema (Martinez, 2014).

1.1.1. Distribuciones de Linux

Las distribuciones de Linux son una recopilación de programas y ficheros, organizados y preparados para su instalación. Dichas distribuciones se las puede adquirir por Internet, o comprando los CDs de instalación, los cuales contendrán todo lo necesario para instalar un sistema Linux completo. En la mayoría de los casos posee un programa inicial que ayuda en la tarea de instalación.

Figura 3. Distribuciones de Linux
Fuente. https://tallerinformatica.wordpress.com/distribuciones-gnulinux/

Existen muchas y variadas distribuciones creadas por diferentes empresas y organizaciones, llegando a un total de 31, de las cuales se cita los principales y más importantes: (Martinez, 2014)

- Ubuntu
- Redhat Enterprise
- Fedora
- Debian
- Open Suse
- Suse Linux Enterprise
- Slackware
- Gentoo
- Kubuntu
- Mandriva

De la lista anterior Fedora es la mejor opción, debido a que es uno de los más estables y seguros

1.1.1.1. Fedora

Figura 4. Logo fedora

Fuente.http://www.cronicasgeek.com/2016/01/futuras-prestaciones-fedora-24-linux/

Distribución gratuita creada y mantenida por la empresa Red Hat que utiliza el sistema de paquetería RPM (Red Hat Package Manager). Destaca por su estabilidad y seguridad gracias al sistema SELinux ("Security-Enhanced Linux"). Tiene tres versiones diferentes: escritorio (Workstation), servidores (Server) y sistemas en la nube (Cloud).

Fedora Workstation es un sistema operativo fácil de usar, funciona para computadoras portátiles y de sobremesa. Cuenta con un conjunto completo de herramientas para desarrolladores y creadores. Es poderoso, confiable y amigable con el usuario. Soporta todo tipo de desarrolladores, sean estos aficionados, estudiantes o profesionales.

El entorno de escritorio GNOME 3 ayuda a concentrarse en el código. Está desarrollado para las necesidades de los desarrolladores y libre de distracciones innecesarias, de manera que pueda concentrarse en lo que de verdad importa.

Figura 5. Escritorio Gnome 3
Datos tomados de la base de datos.
Autor. Juan Toaquiza

Y lo más importante posee el juego completo de lenguajes de código abierto, herramientas y utilidades de Fedora. Para el desarrollo del proyecto se eligió Fedora 24, que era la última versión en ese momento (Red Hat, Inc., 2015).

Para la realización adecuada del proyecto se debe elegir una aplicación informática que proporcione servicios integrales y herramientas al programador, que constituye el Entorno de Desarrollo Integrado, conocido por sus siglas en Ingles IDE

1.2. IDE (Integrated Development Environment)

Figura 6. Escritorio Gnome 3
Fuente: http://www.differencebtw.com/difference-between-eclipse-and-netbeans/

Es un entorno de programación empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Ofrecen un ambiente de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc.

Algunos pueden funcionar con varios lenguajes de programación a la vez. Un ejemplo de esto es Eclipse, al que se le puede añadir soporte de lenguajes adicionales mediante plugins (Ramos I, 2010) (Ecured, 2012)

1.2.1. Características de un IDE:

- Multiplataforma
- Soporte para diversos lenguajes de programación
- Integración con Sistemas de Control de Versiones
- Reconocimiento de Sintaxis
- Extensiones y Componentes para el IDE
- Integración con Framework populares

- Depurador
- Importar y Exportar proyectos
- Múltiples idiomas
- Manual de Usuarios y Ayuda

1.2.2. Componentes:

- Editor de texto.
- Compilador.
- Intérprete.
- Herramientas de automatización.
- Depurador.
- Posibilidad de ofrecer un sistema de control de versiones.
- Factibilidad para ayudar en la construcción de interfaces gráficas de usuarios.

Los IDE más representativos son:

- Eclipse
- NetBeans
- Geany
- CodeRun

1.2.3. Eclipse

Figura 7. Logo Eclipse neón
Fuente. http://www.first8.nl/nieuws/eclipse-neon-grails-2-x-with-groovy-and-gsp-support/

Es un entorno de desarrollo integrado, de Código abierto y Multiplataforma. Es una potente y completa plataforma de Programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Es un entorno de desarrollo integrado (IDE) donde se encuentra todas las herramientas y funciones necesarias para programación, con una atractiva interfaz que lo hace fácil y agradable de usar.

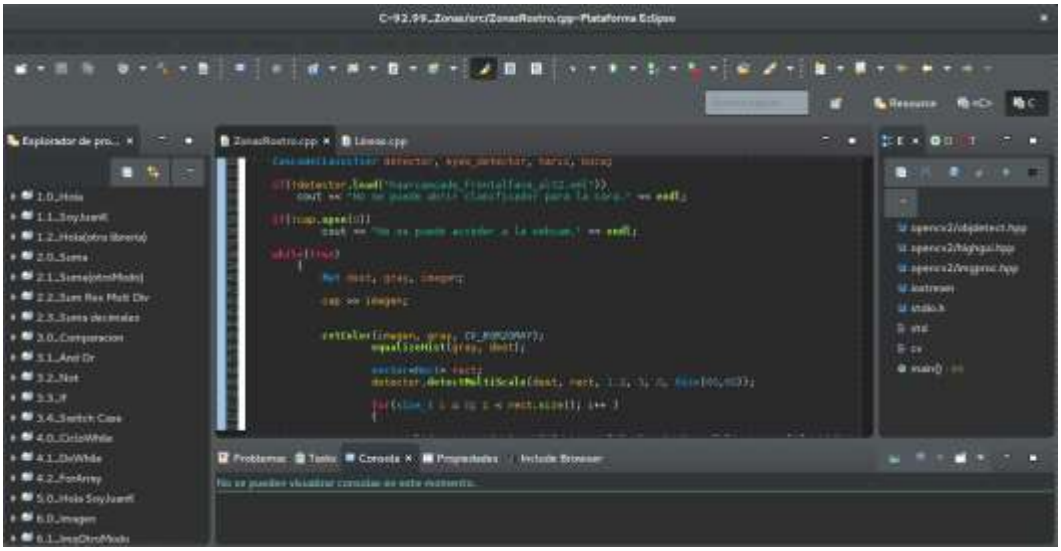


Figura 8. Interfaz Eclipse Neón
Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

1.2.3.1. Características de Eclipse:

- Editor de texto con resaltado de sintaxis donde se puede ver el contenido del fichero en el que se trabaja
- Contiene una lista de tareas y otros módulos similares
- Compilación en tiempo real.
- Tiene pruebas unitarias con JUnit
- Integración con Ant, asistentes (wizards) para creación de proyectos, clases, test, etc., y refactorización.

Estas características son de carácter general, los beneficios del programa se pueden ampliar y mejorar mediante el uso de plug-ins.

1.2.3.2. Ventajas de Eclipse

Entre las ventajas más importantes podemos de la plataforma de programacion eclipse, se puede destacar las siguientes:

- Emplea módulos (plug-in en inglés) para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, sean necesarias o no.

- Los plug-in es una plataforma ligera para componentes de software. Esto le permite a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python. Además manejar lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. (Galli, 2011).

Para el desarrollo del sistema de reconocimiento facial se hizo uso de Eclipse Neon, que era la última versión en ese momento. A este IDE se le debe adaptar un lenguaje de programación.

1.3. LENGUAJE DE PROGRAMACIÓN

Figura 9. Lenguajes de programación
Fuente. http://www.campusmvp.es/recursos/post/Los-lenguajes-de-programacion-mas-demandados-por-las-empresas.aspx

Se refiere a aquel lenguaje especialmente diseñado para describir un conjunto de acciones consecutivas o instrucciones que un equipo informático debe ejecutar. Es así como un lenguaje de programación pasa a ser la manera práctica de lograr que

el equipo ejecute las acciones que el usuario desea. Obedecen a reglas que le permiten expresar las instrucciones que serán interpretadas.

Un lenguaje de programación no es lo mismo que un lenguaje informático, pues estos últimos comprenden otros lenguajes que dan formato a un texto pero no son programación en sí mismos.

Un programador es una persona que maneja los lenguajes de programación y se encarga de utilizarlos para crear secuencias de instrucciones que, en conjunto, conformarán programas informáticos.

Existen diferentes clases de lenguajes de programación, como los funcionales o procedimentales, los imperativos, los lógicos, los híbridos, los orientados a objetos.

Algunos ejemplos de lenguajes de programación son:

- ADA
- BASIC
- C
- C++
- Cobol
- Fortran,
- Java
- LISP
- Pascal
- PHP
- Perl
- Prolog
- ASP
- Action Script
- Python
- JAVA
- Java Script
- Logo (Enciclopedia de Ejemplos, 2016)

1.3.1. Lenguaje de programación C++

Figura 10. Logo C++ Eclipse
Fuente: https://projects.eclipse.org/projects/tools.cdt

C++ es un lenguaje de programación creado por Bjarne Stroustrup, quien tomó como base al lenguaje C que en ese entonces era el lenguaje de programación más popular. Entonces C++ es un derivado del mítico lenguaje C

Este lenguaje nace por la necesidad de ciertas facilidades de programación, incluidas en otros lenguajes pero que C no soportaba, al menos directamente, como son las llamadas clases y objetos, principios usados en la programación actual. Por este motivo se rediseñó C, ampliando sus posibilidades pero permitiendo al programador en todo momento tener controlado el proceso, consiguiendo así una mayor rapidez.

El objetivo de C++ es conducir a C a un nuevo paradigma de clases y objetos con los que se realiza una comprensión más humana basándose en la construcción de objetos, con características propias solo de ellos, agrupados en clases. Por lo que se puede decir que C++ es un lenguaje de programación orientado a objetos que toma la base del lenguaje C.

Al inicio la intención era extender al exitoso C con mecanismos que permitieran la manipulación de objetos. Desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido

Para trabajar con este lenguaje es recomendable tener conocimientos de C, debido a que C++ es una mejora de C. Tener los conocimientos necesarios permitirá un

avance más rápido y mejor comprensión. Además se debe tomar en cuenta que C++, admite C, por lo que se puede programar reutilizando funciones de C, que se puedan usar en C++.

Muchos programadores recomiendan no saber C para saber C++, por ser el primero de ellos un lenguaje imperativo o procedimental y el segundo un lenguaje de programación orientado a objetos. Pero es recomendable tener nociones sobre la programación orientada a objetos en el caso de no tener conocimientos previos de programación estructurada (Zator System , 2016)

1.3.1.1. Lectura, escritura, entrada/salida de datos en C++

La lectura y escritura de datos se realiza por medio de operadores que se encuentran en una librería llamada iostream . La palabra stream quiere decir algo así como canal, flujo o corriente. Las declaraciones de los operadores de esta librería están en un fichero llamado iostream.h. En C++ las entradas son leídas desde streams y las salidas son escritas en streams.

El lenguaje C++ no dispone de sentencias de entrada/salida. En su lugar se utilizan operadores contenidos en la librería estándar y que forman parte integrante del lenguaje. La librería de C++ proporciona algunas herramientas para la entrada y salida de datos que la hacen más versátil. En C++ las entradas son leídas desde streams y las salidas son escritas en streams. Los streams más utilizados para introducir y sacar datos son cin y cout respectivamente.

Para la utilización de dichos streams es necesario incluir, al comienzo del programa, el archivo #include <iostream.h> en el que están definidos sus prototipo (Bustamante P, 2012)

1.3.1.2. Arrays en C++

Un vector, también llamado array (arreglo) unidimensional, es una estructura de datos que permite agrupar elementos del mismo tipo y almacenarlos en un solo bloque de memoria juntos, uno después de otro. Por ejemplo:

```
int a[10];
```

En el código anterior se crea espacio para 10 variables del tipo *int* colocadas una después de la otra, pero sin identificadores únicos para cada variable. En su lugar, todas están englobadas por el nombre de la variable a. Para acceder a cualquiera de los elementos del vector, se utiliza la misma sintaxis de corchetes que se utiliza para definir el vector:

```
a[5] = 47;
```

Se debe recordar que aunque el tamaño de **a** es 10, se seleccionan los elementos del vector comenzando por cero, de modo que sólo se pueden seleccionar los elementos del vector de 0 a 9, indicados en la figura 11.

```
#include <iostream>
using namespace std;
int main()
 {
int a[10];
for (int i = 0; i < 10; i++)
    {
a[i] = i * 10;
cout << "a[" << i << "] = " << a[i] << endl;
    }
}
```

La ventaja, los accesos a vectores son extremadamente rápidos. Sin embargo, si se indexa más allá del final del vector, no hay ninguna red de seguridad, entrará en otras variables. La otra desventaja es que se debe definir el tamaño del vector en tiempo de compilación; si se quiere cambiar el tamaño en tiempo de ejecución no se puede hacer con la sintaxis anterior (Eckel Bruce, 2012)

1.3.1.3. Eclipse CDT (C/C++ Development Tooling)

Consiste en un plugin necesario para el desarrollo de C y C++ en la plataforma Eclipse

El Proyecto CDT ofrece un completo y funcional entorno de desarrollo integrado C y C ++ basado en la plataforma Eclipse. Las características incluyen:

- Soporte para la creación del proyecto y la construcción gestionado por varias cadenas de herramientas.
- Marcas, normas de construcción, navegación fuente
- Varias herramientas de conocimiento de origen, como jerarquía de tipos, gráfico de llamadas (incluyen el navegador)
- El navegador de definición de macro
- Editor de código con resaltado de sintaxis, plegado e hipervínculo navegación
- Código fuente refactorización y la generación de código,
- Herramientas de depuración visuales, incluyendo la memoria, registros, y los espectadores de desmontaje (Eclipse Foundation, 2016)

Hasta el momento se tiene el sistema operativo, el IDE y un lenguaje de programación, pero para el desarrollo del proyecto se hace necesaria la implementación de librerías especiales para su uso en la programación de visión por computador o visión artificial, hablamos de las librerías OpenCV, de las cuales se entra en detalle más adelante.

1.4. VISIÓN POR COMPUTADOR

Visión por computador o visión técnica, es un sub-campo de la inteligencia artificial, por lo que en algunos casos lo conocen también como visión artificial. Su principal objetivo es programar un computador para que este entienda una escena o las características de una imagen, mediante procesos destinados a realizar el análisis de imágenes

Figura 11. Ejemplo de visión por computador
Fuente:http://www.distintiva.com/lab/aplicaciones-de-la-vision-por-computador-cv/

Este tipo de visión ha conseguido facilitar y hacer más productivas ciertas actividades, entre las que podemos mencionar:

- La automatización de tareas repetitivas realizadas por personal humano.
- Controles de calidad de productos a gran velocidad.
- Inspección de objetos sin contacto físico.
- Aumento de velocidad en de ciclos en procesos automatizados.
- Inspeccionar procesos donde existe diversidad de piezas
- Identificación de objetos específicos.
- Guiado de robots
- Dar coordenadas de un determinado objeto.
- Realización de mediciones angulares y tridimensionales (Centro Integrado Politécnico, 2014)

La mencionada visión por computador requiere de un correcto procesamiento de imágenes para cumplir con sus objetivos de manera satisfactoria.

1.4.1. Procesamiento de imágenes

El procesamiento, el análisis y la interpretación de imágenes es un campo de especialización muy importante de la visión artificial, que hace posible que un

ordenador procese imágenes o fotografías bidimensionales, aisladas o conectadas en secuencias dinámicas o temporales

Figura 12. Ejemplo del procesamiento de imágenes
https://www.cs.cmu.edu/~chuck/lennapg/lenna.shtml

Es una clase de procesamiento de señales que puede tomar fotografías o fotogramas de vídeo y someterlos a un procesado de perfil bajo con el fin de extraer determinadas características o parámetros, o bien para elaborar nuevas imágenes procesadas como material de salida.

La visión por computador puede ser considerada como una forma más genérica del procesamiento de imágenes, más cercana a la interpretación de imágenes, cuya finalidad es procesar, comprender y descifrar características o rastrear objetos, con un objetivo claro, en una imagen o una secuencia de imágenes de vídeo.

Para este fin es necesario conocer términos generales que se nombrara durante la programación, como es el histograma y la ecualización del histograma

1.4.1.1. Histograma

Un histograma tal es la base sobre la cual se forman y operan muchos clasificadores de forma común. En este caso, las medidas inexactas de ángulo de gradiente

disminuirán el rendimiento del reconocimiento del clasificador porque los datos que se pueden aprender variarán en función de la rotación del rostro.

Ecualización de histograma

Las cámaras y sensores de imagen por lo general tienen que lidiar no sólo con el contraste en una escena, sino también con la exposición a los sensores de imagen a la luz resultante en esa escena. En una cámara estándar, la configuración de obturación y la abertura de la lente hacen malabarismos entre la exposición de los sensores a la demasiada o poca luz. Después de que la imagen ha sido tomada, no hay nada que podamos hacer sobre lo que el sensor registra. Sin embargo, todavía podemos tomar lo que está allí y tratar de ampliar el rango dinámico de la imagen para aumentar su contraste

Esto es evidente a partir de los valores de intensidad del histograma. Por lo general se trata de una imagen de 8 bits donde sus valores de intensidad pueden variar de 0 a 255, pero el histograma muestra que los valores de intensidad reales están agrupados cerca de la mitad de la gama disponible. La ecualización de histograma es un método para el estiramiento de a partir de este intervalo (Adrian Kaebler, 2008)

Para la detección de expresiones faciales el componente esencial en el procesamiento de imágenes son las bibliotecas de código abierto OpenCV, especializadas en este campo, las cuales se detalla a continuación.

1.4.2. OpenCV

Figura 13. Logo Open CV

Fuente.http://www.incanatoit.com/2015/02/configuracion-opencv-cpp-visual-studio-2013.html

Es una biblioteca de visión por ordenador de código abierto disponible en http://opencv.org. La biblioteca está escrito en C y C ++. Funciona bajo Linux, Windows, Mac OS X, iOS y Android. Posee interfaces disponibles para Python, Java, Ruby, Matlab, y otros idiomas.

OpenCV fue diseñado para la eficiencia computacional con un fuerte enfoque en aplicaciones en tiempo real. Las optimizaciones se hicieron en todos los niveles, desde los algoritmos de multi-núcleo hasta las instrucciones de la CPU.

Uno de los objetivos de OpenCV es proporcionar una infraestructura de visión por ordenador fácil de usar que ayuda a las personas a construir aplicaciones de visión bastante sofisticados rápidamente. La biblioteca OpenCV contiene más de 500 funciones que abarcan muchas áreas, incluyendo la inspección de fábricas de productos, imágenes médicas, la seguridad, la interfaz de usuario, calibración de la cámra, la visión estéreo, y la robótica.

1.4.2.1. Código abierto OpenCV

La licencia de código abierto BSD para OpenCV se ha estructurado de tal manera que se puede construir un producto comercial utilizando la totalidad o parte de OpenCV. No se está bajo ninguna obligación de abrir el código del producto para volver al dominio público.

Figura 14. Símbolo de código abierto
Fuente. http://www.kubos.co/kubos/

En parte debido a estos términos de licencia liberales, hay una gran comunidad de usuarios que incluye a grandes empresas (Google, IBM, Intel, Microsoft, Nvidia, Sony y Siemens, por nombrar sólo unos pocos) y centros de investigación (como Stanford , MIT, CMU, Cambridge, Georgia Tech y el INRIA). OpenCV también

está presente en la web para los usuarios en http://opencv.org, un sitio web que aloja la documentación, información del desarrollador, y otros recursos de la comunidad.

Aunque Intel comenzó OpenCV, la biblioteca está y siempre tenía la intención de promover el uso comercial y la investigación. Por tanto, es abierta y libre, y el propio código se puede utilizar o incrustar (en su totalidad o en parte) en otras aplicaciones, ya sea comercial o de investigación. No obliga al uso de un código de aplicación para ser abierto. No requiere devoluciones o mejoras a la biblioteca.

1.4.2.2. Módulos de OpenCV

Esta biblioteca se divide en varias secciones según las necesidades de cada programador, cada una de las cuales se llama módulo de la biblioteca, que son la estructura organizativa principal dentro de OpenCV.

La lista de módulos exacta ha evolucionado con el tiempo. Cada función en la biblioteca es parte de un módulo. La versión 3.1 de OpenCV presenta los siguientes módulos:

- **opencv_core:** Este módulo tiene definida la estructura básica de datos, Es uno de los módulos que se debe cargar siempre para trabajar con OpenCV
- **opencv_highgui:** Contiene funciones para la lectura y escritura de imagen y video y otras funciones de interfaz de usuario. Es imprescindible, ya que se encarga de ofrecernos una interfaz fácil para poder cargar fotos, capturar videos
- **opencv_imgproc:** Contiene las funciones principales para el procesado de imágenes.
- **opencv_objdetect:** Nos proporciona todas las herramientas para hacer detección de clases ya predefinidas. Contiene funciones para la detección de objetos como caras y personas.
- **opencv_video:** Contiene todas funciones para análisis de vídeos, incluye estimaciones del movimiento, eliminación del fondo y tracking de objetos.

- **opencv_features2d:** Contiene características para detectar puntos y descriptores.
- **opencv_calib3d:** Contiene funciones para la calibración de la cámara, geometrías, etc.
- **opencv_gpu:** Diseñado para sacar provecho de la GPU
- **opencv_flann:** Contiene algoritmos de geometría computacional
- **opencv_ml:** Contiene funciones de aprendizaje Machine Learning
- **opencv_contrib:** Es de código especializado
- **opencv_legacy:** Este módulo contiene los códigos mas antiguos para detección tanto de objetos como de rostros
- **opencv_ocl:** El módulo de OCL es un módulo más moderno, que podría ser considerado análogo al módulo GPU
- **opencv_photo:** Este es un módulo contiene herramientas útiles para la fotografía computacional
- **opencv_stitching:** Este módulo entero implementa un ducto de unión de imágenes sofisticada.
- **opencv_superres:** Súper resolución
- **opencv_videostab:** Estabilización de vídeo (Adrian Kaebler, 2008)

Para el reconocimiento de expresiones faciales es necesario reconocer o identificar un rostro y las partes de la misma, como son los ojos, la nariz y la boca. OpenCV nos facilita la tarea de detectar todo esto pues ya cuenta con clasificadores entrenados para esta tarea almacenados en archivos xml. Este procedimiento a seguir se le conoce como el método de Haar Cascades o Cascade Classification, que en español se traduciría como clasificadores en cascada.

1.4.2.3. Haar Cascades

La detección de objetos usando Haar o clasificadores en cascada es un método de detección de objetos efectivo propuesto por Pablo Viola y Michael Jones en su artículo, “Rapid Object Detection using a Boosted Cascade of Simple Features” en el año 2001. Se trata de un enfoque basado en el aprendizaje de las máquinas donde

una función de cascada es entrenada con una gran cantidad de imágenes positivas y negativas. Se utiliza para detectar objetos en otras imágenes.

Inicialmente, el algoritmo necesita una gran cantidad de imágenes positivas (imágenes de rostros) y las imágenes negativas (imágenes sin caras) para entrenar el clasificador. Entonces se debe extraer las características de la misma. Para esto, se utilizan las características Haar. Cada Característica es un solo valor obtenido restando la suma de los píxeles bajo rectángulo blanco de suma de píxeles bajo rectángulo negro.

Una Característica de Haar considera regiones rectangulares adyacentes en algún lugar específico del área de detección, sumando las intensidades de los pixeles y calculando la diferencia entre ellas para luego clasificar la subregión en alguna categoría.

En la figura se muestra un conjunto de Haar-Features o Características de Haar para tres tipos de características: dos rectángulos (A y B), tres rectángulos (C y D) y cuatro rectángulos (E). Para dos rectángulos el valor calculado es la diferencia entre la suma de pixeles dentro de las dos regiones rectangulares que tienen el mismo tamaño y forma horizontal o verticalmente. En tres rectángulos se calcula la suma dentro de los dos rectángulos exteriores restándole el central. Por último para cuatro rectángulos se calcula la diferencia entre los pares diagonales de rectángulos.

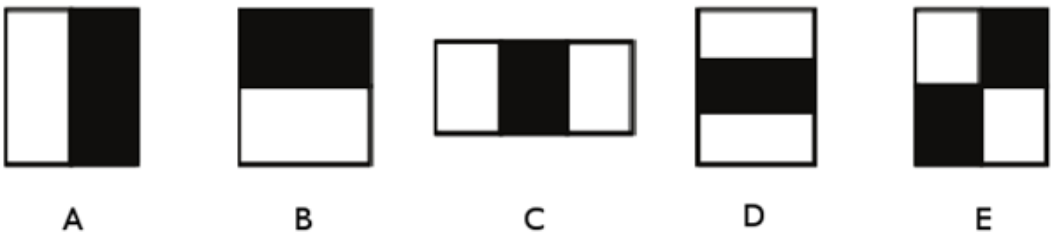


Figura 15. Características de Haar Features
Fuente.http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0

En resumen se podría decir que Haar Cascades es el método más utilizado en la librería de OpenCV, ya que emplea un sistema pre-entrenado para reconocer características, que consiste básicamente en un árbol de decisión degenerado, donde

en cada fase un detector o filtro es entrenado para detectar casi todos los objetos de interés mientras rechaza a los que no lo son (Open CV 3.1.0dev,, 2014)

1.4.2.4. Haar Cascade en OpenCv

OpenCV ya contiene muchos clasificadores pre-formados para cara, ojos, sonrisa, etc. Estos archivos XML se almacenan en la carpeta "data" en los archivos de instalación de OpenCV versión 2.4.9 y 3.1.0 para Linux. También existen versiones para Windows. Para su uso se debe dar la ubicación del archivo XML dentro de Fedora, o colocar el archivo en la carpeta donde se guarda la programación. Cabe recalcar que que estos archivos no funcionan por si solos. Para su uso se hace necesaria una programación que conlleve a la obtención de los resultados deseados.

CAPITULO II

2. METODOLOGÍA

2.1. Tipo de estudio

2.1.1. Descriptivo

El tipo de investigación es descriptiva, debido a que se observa y describe la eficacia de las librerías OpenCV en el reconocimiento de expresiones faciales con programación en lenguaje C++ con señales de video en tiempo real mediante el método de **Haar-Cascade,** Para lograr una mejor respuesta, se somete a un proceso de regionalización del rostro, aumentado significativamente la velocidad del programa.

2.2. Métodos, Técnicas e Instrumentos

2.2.1. Método

2.2.2.1 Experimental

El método empleado en el proyecto es experimental debido que se manipulan las líneas de código para poder apreciar su funcionalidad, la relación que tiene con otras líneas, y la variación que causa en el proceso general. De esta manera se puede hacer correcciones que conlleven a un funcionamiento adecuado del programa.

2.3 Técnicas

2.3.1. Observación

La técnica escogida se caracteriza por recolectar información y datos que proporciona la imagen de video que se muestra en la pantalla del computador. La variación que sufre la imagen en diferentes condiciones de luz, mayor o menor distancia del rostro a la cámara, y cambios que se dan en las líneas de código.

2.3.2. Instrumentos

Los instrumentos necesarios son:

- Bibliografía.- libros, archivos de texto, archivos de video y páginas web.
- Diseño de expresiones faciales.IDE Eclipse Neón, Lenguaje de programación C++, Librerías Open CV, Método Haar Cascade

2.4. Población y muestra

2.4.1. Población

La población consta de las diferentes expresiones faciales que puede presentar un rostro común y corriente captada por la cámara en tiempo real: seriedad, alegría, tristeza, enojo. La captación de las mismas no se deben ver afectadas por el género (hombre o mujer) y tratar en lo posible no verse afectadas por las condiciones externas en la que se encuentre.

2.4.2. Muestra

La muestra tomada para el análisis del proyecto de tesis será la señal que brinda la cámara de la computadora en tiempo real. Aunque cabe mencionar que para pruebas iníciales se hizo uso de imágenes con rostros escogidos al azar

2.5. Hipótesis

Ha: El sistema de reconocimiento de expresiones faciales es aplicable a la interacción Humano-Robot

Ho: El sistema de reconocimiento de expresiones faciales no es aplicable a la interacción Humano-Robot

2.6. Operacionalización de variables

Tabla 1. Operacionalización de variables

VARIABLE	CATEGORÍA	INDICADOR	ÍTEM	TÉCNICA/INSTRUMENTO
V. Independiente Sistema de Expresiones faciales	La cuantificación de las expresiones faciales constituye un tema de interés en el campo de la visión por computador por su utilidad en la construcción de interfaces avanzadas de comunicación con el ordenador. La cara es un objeto difícil de analizar mediante técnicas de Visión por Computador pues presenta amplias zonas con poca textura y sus regiones más expresivas (ojos, cejas y boca) sufren deformaciones no rígidas, añadido la dificultad de modelizar y predecir el movimiento de la cabeza junto a la existencia de oclusiones o de condiciones cambiantes de iluminación.	Se llevara a cabo un análisis de las regiones de la cara para de esta manera obtener información necesaria en la clasificación del estado emocional o sentimiento del individuo.	Identificar relaciones entre las variables existentes	IDE Eclipse Neón Lenguaje de programación C++ Librerías Open CV Método Haar Cascade

V. Dependiente Interacción Humano-Robot	La interacción humano- Robot se refiere al estudio del proceso de interacción entre el ser humano y la maquina (computadora). Su importancia radica en la mejora del uso de la computadora como herramienta de trabajo, ocio y aprendizaje.	Se realizara mediciones de niveles y tiempos en los que se mantiene interacción tiempo real y de manera fluida.	Identificar parámetros influyentes	Distancias entre el usuario y el computador Condiciones de iluminación

Autor. Juan Toaquiza

2.7. Procedimientos

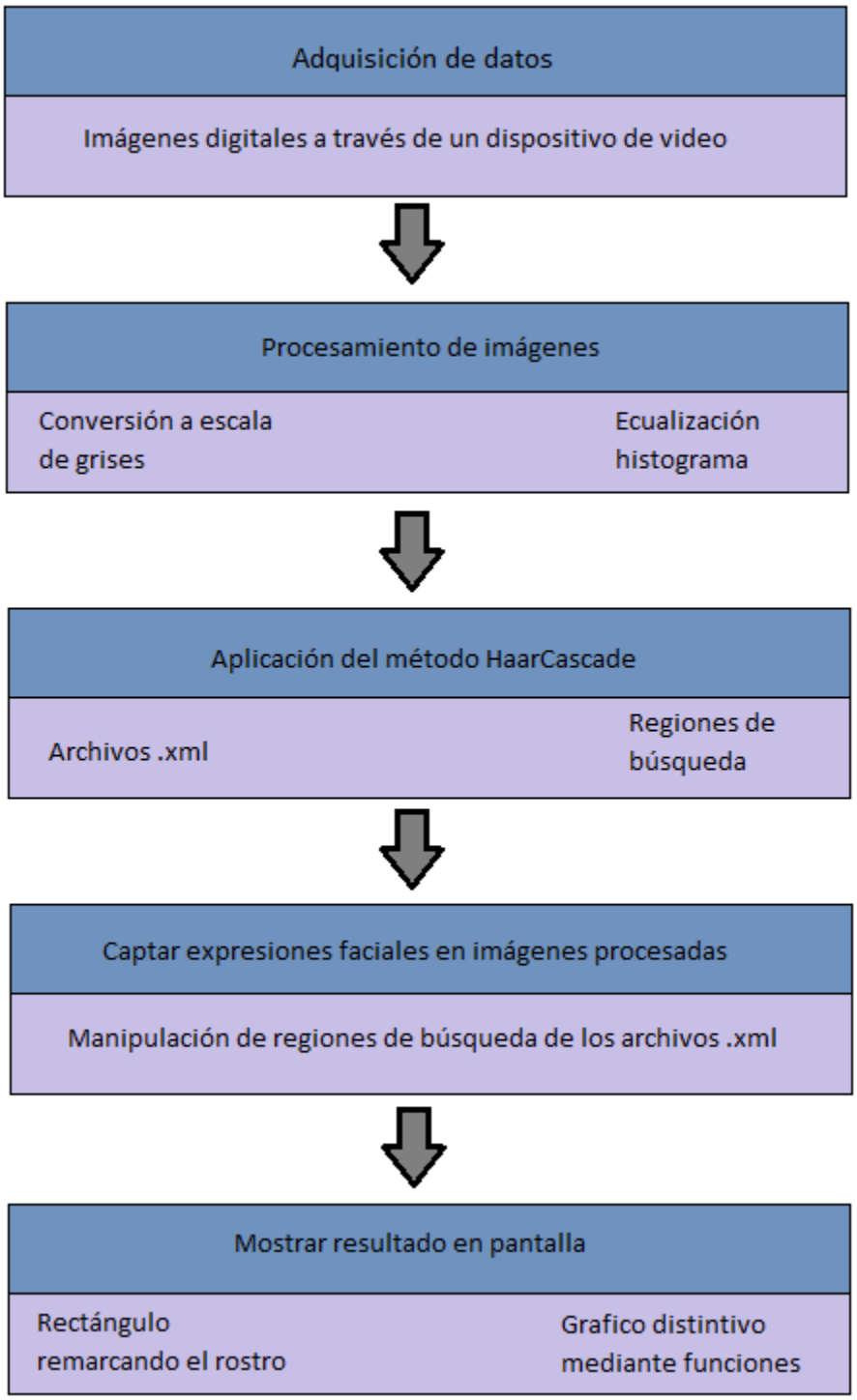


Figura 16. Procedimiento general para la detección de expresiones faciales
Fuente: Base de datos
Autor. Juan Toaquiza

2.8. Procedimiento y análisis

2.8.1. Diseño

2.8.1.1. Configuración del IDE Eclipse para el uso con OpenCV

La manera más fácil de instalar Eclipse fue a través del gestor de paquetes que viene por defecto en Fedora 24. Esta herramienta descargó la última versión de la página oficial. Cabe recalcar que se debe incluyó el plugin CDT, esencial para el funcionamiento con el lenguaje de programación C++

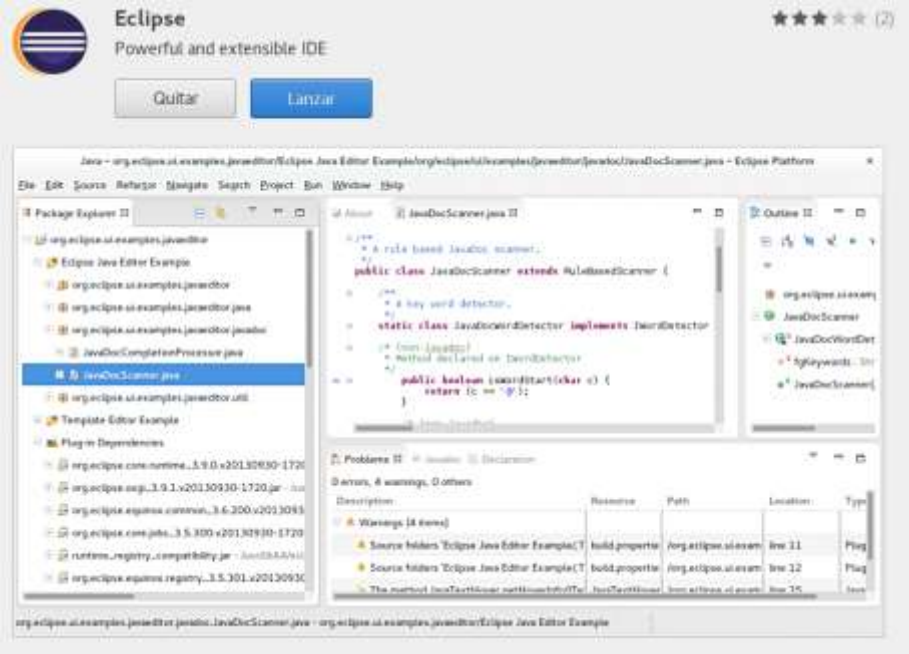


Figura 17. Gestor de paquetes para instalar Eclipse (captura de pantalla)
Fuente. Datos tomados de la base de datos.
Autor. Juan Toaquiza

Las librerías OpenCV fueron instaladas de la misma manera a través de un gestor de paquetes.

2.8.1.2. Activar las librerías Opencv en Eclipse

La activación de las librerías OpenCV se lo realizó a través de Eclipse, ingresando a propiedades de proyecto, hasta llegar a Settings (valores en español)

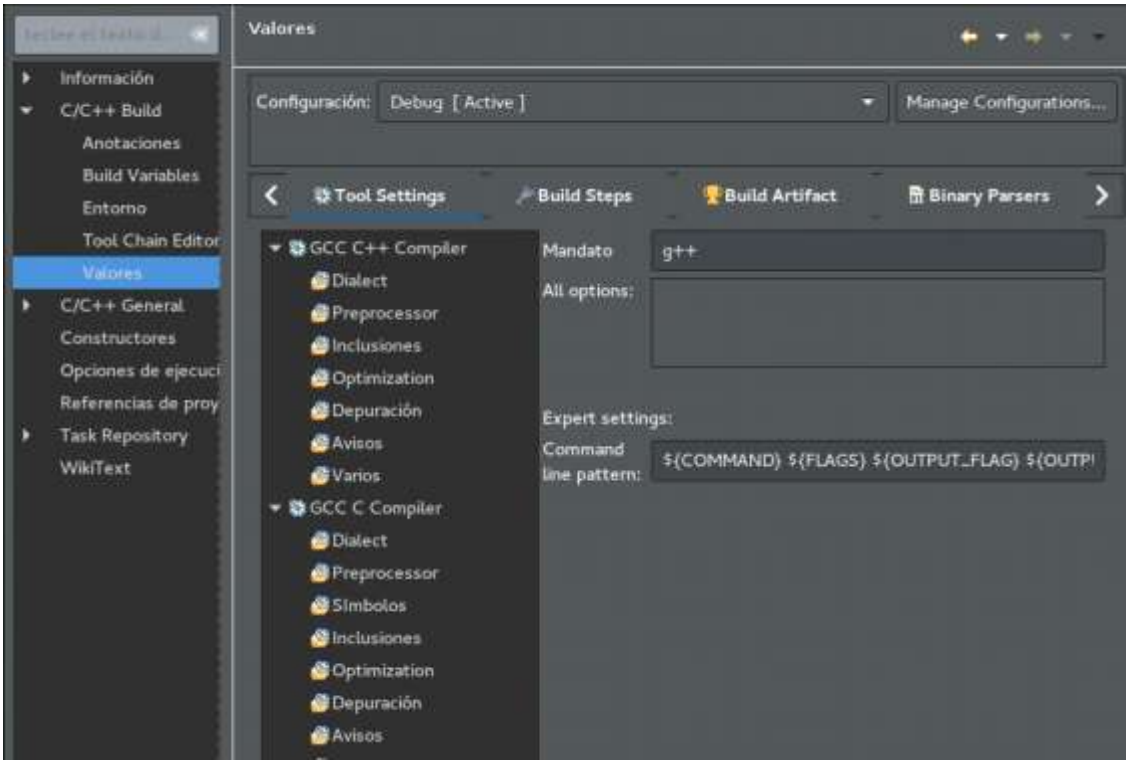


Figura 18. Ventana para añadir el path de OpenCV
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

En la pestaña de Include paths(-I) añadió el path de OpenCV, que en este caso es -I/usr/include/opencv. Posteriormente nos dirigimos hasta GCC C++ Linker y nos ubicamos en Bibliotecas

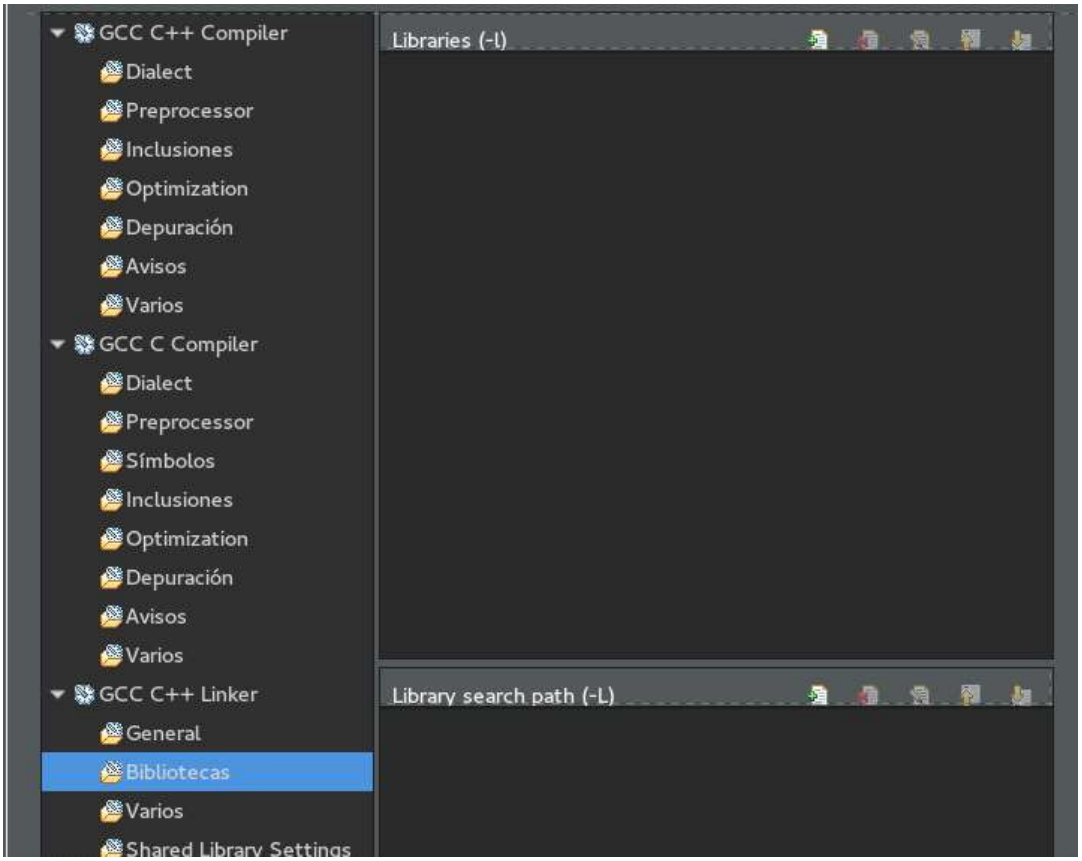


Figura 19. Ventana para ingresar los módulos de OpenCV
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

En la pestaña de Library search path (-L) se añadió el path donde reside OpenCV. Mientras que en la pestaña Libraries(-I) se colocó los módulos de OpenCV disponibles.

En este proyecto se hiso necesario la utilización de siete módulos, los cuales son:

- opencv_core
- opencv_highgui
- opencv_imgproc
- opencv_objdetect
- opencv_video
- opencv_features2d
- opencv_calib3d

2.8.1.3.Comprobación de funcionamiento

Para comprobar si la configuración es correcta se creó un pequeño código el cual imprimirá un texto en pantalla con la utilización de algunas funciones de OpenCV. (Open CV, 2014)

```
#include <cv.h>
#include <highgui.h>
int main ( int argc, char **argv )
{
cvNamedWindow( "My Window", 1 );
IplImage *img = cvCreateImage( cvSize( 640, 480 ), IPL_DEPTH_8U, 1 );
CvFont font;
double hScale = 1.0;
double vScale = 1.0;
int lineWidth = 1;
cvInitFont( &font, CV_FONT_HERSHEY_SIMPLEX | CV_FONT_ITALIC,
hScale, vScale, 0, lineWidth );
cvPutText( img, "Hola Soy JuanK", cvPoint( 200, 400 ), &font,
cvScalar( 255, 255, 0 ) );
cvShowImage( "My Window", img );
cvWaitKey();
return 0;
}
```

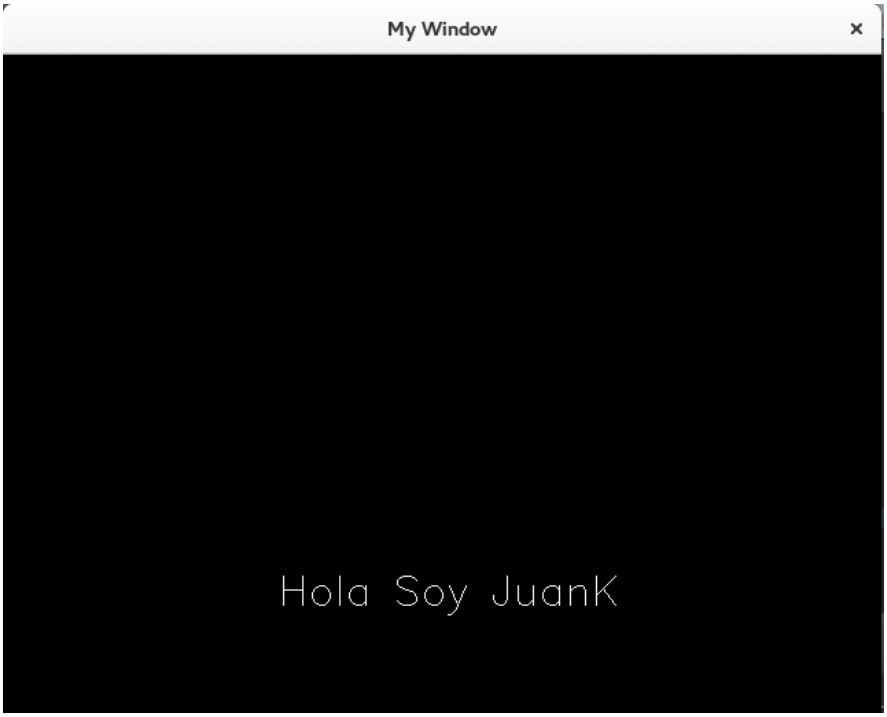


Figura 20. Fuente. Captura de pantalla tomados de la base de datos.
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

De esta manera se consiguió comprobar que las librerías de OpenCV estuvieran funcionando correctamente, lo que permitió comenzar de manera segura el proyecto.

2.8.1.4.Detección de rostros en una imagen con OpenCV

La detección de rostros es una tarea que se facilita con OpenCV debido a que cuenta con clasificadores entrenados para esta tarea los cuales se encuentran almacenados en archivos xml.

Para llegar a utilizar los clasificadores se tuvo que realizar una serie de líneas de código con las cuales se preparó la imagen y demás detalles para poder mostrar la detección del rostro en una imagen. Las líneas de código se las van a detallar tratando de ser lo más explícito posible.

2.8.1.5.Programación para la detección de rostros

Para comenzar con la programación de la detección de rostros en una imagen se tuvo que comenzar por incluir los módulos de las librerías de OpenCV que se van a utilizar, esto se lo hace a través de la función *include*, de la siguiente manera:

```
#include "opencv2/objdetect.hpp"
```

Para esta programación se utilizaron los módulos objedetec, highgui, e imgproc que se van a declarar de manera similar. Estas tuvieron que estar configuradas con anterioridad de manera correcta en el IDE Eclipse neón. De la misma manera se declaran *iostream* y *stdio* propias de C++

a. Using namespaces

Se hizo necesaria la utilización de *using namespace* para poder omitir tanto "std" de lenguaje C++, como "cv" de OpenCV, previo a la escritura de algunas funciones o comandos durante la programación.

```
using namespace std;
using namespace cv;
```

b. Int main

Se comenzó la ejecución del programa a través de *int main*. Abre corchetes al inicio, cierra al final del programa principal.

```
int main()
{
Programa principal
}
```

c. Variables tipo Mat

Se declararon 2 variables *Mat*, que es el tipo de dato para operar con imágenes más empleadas.

```
Mat dest, gray;
```

d. Función imread

Se declaró una variable con el nombre “imagen” y le asignamos los datos de “sonreir.jpg” mediante la función *imread* que sirve para leer una imagen

```
Mat imagen = imread("sonreir.jpg");
```

Figura 21. Imagen “sonreir.jpg”
Fuente. http://homogenesis.com/mujeres-del-futuro/

El dato “sonreir.jpg” es un archivo de imagen que se ubicó en la carpeta donde se guarda la programación. OpenCV puede trabajar con los principales formatos JPG, PNG y BMP.

2.8.1.6.Clasificador en cascada

Se activó nuestra herramienta de detección. Se declaró el clasificador en cascada, definiendo la clase *CascadeClassifier* en el objeto “detector”

```
CascadeClassifier detector;
```

Se aplicó el archivo *xml* haciendo uso de *load* y mediante la estructura condicional *if* controlamos si existe mencionado archivo caso contrario mostramos un mensaje

```
if(!detector.load("haarcascade_frontalface_alt.xml"))
cout << "No se puede abrir clasificador." << endl;
```

Figura 22. Archivo haarcascade_frontalface_alt.xml
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

haarcascade_frontalface_alt.xml Es el tipo de archivo que contiene la base de datos para detectar con el método *CascadeClassifier*. Base de datos para Cara de Frente. Hay algunos otros, tanto para nariz, ojos, etc.

2.8.1.7.Escala de grises

En la escala de grises el valor de cada pixel se representa como un solo valor que lleva solo el valor de la intensidad, se compone una imagen formada exclusivamente a partir de diferentes tonos de gris (Bueno G, 2015)

Se transformó la imagen inicial a escala de grises mediante la función *cvtColor* que permite transformar una imagen RGB a otra en escala de grises, la cual se la guardó con el nombre *gray*

```
cvtColor(imagen, gray, CV_BGR2GRAY);
```

Figura 23. Imagen "sonreir.jpg" en escala de grises
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

2.8.1.8.Ecualización del Histograma

Se aplicó la ecualización de histograma a la imagen en grises, para mejorar el contraste y brillo de la imagen. La ecualización es una forma de manipulación del histograma de la imagen que reduce automáticamente el contraste en las áreas muy claras o muy oscuras de una imagen. Esto se lo hizo a través de la función *equalize* de OpenCV

```
equalizeHist(gray, dest)
```

Figura 24. Ecualización de histograma imagen "sonreir.jpg"
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

a. Elaboración del vector

Se creó un vector de rectángulos para almacenar las caras detectadas, con la función *vector* y *Rect* ambos de OpenCV. Este último almacena las coordenadas de un rectángulo

```
vector<Rect> rect;
```

b. Detección multi escala

Se empezó la detección de caras usando la función *detectMultiScale* (Método de detección multi escala), función propia de OpenCV que detecta objetos (cosas,

rasgos, etc) de diferentes tamaños en una imagen y devuelve una lista de rectángulos donde se encuentran esos objetos, en el vector *rect*

```
detector.detectMultiScale(dest, rect);
```

c. Ciclo For y grafica del rectángulo

Con la sentencia *for* (o ciclo for) se controló el dibujado del rectángulo sobre la imagen en el lugar donde se encuentra el rostro. El número de rectángulos varía dependiendo del número de rostros presentes; un rostro un rectángulo, dos rostros dos rectángulos, etc

El tipo de dato para el contador es *size_t* que puede representar valores no negativos, y mediante el método *.size()* se controla el número de elementos del vector en cuestión. Por cada ciclo se incrementara el valor de 1 a nuestro contador.

```
for(size_t i = 0; i < rect.size(); i++ )
```

Dibujamos el rectángulo sobre la imagen. Es la sentencia del *for*.

```
rectangle(imagen,Point(rect[i].x, rect[i].y),Point(rect[i].x + rect[i].width, rect[i].y + rect[i].height),CV_RGB(0,255,0), 2);
```

se mostró en pantalla una ventana con la imagen original, mediante la herramienta *imshow* que despliega una imagen en una ventana especifica.

```
imshow("Imagen original", imagen);
```

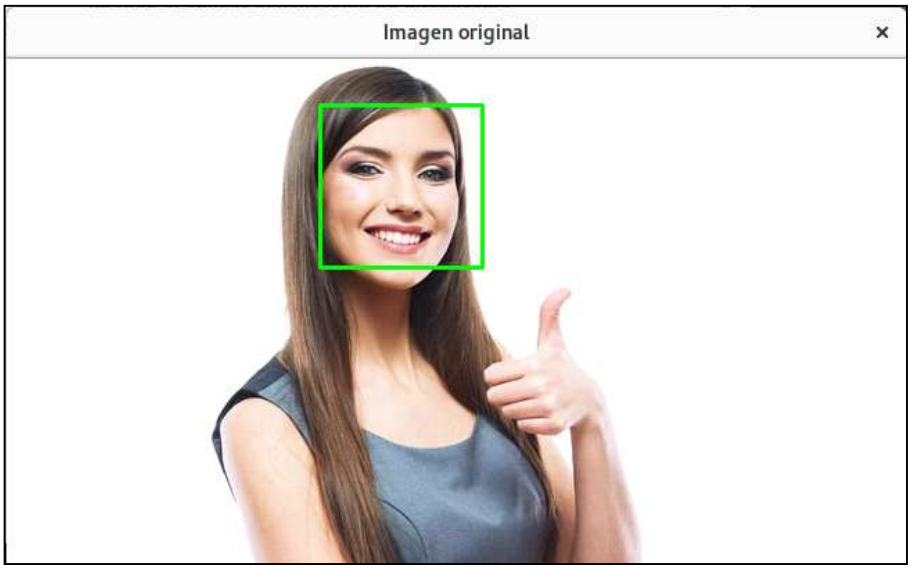


Figura 25. Reconocimiento facial imagen"sonreir.jpg"
Fuente. Captura de pantalla tomados de la base de datos.
Autor. Juan Toaquiza

Además se pudo mostrar en la pantalla otras imágenes que sean de interés como pueden ser en escala de grises o la ecualización del histograma, las cuales se pueden conseguir con la misma función *imshow*

```
imshow("Imagen en escala de grises", gray);

imshow("Imagen al aplicar ecualización de histograma",dest);
```

2.8.1.9.Detectar ojos en una imagen con OpenCV

Al haber detectado un rostro con OpenCV se tiene los conocimientos necesarios para la detección de ojos. OpenCV cuenta con clasificadores en cascada entrenados para la detección de ojos, los cuales pueden actuar tanto para ojos cerrados o abiertos, esto dependerá de la necesidad del programador.

a. Clasificadores para ojos

Clasificadores para detectar ojos abiertos o cerrados:

```
haarcascade_mcs_lefteye.xml, haarcascade_mcs_righteye.xml

haarcascade_lefteye_2splits.xml, haarcascade_righteye_2splits.xml
```

Clasificadores que detectan ojos abiertos:

```
haarcascade_eye.xml

haarcascade_eye_tree_eyeglasses.xml
```

b. Región de búsqueda

Se definió una región de búsqueda para que la detección de ojos funcione correctamente. Lo que hizo es buscar una porción de la imagen donde se desea detectar el ojo, basándose en los valores que deja la región donde se encontró el rostro. En el siguiente cuadro se muestra los puntos de sectorización recomendados para diferentes clasificadores de ojos

Tabla 2.Coordenadas de región para los ojos

Clasificador	**EYE_SX**	**EYE_SY**	**EYE_SW**	**EYE_SH**
haarcascade_eye.xml	0.16	0.26	0.30	0.28
haarcascade_mcs_lefteye.xml	0.10	0.19	0.40	0.36
haarcascade_lefteye_2splits.xml	0.12	0.17	0.37	0.36

Fuente. (Lélis Baggio, 2012)

c. Programación para detección de ojos

Se detalla a continuación la programación para la detección de ojos, basándose en lo visto en la detección de rostros.

Para empezar se estableció una región de búsqueda para el detector, en este caso será el *haarcascade_eye_tree_eyeglasses.xml*, resaltando que, es solo para ojos abiertos. Fueron de tipo de dato *float* o punto flotante que admite decimales.

```
float EYE_SX = 0.16f;
float EYE_SY = 0.26f;
float EYE_SW = 0.30f;
float EYE_SH = 0.28f;
```

En esta ocasión se tuvo otra imagen que sirvió para la detección de ojos, se la nombró como *images*

```
Mat imagen = imread("images.jpg");
```

Figura 26. Grafico "images.jpg"
Fuente. http://listas.20minutos.es/lista/el-rostro-femenino-mas-bello-319783/

d. Clasificador en cascada

Se definió la clase *CascadeClassifier* en el objeto *detector* que fue para la cara y en *eyes_detector* que se utilizó para los ojos. Se declaró el clasificador en cascada.

```
CascadeClassifier detector, eyes_detector;
```

Cargamos el archivo *xml* para los ojos haciendo uso de *load* y mediante la estructura condicional *if* controlamos si existe mencionado archivo caso contrario mostramos un mensaje

```
if(!eyes_detector.load("haarcascade_eye_tree_eyeglasses.xml"))
cout << "No se puede abrir clasificador para los ojos." << endl;
```

En esta parte se continúa con la programación normal para la detección de rostros, la cual llegó hasta la graficación del rectángulo sobre el rostro

```
cvtColor(imagen, gray, CV_BGR2GRAY);
equalizeHist(gray, dest);
vector<Rect> rect;
detector.detectMultiScale(dest, rect);
for(size_t i = 0; i < rect.size(); i++ )
{
rectangle(imagen,Point(rect[i].x, rect[i].y),Point(rect[i].x + rect[i].width, rect[i].y + rect[i].height),CV_RGB(0,255,0), 2);
}
```

e. Condición para la detección de los ojos

En este punto se creó una condición principal *If* para la detección de ojos. La estructura controla la existencia de valores en el vector *rect*, para seguir con la detección de ojos, caso contrario saltar a la parte final del programa principal.

```
if(rect.size() > 0) {
}
```

f. Clonar una imagen existente

El método *.clone()* permite clonar una imagen existente. La imagen clonada tiene el mismo tamaño, tipo y contenido que la original. Es una imagen nueva, que tiene su propia memoria.

Creamos una variable del tipo *Mat*, con el nombre *face*, en la cual se asigna los datos clonados del rectángulo en la primera posición del gráfico donde se aplicó el histograma (*dest*).

```
Mat face = dest(rect[0]).clone();
```

Se creó dos vectores con el tipo de datos *Rect* con los nombres *leftEye* y *rightEye*, correspondientes a los ojos izquierdo y derecho respectivamente.

```
vector<Rect> leftEye, rightEye;
```

g. Definir la región

Se extrajeron los rectángulos que contienen la región para los ojos izquierdo y derecho, ya que se va a usar un detector diferente para el ojo izquierdo y derecho. Este código extrae las regiones para ambos ojos:

```
int leftX = cvRound(face.cols * EYE_SX);
int topY = cvRound(face.rows * EYE_SY);
int widthX = cvRound(face.cols * EYE_SW);
int heightY = cvRound(face.rows * EYE_SH);
int rightX = cvRound(face.cols * (1.0-EYE_SX-EYE_SW));
```

Se creó dos variables del tipo *Mat*, una para la parte superior izquierda de la cara, y otra para la derecha. Estas contienen los datos declarados en *int leftX, int topY, int widthX, int heightY, int rightX*. En estas zonas se realizó la detección de los ojos.

```
Mat topLeftOfFace = face(Rect(leftX, topY, widthX, heightY));
Mat topRightOfFace = face(Rect(rightX, topY, widthX, heightY));
```

Se pasa el detector a cada zona definida, y se guarda los datos en los vectores "leftEye" y "rightEye" respectivamente.

```
eyes_detector.detectMultiScale(topLeftOfFace, leftEye);
eyes_detector.detectMultiScale(topRightOfFace, rightEye);
```

h. Grafica del rectángulo

Mediante la estructura *If* se verifica si hay algún valor dentro de *leftEye*. Si es verdadero, se dibuja el rectángulo, caso contrario se salta a la otra línea. Esto en el caso del ojo izquierdo

```
if((int)leftEye.size() > 0)
{
rectangle(imagen, Point(leftEye[0].x + leftX + rect[0].x, leftEye[0].y + topY +
rect[0].y),
Point(leftEye[0].width + widthX + rect[0].x - 5, leftEye[0].height + heightY +
rect[0].y),
CV_RGB(0,255,255), 2);
}
```

Y para el caso del ojo derecho se prosiguió de igual manera que el dibujo del primer rectángulo para el ojo izquierdo.

```
if((int)rightEye.size() > 0)
{
rectangle(imagen,Point(rightEye[0].x + rightX + leftX + rect[0].x, rightEye[0].y +
topY + rect[0].y), Point(rightEye[0].width + widthX + rect[0].x + 5,
rightEye[0].height + heightY + rect[0].y),CV_RGB(0,255,255), 2);
}
```

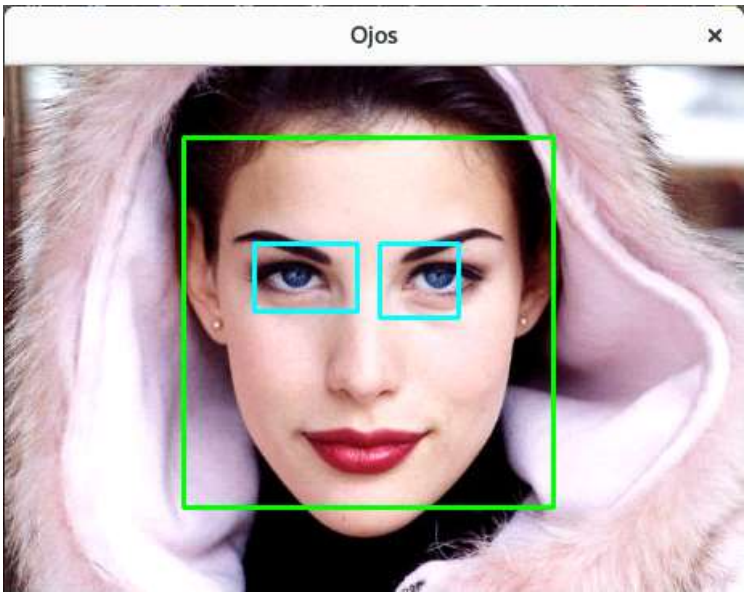


Figura 27. Detección de ojos en "images.jpg"
Fuente: captura de pantalla base de datos
Autor. Juan Toaquiza

En este punto se cierra la estructura condicional *If* de la programación para la detección de los ojos, y se continúa con la parte final de la programación de detección de rostros.

2.8.1.10. Acceso a la webcam

Con la ayuda de OpenCV se pudo acceder a la cámara que está instalada en el computador y procesar las imágenes capturadas, debido a que contiene diferentes funciones que permiten mencionada tarea, pudiendo cambiar una imagen a escala de grises o aplicarle otros algoritmos más complicados según nuestras necesidades.

a. Webcam desde OpenCV

Para acceder a la cámara se lo hizo de una manera muy sencilla. Se debe especificar un archivo de video a la clase *VideoCapture* de OpenCV, indicando el número de dispositivo que se desea usar. Se colocó el número cero si se tiene una sola cámara, de la siguiente manera:

```
VideoCapture cap(0);
```

2.8.1.11. Detección de rostros en tiempo real

Para detectar rostros en tiempo real se requirió leer las imágenes de la webcam y aplicarle a cada una de ellas el procedimiento descrito anteriormente.

En si la webcam envía una imagen tras otra. Lo que significa que se aplica el procedimiento para detectar un rostro a cada imagen una y otra vez.

No confundir. Esto no significa que se tenga una infinidad de rectángulos. El número de rectángulos que se acumularón en el vector corresponden a los rostros presentes ese instante en la imagen.

a. Programación para detección de rostros

Se proporcionó importancia a las líneas de código dedicadas a la detección de rostros en tiempo real, dejando como sobreentendido la programación para detección de rostro en una imagen.

Se inició definiendo la clase *VideoCapture* en el objeto *cap*. La clase *VideoCapture* sirve para la captura de vídeo a partir de archivos de vídeo, secuencias de imágenes o cámaras.

```
int main() {
VideoCapture cap;
```

El acceso a la cámara se colocó dentro de una estructura condicional *If*, mediante la cual comprobó si está conectado o no el dispositivo, caso contrario imprime un mensaje. Mediante la función *cap.open(0)* de OpenCV se abre la cámara por

defecto. El número cero si tenemos solo una cámara, indica el número de dispositivo que deseamos usar.

```
CascadeClassifier detector;
if(!detector.load("haarcascade_frontalface_alt.xml"))
cout << "No se puede abrir clasificador." << endl;
if(!cap.open(0))
cout << "No se puede acceder a la webcam." << endl;
```

b. Bucle while

Para continuar con el programa se crea un bucle que no acaba nunca, hasta forzarlo, en donde ira el procesamiento de las imágenes provenientes de la cámara. Se utiliza el ciclo repetitivo *While*, para salir de este se tiene que acabar el programa o la función principal.

```
while(true)
```

Mediante el operador de extracción leemos la información de *cap* y la almacenamos en variable indicada a la derecha, *imagen*

```
Mat dest, gray, imagen;
cap >> imagen;
```

c. DetectMultiescale

Mediante la función *DetectMultiescale* se comienza la detección de rostros, las cuales almacenamos en el vector *rect*

```
cvtColor(imagen, gray, CV_BGR2GRAY);
equalizeHist(gray, dest);
vector<Rect> rect;
detector.detectMultiScale(dest, rect, 1.2, 3, 0, Size(60,60));
```

Realizado esto se prosigue la programación normal que conlleva crear el ciclo *For* y la subsiguiente grafica del rectángulo

```
for(size_t i = 0; i < rect.size(); i++ )

{
rectangle(imagen,Point(rect[i].x, rect[i].y),Point(rect[i].x + rect[i].width, rect[i].y
+ rect[i].height),CV_RGB(0,255,0), 2);
}
```

Y finalmente se muestra el resultado en una ventana denominada “Detección de Rostros”

```
imshow("Detección de rostros", imagen);
```

2.8.1.12. Detección de ojos en tiempo real

En la detección de ojos en una imagen se observó que prácticamente era la programación de la detección de rostro más la determinación de la zona de los ojos y el uso de un clasificador en mencionado sector, con la elaboración de rectángulos sobre los ojos hallados.

En la detección de los ojos en tiempo real, es básicamente lo mismo. Solo tenemos que aumentar en la programación de detección de rostros en tiempo real, lo siguiente:

Datos para establecer la región de búsqueda. Esto dependerá del clasificador para ojos que se elija, en este caso será el *haarcascade_eye_tree_eyeglasses.xml* cuyos puntos para su búsqueda son:

```
 float EYE_SX = 0.16f;

float EYE_SY = 0.26f;

float EYE_SW = 0.30f;

float EYE_SH = 0.28f;
```

a. Clasificador

Se creó una condición para comprobar que tenemos disponible el clasificador

```
if(!eyes_detector.load("haarcascade_eye_tree_eyeglasses.xml"))
cout << "No se puede abrir clasificador para los ojos." << endl;
```

b. Condición

Condición principal para detectar los ojos

```
if(rect.size() > 0)
```

c. Clonación

Clonación y creación de dos vectores, uno para el ojo derecho y otro para el izquierdo

```
 Mat face = dest(rect[0]).clone();
vector<Rect> leftEye, rightEye;
```

d. Regiones de búsqueda

Extracción de los rectángulos que contienen la región para los ojos izquierdo y derecho

```
int leftX = cvRound(face.cols * EYE_SX);
int topY = cvRound(face.rows * EYE_SY);
int widthX = cvRound(face.cols * EYE_SW);
int heightY = cvRound(face.rows * EYE_SH);
int rightX = cvRound(face.cols * (1.0-EYE_SX-EYE_SW));
Mat topLeftOfFace = face(Rect(leftX, topY, widthX, heightY));
Mat topRightOfFace = face(Rect(rightX, topY, widthX, heightY));
Usar el clasificador en mencionadas zonas.
eyes_detector.detectMultiScale(topLeftOfFace, leftEye);
eyes_detector.detectMultiScale(topRightOfFace, rightEye);
```

e. Grafica de rectángulos

Se graficó de los rectángulos, tanto para el ojo izquierdo como para el derecho

```
if((int)leftEye.size() > 0)

{

rectangle(imagen,Point(leftEye[0].x + leftX + rect[0].x, leftEye[0].y + topY +
rect[0].y),

Point(leftEye[0].width + widthX + rect[0].x - 5, leftEye[0].height + heightY +
rect[0].y),

CV_RGB(0,255,255), 2);

}

if((int)rightEye.size() > 0)

{

rectangle(imagen,Point(rightEye[0].x + rightX + leftX + rect[0].x, rightEye[0].y +
topY + rect[0].y),Point(rightEye[0].width + widthX + rect[0].x + 5,
rightEye[0].height + heightY + rect[0].y),CV_RGB(0,255,255), 2);

}
```

2.8.1.13. Detección de cara, ojos, nariz y boca en tiempo real

Para la detección de todo lo mencionado anteriormente solo falta por añadir, a la actual programación, clasificadores para la nariz y boca. Y dibujar rectángulos sobre las detecciones

```
if(!nariz.load("haarcascade_mcs_nose.xml"))

cout << "No se puede abrir clasificador Nariz." << endl;

if(!boca.load("Mouth.xml"))

cout << "No se puede abrir clasificador boca." << endl;
```

Al no sectorizar se puede apreciar la falencia que presenta el método de Haar Cascade al no dar una región de búsqueda. Y no solo muestra problemas en la detección dentro de la zona del rostro, también reacciona de forma errónea, detectando cualquier objeto fuera de esta zona

Se proporcionó una región de búsqueda dentro del área del rostro es de vital importancia, porque los clasificadores no son efectivos si no se sigue este procedimiento. Por lo cual, se tomó la tarea de buscar las regiones para la boca y nariz, siendo los siguientes los puntos más afectivos:

Naris

```
float Nose_SX= 0.25f;
float Nose_SY= 0.20f;
float Nose_SW= 0.40f;
float Nose_SH= 0.50f;
```

Boca

```
float Mou_SX= 0.27f;
float Mou_SY= 0.79f;
float Mou_SW= 0.42f;
float Mou_SH= 0.16f;
```

2.8.1.14. Detección de la nariz

Al tener los puntos para la región de búsqueda, se realizó el mismo procedimiento que se usó para graficar en la zona de los ojos. Para la región de la nariz, queda de la siguiente manera:

```
if(rect.size() > 0)
{
Mat cara = dest(rect[0]).clone();
vector<Rect> lana;
int left1X = cvRound(cara.cols * Nose_SX);
int top1Y = cvRound(cara.rows * Nose_SY);
int width1X = cvRound(cara.cols * Nose_SW);
int height1Y = cvRound(cara.rows * Nose_SH);
Mat ParteMediaCara = cara(Rect(left1X, top1Y, width1X, height1Y));
nariz.detectMultiScale(ParteMediaCara, lana);
if((int)lana.size() > 0)
{
rectangle(imagen,
Point(lana[0].x + left1X + rect[0].x, lana[0].y + top1Y + rect[0].y),
Point(lana[0].width + width1X + rect[0].x - 5, lana[0].height + height1Y +
rect[0].y),
CV_RGB(0,0,255), 2);
}
}
```

Como se puede observar es exactamente igual a la programación para detección de los ojos, salvo que en esta ocasión solo se grafica un rectángulo.

2.8.1.15. Detección de la boca

Para la boca, de igual manera solo se variaron los puntos para la región. Se debe de tener mucho cuidado de no repetir las variables, porque el proceso puede ser idéntico, pero las variables fueron diferentes.

2.8.1.16. Detección de expresiones faciales con OpenCV

Varias expresiones faciales se pudieron dar, pero se tomó mayor interés a aquellos que se manifiestaron de manera muy pronunciada en el movimiento de las partes de la cara que fue posible captar con las herramientas de OpenCV y el método HaarCascade. Estas vieneron a ser los estados de ánimo como el estado neutral o seriedad, felicidad, tristeza y enojo

Cabe recalcar que mencionadas detecciones no cuentan con un gran nivel de precisión, al contrario, se debe de tratar de ubicar el rostro a una cierta distancia y gozar de condiciones de luz apropiadas, no muy intensa, ni muy baja.

2.8.1.17. Detección de la felicidad

Este estado de ánimo se caracteriza por la generación de una sonrisa, por lo que fue el objetivo a alcanzar. Se debe mencionar que OpenCV cuenta con un archivo .xml destinado para esta tarea, *haarcascade_smile.xml*, el cual después de innumerables pruebas no cumplió con lo esperado, debido que solo captaba rostros, como si se tratase de un detector de rostros; por lo que se tuvo que encontrar la manera de ocupar las detecciones obtenidas hasta este punto.

Se notó que al momento de sonreír, la boca en concreto los labios, cambian de posición, se elevan un poco. Esto se aprovechó colocando el detector de bocas en una región tan ajustada, que al momento de sonreír, la boca salga de la región de detección y esta quede vacía, por lo cual el vector donde se almacenan las detecciones de la boca tendrá una valor de cero.

```
if((int)labo.size() == 0)
```

De esta forma se sabe que está sonriendo. Y caso contrario, cuando se detecte algo, habrá algún valor en el vector, de esta manera se sabe que está en un estado neutral o de seriedad.

```
if((int)labo.size() > 0)
```

Para mostrar dicha estado de ánimo se colocó un pequeño dibujo que representó la seriedad y la felicidad, en lugar de dibujar o no un rectángulo.

Figura 28. Gráficos para detección de seriedad y felicidad
Fuente. http://misimagenesde.com/tag/imagenes-de-caritas/

Dichos gráficos fueron mostrados en conjunto con la imagen principal a través de la siguiente programación:

```
Antes de la función principal int main()
char name0[]="Serio.png";
char name1[]="Feliz.png";
Después de la función principal int main()
IplImage* imag=NULL;
imag=cvLoadImage(name0,1);
cvNamedWindow( "Seriedad", 1);
IplImage* imag1=NULL;
imag1=cvLoadImage(name1,1);
cvNamedWindow( "Happy", 1);
Y  se la muestra en la pantalla de la siguiente manera:
cvShowImage( "Seriedad", imag );
cvShowImage( "Happy", imag1 );
```

a. Coordenadas:

Los puntos para encontrar la región de búsqueda anterior no varían mucho con respecto a los presentes; se los aprecia a continuación:

Coordenadas anteriores:

```
float Mou_SX= 0.27f;
float Mou_SY= 0.79f;
float Mou_SW= 0.42f;
float Mou_SH= 0.16f;
```

Coordenadas actuales:

```
float Mou_SX= 0.30f;
float Mou_SY= 0.80f;
float Mou_SW= 0.40f;
float Mou_SH= 0.15f;
```

2.8.1.18. Detección de la tristeza

Existen varias características que definen esta expresión. Se consideró a una mirada baja como la principal particularidad en la tristeza. Con lo aprendido hasta el momento se puede enfocar en la región de los ojos, y ajustar de tal manera que brinde algún tipo de variación al presenciar una mirada baja.

Para esta parte se tomó en cuenta un solo ojo (izquierdo), bebido a que la presencia de ambos genera una gran inestabilidad en las detecciones, negando en su totalidad el manejo de las detecciones para mostrar algo concreto en pantalla.

a. Detector

Se hizo necesario el cambio de detector para los ojos, debido a que "*haarcascade_frontalface_alt2.xml*" escogido al azar, no sirve para este propósito.

Para escoger el nuevo detector se tomó en cuenta que este no reaccione con los ojos cerrados, por lo cual se escogió "*haarcascade_eye.xml*".

Se varía las coordenadas en la región de búsqueda, colocándola ajustadamente en la región de los ojos

Antes

```
float EYE_SX = 0.16f;
float EYE_SY = 0.26f;
float EYE_SW = 0.30f;
float EYE_SH = 0.28f;
```

Después

```
float EYE_SX = 0.15f;
float EYE_SY = 0.19f;
float EYE_SW = 0.30f;
float EYE_SH = 0.28f;
```

Se colocó una gráfica que distinga la tristeza. La programación varía muy poco con respecto a la sonrisa.

Figura 29. Grafico para detección de tristeza
Fuente. http://misimagenesde.com/tag/imagenes-de-caritas/

```
char name3[]="Tristeza.png";
IplImage* imag2=NULL;
imag2=cvLoadImage(name2,1);
cvNamedWindow( "Seriedad", 1);
cvShowImage("Seriedad",imag2);
```

2.8.1.19. Detección del enojo

El descenso de las cejas fue la principal singularidad que presentó esta expresión facial. En este punto se presentó un inconveniente porque no se pudo detectar las cejas con los conocimientos y técnicas adquiridas hasta ese momento.

Una solución a este problema fue escoger una región de la cara en donde se notó un mayor cambio al momento del enojo y pasar un detector, no importa cuál, que fuera capaz de variar con el movimiento en esta zona.

a. Región de búsqueda

La región comprendió parte de las cejas y el espacio de rostro que hay entre ellas. Estas son las coordenadas para dicho fin.

```
float ENO_SX = 0.35f;
float ENO_SY = 0.20f;
float ENO_SW = 0.26f;
float ENO_SH = 0.40f;
```

Luego de innumerables pruebas, se tuvo un resultado respetable con el archivo *haarcascade_frontalcatface.xml*, el cual reacciona casi todas las veces cuando hay un movimiento en la zona delimitada. El grafico distintivo para esta expresión es:

Figura 30. Grafico para detección de enojo
Fuente. http://misimagenesde.com/tag/imagenes-de-caritas/

Cuya programación fue básicamente la misma que en las dos expresiones mostradas anteriormente.

2.8.1.20. Detección de la furia

Completamente igual al caso anterior, enojo. Lógicamente cambió la zona de análisis y el archivo *xml* del detector que en este caso será *haarcascade_mcs_mouth.xml*. Cabe recalcar que para llegar a determinar tanto la zona de búsqueda así como el analizador, se tuvo que experimentar con una infinidad de combinaciones entre diferentes zonas y todos los clasificadores de OpenCV de la versión 3.1.0 así como de la versión 2.4.9 mencionando que dicha versión trae el mayor número de detectores. Esta tarea se realizó para los casos de enojo y furia

a. Zona de búsqueda

La zona de delimitación comprendió parte del ojo izquierdo y la nariz, siendo las coordenadas las siguientes:

```
float ENO_SX = 0.32f;
float ENO_SY = 0.32f;
float ENO_SW = 0.24f;
float ENO_SH = 0.40f;
```

Misma programación para mostrar la imagen característica, la cual es:

Figura 31. Grafico para detección de furia
Fuente. http://www.infoolavarria.com/2015/07/13/

2.8.1.21. Funciones

Para una mayor fluidez del programa se vio la necesidad de usar funciones, específicamente en la parte que corresponde al grafico distintivo de cada expresión, debido a que generaba un retardo y confusión en la programación. Se utilizó de referencia el caso Seriedad y Felicidad

a. Declaración

Declaración de las funciones:

```
void ImprimirSerio();
void ImprimirFeliz();
```

b. Llamado

Llamado a la función:

```
ImprimirSerio();
ImprimirFeliz();
```

c. Ejecución

La Función, puede ubicarse en una parte del programa que sea conveniente, en este caso se ubicó al final

```
void ImprimirSerio()
  {
IplImage* imag=NULL;
imag=cvLoadImage(name0,1);
cvNamedWindow( "Seriedad", 1);
cvShowImage( "Seriedad", imag );
  }
void ImprimirFeliz()
  {
IplImage* imag1=NULL;
imag1=cvLoadImage(name1,1);
cvNamedWindow( "Happy", 1);
cvShowImage( "Happy", imag1 );
  }
```

CAPITULO III

3. RESULTADOS

3.1. Funcionamiento de la cámara con OpenCV

Cheese es una aplicación para el uso de la cámara web en fotos y videos. Viene por defecto con Fedora 24.

Metódicamente no existió diferencia en la imagen que muestró la cámara a través de Cheese y la que se indicó a través de OpenCV. Por lo que se pudo trabajar con OpenCV sin ningún problema.

Figura 32. Imagen de WebCam con OpenCV (captura de pantalla)
Elaborado por Autor

3.2. Detección de rostros en tiempo real

Con el método de HaarCascade la detección de rostros en tiempo real no mostró ningún inconveniente, pudiendo reaccionar rápidamente a distancia corta de 20 cm

y a una máxima de 1.5m, que fueron las distancias tomadas de la frente del programador hasta la cámara de la computadora.

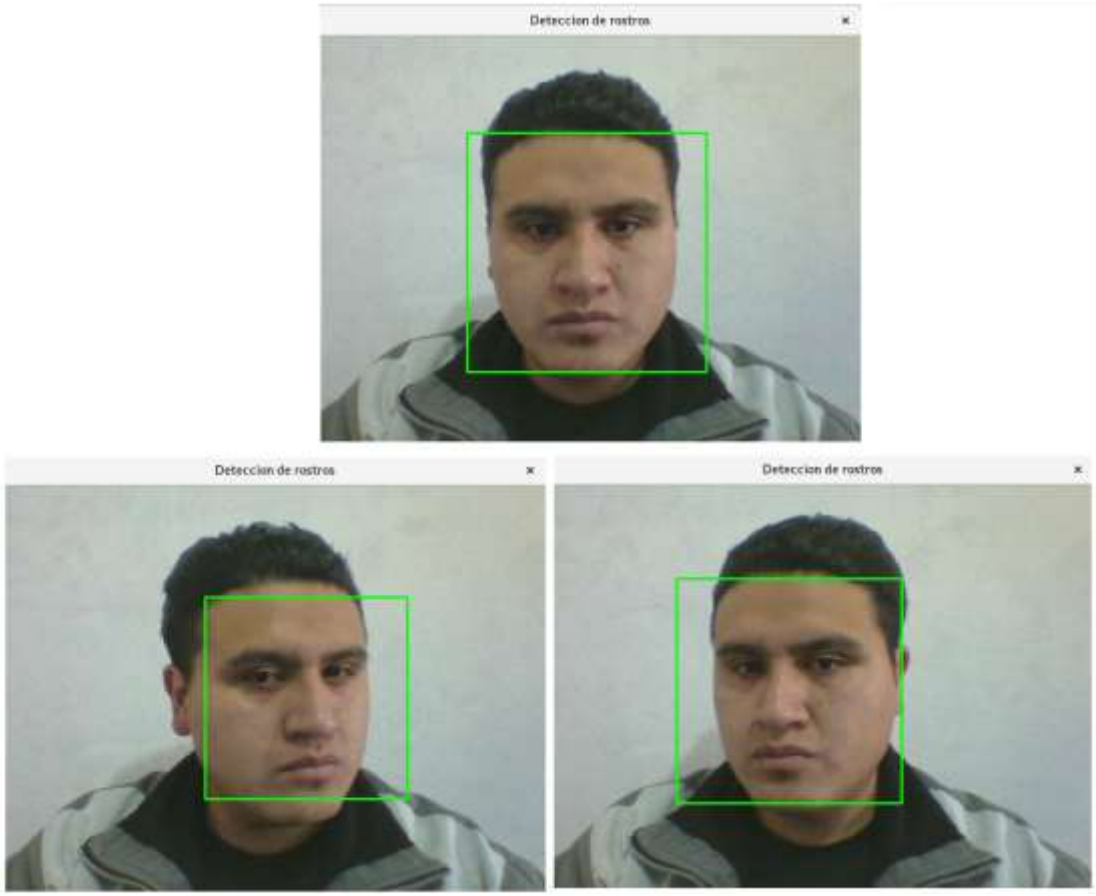


Figura 33. Detección de rostro. De frente y costado (captura de pantalla)
Elaborado por Autor

3.3. Detección de ojos en tiempo real

Al igual que en el caso del rostro, la detección de los ojos no presentó mayor inconveniente, puesto que reaccionó a una distancia de 20cm hasta una máxima de 1m o 1.5 m

Reiterando que tanto para el rostro como para los ojos las condiciones de luz resultaron más flexibles.

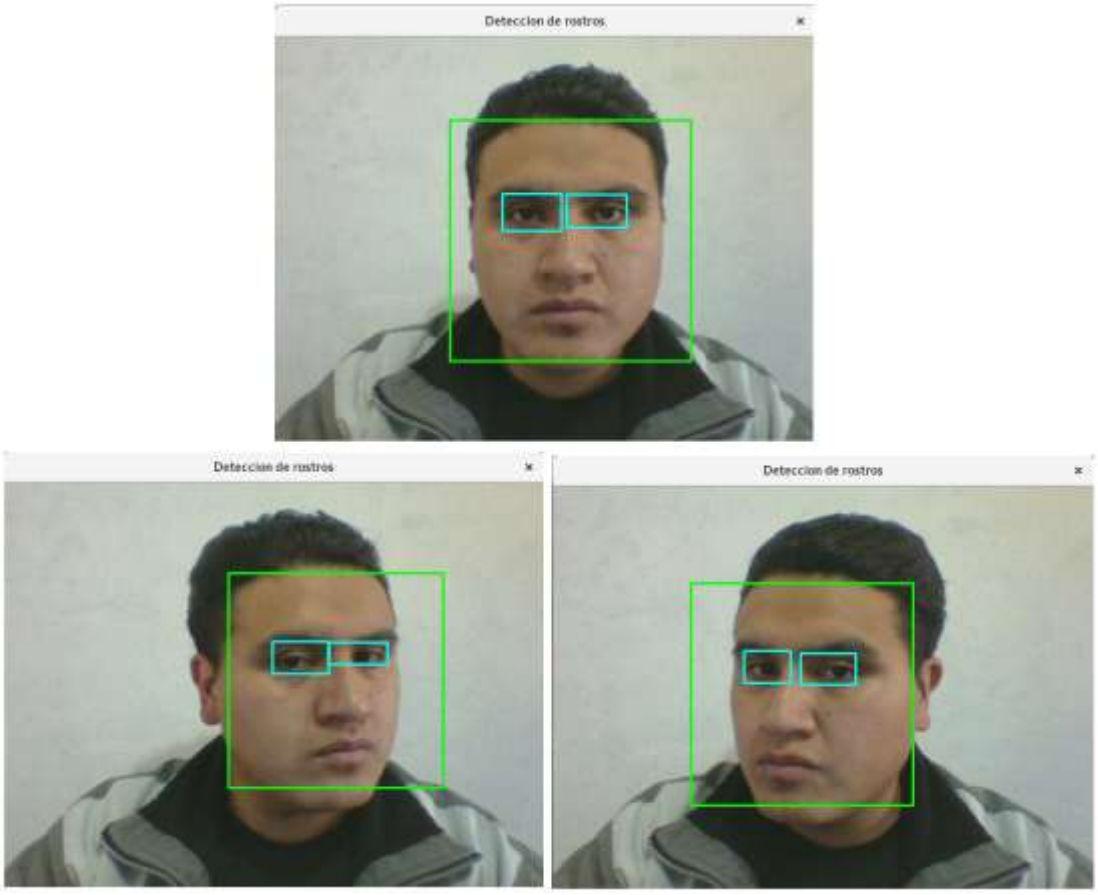


Figura 34. Detección de rostro y ojos. De frente y costado
Elaborado por Autor

3.4. Detección de cara, ojos, nariz y boca en tiempo real

En primer lugar, no se realizó sectorización para enmarcar la inexactitud que presenta el método de Haar Cascade al no proporcionar una región de búsqueda.

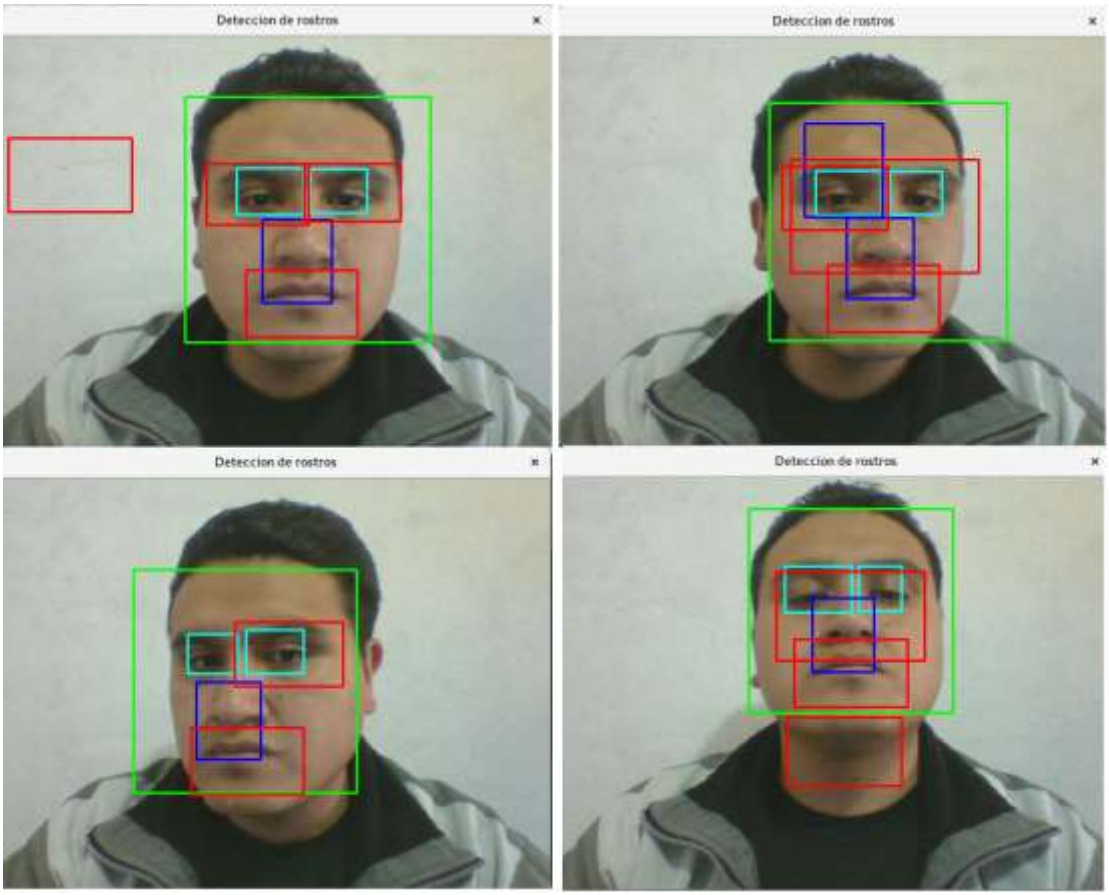


Figura 35. Detección de boca y nariz sin región de búsqueda
Elaborado por Autor.

Sin sectorización este método no solo demostró problemas en la detección dentro de la zona del rostro, también reaccionó de forma errónea, detectando cualquier objeto fuera de esta zona

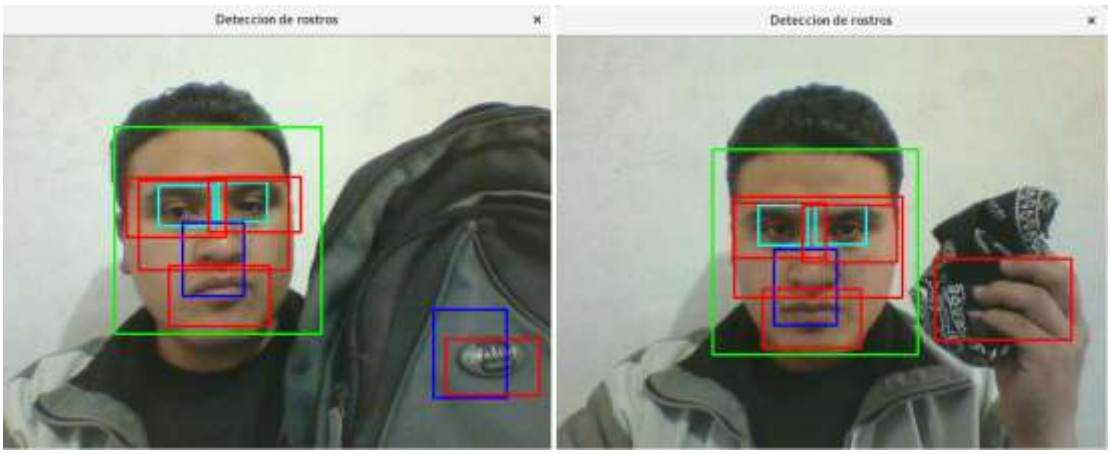


Figura 36. Detección de boca y nariz sin región de búsqueda, con agentes externos
Elaborado por Autor.

Por lo cual se realizó sectorización del rostro, y de esta manera se proporcionó región de búsqueda dentro del área requerida, siendo este proceso de vital importancia

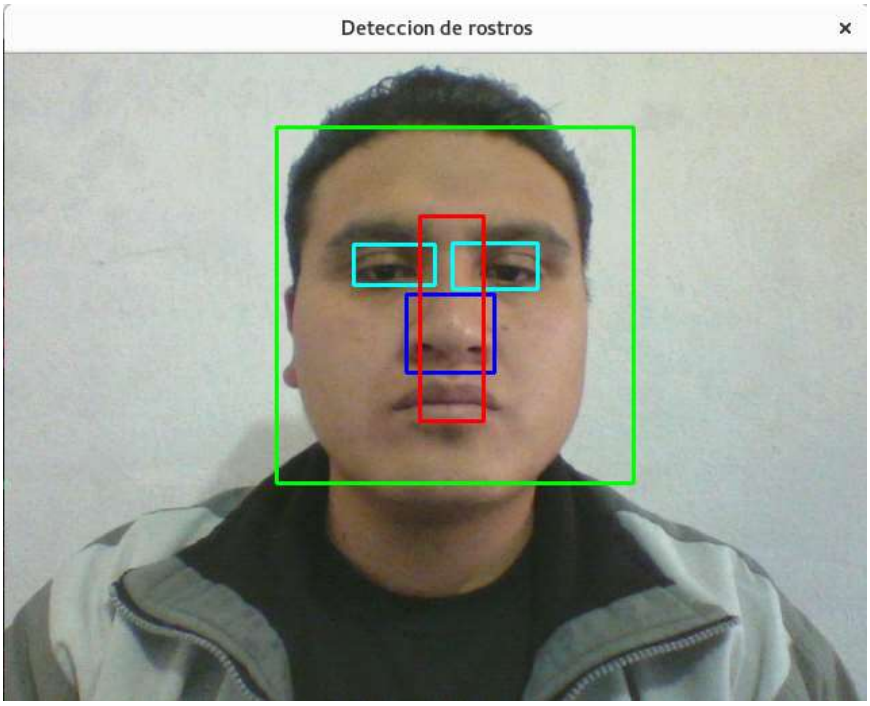


Figura 37. Detección de rostro, ojos, nariz, boca, con región de búsqueda
Elaborado por Autor.

Los recuadros azul y rojo representaron la nariz y boca respectivamente. Esta aclaración se realizó debido a que el recuadro rojo que representaba la boca al momento de realizar la gráfica se desproporcionaba y no era posible modificarlo.

3.5. Detección de expresiones faciales con OpenCV

Para la detección de expresiones faciales fue necesario el análisis por separado de cada expresión facial, porque el proceso de funcionamiento demandó condiciones de distancia e iluminación individuales

3.5.1. Felicidad

Para la detección de las expresiones faciales en primer lugar se empezó por un estado neutral o seriedad, el cual se muestra a continuación:

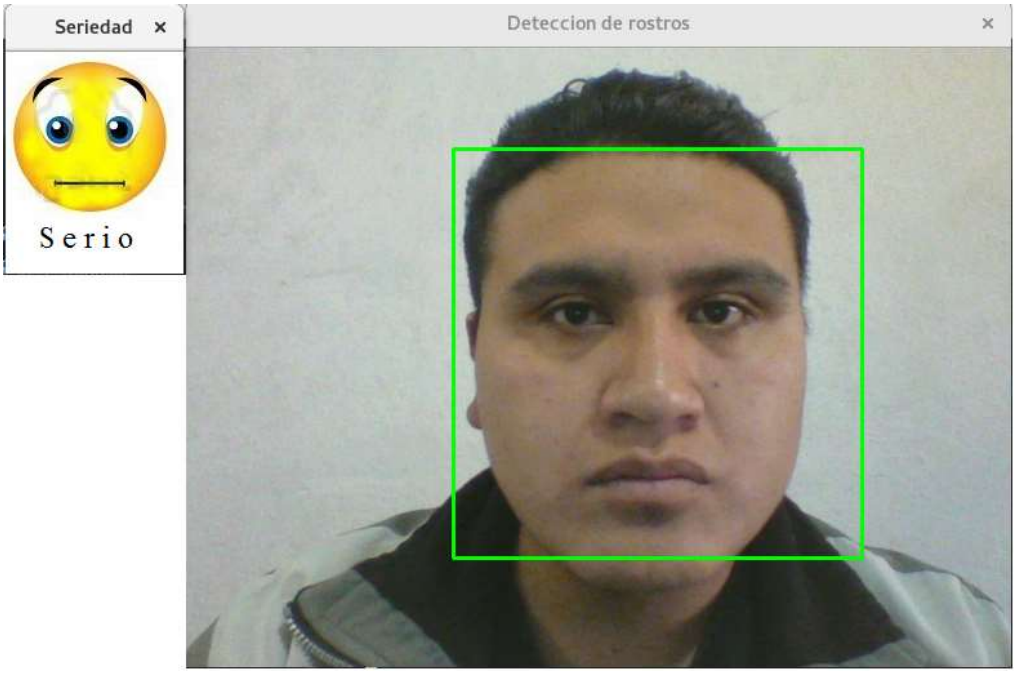


Figura 38. Detección de seriedad o estado neutral
Elaborado por Autor.

La distancia adecuada para la detección de la felicidad fue en un rango de 41 a 42 cm; medida tomada desde la frente del individuo hasta la cámara del computador.

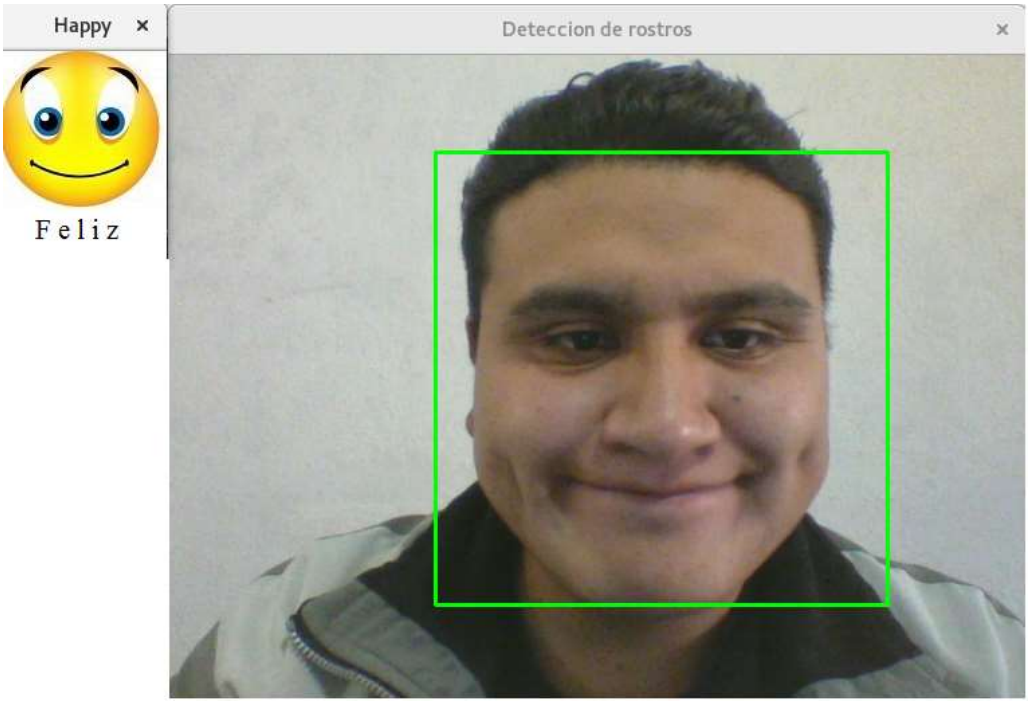


Figura 39. Detección de la felicidad
Elaborado por Autor.

3.5.2. Tristeza

La distancia adecuada para la detección de la tristeza fue de 42.5 cm a 43.5 cm.

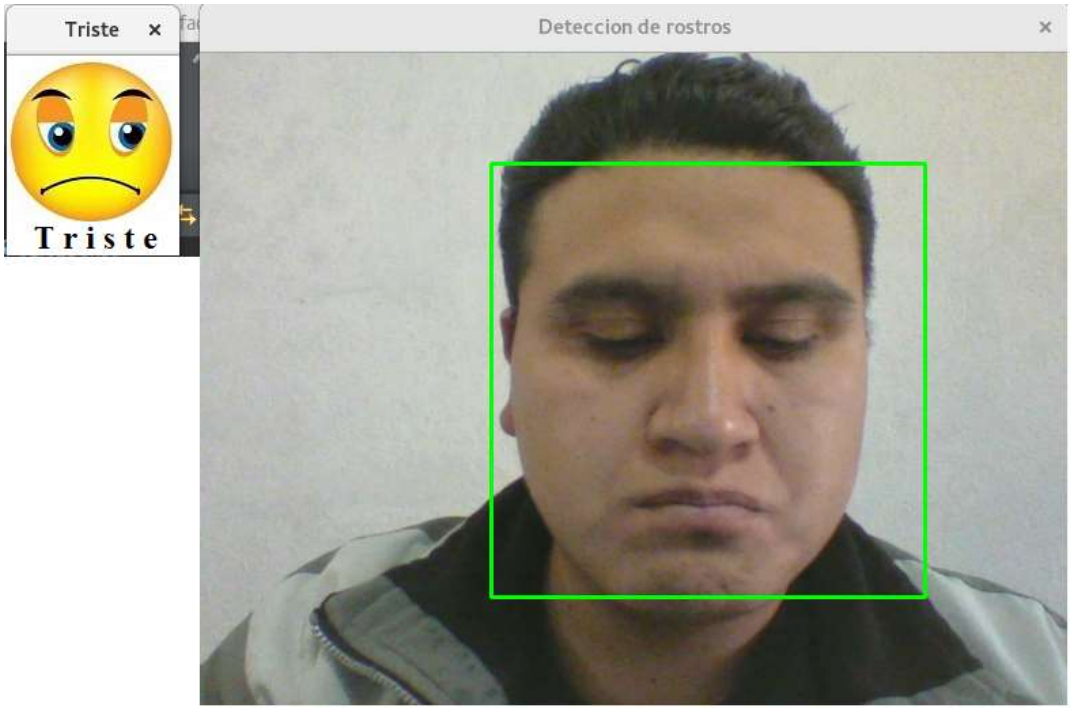


Figura 40. Detección de la tristeza
Elaborado por Autor.

3.5.3. Enojo

Esta expresión facial fue de mayor inconveniente, porque no reaccionó a distancias similares, la distancia aproximada fue en un rango de 45cm a 49 cm.

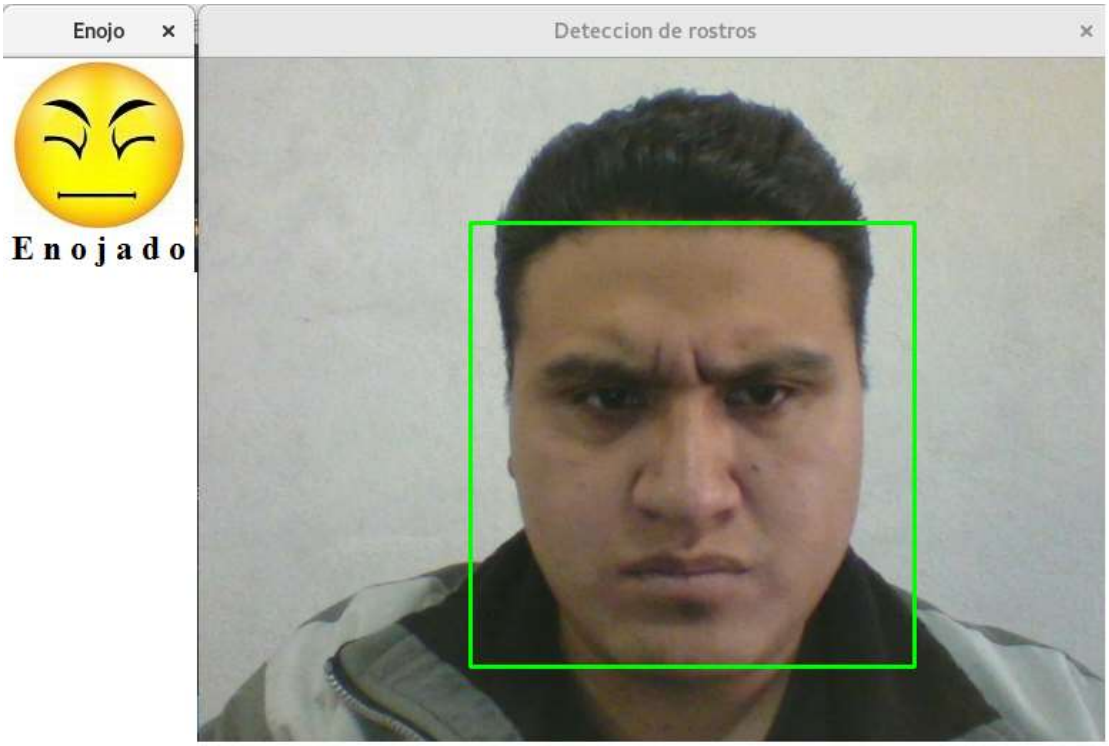


Figura 41. Detección del enojo
Elaborado por Autor.

3.5.4. Furia

Para la detección de la expresión facial la distancia adecuada fue en un rango de 33 cm a 34cm.

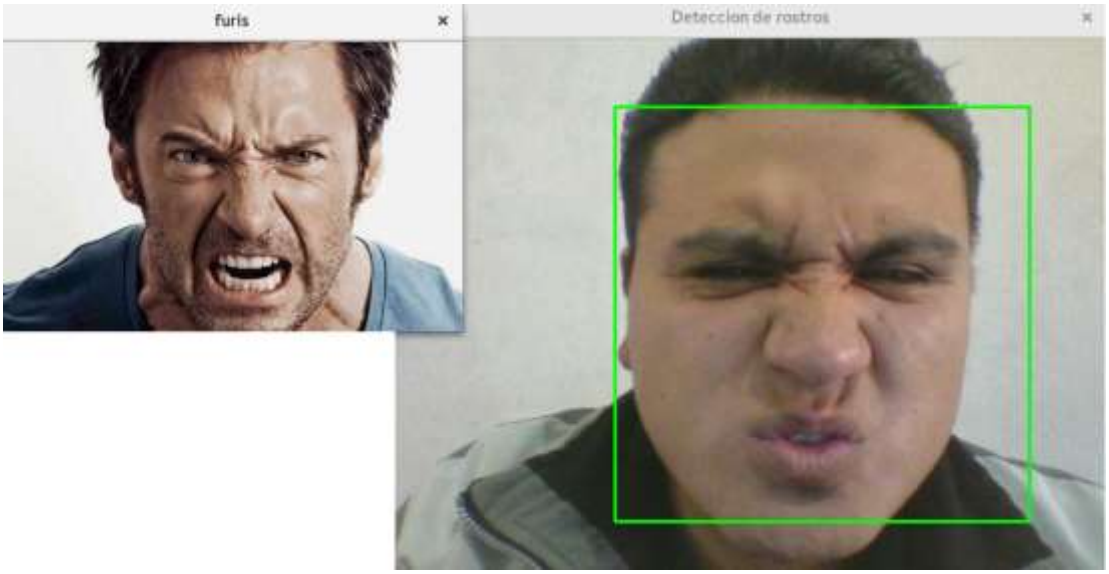


Figura 42. Detección de furia
Elaborado por Autor

3.6. Condiciones de iluminación

En la realización de la investigación se determinaron las condiciones óptimas de iluminación natural y artificial.

3.6.1. Iluminación natural

La fuente de iluminación natural fue proporcionada por una ventana de 1.40 m de ancho y 1.20 m de alto.

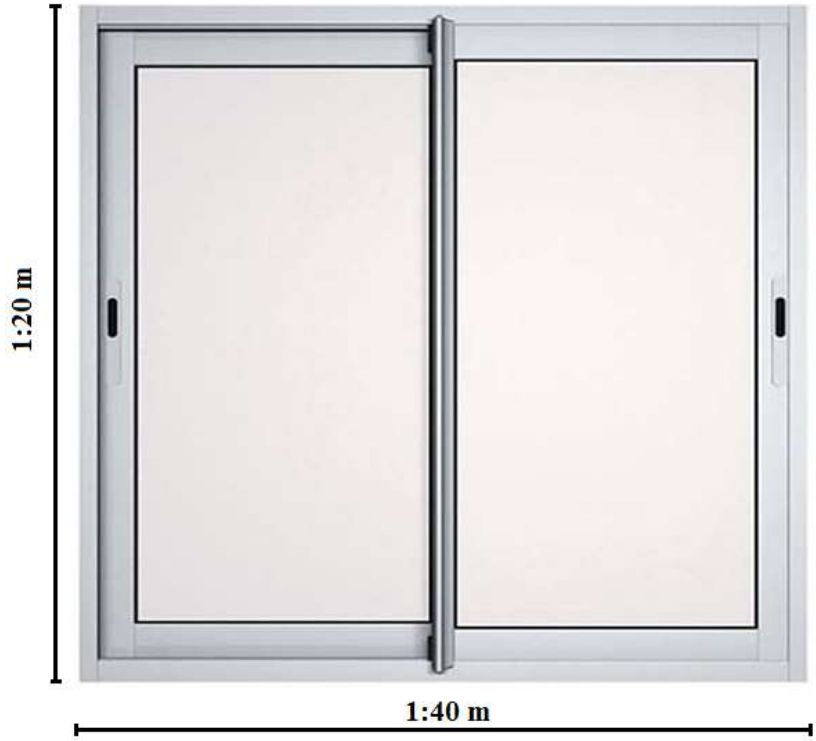


Figura 43. Iluminación natural
Elaborado por Autor

3.6.2. Iluminación artificial

Fue proporcionada por una lámpara con un bombillo incandescente de 60 watts que irradiaba luz amarilla ubicada a conveniencia para las detecciones.

Figura 44. Iluminación artificial
Elaborado por Autor

3.6.3. Medición de la iluminación

Para determinar el nivel de iluminación se utilizó un luxómetro, que fue un instrumento especializado para esta tarea.

Figura 45. Luxómetro CEM DT-1309

El sensor de este instrumento se colocó en la parte frontal del rostro de prueba para la realización de las detecciones.

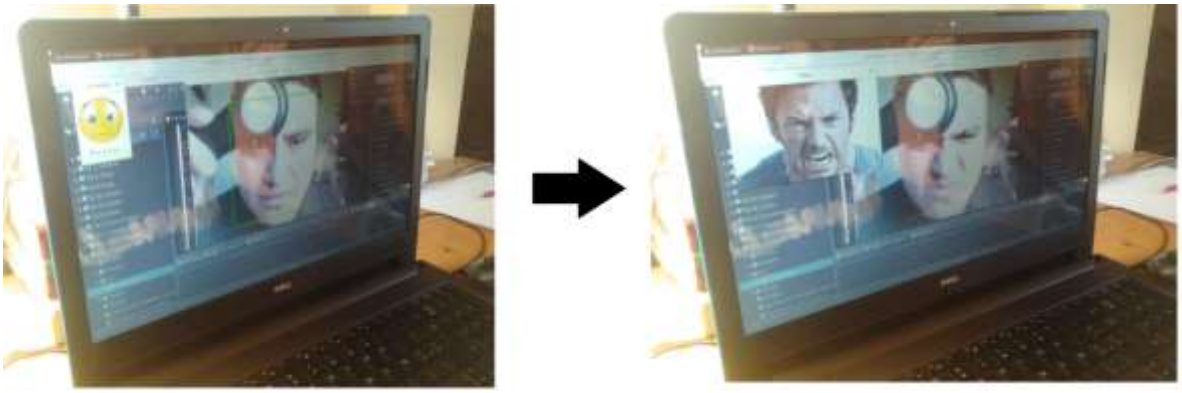


Figura 46. Medición del nivel de iluminación
Elaborado por Autor

Los niveles de iluminación para cada expresión facial fueron los siguientes:

Tabla 3. Medición de niveles de iluminación

Expresión facial	Nivel de iluminación (lux)			
	M1	**M2**	**M3**	**Promedio**
Sonrisa	101	100	98	99.7
Triste	90	88	86	88.0
Enojo	147	144	143	144.7
Furia	141	138	135	138.0

Datos tomados de la base de datos
Elaborado por Autor

3.7. Comprobación de la hipótesis

Para la comprobar la hipótesis se utiliza el método estadístico Chi-Cuadrado, que accede a dos grados de posibilidad, alternativa la que se quiere comprobar y nula (rechaza la hipótesis alternativa) para constatar en cumplimiento de la hipótesis

H_0: El sistema de reconocimiento de expresiones faciales no es aplicable a la interacción Humano-Robot

H_1: El sistema de reconocimiento de expresiones faciales es aplicable a la interacción Humano-Robot

3.7.1. Determinación del valor estadístico de prueba.

Para aceptar o rechazar la hipótesis se tomaron en cuenta dos escenarios, el primero, que toma en cuenta el método HaarCascade sin dar una región de búsqueda. El segundo escenario que brinda los datos del método HaarCascade con una región de búsqueda

El resultado de las detecciones conforma la frecuencia observada. Se calcula el total de filas y columnas, las cuales sumadas deben de dar un mismo valor al que se le conoce como gran total, en este caso el valor es 30

Tabla 4. Frecuencia observada y gran total de filas y columnas

	Método (x)		
Detección (y)	HaarCascade sin región	HaarCascade mas región	Total
Rostro	5	5	10
Ojos	3	5	8
Nariz	0	5	5
Boca	2	5	7
Total	10	20	30

Fuente. Datos tomados de la base de datos
Elaborado por autor

Para encontrar la frecuencia esperada se realiza una regla de tres, para lo cual se multiplica el total de cada columna, por el total de cada fila, y se divide entre el total de cada fila y columna,

Tabla 5. Frecuencia esperada

	Método (x)	
Detección (y)	HaarCascade sin región	HaarCascade con región
Rostro	3.333	6.666
Ojos	2.666	5.333
Nariz	1.666	3.333
Boca	2.333	4.666

Fuente. Datos tomados de la base de datos
Elaborado por autor

Con la tabla completa se procede a calcular Chi Cuadrado, mediante la Formula:

$$x^2 = \sum \frac{(O-E)^2}{E}$$

Donde O es la frecuencia observada y E la frecuencia esperada. Se debe multiplicar el número de filas por el de las columnas para sacar el número de posibles combinaciones entre estas. Dicho dato es de utilidad para formar la tabla XXX, que relaciona el método con la parte detectada, y sigue los pasos para hallar el valor de Chi Cuadrado expuesto en la formula

Tabla 6. Componentes Chi-Cuadrado

Casillas (x/y)	O	E	(O-E)	(O-E)^2	((O-E) ^2)/E
HaarCascade sin región Rostro	5	3.333	1.667	2.779	0.8333
HaarCascade con región Rostro	5	6.666	-1.666	2.776	0.416
HaarCascade sin región Ojos	3	2.666	0.334	0.112	0.042
HaarCascade con región Ojos	5	5.333	-0.333	0.111	0.020
HaarCascade sin región Nariz	0	1.666	-1.666	2.776	1.666
HaarCascade con región Nariz	5	3.333	1.667	2.779	0.813
HaarCascade sin región Boca	2	2.333	-0.333	0.111	0.047
HaarCascade con región Boca	5	4.666	0.334	0.112	0.024
				∑	3.881

Fuente. Datos tomados de la base de datos
Elaborado por autor

El valor de la sumatoria final es el valor de Chi cuadrado

Para verificar cuál de las hipótesis es la adecuada se debe calcular los grados de libertad, que se obtiene restando la unidad al número de filas y de igual manera al número de columnas para finalmente multiplicar los resultados. En este caso el resultado es 3 que viene a ser el grado de libertad

Tabla 7. Grados de libertad vs Probabilidad

	Probabilidad				
Grados de Libertad	0.9	0.5	0.1	0.05	0.01
1	0.02	0.46	2.71	3.84	6.64
2	0.21	1.39	4.61	5.99	9.21
3	0.58	2.37	6.25	7.82	11.35

Fuente. Datos tomados de la base de datos
Elaborado por autor

El grado alfa hace referencia al nivel de confianza que se desea para los cálculos de la prueba. Se trabaja con un grado alfa de 0.5, y la relacionamos con grado de libertad 3. En este caso, para aceptar la hipótesis nula, el resultado de Chi Cuadrado debía tener un valor de hasta 2.37. Al no ser este el caso se puede desechar la hipótesis nula y tomar la hipótesis alternativa.

El sistema de reconocimiento de expresiones faciales es aplicable a la interacción Humano-Robot

CAPITULO IV

4. DISCUSIÓN

El propósito de este trabajo de investigación fue identificar expresiones faciales para que en un momento dado sirva para la interacción entre el humano y un robot, como resultado productivo. Sin embargo quedan fomentadas, las bases para un estudio más profundo con otros métodos y herramientas que permitan un mejor desarrollo de la visión por computador, cuya influencia en el avance tecnológico es innegable, al igual que su uso tanto a nivel académico, comercial e industrial.

Lo primordial fue pretender el desarrollo de este sistema con herramientas poco habituales en el medio, haciendo uso del software libre, cuyo aprendizaje y manejo cubre gran parte del trabajo de investigación. Tomando en cuenta la falta de información sobre el tema.

De los resultados obtenidos con el programa se incorporó como tema de discusión la eficacia del método HaarCascade para la detección de expresiones faciales, o si este constituye solo un excelente detector que fue herramienta útil para encontrar las diferentes partes del rostro.

Culminando este acápite se presenta la controversia existente entre el uso de material libre o el privado, y que elección es mejor para el desarrollo y mejoras en la visión por computador

CAPITULO V

5. Conclusiones y recomendaciones

5.1.Conclusiones

- La elaboración de este trabajo de investigación ha representado un reto interesante, no solo en lo académico sino también a nivel personal. La dedicación y esfuerzo requeridos, han otorgado una nueva visión para la solución de futuros problemas. Los individuos inmiscuidos en la elaboración de esquemas de esta categoría, son conscientes de los inconvenientes y malos momentos que se atraviesan al hacer compilar un programa. Cabe destacar que la solución propuesta en este trabajo de investigación para el reconocimiento de expresiones faciales, no es la única o la más viable, pero si la que se ajustó al conocimiento, recursos económicos y humanos disponibles en este periodo.
- El reconocimiento facial resulta muy útil en cualquier campo de estudio, pero su elaboración requiere de esfuerzo, dependiendo de las herramientas y métodos que se utilicen para este fin. Destacando la riqueza en conocimientos adquiridas durante la investigación sobre el sistema operativo, IDE, y demás mecanismos utilizados en este trabajo.
- Posterior a las pruebas realizadas, la detección del rostro funciona adecuadamente, sin embargo con las detecciones posteriores no se obtuvieron el mismo resultado, evidenciando notables fallas ante cualquier condición adversa, sea esta, baja o alta iluminación, rostros no precisamente frontales, distancia entre el rostro y la cámara. Todo esto conlleva a pensar que este estudio sirve como base para proyectos afines en donde se apliquen conocimientos avanzados y desarrollados en el campo de visión por computador
- El método HaarCascade o clasificadores en cascada no es infalible, debido a que resulta inútil al no especificar una zona de búsqueda para el clasificador, por lo cual queda demostrado que para un reconocimiento de expresiones

faciales tiene muchas limitaciones, pero su conocimiento y manejo es esencial para cualquier estudio de esta naturaleza.

- Finalmente podemos concluir manifestando que se ha cumplido con los objetivos propuestos, en un período de tiempo aceptable y con la optimización de recursos, y las situaciones adversas fueron superadas.

5.2. **Recomendaciones**

- Se recomienda que para la reproducción del trabajo de investigación se debe cerciorar de las características técnicas y físicas del equipo a utilizar, las mismas que deben cumplir satisfactoriamente con el procesamiento de imágenes
- Al momento de realizar las pruebas, cuando se trabaje con luz natural, debe asegurarse que sean durante las horas del día en la cual la luz sea óptima. Si trabaja con luz artificial, esta debe mantener un nivel apropiado para que no influya en los resultados. Además se debe de tener cuidado con la presencia de fuertes fuentes de luz, sean estas en los costados o en la parte posterior del individuo, porque repercute de alguna manera en la detección de la imagen. Las pruebas durante la noche no son recomendables, resultan engañosas y nada fiables.
- En cuanto al software, este no debe ser una barrera, más bien una herramienta útil, por lo que se debe cuidados exhaustivos en su elección y seleccionar aquel que presente un manejo dócil y adecuado, exceptuando en casos extremos en los se deba utilizar aquellos que se desconozcan.

CAPITULO VI

6. PROPUESTA

6.1. Título de la propuesta

Reconocimiento de estados emocionales en tiempo real con las librerías OpenCV a través de la herramienta de software Matlab

6.2. Introducción

En los últimos tiempos el reconocimiento facial se ha convertido en un área de investigación que tiene diferentes ramas de estudio como la clasificación de género, reconocimiento de emociones, evaluación de edad, etc. Involucrando a estudiosos de muchos campos y especializaciones que han encontrado en la visión por computador el punto donde aplicar sus conocimientos para el desarrollo de tecnología

El sistema comenzara funcionando en imágenes para posteriormente añadir dificultad al proceso realizando la detección en tiempo real de manera automática.

Las bibliotecas OpenCV son libres, especializadas en visión por computador utilizadas en una infinidad de aplicaciones. Es multiplataforma siendo funcional para Linux, Windows e inclusive Mac. Para lograr óptimos resultados con este proyecto, las librerías OpenCV necesita de una interface de desarrollo con la cual pueda ser compatible. En este caso se ha optado por Matlab, cuyo entorno fácil de usar permite un avance sin muchas complicaciones en proyectos de cualquier índole.

El reconocimiento de estados emocionales a través de Matlab con el uso de librerías OpenCV, pretende crear un análisis profundo en la forma de interactuar entre estas

dos herramientas, con una ligera comparación de este entorno de desarrollo con otros.

6.3.Objetivos

6.3.1. Objetivo General

Reconocer expresiones faciales en tiempo real con OpenCV a través de Matlab

6.3.2. Objetivos Específicos

- Investigar sobre los métodos para la detección de rostros con OpenCV
- Seleccionar el método adecuado para la elaboración del algoritmo
- Buscar el camino para el correcto funcionamiento entre OpenCV y Matlab
- Desarrollar una interface gráfica de usuario con Guide de Matlab
- Mediante el uso de Matlab facilitar el desarrollo de este tipo de proyectos

6.4.Fundamentación Científico-Técnico

El sistema de reconocimiento para estados emocionales en tiempo real con las librerías OpenCV y Matlab , está encaminada a la correcta detección de los rasgos faciales y la variación que estos presentan en ciertas circunstancias, tratando de hacer este proceso más fácil con la utilización de la herramienta Image Processing Toolbox de Matlab, la cual proporciona un amplio conjunto de algoritmos estándar de referencia, funciones y aplicaciones para el procesamiento de imágenes, análisis, visualización y desarrollo de algoritmos. Puede realizar análisis de imágenes, segmentación de imágenes, mejora de imagen, reducción de ruido, transformaciones geométricas, y el registro de imágenes. Soporta un conjunto diverso de tipos de imágenes, y posee aplicaciones que le permiten explorar imágenes y vídeos, examinar una región de píxeles, ajustar el color y el contraste, crear contornos o histogramas y manipular las regiones de interés.

6.5.Descripción de la propuesta

Este proyecto consiste en la elaboración de un sistema que reconozca estados emocionales, partiendo desde la implementación de las librerías OpenCV en Matlab, para de esta manera beneficiarnos de las facilidades que brinda esta herramienta de software. Esto se lo consigue mediante el paquete de soporte de interfaz de OpenCV que permite la integración de con Matlab.

Ejecutada dicha integración se puede utilizar algoritmos OpenCV con la comodidad de la conexión de datos, la adquisición de imágenes, y capacidades de visualización en Matlab, y de esta manera explorar, analizar, y depurar diseños que incorporan algoritmos OpenCV.

Y lo más importante se puede hacer uso de Computer Vision System Toolbox (Toolbox para sistema de visión por computador) para extender su trabajo en MATLAB y OpenCV. Este Toolbox proporciona algoritmos de Matlab para la extracción de características, la detección y el seguimiento de objetos, reconocimiento de objetos, y la calibración de la cámara y visión geométrica 3D.

Para finalmente crear una interface completa y vistosa para mostrar el funcionamiento del proyecto facilitando la interacción con el usuario a través de Guide

6.6.Diseño organizacional

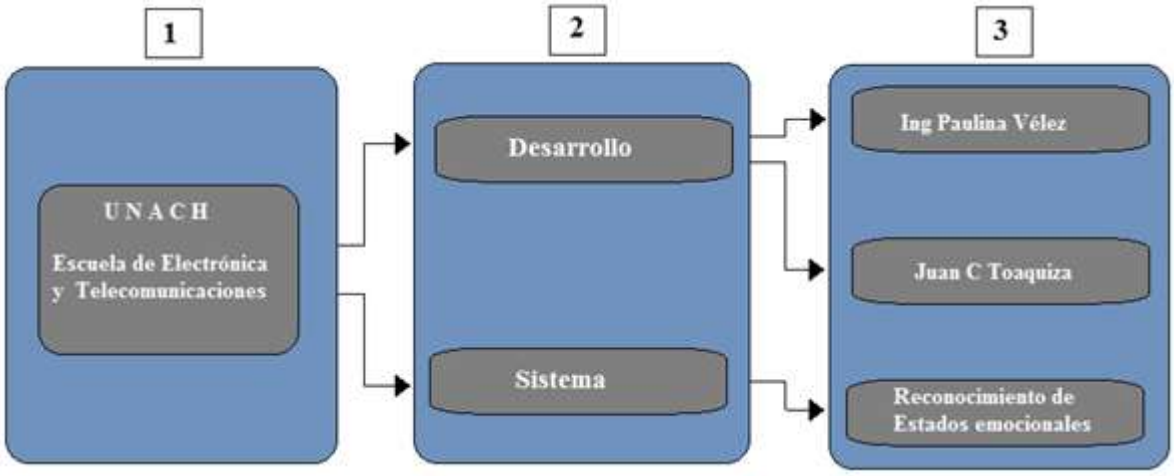


Figura 47. Esquema Organizacional
Elaborado por Autor

6.7.Monitoreo y Evaluación de la propuesta

La evaluación de la propuesta se realizara a través de pruebas en tiempo real, bajo distintas condiciones de luz sea esta natural o artificial, con diferentes distancias y ángulos del rostro con respecto a la cámara. Todo esto será realizado con un equipo apropiado que posea las características necesarias para el procesamiento de imágenes sin ningún problema

El sistema de reconocimiento de estados emocionales en tiempo real con las librerías OpenCV a través de la herramienta de software Matlab, contribuye en el desarrollo de nuevos mecanismos y métodos en el campo de visión por computador, siendo este campo de gran importancia en la actualidad debido a su gran uso en el ámbito académico, comercial e industrial

CAPITULO VII

7. BIBLIOGRAFIA

Adrian Kaebler, G. B. (2008). *Learning OpenCV.* Gravenstein Highway North,, United States of America: O'Really Media. Obtenido de http://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf

Aragón, R. (03 de septiembre de 2011). *El Poder de la mente*. Obtenido de Las expresiones universales de la emoción: http://psiqueviva.com/las-expresiones-universales-de-las-emociones-paul-ekman/

Bueno G, S. O. (2015). *Learning Image Processing with Open CV.* Birmingham: Pack Publishing.

Bustamante P, A. I. (2012). *TECNUM.* España: Escuela Superior de INgerieros en Telecomunicaciones.

Centro Integrado Politécnico. (02 de 02 de 2014). *etitudela*. Obtenido de Visión artificial: http://www.etitudela.com/celula/downloads/visionartificial.pdf

Eckel Bruce. (2012). *Pensar en C++*. España: Proyecto C++. Obtenido de http://arco.esi.uclm.es/~david.villa/pensar_en_C++/pensar_en_cpp-vol1.pdf

Eclipse Foundation. (02 de 02 de 2016). *Eclipse.* Obtenido de Eclipse CDT: https://eclipse.org/cdt/

Ecured. (24 de septiembre de 2012). *Programación*. Obtenido de IDE de Programación: http://www.ecured.cu/IDE_de_Programaci%C3%B3n

Enciclopedia de Ejemplos. (01 de 01 de 2016). *Ejemplos*. Obtenido de Ejemplos de Lenguaje de programación: http://www.ejemplos.co/20-ejemplos-de-lenguaje-de-programacion/

Galli, P. (02 de noviembre de 2011). *Ecured.* Obtenido de Eclipse, entorno del desarrollo integrado: http://www.ecured.cu/Eclipse,_entorno_de_desarrollo_integrado

Lélis Baggio, S. E. (2012). *Mastering OpenCV with Practical Computer Vision Projects.* Birmingham: Packt Publishing.

Martinez, R. (15 de 10 de 2014). *www.Linux-es.org*. Obtenido de El rincón de linux para hispanohablantes: http://www.linux-es.org/sobre_linux

Open CV 3.1.0dev,. (01 de 01 de 2014). *Open Source Computer Vision*. Obtenido de Face Detection using Haar Cascades: http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0

Organización Mundial de la Salud . (12 de 12 de 2011). *Proyectos y programas*. Obtenido de Informe Mundial sobre la Discapacidad: http://www.who.int/disabilities/world_report/2011/es/

Ramos I, L. M. (2010). *Ingenieria del Software y bases de datos: tendencias actuales.* La Mancha: Universidad de Castilla. doi:84842707777

Red Hat, Inc. (01 de 01 de 2015). *Fedora*. Obtenido de Fedora: https://getfedora.org/es/

Vélez J. et al. (2003). *Visión por computador.* S.L. - DYKINSON. doi:9788497720694

Zator System . (01 de 01 de 2016). *Curso C++*. Obtenido de El lenguaje C++: http://www.zator.com/Cpp/E1_2.htm#TOP

CAPITULO VIII

8. APENDICES O ANEXOS

8.1. ANEXO 1. Eye Detection

Eye detection can be very useful for face preprocessing, because for frontal faces you can always assume a person's eyes should be horizontal and on opposite locations of the face and should have a fairly standard position and size within a face, despite changes in facial expressions, lighting conditions, camera properties, distance to camera, and so on. It is also useful to discard false positives when the face detector says it has detected a face and it is actually something else. It is rare that the face detector and two eye detectors will all be fooled at the same time, so if you only process images with a detected face and two detected eyes then it will not have many false positives (but will also give fewer faces for processing, as the eye detector will not work as often as the face detector).

Some of the pretrained eye detectors that come with OpenCV v2.4 can detect an eye whether it is open or closed, whereas some of them can only detect open eyes.

Eye detectors that detect open or closed eyes are as follows:

- haarcascade_mcs_lefteye.xml (and haarcascade_mcs_righteye.xml)
- haarcascade_lefteye_2splits.xml (and haarcascade_ righteye_2splits.xml)

Eye detectors that detect open eyes only are as follows:

- haarcascade_eye.xml
- haarcascade_eye_tree_eyeglasses.xml

Eye search regions

For eye detection, it is important to crop the input image to just show the approximate eye region, just like doing face detection and then cropping to just a small rectangle where the left eye should be (if you are using the left eye detector)

and the same for the right rectangle for the right eye detector. If you just do eye detection on a whole face or whole photo then it will be much slower and less reliable. Different eye detectors are better suited to different regions of the face, for example, the haarcascade_eye.xml detector works best if it only searches in a very tight region around the actual eye, whereas the haarcascade_mcs_lefteye.xml and haarcascade_lefteye_2splits.xml detectors work best when there is a large region around the eye.

The following table lists some good search regions of the face for different eye detectors (when using the LBP face detector), using relative coordinates within the detected face rectangle:

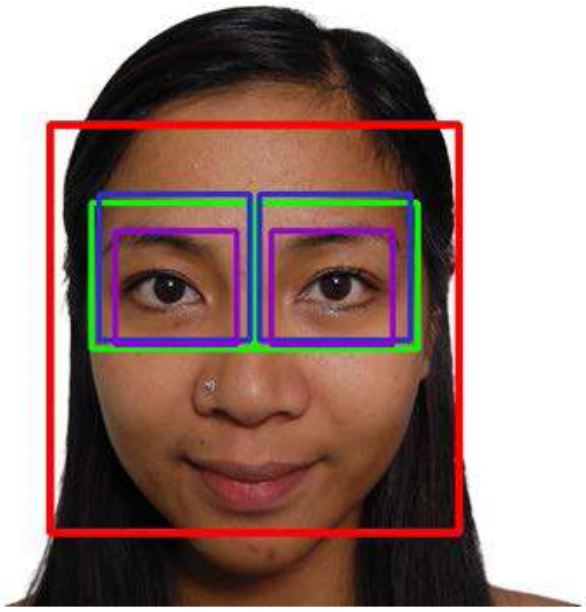
Cascade Classifier	**EYE_SX**	**EYE_SY**	**EYE_SW**	**EYE_SH**
haarcascade_eye.xml	0.16	0.26	0.30	0.28
haarcascade_mcs_lefteye.xml	0.10	0.19	0.40	0.36
haarcascade_lefteye_2splits.xml	0.12	0.17	0.37	0.36

Here is the source code to extract the left-eye and right-eye regions from a detected face:

```
int leftX = cvRound(face.cols * EYE_SX);
int topY = cvRound(face.rows * EYE_SY);
int widthX = cvRound(face.cols * EYE_SW);
int heightY = cvRound(face.rows * EYE_SH);
int rightX = cvRound(face.cols * (1.0-EYE_SX-EYE_SW));
Mat topLeftOfFace = faceImg(Rect(leftX, topY, widthX, heightY));
Mat topRightOfFace = faceImg(Rect(rightX, topY, widthX, heightY));
```

The following image shows the ideal search regions for the different eye detectors, where haarcascade_eye.xml and haarcascade_eye_tree_eyeglasses.xml are best with the small search region, while haarcascade_mcs_*eye.xml and haarcascade_*eye_2splits.xml are best with larger search regions. Note that the

detected face rectangle is also shown, to give an idea of how large the eye search regions are compared to the detected face rectangle:



When using the eye search regions given in the preceding table, here are the approximate detection properties of the different eye detectors:

Cascade Classifier	Reliability*	Speed**	Eyes found	Glasses
haarcascade_mcs_lefteye.xml	80%	18 msec	Open or closed	no
haarcascade_ lefteye_2splits.xml	60%	7 msec	Open or closed	no
haarcascade_eye.xml	40%	5 msec	Open only	no
haarcascade_eye_tree_ eyeglasses.xml	15%	10 msec	Open only	yes

* Reliability values show how often both eyes will be detected after LBP frontal face detection when no eyeglasses are worn and both eyes are open. If eyes are closed then the reliability may drop, or if eyeglasses are worn then both reliability and speed will drop.

** Speed values are in milliseconds for images scaled to the size of 320 x 240 pixels on an Intel Core i7 2.2 GHz (averaged across 1,000 photos). Speed is typically much faster when eyes are found than when eyes are not found, as it must scan the entire image, but the haarcascade_mcs_lefteye.xml is still much slower than the other eye detectors.

8.2.ANEXO 2. Programación sonrisa

Programación Sonrisa

```
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace cv;
char name0[]="Serio.png";
char name1[]="Feliz.png";
void ImprimirSerio();
void ImprimirFeliz();
int main()
{
	float Mou_SX= 0.30f;
	float Mou_SY= 0.80f;
	float Mou_SW= 0.40f;
	float Mou_SH= 0.15f;
	if(!detector.load("haarcascade_frontalface_alt2.xml"))
		cout << "No se puede abrir clasificador para la cara." << endl;
	if(!boca.load("Mouth.xml"))
		cout << "No se puede abrir clasificador para la boca." << endl;
//.............................
	if(!cap.open(0))
			cout << "No se puede acceder a la webcam." << endl;
	while(true)
		{
			Mat dest, gray, imagen;
			cap >> imagen;
			cvtColor(imagen, gray, CV_BGR2GRAY);
					equalizeHist(gray, dest);
					vector<Rect> rect;
					detector.detectMultiScale(dest, rect,  3, 0,
Size(60,60));
					for(size_t i = 0; i < rect.size(); i++ )
					{
					rectangle(imagen,Point(rect[i].x,
rect[i].y),Point(rect[i].x + rect[i].width, rect[i].y +
rect[i].height),CV_RGB(0,255,0), 2);
					}
//.............................
				if(rect.size() > 0)
					{
						vector<Rect> labo;
						int left2X = cvRound(rostro.cols *
Mou_SX);
```

```
                                                int top2Y = cvRound(rostro.rows *
Mou_SY);
                                                int width2X = cvRound(rostro.cols *
Mou_SW);
                                                int height2Y = cvRound(rostro.rows *
Mou_SH);
                                                Mat ParteBajaCara =
rostro(Rect(left2X, top2Y, width2X, height2Y));
                                                boca.detectMultiScale(ParteBajaCara,
labo);
                                                if((int)labo.size() > 0)
                                                {
                                                        ImprimirSerio();
                                                }
                                                 else

cvDestroyWindow("Seriedad");
if((int)labo.size() == 0)
{
ImprimirFeliz();
}
else
cvDestroyWindow("Happy");
}
//.....................
        imshow("Deteccion de rostros", imagen);
                        if(waitKey(2) >= 0) break;
                }
                return 1;
}
void ImprimirSerio()
  {
          IplImage* imag=NULL;
          imag=cvLoadImage(name0,1);
                cvNamedWindow( "Seriedad", 1);
                cvShowImage( "Seriedad", imag );
        }
void ImprimirFeliz()
  {
          IplImage* imag1=NULL;
                imag1=cvLoadImage(name1,1);
                cvNamedWindow( "Happy", 1);
                cvShowImage( "Happy", imag1 );
        }
```