Universidad ORT Uruguay
Facultad de Ingeniería

Sistema de reconocimiento de gestos faciales para aplicaciones de accesibilidad

Entregado como requisito para la obtención del título de Ingeniero en Electrónica

Rachel Schein 180221
Felipe Spoturno 177598

Tutor: André Fonseca

2017

Declaración de Autoría

Nosotros, Rachel Schein y Felipe Spoturno, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el trabajo final de carrera;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Felipe Spoturno | Rachel Schein

18 de Mayo, 2017

Agradecimientos

En primer lugar queremos agradecer a nuestras familias, por el apoyo, el cariño y la presencia incondicional que han tenido con nosotros desde el comienzo de este camino cinco años atrás. Indudablemente sin ellos nada podría haber sido posible.

A nuestros amigos y compañeros que han caminado con nosotros y nos han apoyado con sus consejos en el ámbito académico y en nuestras vidas personales. Su presencia y compañía nos ayudó a sobrellevar, disfrutar y aprender de las dificultades.

Queremos agradecer también a nuestro tutor André Fonseca y la la Universidad ORT Uruguay, por la ayuda brindada y el espacio que nos han dado desde un comienzo para desarrollarnos como profesionales y como personas.

Abstract

En el documento a continuación se presenta un sistema de reconocimiento de gestos faciales para el apoyo a personas con discapacidades motrices y/o del habla, como alternativa al uso de las manos en la interacción humano - computadora.

Hoy en día la tecnología requiere el uso de las manos para interactuar con los dispositivos, sin contemplar a las personas que por diferentes motivos no tienen la motricidad fina necesaria para controlar un *mouse*, seleccionar íconos en una pantalla o escribir en un teclado. La motivación de este proyecto es desarrollar una manera más inclusiva de interactuar con la tecnología existente, permitiéndoles integrarse al mundo moderno que tanto depende de estos dispositivos.

Basado en la tecnología disponible de reconocimiento de imágenes se diseñó un sistema que consiste de cuatro partes. La primera, un bloque de entrenamiento, a correrse al inicio y que guarda la configuración para cada usuario. En segundo lugar, la aplicación principal que hace la detección de gestos faciales en tiempo real. En tercer lugar, las aplicaciones para interactuar con la computadora y por último la base de datos que guarda la configuración de cada usuario.

El sistema implementado es capaz de detectar cuatro gestos en tiempo real (con posibilidades de seguir agregando gestos sencillamente) y cinco orientaciones posibles de la cara, con una exactitud superior al 80 %.

Palabras clave

Reconocimiento de imágenes

Detección de objetos

Modelado de objetos

Reconocimiento de gestos faciales

Landmark

Descriptor

Clasificador

Active Shape Models

Machine learning

Distancia de Mahalanobis

Cámara web

Envío de SMS

Control del *mouse*

Índice

1. Introducción **12**

1.1. Motivación 12

1.2. Objetivos 12

2. Descripción general del sistema **13**

2.1. Descripción de los componentes 13

2.1.1. Entrenamiento 13

2.1.2. Base de Datos 13

2.1.3. Detección de gestos 14

2.1.4. Aplicaciones 14

2.1.4.1. Control del mouse 14

2.1.4.2. Envío de SMS con gestos faciales 15

3. Investigación previa **16**

3.1. Antecedentes 17

3.1.1. Reconocimiento de gestos faciales para el control manos libres de una silla de ruedas inteligente 17

3.1.2. Control del *mouse* a través del *tracking* de la cara 18

3.1.3. Sistema de *tracking* y reconocimiento de gestos faciales para aplicaciones de control vehicular 19

3.2. Introducción a la representación de imágenes 20

3.3. Entrategias de detección de objetos 20

3.4. Modelado de objetos 22
3.5. Librerías disponibles en internet 24
3.5.1. Librería OpenCV 24
3.5.1.1. Clasificadores HAAR 25
3.5.2. Liberia STASM 4.1 26
3.5.2.1. SIFT y MARS 27
3.5.3. Liberia ASMLIB 27
3.5.3.1. BTSM 28
3.5.4. Librería CLM Framework 28
3.5.5. Librería Dlib 29
3.5.6. Visage 30
3.6. Elección de la librería 31

4. Librería Dlib **32**

4.1. Detección de caras 32
4.1.1. Descriptores HOG 32
4.1.2. Cálculo de los descriptores 32
4.2. Detección de *landmarks* 33
4.2.1. Cascada de regresión 34

5. Detección de gestos **36**

5.1. Estudio previo en Scilab 36

5.2. Detección de gestos como problema de clasificación 38
5.3. Modelado del estado de las partes de la cara 38
5.4. Distribución gaussiana multivariable 39
5.5. Gestos, curvas de nivel y distancia de Mahalanobis 40
5.6. Cálculo de radios 41
5.7. Histéresis 42
5.8. Interferencia entre gestos 43
5.9. Entrenamiento del algoritmo 44
5.10. Implementación y verificación del modelo en Scilab 44
5.11. Funcionamiento del algoritmo 46
5.12. Descriptores y gestos utilizados 46
5.12.1. Criterios para la selección de gestos y descriptores 47
5.12.2. Boca 48
5.12.3. Cejas 52
5.12.4. Orientación 54
5.12.5. Efecto de la distancia a la cámara 57
5.13. Determinación de los parámetros de funcionamiento del sistema . . . 59

6. Seguimiento de la cara (*Tracking*) 61

6.1. Algoritmo de *tracking* 61

7. Implementación en C++ 63

7.1. Requerimientos de la librería 63
7.2. Estructura de datos 63
7.2.1. Clase Cara 65
7.2.2. Clase Parte 65
7.2.3. Clase Gesto 67
7.3. Librerías de funciones 67
7.3.1. Funciones de Tratamiento 67
7.3.1.1. Cálculo de longitudes 68
7.3.1.2. Cálculo de áreas 68
7.3.2. Funciones de Estados 69
7.3.3. Funciones de Entrenamiento 69

8. Aplicaciones **71**

8.1. Diagrama de flujo principal 71
8.2. Envío de SMS con gestos faciales 73
8.3. Control del *mouse* 74

9. Análisis final **76**

9.1. Análisis de Efectividad 76
9.1.1. Método de evaluación del calsificador 76
9.1.2. Evaluación del desempeño de un clasificador 76
9.1.3. Resultados 77

9.2. Análisis de tiempo 78
9.2.1. Tiempos del sistema sin *tracking* 79
9.2.2. Tiempos del sistema con *tracking* 79
9.2.3. Piramidación de imagen y mejora en tiempos 80
9.3. Análisis de recursos para el funcionamiento del sistema 80
9.4. Flexibilidad y readaptación de la librería 81
9.4.1. Agregar Aplicaciones 81
9.4.2. Agregar o quitar gestos 81
9.4.2.1. Gestos programados 82
9.4.2.2. Gestos nuevos 82

10. Conclusiones **84**

10.1. Posibles mejoras 84

11. Referencias bibliográficas **86**

A. Planificación **90**

A.1. Marzo - Mayo 90
A.2. Mayo- Julio 91
A.3. Julio-Agosto 91
A.4. Agosto – Diciembre 92
A.5. Diciembre – Febrero 92
A.6. Diagrama de Gantt original 93

B. Aproximación de radios en la detección de gestos **95**

C. Aproximación a la condición de no interferencia entre gestos **98**

D. Análisis de datos en Scilab **100**

D.1. Codificación de la base de datos 100

D.2. Librería de funciones 100

D.3. Caso de uso 101

D.4. Estudio extendido sobre la boca 104

E. Ejemplos de cambios en los gestos a detectar **107**

E.1. Agregar gesto: Mueca derecha 107

E.2. Agregar gesto: Fruncir nariz 108

F. Lista de funciones implementadas en C++ **111**

F.1. Tratamiento.h 111

F.2. Estados.h 111

F.3. Entrenamiento.h 113

G. Instrucciones de instalación para OpenCV y Dlib **114**

1. Introducción

1.1. Motivación

Las tecnologías actuales se han desarrollado dando una preferencia a la interacción a través del uso de las manos. *Mouse*, teclado, botones, pantallas táctiles, los avances tecnológicos parecen exigir al usuario un movimiento manual preciso. Esto conduce a que, en algunos casos, esta interacción pueda verse limitada ya sea por tener las manos ocupadas en otra tarea, o por no tener la motricidad requerida para llevarla a cabo.

Como respuesta a esto en las últimas décadas se han desarrollado opciones alternativas como interacciones basadas en reconocimiento de voz e imagen, que no permiten desplazar completamente el uso de las manos, pero sí dar accesibilidad a quienes lo necesitan.

Este proyecto implementa un sistema de interacción basado en detección de gestos faciales por análisis de imágenes en tiempo real. La ventaja que presentan los sistemas basados en reconocimiento de imágenes es que no requieren la emisión de sonidos (que puede resultar molesto para el entorno), y en caso de aplicaciones de accesibilidad resulta ser más inclusivo, dado que la gente con discapacidades también se ve frecuentemente limitada en el uso correcto de su voz.

En la actualidad existe una gran variedad de sistemas que utilizan reconocimiento de imágenes, que van desde aplicaciones industriales hasta domótica. La mayoría de los sistemas que utilizan el reconocimiento de imágenes para interacción humano-computadora se basan en el análisis del movimiento de las manos del usuario (casos como *Kinect* de Microsoft, *PlayStation Move* de Sony, etc.). Sin embargo, este proyecto propone centrarse en gestos faciales para poder hacer una variedad de interacciones humano-computadora, o más generalmente humano-máquina, universalmente accesibles.

1.2. Objetivos

Implementar un sistema de reconocimiento de gestos faciales con la cámara web de una computadora portátil, con el fin de utilizarse como alternativa al uso de las manos en la interacción con dispositivos tecnológicos. Implementar este sistema además en un código ordenado y fácilmente adaptable, permitiendo agregar nuevas funcionalidades sin trabajo excesivo. El sistema se implementa sobre Linux, sistema operativo abierto y libre, permitiendo que se pueda utilizar en la mayoría de los equipos portátiles sin necesidad de comprar ningún software externo.

2. Descripción general del sistema

El sistema tiene cuatro componentes principales. Entrenamiento, Base de datos, Detección de gestos y Aplicaciones.

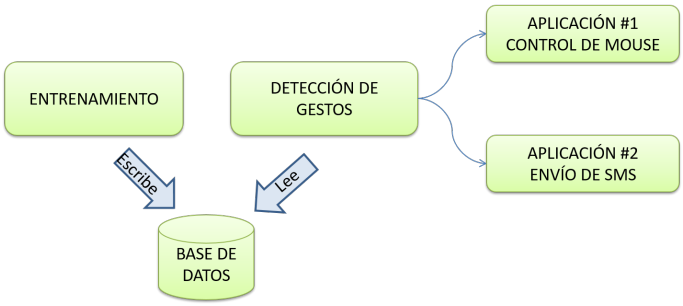


Figura 2-1: Diagrama de bloques del sistema

2.1. Descripción de los componentes

2.1.1. Entrenamiento

El entrenamiento es un proceso que se corre inicialmente cuando un nuevo usuario quiere hacer uso del sistema, y siempre que se necesite una re-calibración. Se diseñó con el objetivo de facilitarle el proceso al usuario en la mayor medida posible, que simplemente tiene que enfrentarse a la cámara y realizar los gestos que se le van indicando. Los gestos a detectar son configurables.

La idea detrás del proceso de entrenamiento es la siguiente. De una lista de posibles gestos a reconocer se escoge uno y se le indica al usuario que lo realice. Se toman un número determinado de medidas relacionadas al gesto elegido y a partir de la distribución de las medidas se calculan los parámetros necesarios para su detección posterior.

La información detallada sobre el entrenamiento se encuentra en el capítulo 5.

2.1.2. Base de Datos

La base de datos guarda la información de configuración del sistema y cada usuario. Incluye los archivos que determinan qué gestos van a ser posibles detectar y los parámetros para la detección de los mismos para cada usuario.

2.1.3. Detección de gestos

El proceso de detección de gestos contiene la inteligencia para el reconocimiento de gestos faciales en tiempo real. El proceso se encarga de:

1. leer la base de datos con los gestos entrenados y configurar la lista de gestos a detectar,
2. localizar al usuario,
3. seguirlo en caso de que se mueva dentro del alcance de la cámara,
4. adaptarse a cambios en la orientación de la cara y
5. comunicarse con la aplicación activa.

Es imprescindible que exista un archivo con los parámetros de los gestos entrenados para iniciar el proceso de detección de gestos.

Como es de esperar, el proceso inicia con una etapa de *setup*. Lee de la base de datos los parámetros y crea una lista con los gestos que es posible detectar. Luego inicia la cámara y comienza la detección y caracterización de caras (el proceso de detección y caracterización de caras se discutirá en el capítulo 4).

Una vez localizada y caracterizada una cara, para cada cuadro el sistema recorre la lista de gestos para verificar si alguno está activo y si es así ejecuta la acción correspondiente.

2.1.4. Aplicaciones

A modo de ejemplo de la utilización del sistema se han implementado dos aplicaciones.

2.1.4.1. Control del mouse

El objetivo de esta aplicación es utilizar la orientación de la cara y algunos gestos faciales para permitir desplazar el cursor a lo largo de la pantalla y lograr realizar un comando que sustituya el clic del *mouse*. Esta aplicación es una alternativa real al uso de las manos en la interacción humano-computadora.

Figura 2-2: Representación de un hombre interactuando con una computadora a través de la cámara web.

La orientación de la cara tendrá como posibilidades: mirar hacia arriba, mirar hacia abajo, mirar hacia cada lado, o mirar hacia el frente. Cada uno de los movimientos horizontales o verticales se corresponderá con un movimiento del cursor en el sentido correspondiente. Por otro lado, si la cara se encuentra en la posición frontal se utilizan distintos gestos faciales para sustituir el clic del *mouse*. Así se logran reemplazar todas las funcionalidades del *mouse* a través de la librería de funciones.

Como lo fijan nuestros objetivos y como será con todas las aplicaciones, para utilizar el *mouse* será necesaria una etapa de entrenamiento previa de las posiciones y los gestos que podrá adoptar la cara, para luego poder ser utilizados.

2.1.4.2. Envío de SMS con gestos faciales

Se ha implementado una aplicación cuyo objetivo es lograr comunicación entre el usuario del sistema y el resto de las personas. Para esto se utiliza un módulo GSM conectado al terminal sobre el que se corre el sistema y permite al usuario enviar un SMS a una o varias personas.

Estos mensajes podrían ser, por ejemplo, mensajes transmitiendo la necesidad de asistencia. Otra alternativa podría ser enviar una señal de alerta si el usuario se encuentra con una expresión de dolor o tristeza.

Como ya se mencionó y como será con todas las aplicaciones todos los mensajes y los gestos de disparo son configurables, así también como el directorio de números al cual se envían los mensajes.

3. Investigación previa

El abordaje clásico a la detección y modelado de objetos se realiza desarticulando el problema en distintas partes según lo muestra la figura 3-3. Es un diagrama de bloques en cascada denominado *pipeline*, en el que los bloques se colocan en serie uno atrás del otro [4].

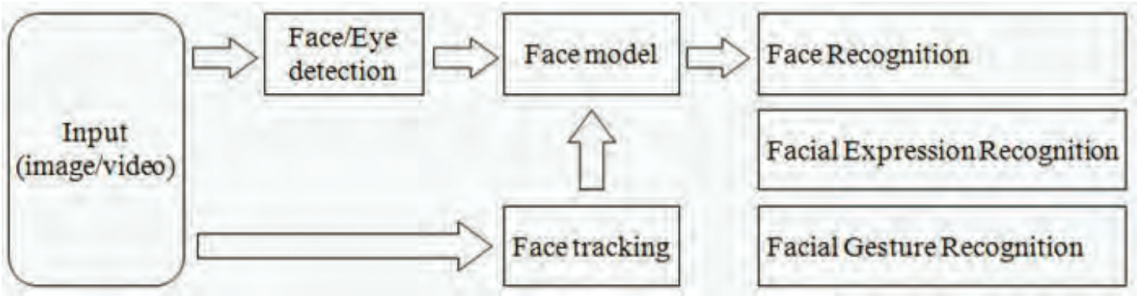


Figura 3-3: Bloques de abordaje al problema [4].

El primer bloque se ocupa de la generación de la imagen digital, el segundo se ocupa de la búsqueda de caras en el cuadro de la imágen, el tercero se ocupa de dar un modelado a la cara en virtud de cuál sea la aplicación que se desea realizar. El último es el bloque de aplicaciones que tiene distintas posibilidades para el caso de análisis de caras: análisis de expresiones faciales, análisis de gestos, etc. Como alternativa al bloque de búsqueda de caras, dado que en general resulta bastante costoso en términos computacionales, se suele utilizar un bloque de *tracking*: un algoritmo que permite seguir la cara una vez que es detectada, y que resulta menos costoso computacionalmente que buscarla en cada cuadro.

Previo al diseño de la solución, y ligado a lo comentado anteriormente, se comenzó realizando un estudio de la tecnología disponible para el reconocimiento de imágenes en general y en relación a la detección de objetos.

En concreto los temas abordados en las búsquedas fueron los siguientes:

- antecedentes,
- introducción a la representación de imágenes,
- estrategias de detección de objetos,
- estrategias de modelado de objetos y
- librerías disponibles en internet.

3.1. Antecedentes

El control de dispositivos electrónicos a través de detección de gestos o movimientos tiene innumerables aplicaciones. Por lo tanto, no es sorprendente que existan otros trabajos de investigación o patentes previas que busquen resolver problemas similares. A continuación se presentan algunos casos:

- una silla de ruedas controlada por gestos faciales,
- un mouse controlado por la posición de la cara y
- un sistema para controlar accesorios dentro de un vehículo mediante el análisis del conductor.

3.1.1. Reconocimiento de gestos faciales para el control manos libres de una silla de ruedas inteligente

Desarrollado en conjunto por investigadores en el Reino Unido y China, este sistema implementa reconocimiento de gestos faciales para sustituir al *joystick* en el control del movimiento de una silla de ruedas inteligente [1].

El sistema está basado en un DSP que se encarga de controlar el movimiento de la silla de ruedas. El DSP puede funcionar tanto en modo manual, con un *joystick*, como en modo automático. En el segundo, se implementa el reconocimiento de gestos faciales desde una PC que envía los comandos correspondientes al microprocesador. Además, cualquiera de los dos modos funciona en paralelo con un detector de obstáculos por ultrasonido.

Los problemas específicos que pretende resolver este proyecto frente a proyectos anteriores son: cómo tratar una cara que esté parcialmente en la imagen o de perfil, independizar el sistema de reconocimiento facial frente a cambios en la iluminación y frente a diferentes tonos de tez posibles, permitir detectar caras aún con obstáculos como lentes o bigotes y por último detectar caras correctamente en ambientes con muchos objetos.

El sistema de reconocimiento de gestos utiliza AdaBoost. Desarrollado en el 2004, presenta una forma rápida y eficiente de detectar caras con una cascada de clasificadores de estilo HAAR (ver Sección 3.5.1.1) [18] .Una de las grandes ventajas de AdaBoost es que permite reconocer caras de perfil. Adicionalmente, como AdaBoost no permite procesamiento en tiempo real, utiliza un algoritmo optimizador para seguir a la cara llamado CamShift.

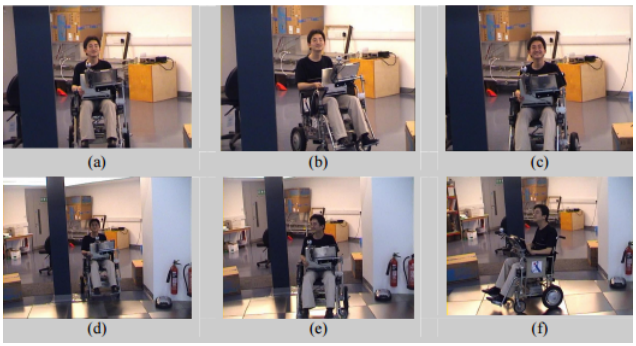


Figura 3-4: Silla de ruedas inteligente pasando por una puerta [1].

En relación a los resultados, el tiempo de procesamiento por cuadro en su sistema integrando AdaBoost y CamShift varía entre 110 y 220 ms dependiendo del tamaño determinado para el rectángulo de la cara.

3.1.2. Control del *mouse* a través del *tracking* de la cara

Este proyecto introduce un controlador del mouse por tracking de la cara basado en un modelado 3D de la misma [2].

Los diferentes comandos del mouse se disparan con gestos faciales. Para identificarlos se analizan: el movimiento vertical del centro del labio superior e inferior, y el movimiento horizontal y vertical de las comisuras de la boca. La acción de clic derecho es disparada por a detección de boca abierta y la acción de clic izquierdo es disparada por el estiramiento de las comisuras (sonrisa).

Desarrollaron tres tipos de mouse diferentes: directo, diferencial y *joystick*. El modo directo posiciona el mouse donde apunta la estimación 3D de la cara. El modo *joystick* permite mover el mouse usando la orientación de la cara, cambiando la velocidad según el ángulo de la misma . Finalmente, el modo diferencial que mueve el cursor de manera similar, pero permite volver a la posición inicial con un gesto adicional.

Las conclusiones revelan que el modo más eficiente es el modo *joystick*. En una computadora con CPU de 2GHz tiene un tiempo de procesamiento entre 17 y 22 fps dependiendo si está detectando solo el movimiento de la cara o si además puede detectar gestos de la misma (ej. abrir boca). Además, el algoritmo de seguimiento funciona bien si el usuario mira hacia el frente pero presenta fallas cuando la dirección de la cara del mismo supera los 40^{o} en relación al centro, o cuando se mueve con velocidad alta.

Figura 3-5: Modelado 3D de la cara mientras el usuario juega al buscaminas [2].

3.1.3. Sistema de *tracking* y reconocimiento de gestos faciales para aplicaciones de control vehicular

Esta patente implementa un sistema de tracking del movimiento de un conductor o pasajero en un vehículo (en tierra, mar o aire) y controla dispositivos de acuerdo a la posición, movimiento o gestos faciales del mismo [3]. Esto puede ser usado tanto como para el confort o entretenimiento del usuario (control de aire acondicionado, ventanas o radio del vehículo) como para su seguridad (*airbag*, ajuste del cinturón de seguridad).

Cuenta con cuatro bloques: una o más cámaras para monitorear al conductor; un sistema de seguimiento de la posición, velocidad o aceleración de distintas partes del cuerpo; un sistema de reconocimiento de gestos; y algoritmos para el control de los distintos dispositivos dentro del vehículo.

El sistema sigue a las diferentes partes del cuerpo basado en color, movimiento y forma del mismo en la imagen. El algoritmo usa una técnica de reconocimiento de color novedosa que requiere computación mínima. Esta técnica calcula tres estimaciones de la ubicación probable del objeto en función a los parámetros mencionados y los pondera. Las ventajas de esta técnica son su robustez frente al ruido y a movimientos veloces del conductor. Además, permite reconocer gestos tanto dinámicos como estáticos.

El sistema funciona en tiempo real. Sin embargo, no hay datos concretos del rendimiento que permitan comparar las aplicaciones.

3.2. Introducción a la representación de imágenes

En el contexto de algoritmos de tratamiento digital, las imágenes vienen representadas por matrices de $m \times n$ donde cada posición tiene la información de un pixel de la cámara. En general, cada pixel tiene almacenados a su vez tres bytes, representando las intensidades de los colores rojo r, verde g, y azul b (modelo RGB24), siendo así el vector $[255, 255, 255]$ el color blanco y el vector $[0, 0, 0]$ el color negro. La cantidad de bits utilizados para representar cada color puede variar, permitiendo representar más o menos colores (dependiendo del modelo) [8].

Para el tratamiento de imágenes en el sentido de detección de objetos, los algoritmos tienden a simplificar la información, llevando la imagen a blancos y negros. Cada pixel vendrá entonces representado por un único valor que puede obtenerse utilizando distintos algoritmos a partir de la imagen original, por ejemplo usando la intensidad media RGB en cada pixel, es decir $y = 1/3 \cdot R + 1/3 \cdot G + 1/3 \cdot B$, o tomando transformaciones específicas, por ejemplo $y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B$ (transformación sugerida en la literatura[7][8] y utilizada por OpenCV[6]).

3.3. Entrategias de detección de objetos

La detección de objetos es el bloque fundamental para lograr cualquier aplicación que requiera realizar *tracking*, reconocimiento de caras, reconocimiento de expresiones faciales o reconocimiento de gestos faciales (nuestro caso). La idea de este bloque es detectar la posición de la cara que envía los comandos gestuales en la imagen en la que se encuentra y estimar sus dimensiones. En general, al bloque de detección de caras lo sigue un bloque de detección de ojos en cascada que es bastante más robusto.

El esquema de detección de objetos más común se compone de tres partes principales[5][9][11]:

1. Un sistema ocupado de recorrer la imagen con la finalidad de determinar si en distintas partes se podría o no encontrar un objeto. La estrategia más común utilizada es el *"sliding window"*.

2. Un sistema para la caracterización de la región de imágen determinada por el gesto anterior. Ejemplos de estos sistemas son los "filtros HAAR", "histogramas de gradiente orientado" y las "redes neuronales convolucionales" (en este último caso ya se realiza también una clasificación).

3. Un último sistema de clasificación, que determina si en la región determinada por el primer sistema, con la caracterización dada por el segundo, se encuentra un objeto. En esta instancia se usan diferentes tipos de clasificadores, siendo

los más comunes los SVM (máquinas de soporte vectorial), aunque pueden utilizarse también redes neuronales artificiales.

El sistema de caracterización (segundo), que es el central, es el más variable en función del objeto que se desea detectar, y el que define la eficiencia del algoritmo. Explicaremos a continuación la estrategia de *sliding window* y los clasificadores SVM que son las partes más estandarizadas de estos algoritmos.

El ***Sliding Window*** es una estrategia que busca simplificar el trabajo del segundo sistema [11]. La idea consiste en seleccionar un rectángulo de la imagen sobre el cual se utilizarán los sistemas dos y tres para determinar si en este rectángulo se encuentra el objeto a detectar. Este rectángulo se desplaza sobre toda la imagen para ver si en determinada posición se encuentra el objeto. Luego se varía el tamaño del rectángulo y se repite el procedimiento, logrando así buscar en toda la imagen todos los objetos posibles de todos los tamaños posibles. Este rectángulo que se utiliza para el *sliding window* suele estar dimensionado de forma que siga más o menos la distribución del cuerpo a detectar. Se parte de un tamaño inicial, y evoluciona hasta un tamaño final, determinado por los tamaños máximos y mínimos del cuerpo que se va a detectar.

Por otro lado los clasificadores **SVM** [10] funcionan de la siguiente manera. Se considera un conjunto de m puntos $x_i \in \mathbb{R}^n$ de entrenamiento (puntos que representan n características de una de las m muestras). Estos puntos están preetiquetados indicando si pertenecen a una clase $y = 1$ o $y = 0$ (si una imagen contiene determinado objeto o no). Los clasificadores SVM intentan encontrar un hiperplano de máxima separación en $\mathbb{R}^n$ entre los subconjuntos de puntos con distintas etiquetas. El hiperplano de separación viene dado por la ecuación:

$$\vec{w} \cdot \vec{x} - b = 0 \tag{3-1}$$

donde $\vec{w}$ representa el vector normal al hiperplano, $\vec{x}$ es el vector de características, $\frac{b}{\|\vec{w}\|}$ representa la distancia del hiperplano al origen entre los conjuntos, y $\frac{2}{\|\vec{w}\|}$ es la distancia entre los dos conjuntos.

Luego este hiperplano se utiliza para determinar si una nueva muestra con sus n características, pertenece al grupo $y = 1$ o $y = 0$. Se tendrá matemáticamente entonces una función:

$$f(\vec{x}) = \vec{w} \cdot \vec{x} - b \tag{3-2}$$

de forma que si $f(\vec{x}) > 0$ se dice que la muestra con el vector de características

$\vec{x}$ pertenece al conjunto de los puntos con $y = 1$. De lo contrario pertenece al de los puntos que tienen $y = 0$.

3.4. Modelado de objetos

En el campo del *computer vision*, los investigadores han propuesto diferentes técnicas para el análisis de la variación de la forma de los cuerpos, como son los "contornos activos" [13] y los *"deformable templates"* [14]. En la medida que se fueron desarrollando otras técnicas para el tratamiento de datos como el análisis de componentes principales (PCA), técnica utilizada para reducir la dimensionalidad de un conjunto de datos eliminando las dimensiones sobre las que se tiene menor varianza [15], y dadas las dificultades presentadas por los primeros algoritmos, se lograron desarrollar las dos técnicas mayormente utilizadas en la actualidad[4]. Estas son los ASM (*Active Shape Models*, 1995), y los AAM *(Active Apprearence Models*, 2001). Técnicas que se creen de importancia y que se explicarán a continuación.

Los **ASM** son modelos matemáticos estadísticos que representan la forma de los objetos [4]. Se entrenan y luego se utiliza un algoritmo que iterativamente deforma una red de puntos para ajustarse a un objeto dentro de la imagen, de forma que luego de la convergencia del algoritmo se tiene un representación matemática para la forma del cuerpo. Fueron desarrollados por Cootes y Taylor a principios de los 90.

Las ventajas del modelo radican en que es extensamente aplicable y permiten pequeñas variaciones del modelo sin crear falsos positivos. Deben ser entrenados previamente, requeriendo el entrenamiento que el programador indique en cada imagen de la base de datos los mismos *landmarks* (puntos claves que aparecen en todas las imágenes). Esto limita las aplicaciones de los modelos ya que requiere que la topología del objeto no cambie y que no sea lo suficientemente amorfo para no poder encontrar *landmarks* distintivos en nuevas imagenes.

La representación matemática básica incluye, entonces, un conjunto l de puntos clave del objeto, que juntos forman el *shape vector* s:

$$s = (x_1, y_1, \cdots, x_l, y_l)^T \tag{3-3}$$

La expresión de un *shape vector* genérico en la mayoría de las implementaciones de este algoritmo, luego del entrenamiento se busca escribir como:

$$s = s_0 + \Sigma_{i=1}^{n} p_i \cdot s_i \tag{3-4}$$

donde s_0 es el *shape* promedio, las componentes s_i son los *shapes* ortogonales obtenidos por *Procrustes analysis* y PCA y los p_i son las componentes que caracterizan al shape s. *Procrustes* es un análisis estadístico de la distribución de un conjunto de formas. Para comparar las formas de uno o más objetos, primero deben ser superpuestos óptimamente mediante la traslación, rotación y escalado (se normalizan en orientación y tamaño). El objetivo es obtener tamaños y posiciones similares minimizando una medida de diferencia de forma llamada distancia de Procrustes entre objetos [16].

Los **AAM** son modelos desarrollados por Cootes y Edwards en 2001 [4]. Son modelos paramétricos estadísticos que ilustran tanto cambios en forma como de apariencia de los objetos. Estos modelos lineales son obtenidos, al igual que los ASM, mediante PCA aplicado sobre un conjunto de entrenamiento previamente etiquetado.

La representación matemática es similar a la de los ASM, con la diferencia que en los AAM los puntos están indexados y configuran una malla que identifica cada sección de la imagen. Los AAM introducen además el concepto de modelo de apariencia. La idea es que se logre establecer un mapeo entre los triángulos de los objetos de entrenamiento s y los del objeto promedio s_0. Se define entonces el vector de apariencia sobre cada uno de los pixeles que forman parte de s_0, como el vector que contiene el valor de intensidad o colores de cada pixel que llamaremos $A(x)$, y de forma análoga al *shape* del objeto se buscará escribir:

$$A(x) = A_0(x) + \Sigma_{i=1}^{m} \alpha_i \cdot A_i(x) \tag{3-5}$$

donde α_i son los parámetros de apariencia, $A_0(x)$ la apariencia del pixel x en la imagen promedio, y $A_i(x)$ son las componentes ortogonales.

Como se puede ver el modelado AAM es más completo que el modelado ASM, puesto que se agrega el modelo de apariencias. Los AAM se utilizan cuando es necesario reconocer algunos aspectos referentes a la apariencia del objeto, pero son mucho más costosos computacionalmente que los ASM que se utilizan cuando se requiere conocer únicamente la forma del objeto.

Dado que el objetivo es implementar aplicaciones que detecten cambios en la forma de la cara, pero no están contenidas las aplicaciones que tengan que ver con la apariencia de la misma se optó por buscar modelados ASM. Además en las aplicaciones que se busca implementar se necesitan algoritmos que sean lo más rápido posibles, por lo tanto queda evidente que esta elección será la que más se ajusta a los requerimientos.

3.5. Librerías disponibles en internet

3.5.1. Librería OpenCV

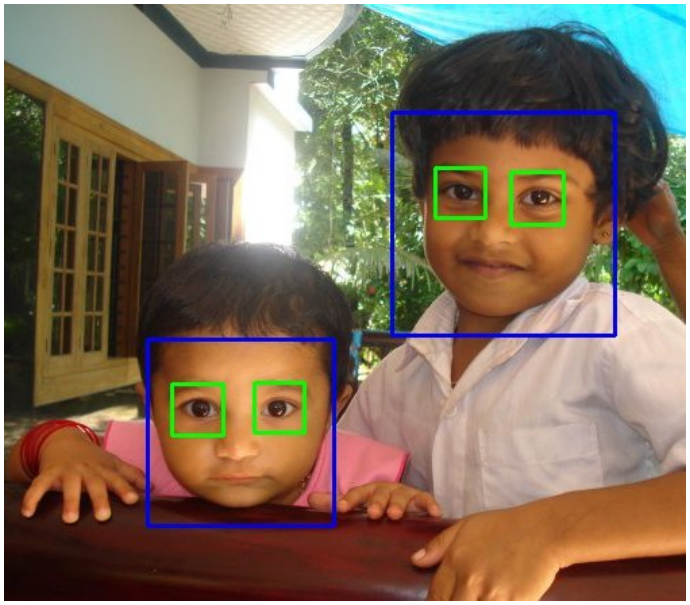


Figura 3-6: OpenCV: Ejemplos de detección de caras [17].

OpenCV (*Open Source Computer Vision Library*) es una librería *open source* para reconocimiento de imágenes y *machine learning* [17]. OpenCV fue creada con el fin de proveer una infraestructura común para aplicaciones de reconocimiento de imágenes y para acelerar el uso de esta tecnología en productos comerciales. Como esta bajo la licencia BSD, el código es fácilmente implementable y modificable.

La librería cuenta con mas de 2500 algoritmos optimizados, entre ellos un exhaustivo grupo de algoritmos *state-of-the-art* de reconocimiento de imágenes y *machine learning*. Por ejemplo, algoritmos de detección y reconocimiento de caras, identificación de objetos, seguimiento de objetos en movimiento y extracción de modelos $3D$ de objetos.

La comunidad de usuarios de OpenCV supera los 47000 personas y el número estimado de descargas es de siete millones.

Está implementada en C++/C, C, Python, Java y MATLAB, soporta Windows, Linux, Android y Mac OS.

3.5.1.1. Clasificadores HAAR

Los algoritmos de detección de caras de OpenCV se basan en clasificadores HAAR en cascada. El algoritmo está basado en el artículo *"Rapid Object Detection using a Boosted Cascade of Simple Features"* [18]. Es un enfoque basado en *machine learning* donde una función cascada es entrenada a partir de un amplio grupo de imágenes positivas y negativas (una imagen es positiva si contiene al objeto y negativa si no lo contiene). Luego, se usa para detectar esos objetos en otras imágenes.

Una vez clasificadas las imágenes se procede a extraer las *features*. Para eso, se usan las HAAR *features* a continuación. Cada *feature* es un valor obtenido mediante la resta de la suma de los pixeles bajo el rectángulo blanco de la suma de pixeles bajo el rectángulo negro.

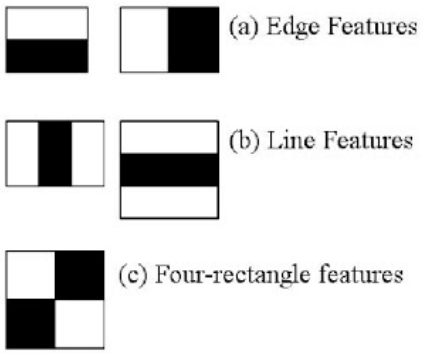


Figura 3-7: Clasificadores HAAR [18].

Para evitar que el procesamiento sea excesivo (pixel a pixel) se introducen las imágenes integrales. Se divide la imagen original en una cuadricula y se suman los valores de todos los pixeles comprendidos en el rectángulo.

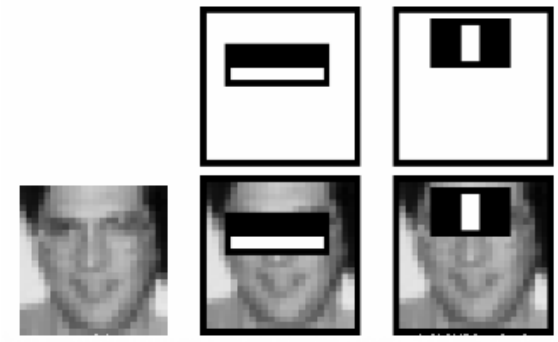


Figura 3-8: Clasificadores HAAR en una imagen [18].

El clasificador es un promedio de varios clasificadores débiles (no pueden cla-

sificar la imagen por sí mismos). De todas maneras esto sigue siendo un proceso ineficiente porque gasta mucho tiempo de procesamiento en secciones donde no hay una cara.

Así surge el concepto de clasificadores en cascada. Se aplican los clasificadores uno a uno y si alguno da resultado negativo se descarta el rectángulo y así se va concentrando en la región donde sí aparece una cara [20].

La aplicación implementada usa algoritmos de OpenCV para acceder a la cámara. La estrategia para reconocer la cara inicial e ignorar otras caras que entren al frame fue desarrollada primero en OpenCV

3.5.2. Liberia STASM 4.1

Figura 3-9: Stasm: Ejemplos de detección de *landmarks* [23].

STASM es una librería en C++ para localizar 77 *landmarks* faciales en imágenes. Recibe una imagen y devuelve las coordenadas de los *landmarks*. Está diseñada para trabajar con la vista frontal de la cara en posiciones relativamente derechas y con expresiones neutrales. El resultado es pobre para caras orientadas hacia los constados o con expresiones. STASM también permite entrenar modelos.

Usa OpenCV y está bajo una licencia del tipo BSD.

La librería está basada en el paper: *"Active Shape Models with SIFT Descriptors and MARS"* [23].

El algoritmo descrito en el *paper* usa ASM para localizar los *landmarks*, pero usa una versión simplificada de descriptores SIFT para encontrar pequeñas partes de una imagen que corresponden con una imagen patrón [22], remplazando los perfiles unidimensionales utilizados en las versiones clásicas. Adicionalmente usa MARS para

determinar eficientemente los descriptores alrededor del *landmark*. También introduce técnicas para disminuir significativamente la carga de computación, haciendo el algoritmo útil en aplicaciones prácticas.

3.5.2.1. SIFT y MARS

SIFT (S*cale Invariant Feature Transform* [26]) propone un método para extraer *features* invariantes distintivas de imágenes. Las *features* son invariantes frente a escalado o rotación y está probado que proveen una identificación robusta frente a distorsión, cambios en la perspectiva 3D, agregado de ruido y cambios en la iluminación.

El reconocimiento de caras se hace identificando *features* individuales en una base de datos de *features* conocidas usando el algoritmo *nearest-neighbor*, un método no paramétrico usado para clasificación donde la salida depende de a qué clase pertenecen la mayoría de sus vecinos [24]. A continuación se aplica una técnica para la detección de figuras en imágenes digitales que puedan ser expresadas matemáticamente llamada transformada de Hough [25] . Finalmente se realiza una verificacion por mínimos cuadrados.

MARS (*Multivariate Adaptive Regression Splines* [27]) es un método de regresión no paramétrica que puede ser considerada una extensión de los modelos lineales que automáticamente modela no linealidades e interacciones entre variables.

3.5.3. Liberia ASMLIB

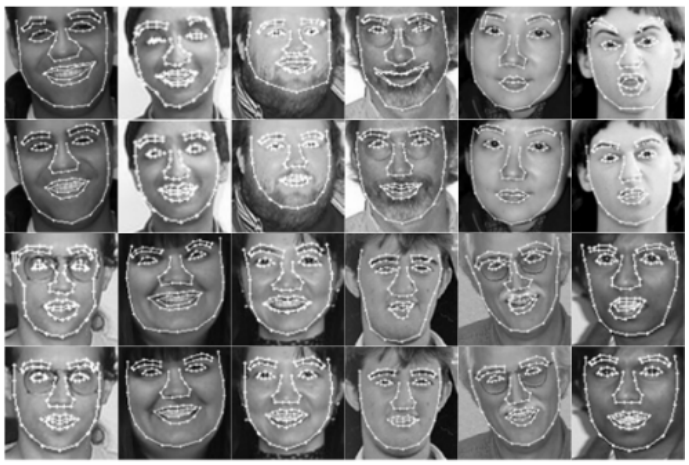


Figura 3-10: ASMlib: Ejemplos de detección de *landmarks* [28].

Es una librería *open source* basada en ASM y desarrollada en C++ usando OpenCV. La librería soporta Linux (32 y 64 bits), Windows y Mac OS.

ASMLIB implementa ASM y BTSM (*Bayesian Tangent Shape Model*). Da un buen resultado para caras de frente y ejecuta en tiempo real con una cámara web. Permite entrenar un modelo propio.

3.5.3.1. BTSM

Desarrollado en el artículo *"Bayesian Tangent Shape Model: Estimating Shape and Pose Parameters via Bayesian Inference"* [28] estudia el problema de análisis de una *shape* y sus aplicaciones para localizar *landmarks* faciales en caras de frente. Proponen una solucion basada en Interferencia Bayesiana y la aproximación por la tangente a la *shape*. Los coeficientes y parámetros son definidos a través de la estimación MAP. Llamada Probabilidad Máxima a posteriori, MAP es una estimación de una cantidad desconocida que es igual a la moda de la distribución posterior. Es usada para obtener un estimativo de una cantidad no observada basado en datos empíricos [29].

3.5.4. Librería CLM Framework

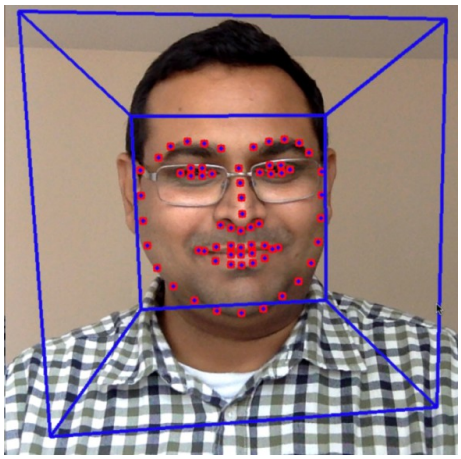


Figura 3-11: CLM Framework: Ejemplo de detección de *landmarks* [30].

CLM Framework, también conocida como *Cambridge Face Tracker*, es una librería *open source* para detección de *landmarks* faciales y estimación de pose de la cara. Fue publicada en el 2012 por un grupo de investigadores de Cambridge, liderado por Tadas Baltrušaitis. La librería está desarrollada en OpenCV y tiene una licencia que impide la comercialización.

CLM Framework está basada en el paper *3D Constrained Local Model for Rigid and Non-Rigid Facial Tracking* [30]. Presenta un modelo 3D constreñido local

(CLM),para tracking de *features* faciales de manera robusta, aún en diferentes poses. El enfoque integra información sobre profundidad e intensidad en un marco de referencia común. Además presenta grandes beneficios sobre métodos CLM comunes en términos de tiempo de convergencia y exactitud.

3.5.5. Librería Dlib

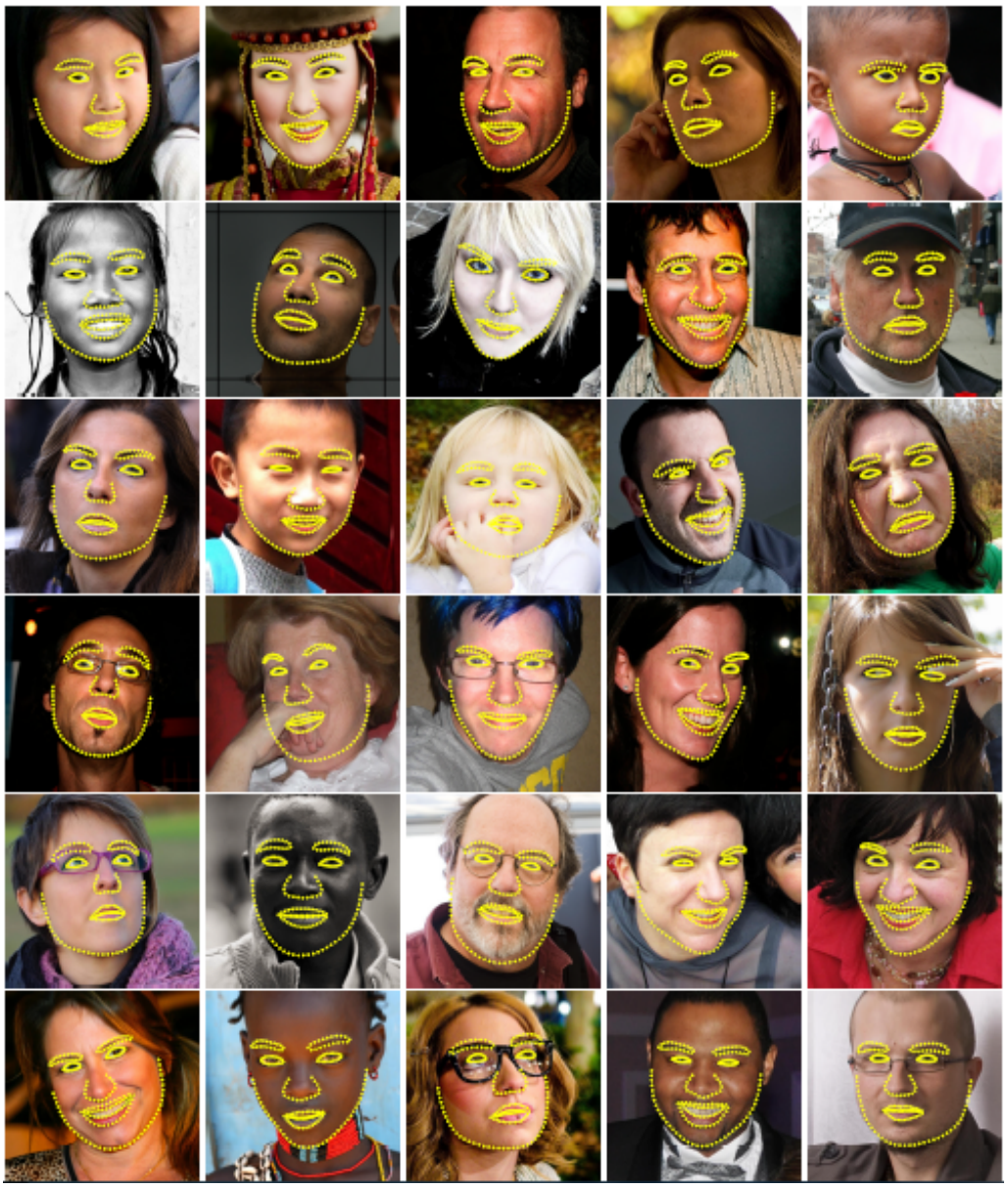


Figura 3-12: Dlib: Ejemplos de detección de *landmarks* [31].

Dlib es una librería *open source* multipropósito desarrollada en el lenguaje C++. Es principalmente una colección de librerías independientes, cada una acompañado por una extensa documentación y modos de *debug*.

Davis King ha sido el autor primario desde el comienzo del desarrollo en el 2002. En este tiempo Dlib ha crecido para incluir una gran variedad de herramientas. En particular, contiene librerías para redes, *threads*, interfaces gráficas, estructuras complejas de datos, álgebra lineal, *machine learning*, procesamiento de imágenes, *data mining*, XML y parseo, optimización numéricas, redes Bayesianas y numerosas otras aplicaciones. En los últimos años un gran porcentaje del desarrollo fue concentrado en la creación de una amplia gama de herramientas de *machine learning* estadísticas. Sin embargo, Dlib sigue siendo una librería multipropósito y acepta contribuciones de librerías de alta calidad en cualquier dominio.

La filosofía de desarrollo de la librería incluye una dedicación a la potabilidad y la facilidad de uso. Por tanto, todo el código esta diseñado para ser lo más portable posible y no requiere al usuario configurar o instalar nada. Actualmente la librería funciona en OS X, MS Windows, Linux, Solaris, the BSDs, y HP-UX.

El detector de *landmarks* de Dlib contiene 68 *landmarks* distribuidos en puntos clave de la cara. Se adapta a cambios de expresión y a orientaciones no frontales. Los algoritmos de detección de caras y de *landmarks* faciales se explicarán con detalle en el próximo capítulo [31].

3.5.6. Visage

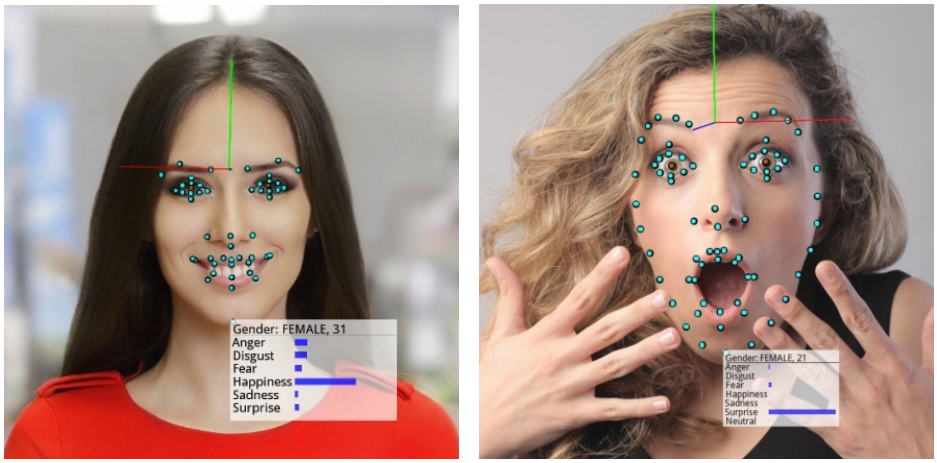


Figura 3-13: Visage: Ejemplos de detección de *landmarks* [33].

Visage Technologies AB es una compañía privada dedicada a desarrollar software de reconocimiento de imágenes para análisis y tracking de la cara. Su tecnología abarca tracking de cara y de *features* faciales, incluyendo detección de caras a nivel *state-of-the-art*, *tracking* de la mirada, estimación de género y de emociones [33].

3.6. Elección de la librería

En la elección de la librería de detección de *landmarks* en la que basar el sistema se tomaron como requerimientos principales:

- open source,
- libre,
- robusta,
- adaptable a cambios de orientación,
- adaptable a expresiones no neutrales y con
- procesamiento en tiempo real.

Para comenzar, Visage fue descartada por ser una aplicación cerrada y paga.

En segundo lugar se descarto CLM Framework porque no es de libre comercialización y aunque incluye estimación de la pose de la cara, el rendimiento es peor que el de Dlib en términos de tiempo de procesamiento.

OpenCV es útil para acceder a la cámara pero solo tiene implementado un detector de caras y de ojos por lo que tampoco se adecuaba a los requerimientos.

Dentro de las tres restantes, STASM y ASMlib no devuelven buenos resultados con caras que no están de frente o con expresiones, mientras que Dlib logra adaptarse a esos cambios.

Por lo tanto, se eligió Dlib. El sistema utiliza algoritmos de Dlib tanto como para detectar caras como para detectar los *landmarks* en las mismas. También aprovecha las librerías de álgebra lineal para hacer la detección de gestos. La información sobre los algoritmos de detección se encuentra en el capítulo siguiente.

4. Librería Dlib

Por su eficacia y versatilidad frente a las otras librerías consideradas Dlib fue elegida para realizar la detección de caras y *landmarks* faciales. A continuación se explicará la teoría detrás de los algoritmos de detección.

4.1. Detección de caras

El algoritmo utilizado por Dlib para encontrar caras está basado en el artículo: *"Histograms of Oriented Gradients for Human Detection"* [32].

4.1.1. Descriptores HOG

El Histograma de Gradientes Orientados (HOG) es un descriptor de *features* usado en procesamiento de imágenes para detectar objetos. La idea principal detrás del los histogramas de gradiente orientado como descriptores radica en que la apariencia y forma local de un objeto puede ser descrita por la distribución de gradientes de intensidad o direcciones de contorno.

La imagen se divide en pequeñas regiones conectadas llamadas celdas, y para los pixeles dentro de cada celda un histograma de gradientes de direcciones es computado. El descriptor es la concatenación de estos histogramas. Para mejorar la exactitud, los histogramas locales pueden ser normalizados por contraste mediante el cálculo de una medida de intensidad a través de una región más grande de la imagen, llamada bloque, y después normalizando todas las celdas del bloque con ese valor. Esta normalización resulta en una mayor invarianza a cambios en la iluminación y la profundidad.

La ventaja de los descriptores HOG sobre otros descriptores es que, como opera a nivel de celdas locales, es invariante a cambios en la geometría y la iluminación.

4.1.2. Cálculo de los descriptores

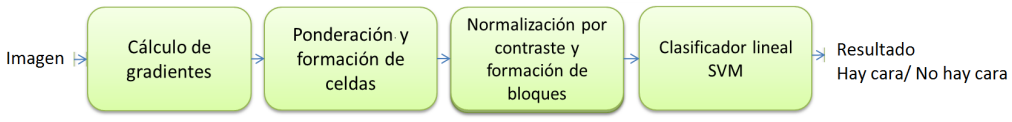


Figura 4-14: Algoritmo de detección.

El primer paso es el cálculo de los gradientes. El método más común es aplicar la máscara derivativa unidimensional en dirección horizontal, vertical o ambas. Una de las máscaras más populares es $[-1, 0, 1]$ y $[-1, 0, 1]^T$.

El segundo paso es crear las celdas. Cada pixel dentro de la celda tiene un voto ponderado para el histograma de orientaciones basado en los valores hallados en el primer paso. En lo que concierne al voto de cada pixel, puede ser el valor del gradiente en sí o alguna función de esta magnitud.

El tercer paso va a resolver las ambigüedades introducidas por posibles cambios en la iluminación y el contraste. Los valores de gradientes deben ser normalizados localmente, lo que requiere agrupar celdas en bloques más grandes. El descriptor HOG es entonces el vector concatenado de los histogramas normalizados de cada región. Es normal que los bloques sean superpuestos, por lo que cada celda contribuye más de una vez al descriptor final.

El paso final es alimentar los descriptores calculados previamente en algún tipo de sistema basado en aprendizaje supervisado. Normalmente se usa un clasificador SVM, pero puede usarse una red neuronal para mayor exactitud.

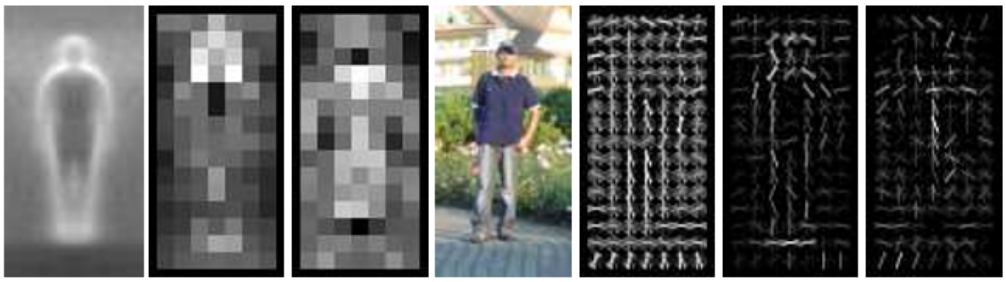


Figura 4-15: Histograma de gradientes orientados [32].

4.2. Detección de *landmarks*

El algoritmo que utiliza Dlib para encontrar los *landmarks* esta desarrollado en el artículo*: One Millisecond Face Alignment with an Ensemble of Regression Trees* [34].

Presenta un nuevo algoritmo que realiza *face alignment* en milisegundos y con exactitud comparable o superior a métodos *state-of-the-art.*

Lo ganado en velocidad es consecuencia de identificar los componentes esenciales de algoritmos previos de *face alignment* e incorporarlos en una cascada de funciones

de regresión de alta capacidad aprendidos via *gradient boosting*[1].

En este caso cada función de regresión estima eficientemente la forma a partir de una estimación inicial y las intensidades de los pixeles indexados relativos al estimativo inicial.

4.2.1. Cascada de regresión

Se denomina $x_i \in \mathbb{R}^2$ a las coordenadas en (x, y) del i-ésimo *landmark* en una imagen I. El vector $S = (x_1^T, x_2^T, ..., x_p^T)^T \in \mathbb{R}^{2p}$ representa las coordenadas de los p *landmarks* en la imagen I. De ahora en más se referirá al vector S como *shape* y a $\hat{S}(t)$ como la estimación actual de S. Cada regresor[2], $r_t(.,.)$ de la cascada predice un vector actualizado de la imagen que se suma a la estimación actual para mejorarla.

$$\hat{S}(t+1) = \hat{S}(t) + r_t(I, S(t)) \tag{4-6}$$

El punto crítico de la cascada es que el regresor r_t hace predicciones basado en características, como por ejemplo la intensidad de los pixeles, computadas de la imagen I e indexadas relativas a la estimación actual $\hat{S}(t)$.

Para entrenar cada regresor se usa la técnica *gradient tree boosting*. Se asume que toda la información para el entrenamiento está disponible en una base de datos $(I_1, S_1), ..., (I_n, S_n)$ donde cada I representa la imagen y cada S representa la *shape*.

El proceso se itera hasta tener una cascada de regresores y el resultado es la última *shape*.

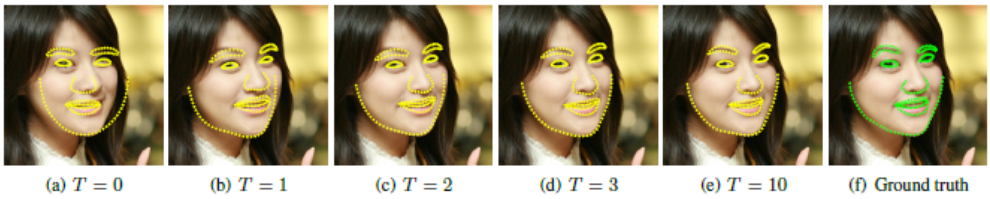


Figura 4-16: Evolución de la *shape* [34].

Los 68 *landmarks* detectados por Dlib son los ilustrados en la figura 4-17.

[1] Técnica de machine learning para problemas de regresión que produce un modelo de predicción a partir de un grupo de modelos débiles, típicamente árboles de decisión [35].

[2] Proceso estadístico para estimar la relación entre variables.

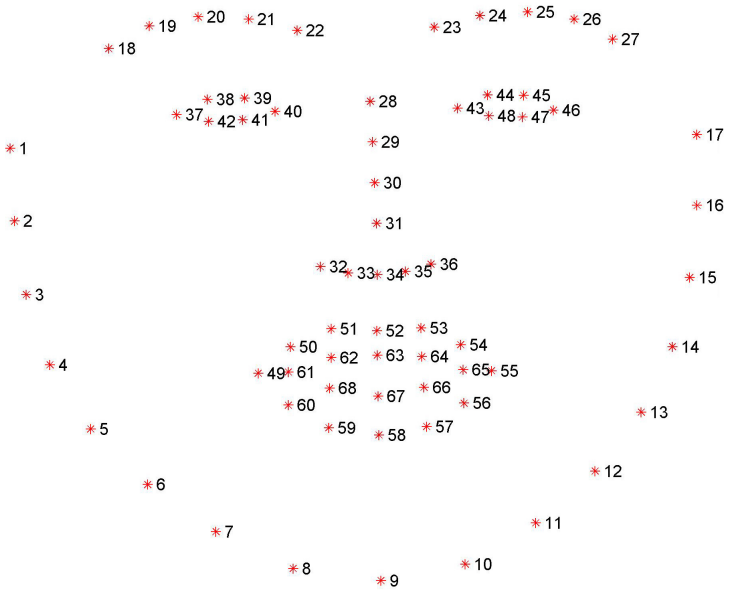


Figura 4-17: Numeración de los *landmarks*.

5. Detección de gestos

5.1. Estudio previo en Scilab

Previo a la proposición de un modelo y algoritmo para la detección de gestos, se realizó un análisis en el *software* de cálculo numérico y prototipado Scilab, en base a un conjunto de muestras de los *landmarks* de diferentes personas almacenados en distintos archivos.

El análisis y la implementación de los algoritmos en Scilab, no solamente permitieron tener una herramienta de *debugging* efectiva, si no que además el *software* introduce una plataforma más simple de desarrollo de algoritmos y permite elaborar gráficas y obtener datos más fácilmente que si los algoritmos se escribieran en C++.

Una vez implementado un simple *loop* de búsqueda de caras y convergencia de *landmarks*, se escribió un programa que permite exportar las posiciones de los *landmarks* en función del tiempo, que se registran durante un lapso de algunos segundos. Se creó así una pequeña base de datos de *landmarks* en función del tiempo con siete personas, cada una de las cuales debió realizar un vídeo con cada uno de los diez gestos que se deseaba estudiar, para cada una de las tres condiciones de distancia, y para algunos además se agregó la condición de distinta luminosidad[3]. Se generaron entonces archivos cuyo nombre identifican cada una de las características anteriores (ver Anexo D.1 para la codificación de los archivos).

Se pidió a los voluntarios que grabaran un vídeo para cada gesto en el que lo realizaran varias veces, volviendo a la posición de reposo. Por ejemplo para el caso de "cejas levantadas" se pidió que se alzaran y bajaran las cejas hasta su posición de reposo unas tres veces durante la toma de datos.

Los datos fueron exportados a un archivo de texto plano (de extensión .csv).

$$\begin{array}{cc} t = 0 & x_0; y_0; x_1; y_1; \cdots; x_n; y_n \\ t = 1 & x_0; y_0; x_1; y_1; \cdots; x_n; y_n \\ \vdots & \vdots \\ t = T_f & x_0; y_0; x_1; y_1; \cdots; x_n; y_n \end{array}$$

Figura 5-18: Esquema representativo del archivo de salida para el programa de generación de datos. La columna de tiempos a la izquierda es solo representativa, los únicos datos salvados son los *landmarks*. T_f representa el último instante de muestreo y n es la cantidad de *landmarks*.

[3]Los archivos con diferente iluminación son meramente para un estudio cualitativo por lo tanto tiene sentido analizarlos únicamente cuando se comparan con la misma persona.

Se tendrán entonces una cantidad de m filas, indicando las coordenadas de todos los n *landmarks* para ese instante (matriz de $m \times 2n$).

Todas las funciones de análisis de *landmarks* (cálculos de distancias, áreas, longitudes, etc.) fueron implementadas y probadas en Scilab, y luego utilizadas para analizar cómo determinar los distintos gestos posibles. La librería de funciones de análisis y cálculos implementadas en Scilab se encuentran adjuntas al proyecto y gran parte de ellas se encuentran documentadas (ver Anexo D.2). En el Anexo D.3 se puede ver también un caso de uso de la librería para el análisis de las cejas.

Luego de analizar algunos gestos en Scilab, se llegó a la conclusión de que cada parte de la cara puede representarse con un conjunto de variables reales que caracterizan su estado. A su vez, en la medida en la que se realizan gestos, estas cantidades varían, describiendo trayectorias.

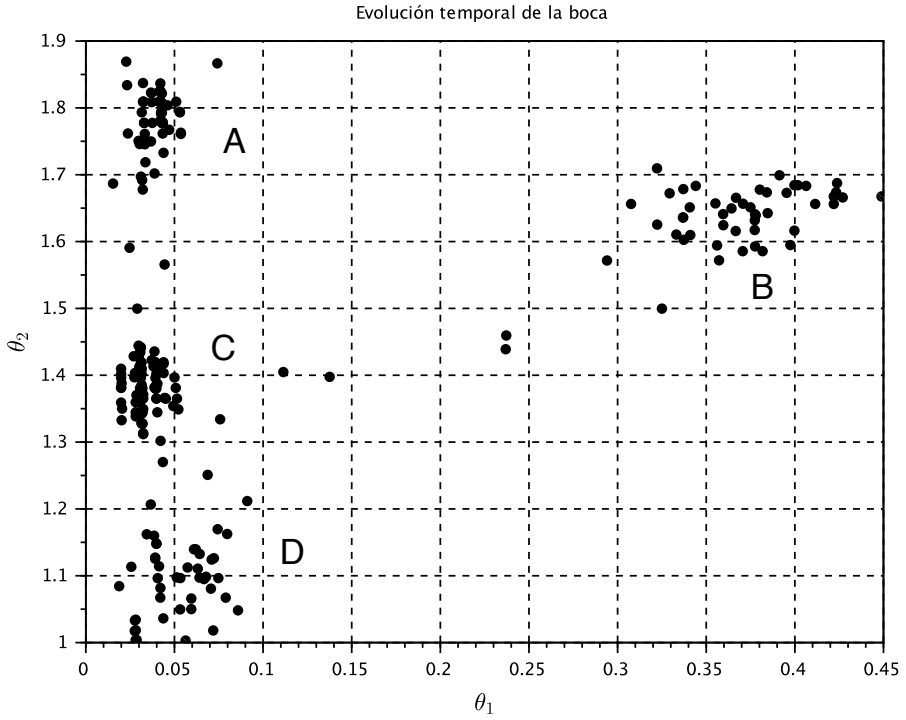


Figura 5-19: La gráfica muestra la trayectoria de la boca paramétrica en la variable del tiempo según dos variables: θ_1 representa la longitud vertical de la boca sobre la longitud horizontal de la misma y θ_2 la longitud horizontal sobre la distancia entre los ojos. Se logran visualizar claramente cuatro regiones. La región A se corresponde con el estado de la boca "sonriendo", la región B se corresponde con el estado "abierta", la región C con el estado "en reposo", y la región D con el estado "beso". Se pueden observar también algunas muestras cuando el usuario transita de un gesto a otro.

5.2. Detección de gestos como problema de clasificación

Independientemente de las observaciones realizadas en Scilab para las medidas de la boca y otras partes de la cara, desde el planteo del problema, se puede identificar la detección de gestos como un problema de clasificación. Puesto que, dada ya una representación del estado de la cara (mediante *landmarks*), se desea saber si se encuentra en determinado estado (boca abierta, boca sonriente, cejas levantadas, etc.).

Los algoritmos clásicos de clasificación como pueden ser los regresores logísticos, SVM, o redes neuronales, exigen que en la etapa de entrenamiento se etiqueten muestras que pertenecen a la clase y muestras que no. Luego, intentan encontrar una función que mejor clasifique nuevos datos. Sin embargo, en la aplicación que será la detección de gestos, este enfoque no es adecuado. Esto es porque no se conocen todos los estados posibles que podrá adoptar la cara cuando no desea representar ningún gesto.

Se busca entonces encontrar un algoritmo que permita clasificar un gesto pura y exclusivamente a partir de los datos recabados al pedir al usuario que lo realice. Esto permite que el algoritmo se adapte perfectamente a sus necesidades, sin requerir que se sepa en qué estados se encontrará la cara cuando no está realizando ningún gesto. Esto, junto con observaciones como las de la Fig. 5-19, sugieren modelar el problema de la forma en la que se explica en las siguientes secciones.

5.3. Modelado del estado de las partes de la cara

Cada parte de la cara sobre la que se quieran detectar gestos concretos (boca, cejas, etc.) puede ser descrita por un conjunto de cantidades reales (medidas de área o longitudes de la cara) determinadas por las medidas realizadas por la cámara, que se denominan descriptores θ_i. Así, cada parte de la cara vendrá caracterizada por un conjunto de n descriptores que serán las componentes de un vector al que se denominará "vector descriptor" Θ:

$$\Theta = [\theta_1 \cdots \theta_n]^T \in \mathbb{R}^n \tag{5-7}$$

El vector se puede expresar como la suma de dos componentes: el estado que desea representar el usuario en el instante t, denominado $s\,(t)$, y una señal de ruido dada por el error introducido por la medida realizada por la cámara, el error en la convergencia de los algoritmos, y el ruido que el usuario podría introducir debido a problemas de motricidad (por lo tanto también depende de s),$n\,(s, x)$. Se tiene entonces:

$$\Theta \sim s + n\left(s, x\right) \tag{5-8}$$

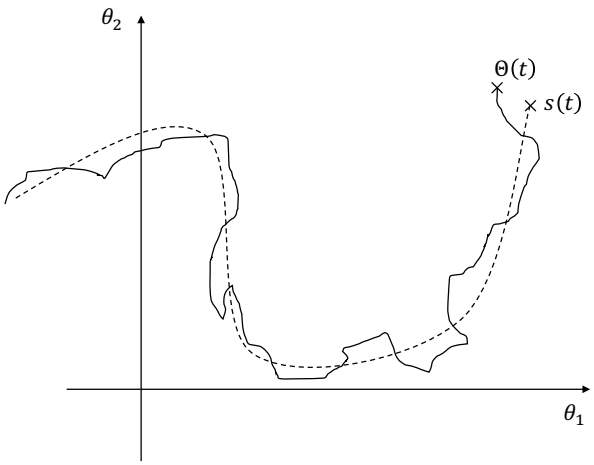


Figura 5-20: Imagen representativa del comportamiento de $s\left(t\right)$ y $\Theta\left(t\right)$, con $\Theta = \left[\theta_1, \theta_2\right]^T \in \mathbb{R}^2$. En la medida en la que evoluciona el tiempo t.

5.4. Distribución gaussiana multivariable

El ruido n introducido por la cámara, la convergencia de los algoritmos, y los posibles problemas motrices que pueda tener el usuario se modelará como una distribución gaussiana multivariable:

$$n\left(s, x\right) \sim N\left(x; 0, \Sigma\left(s\right)\right) = \frac{1}{\sqrt{\left(2 \cdot \pi\right)^n \cdot \left|\Sigma\left(s\right)\right|}} \cdot e^{-\frac{1}{2} \cdot x^T \cdot \Sigma^{-1}(s) \cdot x} \tag{5-9}$$

donde Σ representa la matriz de covarianza, que será también una función de s, puesto que los errores introducidos por los algoritmos o los problemas motrices del usuario pueden depender del gesto que se quiera realizar con la cara. De las ecuaciones anteriores se puede determinar que Θ tendrá una distribución dada por:

$$\Theta \sim N\left(x; s, \Sigma\left(s\right)\right) \tag{5-10}$$

5.5. Gestos, curvas de nivel y distancia de Mahalanobis

Se supone entonces que se quiere representar un gesto, en el que el estado de la parte de la cara pretende mantenerse fijo en un punto $s(t) = \mu \in \mathbb{R}^n$. En este caso, la distribución del vector descriptor de la parte de la cara, de la ecuación 5-10 vendrá dado por:

$$\Theta \sim N(x; \mu, \Sigma) \tag{5-11}$$

donde ahora Σ se mantiene fija. Este gesto, se caracterizará entonces por dos parámetros: el centroide μ y la varianza entorno a μ producida por los algoritmos, la cámara y el usuario Σ.

Igualando la ecuación 5-11 a un valor genérico α, y aplicando logaritmos a ambos lados de la expresión se llega a que las curvas de nivel para la distribución de probabilidades de Θ cuando se desea efectuar un gesto $s(t) = \mu$ vienen dadas por:

$$(x-\mu)^T \cdot \Sigma^{-1} \cdot (x-\mu) = Ln\left[(2\cdot\pi)^n \cdot |\Sigma| \cdot \alpha^2\right] \tag{5-12}$$

y puesto que Σ es definida positiva, la solución a esta ecuación determina elipses en el espacio $\mathbb{R}^n$ donde se mueve Θ. La ecuación anterior se puede interpretar como una medida de distancia llamada "distancia de Mahalanobis" [36][37], dada por:

$$d_{\Sigma_i}(x, \mu_i) = \sqrt{(x-\mu_i)^T \cdot \Sigma_i^{-1} \cdot (x-\mu_i)} \tag{5-13}$$

Esto se ilustra en la imagen: 5-21

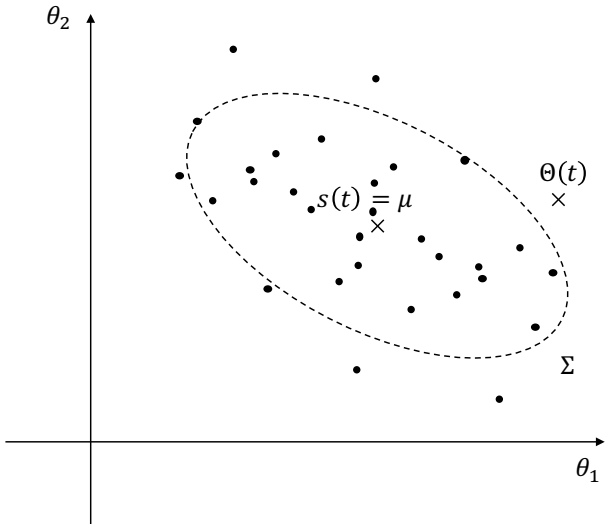


Figura 5-21: Imagen representativa del comportamiento de $s\,(t)$ y $\Theta\,(t)$, teniendo $\Theta = [\theta_1, \theta_2]^T \in \mathbb{R}^2$. En este caso se tiene $s\,(t) = \mu$ constante, representando un gesto, las muestras de Θ se representan como puntos en el espacio y la elipse punteada representa la solución a la ecuación 5-13 para una distancia de Mahalanobis, en la distribución $\Theta \sim N\,(x; \mu, \Sigma)$.

5.6. Cálculo de radios

En base a la ecuación 5-13 se puede determinar una distancia δ (que se denominará radio) en la que si $d_{\Sigma_i}\,(\Theta, \mu_i) < \delta$ el gesto se dará como activo. Los radios se determinan de forma que se tenga por dentro de las elipses cierta fracción $0 < \tau < 1$ del valor de la integral de la gaussiana en todo el espacio. De esta forma, los umbrales $\delta > 0$ para cada τ se determinan como:

$$\min_{\delta} \left\{ \int_{d_{\Sigma_i}(x,\mu_i)<\delta} N\,(x; \mu_i, \Sigma_i)\, dx_1 \cdots dx_n \geq \tau \right\} \tag{5-14}$$

Dado que $N\,(x; \mu_i, \Sigma_i) > 0$ para todo $x \in \mathbb{R}^n$, la integral anterior es estrictamente creciente en δ, y puesto que la integral está acotada entre 0 y 1, el problema tiene solución y es única. Dada la complejidad del problema anterior, se aproxima por otra integral (ver Anexo B para la deducción), que termina siendo:

$$\min_{\delta} \left\{ \int_0^{\delta} N\,(u; 0, 1)\, du \geq \frac{\sqrt[n]{\tau}}{2} \right\} \tag{5-15}$$

De 5-15 se deduce que el límite δ depende únicamente de la dimensión n del vector de descriptor Θ. Se puede ver que esta integral es una integral en una variable, mientras que la anterior es una integral en n variables en un dominio elipsoidal. Nuevamente, dado que $N(u;0,1) > 0$ para todo $u \in \mathbb{R}$, la expresión anterior es estrictamente creciente en δ, y dado que la integral esta acotada entre 0 y 1, el problema tiene solución y es única.

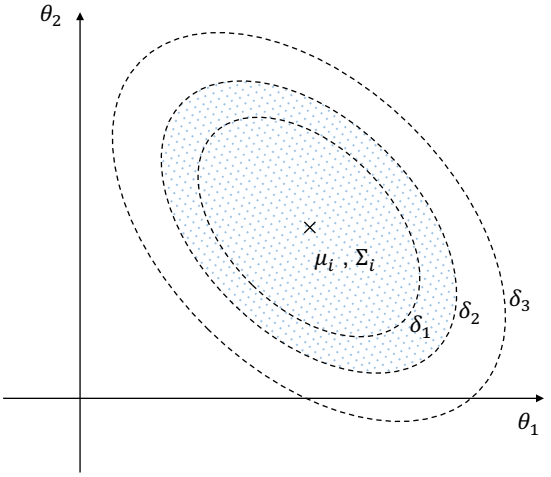


Figura 5-22: Representación de un gesto y diferentes elipses de nivel $\delta_1 < \delta_2 < \delta_3$ dadas por la ecuación 5-13, y sombreado el conjunto $d_{\Sigma_i}(x, \mu_i) < \delta_2$.

5.7. Histéresis

Para eliminar las entradas y salidas continuas a los gestos debido al ruido de $\Theta(t)$ en las cercanías del borde de la elipse, se utilizan dos radios para cada gesto δ_{in} y δ_{out} con $\delta_{in} < \delta_{out}$, de forma que el gesto se detectará como activo cuando $d_{\Sigma_i}(\Theta, \mu_i) < \delta_{in}$, y dejará de determinarse como activo cuando $d_{\Sigma_i}(\Theta, \mu_i) > \delta_{out}$. El radio de salida δ_{out}, se determina de forma que $\delta_{out} = K \cdot \delta_{in}$ con $K > 1$.

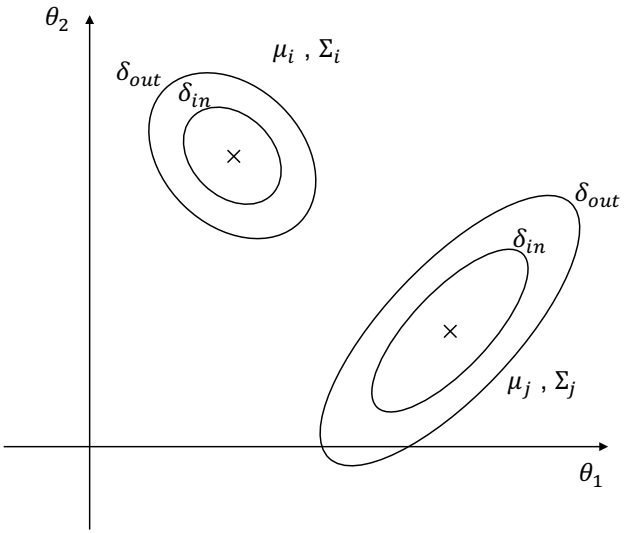


Figura 5-23: Imagen representativa del espacio de Θ con dos gestos y sus elipses de entrada y salida.

5.8. Interferencia entre gestos

La histéresis es útil siempre y cuando se cumpla además que para dos gestos, los conjuntos determinados por el interior de sus elipses de entrada y salida sean disjuntos uno a uno. En otras palabras, se podría decir que dos gestos no interfieren entre sí, sí y solo sí ninguno de los siguientes sistemas tiene solución en x:

$$\begin{cases} d_{\Sigma_i}(x, \mu_i) - \delta_{in} & < 0 \\ d_{\Sigma_j}(x, \mu_j) - \delta_{out} & < 0 \end{cases} \tag{5-16}$$

$$\begin{cases} d_{\Sigma_i}(x, \mu_i) - \delta_{out} & < 0 \\ d_{\Sigma_j}(x, \mu_j) - \delta_{in} & < 0 \end{cases} \tag{5-17}$$

Dado que clasificar este sistema resulta complejo, en la etapa de entrenamiento (luego de entrenar el gesto, pero antes de guardarlo) se utiliza una simplificación a la condición anterior. Se realiza aquí una nueva suposición: el usuario pasa únicamente de un gesto a otro, de modo que $s(t)$ se mueve en linea recta entre centroides de distintos gestos. Es en este sentido en el que se desea que los gestos no interfieran.

Bajo estas hipótesis la condición impuesta en las ecuaciones anteriores pasa a ser (ver Anexo C para la deducción):

$$\begin{cases} \frac{\delta_{in}}{d_{\Sigma_i}(\mu_j,\mu_i)} + \frac{\delta_{out}}{d_{\Sigma_j}(\mu_i,\mu_j)} & < 1 \\ \frac{\delta_{out}}{d_{\Sigma_i}(\mu_j,\mu_i)} + \frac{\delta_{in}}{d_{\Sigma_j}(\mu_i,\mu_j)} & < 1 \end{cases} \tag{5-18}$$

5.9. Entrenamiento del algoritmo

El algoritmo de detección de gestos funciona en dos etapas.

Una primer etapa de configuración de los gestos, en la que, en función de la dimensionalidad del vector Θ, se determinan δ_{in} y δ_{out}.

Luego, una segunda etapa de entrenamiento en la que se determinan los centroides μ_i y las matrices de covarianza Σ_i para cada uno de los gestos. Esto se hará generando un vector de medidas de entrenamiento $V : V_k = \Theta(t_k)$ de m muestras, con $s(t) = \mu$ fijo, y luego se determinan el centroide y la covarianza utilizando los estimadores de los parámetros de la distribución dada en la ecuación 5-11.

Los estimadores vienen determinados por :

$$\mu = \frac{1}{m} \cdot \sum_{i=k}^{m} V_k \tag{5-19}$$

$$\Sigma_{ij} = Cov(\theta_i, \theta_j) \tag{5-20}$$

donde se toman los descriptores θ_i y θ_j generados en sus m muestras. Una vez estimada la matriz de covarianza Σ se procede a calcular su inversa utilizando los algoritmos de Dlib, y se guardan ambas variables en la estructura de datos para luego utilizarlas para clasificar.

Luego de hecho esto se verifica que este nuevo gesto no interfiera con ningún otro.

5.10. Implementación y verificación del modelo en Scilab

Luego de elaborar y trabajar en el modelo, se procedió a analizar e implementar todos los algoritmos en Scilab, para luego pasarlos a C++/C. En particular, dado

lo anterior(ver Fig. 5-19), se puede ver como el modelo puede aplicarse de manera efectiva sobre las medidas realizadas sobre la boca y sus estados.

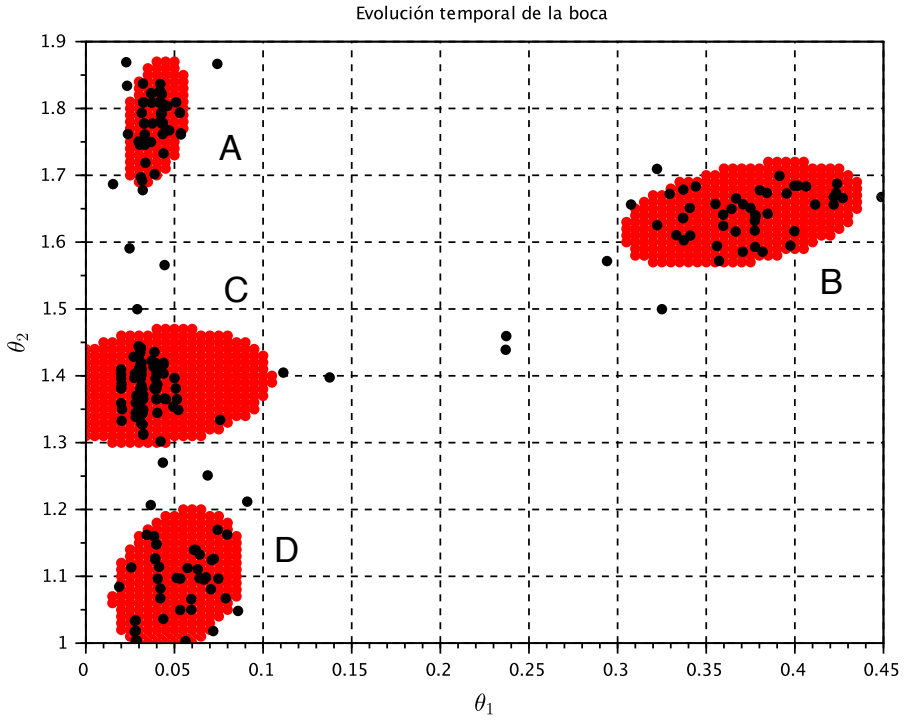


Figura 5-24: La gráfica muestra la trayectoria de la boca en paramétrica en la variable del tiempo según dos variables: θ_1 representa la longitud vertical de la boca sobre la longitud horizontal de la misma y θ_2 la longitud horizontal sobre la distancia entre los ojos. Se logran visualizar claramente cuatro regiones. La región A se corresponde con el estado de la boca «sonriendo», la región B se corresponde con el estado «abierta», la región C con el estado «en reposo», y la región D con el estado «beso». En rojo se representan los puntos del espacio que verifican $d_{\Sigma_i}(x, \mu_i) < \delta$, para alguno de los gestos, con un δ escogido arbitrariamente.

Del análisis de la gráfica anterior se desprenden una serie de conclusiones que verifican algunas de las hipótesis realizadas con respecto al modelo:

1. A la hora de querer expresar un gesto, se puede observar que el conjunto de medidas Θ, oscilan entorno a un punto en concreto μ.

2. Las regiones para cada gesto son aproximables por elipses (en términos visuales), fruto de aproximar el modelo de ruido por un modelo gaussiano multivariable.

3. El ruido entorno a determinado centroide μ_i depende del gesto que se quiere

determinar. Esto verifica la hipótesis de que la matriz de covarianza es una función de s, y se ve reflejado en la variabilidad de la forma de las elipses.

5.11. Funcionamiento del algoritmo

Una vez entrenado el algoritmo, en cada iteración se determina el vector descriptor Θ. Un gesto se considera activo si cumple que la distancia de Mahalanobis a su centroide es menor a la distancia de entrada (δ_{in}). Una vez activo el gesto, se dejará de tener como activo cuando el vector descriptor supere la distancia de salida al centroide (δ_{out}).

5.12. Descriptores y gestos utilizados

A continuación se ilustrarán los gestos que el sistema puede detectar actualmente. Se indicarán los *landmarks* y descriptores utilizados, y cómo cambia el vector Θ en los diferentes gestos.

Estos gestos fueron seleccionados después del estudio realizado en Scilab pero no son los únicos que se pueden detectar. Con un conocimiento básico del funcionamiento del programa debería ser sencillo modificar o agregar gestos y descriptores para que se adapte mejor a las condiciones del usuario.

Las elecciones de partes y gestos fue el resultado del estudio realizado en Scilab sobre la evolución de los *landmarks* mientras distintas personas realizaron los gestos bajo diferentes condiciones. Los descriptores utilizados son los que permiten identificar de forma más adecuada a cada gesto, aunque como se ha mencionado, para detectar nuevos gestos podría hacerse un estudio para ver qué descriptores se pueden agregar.

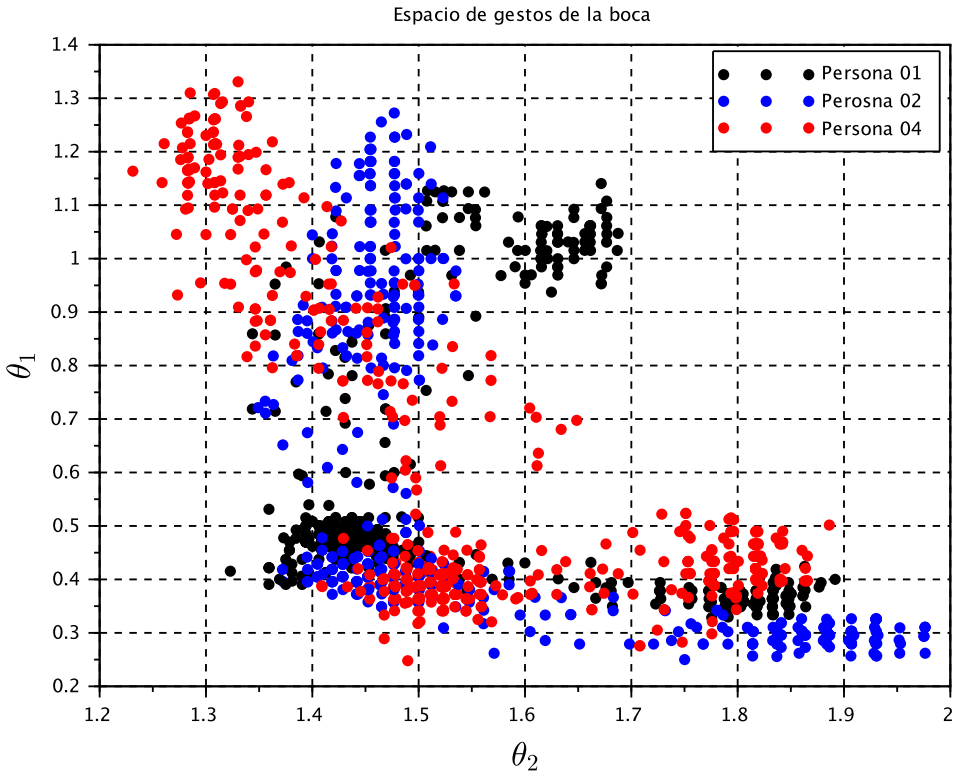


Figura 5-25: La gráfica muestra la trayectoria de la boca de tres personas paramétrica en la variable del tiempo según dos variables: θ_1 representa la longitud vertical de la boca sobre la distancia entre los ojos y θ_2 la longitud horizontal sobre la distancia entre los ojos. Se ve que aunque los centroides no coinciden de persona a persona (haciendo el entrenamiento indispensable) sí se pueden definir tres centroides para cada usuario.

A lo largo de esta sección se denominará l_k al *landmark* de índice k según la figura 4-17.

5.12.1. Criterios para la selección de gestos y descriptores

Al seleccionar un gesto se debe tener en cuenta que los gestos se utilizarán como comandos para distintas aplicaciones. Durante el estudio y el discernimiento sobre los gestos que se tomarán como detectables, se encontraron algunas características importantes que comparten los gestos posibles de detectar. Además se considera necesario definir siempre un gesto "en reposo" asociado al estado de la parte de la cara cuando no se desea representar ningún gesto.

Las características de un gesto detectable se enumeran a continuación.

1. Gesto simple de realizar.
2. Gesto fácil de detectar, teniendo en cuenta el modelado ASM de la cara.
3. Debe encontrarse una medida normalizada que caracterice el gesto cualquiera sea la posición de la cara en el cuadro, y su distancia a la cámara.

A su vez, existen algunos criterios que ayudan a elegir descriptores para la detección de gestos.

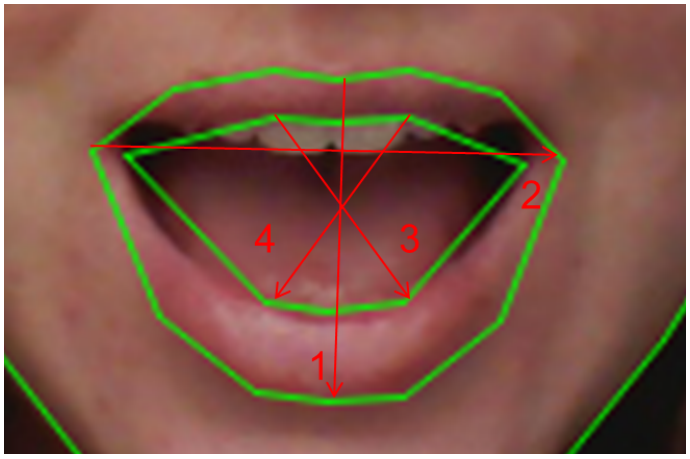
1. Una medida que refleje cambios observables cuando se realiza el gesto, y que el ruido no permita que pase de un estado a otro.
2. Los descriptores suelen normalizarse frente a otra medida de la cara que permanece fija (frente a diferentes orientaciones de la cara y cuando se realizan otros gestos), de forma de dar robustez al detector frente a la distancia del usuario a la cámara.

5.12.2. Boca

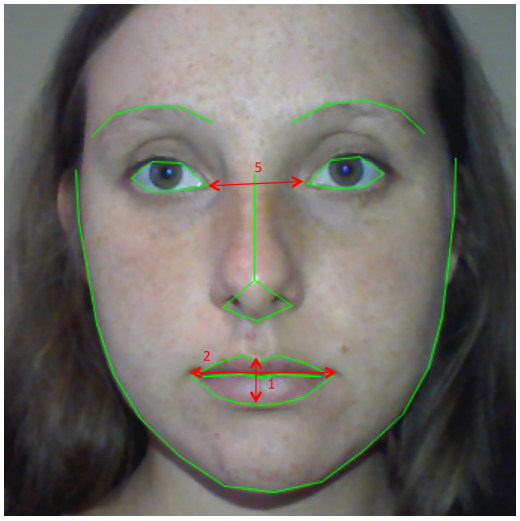
Para el estudio de la boca inicialmente se tomó la relación entre la distancia horizontal entre los bordes de la boca sobre la distancia entre los ojos y la distancia vertical sobre la distancia entre los ojos; que es suficiente para identificar los gestos. Sin embargo, para evitar la interferencia entre ellos, luego de agregar la histéresis se agregaron algunos descriptores.

Los gestos detectables para la boca serán (ver Fig. 5-26):

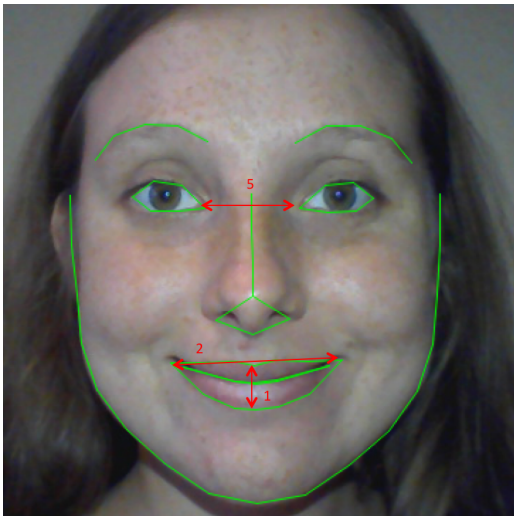
- Boca en reposo
- Boca en sonrisa
- Boca en beso
- Boca abierta



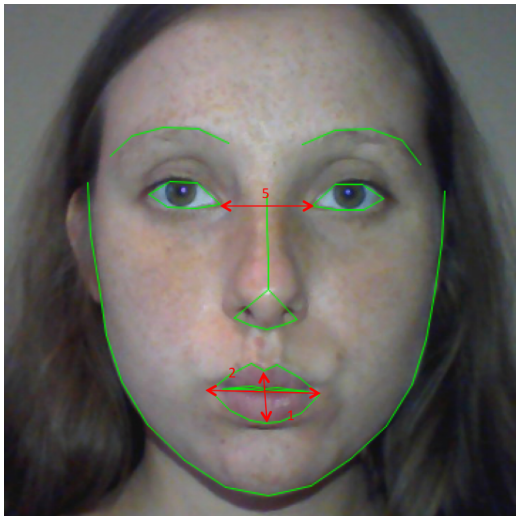
(a) Caracterización de la boca.



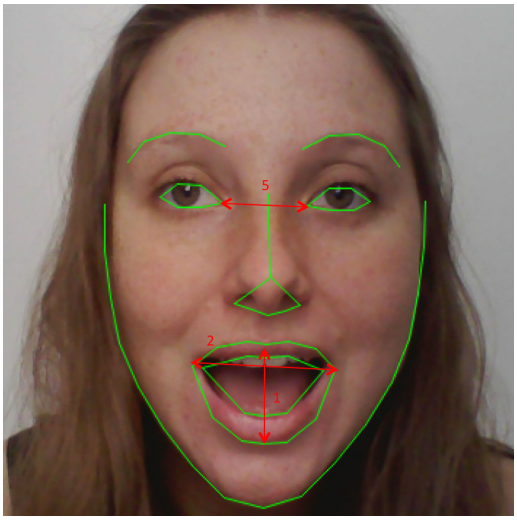
(b) Boca en reposo.



(c) Boca en sonrisa.



(d) Boca en beso.



(e) Boca abierta.

Figura 5-26: Descriptores y gestos posibles para la boca.

Finalmente los descriptores utilizados son:

$$\theta_1 = \frac{\text{Distancia vertical}}{\text{Distancia entre ojos}} = \frac{|\vec{1}|}{|\vec{5}|} = \frac{\|l_{52} - l_{58}\|}{\|l_{42} - l_{39}\|} \tag{5-21}$$

$$\theta_2 = \frac{\text{Distancia horizontal}}{\text{Distancia entre ojos}} = \frac{|\vec{2}|}{|\vec{5}|} = \frac{\|l_{55} - l_{49}\|}{\|l_{42} - l_{39}\|} \tag{5-22}$$

$$\theta_3 = \text{Suma de diagonales} = \frac{|\vec{3}| + |\vec{4}|}{|\vec{5}|} = \frac{\|l_{68} - l_{64}\| + \|l_{66} - l_{62}\|}{\|l_{42} - l_{39}\|} \tag{5-23}$$

$$\theta_4 = \frac{\theta_2}{\theta_1^2} \tag{5-24}$$

Podrían agregarse más gestos para la boca, agregando nuevos descriptores que permitan determinar asimetrías, para detectar por ejemplo muecas hacia la derecha hacia la izquierda, o distintas formas de mover los labios.

En la figura 5-27 se observa un conjunto de muestras de Θ para una tira de datos mientras se registran los cuatro gestos. Se puede observar cómo existen cuatro zonas bien diferenciadas para los cuatro gestos. A partir de la definición de los descriptores y observando la gráfica se puede intuir que el conjunto central, con más puntos, es la boca en reposo, mientras que el conjunto superior es la boca abierta. Los conjuntos más hacia la derecha y a la izquierda se corresponden con la boca en sonrisa y en beso respectivamente (en la sección D.4 se tiene un análisis más profundo realizado sobre la boca en Scilab).

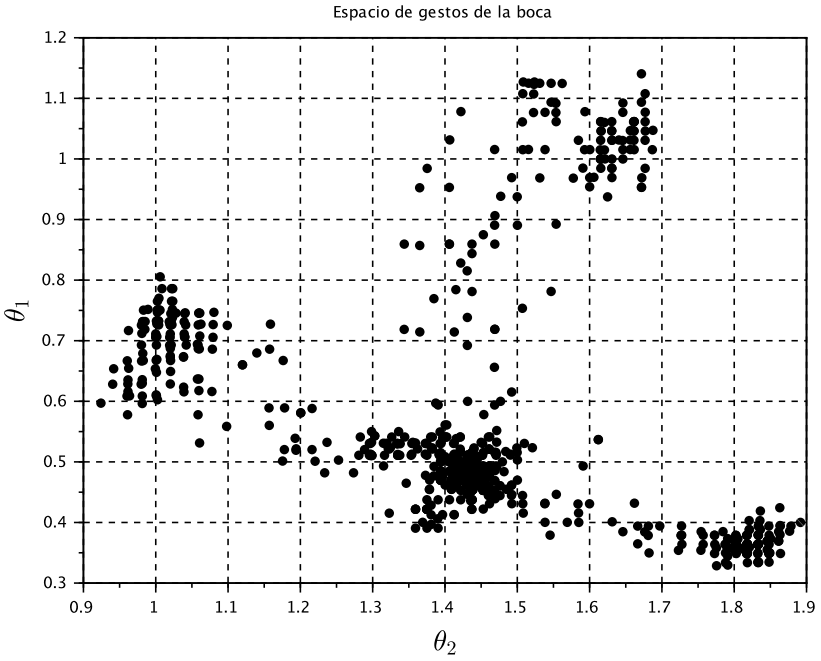
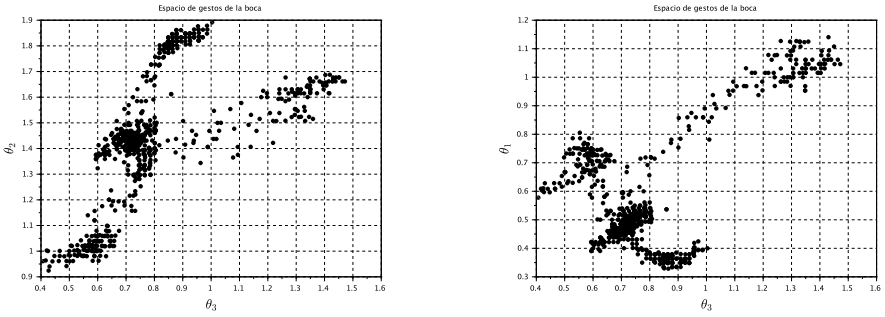


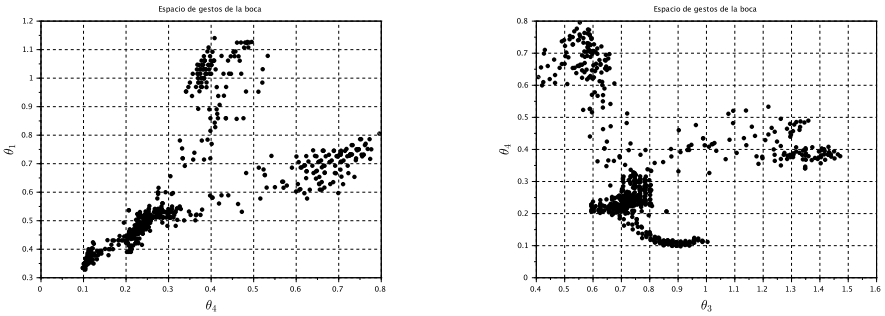
Figura 5-27: Muestras de Θ para la boca, mientras se realizan los cuatro gestos posibles. La gráfica muestra la proyección del vector descriptor en el plano generado por θ_1 y θ_2.

Los descriptores θ_1 y θ_2 no son suficientes para determinar el estado de la boca. En la práctica el gesto "beso" interfiere con el estado de la boca en reposo (se puede observar que la nube de puntos asociados al gesto es bastante difusa), por lo que se tuvieron que agregar los descriptores θ_3 y θ_4 que separan los conjuntos. La figura 5-28 muestra la proyección de Θ para los distintos estados de la boca sobre distintos planos. En la misma se puede ver cómo el gesto beso se separa del estado normal de la boca en la proyección sobre el plano formado por θ_1 y θ_4, o en el plano formado por θ_4 y θ_3, puede ser útil observar para esto que:

$$\mu_{beso} \approx \begin{bmatrix} 0,7 \\ 1 \\ 0,6 \\ 0,7 \end{bmatrix} \tag{5-25}$$



(a) θ_2 y θ_3 (b) θ_1 y θ_3



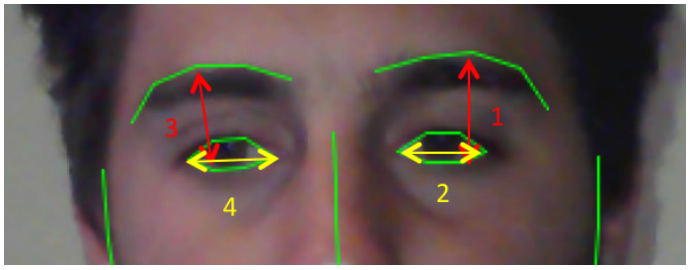
(c) θ_1 y θ_4 (d) θ_4 y θ_3

Figura 5-28: Muestras de Θ para la boca, mientras se realizan los cuatro gestos posibles. Proyecciones en distintos planos.

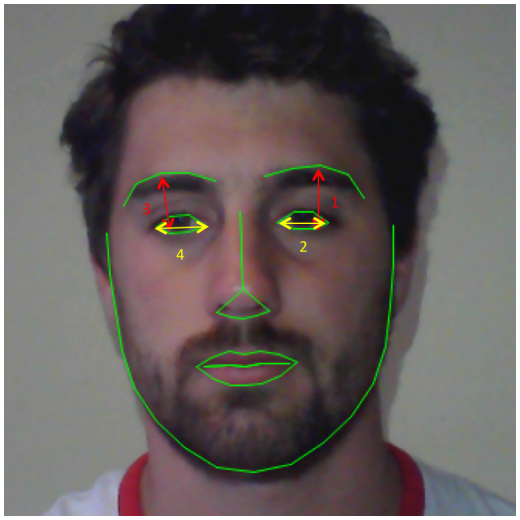
5.12.3. Cejas

Para detectar movimientos de las cejas se analiza la distancia vertical entre las mismas y el borde inferior del ojo. En este caso se usa el largo horizontal de los ojos como normalizador.

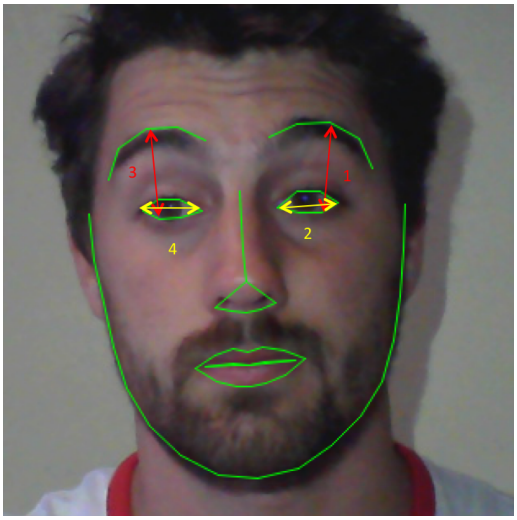
Se extrae la misma información de ambos ojos para tener más datos a analizar y para poder contemplar el caso de que el usuario solo tenga movilidad de un hemisferio de la cara. Los gestos a detectar serán (ver Fig. 5-29):



(a) Caracterización de las cejas.



(b) Cejas en reposo.



(c) Cejas hacia arriba.

Figura 5-29: Descriptores y gestos posibles para las cejas.

Los descriptores utilizados son [39]:

$$\theta_1 = \frac{|\vec{1}|}{|\vec{2}|} = \frac{\|l_{25} - l_{47}\|}{\|l_{43} - l_{46}\|} \tag{5-26}$$

$$\theta_2 = \frac{|\vec{3}|}{|\vec{4}|} = \frac{\|l_{20} - l_{42}\|}{\|l_{40} - l_{37}\|} \tag{5-27}$$

Con estos descriptores se podría agregar la detección de un gesto no simétrico en conjunto con los existentes (por ejemplo levantar solo la ceja derecha). En la Fig. 5-30 se puede observar el espacio de Θ y los gestos posibles.

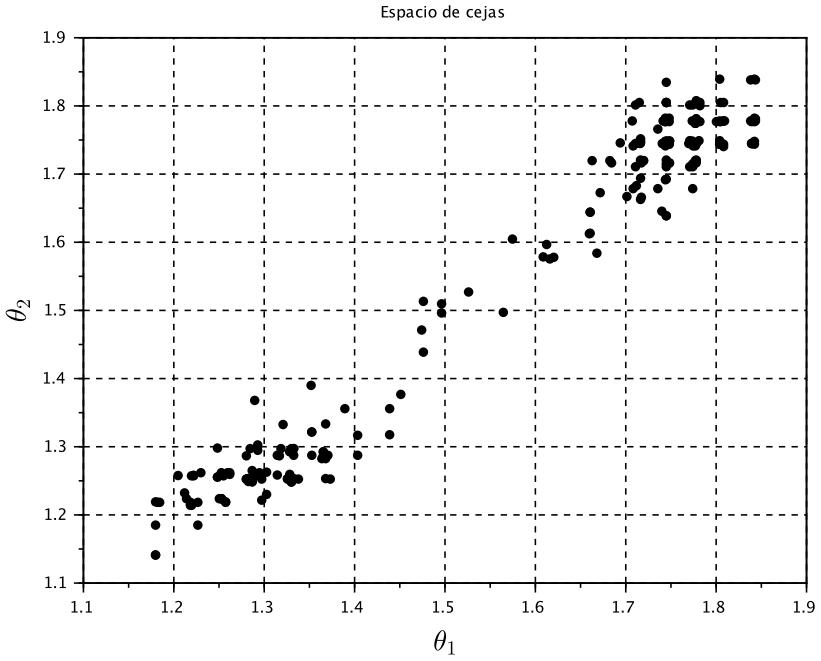
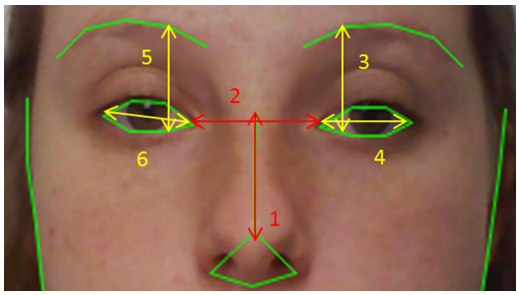


Figura 5-30: Muestras de Θ para las cejas, mientras se realiza el gesto de levantar las cejas y volver al estado en reposo.

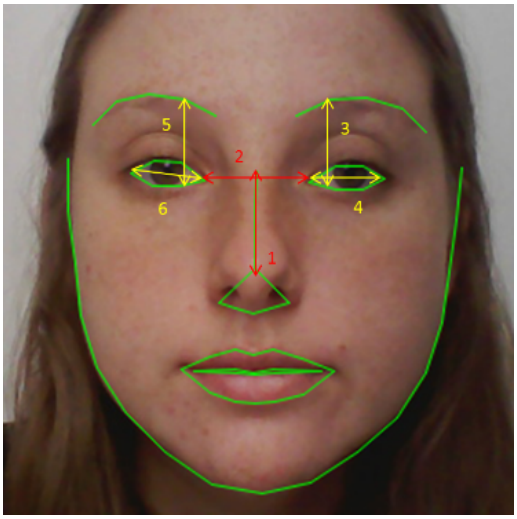
5.12.4. Orientación

Para estudiar la orientación de la cara en sentido horizontal es suficiente con analizar la proyección del vector de la nariz sobre el vector que va de un ojo al otro.

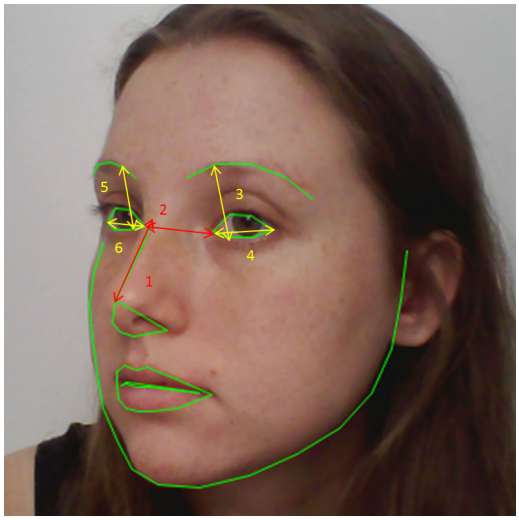
Para estudiar la orientación vertical de la cara se utilizan los mismos vectores pero se analizan sus módulos, es decir, el largo del vector de la nariz sobre la distancia entre los ojos. Para complementar el estudio en esta dirección también se utiliza la distancia entre las cejas y el borde inferior del ojo normalizado con el largo del mismo (ver Fig. 5-31).



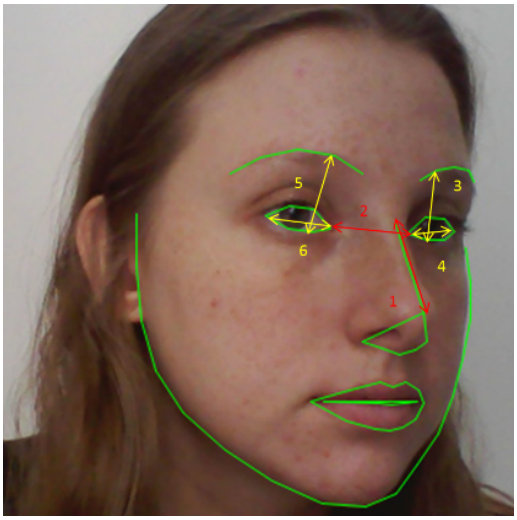
(a) Descriptores para la orientación.



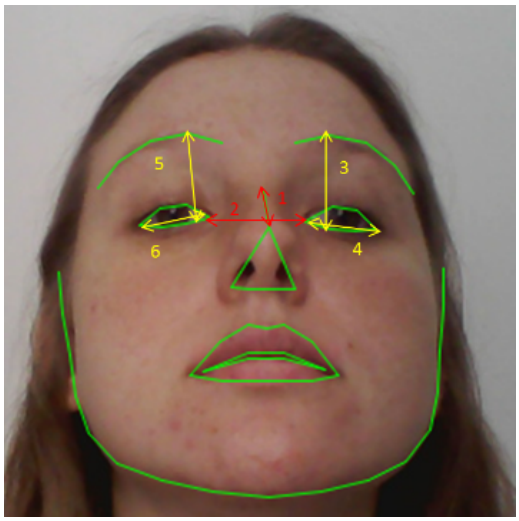
(b) Orientación hacia el frente.



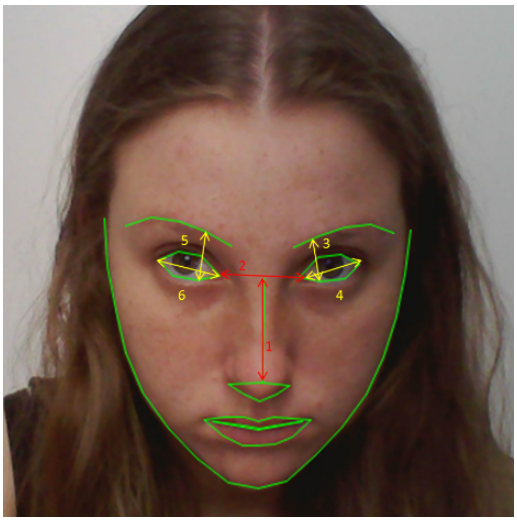
(c) Orientación hacia la derecha.



(d) Orientación hacia la izquierda.



(e) Orientación hacia arriba.



(f) Orientación hacia abajo.

Figura 5-31: Descriptores y gestos de la orientación.

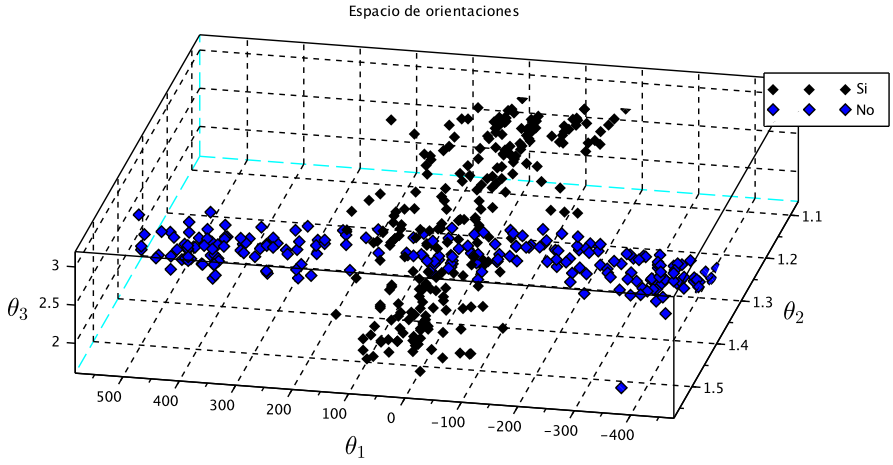
Los descriptores utilizados:

$$\theta_1 = \text{Proyección de nariz sobre ojos} =< \vec{1}, \vec{2} >= (l_{31} - l_{28})^T \cdot (l_{40} - l_{43}) \quad (5\text{-}28)$$

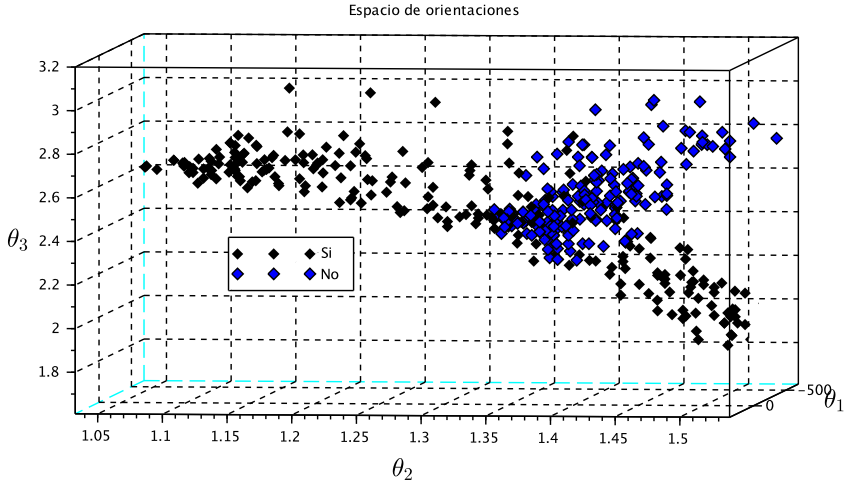
$$\theta_2 = \frac{\text{Longitud de nariz}}{\text{Distancia entre ojos}} = \frac{|\vec{1}|}{|\vec{2}|} = \frac{\|l_{31} - l_{28}\|}{\|l_{40} - l_{43}\|} \quad (5\text{-}29)$$

$$\theta_3 = \text{Altura de cejas} = \frac{|\vec{3}|}{|\vec{4}|} + \frac{|\vec{5}|}{|\vec{6}|} = \frac{\|l_{24} - l_{88}\|}{\|l_{46} - l_{43}\|} + \frac{\|l_{21} - l_{41}\|}{\|l_{40} - l_{37}\|} \quad (5\text{-}30)$$

El estudio en Scilab para la elección de los descriptores se muestra en la Fig. 5-32. Se puede ver que en el plano conformado por θ_1 y θ_2 (ver Fig. 5-32a), la nube de puntos asociada al movimiento de asentir y al movimiento de disentir van en direcciones ortogonales. La zona de intersección entre ambas nubes se corresponde con la cara en orientación hacia el frente. Dado que la orientación hacia abajo se encuentra muy cercana a la orientación hacia al frente, se agregó un tercer descriptor θ_3, que permitió separar los conjuntos de estos dos gestos (ver Fig. 5-32b).



(a)



(b)

Figura 5-32: Muestras de Θ para la orientación, mientras se realizan los gestos de asentir y disentir.

5.12.5. Efecto de la distancia a la cámara

Muchos de los descriptores que se utilizaron, intentan normalizar cierta medida de la cara variable (con la que se recrean gestos), frente a otra fija (con la que se normaliza). Tal es el caso de la boca en sus *landmarks* θ_1 y θ_2, que miden las longitudes horizontales y verticales de la boca en función de la distancia de los ojos (ver ecuaciones 5-21 y 5-22).

Básicamente estos descriptores pueden escribirse genéricamente como:

$$\theta = \frac{L_1}{L_2} \tag{5-31}$$

donde L_i representa longitudes sobre la cara. Aplicando la regla de propagación de errores[4], se tiene que:

$$\Delta\theta \approx \frac{1}{L_2}\sqrt{\Delta L_1^2 + \frac{L_1^2 \cdot \Delta L_2^2}{L_2^2}} \tag{5-32}$$

lo que muestra que el error introducido en los descriptores cuando el objeto es muy lejano ($L_i \to 0$), aumenta linealmente con L_2^{-1} (suponiendo que los errores se mantienen constantes). Esto se puede ver en la Fig. 5-33 que muestra una gráfica comparativa de los gestos "sonreír" y "abrir" para la boca, en función de la distancia[5]. Se ve cómo en la medida en que aumenta la distancia, gracias a la normalización de los centroides, los gestos parecen estar ubicados en zonas similares, pero las nubes de puntos son más difusas que a distancias cortas.[6]

[4]Dada una medida indirecta z determinada por $z = f(x_1, \ldots, x_N)$, donde cada x_i es una medida con error Δx_i, el error introducido por f en z puede aproximarse [38] por $\Delta z \approx \sqrt{\sum_{i=1}^{N} \left(\partial f/\partial x_i\,(x_1, \ldots, x_N)\right)^2 \cdot \Delta x_i^2}$.

[5]Los datos utilizados aquí fueron según la persona 01 de la base de datos compilada.

[6]La razón por la que el centroide azul se ve más lejos de los otros debe ser debido a qué la persona no repitió el gesto perfectamente a esta distancia.

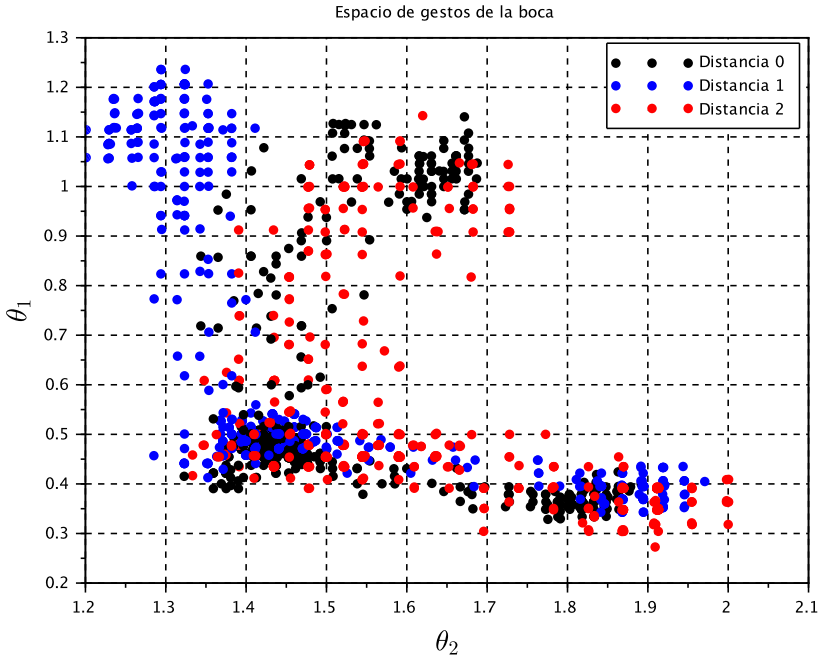


Figura 5-33: Muestras de Θ para la boca, para cada una de las tres distancias mientras se realizan los gestos "abrir" y "sonreír" para la boca.

5.13. Determinación de los parámetros de funcionamiento del sistema

En el diseño del algoritmo de detección de gestos, surgen dos parámetros utilizados para determinar el tamaño de las elipses y su histéresis:

1. El parámetro τ para el cálculo de la integral de la gaussiana, que determina los radios de entrada a las elipses δ_{in}.
2. El parámetro K para el cálculo de los radios de salidas de los gestos δ_{out}.

La forma de determinar estos parámetros fue absolutamente experimental, tomando como criterio la calidad del uso del programa. En definitiva, se determinaron los parámetros mediante prueba y error y se llegó a la conclusión de que los parámetros a utilizar serían:

$$\tau = 0,95 \tag{5-33}$$

$$K = 2,5 \tag{5-34}$$

El software fue testeado en tres computadoras con distintas especificaciones y distintas cámaras web, y se llegó a la conclusión de que la experiencia de usuario para el detector de gestos era exactamente la misma. Estos parámetros sin embargo, son configurables por el programador en la etapa de entrenamiento, permitiéndole incluso seleccionar diferentes parámetros para diferentes gestos.

6. Seguimiento de la cara (*Tracking*)

El mayor desafío introducido por la librería Dlib consistió en que el tiempo de procesamiento era muy lento para las especificaciones propuestas ($110ms$ por cuadro, claramente no tiempo real).

Inicialmente, se implementaron estrategias para acelerar la detección de *landmark*s de Dlib. Por ejemplo, disminuir el tamaño de la imagen para procesar menos información. Reduciendo el tamaño de la imagen a un cuarto del original se acelero la detección de $110ms$ a $70ms$ por cuadro (157 %). De todas formas aún no era suficiente.

Probamos algoritmos de OpenCV y Dlib para el *tracking* de objetos dado un recuadro inicial, pero los tiempos no fueron complacientes a las especificaciones que se buscaban.

Se detectó finalmente, que el cuello de botella del sistema es el tiempo empleado en la búsqueda de caras en todo el cuadro. Búsqueda que devuelve un rectángulo dentro del cual se hace la convergencia de *landmarks*. Por lo tanto, se diseñó un sistema que realimente el resultado obtenido en la convergencia de los *landmarks* para predecir un nuevo rectángulo en el que sea probable encontrar la cara en el siguiente cuadro. Se ahorra de esta forma el procesamiento de la búsqueda de caras en cada cuadro.

6.1. Algoritmo de *tracking*

El algoritmo de *tracking* implementado se basa en definir un área donde es probable que se encuentre la cara en el siguiente cuadro, basada en la posición de la cara en el cuadro actual, dada por sus *landmarks.*

El algoritmo devuelve un rectángulo a partir de los *landmarks*, dejando un margen en caso de que el usuario se desplace en la pantalla. Ese rectángulo va a ser el punto de partida para el algoritmo de Dlib que detecta *landmarks* sobre el siguiente cuadro, de forma que siempre se sigue a la cara.

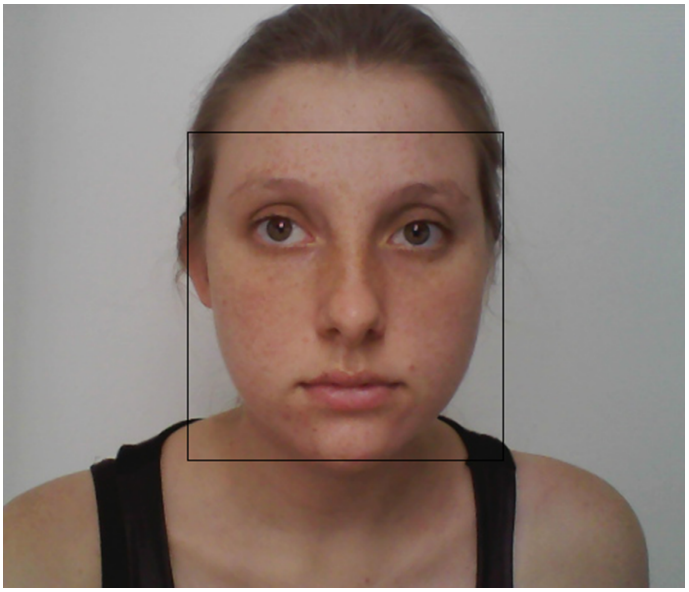


Figura 6-34: Rectángulo de *tracking*

De todas maneras, hay una constante de iteraciones definida después de la cual se vuelve a buscar caras en toda la imagen. Esto es una protección en caso de que el algoritmo de convergencia de *landmarks* no converja correctamente sobre la cara.

El *tracking* sólo funcionará si el usuario está mirando hacia el frente. De lo contrario, el algoritmo de *tracking* se interrumpe y busca caras en todos los cuadros hasta que el usuario vuelva a estar dentro de los límites establecidos para la orientación de la cara. El rectángulo definido para el *tracking* se diseñó con la idea de agilizar el sistema al máximo, siendo lo más pequeño posible (teniendo en cuenta el margen necesario para que la convergencia de los *landmarks* sea exitosa).

Para evitar que localice otra cara (por ejemplo de otra persona pasando por detrás) el algoritmo siempre verifica que el centro de la cara no se haya corrido más de un número constante de unidades de su posición anterior. Esta unidad corresponde a la distancia entre los ojos.

7. Implementación en C++

Este capítulo presenta la implementación en C++ de la librería de algoritmos desarrollados en Scilab. Junto con los algoritmos, se creó una estructura de datos que permite dar una representación apropiada a los objetos del sistema. Estos tres objetos son: Cara, Parte y Gesto.

Previamente a comenzar la explicación detallada de la estructura de datos, los archivos y sus funciones, se recuerdan los requerimientos básicos que impone la librería al sistema sobre el que será instalada.

7.1. Requerimientos de la librería

La librería de funciones implementada, requiere que se tengan previamente instalados OpenCV y Dlib sobre el equipo Linux sobre el que se quiera utilizar. Se requiere a su vez una cámara frontal desde la cual se tomarán los datos. Las inclusiones de los archivos que requiere la librería deben estar bien referenciados a las direcciones en las que se encuentran los archivos de OpenCV y Dlib utilizados (ver Anexo G para la guía de instalación).

7.2. Estructura de datos

La estructura de datos utilizada en la librería de funciones viene dada por la figura 7-35. Previo al diagrama se describirán algunas de las variables que se van a mencionar, que son propias de Dlib.

- **full_object_detection:** Representa la *shape*. Es un vector con la posición de los 68 *landmarks* detectados.
- **dlib::point:** Representa un punto en el espacio bidimensional con dos coordenadas.
- **dlib::matrix<double>:** Representa una matriz de *doubles*.

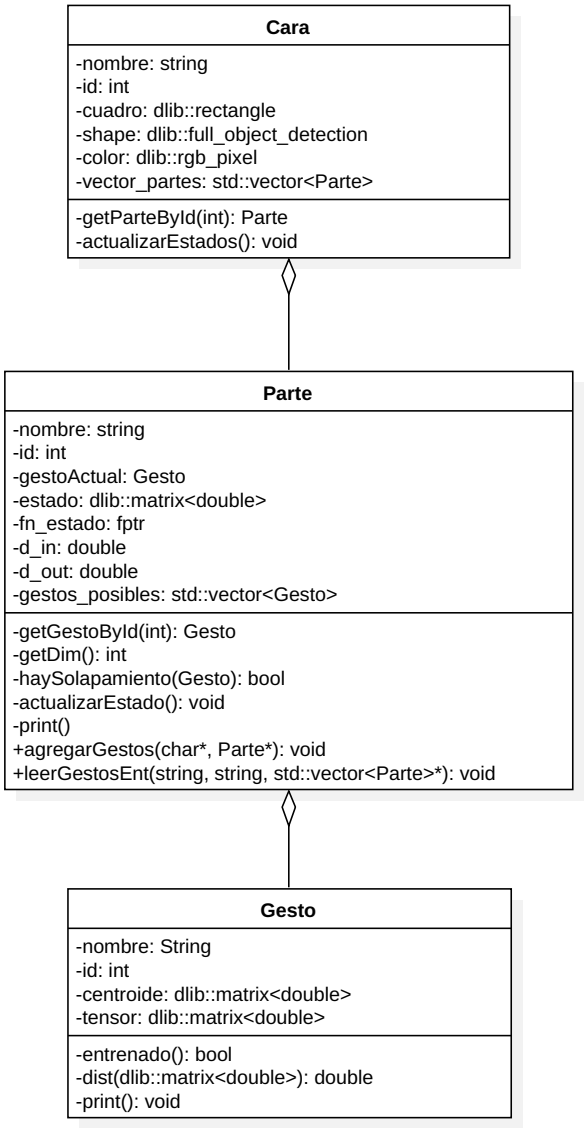


Figura 7-35: Diagrama de clases utilizado en el desarrollo de la librería de funciones.

Las instancias de la clase Cara, representarán el resultado del algoritmo de detección de caras en cada cuadro. La clase Parte, representa las distintas partes de la cara, sobre las que se querrán detectar Gestos.

A continuación se detallan los atributos y funciones de cada una de las clases.

7.2.1. Clase Cara

Como lo sugiere el nombre de la clase, las instancias representarán cada una de las caras presentes en el cuadro de la cámara. Sus atributos son:

- **nombre** e **id:** Atributos que intentan dar una identificación a la cara: nombre del usuario y número de usuario por ejemplo. El id podría utilizarse como identificador en una tabla de Base de Datos, o en una transferencia de datos simple.
- **cuadro:** Representa el rectángulo que contiene a la cara dentro del cuadro. Es directamente el resultado arrojado por el algoritmo de detección de caras.
- **shape:** Es el conjunto de puntos del modelado ASM de la cara.
- **color:** Variable que representa el color con el que se podrían querer dibujar los *landmarks* o cualquier *layout*, sobre el cuadro en una interfaz gráfica para el usuario.
- **vector_partes:** Partes que componen a la cara sobre las que después se detectarán posibles gestos.

Las funciones aplicables sobre una instancia de la clase Cara son:

- **getParteById(int)**: Devuelve la Parte dentro del vector de partes dado su id.
- **actualizarEstados()**: Actualiza en cada paso las variables de las partes de la cara.

7.2.2. Clase Parte

La clase representa los conjuntos de partes de la cara (ojos, boca, etc.) sobre las que se querrán detectar gestos. Los atributos se explican a continuación:

- **nombre** e **id:** Atributos que intentan dar una identificación a la parte. El id podría utilizarse como identificador en una tabla de Base de Datos, o en una transferencia de datos simple.
- **gestoActual:** Representa el gesto que se encuentra activo en la instancia de Parte.

- **estado:** Matriz de n filas y 1 columna (vector de $\mathbb{R}^n$), representando el vector descriptor Θ, con n valores reales representando el estado de la instancia de Parte.
- **fn_estado:** Puntero a la función que recibe por parámetro un *shape* de la clase Cara, y devuelve el vector Θ, de medidas que determinan el estado de la Parte.
- **d_in** y **d_out:** Valores reales positivos indicando los radio de entrada y salida a cada gesto respectivamente (recordar de la ecuación 5-15, que el valor del radio de entrada depende únicamente de la dimensionalidad de Θ, y por lo tanto no es necesario definirlo para cada gesto particularmente).
- **gestos_posibles:** Conjunto de gestos detectables para esta parte.

Las funciones aplicables para cada instancia de Parte se explican a continuación:

- **getGestoById(int):** Devuelve el Gesto dentro de la lista de gestos posibles dado su id.
- **getDim():** Devuelve la dimensionalidad del vector Θ para esta parte (la cantidad de filas de la matriz estado).
- **haySolapamiento(Gesto):** Determina si el gesto que se pasa por parámetros se solapa con alguno de los gestos ya almacenados en gestos_posibles.
- **actualizarEstado():** Una vez actualizada la matriz estado se puede utilizar esta función para determinar qué gesto está activo.
- **print():** Un *print* por consola detallando el estado y los gestos activos de esta parte.

Las funciones públicas contenidas en la clase Parte son:

- **agregarGestos(char*, Parte*):** Agrega a la parte pasada por parámetro, los gestos determinados por el archivo de configuración pasado por parámetro.
- **leerGestosEnt(string, string, std::vector<Parte>*):** Dados los archivos de entrenamiento para los centroides y los tensores, devuelve un vector con las partes y todos sus gestos entrenados.

7.2.3. Clase Gesto

La clase gesto representa los estados detectables para cada una de las partes. A continuación se presenta la explicación de cada uno de los atributos:

- **nombre** e **id:** Atributos que intentan dar una identificación al gesto. El id podría utilizarse como identificador en una tabla de Base de Datos, o en una transferencia de datos simple.
- **centroide:** Matriz de n filas y 1 columna (vector de $\mathbb{R}^n$), representando el centroide μ del gesto.
- **tensor:** Matriz de n filas y n columnas, representando el tensor de medida Σ^{-1} utilizado para medir la distancia de Mahalanobis al centroide de este gesto.

Las funciones aplicables sobre cada gesto son:

- **entrenado():** Función que retorna un booleano indicando si el gesto ya fue entrenado o no.
- **dist(dlib::matrix<double>):** Recibe como parámetro el estado de la parte que contiene a este gesto Θ, y calcula la distancia de Mahalanobis hasta el centroide μ del gesto. En otras palabras devuelve $d_\Sigma(\Theta, \mu)$.
- **print():** Un *print* por consola detallando el resultado del entrenamiento del gesto.

7.3. Librerías de funciones

A continuación se describirán las tres librerías de funciones del sistema de detección de gestos. Para una lista completa de las funciones implementadas ver Anexo F.

7.3.1. Funciones de Tratamiento

Tratamiento contiene las funciones referentes al cálculo de distancias y áreas entre *landmarks* y operaciones con vectores. Las principales funciones son las siguientes.

- **returnPoints(full_object_detection shape, int primero, int ultimo):** Devuelve un vector con la posición de los *landmarks* indexados entre primero y último.
- **longitud (std::vector<dlib::point> puntos):** Devuelve la longitud del contorno definido por el vector puntos.
- **areaSector (std::vector<dlib::point> borde):** Devuelve el área del sector contenido en el vector borde
- **rectTracking(full_object_detection shape):** Devuelve el rectángulo donde va a intentar converger los *landmarks* en el siguiente cuadro.

A continuación se muestran los procedimientos que se utilizan para determinar la longitud de una secuencia de puntos, y el área contenida entre ellos.

7.3.1.1. Cálculo de longitudes

El cálculo de longitudes se usa para calcular contornos de partes de la cara. Por lo tanto, se parte de un vector de *landmarks* ordenados naturalmente siguiendo una línea de la cara.

Para calcular la longitud del vector se calculan las distancias entre los *landmarks* dos a dos y se acumulan para obtener la longitud del vector.

7.3.1.2. Cálculo de áreas

El cálculo de áreas se basa en determinar el área a calcular y dividirla en triángulos con vértices en los *landmarks* del contorno de la misma. Estos triángulos son determinados por vectores entre los *landmarks* como se muestra a continuación.

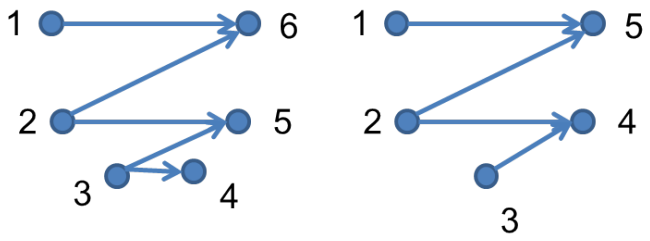


Figura 7-36: Triangulación de un sector

Para crear los vectores se parte del contorno del área a calcular y se segmenta en vectores individuales. Una vez designados los vectores, se calcula el área entre ellos mediante el producto vectorial. El último paso es sumar todas las áreas entre los vectores.

7.3.2. Funciones de Estados

Estados contiene dos tipos de funciones. Por un lado, tiene más de 30 funciones que calculan distintos descriptores θ_i y distintas medidas de la cara dado un *shape*, que podrían utilizarse para agregar gestos. Por otro lado, contiene las funciones de actualización de estado de cada parte definida (boca, cejas y orientación). Las funciones de actualización de estado son llamadas en cada cuadro para determinar el vector descriptor Θ.

Por ejemplo, como se vio en la sección 5.12.3, las cejas tienen dos descriptores θ_1 y θ_2 definidos por las ecuaciones 5-26 y 5-27 respectivamente. Para esto, en Estados están definidas las funciones de cálculo de ambos descriptores, y la función que devuelve el vector de estado Θ. Se tienen entonces las funciones:

- **retornarEstOjoDer_b(full_object_detection shape)**: Devuelve el descriptor θ_1 de las cejas.
- **retornarEstOjoIzq_b(full_object_detection shape)**: Devuelve el descriptor θ_2 de las cejas.
- **estado_cejas(full_object_detection shape)**: Devuelve el vector $\Theta = [\theta_1, \theta_2]^T$ de las cejas.

Si el programador necesita definir nuevos descriptores para determinada parte, o una función de actualización de estado para una parte nueva, éste es el lugar donde debe escribirla.

7.3.3. Funciones de Entrenamiento

Entrenamiento contiene las funciones de cálculo de los parámetros para la detección de gestos. Las funciones principales se describirán a continuación.

- **centroids_vector(std::vector<dlib::matrix<double> > training_set):** Devuelve las coordenadas del centroide de la distribución de puntos de entrenamiento, *training_set.*

- **covariance_matrix_vector(std::vector<dlib::matrix<double> > training_set):** Devuelve la matriz de covarianza de la distribución de puntos de entrenamiento, *training_set.*
- **calcular_rad(double dim, double lim):** Calcula la distancia de Mahalanobis δ_{in} resolviendo la ecuación 5-15, "dim" representa al parámetro la dimensionalidad del vector Θ y "lim" representa a τ.

8. Aplicaciones

8.1. Diagrama de flujo principal

El diagrama de flujo principal comienza utilizando las funciones de OpenCV para obtener acceso a la cámara y comenzar a recoger datos. Inmediatamente se cargan los clasificadores de Dlib para encontrar caras, y la configuración de los gestos entrenados. Una vez hecho esto, se procede a buscar caras.

En este flujo se procurará seguir únicamente una cara, por lo tanto una vez que se detectan las caras utilizando Dlib, y se encuentran los *landmarks* de la cara, se utiliza un pequeño filtro para descartar todas las otras posibles caras que se encuentren lejanas a la cual se le está realizando *tracking* (en el primer paso se toma como cara a seguir la primer cara encontrada por el algoritmo).

Se actualiza luego el rectángulo de *tracking* para saber la nueva posición de la cara, y se determinan los vectores de estado de cada gesto y cuales serán los gestos activos de cada parte.

Luego de esta etapa viene el **bloque de aplicación**, donde se determina qué hacer con el estado de las partes. Más adelante se explicarán los diagramas de flujo utilizados en este bloque para cada una de las aplicaciones.

Finalmente se determina si debe continuarse el *tracking*, dado que se buscan caras nuevas cada N pasos (por seguridad en caso de haber perdido la cara).

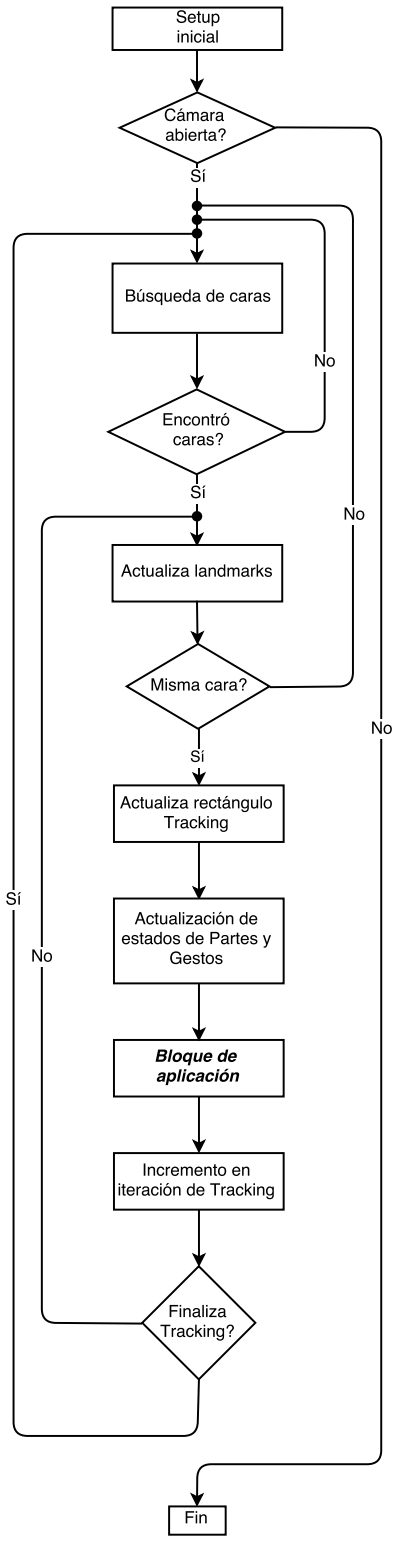


Figura 8-37: Diagrama de flujo principal del sistema. El bloque aplicación es variable según el fin con el que se desee utilizar la librería.

8.2. Envío de SMS con gestos faciales

Para el envío de mensajes con gestos faciales se utilizó un módulo GSM Huawei USB E3131, con el que se establece una comunicación serial a través del puerto USB de la terminal. Se utilizan comandos AT para establecer y monitorear la conexión con el módem, y para enviar mensajes.

Cuando la orientación facial del usuario es hacia el frente y se detecta alguno de los gestos que se determinan como detonantes, se envía el mensaje correspondiente. Esto implica levantar un proceso que establece la conexión con el módulo y envía un mensaje. El proceso de envío de SMS recibe como parámetro un número indicando el mensaje que se desea enviar dentro del directorio de mensajes pre diseñados. El mensaje se envía a un conjunto de números telefónicos predeterminados en un archivo de texto.

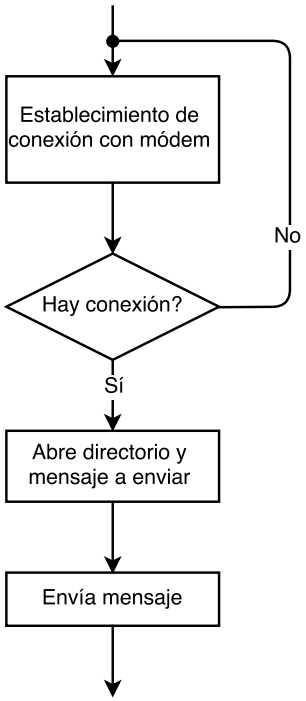


Figura 8-38: Diagrama de flujo del bloque de aplicación para la el caso de uso de la librería para el envío de SMS mediante un módem GSM.

El proceso de envío de mensajes utilizando el módulo GSM se inicia estableciendo conexión con el módem. Luego de superada la etapa de establecimiento de la conexión, se procede a enviar el mensaje indicado por el argumento con el que se convoca al proceso, al directorio de números indicado en el archivo de directorio.

Para evitar envíos de mensajes excesivos se utiliza un *timer* en el diagrama

principal que permite volver a atender la interacción con el módem luego de N segundos.

8.3. Control del *mouse*

Para el control del mouse como se mencionó anteriormente se utilizan las 5 orientaciones posibles: abajo, arriba, izquierda, derecha y hacia el frente. Cuando la orientación se encuentra inclinada en alguna de las posiciones verticales u horizontales el *mouse* se desplazará en esa dirección, mientras que si la orientación de la cara es hacia el frente y se esta realizando algún otro gesto previamente especificado se puede utilizar para simular alguno de las acciones del *mouse*. Para evitar el cliqueo repetitivo cuando se detecta el gesto correspondiente, se programó un *timer* que evita que el *mouse* sea cliqueado dos veces seguidas en menos de N segundos configurables.

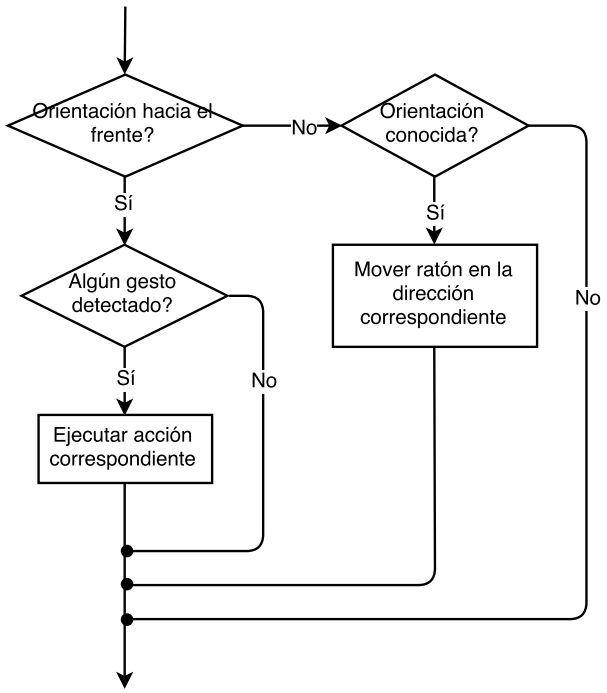


Figura 8-39: Diagrama de flujo del bloque de aplicación para la el caso de uso de la librería para el control del ratón.

Para lograr obtener control del mouse se utilizó la librería de funciones "Xlib" [48]. La librería es un conjunto de macros y funciones en lenguaje C que hace de interfaz a bajo nivel con X11 (librería gráfica de los sistemas Unix). La librería Xlib, tiene funciones que permiten controlar no solamente todas las funcionalidades del

mouse sino también las del teclado. Por lo tanto, sin mucho más trabajo se podría dotar al usuario de control sobre el teclado mediante el uso de gestos.

9. Análisis final

Se analizará el rendimiento del sistema en función a la efectividad y velocidad.

9.1. Análisis de Efectividad

9.1.1. Método de evaluación del calsificador

Para poder evaluar el uso del clasificador en la detección de gestos, se solicitó a las personas utilizadas en la base de datos (7 personas), utilizar el sistema entrenando el gesto y luego repetir el gesto entrenado. Se generaron así algunos archivos como lo muestra la Fig. 9-40.

$$\begin{array}{cc} t=0 & x_0; y_0; x_1; y_1; \cdots; x_n; y_n; lbl \\ t=1 & x_0; y_0; x_1; y_1; \cdots; x_n; y_n; lbl \\ \vdots & \vdots \\ t=T_f & x_0; y_0; x_1; y_1; \cdots; x_n; y_n; lbl \end{array}$$

Figura 9-40: Esquema representativo del archivo de salida para el programa de generación de datos. La columna de tiempos a la izquierda es solo representativa, los únicos datos salvados son los *landmarks*. T_f representa el último instante de muestreo y n es la cantidad de *landmarks*. La última variable *lbl* adopta el valor 1 cuando el gesto debería estar realizandose y 0 en el caso contrario.

A partir de los valores de los landmarks, se determinan los valores de los descriptores de las partes de la cara, y la variable *lbl* indica si el gesto debería estar realizádose o no. A partir de esto se determinan los parámetros que permiten analizar los resultados del clasificador como se explica en las siguientes secciones.

9.1.2. Evaluación del desempeño de un clasificador

Formalmente cada una de las muestras obtenidas por la cámara para una persona pueden etiquetarse con un único elemento del conjunto $\{p, n\}$ que indican muestra positiva o negativa respectivamente para determinado gesto. Un clasificador, mapea cada una de estas muestras al conjunto $\{S, N\}$ que son las predicciones dadas por el clasificador para las etiquetas p y n respectivamente [40].

Se definen así los subconjuntos TP, FN, FP y TF según lo ilustra la Fig.

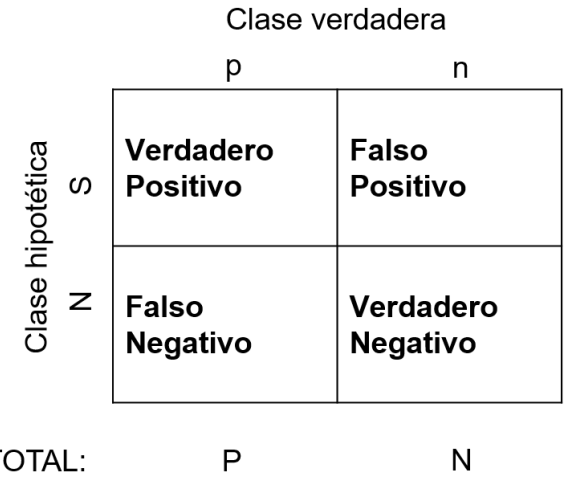


Figura 9-41: Matriz del conjunto $\{p, n\} \times \{S, N\}$

9-41, y juntos con ellos una serie de parámetros que determinan la *performance* de un sistema de clasificación. Estas cantidades son:

$$fp \text{ rate} = \frac{\#FP}{\#n} \tag{9-35}$$

$$\text{recall} = tp \text{ rate} = \frac{\#TP}{\#p} \tag{9-36}$$

$$\text{precision} = \frac{\#TP}{\#TP + \#FP} \tag{9-37}$$

$$\text{accuracy} = \frac{\#TP + \#TN}{\#p + \#n} \tag{9-38}$$

$$\text{F-measure} = \frac{2}{\text{precision}^{-1} + \text{recall}^{-1}} \tag{9-39}$$

9.1.3. Resultados

Se generaron datos etiquetados para correr en Scilab el entrenamiento y evaluar la *performance* de nuestro algoritmo. Los resultados obtenidos para diferentes gestos son los siguientes:

Parámetro a evaluar	fp rate	precision	recall	accuracy	F-measure
Resultado	0	1	0,91	0,98	0,95

Tabla 9-1: Resultados para la *performance* del gesto boca abierta

Parámetro a evaluar	fp rate	precision	recall	accuracy	F-measure
Resultado	0	1	0,99	0,99	0,99

Tabla 9-2: Resultados para la *performance* del gesto boca beso

Parámetro a evaluar	fp rate	precision	recall	accuracy	F-measure
Resultado	0	1	1	1	1

Tabla 9-3: Resultados para la *performance* del gesto boca sonriendo

Parámetro a evaluar	fp rate	precision	recall	accuracy	F-measure
Resultado	0	1	1	1	1

Tabla 9-4: Resultados para la *performance* del gesto levantar cejas

Parámetro a evaluar	fp rate	precision	recall	accuracy	F-measure
Resultado	0	1	0,82	0,91	0,90

Tabla 9-5: Resultados para la *performance* de los gestos orientación.

9.2. Análisis de tiempo

¿Qué se considera procesamiento en tiempo real?

Tiempo real es un concepto amplio y no existe una única definición de lo que se considera real para el procesamiento de imágenes. Un enfoque sostiene que el procesamiento es en tiempo real si es más rápido que el tiempo de la cámara entre cuadro y cuadro [41]. Esta definición no es la más adecuada para la aplicación porque depende de la cámara del usuario y el sistema está pensado para correr en diferentes computadoras. En su lugar, se puede considerar procesamiento de imágenes en tiempo real si la imagen de salida se puede ver continua. Para el ojo humano 13 fps se considera la tasa de imágenes por segundo mínima. Para tener una referencia el estándar para el cine son 24 fps [42].

Se cree que la segunda definición se adecua más al sistema por lo que se intentó tener un tiempo de procesamiento mayor a $77ms$ (equivalente a 13fps).

Para calcular el tiempo de procesamiento del sistema se calculó la diferencia de tiempo entre la primera iteración y la número 500 en una PC con procesador i3, memoria RAM de 4 GB a 1600 MHz con tecnología ddr3 sdram.

El tiempo de procesamiento se calculó dividiendo la diferencia de tiempo entre el número de iteraciones.

Los fps *(frames per second)* se calculan como la inversa del tiempo de procesamiento.

9.2.1. Tiempos del sistema sin *tracking*

El sistema sin *tracking* tiene un tiempo de procesamiento[7] de $160ms$, lo que equivale a $6,25$ fps.

El tiempo de procesamiento es la suma de la detección de caras, el detector de *landmarks* faciales (ambos de Dlib) y el algoritmo implementado de detección de gestos. Como se explicó anteriormente, el mayor cuello de botella es la búsqueda de caras en todo el cuadro por lo que se diseñó el sistema de *tracking* explicado en el capítulo 6.

9.2.2. Tiempos del sistema con *tracking*

Con el algoritmo de *tracking* se disminuye el tiempo de procesamiento de $160ms$ a $60ms$ ($167\,\%$ más veloz). Esto equivale a $16,7$fps que es ligeramente superior al límite inferior de lo que se puede considerar como tiempo real. De todas maneras, se siguieron buscando estrategias para disminuir el tiempo de procesamiento.[8]

Además de entender que el tiempo de procesamiento se asemeja a un tiempo que nosotros consideramos "real", es importante destacar que efectivamente la experiencia de usuario muestra una respuesta "continua" por parte del sistema. Lo que lleva a concluír que la velocidad de procesamiento alcanzada es adecuada.

[7] Todas las pruebas referidas a tiempos fueron realizadas sobre un equipo Lenovo E330, memoria RAM de 4 GB a 1600 MHz con tecnología ddr3 sdram.

[8] Estos tiempos fueron calculados mostrando la imagen por pantalla y los resultados de la detección de gestos por consola.

9.2.3. Piramidación de imagen y mejora en tiempos

Una forma sugerida de mejorar el tiempo de procesamiento de Dlib es disminuir el tamaño del cuadro para que sea más rápido encontrar caras [43]. La única desventaja es que se pierden las caras más pequeñas. Esta estrategia fue utilizada inicialmente. Una vez implementado el algoritmo de *tracking*, que se basa en evitar buscar caras en todos los cuadros, la mejora no es sustancial (menor al 1 %).

La forma encontrada de agilizar la detección de gestos implica reducir la cantidad de información que ve el usuario en la pantalla. Un usuario con experiencia en el sistema que no necesite ver su imagen en la cámara o leer el estado de cada parte todo el tiempo puede utilizar el sistema con un tiempo de procesamiento de 40ms (25fps).

9.3. Análisis de recursos para el funcionamiento del sistema

Una vez implementado el sistema, se procedió a utilizarlo en varios terminales y se llegó a la conclusion de que la memoria RAM utilizada por el sistema es de aproximadamente $130Mb$. Los tiempos de obtención y análisis de cuadros por segundo dependerán de la capacidad de procesamiento del sistema.

Se presentan a continuación los datos obtenidos en un sistema con un procesador $1,4GHz$ Intel Core i5 (de dos núcleos), mientras se realizaban pruebas sobre el sistema utilizando la herramienta "*cpulimit*" del Linux, que permite limitar el uso del CPU para analizar los tiempos del sistema.

Límite al uso CPU (%)	**Uso del CPU (%)**		**Tiempo medio de cuadro (ms)**	
	Sin tracking	**Con tracking**	**Sin tracking**	**Con tracking**
Sin límite	108	85	175	50
60	60	60	280	60
40	40	40	415	80
20	20	20	830	180
10	10	10	450	360

Tabla 9-6: Resultados para el tiempo de obtención y análisis de un cuadro en función de la capacidad de procesamiento disponible para el proceso. Las medidas porcentuales están dadas con respecto al uso de uno de los dos núcelos del procesador, por lo que el uso máximo se corresponde con un 200 %.

A partir de las pruebas realizadas sobre la *performance* del sistema se sugiere utilizar procesadores que no sean inferiores a un 55 % de la capacidad dada por el procesador utilizado para las pruebas.

Como observación a la tabla anterior se puede ver que el porcentaje de uso del CPU y los tiempos del sistema son inversamente proporcionales, por lo tanto a partir de las especificaciones del procesador se podrían estimar los requerimientos mínimos necesarios sobre otro procesador para alcanzar el tiempo que se desee.

La ecuación obtenida mediante mínimos cuadrados de la relación anterior es:

$$C_{\%} \approx \frac{3471}{T_{CT}} \tag{9-40}$$

donde $C_{\%}$ representa la capacidad de procesamiento porcentual referido al procesador con que se realizaron las pruebas, y T_{CT} representa el tiempo con tracking que se desea que tenga el sistema.

9.4. Flexibilidad y readaptación de la librería

La librería se programó con la idea de que sea fácilmente extensible. Con un conocimiento básico de los *landmarks* se pueden agregar gestos a detectar tanto para las partes previamente definidas cómo para otras a definir en el futuro. Por otro lado, se pueden agregar aplicaciones.

A continuación se explicará como realizar estas acciones en orden creciente de complejidad.

9.4.1. Agregar Aplicaciones

Para agregar aplicaciones sin modificar los gestos a detectar solamente hay que sustituir el bloque aplicación por uno nuevo.

9.4.2. Agregar o quitar gestos

La librería permite la definición de nuevos gestos y partes según se detalla en las siguientes secciones.

9.4.2.1. Gestos programados

Para agregar o quitar gestos programados (descritos en 5.12.2,5.12.3 y 5.12.4) es tan sencillo como entrar al archivo de configuración correspondiente a la parte ("Boca.txt", "Cejas.txt" u "Orientacion.txt") y eliminar o agregar una línea correspondiente al gesto con el siguiente formato:

<nombre_del_gesto>;<id_del_gesto>;1

Por ejemplo el archivo "Boca.txt" contiene las líneas:

Normal;1;1; Beso;2;1; Sonrisa;3;1; Abierta;4;1;

9.4.2.2. Gestos nuevos

Antes de agregar gestos nuevos se deben tomar en cuenta tres consideraciones

1. ¿El gesto pertenece a una parte ya definida (boca o cejas)?
2. ¿Puedo detectar el gesto con los descriptores correspondientes a esa parte?
3. ¿Tengo una idea fundada de que no va a interferir con otros gestos ya definidos para esa parte?

Si la respuesta a las tres preguntas es sí, entonces hay que seguir el mismo procedimiento descrito en 9.4.2.1.

Si alguna de las respuestas es no, el programador va a tener que acceder al archivo "estados.cpp" (y "estados.h" respectivamente).

Si se quiere agregar un gesto que no puede ser detectado con los descriptores actuales de la parte deben agregarse uno o más descriptores nuevos. Para agregar un descriptor, se debe modificar la función de actualización de estado de esa parte. Las funciones tienen el nombre "estado_<nombre_de_la_parte>" (ver 7.3.2). Muchas funciones para crear descriptores están definidas en "estados.cpp" aunque no estén en uso.

Si se quiere agregar un gesto para una parte no definida previamente (por ejemplo: nariz), se debe crear la función "estado_<nombre_de_la_parte>" y agregar

un archivo de configuración correspondiente (“Nariz.txt”). También se deberá modificar el archivo de configuración global (“Configuracion.h”) para agregar el nombre, ID y archivo asociado de la parte. Por último en el archivo de entrenamiento y en el de detección se debe crear la parte y agregarla a un vector de partes, siguiendo la forma en la que están creadas las partes ya existentes.

Para ver más detalladamente cómo hacer estos cambios se sugiere ver Anexo E.

10. Conclusiones

El objetivo era implementar un sistema de reconocimiento de gestos faciales con una cámara web como alternativa al uso de las manos en la interacción con dispositivos tecnológicos para aplicaciones convenientes para personas con discapacidad motora y/o del habla y fue cumplido.

El sistema puede detectar cuatro gestos (con opción de extender a más) cuando la persona está mirando la cámara de frente y detectar la orientación de la cara cuando no está mirando de frente (arriba, abajo, izquierda o derecha). El procesamiento es de 40ms por cuadro, por lo que es posible asegurar que es en tiempo real.

Con respecto a la performance del sistema, se logró un rendimiento mayor al 80 % (ver 9.1.3).

Adicionalmente se diseñaron dos aplicaciones para interactuar con dispositivos tecnológicos: control del *mouse* y envío de SMS.

En reiteradas ocasiones a lo largo del proyecto se observó que la cantidad de publicaciones y librerías referidas al procesamiento de imágenes, está en aumento. A modo de ejemplo en Febrero del 2017 se publicó un artículo sobre la implementación de un controlador de *mouse* para un explorador web [44]. Un ejemplo de librería actualizada es el de OpenFace (ex-CLM Framework), que desde mediados del 2016 es capaz de detectar y modelar caras, estimar la pose de la misma, estimar la dirección de la mirada y analizar las acciones de diferentes unidades dentro de la cara [45]. Lo cual muestra que el análisis de imágenes aplicado al reconocimiento facial es un campo en auge.

El sistema implementado es superior a sistemas previos (analizados en la sección 3.1) tanto en rendimiento como en tiempo de ejecución. Además, la fase de entrenamiento (no implementada en los proyectos mencionados) agrega a el sistema una robustez y flexibilidad frente a problemas motrices que no poseen los otros sistemas.

10.1. Posibles mejoras

En esta sección se propondrán posibles mejoras al sistema.

- En lo que refiere al algoritmo podría buscarse una mejor aproximación al cálculo de los radios δ que la obtenida en la ecuación 5-15. También podría

estudiarse en mayor profundidad el problema planteado por la ecuación 5-16 o la ecuación 5-17 para la intersección de las elipses en n-dimensiones.

- Implementar un *tracking* capaz de seguir la cara cuando se encuentra en orientaciones diferentes a la frontal.
- Estudiar la adaptación de la librería para sistemas operativos de celular. Tiene la ventaja de que es más barato por lo que sería accesible a más personas. Por otro lado, es más fácil y liviano de transportar que una computadora. Existe una librería para ejecutar Dlib en Android [46].
- Estudiar OpenFace y nuevas tecnologías e investigar qué módulos de Dlib podrían ser reemplazados (estudiar las licencias puesto que no es una librería comercializable).
- Programar una interfaz más amigable al usuario. Actualmente el sistema interactúa con el usuario mediante las aplicaciones o la consola. Sería ventajoso tener un avatar en la fase de entrenamiento que muestre qué gesto hay que entrenar, o qué gesto está activo en la detección. Otra fase que se podría mejorar con una interfaz es la de configuración. En vez de ir a modificar archivos del programa, se podrían agregar o quitar gestos a través de una ventana.
- Entrenar gestos a ignorar. Por ejemplo si el usuario tiene un tic y alguno de los gestos se superponen en el entrenamiento, se podría sugerir al usuario una forma más apropiada de realizar el gesto, en función de cómo se dio la superposición.
- Implementar una estimación de la posición de la cara utilizando no un espacio discreto como el que se maneja actualmente (arriba, abajo, izquierda, derecha, frente), sino que un estimador de la posición de la cara que se almacene en un vector (perteneciente a $\mathbb{R}^2$ o $\mathbb{R}^3$) para lograr mover el mouse en cualquier dirección, por ejemplo.
- Ligado a la posible mejora anterior y extendiendo el concepto de distancia de Mahalanobis, podría implementarse un detector basado en la semi-distancia de Mahalanobis para funcionales [47]. Así, se detectarían determinadas trayectorias en el espacio de Θ (permitiendo alguna lectura de labios simple por ejemplo) o posibles movimientos en la orientación de la cara (mover la cara en círculos en un sentido o en el otro, etc.).

11. Referencias bibliográficas

Referencias

[1] P. Jia, H. Hu, T. Lu y L. Yuan. "Head gesture recognition for hands-free control of an intelligent wheelchair". *Industrial Robot: An International Journal ,* vol 34, pp 60-68, 2007.

[2] J. Tu, H. Tao y T. Huang (2007). "Face as mouse through visual face tracking". *Computer Vision and Image Understanding* , vol 108, pp 35-40, 2007.

[3] G. V. Paul, G. J. Beach, C. J. Cohen y C. J. Jacobus "Tracking and gesture recognition system particularly suited to vehicular control applications," U.S. Patent 7,050,606. Mayo 23, 2016

[4] . D. Kim and J. Sung. *Automated Face Analysis: Emerging Technologies and Research*. New York: Hershey, 2009, pp 1-4, 45-50.

[5] "face_detection_ex." [Online]. Available: http://dlib.net/face_detection_ex.cpp.html, [6 Junio, 2017].

[6] "Color conversions." [Online]. Available: http://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html, 18 Diciembre 2015 [27 Febrero, 2017].

[7] "YUV to RGB Conversion." [Online]. Available: http://www.fourcc.org/fccyvrgb.php, [27 Febrero, 2017].

[8] N. Ohta y A. Robertson. *Colorimetry: fundamentals and applications*. Chinchester: John Wiley & Sons Ltd., 2006, pp 63-71, 176-180.

[9] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. California: O'Reilly Media, Inc, 2008, pp 209-214.

[10] A. Ng. Support Vector Machines: Large magin intuition, Topic: "*Machine learning*", location: Computer Science Department, Stanford University, 21 Julio 2016.

[11] P. Viola y M Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". *Computer vision and pattern recognition*, 2001.

[12] "Statistical classification." [Online]. Available: https://en.wikipedia.org/wiki/Statistical_classification, 18 Noviembre 2016 [26 Febrero, 2017] .

[13] M. Kass, A. Witkin and D. Terzopoulos. "Snakes: Active contour models". *International journal of computer vision* vol 1, pp.321-331, 1988.

[14] A. Yuille, P. Hallinan, and D. Cohen. "Feature extraction from faces using deformable templates". *International journal of computer vision* vol 8, pp. 99-111, 1992.

[15] "Principal component analysis." [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis, 15 Febrero 2017 [26 Febrero, 2017].

[16] "Procrustes analysis." [Online]. Available: https://en.wikipedia.org/wiki/Procrustes_analysis, 5 Febrero 2017 [26 Febrero, 2017] .

[17] OpenCV Developers Team. "ABOUT." [Online]. Available: http://opencv.org/about.html, 2017 [26 Febrero, 2017] .

[18] P. Viola and M. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features" in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001.

[19] "Feature (computer vision)." [Online]. Available: https://en.wikipedia.org/wiki/Feature_(computer_vision), 27 Julio 2016 [26 Febrero, 2017] .

[20] "Face Detection Using HAAR Cascades." [Online]. Available: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html, 24 Febrero 2017 [26 Febrero, 2017] .

[21] S. Milborrow. "Stasm 4 User Manual." [Online]. Available: http://www.milbo.org/stasm-files/stasm4.pdf, 27 Diciembre 2013 [10 Marzo, 2016] .

[22] "Template matching." [Online]. Available:https://en.wikipedia.org/wiki/Template_matching, 4 Enero 2017 [26 Febrero, 2017].

[23] S. Milborrow and F. Nicolls. "Active Shape Models with SIFT Descriptors and MARS" in International Conference on Computer Vision Theory and Applications (VISAPP), 2014.

[24] "k-neartes neighbor algorithm." [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, 10 Febrero 2017, [26 Febrero, 2017] .

[25] "Hough transform." [Online]. Available: https://en.wikipedia.org/wiki/Hough_transform, 24 Febrero 2017, [26 Febrero, 2017] .

[26] D. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision*, vol 60, November 2004.

[27] J. Friedman. "Multivariate adaptive regression splines." *The annals of statistics*, pp.1-67, 1991.

[28] L. Gu, Y. Zhou and H.Zhang. “Bayesian Tangent Shape Model: Estimating Shape and Pose Parameters via Bayesian Inference” in CVPR’03 Proceedings of the 2003 IEEE computer society conference on Computer vision and pattern recognition, 2003 pp.109-116.

[29] “Maximum a posteriori estimation.” [Online]. Available: https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation, 16 Diciembre 2017, [26 Febrero, 2017] .

[30] T. Baltrušaitis, P. Robinson and L. Morency. “3D Constrained Local Model for Rigid and Non-Rigid Facial Tracking” in Computer Vision and Pattern Recognition (CVPR), IEEE Conference on, 2012.

[31] “Introduction.” [Online]. Available: http://dlib.net/intro.html , 28 Octubre 2015 [25 Febrero, 2017].

[32] “Visage Technologies.” [Online]. Available: http://visagetechnologies.com/, 2017 [25 Febrero, 2017]. N. Dalal and B. Triggs. “Histograms of Oriented Gradients for Human Detection” Computer Vision and Pattern Recognition, CVPR. IEEE Computer Society Conference on, 2005

[33] “Visage Technologies.” [Online]. Available: http://visagetechnologies.com/, 2017 [25 Febrero, 2017].

[34] V. Kazemi and J. Sullivan. “One Millisecond Face Alignment with an Ensemble of Regression Trees”. Computer Vision and Pattern Recognition (CVPR), IEEE Conference on, 2014

[35] ”Gradient_boosting.” [Online]. Available: https://en.wikipedia.org/wiki/ Gradient _boosting, 16 Enero 2017 [25 Febrero, 2017].

[36] P. C. Mahalanobis. “On the generalised distance in statistics” in *Proceedings National Institute of Science*, Vol. 2, pp. 49-55, Abril 1936.

[37] ”Mahalanobis distance” [Online]. Available: https://en.wikipedia.org/wiki/ Mahalanobis _distance, 19 Febrero 2017, [26 Febrero 2017].

[38] P. Fornasini. *The Uncertainty in Physical Measurements*, USA:Springer, 2008, pp 155-163.

[39] “Multivariate_normal_distribution” [Online]. Available: https://en.wikipedia.org/wiki/ Multivariate_normal_distribution, 27 Enero 2017 [26 Febrero 2017].

[40] T. Fawcett. “An introduction to ROC analysis.” Pattern recognition letters, Vol. 27, pp. 861-874, 2006.

[41] A. Bovik. *Handbook of Image &VideoProcessing*. New York: AcademicPress, 2005, pp 3-17.

[42] “Frame rate”. [Online]. Available: https://en.wikipedia.org/wiki/Frame_rate#How_many_frames_per_second_can_the_human_eye_see.3 F, 23 Febrero 2017 [25 Febrero, 2017].

[43] S. Mallik. “Speeding up Dlib’s Facial Landmark Detector.” [Online]. Available: http://www.learnopencv.com/speeding-up-dlib-facial-landmark-detector/, 2017 [25 Febrero, 2017].

[44] H. Mosquera, et al. "Identifying facial gestures to emulate a mouse: Control application in a web browser." Signal Processing, Images and Artificial Vision (STSIVA), 2016 XXI Symposium on. IEEE, 2016.

[45] T. Baltrušaitis, P. Robinson and L. Morency. “OpenFace: an open source facial behavior analysis toolkit.” [Online]. Available: https://www.cl.cam.ac.uk/~tb346/res/openface.html , 2016 [25 Febrero, 2017]

[46] “Port dlib to Android.” [Online]. Available: https://github.com/tzutalin/dlib-android , Enero 2012 [6 Junio, 2016]

[47] J. Esdras, P. Galeano, and R. E. Lillo. "The Mahalanobis distance for functional data with applications to classification." *Technometrics* Vol 57 pp. 281-291, 2015.

[48] C. Tronche. “The Xlib Manual.” [Online]. Available: https://tronche.com/gui/x/xlib/ , Octubre 2005 [20 Enero, 2017]

[49] T. Fawcett. “An introduction to ROC analysis.” Pattern recognition letters, Vol. 27, pp. 861-874, 2006.

Apéndices

A. Planificación

A continuación: el diagrama de Gantt del proyecto.

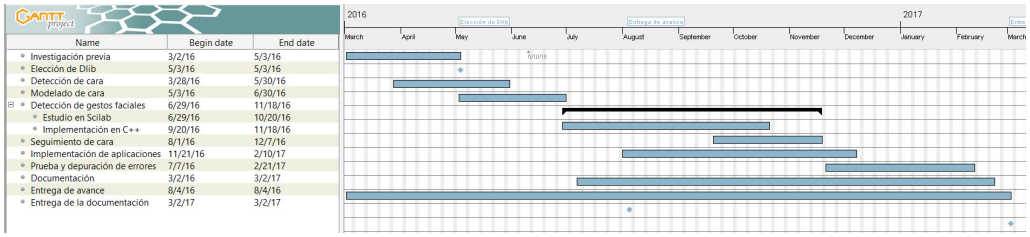


Figura A-42: Gantt (completo)

Figura A-43: Gantt (solo fechas)

A.1. Marzo - Mayo

- Investigación de algoritmos existentes. Estudio del lenguaje de programación C++. Estudio de OpenCV y de sus aplicaciones.
- Se implementó el primer algoritmo de búsqueda de caras a tiempo real en openCV con filtros HAAR.
- Búsqueda de librerías *open source* que resolvieran parte del problema. (stasm, asm, Dlib y CLM framework)

Se comenzó la investigación de algoritmos existentes para reconocimiento de imágenes, detección de cara, modelado de la cara y *tracking* de objetos. La investigación

llevó a leer sobre openCV, clasificadores, filtros HAAR, *adaboost*, *Active Shape Models (ASM)* y *Active Appereance Models* (AAM). Como se encontraron ejemplos de reconocimiento de cara implementados con openCV y filtros HAAR se empezaron a implementar pequeños sistemas basados en los ejemplos. A partir del código encontrado se desarrollaron programas para acceder a la cámara, detectar caras y detectar una cara y seguirla. A su vez se dedicó gran parte de abril a familiarizarnos con el lenguaje C++.

A.2. Mayo- Julio

- Se decidió usar Dlib.
- Estudio de qué gestos eran posibles detectar con la librería. Codificación de algoritmos necesarios para adaptar Dlib a nuestras necesidades, entre ellos: recoger los *landmarks* pertenecientes a una parte de la cara, calcular perímetros y/o áreas, aislar una parte del cuadro para su análisis.
- Análisis en Scilab. Estudio de tiempos de procesamiento.

Una vez elegida Dlib para la detección de gestos se empezó a estudiar qué gestos se podían detectar. Por ejemplo, el algoritmo se ajusta bien a cambios en la boca (abierta, cerrada, sonriendo) y a movimientos en las cejas, pero no reconoce una guiñada. Se hicieron algoritmos que permiten sacar información sobre distancias y áreas y otros algoritmos auxiliares que facilitan las tareas nombradas (recoger los *landmarks* de una parte de la cara o asilar una parte del cuadro.) Los datos recogidos gracias a los algoritmos fueron exportados a Scilab para analizarlos con facilidad. Se comprobó que se podía verificar el estado de la boca y el movimiento de cejas. Se estudió el tiempo de procesamiento del sistema.

A.3. Julio-Agosto

- Implementar un objeto cara para guardar los *landmarks* y facilitar el *tracking*.
- *Tracking*.
- Introducción a redes neuronales y en especial los algoritmos que provee Dlib para trabajar con ellas.

El objetivo de estos dos meses fue reducir el tiempo de procesamiento. Para eso se decidió implementar un *tracking* de la cara y evitar que el algoritmo tenga que volver a encontrar la cara en toda la imagen cada vez que corre para buscar los *landmarks*. Aunque Dlib cuenta con un *tracker*, su implementación aumenta el tiempo

de procesamiento en gran medida por lo que se decidió implementar un objeto cara donde se archive la última posición de los *landmarks* y buscar la próxima posición dentro de un nuevo rectángulo que contenga a la cara.

Por otro lado se quiere llegar a un sistema de reconocimiento de gestos que no sea tan simple como depender de un umbral (por ejemplo si el área de la boca es mayor a un x % del área de la cara está abierta), sino hacer una clasificación a partir de redes neuronales. Aunque Dlib ofrece algoritmos de redes neuronales simples todavía se está estudiando la mejor forma de implementar la clasificación.

A.4. Agosto – Diciembre

- Terminar y probar la detección de gestos simples a una persona.
- *Tracking.*
- Implementación de aplicaciones.

Se mejoró el sistema de detección de gestos implementando la estrategia descrita. A partir de la nueva detección de gestos se modificó la estrategia de *tracking* para que solo funcione cuando está mirando hacia el frente, reduciendo así la cantidad de errores por no poder converger los *landmarks.* También se comenzó con la implementación de las aplicaciones: control del *mouse* y envío de SMS mediante un módulo GSM.

A.5. Diciembre – Febrero

- Fin de la implementación de aplicaciones.
- Perfeccionamiento del sistema.
- Versión final de la documentación.

Se terminó la implementación de las aplicaciones. Se realizaron pruebas para el perfeccionamiento del sistema. Se definieron qué gestos se podían detectar.

Finalmente, se realizó la versión final de la documentación.

A.6. Diagrama de Gantt original

Nombre	Fecha de in...	Fecha de fin
Investigación previa	10/03/16	17/04/16
Estudio de cámara web	04/04/16	24/04/16
Detección de cara	04/04/16	02/05/16
Seguimiento de cara	03/05/16	04/07/16
Modelado de la cara	18/04/16	05/06/16
Detección de gestos faciales	06/06/16	05/08/16
Prueba y depuración de err...	18/07/16	12/09/16
Implementación de acciones	29/08/16	28/10/16
Prueba y depuración de err...	03/10/16	11/12/16
Perfeccionamiento del sist...	12/12/16	31/01/17
Documentación	10/03/16	02/03/17
Entrega de avance	04/08/16	04/08/16
Entrega de la Documentaci...	02/03/17	02/03/17

Figura A-44: Gantt inicial

La investigación previa se extendió hasta el final de Abril.

El estudio de la cámara web no se realizó formalmente sino que el enfoque fue en encontrar los comandos para acceder a la cámara y capturar los cuadros. Esto se completó aproximadamente a finales de Marzo.

Los primeros acercamientos a la detección de cara fueron realizados entre fines de Marzo y principios de Abril con OpenCV pero no corresponden a los algoritmos que se usaron en la versión final. En general todas las librerías de detección de *landmarks* traían su propio detector de cara y quedó definido cuando se decidió usar Dlib a fines de mayo.

Como al encontrar Dlib el enfoque cambió inmediatamente a la detección de *landmarks* (o modelado de la cara), el seguimiento quedó desplazado y no lo fue retomado hasta principios de Agosto cuando se estaba buscando maneras de mejorar el tiempo de procesamiento. La versión final de *tracking* se basa en el reconocimiento del gesto mirar hacia adelante por lo que fue terminado en Diciembre.

El modelado de la cara lo realiza Dlib y quedó pronto a fines de Mayo.

La detección de gestos faciales empezó a principios de Junio y terminó en Noviembre. Se le dio mucha importancia a que el sistema sea robusto y adaptable.

Las aplicaciones llevaron de Noviembre a Febrero.

B. Aproximación de radios en la detección de gestos

Se busca determinar la mínima distancia de Mahalanobis cuya elipse de nivel permita encerrar una fracción τ del volumen encerrado por una gaussiana n-dimensional. Para esto, se parte de la ecuación 5-14 y se realiza una sucesión de cambios de variables que transformarán este problema en el problema de la ecuación 5-15 que es más simple.

La idea básica será transformar una distribución genérica $N(x; \mu_i, \Sigma_i)$ de un gesto i, en una distribución estándar multivariable $N(z; O, Id)$, donde O representa el origen, e Id la matriz identidad, como muestra la siguiente imagen:

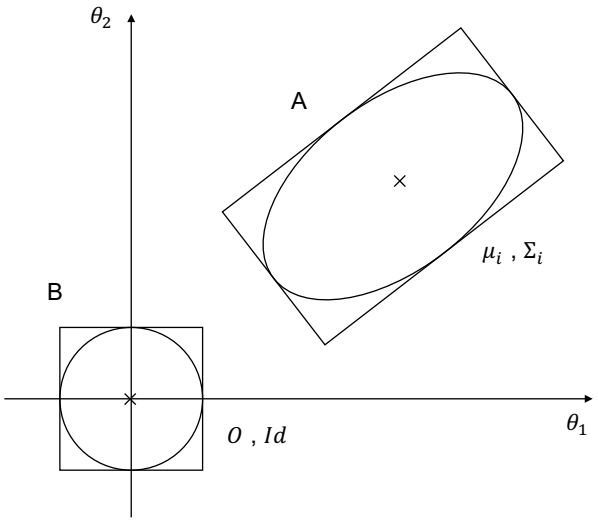


Figura B-45: Imagen representativa en $\mathbb{R}^2$ de los cambios de variables a realizar que llevan la figura A en la figura B, ambos con el conjunto con el que se aproxima la elipse.

El conjunto de transformaciones a hacer para llevar la elipse A de la imagen anterior en la elipse B, esta dado por lo siguientes pasos (denominada aquí $\alpha = [(2 \cdot \pi)^n \cdot |\Sigma|]^{-1/2}$):

1. Una **traslación** que lleva μ_i a O. Esto es un cambio de variable[9] $\nu_1 = x - \mu_i$. Este cambio de variable no produce cambios en los diferenciales de la integral,

[9]Denominado ν_l al cambio de variable auxiliar realizado en el paso l, y ν_{l_k} a su k-ésima componente.

dado que $d\nu_{1_k} = dx_k$.

$$\int_{d_{\Sigma_i}(x,\mu_i)<\delta} N(x;\mu_i,\Sigma_i)\,dx_1\cdots dx_n = \alpha\cdot\int_{d_{\Sigma_i}(\nu_1,O)<\delta} e^{-\frac{1}{2}\cdot\nu_1^T\cdot\Sigma_i^{-1}\cdot\nu_1} d\nu_{1_1}\cdots d\nu_{1_n} = \tag{2-41}$$

2. Una **rotación** del sistema que lleve los ejes principales de A, paralelos a los ejes. Esto es un cambio de variable que diagonialice la matriz Σ_i^{-1} que se denominará matriz M_i, esto es: $M_i = P\cdot\Sigma_i^{-1}\cdot P^t$, donde P es una matriz orto-normal. El cambio de variable necesario será entonces: $\nu_2 = P\cdot\nu_1$. Dado que P es una transformación ortogonal, no modifica las medidas de volumen y por lo tanto no produce cambios en el diferencial.

$$= \alpha\cdot\int_{d_{M_i}(\nu_2,O)<\delta} e^{-\frac{1}{2}\cdot\nu_2^T\cdot M_i^{-1}\cdot\nu_2} d\nu_{2_1}\cdots d\nu_{2_n} = \tag{2-42}$$

3. Una **homotecia** sobre el espacio que lleve a la elipse centrada y con ejes principales sobre los ejes de coordenadas a una esfera unitaria. Dado que M_i es diagonal, puede escribirse $\nu_2^t\cdot M_i\cdot\nu_2 = \sum_k \lambda_k\nu_{2_k}^2$, donde λ_k representa el k-ésimo valor propio de M_i, y puede verse que el cambio de variable a realizar será $\nu_{3_k} = \lambda_k^{-1/2}\cdot\nu_{2_k}$, de modo que el cambio en los diferenciales será $d\nu_{3_k} = \lambda_k^{-1/2}\cdot d\nu_{2_k}$. Como resultado $\sum_k \lambda_k\nu_{2_k}^2 = \sum_k \nu_{3_k}^2 = \nu_3^t\cdot Id\cdot\nu_3 = \nu_3^t\cdot\nu_3$, y la distancia de Mahalanobis con tensor métrico Id es exactamente igual a la distancia euclidiana en $\mathbb{R}^n$, es decir $d_{Id}(x,y) = d(x,y)$.

$$= \frac{\alpha}{\sqrt{\prod_k \lambda_k}}\cdot\int_{d(\nu_3,O)<\delta} e^{-\frac{1}{2}\cdot\nu_3^T\cdot\nu_3} d\nu_{3_1}\cdots d\nu_{3_n} \tag{2-43}$$

Se tiene en cuenta entonces que los valores propios λ_k de la matriz M_i son los mismos que los de la matriz Σ_i^{-1}, y los inversos a los de la matriz Σ_i. Con esto se puede escribir $|\Sigma_i| = \prod_k \lambda_k^{-1}$. Por lo tanto s tiene $\alpha/\sqrt{\prod_k \lambda_k} = (2\cdot\pi)^{-n/2}$. De esta forma, escribiendo $z = \nu_3$ se llega finalmente al resultado:

$$\int_{d_{\Sigma_i}(x,\mu_i)<\delta} N(x;\mu_i,\Sigma_i)\,dx_1\cdots dx_n = \int_{d(z,O)<\delta} N(z;O,Id)\,dz_1\cdots dz_n \tag{2-44}$$

Luego de estos cambios, la dificultad en el cálculo de la integral sigue siendo alta debido a que el dominio de integración es esférico. Pero se aproximará $d(z,O) < \delta$ (que ahora resulta ser la distancia euclidiana en $\mathbb{R}^n$), por un rectángulo dado por $[-\delta,\delta]^n$. Teniendo en cuenta esto y utilizando propiedades de exponenciales, se tiene:

$$\int_{z\in[-\delta,\delta]^n} N(z;O,Id)\,dz_1\cdots dz_n = \left(\int_{-\delta}^{\delta} N(u;0,1)\,du\right)^n = \left(2\cdot\int_0^{\delta} N(u;0,1)\,du\right)^n \tag{2-45}$$

Haciendo la aproximación en estos dos dominios se llega, despejando de la ecuación anterior, al problema que plantea la ecuación 5-15. Volviendo hacia atrás en los cambios realizados en el cálculo de la integral, se puede observar ver (Fig. B-45) que el dominio A elipsoidal original, se aproxima por el rectángulo que circunscribe a la elipse. Se puede ver además que el resultado depende únicamente de la dimensionalidad del espacio, no depende de Σ_i ni de μ_i.

C. Aproximación a la condición de no interferencia entre gestos

Se parte de la hipótesis de que $s\,(t)$ se mueve en linea recta entre centroides de distintos gestos, y es en este sentido en el que se desea que los gestos no interfieran.

Dados entonces dos gestos i y j, se buscará que el segmento que va desde μ_i hasta μ_j no tenga ningún punto que satisfaga el sistema 5-16 y ningún punto que satisfaga 5-17. Para esto se verá que para cualquiera de los dos sistemas, el segmento que va desde un centroide al otro se puede dividir en tres sub segmentos: un segmento que se encuentra dentro de la elipse del gesto i, un segmento dentro de la elipse del gesto j y un tercer segmento (que podría existir o no) que se encuentra fuera de las elipses.

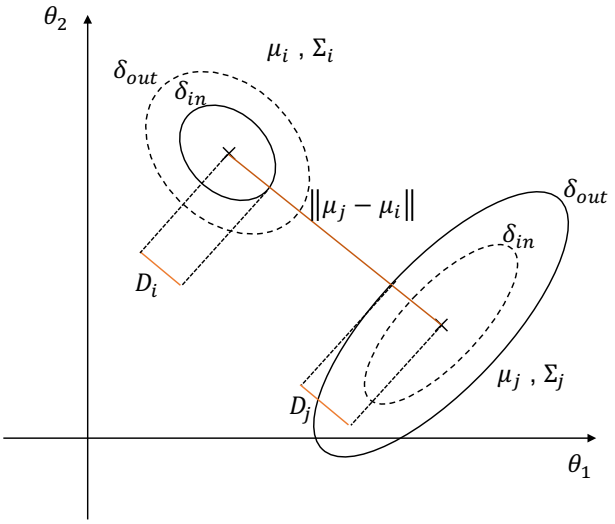


Figura C-46: Imagen representativa en $\mathbb{R}^2$ de la condición de no interferencia impuesta por la ecuación 5-16 entre un gesto i y un gesto j.

De la figura C-46 se entiende que la condición de no interferencia según la ecuación 5-16 será $D_i + D_j < \|\mu_j - \mu_i\|$. Los valores D_i y D_j son las medidas del segmento entre los centroides, que se encuentran dentro de las elipses de cada gesto.

Para determinar estas distancias se construye un versor $\hat{v} = {}^{(\mu_j - \mu_i)}/_{\|\mu_j - \mu_i\|}$, de modo que se cumplirá:

$$D_i^2 \cdot \hat{v}^T \cdot \Sigma_i^{-1} \cdot \hat{v} = \delta_{in}^2 \tag{3-46}$$

$$D_j^2 \cdot \hat{v}^T \cdot \Sigma_i^{-1} \cdot \hat{v} = \delta_{out}^2 \tag{3-47}$$

De lo anterior, se deduce entonces que la condición para que no haya interferencia según la ecuación 5-16 será:

$$\sqrt{\frac{\delta_{in}^2}{\hat{v}^T \cdot \Sigma_i^{-1} \cdot \hat{v}}} + \sqrt{\frac{\delta_{out}^2}{\hat{v}^T \cdot \Sigma_j^{-1} \cdot \hat{v}}} < \|\mu_j - \mu_i\| \tag{3-48}$$

utilizando la definición del versor $\hat{v}$, y recordando la ecuación 5-13, se puede modificar esta expresión llegando a la condición:

$$\frac{\delta_{in}}{d_{\Sigma_i}(\mu_j, \mu_i)} + \frac{\delta_{out}}{d_{\Sigma_j}(\mu_i, \mu_j)} < 1 \tag{3-49}$$

el razonamiento será análogo para satisfacer la ecuación 5-17, la condición de no interferencia será entonces:

$$\begin{cases} \frac{\delta_{in}}{d_{\Sigma_i}(\mu_j,\mu_i)} + \frac{\delta_{out}}{d_{\Sigma_j}(\mu_i,\mu_j)} & < 1 \\ \frac{\delta_{out}}{d_{\Sigma_i}(\mu_j,\mu_i)} + \frac{\delta_{in}}{d_{\Sigma_j}(\mu_i,\mu_j)} & < 1 \end{cases} \tag{3-50}$$

D. Análisis de datos en Scilab

D.1. Codificación de la base de datos

Los archivos generados por el programa para la base de datos tiene el nombre "<gesto>#<persona>#<distancia>#<iluminación>.csv". Cada uno de los indicadores en el nombre del archivo se explican en la siguiente tabla:

Identificadores de gesto	
Gesto	Código
Abrir boca	01
Sonreír	02
Mueca derecha	03
Mueca izquierda	04
Fruncir ceño	05
Levantar cejas	06
Guardar labios	07
Disentir con cabeza	08
Asentir con cabeza	09
Beso	10

(a)

Identificadores de persona
Código
001
002
003
004
005
006
007

(b)

Identificadores de distancia	
Distancia	Código
Cercana ($\approx 70cm$)	0
Mediana ($\approx 120cm$)	1
Lejana ($\approx 150cm$)	2

(c)

Identificador de iluminación	
Condición de luminosidad	Código
Escasa	0
Abundante	1

(d)

Tabla D-7: Tablas de etiquetas para el nombre de los archivos de la base de datos de prueba.

Teniendo en cuenta esto, por ejemplo, un archivo con el nombre "01#004#2#1.csv" indica que el archivo se corresponde con el gesto "abrir boca", realizado por la persona 4, a una distancia de aproximadamente $120cm$ de la cámara, con una condición de luminosidad buena.

D.2. Librería de funciones

La librería de análisis de datos en Scilab se compone de los siguientes archivos:

- **DataBase.sci**: Funciones para levantar los archivos desde la base de datos. Contiene funciones que ayudan a filtrar los archivos indicando la persona que los realizó, o el gesto que se desea estudiar para levantar los archivos correspondientes.
- **Tratamiento.sce**: Contiene todo el conjunto de funciones que ayudan a tratar a los *landmarks*. Desde devolver un vector de puntos obteniendo la evolución en la posición de determinado *landmark* en el tiempo hasta funciones que determinan el área encerrada por cierto conjunto de *landmarks*.
- **Animación.sci**: Función que permite animar una muestra de *landmarks* para reproducir en un pequeño vídeo el movimiento de los *landmarks* en la muestra de datos.
- **Funciones.sci**: Conjunto de funciones para crear gráficas nuevas con cuadriculado, y para graficar la TFTD de un vector de datos.
- **pasabanda_kaiser.sci**: Función que devuelve la respuesta al impulso de un filtro pasabanda kaiser pasando por parámetros sus características.
- **Analisis.sci**: Conjunto de funciones que determinan la evolución en el tiempo de determinadas medidas en base a los *landmarks* dado un archivo (toma de datos), con el fin de luego analizarlas o compararlas con los resultados obtenidos en otras muestras. Las funciones definidas en este archivo pueden utilizarse como andamiaje a la hora de querer implementar una función de análisis propia para los *landmarks*.
- **interseccion.sci**: Funciones para el cálculo de la distancia de Mahalanobis y algoritmos de intersección.
- **multidimensional.sci**: Definición de la gaussiana estándar y algoritmo de determinación de radios.

D.3. Caso de uso

Se busca encontrar un vector descriptor adecuado para determinar si las cejas del usuario se encuentran alzadas o no. Para eso se ve que podría ser útil utilizar para cada uno de los ojos un descriptor, indicando la altura de la ceja del ojo izquierdo, y lo mismo para el ojo derecho, normalizada en su longitud horizontal. De esta forma se podrá caracterizar a las cejas mediante dos descriptores $\theta_1 = {}^{\|l_{25}-l_{48}\|}/_{\|l_{43}-l_{46}\|}$ y $\theta_2 = {}^{\|l_{20}-l_{42}\|}/_{\|l_{37}-l_{40}\|}$

Se define entonces en el archivo "analisis_cejas.sci" una función "datosLongCeja_der(archivo)" que devuelve θ_1, y una función "datosLongCeja_izq(archivo)" que devuelve θ_2. A continuación se puede ver el código de Scilab utilizado para la función de la ceja derecha.

```
//Datos de longitudes de cejas
function y = datosLongCeja_der(archivo)
        //Cargado de datos
        datos = csvRead(direccion_BBDD + archivo,';');
        l1 = posicion_tiempo(25, datos);
        l2 = posicion_tiempo(48, datos);
        h1 = posicion_tiempo(43, datos);
        h2 = posicion_tiempo(46, datos);
        //Longitudes de ceja
        long = mod(l1- l2);
        horizontal = mod(h2 - h1);
        y = long./horizontal;
endfunction
```

Luego se procede a definir en un nuevo archivo que se llamará "cejas.sce", donde se analizará la elección de estos descriptores y se verá si es apropiada o no. El código que se implementará deberá incluir las librerías de funciones para poder realizar el análisis, levantar el archivos que se desea analizar (se buscará analizar el archivo "06#002#0#0.csv "), y luego, se guardará en dos variables y_1 e y_2 cada uno de los descriptores. Luego se graficarán las dos variables, para ver la trayectoria en el tiempo para el vector $\Theta = [\theta_1, \theta_2]^T$.

```
// ---------- Analisis de cejas ----------

// Configuraciones iniciales
//Limpia memoria y cierra ventanas
clear; clc; xdel(winsid());

//Incluye las librerias
direccion_Librerias = "../Librerias/";
exec(direccion_Librerias + 'Funciones.sce');
exec('analisis_cejas.sci');
exec(direccion_Librerias + 'Tratamiento.sce');
exec(direccion_Librerias + 'DataBase.sci');
exec(direccion_Librerias + 'Animacion.sci');
//Filtra los archivos a analizar
archivos = archivosCSV();
archivos = filtroPers(archivos, 2);
archivos = filtroGesto(archivos, 6);
archivos = filtroDist(archivos, 0);
//Carga datos datos = csvRead(direccion_BBDD + archivos(1),';');

//Funciones para determinar altura de cejas
y1 = datosLongCeja_der(archivos(1));
y2 = datosLongCeja_izq(archivos(1));

//Hace las graficas crearFig();
plot(y1,y2,'black.');
xtitle('Evolución de las cejas en el tiempo',
'Altura ceja der.', 'Altura ceja izq.');
```

La salida del algoritmo una vez que se corre en Scilab se muestra en la figura D-47, de la cual se puede ver que existen dos lóbulos bien diferenciados correspondientes al estado “cejas levantadas” y “cejas en reposo”. Se concluye entonces que este vector descriptor permite determinar el estado de las cejas.

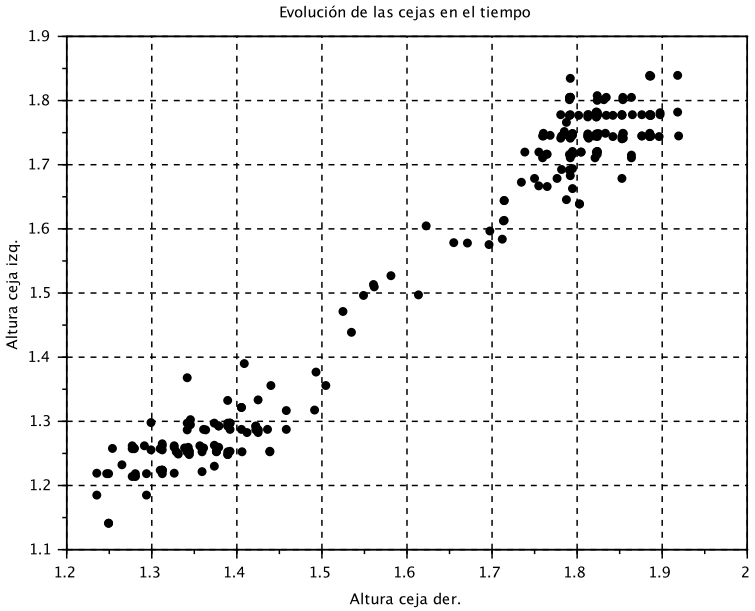


Figura D-47: Evolución del vector Θ de las cejas para la toma de datos del archivo "06#002#0#0.csv ".

D.4. Estudio extendido sobre la boca

En la gráfica de la sección 5.12.2 (ver Fig. 5-27) se puede ver que el gesto beso está bastante cerca del conjunto de puntos asociados al gesto normal,es por eso que se decidió agregar el descriptor θ_4. En la Fig. D-48se puede ver como las muestras asociadas al gesto beso se separan del conjunto de reposo.

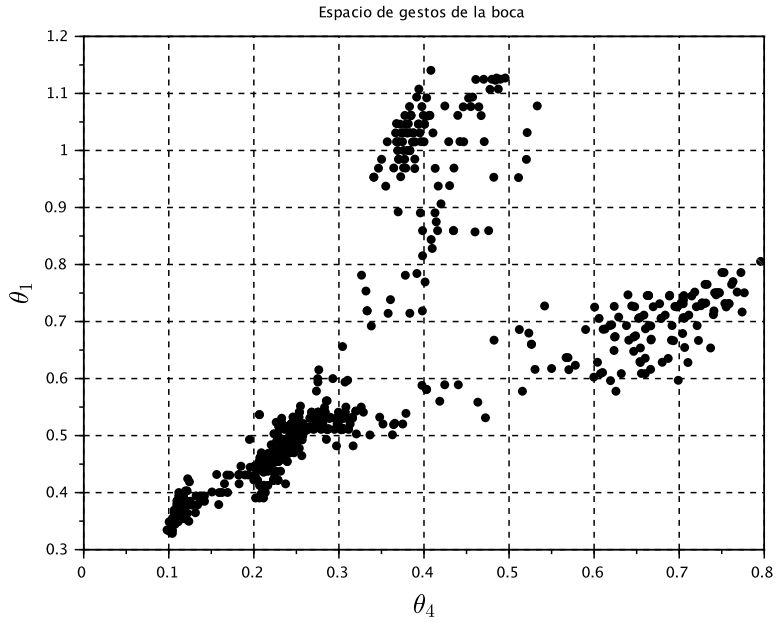
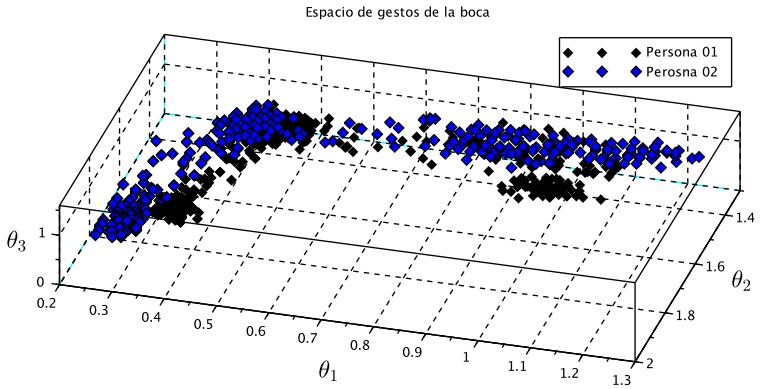
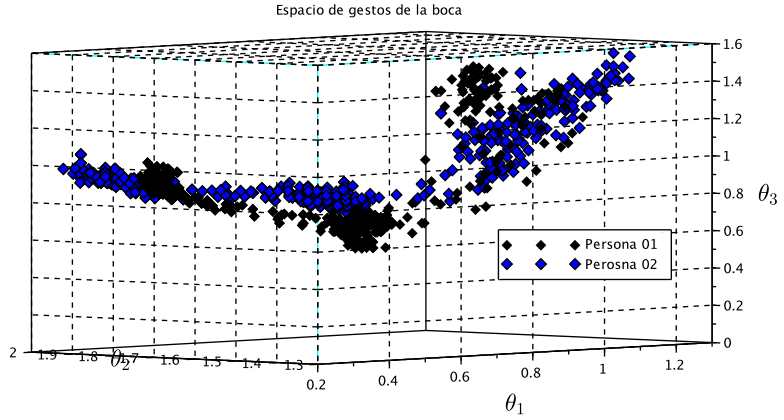


Figura D-48: Proyección de la evolución del vector Θ en el plano formado por los descriptores θ_1 y θ_4 para los cuatros gestos.

Para verificar la robustez del conjunto frente a cambios de persona, la Fig. D-49 muestra les conjunto de puntos en el espacio para dos de las personas del conjunto de la base de datos, mientras hacen los gestos de abrir boca y sonreír.



(a)



(b)

Figura D-49: Proyección de las muestras de Θ en el espacio conformado por θ_1, θ_2 y θ_3, mientras se realizan los gestos de abrir, sonreír y dejar boca en reposo.

E. Ejemplos de cambios en los gestos a detectar

A continuación se detallan ejemplos que podrían utilizarse para agregar nuevos gestos o partes a la librería de detección de gestos. Ninguno de los ejemplos está probado por lo que no se garantiza que puedan funcionar, son solamente ejemplos ilustrativos.

E.1. Agregar gesto: Mueca derecha

En este caso el gesto pertenece a una parte ya definida (la boca) por lo que no se requiere definir una nueva.

Los descriptores predefinidos para la boca no son suficientes como para detectar la mueca derecha por lo que se agregará un descriptor nuevo: distancia entre la comisura derecha y la nariz, normalizada con la distancia entre los ojos. Se define la función llamada **cociente_MuecaDer** en "estados.cpp" para devolver el descriptor adicional:

```
//Mueca derecha
float cociente_MuecaDer(full_object_detection shape){
        float cociente = 0;
        float mod_boca, mod_ojos;
        dlib::point ojos = retornarVectorOjos(shape);
        dlib::point boca = retornarVectorBocaNariz_Der(shape);
        mod_boca = longitud(boca); mod_ojos = longitud(ojos);
        cociente = mod_boca/mod_ojos;
        return cociente;
}
```

Se agrega además a la función **estado_boca** en "estados.cpp", una nueva dimensión al vector descriptor Θ de la boca, para poder detectar el gesto, donde se llama a la función recién definida.:

```
// Retorno de estado de la boca
dlib::matrix<double> estado_boca(full_object_detection shape){
        std::vector<dlib::point> boca_vertical = retornarBocaInterna_v(shape);
       std::vector<dlib::point> boca_horizontal = retornarBocaInterna_h(shape);
        dlib::matrix<double> ret(5,1); //Matriz de retorno (DIMENSION 5)
        ret(0,0) = longitud(boca_horizontal)/length(retornarVectorOjos(shape));
        ret(1,0) = longitud(boca_vertical)/length(retornarVectorOjos(shape));
        ret(2,0) = retornarBoca_cruz1(shape) + retornarBoca_cruz2(shape);
        ret(3,0) = ret(1,0)/(ret(0,0)*ret(0,0));
        ret(4,0) = cociente_MuecaDer(shape); //NUEVO DESCRIPTOR
        return ret;
}
```

Recordar también agregar las firmas de las funciones en el archivo "estados.h". En el archivo de configuración de la boca, "Boca.txt" quedará entonces modificado indicando que el gesto "Mueca derecha" es el gesto 5 con las líneas:

```
Normal;1;1;
Beso;2;1;
Sonrisa;3;1;
Abierta;4;1;
MuecaDerecha;5;1;
```

E.2. Agregar gesto: Fruncir nariz

Dado que nariz no es una parte incluida en el sistema. Lo primero a hacer es elegir los descriptores y crear la función de estados. Los descriptores pueden ser:

- distancia del borde derecho de la nariz al borde derecho del ojo normalizados por la distancia entre los ojos,
- ídem para el lado izquierdo,
- distancia del borde derecho de la nariz al borde izquierdo del ojo normalizados por el largo de un ojo o
- área del triángulo de la nariz sobre el área de la cara

Una vez elegidos los primeros descriptores hay que crear una función que los devuelva. Asumiendo que se eligen el primero y el segundo mencionados, se consulta en el mapa de *landmarks* (ver fig. 4-17) para identificar cuáles se necesitan (bordes de la nariz: l_{31} y l_{35}, bordes de los ojos l_{39} y l_{42}). En "estados.cpp" se agrega la función **cociente_NarizIzq** para crear el descriptor.

```
//Descriptor de nariz
float cociente_NarizIzq(full_object_detection shape){
	float cociente = 0;
	float mod_nariz, mod_ojos;
	Dlib::point ojos = retornarVectorOjos(shape);
	Dlib::point nariz = shape.part(39)-shape.part(31);
	mod_nariz = Dlib::length(nariz);
	mod_ojos = Dlib::length(ojos);
	cociente = mod_nariz/mod_ojos;
	return cociente;
}
```

Se crea una función análoga para el lado derecho llamada **cociente_NarizDer**. Solo falta crear la función de actualización de estado **estado_nariz** que devuelve el vector descriptor de la nariz:

```
//Estado nariz
Dlib::matrix<double> estado_nariz(full_object_detection shape){
	Dlib::matrix<double> ret(2,1); //Matriz de retorno
	double aux1 = cociente_NarizIzq(shape);
	double aux2 = cociente_NarizDer(shape);
	ret(0,0) = aux1;
	ret(1,0) = aux2;
	return ret;
}
```

Como no se tiene ninguna parte definida para la nariz, hay que agregar el archivo "Nariz.txt" en el directorio definiendo los estados posibles. En el archivo se tendrán las líneas:

```
Normal;1;1;
Fruncido;2;1;
```

En el archivo "configuracion.h" se agrega un nombre, un ID y el archivo en el que se tendrán los gestos definidos (referencia al archivo "Nariz.txt"):

```
#define NARIZ "Nariz";
#define NARIZ_ID 4;
#define ARCHIVO_NARIZ "./Nariz.txt";
```

Por otro lado, en el archivo "entrenamiento.cpp" se agregan las líneas necesarias para que el sistema entrene este gesto para esta parte:

```
Parte parte_nariz(NARIZ, NARIZ_ID, &estado_nariz);
agregarGestos(ARCHIVO_NARIZ, &parte_nariz);
vector_partes.push_back(parte_nariz);
```

Análogamente en el archivo “aplicacion.cpp” deben agregarse las lineas necesarias para agregar la nariz como parte:

```
Parte parte_nariz(NARIZ, NARIZ_ID, &estado_nariz);
vector_partes.push_back(parte_nariz);
```

F. Lista de funciones implementadas en C++

F.1. Tratamiento.h

```
std::vector<dlib::point> returnPoints(
        full_object_detection shape, int prim, int ult);
std::vector<image_window::overlay_line> lineas(
        const std::vector<dlib::point> puntos,
        const rgb_pixel color);
float areaVectores(
        const dlib::point punto1,
        const dlib::point punto2);
std::vector<dlib::point> segmentar(
        const std::vector<dlib::point> puntos,
        int inicio, int final);
float areaFace(
        full_object_detection shape);
float longitud(
        const std::vector<dlib::point> puntos);
float areaSector(
        std::vector<dlib::point> borde);
std::vector<dlib::point> concatenar(
        const std::vector<dlib::point> vector1,
        const std::vector<dlib::point> vector2);
std::vector<dlib::point> puntosRect(rectangle rect);
rectangle rectTrakking(full_object_detection shape);
```

F.2. Estados.h

```
// Funciones de descriptores
std::vector<dlib::point> retornarOjoIzquierdo(
        full_object_detection shape);
std::vector<dlib::point> retornarOjoDerecho(
        full_object_detection shape);
std::vector<dlib::point> retornarBocaInterna(
        full_object_detection shape);
std::vector<dlib::point> retornarBocaInterna_h(
        full_object_detection shape);
std::vector<dlib::point> retornarBocaInterna_v(
        full_object_detection shape);
float modulo(dlib::point vect);
dlib::point retornarVectorNariz(
        full_object_detection shape);
dlib::point retornarVectorOjos(
```

```
        full_object_detection shape);
dlib::point retornarVectorBoca_v(
        full_object_detection shape);
dlib::point retornarVectorBoca_h(
        full_object_detection shape);
dlib::point retornarVectorBocaNariz_Der(
        full_object_detection shape);
dlib::point retornarVectorBocaNariz_Izq(
        full_object_detection shape);
double orientacionHorizontal(
        full_object_detection shape);
double orientacionVertical(
        full_object_detection shape);
float retornarAreaNariz(
        full_object_detection shape);
float cociente_Boca(
        full_object_detection shape);
float cociente_Sonrisa(
        full_object_detection shape);
float cociente_MuecaDer(
        full_object_detection shape);
float cociente_MuecaIzq(
        full_object_detection shape);
float areaBoca(
        full_object_detection shape);
float areaCejaIzq(
        full_object_detection shape);
float areaCejaDer(
        full_object_detection shape);
float areaOjo(
        full_object_detection shape);
//Funciones de estados
dlib::point retornarVectorCejaIzq_v(
        full_object_detection shape);
dlib::point retornarVectorCejaIzq_h(
        full_object_detection shape);
double estCejaIzq(
        full_object_detection shape);
double retornarBoca_cruz1(
        full_object_detection shape);
double retornarBoca_cruz2(
        full_object_detection shape);
dlib::matrix<double> estado_boca(
        full_object_detection shape);
dlib::matrix<double> estado_ojos(
        full_object_detection shape);
```

```
dlib::matrix<double> estado_expresion(
        full_object_detection shape);
dlib::matrix<double> estado_orientacion(
        full_object_detection shape);
double retornarEstOjoDer(
        full_object_detection shape);
double retornarEstOjoDer_b(
        full_object_detection shape);
double retornarEstOjoDer_c(
        full_object_detection shape);
double retornarEstOjoIzq(
        full_object_detection shape);
double retornarEstOjoIzq_b(
        full_object_detection shape);
double retornarEstOjoIzq_c(
        full_object_detection shape);
double sgn(double unNum);
```

F.3. Entrenamiento.h

```
dlib::matrix<double> covariance_matrix(
        std::vector<std::vector<double> > training_set);
dlib::dpoint centroids(
        std::vector<std::vector<double> > training_set);
dlib::matrix<double> centroids_d(
        std::vector<std::vector<double> > training_set);
double mod(std::vector<double> v);
double mean_value(std::vector<double> datos);
double max_vp(dlib::matrix<double> elipsoide);
bool gesto_activo(
        dlib::matrix<double> estado,
        dlib::matrix<double> centroide,
        double radio);
dlib::matrix<double> covariance_matrix_vector(
        std::vector<dlib::matrix<double> > training_set);
dlib::matrix<double> centroids_vector(
        std::vector<dlib::matrix<double> > training_set);
double criterio90(double dim);
double gaussiana(double x);
double calcular_rad(double dim, double lim);
```

G. Instrucciones de instalación para OpenCV y Dlib

Para instalar OpenCV y Dlib, los programas necesarios para la ejecución del sistema, se debe seguir las siguientes instrucciones de instalación. Por consola y en la carpeta **/usr/local/lib** ejecutar:

```
#INSTALL OPENCV
sudo apt-get install build-essential
sudo apt-get install
cmake git libgtk2.0-dev pkg-config
libavcodec-dev libavformat-dev libswscale-dev
git clone https://github.com/opencv/opencv.git
cd ~/opencv
mkdir release
cd release
cmake
-D CMAKE_BUILD_TYPE=RELEASE -D
      CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
cd..

#INSTALL DLIB IN OPENCV
mkdir SharedLibs
cd SharedLibs
git clone https://github.com/davisking/Dlib.git
cd Dlib/examples
mkdir build
cd build
cmake --build . --config Release
cd examples/build/
wget http://Dlib.net/files/
shape_predictor_68_face_landmarks.dat.bz2
bzip2 -d shape_predictor_68_face_landmarks.dat.bz2
```