Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Manejo mediante gestos faciales de una aplicación para la programación en Java para personas con diversidad funcional

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Aparicio Alonso, Eduardo

Tutor: Martínez Hinarejos, Carlos David

2020-2021

Resumen

La programación en computadoras requiere generalmente del uso de pantallas, teclados o ratones para ver el código implementado o para ingresar nuevo código. Sin embargo, estos dispositivos a menudo no son adecuados ni cómodos para personas con diversidad funcional. Por ello, en este trabajo se va a realizar el desarrollo del reconocimiento de gestos faciales para el manejo de COPS, entorno de desarrollo software en Eclipse que facilita la programación a usuarios con diversidad funcional. Esto se conseguirá mediante la localización de los puntos más significativos de la cara de una persona (puntos en los ojos, cejas, contorno de la mandíbula, nariz, boca, etc.) con el objetivo de, a través del movimiento y las distancias entre estos, entrenar modelos correspondientes a cada gesto facial. Estos modelos servirán para posteriormente reconocer qué gesto es el que el usuario está realizando a través de la cámara y, de la misma manera, cuál es la funcionalidad de la aplicación que se desea poner en marcha. Con esto se conseguirá el objetivo de que cualquier usuario, sin exclusiones, pueda disfrutar de este gran mundo de la programación sin ningún tipo de dificultad ni inconveniente.

Palabras clave: reconocimiento, gestos faciales, COPS, Eclipse, diversidad funcional.

Abstract

Computer programming generally requires the use of screens, keyboards or mice to view the implemented code or to enter new code. However, these devices are often not suitable or comfortable for people with functional diversity. For this reason, in this work, the development of facial gesture recognition for the management of COPS, a software development environment in Eclipse which facilitates programming for users with functional diversity, will be carried out. This will be achieved by locating the most significant points on human's face (points on the eyes, eyebrows, jaw, nose, mouth, etc.) with the aim of, with the movement and the distances between these train models corresponding to each facial gesture. These models will serve to recognize which gesture the user is performing through the camera and, in the same way, which is the function of the application to be launched. With this, the main goal will be achieved, which is that any user, without exclusions, can enjoy this great world of programming without any kind of difficulty or inconvenience.

Keywords: recognition, facial gesture, COPS, Eclipse, functional diversity.

Resum

La programació en computadores requereix generalment de l'ús de pantalles, teclats o ratolins per a veure el codi implementat o per a ingressar nou codi. No obstant això, aquests dispositius sovint no són adequats ni còmodes per a persones amb diversitat funcional. Per això, en aquest treball es realitzarà el desenvolupament del reconeixement de gestos facials per al maneig de COPS, entorn de desenvolupament de programari en Eclipse que facilita la programació a usuaris amb diversitat funcional. Això s'aconseguirà mitjançant la localització dels punts més significatius de la cara d'una persona (punts en els ulls, celles, contorn de la mandíbula, nas, boca, etc.) amb l'objectiu de, a través del moviment i les distàncies entre aquests, entrenar models corresponents a cada gest facial. Aquests models serviran per a posteriorment reconéixer quin gest és el que l'usuari està realitzant a través de la càmera i, de la mateixa manera, quina és la funcionalitat de l'aplicació que es desitja posar en marxa. Amb això, s'aconseguirà l'objectiu de que qualsevol usuari, sense exclusions, puga gaudir d'aquest gran món de la programació sense cap mena de dificultat ni inconvenient.

Paraules clau: reconeixement, gestos facials, COPS, Eclipse, diversitat funcional.

Agradecimientos

Han sido varias las personas que, ya sea académica o moralmente, me han proporcionado su apoyo en la realización de este trabajo y, como es debido, he de expresarles mi gratitud.

En primer lugar, me gustaría agradecer a mi tutor Carlos David Martínez Hinarejos por todo su esfuerzo y dedicación durante estos meses para ayudarme a realizar un buen proyecto, por estar siempre disponible para resolver todas mis dudas y por aportar su gran experiencia en el tema que trata este trabajo.

En segundo lugar, quería también agradecer a todas las personas que, sin ser de mi familia más cercana, me han ayudado y motivado durante este periodo. Ellos son todos mis amigos, tanto los amigos que he tenido la suerte de conocer durante estos años en Valencia como los amigos de mi pueblo natal. Quería darles las gracias por estar siempre que se les ha necesitado.

Y, por último, no podía olvidarme de mi familia, mis padres, mi hermana, tíos, primos y abuelos, que son los que siempre están tanto en lo bueno como en lo malo y son los que me han enseñado la importancia de creer en uno mismo y de resolver todo lo que te propongas con humildad y sencillez. Quería dedicarles este trabajo a todos ellos.

Abreviaturas

LC – Levantar cejas

CO – Cerrar ojos

GI – Guiño izquierdo

GD – Guiño derecho

NO – Normal

HMM - *Hidden Markov Models*

HCI – *Human-Computer Interaction*

AU – *Action Unit*

SVM - *Support Vector Machines*

Índice

Índice de figuras

Índice de tablas

1. Introducción

Eclipse[1] es un entorno de desarrollo software de código abierto con la posibilidad de ser extendido indefinidamente mediante *plugins*. De hecho, se consideró desde un principio como una plataforma de integración de herramientas de desarrollo. Al tratarse de un entorno de desarrollo integrado genérico, da soporte a una gran cantidad de lenguajes. Sin embargo, el más habitual de todos ellos es Java, el cual se apoya en el plugin JDT[2], incluido en la distribución estándar de la herramienta. Este entorno cuenta con una comunidad de usuarios grande e implicada, que extiende constantemente las funcionalidades con nuevos *plugins*. Algunos de estos *plugins* son JDT (ya comentado), EGit[3] (encargado de integrar GIT en Eclipse) y ADT[4] (encargado del desarrollo de aplicaciones Android).

COPS [1] (Computer Programming using Speech) es uno de estos *plugins* para Eclipse, encargado de facilitar a usuarios con diversidad funcional su experiencia programando. Algunas de sus funcionalidades más destacables son el reconocimiento por voz, que permite al programador dictar el código que quiere escribir, y la síntesis por voz, que por otro lado permite a éste escuchar el fragmento de código seleccionado.

El funcionamiento de este *plugin* es sencillo. Cuando abrimos Eclipse, al seleccionar la vista de COPS, se va a mostrar el código del fichero que se haya abierto en el área de edición de texto habitual del IDE de Eclipse clonado en una nueva pestaña en la parte inferior, de manera que ambas áreas estarán coordinadas para que cualquier modificación en una de ellas venga reflejada también en la otra. Esta pestaña inferior cuenta con la configuración inicial de un fondo negro con la fuente en color amarillo para conseguir un buen contraste (Figura 1), aunque es posible modificar esta configuración desde las preferencias de vista (Figura 2).

COPS cuenta con cuatro funcionalidades principales, las cuales se podrán poner en marcha interactuando con los cuatro iconos correspondientes (que aparecen en la esquina superior derecha de la pestaña de vista de COPS en la Figura 1). El primer y segundo icono se corresponden con el aumento y decremento del tamaño de la fuente, respectivamente. Con el tercer icono, al hacer clic una vez sobre él se iniciará una grabación de voz, la cual se parará con un segundo clic sobre el mismo y, de esta manera, con el audio guardado, se escribe el código dictado. Por último, con el cuarto icono, tras seleccionar un fragmento de texto y pulsarlo, se procede a escuchar el código seleccionado. El proceso de síntesis se apoya en FreeTTS[5], mientras que el dictado por voz se trata de una versión modificada del sistema iATROS [2], desarrollado de forma modular con un motor de reconocimiento central y varias funciones de utilidad que se podrán usar en la construcción de aplicaciones basadas en voz y escritura manuscrita, además de aplicaciones multimodales e interactivas.

[1] https://www.eclipse.org
[2] https://www.eclipse.org/jdt/overview.php
[3] https://projects.eclipse.org/projects/technology.egit
[4] https://stuff.mit.edu/afs/sipb/project/android/docs/tools/sdk/eclipse-adt.html
[5] https://freetts.sourceforge.io/

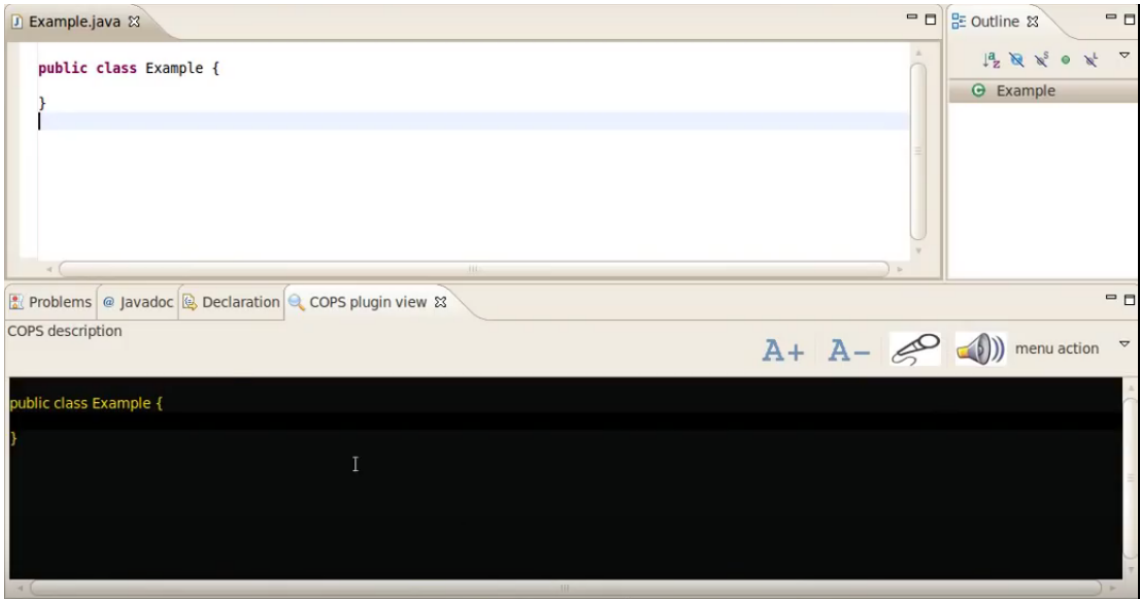

Figura 1. Vista general de COPS.

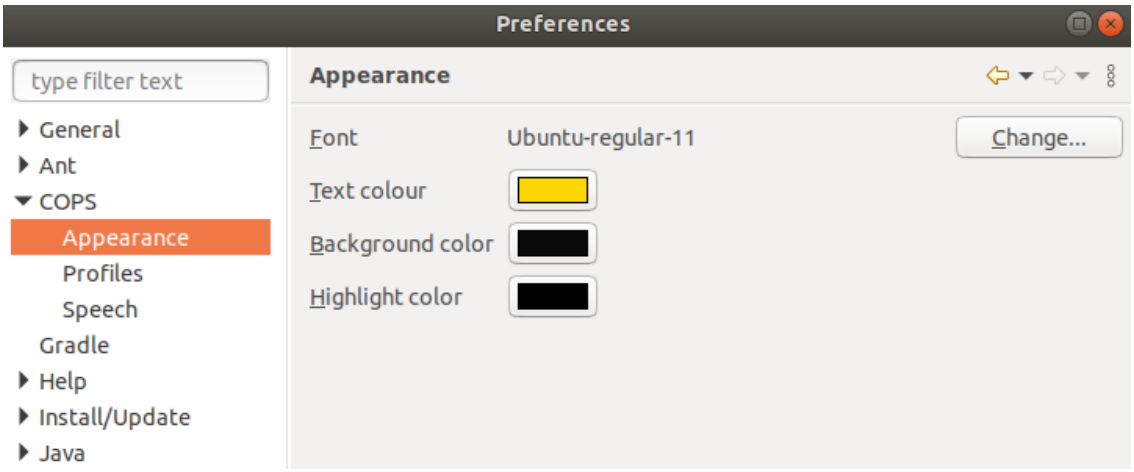

Figura 2. Preferencias de la vista de COPS.

Sabiendo esto, es aquí donde entra el papel de este proyecto: permitir que estas cuatro funcionalidades principales sean más accesibles para los usuarios a los que va dirigida la aplicación. La idea es que, mediante gestos realizados con la cara, cada una de estas funcionalidades puedan activarse y desactivarse, facilitando así el uso de las mismas. Se verá en los siguientes apartados cómo se ha conseguido este objetivo.

1.1. Motivación

El motivo de elección de este trabajo ha sido el hecho de pensar en la gran importancia que tiene actualmente el desarrollo software en nuestras vidas y lo imprescindible que será en el futuro, y en las barreras y dificultades con las que personas con diversidad funcional pueden encontrarse a la hora de llevarlo a cabo. Hoy en día son cada vez más los usuarios interesados en este ámbito, y es por ello por lo que es importante no excluir a aquellos con alguna dificultad. Mediante este proyecto se ofrece a estas personas alternativas adaptadas para poder acceder y disfrutar de este gran mundo que es la programación con las mejores condiciones posibles.

1.2. Objetivos

El primer objetivo de este proyecto es conseguir el correcto reconocimiento mediante cámara de cada uno de los gestos faciales elegidos, a través del uso de las librerías `hmmlearn`[6], `opencv`[7] y `dlib`[8]. Una vez conseguido esto, el objetivo principal es ampliar la funcionalidad del *plugin*, integrando este reconocimiento en el código del *plugin* a través del uso de *sockets* que permitan comunicar ambos procesos mediante mensajes. De esta manera, con un guiño de su ojo izquierdo o derecho el usuario logrará disminuir o aumentar el tamaño de la fuente, respectivamente; con el cierre de sus ojos durante un determinado tiempo, el usuario podrá activar la síntesis de voz; y, por último, levantando las cejas se logrará activar y desactivar el dictado de texto por voz.

1.3. Estructura de la memoria

Respecto a la estructura sobre la que se apoya esta memoria, se comenzará en el capítulo 2 explicando el estado del arte, con la introducción de las técnicas más utilizadas en el reconocimiento de gestos faciales hasta ahora y las alternativas a COPS que se pueden encontrar hoy en día en el mercado. En el capítulo 3 se va a introducir el desarrollo del sistema de reconocimiento de gestos faciales y se va a mostrar paso a paso cómo se ha conseguido. Además, en este capítulo veremos las tecnologías y herramientas que se han utilizado para el desarrollo en cada una de las fases, así como una evaluación final del sistema que nos mostrará su tasa de acierto. Tras esto, en el capítulo 4 se verá cómo se ha incluido este reconocimiento al código de la aplicación COPS en Eclipse y cómo para cada uno de los gestos se podrá activar una funcionalidad de ésta. Posteriormente, se comentarán en el capítulo 5 las conclusiones que se han podido obtener del proyecto, así como la relación de los contenidos con los estudios cursados en el grado. Para finalizar, se hablará en el capítulo 6 acerca de posibles trabajos futuros y extensiones a realizar sobre COPS.

[6] https://hmmlearn.readthedocs.io/en/latest/tutorial.html
[7] https://docs.opencv.org/master/
[8] http://dlib.net

2. Estado del arte

Antes de abordar el sistema de reconocimiento desarrollado en este proyecto, en este capítulo se hará una breve introducción acerca del contexto actual en el que se encuentra el campo de la visión por computador en relación con el reconocimiento de gestos y su análisis. Para ello, se adoptará una visión general de los distintos algoritmos, técnicas, herramientas y fundamentos teóricos utilizados actualmente para dicho propósito.

Se comenzará con una breve introducción al *machine learning*, para posteriormente hablar de la situación actual en los sistemas de ayuda a la programación para discapacitados, donde se verán algunas alternativas al *plugin* COPS. Con esto, se comprenderá mejor el entorno en el que se encuentra este proyecto.

2.1. Introducción al *Machine Learning*

El aprendizaje automático[9] o *machine learning* es un campo de inteligencia artificial que, mediante el uso de técnicas estadísticas, permite que los sistemas informáticos puedan aprender, tomar decisiones o identificar patrones a partir de un conjunto de datos, sin ser programados explícitamente. Se dice que una computadora aprende cuando es capaz de mejorar sus resultados o disminuir su error por medio de la experiencia. Los investigadores del *machine learning* buscan algoritmos que conviertan muestras de datos en programas de computadora, algoritmos que serán capaces de desarrollar nuevos algoritmos. Los modelos resultantes deben poder generalizar comportamientos, es decir, predecir la salida o resultado para un conjunto más amplio de datos.

En este campo se distinguen principalmente dos tipos de aprendizaje en función de la naturaleza del conjunto de entrenamiento, el supervisado y el no supervisado.

2.1.1. Aprendizaje supervisado

El algoritmo de aprendizaje supervisado cuenta con información que le permite conocer qué conjuntos de datos son satisfactorios para el objetivo del aprendizaje, es decir, el conjunto de entrenamiento cuenta tanto con los datos de entrada como de salida. Su aprendizaje se basa en tratar de resolver una tarea, cotejarla con la solución correcta y, de esta forma, reajustar sus parámetros para disminuir el error. Un posible ejemplo sería el proyecto actual. Como se verá a continuación, para el aprendizaje de los modelos se han realizado videos con un gesto determinado, especificando de qué gesto se trata en cada video.

[9] https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

2.1.2. Aprendizaje no supervisado

Éste, por el contrario, no cuenta con la información que le permite conocer si un conjunto de datos es satisfactorio. Su objetivo principal es analizar patrones que separen y clasifiquen los datos en diferentes grupos, en función de sus atributos. Para dar utilidad a la información obtenida por el algoritmo de aprendizaje no supervisado, ésta será interpretada por una persona posteriormente. Si en este proyecto no se especificase para cada video qué gesto es el que se realiza, el aprendizaje sería no supervisado.

2.2. Introducción al reconocimiento de gestos

El reconocimiento de gestos es un área en informática y un problema de reconocimiento de formas con el objetivo de interpretar gestos humanos mediante algoritmos matemáticos. Los gestos pueden realizarse con cualquier movimiento o estado del cuerpo, pero normalmente se originan en la cara o en la mano.

Uno de los primeros trabajos aceptados por la comunidad en incorporar gestos en HCI (*Human-Computer Interaction*) fue [3], donde Sutherland presentó un sistema llamado Sketchpad, que permitía manejar un marcador tipo bolígrafo para manipular objetos gráficos en un dispositivo tipo tableta. De esta manera, ser humano y computadora conversarían a través de dibujos lineales.

Sin embargo, en este proyecto se hará más hincapié, como es obvio y debido a los requisitos, en aquellos gestos originados en la cara.

Hoy en día, el reconocimiento de expresiones faciales tiene numerosas aplicaciones en ámbitos como la interacción humano-computador (HCI), juegos de computadora interactivos o en la investigación psicológica (debido a los sentimientos o emociones que una persona puede expresar en base a los gestos de su cara). Es por esto por lo que el reconocimiento de expresiones faciales es un componente esencial en cualquier sistema diseñado para reconocer un lenguaje de gestos en tiempo real.

Un sistema de codificación de acciones faciales (FACS en inglés) muy conocido es el de Ekman y Friesen [4], comentado en [5]. En éste se definen las expresiones en términos de la presencia o ausencia de 44 movimientos musculares elementales, llamados unidades de acción (*action unit* - AU). Vemos algunos ejemplos en la Figura 3.

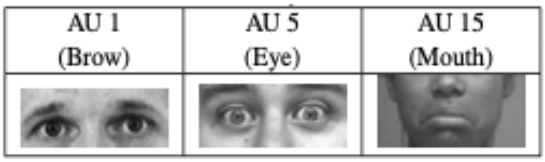

Figura 3. Ejemplos de AUs.

Actualmente, existen dos enfoques distintos para el reconocimiento automático de expresiones faciales. El primero de ellos es el "sistema de puntos característicos", en el cual se extrae la ubicación de varios puntos de referencia de la cara (ojos, cejas, boca...), y los vectores de características de éstos se calculan como alguna función de las posiciones y distancias entre puntos. El segundo es el "sistema basado en la apariencia", en el cual se extraen características a partir de imágenes integrales mediante las máscaras de Haar [6]. En estas imágenes integrales el objetivo se basa en considerar regiones rectangulares adyacentes en una ventana de detección y para cada región sumar la intensidad de los píxeles, con el propósito de calcular después la diferencia entre estas sumas. Esto permite clasificar distintas secciones de la imagen.

Otro de los enfoques más exitosos para el reconocimiento de expresiones es aplicar filtros de Gabor [7] para extraer características y luego usar máquinas de vectores soporte (*Support Vector Machines* - SVM) [8] para clasificarlas en AU (por ejemplo, en [9] y [10]). Por un lado, las SVM son un modelo propio del aprendizaje supervisado que pertenece a la familia de los clasificadores lineales desarrollado por Vladimir Vapnik y es capaz de resolver problemas de clasificación y regresión mediante un conjunto de datos de entrenamiento. Y por el otro, los filtros de Gabor son filtros lineales cuya respuesta de impulso es una función sinusoidal multiplicada por una función gaussiana. A pesar de que las tasas de reconocimiento son altas (más del 90%), este último enfoque es ineficiente en el uso de memoria y lento debido a la alta redundancia de la representación de Gabor.

El primero de los enfoques comentados es el que se ha llevado a cabo para el desarrollo de este proyecto, por lo que se irá comentando y mostrando más detalladamente en el capítulo 3. Sin embargo, el segundo de los enfoques se introducirá brevemente en el capítulo actual, en vistas a tener una idea general más detallada de su funcionamiento.

2.2.1. Sistema basado en la apariencia

El sistema basado en la apariencia, siguiendo la investigación descrita en [5], se basa en la combinación de las características de Haar y el algoritmo de Adaboost [11], y fue empleado por Paul Viola y Michael Jones en [12] para la detección de rostros. Esta combinación ha demostrado una alta precisión de reconocimiento y un rendimiento rápido en tiempo de ejecución. Las características de Haar, a diferencia de las de Gabor, se pueden extraer rápidamente sin una transformada de Fourier. Además, ya que el algoritmo de aprendizaje por refuerzo [13] lleva a cabo implícitamente la selección de características, el número de características de Haar extraídas será mucho menor que el número de características de Gabor, lo que permite ahorrar memoria. Para entenderlo mejor, se explicará a continuación más detalladamente las fases en las que se compone este sistema (siguiendo [6]). Éstas se pueden ver en la Figura 4 en forma de esquema.

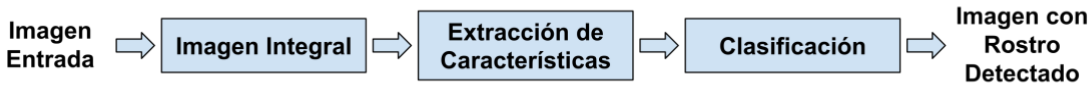

Figura 4. Fases del sistema basado en la apariencia.

2.2.1.1. Imagen integral

En la primera fase, la representación en imagen integral permite extraer rápidamente características a diferentes escalas debido a que no se trabaja con los valores de intensidad de forma directa, si no con una imagen acumulativa que se formará a partir de operaciones básicas. Una imagen integral, en la localización (x, y), contiene la suma de los píxeles de la parte superior izquierda de la imagen.

2.2.1.2. Extracción de características

En la segunda fase, extracción de características, es necesario comentar que el algoritmo, para entrenar al clasificador, necesita imágenes que contengan la cara buscada (positivas) y que no la contengan (negativas), y es para esto para lo que se utilizan las características de Haar.

Los filtros Haar son ventanas con dos regiones diferenciadas que se aplican sobre un área de detección de la imagen, con las cuales se computan valores definidos como características Haar. Cada una de estas características será un valor único obtenido tras restar la suma de píxeles en el rectángulo blanco a la del rectángulo negro, como se puede ver en la Figura 5. La idea tras esto es formar un modelo con las características de Haar del objeto a detectar.

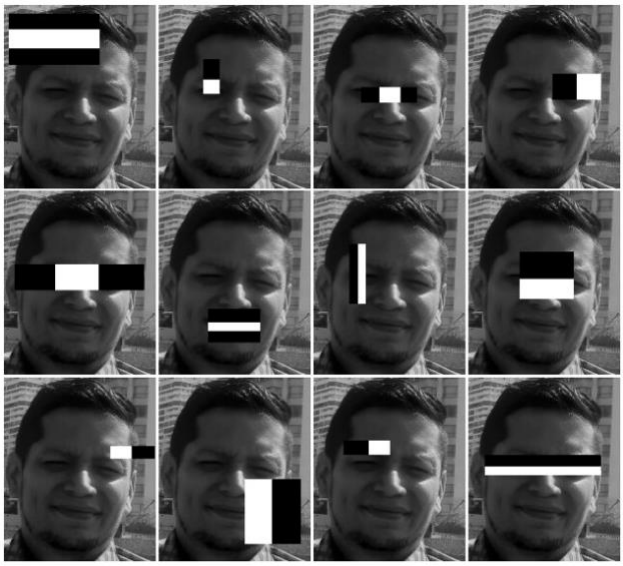

Figura 5. Filtros Haar sobre una misma imagen.

2.2.1.3. Clasificación

Por último, en la tercera fase se usó el concepto de cascada de clasificadores (*boosting*, introducido por [14]), el cual consiste en tomar clasificadores débiles o básicos, cada uno de ellos con una precisión ligeramente superior a una clasificación aleatoria, y combinarlos para construir uno fuerte con la precisión que se requiere. Se puede ver un esquema sencillo en la Figura 6.

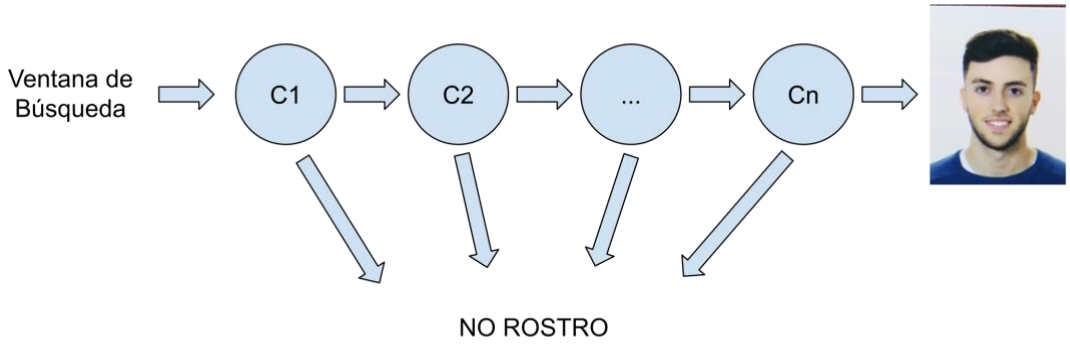

Figura 6. Clasificador en cascada.

Se debe seguir un proceso de entrenamiento supervisado para crear esta cascada de clasificadores y se conseguirá mediante un algoritmo basado en Adaboost. Éste último es un meta-algoritmo adaptativo de *machine learning* que resulta efectivo en la búsqueda de un pequeño número de buenas características que no tienen variación significante. De esta manera, se asegura una clasificación rápida, ya que se excluye a una gran cantidad de características insignificantes, y se centra en un pequeño grupo de características críticas, las cuales serán suficientes para percibir si la imagen o fragmento de imagen se corresponde con el objeto buscado.

2.3. Situación actual en sistemas de ayuda a la programación para personas con diversidad funcional

En este proyecto se propone y se desarrolla una buena solución para que usuarios con diversidad funcional puedan programar con las mayores facilidades posibles. Por ello, antes de comenzar con éste, se comentará cómo se encuentra en la actualidad este campo y qué avances o alternativas se pueden encontrar en él.

2.3.1. SpeechEclipse

Aunque actualmente SpeechEclipse [15] no está disponible, este *plugin* de Eclipse dispone de un sistema de reconocimiento de voz que permite la navegación por menús. Cuenta con una implementación de la API Java Speech sobre la herramienta de reconocimiento de voz de Windows y su gramática es definida con Java Speech Gramar Formal. Esta gramática será la encargada de definir qué entiende el reconocedor de voz e incluye una parte de dictado libre y otra basada en reglas para un funcionamiento más preciso. Con ello, el funcionamiento del *plugin* es bastante sencillo: comprueba si lo que dicta el usuario es un evento (por ejemplo, abrir un menú) y, si es así, pasa este evento a “Robot Class” para transformarlo en una orden de teclado y ejecutar la orden.

En la Figura 7 se representa más detalladamente en forma de esquema su funcionamiento.

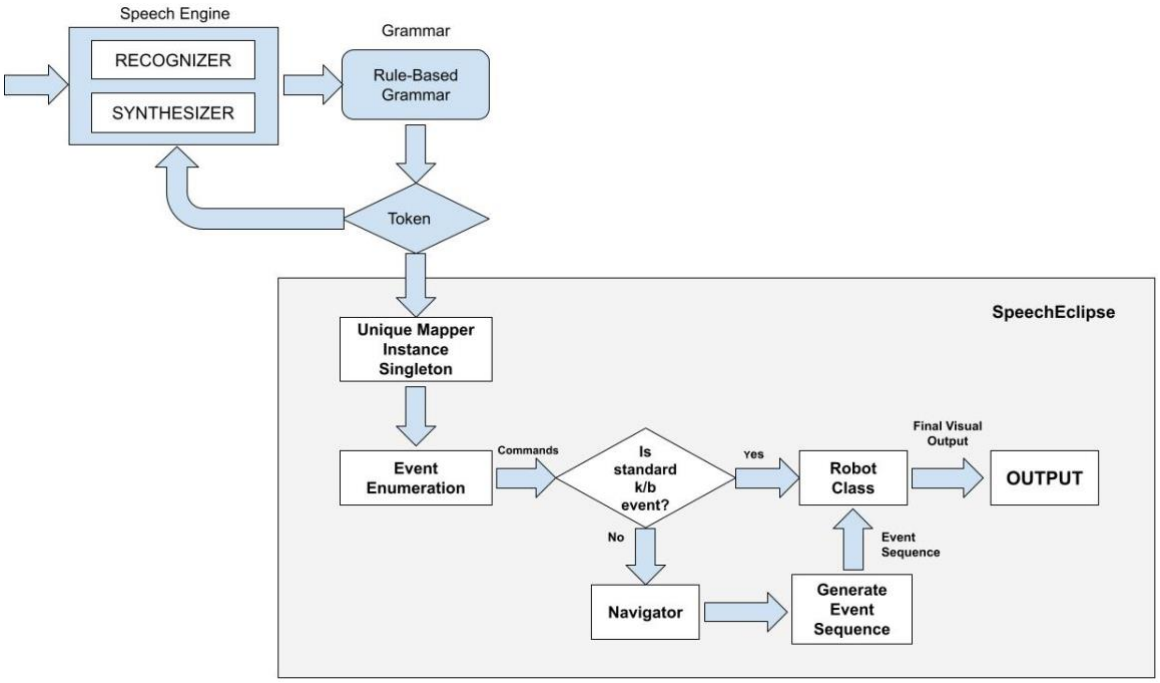

Figura 7. Esquema de funcionamiento de SpeechEclipse.

2.3.2. Polaris

Polaris[10] es otro *plugin* que permite, a través de la voz también, realizar acciones como copiar o cortar texto, pegarlo y abrir diálogos para nuevos archivos. En la Figura 8 se puede ver su interfaz, con un botón integrado en la pestaña superior izquierda.

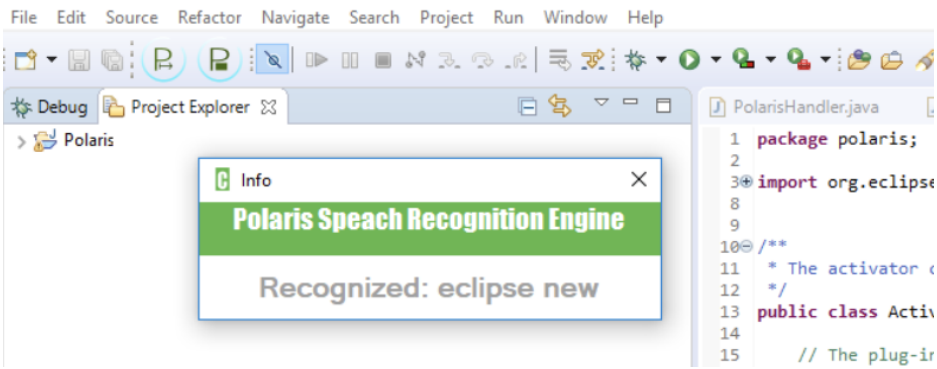

Figura 8. Interfaz de Polaris.

[10] https://sourceforge.net/projects/polarisspeachrecognition/

2.3.3. Ok Eclipse

Ok Eclipse[11], a través de ordenes de voz al igual que los dos anteriores, puede ejecutar acciones como crear una clase con cuatro variables "*integer*" o transformar de código Java a código Python con una orden "*convert*", entre muchas otras. En esta herramienta integrada de Eclipse el reconocimiento de voz se basa en la API de Google Speech.

2.3.4. Dragon

Dragon[12] es una herramienta de la empresa Nuance, dedicada a la creación de herramientas de reconocimiento de voz y procesamiento avanzado del lenguaje natural mediante inteligencia artificial, capaces de convertir conversaciones en textos de forma automática. Dragon puede ser usada para programar en cualquier IDE y en cualquier lenguaje, a pesar de que a veces pueda resultar difícil. Pero lo importante de esta herramienta es que trabaja conjuntamente con VoiceCode[13], la cual sí fue completamente desarrollada para la programación por voz. Además, esta última incluye soporte para accesorios de control por manos libres para manejo de ratón como SmartNav (Figura 9), que permite al usuario un control completo de un ordenador moviendo la cabeza de forma natural.

Figura 9. Accesorio SmartNav para control con movimiento de cabeza.

Como se puede ver, existen varias alternativas a COPS en la actualidad. Todas ellas pueden resultar favorables para la programación para personas con discapacidades, pero ninguna de éstas cuenta con la ampliación desarrollada en este proyecto. Este manejo mediante gestos de la aplicación supondrá un avance mucho mayor en el campo, ayudando a un nivel notablemente superior a estos usuarios.

[11] https://github.com/dharmangbhavsar/ok-eclipse
[12] https://www.nuance.com/es-es/dragon.html
[13] https://voicecode.io/

3. Sistema de reconocimiento de gestos faciales

Durante este capítulo se van a abordar las diferentes fases que componen todo el proceso de construcción del sistema capaz de reconocer determinados gestos que el usuario va a realizar con la cara a través de la cámara.

Para facilitar la comprensión del sistema y todo el proceso que conlleva, el capítulo se dividirá en distintas secciones en función de los pasos que se han llevado a cabo para su desarrollo. También se incluirán en cada apartado las tecnologías usadas en cada caso.

Todo este proceso se puede observar de manera sintetizada en la Figura 10, donde se pueden ver recopiladas 6 fases que se describen a lo largo de las siguientes secciones del capítulo.

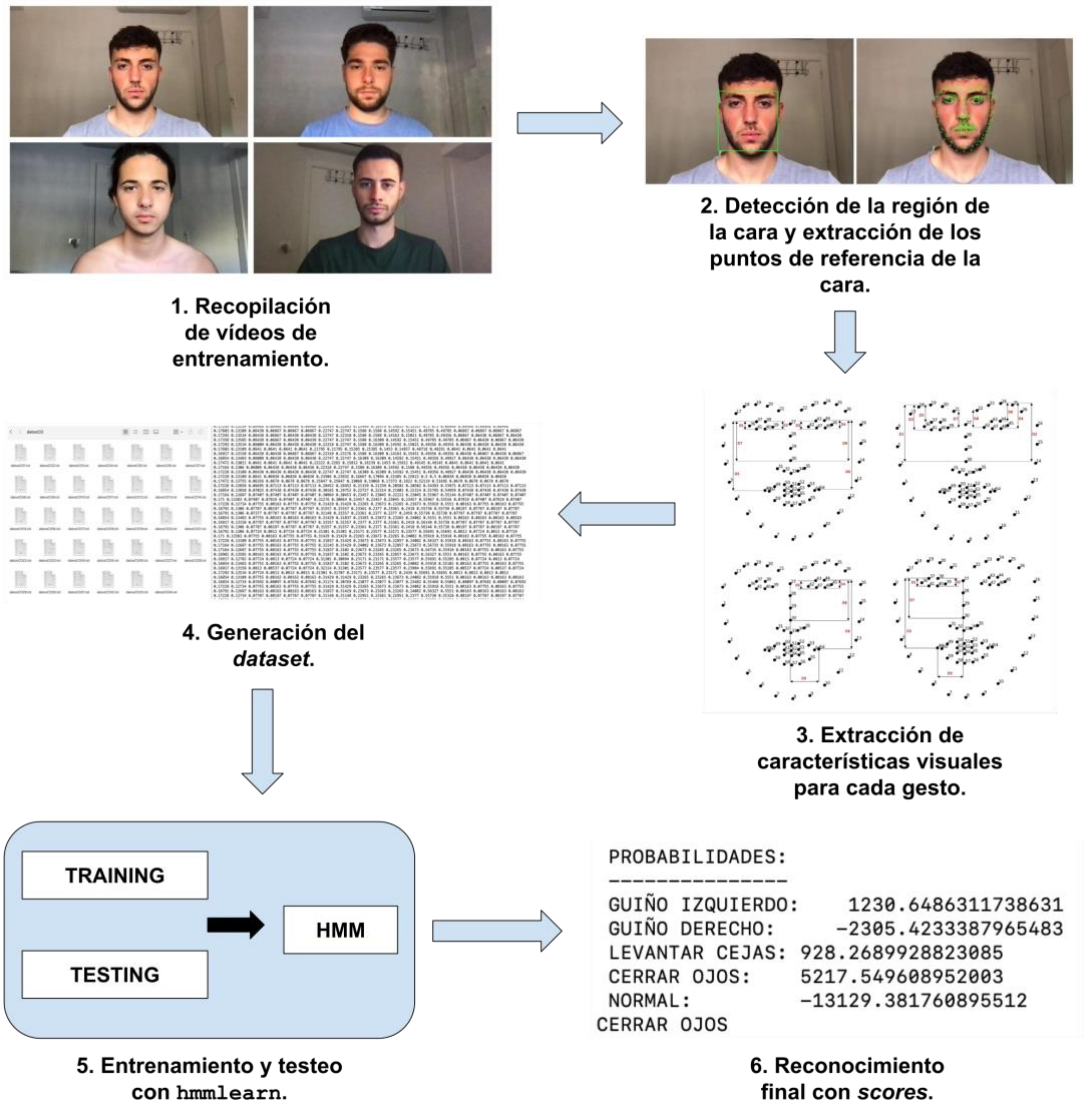

Figura 10. Esquema de las fases del sistema de desarrollo de gestos faciales.

3.1. Recopilación de videos de entrenamiento

El *dataset* propuesto para la construcción del sistema se creará a partir de una serie de videos de entrenamiento realizados con la cámara del ordenador a varias personas. Se trata de videos de 2 segundos exactos, en los cuales estas personas se encuentran realizando el gesto correspondiente.

Sin embargo, para la extracción de estos videos de 2 segundos se han realizado videos más largos en los que la persona va realizando todos los gestos sin ningún orden determinado, y posteriormente se han recortado los fragmentos en los que se encuentran los gestos en cuestión.

La posición de las personas frente a la cámara es la que una persona normalmente adoptaría cuando está frente al ordenador programando o realizando alguna tarea. De esta manera, y teniendo en cuenta lo anterior respecto a los recortes de un video de mayor duración, se ha intentado aportar los escenarios más realistas posibles para un mejor reconocimiento final.

Cabe destacar aquí que durante los 2 segundos del video es necesario asegurarse de que únicamente aparece en cámara la cara del usuario y ninguna otra, ya que, si esto ocurre, los *frames* durante los que aparecen dos o más caras no serían válidos para nuestro *dataset* final. También habrá que tener en cuenta que la posición de la cara sea una posición correcta para que el detector del *software* pueda reconocerla correctamente, es decir, estar en todo momento mirando a la pantalla sin girarse demasiado o subir o bajar mucho la mirada, como se puede ver en las imágenes mostradas en la Figura 11.

Figura 11. Algunos de los usuarios que aparecen en los videos.

Finalmente, para la obtención del *dataset* procesado (que constará de una serie archivos extraídos a partir de estos videos), se puede ver en la Tabla 1 con cuántos videos de 2 segundos cuenta cada uno de los gestos.

Tabla 1. Número de videos por gesto.

Gesto	Número de videos
Levantar cejas	33
Cerrar ojos	34
Guiño izquierdo	33
Guiño derecho	31
Normal	32

3.2. Detección de los puntos característicos de la cara

Una vez recopilados los videos de 2 segundos de los gestos característicos en el apartado anterior, es el momento de extraer el área donde se ubica la información más relevante para conseguir nuestro objetivo: la cara.

En primer lugar, sobre los datos en formato de video se aplicará un software encargado de detectar esta zona de la cara, al cual se le han realizado una serie de modificaciones en vistas a obtener una secuencia de *frames* de la región de la cara que represente la totalidad del gesto. Sabiendo que la frecuencia con la que se tratan los videos en este proyecto es de 30 *frames* por segundo, y mediante el uso de las bibliotecas `opencv`[14] y `dlib`[15], se conseguirá detectar las caras en primera instancia. Al detectar correctamente una cara en el *frame*, como se puede ver en la Figura 12, se empleará ahora un predictor para determinar los puntos clave de la misma.

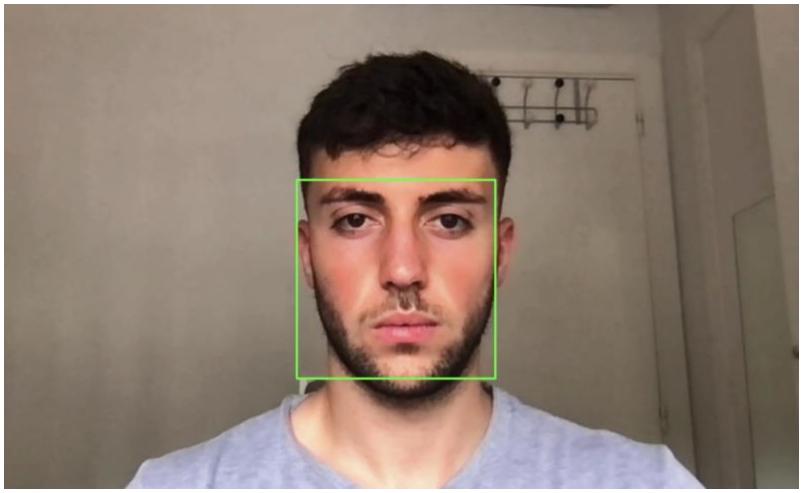

Figura 12. Detección de la zona de la cara.

[14] https://docs.opencv.org/master/
[15] http://dlib.net

Por lo tanto, si en cada *frame* se encuentra una única cara y es posible extraer correctamente los puntos característicos de la misma, se construirá la secuencia de *frames* sobre la que se extraerán las características significativas del gesto realizado.

Para comprender de mejor manera cómo se ha realizado este proceso, es necesario describir brevemente la tecnología empleada para el mismo, es decir, las bibliotecas mencionadas anteriormente usadas para la detección de la región de la cara.

- OpenCV (Open Source Computer Vision Library): Se trata de una biblioteca software de código abierto relacionada con la visión artificial y el *machine learning*, que permite resolver problemas de visión por computador. El objetivo de su creación fue facilitar un entorno de desarrollo dirigido a aplicaciones de visión por computador que fuese fácil de utilizar y altamente eficiente. Cuenta con más de 2000 algoritmos optimizados, entre los que se destacan la detección y reconocimiento de caras, clasificación de acciones en videos, identificación de objetos determinados, seguimiento del movimiento de alguna parte del cuerpo... En este proyecto en concreto, ha permitido el tratamiento de videos, consiguiendo el conjunto de *frames* deseado.

Tras esto será posible trabajar con cada uno de los *frames* que componen la entrada del programa individualmente, y es ahora cuando toma lugar la tecnología `Dlib`.

- Dlib: Se trata de una biblioteca multiplataforma de código libre escrita en C++ que contiene una serie algoritmos y herramientas de aprendizaje automático con el fin de crear software complejo que pueda resolver problemas del mundo real. Es muy apropiado en investigación, en la industria y en el mundo académico, e incluye herramientas y entornos enfocados a las estructuras de datos, álgebra lineal, *machine learning* y procesado de imágenes, entre otros. En este proyecto se ha hecho uso de esta biblioteca para detectar la cara del usuario en cuestión con la función `get_frontal_face_detector()` en primer lugar y, posteriormente, mediante un modelo entrenado, para predecir el área donde se ubican los puntos característicos de la cara con la función `shape_predictor()`. Este modelo predictor se entrena basándose en los 68 puntos de referencia de la cara humana (*facial landmarks*), los cuales se pueden observar en la Figura 13, y se usa para estimar su ubicación en (x,y)-coordenadas. En la sección 3.3 se verá cuáles de estos 68 puntos se han usado para extraer las características más importantes de los gestos determinados.

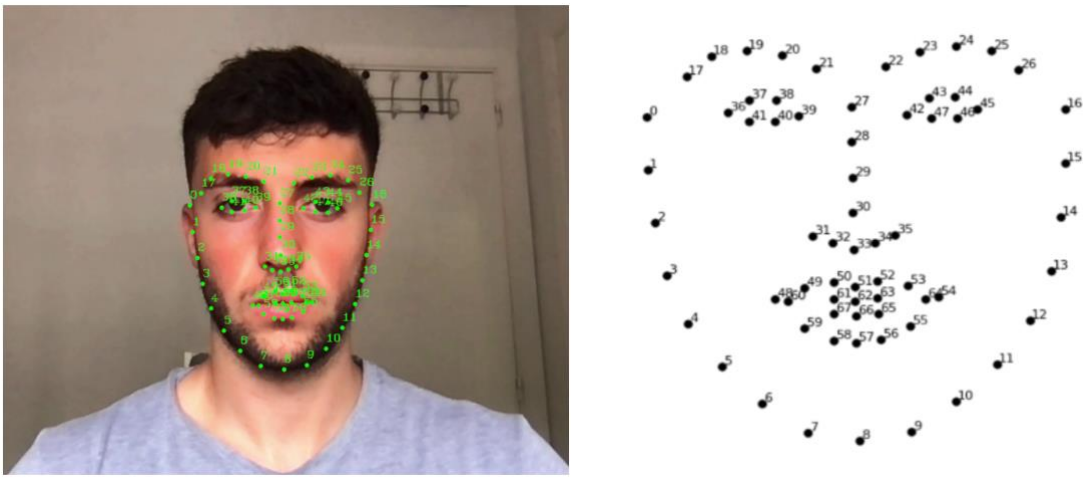

Figura 13. Índices de los 68 puntos de referencia faciales del predictor.

3.3. Extracción de características visuales del movimiento facial

Una vez terminado todo el pre-proceso de los datos descrito hasta el momento, será necesario estudiar un último paso con el que concluiremos nuestro *dataset*. Se trata de aplicar cierto código que permita extraer las características más relevantes para cada uno de los gestos que se ha elegido. Esta fase es un punto fundamental para la construcción de nuestro sistema.

Esto se va a conseguir mediante el tratamiento de las distancias y el movimiento de los puntos escogidos entre los 68 totales mencionados previamente.

Para entender mejor el proceso, vamos a ver para cada gesto qué distancias y puntos hemos decidido que sean característicos para posteriormente entrenar los modelos. Antes de pasar con esto, es necesario destacar que para el tratamiento equitativo de los videos y para tener en cuenta la distancia que tiene el usuario frente a la cámara, hemos tratado todas las distancias normalizándolas. Esto quiere decir que, para evitar confusiones a la hora de entrenar los modelos, se ha extraído de estos puntos de referencia faciales dos distancias de normalización:

- Distancia de normalización de las distancias en el eje Y: esta distancia será en todo momento la distancia entre las coordenadas “y” de los puntos 27 y 8 de la Figura 13, es decir, la altura de la cara hasta la terminación de la nariz.

- Distancia de normalización de las distancias en el eje X: esta distancia será en todo momento la distancia entre las coordenadas “x” de los puntos 0 y 16 de la Figura 13, es decir, la anchura de la cara.

De esta manera, y como se ha comentado anteriormente, no se encontrarán problemas de reconocimiento debido a la distancia del usuario a la cámara del ordenador de trabajo.

3.3.1. Características del gesto “Levantar cejas”

Para el gesto de levantar las cejas, se ha pensado que podrían ser buenas características las siguientes:

- Una distancia de eje Y en cada lado de la cara, izquierdo y derecho, entre el punto superior de la ceja y el punto inferior de la nariz, como vemos en la Figura 14 llamadas D1 y D2.

- Una distancia de eje Y también en cada lado de la cara, izquierdo y derecho, entre el punto más alto de la ceja y un punto del parpado inferior del ojo del lado correspondiente. En la Figura 14 llamadas D3 y D4.

- Otra distancia de eje Y también en cada lado de la cara, izquierdo y derecho, entre el punto más alto de la ceja y un punto del parpado superior del ojo del lado correspondiente. En la Figura 14 llamadas D5 y D6.

- Por último, otra distancia de eje Y en cada lado, entre el punto más interno de la ceja y el más interno del ojo en el lado correspondiente. En la Figura 14 llamadas D7 y D8. Estas distancias son muy importantes debido a que, para la mayoría de las personas, cuando tratan de levantar las cejas, la mayor variación de distancia que se produce entre la ceja y el ojo es en esta parte interna. Sin embargo, en la parte externa de éstos la distancia a veces ni siquiera varía.

Como se puede ver en la Figura 14, todas estas distancias tienen en cuenta el movimiento en el eje Y de las cejas, y en todo momento tratan con puntos en los que el levantamiento de las cejas será más notorio, como se ha explicado.

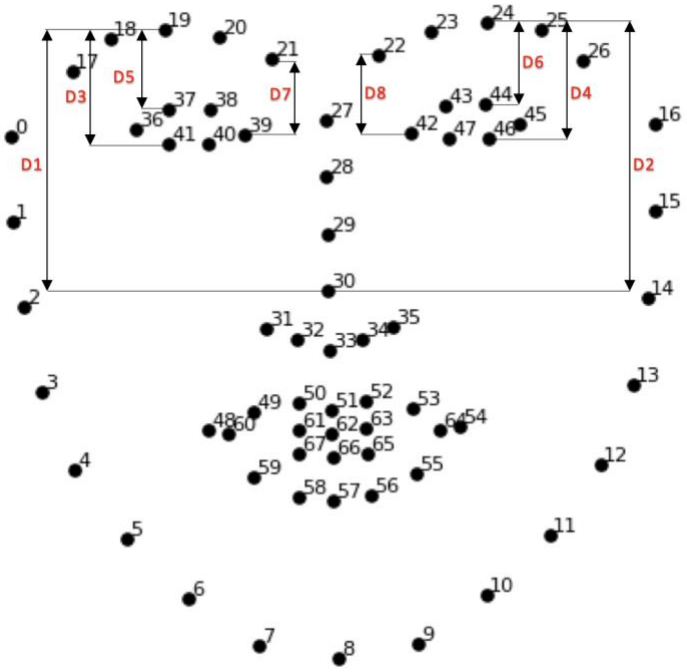

Figura 14. Distancias más destacables en el gesto de “Levantar cejas”.

3.3.2. Características del gesto “Cerrar ojos”

Para el gesto de cerrar los ojos durante un segundo aproximadamente, se ha pensado que podrían ser buenas características las siguientes:

- Dos distancias de eje Y en cada ojo, izquierdo y derecho, que nos permiten conocer cuándo el ojo se está cerrando, ya que estas distancias disminuirían considerablemente. Éstas serían D1 y D2 para el ojo izquierdo y D3 y D4 para el ojo derecho en la Figura 15.

- Una distancia de eje Y también en cada lado, izquierdo y derecho, entre un punto superior del ojo y el exterior del labio de ese lado. Éstas serán D5 y D6 en la Figura 15.

- Otra distancia de eje Y en cada lado, izquierdo y derecho, entre el mismo punto superior del ojo de ese lado y el punto inferior de la nariz. Serán D7 y D8 en la Figura 15.

Estas cuatro últimas distancias son características del gesto de cerrar los ojos, ya que en todo momento tienen en cuenta el movimiento del parpado superior, y en caso de que las distancias disminuyan, significará que el ojo se está cerrando, puesto que el párpado estaría bajando (por ser distancias en todo momento de la coordenada Y).

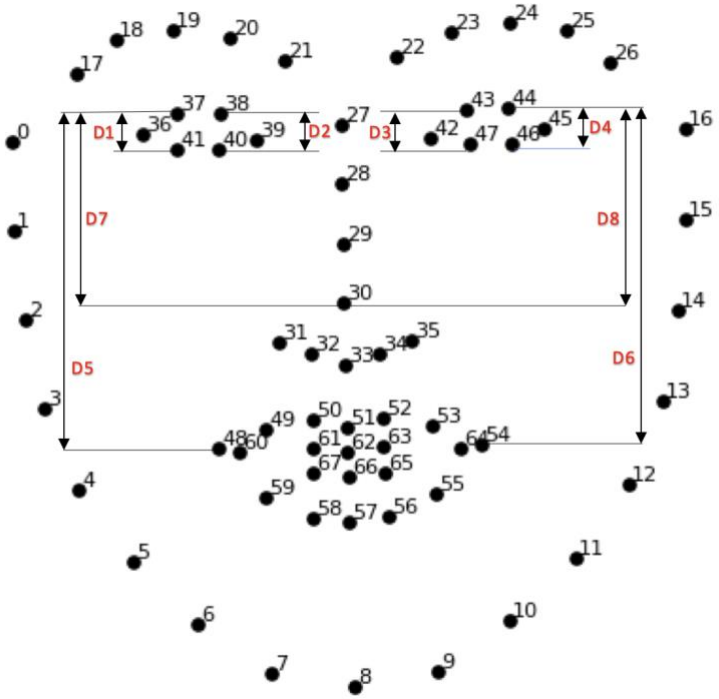

Figura 15. Distancias más destacables en el gesto de “Cerrar ojos”.

3.3.3. Características de los gestos “Guiño izquierdo” y “Guiño derecho”

Para los gestos de guiñar cada ojo durante un segundo aproximadamente, hay que comentar que se ha decidido, con el objetivo de hacer el gesto más característico y diferencial, añadir al guiño del ojo correspondiente un pequeño movimiento del labio hacia el lado en el que se está realizando el guiño. De esta manera, las distancias en estos gestos van a ser las mismas que para el gesto de cerrar los ojos en el lado correspondiente al guiño únicamente, pero añadiendo este último detalle:

- Una distancia de eje X en cada lado, izquierda y derecha, que va desde el punto exterior del labio de cada lado hasta el punto superior de la nariz. De esta manera, como vemos en D9 en las Figuras 16 y 17, miramos la distancia desde el centro de la cara hasta el exterior del labio, y podemos ver el movimiento lateral del mismo.

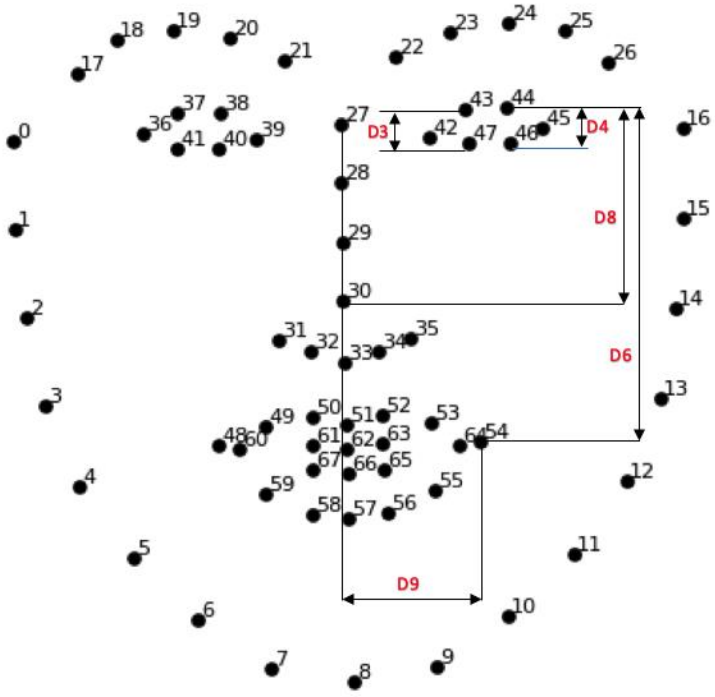

Figura 16. Distancias más destacables en el gesto de “Guiño derecho”.

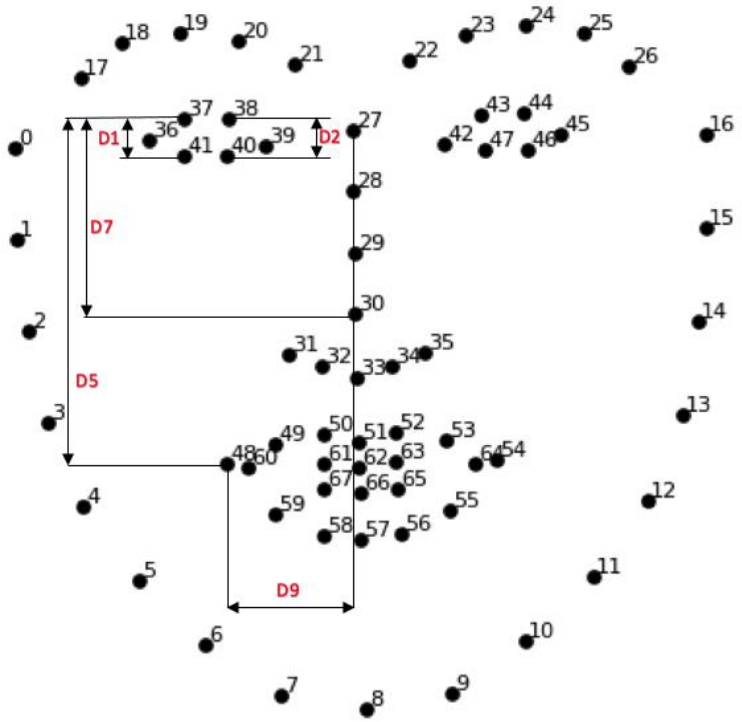

Figura 17. Distancias más destacables en el gesto de "Guiño izquierdo".

Para finalizar con la fase de extracción de características visuales del movimiento facial, se ha de destacar una serie de cosas. La primera de ellas es que en estos apartados se ha visto cada una de las distancias características más importantes y significativas que se ha añadido al modelo pensando en ese gesto determinado. Sin embargo, para entrenar posteriormente los modelos de cada gesto, se van a tener en cuenta el conjunto de todas estas características, independientemente de cuál sea el gesto. Esto se debe a que para el futuro reconocimiento necesitamos tener en cuenta todo el conjunto, para, a partir de éste, poder obtener qué características son las que más se asemejan al modelo del gesto realizado. Por ello, la segunda de las cosas que se quería destacar es que el gesto normal no tiene características que sean significativas únicamente de él (por ello no se ha dedicado un apartado exclusivamente al mismo), si no que, como se ha comentado, su modelo se entrenará con el conjunto de todas, al igual que los demás.

Para tener una visión general del conjunto de 18 características totales recopiladas con el que entrenaremos los modelos, se muestra en la Tabla 2 el vector que reúne todas en el código usado.

Tabla 2. Vector de características de los gestos para entrenar los modelos.

PUNTOS CLAVE								
0	1	2	3	4	5	6	7	8
D9(GI)	D9(GD)	D3(CO)	D4(CO)	D1(CO)	D2(CO)	D4(LC)	D3(LC)	D6(LC)
9	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
D5(LC)	D8(LC)	D7(LC)	D2(LC)	D1(LC)	D8(CO)	D7(CO)	D6(CO)	D5(CO)

Es muy importante destacar aquí para evitar confusiones que, en el código desarrollado en el proyecto, debido a que la cámara usada cuenta con el modo espejo, tomamos la parte derecha de la cara mostrada por pantalla como la parte izquierda de la cara real, y viceversa.

Veremos, en la sección 3.4, cómo a partir de estas características y de los videos recopilados de entrenamiento obtendremos el *dataset* del sistema.

3.4. Generación del *dataset*

Tras la obtención de los videos y la decisión final de cuáles son las características más significativas de cada gesto, es el momento de generar el *dataset*. Debido al formato que se ha decidido emplear para el reconocimiento, el de entrenamiento mediante Modelos ocultos de Markov (explicado en las secciones 3.5 y 3.6), el *dataset* de este proyecto constará de un documento de texto (txt) por cada uno de los vídeos recopilados.

El número de líneas del documento de texto será igual al número de *frames* con los que cuenta el vídeo. Esto es debido a que, con el código desarrollado para la extracción de características con los puntos clave de la cara, se tratan *frame* a *frame* todos los videos, y para cada uno de estos *frame*s se obtendrá el vector de características correspondiente a ese instante. De esta manera, el sistema, mediante una serie de funciones básicas de Python, se encargará de ir guardando y escribiendo el vector de características correspondiente a cada instante en el fichero de texto plano propio. Así, el documento de texto de cada video contará con "x" (entre 59 y 61) vectores de características escritos en líneas distintas, siendo "x" el número de *frames* con los que cuenta el video.

Un pequeño apunte que comentar aquí es que, como se ha explicado en el párrafo anterior, el número de *frames* puede variar de 59 a 61 ya que, a pesar de que los recortes de los videos han sido exactamente de 2 segundos y de que la frecuencia de la cámara de grabación es de 30 *frames* por segundo, existe la posibilidad de una breve variación de 1 *frame*. Sin embargo, este pequeño detalle no resultará problemático a la hora de entrenar los modelos, ya que es insignificante.

Se puede ver en la Figura 18 parte de uno de los ejemplos de fichero del *dataset* generado para este proyecto. Cada línea cuenta con 18 números reales, que siguen el formato del vector de la Tabla 2.

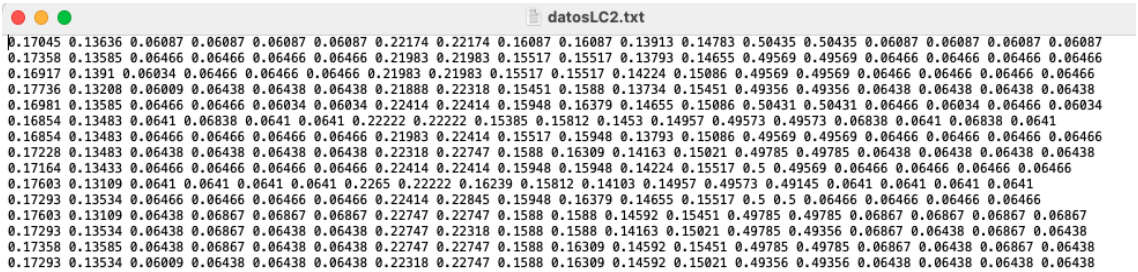

Figura 18. Ejemplo de documento de texto para el *dataset*.

3.5. Entrenamiento del sistema

Tras obtener todos los documentos de texto necesarios, es ahora el momento de la fase de entrenamiento de sistema. En ésta, para el entrenamiento de los modelos de cada gesto, se ha decido usar la librería de Python `hmmlearn`[16], que implementa los modelos ocultos de Markov (HMM – abreviatura en inglés) [16]. Para entender mejor el procedimiento posteriormente, se hará una breve introducción de los HMM.

3.5.1. Introducción a los modelos ocultos de Markov

El origen de los HMM tiene lugar en los años 50 del siglo pasado, con un grupo de investigadores tratando el problema de caracterización de procesos estocásticos para los cuales no contaban con las observaciones necesarias. De esta manera surgió la idea de modelar un proceso estocástico particular como un modelo estocástico doble, en el cual las realizaciones de un primer proceso oculto originaban al segundo proceso observado. Ambos procesos se lograban caracterizar usando sólo el observado.

Un HMM es un modelo probabilístico o estadístico generativo en el que una secuencia de variables observables "X" es generada por una secuencia de estados ocultos internos "Z". Pero estos estados ocultos no se observan directamente, si no que se supone que las transiciones entre estados ocultos tienen la forma de una cadena de Markov, la cual vendrá especificada por una matriz de probabilidades de inicio "π" y una matriz de probabilidades de transición "A". Por otro lado, la probabilidad de emisión de un estado observable puede ser cualquier distribución con parámetros "O" condicionado por el estado oculto actual. Por ello, cualquier HMM está siempre determinado por los parámetros "π", "A" y "O".

Sin embargo, con esto existen tres problemas fundamentales en los HMM. Dados los parámetros del modelo y los estados observables, el primero de los problemas es estimar la secuencia óptima de estados ocultos (y su probabilidad asociada), y el segundo es calcular la probabilidad de una secuencia observable. Dados únicamente los estados observables se tiene el tercer problema, que es estimar los parámetros del modelo.

Los dos primeros problemas se resolverán mediante los algoritmos de programación dinámica de Viterbi y Forward-Backward, respectivamente [17]. El último de los problemas se resolverá mediante el algoritmo iterativo de Baum-Welch, de expectación-maximización (EM) [18].

[16] https://hmmlearn.readthedocs.io/en/latest/tutorial.html

3.5.2. Entrenamiento con `hmmlearn`

El proceso de entrenamiento con `hmmlearn` en Python comienza creando cada uno de los modelos. En este proyecto serán cinco los modelos definidos, uno para el gesto de levantar las cejas, otro para cerrar ojos, guiño izquierdo, guiño derecho, y finalmente, otro para el gesto normal. La función `GaussianHMM`[17] es la que nos permite crear estos modelos, mediante HMM con emisiones Gaussianas, como su propio nombre indica. Los parámetros de esta función para todos los modelos serán:

- Número de estados (`n_components` en la Figura 19) = 3
- Tipo de covarianza (`covariance_type` en la Figura 19) = "full", que permite a cada estado usar la matriz de covarianzas completa.

En la Figura 19 se observan el resto de los parámetros, que se completarán de forma automática con unos valores determinados, al contrario de los dos mencionados, los cuáles se han tenido que pasar en forma de parámetros en la función.

```
class hmmlearn.hmm.GaussianHMM(n_components=1,
covariance_type='diag', min_covar=0.001, startprob_prior=1.0,
transmat_prior=1.0, means_prior=0, means_weight=0,
covars_prior=0.01, covars_weight=1, algorithm='viterbi',
random_state=None, n_iter=10, tol=0.01, verbose=False,
params='stmc', init_params='stmc')
```

Figura 19. Estructura de la función `GaussianHMM`.

Esta topología ha sido la escogida para los HMM´s ya que está basada en la que se usa normalmente en el reconocimiento de gestos.

Una vez han sido creados los modelos, llega el momento de entrenarlos. Para esto, se ha requerido de la función de `hmmlearn`, `fit`. Ésta requiere de entrada una matriz de secuencias de muestras concatenadas, y es esto lo que se ha realizado en el proyecto.

Para cada uno de los documentos de texto del *dataset*, mediante una serie de funciones básicas de tratamiento de documentos y de matrices en Python, se ha accedido a ellos para leer e ir guardando sus líneas (en forma de vector sin espacios) en una matriz. Finalmente, las matrices de todos los ficheros se concatenan en otra matriz final para cada modelo, la cual se pasará por parámetro a la función de entrenamiento `fit` comentada en el párrafo anterior.

[17]https://hmmlearn.readthedocs.io/en/latest/api.html#hmmlearn.hmm.GaussianHMM

Una vez terminada esta fase, se tienen correctamente entrenados cada uno de los modelos con sus respectivas muestras, y con esto se hace posible la ejecución del paso final que conlleva al objetivo principal del proyecto, el reconocimiento de los gestos.

3.6. Reconocimiento final

Esta fase va a ser la que permita, a través de una grabación o de una captura en directo de la cámara, reconocer qué gesto es el que se encuentra realizando el usuario en un instante determinado.

Este proceso va a requerir del uso de la función `score` de `hmmlearn`. El objetivo con esta función es, pasando una matriz con el mismo formato que el visto en el *dataset* y en el entrenamiento de los modelos (matrices que recojan todas las características de todos los *frames* de videos de 2 segundos recortados del directo o de la grabación), extraer la probabilidad de la muestra en cada modelo.

Para que esto sea posible, se ha decidido que al comenzar la grabación la primera matriz que se le pasa a la función `score` será la correspondiente al fichero de uno de los videos de muestra del gesto normal. De esta manera, al comienzo, la persona se encontrará realizando el gesto normal y, posteriormente, cada 0,5 segundos, es decir, 15 *frames*, esta matriz se irá actualizando y pasando a la función continuamente.

Sabiendo esto, y teniendo los resultados de los *scores* realizados, se elegirá ganador a aquel modelo con la probabilidad más alta. Sin embargo, hay que destacar un pequeño detalle que se ha debido de tener en cuenta para el correcto reconocimiento.

Tras la realización de una serie de pruebas en el sistema desarrollado, se ha notado que la diferenciación de las probabilidades de cada modelo que no es el de gesto normal es bastante clara, es decir, el *score* del gesto que se esta realizando en el momento destaca por encima de los demás notablemente. Sin embargo, cuando el usuario no se encuentra realizando ningún gesto característico de una funcionalidad, es decir, está normal, los *scores* son bastante similares para todos los modelos. Es por esto por lo que se ha adoptado la siguiente solución: cuando las probabilidades no varían más de una cantidad “X” (que se ha determinado tras la realización de una serie de pruebas para ajustar adecuadamente este valor), se reconocerá que el gesto que el usuario está realizando en ese momento es el normal. Cuando existe una probabilidad que resalta por encima de las demás una cantidad mayor que ese valor “X”, éste será el gesto reconocido. En el proyecto se ha decidido que este valor de “X” sea 2000.

En la Figura 20 se aprecian varios resultados de una de las pruebas realizadas donde se ve qué gesto ha resultado finalmente ganador teniendo en cuenta los aspectos comentados en este apartado.

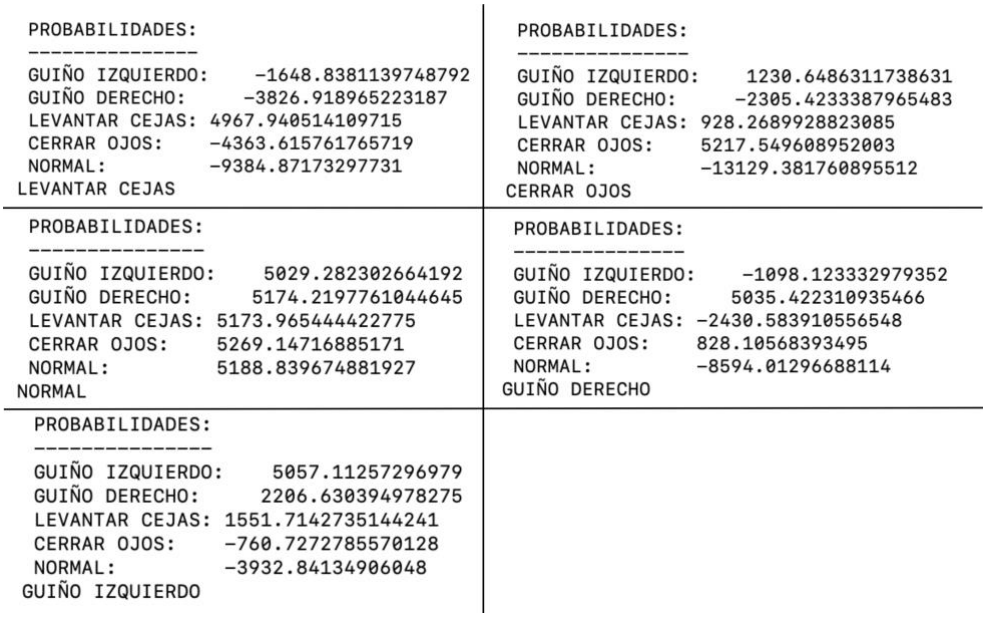

Figura 20. *Scores* para cada modelo y gesto reconocido en una muestra.

3.7. Evaluación del rendimiento del sistema

Para finalizar con el sistema de reconocimiento de gestos desarrollado, se va a realizar una evaluación de sus resultados con el objetivo de conocer un valor aproximado de la precisión o el rendimiento de éste. Para ello, se ha decidido emplear una validación cruzada. La validación cruzada[18] es un método de evaluación de los resultados de un análisis estadístico, muy utilizada en proyectos de inteligencia artificial para validar modelos generados, que permite garantizar la independencia entre el subconjunto de datos de entrenamiento y el de prueba. Consiste en iterar y calcular la media aritmética de las precisiones de predicción de las diferentes particiones. Esta técnica se usará en casos donde el principal fin es la predicción y se desea estimar la precisión de un modelo que se llevará a cabo a la práctica, como es nuestro caso en este trabajo. El tipo de validación cruzada que se va a llevar a cabo en esta memoria es la de K iteraciones, siendo K igual a tres para nuestro conjunto de datos. Esto supone que los datos de muestra se dividirán en tres subconjuntos (especificados en la Tabla 3), de manera que uno de ellos se utilizará como datos de prueba y los dos restantes como datos de entrenamiento. Este proceso se repetirá durante tres iteraciones con cada uno de los posibles subconjuntos de datos de prueba. Finalmente, para obtener el resultado definitivo se calculará la media aritmética de los resultados de cada iteración. Este método es muy preciso, ya que la evaluación se realiza a partir de K combinaciones de datos de entrenamiento y de prueba, pero a pesar de esto tiene la desventaja de que puede resultar lento desde el punto de vista computacional.

[18] https://es.wikipedia.org/wiki/Validación_cruzada

Tabla 3. Subconjuntos de cada gesto para la validación cruzada.

Gesto	Número de videos	Subconjunto 1	Subconjunto 2	Subconjunto 3
Levantar cejas	33	1-11	12-22	23-33
Cerrar ojos	34	1-11	12-22	23-34
Guiño izquierdo	33	1-11	12-22	23-33
Guiño derecho	31	1-10	11-20	21-31
Normal	32	1-11	12-22	23-32

Tras conocer el funcionamiento de esta técnica, se verá detalladamente cómo se ha aplicado la misma a nuestro sistema. Como se puede ver en la Tabla 3, se ha dividido el conjunto de datos en 3 subconjuntos, por lo que la validación cruzada contará con 3 iteraciones, como se ha comentado al principio de la sección. Se verán a continuación cada una de ellas.

- **Primera iteración**: se usan los subconjuntos 1 y 2 como datos de entrenamiento y el subconjunto 3 como datos de prueba. Los resultados obtenidos se pueden observar en la Tabla 4.

Tabla 4. Primera iteración de la validación cruzada.

Gesto	Datos de entrenamiento	Datos de prueba	Nº aciertos	Nº fallos
Levantar cejas	Subconjuntos 1 y 2	Subconjunto 3	11	0
Cerrar ojos			10	2
Guiño izquierdo			11	0
Guiño derecho			11	0
Normal			8	2

- **Segunda iteración**: se usan los subconjuntos 1 y 3 como datos de entrenamiento y el subconjunto 2 como datos de prueba. Los resultados obtenidos se pueden observar en la Tabla 5.

Tabla 5. Segunda iteración de la validación cruzada.

Gesto	Datos de entrenamiento	Datos de prueba	Nº aciertos	Nº fallos
Levantar cejas	Subconjuntos 1 y 3	Subconjunto 2	11	0
Cerrar ojos			6	5
Guiño izquierdo			11	0
Guiño derecho			10	0
Normal			9	2

- **Tercera iteración**: se usan los subconjuntos 2 y 3 como datos de entrenamiento y el subconjunto 1 como datos de prueba. Los resultados obtenidos se pueden observar en la Tabla 6.

Tabla 6. Tercera iteración de la validación cruzada.

Gesto	Datos de entrenamiento	Datos de prueba	Nº aciertos	Nº fallos
Levantar cejas	Subconjuntos 2 y 3	Subconjunto 1	11	0
Cerrar ojos			10	1
Guiño izquierdo			11	0
Guiño derecho			10	0
Normal			11	0

Una vez se han realizado todas las pruebas necesarias de cada iteración, para calcular la tasa de acierto se dividirá el número total de aciertos por el número total de muestras. Esto es 151 dividido por 163, lo que resulta en una tasa de acierto de 92.64 %. Se puede asumir que esta tasa de acierto es lo suficientemente buena para integrar este sistema de reconocimiento de gestos en COPS.

4. Inclusión del sistema de reconocimiento en el código de la aplicación

Tras terminar y comprobar con una serie de pruebas que el sistema de reconocimiento de gestos faciales cumple con los requisitos propuestos y funciona correctamente, es el momento de integrarlo o comunicarlo de alguna forma con el *plugin* de Eclipse. Para ello, se ha determinado que una buena forma de hacerlo sería mediante el uso de *sockets*, que a través de la comunicación de procesos permitan, cuando sea necesario, activar las funcionalidades que se requieren. Antes de pasar a introducir y describir más detalladamente el uso de los *sockets* en este proyecto, se va a explicar y analizar la estructura de COPS a nivel superficial, con el objetivo de entender mejor la integración posteriormente.

4.1. Estructura y análisis del sistema COPS

Siguiendo la estructura del plugin vista en [19], se puede afirmar que COPS se divide en dos procedimientos claramente diferenciados: iATROS y el *plugin* en sí.

El encargado de todo el proceso de reconocimiento de voz es iATROS, que en nuestra ampliación se activaría con el gesto de levantar las cejas. Para utilizarlo, es necesario ejecutar `iatros-run`, un archivo ejecutable encargado de arrancar su sistema de reconocimiento. Para realizar cualquier modificación en la configuración será necesario modificar el archivo `cops.conf`, donde se determinan algunas de las preferencias para la ejecución del reconocedor como, por ejemplo, el vocabulario usado (disponible en español o inglés), la ruta donde debe encontrarse el archivo de entrada para el reconocimiento o los modelos de lenguaje. Tras decidir la configuración y ejecutar `iatros-run`, el sistema de reconocimiento podrá tomar una entrada y devolver el texto correspondiente. Esta entrada debe ser un cepstrum (representación de una señal al aplicarle la transformada de Fourier inversa) para poder realizar la conversión a texto. Para conseguir un cepstrum, se debe comenzar con un audio en formato de audio sin compresión, en nuestro caso WAV (*Waveform Audio Format*). Con esto, para la transformación de formato WAV a RAW (otro formato de audio sin compresión), se debe ejecutar el archivo `wav2raw`. Con el archivo de audio RAW y al ejecutar `raw2CC` se llamará a `iatros-speech-cepstral`, que generará el archivo `cops.CC`, el cual es el cepstrum que se buscaba. Si iATROS se encuentra en ejecución tras la creación de éste, llevará a cabo el reconocimiento y devolverá como salida final `cops.out`, que contendrá el texto reconocido a partir del audio inicial. Podemos ver en la Figura 21 un esquema con todo este procedimiento.

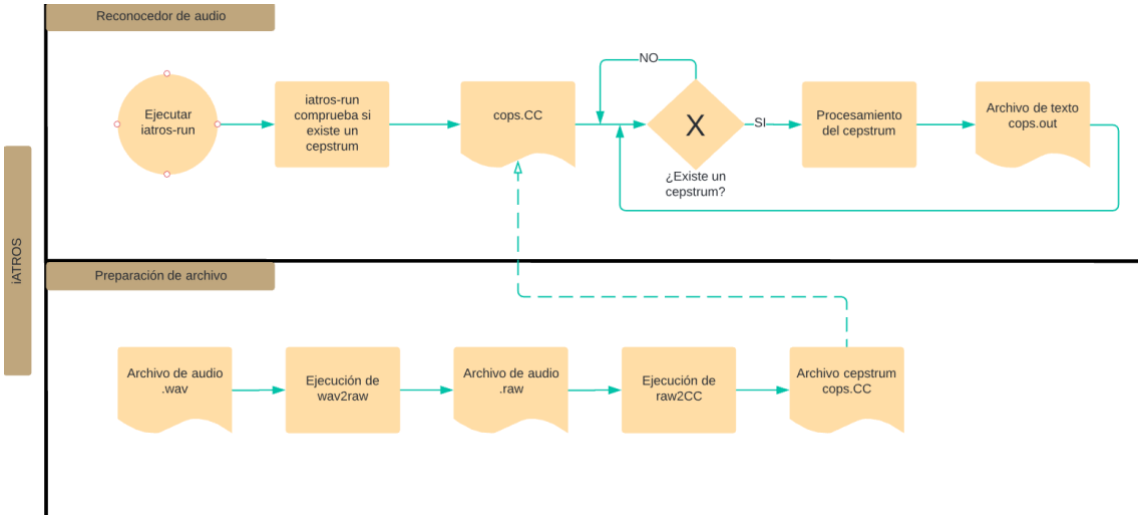

Figura 21. Esquema del funcionamiento de iATROS.

Sin embargo, para la ampliación de este proyecto no es necesario conocer a fondo el subsistema iATROS, ya que éste no ha sido modificado. A pesar de esto, se ha comentado brevemente para poder tener una visión a nivel superficial de su funcionamiento.

Dicho esto, se pasará a estudiar el *plugin* en sí internamente. COPS está compuesto por los siguientes elementos: una carpeta `icons` donde se encuentran las imágenes de los iconos que se utilizan en el *plugin* (micrófono, menú, aumento y decremento de fuente...), otra carpeta `lib` con las librerías que se usan a lo largo el plugin, un archivo `plugin.xml` para precisar algunos de los aspectos visuales y, finalmente, las fuentes divididas en cinco paquetes: `codeshine`, `codeshine.preferences`, `codeshine.speech`, `codeshine.utils` y `codeshine.views`. Esta estructura se puede ver en la Figura 22.

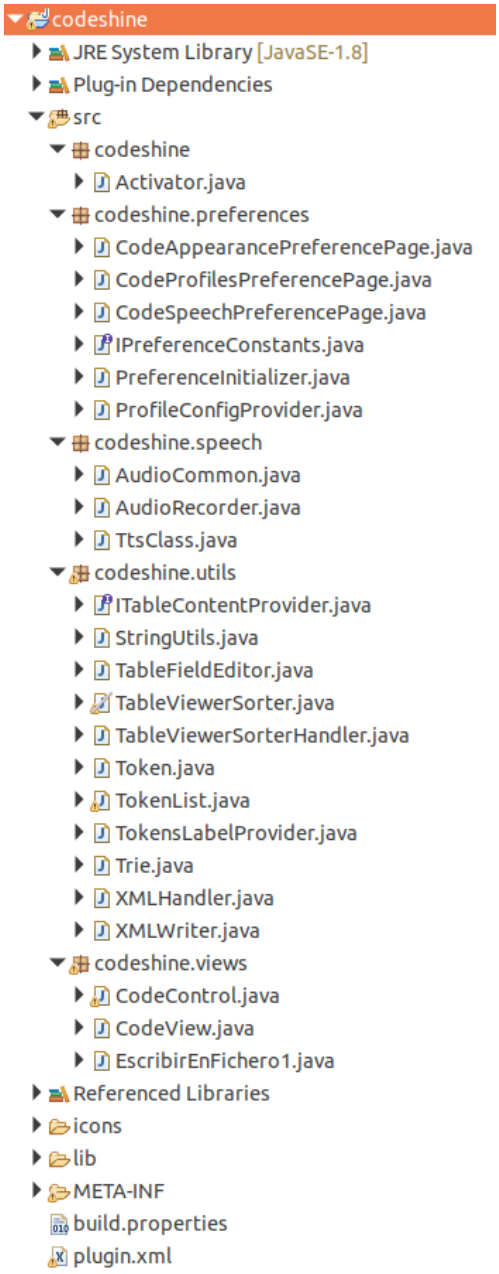

Figura 22. Estructura de COPS.

En el paquete `codeshine` se encuentra la clase `Activator`, la cual es el punto de partida ya que se carga en el arranque y es la encargada de informar de que estamos ante un *plugin* de interfaz de Eclipse.

En el paquete `preferences` se encuentran las clases de gestión de preferencias y perfiles del *plugin*. La primera de ellas es `CodeAppearancePreferencePage`, la cual determina la página de preferencias de visualización del código que también podemos observar accediendo a las preferencias, donde se elige el color de la letra y del fondo, y la fuente. La clase `CodeSpeechPreferencePage` define las preferencias de la síntesis de voz, en la cual se puede modificar su tono, entre otros aspectos. Luego, `CodeProfilesPreferencePage` define la ventana donde es posible guardar y cargar

distintos perfiles, que permiten guardar la configuración de la sesión y cargarla en la siguiente. Para modificar un perfil, se encuentra la clase `ProfileConfigProvider`. Cabe destacar que los perfiles se guardan como archivos XML en la carpeta elegida por el usuario. Y para finalizar con este paquete, encontramos las clases `PreferenceInitializer`, que carga las opciones predeterminadas al iniciar el plugin, e `IPreferenceConstants`, interfaz donde se encuentran las variables que se pueden modificar en las preferencias.

Otro paquete es `speech`, y en él se hallan las clases relacionadas con la síntesis y reconocimiento de voz. La primera es `TtsClass`, la cual maneja la reproducción de audio en COPS, basándose primordialmente en FreeTTS (comentado en el Capítulo 1), un paquete de recursos *open source* para la síntesis de voz construido totalmente en Java. `AudioCommon` es la clase que cuenta con métodos para la gestión de audio. Y, por último, la clase `AudioRecorder` es donde se lleva a cabo todo el proceso de reconocimiento de voz del subsistema iATROS explicado al principio de esta sección.

Le sigue el paquete `utils`, en el cual las clases `Token`, `TokenList` y `TokenLabelProvider` contribuyen a la gestión y visualización de la selección de perfiles de preferencias, por ejemplo. El resto de clases, como `TableFieldEditor` o `TableViewerSorter`, también se encargan de facilitar la gestión y visualización de datos para el *plugin*.

Para finalizar, en el paquete `views` se encuentran las clases encargadas de mostrar y ejecutar correctamente la interfaz de COPS, y es aquí donde se ha incluido la nueva clase creada para la ampliación de este proyecto que se comentará en las secciones 4.3 y 4.4.

La primera clase que se puede encontrar en él es `CodeControl`, cuyo constructor cuenta con un *listener* encargado de controlar los eventos en caso de que estos sean activados a través de atajos de teclado. Cabe destacar también en esta clase el método `speaker`, encargado de preparar la síntesis por voz y que se llamará desde la clase `CodeView` (vista más adelante), y los métodos `set` de cada parámetro de visualización, como la fuente o el color de la letra.

La clase `EscribirEnFichero1` es la encargada de realizar las llamadas necesarias para el proceso de síntesis de voz y su constructor debe invocarse con la voz finalmente elegida para la reproducción del texto. Cuando esto último ocurre, en el constructor se crea una instancia de la clase `Voice` (clase del paquete de FreeTTS donde se encuentra el proceso de pasar de texto a voz) y se le asigna una voz mediante una instancia de `VoiceManager` (clase del paquete de FreeTTS para la gestión de voces). A ésta se le pasa la cadena de caracteres de la voz deseada, indicada anteriormente en el constructor, y se consulta si la voz está disponible para la reproducción. Un método a destacar en esta clase es `toFile`, que llama al método `speak` de `Voice` donde se realiza la síntesis y reproducción de voz, y graba el resultado en el archivo que se le indique como parámetro. Otros métodos son `listAllVoices`, que da la lista de las posibles voces a través de `VoiceManager`, o `getExtension`, que obtiene el formato y la extensión de un archivo.

Para terminar, la clase `CodeView` implementa los métodos encargados de que la interfaz funcione. El primero de ellos es `selectionChanged`, que comunica al *listener* de la clase cuándo se cambia la selección en el editor de Eclipse para que la pantalla clonada de COPS también cambie. Los siguientes métodos son `init`, que inicializa la interfaz de COPS y crea un objeto de tipo `AudioRecorder` (del paquete `speech` comentado anteriormente y cuyo objetivo es ejecutar la grabación y los procesos necesarios para que el audio sea reconocido y convertido a texto), y `dispose`, que finaliza tanto el `AudioRecorder` como los *listener* de la interfaz. Otro método es `makeActions`, cuya función es crear los botones de la interfaz de COPS. Éstos son objetos de tipo `Action` creados para cada funcionalidad, a los cuales se les destina una imagen, un texto y la acción a realizar. Para las funcionalidades de aumentar y disminuir texto o síntesis por voz, únicamente será necesario llamar al método correspondiente de la clase `CodeControl`. Sin embargo, la funcionalidad del reconocimiento de voz es más complicada y, por lo tanto, es necesario más código para su funcionamiento. Al principio de esta ejecución se comprueba el estado de la acción de reconocimiento de voz mediante una variable de control, ya que ésta cuenta con dos etapas: el inicio de la grabación al pulsar el botón la primera vez y su final y procesamiento al pulsarlo la segunda vez. En la primera etapa se llama al método `startRecording` de la clase `AudioRecorder` y se cambia la imagen del icono para que el usuario sea consciente de que la grabación ha comenzado. Tras esto se actualizará la variable de control para conocer la fase en la que nos encontramos. En la segunda etapa, se vuelve a la imagen original, se actualiza la variable de control y se llama a `stopRecording`, a `performRecognition` para iniciar el proceso de transformación del audio y a `restoreRecording` para restaurar el estado de `AudioRecorder`. Finalmente, el método `updateText` escribirá el texto en el editor de Eclipse.

La estructura de llamadas al método correspondiente de la clase `CodeControl` y el tratamiento de las dos fases para el reconocimiento de voz ha servido de inspiración para la nueva clase creada como ampliación, como se podrá ver en la sección 4.4.

4.2. Introducción a los *sockets*

Una vez entendida la estructura interna del *plugin*, es el momento de introducir el concepto de *socket* brevemente antes de pasar a su uso en este trabajo.

Un *socket* [20] es un concepto abstracto por el cual dos procesos, que pueden estar en la misma o en distintas computadoras, pueden intercambiar cualquier flujo de datos de manera fiable y ordenada. Para que dos procesos puedan comunicarse es necesario que se puedan localizar, y para ello se necesitan las direcciones IP de las computadoras y los números de puerto por los que se realizará el intercambio de mensajes.

Cuando se tiene este concepto de *socket*, el de comunicación de procesos, la arquitectura que se debe adoptar es la cliente y servidor (usada en este proyecto). El servidor en el *socket* es el encargado de recibir la información del cliente y responder a éste. Para entender mejor el funcionamiento de un *socket* se va a explicar detalladamente los pasos que cumplen al comunicarse:

1. Se pone en funcionamiento el servidor, de manera que éste se encuentra a la espera de la comunicación de un cliente.
2. Cuando el proceso cliente se ejecuta, realizará una petición al servidor y éste último gestionará la respuesta que quiere darle.
3. Tras esto, el cliente recibe la respuesta del servidor.

Existen dos tipos de *socket* dependiendo del protocolo con el que se realiza la conexión:

- *Socket* con protocolo TCP [20]: este protocolo está orientado a la conexión y garantiza la correcta transmisión de ficheros, manteniendo el orden de éstos correctamente. Esto será posible gracias a que cuando llega un paquete de datos, el receptor emite un mensaje de recepción ACK. Este es el usado en el proyecto.

- *Socket* con protocolo UDP [20]: este protocolo no está orientado a la conexión. De hecho, los paquetes de datos pueden viajar en cualquier orden y no se garantiza la correcta llegada de todos.

4.3. Implementación de los *sockets* en el proyecto

Este proyecto ha requerido del uso de *sockets* para la comunicación entre el sistema desarrollado en Python para reconocimiento de gestos y el *plugin* de Eclipse. Sin embargo, el uso de éstos para nuestros objetivos es muy sencillo.

El servidor en este caso se ha implementado en una nueva clase de Eclipse mediante el uso de la librería de Java `java.net.*`. De esta manera, cuando el *plugin* se lanza para su uso, este servidor quedará siempre activo tras crearse con la función `new ServerSocket()`, a la espera de mensajes de clientes. Cuando el servidor en Eclipse recibe un mensaje de un cliente, lo acepta mediante la función `accept()` y lo tratará de la manera oportuna. Tras examinar el mensaje, cerrará la conexión con el cliente sin ser necesario que éste último obtenga una respuesta.

Por el otro lado, los clientes se han implementado en Python dentro del código de reconocimiento final, mediante el uso de la librería `socket`[19].

[19] https://docs.python.org/es/3/howto/sockets.html

Cada vez que el sistema reconoce el gesto que se encuentra realizando el usuario cada 0,5 segundos (15 *frames*) se crea un nuevo cliente que se conectará con el servidor anterior mediante la función `connect((host,port))`, siendo ambos parámetros el *host* (en este caso `localhost` ya que ambos procesos se encuentran en la misma computadora) y el puerto que se ha usado para el intercambio de mensajes, respectivamente. Una vez conectado, éste enviará al servidor cuál ha sido el resultado del reconocimiento en ese instante mediante la función `send()`. `close()` será la función que cierre cada cliente tras enviar su mensaje.

Los mensajes que mandan los clientes son cadenas de caracteres que contienen el nombre del gesto que se ha reconocido en ese instante. Con esto, cuando el servidor recibe en Eclipse estas cadenas, en función del gesto ordenará la acción correspondiente en el código del *plugin* dentro de esta nueva clase.

4.4. Uso de `Thread` para trabajo en paralelo

Para que todo el sistema de *sockets* descrito anteriormente funcione de manera correcta, es totalmente necesario que la clase en la que se implementa el servidor y en la que se tratan los mensajes con los gestos reconocidos de los clientes se ejecute de forma paralela con el resto del funcionamiento del *plugin*. Esto será posible gracias al uso de la clase `Thread`[20] de Java.

Con la extensión de esta clase se consigue que desde la clase `CodeView`, que pone en marcha todas las funcionalidades del *plugin*, al crear el hilo y llamar a la función `start()`, todo el sistema de *sockets* que se encuentra en el método `run()` de la nueva clase se ejecute en este nuevo hilo, sin interrumpir la ejecución paralela del resto del *plugin* en ningún momento.

Así, en caso de que el reconocedor de gestos detecte un gesto en particular, la funcionalidad que correspondiera a ese gesto se realizaría también desde este nuevo hilo. La llamada a cada funcionalidad en caso de la detección de un gesto se llevará a cabo mediante llamadas al método correspondiente de la clase `CodeControl` para el aumento y decremento de fuente y la síntesis por voz, y mediante el tratamiento de las dos fases comentado al final de la sección 4.1 para el reconocimiento de voz (reconocedor activado o desactivado).

Esta es una buena forma de que no se encuentren errores de confusión ni se interrumpa ninguna de las funcionalidades.

[20] https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

5. Conclusiones.

Una vez finalizado el desarrollo del software y su explicación en la memoria se pueden extraer varias conclusiones.

En el apartado 1.2 se proponen unos objetivos claros para este proyecto: el desarrollo de un sistema de reconocimiento de gestos con una precisión adecuada y su correcta implementación en el *plugin* COPS. A pesar de que la calidad de los resultados no es perfecta, el balance general del proyecto es positivo, ya que estos objetivos se han alcanzado con éxito.

Para esto, en primer lugar se ha construido un *dataset* desde cero, enfocado a la detección de gestos faciales a partir de videos realizados a varios usuarios. A pesar de no ser un *dataset* de gran tamaño, se puede considerar un conjunto de datos importante para nuestro objetivo. Por otro lado, gracias a este *dataset* se ha podido desarrollar el sistema de modelos de Markov para cada uno de los gestos, el cual permite posteriormente para una muestra, conocer qué gesto se ha realizado en ésta exactamente. Como se ha podido ver en la sección 3.7, la precisión de este sistema no es del 100%, pero sí la suficiente para que un buen tratamiento de los resultados permita el correcto funcionamiento del *plugin* COPS una vez se ha añadido esta ampliación mediante el uso de *sockets*.

Se puede concluir respecto a esto que con un mayor *dataset*, con una cantidad mayor de videos de entrenamiento para cada gesto, la precisión del sistema se incrementaría, gracias a que los *scores* resultantes serían más precisos.

También se ha llegado a la conclusión mediante el periodo de prueba del proyecto de que, dependiendo de las características de la computadora en la que se use el *plugin* ampliado, el sistema de reconocimiento será más o menos lento. Se ha podido comprobar que el procesamiento del video a reconocer determina cómo de rápido se detectará la realización de un gesto u otro. Sin embargo, la precisión del sistema aquí no se ve afectada, ya que, a pesar de un reconocimiento más tardío, el resultado es el mismo.

Por último, es destacable comentar que, antes de comenzar con este *plugin,* es necesario que el usuario que vaya a hacer uso de él aprenda y se familiarice con los gestos requeridos. A pesar de que se ha tenido en cuenta la facilidad de estos gestos, antes de iniciar su experiencia programando, alguna persona que conozca la aplicación debería ayudar y enseñar al nuevo usuario. Se ha llegado a la conclusión de que los gestos de levantar las cejas y cerrar los ojos son relativamente sencillos a todo el mundo; pero, sin embargo, los guiños no son tan familiares, debido a que éstos incluyen el pequeño movimiento del labio que se ha comentado en la sección 3.3.3. Una vez el programador conoce todos los gestos, el uso del *plugin* no supondrá ningún tipo de problema ni confusión.

5.1. Relación del proyecto con los estudios cursados

A lo largo del grado se han estudiado materias que han permitido y ayudado al desarrollo de este trabajo. Todas ellas han aportado al proyecto de alguna forma, desde asentar los conocimientos básicos de la informática hasta conceder las bases en inteligencia artificial y *machine learning*. Sin embargo, cabe destacar algunas de ellas por una mayor aportación.

En primer lugar, se destacan asignaturas como IIP (Introducción a la Programación), PRG (Programación), EDA (Estructuras de Datos) y ALT (Algorítmica), las cuales han permitido un gran avance en el aprendizaje del alumno tanto en la interpretación como en la generación código, además de en la gestión de las distintas estructuras de datos disponibles para el programador. Por ello, se podría decir que han sido una de las bases primordiales a la hora de generar y modificar los programas que se han empleado a lo largo del proyecto.

En segundo lugar, APR (Aprendizaje Automático) y PER (Percepción) han establecido las bases del aprendizaje automático, permitiendo al alumno familiarizarse con los procedimientos que conllevan este tipo de proyectos. No obstante, se destaca PER debido a que ha sido la fuente principal para la consolidación de los principios del tratamiento de la información que nutrirán los sistemas de reconocimiento, entre otros aspectos.

CPA (Computación Paralela), por su parte, ha sido la encargada de dar a conocer el posible uso de hilos y ejecución de procesos en paralelo.

Respecto al uso de los modelos ocultos de Markov (HMM), se destaca el papel de SIN (Sistemas Inteligentes), que dió a conocer estos modelos junto a las bases estadísticas relacionadas.

Por último, es necesario destacar RED (Redes) y TSR (Tecnologías de sistemas de información en la red) que, a pesar de no ser asignaturas directamente relacionadas con la rama de Computación, han proporcionado los conocimientos necesarios para el uso de *sockets*, encargados de la comunicación de procesos en este proyecto.

Respecto a las competencias transversales, durante la realización de este proyecto se han consolidado aquellas relacionadas con la ”Comunicación efectiva”, “Diseño o proyecto”, “Conocimiento de problemas contemporáneos” o “Comprensión e integración” entre otras.

6. Trabajos futuros.

Una vez visto todo el desarrollo realizado y las conclusiones extraídas del mismo, se comentan posibles trabajos y ampliaciones futuras que se podrían aplicar a este proyecto y a COPS.

La primera de ellas podría tratarse de la integración de un funcionamiento conjunto del manejo mediante botones y gestos en el uso del reconocimiento de voz. Actualmente, la aplicación únicamente es capaz de poner en marcha y terminar el reconocimiento de voz con el mismo mecanismo, es decir, si el reconocimiento de voz se inicia con el botón correspondiente, éste deberá ser terminado con el mismo, y lo mismo ocurre si se iniciara mediante el gesto de levantar las cejas, habría que pararlo con este mismo gesto. Por ello, la idea sería que el *plugin* admitiese el uso conjunto de ambos mecanismos, por ejemplo, iniciar el reconocimiento con el gesto y posteriormente terminarlo pulsando el botón, o viceversa.

Otra ampliación que podría resultar interesante sería implementar el uso del menú de preferencias para elección de fuente y otras configuraciones también mediante el uso de gestos faciales.

En cuanto a COPS, aunque con los avances conseguidos hasta el momento resulta muy cómoda, se podrían añadir más botones que tengan acceso más directo a algunas funcionalidades, como por ejemplo sería el de ejecutar el proyecto, o simplemente para cambiar la configuración de fuente de forma más sencilla y directa. Este paso ayudaría a aquellas personas con diversidad funcional a, con el menor número de movimientos posibles, poder acceder a cualquiera de las funcionalidades que este *plugin* ofrece.

Por último, teniendo en cuenta las conclusiones obtenidas en el capítulo 5, sería de gran ayuda ampliar el *plugin* con una pequeña prueba de aprendizaje inicial, la cual apareciera al iniciar COPS con el objetivo de enseñar al usuario a realizar de forma correcta cada uno de los gestos especificados en este proyecto. Esta funcionalidad contaría con varias fases que el usuario debería de ir superando, en las cuales aparece una pequeña descripción del gesto a realizar y el impacto que éste tiene en la aplicación. La persona realizaría el gesto hasta que el sistema lo reconociera como correcto. Con esto, el objetivo es que el usuario se familiarice con los gestos y aprenda su correcta realización antes de adentrarse al uso del *plugin*.

Bibliografía

[1] CARLOS-D. MARTÍNEZ-HINAREJOS, SANTIAGO SÁNCHEZ-ALEPUZ AND NATIVIDAD PRIETO-SÁEZ. COPS: A COMPUTER PROGRAMMING TOOL TO COPE WITH FUNCTIONAL DIVERSITY. PROCEEDINGS OF IBERSPEECH 2012, PP. 371-376, UNIVERSIDAD AUTÓNOMA DE MADRID, MADRID, 2012.

[2] MÍRIAM LUJÁN-MARES, VICENT TAMARIT, VICENT ALABAU, CARLOS-D. MARTÍNEZ-HINAREJOS, MOISÉS PASTOR, ALBERTO SANCHIS, AND ALEJANDRO TOSELLI. IATROS: A SPEECH AND HANDWRITTING RECOGNITION SYSTEM. V JORNADAS EN TECNOLOGÍAS DEL HABLA (VJTH'2008), PP. 75-78, BILBAO, NOV 2008.

[3] IVAN E. SUTHERLAND. SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM. MASSACHUSETTS INSTITUTE OF TECHNOLOGY, VOL. 2, Nº 5, PP. 507-524, 1963.

[4] P. EKMAN AND W.V. FRIESEN. THE FACIAL ACTION CODING SYSTEM: A TECHNIQUE FOR THE MEASUREMENT OF FACIAL MOVEMENT. SAN FRANCISCO: CONSULTING PSYCHOLOGISTS PRESS, 1978.

[5] JACOB WHITEHILL AND CHRISTIAN W. OMLIN. HAAR FEATURES FOR FACS AU RECOGNITION. PROC. IEEE INT'L CONF. FACE AND GESTURE RECOGNITION, 5 PP. 2006.

[6] GUEVARA, M., ECHEVERRY, J., & URUEÑA, W. (2008). DETECCIÓN DE ROSTROS EN IMÁGENES DIGITALES USANDO CLASIFICADORES EN CASCADA. SCIENTIA ET TECHNICA, VOL. 1, N.º 38, 6 PP. JUN. 2008.

[7] R. SANTIAGO MONTERO, J. M. LÓPEZ MÁRQUEZ, AND M. ORNELAS RODRÍGUEZ. APLICANDO FILTROS DE GABOR A SEGMENTACIÓN DE CARACTERES ALFANUMÉRICOS EN PLACAS VEHICULARES. CIINDET, 4 PP. MÉXICO, 2011.

[8] AURIA, LAURA AND MORO, R. A., SUPPORT VECTOR MACHINES (SVM) AS A TECHNIQUE FOR SOLVENCY ANALYSIS (AUGUST 1, 2008). DIW BERLIN DISCUSSION PAPER Nº 811.

[9] G. DONATO, M.S. BARTLETT, J. C. HAGER, P. EKMAN, AND T.J. SEJNOWSKI. CLASSIFYING FACIAL ACTIONS. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. VOL 21, Nº 10, PP. 974-989, 1999.

[10] M. BARTLETT, G. LITTLEWORT, B. BRAATHEN, T. SEJNOWSKI, AND J. MOVELLAN. A PROTOTYPE FOR AUTOMATIC RECOGNITION OF SPONTANEOUS FACIAL ACTIONS. IN S. BECKER AND K. OBERMAYER, EDITORS, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, VOL 15. MIT PRESS, 2003.

[11] IVANA BARBONA, AND CELINA BELTRÁN. APLICACIÓN DEL ALGORITMO BOOSTING ADAPTATIVO (ADABOOST) A UN PROBLEMA DE CLASIFICACIÓN AUTOMÁTICA DE TEXTOS. REVISTA DE EPISTEMOLOGÍA Y CIENCIAS HUMANAS, PP. 37-44, 2018.

[12] P. VIOLA AND M. JONES. ROBUST REAL-TIME OBJECT DETECTION. INTERNATIONAL JOURNAL OF COMPUTER VISION, VOL. 57, Nº. 2, PP. 137-154. 2004.

[13] M.J.L. BOADA, BEATRIZ LÓPEZ BOADA, AND VICENTE DÍAZ LÓPEZ. ALGORITMO DE APRENDIZAJE POR REFUERZO CONTINUO PARA EL CONTROL DE UN SISTEMA DE SUSPENSIÓN SEMIACTIVA. REVISTA IBEROAMERICANA DE INGENIERÍA MECÁNICA. VOL. 9, Nº 2, PP. 77-91, 2005.

[14] SCHAPIRE, R AND FREUND, Y. A DECISION THEORETIC GENERALIZATION OF ON-LINE LEARNING AND APPLICATION TO BOOSTING. AT&T BELL LABORATORIES. USA, 1995.

[15] S. SHAIK, R. CORVIN, R. SUDARSAN, F. JAVED, Q. IJAZ, S. ROYCHOUDHURY, J. GRAY, AND B. R. BRYANT. SPEECHCLIPSE: AN ECLIPSE SPEECH PLUG-IN. IN OOPSLA WORKSHOP ON ECLIPSE TECHNOLOGY EXCHANGE, PP. 84-88, 2003.

[16] RODRÍGUEZ, F., & BAUTISTA, S. (2007). MODELOS OCULTOS DE MARKOV PARA EL ANÁLISIS DE PATRONES ESPACIALES. ECOSISTEMAS, VOL. 15, Nº 3, PP. 68-75, 2006.

[17] GARY D. BRUSHE, ROBERT E. MAHONY, AND JOHN B. MOORE. A FORWARD BACKWARD ALGORITHM FOR ML STATE AND SEQUENCE ESTIMATION. INTERNATIONAL SYMPOSIUM ON SIGNAL PROCESSING AND ITS APPLICATIONS, ISSPA, PP. 224-227, GOLD COAST, AUSTRALIA, 25-30 AUGUST,1996.

[18] RICARDO ANTONIO MENDOZA LEÓN. MÉTODOS DE INFERENCIA ESTADÍSTICA PARA ENTRENAMIENTO DE MODELOS OCULTOS DE MARKOV. PÁG. 60, REVISTA ELEMENTOS, Nº 1, JUNIO DE 2011.

[19] ANDRÉS FONT VICEDO. EXTENSIÓN DE FUNCIONALIDADES DE UNA APLICACIÓN PARA LA PROGRAMACIÓN EN JAVA PARA PERSONAS CON DIVERSIDAD FUNCIONAL. TRABAJO DE FIN DE GRADO. GRADO EN INGENIERÍA INFORMÁTICA. UNIVERSIDAD POLITÉCNICA DE VALENCIA. 2020.

[20] LIMI KALITA. SOCKET PROGRAMMING. (IJCSIT) INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES, VOL. 5, Nº 3, PP. 4802-4807, 2014.

Anexo

En el anexo de este proyecto se encontrará tanto el código empleado para el sistema de reconocimiento de gestos faciales como la nueva clase creada para su inclusión en COPS. El objetivo de este apartado es poder consultar el código para entender mejor la explicación desarrollada en los capítulos 3 y 4.

• SISTEMA DE RECONOCIMIENTO DE GESTOS FACIALES EN PYTHON

- Ejemplo de generador de *dataset* a partir de los videos de entrenamiento.

Aquí vemos un ejemplo de generador de archivos para el *dataset*, en este caso para el gesto de guiño izquierdo. La estructura para los demás gestos sería exactamente igual.

```
import cv2
import numpy as np
import dlib

for i in range(1,34):

    # Recorremos todos los videos de entrenamiento para sacar el txt
de cada uno
    # Este es el ejemplo de recorrer todos los videos del gesto guiño
izquierdo.
    cap = cv2.VideoCapture("videosentrenamiento/guiño_izquierdo" +
str(i) + ".mp4")

    detector = dlib.get_frontal_face_detector()
    predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

    puntosClaveGI = []

    # Archivo de texto del dataset
    txtGI = open("txt/datosGI/datosGI" + str(i) + ".txt", "w")

    while True:
        try:
            _, frame = cap.read()
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            faces = detector(gray)
            for face in faces:
                x1 = face.left()
                y1 = face.top()
                x2 = face.right()
                y2 = face.bottom()
```

```
                landmarks = predictor(gray, face)
                myPoints = [0] * 68
                for n in range(0, 68):
                    x = landmarks.part(n).x
                    y = landmarks.part(n).y
                    myPoints[n] = [x,y]
                    cv2.circle(frame, (x, y), 3, (0, 255, 0), -1)
                    cv2.putText(frame,str(n),(x,y-10),
cv2.FONT_HERSHEY_COMPLEX_SMALL,0.8,(0,255,0), 1)

                # Para tener mejor situados los puntos clave de la
cara
                  ojoD = myPoints[36:42]
                cejaD = myPoints[17:22]
                ojoI = myPoints[42:48]
                cejaI = myPoints[22:27]
                # Al ser modo espejo el derecho aparece en la
izquierda y vicebersa.
                nariz = myPoints[27:36]
                bocaExt = myPoints[48:60]
                bocaInt = myPoints[60:68]
                mandibula = myPoints[0:17]

                  # Para la normalización como se ha comentado en el
capítulo 3.
                normY = myPoints[27][1] - myPoints[8][1]
                normX = myPoints[16][0] - myPoints[0][0]

                # Características comunes

                distOjoI1 = (myPoints[43][1] - myPoints[47][1])/normY
                distOjoI1 = round(distOjoI1, 5)
                distOjoI2 = (myPoints[44][1] - myPoints[46][1])/normY
                distOjoI2 = round(distOjoI2, 5)
                distOjoD1 = (myPoints[37][1] - myPoints[41][1])/normY
                distOjoD1 = round(distOjoD1, 5)
                distOjoD2 = (myPoints[38][1] - myPoints[40][1])/normY
                distOjoD2 = round(distOjoD2, 5)

                distOjoBocaI1 = (myPoints[44][1] -
myPoints[54][1])/normY
                distOjoBocaI1 = round(distOjoI2, 5)
                distOjoBocaD1 = (myPoints[37][1] -
myPoints[48][1])/normY
                distOjoBocaD1 = round(distOjoD1, 5)

                distOjoNarizI1 = (myPoints[44][1] -
myPoints[30][1])/normY
                distOjoNarizI1 = round(distOjoI2, 5)
                distOjoNarizD1 = (myPoints[37][1] -
myPoints[30][1])/normY
                distOjoNarizD1 = round(distOjoD1, 5)

                distCejaNarizI1 = (myPoints[24][1] -
myPoints[30][1])/normY
                distCejaNarizI1 = round(distCejaNarizI1, 5)
                distCejaNarizD1 = (myPoints[19][1] -
myPoints[30][1])/normY
                distCejaNarizD1 = round(distCejaNarizD1, 5)
```

```
                distCejaOjoI1 = (myPoints[24][1] -
myPoints[46][1])/normY
                distCejaOjoI1 = round(distCejaOjoI1, 5)
                distCejaOjoD1 = (myPoints[19][1] -
myPoints[41][1])/normY
                distCejaOjoD1 = round(distCejaOjoD1, 5)

                distCejaOjoI3 = (myPoints[22][1] -
myPoints[42][1])/normY
                distCejaOjoI3 = round(distCejaOjoI3, 5)
                distCejaOjoD3 = (myPoints[21][1] -
myPoints[39][1])/normY
                distCejaOjoD3 = round(distCejaOjoD3, 5)

                distCejaOjoI2 = (myPoints[24][1] -
myPoints[44][1])/normY
                distCejaOjoI2 = round(distCejaOjoI2, 5)
                distCejaOjoD2 = (myPoints[19][1] -
myPoints[37][1])/normY
                distCejaOjoD2 = round(distCejaOjoD2, 5)

                distLabioNarizI1 = (myPoints[54][0] -
myPoints[27][0])/normX
                distLabioNarizI1 = round(distLabioNarizI1, 5)
                distLabioNarizD1 = (myPoints[27][0] -
myPoints[48][0])/normX
                distLabioNarizD1 = round(distLabioNarizD1, 5)

                  # Formato seguido para los documentos de texto del
dataset
                puntosClaveGI = [distLabioNarizI1, " ",
distLabioNarizD1, " ", distOjoI1, " ", distOjoI2, " ", distOjoD1, " ",
distOjoD1, " ", distCejaOjoI1, " ", distCejaOjoD1, " ", distCejaOjoI2,
" ", distCejaOjoD2, " ", distCejaOjoI3, " ", distCejaOjoD3, " ",
distCejaNarizI1, " ", distCejaNarizD1, " ", distOjoNarizI1, " ",
distOjoNarizD1, " ", distOjoBocaI1, " ", distOjoBocaD1]

                txtGI.write("".join(map(str,puntosClaveGI)))
                txtGI.write("\n")
                cv2.imshow("Frame", frame)
                key = cv2.waitKey(1)
                if key == 27:
                    break

        except:
            break
```

- **Entrenador de modelos a partir de los documentos de texto del *dataset*.**

```
import numpy as np
import pickle
from hmmlearn import hmm
from numpy.random import default_rng
import cv2
import dlib

np.random.seed(42)

cerrar_ojos = hmm.GaussianHMM(n_components=3, covariance_type="full")
cerrar_ojos.n_features = 18

levantar_cejas = hmm.GaussianHMM(n_components=3,
covariance_type="full")
levantar_cejas.n_features = 18

guiño_izquierdo = hmm.GaussianHMM(n_components=3,
covariance_type="full")
guiño_izquierdo.n_features = 18

guiño_derecho = hmm.GaussianHMM(n_components=3,
covariance_type="full")
guiño_derecho.n_features = 18

normal = hmm.GaussianHMM(n_components=3, covariance_type="full")
normal.n_features = 18

Sco = []
Slc = []
Sgd = []
Sgi = []
Sno = []

for i in range(1,41):

    if i < 34:
        # Guiño izquierdo
        f = open('txt/datosGI/datosGI' + str(i) +'.txt', 'r')
        csGI = []
        for s in f:
            csGI.insert(len(csGI)+1,[float(x) for x in s.split()])
        f.close()
        Sgi.insert(len(Sgi)+1, csGI)

    if i < 32:
        # Guiño derecho
        f = open('txt/datosGD/datosGD' + str(i) +'.txt', 'r')
        csGD = []
        for s in f:
            csGD.insert(len(csGD)+1,[float(x) for x in s.split()])
        f.close()
        Sgd.insert(len(Sgd)+1, csGD)

    if i < 35:
        # Cerrar ojos
        f = open('txt/datosCO/datosCO' + str(i) +'.txt', 'r')
        csCO = []
        for s in f:
```

```
            csCO.insert(len(csCO)+1,[float(x) for x in s.split()])
        f.close()
        Sco.insert(len(Sco)+1, csCO)

    if i < 33:
        # Normal
        f = open('txt/datosNO/datosNO' + str(i) +'.txt', 'r')
        csNO = []
        for s in f:
            csNO.insert(len(csNO)+1,[float(x) for x in s.split()])
        f.close()
        Sno.insert(len(Sno)+1, csNO)

    if i < 34:
        # Levantar cejas
        f = open('txt/datosLC/datosLC' + str(i) +'.txt', 'r')
        csLC = []
        for s in f:
            csLC.insert(len(csLC)+1,[float(x) for x in s.split()])
        f.close()
        Slc.insert(len(Slc)+1, csLC)

Xco = np.concatenate([ s for s in Sco ])
Xlc = np.concatenate([ s for s in Slc ])
Xgi = np.concatenate([ s for s in Sgi ])
Xgd = np.concatenate([ s for s in Sgd ])
Xno = np.concatenate([ s for s in Sno ])

cerrar_ojos.fit(Xco)
levantar_cejas.fit(Xlc)
guiño_izquierdo.fit(Xgi)
guiño_derecho.fit(Xgd)
normal.fit(Xno)

with open("hmm/cerrar_ojos.hmm", "wb") as file:
pickle.dump(cerrar_ojos, file)
with open("hmm/levantar_cejas.hmm", "wb") as file:
pickle.dump(levantar_cejas, file)
with open("hmm/guiño_izquierdo.hmm", "wb") as file:
pickle.dump(guiño_izquierdo, file)
with open("hmm/guiño_derecho.hmm", "wb") as file:
pickle.dump(guiño_derecho, file)
with open("hmm/normal.hmm", "wb") as file: pickle.dump(normal, file)
```

- **Decodificación y obtención de *scores* de videos de prueba a partir de los modelos entrenados con el uso de *sockets* necesario para la comunicación con Eclipse.**

```
import numpy as np
import pickle
from hmmlearn import hmm
from numpy.random import default_rng
import cv2
import dlib
import socket

host = "localhost"
port = 50006

np.random.seed(42)

with open("hmm/cerrar_ojos.hmm", "rb") as file: cerrar_ojos =
pickle.load(file)
with open("hmm/levantar_cejas.hmm", "rb") as file: levantar_cejas =
pickle.load(file)
with open("hmm/guino_izquierdo.hmm", "rb") as file: guiño_izquierdo =
pickle.load(file)
with open("hmm/guino_derecho.hmm", "rb") as file: guiño_derecho =
pickle.load(file)
with open("hmm/normal.hmm", "rb") as file: normal = pickle.load(file)

cap = cv2.VideoCapture(0)

detector = dlib.get_frontal_face_detector()
predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

puntosClaveMP = []
# Iniciamos el array como uno de los videos de normal que tenemos para
entrenamiento.
csMP = [[0.16475, 0.14176, 0.06009, 0.06009, 0.06867, 0.06867,
0.22747, 0.23176, 0.16738, 0.16309, 0.15451, 0.15451, 0.48069,
0.48498, 0.06009, 0.06867, 0.06009, 0.06867], [0.17176, 0.14504,
0.05957, 0.05957, 0.06383, 0.06383, 0.22553, 0.22553, 0.16596, 0.1617,
0.15319, 0.14894, 0.4766, 0.4766, 0.05957, 0.06383, 0.05957, 0.06383],
[0.17176, 0.1374, 0.06438, 0.06438, 0.06867, 0.06867, 0.23176,
0.23176, 0.16738, 0.16309, 0.15451, 0.15451, 0.47639, 0.47639,
0.06438, 0.06867, 0.06438, 0.06867], [0.17308, 0.14231, 0.06438,
0.06438, 0.06867, 0.06867, 0.23176, 0.23176, 0.16738, 0.16309,
0.15451, 0.15451, 0.47639, 0.48069, 0.06438, 0.06867, 0.06438,
0.06867], [0.17241, 0.1341, 0.06009, 0.06438, 0.06438, 0.06438,
0.22747, 0.22747, 0.16309, 0.16309, 0.15451, 0.1588, 0.47639, 0.47639,
0.06438, 0.06438, 0.06438, 0.06438], [0.1673, 0.13308, 0.06009,
0.06009, 0.06867, 0.06867, 0.22747, 0.23176, 0.16738, 0.16309,
0.15451, 0.15451, 0.47639, 0.48069, 0.06009, 0.06867, 0.06009,
0.06867], [0.1673, 0.13308, 0.0641, 0.05983, 0.0641, 0.0641, 0.2265,
0.2265, 0.16667, 0.16239, 0.16239, 0.15385, 0.47436, 0.47436, 0.05983,
0.0641, 0.05983, 0.0641], [0.1711, 0.12928, 0.05957, 0.05957, 0.05957,
0.05957, 0.22553, 0.22128, 0.16596, 0.1617, 0.15319, 0.15319, 0.4766,
0.4766, 0.05957, 0.05957, 0.05957, 0.05957], [0.17557, 0.12595,
0.05983, 0.0641, 0.0641, 0.0641, 0.2265, 0.2265, 0.16239, 0.16239,
0.15385, 0.15385, 0.47436, 0.47436, 0.0641, 0.0641, 0.0641, 0.0641],
[0.17241, 0.1341, 0.0641, 0.0641, 0.0641, 0.0641, 0.23077, 0.2265,
0.16667, 0.16239, 0.15812, 0.15812, 0.47436, 0.47436, 0.0641, 0.0641,
```

```
0.0641, 0.0641], [0.17045, 0.13258, 0.05957, 0.05957, 0.06383,
0.06383, 0.22553, 0.22553, 0.16596, 0.1617, 0.15745, 0.15319, 0.4766,
0.4766, 0.05957, 0.06383, 0.05957, 0.06383], [0.17241, 0.13027,
0.05983, 0.05983, 0.0641, 0.0641, 0.2265, 0.2265, 0.16667, 0.16239,
0.15385, 0.15812, 0.47436, 0.47436, 0.05983, 0.0641, 0.05983, 0.0641],
[0.17241, 0.13793, 0.05983, 0.0641, 0.06838, 0.06838, 0.2265, 0.2265,
0.16239, 0.15812, 0.15385, 0.15812, 0.47436, 0.47436, 0.0641, 0.06838,
0.0641, 0.06838], [0.17625, 0.13027, 0.06009, 0.06867, 0.06867,
0.06867, 0.23176, 0.22747, 0.16309, 0.1588, 0.15021, 0.1588, 0.47639,
0.47639, 0.06867, 0.06867, 0.06867, 0.06867], [0.16794, 0.1374,
0.05983, 0.0641, 0.0641, 0.0641, 0.23077, 0.2265, 0.16667, 0.16239,
0.15385, 0.15812, 0.47436, 0.47436, 0.0641, 0.0641, 0.0641, 0.0641],
[0.17625, 0.13793, 0.06438, 0.06009, 0.06438, 0.06438, 0.22747,
0.22747, 0.16738, 0.16309, 0.1588, 0.1588, 0.47639, 0.47639, 0.06009,
0.06438, 0.06009, 0.06438], [0.1711, 0.13308, 0.05983, 0.05983,
0.0641, 0.0641, 0.2265, 0.2265, 0.16667, 0.16239, 0.15385, 0.15385,
0.47436, 0.47436, 0.05983, 0.0641, 0.05983, 0.0641], [0.1673, 0.13688,
0.05983, 0.05983, 0.06838, 0.06838, 0.2265, 0.2265, 0.16667, 0.15812,
0.15385, 0.15385, 0.47436, 0.47436, 0.05983, 0.06838, 0.05983,
0.06838], [0.1673, 0.13308, 0.0641, 0.06838, 0.0641, 0.0641, 0.23077,
0.2265, 0.16239, 0.16239, 0.15812, 0.15385, 0.47863, 0.47863, 0.06838,
0.0641, 0.06838, 0.0641], [0.17176, 0.13359, 0.06438, 0.06438,
0.06438, 0.06438, 0.23176, 0.22747, 0.16738, 0.16309, 0.15021,
0.15451, 0.48069, 0.48069, 0.06438, 0.06438, 0.06438, 0.06438],
[0.1635, 0.13688, 0.06466, 0.06466, 0.06466, 0.06466, 0.22845,
0.22845, 0.16379, 0.16379, 0.15948, 0.15517, 0.47845, 0.48276,
0.06466, 0.06466, 0.06466, 0.06466], [0.17308, 0.13846, 0.06466,
0.06466, 0.06466, 0.06466, 0.23276, 0.22845, 0.1681, 0.16379, 0.15517,
0.15517, 0.48276, 0.48707, 0.06466, 0.06466, 0.06466, 0.06466],
[0.16794, 0.1374, 0.0641, 0.05983, 0.0641, 0.0641, 0.2265, 0.2265,
0.16667, 0.16239, 0.14957, 0.15385, 0.47436, 0.48291, 0.05983, 0.0641,
0.05983, 0.0641], [0.16923, 0.15, 0.06061, 0.06061, 0.06494, 0.06494,
0.22944, 0.22944, 0.16883, 0.1645, 0.15152, 0.16017, 0.48052, 0.48918,
0.06061, 0.06494, 0.06061, 0.06494], [0.16412, 0.14504, 0.06438,
0.06438, 0.06867, 0.06867, 0.22747, 0.23176, 0.16309, 0.16309,
0.15021, 0.15021, 0.4721, 0.48498, 0.06438, 0.06867, 0.06438,
0.06867], [0.1597, 0.14449, 0.06466, 0.06466, 0.06466, 0.06466,
0.22845, 0.22845, 0.16379, 0.16379, 0.15517, 0.15517, 0.48276,
0.49569, 0.06466, 0.06466, 0.06466, 0.06466], [0.15849, 0.14717,
0.06061, 0.06494, 0.06494, 0.06494, 0.22511, 0.22944, 0.16017, 0.1645,
0.15584, 0.15584, 0.48052, 0.49351, 0.06494, 0.06494, 0.06494,
0.06494], [0.15909, 0.14394, 0.06061, 0.06494, 0.06061, 0.06061,
0.22511, 0.22511, 0.16017, 0.1645, 0.15152, 0.15584, 0.47619, 0.49351,
0.06494, 0.06061, 0.06494, 0.06061], [0.15849, 0.1434, 0.06061,
0.06061, 0.06061, 0.06061, 0.22511, 0.22511, 0.1645, 0.1645, 0.15152,
0.15584, 0.48485, 0.49784, 0.06061, 0.06061, 0.06061, 0.06061],
[0.1553, 0.14773, 0.06466, 0.06466, 0.06466, 0.06466, 0.22845,
0.22845, 0.16379, 0.16379, 0.15517, 0.15948, 0.48276, 0.49569,
0.06466, 0.06466, 0.06466, 0.06466], [0.15472, 0.15094, 0.06034,
0.06466, 0.06466, 0.06466, 0.22414, 0.22414, 0.15948, 0.15948,
0.15086, 0.15517, 0.48276, 0.49569, 0.06466, 0.06466, 0.06466,
0.06466], [0.15909, 0.14773, 0.06034, 0.06466, 0.06466, 0.06466,
0.22414, 0.22845, 0.15948, 0.16379, 0.15517, 0.15517, 0.47845,
0.49569, 0.06466, 0.06466, 0.06466, 0.06466], [0.1553, 0.15152,
0.06061, 0.06494, 0.06494, 0.06494, 0.22944, 0.22511, 0.1645, 0.16017,
0.15152, 0.15152, 0.48485, 0.49784, 0.06494, 0.06494, 0.06494,
0.06494], [0.1553, 0.15152, 0.06061, 0.06926, 0.06494, 0.06494,
0.22511, 0.22944, 0.15584, 0.1645, 0.14719, 0.15152, 0.48052, 0.49784,
0.06926, 0.06494, 0.06926, 0.06494], [0.15909, 0.14773, 0.06034,
0.06897, 0.06466, 0.06466, 0.22845, 0.22845, 0.15948, 0.16379,
0.15086, 0.15517, 0.48276, 0.49569, 0.06897, 0.06466, 0.06897,
```

0.06466], [0.15589, 0.14829, 0.05579, 0.06438, 0.06009, 0.06009,
0.22747, 0.22318, 0.16309, 0.16309, 0.14592, 0.15451, 0.48069,
0.49356, 0.06438, 0.06009, 0.06438, 0.06009], [0.15849, 0.14717,
0.06034, 0.06034, 0.06034, 0.06034, 0.21983, 0.22414, 0.15948,
0.16379, 0.15086, 0.15517, 0.47845, 0.49569, 0.06034, 0.06034,
0.06034, 0.06034], [0.15849, 0.15094, 0.06034, 0.06034, 0.06034,
0.06034, 0.22414, 0.22414, 0.16379, 0.16379, 0.15086, 0.15517,
0.48707, 0.5, 0.06034, 0.06034, 0.06034, 0.06034], [0.15789, 0.14662,
0.05603, 0.06466, 0.06034, 0.06034, 0.22414, 0.22414, 0.15948,
0.16379, 0.15086, 0.15086, 0.48276, 0.5, 0.06466, 0.06034, 0.06466,
0.06034], [0.15472, 0.15094, 0.05556, 0.0641, 0.0641, 0.0641, 0.22222,
0.22222, 0.15812, 0.15812, 0.1453, 0.15812, 0.47863, 0.49145, 0.0641,
0.0641, 0.0641, 0.0641], [0.15414, 0.15038, 0.05983, 0.0641, 0.0641,
0.0641, 0.2265, 0.22222, 0.16239, 0.15812, 0.1453, 0.15385, 0.48291,
0.49573, 0.0641, 0.0641, 0.0641, 0.0641], [0.15094, 0.15094, 0.06009,
0.06438, 0.06438, 0.06438, 0.22747, 0.22747, 0.16309, 0.16309,
0.14592, 0.15021, 0.48498, 0.50215, 0.06438, 0.06438, 0.06438,
0.06438], [0.15789, 0.14662, 0.06466, 0.06466, 0.06466, 0.06466,
0.22414, 0.22414, 0.15948, 0.15948, 0.15086, 0.15086, 0.48276,
0.49569, 0.06466, 0.06466, 0.06466, 0.06466], [0.15356, 0.14981,
0.06034, 0.06897, 0.06897, 0.06897, 0.22414, 0.22845, 0.15517,
0.15948, 0.15086, 0.15517, 0.48276, 0.5, 0.06897, 0.06897, 0.06897,
0.06897], [0.15849, 0.15094, 0.06466, 0.06466, 0.06897, 0.06897,
0.22414, 0.22414, 0.15948, 0.15517, 0.15086, 0.15086, 0.48276,
0.49569, 0.06466, 0.06897, 0.06466, 0.06897], [0.15472, 0.15094,
0.06034, 0.06466, 0.06466, 0.06466, 0.22414, 0.22845, 0.15948,
0.16379, 0.15086, 0.15948, 0.48276, 0.5, 0.06466, 0.06466, 0.06466,
0.06466], [0.14981, 0.14607, 0.06466, 0.06897, 0.06897, 0.06897,
0.22414, 0.22845, 0.15517, 0.15948, 0.15086, 0.15517, 0.48276, 0.5,
0.06897, 0.06897, 0.06897, 0.06897], [0.15472, 0.15094, 0.06494,
0.06494, 0.06926, 0.06926, 0.22511, 0.22944, 0.16017, 0.16017,
0.14719, 0.16017, 0.48485, 0.50216, 0.06494, 0.06926, 0.06494,
0.06926], [0.15472, 0.15094, 0.06466, 0.06466, 0.06466, 0.06466,
0.22414, 0.22845, 0.15948, 0.16379, 0.15086, 0.15517, 0.47845,
0.49569, 0.06466, 0.06466, 0.06466, 0.06466], [0.15356, 0.14607,
0.06034, 0.06897, 0.06897, 0.06897, 0.22845, 0.22845, 0.15948,
0.15948, 0.15086, 0.15517, 0.48707, 0.5, 0.06897, 0.06897, 0.06897,
0.06897], [0.15356, 0.14607, 0.06009, 0.06867, 0.06867, 0.06867,
0.22318, 0.22747, 0.15451, 0.1588, 0.15021, 0.1588, 0.47639, 0.49356,
0.06867, 0.06867, 0.06867, 0.06867], [0.15038, 0.15038, 0.06034,
0.06466, 0.06034, 0.06034, 0.22414, 0.22414, 0.15948, 0.16379,
0.15517, 0.15517, 0.47845, 0.49569, 0.06466, 0.06034, 0.06466,
0.06034], [0.15038, 0.14662, 0.06034, 0.06897, 0.06897, 0.06897,
0.22414, 0.22845, 0.15517, 0.15948, 0.15086, 0.15517, 0.47845,
0.49569, 0.06897, 0.06897, 0.06897, 0.06897], [0.1573, 0.14232,
0.06009, 0.06438, 0.06438, 0.06438, 0.22747, 0.22747, 0.16309,
0.16309, 0.14592, 0.15021, 0.48069, 0.49356, 0.06438, 0.06438,
0.06438, 0.06438], [0.15472, 0.14717, 0.06438, 0.06438, 0.06438,
0.06438, 0.22747, 0.22747, 0.16309, 0.16309, 0.15021, 0.1588, 0.48069,
0.49356, 0.06438, 0.06438, 0.06438, 0.06438], [0.15789, 0.14662,
0.05603, 0.06466, 0.06466, 0.06466, 0.22414, 0.22845, 0.15948,
0.16379, 0.15086, 0.15517, 0.47845, 0.49569, 0.06466, 0.06466,
0.06466, 0.06466], [0.15849, 0.14717, 0.06009, 0.06438, 0.06438,
0.06438, 0.22318, 0.22747, 0.1588, 0.16309, 0.15021, 0.15451, 0.48069,
0.49785, 0.06438, 0.06438, 0.06438, 0.06438], [0.15909, 0.14394,
0.05983, 0.05983, 0.0641, 0.0641, 0.22222, 0.2265, 0.16239, 0.16239,
0.1453, 0.15385, 0.47863, 0.49573, 0.05983, 0.0641, 0.05983, 0.0641],
[0.15414, 0.15038, 0.06494, 0.06494, 0.06494, 0.06494, 0.22511,
0.22944, 0.16017, 0.1645, 0.15152, 0.16017, 0.48485, 0.50216, 0.06494,
0.06494, 0.06494, 0.06494], [0.15909, 0.15152, 0.06466, 0.06466,

```
0.06466, 0.06466, 0.22414, 0.22414, 0.15948, 0.15948, 0.15086,
0.15948, 0.47845, 0.49138, 0.06466, 0.06466, 0.06466, 0.06466]]

contador = 0

while True:
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = detector(gray)
    for face in faces:
        x1 = face.left()
        y1 = face.top()
        x2 = face.right()
        y2 = face.bottom()

        landmarks = predictor(gray, face)
        myPoints = [0] * 68
        for n in range(0, 68):
            x = landmarks.part(n).x
            y = landmarks.part(n).y
            myPoints[n] = [x,y]
            cv2.circle(frame, (x, y), 3, (0, 255, 0), -1)
            cv2.putText(frame, str(n), (x, y-10),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.8, (0,255,0), 1)

        ojoD = myPoints[36:42]
        cejaD = myPoints[17:22]
        ojoI = myPoints[42:48]
        cejaI = myPoints[22:27]
        # Modo espejo, el derecho aparece en la izquierda y vicebersa.
        nariz = myPoints[27:36]
        bocaExt = myPoints[48:60]
        bocaInt = myPoints[60:68]
        mandibula = myPoints[0:17]

        normY = myPoints[27][1] - myPoints[8][1]
        normX = myPoints[16][0] - myPoints[0][0]

        # CREAR ARRAY PUNTOS CLAVE PROBABILIDADES
        """
        distNarBarbY = (myPoints[30][1] - myPoints[8][1])/normY
        distNarBarbY = round(distNarBarbY, 5)

        distNarizY = (myPoints[30][1] - myPoints[27][1])/normY
        distNarizY = round(distNarizY, 5)
        """
        distOjoI1 = (myPoints[43][1] - myPoints[47][1])/normY
        distOjoI1 = round(distOjoI1, 5)
        distOjoI2 = (myPoints[44][1] - myPoints[46][1])/normY
        distOjoI2 = round(distOjoI2, 5)
        distOjoD1 = (myPoints[37][1] - myPoints[41][1])/normY
        distOjoD1 = round(distOjoD1, 5)
        distOjoD2 = (myPoints[38][1] - myPoints[40][1])/normY
        distOjoD2 = round(distOjoD2, 5)

        distOjoBocaI1 = (myPoints[44][1] - myPoints[54][1])/normY
        distOjoBocaI1 = round(distOjoI2, 5)
        distOjoBocaD1 = (myPoints[37][1] - myPoints[48][1])/normY
        distOjoBocaD1 = round(distOjoD1, 5)
```

```
        distOjoNarizI1 = (myPoints[44][1] - myPoints[30][1])/normY
        distOjoNarizI1 = round(distOjoI2, 5)
        distOjoNarizD1 = (myPoints[37][1] - myPoints[30][1])/normY
        distOjoNarizD1 = round(distOjoD1, 5)

        distCejaNarizI1 = (myPoints[24][1] - myPoints[30][1])/normY
        distCejaNarizI1 = round(distCejaNarizI1, 5)
        distCejaNarizD1 = (myPoints[19][1] - myPoints[30][1])/normY
        distCejaNarizD1 = round(distCejaNarizD1, 5)

        distCejaOjoI1 = (myPoints[24][1] - myPoints[46][1])/normY
        distCejaOjoI1 = round(distCejaOjoI1, 5)
        distCejaOjoD1 = (myPoints[19][1] - myPoints[41][1])/normY
        distCejaOjoD1 = round(distCejaOjoD1, 5)

        distCejaOjoI3 = (myPoints[22][1] - myPoints[42][1])/normY
        distCejaOjoI3 = round(distCejaOjoI3, 5)
        distCejaOjoD3 = (myPoints[21][1] - myPoints[39][1])/normY
        distCejaOjoD3 = round(distCejaOjoD3, 5)

        distCejaOjoI2 = (myPoints[24][1] - myPoints[44][1])/normY
        distCejaOjoI2 = round(distCejaOjoI2, 5)
        distCejaOjoD2 = (myPoints[19][1] - myPoints[37][1])/normY
        distCejaOjoD2 = round(distCejaOjoD2, 5)

        distLabioNarizI1 = (myPoints[54][0] - myPoints[27][0])/normX
        distLabioNarizI1 = round(distLabioNarizI1, 5)
        distLabioNarizD1 = (myPoints[27][0] - myPoints[48][0])/normX
        distLabioNarizD1 = round(distLabioNarizD1, 5)

        puntosClaveMP = [distLabioNarizI1, distLabioNarizD1,
distOjoI1, distOjoI2, distOjoD1, distOjoD1, distCejaOjoI1,
distCejaOjoD1, distCejaOjoI2, distCejaOjoD2, distCejaOjoI3,
distCejaOjoD3, distCejaNarizI1, distCejaNarizD1, distOjoNarizI1,
distOjoNarizD1, distOjoBocaI1, distOjoBocaD1]

        csMP.append(puntosClaveMP)
        csMP.pop(0)

        contador = contador + 1

        if contador == 15:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.connect((host, port))

            # COMPROBAR PROBABILIDADES

            print(" PROBABILIDADES: ")
            print(" --------------- ")

            scoreGI = guiño_izquierdo.score(csMP)
            print(" GUIÑO IZQUIERDO:    " + str(scoreGI))

            scoreGD = guiño_derecho.score(csMP)
            print(" GUIÑO DERECHO:     " + str(scoreGD))

            scoreLC = levantar_cejas.score(csMP)
            print(" LEVANTAR CEJAS: " + str(scoreLC))

            scoreCO = cerrar_ojos.score(csMP)
            print(" CERRAR OJOS:    " + str(scoreCO))
```

```
            scoreNO = normal.score(csMP)
            print(" NORMAL:          " + str(scoreNO))

            scores = [scoreGI, scoreGD, scoreCO, scoreLC, scoreNO]
            winner = max(scores)

            if (max(scores)-min(scores) < 2000) or winner == scoreNO:
                sock.send("NORMAL".encode())
                print("NORMAL")
            elif winner == scoreGI:
                sock.send("GUIÑO IZQUIERDO".encode())
                print("GUIÑO IZQUIERDO")
            elif winner == scoreGD:
                sock.send("GUIÑO DERECHO".encode())
                print("GUIÑO DERECHO")
            elif winner == scoreCO:
                sock.send("CERRAR OJOS".encode())
                print("CERRAR OJOS")
            elif winner == scoreLC:
                sock.send("LEVANTAR CEJAS".encode())
                print("LEVANTAR CEJAS")

            contador = 0
            sock.close()


    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1)
    if key == 27:
        break
```

- **NUEVA CLASE CON THREAD Y *SOCKET* EN JAVA**

```
public class hiloControl extends Thread{

	static Logger logger = Logger.getLogger(hiloControl.class);

	private CodeControl cc;

	private AudioRecorder ar;
	public boolean recogActive;
	private org.eclipse.jface.action.Action recogAction;

	protected ServerSocket ss;
	protected Socket cs;

	public final int port = 50006;
	public boolean VieneDeNormal;

	public hiloControl(CodeControl aux, AudioRecorder araux,
org.eclipse.jface.action.Action recogActionaux) {
		cc = aux;
		ar = araux;
		recogAction = recogActionaux;
		recogActive = false;
	}

	@Override
	public void run() {

		try {
		  ss = new ServerSocket(port);
		} catch (IOException e1) {
		  e1.printStackTrace();
		}

		System.out.println("SERVIDOR CREADO");
		VieneDeNormal = true;

		while(true) {
		  try {
			System.out.println("ESPERANDO");
			cs = ss.accept();
			System.out.println("CLIENTE ENCONTRADO");

			PrintWriter out = new
PrintWriter(cs.getOutputStream(), true);
		    BufferedReader in = new BufferedReader(new
InputStreamReader(cs.getInputStream()));

			String message = in.readLine();
			System.out.println("SERVIDOR RECIBE DE CLIENTE: " +
message);


			if (message.equals("NORMAL")) {
			  VieneDeNormal = true;
			}
			if (message.equals("GUIÑO DERECHO") && VieneDeNormal
== true) {
```

```
                    Display.getDefault().syncExec(new Runnable() {
                      public void run() {
                         cc.increaseFontSize();
                      }
                    });
                    VieneDeNormal = false;
                  }
                  if (message.equals("GUIÑO IZQUIERDO") &&
VieneDeNormal == true) {
                    Display.getDefault().syncExec(new Runnable() {
                      public void run() {
                        cc.decreaseFontSize();
                      }
                    });
                    VieneDeNormal = false;
                  }
                  if (message.equals("CERRAR OJOS") && VieneDeNormal ==
true) {
                    Display.getDefault().syncExec(new Runnable() {
                      public void run() {
                        cc.speaker();
                      }
                    });
                    VieneDeNormal = false;
                  }
                  if (message.equals("LEVANTAR CEJAS")) {
                    Display.getDefault().syncExec(new Runnable() {
                      public void run() {
                        if (recogActive) {
                           logger.info("Stopping recording");
                           recogAction.setImageDescriptor(Activat
or.getImageDescriptor("icons/micro.png"));

                           ar.stopRecording();
                           logger.info("Performing recognition");
                           final String r=ar.performRecognition();
                          logger.info(r);
                           ar.restoreRecording();
                           logger.info("Restoring recogniser");
                           cc.updateText(r);
                           recogActive = false;
                         }
                         else {
                           logger.info("Starting recording");
                           recogAction.setImageDescriptor(Activa
tor.getImageDescriptor("icons/micro2.png"));
                           ar.startRecording();
                           recogActive = true;
                         }
                      }
                    });
                   VieneDeNormal = false;
                  }


                  out.close();
                  in.close();

                  cs.close();

                } catch (IOException e) {
```

```
                        e.printStackTrace();
                    }
                }
            }
}
```