Reconocimiento Facial con DLib y OpenCV

Curso 2k18/2k19

Apellidos, nombre	Javier Riera Chirivella (jariechi@inf.upv.es) Francisco Mayol Alonso (framaal5@inf.upv.es)
Titulación	Grado de Ingeniería informática
Fecha	Abril 2019

Índice

Índice de figuras

Índice de imágenes

Índice de códigos

1 Resumen de las ideas clave

En este documento vamos a explicar los pasos que hemos seguido para crear una aplicación capaz de utilizar el reconocimiento facial integrado en la librería de C++ DLib y aplicarlo a un proyecto de OpenCV.

Con esto queremos demostrar que no es necesario arrinconarse en solo una decisión, y que con suficiente trabajo e investigación se pueden superar las limitaciones de un sistema, aunque a veces se requiera pensamiento lateral.

2 Introducción

La idea inicial de este proyecto era, primero, ser capaces de utilizar el reconocimiento facial de DLib en alguna forma sustancial, y ver resultados que fuesen interesantes.

Una vez eso estuviese hecho, la idea era aplicarlo a un proyecto ya existente de OpenCV, y el proyecto de ejemplo GeoFace servía muy bien debido a que es literalmente la reproducción virtual de una cara.

Con estos dos elementos, lo que quedaba por decidir era en qué forma iban a interactuar, y una buena idea era hacer que la máscara de GeoFace se moviese en relación a la cara que se reconoce con DLib.

Dado que somos dos estudiantes que hemos trabajado arduo en este proyecto, y que este documento va dirigido a otros estudiantes, intentaremos no hacerlo todo demasiado pesado. Queremos que no cometáis los mismos errores que nosotros y que aprendáis de las cosas en que nosotros nos equivocamos, así como aquellas que hicimos bien. No esperéis un vocabulario técnico ni, sinceramente, demasiado serio en muchas ocasiones. Lo importante es que no os muráis de aburrimiento mientras intentáis aprender a hacer cosas chulas con OpenCV y DLib… Adelante, pues.

3 Objetivos

Lo mejor para mantenernos organizados es hacer una lista de las cosas que necesitamos hacer para cumplir nuestro objetivo final, e ir cumpliéndolos poco a poco conforme los vayamos solventando. He aquí la lista:

1. Instalar DLib.
2. Instalar OpenCV.
3. Instalar OpenGL.
4. Conseguir que GeoFace funcione.
5. Hacer que el reconocimiento facial funcione.
6. Conseguir que el script de estimación del ángulo de la cara funcione.
7. Conseguir una estimación de ángulo continua, no solo en una foto.
8. Establecer qué datos se pueden obtener del programa de estimación de ángulo facial para mover GeoFace.
9. Establecer una manera de intercambiar datos entre ambos programas.

4 Instalación y Errores cometidos

No vamos a mentir: Ha sido un viaje movidito. Nos hemos topado con gran cantidad de baches e imprevistos que no pensábamos que nos llevarían tanto tiempo de solucionar como al final resultó ser. Pero si algo bueno podemos sacar de esto, es que hemos aprendido a lidiar con este tipo de situaciones (O al menos eso esperamos).

De lo primero que nos dimos cuenta era que tener un **conocimiento previo de Linux** nos habría ahorrado muchos quebraderos de cabeza. Casi cualquier proyecto es más sencillo de programar en Linux… Si sabes cómo funciona este.

Siguiendo los tutoriales de internet[1] que nos facilitó el profesor, intentamos instalar en un portátil la librería DLib con sus bindeos en Python, incluyendo los entornos virtuales que se recomiendan para usar Python. Por desgracia, esto no funcionó y supuso bastante tiempo perdido simplemente intentando conseguir una instalación de Python funcional.

Decidimos entonces descartar Python… Mala idea, creednos.

Tras una breve incursión en el mundo de usar DLib sin Python, nos deshicimos de la idea e intentamos utilizar Windows, dado que es una plataforma con la que estamos más familiarizados. Para esto teníamos que configurar un compilador de C++ como MinGW o cygWin, pero fuimos incapaces porque necesitábamos hacer uso de CMake para que esto funcionase, así que volvimos con el rabo entre las piernas a Linux, pero esta vez decidimos hacerlo sobre una instalación limpia de Ubuntu 18.04 LTS.

Esta vez sí lo conseguimos. Fuimos capaces de instalar Tanto OpenCV como DLib a base de prueba y error y entender qué se nos estaba pidiendo tener instalado en lugar de seguir los tutoriales de internet sin ton ni son. Las cosas importantes para instalarlo todo son:

- Tener GCC instalado
- CMake instalado y configurado
- Python3 instalado
- Pip de Python3 instalado
- VirtualEnv y VirtualEnvWrapper de Python3 instalados y configurados

Los pasos a seguir para esto son:

```
sudo apt install gcc

sudo apt install cmake

sudo apt install python3

sudo apt install python3-pip

sudo apt-get install build-essential cmake pkg-config

sudo apt-get install libx11-dev libatlas-base-dev
```

[1] https://www.pyimagesearch.com/2017/03/27/how-to-install-dlib/

```
sudo apt-get install python3-dev

sudo pip3 install virtualenv virtualenvwrapper
```

Esto nos servirá para instalar todo lo que haya que instalar en Linux, pero ahora es cuando entramos en los entornos virtuales. Un entorno virtual de Python sirve para poder instalar en cada entorno únicamente lo que se necesita y en la versión que se necesita sin preocuparte por afectar a otros proyectos que puedan compartir el mismo ordenador. Así que lo primero que tenemos que hacer es crear nuestro propio entorno virtual utilizando virtualenv y virtualenvwrapper, tal que así:

```
mkvirtualenv nombreDelEntorno Python3

workon nombreDelEntorno
```

Así ya lo hemos creado y estamos dentro de él, por lo que podemos empezar a instalar cosas en él. Es aquí donde vamos a instalar DLib. Necesitamos varias librerías de Python para poder hacer ciertos cálculos matemáticos, así que ahora nos toca instalarlas:

```
pip install numpy scipy matplotlib scikit-image scikit-learn ipython

pip install dlib
```

Y ahora una serie de librerías generales que no son de Python para poder tratar con los archivos de imagen y otras cosas complementarias que se recomiendan por internet, además de OpenGL:

```
sudo apt-get install libjpeg-dev libpng-dev libtiff-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
libv4l-dev

sudo apt-get install libxvidcore-dev libx264-dev

sudo apt-get install libgtk-3-dev

sudo apt-get install libatlas-base-dev gfortran
```

Con todo eso ya instalado toca ponerse manos a la obra con OpenCV, que es posiblemente una de las cosas que más trabajo da. Tras mucho buscar y probar, al final esto es lo que nos dio resultado:

```
cd ~

wget -0 opencv.zip
https://github.com/opencv/opencv/archive/4.0.0.zip

wget -O opencv.zip
https://github.com/opencv/opencv/archive/4.0.0.zip

wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.0.0.zip

unzip opencv.zip

unzip opencv_contrib.zip

mv opencv-4.0.0 opencv

mv opencv_contrib-4.0.0 opencv_contrib

cd opencv

mkdir build
```

```
cd build/

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local
-D INSTALL_PYTHON_EXAMPLES=ON -D INSTALL_C_EXAMPLES=OFF -D
OPENCV_ENABLE_NONFREE=ON -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules -D
PYTHON_EXECUTABLE=~/.virtualenvs/cv/bin/python -D BUILD_EXAMPLES=ON
..

make -j4

sudo make install

sudo ldconfig
```

Esto instala y configura en CMake todo lo necesario para utilizar OpenCV, así que nos hemos quitado de encima los objetivos del 1 al 3 de nuestra lista. Y con esto y un bizcocho…

A programar.

5 Proyectos Intermedios

Está claro que, aunque lo tengamos todo instalado ya, no podemos lanzarnos al vacío sin dar pequeños pasos. Así que en este apartado vamos a intentar cumplir nuestro objetivo número 5, y haremos un total de tres proyectos que luego modificaremos para conseguir el final.
Lo primero y posiblemente más sencillo es hacer que GeoFace funcione. Tenemos que tener en cuenta, y esto es muy importante, que **el programa GeoFace viene con errores en la página de demos** de OpenGL. Esto es algo que se aborda en las prácticas de la asignatura, por lo que aquí no lo vamos a explicar, pero es mejor tenerlo en cuenta y no cometer el error que tuvimos nosotros de usar el proyecto sin arreglar y tirarnos de los pelos por no encontrar el error en nuestros cambios o en la compilación.
Sin más demora, lo que hace falta para compilar el programa es descargarlo de la web de Demos de OpenGL[2].
Una vez esté descargado y descomprimido en una carpeta que queramos, y tras solucionar el error que se nos presenta en la práctica, tenemos que abrir un terminal en esa dirección y ejecutar el siguiente comando:

```
gcc display.c fileio.c main.c make_face.c muscle.c -o main -lGLU
-lGL -lglut -lm
```

Podemos sustituir "main" por el nombre que queramos darle a nuestro programa. Acto seguido, lo que queda por hacer es ejecutar nuestro archivo ejecutable recién creado, para ello tan solo hay que hacer:

```
./main
```

Y tachán, con esto ya deberíamos tener un proyecto funcional de GeoFace, que debería quedar como esto:

[2] https://www.opengl.org/archives/resources/code/samples/glut_examples/demos/demos.html

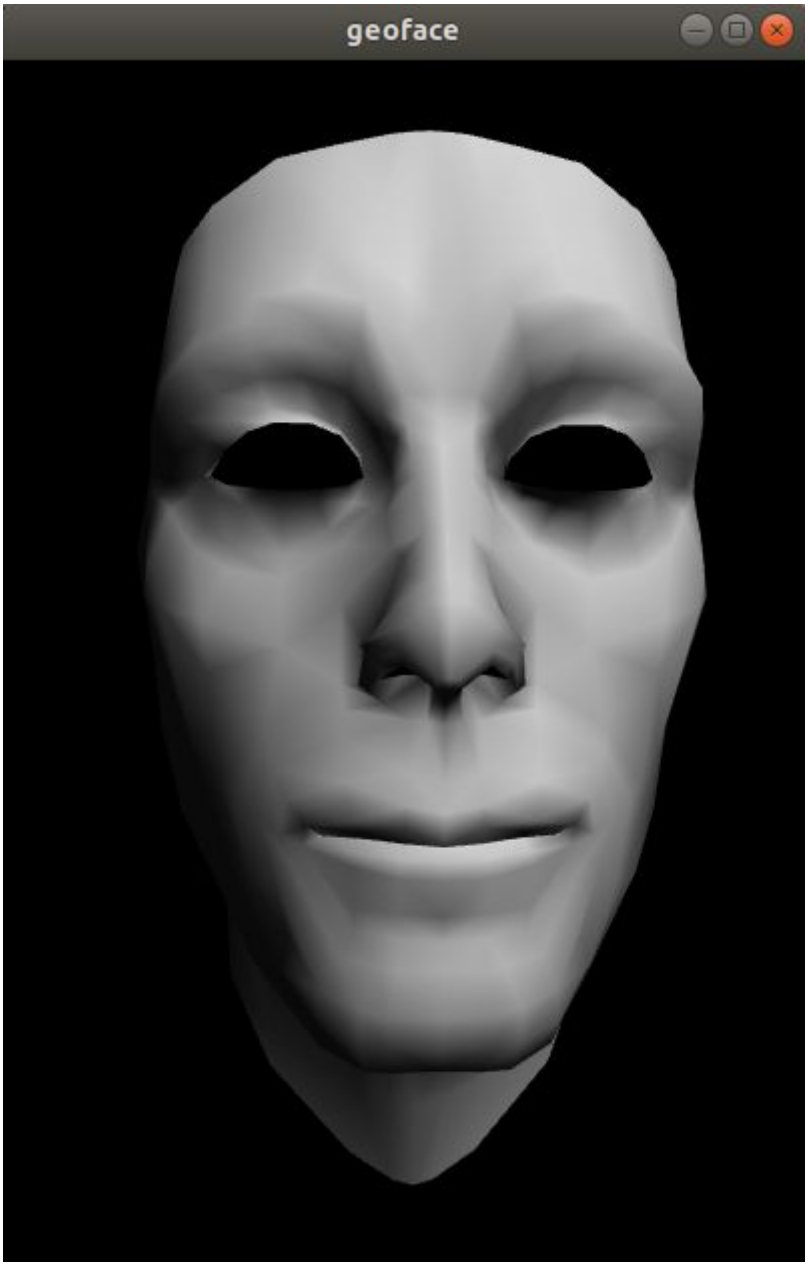

Imagen 1: GeoFace en funcionamiento.

Lo siguiente es realizar la identificación de puntos faciales con DLib y OpenCV, siguiendo el tutorial de Adrian Rosebrock[3]. Hay que suscribirse a su boletín de información para conseguir el código completo, así como el archivo ya entrenado de puntos faciales que se va a utilizar. Una vez esté descargado poco más queda que ejecutarlo desde nuestro entorno virtual de Python y ver que, en efecto, funciona con los 68 puntos:

[3] https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

Imagen 2: Mi preciosa cara en bata siendo identificada por el programa.

¡Perfecto! Todo va viento en popa. Pero para identificar el ángulo en que estamos mirando no hacen falta 68 puntos, sino apenas 6, como se explica en el tutorial de Satya Mallick[4].

Así pues, siguiendo su código y haciendo cierto batiburrillo con el archivo de entrenamiento del programa anterior, escogiendo solo aquellos que necesitamos, podemos ejecutar un programa de ejemplo en el que se detecten los extremos de los ojos, la nariz, las comisuras de los labios y la barbilla, y a partir de ahí se obtenga una estimación de hacia dónde estamos mirando.

Si queréis una explicación en profundidad sobre cómo se calcula esto a nivel matemático, en el tutorial que acabamos de mencionar viene una maravillosa.

[4] https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/

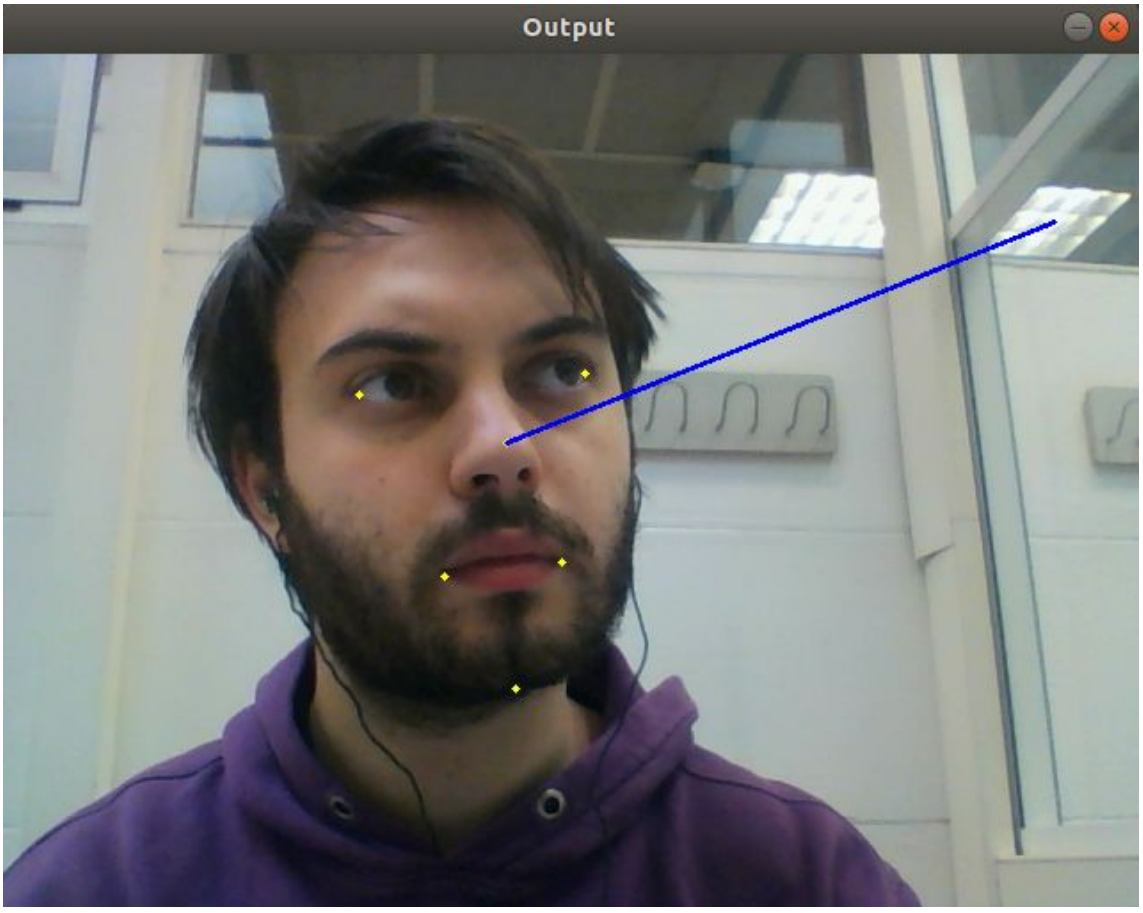

Imagen 3: estimación de ángulo facial funcionando.

Y con esto ya lo tenemos todo preparado para empezar a programar nosotros, y hemos conseguido cumplir los objetivos 4, 5 y 6.

6 Modificaciones para el Proyecto Final

Está muy bien que nos funcione todo lo anterior, pero por separado nos sirve de poco. Además, los programas que usan imágenes nuestras no funcionan con la webcam sino con fotos ya guardadas en memoria. Es por eso que tenemos que modificarlos para que podamos usarlos en tiempo real.

No es demasiada complicación, la verdad. En realidad se parece mucho a lo que se hace con las fotos ya guardadas pero metiéndolo en un bucle, tal que así (Código 1):

```
#Aquí iniciamos el capturador de vídeo de la webcam
vs = VideoStream(src=0).start()
#Este es el loop principal
while True:
    #Capturamos el fotograma actual
    frame = vs.read()
```

```
#Hacemos las operaciones necesarias
```

Si movemos todas las operaciones que se hacen con una foto a dentro del bucle, seremos capaces de obtener un feed en tiempo real rápidamente, y así habremos cumplido el objetivo 7.

Una vez eso lo tengamos, hay que ver qué datos se le pueden pasar al programa de GeoFace para cambiar la orientación de la cara, y para ello hay que entender cómo modificar la orientación de la cara.

Inspeccionando el código de GeoFace hemos podido comprobar que lo que más nos interesa modificar son los atributos spinxface y spinyface, que se modifican según cómo y cuánto arrastremos el ratón. De esta manera, estableceremos un sistema en que, según cuánta distancia haya del punto de inicio a punto de fin en nuestra línea de indicación de orientación, podremos deducir en qué ángulo está mirando. Hemos creado una pequeña proporción para conseguirlo, que es:

$$\Delta x/1000 * 90 = anguloX$$

$$\Delta y/1000 * 90 = anguloY$$

Así podemos simular un arrastre de ratón con los datos de la cámara. Ale, objetivo 8 solucionado.

Vamos a por lo último: cómo compartir estos datos. Dado que un programa está en Python y el otro en C, no hemos sido capaces de ejecutarlos al unísono con solo un comando, ni hemos podido establecer una pipeline que intercambie datos. Por esta razón hemos recurrido a un método que, aunque poco elegante, cumple su cometido y nos sirve para crear una muestra de concepto. Vamos a crear un archivo .txt en el que el detector facial esté continuamente escribiendo el delta de x e y y sustituyéndolo por los datos más recientes, mientras que GeoFace lo irá leyendo y actualizando la rotación de la máscara continuamente.

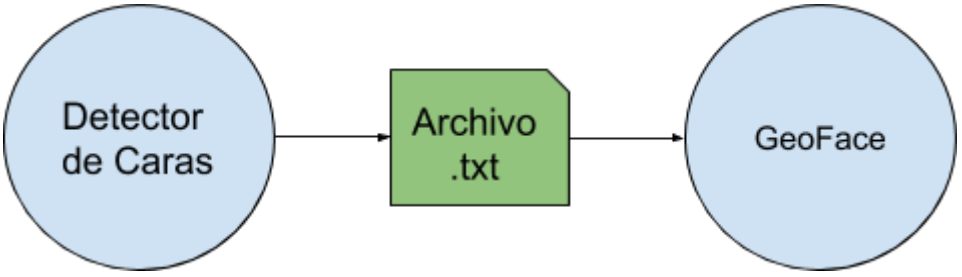

Figura 1: Esquema de intercambio de datos

No ha hecho falta modificar demasiado, simplemente entender cómo funcionan las librerías de lectura y escritura de archivos en Python y C.

Una vez hecho esto, solo queda ejecutar ambos programas de forma paralela y comprobar cómo funcionan en conjunto.

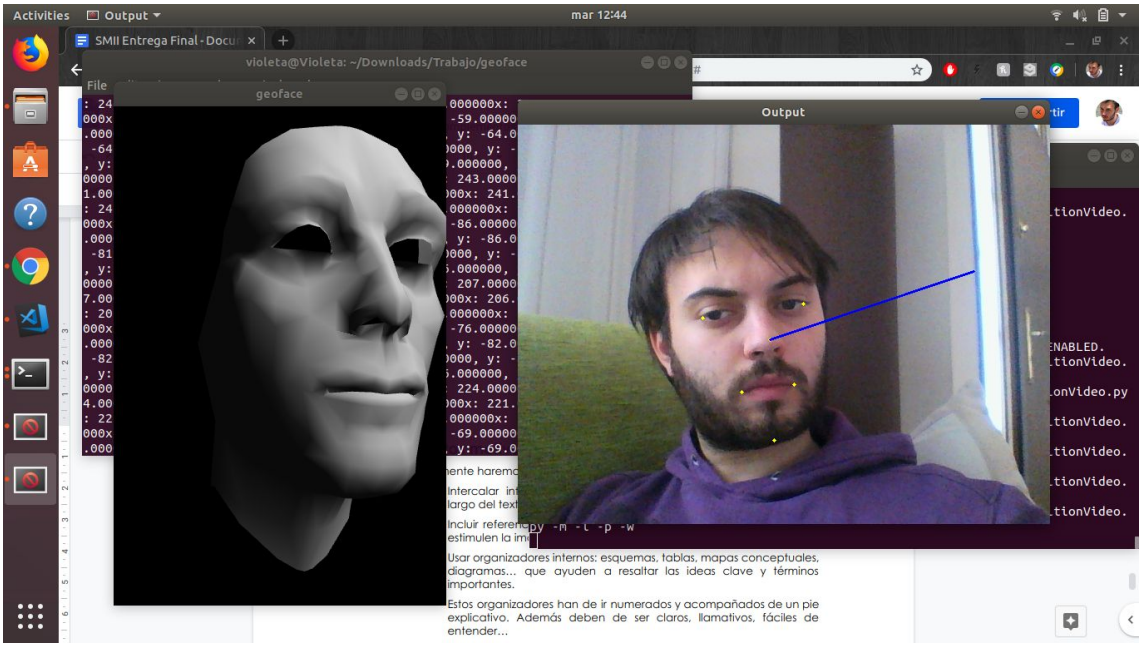

Imagen 4: Ambos programas funcionando juntos.

7 Aspectos a mejorar

No todos los proyectos son perfectos, y este no iba a ser la excepción. Hay muchas cosas que nos habría gustado hacer de forma diferente, y que podrían haber salido mejor, así que vamos a intentar hacer una lista de esos detalles que podrían mejorarse.

- **Rotación en el eje z:** La cara de GeoFace rota en dos ejes, el x e y, pero la cara detectada en el otro programa puede rotar en el eje z sin que se vea afectado en el resultado final. Quizá crear una referencia con la unión de los ojos o similar sería una buena idea para poder poner ese eje en uso y que la traducción de movimientos fuese más 1:1.
- **Comunicación ineficiente:** Ya lo hemos comentado, un documento de texto es simplemente una excusa de comunicación. Aprender a utilizar sockets o pipelines sería preferible en este tipo de proyectos, o…
- **Usar un único programa:** Esto puede que hiciese las cosas más difíciles a corto plazo, pero con vistas al futuro es insostenible usar dos programas para un único uso, o al menos tener que ejecutarlos desde dos terminales diferentes.
- **Detección poco estable:** Hay muchas ocasiones en las que se pierde la orientación de la cara o simplemente no se identifica de forma correcta, haciendo que la máscara de GeoFace se mueva de forma casi errática. Quizá utilizar más de 6 puntos solucionase ese problema, pero puede que también cargase más la ejecución. De todas formas, es algo que habría que investigar.

8 Conclusión

Este proyecto ha sido una gran experiencia de auto-educación y resolución de problemas. Nos hemos enfrentado a tecnologías que hasta este año no habíamos estudiado y nos hemos tenido que ver las caras con problemas de instalación en sistemas operativos que no conocíamos demasiado.

Aunque, sobre todo, lo más importante ha sido la experimentación y la realización que ha supuesto. Cada vez que lográbamos un pequeño avance, aunque fuese siguiendo un tutorial, nos sentíamos orgullosos de haber conseguido que el ordenador pudiese identificar nuestras caras y pintar unos cuantos puntos encima.

Ha resultado ser un proyecto que, aunque desafiante, nos ha servido para aprender y aplicar muchos ámbitos nuevos. Entre ellos, lo más importante podría ser tener que programar en Python, que es un lenguaje que está cada vez más en uso y es un estándar de la industria, así que aprender aunque sea los conceptos básicos ha sido un gran paso.

Queremos agradecerte que hayas leído hasta el final y esperamos que hayas aprendido algo de nuestra experiencia, y que si te decides a ir por el mismo camino que nosotros, al menos no cometas los mismos errores. ¡Suerte!

9 Bibliografía

Cómo Instalar DLib:
https://www.pyimagesearch.com/2017/03/27/how-to-install-dlib/
Instalación de VirtualEnv en Linux:
https://stackoverflow.com/questions/13855463/bash-mkvirtualenv-command-not-found
Instalar OpenCV 4 en Ubuntu:
https://www.pyimagesearch.com/2018/08/15/how-to-install-opencv-4-on-ubuntu/
Instalar OpenCV 3 y DLib en Windows:
https://www.learnopencv.com/install-opencv-3-and-dlib-on-windows-python-only/
Instalar DLib y ejemplo de reconocimiento facial:
https://www.pyimagesearch.com/2018/01/22/install-dlib-easy-complete-guide/
Estimación del ángulo facial:
https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/