CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Análisis de imágenes del rostro humano

Autor: Alejandro Rabadán Ureña

Director: Luis Baumela Molina

MADRID, JUNIO DE 2015

RESUMEN

El objetivo general de este trabajo es el correcto funcionamiento de un sistema de reconocimiento facial compuesto de varios módulos, implementados en distintos lenguajes. Uno de dichos módulos está escrito en Python y se encargará de determinar el género del rostro o rostros que aparecen en una imagen o en un fotograma de una secuencia de vídeo. El otro módulo, escrito en C++, llevará a cabo el reconocimiento de cada una de las partes de la cara (ojos, nariz, boca) y la orientación hacia la que está posicionada (derecha, izquierda).

La primera parte de esta memoria corresponde a la reimplementación de todas las partes de un analizador facial, que constituyen el primer módulo antes mencionado. Estas partes son un analizador, compuesto a su vez por un reconocedor (Tracker) y un procesador (Processor), y una clase visor para poder visualizar los resultados. Por un lado, el reconocedor o "Tracker.es el encargado de encontrar la cara y sus partes, que serán pasadas al procesador o Processor, que analizará la cara obtenida por el reconocedor y determinará su género. Este módulo estaba diseñado completamente en C y OpenCV 1.0, y ha sido reescrito en Python y OpenCV 2.4.

Y en la segunda parte, se explica cómo realizar la comunicación entre el primer módulo escrito en Python y el segundo escrito en C++. Además, se analizarán diferentes herramientas para poder ejecutar código C++ desde programas Python. Dichas herramientas son PyBindGen, Cython y Boost. Dependiendo de las necesidades del programador se contará cuál de ellas es más conveniente utilizar en cada caso.

Por último, en el apartado de resultados se puede observar el funcionamiento del sistema con la integración de los dos módulos, y cómo se muestran por pantalla los puntos de interés, el género y la orientación del rostro utilizando imágenes tomadas con una cámara web.

The main objective of this document is the proper functioning of a facial recognition system composed of two modules, implemented in different languages. One of these modules is written in Python, and his purpose is determining the gender of the face or faces in an image or a frame of a video sequence. The other module is written in C ++ and it will perform the recognition of each of the parts of the face (eyes, nose , mouth), and the head pose (right, left).

The first part of this document corresponds to the reimplementacion of all components of a facial analyzer , which constitute the first module that I mentioned before. These parts are an analyzer , composed by a tracke) and a processor, and a viewer to display the results. The tracker function is to find and its parts, which will be passed to the processor, which will analyze the face obtained by the tracker. The processor will determine the face's gender. This module was completely written in C and OpenCV 1.0, and it has been rewritten in Python and OpenCV 2.4.

And in the second part, it explains how to comunicate two modules, one of them written in Python and the other one written in C++. Furthermore, it talks about some tools to execute C++ code from Python scripts. The tools are PyBindGen, Cython and Boost. It will tell which one of those tools is better to use depend on the situation.

Finally, in the results section it is possible to see how the system works with the integration of the two modules, and how the points of interest, the gender an the head pose are displayed on the screen using images taken from a webcam.

ÍNDICE

1. INTRODUCCIÓN

En los últimos años, con el desarrollo de la tecnología se han ido perfeccionando los mecanismos de seguridad a la hora de acceder a un dispositivo con el fin de evitar robos, pérdidas o accesos no deseados a información privada. Algunos de estos sistemas de autenticación son claves de acceso o contraseñas, patrones gráficos, lectura de huellas dactilares, y una de los más recientes, el reconocimiento facial.

Centrándonos más en los sistemas de reconocimiento facial, ya que es en lo que se centra este proyecto, podríamos decir que cada vez se utilizan en un mayor número de dispositivos, como pueden ser smartphones, tablets, ordenadores o cámaras inteligentes. El problema de esta tecnología son las condiciones en que son tomadas las imágenes, por lo que si el rostro no está en una posición más o menos concreta, o la iluminación no es adecuada, el sistema podría inducir a errores al analizar las características del rostro o rostros. Por esta razón, el reconocimiento facial es un área de investigación activa hoy en día que continúa evolucionando.

El reconocimiento facial aunque se use principalmente en sistemas de seguridad, también se utiliza en aplicaciones de interacción persona-ordenador, en gestión multimedia, y en software como Google's Picasa, Apple iPhoto, Sony's Picture Motion Browser (PMB), Facebook (Figura 1), entre otros. Estos sistemas, cuando un usuario sube una foto, automáticamente reconoce los rostros que aparecen en la imagen para que el usuario pueda nombrarlos de forma rápida y sencilla, haciendo que sean aplicaciones cómodas con una alta usabilidad.

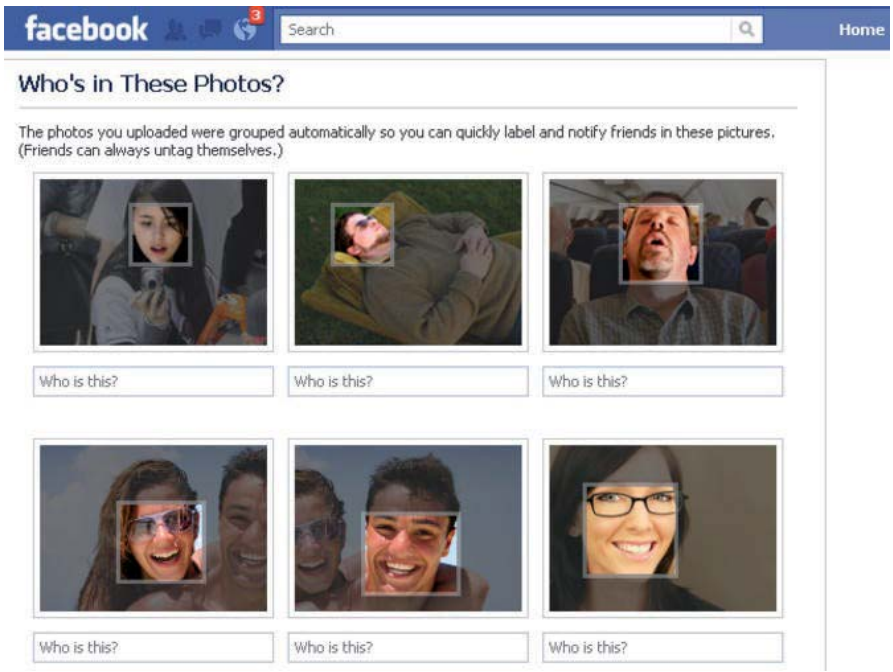


Figura 1: Reconocedor facial usado en Facebook.

1.1. Planteamiento

El objetivo del trabajo de fin de grado (TFG) es la reimplementación al lenguaje Python y la optimización de un algoritmo de reconocimiento y análisis automático de imágenes del rostro humano tomadas en condiciones no controladas escrito en C++ [1]. Este algoritmo extraerá de las imágenes información demográfica sobre el género del rostro e información geométrica sobre la orientación del mismo. El programa recibirá una imagen o un vídeo de entrada para procesar su contenido en busca de los rostros. Para esto, se desarrollarán las siguientes tareas:

- *Análisis del código en C++ y diseño de un diagrama UML para el entendimiento de cómo funciona el sistema.* El inicio del TFG ha consistido en el montaje del código del reconocedor, y su posterior ejecución usando como entrada imágenes, archivos de vídeo y el vídeo en directo de una cámara. Una vez que todo funcionaba de la manera esperada, leí detenidamente el código para diseñar un diagrama UML con el funcionamiento del sistema y poder estructurar el trabajo de una forma más cómoda. De esta forma, la parte de traducción de código fue dividida en dos ramas independientes, reconocimiento y clasificación.

- *Reimplementación del sistema de reconocimiento y selección de la región facial (Tracking).* Esta es la primera parte en lo que a implementación se refiere, y consiste en la recodificación de los algoritmos usados para el análisis de imágenes de entrada para saber el número de rostros que aparecen en una imagen o frame, y seleccionar la región de interés de dichos rostros para su posterior clasificación.

- *Reimplementación del sistema de clasificación de la región seleccionada.* Segunda parte de la recodificación del reconocedor, en ella se usan los algoritmos de clasificación explicados en el apartado 3 sobre la región de interés extraída del paso anterior.

- *Encapsulación de código C++ en Python.* En esta parte se usarán fragmentos de código implementados en C++, a través del código Python desarrollado anteriormente.

- *Respuesta al usuario en tiempo real, con la información correspondiente al Tracking, al género y a la orientación del rostro sobre la imagen o vídeo pasado como entrada.* Tras la realización de cada una de las partes descritas, el sistema se encarga de ir mostrando en la misma pantalla en la que se van procesando los rostros los resultados que ha ido obteniendo, como son la región de interés del rostro, el género, y la orientación. Con esto el usuario puede ver los resultados del programa en tiempo real.

En el desarrollo de estas tareas se han utilizado los entornos Eclipse con el plugin CDT para la utilización del código ya existente en C++, y el entorno Pycharm para el desarrollo del reconocedor en Python. También ha sido necesaria la instalación de la librería de OpenCV para el análisis y procesamientos de imágenes en los entornos anteriores. De forma más clara la integración de los entornos y las herramientas que he utilizado son los siguientes:

- Instalación de Eclipse con el plugin CDT
- Instalación de Pycharm para el desarrollo en Python
- Instalación de OpenCV para el análisis de imágenes
- Instalación de librerías de apoyo de Python, para facilitar la programación del reconocedor como matplotlib, numpy y math

1.2. Posibles problemas

El sistema de reconocimiento facial utilizado en este TFG se basa en la idea de identificar rostros extrayendo las características de cada una de sus partes. Esto significa que deberemos tener especial cuidado con las condiciones en que se toman las imágenes que pasaremos al sistema para su posterior procesamiento. A continuación, se van a citar varias de las situaciones problemáticas más comunes a la hora de capturar una imagen o un vídeo desde una cámara, y con las que tendremos que tener cuidado si queremos que el sistema funcione de la forma más correcta posible.

- *Iluminación de la imagen.* La iluminación es una de las características más importantes a la hora de trabajar con un sistema que procese imágenes. Por consiguiente, será necesario que los rostros estén correctamente iluminados, porque si no en los casos de iluminación excesiva o de iluminación escasa el sistema no será capaz de detectar correctamente la cara o algunas partes de la misma. Como subcategoría de este problema, también se ha de mencionar la tonalidad de la piel. Esto se debe a que el reconocedor se basa para distinguir la cara las diferentes sombras que proyectan cada una de sus partes, por lo que su funcionamiento será peor en imágenes tomadas a personas con una tez oscura.

- *Orientación de la cabeza.* Esta es una de las principales debilidades del sistema. En cuanto ya no sea visible en la imagen la mitad del rostro aproximadamente (imágenes de perfil), el sistema ya no será capaz de determinar que esa forma de la imagen sea una cara, cuando en realidad si lo es.

- *Resolución de la captura.* Otro aspecto a considerar es la resolución con la que la cámara realizará las capturas. Si la resolución es muy baja el sistema tendrá problemas para la detección de los contornos del rostro y sus partes. En cambio si la resolución es demasiado alta, el reconocedor tardará en procesar la imagen mucho tiempo debido a que tiene que recorrer todos y cada uno de los píxeles de la imagen en busca de posibles caras.

- *Imágenes movidas.* En este caso, la nitidez del rostro capturado es mínima, incluso una mancha difuminada y desenfocada en una imagen es difícil de reconocer hasta par el ojo humano. Por tanto, hay que evitar realizar movimientos bruscos en caso de que estemos utilizando el sistema para procesar los fotogramas de un vídeo tomado en tiempo real.

- *Uso de complementos.* Todos los objetos que estén puestos en la cara provocarán que el reconocedor no funcione correctamente. Por ejemplo, el uso de gafas, gorras, sombreros entre otros, hacen que no se vean algunas partes que el procesador requiere para distinguir el sexo o la orientación del rostro.

En la siguiente figura podemos observar varios ejemplos de imágenes tomadas con algunos de los problemas previamente descritos.

(a) Escasa iluminación (b) Orientación facial (c) Baja resolución

(d) Imagen movida (e) Uso de complementos

Figura 2: Imágenes que se deberían evitar en el sistema.

2. ESTADO DEL ARTE

En este capítulo se describirán los entornos y las herramientas utilizadas en la ejecución y desarrollo del reconocedor facial, y la finalidad para la que se ha usado cada uno de ellos de forma precisa.

2.1. Entornos de desarrollo

Para entrar en materia empezaremos con la definición de un entorno de desarrollo. Un entorno de desarrollo, o más conocido por sus siglas en inglés IDE (Integrated Development Environment), es un entorno de programación que ha sido empaquetado como una aplicación y suelen consistir en un editor de texto, un compilador, un depurador y una interfaz gráfica de usuario (GUI) [2]. Estos pueden dedicarse en exclusiva a un solo lenguaje de programación o para varios.

Una de las ventajas de los IDE es que mientras el código es editado puede ser montado, y de esta forma el programa te avisa de posibles errores de sintaxis.

En cuanto a distinción de IDEs, podemos clasificarlos dependiendo del número de lenguajes a los que están dedicados. Algunos IDEs están dedicados específicamente a un lenguaje de programación, permitiendo que las características sean lo más cercanas al paradigma de programación de dicho lenguaje o a un tipo de ajustes de tipos de lenguajes de programación como Xcode, Xojo y Delphi. Por otro lado, existen muchos IDEs de múltiples lenguajes tales como Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Codenvy y Microsoft Visual Studio.

2.1.1. Eclipse

La definición y objetivo de Eclipse según su web oficial [3]:

"Eclipse es una comunidad de individuos y organizaciones que colaboran en la fácil comercialización de software libre. Sus proyectos están centrados en la construcción de una plataforma de desarrollo abierta compuesta por frameworks extensibles, herramientas y diferentes ejecuciones de los programas para construir, implementar y gestionar software en su ciclo de vida. Eclipse Foundation es una corporación sin ánimo de lucro, que alberga los proyectos de Eclipse y ayuda al crecimiento de una comunidad de código abierto y a un ecosistema de productos y servicios complementarios.

El proyecto Eclipse fue originalmente creado en noviembre de 2001 por IBM y apoyado por un consorcio de proveedores de software. Eclipse Foundation fue creada en enero de 2004 como una corporación sin ánimo de lucro independiente para actuar como el administrador de la comunidad de Eclipse. La corporación sin ánimo de lucro independiente fue creada para permitir un proveedor neutral y libre, y una comunidad transparente para establecerse alrededor de Eclipse. Actualmente, la comunidad de Eclipse está formada por individuos y organizaciones de una sección transversal de la industria del software."

Eclipse es el entorno que he elegido para la parte de montaje, ejecución y pruebas del reconocedor facial ya completo en el lenguaje C++. Una de las razones por las que lo he elegido es porque es el entorno con el que más he trabajado desde que entré en el grado. Además, es gratuito y con muchas extensiones o pluggins para adaptarlo a las necesidades del desarrollador. Por ejemplo, este IDE lo he usado tanto para Java como para C/C++.

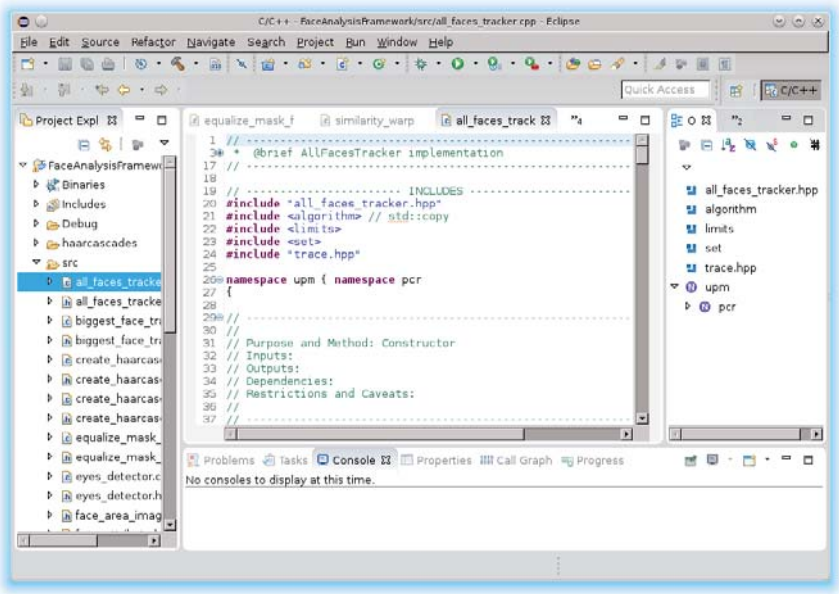


Figura 3: Ejemplo de la interfaz del IDE Eclipse.

Plugin CDT

Un plugin es una aplicación desarrollada para añadir una funcionalidad adicional o una nueva característica a otra aplicación software ya creada. Su traducción al castellano sería complemento. Los complementos son usados para navegadores web, aplicaciones, juegos, reproductores y los IDEs de los que se ha hablado anteriormente. Algunos ejemplos de complementos o plugins:

- *Navegadores web:* Adobe Flash Player para la visualización de vídeos
- *Reproductores de audio:* DFX aporta funciones específicas para la configuración de sonido
- *IDE:* Para adaptar un entorno de desarrollo a un nuevo lenguaje

El plugin CDT, en nuestro caso, se integra en Eclipse y le permite gestionar, montar, depurar y ejecutar proyectos en los lenguajes C y C++. Otras utilidades que aporta este complemento son la detección automática de errores de sintaxis mientras se está escribiendo el código, la adición de menús visuales para la configuración de la ejecución y posible depuración del proyecto, y para la inclusión de librerías externas, como es el caso de OpenCV que se explicará posteriormente.

Este plugin permitirá de esta forma montar, compilar, ejecutar y probar de manera rápida y cómoda, el código del reconocedor facial existente con los parámetros y argumentos requeridos.

2.1.2. Pycharm

El objetivo de Pycharm según JetBrains (compañía que lo creó) es [4]:

"Nosotros ayudamos a los desarrolladores a trabajar más rápido mediante la automatización de tareas comunes y repetitivas para permitirles estar centrados en el diseño del código y el panorama. Proporcionamos herramientas para explorar y familiarizarse con las bases del código más rápidamente. Nuestros productos lo facilitan de tal manera para que se puede cuidar la calidad durante todas las etapas de desarrollo y pasar menos tiempo en tareas de mantenimiento."

Las razones por las que he elegido Pycharm, para la parte de implementación en Python, esta vez se basan en que compañeros del laboratorio de Percepción y visión por computador, en el que estoy colaborando en este trabajo, me lo recomendaron. También es un entorno gratuito, y da bastantes facilidades para la inclusión de librerías, sin contar que ya en la instalación por defecto se añaden gran cantidad de las librerías más utilizadas en Python.

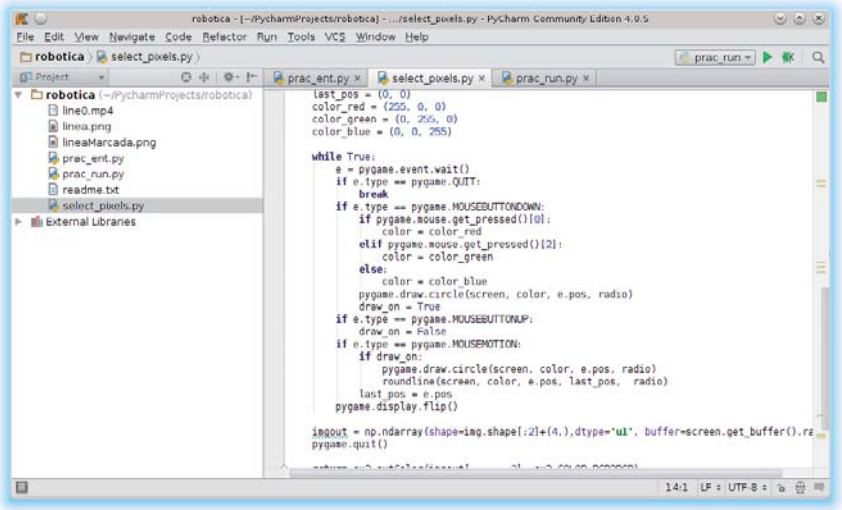


Figura 4: Ejemplo de la interfaz del IDE Pycharm.

2.2. Bibliotecas de apoyo para Python

Una biblioteca (del inglés library) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para una funcionalidad específica.

A diferencia de un programa ejecutable, el comportamiento que implementa una librería no espera ser utilizada de forma autónoma, sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea. Por otra parte, el comportamiento de una biblioteca no tiene porqué diferenciarse en demasía del que pudiera especificarse en un programa. Es más, unas bibliotecas pueden requerir de otras para funcionar, pues el comportamiento que definen refina, o altera, el comportamiento de la biblioteca original; o bien la hace disponible para otra tecnología o lenguaje de programación [5].

Las librerías de apoyo de Python para el desarrollo del proyecto han sido las siguientes:

- *Numpy:* Permite el uso de funcionalidades para la computación científica. Como por ejemplo un objeto array n-dimensional, herramientas para la integración de código C/C++, y funciones de álgebra lineal [6].
- *Math:* Permite el uso de las funciones matemáticas definidas en el estándar de C. Como por ejemplo redondeos, raíces, potencias entre otras [7].
- *Distutils:* Proporciona soporte para la construcción o instalación de módulos adicionales mediante el uso de Python. Estos nuevos módulos pueden estar escritos en Python, en C o pueden se colecciones de paquetes de Python que incluyen otros módulos implementados en Python y C [8].
- *Subprocess:* Permite generar nuevos procesos, conectar sus entradas, salidas y salidas de error, y obtener los códigos que devuelven [9].

2.3. OpenCV

OpenCV es una biblioteca libre que proporciona una framework de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real desarrollada originalmente por Intel.. Entre sus muchas áreas de aplicación destacarían: interacción hombre-máquina, segmentación y reconocimiento de objetos, reconocimiento de gestos, seguimiento del movimiento, estructura del movimiento, y robots móviles [10].

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows [11]. Su API está disponible para distintos lenguajes de programación tales como C, C++, Python.

La librería es tan eficiente debido a que su código está en C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multiprocesador. OpenCV también puede utilizar el sistema de primitivas de rendimiento integradas de Intel, al haber sido diseñada por ellos, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

Su funcionalidad para el desarrollo de este trabajo es la del procesamiento de imágenes para el reconocimiento y selección de rostros y las características de los mismos, como puede ser la localización de los ojos, la nariz y la boca.

En capítulos posteriores se especificará de forma más precisa los algoritmos y las funciones necesarias para la obtención de las regiones o posiciones que ocupan los elementos mencionados previamente. A continuación vemos un ejemplo de reconocimiento facial usando OpenCV en la Figura 5.

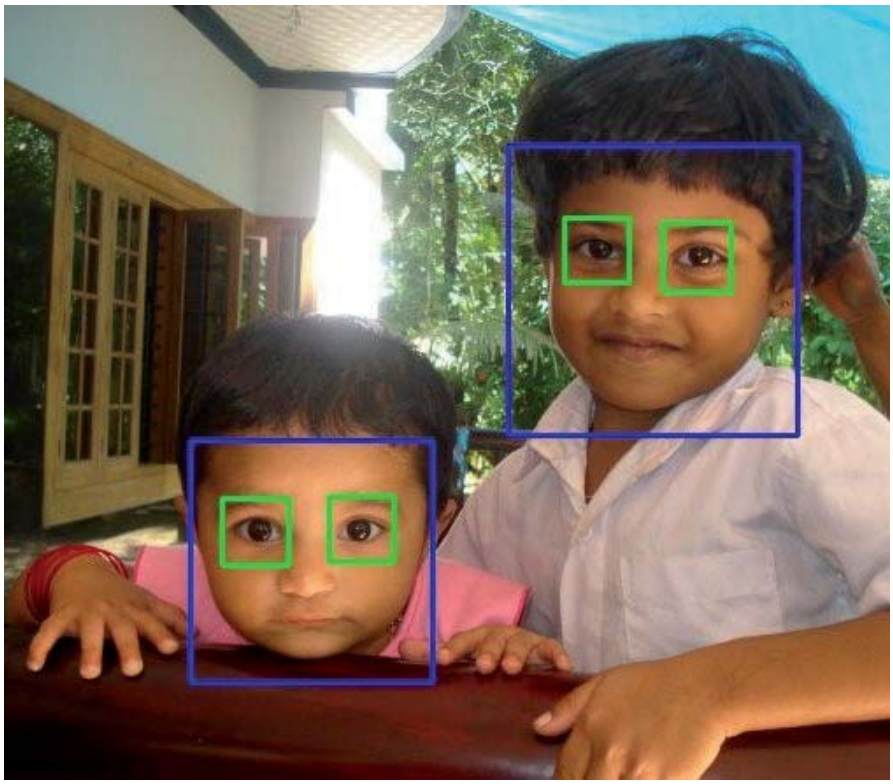


Figura 5: Ejemplo de reconocimiento del rostro y los ojos usando OpenCV

3. FUNDAMENTOS BÁSICOS

3.1. Visión por computador

La visión por computador es un campo que consiste en métodos para el análisis y el procesamiento de imágenes obtenidas del mundo real para convertirlas en datos numéricos o simbólicos que podamos usar posteriormente. Las imágenes pueden ser tomadas de muchas formas, como en una secuencia de imágenes o fotogramas de un vídeo, vistas desde múltiples cámaras o información multidimensional obtenida desde un escáner médico. Hay gran variedad de aplicaciones para la visión por computador, por ejemplo la reconstrucción de imágenes, estimación de movimientos, realidad aumentada, reconocimiento de objetos y la estimación de su orientación entre otros. Estos dos últimos son el tema principal de este trabajo, sólo que el objeto a reconocer es el rostro y lo que queremos obtener son su orientación, su género y la posición de sus partes (ojos, nariz y boca).

Existen otros campos o ramas de conocimiento que están muy relacionados con la visión por computador, como la inteligencia artificial, la estadística, la geometría, y la óptica entre otros. Un ejemplo sería un robot sigue-líneas que tuviese una cámara encima que actuara como sus ojos. En tiempo real, las imágenes capturadas por la cámara tendrían que ser procesadas para distinguir la línea del suelo. Si quisieramos un problema un poco más complejo, habría bifurcaciones en la línea y flechas que indicarían qué camino debería escoger el robot. Para esto, sería necesario un clasificador para el reconocimiento de la forma de una flecha, y contrastarla estadísticamente con un universo de trabajo previamente aprendido, formado con muchas flechas diferentes. De esta forma determinaríamos si esa forma a priori desconocida es una flecha o no.

En este trabajo de fin de grado, el papel de la visión por computador sería el de analizar imágenes o fotogramas de un vídeo en tiempo real para extraer datos relativos a los diferentes rostros que aparezcan en ellas. Una vez se han extraído los datos del análisis previo, se procederá a un procesamiento de estos datos para determinar la orientación hacia la que está posicionada el rostro y su género. A continuación, se explicarán los diferentes procedimientos de un sistema de reconocimiento de formas que se han utilizado tanto para determinar la orientación como el género. Para el desarrollo de esta parte, he utilizado como guía el material de la asignatura de *Reconocimiento de formas* [13] que ya he cursado. El material me lo proporcionó el tutor de este TFG, ya que es uno de los profesores que imparte dicha asignatura.

3.2. Sistema de reconocimiento de formas

El reconocimiento de formas es una disciplina con el objetivo de diseñar algoritmos capaces de encontrar regularidades en conjuntos datos para poder agruparlos en

diferentes categorías. Algunas de sus aplicaciones son reconocimiento de objetos, lugares, expresiones, diagnósticos clínicos, predicción de catástrofes, entre otros.

Hay diversos conceptos en un reconocedor que utilizaremos posteriormente y que es importante mencionar:

- *Universo de trabajo:* Conjunto de objetos disponibles para la construcción de un clasificador.
- *Clase:* Agrupación de objetos dentro del universo de trabajo.
- *Objeto:* Instancia de una clase.
- *Vector de características:* Conjunto de descriptores que representan un objeto y que permiten clasificarlo o etiquetarlo.

3.3. Técnicas de transformación de las características principales

Debido a la gran cantidad de características discriminantes que se pueden determinar de cada objeto, se utilizarán dos técnicas para reducir la complejidad del problema. Estas dos técnicas son:

- *Análisis Componentes Principales (ACP).* Maximiza la varianza de las nuevas características discriminantes.
- *Análisis Discriminante Lineal (ADL).* Maximiza un índice de separabilidad de las clases en el espacio transformado.

4. ALGORITMOS UTILIZADOS

El objetivo de esta aplicación es detectar los rostros que aparecen de frente o ligeramente girados hacia un lado en la escena, a partir de una imagen o vídeo que es introducido como entrada. Posteriormente, esa detección servirá para analizar los rostros y determinar la orientación y el género de los mismos. Para ello, se va a especificar en cada apartado de esta sección, el funcionamiento del conjunto de técnicas y algoritmos que se han utilizado para desarrollar el sistema. Primero, se explicará el funcionamiento del algoritmo encargado de detectar y enmarcar las caras tanto en imágenes como en fotogramas de un vídeo a tiempo real. Y finalmente se explicará como se procesan los datos extraídos del rostro detectado para determinar su género y su orientación. En la figura siguiente podemos observar cómo es el paso de una imagen por el sistema.

He de mencionar que explicaré el funcionamiento de los algoritmos, pero no ha sido tarea de este trabajo de fin de grado el diseño de los mismos sino que ya habían sido desarrollados anteriormente.

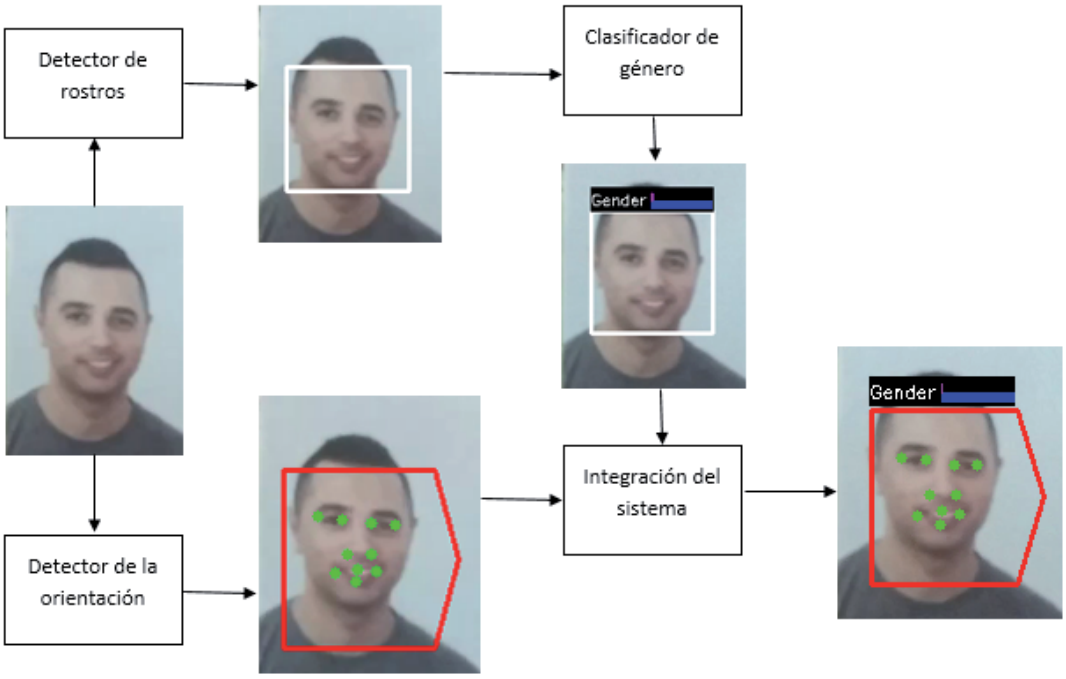


Figura 6: Esquema que explica el funcionamiento del sistema

4.1. Reconocer un rostro de una imagen

El primer algoritmo a implementar permitirá localizar la posición exacta de todos y cada uno de los rostros que aparecen en una imagen o vídeo que ha sido introducido como entrada al sistema. Para la detección de los rostros se utilizará la técnica de clasificación de *Haar Feature-based Cascade Classifier for Object Detection*, desarrollada en las publicaciones de Paul Viola y Michael Jones [14] y de Rainer Lienhart,Alexander Kuranov y Vadim Pisarevsky [15]. La idea se basa en entrenar cada clasificador con un conjunto de imágenes de un objeto particular a clasificar, como puede ser una cara o un coche (ejemplos positivos), y con otro conjunto de imágenes con cualquier contenido (ejemplos negativos). Todas las imágenes del entrenamiento, tanto los ejemplos positivos como los negativos tienen que tener el mismo tamaño. Una vez que el clasificador ha sido entrenado, el sistema buscará el objeto definido, rostros en nuestro caso, por toda la imagen utilizando una ventana que recorrerá toda la imagen aplicando dicho clasificador a diferentes escalas.

Una vez encontrado uno o varios rostros, en caso de querer obtener la localización de las diferentes partes de la cara (ojos, nariz y boca) se utilizarán otros tres clasificadores entrenados de la misma manera descrita en el párrafo anterior.

Cabe destacar que esta fase del sistema es de vital importancia debido a que los datos obtenidos por el clasificador de rostros serán procesados por los algoritmos de determinación de género y de orientación de un rostro. Por tanto, cualquier error en la detección de un rostro haría fallar al resto del sistema.

4.1.1. Algoritmo de ventana deslizante

El funcionamiento de este algoritmo es de fuerza bruta, ya que a priori no sabemos ni cuántos rostros hay ni dónde están situados en una imagen. Esto supone que la localización será a ciegas, con lo que la ventana, de tamaño ajustable, iterará por toda la imagen que le pasemos como entrada a diferentes escalas, de aquí su nombre "ventana deslizante". Como este algoritmo es de fuerza bruta se reducirá su complejidad cuanto menor sea la imagen pasada como entrada al sistema. Por tanto, la imagen de entrada para la detección de rostros será la escena completa, mientras que para la detección de cada una de sus partes (ojos, nariz y boca) la entrada será sólo la sección de la cara o caras obtenidas de la fase de detección anterior. La premisa utilizada para el funcionamiento de esta fase del sistema es que un ojo, una nariz o una boca tienen que estar única y exclusivamente dentro de la región de una cara.

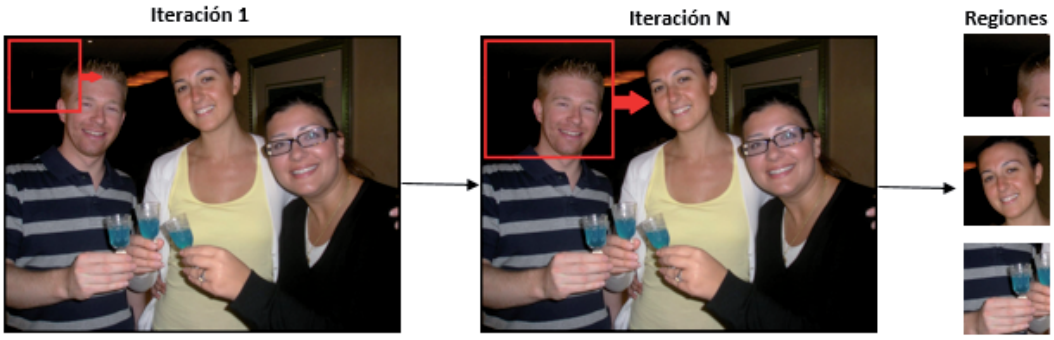


Figura 7: Funcionamiento de la ventana deslizante

4.2. Determinar género de un rostro

Una vez el detector ha extraído las diferentes secciones que contienen a cada uno de los rostros que aparecen en la escena original, se procederá a analizar cada uno individualmente. Para empezar la imagen del rostro se transforma en una imagen cuadrada y se pasa a escala de grises. Después, se mejorará el contraste de la imagen ecualizando su histograma de intensidad. En las siguientes figuras se puede ver de forma más clara en qué consiste este método.

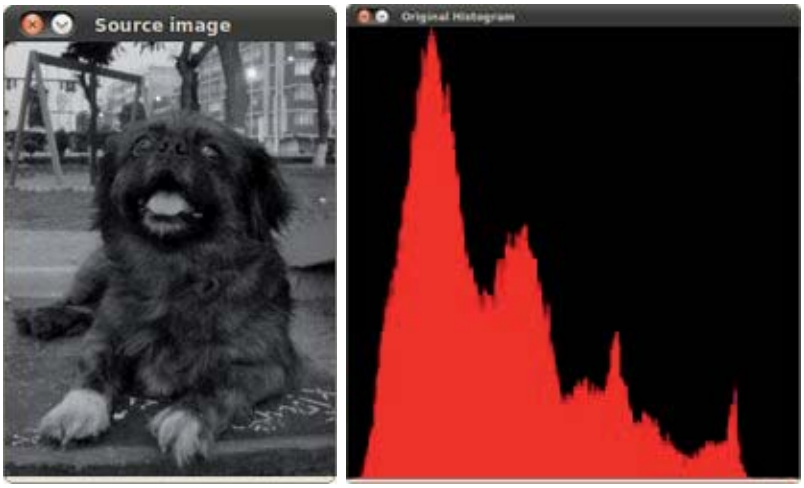


Figura 8: Ejemplo de una imagen y su histograma de intensidad

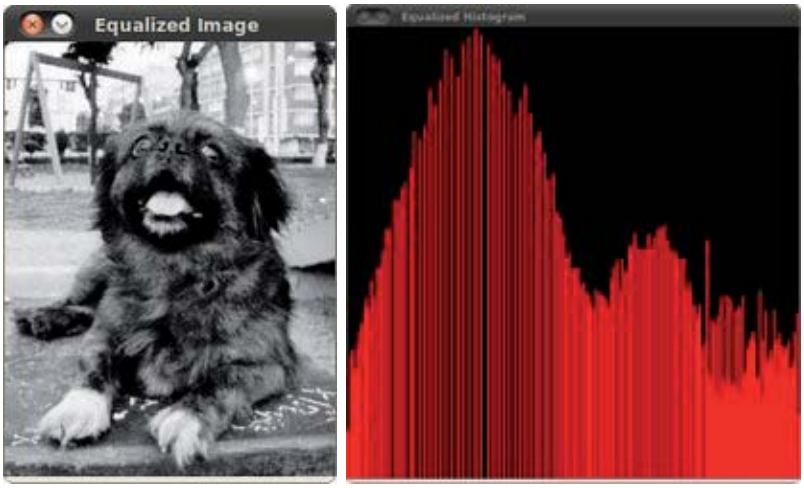


Figura 9: Ejemplo de ecualizar el histograma de intensidad de la figura anterior.

Cuando ya se ha ecualizado la imagen del rostro se le aplica una máscara para dejar los bordes de la imagen en negro y dejar una región circular de la cara desde las cejas hasta la barbilla aproximadamente. A continuación, se aplicarán las técnicas de transformación de las características principales mencionadas en un apartado anterior: análisis de las componentes principales y análisis discriminante lineal. Al aplicar estas técnicas se procede al cálculo de la pertenencia a cada uno de los géneros. En caso de ser un vídeo, estas pertenencias se irán acumulando para cada rostro, de manera que conforme se vayan procesando más fotogramas cada vez se aproximará más al resultado correcto, y aunque en alguno de ellos el detector falle no significará nada en el resultado final.

5. DESARROLLO

En este capítulo contaré de forma detallada las diferentes partes en las que se ha dividido el trabajo de fin de grado. Cada parte incluirá los fragmentos de código más importantes con sus correspondientes explicaciones.

Como el proyecto es modular, analizaré dentro de cada parte tanto la funcionalidad del código como la evaluación de los resultados obtenidos

5.1. Reimplementación de código C++ a Python

La primera parte consiste en la reimplementación del reconocedor de rostros incluyendo la detección de ojos, nariz y boca, y del clasificador de género.

5.1.1. Tipos complejos de datos

Hay varios tipos complejos de datos utilizados para la comunicación entre los métodos y el almacenamiento de una forma cómoda toda la información.

- **Image_Region.** Objeto que contiene cuatro parámetros que definen la región rectangular de un rostro. Partimos de que la esquina superior izquierda de la ventana es la posición (0,0) y que el eje Y está invertido, es decir, que el eje Y va desde cero hasta la altura de la imagen conforme se desciende.
  - *x:* Número entero que determina la posición en el eje X de la esquina superior izquierda del rectángulo que contendrá el rostro.
  - *y:* Número entero que determina la posición en el eje Y de la esquina superior izquierda del rectángulo que contendrá el rostro.
  - *width:* Número entero que determina la anchura (distancia en el eje X) del rectángulo que contendrá el rostro.
  - *height:* Número entero que determina la altura (distancia en el eje Y) del rectángulo que contendrá el rostro.
- **Face.** Objeto en forma de lista que contiene los siguientes elementos:
  - *region:* Objeto *Image_Region.* con los parámetros que definen el rectángulo que contiene un rostro.
  - *left_eye_coords:* Lista de formada por dos números enteros que definen las coordenadas relativas del ojo izquierdo en el rectángulo que contiene el rosto.
  - *right_eye_coords:* Lista de formada por dos números enteros que definen las coordenadas relativas del ojo derecho en el rectángulo que contiene el rosto.
  - *nose_coords:* Lista de formada por dos números enteros que definen las coordenadas relativas de la nariz en el rectángulo que contiene el rosto

- *mouth_coords:* Lista de formada por dos números enteros que definen las coordenadas relativas de la boca en el rectángulo que contiene el rosto
- *processor:* Objeto procesador que contendrá entro otros parámetros, las probabilidades acumuladas que indicarán la pertenencia del rostro a cada género.

5.1.2. Estructura y funcionalidad del código

Para el desarrollo del modelo de esta sección se han aplicado diversos principios básicos de diseño de la ingeniería del software que facilitarán el entendimiento y el posterior uso del código de la aplicación.

- *Modularidad.* Dividir un sistema complejo en partes más simples llamadas módulos. De manera que el cambio de algún módulo afecte de la mínima manera posible a la totalidad del sistema.
- *Abstracción:* Permitir la comprensión de la esencia de los subsistemas sin tener que conocer detalles innecesarios. Así se facilita el mantenimiento y favorece la encapsulación.
- *Cohesión:* Procurar que todos los elementos que componen cada módulo están relacionados en el desarrollo de una función única y perfectamente definida.
- *Acoplamiento:* Eliminación o reducción de relaciones innecesarias entre módulos para mantener el sistema.

Antes de empezar a explicar el funcionamiento del código en sí, veremos el diagrama de clases UML del reconocedor facial y del clasificador de género.

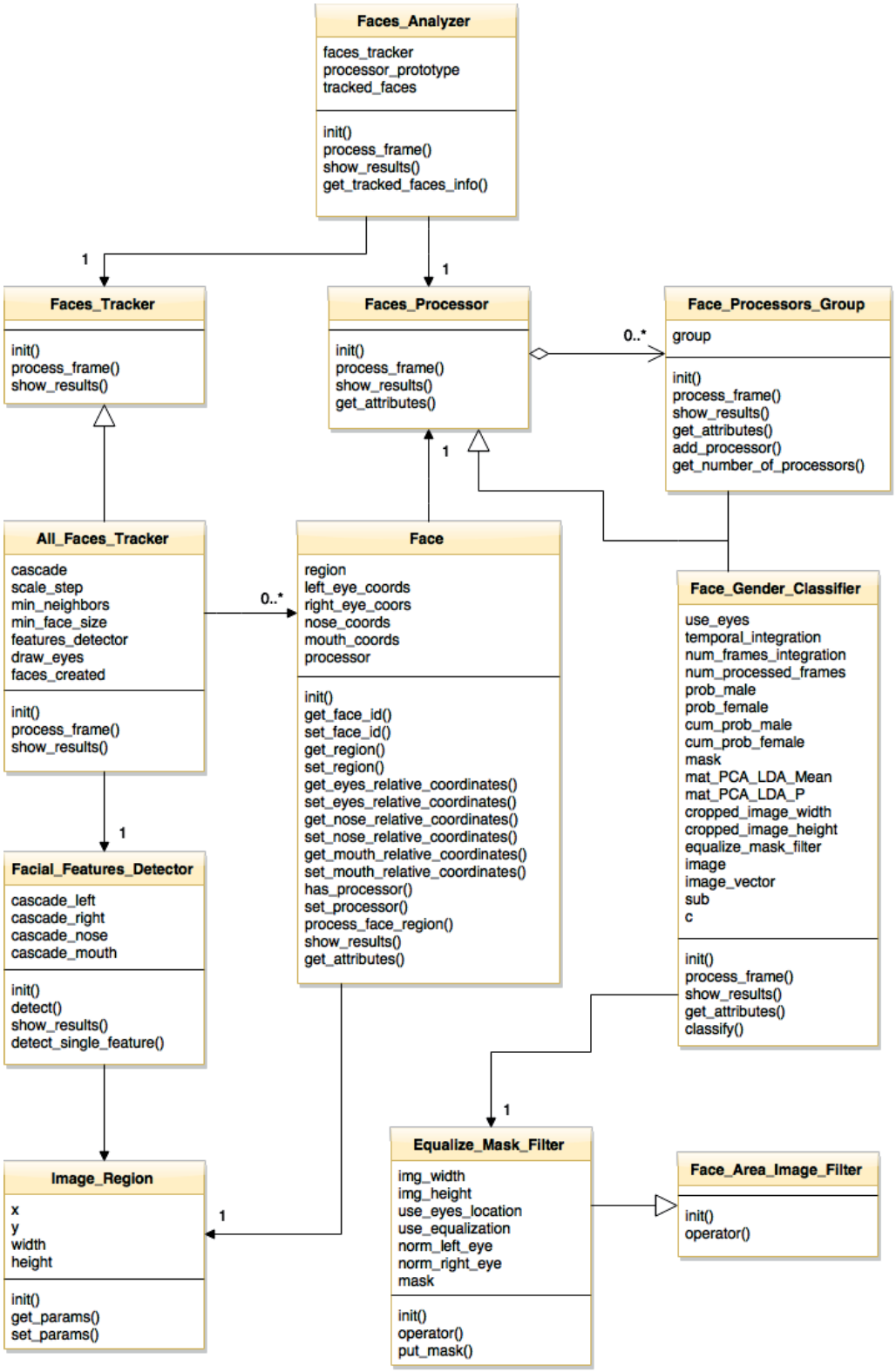


Figura 10: Diagrama de clases UML

Se puede observar en el diagrama de la figura anterior que esta aplicación se divide en dos partes. La primera, que sería la rama izquierda del árbol del diagrama, encabezada por la interfaz *Faces_Tracker*, será la encargada a grandes rasgos de reconocer y situar los rostros de una imagen o vídeo y sus ojos, nariz y boca. Y la segunda rama, situada a la derecha en el árbol y encabezada por la interfaz *Faces_Processor*, se encargará del procesamiento de los datos de interés obtenidos por el detector facial, y así determinar el género de cada uno de los rostros que aparecen en la imagen. Ambas partes estarán gestionadas por la clase abstracta *Faces_Analyzer*.

Una vez explicado la funcionalidad un poco por encima, ya podemos adentrarnos en cómo se ejecutará el código de principio a fin. Para facilitar el entendimiento de cómo funcionan las interacciones entre los diferentes módulos del sistema y las estructuras de datos que manejan, utilizaré diagramas de secuencia. Los diagramas de secuencia muestran las instancias de las clases que intervienen en el funcionamiento de la aplicación con una línea vertical, y los mensajes pasados entre los diferentes objetos serán representados con líneas horizontales. Estas últimas representan las llamadas de los métodos de las clases en orden cronológico.

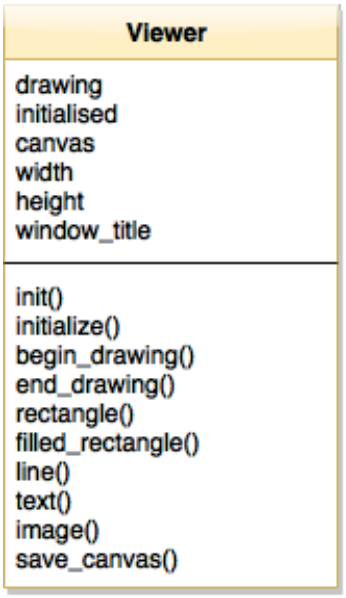


Figura 11: Clase Viewer

El módulo principal será *Test_Framework* y será el encargado de lanzar el programa, cargando los diferentes clasificador en cascada (*CascadeClassifier*) para la localización de los rostros y sus partes, y de gestionar la recepción de imágenes o vídeo. En primer lugar creará una instancia de la clase *Viewer* que se encargará de generar una nueva imagen sobre la que pintará los resultados que se obtengan del detector de rostros y sus partes, y del procesador de género. Los resultados se

dibujarán en forma de texto y rectángulos sobre la imagen de entrada, a través del método **show_results()** que se explicará posteriormente cuando sea utilizado.

Después comprobará el tipo de archivo que se le está pasando por la entrada: una imagen, un archivo de vídeo o un vídeo en tiempo real desde una cámara web. Ya está todo listo para empezar a crear el detector de rostros y el procesador de género.

Creación de un analizador de rostros

El siguiente paso será una llamada al constructor de *All_Faces_Tracker* con una serie de parámetros para ajustar la localización de todos los rostros que aparezcan en la imagen o fotograma de entrada. Los parámetros que recibe el constructor de *All_Faces_Tracker* son:

- *cascade:* Objeto *CascadeClassifier* encargado de localizar los rostros de una imagen.
- *cascade_left_eye:* Objeto *CascadeClassifier* encargado de localizar el ojo izquierdo de un rostro.
- *cascade_right_eye:* Objeto *CascadeClassifier* encargado de localizar el ojo derecho de un rostro.
- *cascade_nose:* Objeto *CascadeClassifier* encargado de localizar la nariz derecho de un rostro.
- *detection_scale_step:* Número decimal que determina cuanto se reduce la dimensión de la imagen de entrada en cada paso del clasificador en cascada.
- *detection_min_neighbors:* Número entero que determina el número de vecinos que se consideran como un único ejemplar, con el fin de evitar solapamientos en una misma cara detectada repetidas veces en píxeles muy próximos.
- *detection_min_face_size:* Número entero que determina la dimensión de la ventana deslizante que utilizará el clasificador en cascada para filtrar toda la imagen.
- *draw_eyes:* Variable booleana que determina si se quiere pintar en la imagen de salida del sistema la ubicación de los ojos, nariz y boca de todas las caras que aparezcan en la imagen de entrada.

A continuación, creará una instancia de la clase *Face_Processor_Group*, que es simplemente una lista que contendrá los diferentes procesadores que se quieran añadir para determinar cualquier característica con la información de un rostro. De esta manera, el código permite que no sólo se pueda utilizar el procesador de género, sino que se podría programar uno nuevo como un procesador para determinar la edad, y sólo con añadirlo a la lista del objeto *Face_Processor_Group* ya se podría utilizar.

Una vez que ya tenemos creados las instancias, llamaremos al constructor de la clase *Faces_Analyzer* que se encargará de relacionar el detector de rostros o tracker con el procesador de género.

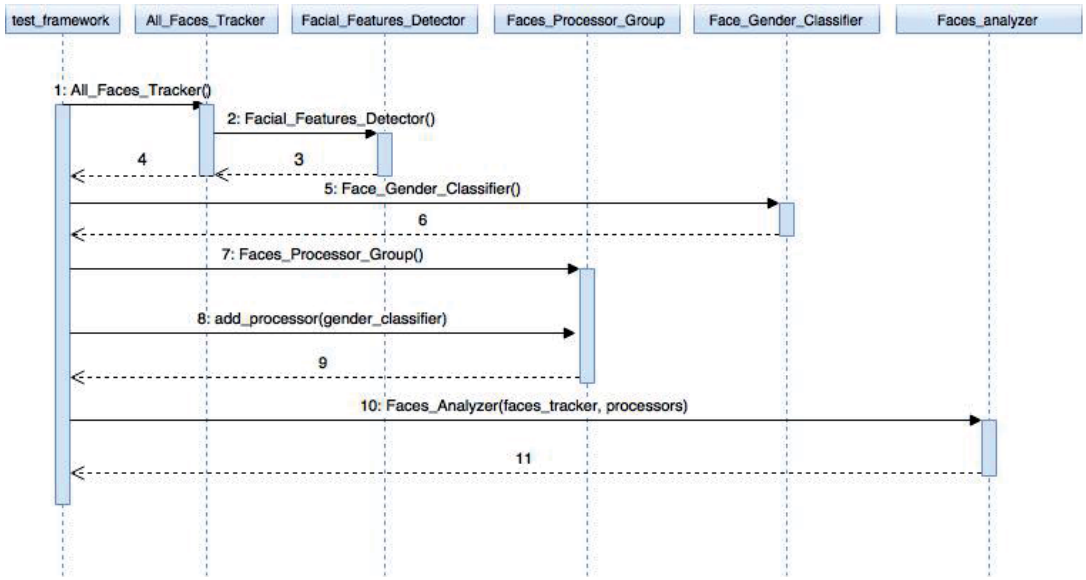


Figura 12: Diagrama de secuencia para crear un analizador de rostros

Antes de empezar a usar el detector sobre la imagen es necesario inicializar el objeto Viewer creado anteriormente para que genere una nueva ventana en el escritorio para poder visualizar los resultados.

Procesamiento de cada imagen

No importará si la entrada es una imagen o un vídeo, en caso de que sea una imagen todos los métodos se ejecutarán en la misma secuencia pero sólo una única vez, mientras que para un vídeo habrá un bucle para procesar cada uno de los fotogramas hasta que termine el archivo de vídeo o se pare de forma manual.

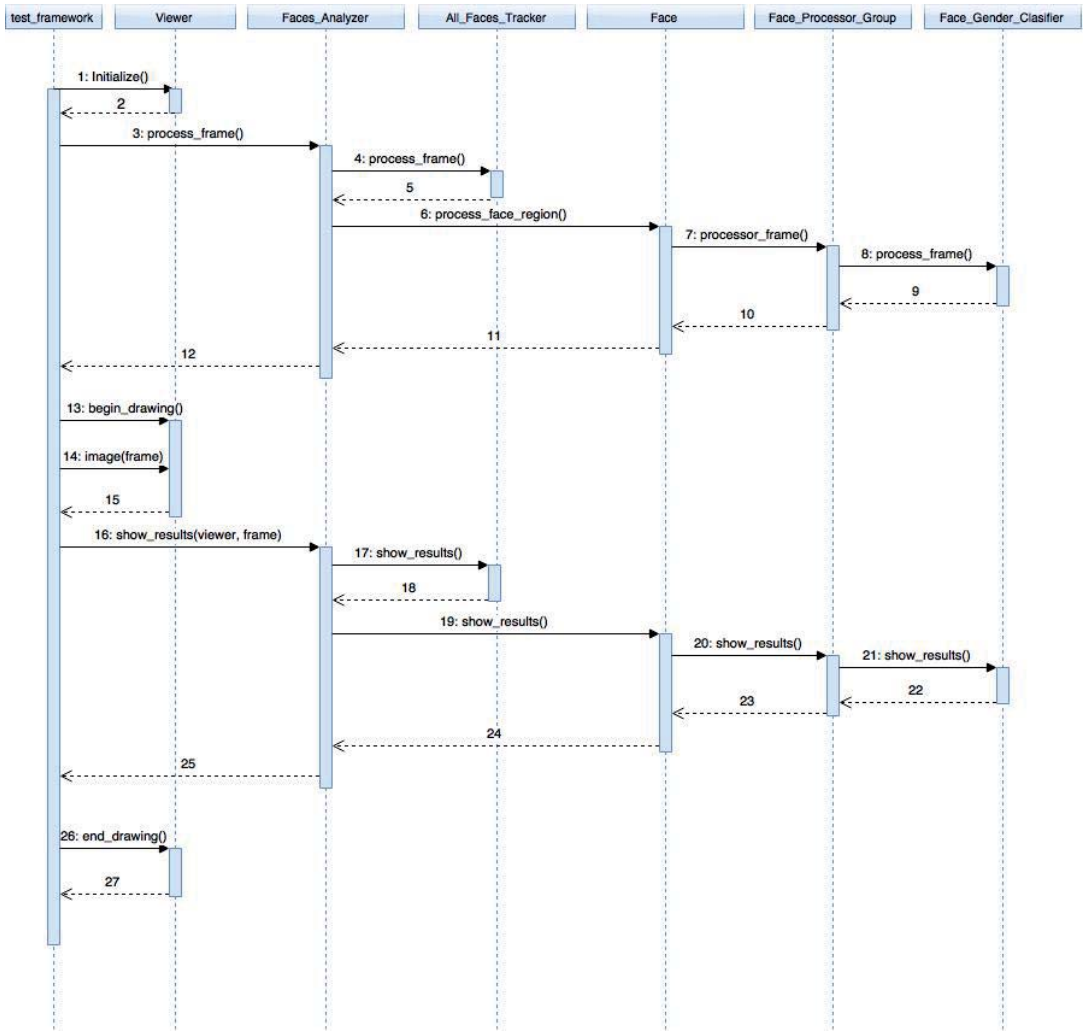


Figura 13: Diagrama de secuencia del procesamiento de cada imagen

En la llamada al método *process_frame* de la clase *Faces_Analyzer* se incluyen tanto el detector de rostros como el procesador de género. Primero se ejecutará el detector de rostros y justo después el procesador de género.

```
def process_frame(self, image):
    self.tracked_faces = self.faces_tracker.process_frame(image, self.tracked_faces)

    # Set a new copy of the image processor to each new face (hasProcessor()
    # will be false for new faces) and apply processFaceRegion to each of
    # the faces.
    for tracked_face in self.tracked_faces:
        if not tracked_face.has_processor():
            tracked_face.set_processor(c.deepcopy(self.processor_prototype))
        tracked_face.process_face_region(image)
    return self.tracked_faces
```

Figura 14: Código de *process_frame*

La primera línea, es una llamada al método *process_frame* de la clase *Faces_Tracker*, que corresponde al detector facial, mientras que las líneas siguientes corresponden al procesamiento de la información contenida para cada rostro detectado para determinar su género.

Detector de rostros

Para la detección de rostros, según podemos observar en la Figura 13 se acaba llamando al método *process_frame* de la clase *All_Faces_Tracker* que realmente será el encargado de detectar todas y cada una de las caras que aparezcan en la imagen. A este método se le pasarán una imagen y las caras detectadas en la iteración anterior.

El método *process_frame* primero empezará poniendo la imagen en escala de grises para después llamar al método *detectMultiScale* que se encargará de detectar los rostros de la imagen. El valor que nos devolverá sera una lista de listas de cuatro elementos, una sublista por cada cara detectada. Una sublista representa la región que ocupa cada rostro, y está formada por los elementos [x, y, width, height], que significan lo mismo que en un objeto *Image_Region*.

A continuación se comprobará el número de caras que se han detectado en esta iteración, y si este es mayor que cero, porque si no lo es devolvería una lista vacía y concluiría el método. En caso de detectar un número mayor que cero, comprobará si en la iteración anterior había alguna cara o no. Si no había ninguna cara antes, significa que todas las caras detectadas en esta imagen son nuevas y las añadirá a la variable de clase *faces_created*, y calculará las posiciones relativas de sus ojos, nariz y boca. Para esto llamará al método *detect_created* de la clase *Facial_Features_Detector* que recibe los siguientes parámetros:

- *frame:* Imagen o fotograma de un vídeo de tipo *numpy.ndarray*
- *region:* Objeto *Image_Region* con los parámetros de la región de la cara.
- *left_eye:* Lista de formada por dos números enteros que definen las coordenadas relativas del ojo izquierdo en el rectángulo que contiene el rosto.
- *right_eye:* Lista de formada por dos números enteros que definen las coordenadas relativas del ojo derecho en el rectángulo que contiene el rosto.
- *nose:* Lista de formada por dos números enteros que definen las coordenadas relativas de la nariz en el rectángulo que contiene el rosto.
- *mouth:* Lista de formada por dos números enteros que definen las coordenadas relativas de la boca en el rectángulo que contiene el rosto.

En caso contrario, es decir que ya había caras detectadas en la iteración anterior, comprobará para cada cara detectada en la iteración actual cuál es la cara antigua que esté más próxima. Si el ratio de los parámetros entre los rostros antiguos y los nuevos que estén más cercanos entre sí es menor a 0.3 se actualizarán las dimensiones del rectángulo contenedor del rostro. Si es mayor de 0.3 entonces es un rostro nuevo, calculará las coordenadas de sus partes y lo añadirá a la lista de caras detectadas *tracked_faces*.

Clasificador de género

Una vez añadidas contrastadas todas las caras, añadido las nuevas y actualizado las que se han movido ligeramente habrá concluido la tarea del detector y el método *process_frame* de la clase *All_Faces_Tracker* devolverá la lista de caras detectadas *tracked_faces*. Ahora el método *process_frame* de la clase *Faces_Analyzer* comprobará si cada cara tiene asignado un procesador de género, si es una cara nueva le asignará una copia del procesador de género, y se ejecutará el método *process_face_region* de cada uno de los rostros. Este método recibe como parámetros la imagen de entrada y un rostro, y se encarga de llamar al método *process_frame* de la clase *Face_Gender_Clasifier* con los mismos parámetros.

En este caso, el método *process_frame* primero llamará a la función *operator* de la clase *Equalize_Mask_Filter* introduciendo como parámetros de entrada la imagen y un rostro. Este método primero convierte la imagen recortada de sólo el rostro a escala de grises usando la función *cvtColor*, y después se ecualiza su histograma de intensidad con *equalizeHist*. Para finalizar, se le aplica una máscara a la sección de la imagen del rostro para que solamente el centro de la cara sea visible dejando los bordes en negro.

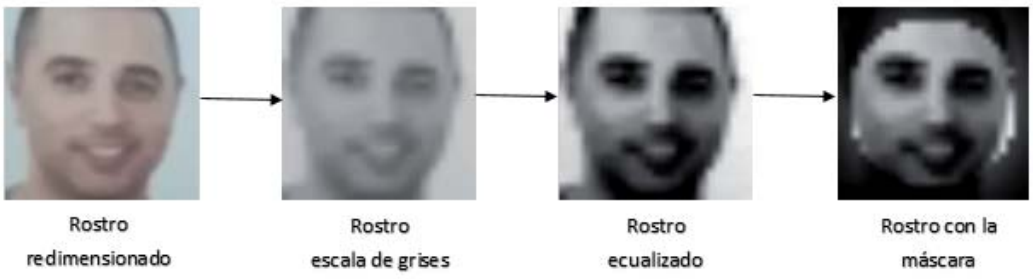


Figura 15: Etapas de la imagen de un rostro para la detección de género

Después de haber aplicado las modificaciones pertinentes a la nueva imagen con el rostro, ahora se redimensionará a una nueva imagen de 25x25 píxeles y se aplicará ACP y ADL. Por último, se procederá a clasificar la imagen para determinar la probabilidad de que pertenezca a cada uno de los géneros.

Mostrar resultados

Para finalizar con la ejecución del sistema sólo falta interpretar los resultados obtenidos del método *process_frame* de la clase *Faces_Analyzer* y mostrarlos por pantalla. Para ello, seguidamente se llamará al metodo *show_results* de la misma clase que llamará a su vez a los métodos con el mismo nombre de las clases *All_Faces_Tracker* y de *Face_Gender_Clasifier*

Para pintar por pantalla los datos relativos al detector facial, se utilizará el método *show_results* de la clase *All_Faces_Tracker*. En este método se pintará un rectángulo blanco correspondiente a las dimensiones del objeto *Image_Region* contenido dentro de cada cara detectada, y en caso de haberlo seleccionado pintará la localización de los ojos, la nariz y la boca de que haya detectado de cada rostro.

Figura 16: Resultado del detector de rostros

Y para pintar los datos relativos al género de cada rostro, se utilizará el método *show_results* de la clase *Face_Gender_Clasifier*. En este método primero comprobará si si han pasado los suficientes fotogramas de un vídeo para saber si las probabilidades de pertenencia a cada género son fiables. Para mostrar ambas probabilidades, se pintará un rectángulo azul (masculino) y uno rosa (femenino) encima del recuadro del rostro. El tamaño de estos dos rectángulos será directamente proporcional a la probabilidad de que pertenezca el rostro a cada uno de los géneros.

Figura 17: Resultado del detector de rostros y clasificador de género

Funcionamiento de los métodos

clase: Face_Analyzer

- **process_frame(image, tracked_faces);** Método encargado de procesar una imagen o fotograma de un vídeo de tipo *numpy.ndarray*. Dependiendo de su implementación su finalidad es la de determinar la región que ocupa cada rostro y las coordenadas relativas sus partes (ojos, nariz y boca), y la de clasificar dicho rostro para poder conocer su género. El valor de retorno será una lista de objetos de tipo *Face*.

- **show_results(viewer, image);** Método encargado de mostrar por pantalla los resultados obtenidos por *process_frame* a partir de un objeto de la clase *Viewer* y la imagen del tipo *numpy.ndarray* sobre la que plasmará los resultados. Para ser exactos, pintará el rectángulo que se ajusta a la posición del rostro, la posición de las partes del mismo si se ha indicado, y los rectángulos asociados a su género.

clase: All_Faces_Tracker

- **process_frame(image, tracked_faces);** Método encargado de procesar una imagen o fotograma de un vídeo de tipo *numpy.ndarray* para determinar la región que ocupa cada rostro y las coordenadas relativas sus partes (ojos, nariz y boca). El valor de retorno será una lista de objetos de tipo *Face*.

- **show_results(viewer, image, tracked_faces);** Método encargado de mostrar por pantalla los resultados obtenidos por *process_frame* a partir de un objeto de la clase *Viewer*, una imagen de tipo *numpy.ndarray* sobre la que plasmarán los resultados y las caras detectadas (*tracked_faces*) como lista de objetos *Face*. Su objetivo es el de pintar el rectángulo que se ajusta a la posición del rostro y la posición de las partes del mismo si se ha indicado.

clase: Facial_Features_Detector

- **detect(frame, region, left_eye, right_eye, nose, mouth);** Método encargado de procesar la región que ocupa un rostro (*region*) de una imagen o fotograma de un vídeo (*frame*) de tipo *numpy.ndarray* para determinar las coordenadas relativas de sus partes (ojos, nariz y boca). El valor de retorno será una lista con las coordenadas de cada parte.

- **show_results(viewer, region, left_eye, right_eye, nose, mouth);** Método encargado de mostrar por pantalla a partir de un objeto de la clase *Viewer* la posición de los ojos, la nariz y la boca mediante una cruz de diferentes colores sobre la región de un rostro.

clase: Face_Processor_Group

- **add_processor(processor);** Método encargado de añadir instancias de la clase *Face_Processor* a la lista que almacena los procesadores que se van a aplicar a cada rostro.
- **get_number_of_processor();** Método que devuelve el número de procesadores que se van a aplicar a cada rostro.

clase: Face_Gender_Classifier

- **process_frame(frame, f);** Método encargado de procesar la región que ocupa un rostro (f) de una imagen o fotograma de un vídeo (*frame*) de tipo *numpy.ndarray* para determinar la probabilidad de pertenencia a cada género. El valor de retorno será una lista que contendrá las probabilidades de que el rostro sea masculino y femenino.
- **show_results(viewer, image, f);** Método encargado de mostrar por pantalla los resultados obtenidos por *process_frame* a partir de un objeto de la clase *Viewer* (*viewer*), una imagen del tipo *numpy.ndarray* sobre la que se plasmarán los resultados (*image*) y la cara que ha sido clasificada (f). Su objetivo es el de pintar dos rectángulos, encima de la región del rostro, que representen la pertenencia de dicho rostro a cada género.
- **classify(value);** Método encargado de clasificar el valor numérico decimal *value* obtenido de aplicar el ecualizado al histograma de intensidad, una máscara, ACP y ADL a la imagen de la región que ocupa el rostro. El valor de retorno será una lista que contendrá las probabilidades de que el rostro sea masculino y femenino.

clase: Equalize_Mask_Filter

- **operator(frame, face);** Método encargado de procesar la región que ocupa un rostro (*face*) de una imagen o fotograma de un vídeo (*frame*) de tipo *numpy.ndarray* convertida a escala de grises para ecualizar su histograma de intensidad. El valor de retorno será la imagen de tipo *numpy.ndarray* ecualizada.
- **put_mask(image, mask);** Método encargado de aplicar una máscara a una imagen. El parámetro imagen tiene que ser de 25x25 píxeles.

5.1.3. Diferencias de OpenCV entre C++ y Python

Las diferencias más notables que he encontrado a la hora de reescribir el código tienen que ver con OpenCV. El problema estaba en que el código inicial escrito en C++ utilizaba funciones de la versión OpenCV 1.0, mientras que para Python la última versión estable de OpenCV es la 2.4. Esta situación provocaba que algunas

de las funciones utilizadas en el código C++ ya no existan, o hayan sufrido cambios tanto en los parámetros que reciben como en los que devuelven. Incluso había funciones de OpenCV que solamente se podían utilizar en C++, como por ejemplo el manejo de las regiones de interés, que suponían una función específica en C++ mientras que en Python era una sublista de la imagen completa usando la biblioteca numpy.

El resto de diferencias están relacionadas directamente con funcionalidades y estructuras de datos que se utilizan en C++ que no son necesarias en Python. Por ejemplo, la reserva y liberación de memoria cada vez que se carga una imagen. Y con respecto a las estructuras de datos, era necesario utilizar clases auxiliares para la gestión de objetos complejos como un punto, o un rectángulo entre otros. Sin embargo, estas clases en Python son listas sencillas de elementos o como mucho listas de listas, sin necesidad de crear clases auxiliares. Además el acceso a estas estructuras de datos es muy dispar, en Python es utilizar índices para acceder al elemento oportuno de una lista, mientras que en C++ necesitas conocer el tipo de datos de cada elemento para poder extraerlo en una variable del mismo tipo.

5.2. Encapsulación de código C++ a Python

Esta es la segunda parte del trabajo, y consiste en cómo manejar un módulo implementado en C++ (detector de la orientación de un rostro) desde Python.

5.2.1. Objetivo

Lo que se pretende con el desarrollo de esta parte del proyecto es la gestión desde Python de un módulo implementado en C++. Se parte de la premisa de que tenemos dos módulos, uno escrito en Python y otro en C++. El módulo de Python, es el detector facial y el clasificador de género de un rostro, ambos explicados en la sección anterior. Y el módulo de C++ tiene la funcionalidad de detectar la orientación hacia la que está girado un rostro. El objetivo por tanto es combinar ambos módulos y usar sus funcionalidades al mismo tiempo sin tener que reescribir ninguno de ellos al otro lenguaje, ya que sería muy costoso.

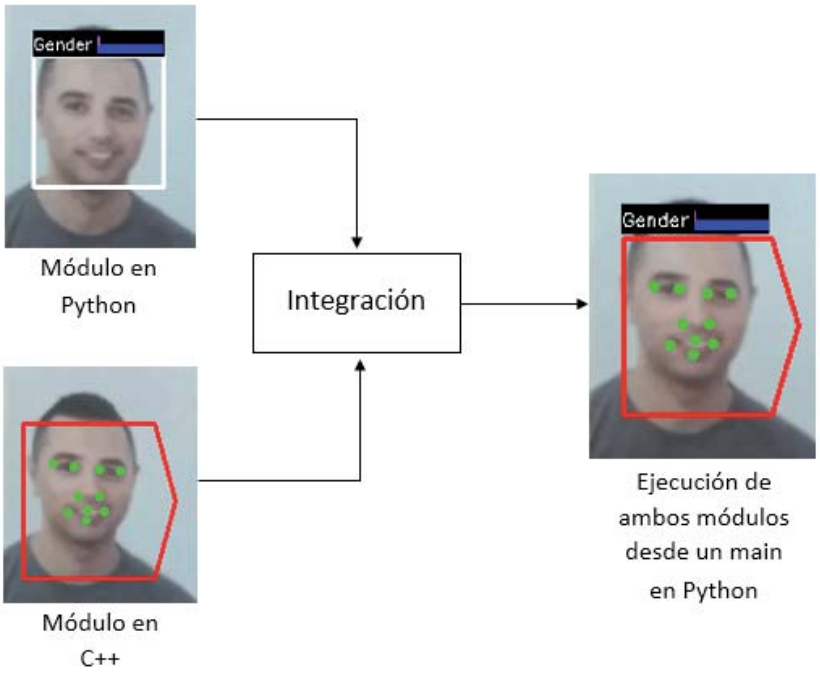


Figura 18: Resultado de combinar la funcionalidad de ambos módulos

Para conseguir la integración de ambos módulos y poder ejecutarlos desde un mismo método *main* escrito en Python, necesitamos saber como poder ejecutar código C++ desde Python. Aquí reside uno de los problemas más importantes de esta parte.

Primero hay que especificar bien el cómo vamos a utilizar desde Python el módulo escrito en C++. Esto es muy importante porque dependiendo de estos requisitos de utilización elegiremos la herramienta más adecuada para ellos. Digo esto porque ha sido uno de mis errores cometidos en el diseño y desarrollo de esta tarea. Por tanto, el objetivo es poder crear instancias desde Python de una clase compleja implementada en C++, y manejarlas como si fuesen instancias de una clase de Python. Lo que quiero decir con esto, es que con la instancia almacenada en una variable en Python, podamos llamar a métodos de esa clase (escritos en C++) como lo hacemos en Python *instancia.método(parámetros)* y que funcione correctamente.

Lo que se puede prever a priori es que en caso de que consigamos que funcione correctamente, va a haber un problema de tipos de datos entre Python y C++. Por consiguiente, este sería el segundo objetivo, hacer una clase que sea capaz de transformar los tipos de datos de C++ a tipos de Python y viceversa.

5.2.2. Funcionalidad del código

En este caso, como simplemente me he dedicado a hacer el encapsulamiento del código C++, no sé cómo es el funcionamiento de este módulo a nivel de código. Por esta razón, explicaré un poco por encima el funcionamiento general de este módulo.

El funcionamiento de este módulo es muy parecido al del otro módulo, ya que también tiene un detector facial, y un detector de ojos, nariz y boca aunque este último es más preciso y obtiene más puntos por cada parte. Por tanto, este módulo también recibe como parámetro de entrada una imagen o un vídeo (en tiempo real o un archivo) y procede a detectar todos los rostros que aparecen en la escena. Una vez las ha detectado, extrae la información útil de cada uno y la procesa para saber hacia dónde está orientado el rostro, hacia la derecha o hacia la izquierda. En la figura siguiente se puede ver el resultado final de ejecutar este módulo.

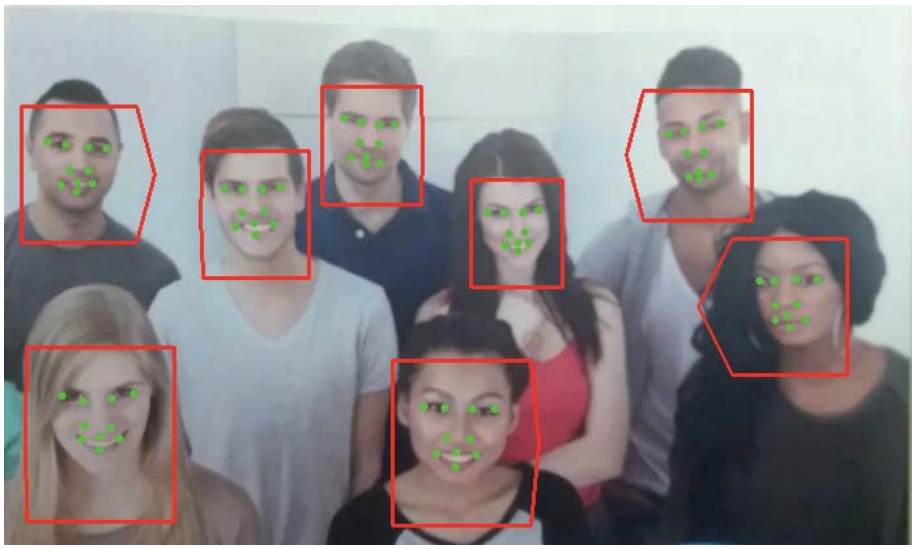


Figura 19: Resultado de aplicar el módulo escrito en C++ a una imagen

5.2.3. Herramientas posibles y utilizadas

El primer paso, fue una búsqueda exhaustiva de información sobre cómo funcionaba la encapsulación entre diferentes códigos porque nunca había hecho nada parecido antes. Una de las primeras páginas útiles que encontré incluía una gran cantidad de herramientas para diferentes comunicaciones entre código escrito en C++ y Python [18]. Como no tenía mucho conocimiento de como funcionaba esta comunicación fui seleccionando las herramientas en función de los primeros objetivos.

Estas primeros objetivos no son más que formas erróneas de entender el objetivo antes descrito. Hay mucha diferencia entre cada una de las herramientas existentes, el caso está en saber específicamente para qué se necesita la herramienta auxiliar.

De este modo, al principio pensé que solamente era necesario hacer una llamada a un método *main* escrito en C++ que ejecutase todo elcódigo del módulo y el mismo método mostrase por pantalla los resultados. Esto obviamente es lo más sencillo de hacer, y es lo que se llama hacer un *binding* entre dos lenguajes. Así que la herramienta que consideré que era la más adecuada era PyBindGen.

Pasaron unos días, y después de estudiarme como funcionaba esta herramienta conseguí ejecutar un ejemplo sencillo con alguna clase escrita en código C++. Sin embargo, cuando quise utilizar la herramienta con el módulo completo empezó a fallar debido a que no soporta archivos de cabeceras *.hpp*, sólo soporta cabeceras de lenguaje C *.h*. Así que tuve que buscar otra herramienta que trabajase con cabeceras de código C++ *.hpp* que eran las que tenía implementadas el módulo de la detección de la orientación de un rostro.

La siguiente herramienta que escogí fue Cython, teniendo en mente el mismo objetivo que con PyBindGen, una llamada a un *main* en C++ que se encargase de todo. Con esta herramienta conseguí hacerlo, desde un método en Python lograba invocar el método *main* del módulo y funcionaba perfectamente. Pero aquí existía un problema, y era que de esta manera no era posible integrar ambos sistemas porque como todo lo hacía el código de C++, desde Python no se podía procesar la misma imagen. Era como si el módulo fuese un archivo ejecutable y lo arrancásemos, te devuelve el resultado pero no se puede interactuar con él. Esto supuso un gran imprevisto porque el aprendizaje de esta herramienta fue bastante complicado, y muy costoso en lo que a tiempo se refiere. En principio, me dio la impresión de que había perdido mucho tiempo aprendiendo herramientas que no sirvieron finalmente para nada, pero en realidad ese tiempo estuvo bien invertido, ya que me facilitó la utilización de la última herramienta.

Al comentar el error en el laboratorio, me explicaron con todo lujo de detalles cómo debían interactuar el módulo de Python con el de C++. Por consiguiente, me puse inmediatamente a buscar otra herramienta que fuera capaz de crear y almacenar una instancia de una clase en C++, y manejarla desde Python como si fuese un objeto propio de Python. Así que, la última herramienta que elegí fue Boost y con esta conseguí concluir esta tarea de manera satisfactoria.

Para terminar esta introducción sobre estas tres herramientas, resumiré un poco su funcionamiento ya que es muy similar en las tres. Las tres herramientas aparte de sus propias bibliotecas requieren los siguientes componentes: el código en C++ y sus cabeceras, una clase en C++ que se encargará de la conversión de tipos y de encapsular el código C++, y un fichero Python *setup.py* que se encargara de compilar todo generando una biblioteca dinámica que podrá ser importada directamente en cualquier fichero Python para su posterior uso.

5.2.4. Generación de biblioteca dinámica en C++

Para la generación de la biblioteca dinámica serán necesarios los siguientes ficheros: *setup.py*, un fichero *.cpp* que encapsule los métodos y convierta los tipos de datos. El segundo fichero no tiene por qué ser único, se puede dividir en varios ficheros *.cpp* y sus cabeceras *.hpp* para tener el código de una forma más modular y clara. En los siguientes apartados explicaré para qué sirve cada uno de estos ficheros, cómo implementarlos y cómo utilizarlos para que se genere la biblioteca dinámica correctamente.

Fichero setup.py

El fichero setup.py contiene un pequeño script en Python para la creación de módulos para Python con la Ptyhon C API. Esta API permite la implementación de módulos con funcionalidades específicas, escritos en C/C++, y extienden la funcionalidad del intérprete de Python.

Con el fin de automatizar el proceso de compilación de todos los ficheros que compondrán nuestro nuevo módulo, utilizaremos el paquete de Python *Distutils* [22]. Con este paquete podremos describir en el script *setup.py* de qué estará compuesto nuestro módulo, y al ejecutarlo, él mismo se encargará de compilar todo el módulo con las opciones pertinentes. En dicho script indicaremos las bibliotecas y ficheros de código C/C++ que compondrán el nuevo módulo, así como el nombre que tendrá el mismo.

Como he trabajado en un entorno Linux, he utilizado también en este script el paquete *subprocess* de Python [23]. Este paquete permite la ejecución de comandos que se utilizan en bash (consola). Su uso es para localizar las bibliotecas de OpenCV y Boost instaladas en el sistema, de esta manera no será necesario saber o buscar dónde están instalados.

Una vez introducidas todas las partes de este script, voy a explicar detalladamente las funciones que he utilizado en su implementación.

```
1  from distutils.core import setup
2  from distutils.extension import Extension
3  import subprocess
4
5  proc_libs = subprocess.check_output("pkg-config --libs opencv".split())
6  libs = [lib for lib in str(proc_libs).split()]
7
8  libs.append('/usr/lib/x86_64-linux-gnu/libboost_serialization.so')
9  libs.append('/usr/lib/x86_64-linux-gnu/libboost_system.so')
10 libs.append('/usr/lib/x86_64-linux-gnu/libboost_filesystem.so')
11 libs.append('/usr/lib/x86_64-linux-gnu/libboost_thread.so')
12 libs.append('/usr/lib/x86_64-linux-gnu/libpthread.so')
13 libs.append('/usr/lib/x86_64-linux-gnu/libboost_python.so')
14
15 setup(
16   ext_modules = [Extension("faceforest",
17     sources = ["./modulos.cpp","./src/face_utils.cpp","./src/FaceForest.cpp",
18                "./src/HeadPoseSample.cpp","./src/ImageSample.cpp",
19                "./src/MPSample.cpp","./src/Viewer.cpp", "./Conversion.cpp" ],
20     include_dirs=[".", "./include", "/usr/local/include/opencv","/usr/include"],
21     extra_link_args=libs
22   )]
23 )
```

Figura 20: Código del script setup.py

En el código se pueden apreciar tres partes: la inclusión de paquetes de Python que se utilizan en este fichero, la búsqueda y almacenamiento de las rutas de las bibliotecas de OpenCV y Boost, y el setup.

La sección que se encarga de la búsqueda y almacenamiento de las bibliotecas de herramientas auxiliares utilizadas en el código C/C++, OpenCV y Boost en mi caso, va desde la línea 5 hasta la 13 del código que muestra la [Figura 20]. La variable *proc_lib* contiene una lista con las rutas de todas las bibliotecas de OpenCV. Estas rutas se han obtenido con el comando de bash *pkg-config –libs opencv*, que devuelve las rutas absolutas de dónde se encuentran dichas bibliotecas instaladas en el sistema. Estas rutas para poder ser utilizadas necesitan ser procesadas para convertirlas en un formato que Python pueda manejar y entender. Así que, la línea 6 se encarga de procesar la cadena con todas las rutas juntas separadas por un espacio, en una lista de cadenas de texto. Esta sería la forma automática de añadir librerías, pero en caso de que este comando no las encuentre, o surja cualquier problema que no supiéramos solucionar tendríamos que introducirlas manualmente, como es el caso de la parte del código desde la línea 8 a la 13 de la figura anterior. Como ya tenemos una lista en la cual cada elemento es una cadena de texto que representa una ruta, simplemente tenemos que añadir más elementos a esa lista.

Y la última parte, sería la de la función *setup* que comprende de la línea 15 a la 22 del código del script.

```
15  setup(
16    ext_modules = [Extension("faceforest",
17      sources = ["./modulos.cpp","./src/face_utils.cpp","./src/FaceForest.cpp",
18                 "./src/HeadPoseSample.cpp","./src/ImageSample.cpp",
19                 "./src/MPSample.cpp","./src/Viewer.cpp", "./Conversion.cpp" ],
20      include_dirs=[".", "./include", "/usr/local/include/opencv","/usr/include"],
21      extra_link_args=libs
22    )]
```

Figura 21: Código de la función setup del paquete Distutils

Ya hemos localizado todas las bibliotecas de las herramientas auxiliares que utiliza el módulo escrito en C++. Ahora falta añadir al nuevo módulo los ficheros *.cpp* y cabeceras *.hpp* que se encargan del funcionamiento principal del módulo. Como se puede observar en la Figura 21 dentro de la función *setup* se encuentra el argumento *ext_modules*, que es una lista de los módulos que se van a generar. Cada uno de estos módulos es una instancia de *Extension*, y contiene la siguiente información:

- *name:* Cadena de texto que representará el nombre completo que tendrá el módulo. En este caso será *faceforest*.
- *sources:* Lista de cadenas de texto que contiene las rutas de los ficheros de código fuente escritos en C++. No es posible la introducción de la ruta del directorio que contiene a todos los ficheros, es necesario añadir la ruta de la ubicación de cada fichero *.cpp*. Las rutas a introducir pueden ser tanto absolutas como relativas. En caso de ser relativas partirían del directorio donde está contenido el script *setup.py*.
- *include_dirs:* Lista de cadenas de texto que contiene las rutas de los directorios donde se encuentran las cabeceras del código C++. Es necesario incluir las cabeceras *.h* utilizadas de las herramientas auxiliares. En linux suelen encontrarse en el directorio */usr/include* o */usr/local/include*.
- *extra_link_args:* Lista de cadenas de texto que contiene las rutas de las bibliotecas de las herramientas externas utilizadas ya extraídas anteriormente.

Con esto tendríamos listo para ejecutar el fichero *setup.py* y nos compile y monte nuestro nuevo módulo. Solamente nos falta una clase que encapsule las funciones necesarias del módulo de C++ y realice la conversión de tipos de C++ a Python.

Clase de encapsulamiento utilizando Boost

La clase de encapsulamiento o envoltura, conocida en inglés como *wrapper*, se encargará de envolver las funciones del módulo de C++ que queramos utilizar desde Python. Esta clase está implementada en C++ y también se encargará de realizar la conversión de tipos pertinente, ya que se van a hacer llamadas desde Python a métodos de C++, y tanto los parámetros de entrada como los de salida tienen que

ser los propios para cada sea capaz de utilizarlos. Si la conversión de tipos es muy compleja, siempre se puede dividir esta clase en varias para facilitar el entendimiento del código, como es este caso. En cambio la conversión de los tipos básicos de C++ a Python, y viceversa es automática gracias a Boost y no requiere ningún método adicional.

Como apunte, no es necesario envolver todas las funciones del módulo que queramos utilizar, de esta manera se perdería mucho tiempo en implementar una gran cantidad de código que luego no tendría ninguna utilidad. Por esta razón, es necesario saber qué objetos, qué métodos, y qué tipos de datos usarán las variables queremos manejar de manera remota desde Python.

Dicho esto, explicaré detalladamente los objetos y los métodos que he necesitado encapsular, y las conversiones de datos que he tenido que desarrollar.

Las única clase encapsulada para el manejo de forma remota del módulo C++ ha sido la siguiente:

- **FaceForest.** Clase encargada de todo el funcionamiento del sistema, es similar a *Faces_Analyzer* del módulo de la parte de la reimplementación. Su método principal es *analyze_image()*, desde el cual ejecuta el detector facial y el procesador de la detección de la orientación. Éste será el único método que será necesario encapsular de esta clase.

No sólo son necesarias las clases del módulo C++, sino que también necesitamos envolver las clases que componen estructuras de datos compuestas, empleadas pro ejemplo, para almacenar una imagen. Por consiguiente, las clases encapsuladas que están relacionadas con estructuras de datos complejas son las siguientes:

- **FaceForestOptions.** Está compuesto por las opciones de configuración que se usarán para la detección de la orientación de un rostro.
- **ForestParam.** Contiene las opciones de cómo se realizará la búsqueda en los diferentes conjuntos de arboles o bosques que se introducen en el sistema.
- **FaceDetectionOption** Contiene variables con valores relativos a cómo se realizará la detección de rostros en una imagen.
- **Mat.** Es un tipo de estructura de datos de C++ que usa OpenCV para manejar las imágenes en el sistema.
- **Vector<Face>.** Consiste en un vector de objetos Face. Lo usa el detector facial del módulo C++ para almacenar información relativa de cada rostro detectado en una imagen. No será necesaria la envoltura del tipo Face, ya que será convertido en la clase de encapsulación a un tipo lista de Python.

Por último, quedaría explicar los tipos de datos que he necesitado convertir para el correcto funcionamiento de comunicación entre Python y C++. Sólo ha sido necesario la conversión de dos tipos y en una única dirección. Una ha sido de un objeto de tipo *numpy.ndarray* (Python) a un objeto *Mat* (OpenCV-C++), y la otra de un objeto *vector<Face>*(C++) a un objeto lista de listas (Python). La razón de la primera conversión es que como el sistema de recepción de imágenes está en código Python, esta imagen será capturada en una variable con el tipo *numpy.ndarray*. Esta imagen tendrá que ser introducida en el módulo que queremos crear para que la procese y nos devuelva los resultados de la orientación de los rostros que aparecen en ella. Pero aquí tenemos el problema, si le pasamos la imagen tal como está, al intentar procesarla y no ser del tipo *Mat* producirá un error.

En cuanto a la otra conversión, los resultados al procesar la imagen por parte del módulo, están almacenados en un objeto de tipo *vector<Face>*. Este objeto contiene en cada elemento los datos de la región que ocupa cada rostro y las coordenadas relativas de diversos puntos de interés. Esos datos para poder ser mostrados por pantalla tienen que ser devueltos a Python al terminar la ejecución del método de procesamiento. Pero si no se convierten antes, cuando en el código de Python se intenta acceder a la variable que contiene ese objeto *vector<Face>*, devolverá un error al no ser un tipo de datos utilizado en Python.

A continuación, al igual que con el fichero *setup.py*, detallaré todos los pasos que he llevado a cabo para el desarrollo de la clase de encapsulamiento o envoltura. Para empezar, tomaremos una visión general de la clase completa en la Figura 22.

```cpp

#include <FaceForest.hpp>
#include <face_utils.hpp>
#include <Constants.hpp>
#include <boost/python.hpp>
#include <opencv2/core/core.hpp>
#include <Conversion.hpp>


using namespace boost::python;

// Converts a std::vector<Face> into a pylist and returns 3 lists:
//      -hp_list: list of headposes from each face
//      -bbox_list: list of bboxes from each face
//      -ffd_list: list off ffd_coordinates from each face
void vector_to_pylist (
    const std::vector<Face>& vect, list faces
)
{
    for (unsigned long i = 0; i < vect.size(); ++i)
    {
        list aux;
        list hp_list;
        list bbox_list;
        list ffd_list;
        hp_list.append(vect[i].headpose);
        bbox_list.append(vect[i].bbox.x);
        bbox_list.append(vect[i].bbox.y);
        bbox_list.append(vect[i].bbox.width);
        bbox_list.append(vect[i].bbox.height);
        for (unsigned long j = 0; j < vect[i].ffd_cordinates.size(); ++j){
            list ffd_local;
            ffd_local.append(vect[i].ffd_cordinates[j].x);
            ffd_local.append(vect[i].ffd_cordinates[j].y);
            ffd_list.append(ffd_local);
        }
        aux.append(hp_list);
        aux.append(bbox_list);
        aux.append(ffd_list);
        faces.append(aux);
    }
}

// Convert python list to cv::Mat
cv::Mat numpy_to_Mat(PyObject *np_array)
{
    NDArrayConverter cvt;
    cv::Mat mat = cvt.toMat(np_array);
    return mat;
}

BOOST_PYTHON_MODULE(faceforest)
{
    class_<FaceForest>("FaceForest",init<FaceForestOptions>())
        .def("analyzeImage", &FaceForest::analyzeImage)
    ;

    class_<cv::Mat>("Mat")
    ;

    class_<std::vector<Face> >("vectorFace")
    ;

    class_<FaceForestOptions>("FaceForestOptions")
        .def_readwrite("hp_forest_param", &FaceForestOptions::hp_forest_param)
        .def_readwrite("mp_forest_param", &FaceForestOptions::mp_forest_param)
        .def_readwrite("fd_option", &FaceForestOptions::fd_option)
    ;

    class_<ForestParam>("ForestParam")
    ;

    class_<FaceDetectionOption>("FaceDetectionOption")
        .def_readwrite("path_face_cascade", &FaceDetectionOption::path_face_cascade)
    ;

    // CONVERTER FUNCTIONS
    def("numpy2Mat", &numpy_to_Mat);
    def("vector2pylist", &vector_to_pylist);
    def("loadConfigFile", &loadConfigFile);
}
```

Figura 22: Código de la clase de encapsulamiento

Lo primero que se debe hacer es incluir la cabecera de *boost/python.hpp* para poder utilizar el código de envoltura. El resto de *#include* pertenecen a las cabeceras de las clases, de las cuales vamos a encapsular alguna parte que contengan, ya sea un tipo de datos compuesto o un método. Excepto la cabecera *Conversion.hpp*, que contiene en varios métodos para convertir un objeto de tipo *numpy.ndarray* a un objeto de tipo *Mat* de OpenCV-C++.

También para el desarrollo de esta clase ha sido necesario el uso de varios espacios de nombres: el de la herramienta Boost,*boost::python* para todo el sistema de conversión de tipos y objetos Python en esta clase de C++, el de la biblioteca estándar de C++, *std::vector* para el manejo de la estructura de datos compuestos antes mencionada de *vector<Face>*, y el de la herramienta OpenCV *cv::Mat* para la gestión los tipos de datos que tienen las imágenes en C++.

Ahora continuaremos con las funciones para la conversión de datos. Como ya comenté anteriormente tenemos dos conversiones diferentes. La más sencilla es de *vector<Face>* a una lista de listas. Para conseguir esto se utilizará el método:

- **vector_to_pylist(std::vector<Face> vect, list faces);** Este método tiene dos parámetros, *vect* que es el de entrarda y *list* el de salida. Este método accederá a cada objeto *Face* contenido en el vector, y extraerá sus tres estructuras de datos que lo componen: *headpose*, *bbox* y *ffd_coordinates*. Headpose es un número decimal que indica la orientación de la cara, bbox es una estructura de cuatro elementos que representan un rectángulo que contiene el rostro, y ffd_coordinates es un vector de puntos que indican la posición relativa de cada punto de interés del rostro.

  Como bbox y ffd_coordinates no son tampoco tipos básicos será necesario acceder a cada uno de sus elementos de forma individual. El formato final en el que se convierte este vector es una lista de listas, dónde cada sublista representa un objeto *Face* que contiene tres listas. Una con el número que representa la orientación de la cara, otra con los cuatro elementos que definen el rectángulo que contiene el rostro y la última contendrá por pares [x,y] todos los puntos de interés de la cara.

Y la segunda conversión, de un objeto *numpy.ndarray* a un objeto *Mat* de OpenCV-C++. Debido a la complejidad de este procedimiento fue necesaria la creación de una clase aparte que se encargase de la conversión. En el código de la Figura 22 en el método **numpy_to_Mat(PyObjec *np_array)** sólo se encuentra la creación de un objeto *NDArrauConverter*, y la ejecución del método de ese mismo objeto *toMat()*. Esta clase empecé a implementarla en un principio, pero existían una gran cantidad de problemas a la hora de gestionar los tipos básicos de los elementos de un objeto *Mat* de OpenCV debido a que se requería un conocimiento superior de C++. Así que, indagando por internet, en github encontré un par de clases [24] que habían extraído y rediseñado la parte del código que proporciona OpenCV para

convertir listas en matrices y como funcionaban perfectamente decidí utilizarlas en el proyecto.

Para concluir esta clase de envoltura solo falta definir cómo vamos a encapsular todos los componentes (clases, tipos de datos y métodos) definidos en C++ desde Python. Para esto será necesario la creación de la siguiente sección de código (líneas 52-80 de la clase de encapsulamiento):

```
51
52  BOOST_PYTHON_MODULE(faceforest)
53  {
54      class_<FaceForest>("FaceForest",init<FaceForestOptions>())
55          .def("analyzeImage", &FaceForest::analyzeImage)
56      ;
57
58      class_<cv::Mat>("Mat")
59      ;
60
61      class_<std::vector<Face> >("vectorFace")
62      ;
63
64      class_<FaceForestOptions>("FaceForestOptions")
65          .def_readwrite("hp_forest_param", &FaceForestOptions::hp_forest_param)
66          .def_readwrite("mp_forest_param", &FaceForestOptions::mp_forest_param)
67          .def_readwrite("fd_option", &FaceForestOptions::fd_option)
68      ;
69
70      class_<ForestParam>("ForestParam")
71      ;
72
73      class_<FaceDetectionOption>("FaceDetectionOption")
74          .def_readwrite("path_face_cascade", &FaceDetectionOption::path_face_cascade)
75      ;
76
77      // CONVERTER FUNCTIONS
78      def("numpy2Mat", &numpy_to_Mat);
79      def("vector2pylist", &vector_to_pylist);
80      def("loadConfigFile", &loadConfigFile);
81  }
```

Figura 23: Código de envoltura de la clase de encapsulamiento

Para definir la envoltura primero tenemos que escribir la línea

BOOST_PYTHON_MODULE(nombre del módulo)

IMPORTANTE: Para evitar problemas en la generación de la biblioteca dinámica el nombre del módulo introducido en la clase de encapsulamiento como parámetro de *BOOST_PYTHON_MODULE* **debe ser el mismo** que el nombre que se le dio como parámetro a *Extension* en el fichero *setup.py*.

Una vez definido el módulo, sólo tenemos que ir definiendo cada clase de la siguiente forma:

class_<NombreDeLaClaseEnC++>("NombreParaEjecutarDesdePython")

En caso de que necesitemos utilizar el constructor de alguna clase como en la línea 54 de la Figura 23, habría que añadir:

class_<NomClassC++>("NomPython",**init<Parámetro><Parámetro>()**)

Para la definición de funciones o métodos, dentro de la envoltura de una clase se tendrá que escribir:

.def("NombreEjecutarDesdePython", &NombreClaseC++::MétodoC++)

En cuanto a la envoltura de variables que están dentro de una clase:

.def("NombreVariableEnPython", &NombreClaseC++::VariableC++)

Y por último, necesitamos encapsular los métodos que hemos en la parte superior de esta clase para la conversión de los tipos complejos de datos. Estos métodos al haber sido definidos en esta misma clase van dentro del módulo pero fuera de cualquier envoltura de clase.

.def("NombreMetodoEnPython", &NombreMétodoC++)

Con este último paso ya tenemos terminados los dos ficheros necesarios, setup.py y la clase de encapsulamiento, para poder generar la biblioteca dinámica. Ahora simplemente lo que hay que hacer es abrir una instancia de la consola de Linux y situarnos en el directorio donde se encuentran los dos ficheros antes mencionados. Ahora ejecutaremos la siguiente línea en la consola:

$ python setup.py build_ext –inplace

Después de esperar un tiempo, ya que tiene que compilar una gran cantidad de bibliotecas y ficheros, si todo funciona correctamente habrá generado una biblioteca dinámica **nombre.so** con el nombre que le dimos en el fichero *setup.py* en el mismo directorio. Este archivo es directamente válido para ser importado en cualquier fichero de código Python de la misma manera que el resto de paquetes o bibliotecas.

5.2.5. Funcionamiento del nuevo módulo

Al igual que la inclusión de la nueva biblioteca dinámica generada en cualquier clase de Python no necesita ningún tratamiento especial, la creación de instancias y la llamada a sus métodos, definidos en la clase de encapsulamiento, es de la misma forma que una clase definida en Python. Seguidamente, veremos como se comporta el nuevo módulo siendo gestionado desde una clase Python.

La clase de Python que se encargará de controlar la ejecución necesitará de las siguientes partes para poder ejecutar correctamente el nuevo sistema de reconocimiento facial y el detector de orientación de un rostro:

- **Gestión de los parámetros de entrada: imágenes y vídeo**
- **Invocación de los métodos de la nueva biblioteca**
- **Creación de una clase para devolver todos los resultados por pantalla**
- **Implementación de un método para mostrar la orientación de un rostro**

De las partes anteriores, ya están implementadas dos de ellas en la primera parte de este trabajo (reconocedor facial y clasificador de género). Estas son la gestión de las imágenes y vídeo en la entrada del sistema, y la creación de una clase para pintar todos los resultados (la clase *Viewer*). Por consiguiente, sólo quedaría la implementación relacionada con ejecutar los métodos de la biblioteca y la interpretación de los resultados que devuelve.

En cuanto cómo interpretar los datos relativos a la orientación del rostro se utilizará el siguiente método:

- **show_results(proc_face);** Este método recibe como parámetro de entrada una lista de listas, que se corresponde con un objeto *Face* como ya comentamos en el apartado de la clase de encapsulamiento. Lo que hace primero este método es extraer la región que contiene el rostro, todos sus puntos de interés y el número que indicaba la orientación del mismo. Luego, pinta todos los puntos de interés y comprueba el valor de la orientación. En función de si ese valor es mayor o menor que 0, se dibujará una flecha hacia la izquierda o la derecha respectivamente.

El diagrama de secuencia de la figura siguiente muestra como es la interacción entre la clase principal de Python, la biblioteca dinámica, y la clase *Viewer* para crear una instancia del *Viewer* que se encargará de mostrar los resultados, y de la clase *FaceForest* que se encargará de controlar el detector facial e el de orientación de un rostro.

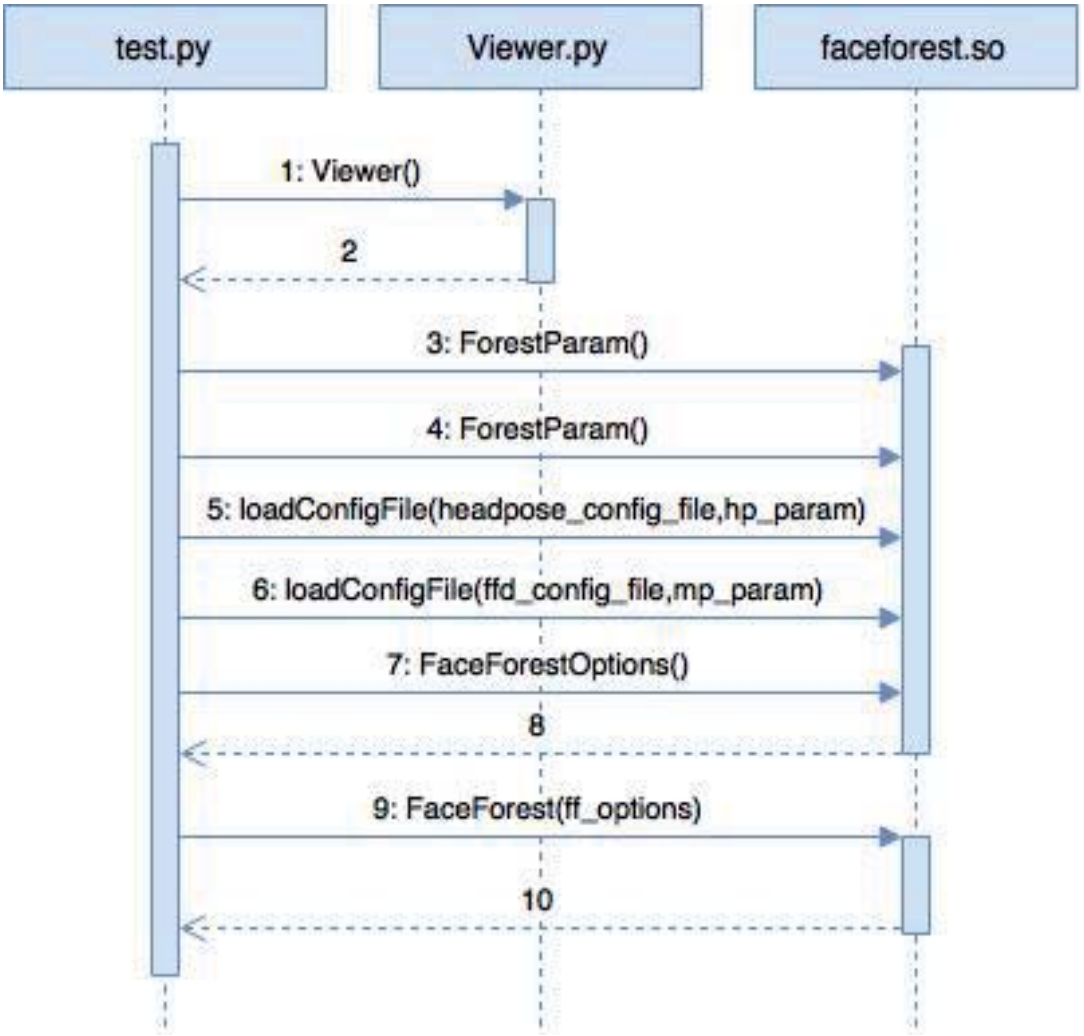


Figura 24: Diagrama de secuencia para representar la inicialización de las instancias de la clase Viewer y FaceForest

Si nos fijamos en el diagrama hasta ahora sólo se ha ejecutado la primera parte del código, todavía falta la captura de imágenes y su posterior procesamiento. De nuevo, utilizaré un diagrama de secuencia para ver como es la interacción entre las diferentes partes del sistema a la hora de procesar una imagen o fotograma.

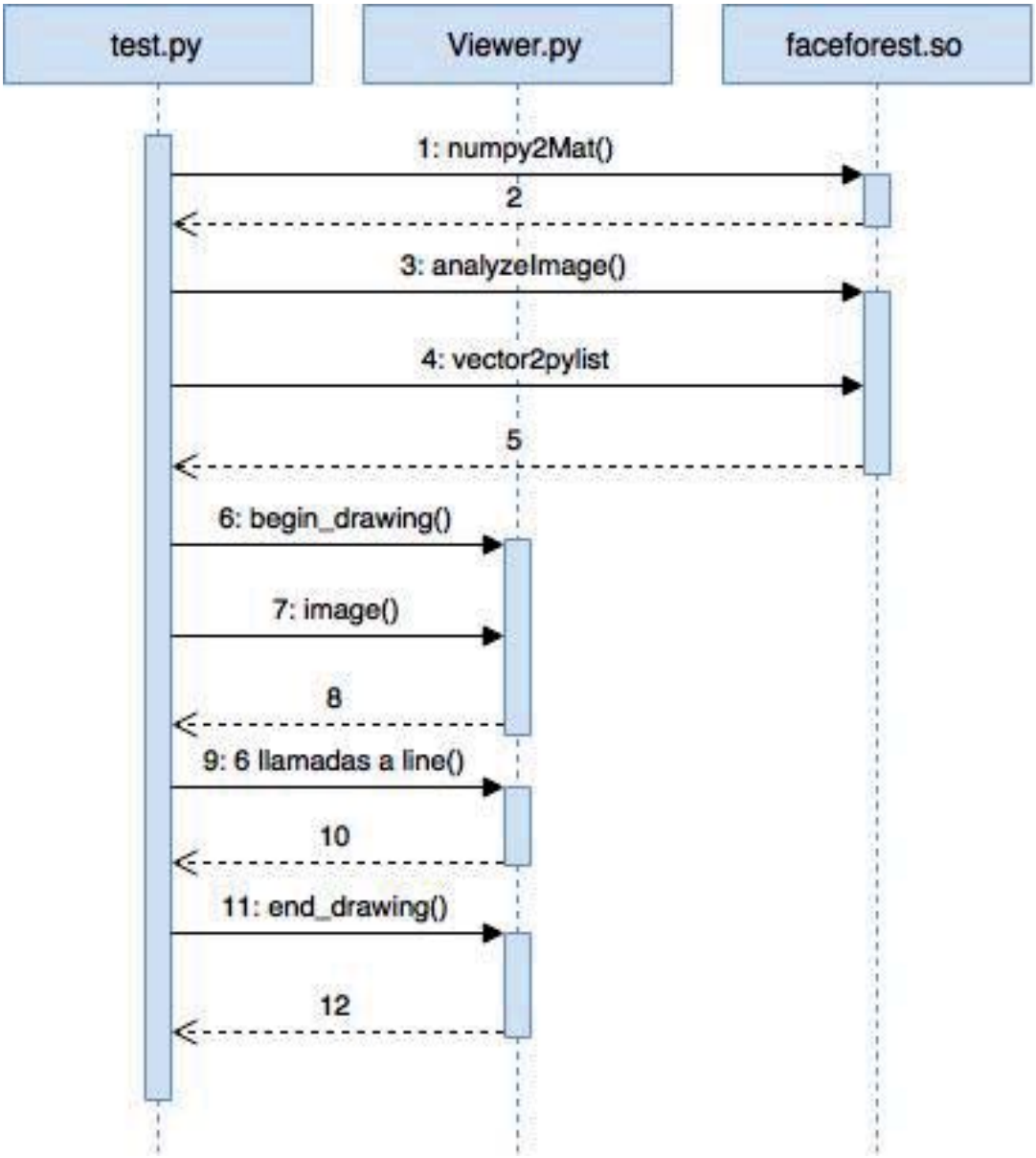


Figura 25: Diagrama de secuencia para representar el procesamiento de una imagen en el sistema

5.2.6. Análisis de resultados

Lo único que quedaría por analizar con respecto a esta parte sería el tiempo que tarda el código y la visualización de los resultados. El tiempo de ejecución es aproximadamente el mismo si lo ejecutamos con la librería dinámica, que si lo ejecutamos como estaba diseñado el módulo originariamente en C++. Donde podría consumir más tiempo sería a la hora de mostrar los resultados por pantalla, pero he utilizado el método de Python *map()* que permite aplicar una función simultáneamente a todos los elementos de una lista. Con esto consigo que la función descrita anteriormente *show_results()* no tenga apenas coste de ejecución.

Los resultados de la ejecución de este código se pueden ver en la Figura 26 sobre un vídeo grabado por mi mismo sobre un documento de publicidad en el que aparecen ocho personas jóvenes.

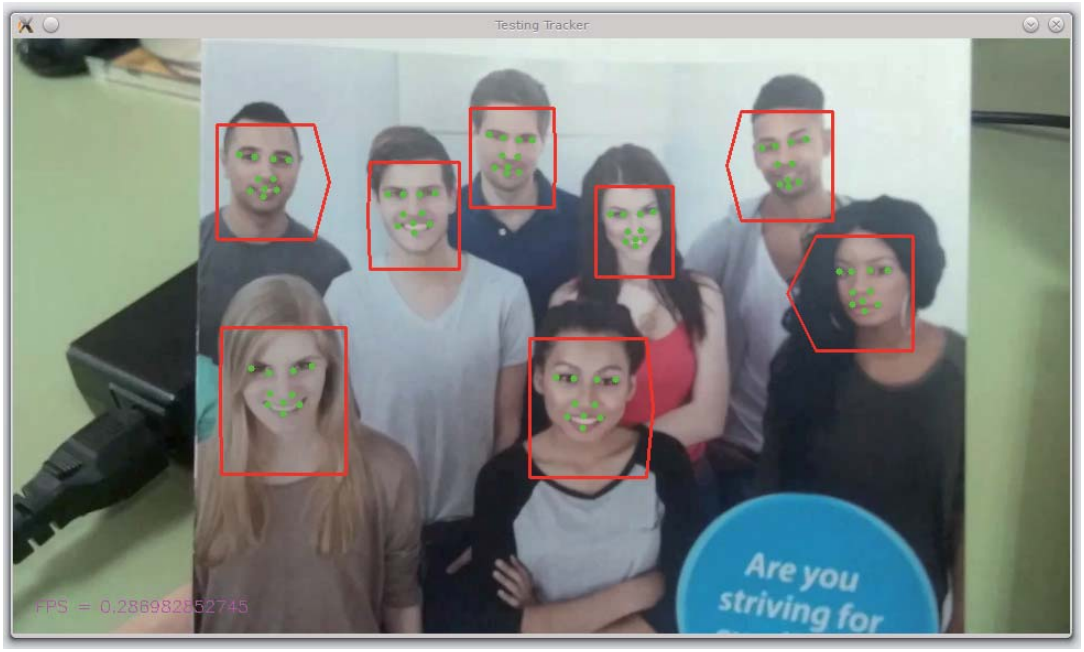


Figura 26: Resultado obtenido al procesar con este módulo un vídeo casero

Se puede observar en la figura anterior que detecta todos los rostros y todos los puntos de interés de cada parte, cosa que no ocurría en la primera parte de este trabajo. En el primer reconocedor facial había muchas partes que no se llegaban a localizar a no ser que estuviera en condiciones óptimas de iluminación. Sin embargo, en el segundo acierta prácticamente en cualquier situación salvando algunos casos particulares. A continuación, podemos observar el máximo número de partes que es capaz de localizar el primer detector facial, porque no en todos los fotogramas detecta todos los que aparecen en la Figura 27.

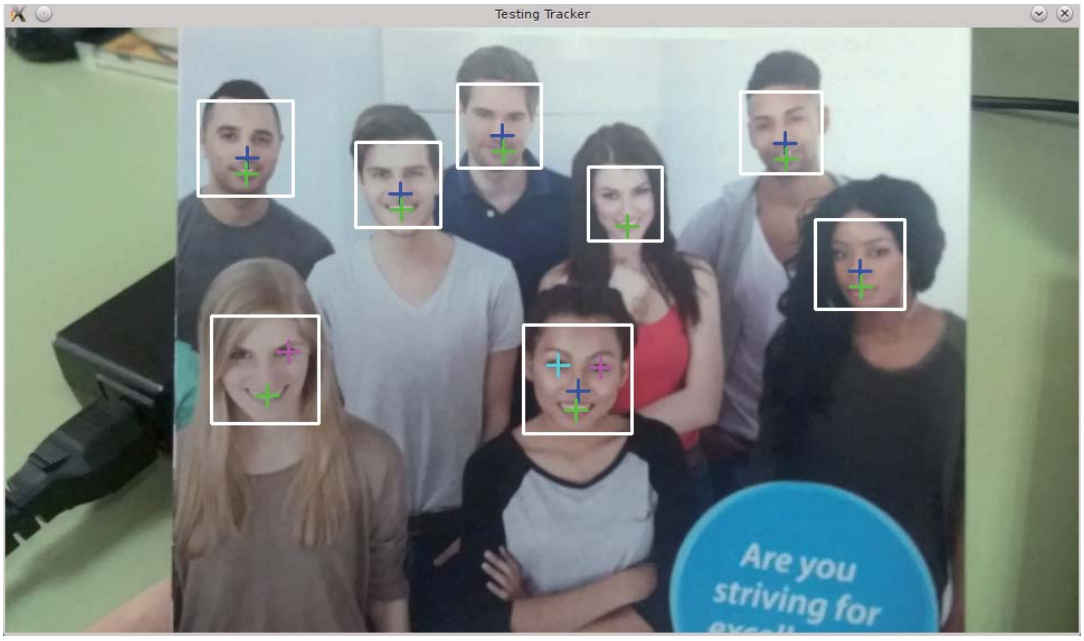


Figura 27: Resultado obtenido al procesar un vídeo casero con el primer módulo

5.3. Integración

Ya en este punto de desarrollo del trabajo de fin de grado hay desarrollados dos módulos diferentes, el primero encargado de detectar rostros y clasificar al género al que pertenecen, y el segundo capaz también de localizar rostros en una imagen como de detectar la posición hacia la que están orientados. Ahora la parte que faltaría sería la de combinar las funcionalidades que nos ofrecen ambos módulos y tener un sistema más o menos completo.

La integración de estos dos módulos o sistemas, no ha sido para nada complicada. La razón es que la clase principal de ambos reconocedores hacen lo mismo a grandes rasgos. Por ejemplo, ambas clases tienen que crear una instancia de una clase *Viewer* (que es la misma en ambos módulos) para mostrar los resultados por pantalla, y ambas tienen tienen que procesar tanto imágenes como vídeos. El dato más importante es que el funcionamiento del segundo módulo reside en la biblioteca dinámica, ya que puede ser utilizada en cualquier programa escrito en Python. Esto significa que simplemente con un *import* de la biblioteca, y la combinación de las clases principales de ambos módulos ya tendríamos lista la integración para funcionar.

5.3.1. Modificaciones aplicadas a código anterior

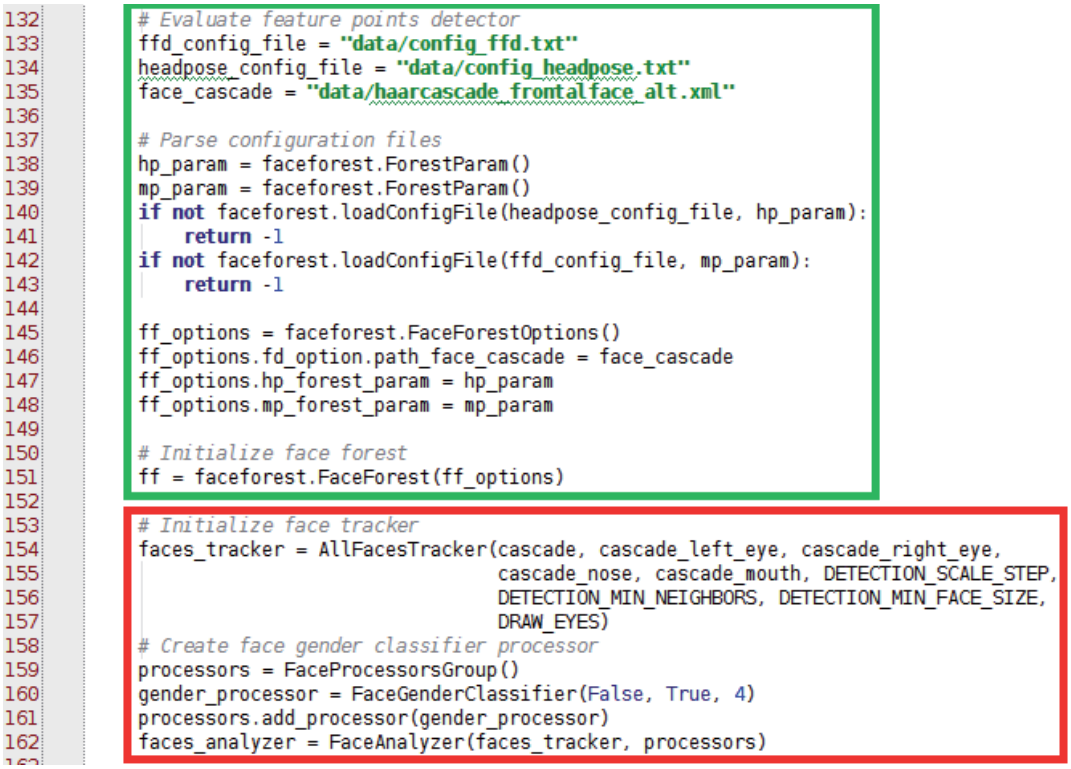
Para explicar de una forma más detallada en qué ha consistido la combinación antes mencionada comentaré algunos fragmentos de código destacable. De cada fragmento indicaré en rojo las partes que pertenecían al primer módulo y en verde las

que pertenecían al segundo.

```
25  viewer = Viewer()
26
27
28  def show_results(proc_face):
29      red_color = [0, 0, 255]
30      green_color = [0, 255, 0]
31
32      # Print points
33      for point in proc_face[2]:
34          x = point[0] + proc_face[1][0]
35          y = point[1] + proc_face[1][1]
36          viewer.circle(x, y, 3, -1, green_color)
37      x, y, width, height = proc_face[1]
38      a = [x, y]
39      b = [x+width, y]
40      c = [x, y+height]
41      d = [x+width, y+height]
42      y1 = [x, y+(height/2)]
43      y2 = [x+width, y+(height/2)]
44      yaw = int(proc_face[0][0] * 80)
45      if yaw < 0:
46          y2[0] -= yaw
47      else:
48          y1[0] -= yaw
49
50      viewer.line(a[0], a[1], b[0], b[1], red_color, 2)
51      viewer.line(c[0], c[1], d[0], d[1], red_color, 2)
52      viewer.line(a[0], a[1], y1[0], y1[1], red_color, 2)
53      viewer.line(y1[0], y1[1], c[0], c[1], red_color, 2)
54      viewer.line(b[0], b[1], y2[0], y2[1], red_color, 2)
55      viewer.line(y2[0], y2[1], d[0], d[1], red_color, 2)
```

Figura 28: Primera sección destacable de la clase principal de la integración

En esta figura podemos observar la creación de la instancia de la clase *Viewer* como variable global de esta clase. Es necesario que sea global y no local, porque si no, no se puede utilizar la función *map* que permitía aplicar este método de forma simultánea a todas las caras a la vez. Es instancia no está remarcada con ningún color ya que existía en ambos módulos. En cuanto a *show_results*, es un método que fue necesario implementar para mostrar los resultados del detector de la orientación en la ejecución del segundo módulo de forma aislada. Por esa razón ha sido directamente copiado en la integración y tendrá exactamente la misma función.



```
# Evaluate feature points detector
ffd_config_file = "data/config_ffd.txt"
headpose_config_file = "data/config_headpose.txt"
face_cascade = "data/haarcascade_frontalface_alt.xml"

# Parse configuration files
hp_param = faceforest.ForestParam()
mp_param = faceforest.ForestParam()
if not faceforest.loadConfigFile(headpose_config_file, hp_param):
    return -1
if not faceforest.loadConfigFile(ffd_config_file, mp_param):
    return -1

ff_options = faceforest.FaceForestOptions()
ff_options.fd_option.path_face_cascade = face_cascade
ff_options.hp_forest_param = hp_param
ff_options.mp_forest_param = mp_param

# Initialize face forest
ff = faceforest.FaceForest(ff_options)

# Initialize face tracker
faces_tracker = AllFacesTracker(cascade, cascade_left_eye, cascade_right_eye,
                                cascade_nose, cascade_mouth, DETECTION_SCALE_STEP,
                                DETECTION_MIN_NEIGHBORS, DETECTION_MIN_FACE_SIZE,
                                DRAW_EYES)
# Create face gender classifier processor
processors = FaceProcessorsGroup()
gender_processor = FaceGenderClassifier(False, True, 4)
processors.add_processor(gender_processor)
faces_analyzer = FaceAnalyzer(faces_tracker, processors)
```

Figura 29: Segunda sección destacable de la clase principal de la integración

Esta figura contiene el código relacionado con la creación de instancias de todos y cada uno de los detectores y procesadores. El código del recuadro verde, empieza por la carga de los ficheros de configuración en una variable de tipo *FaceForestOptions*, para luego utilizarla para crear una instancia de la clase *FaceForest* que invocará su método clase *analyzeImage()* de la biblioteca dinámica para procesar la imagen. Mientras que el código del recuadro rojo, se encarga de inicializar una instancia del detector facial y de las partes del rostro, otra instancia del procesador de género, y finalmente una instancia del reconocedor facial completo, que contiene al detector y al procesador.

En este momento de la ejecución de esta clase ya está todo listo para recibir una imagen o fotograma y que el sistema pueda procesarlo correctamente.

```
180             # Convert numpy array to Mat
181             mat = faceforest.numpy2Mat(frame)
182
183             cfaces = faceforest.vectorFace()
184             ticks = cv2.getTickCount()
185             ff.analyzeImage(mat, cfaces)
186
187             # Convertir vector a pylist
188             faceforest.vector2pylist(cfaces, faces_list)
189
190             # Process frame
191             faces_analyzer.process_frame(frame)
192             ticks = cv2.getTickCount()-ticks
193
194             # Drawing results
195             viewer.begin_drawing()
196             viewer.image(frame, 0, 0, width, height)
197             map(show_results, faces_list)
198
199             # View FPS
200             fps = cv2.getTickFrequency()/ticks
201             str_fps = 'FPS = ' + str(fps)
202             viewer.text(str_fps, 20, frame.shape[0]-20, (255, 0, 255), 0.5, 1)
203
204             faces_analyzer.show_results(viewer, frame)
205             k = viewer.end_drawing(msec)
```

Figura 30: Tercera sección destacable de la clase principal de la integración

Para concluir este apartado, terminaré explicando como funciona el procesamiento de cada imagen o fotograma introducido en el sistema que se puede apreciar en la figura anterior. El código esta dividido en dos partes de la línea 180 a la 191, está el análisis y procesamiento de la imagen, mientras que de la línea 195 a la 205 está el paso de resultados por pantalla.

Una vez la imagen ha sido leída se almacena en una variable llamada *frame*. Ahora se invoca a la función *numpy2Mat()* de la biblioteca que se encarga de convertir esa imagen formateada en Python, al tipo Mat de C++. Hecho esto ahora se llama a la función *analyzeImage()* para que el segundo módulo procese la imagen y nos devuelva el vector de caras, los puntos de interés de cada una y la orientación hacia la que están giradas. El siguiente paso es convertir el vector de rostros a una lista de listas en Python, y para ello se llama al método *vector2pylist()*. Ya sólo quedaría que el detector y procesador del primer módulo analizasen la imagen. Cuando han terminado, hacen las llamadas pertinentes a los métodos *show_results()* de cada procesador para que podamos ver los resultados finales de ambos módulos integrados.

5.3.2. Resultado

Por último, este es el resultado final al procesar una imagen con la integración del código de ambos módulos.

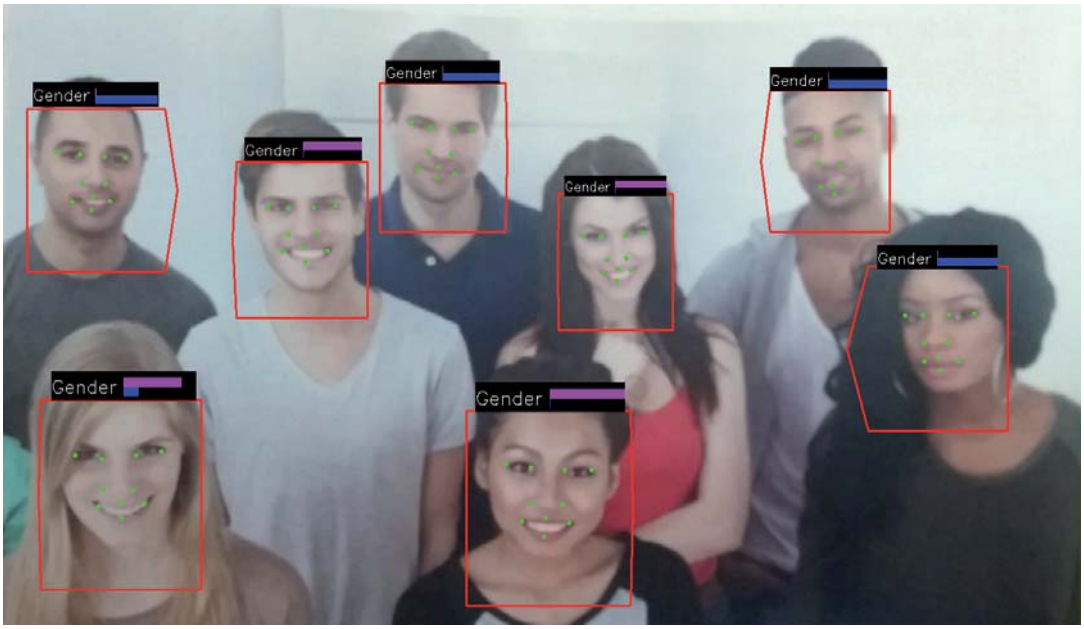


Figura 31: Resultado obtenido al procesar un vídeo casero con la integración de los dos módulos

Se puede observar que el clasificador de género falla en algunas ocasiones. Pero haciendo un análisis exhaustivo para descartar posibles errores como, por ejemplo,que no almacenase correctamente el orden de las caras, o que hubiera errores de cálculo, pude observar que las caras las procesaba correctamente. Sin embargo, al aplicar la ecualización y la máscara a la imagen de la región de la cara, no era tan fácil distinguir el género de esa persona ni para mí mismo.

6. CONCLUSIONES

En este capítulo se exponen las conclusiones que he sacado después de la realización de las distintas tareas de este trabajo. Para organizar un poco el contenido de las conclusiones y que las ideas no estén demasiado mezcladas he dividido el contenido en varias partes.

La primera parte sería lo que me ha supuesto este trabajo en cuanto a aprendizaje se refiere.

En un principio tenía nociones muy básicas tanto de Python como de C++, ya que estos lenguajes apenas los he utilizado en alguna asignatura del grado. Pero la verdad es que de C++ no hacía falta saber demasiado, ya que es muy parecido en cuanto a estructuración y funcionalidad a Java. El único obstáculo para comprender el detector de rostros y el clasificador de género fue producido por funciones y estructuras de OpenCV, más que por el propio lenguaje en sí. De todas maneras todo se hizo más llevadero gracias a la ayuda de mi tutor Luis Baumela y varios compañeros del laboratorio como Roberto Valle y Jose Miguel Buenaposada, que cualquier duda que tuviera me la resolvían sin ningún problema.

Con respecto a Python he notado una mejoría enorme. Al principio estaba un poco perdido, ya que muchas funcionalidades de Python no las conocía y digamos que perdía tiempo al no aplicar los métodos idóneos para cada situación. Pero con todas las horas aplicadas al desarrollo de este trabajo he mejorado mi nivel de partida muy satisfactoriamente.

También hay que mencionar la biblioteca de OpenCV, una herramienta muy útil para el procesamiento de imágenes. Al principio pensé que era complicado todo el sistema de procesado de imágenes y bastante lento. Sin embargo, con el paso del tiempo y con algunos métodos en Python que permiten aplicar una operación o función a todos los elementos de una matriz al mismo tiempo.

Por último, en cuanto a la adquisición de conocimientos, quería hacer referencia a las herramientas de encapsulamiento de códigos diferentes. En este caso me pasó al revés que con OpenCV, pensé al ver el planteamiento que era sencillo pero me equivoqué. Realmente, depende de cómo se vayan a comunicar los diferentes módulos de una aplicación implementados en distintos lenguajes. Digamos que en esta parte es donde más atascado me he quedado de todo el trabajo. Tuve que cambiar varias veces de herramienta y desechar semanas de trabajo. Pero aún así, me sirvió para entender en profundidad cómo funcionaban los encapsulamientos en situaciones diversas, y qué herramientas es la más recomendada para cada caso. Situaciones como ejecutar desde Python un módulo escrito en C++, crear objetos implementados en C++ desde Python, o en el caso de este trabajo, manejo de instancias de clases de C++

en Python como si fuesen de una clase de Python. Parecen situaciones muy parecidas pero hay gran diversidad de herramientas que sólo contemplan una de ellas.

Y la segunda parte trata de observaciones que tengo sobre las tareas que he desarrollado. En primer lugar, estoy contento con el trabajo que he realizado, pero si que me hubiera gustado más diseñar yo mismo un sistema de reconocimiento facial. De esta forma, hubiera tenido que esforzarme en pensar cómo hacer las cosas y no tanto en copiar lo que ya estaba escrito y ponerlo de otra manera. Por esta razón me gustó más la segunda tarea del trabajo (encapsulación de código C++ en Python) ya que no había ningún dato previo y tenía que darle vueltas hasta que saliese.

Para finalizar este capítulo, querría mencionar que un sistema de reconocimiento facial es demasiado dependiente de las características de la imagen. Como ya conté en el Capítulo 1, los problemas que afectan a las imágenes son de vital importancia. Cuando ves un sistema de reconocimiento facial por primera vez uno tiende a pensar que es muy sencillo su funcionamiento, ya que nosotros gracias nuestros ojos somos capaces de reconocer cualquier objetos en tiempos casi inexistentes. En este trabajo me he dado cuenta que si queremos detectar varios rostros simultáneamente en una misma imagen, determinar su género y la orientación hacia donde miran se necesita más que un ordenador convencional si queremos que vaya a la velocidad que captura imágenes una cámara web.

6.1. Posibles mejoras aplicables

En este apartado voy a enumerar una serie de mejoras que podrían mejorar el rendimiento del sistema:

- **Optimización del reconocedor facial.** Como ya he explicado, debido a mis pocos conocimientos sobre el lenguaje la primera parte que implementé es la parte más pobre en cuanto a optimización en Python.
- **Optimización del código C++ que se utiliza desde Python.** Una vez integrado el código, el código de C++ que se encarga de calcular la orientación del rostro y varios puntos para cada parte del mismo se aleja bastante de poder ser usado en tiempo real.
- **Detección de rostros de perfil.** La idea sería buscar otros clasificadores en cascada o mejorar los que ya existen par que sea posible reconocer un rostro girado más de 45 grados.
- **Incorporación de nuevos módulos con otros procesadores.** Ahora que es posible utilizar módulos tanto en Python como en C++, cabe la posibilidad de añadir otros módulos que ya hubiesen sido implementados o elegir el lenguaje que mejor nos parezca para hacer uno nuevo. Por ejemplo, un procesador que interprete gestos faciales.

- **Evitar redundancia de código.** Habría que retocar alguno de los módulos ya existentes, ya que en ambos se localizan los rostros y sus partes. Esto supone un gasto computacional innecesario que se podría reducir.

BIBLIOGRAFÍA

[1] Código del algoritmo de reconocimiento facial escrito en C++, por José Miguel Buenaposada.

[2] Definición de IDE.
http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

[3] Información sobre el IDE Eclipse.
https://eclipse.org/org/

[4] Información sobre el IDE Pycharm y software de JetBrains.
https://www.jetbrains.com/company/

[5] Definición de librería.
https://en.wikipedia.org/wiki/Library_(computing)

[6] Información sobre la biblioteca Numpy.
http://www.numpy.org/

[7] Información sobre la biblioteca Math.
https://docs.python.org/2/library/math.html

[8] Información sobre la biblioteca Distutils.
https://docs.python.org/3/library/distutils.html

[9] Información sobre la biblioteca Subprocess.
https://docs.python.org/3/library/subprocess.html?highlight=subprocess#module-subprocess

[10] Información sobre OpenCV.
http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf

[11] Más información sobre OpenCV.
http://es.wikipedia.org/wiki/OpenCV
[12] Información sobre la visión por computador.
https://en.wikipedia.org/wiki/Computer_vision

[13] Material de la asignatura de Reconocimiento de formas"proporcionado por Luis Baumela Molina.

[14] P.Viola y M. Jones, *Rapid object detection using a boosted cascade of simple features*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2001, Pag: 511-518.

[15] R. Lienhart, A. Kuranov y V. Pisarevsky, *Empirical analysis of detection cascades of boosted classifiers for rapid object detection*, MRL Technical Report, 2002.

[16] Como funciona el algoritmo Haar Cascade.
`http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html`

[17] Aplicación online para el diseño de los diagramas de secuencia y UML.
`https://drive.draw.io`

[18] Página con información sobre herramientas para la comunicación entre código Python y otros lenguajes.
`https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages`

[19] Documentación sobre PyBindGen.
`http://pybindgen.readthedocs.org/en/latest/`

[20] Documentación sobre Cython.
`http://cython.org/#documentation`
`http://docs.cython.org/`

[21] Documentación sobre Boost.
`https://wiki.python.org/moin/boost.python`
`http://www.boost.org/doc/libs/1_58_0/libs/python/doc/tutorial/doc/html/index.html`

[22] Documentación sobre el paquete Distutils.
`https://docs.python.org/2/distutils/`

[23] Documentación sobre el paquete Subprocess.
`https://docs.python.org/2/library/subprocess.html`

[24] Código sobre el método que convierte una lista de Python en un objeto Mat de OpenCV-C++.
`https://github.com/yati-sagade/opencv-ndarray-conversion`

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jun 25 23:19:20 CEST 2015
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)