Trabajo de Fin de Grado:

Estudio e implementación de algoritmos para la detección de rostros en la banda del infrarrojo lejano.

Grado en Física.

Autor: Adrián Cordones Martínez.
Tutor: Juan Antonio Leñero Bardallo.
16 de octubre de 2023.

Agradecimientos

Me gustaría agradecer a toda mi familia y amigos por su apoyo constante e incondicional a lo largo de toda mi trayectoria formativa.

También a mi tutor en este proyecto, Juan Antonio Leñero, por brindarme esta oportunidad y por dedicarme su tiempo y conocimientos.

Abstract

In this Final Degree Project (FDP) we aim to detect faces in the infrared range and determine their average temperature. To achieve this, we have studied the operation of microbolometer infrared sensors and some common methods for object detection. We have used some of the most relevant functions and classes from OpenCV in C++ to develop some useful algorithms that can be included into the final program.

For this task, an experimental device has been characterized, which uses an embedded system, a Raspberry Pi 3B+, capable of using two cameras, one operating in the visible range and another in the infrared (LWIR). Taking this into account, the requirements that this device must meet are raised, including a balance between efficiency and economic feasibility.

Several methods have been employed to address this task and a series of tests have been applied to verify the proper functioning of the experimental device. Finally, we present the final appearance of the program along with the improvements that have been made and it is considered the possibility of making future changes to the system.

Resumen

En este Trabajo de Fin de Grado (TFG) se busca detectar rostros en el rango infrarrojo y determinar su temperatura media. Para ello, se han estudiado el funcionamiento de los sensores infrarrojos de microbolómetro y algunos métodos habituales para la detección de objetos. Se han utilizado algunas de las funciones y clases más relevantes de OpenCV en C++ para desarrollar distintos algoritmos útiles que puedan ser incorporados en el programa final.

Para esta labor, se ha caracterizado un dispositivo experimental que hace uso de un sistema embebido, una Raspberry Pi 3B+, que incorpora la posibilidad de utilizar dos cámaras, una que trabaja en el rango visible y otra en el infrarrojo (LWIR). Teniendo esto en cuenta, se plantean los requisitos que este dispositivo debe cumplir, entre ellos un equilibrio entre eficiencia y viabilidad económica.

Se han empleado varios métodos para abordar dicha tarea y se han realizado una serie de pruebas para comprobar el correcto funcionamiento del dispositivo experimental. Por último, se muestra la apariencia final del programa junto a las mejoras que se han realizado y se contempla la posibilidad de realizar cambios a futuro en el sistema.

Índice general

Índice de tablas

Índice de figuras

Lista de abreviaturas

IR	*Infrared*
FLIR	*Forward Looking Infrared*
LWIR	*Longwave Infrared*
IRT	*Infrared Termography*
FPA	*Focal Plane Array*
ROIC	*Readout Circuit*
USB	*Universal Serial Bus*
I2C	*Inter-Integrated Circuit*
CCI	*Camera Control Interface*
RGB	*Red, Green and Blue*
HSV	*Hue, Saturation and Intensity value*
YCbCr	*Luminance (Y) and Chrominance (Cb and Cr)*
GPIO	*General Purpose Input/Output*
SBC	*Single-Board Controller*
ROI	*Region of Interest*

Capítulo 1

Introducción

En este capítulo se expondrán las razones que motivan el estudio y desarrollo de este proyecto y, posteriormente, los objetivos que se buscan alcanzar.

1.1. Motivación

En los últimos años, el mundo ha enfrentado varias situaciones epidémicas, de urgencia sanitaria o de pandemia mundial: COVID-19, ébola, viruela símica, etc. Todas estas enfermedades presentan como síntoma habitual común la fiebre [1–3].

La población tiende a concentrarse cada vez más en las grandes metrópolis, las cuales presentan una mayor densidad de población, en la Figura 1.1.1 se muestran distintas ciudades españolas como ejemplo. Estas condiciones las hace un entorno favorable para la propagación de enfermedades epidémicas, facilitando su transmisión y aumentando el riesgo de contagio. Además, en un mundo globalizado cada vez más conectado las enfermedades pueden propagarse a otros países más fácilmente por avión o por barco.

Estos motivos, junto a la ausencia de medidas preventivas, nos llevan a la necesidad de buscar formas efectivas de detectar estos focos de contagios y proceder a su tratamiento médico y, si fuera necesario, a su posterior cuarentena. Necesitamos una tecnología que nos permita medir la fiebre; un síntoma cuantificable y relativamente fácil de detectar.

Esta tecnología debe cumplir una serie de requisitos [5]. Buscamos que sea fácil de usar, portátil, autónoma y lo suficientemente precisa para distinguir la fiebre de una simple destemplanza. Además los resultados deben ser fácilmente interpretables tras su medición de forma inmediata y estos deben poder ser almacenados para hacer un seguimiento de los casos y la realización de cálculos estadísticos.

Para el desarrollo de este Trabajo de Fin de Grado, se ha optado por el uso de sistemas embebidos, debido a que son muy versátiles por su pequeño tamaño y bajo consumo, lo que favorece su instalación y comercialización. En concreto, se ha implementado una Raspberry Pi $3B+$ en el sistema experimental utilizado. El único inconveniente posible según el modelo, y por tanto su precio, es su baja capacidad de cómputo, por lo que para ello es crucial la optimización y depuración del software desarrollado.

Teniendo en cuenta los requisitos anteriores y el entorno en el que se va a usar, ha sido conveniente el uso de C++ debido a que los niveles de optimización alcanzables en este lenguaje son superiores a otros de uso habitual. Se hará uso de la librería OpenCV [6–8] dado que nos permite realizar todas las tareas abordadas.

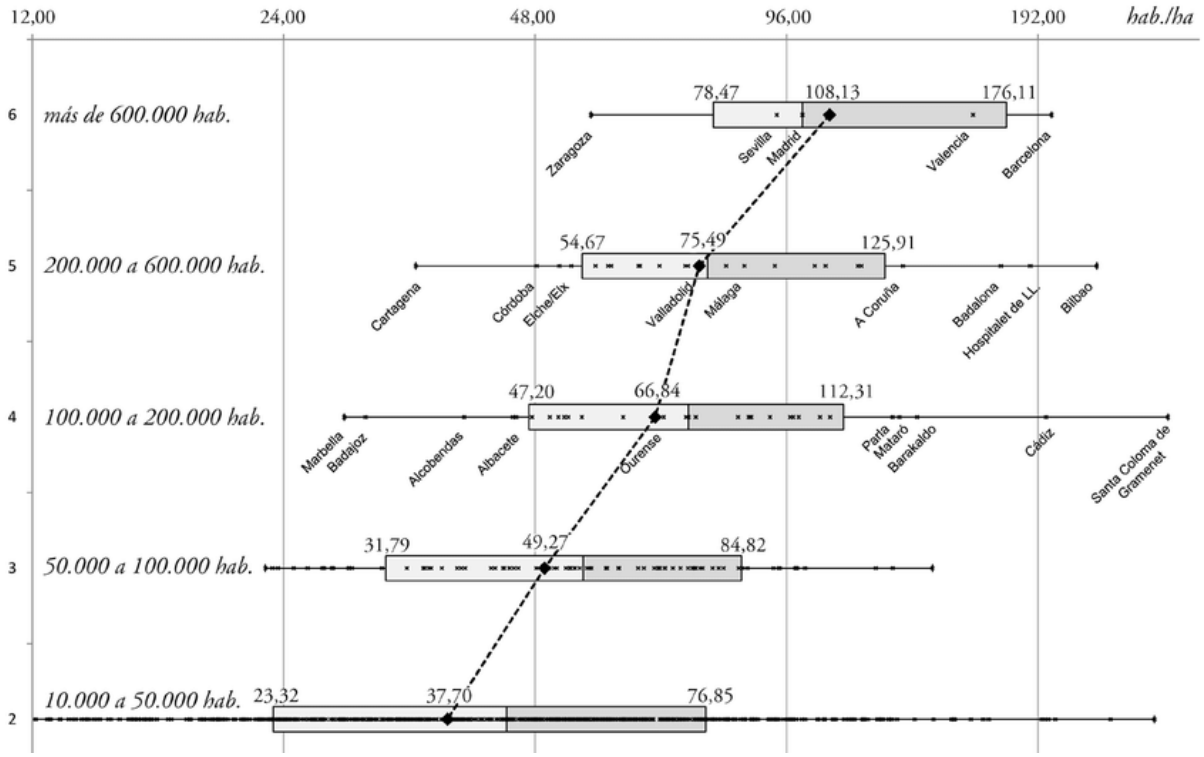


Figura 1.1.1: Densidad de población (en $hab./hm^2$) de distintas ciudades de España. Con línea discontinua: valores medios; y en gráfico box plot: valores mínimos, percentil 15 %, mediana, percentil 85 % y máximo [4]

1.2. Objetivos

El objetivo principal es desarrollar un programa en una Raspberry $3B+$ que pueda utilizar dos cámaras, una del rango visible y otra del IR, y sea capaz de tomar fotografías y, a partir de estas, detectar caras y objetos anidados, como ojos o gafas, para medir su temperatura media tras una calibración previa. Para lograr esto, el programa debe abordar las siguientes labores:

- Acceder a una cámara mediante bus I2C o por USB y tomar fotografías.
- Leer fotografías y almacenarlas para la posterior aplicación de algoritmos y funciones que nos permitan un adecuado procesamiento de datos.
- Cambiar el espacio de color de las imágenes (RGB, gray, etc.) para facilitar la aplicación de estas funciones.
- Detectar rostros en el espectro infrarrojo, separarlas del resto de la imagen y medir su temperatura media.
- Almacenar las imágenes tomadas y procesadas en un fichero en el disco para poder visualizarlas a posterior o para volver a realizar operaciones con ellas.
- Mostrar las imágenes en pantalla para facilitar su interpretación en tiempo real.
- Optimizar y depurar el programa. Liberar memoria en nuestro sistema embebido siempre que sea posible para no saturarla.

Capítulo 2

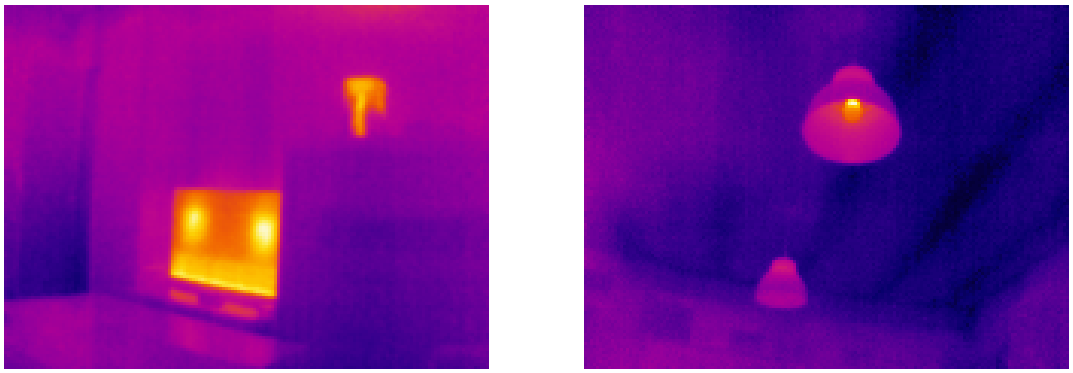
Fundamento teórico

En este capítulo se introducirán todos los conceptos teóricos necesarios para abordar los objetivos planteados en este proyecto. Para ello, se tratarán previamente las nociones físicas necesarias y se presentarán todas las herramientas que serán de utilidad.

2.1. Captura de imágenes térmicas

Hoy en día, las cámaras térmicas tienen una gran cantidad de aplicaciones en campos como la industria, la educación, la investigación o la medicina, entre otros. Por ejemplo, vemos su uso en la medicina moderna para tratar desde trastornos como la artritis inflamatoria o la osteoartritis hasta algunos síndromes que afectan a la circulación y otras malformaciones vasculares [5], [9]. Esta tecnología también puede aplicarse con motivos educativos, en la realización de experimentos cualitativos que se adapten al nivel de formación de estudiantes, sin que se requiera un entendimiento profundo de la teoría de transferencia del calor [10].

Aunque en un principio podría parecer que esta tecnología presenta un coste elevado, se puede ver que esto es una realidad relativa, ya que en muchos de estos campos su uso permite evitar tecnologías más complejas, y por tanto de mayor coste, e incluso en algunos casos automatizar procesos que por otras vías requerirían de la participación de operarios, acortando los tiempos en los que se procesan datos y se obtienen resultados. El coste dependerá también del tipo de tecnología que incorpora cada sensor infrarrojo.



(a) (b)

Figura 2.1.1: Ejemplos de imágenes térmicas: (a) Electrodomésticos (b) Lámparas

La imágenes térmicas se generan en un proceso por el cual se convierte radiación infrarroja, el calor que emiten los objetos presentes, en imágenes visibles que plasman con fidelidad la distribución espacial de las diferencias de temperatura en la escena captada [11]. La imagen final es una representación visual pero no real, puesto que está en el rango visible. El rango de longitud de onda en el que opera habitualmente una cámara térmica comercial abarca desde los 0,8 μm hasta los 14 μm.

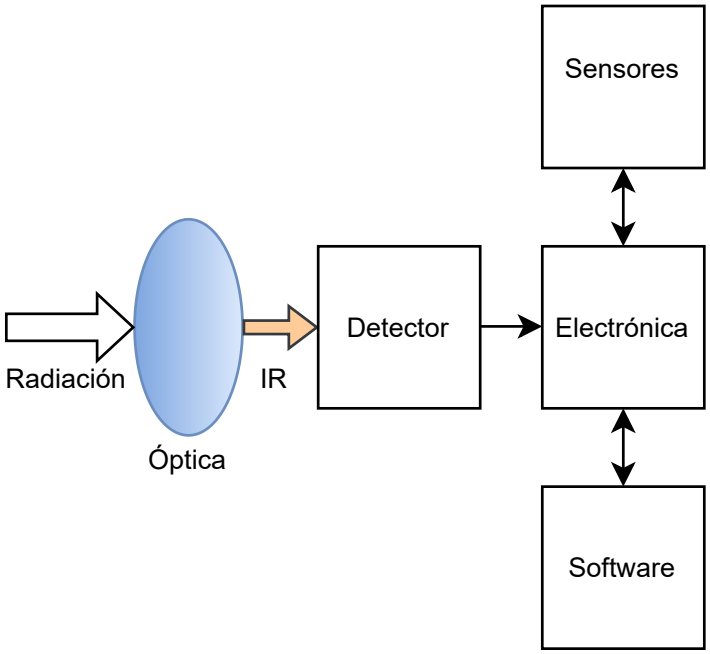


Figura 2.1.2: Diagrama de bloques de una cámara térmica [11]

Las cámaras térmicas tienen varias partes diferenciadas que participan en la captura y procesamiento de imágenes [11], representadas en la Figura 2.1.2:

- **Óptica:** La radiación incidente abarca todo el espectro, por tanto, es necesario hacer uso de un sistema óptico, como lentes o filtros, que transmita únicamente la radiación infrarroja. A diferencia de las cámaras que trabajan en el rango visible, en este tipo de tecnología se incorporan semiconductores en lugar de vidrio.
- **Detector:** Existen distintos tipos: de matriz de plano focal, de barrido, etc. Se encargan de convertir la radiación infrarroja absorbida en señales eléctricas. Por lo general, no están basados en silicio como los detectores usados en el rango visible, aunque existen algunas aplicaciones como los detectores de silicio amorfo (a-Si).
- **Electrónica:** Todo el hardware y circuitos adicionales para cálculos, correcciones, procesamiento de imágenes, etc. Por ejemplo, es posible incluir hardware externo para almacenar y procesar datos, como CPU, RAM y memorias, o para visualizar las imágenes, como GPU y pantallas.
- **Sensores:** No son estrictamente obligatorios, pero se suelen incluir sensores para automatizar procesos de calibración (temperatura ambiente, humedad relativa, emisividad, distancia relativa a objetos, etc.). Según la aplicación, esta calibración puede realizarse de forma externa o tenerla en cuenta en la incertidumbre de las medidas.

- **Software:** Engloba todo el conjunto de rutinas utilizadas: interfaz de usuario para visualizar las imágenes, programas de procesamiento de datos, controles, etc.

2.1.1. Relación entre el nivel de radiación y la temperatura

Hemos visto que una cámara térmica traduce la radiación infrarroja que recibe en una imagen que representa las diferencias de temperatura en los objetos fotografiados, pero para realizar una correcta calibración de nuestros sistemas es necesario que entendamos las variables implicadas.

Partimos de la ley física que relaciona la potencia radiada por unidad de superficie con la temperatura, la ley de Stefan-Boltzmann [12]:

$$P = \epsilon \cdot \sigma \cdot (T_e)^4 \tag{2.1.1}$$

donde T_e es la temperatura efectiva de la superficie, ϵ es la emisividad, $0 \leq \epsilon \leq 1$, que es característica de la superficie que radia e igual a la unidad en el caso de un radiador ideal o cuerpo negro y σ es la constante de Stefan-Boltzmann.

$$\sigma = \frac{\pi^2 k_B^4}{60\hbar^3 c^2} = 5{,}670373(21) \cdot 10^{-8} \, \frac{W}{m^2 \cdot K^{-4}} \tag{2.1.2}$$

Sin embargo, para la captura de imágenes térmicas se ha de tener en cuenta que la radiación que atraviesa la lente no procede únicamente de nuestro objeto de estudio. En la Figura 2.1.3 se muestran los distintos rayos de luz que llegan, en general, a la cámara infrarroja: radiación emitida y reflejada por el objeto de estudio y radiación ambiental.

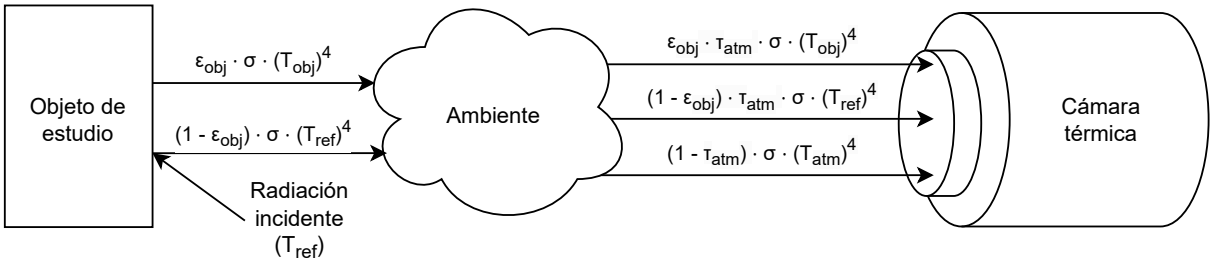


Figura 2.1.3: Trazado de rayos para una cámara térmica [13]

La potencia por unidad de superficie total incidente en la cámara térmica vendrá dada por la expresión:

$$P_{tot} = P_{obj} + P_{ref} + P_{atm} \tag{2.1.3}$$

Partiendo de esta situación queremos obtener una expresión para la temperatura a determinar, T_{obj}. Aplicamos la ley de Stefan-Boltzmann (2.1.1) a cada caso:

- Para la radiación emitida por el objeto de estudio:

$$P_{obj} = \epsilon_{obj} \cdot \tau_{atm} \cdot \sigma \cdot (T_{obj})^4 \tag{2.1.4}$$

- Para la radiación reflejada por el objeto de estudio:

$$P_{ref} = \tau_{obj} \cdot \tau_{atm} \cdot \sigma \cdot (T_{ref})^4 = (1 - \epsilon_{obj}) \cdot \tau_{atm} \cdot \sigma \cdot (T_{ref})^4 \tag{2.1.5}$$

- Para la radiación emitida por el ambiente:

$$P_{atm} = \epsilon_{atm} \cdot \sigma \cdot (T_{atm})^4 = (1 - \tau_{atm}) \cdot \sigma \cdot (T_{atm})^4 \tag{2.1.6}$$

  donde ϵ_{obj} es la emisividad del objeto, τ_{obj} es la transmitancia del objeto, ϵ_{atm} es la emisividad del ambiente y τ_{atm} es la transmitancia del ambiente.

Sustituimos las Ecuaciones ecuaciones (2.1.4) a (2.1.6) en la Ecuación (2.1.3):

$$P_{tot} = \epsilon_{obj} \cdot \tau_{atm} \cdot \sigma \cdot (T_{obj})^4 + (1 - \epsilon_{obj}) \cdot \tau_{atm} \cdot \sigma \cdot (T_{ref})^4 + (1 - \tau_{atm}) \cdot \sigma \cdot (T_{atm})^4 \tag{2.1.7}$$

Despejamos T_{obj} en la Ecuación (2.1.7) y obtenemos la expresión que buscábamos para la temperatura del objeto a estudiar:

$$T_{obj} = \sqrt[4]{\frac{P_{tot} - (1 - \varepsilon_{obj}) \cdot \tau_{atm} \cdot \sigma \cdot (T_{ref})^4 - (1 - \tau_{atm}) \cdot \sigma \cdot (T_{atm})^4}{\varepsilon_{obj} \cdot \tau_{atm} \cdot \sigma}} \tag{2.1.8}$$

En el caso de que la temperatura esperable esté en un intervalo no muy amplio de temperaturas, $T_{obj} \in [T_0 - \Delta T, T_0 + \Delta T]$ con $\Delta T \sim T_0$, podemos expandir en serie de Taylor en torno a T_0 la Ecuación (2.1.8):

$$T_{obj} \simeq T_0 + \left.\frac{dT_{obj}}{dP_{tot}}\right|_{P_0} \cdot (P_{tot} - P_0) \tag{2.1.9}$$

donde T_0 es el valor promedio de temperatura, P_0 es la potencia radiada por unidad de superficie correspondiente a ese valor de temperatura y la derivada evaluada queda como:

$$\left.\frac{dT_{obj}}{dP_{tot}}\right|_{P_0} = \frac{1}{4} \frac{1}{\sqrt[4]{\epsilon_{obj}\tau_{atm}\sigma \left(P_0 - (1 - \varepsilon_{obj})\, \tau_{atm}\sigma\, (T_{ref})^4 - (1 - \tau_{atm})\, \sigma\, (T_{atm})^4\right)^3}} \tag{2.1.10}$$

Vemos, por tanto, que si trabajamos en un entorno controlado se puede utilizar la Ecuación (2.1.9) en lugar de la Ecuación (2.1.8); esto es, que se verifiquen las condiciones adecuadas para poder utilizar dicha aproximación. Las condiciones que se deben verificar son las siguientes:

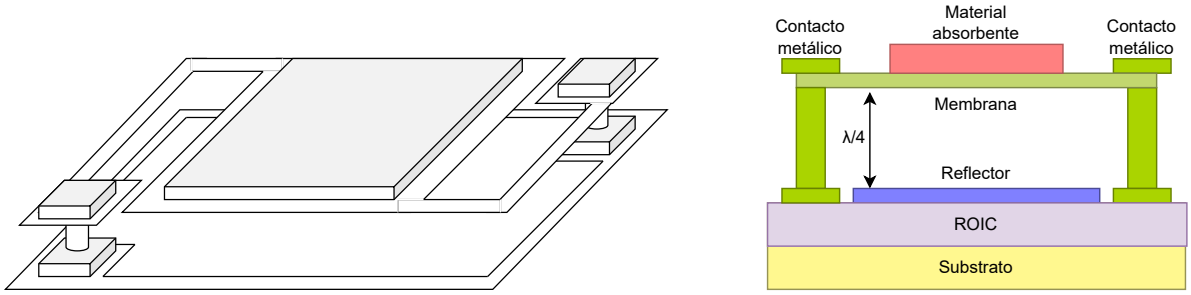
- Temperatura ambiente aproximadamente constante: $T_{atm} \sim cte$.
- Objetos de estudio con misma propiedades térmicas: $\epsilon_{obj}^{(1)} \simeq \epsilon_{obj}^{(2)} \simeq ....$
- Rango de temperaturas medibles no muy grande: $\Delta T \sim T_0$.

En esta situación, podemos calibrar el sistema de imágenes térmicas mediante un ajuste lineal, midiendo dos o más puntos de referencia (P_j, T_j) obteniendo los parámetros de calibración a y b:

$$\boxed{T_{obj} = a \cdot P_{tot} + b} \tag{2.1.11}$$

2.1.2. Detector de microbolómetro

El sensor infrarrojo más utilizado en aplicaciones de captura de imágenes térmicas es el sensor de matriz de plano focal (Focal Plane Array, FPA). Este tipo de sensor utiliza una matriz de $m \times n$ elementos detectores sensibles a la radiación infrarroja. Entre estos elementos detectores el más habitual es el microbolómetro.



(a) (b)

Figura 2.1.4: Detector microbolométrico individual [14]: (a) Estructura simplificada de un píxel (b) Sección transversal con las partes más relevantes indicadas

Los microbolómetros son un tipo de sensores no refrigerados, es decir, operan a temperatura ambiente sin la necesidad de refrigeración para su funcionamiento. Esto hace que no sean los candidatos más precisos para detectar radiación infrarroja puesto que la sensibilidad del sensor estará limitada por el ruido térmico generado por la propia estructura del sensor. Sin embargo, los microbolómetros tienen la ventaja de ser más compactos, económicos y eficientes energéticamente que los sensores refrigerados, lo que los hace más adecuados para aplicaciones comerciales y de consumo. En otro tipo de aplicaciones más técnicas es habitual encontrar sensores que trabajan en longitudes de onda más cortas, como sensores basados en fotodiodos formados con compuestos tenarios como InGaAs.

Un microbolómetro consta de varias partes importantes [14], incluyendo:

- **Material absorbente:** Cuando la radiación IR incide directamente sobre el microbolómetro, esta es absorbida por un material semiconductor. Generalmente se utiliza óxido de vanadio (VO_x, con x entre 1,8 y 2,2) o, en menor medida, silicio amorfo (a-Si). Los fotones absorbidos por el material excitan electrones de la banda de valencia a la banda de conducción, dando lugar a un aumento en la temperatura y, por tanto, una disminución de la resistencia eléctrica. Esta relación viene dada por el coeficiente de temperatura de resistencia (TCR) del material:

$$TCR = \frac{1}{R}\frac{dR}{dT} \tag{2.1.12}$$

  donde R es la resistencia eléctrica y T es la temperatura.

- **Reflector:** Como parte de la radiación IR es transmitida por el material absorbente, se coloca un material reflector justo debajo para reflejarla de vuelta. Esto evita que la radiación IR no deseada alcance el substrato y produzca señales eléctricas espurias.

- **Membrana:** El material absorbente está colocado encima de una membrana muy fina (en torno a 0,1 μm de grosor). Esta membrana está aislada térmicamente del substrato mediante una cavidad intermedia en la que se da vacío. La respuesta espectral máxima se da para radiación IR cuya longitud de onda es aproximadamente cuatro veces la anchura de la cavidad ($e = \lambda/4$). Por ejemplo, es habitual anchuras de $e \simeq 2{,}5\ \mu m$ para obtener una respuesta máxima en longitudes de onda en torno a $10\ \mu m$ (LWIR: $8 - 14\ \mu m$).
- **Pilares metálicos:** La membrana está conectada mediante unos puentes estrechos a unos pilares metálicos que hacen de contactos al ROIC.
- **ReadOut Integrated Circuit (ROIC):** La señal eléctrica producida por el microbolómetro es recogida y procesada mediante circuitos electrónicos que amplifican y filtran la señal para mejorar la relación señal/ruido y la resolución.

2.2. Métodos para la detección de rostros

Denominamos detección de rostros al proceso de identificar y extraer la región de caras, su posición y tamaño, mediante software a partir de muestras digitales, ya sean, imágenes estáticas o vídeos [15]. La detección de rostros es un subcampo específico de la detección de objetos, técnicas aplicadas en el campo de la visión artificial, o computer vision. Entre sus aplicaciones encontramos la seguridad, la vigilancia, el reconocimiento facial, la fotografía, etc.

El desarrollo de un algoritmo enfocado a detección de caras implica la creación de un modelo capaz de reconocer y detectar caras en nuevas imágenes. Este modelo englobará distintas técnicas según el tipo de aplicación y sus requisitos. Estos métodos siguen un enfoque común, un proceso previo donde se entrena el modelo y la posterior detección de rostros en nuevas imágenes.

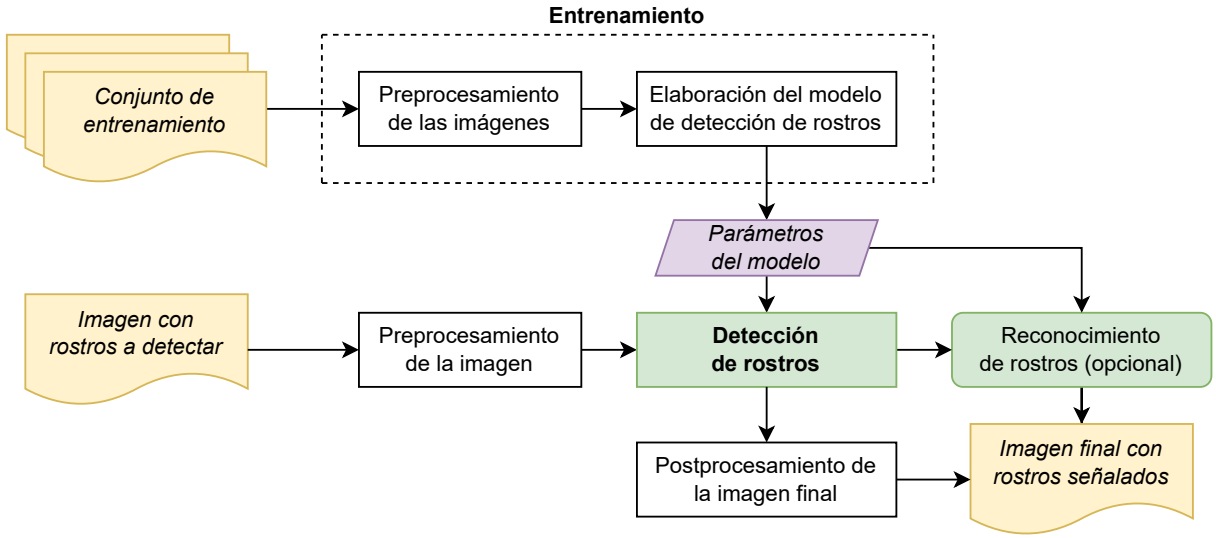


Figura 2.2.1: Diagrama de bloques para un algoritmo de detección de rostros

El entrenamiento del modelo implica seguir un procedimiento en el cual se ajusta y refina de forma iterativa a medida que se procesan más datos de un conjunto

proporcionado previamente. Es importante destacar que el proceso de entrenamiento requiere tiempo y recursos adecuados, ya que puede ser computacionalmente intensivo, especialmente en el caso de modelos más complejos, como en el caso de redes neuronales. A continuación, se describen los pasos más comunes en el proceso de entrenamiento:

1. **Conjunto de entrenamiento:** El primer paso es recopilar un conjunto de datos de entrenamiento que contenga imágenes que representen una variedad de caras y tomadas en las distintas condiciones de captura que contemple nuestro sistema de detección. A veces, según el tipo de técnica aplicada, también puede ser necesario aportar imágenes que contengan otros objetos en lugar de rostros. Es importante tener un conjunto de datos equilibrado y representativo para la obtención de resultados precisos.

2. **Preprocesamiento de las imágenes:** Tras proporcionar las imágenes para el entrenamiento, es común realizar ciertas operaciones para disminuir el coste computacional. Esto puede incluir correcciones en brillo y contraste, eliminación de ruido y la redimensión de la imagen a un tamaño adecuado para el algoritmo. También es habitual cambiar a escala de grises para reducir el número de variables.

3. **Elaboración del modelo:** Se selecciona y se elabora un modelo adecuado para la detección de caras. El modelo se diseña y entrena para aprender patrones y características distintivas que permitan identificar las caras en las imágenes. Cada técnica requerirá un proceso de entrenamiento distinto que llevará un tiempo diferente según la complejidad de los cálculos y la capacidad de cómputo del sistema.

4. **Obtención de parámetros:** El objetivo final del proceso de entrenamiento es el de optimizar y obtener los parámetros necesarios para la detección. Para ello, se realizará el entrenamiento, supervisado o no, del modelo. Se obtendrán los parámetros necesarios para realizar la detección de rostros que podrán variar según el tipo de técnicas contempladas: clasificadores de características, pesos, autovalores asociados, etc.

Como hemos indicado, los modelos de detección de rostros en imágenes siguen un proceso secuencial para que la detección sea precisa, efectiva y óptima. Todos los modelos deben tener en común varias etapas cruciales que se muestran a continuación:

1. **Captura de imagen:** El primer paso es adquirir y almacenar una imagen que contenga uno o varios rostros. Esto se puede lograr tomando la imagen mediante una cámara presente o cargando una imagen tomada previamente.

2. **Preprocesamiento de la imagen:** Una vez que se ha capturado la imagen, se realiza el mismo tipo de operaciones que hayan sido realizadas sobre las imágenes proporcionadas en el entrenamiento. Esto implica cambiar a la misma escala de color que el conjunto de entrenamiento y realizar las correcciones necesarias.

3. **Detección de rostros:** El siguiente paso es detectar las regiones de interés que contengan rostros en la imagen preprocesada. Para ello, se aplicarán técnicas refinadas en el entrenamiento del algoritmo de detección. El algoritmo proporcionará las coordenadas de las regiones espaciales de la imagen que

contienen caras. Según la técnica aplicada, estas regiones pueden ser los cuadros que contienen rostros o directamente la segmentación de los propios rostros.

4. **Reconocimiento** (opcional)**:** Una vez que se han detectado las regiones faciales en la imagen, se podría proceder a identificar a qué individuo pertenece cada rostro si el algoritmo permite hacerlo. Para que el reconocimiento sea posible, tendrá que haberse entrenado previamente al algoritmo con los individuos a identificar, incluyendo un correcto etiquetado de los rostros de cada individuo.

5. **Postprocesamiento de la imagen final:** Finalmente, es posible realizar las operaciones convenientes sobre la imagen proporcionada como entrada. Esto puede incluir la delimitación precisa de los rostros mediante la creación de los cuadros calculados en la detección, la eliminación de falsas detecciones o la aplicación de filtros adicionales para mejorar la calidad visual de la imagen. También es posible realizar una segmentación de las caras a posterior, si esta no venía incluida en el proceso de detección.

La detección de caras presenta una serie de factores limitantes que pueden dificultar su precisión y rendimiento. Cada una de estas dificultades plantea desafíos a tener en cuenta a la hora de diseñar el sistema termográfico y los algoritmos de detección, ya que deben ser capaces de reconocer rostros de manera precisa y óptima. Los factores más habituales que limitan la detección son los mostrados a continuación [5, 15]:

- **Distancia:** La distancia máxima para la cual el dispositivo puede medir la temperatura depende de tres factores: la óptica del dispositivo, la resolución de píxeles infrarrojos del dispositivo y el tamaño del objeto a medir. El flujo de radiación infrarroja emitida por un objeto disminuye cuadráticamente con la distancia, de modo que la aproximación realizada en la Ecuación 2.1.11 proporcionará mediciones de temperatura menos precisas conforme aumente la distancia al sujeto.

- **Orientación:** Dependiendo del tipo de aplicación, la orientación de los rostros puede variar en términos de inclinación. Las caras pueden estar en posición frontal, de perfil, vistas desde arriba o abajo y en ángulos intermedios. La detección de caras puede ser más desafiante cuando los rostros están en ángulos no frontales, ya que los rasgos faciales pueden aparecer deformados o parcialmente ocultos.

- **Iluminación:** La iluminación incorrecta o variable puede afectar la detección de caras. La iluminación intensa o las sombras fuertes pueden ocultar o distorsionar los rasgos faciales, dificultando la detección precisa. Además, las diferencias en la iluminación entre las imágenes proporcionadas para el entrenamiento y las imágenes de prueba pueden afectar a la eficiencia del algoritmo.

- **Resolución:** La baja resolución de una imagen puede afectar la detección de caras de varias maneras. La escala de los rostros en relación con la resolución puede generar problemas de distorsión, pérdida de detalles o superposición. Por otro lado, la localización precisa de los límites y contornos de un rostro puede ser más compleja. Sin embargo, una resolución muy alta requieren más recursos computacionales para procesar y analizar, lo que puede afectar al rendimiento de los algoritmos, especialmente en sistemas con limitaciones de potencia de cálculo.

En particular, las cámaras IR suelen tener resoluciones muy bajas en general y las que tienen mayor resolución suelen reflejarlo presentando un coste elevado, lo cual es algo muy a tener en cuenta a la hora de diseñar dispositivos experimentales para detección de caras.

- **Ruido óptico:** La presencia de ruido en la imagen puede confundir al sistema y hacer que interprete incorrectamente los rasgos faciales o incluso los identifique erróneamente. El ruido óptico puede ser causado por varios problemas del sistema, como aberraciones ópticas, desenfoque de la lente o problemas en la compresión de las imágenes que pueden generar bloqueo de colores o pérdida de detalles sutiles. Todas estas condiciones pueden causar imperfecciones en los rostros que compliquen al algoritmo detectarlos, lo que puede afectar la precisión de la detección.

- **Complejidad del fondo:** Si hay un fondo complejo o ruidoso en la imagen, puede haber una mayor posibilidad de falsas detecciones o una menor precisión en la detección de caras. Los algoritmos de detección deben poder distinguir claramente entre el rostro y el entorno circundante para obtener resultados precisos.

- **Obstrucción:** Cuando una parte del rostro está cubierta u obstruida, ya sea por objetos externos, como gafas, mascarillas o manos, o por otras partes del propio rostro, como el cabello, la detección de caras se vuelve más desafiante. Las oclusiones parciales pueden afectar la integridad de los rasgos faciales y hacer que la detección sea más difícil.

- **Variabilidad:** La variaciones en los rasgos de los rostros en términos de expresiones faciales, poses, edades y razas puede presentar desafíos adicionales en la detección de caras. Los algoritmos de detección deben ser capaces de adaptarse a esta diversidad para obtener resultados precisos y robustos.

  En el caso de imágenes en el rango IR hay que tener en cuenta que la variación de la emisividad con el tono de piel. En particular, la emisividad de la piel humana suele estar en el rango de $[0{,}96, 0{,}99]$, de modo que no afecta demasiado a los resultados finales [16].

- **Número de rostros:** La detección de múltiples rostros en una imagen puede ser complicada, especialmente cuando los rostros se superponen o están muy cerca unos de otros. La presencia de varios rostros puede requerir estrategias adicionales para la detección y el seguimiento adecuados.

Los métodos de detección de rostros pueden clasificarse en dos categorías principales: basados en imágenes y basados en características. A continuación, se describen las diferencias clave entre estos dos enfoques [15, 17, 18]:

- **Métodos basados en imágenes:** Analizan directamente las características visuales globales de la imagen completa, como el contraste, la textura, la forma y las relaciones espaciales. Son rápidos, eficientes y muchos de estos métodos tienen una adaptación favorable ante diferentes condiciones. Sin embargo, pueden ser más propensos a errores en casos de variaciones extremas de apariencia y obstrucciones parciales del rostro. Además, pueden tener dificultades para distinguir entre un rostro y otros objetos que comparten características visuales similares.

- **Métodos basados en características:** Se centran en extraer y analizar características específicas locales del rostro como formas, ya sean ojos, nariz, boca o accesorios, patrones locales o cambios de textura e iluminación. Ofrecen una mayor precisión en la detección y son más resistentes a variaciones en la apariencia de los rostros o a cambios externos. Sin embargo, algunos de estos métodos pueden ser más costosos computacionalmente o ser menos flexibles en algunas situaciones.

Tipo de técnica	**Basado en**
Redes neuronales	*Imágenes*
Subespacios lineales	
Enfoque estadístico	
Modelos de silueta activa (ASM)	*Características*
Análisis a bajo nivel	
Análisis de características	

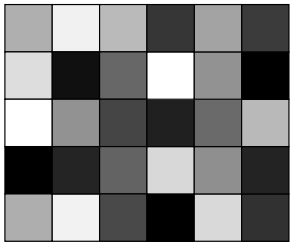
Tabla 2.1: Tipos de técnicas aplicables a detección de rostros [15, 17]

Aunque esta es la forma más sencilla a primera vista a la hora de distinguir entre las distintas técnicas de detección de objetos, hay que destacar que no son excluyentes entre si. De hecho, hay técnicas que pertenecen a varias de estas categorías y pueden ser en parte redundantes, pero se sigue realizando esta distinción al ser la que se ha seguido históricamente siguiendo el avance tecnológico en este sector.

Por otro lado, los modelos desarrollados no tienen que incorporar una única técnica. En algunos casos es conveniente combinar varias de estas para obtener una mejora notable en eficiencia o rendimiento. Veremos más adelante que este ha sido el caso en el desarrollo de nuestro dispositivo experimental.

• Espacio de características:

Como hemos visto, una vez realizado el preprocesamiento de la imagen, el procedimiento más habitual en imágenes en el rango visible es convertir la imagen ya normalizada en escala de grises para simplificar su estudio. En el caso de la radiación infrarroja, las imágenes a veces vienen dadas en un espacio de falso color, como se muestra por ejemplo en la Figura 2.1.1, pero hay que tener en cuenta que la información cuantitativa de cada píxel proporcionada por el sensor alude a una única intensidad. Bajo este contexto, la imagen preprocesada tendrá un tamaño fijo de $M \times N$ píxeles.



(a)

$$\begin{pmatrix} 175 & 241 & 186 & 54 & 163 & 59 \\ 221 & 16 & 103 & 255 & 146 & 0 \\ 255 & 146 & 69 & 32 & 106 & 185 \\ 0 & 36 & 99 & 216 & 143 & 35 \\ 174 & 242 & 73 & 0 & 217 & 49 \end{pmatrix}$$

(b)

Figura 2.2.2: Ejemplo de imagen arbitraria en escala de grises: (a) Imagen de 5×6 píxeles (b) Matriz asociada de intensidades de gris

Cada píxel tendrá un valor de intensidad que podemos agrupar en una matriz de $M \times N$ elementos, de modo que cada elemento I_{ij} representa la intensidad de gris asociada a cada píxel (i, j) de la imagen. Igualmente, podemos tomar todas las intensidades y agruparlas siguiendo un orden fijo, por ejemplo, tomando todos los elementos en fila tras fila, colocándolos en forma de vector o matriz columna, asociando a cada intensidad una coordenada: $\boldsymbol{x}^T = (x_1, ..., x_d)$. De esta forma, la imagen pasaría a representar un punto en un espacio de $d = M \cdot N$ dimensiones, al que denominamos *espacio de características*, o espacio de imágenes.

Para imágenes visibles, vemos que si en el preprocesamiento no se convirtiera la imagen a escala de grises, cada píxel tendría un valor asociado para las k variables, generalmente $k = 3$, por ejemplo: RGB, HSV, $YCbCr$, etc. Por tanto, el espacio de imágenes pasaría a ser de dimensión $d = k \cdot M \cdot N$ y el coste computacional sería mayor. Vemos entonces que la escala de grises es un caso particular donde se tiene que $k = 1$, el valor más sencillo.

En el caso de las imágenes en el espectro infrarrojo, la imagen digital resultante en el proceso de captura se presenta en escala de grises, salvo que se utilice un espacio de falso color para mejorar la presentación y facilitar su entendimiento. La intensidad de gris en cada píxel representa el nivel de radiación infrarroja captada por ese píxel en particular.

De ahora en adelante, nos centraremos en explorar las distintas técnicas de detección de caras, tanto en el espectro visible como en el infrarrojo, pues aunque el objetivo final es la detección en el infrarrojo, veremos que tener en cuenta ambos espectros podrá facilitarnos el proceso. Una vez explorados los distintos modelos, nos centraremos en idealizar y diseñar el dispositivo experimental que nos permita medir la temperatura facial de forma eficiente, rápida y óptima.

2.2.1. Redes neuronales

Dado su alto porcentaje de aciertos, las redes neuronales se han convertido en uno de los métodos más aplicados para realizar una amplia variedad de tareas como la clasificación de objetos, la detección de patrones, el procesamiento del lenguaje natural, la predicción de resultados, entre otras.

Las redes neuronales [19] consisten en modelos matemáticos inspirados por la estructura y el funcionamiento del cerebro humano, compuesta por un conjunto de nodos interconectados llamados *neuronas artificiales* o simplemente *neuronas*, que procesan la información y la transfieren a otras neuronas a través de conexiones llamadas *sinapsis artificiales*.

El funcionamiento de una red neuronal se basa en el procesamiento de información mediante el aprendizaje y la adaptación, cuyo objetivo en este caso es el de buscar patrones visuales en las imágenes. La red recibe una entrada de datos, que se procesa a través de las diferentes capas de neuronas. Cada capa extrae características cada vez más complejas de los datos y las utiliza para hacer predicciones o tomar decisiones.

Durante el proceso de entrenamiento, la red ajusta los pesos de las conexiones entre las neuronas para optimizar el rendimiento y mejorar la precisión de las predicciones. Esto se logra mediante el uso de algoritmos de aprendizaje supervisado o no supervisado, que permiten a la red adaptarse a diferentes situaciones y aprender a partir de los datos.

En general, el proceso de entrenamiento de una red neuronal consiste en [19, 20]:

1. **Inicialización de los pesos:** Al inicio del entrenamiento, los pesos de las conexiones se inicializan con valores pequeños seleccionados de forma arbitraria, $\{\omega_{ij}\}$.

2. **Propagación:** La entrada se propaga a través de la red neuronal, calculando la salida de cada neurona hasta llegar a la capa de salida.

3. **Cálculo del error:** Se compara la salida de la red neuronal con la salida esperada para el dato de entrada, y se calcula el error de predicción utilizando una función de pérdida como el error cuadrático medio.

$$\epsilon = \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{y_i} - \boldsymbol{\hat{y}_i}\|^2 \tag{2.2.1}$$

   donde y_i es la salida esperada y $\hat{y}_i$ es la salida obtenida por la red neuronal.

4. **Retropropagación:** Se utiliza un algoritmo para calcular la contribución de cada neurona al error total de la red y actualizar los pesos de las conexiones en la dirección opuesta al gradiente de la función de pérdida.

5. **Actualización de los pesos:** Se incrementa el valor de los pesos de las conexiones de acuerdo a la siguiente regla:

$$\omega'_{ij} = \omega_{ij} + \Delta\omega_{ij} = \omega_{ij} - \alpha\frac{\partial\epsilon}{\partial w_{ij}} \tag{2.2.2}$$

   donde ω'_{ij} es el nuevo valor para el peso ω_{ij}, α es la tasa de aprendizaje y $\partial\epsilon/\partial\omega_{ij}$ es el gradiente de la función de pérdida con respecto al peso ω_{ij}.

6. **Reiteración:** Se repiten los pasos 2 a 5 para todos los datos de entrenamiento en una nueva iteración. Se sigue reiterando hasta que el error de predicción se reduzca a un valor aceptable o hasta que se alcance un número máximo establecido de iteraciones.

En el caso de aplicar redes neuronales a la detección de rostros hay que aportar para su entrenamiento, por un lado, un conjunto de imágenes con caras variadas, con distintos rasgos faciales, algunas con accesorios como gafas, pendientes, mascarillas o bufandas y otras sin ellas, capturadas desde distintos ángulos, y otro conjunto de imágenes sin caras presentes. La respuesta o salida de la red neuronal ante una imagen o entrada es la de determinar si existe o no una cara presente en la imagen, es decir, una respuesta binaria.

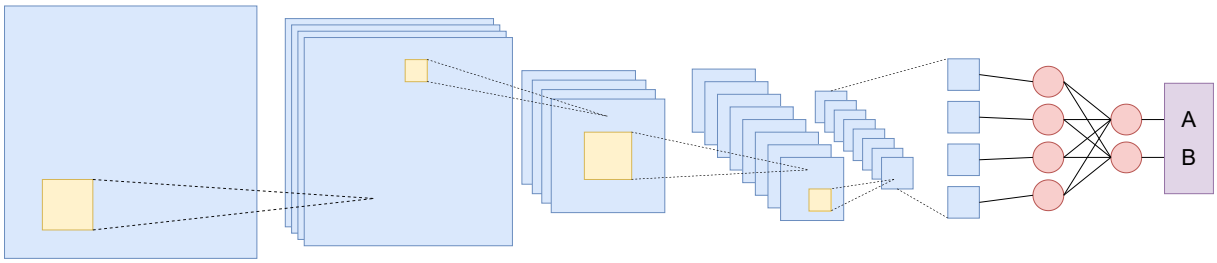


Figura 2.2.3: Ejemplo esquemático de CNN: Capas de convolución y pooling, capas de clasificación y distribución probabilística

En particular, hay muchos tipos de arquitecturas de redes neuronales pero en la actualidad, los usados habitualmente en detección de objetos son las Redes Neuronales

Convolucionales (Convolutional Neural Network, CNN). Estas utilizan capas convolucionales para extraer características locales de las imágenes en combinación con capas de pooling para reducir la dimensionalidad y preservar las características más importantes [20, 21]. Estas características extraídas se alimentan a posteriori a capas completamente conectadas para realizar la clasificación.

La convolución se realiza mediante el uso de filtros o kernels que se deslizan sobre la imagen de entrada y realizan multiplicaciones y sumas ponderadas. Los filtros son matrices de pesos que se utilizan para realizar la convolución. Estos filtros son aprendidos durante el entrenamiento de la red y se utilizan para extraer características específicas de la imagen, como bordes, texturas o formas.

La operación básica realizada por una capa de pooling es agrupar localmente las activaciones de las capas convolucionales y calcular una estadística resumida, como el valor máximo o max pooling, o el promedio o average pooling, en cada región de píxeles agrupada. El propósito principal de las capas de pooling es lograr invariancia ante pequeñas variaciones en la posición de las características dentro de una región. Al aplicar pooling, se reduce la resolución espacial de las características, lo que disminuye la cantidad de parámetros y computaciones necesarias en la red y ayuda a controlar el sobreajuste, u overfitting. Además, las capas de pooling permiten capturar características más generales y robustas, ya que se enfocan en la presencia o ausencia de características en lugar de su ubicación exacta.

2.2.2. Subespacios lineales

El modelo de subespacios lineales en el contexto de la detección de rostros se basa en la idea de que gran parte de los puntos en el espacio de imágenes no representarán caras físicamente posibles. Por lo tanto, debido a estas restricciones, tenemos que las imágenes de las caras estarán contenidas en un subespacio del espacio de imágenes, al que denominamos *subespacio de caras*.

Debido a la naturaleza matemática de estas técnicas, algunas de estas pueden englobarse también como métodos que siguen un enfoque estadístico. En particular, vamos a estudiar los subespacios resultantes de aplicar PCA (*Eigenfaces*), LDA (*Fisherfaces*), ICA y LPP, observando las características y particularidades de cada uno y las diferencias entre estos métodos [22, 23, 34–36].

• Análisis de componentes principales (PCA):

El método de Análisis de Componentes Principales (Principal Components Analysis, PCA) es una técnica planteada por primera vez en 1901 por K. Pearson [24] y utilizada típicamente en el campo de la estadística y el análisis de datos para reducir la dimensionalidad de un conjunto de datos, manteniendo la información más relevante.

Si tenemos un conjunto de partida con m datos de estudio $\{\boldsymbol{x_i}\}_{i=1,\dots,m}$ en un espacio de características de dimensión d, este método tiene como objetivo encontrar un subespacio de dimensión $s < d$ que los describa con la menor pérdida de información posible. Si nos centramos en el ejemplo de dos dimensiones ($d = 2$), buscamos encontrar la recta que recoja la máxima variación de los datos de estudio, es decir, aquella que minimice las distancias entre los datos de partida y sus proyecciones en la recta.

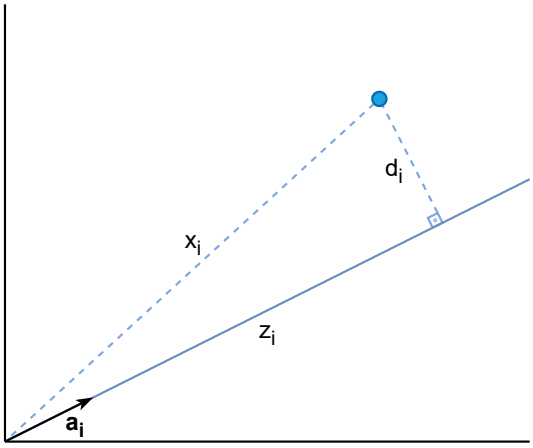


Figura 2.2.4: Proyección de un punto sobre el subespacio de PCA.

Aplicando el teorema de Pitágoras, se cumple que la relación entre coordenadas es:

$$x_i^2 = z_i^2 + d_i^2 \tag{2.2.3}$$

Por tanto, el problema se basará en encontrar el vector unitario $\boldsymbol{a_i}$ que caracterice a la recta que minimiza la distancia de los puntos al subespacio:

$$min \sum_{i=1}^{m} d_i^2 = \sum_{i=1}^{m} \|\boldsymbol{x_i} - z_i \boldsymbol{a_i}\|^2 \tag{2.2.4}$$

donde $\boldsymbol{x_i}$ es el vector asociado a cada dato en el espacio de características, $z_i \boldsymbol{a_i}$ es el vector que apunta a la proyección de $\boldsymbol{x_i}$ en el nuevo subespacio y d_i es la distancia entre un punto y su proyección. Como la distancia $|\boldsymbol{x_i}|$ es un valor fijo dado en el problema, minimizar la distancia d_i es equivalente a maximizar el parámetro z_i, es decir, maximizar la coordenada de la proyección en el subespacio resultante.

La covarianza entre dos variables discretas X e Y se define como:

$$Var(X, Y) = \frac{1}{m} \sum_{i=1}^{m} (X_i - \bar{X})(Y_i - \bar{Y}) \tag{2.2.5}$$

donde $\bar{X}$ representa el valor medio de X y $\bar{Y}$ el valor medio de Y. En el caso de $X = Y$, $V(X, X)$ se denomina la varianza de la variable X. La covarianza es una medida estadística que describe la relación entre dos variables, indicando cómo varían conjuntamente. Un valor positivo de covarianza indica que las variables tienden a cambiar en la misma dirección, mientras que un valor negativo indica que tienden a cambiar en direcciones opuestas.

Podemos aplicar este concepto al espacio de imágenes y buscar aquellas variables, asociadas a los píxeles, que presenten mayor relación. Esto es lo mismo que resolver el siguiente problema de autovalores:

$$C\boldsymbol{a_i} = \lambda_i \boldsymbol{a_i} \tag{2.2.6}$$

donde C es la matriz de varianzas y covarianzas de los puntos de estudio $\{\boldsymbol{x_i}\}_{i=1,\dots,m}$, λ_i los autovalores del problema y $\boldsymbol{a_i}$ los autovectores normalizados. A los vectores $\boldsymbol{a_i}$ se les denomina *Componentes Principales* de la matriz de covarianza C.

En el ejemplo bidimensional ($d = 2$), tras resolver el problema de autovalores obtendríamos dos autovectores asociados a dos autovalores diferentes. Como buscamos la recta que presente la menor pérdida de información, nos quedaremos con el subespacio cuyo autovalor sea más grande, descartando el otro ya que sería el más próximo a cero que será menos relevante. En el caso general, haremos esto de forma generalizada, quedándonos con los autovalores de mayor valor.

• **Análisis discriminante lineal (LDA):**

El método de Análisis Discriminante Lineal (Linear Discriminant Analysis, LDA) es una generalización del método de Discriminante Lineal de Fisher (Fisher's Linear Discriminant, FLD), el cual se planteó en 1936 por R. A. Fisher [25]. Esta técnica además es muy útil cuando se trabaja con conjuntos de datos con múltiples clases ya que, a diferencia de PCA, el objetivo de esta técnica es lograr una mejor discriminación y clasificación de los datos, maximizando la separabilidad entre clases.

En este contexto, el término clase se refiere a las categorías o grupos a los que pertenecen los datos de entrada. En particular, en detección de caras las clases serán los distintos individuos de entrenamiento de los que se disponen imágenes. Este método nos permitirá tomar distintas imágenes de cada clase, con distinta iluminación y expresiones faciales, posibilitando diferenciarlas entre si, algo que PCA no nos permite hacer.

Este método es similar al de PCA, sólo que en vez de maximizar una única matriz de covarianza, definiremos por separado una matriz que caracterice la covarianza dentro de una clase y otra que caracterice la covarianza entre clases. Nuestro objetivo será el de maximizar la matriz de covarianza entre clases a la vez que buscamos minimizar la matriz de covarianza dentro de clases, es decir, obtener el subespacio que mejor discrimine a las clases [22].

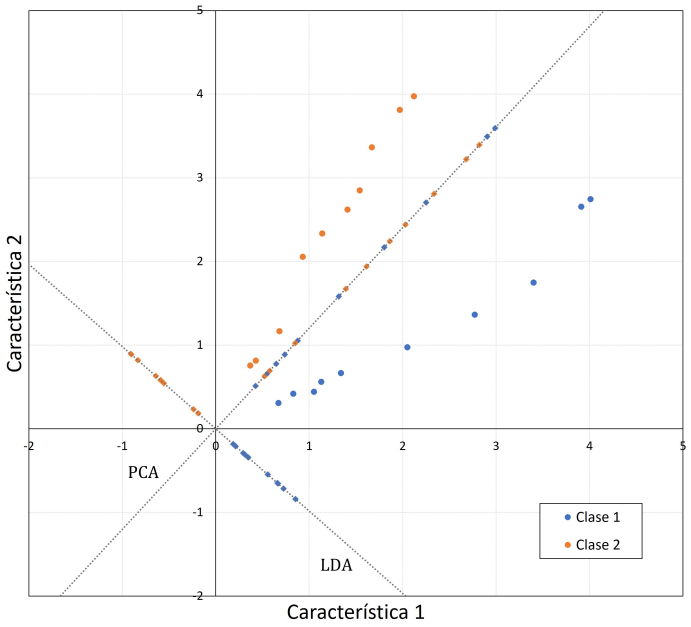


Figura 2.2.5: Comparación entre el método de PCA y el de LDA: Ejemplo particular en el que se clasifican dos clases ($c = 2$) con diez muestras cada una ($n_j = 10$, con $j = 1, 2$) en un espacio de características de dos dimensiones ($d = 2$).

La matriz de varianzas y covarianzas estudiada en PCA viene dada por:

$$C = S_b + S_w \tag{2.2.7}$$

donde S_b representa la matriz de covarianza entre clases, S_w la matriz de covarianza dentro de clases.

En este caso, el problema de autovalores equivalente sería:

$$S_b \boldsymbol{a_i} = \lambda_i S_w \boldsymbol{a_i} \tag{2.2.8}$$

donde hay que destacar que S_w debe ser una matriz no singular para que exista su inversa.

• Análisis de componentes independientes (ICA):

El método de Análisis de Componentes Independientes (Independent Component Analysis, ICA) fue formulado por primera vez por J. Herault y C. Jutten en 1986 [26] en un intento de resolver el problema de Separación Ciega de Fuentes (Blind Source Separation, BSS).

El problema de Separación Ciega de Fuentes (BSS) consiste en descomponer una mezcla de señales desconocidas en sus fuentes originales sin ninguna información previa sobre las señales o el proceso de mezcla [27]. El desafío radica en identificar y separar las señales individuales sin información explícita sobre ellas o el proceso de mezcla, lo que requiere técnicas avanzadas de procesamiento de señales y análisis estadístico.

Por ejemplo, tenemos s fuentes sonoras y d micrófonos. Conociendo las señales $\{x_i\}_{i=1,\dots,d}$ medidas por los micrófonos, el problema residiría en obtener las señales originales $\{r_j\}_{j=1,\dots,s}$ provenientes de las fuentes. Hay que tener en cuenta que en este caso cada señal x_i representa una componente en el espacio de características de d dimensiones.

Queremos construir x_i como suma de las s componentes independientes r_j:

$$x_i = \sum_{j=1}^{s} a_{ij} r_j \quad \Longrightarrow \quad \boldsymbol{x} = A\boldsymbol{r} \tag{2.2.9}$$

donde x_i son las d señales conocidas y A es la matriz de mezcla, pues en el problema citado sería la matriz de coeficientes que suma todas las señales r_j provenientes de las distintas fuentes en cada sensor.

El problema de separación se traduce en encontrar la transformación lineal W (matriz de t que aplicada sobre un vector del espacio de características nos devuelva el vector del subespacio de componentes independientes [23]:

$$r_j = \sum_{i=1}^{m} w_{ij} x_i \quad \Longrightarrow \quad \boldsymbol{r} = W\boldsymbol{x} \tag{2.2.10}$$

Para encontrar la matriz de separación W existen múltiples algoritmos [28]: *FastICA*, *JADE* (Joint Approximate Diagonalization of Eigenmatrices), *Infomax*, *SOBI* (Second Order Blind Identification), etc.

• **Proyecciones que preservan la localidad (LPP):**

El método de Proyecciones que Preservan la Localidad (Locality Preserving Projections, LPP) fue planteada en 2003 por X. He y P. Niyogi [29] como una alternativa a los métodos estudiados previamente, PCA y LDA. El principal objetivo de esta técnica es preservar la estructura local de los datos en un subespacio de menor dimensión.

Cuando tenemos conjuntos de datos de alta dimensionalidad, es común que los datos exhiban una estructura local o una relación entre los vecinos más próximos. Esto significa que los puntos de datos similares tienden a agruparse en regiones cercanas del espacio. Esta técnica aprovecha esta propiedad para encontrar una representación de menor dimensión que preserve la relación de vecindad entre las muestras [22].

El primer paso de este algoritmo es construir un grafo de adyacencia G con m nodos, siendo m el número de datos de disponibles de partida. Este grafo se construirá de manera que dos nodos i y j estén conectados si se verifica que $\boldsymbol{x_i}$ y $\boldsymbol{x_j}$ están lo suficientemente próximos en el espacio de características. Podemos establecer distintos tipos de condiciones para establecer que se da la proximidad, como las siguientes [22]:

1. ***k*-vecinos más cercanos:** Los nodos i y j están conectados si j está entre los k-vecinos más cercanos de i, o viceversa.

2. **Vecinos próximos en una distancia $\boldsymbol{\delta}$:** Los nodos i y j están conectados si la distancia euclídea es inferior a una cota previamente establecida:

$$d_{ij} = \|\boldsymbol{x_i} - \boldsymbol{x_j}\| < \delta \tag{2.2.11}$$

El siguiente paso será el de construir una matriz simétrica de pesos W, de modo que cada elemento de matriz W_{ij} es un peso asociado a la relación entre los nodos i y j, siendo 0 si no existe unión entre esos nodos en el grafo G. Para establecer el valor de los pesos de los nodos que sí están conectados utilizaremos una función kernel.

Existen distintas funciones kernel que se suelen usar para este contexto [22]:

1. **Función de base radial Gaussiana:** $W_{ij} = exp\left(\frac{-\|\boldsymbol{x_i}-\boldsymbol{x_j}\|^2}{2\sigma^2}\right)$

2. **Función unidad:** $W_{ij} = 1$

Por último, resolvemos el siguiente problema de autovalores:

$$XLX^T\boldsymbol{a_i} = \lambda_i XDX^T\boldsymbol{a_i} \tag{2.2.12}$$

donde $L = D - W$ es la matriz laplaciana y D es una matriz diagonal cuyos elementos son $D_{ii} = \sum_j w_{ji}$. Al igual que pasaba en LDA, se asume que XDX^T es una matriz no singular, es decir, una matriz invertible.

2.2.3. Enfoque estadístico

Este modelo parte de la idea de que toda imagen que presente similitudes con lo que sabemos que es una cara será probablemente otra cara. Por tanto, buscamos algoritmos

que utilicen métodos estadísticos para hallar similitudes con los rostros del conjunto de entrenamiento establecido de partida, centrándonos en aquellos métodos que reduzcan la cantidad de variables implicadas en el problema de detección de caras.

• Máquinas de vectores de soporte (SVM):

La técnica de Máquinas de Vectores de Soporte (Support Vector Machines, SVM) fue propuesta originalmente en 1964 por V. N. Vapnik y A. Y. Chervonenkis [30]. Posteriormente, en 1992 V. N. Vapnik, I. Guyon y B. Boser sugirieron una forma de crear clasificadores no lineales utilizando funciones kernel [31]. Finalmente, C. Cortes y V. N. Vapnik introdujeron en 1995 la versión de "margen suave" (soft margin) que se utiliza más habitualmente en algoritmos de clasificación [32].

Este método, a diferencia de los últimos mostrados en esta sección, no busca reducir la dimensionalidad de los datos para simplificar su estudio, sino que su objetivo es encontrar un hiperplano óptimo que pueda separar las diferentes clases de datos en el espacio de características [33]. Este hiperplano se selecciona de tal manera que maximiza el margen entre las clases, es decir, la distancia entre el hiperplano y los puntos de datos más cercanos de cada clase. Para ello se utilizan vectores de soporte cuyo módulo es la distancia entre los puntos más cercanos y el hiperplano buscado.

Consideremos un hiperplano en el espacio de características d-dimensional. Dado cualquier punto $\boldsymbol{x}$ del hiperplano, donde $\boldsymbol{x}^T = (x_1, ..., x_d)$, se verifica que:

$$H: \quad \boldsymbol{w}^T\boldsymbol{x} + b = 0 \tag{2.2.13}$$

donde $\boldsymbol{w}$ es un vector de pesos normal al hiperplano y b es un término de sesgo o desplazamiento.

Queremos encontrar los valores óptimos para $\boldsymbol{w}$ y b que permitan la separación más amplia entre las clases de datos. La separación se logra maximizando el margen entre el hiperplano de separación máximo y los puntos de datos más cercanos de ambas clases. De modo que para un dato x_i, sabremos a qué clase pertenece por el valor de $\boldsymbol{w}^T\boldsymbol{x_i} + b$.

Para formalizar esto, introducimos la etiqueta de clase que nos caracterizará a cual de las clases pertenece un dato $\boldsymbol{x_i}$ dado. Tenemos que $y_i = +1$ si $\boldsymbol{x_i}$ pertenece a la clase positiva y $y_i = -1$ si $\boldsymbol{x_i}$ pertenece a la clase negativa. Para definir el margen, necesitamos introducir una serie de restricciones:

- Los puntos de datos $\boldsymbol{x_i}$ más cercanos al hiperplano deben satisfacer la ecuación $\boldsymbol{w}^T\boldsymbol{x_i} + b = \pm 1$. Los hiperplanos paralelos al hiperplano original que contienen a estos puntos se denominan *hiperplanos de apoyo*:

$$H_+: \quad \boldsymbol{w}^T\boldsymbol{x} + b = +1 \tag{2.2.14}$$
$$H_-: \quad \boldsymbol{w}^T\boldsymbol{x} + b = -1 \tag{2.2.15}$$

- Los vectores que unen el hiperplano original a estos puntos más cercanos se denominan *vectores de soporte*. La distancia entre los hiperplanos de apoyo será lo que definimos como margen:

$$d_+ = \frac{|\boldsymbol{w}^T\boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|} = \frac{|+1|}{\|\boldsymbol{w}\|} \qquad d_+ + d_- = \frac{2}{\|\boldsymbol{w}\|} \tag{2.2.16}$$
$$d_- = \frac{|\boldsymbol{w}^T\boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|} = \frac{|-1|}{\|\boldsymbol{w}\|}$$

donde $\|\boldsymbol{w}\|$ es la norma euclídea de $\boldsymbol{w}$.

Generalizando estas restricciones al resto de puntos de cada clase se cumple que:

$$\begin{aligned} y_i = +1 \quad &\rightarrow \quad \boldsymbol{w}^T\boldsymbol{x_i} + b \geq +1 \\ y_i = -1 \quad &\rightarrow \quad \boldsymbol{w}^T\boldsymbol{x_i} + b \leq -1 \end{aligned} \qquad y_i \cdot (\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq +1 \tag{2.2.17}$$

Como se indicó al principio podemos distinguir entre dos versiones de SVM:

1. **Margen fuerte:** Buscamos el hiperplano óptimo que maximiza el margen sin permitir excepción alguna sobre ningún punto. Todos los datos del conjunto de entrenamiento deben quedar aislados correctamente en el lado del hiperplano que le correspondan según a qué clase pertenezcan [30, 33].

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|^2 \tag{2.2.18}$$

$$\text{sujeto a} \quad y_i \left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1, \quad \forall i \in \{1, \ldots, m\} \tag{2.2.19}$$

2. **Margen suave:** Ahora se introduce cierta flexibilidad en la clasificación permitiendo que algunos datos del conjunto de entrenamiento se encuentren dentro del margen o incluso en el lado incorrecto del hiperplano. Esta flexibilidad se consigue mediante la introducción de *variables de holgura* ζ_i, o slack variables, en el margen [32, 33].

$$\min_{\boldsymbol{w},b,\zeta} \quad \|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m} \zeta_i \tag{2.2.20}$$

$$\text{sujeto a} \quad y_i \left(\boldsymbol{w}^T\boldsymbol{x_i} + b\right) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \quad \forall i \in \{1, \ldots, m\} \tag{2.2.21}$$

Este problema se puede resolver utilizando técnicas de optimización convexa, como el método de multiplicadores de Lagrange. En caso de que los datos no sean linealmente separables se aplica una función kernel $K(\boldsymbol{x_i}, \boldsymbol{x})$ que transforma los datos de manera que sí lo sean [31].

2.2.4. Modelos de silueta activa

Los modelos de silueta activa, también conocidos como modelos de contorno activo, son una técnica ampliamente utilizada en el campo de la visión artificial y el procesamiento de imágenes para la segmentación y extracción de contornos precisos de objetos en imágenes. A través de su capacidad para adaptarse a las características locales y globales de la imagen, los modelos de silueta activa se han convertido en una herramienta ampliamente utilizada en detección de objetos.

• Modelo de Distribución de Puntos (PDM):

El Modelo de Distribución de Puntos (Point Distribution Model, PDM) es una técnica utilizada en visión artificial para el análisis de formas y la segmentación de figuras geométricas en imágenes. Esta técnica fue presentada en 1995 por T. F. Cootes, C. J. Taylor, D. H. Cooper y J. Graham [37].

El objetivo de esta técnica es construir un modelo estadístico de la variabilidad de la forma de un objeto y se utiliza reposicionar esos puntos clave en nuevas imágenes. En

el caso de detección de rostros, el modelo presenta la apariencia global de una cara, que incluye todas las características faciales como las cejas, la nariz y los ojos.

Para este método partimos de k puntos de referencias (x_i, y_i) colocados manualmente, de modo que engloben de manera suficiente aproximada la forma de un rostro. Estos puntos se alinean utilizando métodos estadísticos que minimicen el error mínimo cuadrado entre puntos, como el análisis generalizado de Proscrutes (GPA).

Podemos presentar todos estos puntos de referencia en un vector que los engloble:

$$\boldsymbol{x} = (x_1, y_1, \ldots, x_k, y_k) \in \mathbb{R}^{2k} \tag{2.2.22}$$

Asumiendo que la dispersión es Gaussiana, se utiliza el Análisis de Componentes Principales (PCA) para calcular los autovectores y autovalores normalizados de la matriz de covarianza de todas las formas dadas para el entrenamiento del modelo. Cada autovector describe un modo principal de variación a lo largo del conjunto.

Podemos definir una nueva forma como suma de la forma promedio $\bar{\boldsymbol{x}}$ y una combinación lineal de autovectores:

$$\boldsymbol{x'} = \bar{\boldsymbol{x}} + P\boldsymbol{b} \tag{2.2.23}$$

donde $P \in \mathbb{R}^{2k \times s}$ es la matriz de los s mejores autovectores agrupados en columnas y $\boldsymbol{b}$ es un vector de los s pesos asociados a cada autovector.

Por lo tanto, al modificar la variable $\boldsymbol{b}$ se pueden definir un número infinito de formas. Para asegurarse de que las nuevas formas se encuentren dentro de las variaciones observadas en el conjunto de entrenamiento, es común permitir que cada elemento de $\boldsymbol{b}$ esté dentro de ± 3 desviaciones estándar, donde la desviación estándar de un componente principal dado se define como la raíz cuadrada de su correspondiente autovalor.

2.2.5. Análisis a bajo nivel

Los métodos de análisis de bajo nivel para la detección de caras se enfocan en características específicas de la imagen para identificar regiones que puedan contener caras. Estos tipos de métodos buscan realizar la segmentación de características visuales utilizando propiedades específicas de los píxeles, como el color o la escala de grises, entre otros [17]. Sin embargo, debido a la naturaleza de bajo nivel, las características generadas pueden resultar ambiguas en su interpretación. De modo que por si solos estos métodos de análisis no son muy eficientes en la detección.

• Algoritmo de Otsu:

En el caso de imágenes en el espectro infrarrojo, como se ha estudiado, los píxeles tienen una única intensidad asociada, de modo que aunque podemos visualizar la imagen utilizando cualquier gradiente de color, para facilitar su comprensión, esto será equivalente a estudiar una imagen en escala de grises. En la Figura 2.2.6 observamos dos ejemplos de gradientes utilizados en imágenes térmicas.

El thresholding de Otsu [38] es un método automático utilizado para realizar la segmentación de imágenes en dos clases diferentes: frente (*foreground*) y fondo (*background*). El objetivo de este algoritmo es el de clasificar los píxeles de una imagen en estas dos clases buscando el valor umbral (*threshold*) que los separa. En el caso

particular de imágenes infrarrojas, buscamos segmentar los rostros puesto que emitirán mayor cantidad de radiación infrarroja que el fondo [39].

(a) (b)

Figura 2.2.6: Gradientes de color usados en IR: (a) Falso color (b) Escala de grises

Dado el número total de píxeles en la imagen, podemos definir la probabilidad de encontrar un pixel con intensidad de grises i (donde $i \in [0, L-1]$) como:

$$p_i = n_i/d \quad (n_i \geq 0) \; ; \qquad \sum_{i=0}^{L-1} p_i = 1 \; ; \quad d = M \cdot N = \sum_{i=0}^{L-1} n_i \tag{2.2.24}$$

donde n_i es el número de píxeles con intensidad de grises i y d es el número total de píxeles en la imagen.

Definimos la probabilidad de cada clase como la probabilidad acumulada de la intensidad de grises hasta el valor umbral k buscado:

$$\begin{aligned} w_0 &= Pr(C_0) = \sum_{i=0}^{k} p_i = w(k) \\ w_1 &= Pr(C_1) = \sum_{i=k+1}^{L-1} p_i = 1 - w(k) \end{aligned} \qquad\qquad w(k) = \sum_{i=0}^{k} p_i \tag{2.2.25}$$

Igualmente, podemos calcular las medias de cada clase de la siguiente forma:

$$\begin{aligned} \mu_0 &= i \cdot Pr(i|C_0) = \sum_{i=0}^{k} i p_i / w_0 = \mu(k)/w(k) \qquad & \mu(k) &= \sum_{i=0}^{k} i \cdot p_i \\ \mu_1 &= i \cdot Pr(i|C_1) = \sum_{i=k+1}^{L-1} i p_i / w_1 = \frac{\mu_T - \mu(k)}{1 - w(k)} \qquad & \mu_T &= \sum_{i=0}^{L-1} i \cdot p_i \end{aligned} \tag{2.2.26}$$

De todo lo anterior, se deduce que se cumplen las siguientes igualdades:

$$w_0\mu_0 + w_1\mu_1 = \mu_T \qquad\qquad w_0 + w_1 = 1 \tag{2.2.27}$$

Las varianzas de las clases vienen dadas por:

$$\begin{aligned} \sigma_0^2 &= \sum_{i=0}^{k} (i - \mu_0)^2 Pr(i|C_0) = \sum_{i=0}^{k} (i - \mu_0)^2 p_i / w_0 \\ \sigma_1^2 &= \sum_{i=k+1}^{L-1} (i - \mu_1)^2 Pr(i|C_1) = \sum_{i=k+1}^{L-1} (i - \mu_1)^2 p_i / w_1 \end{aligned} \tag{2.2.28}$$

El método de Otsu es un caso particular del discriminante lineal de Fisher (FLD) en una dimensión, de modo que sólo tenemos que encontrar la intensidad que minimiza σ_w^2,

o lo que es lo mismo, que maximiza σ_b^2.

$$\begin{aligned} \sigma_w^2 &= w_0\sigma_0^2 + w_1\sigma_1^2 \\ \sigma_b^2 &= w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2 = w_0 w_1(\mu_1 - \mu_0)^2 \end{aligned} \tag{2.2.29}$$

Finalmente, el objetivo de este algoritmo es evaluar la varianza entre clases para todos los valores posibles de intensidad de grises y quedarnos con el valor umbral que la maximice, que será el que marque el límite entre frente y fondo:

$$\sigma_b^2 = \frac{(\mu_T w(k) - \mu(k))^2}{w(k)(1 - w(k))} \quad \rightarrow \quad \boxed{t \ : \ \sigma_b^2(t) = \max_{0 \leq k \leq L-1} \sigma_b^2(k)} \tag{2.2.30}$$

2.2.6. Análisis de características

El análisis de características se basa en la idea de que ciertas propiedades locales o patrones específicos son comunes en las regiones faciales. Estos patrones pueden ser capturados y utilizados para diferenciar entre caras y otras partes de la imagen. También es posible distinguir y capturar accesorios locales como gafas o mascarillas. A diferencia de otras técnicas que se centran en el análisis de píxeles, el enfoque de análisis de características se basa en la detección de elementos específicos que componen un rostro humano, como los ojos, la nariz, la boca y las orejas.

• Patrones Binarios Locales (LBP):

Los patrones binarios locales (Local Binary Patterns, LBP) son un tipo de operador de textura propuesto originalmente por T. Ojala, M. Pietikainen y D. Harwood en 1994 [40, 41]. La idea básica de un operador LBP es que las texturas de superficie bidimensionales pueden describirse mediante dos medidas complementarias: patrones espaciales locales y contraste de escala de grises.

El operador LBP original [40, 41] forma etiquetas para los píxeles de la imagen relacionando cada píxel con su entorno de 3×3 píxeles alrededor. Para ello, se calcula la diferencia de intensidades de estos píxeles del entorno con la del píxel central. Después, se les asocia el valor 0 o 1 según si la diferencia es negativa o positiva (o cero) respectivamente. Finalmente, se dispone el resultado como un número binario, multiplicando en orden por potencias de 2 y sumando todo (véase Figura 2.2.7).

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s\left(i_p - i_c\right) 2^p \qquad \text{donde} \quad s(x) = \begin{cases} 1, \text{ si } x \geq 0 \\ 0, \text{ en otro caso} \end{cases} \tag{2.2.31}$$

Sin embargo, el operador LBP se extendió [42] para calcularse sobre entornos de distintos tamaños (en la Ecuación 2.2.31, P indica el número de puntos y R el radio del círculo de medición, donde el LBP original sería $P = 8$ y $R = 1,0$).

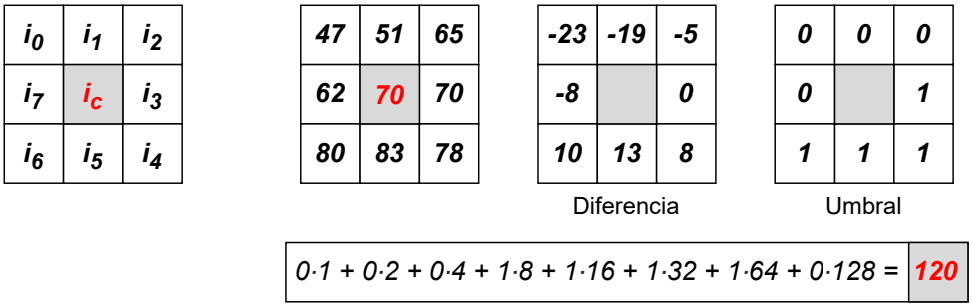


Figura 2.2.7: Ejemplo del proceso de cálculo de un LBP

Tras aplicar el operador LBP en todos los píxeles de la imagen se obtiene la imagen final etiquetada $f_l(x, y)$. Calculamos el histograma en dicha imagen de la siguiente manera:

$$H_i = \sum_{x,y} I\{f_l(x, y) = i\}; \qquad \text{donde} \quad I\{A\} = \begin{cases} 1, & \text{si } A \text{ es verdadero} \\ 0, & \text{si } A \text{ es falso} \end{cases} \tag{2.2.32}$$

donde $i_c = 0, \ldots, n-1$ es la etiqueta dada para cada píxel por el operador LBP (siendo n el valor máximo).

Finalmente, obtenemos los histogramas normalizados que se usarán para entrenar otro método de clasificación, ya sean las estudiadas previamente (PCA, LDA, SVM, CNN, etc.) o el algoritmo *AdaBoost* que estudiaremos más adelante en esta sección:

$$N_i = \frac{H_i}{\sum_{j=0}^{n-1} H_j} \tag{2.2.33}$$

• Algoritmo de Viola-Jones (Haar):

El método de Viola-Jones fue propuesto originalmente en 2001 por P. Viola y M. Jones, quienes le dan nombre al algoritmo [43, 44]. Este método introduce el uso de características Haar (que reciben el nombre de las funciones Haar), que se tratan de patrones rectangulares contiguos (véase Figura 2.2.8) en los que se evalúa la diferencia entre las sumas de intensidades en las zonas claras con las oscuras.

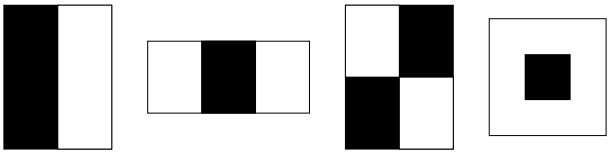


Figura 2.2.8: Tipos de clasificadores Haar [45]

Estas características pueden ser calculadas muy rápidamente usando una interpretación intermedia, la imagen integral. La imagen integral en un píxel (x, y) contiene la suma de píxeles por encima y por su izquierda, siendo su definición [43, 44]:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i\left(x', y'\right), \tag{2.2.34}$$

donde $ii(x, y)$ es el valor del píxel (x, y) en la imagen integral y $i(x, y)$ es el valor del píxel (x, y) en la imagen original.

Usando las siguientes expresiones, la imagen integral se puede calcular fácilmente a partir de la imagen original:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{2.2.35}$$
$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{2.2.36}$$

donde $s(x, y)$ es la suma acumulada de una fila, $s(x, -1) = 0$ y $ii(-1, y) = 0$.

P. Viola y M. Jones proponían una clasificación en cascada como alternativa para mejorar la eficiencia del clasificador final. La idea básica es dividir la imagen completa en sub-ventanas en las que iremos realizando la clasificación, que constará de varias etapas. Cada etapa consta de varios clasificadores, de modo que se pueda descartar rápidamente aquellas subregiones que no tengan rostros en las primeras etapas. Los primeros clasificadores se eligen de forma selectiva para descartar imágenes negativas con un menor coste computacional. Además, este modo de operación permite detectar múltiples rostros en una misma imagen de forma eficiente y precisa.

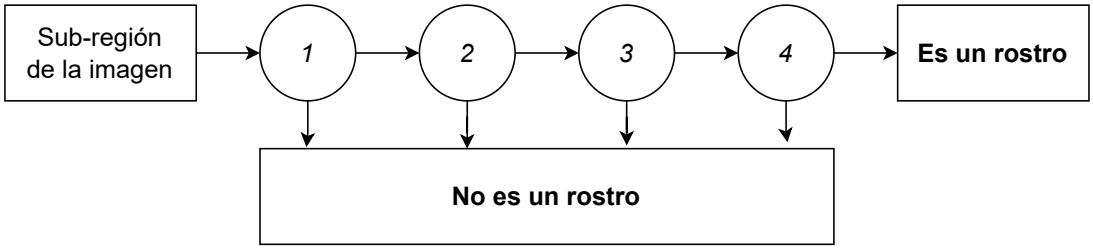


Figura 2.2.9: Diagrama de flujo de la clasificación en cascada [43, 44]

Principalmente, el uso de este tipo de características tiene varias ventajas [44]. Por ejemplo, la información de estas características es más manejable e interpretable que la proporcionada por los métodos que operan con grandes cantidades de datos en crudo. Además, esta técnica es más eficiente que todos los métodos basados en estudiar características de los píxeles, puesto que con pocas de estas características se obtiene un clasificador eficiente.

Para lograr este clasificador se propone el uso de *AdaBoost*, un algoritmo de entrenamiento que nos permite crear un clasificador fuerte a base de varios clasificadores más simples, como los basados en Haar. Para ello, sólo necesitaremos proporcionar imágenes positivas (con rostros) y negativas (sin rostros) en el entrenamiento.

• Algoritmo AdaBoost:

AdaBoost (*Adaptive Boosting*) es un algoritmo de aprendizaje automático (*machine learning*) propuesto por Yoav Freund y Robert Schapire en 1996 [46]. El objetivo principal de AdaBoost es mejorar la precisión de la detección construyendo un clasificador fuerte mediante la combinación ponderada de clasificadores débiles, como LBP o Haar.

El algoritmo de AdaBoost funciona siguiendo las siguientes etapas [44, 47]:

1. **Inicialización:**

   a) Se considera un conjunto de entrenamiento (X, Y) con m muestras $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, donde $x_i \in X = \mathbb{R}^d$ es el vector de características e $y_i \in Y = \{-1, +1\}$ es la etiqueta correspondiente que denota a qué clase pertenece cada vector x_i.

   b) Se inicializan los pesos iniciales para cada dato del conjunto de entrenamiento como $w_1(x_i) = 1/m, \forall i = [1, m]$, donde m es el número total de datos. Los pesos siempre sumarán 1 en total, de modo que para cualquier iteración posterior se seguirá cumpliendo siempre que $w_t(x_i) \in [0, 1], \ \forall t \in [1, T]$.

2. **Entrenamiento de clasificadores débiles:**

   Iniciamos un bucle donde para cada iteración $t = 1, \ldots, T$ se realiza lo siguiente:

   a) Se entrena un clasificador débil en el conjunto de entrenamiento ponderado utilizando los pesos actuales.

   $$\begin{aligned} h_t : X &\to Y = \{-1, +1\} \\ x_i &\to h_t(x_i) \end{aligned} \tag{2.2.37}$$

   b) Se calcula el error ponderado del clasificador débil como:

   $$\epsilon_t = \sum_{i=1}^{m} w_t(x_i) \cdot \delta_t(x_i) \tag{2.2.38}$$

   donde $\delta_t(x_i) = 0$ si se verifica que $h_t(x_i) = y_i$, es decir, si la clasificación se dió correctamente, y $\delta_t(x_i) = 1$ en caso contrario.

   c) Se calcula el peso del clasificador débil como:

   $$\alpha_t = \frac{1}{2} \cdot \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \tag{2.2.39}$$

   d) Se actualizan los pesos normalizados de los ejemplos de entrenamiento:

   $$w_{t+1}(x_i) = \frac{w_t(x_i) \cdot \exp\left((-1)^{\delta_t(x_i)}\alpha_t\right)}{\sum_{j=1}^{m} w_t(x_j) \cdot \exp\left((-1)^{\delta_t(x_j)}\alpha_t\right)} \tag{2.2.40}$$

   donde $(-1)^{\delta_t(x_i)}$ será por definición $+1$ cuando la clasificación sea correcta y -1 en caso de que no lo sea.

3. **Combinación de clasificadores débiles:**

   Se combina el conjunto de clasificadores débiles entrenados en un clasificador fuerte final mediante una ponderación lineal:

   $$H_{final}(x_i) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x_i)\right) \tag{2.2.41}$$

4. **Predicción:**

   Para realizar una predicción con el clasificador fuerte final, simplemente alimentamos al modelo con el nuevo ejemplo problema y observamos la salida.

   a) Si la suma ponderada es positiva, se asigna la clase positiva.

   b) Si la suma ponderada es negativa, se asigna la clase negativa.

Capítulo 3

Dispositivo experimental y resultados

Tras estudiar distintas formas de abordar el problema de la detección de rostros, será necesario encontrar aquel dispositivo experimental que pueda ponerlas en práctica. Para ello, en este capítulo se plantearán los requisitos necesarios para el sistema buscado. A su vez, se incorporará el dispositivo presentado por J. A. Leñero-Bardallo, R. De la Rosa-Vidal, R. Padial-Allué, J. Ceballos-Cáceres, A. Rodriguez-Vázquez y J. Bernabéu-Wittel en el artículo "*A Customizable Thermographic Imaging System for Medical Image Acquisition and Processing*" [5], y veremos varios algoritmos posibles junto a algunas pruebas para evaluar su rendimiento.

3.1. Revisión del dispositivo

Como se ha tratado previamente, queremos encontrar un sistema que sea capar de medir la temperatura facial en tiempo real, que se adapte a las distintas condiciones que surgen en una situación de emergencia sanitaria. En definitiva, buscamos un sistema que verifique los siguientes requisitos:

- **Fácil de usar:** El dispositivo debe ser planteado para usar en cualquier tipo de entorno profesional; hospitales, empresas, tiendas, centros educativos, etc.
- **Rápido:** Los resultados deben ser procesados en tiempos relativamente cortos, para proceder con los protocolos necesarios en caso de detectar fiebre y evitar potenciales futuros contagios.
- **Eficiente:** Los datos proporcionados por el dispositivo deben ser lo suficientemente fiables. En esto influirá tanto el hardware como los algoritmos utilizados.
- **Autónomo:** El sistema debe funcionar sin la necesidad de operarios externos. El individuo será el que se coloque e interactúe con un activador, como un botón o un comando de ejecución por un terminal, y el sistema deberá hacer el resto sin necesidad de interacción externa.
- **Económico:** Dado que el dispositivo está planteado para poder ser utilizado en cualquier tipo de entorno, debe ser asequible en su forma más simple.
- **Información almacenable:** Como se busca identificar a individuos con fiebre, será necesario almacenar datos como la hora y día de detección, el rostro del individuo y la temperatura facial entre otros.

- **Personalizable:** Se busca la adaptabilidad a múltiples escenarios posibles, por lo que es estrictamente necesario la posibilidad de modificar el dispositivo, ya sea mediante la adición de nuevos periféricos o incorporando nuevos algoritmos.
- **Portátil:** Como queremos un sistema que se pueda aplicar a todo tipo de entornos y escenarios, debe ser también fácilmente transportable, es decir, ligero y poco voluminoso.

Como propuesta, se ha optado por utilizar sistemas embebidos, debido a su versatilidad, su bajo consumo, su pequeño tamaño y su bajo coste en comparación a otros sistemas computacionales. De todas las posibles opciones, se ha incorporado una *Raspberry Pi* $3B+$. Como sensor infrarrojo se ha utilizado el FLIR Lepton 3, un sensor de matriz de plano focal de microbolómetro no refrigerado de VOx. Además, se ha aprovechado a su vez el módulo para cámaras presente en las placas *Raspberry Pi* para incorporar también una cámara que actúa en el rango visible, siendo posible así previsualizar rostros o capturar imágenes en dicho rango.

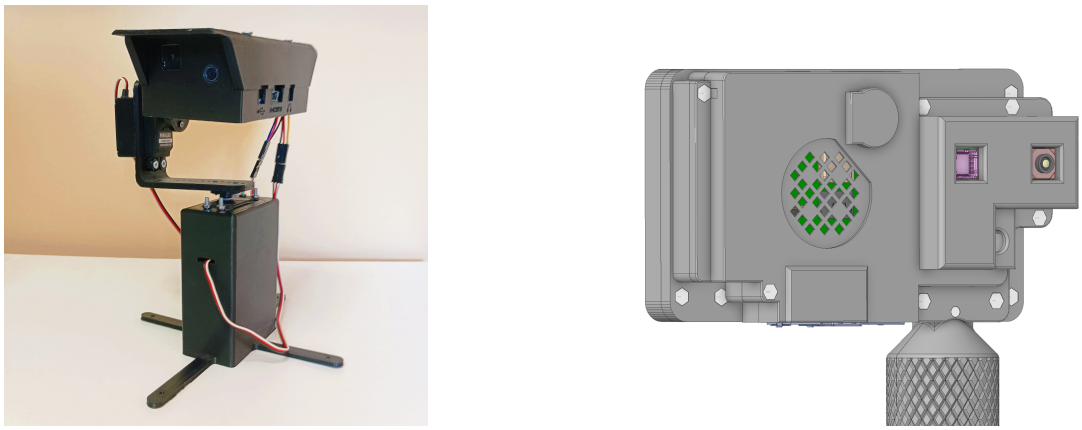
La carcasa en la que se incorpora todo el sitema se ha realizado mediante impresión 3D, constando de una visera por encima de los sensores para facilitar la captura de imágenes, bloqueando el posible exceso de luz. Esta carcasa consta de 4 puntos de apoyos, con la posibilidad ser fijados al suelo o al techo mediante tornillos. También incluye un brazo móvil con 2 ejes de rotación perpendiculares entre si, que pueden ser controlados por dos motores conectados a los pines de la GPIO de la *Raspberry Pi*.

Tanto la elección de elementos que conforman el sistema como el diseño de su carcasa fueron abordados en el artículo "*A Customizable Thermographic Imaging System for Medical Image Acquisition and Processing*" [5].

Teniendo todas estas incorporaciones en cuenta, las características del dispositivo térmico utilizado en este proyecto son las mostradas en la Tabla 3.1.

CPU	Broadcom $BCM2837B0$ Cortex-$A53$ 64-*bit* SoC @1,4 GHz
GPU	Broadcom Videocore-IV
RAM	1 GB LPDDR2 SDRAM
Memoria SD	64 GB microSDXC
Consumo medio	890 mA @5 V
Sistema operativo	Raspbian
Resolución (IR)	160×120 píxeles
Frame rate (IR)	> 2 FPS (Efectivo: 8,7 FPS)
Exactitud	0,1 $°C$ (tras calibración)
Resolución (RGB)	3280×2464 píxeles (imágenes estáticas)
Frame rate (RGB)	1920×1080 @30 FPS
Peso	333 g
Conectividad WiFi	Sí
Conectividad Bluetooth	Sí
Conectividad Ethernet	Sí

Tabla 3.1: Especificaciones del sistema [5]



(a) (b)

Figura 3.1.1: Diseño del dispositivo experimental: (a) Sistema utilizado (b) Renderizado 3D de una versión alternativa del sistema con pantalla y batería [5]

Sería posible actualizar el hardware del sistema, implementando modelos superiores de los sensores o de Raspberry Pi. Para ello, únicamente habría que editar la carcasa incorporando nuevos espacios para estos o redistribuyendo el espacio si fuera necesario, y posteriormente, proceder la impresión de la nueva versión de la carcasa. En la Figura 3.1.1 se muestran el dispositivo utilizado en este proyecto y una versión alternativa procedente del mismo artículo.

En la Tabla 3.2, se compara este dispositivo térmico con múltiples alternativas de distintos fabricantes y de mayor coste. Podemos concluir que el sistema propuesto es bastante competitivo frente a otros comerciales, siempre y cuando el individuo a capturar se posicione a menos de 2 metros del sensor infrarrojo y el sistema sea calibrado con un termómetro de alta precisión.

Sensor	**Sensibilidad (°C)**	**Exactitud (°C)**	**Precisión (°C)**	**Rango (°C)**	**Rango (λ)**	**Resolución**	**Programable**	**Expansión de hardware**
FLUKE TiX560	$\leq 0{,}03$	*ND* (calibrado por fabricante)	± 2	$[-20, 1200]$	LWIR & Visible	320×240	No	Limitada
FLIR A320	$\leq 0{,}05$	$\pm 0{,}5$ (calibrado por fabricante)	± 2	$[-20, 120]$	LWIR	320×240	No	No
Mediterm IRIS 640	0,01	$< 0{,}5$ (calibración manual)	*ND*	$[17, 38]$	LWIR	640×480	No	No
Este sistema	0,05	$< 0{,}1$ (calibración manual)	± 1	$[-10, 80]$	LWIR & Visible	160×120	Sí	Sí

Tabla 3.2: Comparación de diferentes sistemas infrarrojos para la medición de la temperatura facial [5]

Como se muestra en la Tabla 3.1.1, bajo una calibración adecuada, el sensor FLIR Lepton 3 en este sistema ha demostrado una incertidumbre de $\pm 0{,}1$ °C [5]. Sin embargo, para la calibración de este proyecto se disponía un termómetro *HTD8813*, cuya incertidumbre en la medida es de $\pm 0{,}2$ °C, superior a la incertidumbre demostrada por el dispositivo. Esto significa que la incertidumbre en las pruebas realizadas serán mayores, pero que sería posible mejorar las medidas mediante una correcta calibración con un termómetro de mayor precisión.

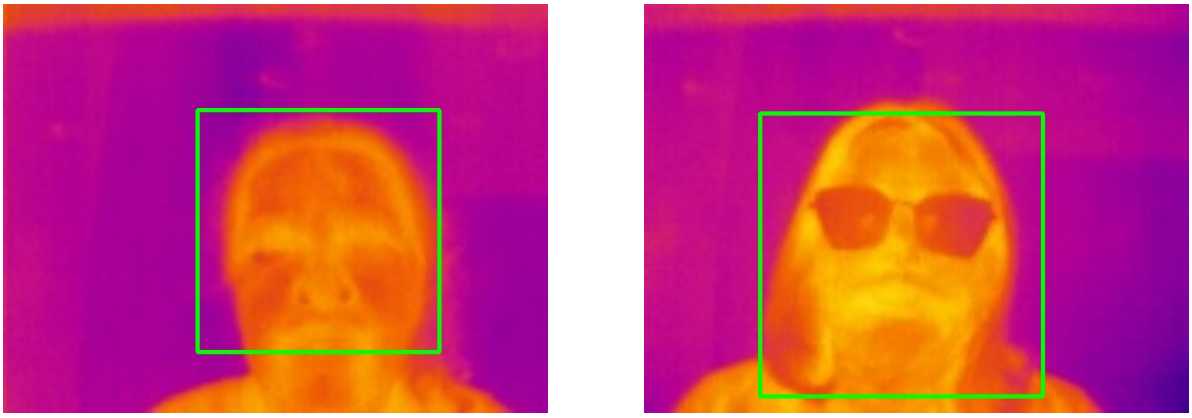
La incorporación de una Raspberry Pi $3B+$ facilitará la tarea de incluir software personalizado en el dispositivo. Para ello, se ha instalado el sistema operativo Raspbian, junto a una librería que incorpora herramientas dedicadas a la visión por computadora, *OpenCV* [6–8]. A continuación se introducirán los distintos métodos estudiados para abarcar el problema de la detección de rostros y la medición de la temperatura facial.

3.2. Método I: Detección directa

Para empezar a desarrollar nuestro algoritmo es necesario establecer el método de detección que vamos a seguir. Es conveniente que la técnica escogida sea basada en características, puesto que el objetivo final es medir la temperatura para rostros que pueden presentar gafas, mascarillas y otros accesorios, mientras que con métodos basados en imágenes esto puede resultar más complejo.

Podríamos plantearnos utilizar redes neuronales y escoger filtros apropiados para abordar dicha tarea, pero es probable que nos encontremos con problemas de rendimiento en este sistema, debido a su menor capacidad de cómputo en comparación a otros sistemas computacionales, u otros problemas que requieran de un conocimiento más profundo en este tipos de sistemas. Por lo tanto, una elección conveniente será la de utilizar análisis de características, Haar o LBP, puesto que nos ofrece un equilibrio entre alta precisión en detecciones y bajo consumo de memoria y, por tanto, alta velocidad en las detecciones. En nuestro caso se utilizaron clasificadores del tipo Haar, pero también es posible utilizar el mismo algoritmo para clasificadores LBP, pues *OpenCV* ofrece esa posibilidad.

La detección mediante clasificadores Haar nos proporcionará como resultado un recuadro que contiene el rostro detectado, como se muestra en la Figura 3.2.1. Sin embargo, esto no será suficiente para medir su temperatura media, puesto que aparecerán pelo, accesorios y parte del fondo que contribuirían al cálculo, dando lugar a una medición errónea. Para realizar esta tarea se probó a utilizar un thresholding de Otsu, que hallaría el valor umbral de IR que separa al rostro del resto de elementos que no contribuyen a la temperatura facial. Por tanto, nos valdría con utilizar el algoritmo de Otsu y realizar una segmentación binaria imponiendo el valor obtenido como umbral.



(a) (b)

Figura 3.2.1: Ejemplos de rostros detectados utilizando clasificadores Haar en el rango infrarrojo: (a) Rostro descubierto (b) Rostro con gafas

Finalmente, sólo quedaría realizar el cálculo del valor promedio de píxeles IR de la imagen segmentada y calcular la temperatura equivalente, siendo posible su determinación tras una previa calibración del dispositivo. Uniendo todos estos procesos, tenemos un resumen de los pasos necesarios en nuestro algoritmo para obtener la temperatura media facial, aunque posteriormente veremos que podremos añadir otros procesos para mejorar tanto la presentación como la comodidad en el uso del programa.

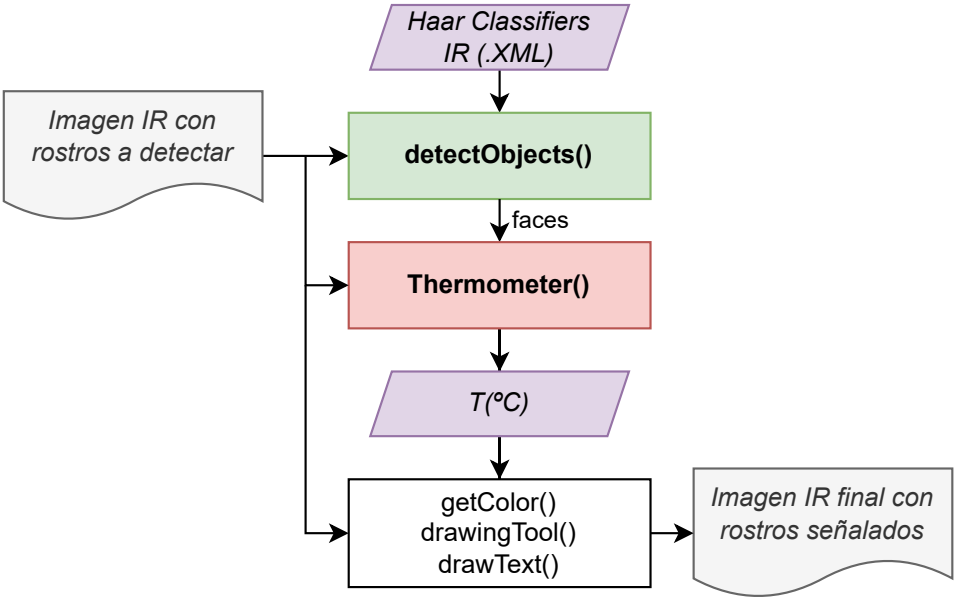


Figura 3.2.2: Diagrama de bloques del algoritmo personalizado para la detección facial en el rango *LWIR* y la medición de su temperatura

En la Figura 3.2.2 se muestra un diagrama que resume la implementación del algoritmo planteado para la detección de rostros en el IR. A este algoritmo lo hemos denominado de *detección directa*, puesto que busca realizar la detección mediante clasificadores Haar directamente en el rango IR.

La función *detectObjects*( ) es la encargada de detectar los rostros mediante clasificadores, Haar o LBP, haciendo uso de cv::CascadeClassifier::*detectMultiScale*( ), función proporcionada por *OpenCV* para detectar objetos. Como resultado, proporciona un vector en cuyos elementos se indica las coordenadas, el ancho y el alto del rectángulo que contiene cada rostro detectado. También se aportan estos datos para los rectángulos que contienen objetos anidados, como los ojos, si al ejecutar el programa se le proporcionan como entrada los clasificadores necesarios para ello.

La función *Thermometer*( ) hace uso de cv::*threshold*( ), función de *OpenCV* que nos permite aplicar distintos tipos de segmentación en la imagen. Primero, se obtiene el valor umbral de Otsu para cada rectángulo con un rostro detectado. Después, se aplica una segmentación binaria que elimina de la imagen los píxeles por debajo de dicho umbral. En este proceso se determina una máscara binara que contiene píxeles completamente blancos en la posición de los que se quieren conservar y otros completamente negros para los que se quieren borrar. Esta máscara será necesaria tanto para obtener la imagen final segmentada como para el cálculo de la media final. Para ello, se utiliza la función cv::*mean* para determinar el valor promedio de los píxeles en la imagen segmentada indicados por la máscara, descartando los píxeles negros del cálculo. Para la calibración de la temperatura,

se ha utilizado el ajuste lineal estudiado en la Ecuación 2.1.11. Para ello, se ha utilizado un termómetro *HTD8813* cuya incertidumbre es de $\pm 0{,}2\ ^{\circ}C$ en el rango de operación, tal y como se ha indicado anteriormente en la revisión del dispositivo.

Finalmente, se crearon distintas funciones que actuarían sobre la imagen capturada. Por un lado, *drawingTool*( ) dibuja los rectángulos que contienen los rostros y líneas cruzadas '+' encima de los ojos. Después de esto, *drawText*( ) se encarga de escribir la temperatura facial debajo del rectángulo. Previamente a esto, la función *getColor*() toma con entrada el valor de la temperatura medida y devuelve el color que utilizarán las funciones anteriores según si superan o no dos valores umbrales establecidos manualmente: verde si no los supera, naranja para una destemplanza ligera y rojo si está por encima de lo que se considera fiebre.

A la hora de probar nuestro algoritmo encontramos una serie de adversidades que tuvimos que afrontar de distinto modo. El principal problema que encontramos es que no hay clasificadores Haar de uso público para rostros en el IR, por lo que fue necesario buscar vías alternativas. En primer lugar, se exploró la posibilidad de entrenar nuestros propios clasificadores Haar en el rango IR, utilizando los programas de muestra de *OpenCV*, *opencv_createsamples* y *opencv_traincascade*. Estos programas sirven para entrenar nuestros propios clasificadores, proporcionando imágenes positivas que contengan los rostros e imágenes negativas que guarden cierta relevancia con rostros pero que no los contengan, ni siquiera parcialmente. Para ello, se utilizaron distintas imágenes de *OTCBVS Benchmark dataset* [48, 49], una base de datos con imágenes de referencia en el IR. Sin embargo, los clasificadores entrenados no daban lugar a ninguna detección en las imágenes de prueba. Esto puede deberse a varias razones:

- No se proporcionó una variedad lo suficientemente amplia de rostros para el entrenamiento. Es necesario proporcionar imágenes de rostros de más individuos y no repetir tanto de un mismo individuo.
- Las imágenes negativas usadas en el entrenamiento no eran lo suficientemente adecuadas, deben tener más detalles que se asemejen a rostros.

En ambos casos, el problema a afrontar era el mismo, pues la mayoría de bases de datos IR encontradas eran privadas o requerían de una acreditación como investigador para solicitar obtener acceso a ellas. Por tanto, no fue posible acceder a un repertorio de rostros lo suficientemente amplio ni obtener imágenes negativas variadas para el entrenamiento. Como último intento se probó a utilizar los clasificadores Haar proporcionados por *OpenCV* que habían sido entrenados en el rango visible. Se encontró que el clasificador *haarcascade_frontalface_alt2.xml* era el que ofrecía mejores resultados, pero aún así veremos a continuación que no eran lo suficientemente satisfactorios.

Para comprobar la eficacia de nuestro algoritmo, se han tomado imágenes de rostros a una distancia fija de 50 *cm* entre el individuo y el dispositivo. Las imágenes eran tomadas por pares, capturando simultáneamente tanto en el rango IR como en el rango visible. Veremos que las imágenes en el rango visible serán relevantes más adelante en el siguiente método. En total, se tomaron 300 pares de imágenes, de los que se incluyeron distintas categorías: rostro descubierto, mascarilla, cobertura parcial o completa del pelo, gafas y combinaciones entre estas. Cada 5 imágenes, se giraba el rostro en torno a los 45° hacia cada lado, tomando 5 de frente, 5 para la cara izquierda y 5 para la cara derecha.

La razón de tomar 5 imágenes seguidas en cada caso es la de comprobar la precisión del dispositivo en medidas continuadas. También habría que tener en cuenta que el dispositivo está pensado para situaciones en las que se toman imágenes frontales de los rostros, para maximizar la superficie de medición y disminuir potenciales errores en el segmentado de Otsu. Aún así, resultará interesante tomar capturas de los lados de los rostros para comprobar cuánto disminuye el número de detecciones.

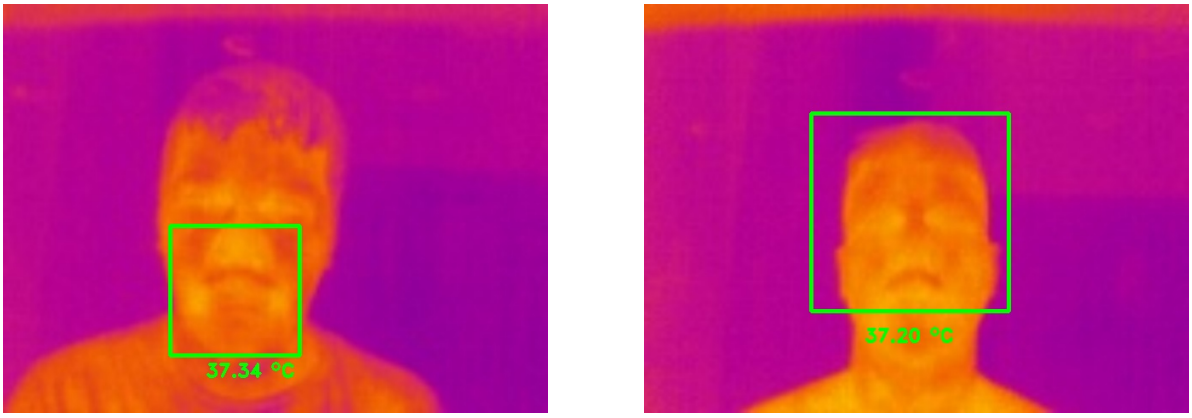
	Número de imágenes
N	75
M	60
C	75
M+C	60
G	15
M+C+G	15

	FRENTE	*LADO IZQ.*	*LADO DER.*	**SUB.**
N	68,00 %	20,00 %	0,00 %	29,33 %
M	35,00 %	0,00 %	0,00 %	11,67 %
C	80,00 %	36,00 %	52,00 %	56,00 %
M+C	15,00 %	0,00 %	20,00 %	11,67 %
G	100,00 %	0,00 %	0,00 %	33,33 %
M+C+G	100,00 %	0,00 %	100,00 %	66,67 %
SUB.	57,00 %	14,00 %	22,00 %	

Tabla 3.3: Tasas de aciertos obtenidas por el método de *detección directa* separadas por categorías: por el tipo de vestimenta (en filas) y por la parte del rostro que se mostraba a la cámara (en columnas). Nomenclatura utilizada: *N* - Nada, *M* - Mascarilla, *C* - Cobertura parcial, *G* - Gafas.

En la Tabla 3.3 se muestran el número de detecciones para cada categoría, separando por columnas según la parte del rostro capturada y por filas según los tipos de accesorios que se portaban. Además, también se aportan los subtotales relativos en cada caso. La tasa de detecciones total sería de un 31,00 % aunque, como se ha indicado anteriormente, la tasa relevante será el subtotal de detecciones de frente. Para las detecciones de lado se preparó al algoritmo para que iterara probando tanto con el clasificador originalmente mencionado como con otro también proporcionado por *OpenCV* para rostros de perfil, *haarcascade_profileface.xml*. Aún así, vemos que los subtotales relativos de detecciones para los lados de los rostros, aunque son parecidas, difieren bastante con el valor para las detecciones de frente. El tiempo de detección ha sido en promedio de 0,04 s, con un valor máximo de 0,06 s y un valor mínimo de 0,03 s. En cuanto a la variación en el valor de la temperatura en medidas simultáneas, la variación máxima ha sido de 0,03°C.

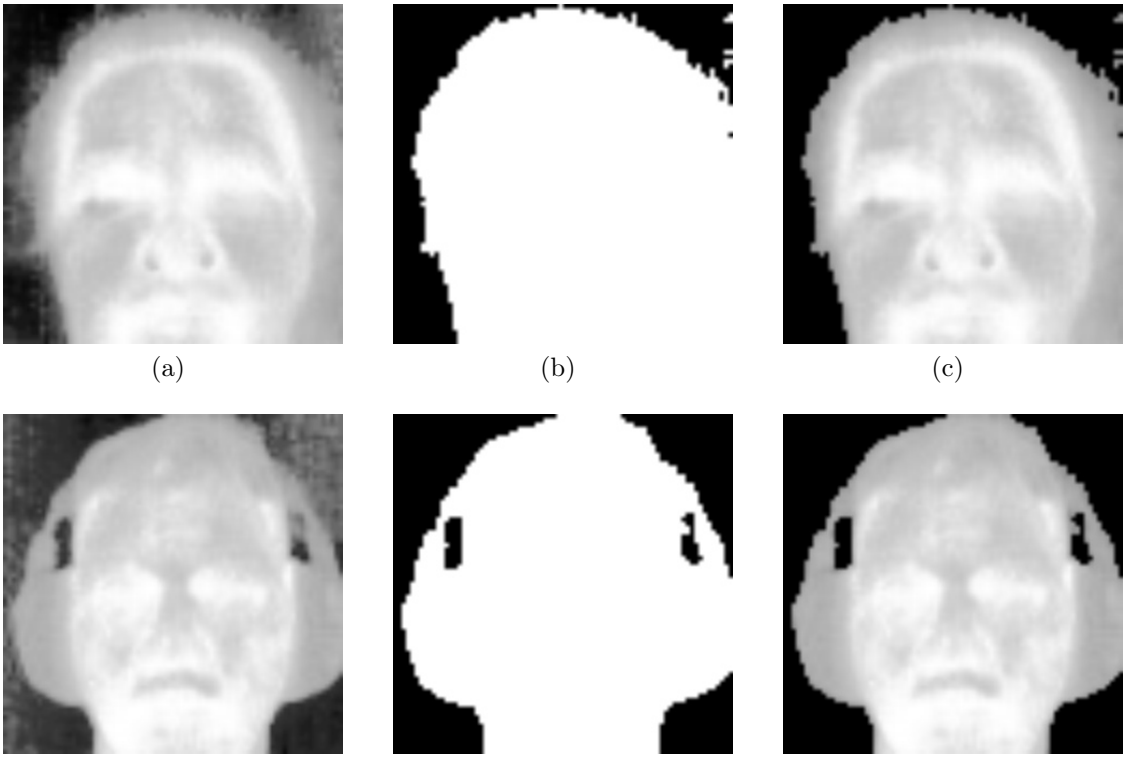
La baja tasa de detecciones no es el único problema presente en este primer método, pues también se encontraron distintos inconvenientes que resultaban en una medición incorrecta de la temperatura facial. Entre estos obstáculos, tenemos que a veces al aplicar el clasificador la detección no se daba del rostro completo, es decir, entendía por rostros fracciones de la cara o incluso objetos que no lo eran. Esto es, entre otros factores, debido a que el clasificador había sido entrenado en el rango visible, y las imágenes infrarrojas aunque geométricamente se asemejen no contienen la misma información relativa entre píxeles. Por otro lado, también era habitual que cuando la detección se daba correctamente no encuadrara la cara de forma adecuada, por lo que el thresholding de Otsu era menos satisfactorio. En la Figura 3.2.3 se muestran varios de estos problemas encontrados en el proceso de detección.



(a) (b)

Figura 3.2.3: Problemas encontrados en el testeo del dispositivo mediante el método de "detección directa": (a) Detección errónea o incompleta (b) Detección no centrada correctamente

Otro de los inconvenientes que presenta este método es que, debido a que la detección no encuadraba los rostros en su completitud, el segmentado de Otsu no daba resultados adecuados. Esto ocurría porque no se discriminaban adecuadamente el "foreground", la piel, del "background", accesorios, pelo y pared.



(a) (b) (c)

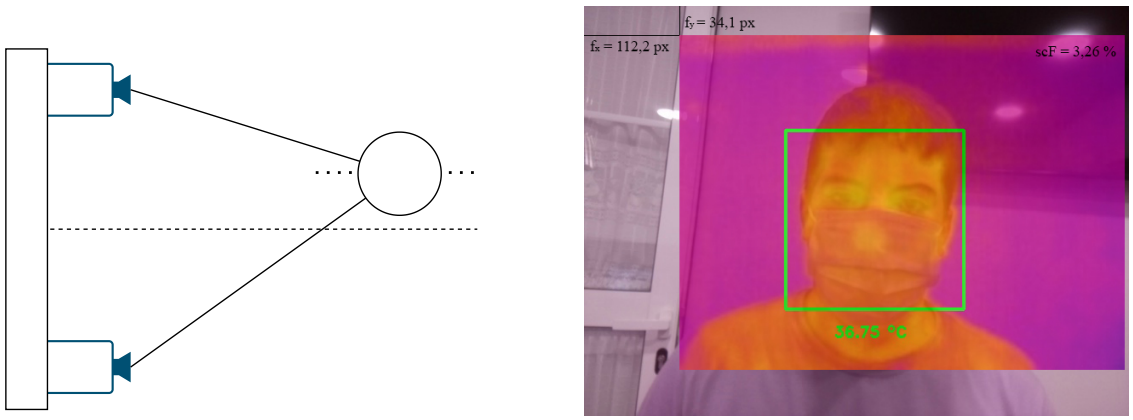
(d) (e) (f)

Figura 3.2.4: Segmentación de rostros detectados: i) Rostro detectado (ROI) ii) Máscara dada por la segmentanción de *Otsu* iii) ROI segmentado tras la aplicación de la máscara. Las figuras (a)-(c) corresponden a un rostro despejado y las figuras (d)-(f) a un rostro con auriculares.

3.3. Método II: Detección indirecta

Debido a los problemas abordados anteriormente, la medición de la temperatura mediante nuestro algoritmo era imprecisa e inexacta. Por tanto, es necesario considerar una forma alternativa de cumplir nuestros objetivos. Para ello, se plantea la posibilidad de utilizar la cámara visible presente en el dispositivo para realizar la detección y, posteriormente, remapear las coordenadas, el ancho y el largo del rectángulo que contiene el rostro detectado a la imagen infrarroja tomada de forma simultánea. A partir de este punto se procede de forma análoga al método anterior, es decir, realizando un segmentado de Otsu y determinando la temperatura facial a partir del valor promedio. Debido a que la detección se produce en el rango visible, denominamos a este algoritmo como método de *detección indirecta*.

El problema presente en este método es que la imagen captada por la cámara infrarroja no es un duplicado con distinta resolución de un segmento de la imagen visible. Esto es así porque, aunque las cámaras están contenidas en un mismo plano, están desplazadas una distancia la una de la otra y, dado que los rostros no son objetos planos, las cámaras observan el rostro desde un ángulo ligeramente distinto, tal y como se muestra de forma esquematizada en la Figura 3.3.1 (a). Por otro lado, las cámaras ni siquiera poseen las mismas especificaciones y poseen resoluciones muy diferentes entre si, lo cual resulta en una tarea aún más compleja de sobrellevar.



(a) (b)

Figura 3.3.1: Ilustración del proceso de "remapping" facial: (a) Problema óptico (b) Esquematización: imagen LWIR superpuesta con un 75 % de opacidad

Dado que el error cometido en la medida aumenta cuadráticamente con la distancia [5], es necesario tomar las imágenes faciales lo más próximas posibles. En nuestro caso, como se ha indicado en el método anterior, se ha procedido con una distancia fija de unos 50,0 *cm* entre el rostro y el sistema para cada individuo. Para estas distancias, considerar que la imagen infrarroja y la imagen visible contemplan objetos planos ha demostrado ser una buena aproximación, pues la incertidumbre es inferior al 5 %. Esta incertidumbre no es relevante pues la medición final de la temperatura dependerá de la bondad en la discriminación del thresholding de Otsu, que será viable siempre que la detección englobe correctamente el rostro y comprenda una buena parte del fondo.

Considerar objetos planos colocados de frente facilita mucho los cálculos. En el caso de un punto dado en la imagen visible (x_{Vis}, y_{Vis}), sus coordenadas (x_{IR}, y_{IR}) en la imagen infrarroja vendrán dadas por:

$$\begin{cases} x_{Vis} = f_x + ScF \cdot x_{IR} \\ y_{Vis} = f_y + ScF \cdot y_{IR} \end{cases} \implies \begin{cases} x_{IR} = (x_{Vis} - f_x)/ScF \\ y_{IR} = (y_{Vis} - f_y)/ScF \end{cases} \tag{3.3.1}$$

donde f_x y f_y son factores de desplazamiento y ScF es un factor de escala de resolución entre ambas imágenes. Hay que tener en cuenta que en *OpenCV* el origen está colocado en el píxel de la esquina superior izquierda de la imagen y el resto de píxeles tendrán coordenadas (x, y) positivas.

Las longitudes en la imagen infrarroja se obtienen del siguiente modo:

$$\begin{cases} \Delta x_{IR} = (x_{2Vis} - f_x)/ScF - (x_{1Vis} - f_x)/ScF = \Delta x_{Vis}/ScF \\ \Delta y_{IR} = (y_{2Vis} - f_y)/ScF - (y_{1Vis} - f_y)/ScF = \Delta y_{Vis}/ScF \end{cases} \tag{3.3.2}$$

Se han utilizado 57 imágenes de rostros de muestra para calibrar el "remapping", obteniendo los siguientes factores: $f_x = 112,2\ px$, $f_y = 34,1\ px$ y $ScF = 327\,\%$. Las décimas en las coordenadas de un píxel no representan nada, ni en f_x o f_y, ni en los productos con el factor de escala $1/ScF$, por ello, la función cv::*cvRound*( ) redondea las coordenadas finales al número entero más cercano.

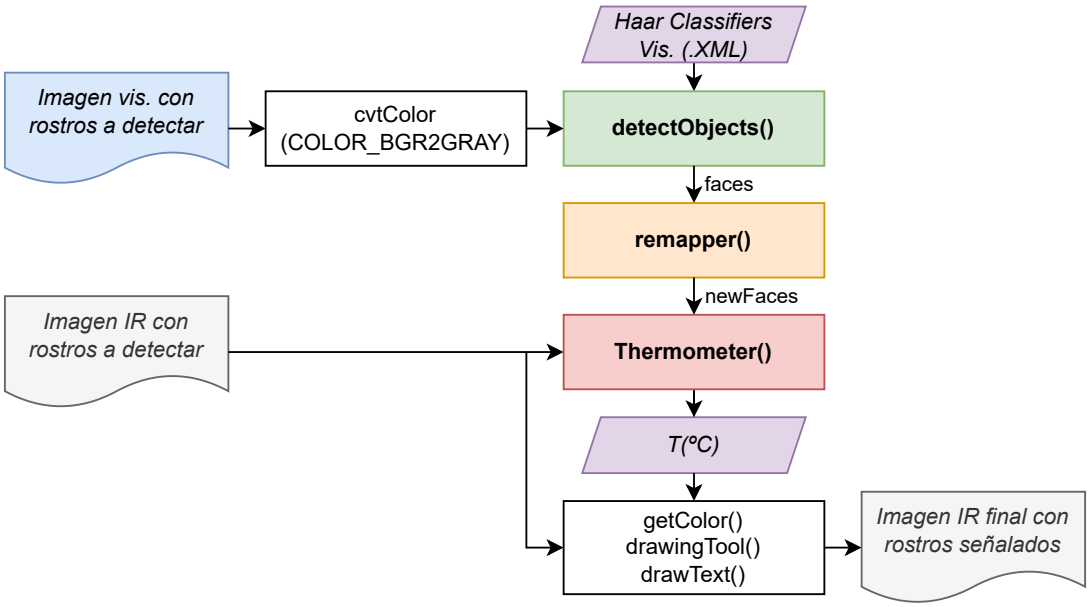


Figura 3.3.2: Diagrama de bloques del algoritmo personalizado para la detección facial en el rango visible, posterior reasignación al rango *LWIR* y la medición de su temperatura

En la Figura 3.3.2 se muestran las nuevas incorporaciones al diagrama de bloques del método anterior presentado en la Figura 3.2.2. En este caso, se hace primero la detección en el rango visible utilizando los clasificadores Haar entrenados en el mismo rango y proporcionados por *OpenCV*. Posteriormente, se realiza el segmentado en la imagen infrarroja y se mide la temperatura. La función *remapper*( ) es la encargada de trasladar a la imagen infrarroja las coordenadas y longitudes de los rectángulos que contienen los rostros detectados en el rango visible. También es posible realizar el mismo cambio para

los objetos anidados. Para este propósito, se diseñó la función *nestedRemapper*( ) que actúa sobre cada rostro reasignando las coordenadas para cada ojo señalado.

Para el testeo de este método, se han utilizado los mismos pares de imágenes de rostros que se usaron para probar el método anterior. Sin embargo, la diferencia ahora es que las imágenes en el rango visible también se utilizarán. Para la detección se utilizaron los clasificadores aportados por *OpenCV*, *haarcascade_frontalface_ alt2.xml* para los rostros de frente y *haarcascade_ profileface.xml* para los lados. En la Tabla 3.4 se presentan las tasas de aciertos siguiendo el mismo criterio que se utilizó en el método anterior.

	Número de imágenes
N	75
M	60
C	75
M+C	60
G	15
M+C+G	15

	FRENTE	*LADO IZQ.*	*LADO DER.*	**SUB.**
N	100,00 %	80,00 %	92,00 %	90,67 %
M	75,00 %	40,00 %	20,00 %	45,00 %
C	100,00 %	80,00	100,00 %	93,33 %
M+C	50,00 %	25,00 %	25,00 %	33,33
G	100,00 %	100,00 %	100,00 %	100,00 %
M+C+G	0,00 %	0,00 %	0,00 %	0,00 %
SUB.	80,00 %	58,00 %	62,00 %	

Tabla 3.4: Tasas de aciertos obtenidas por el método de *detección indirecta* separadas por categorías: por el tipo de vestimenta (en filas) y por la parte del rostro que se mostraba a las cámaras (en columnas). Nomenclatura utilizada: *N* - Nada, *M* - Mascarilla, *C* - Cobertura parcial, *G* - Gafas.

Tras probar el algoritmo mediante el método de "*detección indirecta*" se tuvo un 66,67 % de detecciones sobre el total de imágenes de prueba. Vemos que detectando el rostro de frente el porcentaje de aciertos asciende a un 80,00 % y que para ambos lados, la diferencia con el caso frontal es inferior a la que se tenía mediante el método anterior. También, si nos fijamos en los porcentajes por categorías, se observa que las detecciones disminuyen cuando hay una mascarilla presente y cuando hay cobertura total del pelo. Esto es previsible puesto que es más difícil detectar rasgos del rostro si parte de este está cubierto. A pesar de haber utilizado un análisis de características, los clasificadores Haar de *OpenCV* no están entrenados con rostros que porten mascarillas o mayor cobertura facial. Por tanto, es posible entrenar un clasificador que tenga en cuenta estos accesorios dando lugar a mejores resultados.

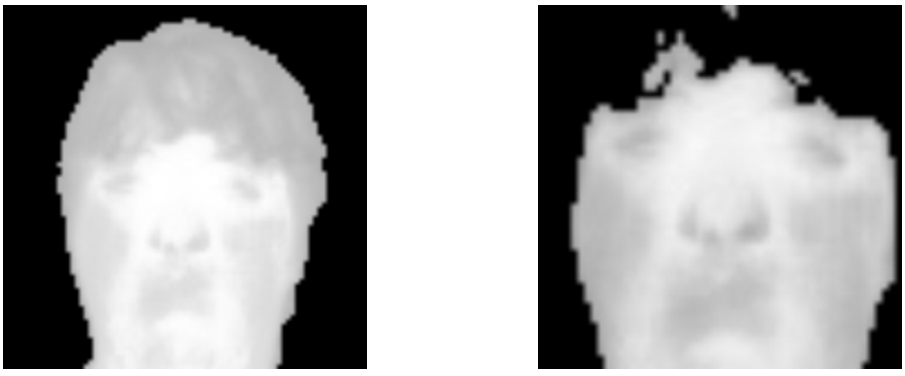
Estas detecciones se han realizado en un tiempo promedio de 0,50 *s* por ejecución, con un valor máximo de 0,79 *s* y un valor mínimo de 0,34 *s*. Vemos que ahora el tiempo empleado es superior al del algoritmo anterior, pero esto no tiene que ver con el remapeo de coordenadas puesto que sólo se está teniendo en cuenta el tiempo empleado en la detección. Esto ocurre debido a que la resolución de las imágenes del rango visible empleadas es superior al de las imágenes en el infrarrojo. En cuanto a la variación en el valor de medidas simultáneas, la variación máxima ha sido de 0,10°*C*. Hay que tener en cuenta que a esta variación contribuyen tanto la precisión del sensor, como la precisión de los algoritmos aplicados.

Por tanto, podemos concluir que, en comparación al método anterior, la precisión en la detección de rostros es mucho mayor mediante este método. Además, mediante este método también nos deshacemos del problema de la baja resolución en sensores infrarrojos.

(a) (b) (c)

Figura 3.3.3: Segmentación de rostro con auriculares por el método de : (a) Rostro detectado (*ROI*) (b) Máscara dada por la segmentanción de *Otsu* (c) *ROI* segmentado tras la aplicación de la máscara

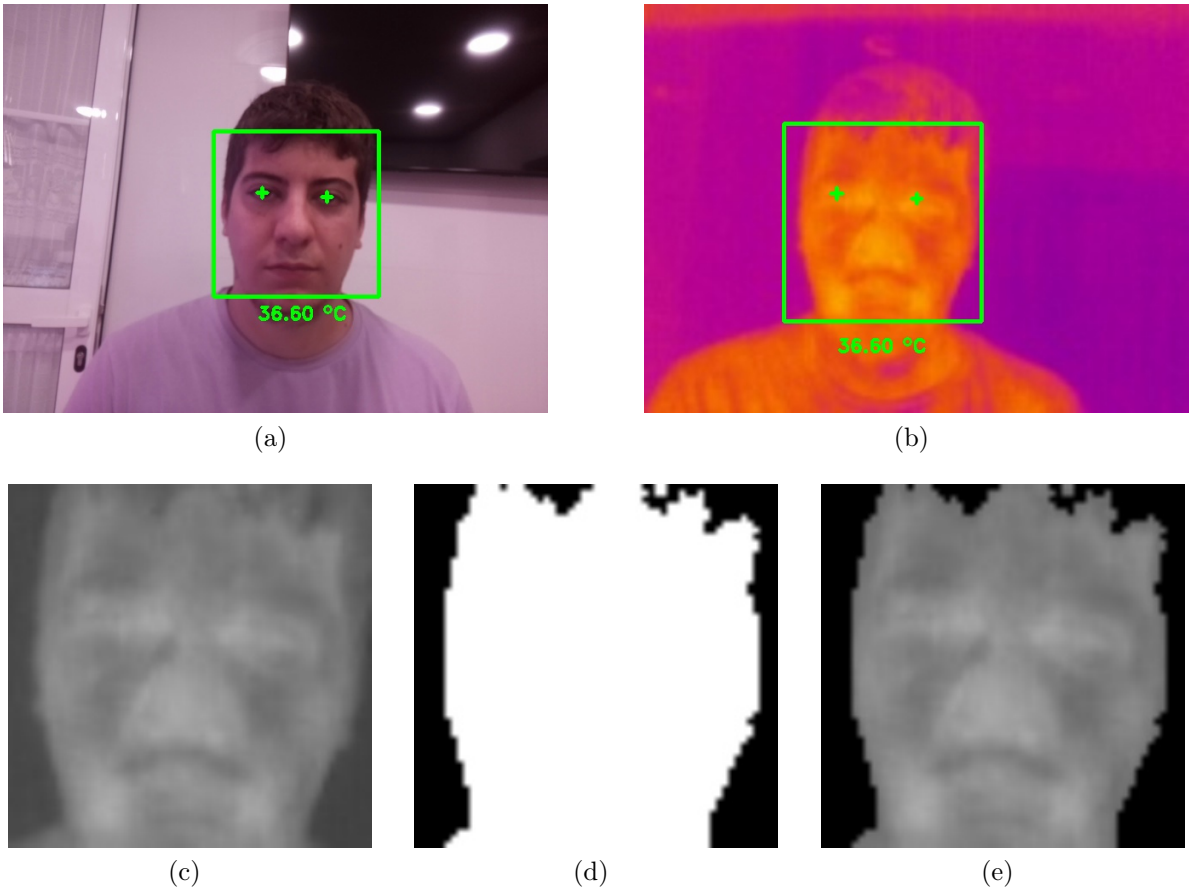
En cuanto a la segmentación, vemos que, al mejorar la calidad en la detección y aislar mejor la piel, el algoritmo de Otsu ofrece mejores resultados. Por un lado, en la Figura 3.3.3 volvemos a ver el ejemplo de un rostro con auriculares. Vemos como ahora los auriculares no se tienen en cuenta en la segmentación. Durante la realización de este proyecto, también fue posible tomar una muestra de un rostro con fiebre, tal y como se muestra en la Figura 3.3.4. Las temperaturas medidas de forma externa fueron de $(37{,}3 \pm 0{,}2)^{\circ}C$ en el cuello y de $(38{,}3 \pm 0{,}1)^{\circ}C$ en la axila, mientras que la temperatura facial medida por el sistema fue de $(37{,}1 \pm 0{,}2)^{\circ}C$. Se ha probado a realizar la segmentación mediante los dos métodos planteados: detección directa y detección indirecta. Se observa que el segmentado de Otsu realizado por el método de detección indirecta es más satisfactorio, puesto que discrimina mejor la piel del resto de elementos.



(a) (b)

Figura 3.3.4: Imagen facial de una persona con fiebre segmentada siguiendo ambos métodos: (a) Detección directa (b) Detección indirecta

En la Figura 3.3.5 se muestra un ejemplo donde se presentan todas las imágenes que muestra el programa en su ejecución: el rostro detectado tanto en el rango visible como en el infrarrojo, el rostro aislado, la máscara que muestra en blanco todos los píxeles por encima del valor umbral de Otsu y la imagen segmentada final aplicando dicha máscara sobre el rostro aislado. En este ejemplo volvemos a ver cómo el algoritmo de Otsu discrimina correctamente la piel del pelo y el fondo.



(a) (b)

(c) (d) (e)

Figura 3.3.5: Detección facial y medición de la temperatura: (a) Detección en el rango visible (b) Remapping al rango LWIR (c) Rostro detectado (*ROI*) (d) Máscara dada por la segmentanción de *Otsu* (e) *ROI* segmentado tras la aplicación de la máscara

3.4. Cambios en el programa final

Para facilitar el uso del programa, se incluyeron distintas mejoras. En primer lugar, se han incluido ambos métodos en el mismo código del mismo, pudiendo elegir entre el uso de uno u otro según si se incluye o no el indicador *–remapping* en su ejecución. También se ha proporcionado la posibilidad de cambiar las escalas de las imágenes finales de forma independiente en ambos rangos, infrarrojo y visible, reescalando mediante interpolación lineal. Esto fue necesario puesto que al tener la cámara infrarroja menor resolución, el texto de la temperatura dibujado sobre la imagen final era apenas visible. Por otro lado, se ha incorporado la posibilidad de incluir el indicador *–try-flip* para detectar caras invertidas, en caso de que se quiera usar el dispositivo anclado al techo y dado la vuelta.

Entre otras de las características del programa, se incluye un parser que hace posible incluir estos indicadores en su ejecución. También, se tienen en cuenta en el código los posibles errores en su ejecución y han sido clasificados adecuadamente, haciendo que cada uno dé como resultado un número negativo concreto en retorno del programa.

Es posible elegir la carpeta en la que se guardarán las imágenes capturadas y procesadas, incluyéndose en el nombre de estas la fecha y hora de procesado. También se puede elegir entre procesar imágenes proporcionadas de forma externa o que sean capturadas en ese mismo instante. Para mayor comodidad en este proceso, se ha incluido una previsualización en el rango visible que se muestra en una ventana externa que incluye un botón para realizar la captura y comenzar a procesar los datos. Si se desea, es posible pulsar la tecla 'S' para guardar las imágenes detectadas. En la Figura 3.4.1 se muestra la interfaz de usuario de dicha ventana.

Por último, quedaría remarcar que aunque el número de detecciones falsas en ambos métodos fue bajo, dándose en total 18 mediante detección directa y 29 mediante detección indirecta, es posible corregirlas. En el primer caso, estas se debieron a que el clasificador no estaba entrenado en el rango infrarrojo y en el segundo método, estas falsas detecciones se debieron en su totalidad a reflejos que el algoritmo interpretaba como rostros. Para solucionarlos ha sido tan sencillo como establecer un mínimo en los tamaños de los rostros o tenerlo en cuenta a la hora de elegir el lugar donde se utilizará el sistema. Para los objetos anidados se realizó la misma corrección.

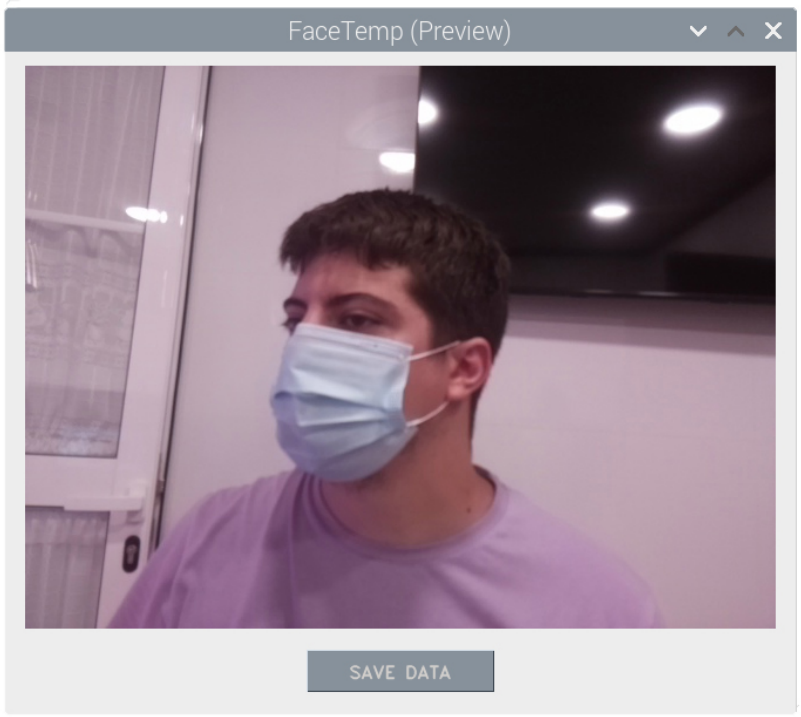


Figura 3.4.1: Interfaz final del programa: previsualización en el rango visible

Para más información es posible consultar el Anexo I, en el que se presenta el código completo del programa final. También existe la posibilidad de ver el código en el siguiente enlace de Github:

https://github.com/adriancordones/FaceTemp

3.5. Posibles mejoras en el sistema

En las secciones anteriores se han explorado los requisitos que debe tener nuestro sistema y todos los componentes que lo constan, pero como se ha indicado esta elección no es única, pues existen vías alternativas de afrontar la tarea presente en este proyecto. Una de las principales características era que el sistema fuera personalizable, y esto es posible atendiendo tanto al software como el hardware utilizado.

Si nos centramos en la viabilidad económica del sistema, los precios han fluctuado constantemente en los últimos meses debido a la reciente escasez de semiconductores y a los cambios constantes en la economía global. Por eso, sería útil estudiar distintas alternativas a los componentes más destacables que se han utilizado en este proyecto, el sensor térmico y el sistema embebido. Tenemos también como problema añadido que, para estos componentes, es habitual que los precios suban en modelos antiguos con la salida al mercado de nuevos modelos, debido a que su oferta decrece con el tiempo. En las Tablas 3.5 y 3.6 se muestran las especificaciones y precios estimados para distintos sistemas embebidos y sensores, incluyendo en la primera columna el precio original de los productos cuando se comenzó el proyecto y en el resto de columnas otras alternativas.

	Lepton FLIR 3	**Lepton FLIR 3.5**	**LI-LEPTON-USB2 (*Lepton FLIR 3.5*)**	**MLX90640**
Resolución	160×120	160×120	160×120	32×24
Frame rate	$> 2\ FPS$ (*Eff.*: 8,7 FPS)	$> 2\ FPS$ (*Eff.*: 8,7 FPS)	$> 2\ FPS$ (*Eff.*: 8,7 FPS)	$> 0{,}25\ FPS$ Máx.: 16 FPS
Precio estimado	245 € (antes de 2021) con PureThermal board	390 € con PureThermal board	263 €	63 €

Tabla 3.5: Tabla de precios estimados y especificaciones para el sensor IR utilizado y distintas posibles alternativas [5, 50]

	Raspbery Pi 3*B*+	**Raspberry Pi 5 (4 *GB* RAM)**	**Orange Pi 3*B* (2 *GB* RAM)**	**ROCK64**
CPU	Broadcom $BCM2837B0$ Cortex $A53$ 64-*bit* SoC @$1,4\ GHz$	Broadcom $BCM2712$ Cortex-$A76$ 64-*bit* SoC @$2,4\ GHz$	Rockchip $RK3566$ Cortex-$A55$ 64-*bit* SoC @1,8 GHz	Rockchip $RK3328$ Cortex $A53$ 64-*bit* SoC @1,5 GHz
GPU	Broadcom Videocore-IV	Broadcom Videocore-VII	ARM Mali $G52\ 2EE$	ARM Mali 450 $MP2$
RAM	1 GB LPDDR2 SDRAM	4 GB LPDDR4X SDRAM	2 GB LPDDR4/4X SDRAM	4 GB LPDDR3 SDRAM
Precio estimado	35 € (antes de 2021)	74 €	42 €	54 €

Tabla 3.6: Tabla de precios estimados y especificaciones para el sistema de placa única (SBC) utilizado y distintas posibles alternativas [5, 51–53]

En la Tabla 3.5 se muestra el *LI-LEPTON-USB2* que incluye la placa necesaria para su funcionamiento por un precio inferior a la otra alternativa presentada. Además, el sensor incluido en este producto es un modelo superior al utilizado en este proyecto. En el caso del *MLX90640*, se ofrece una posibilidad más económica pero que no ofrece demasiada versatilidad dada su resolución aún más baja, por lo que ni siquiera sería viable la parte del algoritmo de detección directa. En el caso de sistemas embebidos, en la Tabla 3.6 se muestran como alternativas el modelo superior de *Raspberry Pi* y otros dos sistemas de distinta marca, que por precios similares tienen características a tener en cuenta.

Para el software, tenemos distintas vías de mejoras que pueden ser consideradas. Incorporando una *Raspberry Pi 5* podríamos considerar implementar un programa más complejo que aproveche la mayor capacidad de dicho sistema. Por ejemplo, podríamos comprobar si es viable implementar detección de rostros mediante una CNN (Convolutional Neural Network). También sería posible utilizar otras liberías como *TensorFlow* [54], con la que podríamos probar a reentrenar un modelo de aprendizaje profundo por "transfer learning". Este método consistiría en tomar un modelo de partida, como una CNN entrenado para detección de caras, y reentrenar la última capa con nuevos datos.

Capítulo 4

Conclusiones

El objetivo final de este proyecto era el de medir la temperatura media facial, el cual se ha abordado de forma teórica y práctica a lo largo de todo el documento. A pesar de la capacidad de cómputo limitada del dispositivo se han obtenido tiempos muy favorables, cosa que ha sido posible mediante una gestión óptima de la memoria, liberándola siempre que ha sido posible.

El dispositivo planteado presenta la posibilidad de introducir modificaciones en caso de desearlo, pudiendo incluso incluir nuevas funciones que no se tenían contempladas en primer lugar, aportando flexibilidad para todo tipo de escenarios y aplicaciones. También es posible añadir nuevas funcionalidades al programa desarrollado.

Tras explorar dos vías alternativas de abordar el problema propuesto, se han obtenido unos resultados finales bastante satisfactorios, obteniendo un 80,00 % de detecciones mediante detección indirecta en rostros capturados de frente. Sin embargo, se han encontrado distintos problemas que, si se hubieran dado las condiciones adecuadas, no habrían existido y se habría llegado a mejores resultados. Por ello, se plantean distintas posibilidades a tener en cuenta para mejorar el sistema:

- Entrenar clasificadores en el rango infrarrojo con una base de datos de rostros e imágenes negativas lo suficientemente amplia y variada.
- Entrenar clasificadores en el rango visible que incluyan la posibilidad de portar mascarillas o cobertura total del pelo.
- Mejorar el hardware del sistema y utilizar métodos alternativos que requieran mayor capacidad de cómputo, como por ejemplo, reentrenar una CNN por "transfer learning".
- Aunque el número de detecciones falsas ha sido pequeño, ayuda escoger un entorno adecuado para la detección con ninguna superficie reflectante captada por las cámaras.
- Utilizar un termómetro con menor incertidumbre en la medida para calibrar mejor la temperatura que medirá el sistema.

Bibliografía

[1] "Coronavirus (COVID-19)", *Organización Mundial de la Salud (WHO)* URL: https://www.who.int/es/health-topics/coronavirus Last accessed: Apr. 2023.

[2] "Enfermedad por el virus del Ébola", *Organización Mundial de la Salud (WHO)* (Apr. 20, 2023)
URL: https://www.who.int/es/news-room/fact-sheets/detail/ebola-virus-disease
Last accessed: Apr. 2023.

[3] "Viruela símica", *Organización Mundial de la Salud (WHO)* (Apr. 18, 2023) URL: https://www.who.int/es/news-room/fact-sheets/detail/monkeypox Last accessed: Apr. 2023.

[4] F. García, "Compacidad y densidad de las ciudades españolas", *EURE (Santiago)*, vol. 42, no. 127. Pontificia Universidad Católica de Chile, pp. 5–27, Sep. 2016. doi: 10.4067/s0250-71612016000300001.

[5] J. A. Leñero-Bardallo, R. De la Rosa-Vidal, R. Padial-Allué, J. Ceballos-Cáceres, A. Rodriguez-Vázquez and J. Bernabéu-Wittel, "A Customizable Thermographic Imaging System for Medical Image Acquisition and Processing", *IEEE Sensors Journal*. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–1, 2021. doi: 10.1109/jsen.2021.3080035.

[6] "OpenCV: About" (2023) URL: https://opencv.org/about/ Last accessed: Apr. 2023.

[7] "OpenCV: Repository" (Apr. 27, 2023) URL: https://github.com/opencv/opencv Last accessed: Apr. 2023.

[8] "OpenCV: OpenCV modules" (Dec. 25, 2021) URL: https://docs.opencv.org/4.5.5/ Last accessed: Apr. 2023.

[9] E. F. J. Ring and K. Ammer, "Infrared thermal imaging in medicine", *Physiological Measurement*, vol. 33, no. 3. IOP Publishing, pp. R33–R46, Feb. 28, 2012. doi: 10.1088/0967-3334/33/3/r33.

[10] J. Haglund, F. Jeppsson, E. Melander, A.-M. Pendrill, C. Xie and K. J. Schönborn, "Infrared cameras in science education", *Infrared Physics & Technology*, vol. 75. Elsevier BV, pp. 150–152, Mar. 2016. doi: 10.1016/j.infrared.2015.12.009.

[11] M. Vollmer, "Infrared Thermal Imaging", *Computer Vision: A Reference Guide*, Springer International Publishing, pp. 666–670, 2021. doi: 10.1007/978-3-030-63416-2_844.

[12] J. Stefan, "Uber die Beziehung zwischen der Warmestrahlung und der Temperatur", *Sitzungsber. Kaiserl. Akad. Wiss. Math. Naturwiss.*, Cl. II., Abth. 79, no. 3, pp. 391-428, 1879.

[13] R. Usamentiaga, P. Venegas, J. Guerediaga, L. Vega, J. Molleda and F. G. Bulnes, "Infrared Thermography for Temperature Measurement and Non-Destructive Testing", *Sensors*, vol. 14, no. 7. MDPI AG, pp. 12305–12348, Jul. 10, 2014. doi: 10.3390/s140712305.

[14] J. L. Tissot, C. Trouilleau, B. Fieque, A. Crastes and O. Legras, "Uncooled microbolometer detector: recent developments at ULIS", *Opto-Electronics Review*, vol. 14, no. 1. Polish Academy of Sciences Chancellery, Jan. 01, 2006. doi: 10.2478/s11772-006-0004-2.

[15] A. Kumar, A. Kaur and M. Kumar, "Face detection techniques: a review", *Artificial Intelligence Review*, vol. 52, no. 2. Springer Science and Business Media LLC, pp. 927–948, Aug. 04, 2018. doi: 10.1007/s10462-018-9650-2.

[16] M. Charlton, S. A. Stanley, Z. Whitman, V. Wenn, T. J. Coats, M. Sims and J. P. Thompson, "The effect of constitutive pigmentation on the measured emissivity of human skin", *PLOS ONE*, vol. 15, no. 11. Public Library of Science (PLoS), p. e0241843, Nov. 25, 2020. doi: 10.1371/journal.pone.0241843.

[17] E. Hjelmås and B. K. Low, "Face Detection: A Survey", *Computer Vision and Image Understanding*, vol. 83, no. 3. Elsevier BV, pp. 236–274, Sep. 2001. doi: 10.1006/cviu.2001.0921.

[18] W. Zhao, R. Chellappa, P. J. Phillips and A. Rosenfeld, "Face Recognition: A Literature Survey", *ACM Computing Surveys*, vol. 35, no. 4. Association for Computing Machinery (ACM), pp. 399–458, Dec. 2003. doi: 10.1145/954339.954342.

[19] A. Lapedes and R. Farber, "How Neural Nets Work", *Neural Information Processing Systems*, American Institute of Physics, pp. 442-456, 1987. URL: https://proceedings.neurips.cc/paper_files/paper/1987/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf

[20] H. Wang, Z. Li, X. Ji and Y. Wang, "Face R-CNN", *arXiv*, 2017. doi: 10.48550/ARXIV.1706.01061.

[21] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1. Institute of Electrical and Electronics Engineers (IEEE), pp. 23–38, 1998. doi: 10.1109/34.655647.

[22] A. Rao and S. Noushath, "Subspace methods for face recognition", *Computer Science Review*, vol. 4, no. 1. Elsevier BV, pp. 1–17, Feb. 2010. doi: 10.1016/j.cosrev.2009.11.003.

[23] F. Anowar, S. Sadaoui, and B. Selim, "Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE)", *Computer Science Review*, vol. 40, Elsevier BV, p. 100378, May 2021. doi: 10.1016/j.cosrev.2021.100378.

[24] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space", *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11. Informa UK Limited, pp. 559–572, Nov. 1901. doi: 10.1080/14786440109462720.

[25] R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems", *Annals of Eugenics*, vol. 7, no. 2. Wiley, pp. 179–188, Sep. 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x.

[26] J. Herault and C. Jutten, "Space or time adaptive signal processing by neural network models", *AIP Conference Proceedings*, vol. 151, AIP, pp. 206-211, 1986. doi: 10.1063/1.36258.

[27] L. E. Di Persia, "Separación ciega de fuentes sonoras: revisión histórica y desarrollos recientes", *Academia Nacional de Ciencias Exactas, Físicas y Naturales*, vol. 68, Anales de la Academia Nacional de Ciencias Exactas, Físicas y Naturales de Buenos Aires, pp. 45-62, May 2017. URL: https://ri.conicet.gov.ar/handle/11336/47060

[28] G. Sahonero-Alvarez, and H. Calderon. .ª comparison of SOBI, FastICA, JADE and Infomax algorithms", *Proceedings of the 8th International Multi-Conference on Complexity*, Informatics and Cybernetics, pp. 17-22, 2017. URL: https://www.imt.ucb.edu.bo/documents/publications/A-Comparison-of-SOBI-Fast ICA-JADE-and-Infomax-Algorithms.pdf

[29] X. He and P. Niyogi, "Locality Preserving Projections", *Advances in Neural Information Processing Systems*, vol. 16, MIT Press, 2003. URL: https://proceedings.neurips.cc/paper_files/paper/2003/file/d69116f8b0140cdeb1f99a 4d5096ffe4-Paper.pdf

[30] V. N. Vapnik and A. Y. Chervonenkis, "A class of algorithms for pattern recognition learning", *Avtomatika i Telemekhanika*, vol. 25, pp. 937–945, 1964. URL: http://mi.mathnet.ru/at11678

[31] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers", *Proceedings of the fifth annual workshop on Computational learning theory*, ACM, Jul. 1992. doi: 10.1145/130385.130401.

[32] C. Cortes and V. Vapnik, "Support-vector networks", *Machine Learning*, vol. 20, no. 3. Springer Science and Business Media LLC, pp. 273–297, Sep. 1995. doi: 10.1007/bf00994018.

[33] N. Cristianini and E. Ricci, "Support Vector Machines", *Encyclopedia of Algorithms*. Springer US, pp. 928–932, 2008. doi: 10.1007/978-0-387-30162-4_415.

[34] V. P. Kshirsagar, M. R. Baviskar, and M. E. Gaikwad, "Face recognition using Eigenfaces", *2011 3rd International Conference on Computer Research and Development*. IEEE, Mar. 2011. doi: 10.1109/iccrd.2011.5764137.

[35] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7. Institute of Electrical and Electronics Engineers (IEEE), pp. 711–720, Jul. 1997. doi: 10.1109/34.598228.

[36] D. Wahyuningsih, C. Kirana, R. Sulaiman, Hamidah, and Triwanto, "Comparison Of The Performance Of Eigenface And Fisherface Algorithm In The Face Recognition Process", *2019 7th International Conference on Cyber and IT Service Management (CITSM)*. IEEE, Nov. 2019. doi: 10.1109/citsm47753.2019.8965345.

[37] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active Shape Models-Their Training and Application", *Computer Vision and Image Understanding*, vol. 61, no. 1. Elsevier BV, pp. 38–59, Jan. 1995. doi: 10.1006/cviu.1995.1004.

ic-thresholding.pdf

[38] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1. Institute of Electrical and Electronics Engineers (IEEE), pp. 62–66, Jan. 1979. doi: 10.1109/tsmc.1979.4310076.

[39] S. Zhou, P. Yang and W. Xie, "Infrared image segmentation based on Otsu and genetic algorithm", *2011 International Conference on Multimedia Technology*. IEEE, Jul. 2011. doi: 10.1109/icmt.2011.6003109.

[40] T. Ojala, M. Pietikainen and D. Harwood, "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions", *Proceedings of 12th International Conference on Pattern Recognition*. IEEE Comput. Soc. Press. doi: 10.1109/icpr.1994.576366.

[41] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions", Pattern Recognition, vol. 29, no. 1. Elsevier BV, pp. 51–59, Jan. 1996. doi: 10.1016/0031-3203(95)00067-4.

[42] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7. Institute of Electrical and Electronics Engineers (IEEE), pp. 971–987, Jul. 2002. doi: 10.1109/tpami.2002.1017623.

[43] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. CVPR 2001. IEEE Comput. Soc. doi: 10.1109/cvpr.2001.990517.

[44] P. Viola and M. J. Jones, "Robust Real-Time Face Detection", *International Journal of Computer Vision*, vol. 57, no. 2. Springer Science and Business Media LLC, pp. 137–154, May 2004. doi: 10.1023/b:visi.0000013087.49260.fb.

[45] K. Kadir, M. K. Kamaruddin, H. Nasir, S. I. Safie, and Z. A. K. Bakti, "A comparative study between LBP and Haar-like features for Face Detection using OpenCV", *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*. IEEE, Aug. 2014. doi: 10.1109/ice2t.2014.7006273.

[46] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting", Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 23–37, 1996. doi: 10.1007/3-540-59119-2_166.

[47] W. Gao and Z.-H. Zhou, "On the doubt about margin explanation of boosting", *Artificial Intelligence*, vol. 203. Elsevier BV, pp. 1–18, Oct. 2013. doi: 10.1016/j.artint.2013.07.002.

[48] R. Miezianko, "Terravic Research Infrared Database," *IEEE OTCBVS WS Series Bench*. URL: https://vcipl-okstate.org/pbvs/bench/Data/04/download.html Last accessed: Aug. 2023.

[49] J. Liu, X. Fan, Z. Huang, G. Wu, R. Liu, W. Zhong and Z. Luo, "Target-aware Dual Adversarial Learning and a Multi-scenario Multi-Modality Benchmark to Fuse Infrared and Visible for Object Detection", *arXiv*, 2022. doi: 10.48550/ARXIV.2203.16220.

[50] "Mouser electronics". URL: https://www.mouser.es Last accessed: Oct. 2023.

[51] "Raspberry Pi 5", *Raspberry Pi Foundation*.
URL: https://www.raspberrypi.com/products/raspberry-pi-5/
Last accessed: Oct. 2023.

[52] "Orange Pi 3B", *Orange Pi*.
URL: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-3B.html Last accessed: Oct. 2023.

[53] "ROCK64-4GB Single Board Computer", *Pine64*.
URL: https://pine64.com/product/rock64-4gb-single-board-computer/
Last accessed: Oct. 2023.

[54] "TensorFlow". URL: https://www.tensorflow.org/ Last accessed: Oct. 2023.

ANEXO I: Código del programa

```
//============================================================================
// Name        : FaceTemp.cpp
// Author      : Adrián Cordones Martínez
// Version     : 1.0
// Description : Program created for my 'FDP' on my Physics grade in the
//               University of Seville (US). Type "./FaceTemp --help" for more
//               information about this program.
//============================================================================

#include <iostream>
#include <iomanip>
#include <unistd.h>
#include <string>
#include <sstream>

#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/freetype.hpp"

// OpenCV external UI (CVUI):
#define CVUI_IMPLEMENTATION
#include "cvui.h"

#include "wiringPi.h"
#include "Calib.hpp"

using namespace std;
using namespace cv;

// Custom namespace (from "Calib.hpp") for camera calibration:
using namespace cd;
// Custom class (from "Calib.hpp") for assigning Cameras IDs:
CameraDetector CD;

// Error values:
#define       CASCADE_ERROR   -1 // Could not load classifier cascade.
#define     NOIRIMAGE_ERROR   -2 // Could not read IR image.
#define   NOIRCAPTURE_ERROR   -3 // Capture from IR camera didn't work.
#define    NOVISIMAGE_ERROR   -4 // Could not read Visible image.
#define  NOVISCAPTURE_ERROR   -5 // Capture from Visible camera didn't work.
#define        ORIGIN_ERROR   -6 // Images must have the same origin (capturing from cameras or external images).
#define      EMPTYVIS_ERROR   -7 // Empty image in the visible range when capturing from camera.
#define       EMPTYIR_ERROR   -8 // Empty image in the IR range when capturing from camera.

// Temperature-IR calibration values:
// This calibration may change due to ambient temperature.
#define T1  36.6
#define T2  36.9
#define R1  8418.8
#define R2  11348.92
#define IRSensorTo8Bit  16383.0/255

// Temperature bounds:
#define LOWER_TEMP_THRES      37.5 //Slight fever
#define HIGHER_TEMP_THRES     38.3 //Fever

// Remapping factors:
```

```
#define scaleFactor  3.27
#define     xFactor  112.2
#define     yFactor  34.1

// Use "./FaceTemp --help" for more information about this program.
static void help(const char** argv)
{
    cout << "\nThe main purpose of this program is detecting faces and"
        " measuring their mean temperature using OpenCV library.\n"
        "This program uses the cv::CascadeClassifier class to"
        " detect objects (faces and eyes) using Haar features.\n"
            "\n"
        "There are two forms of operating:\n"
        "a) Using only one image on the IR light range and detecting"
        " the faces and measuring their mean temperature.\n"
        "b) Using two images, one on the visible light range to detect"
        " faces and another on the IR light range for measuring\n  "
        " the mean temperature by remapping the detected objects."
        " By using this mode, visible camera is used as a preview.\n"
            "\n"
        "Usage:\n"
        <<  argv[0]
        <<  "\n"
        "   @IR_input          <IR_file_or_cam_ID>    "
        "   -- IR range: Camera index or Image path.\n"

        "   @Vis_input         <Vis_file_or_cam_ID>   "
        "   -- Visible range: Camera index or Image path.\n"

        "   --cascade          = <cascade_path>       "
        "   -- Primary trained classifier (frontal face).\n"

        "   --nested-cascade   = <nested_cascade_path>"
        "   -- Optional secondary classifier (eyes)"
        " [only for visible range].\n"

        "   --pathToSave       = <save_path>          "
        "   -- Folder where the exported images are saved."
        " Default: data/Images\n"

        "   --IR_scale         = <IR_img_scale>       "
        "   -- Final IR Image scale factor. Default: 1.\n"

        "   --Vis_scale        = <Vis_img_scale>      "
        "   -- Final Vis. Image scale factor. Default: 1.\n"

        "   --try-flip                                "
        "   -- Use this when you want to search for inverted faces.\n"

        "   --remapping                               "
        "   -- Use this if you want to use the remapping method.\n"
        "\n"
        "When remapping, images must have the same origin"
        " (capturing from cameras or external images).\n"
        "\n"
        "Example:\n"
        <<  argv[0]
        <<  " --cascade=/usr/local/share/opencv4/haarcascades/"
        "haarcascade_frontalface_alt.xml --nested-cascade=/usr/local/"
        "share/opencv4/haarcascades/haarcascade_eye_tree_eyeglasses.xml"
        " --pathToSave=/home/pi/Documents/exportedImagesFromOpenCV/ "
        "--IR_scale=4 --try-flip --remapping\n"
        "\n"
        "During execution:\n"
        "\t- Press 'S' to save all images.\n"
        "\t- Press any other key to exit.\n"
        "\n"
        "Using OpenCV version " << CV_VERSION << "\n" << endl;
}

// Functions declaration:
void DisplayHelp(const char** argv);
const string currentDateTime();
```

```
void remapper(vector<Rect> faces, vector<Rect>& newFaces, Mat img, double scF, double xF, double yF);
void nestedRemapper(vector<vector<Rect>> nestedObjects, vector<vector<Rect>>& newNestedObjects,
                    Mat img, double scF);
void detectObjects(Mat& gray, CascadeClassifier& cascade, CascadeClassifier& nestedCascade, bool tryflip,
                    vector<Rect>& faces, vector<vector<Rect>>& nestedObjects);
void drawingTool(Mat& img, vector<Rect> faces, vector<vector<Rect>> nestedObjects, Scalar color,
                double temperature, double scale);
void drawText(Mat& img, vector<Rect> faces, Scalar color,
                double temperature, double scale);
double IRtoTempComversion(const double A1, const double A2, const double B1, const double B2, const double IR);
double GetTempFromIRImage(const Mat& Image, const Rect& ROI, double& otsuThreshold, Mat& ImgROI,
                Mat& ImgROISegmented, Mat& Mask);
void Thermometer(Mat img, Mat& gray, vector<Rect> faces, double& meanTemp, double scale);
void getColor(const double& highThreshold, const double& lowerThreshold,
                const double& temperature, Scalar& color);
void saveData(Mat img1, Mat img2, Mat IR_image, Mat Vis_image, string savedImagesPath, bool remapping_mode);

int main(int argc, const char** argv)
{
    // Input arguments declaration:
    string cascadeName, nestedCascadeName;
    string IR_inputName, Vis_inputName, savedImagesPath;
    double IR_scale, Vis_scale;
    bool tryflip, remapping_mode;

    // Video capture, frames and images:
    VideoCapture IR_capture, Vis_capture;
    Mat IR_image, Vis_image;

    // Cascades Classifiers (Haar features):
    CascadeClassifier cascade, nestedCascade;

    char c; // 'Save' and 'Exit' key

    cv::CommandLineParser parser(argc, argv,
        "{help h||}"
        "{@IR_input||}"
        "{@Vis_input||}"
        "{cascade||}"
        "{nested-cascade||}"
        "{pathToSave|data/Images/|}"
        "{IR_scale|1|}"
        "{Vis_scale|1|}"
        "{try-flip||}"
        "{remapping||}"
    );

    if (parser.has("help"))
    {
        help(argv);
        return 0;
    }

    // Call for input arguments:
    cascadeName = parser.get<string>("cascade");
    nestedCascadeName = parser.get<string>("nested-cascade");
    IR_scale = parser.get<double>("IR_scale");
    Vis_scale = parser.get<double>("Vis_scale");
    tryflip = parser.has("try-flip");
    remapping_mode = parser.has("remapping");
    IR_inputName = parser.get<string>("@IR_input");
    Vis_inputName = parser.get<string>("@Vis_input");
    savedImagesPath = parser.get<string>("pathToSave");

    if (!parser.check())
    {
        parser.printErrors();
        return 0;
    }

    // Checking for ERRORS and WARNINGS:
    if(nestedCascadeName.empty() || !nestedCascade.load(samples::findFileOrKeep(nestedCascadeName,1)))
        cout << "[WARN]: Could not load classifier cascade for nested objects." << endl;
```

```
if(cascadeName.empty() || !cascade.load(samples::findFileOrKeep(cascadeName,1)))
{
    cerr << "ERROR (" << CASCADE_ERROR << "): Could not load classifier cascade." << endl;
    DisplayHelp(argv);

    return CASCADE_ERROR;
}

// We must check if IR and Visible images have the same origin
// (capturing from cameras or external images, but not alternating both when remapping):
bool IR_type, Vis_type;

if(IR_inputName.empty() || (isdigit(IR_inputName[0]) && IR_inputName.size() == 1))
{
    IR_type = 1; // IR camera

    // If @IR_filename is empty, search for IR camera:
    int IR_index = IR_inputName.empty() ? CD.DetectCameraID(CameraType::IR) : IR_inputName[0] - '0';

    if(!IR_capture.open(IR_index))
    {
        cerr << "ERROR (" << NOIRCAPTURE_ERROR << "): Capture from IR camera (#"
        <<  IR_index << ") didn't work." << endl;
        DisplayHelp(argv);

        return NOIRCAPTURE_ERROR;
    }
}
else
{
    IR_type = 0; // IR image
    IR_image = imread(samples::findFileOrKeep(IR_inputName,1), IMREAD_COLOR);
    if(IR_image.empty())
    {
        if(!IR_capture.open(samples::findFileOrKeep(IR_inputName,1)))
        {
            cerr << "ERROR (" << NOIRIMAGE_ERROR << "): Could not read IR image: "
            << IR_inputName << endl;
            DisplayHelp(argv);

            return NOIRIMAGE_ERROR;
        }
    }
}
if(remapping_mode)
{
    if(Vis_inputName.empty() || (isdigit(Vis_inputName[0]) && Vis_inputName.size() == 1))
    {
        Vis_type = 1; // Visible camera

        // If @Vis_filename is empty, search for Visible camera:
        int Vis_index = Vis_inputName.empty() ? CD.DetectCameraID(CameraType::Visible) :
                        Vis_inputName[0] - '0';

        if(!Vis_capture.open(Vis_index  ))
        {
            cerr << "ERROR (" << NOVISCAPTURE_ERROR << "): Capture from Visible camera (#"
            <<  Vis_index << ") didn't work." << endl;
            DisplayHelp(argv);

            return NOVISCAPTURE_ERROR;
        }
    }
    else
    {
        Vis_type = 0; // Visible image
        Vis_image = imread(samples::findFileOrKeep(Vis_inputName,1), IMREAD_COLOR);
        if(Vis_image.empty())
        {
            if(!Vis_capture.open(samples::findFileOrKeep(Vis_inputName,1)))
            {
                cerr << "ERROR (" << NOVISIMAGE_ERROR << "): Could not read Visible image: "
                << Vis_inputName << endl;
```

```
                DisplayHelp(argv);

                return NOVISIMAGE_ERROR;
            }
        }
    }

    if(IR_type != Vis_type)
    {
        cerr << "ERROR (" << ORIGIN_ERROR <<
                "): Images must have the same origin (capturing from cameras or external images)." << endl;
        cout << "\t    Alternating both have no sense because images will not be correlated!" << endl;
        DisplayHelp(argv);

        return ORIGIN_ERROR;
    }
}

// PREVIEW mode:
if(remapping_mode && Vis_type)
{
    cout << "\nStarting preview, press 'SAVE DATA' to start.\nPress 'ESC' or 'Q' to exit preview mode."
         << endl;

    // Loading picture from camera:
    Mat image;
    Vis_capture.read(image);

    // Create a frame where UI components will be rendered to:
    Mat frame;

    // CONTROL VALUES ADJUSTED MANUALLY...
    // Initialize arguments for the UI window:
    int top = 0.025*image.rows;
    int left = 0.025*image.cols;
    int right = left;
    int bottom = 0.15*image.rows; // Botton margin
    int borderType = BORDER_CONSTANT;
    Scalar value(237, 237, 237); // Window color
    // Button parameters:
    int s_x = 0.25*image.cols;
    int s_y = 0.075*image.rows;
    int x = 0.5*(image.cols + left + right) - s_x/2;
    int y = 1.1*image.rows - s_y/2;
    double button_font_size = 0.5;
    unsigned int inside_color = 0xFF87919B;
    string button_name = "SAVE DATA";

    // Init CVUI and tell it to create a OpenCV window, i.e. cv::namedWindow(WINDOW_NAME):
    string win_name = "FaceTemp (Preview)";
    cvui::init(win_name);

    for(;;)
    {
        Vis_capture.read(image);

        // Create frame border:
        copyMakeBorder(image, frame, top, bottom, left, right, borderType, value);

        if (cvui::button(frame, x, y, s_x, s_y, button_name, button_font_size, inside_color)
            || c == 's' || c == 'S')
        {
            break;
        }

        cvui::imshow(win_name, frame);

        c = waitKey(20);
        if(c == 27 || c == 'q' || c == 'Q')
        {
            exit(0);
        }
    }
```

```
    destroyAllWindows();

    // Release memory:
    image.release();
    frame.release();
}

// Needed variables:
vector<Rect> faces;
vector<vector<Rect>> nestedObjects;
Mat img1, img2, gray;
double meanTemp;
Scalar color;

// Program initialization:
if(IR_capture.isOpened())
{
    cout << "\nTaking facial picture, stay still and look at camera(s)..." << endl;

    // Take picture(s):
    IR_capture.read(IR_image);
    if(remapping_mode)
    {
        Vis_capture.read(Vis_image);

        if(Vis_image.empty())
        {
            cout << "ERROR (" << EMPTYVIS_ERROR
                 << "): Empty image in the visible range when capturing from camera." << endl;
            return EMPTYVIS_ERROR;
        }
    }
    if(IR_image.empty())
    {
        cout << "ERROR ("
             << EMPTYIR_ERROR << "): Empty image in the IR range when capturing from camera." << endl;
        return EMPTYIR_ERROR;
    }
}
else
{
    if(remapping_mode)
        cout << "\nDetecting face(s) from " << Vis_inputName << endl;
    else
        cout << "\nDetecting face(s) from " << IR_inputName << endl;
}

img1 = IR_image.clone();

if(remapping_mode)
{
    img2 = Vis_image.clone();

    // Convert Vis. space color to Gray Scale:
    cvtColor(img2, gray, COLOR_BGR2GRAY);
    equalizeHist(gray, gray);

    // Detecting faces in the Visible light range:
    detectObjects(gray, cascade, nestedCascade, tryflip, faces, nestedObjects);

    vector<Rect> newFaces;
    vector<vector<Rect>> newNestedObjects;

    // Remapping faces and eyes:
    remapper(faces, newFaces, img1, scaleFactor, xFactor, yFactor);
    nestedRemapper(nestedObjects, newNestedObjects, img1, scaleFactor);

    // Convert IR space color to Gray Scale:
    cvtColor(img1, gray, COLOR_BGR2GRAY);

    // Measure mean temperature from faces:
    Thermometer(img1, gray, newFaces, meanTemp, IR_scale);
```

```
        // Color for Drawing Tool:
        getColor(HIGHER_TEMP_THRES, LOWER_TEMP_THRES, meanTemp, color);

        // Resize and draw objects for Vis. image:
        resize(img2, img2, Size(), Vis_scale, Vis_scale, INTER_LINEAR);
        drawingTool(img2, faces, nestedObjects, color, meanTemp, Vis_scale);
        drawText(img2, faces, color, meanTemp, Vis_scale);

        // Resize and draw objects for IR image:
        resize(img1, img1, Size(), IR_scale, IR_scale, INTER_LINEAR);
        drawingTool(img1, newFaces, newNestedObjects, color, meanTemp, IR_scale);
        drawText(img1, newFaces, color, meanTemp, IR_scale);

        // Release memory:
        newFaces.clear();
        newNestedObjects.clear();
    }
    else
    {
        // Convert IR space color to Gray Scale:
        cvtColor(img1, gray, COLOR_BGR2GRAY);
        equalizeHist(gray, gray);

        // Detect faces in the IR light range:
        detectObjects(gray, cascade, nestedCascade, tryflip, faces, nestedObjects);

        // Measure mean temperature from faces:
        Thermometer(img1, gray, faces, meanTemp, IR_scale);

        // Color for Drawing Tool:
        getColor(HIGHER_TEMP_THRES, LOWER_TEMP_THRES, meanTemp, color);

        // Resize and draw objects for IR image:
        resize(img1, img1, Size(), IR_scale, IR_scale, INTER_LINEAR);
        drawingTool(img1, faces, nestedObjects, color, meanTemp, IR_scale);
        drawText(img1, faces, color, meanTemp, IR_scale);
    }

    // Release memory:
    gray.release();
    faces.clear();
    nestedObjects.clear();

    cout << "\t- Press 'S' to save all images.\n\t- Press any other key to exit.\n" << endl;

    imshow("IR Image", img1);
    if(remapping_mode)
        imshow("Visible Image", img2);

    c = waitKey(0);

    if(c == 's' || c == 'S')
        saveData(img1, img2, IR_image, Vis_image, savedImagesPath, remapping_mode);

    // Close all windows:
    destroyAllWindows();

    // Release images from memory:
    IR_image.release();
    Vis_image.release();
    img1.release();
    img2.release();

    // Release Video capture:
    IR_capture.release();
    Vis_capture.release();

    return 0;
}

// Functions definition:
void DisplayHelp(const char** argv)
```

```
{
    cout << "\nPlease, make sure you have introduced the correct arguments.
             This is how this program works..." << endl;
    help(argv);
}

const string currentDateTime()
{
    time_t     now = time(0);
    struct tm  tstruct;
    char       buf[80];
    tstruct = *localtime(&now);
    // Visit http://en.cppreference.com/w/cpp/chrono/c/strftime
    // for more information about date/time format
    strftime(buf, sizeof(buf), "%d-%m-%Y.%H-%M-%S", &tstruct);

    return buf;
}

void remapper(vector<Rect> faces, vector<Rect>& newFaces, Mat img, double scF, double xF, double yF)
{
    Rect face, newFace;

    for(size_t i = 0; i < faces.size(); i++)
    {
        face = faces[i];

        cout << "\nFace #" << i+1 << " detected at: x = " << face.x << "; y = " << face.y
             << "; height = " << face.height << "; width = " << face.width << endl;

        // Translation movement:
        newFace.x = cvRound((face.x - xF + 1) / scF);
        newFace.y = cvRound((face.y - yF + 1) / scF);

        // Remapping scale factor:
        newFace.width = cvRound(face.width / scF);
        newFace.height = cvRound(face.height / scF);

        // Check if part of the face is outside the image and correct it:
        if(newFace.x < 0)
            newFace.x = 0;
        if(newFace.y < 0)
            newFace.y = 0;
        if((newFace.x + newFace.width-1) > img.cols)
            newFace.width = img.cols - newFace.x;
        if((newFace.y + newFace.height-1) > img.rows)
            newFace.height = img.rows - newFace.y;

        cout << "Remapping to: x' = " << newFace.x << "; y' = " << newFace.y << "; height' = "
             << newFace.height << "; width' = " << newFace.width << endl;

        newFaces.push_back(newFace);
    }
}

void nestedRemapper(vector<vector<Rect>> nestedObjects, vector<vector<Rect>>& newNestedObjects,
                    Mat img, double scF)
{
    vector<Rect> eyes, newEyes;
    Rect eye, newEye;

    if(!nestedObjects.empty())
    {
        for(size_t i = 0; i < nestedObjects.size(); i++)
        {
            eyes = nestedObjects[i];

            if(eyes.empty())
            {
                newNestedObjects.push_back(eyes);
                continue;
            }
```

```
            for(size_t j = 0; j < eyes.size(); j++)
            {
                eye = eyes[j];

                // Remapping scale factor:
                newEye.x = cvRound(eye.x / scF);
                newEye.y = cvRound(eye.y / scF);
                newEye.width = cvRound(eye.width / scF);
                newEye.height = cvRound(eye.height / scF);

                // Check if part of the eye is outside of the image and correct it:
                if(newEye.x < 0)
                    newEye.x = 0;
                if(newEye.y < 0)
                    newEye.y = 0;
                if((newEye.x + newEye.width-1) > img.cols)
                    newEye.width = img.cols - newEye.x;
                if((newEye.y + newEye.height-1) > img.rows)
                    newEye.height = img.rows - newEye.y;

                newEyes.push_back(newEye);
            }

            newNestedObjects.push_back(newEyes);
        }
    }

    // Release memory for objects that will not be used anymore:
    eyes.clear();
    newEyes.clear();
}

void detectObjects(Mat& gray, CascadeClassifier& cascade, CascadeClassifier& nestedCascade, bool tryflip,
                   vector<Rect>& faces, vector<vector<Rect>>& nestedObjects)
{
    // Init time (in ticks):
    double t = (double)getTickCount();

    // Detect faces of different sizes using cascade classifier:
    cascade.detectMultiScale(gray, faces, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30));

    if(tryflip)
    {
        vector<Rect> faces2;

        flip(gray, gray, 0); // Vertical flip (across x-axis)
        cascade.detectMultiScale(gray, faces2, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30));

        for(vector<Rect>::const_iterator r = faces2.begin(); r != faces2.end(); r++)
        {
            faces.push_back(Rect(r->x, gray.rows - r->y - r->height, r->width, r->height));
        }

        // Release memory:
        faces2.clear();

        // Turn back flip (across x-axis)
        flip(gray, gray, 0);
    }

    Mat grayROI;
    vector<Rect> nfaces;

    // Delete false detections:
    for(size_t i = 0; i < faces.size(); i++)
    {
        Rect r = faces[i];
        double correctionFactor = 0.25 * gray.rows;

        if(r.width < correctionFactor || r.height < correctionFactor)
            continue;

        nfaces.push_back(r);
```

```
    }

    faces = nfaces;

    // Release memory:
    nfaces.clear();

    if(!nestedCascade.empty())
    {

        // Detect eyes:
        for(size_t i = 0; i < faces.size(); i++)
        {
            Rect r = faces[i];
            vector<Rect> eyes;
            grayROI = gray(r);

            // Detection of eyes for every face:
            nestedCascade.detectMultiScale(grayROI, eyes, 1.1, 2, 0|CASCADE_SCALE_IMAGE, Size(30, 30));

            nestedObjects.push_back(eyes);
        }
    }
    // Release memory for objects that will not be used anymore:
    grayROI.release();

    // Detection time (in seconds):
    t = ((double)getTickCount() - t)/getTickFrequency();
    cout << "Elapsed time in detection = " << t << " s" << endl;
}

void drawingTool(Mat& img, vector<Rect> faces, vector<vector<Rect>> nestedObjects, Scalar color,
                 double temperature, double scale)
{
    double linewidth = 0.009 * img.rows;
    double correctionFactor = 10 * linewidth;

    for(size_t i = 0; i < faces.size(); i++)
    {
        Rect r = faces[i];

        // Draw rectangles around faces:
        rectangle(img, Point(cvRound(r.x*scale), cvRound(r.y*scale)), Point(cvRound((r.x + r.width-1)*scale),
                    cvRound((r.y + r.height-1)*scale)), color, linewidth, 8, 0);

        vector<Rect> eyes = nestedObjects[i];
        if(eyes.empty() || nestedObjects.empty())
            continue;

        Point eyeCenter, x1, x2, y1, y2;
        int radius;

        // Draw "+" on eyes for every face:
        for(size_t j = 0; j < eyes.size(); j++)
        {
            Rect nr = eyes[j];

            eyeCenter.x = cvRound((r.x + nr.x-1 + (nr.width-1)*0.5)*scale);
            eyeCenter.y = cvRound((r.y + nr.y-1 + (nr.height-1)*0.5)*scale);
            radius = cvRound((nr.width + nr.height)*0.0625*scale);

            // CORRECTION: when the light is not optimal, sometimes an eye is detected two times.
            if(abs(eyeCenter.x - eyes[j-1].x) < correctionFactor &&
               abs(eyeCenter.y - eyes[j-1].y) < correctionFactor)
                continue;
            if(abs(eyeCenter.x - eyes[j-2].x) < correctionFactor &&
               abs(eyeCenter.y - eyes[j-2].y) < correctionFactor)
                continue;

            // Point X1:
            x1.x = eyeCenter.x - radius;
            x1.y = eyeCenter.y;
            // Point X2:
```

```
            x2.x = eyeCenter.x + radius;
            x2.y = eyeCenter.y;
            // Point Y1:
            y1.x = eyeCenter.x;
            y1.y = eyeCenter.y - radius;
            // Point Y2:
            y2.x = eyeCenter.x;
            y2.y = eyeCenter.y + radius;

            // Draw lines:
            line(img, x1, x2, color, linewidth, 8, 0);
            line(img, y1, y2, color, linewidth, 8, 0);
        }
    }
}

void drawText(Mat& img, vector<Rect> faces, Scalar color, double temperature, double scale)
{
    double linewidth = 0.009 * img.rows;

    for(size_t i = 0; i < faces.size(); i++)
    {
        if(faces.size() > 1)
            break;

        Rect r = faces[i];
        r.x = r.x * scale;
        r.y = r.y * scale;
        r.width = r.width * scale;
        r.height = r.height * scale;

        // Draw temperature:
        stringstream ssTemp;
        ssTemp << std::fixed << std::setprecision(2) << temperature << "  C";
        Point textPoint;
        textPoint.x = cvRound(r.x + (r.width - 1)*0.5 - r.width*0.23);
        textPoint.y = cvRound(r.y + (r.height - 1) + r.height/7);
        putText(img, ssTemp.str(), textPoint, FONT_HERSHEY_DUPLEX, linewidth/6, color, 0.5*linewidth, 8, 0);

        // Draw degree symbol:
        Point degree;
        degree.x = cvRound(textPoint.x + img.rows*41/240);
        degree.y = cvRound(textPoint.y - img.cols/50);
        circle(img, degree, linewidth, color, 0.5*linewidth, 8, 0);

    }
}

double IRtoTempComversion(const double A1, const double A2, const double B1, const double B2, const double IR)
{

    double a, b, result;

    a = (A2-A1)/(B2-B1);
    b = A2 - a*B2;

    result = IR*IRSensorTo8Bit*a + b;

    return result;
}

double GetTempFromIRImage(const Mat& Image, const Rect& ROI, double& otsuThreshold, Mat& ImgROI,
                          Mat& ImgROISegmented, Mat& Mask)
{
    double IR, temperature, threshold;
    ImgROI = Image(ROI);

    // Obtain the Otsu's threshold:
    otsuThreshold = cv::threshold(ImgROI, ImgROISegmented, 0, 255, THRESH_TOZERO|THRESH_OTSU);
    cout << "\nOTSU's Threshold = " << otsuThreshold << endl;
    // Binary segmentation using Otsu's threshold:
    cv::threshold(ImgROI, Mask, otsuThreshold, 255, THRESH_BINARY);
```

```
    // Compute ROI mean:
    IR = cv::mean(ImgROI, Mask)[0];
    cout << "IR = " << IR << "\n" << endl;

    // IR to temperature conversion:
    temperature = IRtoTempComversion(T1, T2, R1, R2, IR);

    return temperature;
}

void Thermometer(Mat img, Mat& gray, vector<Rect> faces, double& meanTemp, double scale)
{
    Mat ImgROI, ImgROISegmented, Mask;
    double thresholdSeg;

    for(size_t i = 0; i < faces.size(); i++)
    {
        Rect r = faces[i];

        // Measuring Temperature from face [i]:
        meanTemp = GetTempFromIRImage(gray, r, thresholdSeg, ImgROI, ImgROISegmented, Mask);
        cout << "The temperature for the face #" << i+1 << " is: T(Â°C) = " << meanTemp << endl;

        // Resize all processed IR images:
        resize(ImgROI, ImgROI, Size(), scale, scale, INTER_LINEAR);
        resize(ImgROISegmented, ImgROISegmented, Size(), scale, scale, INTER_LINEAR);
        resize(Mask, Mask, Size(), scale, scale, INTER_LINEAR);

        stringstream ssNum;
        ssNum << i+1;

        string ROI_name = "ROI (face #" + ssNum.str() + ")";
        string OTSU_name = "OTSU (face #" + ssNum.str() + ")";
        string Mask_name = "Mask (face #" + ssNum.str() + ")";

        imshow(ROI_name, ImgROI);
        imshow(OTSU_name, ImgROISegmented);
        imshow(Mask_name, Mask);
    }
    cout << endl;

    // Release memory:
    ImgROI.release();
    ImgROISegmented.release();
    Mask.release();
}

void getColor(const double& highThreshold, const double& lowerThreshold,
                const double& temperature, Scalar& color)
{
    if(temperature >= highThreshold)
        // Red when fever:
        color = CV_RGB(255,0,0);
    else if(temperature >= lowerThreshold)
        // Orange when slight fever:
        color = CV_RGB(255,153,51);
    else
        // Green when normal:
        color = CV_RGB(0,255,0);

}

void saveData(Mat img1, Mat img2, Mat IR_image, Mat Vis_image, string savedImagesPath, bool remapping_mode)
{
    string timeStamp = currentDateTime();
    string filename = savedImagesPath + "IR_raw_image_" + timeStamp + ".jpg";
    imwrite(filename, IR_image);

    if(remapping_mode)
    {
        filename = savedImagesPath + "Vis_raw_image_" + timeStamp + ".jpg";
        imwrite(filename, Vis_image);
```

```
        filename = savedImagesPath + "Vis_FaceDetect_image_" + timeStamp + ".jpg";
        imwrite(filename, img2);
    }

    filename = savedImagesPath + "IR_FaceDetect_image_" + timeStamp + ".jpg";
    imwrite(filename, img1);
}
```

ANEXO II: Clases y funciones relevantes de OpenCV

Lista 1. Clases relevantes de *OpenCV* [6–8]:

- cv::**Mat**. "Array numérico denso n-dimensional de uno o múltiples canales. Permite almacenar vectores y matrices de valores reales o complejos, imágenes en escala en color o escala de grises, volúmenes de píxeles, tensores, histogramas, etc".
- cv::**VideoCapture**. "Clase para la captura de vídeo desde archivos de vídeo, secuencias de imágenes o cámaras. La clase proporciona una API en C++ para capturar vídeo desde cámaras o para leer archivos de vídeo y secuencias de imágenes".
- cv::**CascadeClassifier**. "Clase para clasificadores en cascada destinados a la detección de objetos".
- cv::**Rect**. "Clase que actúa como plantilla para rectángulos, que vienen dados por las coordenadas de la esquina superior izquierda y por los valores del ancho y alto del rectángulo".
- cv::**Point**. "Clase que actúa como plantilla para puntos $2D$, que vienen dados por sus coordenadas x e y".
- cv::**Size**. "Clase que actúa como plantilla para dar el tamaño de una imagen o un rectángulo".
- cv::**Scalar**. "Clase que actúa como plantilla para un vector de 4 elementos derivada de *Vec*. El tipo *Scalar* se utiliza ampliamente en *OpenCV* para pasar valores de píxeles, por ejemplo, para pasar el color de un píxel".

Lista 2. Funciones relevantes de *OpenCV* [6–8]:

- Mat cv::**imread**( ). "Carga una imagen desde el archivo especificado y la devuelve. Si la imagen no puede leerse, debido a que falta el archivo, permisos incorrectos, formato no compatible o inválido, la función devuelve una matriz vacía *Mat::data==NULL*".
- void cv::**imshow**( ). "Muestra una imagen en la ventana especificada. Si la ventana se crea con el indicador *cv::WINDOW_AUTOSIZE*, la imagen se muestra con su tamaño original, aunque limitada por la resolución de la pantalla".

- bool cv::**imwrite**( ). "Guarda la imagen en el archivo especificado y en el formato indicado en el nombre, si está entre los permitidos".
- void cv::**cvtColor**( ). "Convierte una imagen de entrada de un espacio de color a otro. En el caso de una transformación desde o hacia el espacio de color RGB, el orden de los canales debe especificarse explícitamente, RGB o BGR. En una imagen de color BGR estándar de 24 bits, el primer byte será un componente Azul de 8 bits, el segundo byte será Verde y el tercer byte será Rojo. El cuarto, quinto y sexto bytes serían entonces el segundo píxel y así sucesivamente".
- void cv::**resize**( ). "Redimensiona la imagen de origen hacia abajo o hacia arriba al tamaño especificado y la guarda".
- void cv::CascadeClassifier::**detectMultiScale**( ). "Detecta objetos de diferentes tamaños en la imagen de entrada. Los objetos detectados se devuelven como una lista de rectángulos".
- void cv::**threshold**( ). "Aplica un thresholding de nivel fijo a cada elemento del array. Esta función admite varios tipos de thresholding, que son determinados por el parámetro 'type'. Además, estos pueden combinarse con distintos indicadores especiales para obtener el valor umbral del thresholding mediante algoritmos externos, como $THRESH\_OTSU$ para el algoritmo de Otsu".
- void cv::**mean**( ). "La función cv::mean calcula el valor medio M de los elementos del array, de forma independiente para cada canal".
- int cv::**cvRound**( ). "Redondea un número al entero más cercano".
- void cv::**line**( ). "Dibuja el segmento de línea entre los puntos *pt1* y *pt2* en la imagen".
- void cv::**rectangle**( ). "Dibuja un contorno de rectángulo o un rectángulo relleno cuyas dos esquinas opuestas son *pt1* y *pt2*".
- void cv::**putText**( ). "Renderiza la cadena de texto especificada en la imagen. Los símbolos que no pueden ser representados usando la fuente especificada se reemplazan por signos de interrogación".
- void cv::Mat::**release**( ). "Decrementa el contador de referencias y desasigna la matriz si es necesario".
- void cv::VideoCapture::**release**( ). "Cierra el archivo de vídeo o la cámara".