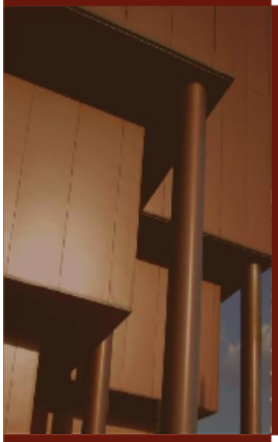
Trabajo Fin de Grado
Grado en Ingeniería de Electrónica, Robótica y Mecatrónica

Plataforma móvil para la detección y el seguimiento de personas con alta temperatura corporal a través del procesamiento en tiempo real de imágenes en el espectro infrarrojo

Autor: Daniel Clavijo Calvo
Tutor: Jesús Capitán Fernández
Rafael de la Rosa Vidal



Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.
Intensificación de Robótica

Plataforma móvil para la detección y el seguimiento de personas con alta temperatura corporal a través del procesamiento en tiempo real de imágenes en el espectro infrarrojo

Autor:
Daniel Clavijo Calvo

Tutor:
Jesús Capitán Fernández
Profesor Contratado doctor
Rafael de la Rosa Vidal
Contratado predoctoral

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Plataforma móvil para la detección y el seguimiento de personas con alta temperatura corporal a través del procesamiento en tiempo real de imágenes en el espectro infrarrojo

Autor: Daniel Clavijo Calvo

Tutor: Jesús Capitán Fernández

Rafael de la Rosa Vidal

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Después de los años de esfuerzo dedicados, toca cerrar una etapa muy importante de mi vida. Simplemente dar las gracias a todas las personas que de un modo u otro han sido partícipes de haber llegado hasta aquí. Recordaré siempre esos grandes profesores que me he encontrado en el camino, que disfrutan de su trabajo y que siempre tienen la mejor actitud ante sus alumnos, incluso en días no tan buenos.

En especial, dar las gracias a mis tutores, Rafael de la Rosa, quién ha puesto a mi disposición todo el conocimiento necesario para poder llevar a cabo este trabajo y me ha prestado su ayuda siempre que ha sido necesario. A Jesús Capitán, por haberme prestado su atención cuando lo he necesitado y darme los mejores consejos en los momentos imprescindibles para seguir avanzando en la dirección correcta.

Dar las gracias también a Juan Antonio Leñero, quién ha tenido un papel muy importante en el desarrollo de este trabajo, aportando su conocimiento, herramientas y materiales necesarios para hacerlo posible. A Francisco José Garrido, por ser una gran persona y hacer más llevaderas las mañanas de prácticas en el IMSE (Instituto de Microelectrónica de Sevilla), además de contribuir con su trabajo al desarrollo de este, aportando parte del desarrollo de la tarea de detección de rostros.

Por último, agradecer el apoyo de mi familia, quienes me soportan día a día y han hecho de mí la persona que soy hoy.

Resumen

En este Trabajo de Fin de Grado se ha desarrollado e implementado un sistema de medida de la temperatura corporal. En el último año, con la aparición de la pandemia de COVD-19, causada por el virus SARS-CoV-2, ha sido necesario el desarrollo de sistemas capaces de detectar personas sospechosas de portar la enfermedad en entornos públicos. Uno de los indicadores de esta enfermedad es la fiebre alta. Por esto, se ha desarrollado un sistema de bajo coste para realizar un seguimiento de personas y estimar, en tiempo real, su temperatura corporal.

En términos de visión, el sistema consta de dos sensores de cámara: un sensor infrarrojo y un sensor de espectro visible para llevar a cabo la tarea de reconocimiento de rostros. Al contar con ambos sensores se detallarán dos opciones o métodos para llevar a cabo esta tarea, desarrollando las ventajas y desventajas de cada uno de ellos. Ambos métodos se basan en el uso de OpenCV, una librería que implementa los algoritmos de detección necesarios. Todo ello trabajando sobre la plataforma Raspberry Pi 3B. Esta tarea de reconocimiento de rostros es la tarea limitante respecto a rendimiento y fluidez del sistema, ya que esto conlleva un gran coste computacional. El procedimiento seguido para estimar la temperatura corporal consta de varias fases. La primera de ellas consiste en la detección del rostro de la persona mediante el uso de redes neuronales. Una vez se tiene delimitada la región de interés, se le aplica un filtrado de Gauss para suavizar la imagen, y se realiza una segmentación empleando el método de Otsu para eliminar aquellas zonas más frías de la imagen que pudieran alterar el cálculo del valor de temperatura. Finalmente, y tras la calibración correcta del sensor de imagen infrarrojo, podemos obtener valores de temperatura a partir de valores de intensidad de píxel.

Para llevar a cabo el seguimiento de rostros ha sido necesario diseñar y construir una plataforma que integre todos los componentes hardware empleados para la visión, junto con dos servomotores con los que variaremos las posiciones de Pitch y Yaw del sistema de cámaras. Aprovechando la tarea de detección, aplicaremos un control sobre estos servomotores con el objetivo de que el centro de la ROI detectada quede lo más centrada posible en la imagen, es decir, se intentará que el centro de la ROI coincida con el centro de la imagen. Veremos las limitaciones que tenemos a la hora de cumplir este objetivo, debido a la resolución mínima de los servomotores.

El sistema se ha construido y testado de forma satisfactoria. Es capaz de realizar la detección de rostros a velocidades de 3 frames por segundo cuando se detectan uno o dos rostros en imagen. Este rendimiento baja un poco cuando se detectan más rostros (en torno a 5 rostros o más) en la misma imagen, lo que podemos considerar como un buen rendimiento teniendo en cuenta que es un sistema de especificaciones muy limitadas, y que además de la tarea de detección, también realiza el seguimiento de la persona.

Abstract

In this final Degree project, a body temperature measurement system was developed and implemented. In the last year, because of the COVID-19 pandemic, the development of systems that can detect sick people has been necessary. Fever is one of the indicators of this disease, and that is why a low-cost system has been developed for people tracking and body temperature measurement in real-time.

The system has two camera sensors: an infrared sensor and a normal vision sensor to do the face recognition work. Two methods will be detailed, with their advantages and disadvantages. Both of them are based on the use of OpenCV, which is a library that implements the necessary detection algorithm, working on the Raspberry Pi 3B. This faces recognition work needs a big computational cost, so is the principal restriction for the system's efficiency. To estimate the body temperature, first of all, we need to detect the face using neural networks. When the region of interest is delimitated, the image is filtered with a Gaussian filter. Then, the Otsu method is used to remove the coldest areas that could deteriorate the temperature value. Finally, after the correct calibration of the infrared sensor, we can get a temperature value from the intensity pixel value.

The faces tracking function is added too. The design and construction of a platform were necessary to integrate all hardware devices used. Pitch and Yaw positions of the cameras system will be modified with two servo motors. Using the detection work, a servomotor control will be applied. The goal is to match the ROI's center with the center of the image. That is limited by the servomotor resolution.

The systems have been built and tested successfully. It can detect faces at 3 frames per second when one or two faces appear in the image. This performance decreases a bit when a few faces are detected at the same time, but that is a good result because also the platform control is implemented in this low-cost system.

Índice de contenidos

Índice de figuras

Lista de abreviaturas

PWM Pulse Width Modulaltion

ROI Region Of Interest

IR Infrarrojo

Hz Hertzios

XML Extensible Markup Language

GPIO General Purpose Input/Output

CSI Camera Serial Interface

USB Universal Serial Bus

RAM Random Access Memory

Wi-Fi Wireless Fidelity

HDMI High-Definition Multimedia Interface

GPU Graphics Processing Unit

CPU Central Processing Unit

SWIR Short-wavelength infrared

MWIR Medium-wavelength infrared

LWIR Long-wavelength infrared

RGB Red-Green-Blue

AGC Automatic Gain Control

LBP Local Binary Pattern

DC Direct Current

FPS Frames per Second

Capítulo 1

Introducción y motivación

Con la aparición del virus SARS-CoV-2 en diciembre de 2019, causante de la COVID-19, la OMS ha establecido como rutina ciertos protocolos [1], o recomendaciones, que se deben cumplir a modo de prevenir la expansión de dicha enfermedad [2]. Estas recomendaciones, seguidas por la mayoría de personas, van desde el uso de mascarilla en lugares públicos, mantener una distancia de 1.5 a 2 metros y evitar las aglomeraciones de personas, mantener una correcta higiene de manos y de las superficies de contacto, así como mantener un control de nuestra salud. Relacionado con esto último, uno de los protocolos más habituales consiste en la medición de la temperatura corporal. Por esto, las personas que acuden a lugares frecuentados por otras personas, como puede ser su puesto de trabajo, deben comprobar que su temperatura corporal se encuentra en niveles correctos (entre 35.5 y 37.2ºC). Esto es, no por encima de 38 grados, que es el momento en el que se considera que una persona tiene fiebre a causa de una infección o enfermedad [3].

Debido a esto, el uso de termómetros de pistola (Figura 1-1) se ha incrementado sustancialmente por su facilidad de uso, precio reducido, medición en pocos segundos y precisión. Además, este tipo de termómetro permite realizar una medición a distancia, sin entrar en contacto con la persona que sea objeto de estudio, lo que hace que la distancia de seguridad entre personas mencionada anteriormente se siga cumpliendo.

Figura 1-1: Termómetro infrarrojo de pistola.

La finalidad de este trabajo es construir un sistema automático para la medición del valor de temperatura corporal, añadiendo además la funcionalidad de seguimiento de rostros. De esta forma tendremos la funcionalidad del termómetro infrarrojo de pistola sin tener que hacerlo manualmente. Esto se llevará a cabo empleando componentes de fácil adquisición en el mercado (productos "off-

the-shelf"). Se han cumplido los siguientes objetivos:

- Estudiar el estado del arte de los diferentes sistemas existentes que cumplen funciones parecidas, tanto de medición de temperatura como del seguimiento de objetos. De este análisis podremos extraer, por ejemplo, algoritmos de detección de rostros.
- Realizar la calibración térmica del sensor infrarrojos montado en el sistema.
- Justificar la elección del hardware empleado.
- Estudiar, diseñar y construir la plataforma que permitirá el seguimiento automático de rostros.
- Implementar el controlador para la función de seguimiento de rostros.
- Ajuste y puesta en marcha del sistema.

En este trabajo se ha organizado la información de la siguiente forma: en el Capítulo 2 se desarrolla el estado del arte de los distintos sistemas que realizan operaciones similares, así como de los dispositivos que desempeñen por separado las dos funciones más destacadas de nuestro sistema, como son: la medición de la temperatura y el seguimiento de objetos. En el Capítulo 3 se verá en detalle cada elemento de hardware que se ha elegido, de forma justificada, para cumplir con los objetivos propuestos. El Capítulo 4 nos presenta las diferentes formas que se han estudiado para realizar el reconocimiento de rostros, trabajando en un primer método basado en la creación y entrenamiento de una red neuronal en el espectro infrarrojo, y un segundo método en el que a partir del uso de redes neuronales ya existentes en el espectro visible, aplicaremos correspondencia de puntos entre imagen visible e infrarroja. En el Capítulo 5 veremos el trabajo realizado para el procesamiento de la imagen infrarrojos y la forma en que se obtiene un valor de temperatura a partir de niveles de intensidad de píxeles. En el Capítulo 6 se detalla el diseño mecánico de la plataforma propuesta para poder acoplar los servomotores al sistema, así como el algoritmo de control aplicado a cada servomotor para conseguir el movimiento necesario en el seguimiento de rostros. Por último, en el Capítulo 7 se hace un repaso de los objetivos alcanzados y de las posibles líneas de trabajo futuras, que mejorarán el funcionamiento del sistema.

Capítulo 2

Estado del arte

El hombre siempre ha tratado de mejorar su calidad de vida, y por ello, el desarrollo y construcción de máquinas que resolviesen operaciones complejas de forma automática y rápida, ha jugado un papel muy importante. Fue en 1946 cuando se construyó ENIAC, la primera computadora electrónica de propósito general, y fue entonces cuando el desarrollo de este campo entró en auge [4, 5]. Estas máquinas implementan algoritmos que permiten resolver problemas de dificultad elevada, pero están limitadas a resolver problemas que admitan la aplicación de dichos algoritmos. Si nos enfrentamos a problemas como la clasificación de objetos por rasgos similares, no podremos resolverlos con estas máquinas. Por eso, estas máquinas debían evolucionar.

Esta evolución nos lleva al concepto de inteligencia artificial (IA), que podría definirse como la capacidad de las máquinas para, mediante el uso de ciertos algoritmos, aprender de esos datos y tomar decisiones de la forma en que lo haría un ser humano. Las ventajas de las máquinas sobre las personas las hacen mucho más eficientes y rentables, ya que no necesitan descansar y pueden analizar grandes volúmenes de información al mismo tiempo con una tasa de error mucho menor [6, 7, 8]. La inteligencia artificial surgió en 1956, cuando en una conferencia en Estados Unidos, Allen Newell y Herbert Simon presentaron el primer programa de ordenador que emulaba características del cerebro humano, llamado Logic Theorist [9].

Dentro del ámbito de la inteligencia artificial, se suelen utilizar las redes neuronales, como herramientas para la predicción de tendencias y clasificación de objetos. Además, tienen la capacidad de aprender y mejorar su funcionamiento [10, 4]. También se introduce el concepto de Visión por Computador, que se entiende como el estudio y entendimiento de los procesos de la visión para poder construir máquinas que sean capaces de realizar tareas similares [11].

Entre los objetivos perseguidos por la inteligencia artificial se encuentra la clasificación de objetos en tiempo real. En nuestro caso, nos centramos en la detección de personas, y más concretamente de rostros. En visión artificial y machine learning, un software muy popular es OpenCV, una biblioteca utilizada en multitud de proyectos para llevar a cabo la detección de objetos en imágenes o vídeos y será también la que nos permitirá realizar este trabajo. En nuestro caso nos centraremos en la detección de rostros en tiempo real, para lo cual, OpenCV dispone de numerosos clasificadores entrenados y algoritmos optimizados.

Para el reconocimiento facial en el espectro visible existen varios algoritmos capaces de obtener la información necesaria de las imágenes que luego se usarán para el entrenamiento de la red neuronal. Entre los algoritmos más conocidos se encuentran: EigenFace, FisherFace y Viola-Jones [12, 13]. Todos ellos implementan el mismo método de extracción de características, conocido como PCA [14], en el que se eliminan las características que aportan poca o nula información, reduciendo así la cantidad de datos necesaria para el posterior entrenamiento.

En concreto, el algoritmo que implementa OpenCV es el de Viola-Jones [15], que introduce el concepto de *"Cascade of Classifiers"*. Este método consiste en llevar a cabo un algoritmo de detección escalonado en diferentes etapas, condicionadas por las denominadas características de *"Haar"*. Las

funciones de Haar extraen información (características) de la imagen de rostros que luego se utiliza para el entrenamiento de una red neuronal. Cada una de estas características definen una etapa de un clasificador en cascada, y para determinar si una imagen contiene rostro o no, deberá pasar por cada una de las etapas. Si en alguna etapa la imagen no cumple la característica, se descarta y pasa a analizarse la siguiente imagen. Esto hace que sea un algoritmo más eficiente que si se comprobase el cumplimiento de todas las características en una misma etapa.

Los algoritmos mencionados tienen buenos resultados cuando se trabaja en el espectro visible, pero para el caso que nos ocupa, el espectro infrarrojo de onda larga (LWIR), ninguno de ellos resulta válido. Esto se debe a que en esta banda del espectro infrarrojo, algunos elementos como la ropa, gafas y distintos complementos filtran la radiación IR, lo que hace que estas zonas aparezcan en tonos oscuros (zonas frías) y resulten imposibles de detectar térmicamente [16]. Por lo tanto, para llevar a cabo la tarea de detección de rostros en el espectro infrarrojo, necesitaremos una base de datos de imágenes infrarrojas, con la que podremos crear una base de datos específica para este uso.

La tarea de encontrar una base de datos de imágenes de rostros de personas en el espectro infrarrojo ha resultado complicada y finalmente sin éxito. Se ha intentado pedir permiso para acceder a bases de datos (*"LWIR UdeC"* y *"UCHThermalFace"*) de un repositorio privado de la Universidad de Chile [17], pero también sin éxito. Las pocas bases de datos públicas encontradas se encuentran en la bibliografía [18], un repositorio donde podemos encontrar diferentes bases de datos en el espectro infrarrojo, pero necesitan ser procesadas antes de utilizarlas para el entrenamiento de la red neuronal, ya que cada imagen está enmarcada con una serie de datos numéricos que no deberían aparecer a la hora de comenzar el entrenamiento.

Por otro lado, La detección de rostros en el espectro visible ha sido un tema muy desarrollado y con una gran cantidad de trabajos realizados sobre este tema. Sin embargo, la detección de rostros en el espectro infrarrojo no ha sido un tema muy tratado, y por eso la escasez de bases de datos de este tipo. Aunque sí existen varios trabajos, todos ellos utilizan métodos distintos a las redes neuronales para la detección de rostros. El método en el que se basan estos trabajos [19, 20, 21, 22] consiste en la segmentación de la imagen por temperatura, partiendo de la idea de que la temperatura corporal se encuentra normalmente en un rango superior a la temperatura del ambiente. Considerando que la temperatura corporal se encuentra por encima de los 30ºC aproximadamente, y que el resto de la imagen (ambiente y fondo) se encuentra a valores de temperatura inferiores. De esta forma, se puede segmentar la imagen para los valores de temperatura más altos, quedándonos así con imágenes en la que las intensidades de píxeles bajas (inferiores al umbral establecido) se toman como regiones a 0ºC. Una vez segmentada la imagen, se recorre para el encasillamiento encontrando los segmentos horizontales y verticales con intensidades correspondientes al rostro de la persona.

Otro método utilizado para la detección de rostros en el espectro infrarrojo consiste en aplicar un descriptor de texturas para la segmentación de la piel en imágenes [23]. Al fin y al cabo, este método, como el de segmentación de piel por temperatura, son métodos que funcionan, pero tienen problemas similares, ya que en determinadas condiciones pueden dar un alto número de falsos positivos.

Por último, existe un trabajo en el que se ha usado una red neuronal para llevar a cabo la detección de rostros en el espectro infrarrojo [24]. Esta red neuronal ha sido entrenada con las bases de datos nombradas anteriormente, *"LWIR UdeC"* y *"UCHThermalFace"*. La primera está formada por 612 imágenes infrarrojas en las que participaron 102 personas. La segunda es una colección de 1484 imágenes tomadas a 53 personas. En este caso se obtienen buenos resultados en la detección. Este trabajo se implementó en un *System on Chip* Zynq-7000, que integra un procesador ARM con una FPGA. La ejecución del detector de rostros se lleva a cabo con OpenCV (implementación en software), lo que hace que sea la parte limitante del sistema, obteniendo como resultado una tasa de 1 fps.

Dadas las dificultades para conseguir una base de datos de imágenes infrarrojas, en este trabajo se expondrán y detallarán dos métodos posibles a seguir para la detección de rostros en el espectro infrarrojo: la creación de una base de datos propia de imágenes en el infrarrojo para luego poder entrenar una red neuronal(ver apartado 4.1), y la utilización de una red neuronal ya entrenada con una base de datos en el espectro visible, disponible en el repositorio de OpenCV (ver apartado 4.2). Con

este segundo método podremos encontrar puntos significativos en la imagen IR análogos a la imagen de espectro visible.

El trabajo desarrollado se ha implementado en la plataforma Raspberry Pi, siendo OpenCV, el software encargado de llevar a cabo la detección de rostros. Por lo que estaríamos realizando una implementación software en hardware de propósito general. Además, aunque la generación de Raspberry Pi usada no sea la más actual, se consiguen tasas de ejecución de unos 3 fps cuando en la imagen se detecta rostro, y de 5 fps cuando no se da ninguna detección, por lo que podemos darnos cuenta de que el cuello de botella del sistema sigue siendo la detección de rostros. Como vemos, la ejecución de este sistema es 3 veces mayor que el sistema comentado anteriormente en el que se utiliza una red neuronal específica del espectro infrarrojo.

Para crear la plataforma en la que integraremos el sistema, se ha optado por una plataforma tipo gimbal [25]. Este tipo de plataformas, en las que el eje del motor se suele utilizar directamente como anclaje de las distintas piezas que forman la estructura, pueden llegar a ser bastante complejas en función de los grados de libertad que tenga. Esta plataformas son muy usadas para la estabilización de sensores de cámara, utilizándose tanto en drones o helicópteros [26, 27], como en teléfonos móviles a modo de "palo selfie" [28]. En nuestro caso, será un gimbal de dos ejes, con el que podremos modificar los valores de Pich y Yaw mediante la inclusión de dos servomotores.

Capítulo 3

Hardware

En este capítulo se va a desarrollar la arquitectura hardware del dispositivo, primero de forma general en la que se verá la unidad de control y periféricos empleados para la adquisición de imágenes y posterior movimiento del dispositivo. Seguidamente, se pasará a la descripción de las características técnicas de cada uno de los elementos empleados.

3.1. Arquitectura hardware del sistema

La Figura 3-1 recoge en un esquema todos los componentes hardware que se usarán en el diseño del dispositivo. Posteriormente veremos cada uno en detalle, tanto sus especificaciones técnicas como la conexión de cada uno de los elementos con la unidad central de control, la Raspberry Pi 3 B.

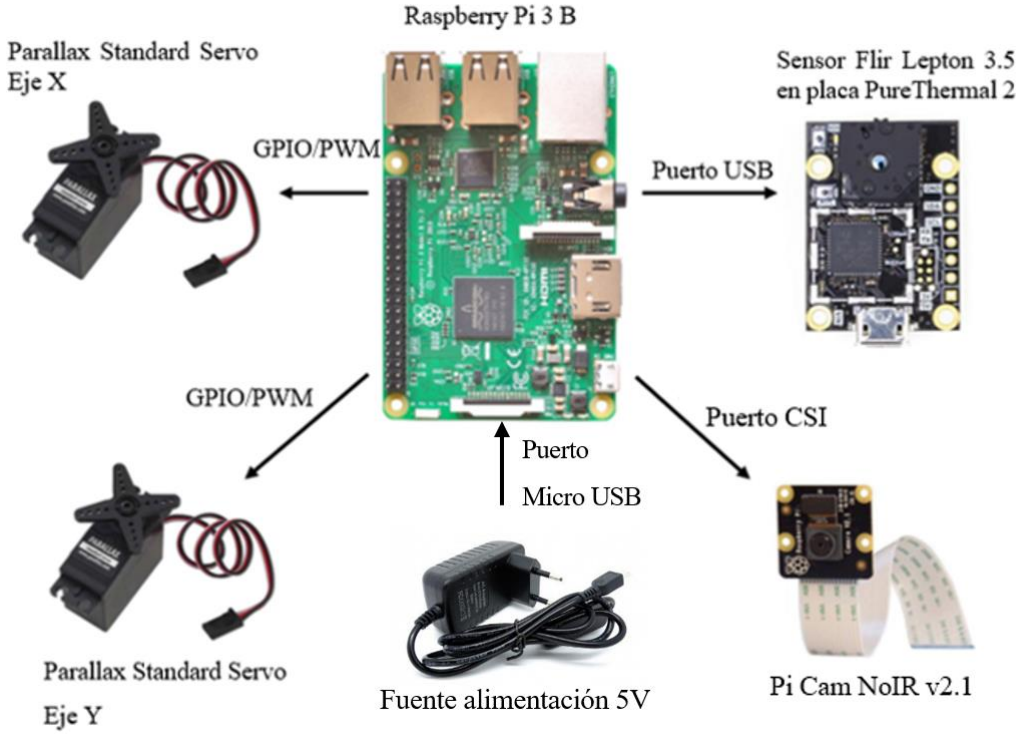


Figura 3-1:Arquitectura hardware del dispositivo.

En este proyecto vamos a usar la tercera generación de la serie de Raspberry Pi, en concreto la Raspberry Pi 3B, que podemos encontrarla por alrededor de 30€, lo que hace de este, un dispositivo muy económico que reducirá el presupuesto total del proyecto de manera considerable. Para la elección del modelo de generación anterior (Raspberry Pi 3B) nos hemos basado en el consumo energético de ambas versiones. Según la información oficial de Raspberry [29], tenemos los valores

de consumo en Amperios. Como ambos dispositivos se alimentan a 5V, podemos expresar el consumo energético en Vatios (W) como se muestra en la Tabla 3-1.

	Raspberry Pi 3B	Raspberry Pi 4B
Reposo	1.5	3
Reproducción de vídeo	2.75	5.85
Estrés máximo	6.7	6.25

Tabla 3-1:Consumos energéticos de las versiones Raspberry Pi 3B/4B en Vatios.

Puede comprobarse que la Raspberry Pi 3B es mucho más eficiente que el dispositivo de última generación, excepto en el caso de tareas muy exigentes, para las que el consumo es un poco mayor para la versión elegida. El hecho de que el sistema sea más eficiente, sería más favorable en el caso de que quisiéramos tener un sistema portable conectándolo a fuentes de alimentación por baterías, aumentando su autonomía.

Además de su mayor eficiencia energética, el modelo Raspberry Pi 3B aporta todos los recursos que necesitaremos para nuestra aplicación.

3.2. Raspberry Pi 3 Modelo B

Como base de este proyecto y motor de cálculo del sistema, se ha elegido la *"Raspberry Pi 3B v1.2"* como principal componente del hardware del proyecto. Por la importancia que tiene este componente, vamos a dedicar un apartado de esta memoria a la descripción de algunas de sus funciones y capacidades.

3.2.1. Historia

La Raspberry Pi Fundation es una organización benéfica con sede en Reino Unido que surge en 2009 con la intención de acercar la tecnología y el interés por la informática a los jóvenes de la época. Raspberry Pi nace de la necesidad de crear un ordenador pequeño, básico y asequible [30]. Esto daría la libertad a cualquier joven de poder experimentar con ella sin miedo a romper un dispositivo caro.

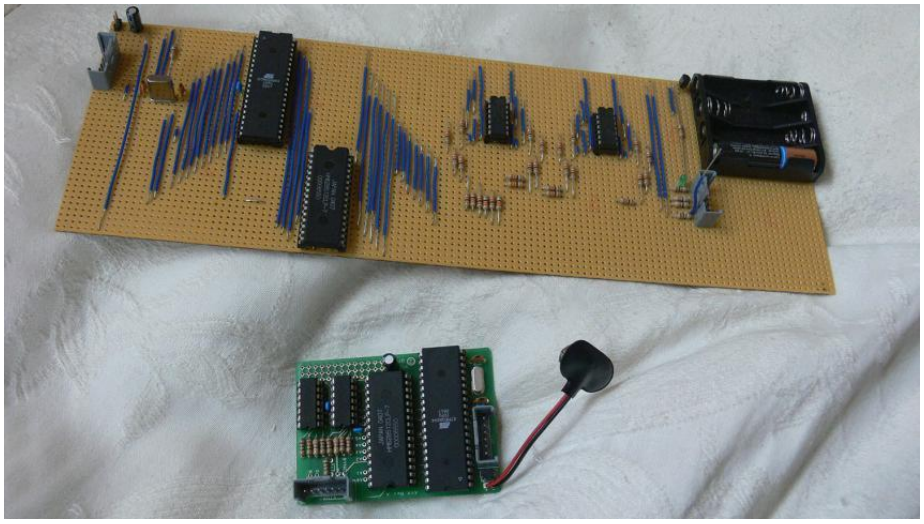


Figura 3-2: Primer prototipo de Raspberry Pi.

En febrero de 2012 se comercializaba la Raspberry Pi 1 B (ver Figura 3-2) por 35 dólares, y se trataba de un ordenador completo con potencia suficiente para programar. Su secreto para ser tan económica fue la reducción del número de componentes que se hizo hasta dejar solo lo imprescindible,

contando con un conector GPIO de 26 pines [31]. Más tarde, en el año 2015 se presentó el modelo de segunda generación, mejorando el procesador, y aumentando la cantidad de pines hasta los 40 pines GPIO.

En el año 2016, la Raspberry Pi Fundation saca a la luz la Raspberry Pi 3 B, modelo que usamos en este proyecto. Renovó su procesador respecto a la versión anterior, y su aporte más novedoso fue la inclusión de Bluetooth 4.1 y de antenas Wi-Fi sin necesidad de adaptadores.

Ya en el año 2019 se anunció la generación más reciente, la Raspberry Pi 4 B. Esta introdujo muchos cambios, entre ellos la mejora de procesador por uno más potente y eficiente, y la elección de la capacidad de memoria RAM, que va desde los 2GB hasta los 8GB. Pasó de tener 1 puerto HDMI completo a 2 puertos microHDMI y un puerto USB 3.0. Además, es capaz de controlar pantallas a una resolución de 4K.

Cada una de las versiones mencionadas cuentan a su vez con una versión más recortada en especificaciones y precio. Se trata de los modelos A. Además de la versión recortada en especificaciones, también se fabricaron versiones más potentes de cada una de ellas. Se denominaron A+ y B+.

3.2.1. Especificaciones técnicas

	Raspberry Pi 3 B	Raspberry Pi 4 B
SoC	BCM2837	BCM2711
Fabricante	Broadcom	Broadcom
Instrucciones	64 bits (AR Mv8)	64 bits (AR Mv8)
CPU	ARM Cortex-a53	ARM Cortex-a72
RAM	1 GB	2/4/8 GB
Cores	Quad-Core	Quad-Core
Velocidad	1200 MHz	1500 MHz
GPU	VideoCore IV (400 MHz)	VideoCore VI (MHz)
Conexiones	HDMI 100 Ethernet (100 Mbps) 4x USB 2.0 40 pins GPIO MIPI DSI display MIPI CSI camera Wi-Fi Bluetooth 4.1 Audio estéreo de 4 polos Puerto de vídeo compuesto	2x micro-HDMI Gigayte Ethernet (1000 Mbps) 2x USB 3.0; 2x USB 2.0 40 pins GPIO 2x MIPI DSI display 2x MIPI CSI camera Wi-Fi Bluetooth 5.0 Audio estéreo de 4 polos Puerto de vídeo compuesto Power over Ehthernet (PoE)

Tabla 3-2:Especificaciones de la RPi 3 vs RPi 4.

En la web de *raspberrypi.org* podemos encontrar toda la información técnica de nuestro modelo. Para poner en contexto la diferencia de especificaciones entre el modelo con el que se ha llevado a cabo este proyecto (*Raspberry Pi 3 B*) [32] y el modelo de última generación (*Raspberry Pi 4 B*) [33] podemos consultar la Tabla 3-2.

Estas diferencias que vemos sobre el papel se traducen en un rendimiento mucho mayor para el modelo de la nueva generación. Para ver con más detalle la diferencia real entre estos dos dispositivos podemos ir al blog *'geeks3d.com'* [34] en el que el usuario JEGX ha realizado diferentes pruebas de rendimiento de CPU y GPU en las que enfrenta estos dos modelos, siendo la Raspberry Pi 4 B la versión de 4 GB (existe también una versión superior de 8 GB). Teniendo en cuenta que en ambos modelos se tiene instalado el mismo sistema operativo, la última versión de Raspbian, se han obtenido los siguientes resultados:

Pruebas de CPU

Para comparar del rendimiento de CPU se ha utilizado el programa *Sysbench* [35], que inicia una prueba multitarea con 4 subprocesos y se mide el tiempo empleado en la ejecución (a menor tiempo, mejor rendimiento). Los resultados fueron:

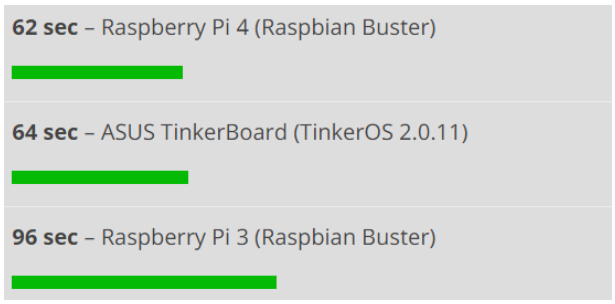


Figura 3-3:Primera prueba CPU.

Este resultado muestra un aumento de rendimiento del **+36%** para la nueva generación, incluso mejora el rendimiento de un procesador de ASUS.

Prueba de GPU

En este caso, el usuario JEGX usó la herramienta GeeXLab 0.29.2 [36]. Recordemos que la RPi 4 viene con una nueva GPU, la VideoCore VI. Mientras que la RPi 3 tiene la VideoCore IV.

La primera prueba consistió en medir la velocidad de fotogramas al lanzar dicho programa.

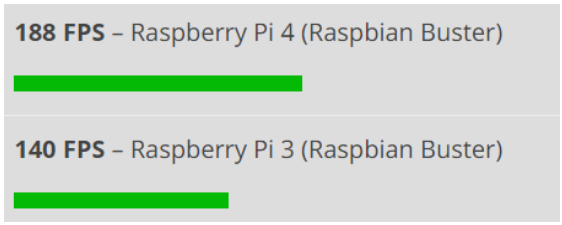


Figura 3-4:Primera prueba GPU.

➔ Esto muestra un rendimiento de **+34%** mayor a favor de la RPI 4.

Una segunda prueba consistió en volver a medir la velocidad de fotogramas, esta vez con una prueba de sombras para evaluar la potencia bruta.

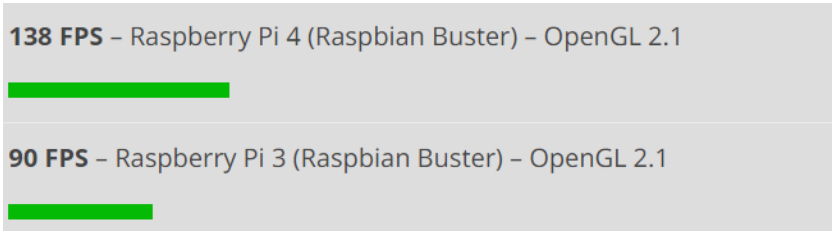


Figura 3-5:Segunda prueba GPU.

➔ En esta prueba, la Raspberry Pi 4 arroja un rendimiento de **+53%** mayor al de la generación anterior.

Realizó una tercera prueba, también de medición de velocidad de fotogramas.

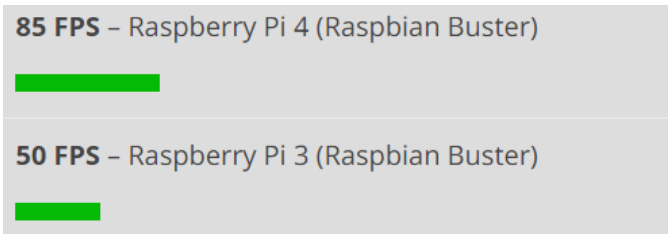


Figura 3-6:Tercera prueba GPU.

➔ La RPi 4 termina la prueba con un rendimiento un **+70%** superior al de la RPi 3.

En la última prueba obtuvo los siguientes resultados:

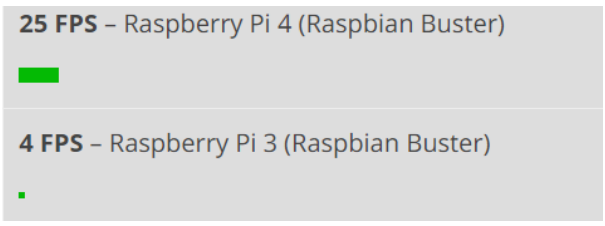


Figura 3-7:Cuarta prueba GPU.

➔ Este resultado muestra un rendimiento de **+525%** superior al de la RPi 3.

Todas estas pruebas las muestra JEGX en el blog *'geeks3d.com'* [34]. Como vemos, la Raspberry Pi 4 presenta un mejor desempeño en todas las tareas, pero como se comentó en el apartado 3.1 Arquitectura hardware del sistema, usaremos la versión de tercera generación por presentar un menor consumo energético, además de disponer de todos los recursos que necesitaremos para la aplicación.

3.2.2. GPIO

GPIO son las siglas de "General Purpose Input/Output", es decir, Entrada/Salida de propósito general. Tanto la generación de Raspberry Pi 3 y 4 poseen 40 puertos GPIO. Sus principales características son, por ejemplo, que pueden ser configurados como entrada o salida según nos convenga, al igual que ocurre en Arduino [37]; pueden activarse o desactivarse mediante código; pueden leer datos binarios, es decir, detectan señal de voltaje o su ausencia; y entre otras cosas, se puede configurar los valores de entrada como eventos que generen alguna acción. Esto es lo que hacemos, por ejemplo, al leer el valor de sensores que impliquen alguna acción.

De todo el conjunto de 40 pines nos interesan aquellos que son compatibles con el hardware PWM (Pulse Width Modulation), que son los pines 12/32/33/35 correspondientes a GPIO 18/12/13/19 respectivamente (ver Figura 3-8). De estos pines, solo podremos usar dos de ellos simultáneamente [38].

Las señales PWM nos permiten simular una señal analógica usando un sistema digital (un ordenador). Si se divide 1 segundo en 500 partes (suficiente para que nuestros sentidos no lo noten),

y en cada división se mantiene la tensión abierta únicamente un 5%, encontramos que la tensión promedio a la salida es un 5% del valor de mantener la señal a valor alto. Entonces, el resultado final de la tensión varía proporcionalmente al porcentaje de tiempo que mantenemos a valor alto la señal, es decir, el valor eficaz de la tensión depende directamente del ancho de pulso. Por eso, a esta técnica se le llama *modulación de anchos de pulsos (PWM)*. Se introducen entonces dos conceptos: la frecuencia de la señal PWM, y el Duty Cycle.

La frecuencia de la señal PWM se corresponde con el tiempo entre disparo de pulsos. Si dividimos 1 segundo en 500 divisiones, la frecuencia de PWM será de 500Hz (el período entre pulsos será 1/500 segundos). En cambio, el Duty Cycle es un valor en porcentaje que representa el tiempo que tenemos en valor alto la señal respecto del tiempo máximo posible [39]. Para el caso anterior, en el que tenemos en alta la señal un 5% en cada división, ese será el valor de Duty Cycle (ver Figura 3-9).

Para entender el funcionamiento de las señales PWM como un ejemplo de la vida real y que realizamos todos los días sin darnos cuenta y sin pensar en ello, podemos pensar en el caso práctico de llenar un vaso de agua del grifo. Cuando veamos que el vaso está casi lleno, cerraremos el grifo poco a poco y no por completo de manera instantánea, para evitar que el agua se derrame del vaso. Esta regulación del caudal de agua es algo sencillo si se trata de hacerlo en modo analógico. En sistemas digitales es algo complejo, ya que digitalmente podemos abrir (1) o cerrar (0) el grifo. Es entonces cuando usamos señales PWM, cuando queremos simular, con un sistema digital, el comportamiento de un sistema analógico.

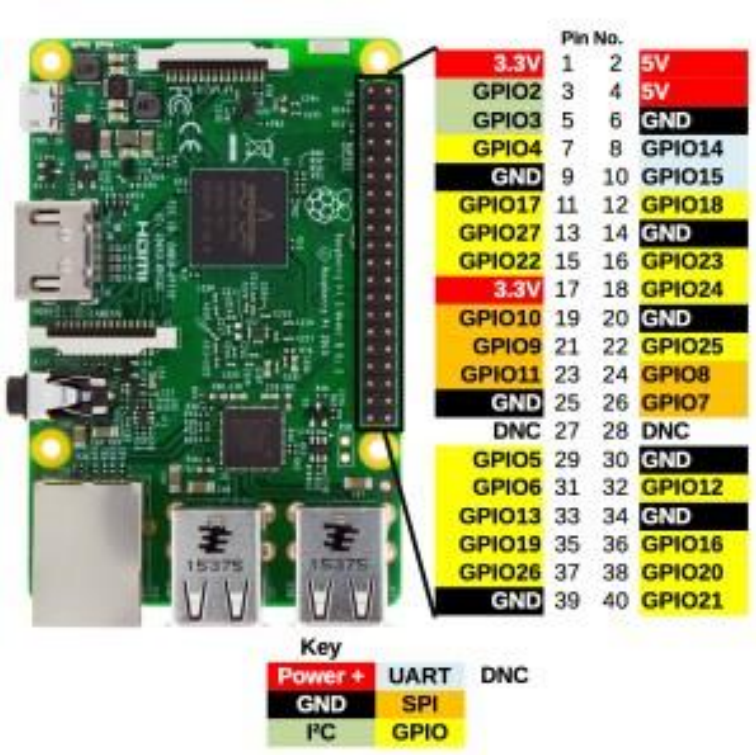


Figura 3-8:GPIO de la Raspberry Pi 3 B.

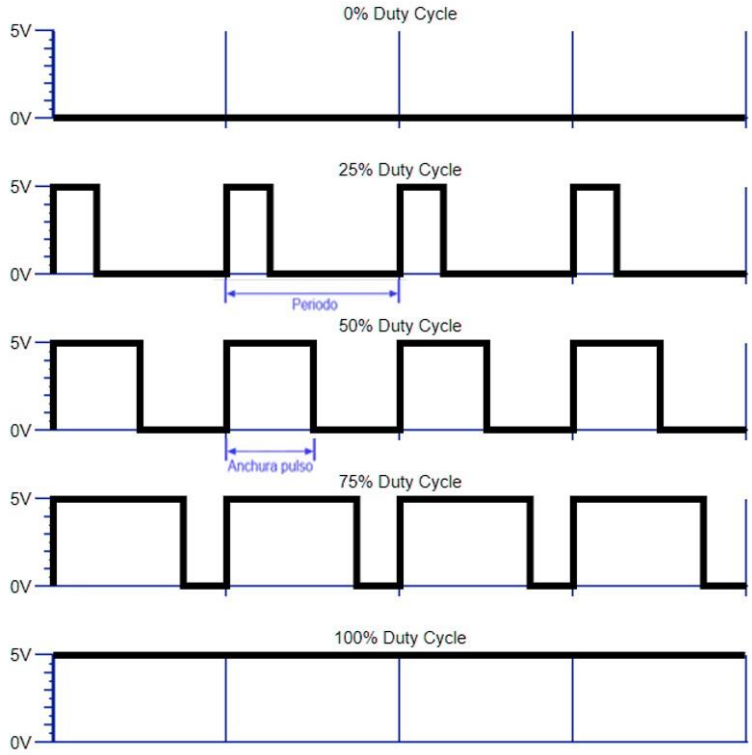


Figura 3-9: Representación de Duty Cycle, periodo y anchura de pulso de señal PWM.

Para el caso de laFigura 3-9, si tenemos una tensión de alimentación Vcc de 5V y tan sólo queremos una señal de 1V, podemos generar una señal PWM que el 20% del tiempo valdrá 5V y el 80% restante valdrá 0V siguiendo las siguientes ecuaciones que modelan el valor de Duty Cycle y el valor de la señal PWM [40].

Teniendo en cuenta la gráfica c) (50% Duty Cycle) de la Figura 3-9, y a partir de la ecuación 3-1, obtenemos la ecuación 3-2. De forma general podemos obtener el valor de salida de la señal PWM (Vsalida), como indica la ecuación 3-3.

$$V_{med} = \frac{1}{T}\int_0^T V(t)dt \tag{3-1}$$

$$V_{med} = \frac{1}{T}\left(\int_0^{0.5} V(t)dt + \int_{0.5}^{1} V(t)dt\right) \tag{3-2}$$

$$V_{salida} = (Vcc_{máx} + Vcc_{min}) * Duty\ Cycle \tag{3-3}$$

$$Duty\ cycle = \frac{T_{ON}}{T} \tag{3-4}$$

Donde: T_{ON} = tiempo con señal a valor alto.

T = tiempo total o Periodo.

3.3. Sensor Flir Lepton 3.5

Gracias a la termografía infrarroja, somos capaces de medir temperaturas a distancia sin tener un contacto físico con el objeto a analizar. Esto se hace mediante la captación de la intensidad de radiación infrarroja que emiten los cuerpos. La termografía es un campo de la ciencia que está evolucionando muy rápidamente en las últimas décadas debido a los grandes avances realizados en la tecnología de

microsistemas y el diseño de los detectores infrarrojos, entre otros [41]. En la actualidad se puede aplicar estos conocimientos en diferentes campos de la industria, como por ejemplo la monitorización de sistemas y el mantenimiento predictivo de estos, lo que puede significar la reducción de costes. Por ejemplo, la termografía permite la detección de escapes de gas invisibles para el ojo humano. Utilizando cámaras termográficas podemos convertir la energía radiada, que es invisible al ojo humano, en una imagen visible formada por la temperatura superficial de los objetos captados por la cámara. Estos dispositivos a los que llamamos *'cámaras'* son en realidad sensores que detectan el calor radiado y nos lo muestra en forma de imagen. A los dispositivos que operan en la longitud de onda infrarroja se les conoce como *'generadores de imágenes'* [41, 42, 43].

En termografía infrarroja nos podemos encontrar tres longitudes de onda: infrarrojos de longitud de onda corta (SWIR), infrarrojos de longitud de onda media (MWIR) e infrarrojos de longitud de onda larga (LWIR).

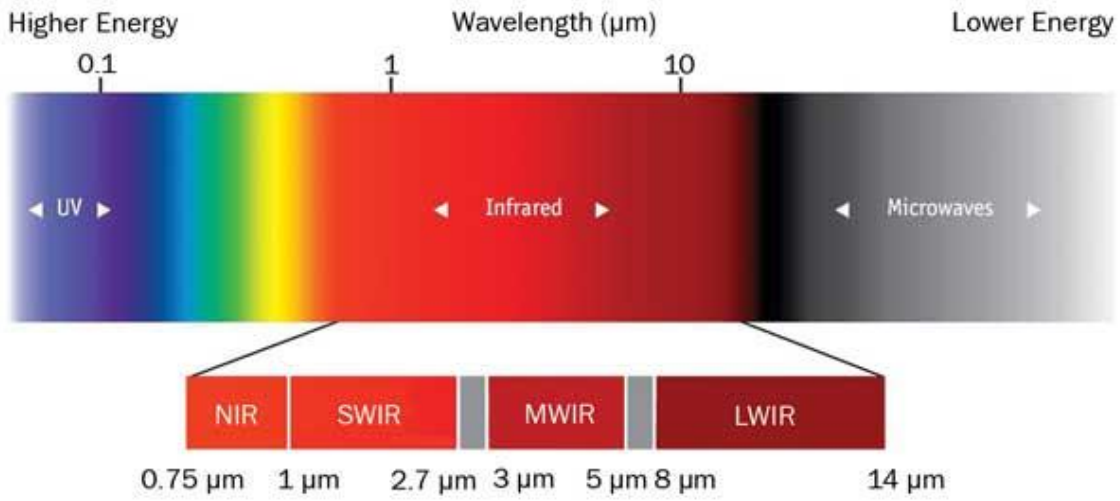


Figura 3-10:División del espectro infrarrojos.

Como vemos en la Figura 3-10, el infrarrojo de longitud de onda corta (SWIR) se define en el rango de longitud de onda de 1 a 2.7 micras. El infrarrojo de longitud de onda media (MWIR) se define en el rango de 3 a 5 micras [44].

Por su parte, el infrarrojo de longitud de onda larga (LWIR) se define entre las 8 y 14 micras. Este será nuestro rango de interés, ya que para la aplicación con la que vamos a trabajar, implementaremos un sensor de espectro infrarrojos LWIR. Este tipo de dispositivos son los que se suelen utilizar en las prácticas de inspección de temperatura. La forma en que lo hacen es determinando si un punto está más frío o más caliente que otro captando su emisión infrarroja [45].

En este proyecto usaremos el sensor *"Lepton 3.5"* desarrollado por FLIR junto con la placa de conexiones *"PureThermal 2"* como se muestra en la Figura 3-11, que deberá ir conectado a uno de los puertos USB de los que dispone la Raspberry Pi 3 B.

Figura 3-11:PureThermal 2 junto con sensor Lepton 3.5 by FLIR.

3.3.1. Características del sensor Lepton 3.5 de FLIR

En la web del sensor podemos encontrar las siguientes características para este sensor [46] (Figura 3-11):

- Sensibilidad térmica: <50 mK (0.050˚C).
- Rango espectral: 8-14 micrones (nominal) Infrarrojos de onda larga (**LWIR**).
- Resolución imagen: 160 x 120 píxeles.
- Precisión radiométrica[1] [47]:

  Modo de alta ganancia: mayor de +/- 5 ° C o 5%;

  Modo de baja ganancia: mayor de +/- 10 ° C o 10%.

- Rango dinámico[2] de escena [48]:

  Modo de alta ganancia: -10 ° a + 140 ° C;

  Modo de baja ganancia: -10 ° a + 400 ° C (a temperatura ambiente)

  -10 ° a + 450 ° C (típico).

- Tamaño de píxel: 12 micrómetros.
- Velocidad de fotogramas: 8,7 Hz (efectivo).
- Formato de salida: seleccionable por el usuario de 14 bits, 8 bits (AGC aplicado) o RGB de 24 bits (AGC y coloración aplicados).
- Campo de visión horizontal (HFOV): 57 °.
- Tipo de lente: f / 1.1
- Consumo de energía: 150 mW típico, 650 mW durante el evento del obturador, 5mW en espera.

[1] La **precisión radiométrica** hace referencia al número de niveles digitales utilizados para representar los datos recogidos por el sensor.

[2] El **rango dinámico** de un sensor hace referencia a la capacidad de éste para obtener una gran variedad de blancos, grises y negros. A mayor rango dinámico, más detalles será capaz de detectar el sensor.

- Dimensiones (ancho x largo x alto): 10,50 x 12,70 x 7,14 mm.
- Peso: 0,9 gramos.

3.4. Sensor Pi Cam NoIR 2.1

Para capturar imágenes en el espectro visible, usaremos la cámara Raspberry Pi Cam de 8 megapíxeles. Más concretamente la versión NoIR v2.1 que utiliza un sensor SONY IMX219 [49]. Este modelo de cámara es compatible con todos los modelos de Raspberry Pi con cámara, y es prácticamente idéntico al sensor del modelo anterior (v1.3). La serie de dispositivos Pi Cam están fabricados especialmente para su uso con los modelos de Raspberry, por lo que cuentan con el conector CSI para permitir la comunicación entre ambos dispositivos. En la Figura 3-12 se muestra dicha conexión. Este puerto CSI ofrece un bus de conexión eléctrica unidireccional (de la cámara al procesador) entre dos dispositivos con 15 conductores, con un ancho de banda de 4 GBps para la transferencia de datos entre cámara y procesador [31].

Figura 3-12:Conexión puerto CSI entre Pi Cam y Raspberry Pi 3 B.

Podemos ver en la web de *"raspi.tv"* sus especificaciones principales [49]:

Lente: Enfoque fijo (No ajustable).

Megapíxeles: 8 MP.

Resolución de cámara: 3280 x 2464 píxeles.

Resolución de vídeo: 1080p30, 720p60 y 640x480p90

Interfaz: CSI.

Conector: conector plano.

Dimensiones: 25 mm x 23 mm x 9 mm.

Peso: ~3 g.

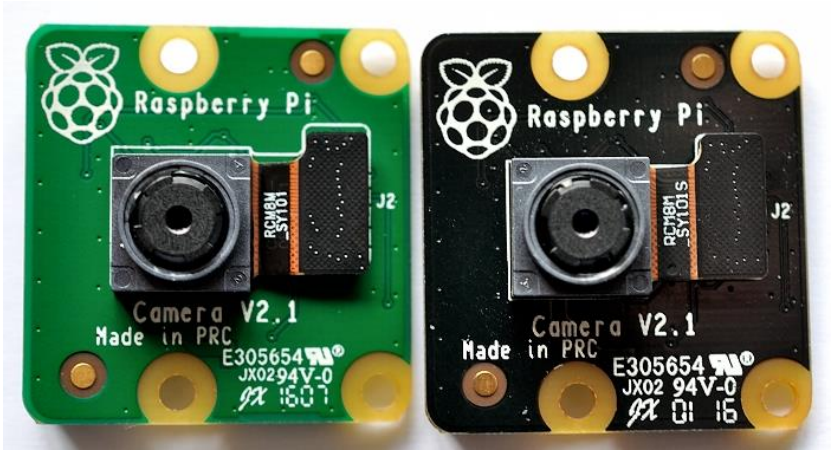


Figura 3-13:Pi Cam v2.1 (izquierda) y Pi Cam NoIR v2.1 (derecha).

Como vemos en la Figura 3-13, existen dos versiones de esta Pi Cam v2.1 [49]. La que usaremos nosotros en este proyecto será la versión NoIR v2.1. Esta denominación NoIR nos indica que el sensor no incluye filtro de infrarrojos, y es sensible a la radiación por infrarrojos de onda corta (SWIR), pero no es sensible a la radiación por infrarrojos de longitud de onda larga (calor, LWIR) [50]. Por tanto, no sería posible utilizar este sensor para detectar a personas mediante su calor corporal. Además, para poder ver en la oscuridad con este sensor, se necesita iluminación mediante LEDs infrarrojos, con los que podemos iluminar la escena en el espectro infrarrojo y captar imágenes claras en la oscuridad.

En la Figura 3-14 podemos ver la diferencia entre ambos modelos mostrados cuando se enfoca a una pared de ladrillos en la oscuridad utilizando una iluminación de LEDs infrarrojos [51].

Figura 3-14:Pi Cam v2.1 (derecha) y Pi Cam NoIR v2.1 (izquierda). Imagen en condiciones de visibilidad nula, con iluminación de LEDs infrarrojos.

Se puede apreciar que el sensor NoIR recoge una imagen bastante clara en el espectro infrarrojos mientras que el sensor de visión normal no es capaz de recoger ningún tipo de información de la escena.

Otra diferencia a destacar entre ambos sensores, es la calidad de imagen que se obtiene en condiciones normales con clara visibilidad. El sensor NoIR, al no tener filtro de infrarrojos obtiene una imagen con colores más “desgastados” (más rosados) que la imagen que podemos ver con el sensor que sí incluye este filtro, el sensor de visión normal.

Figura 3-15:Pi Cam v2.1 (derecha) y Pi Cam NoIR v2.1 (izquierda). Imagen en condiciones normales.

Capítulo 4

Algoritmos de detección de rostros

Como ya sabemos, llevar a cabo la tarea de detección de caras será el primer objetivo y más básico e importante de todos los que componen este proyecto, ya que sobre él se irán añadiendo el resto de funcionalidades del sistema. Para llevar a cabo este primer objetivo, se proponen dos métodos totalmente diferentes uno de otro. Como tenemos dos sensores de cámara instalados en nuestro sistema, el sensor de infrarrojos y el sensor de espectro visible, disponemos de varias posibilidades y caminos a seguir para tener un buen reconocimiento de rostros. Se describirán los dos métodos que se han implementado y testado, y que se proponen como solución a este problema.

El primero de los métodos de detección de caras que se propone se basa en el uso del sensor de espectro infrarrojos para conseguir una detección rápida y precisa, que, aunque a priori parece ser la manera más óptima para nuestra aplicación, también presenta algunos inconvenientes, como veremos en la sección correspondiente.

El segundo método que veremos se basa en la utilización simultánea de ambos sensores de cámara para llevar a cabo la detección de rostros con uno de ellos, y representar esa cara detectada en la imagen recogida por el otro sensor. Es decir, se utilizará el sensor de espectro visible para reconocer caras y encuadrar cada una de ellas en un Bounding-Box. Seguidamente, esas coordenadas de los puntos que conforman ese Bounding-Box se pasará de una imagen (espectro visible) a otra (espectro infrarrojo) mediante correspondencia de puntos. Este método tampoco es perfecto y se detallarán sus carencias en su respectiva sección.

4.1. Método 1: Creación y entrenamiento de un detector Haar Cascade

El principal problema del proyecto consiste en poder dotar al sistema de la capacidad de detección de rostros de personas en el espectro infrarrojo. Para poder llevarlo a cabo, primero se debe explicar el concepto de *"Haar Cascade"* [52].

En 2001, Paul Viola y Michael Jones proponen en su artículo *"Rapid Object Detection using a Boosted Cascade of Simple Features"* [15] un método eficaz de detección de objetos utilizando clasificadores en cascada basados en características de *"Haar"*. Al trabajar con detecciones de rostros, necesitaremos imágenes para entrenar el clasificador. Se necesitan muchas imágenes positivas (imágenes de rostros) e imágenes negativas (imágenes sin rostros). Las funciones de Haar se utilizan para extraer características del algoritmo y entrenar así el clasificador [53].

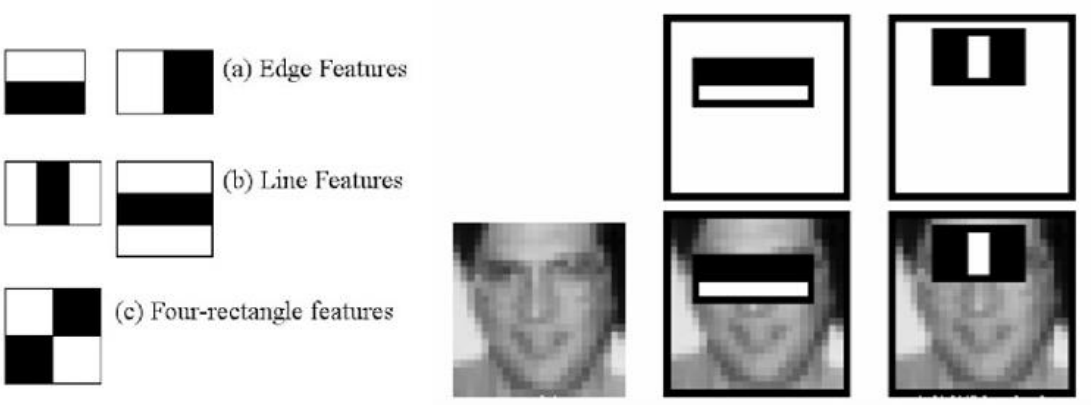


Figura 4-1:Funciones de Haar (izquierda). Aplicación de funciones de Haar (derecha).

Cada característica representa un valor obtenido de realizar la resta del valor de los píxeles que se encuentran bajo el rectángulo blanco a la suma de los píxeles que se encuentran bajo el rectángulo negro. Estas funciones de Haar se aplican sobre la matriz de píxeles como una máscara, dependiendo de si se trata de una función de dos, tres o cuatro rectángulos. Esto provoca que el número de características sea muy elevado. La imagen que cumpla con las características que se consideren como más significativas de una imagen en la que se encuentra rostro, se considerará imagen que contiene rostro.

Para hacer la detección más eficiente y no tener que aplicar todas las características en una misma ventana, es decir, en cada región de la imagen para detectar cara, el método de *'Viola-Jones'* introduce el concepto de *"Cascade of Classifiers"*, que propone hacerlo de forma escalonada y por etapas. Esto es, aplicar una serie de características básicas a la región que se analiza, y si no las cumple, desecharla y pasar a analizar la siguiente región. La región que cumpla con las características de la primera etapa pasa a la segunda etapa donde se le aplican nuevas características. Así de manera sucesiva hasta cumplir con todas las etapas. La región que pasa por todas ellas es una región de cara [15].

Existen varias *"Haar Cascade*" ya entrenadas y con la información de dicho entrenamiento almacenada en archivos *XML* en el repositorio de OpenCV en GitHub [54]. Estas redes neuronales son específicas para la detección de personas o partes de ella (rostros, cuerpos completos, ojos, etc.) disponibles para todo aquel que desee usarlas. Además, OpenCV también pone a nuestra disposición ficheros de código que hacen uso e implementan estos archivos *"Haar Cascade"*. Estos archivos XML se han creado a partir de bases de datos formadas por imágenes del espectro visible convertidas a escala de grises como podemos ver en las imágenes de explicación del funcionamiento de las características de Haar en la web de OpenCV [52], para usarse con sensores de cámara de espectro visible, y no para su uso con cámaras térmicas como es el caso de este proyecto. Por este motivo, el uso de estas *"Haar Cascade*" en el infrarrojo ofrecen un rendimiento muy bajo.

Este problema nos lleva a la creación de nuestro propio archivo XML, que debe ser específico para nuestra aplicación y para ello debemos emplear una base de datos compuesta por imágenes en el espectro infrarrojo. Vamos a describir paso a paso cómo se ha creado y entrenado una red neuronal, y con la información de dicho entrenamiento se genera el fichero que finalmente usaremos, llamado "*cascade.xml*".

4.1.1. Creación de la base de datos

Los archivos XML que podemos encontrar en el repositorio de OpenCV se han creado a partir de una base de datos de imágenes tomadas con una cámara de espectro visible y convertidas a escala de grises. Para crear nuestro propio clasificador Haar Cascade, y que sea específico para su uso en el espectro infrarrojo, primero debemos crear una base de datos compuesta de imágenes en el infrarrojo.

Crearemos una carpeta de trabajo específica para el desarrollo de nuestro archivo XML, que

llamaremos por ejemplo *"IRFacesCascade"*.

Para tomar estas imágenes térmicas se ha usado el propio sistema de cámaras que tenemos ya montado. Tendremos que usar dos tipos de imágenes: imágenes positivas (imágenes de caras) e imágenes negativas (imágenes que no contienen cara) [55].

En nuestra carpeta de trabajo "*IRFacesCascade*" creamos ahora una carpeta "p" en la que guardaremos todas las imágenes positivas, y una carpeta "n" en la que se guardarán todas las imágenes negativas. La Figura 4-2 y la Figura 4-3 son ejemplos de lo que guardaremos en las carpetas "n" y "p":

Figura 4-2: Imagen negativa.

Figura 4-3: Imagen positiva.

Para tomar estas imágenes haciendo uso del propio sensor de infrarrojos de nuestro sistema, utilizamos un fichero de código escrito en lenguaje *'Python'* [56] que podemos ver en la sección de 'Anexos' donde se da una breve explicación de su funcionamiento (ver Anexo A). De esta forma podemos añadir imágenes reducidas de tamaño a las carpetas *"n"* y *"p"* de manera sencilla y rápida.

Las imágenes que guardamos en estas carpetas son en realidad la región de interés de la imagen captada por el sensor, es decir, en el caso de tener una imagen positiva, guardaremos en la carpeta *"p"* (cuando pulsemos la tecla 'p') únicamente la ROI.

Además, la ROI se almacena en las carpetas de imágenes correspondientes con un tamaño reducido, concretamente de un ancho de 38 píxeles y alto de 46 píxeles. Esto ayudará a que la base de datos ocupe un menor espacio, y el posterior entrenamiento de la red neuronal tendrá un coste computacional menor, ya que si no reducimos el tamaño de estas imágenes, tendría que entrenar con imágenes de 65 píxeles de ancho y 80 de alto (tienen este tamaño porque sólo almacenamos la ROI, serían de 160x120 píxeles si almacenamos la imagen completa, como vimos en las especificaciones del sensor en el apartado 3.3.1 Características del sensor Lepton 3.5 de FLIR. En esta reducción se ha tenido en cuenta que en la imagen guardada para el entrenamiento se sigan apreciando de manera clara los detalles que harán característicos cada rostro que se pueda almacenar como ojos, cejas, nariz, etc, (ver Figura 4-4).

Figura 4-4:ROI del frame (imagen positiva).

Figura 4-5: Frame completo.

En definitiva, se ha creado una base de datos compuesta de 477 imágenes positivas y 1000 imágenes negativas. Como imágenes positivas, se han tomado imágenes de personas con gafas y sin ellas para tener en cuenta estos dos posibles escenarios. Aun así, la carpeta de imágenes positivas es bastante escasa y la variedad de rostros que incluye es muy pobre. La gran variedad de casos posibles que se pueden dar en la realidad, con rostros y expresiones distintas, hace que el funcionamiento de esta red neuronal no sea muy efectivo, excepto con las personas a las que se le tomaron fotografías de sus rostros para la creación de la base de datos, a las que detectará de manera rápida y eficiente. A este conjunto de imágenes positivas podríamos incluir también imágenes de personas con mascarilla para hacerla más completa, mejorando la capacidad detección.

Por otro lado, la carpeta de imágenes negativas incluye imágenes del entorno en el que se tomaron las imágenes positivas. Las imágenes negativas que se usarán en el entrenamiento de la red neuronal deben tomarse en el ambiente en el que finalmente se usará el sistema para conseguir mejores resultados. Como comenta Gabriela Solano (apasionada del mundo de la visión por computador) en su Blog [56]:

> *"Toma en cuenta además, que para obtener mejores resultados de detección, las imágenes de entrenamiento deben tener la mayor variedad posible, tomando en cuenta el ambiente en donde va a trabajar el detector".*

4.1.2. Cascade Trainer GUI

Para crear nuestro propio detector utilizando Haar Cascade usaremos el programa "*Cascade Trainer GUI*", el cuál a partir de una base de datos separada en carpetas 'n' y 'p', como ya hemos hecho, realiza el entrenamiento de una red neuronal y guarda toda la información del entrenamiento en un archivo que por defecto tiene de nombre "*cascade.xml*". Este archivo será el que finalmente usemos para realizar la detección de rostros en el espectro infrarrojo.

Para descargar este programa, lo hacemos de manera rápida y sencilla directamente desde la web [57], donde también nos explican de una manera muy sencilla cómo usar este programa. Vamos a ver todos los pasos seguidos para llevar a cabo el entrenamiento de nuestra red, los cuales vienen explicados cada uno de ellos en la web.

4.1.2.1 Paso 1

Una vez instalado el programa, lo ejecutamos y aparecerá la siguiente ventana:

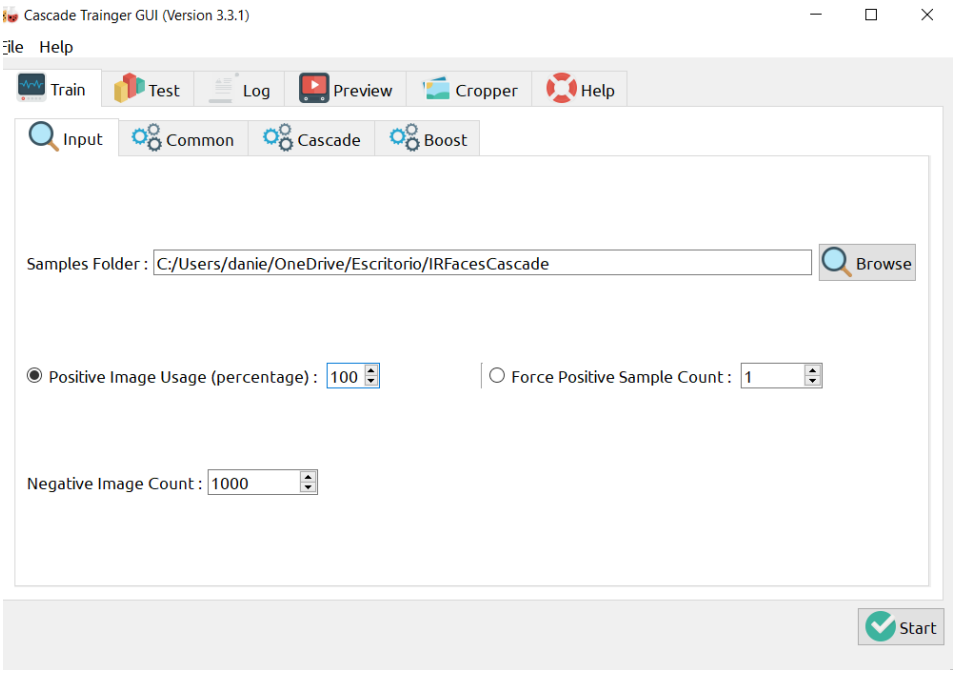


Figura 4-6: Paso 1 del entrenamiento de la red. (Captura de pantalla de mi PC).

Primero nos centraremos en la pestaña *"Input"*. Por defecto, Cascade Trainer GUI nos aporta unos parámetros de ajuste que considera como más optimizados y recomendados para el entrenamiento, pero algunos de ellos es necesario ajustarlos para cada entrenamiento en concreto. Indicaremos en *"Samples Folder"* nuestra carpeta de trabajo principal, que es la que contendrá las carpetas 'n' y 'p'. En este caso nos referimos a la careta "IRFacesCascade".

Marcaremos la opción "*Positive Image Usage (porcentage)"* con valor 100. Esto significa que usaremos el 100% de las imágenes contenidas en la carpeta 'p' para el entrenamiento. En *"Negative*

Image Count" indiaremos el número de imágenes de la carpeta de imágenes negativas, en este caso es de 1000 imágenes.

4.1.2.2 Paso 2

Pasamos ahora a la pestaña *"Common"*, donde nos encontraremos los siguientes parámetros:

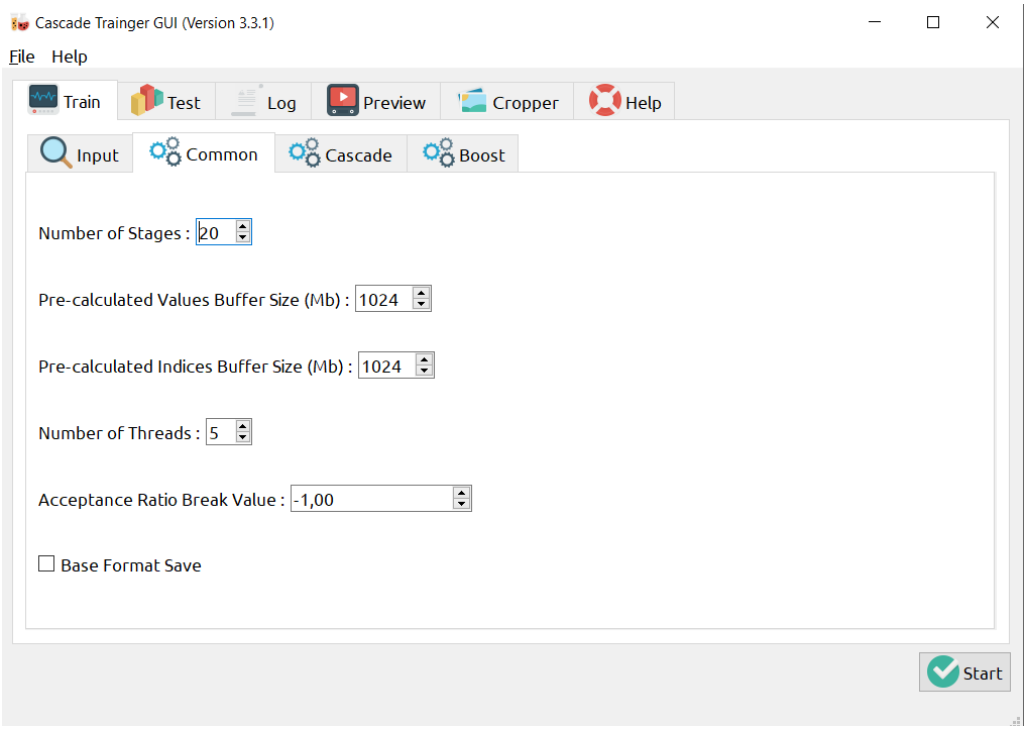


Figura 4-7: Paso 2 del entrenamiento de la red. (Captura de pantalla de mi PC)

En esta ventana, aparece *"Num of Stages"*, que son el número de iteraciones del entrenamiento. Si indicamos un valor muy alto, la precisión del clasificador mejorará, pero también aumentará el tiempo de entrenamiento empleado. Se ha dejado el valor 20, que es el que el programa considera por defecto como valor óptimo.

Los dos siguientes apartados *"Pre-calculate Values Buffer Size (Mb)"* y *"Pre-calculated Indices Buffer Size (Mb)"* hacen referencia a la cantidad de memoria que vamos a dedicar al entrenamiento. Si nuestro PC tiene una memoria de 8GB podríamos dedicar 2048 Mb a cada uno de estos apartados, lo que reduciría el tiempo de entrenamiento. Los dos siguientes parámetros se han mantenido también por defecto.

4.1.2.3 Paso 3

En este tercer paso configuraremos la pestaña *"Cascade"*.

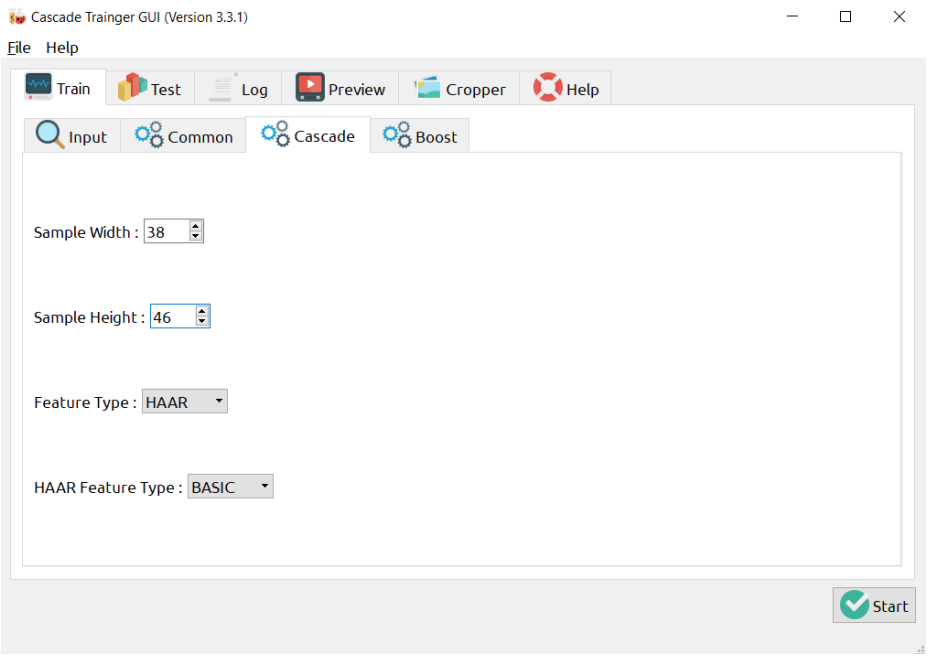


Figura 4-8: Paso 3 del entrenamiento de la red. (Captura de pantalla de mi PC).

En este apartado configuraremos el parámetro *"Sample Width"* a valor 38, que será el ancho de las imágenes que usaremos para el entrenamiento. *"Sample Heigh"* será el alto de las imágenes, que en este caso es de 46 píxeles. Estos valores se han tomado como referencia del entrenamiento de una red que hizo Gabriela Solano en su blog [56]. Es muy importante que todas las imágenes de nuestra base de datos tengan el mismo tamaño para que durante el entrenamiento no se produzcan fallos.

En *"Feature Type"* podemos elegir entre tres tipos de clasificador: HAAR, LBP o HOG (cuyas definiciones aparecen en la web del programa que estamos usando "Cascade Trainer GUI" [57]). HOG lo utilizaremos únicamente si tenemos la versión 3.1 o posterior de OpenCV. Nuestra versión de OpenCV es la 4.5 [58].

LBP lo usaremos cuando nuestra base de datos contenga muchas imágenes y queramos obtener menor tiempo de entrenamiento.

HAAR lo usaremos cuando necesitemos que el clasificador sea más preciso, por el contrario, necesita mayor tiempo de entrenamiento. Este es el método que emplearemos.

El parámetro *"HAAR Feature Type"* lo establecemos en BASIC para indicar que todas las imágenes están en posición vertical. Si lo establecemos en ALL, estaremos indicando que también puede haber imágenes rotadas 45 grados en esa base de datos.

En la pestaña "*Boost"* mantendremos todos los parámetros por defecto, como así recomiendan en la web de *"Cascade Traiener"* GUI. Presionamos *"Start"* y comenzará a ejecutarse el entrenamiento. Una vez finalizado, nos habrá generado el archivo *'cascade.xml'* en la carpeta de trabajo seleccionada en la primera pestaña del paso 1.

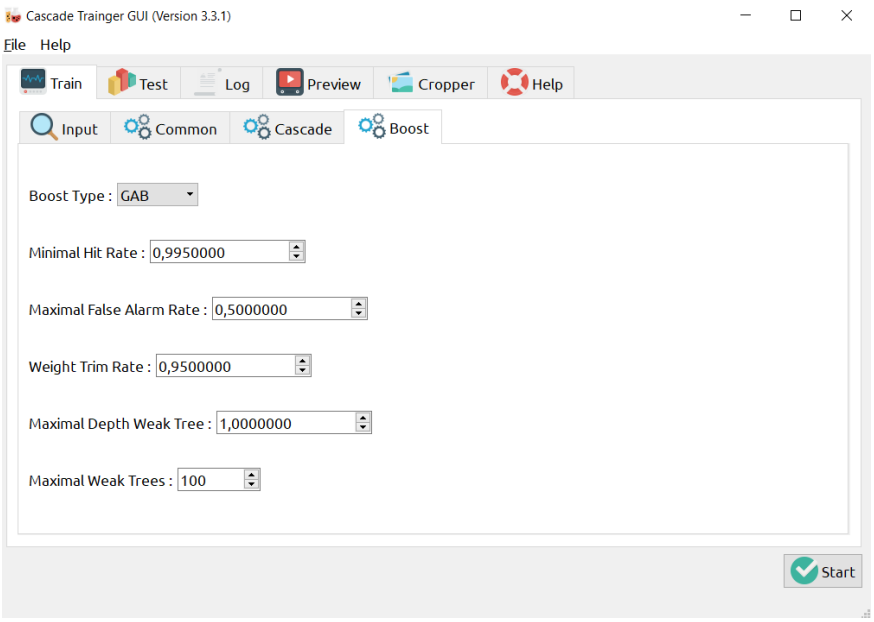


Figura 4-9: Pestaña "Boost". Se dejan los parámetros propuestos por defecto. (Captura de pantalla de mi PC).

Este programa también nos da la opción de probar el funcionamiento de la red que se ha entrenado, en la pestaña *"Test"*, y se pueden ver los resultados en la pestaña *"Preview"*. Nosotros la probaremos directamente haciendo uso de nuestro código.

4.1.3. Resultados del entrenamiento de la red

Para probar el archivo *"cascade.xml"* que se ha generado con la información del entrenamiento, hacemos uso del código que nos proporciona OpenCV en lenguaje C++ para la detección de rostros [59]. Este código está pensado para usarse directamente con los archivos Haar Cascade ya entrenados que podemos encontrar en el repositorio de OpenCV [54], pero podemos aprovecharlo también para nuestra red entrenada en el espectro infrarrojo. Simplemente debemos tener en cuenta que al cambiar la detección del espectro visible al infrarrojo, tenemos que indicarlo pasando como último argumento, en la orden de ejecución del programa, el índice del sensor que estaremos utilizando. Para el sensor del visible se corresponde el índice 0, mientras que para el que usaremos se corresponde el índice 1 (ver Figura 4-10).

```
pi@raspberrypi:~ $ ./peopledetect --cascade=./cascade.xml --scale=1.3 1
```

Figura 4-10:Orden de ejecución.

También debemos tener en cuenta que tendremos que modificar la función utilizada para la detección de rostros en este caso, ya que le pasaremos como argumento nuestra red neuronal y no una de las redes que ya nos ofrece OpenCV en su repositorio. Ésta, es la función *"detectMultiScale"*, que pertenece a la clase *"CascadeClassifier"* [60] (línea 315).

Esta función recibe varios argumentos de entrada. El primer argumento será la imagen sobre la que se quiere realizar la detección de rostros. Esta imagen es la matriz correspondiente a cada frame recogido por el sensor de infrarrojos, que llamaremos en nuestro código *'smallImg'*. Tiene este nombre porque es la imagen original pero reducida un factor de escala que nosotros aportaremos como argumento de entrada para nuestro programa. En nuestro caso, la imagen se ha reducido un factor de escala 1.3 como se ve en la Figura 4-10, lo que aportará más rapidez a la ejecución del programa al tratar con imágenes más reducidas. Más adelante se explicará un poco más en detalle cómo se usa internamente este factor de escala (ver apartado 4.2.1).

El segundo argumento que recibe la función es el vector de rectángulos, donde guardaremos las

coordenadas del Bounding-Box que contiene cada rostro detectado, y que pertenece a la clase *"Rect"* [61]. Este vector es en nuestro código la variable *"faces"*, y contiene las siguientes coordenadas y dimensiones: *'faces(esquina superior izquierda X, Esquina superior izquierda Y, Ancho, Alto)'*.

El tercer argumento de la función es un nuevo factor de escala distinto al descrito anteriormente, ya que, el anterior es un dato global para todo el código del programa y que nos reduce la imagen captada por el sensor dando como resultado *'smallImg'*. Por el contrario, este factor de escala es propio de la función *"detectMultiScale"* y lo que hace es volver a reducir la imagen de entrada dada como primer argumento.

El cuarto argumento es el número mínimo de vecinos que debe tener cada rectángulo candidato a ser ROI definitiva para darlo como válido. El funcionamiento consiste en una ventana que recorrerá la imagen en busca de rostros. Al acabar, posiblemente nos encontremos con varias detecciones de rostros, pero puede que algunas pertenezcan a la misma persona (ver Figura 4-11). Este parámetro indica entonces cuántas detecciones del mismo rostro se tienen que dar como mínimo para considerarlo definitivamente como una detección exitosa. Esto nos ayudará también a reducir el número de falsos positivos.

El siguiente argumento de entrada se corresponden los flags o banderas, pero no lo vamos a usar y lo dejaremos a valor 0.

Como último argumento de entrada de la función, indicaremos el tamaño mínimo que debe tener el objeto a detectar para considerarlo como detección aceptable, que será un cuadrado de 30x30 píxeles.

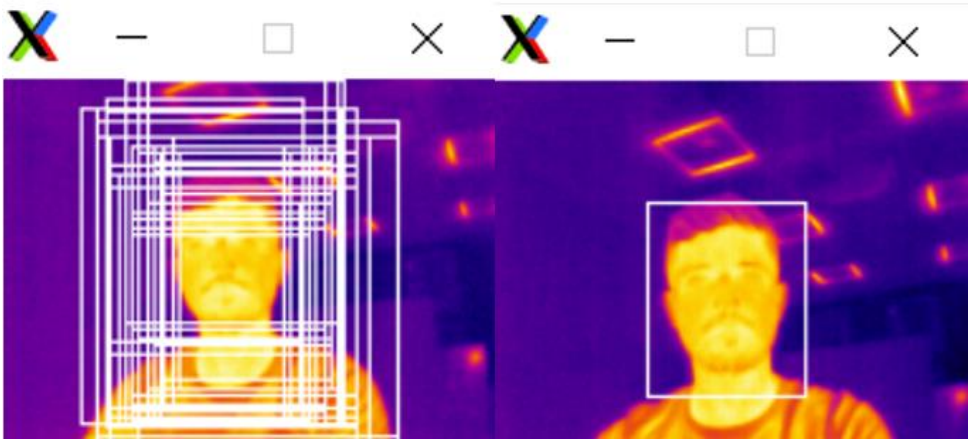


Figura 4-11: número mínimo de vecinos 0 (izquierda); número mínimo de vecinos 50 (derecha).

Esto es un ejemplo de varios frames en los que la red detecta rostro en la imagen. En vídeo se puede apreciar cómo es capaz de mostrar una detección muy continua, rápida y sin cortes incluso cuando se realizan movimientos bruscos.

Figura 4-12: Resultados de la detección usando "cascade.xml".

4.2. Método 2: Mapeo entre los píxeles de la cámara visible y los de la infrarroja.

Como hemos visto, el método anterior es el óptimo para llevar a cabo la tarea de la detección de rostros en el infrarrojo, basado en la creación de una base de datos y entrenamiento de una red neuronal a partir de dicha base de datos, compuesta de imágenes de rostros en el infrarrojo.

Viendo las carencias que presenta dicha red neuronal, se ha optado por emplear otro método para el reconocimiento de caras. La idea de este nuevo método es combinar el uso de las dos cámaras que están disponibles en nuestro sistema: cámara de visible y cámara de visión infrarroja. Es decir, realizaremos la detección de rostros con la cámara de espectro visible y luego, mediante correspondencia de puntos entre las dos imágenes, infrarroja y visible, obtendremos la detección de rostros en la imagen infrarroja. Una tarea similar a esta, utilizando también en conjunto estos dos tipos de sensores, es la que propone '*Wilfrido Rolando Guerrero Albán'* en la *"Revista Tecnológica ESPOL-RTE"* [62].

Este método es más sencillo de implementar que el anterior, ya que emplearemos ahora una red neuronal ya entrenada, proporcionada por OpenCV en su repositorio de "Haar Cascades" [54] por lo que no tendremos que hacer toda la parte de la creación de una base de datos específica para el entrenamiento ni llevar a cabo el entrenamiento de la misma, lo que nos ahorrará un poco de tiempo.

En este caso se ha usado la red neuronal del repositorio de OpenCV denominada *"haarcascade_frontalface_default.xml"*. El código que nos proporciona OpenCV escrito en lenguaje C++ [59], junto con el uso de esta red neuronal, da un buen resultado para la detección de rostros en el espectro visible. Esta implementación nos devuelve una imagen en la que se destaca la detección de caras encuadrando cada cara que aparece en la imagen con un recuadro o Bounding-Box.

Nuestro método consistirá en transportar ese Bounding-Box de la imagen de visible a la imagen de infrarrojos. Para poder hacer esto se calcularán de manera empírica los valores de offset que necesitaremos añadir debido a la colocación de ambos sensores en la estructura soporte.

4.2.1. Offset entre ambos sensores.

Como este método de detección de rostros requiere trabajar con el sensor de imagen visible y el sensor de imagen infrarroja simultáneamente, se debe iniciar la captura de imágenes de ambos sensores, tal y como se muestra en las líneas 123 y 124 del apartado Anexo D.

Vamos a considerar como primer paso de este método el cálculo de estos valores de offset. Las variables definidas como *"X_OFFSET"* e *"Y_OFFSET"* (líneas 120 y 121) serán esos valores de offset en el eje X e Y, respectivamente, que debemos aplicar a los puntos de la imagen en infrarrojos para que se correspondan con los mismos puntos de la imagen de visible. Estos puntos a los que le aplicaremos el offset correspondiente serán los puntos que forman el Bounding-Box que aparecerá en la imagen de espectro visible cuando se detecte algún rostro gracias al uso de la red neuronal específica para esta aplicación, mencionada anteriormente como *"haarcascade_frontalface_default.xml"*. Las coordenadas de este Bounding-Box las tenemos que pasar a la imagen en el infrarrojo.

El cálculo de los valores adecuados de las variables *"X_OFFSET"* e *"Y_OFFSET"* se realizará de manera empírica y se obtendrán de una forma rápida con ayuda de dos "sliders" creados con la función *'createTrackbar'* [63] (líneas 130 y 133), con los que podremos cambiar el valor de estas dos variables y comprobar el resultado en tiempo real. Al crear el slider, sólo nos deja opción de indicar el valor inicial (tercer argumento) y el valor máximo que podemos alcanzar (cuarto argumento). Si suponemos que nuestros valores de offset estarán comprendidos entre [-50, 50] píxeles, podríamos crear un slider que cambie el valor de las variables en torno a ese rango de píxeles. Pero con la función *'createTrackbar'* no podemos elegir el límite inferior que queremos que tenga el slider y este siempre tendrá un valor mínimo de 0, por lo tanto podemos crear uno que comprenda el rango [0, 100].

Se han creado también las variables *"iSliderValue1"* y *"iSliderValue2"* (líneas 129 y 132), que nos servirán para pasar el rango [0, 100] del slider al rango [-50, 50] en el que queremos que se encuentre *"X_OFFSET"* e *"Y_OFFSET"*. Una vez que tenemos los valores adecuados de *"iSliderValue1"* y

"iSliderValue2" hacemos el cambio contrario para obtener los valores reales de offset (líneas 306 y 307) mostrado en las ecuaciones 4-1 y 4-2.

$$X_{OFFSET} = iSliderValue_1 - 50; \quad (4\text{-}1)$$

$$Y_{OFFSET} = iSliderValue_2 - 50; \quad (4\text{-}2)$$

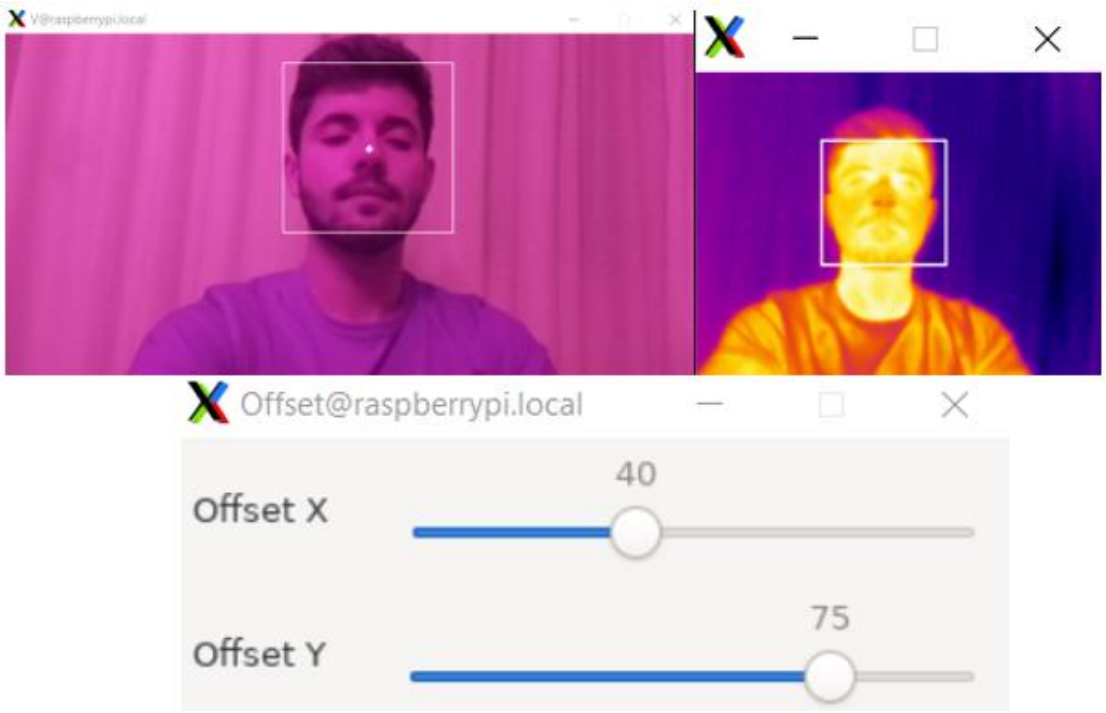


Figura 4-13:Imagen de Trackbars en sus valores correctos junto con imágenes de ambos sensores.

Para tener el Bounding-Box correctamente centrado en la imagen IR, como ocurre en la imagen del sensor de espectro visible, los valores correctos de las variables *"iSliderValue1" y "iSliderValue2"* son 40 y 75 respectivamente. Por lo que *"X_OFFSET"* e *"Y_OFFSET"*, al despejar de la ecuación vista más arriba, nos quedan con los valores (en píxeles) de -10 y 25 respectivamente.

Los valores de *"X_OFFSET"* e *"Y_OFFSET"* dependerán del tamaño de la imagen de espectro visible. Este tamaño podemos modificarlo haciendo uso de un parámetro de entrada que debemos aportarle al código de OpenCV que estamos usando. Esto lo indicamos en la línea de comandos junto con el ejecutable del programa y la red neuronal que vamos a emplear para el reconocimiento de caras:

```
pi@raspberrypi:~ $ ./peopledetect --cascade=./haarcascade_frontalface_default.xml --scale=1.3
```

Figura 4-14: Orden de ejecución mostrando nombre del programa, cascada y escala empleadas.

El parámetro *'scale'* se usa para reducir la imagen un factor de escala que nosotros indiquemos. Por ejemplo, si indicamos en la línea de comandos un valor *'scale=1.3'*, la imagen se reducirá en ambos ejes por igual. Cuanto más reduzcamos la imagen menor coste computacional supondrá el tratamiento de la misma. Esta reducción de tamaño se lleva a cabo con la función *"resize()"* [64], a la que se le indican como parámetros la matriz de entrada (frame original), la matriz de salida (frame redimensionado), el tamaño que queremos que tenga la matriz de salida, el cuál se puede indicar directamente con el propio tamaño de la matriz que queremos, o mediante factores de escala para cada eje de manera independiente. También se indica la forma de interpolación usada, y en nuestro caso usamos la interpolación bilineal. Para reducir la imagen (o ampliarla si quisiéramos) se han usado los factores de escala, aplicando el mismo factor a ambos ejes. El factor de escala que se emplea tiene la forma indicada en la ecuación 4-3.

$$f_x = \frac{1}{scale} \tag{4-3}$$

En el método de *interpolación bilineal* [65] se toman los cuatro vecinos más cercanos al píxel, interpolando en primer lugar linealmente en la dirección de las filas de una imagen y posteriormente el resultado se interpola linealmente en la dirección de las columnas. En el método de interpolación al *vecino más cercano*, por el contrario, se toma el valor del pixel más cercano al pixel que estamos calculando.

4.2.2. Bounding-Box en la imagen de infrarrojos.

Hemos visto en el apartado anterior la forma en la que podemos obtener el offset que debemos aplicar a cualquier punto que tengamos en la imagen del espectro visible para obtener el punto correspondiente en la imagen de infrarrojos. Ahora veremos el segundo paso, o paso que engloba al anterior, ya que explicaremos cómo calcular dicha correspondencia de puntos, mientras que en el apartado anterior vimos únicamente una explicación de los valores de offset y el cómo poder variar sus valores de forma sencilla.

Una vez conocidos esos valores de offset, tenemos que aplicárselos a los puntos de interés en nuestra matriz de visión normal, *"VMat"*, para encontrar esos mismos puntos en el frame correspondiente de la imagen de infrarrojos, *"IRMat"*. Esos puntos de interés serán para nosotros los que conforman cada rectángulo de cada frame de la imagen en el que se produce una detección de rostro, es decir, el Bounding-Box.

Se puede observar que lo primero que tenemos que hacer es detectar los rostros de las personas que aparezcan en la imagen. Para ello hacemos uso de nuevo de la función "*detectMultiScale*" (línea 315), ahora junto con la red neuronal de OpenCV ("*haarcascade_frontalface_default.xml*"). El uso de esta red neuronal se la indicamos al programa desde la línea de comandos. La región formada por estas caras detectadas, se guardarán en una variable definida como un vector de rectángulos, *"faces"* (segundo argumento de la función *detectMultiScale*), en la que se almacena toda la información de un rectángulo: coordenadas X e Y de la esquina superior izquierda (*"faces.x, faces.y"*) y el ancho y alto (*"faces.width, faces.heigh"*).

Una vez hayamos obtenido el Bounding-Box de la imagen de espectro visible (los cuatro puntos que se han almacenado en la variable *"faces"*) podemos pasar, mediante los valores de offset calculados anteriormente, los puntos que forman la esquina superior izquierda e inferior derecha, a la imagen infrarroja. Esto lo hacemos con cada una de las esquinas de nuestro Bounding-Box.

El método seguido consiste en dividir cada coordenada (X e Y) de las dos esquinas de cada rectángulo por una constante a la que se ha llamado *"PROP"* (línea 350), cuyo valor es la proporción que hay entre los tamaños de la imagen tomada por el sensor de espectro visible (640x320 píxeles) y la imagen tomada por el sensor infrarrojos (160x80 píxeles). Podemos ver que la proporción que existe entre cada imagen es de 4 veces menor la imagen infrarroja que la imagen visible, por tanto, *PROP = 4.*

Para obtener el punto en la imagen térmica, dividimos las coordenadas X e Y de cada uno de los elementos del vector *"faces"* por esa constante de proporción. Esto nos daría el punto correspondiente en la imagen térmica si suponemos que ambos sensores están superpuestos, es decir, con offset 0. Para ajustar ese offset no nulo entre sensores, sumamos a las coordenadas resultantes en la imagen IR el offset calculado en el apartado anterior, las variables "*X_OFFSET"* y *"Y_OFFSET*".

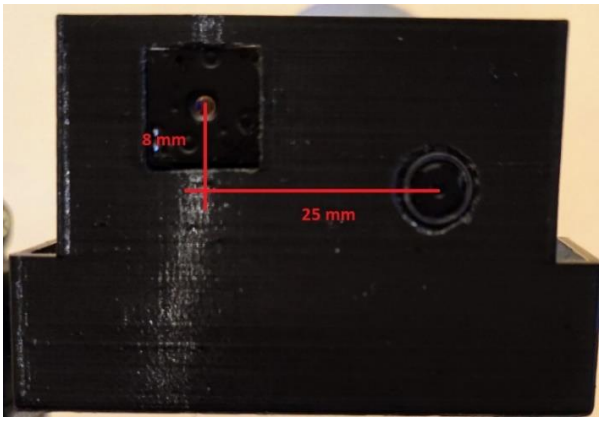


Figura 4-15:Medidas de Offset entre sensores en la pieza real.

Se puede apreciar en la Figura 4-15 que el offset entre los objetivos de cada sensor es de 25 mm en horizontal y de 8 mm en vertical. El sensor colocado más arriba es el sensor de espectro infrarrojo (cuadrado), y el otro el sensor de visible (redondo).

Además, se usa también otra constante, *"MARGIN"*, con valor 5. Esta constante representa un margen de píxeles que le damos a cada esquina del Bounding-Box en la imagen infrarrojos para agrandarlo un poco más y que encuadre por completo la cara de la persona. Para ello se debe restar este valor a la esquina superior izquierda del rectángulo, y sumarlo a la esquina inferior derecha del mismo. De esta forma conseguimos un rectángulo de mayor dimensión y un encuadre de la cara más preciso, como ocurre en la imagen de espectro visible.

Finalmente comprobamos que el rectángulo que vamos a dibujar en la imagen térmica no se salga de los límites de la matriz "*IRMat*", limitando así los valores de las esquinas de cada rectángulo a los valores máximos que pueden tomar (línea 356).

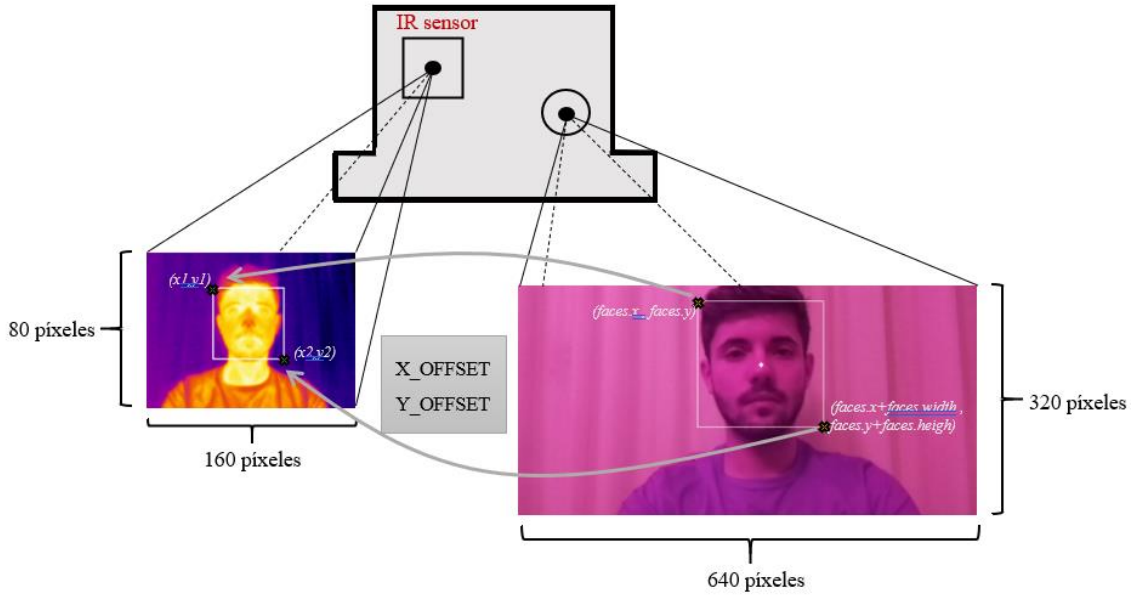


Figura 4-16:Resultado de la correspondencia de puntos entre ambas imágenes.

4.3. Ventajas e inconvenientes de ambos métodos.

Entendiendo que el método 1 se corresponde con la creación y entrenamiento de una red neuronal específica del infrarrojo, y que el método 2 hace referencia al mapeo entre píxeles de ambos sensores de imagen, veremos las ventajas y desventajas que suponen la utilización de cada uno de ellos.

El método 1 sería el óptimo, ya que, al crear una red neuronal específica para nuestra aplicación, el funcionamiento será más eficiente. Además, presenta una ventaja importante a tener en cuenta, como puede ser: el uso del sistema en situaciones de poca luz ambiente, ya que, al usar directamente el sensor de infrarrojos, se podrá detectar a la persona en cualquier situación de luminosidad que se pudiera dar. Esto es así porque el sensor de infrarrojos utilizado, descrito en el apartado Sensor Flir Lepton 3.5 de este trabajo, es un sensor que capta información del espectro infrarrojos de longitud de onda larga. Es decir, es capaz de captar la información del calor superficial de los objetos del entorno [45] por lo que incluso en una situación completamente a oscuras, se seguiría reconociendo a la persona que se enfoca.

Como características a destacar del método 1, es capaz de reconocer rostros estando la persona a una distancia máxima de unos 2 metros aproximadamente. A esta distancia empieza a detectar algunos rostros con una tasa de acierto baja. Pero si la persona se acerca al dispositivo hasta los 1.5 metros aproximadamente, el funcionamiento mejora y la tasa de acierto aumenta considerablemente, reconociendo el rostro de la persona con una tasa de acierto de más del 90%. Para detectar rostros a esta distancia, es necesario bajar un poco el valor del número mínimo de vecinos en la función *"detectMultiScale"*, ya que, al encontrarse la persona más alejada del sensor se producirán menos reconocimientos de una misma persona en el barrido de la ventana, como se explicó en el apartado 4.1.3 Esto también provoca que al acercarnos más al dispositivo, se produzcan más detecciones falsas, por lo que debemos ajustar este parámetro en función de la cercanía a la que queramos obtener las detecciones más fiables. Se ha optado por un valor de número mínimo de vecinos de 50, que nos dará buenas detecciones a distancias cercanas.

Además, si la persona usa mascarilla y gafas al mismo tiempo, el funcionamiento de la red a distancias más alejadas empeora un poco, pero a distancias cortas (1 metro o inferior) también tiene una tasa de reconocimiento elevada, como para el caso anterior. El tiempo de ejecución para este algoritmo es bastante reducido, ya que se han medido tiempos que varían entre los 1,5 ms y 2,5 ms, lo que podemos decir que se tarda una media de 2 ms en reconocer el rostro de la persona que aparece en imagen.

Pese a ser el método para reconocimiento de rostros más adecuado, contamos con varios inconvenientes en el momento de ponerlo en práctica en lo que sería su entorno de funcionamiento.

El tipo de entrenamiento que se ha realizado con esta red neuronal (a partir de una base de datos reducida) es válido para aplicaciones en las que se busque el reconocimiento de objetos que no van a cambiar de forma, o que en todas aquellas situaciones posibles en las que pudiera aparecer, lo hiciera siempre con esa misma forma. Así, con una base de datos con pocas imágenes (alrededor de 1500 aproximadamente) podemos cubrir casi la totalidad de situaciones posible, tomando imágenes de dicho objeto en diferentes posiciones y ángulos respecto del sensor. Esto no es así con la aplicación que queremos abordar en este proyecto, ya que cada rostro posee unos rasgos determinados (barba, pelo largo, bigote, etc), y hace que la cantidad de situaciones posibles que podemos encontrar sea muy grande.

El mayor inconveniente en este caso, es la necesidad de crear una base de datos lo suficientemente grande y diversa, es decir, se necesitan miles de imágenes tanto positivas como negativas de distintas personas manteniendo diferentes gestos y posiciones. Cuantas más personas participen en la creación de dicha base de datos, mayor diversidad de rostros posibles será capaz de detectar y mejor entrenada estará la red, consiguiendo así una precisión mayor.

Por este motivo, emplearemos el método 2 para la tarea de reconocimiento facial, en el que usaremos una de las redes neuronales que podemos encontrar en el repositorio de OpenCV, que han sido entrenadas con bases de datos mucho más extensas (método 2) y que cubren casi la totalidad de casos posibles. Esto nos ahorra todo el trabajo que conlleva la creación de una base de datos propia para nuestra aplicación, además de entrenar una red neuronal y guardar esa información del

entrenamiento en un fichero de tipo XML, tal y como se ha visto en el apartado 4.1.

A pesar de esta ventaja que nos aporta y, aunque resulta efectivo para nuestra aplicación, presenta varios problemas a la hora de llevarlo a la práctica. Estos problemas se deben sobre todo a la iluminación que necesita la cámara de espectro visible para funcionar correctamente. Al no tener ningún foco de luz integrado en el sistema, a modo de flash como en cualquier teléfono móvil moderno, es necesario tener una buena iluminación en el ambiente en el que se va a colocar el conjunto de cámaras, para que el sensor de espectro visible sea capaz de obtener una imagen lo suficientemente nítida y clara como para que la red neuronal que estamos empleando pueda realizar detecciones de la manera más eficiente posible. En definitiva, la cantidad de iluminación del ambiente, ya sea excesiva o insuficiente, influye sobre la capacidad de detección en el caso de emplear una red neuronal de espectro visible, por lo que es necesario encontrar el punto óptimo para que el funcionamiento sea el correcto.

Por lo tanto, y a modo de conclusión, el método 2 podría servir de reemplazo al método 1, siempre y cuando conozcamos las condiciones del ambiente en el que trabajará nuestro sistema, y estas cumplan con los requisitos de iluminación desarrollados. Es decir, si este sistema finalmente se coloca en una recepción de alguna empresa, en la que siempre se tendrá buenas condiciones de luz ambiente durante el período laboral, y su función es únicamente medir la temperatura corporal de los trabajadores a su llegada, cumplirá con éxito su misión.

Por otra parte, conociendo las características del sensor de espectro visible que estamos usando (Pi Cam NOIR v2.1, sensible a la radiación infrarroja de onda corta, SWIR), podríamos colocar iluminación mediante LEDs infrarrojos (que emitan en la banda del espectro infrarrojo de onda corta) en el lugar en el que se utilizará el sistema. Esto ayudará a que el sensor pueda obtener información del entorno en la oscuridad, gracias a la ausencia del filtro de radiación infrarroja, y así poder detectar rostros, calcular el Bounding-Box correspondiente y funcionar como lo hace en condiciones de buena luminosidad. La inclusión de LEDs infrarrojos no influye en la medida de la temperatura corporal, ya que esta medida la obtenemos con el sensor infrarrojos de FLIR, el cuál es sensible a la radiación infrarroja de onda larga (LWIR) pero no a la de onda corta (SWIR). Esta solución es una propuesta que no ha sido probada, dado que no se conoce el funcionamiento de la red neuronal en estas condiciones de iluminación infrarroja, su respuesta puede que no sea óptima.

Capítulo 5

Estimación de la temperatura en la región de interés

Como se dijo al inicio de este trabajo, el objetivo es llegar a calcular la temperatura corporal de la persona que aparece en la imagen, y de la cual se ha obtenido la región de interés gracias al uso de una red neuronal, es decir, la cara de la persona se separa del resto de la imagen en una imagen más pequeña (ROI).

Este cálculo de temperatura es posible gracias a la termografía. Como se puede leer en el libro "Termografía, guía de bolsillo" [42], cualquier objeto cuya temperatura sea mayor a 0 Kelvin (-273ºC) emite radiación infrarroja. El físico Max Planck demostró que existe una relación entre la intensidad de la radiación infrarroja que emite un cuerpo y su temperatura. Basándose en esto, se define la termografía como la medición de la temperatura superficial de un objeto con una cámara termográfica, sin necesidad de contacto.

Para calcular la temperatura corporal de una persona adulta existe la posibilidad de hacerlo en diferentes partes del cuerpo, obteniendo una medida más o menos precisa dependiendo del lugar en que se realice dicha medición. Como nos informa la enciclopedia médica "MedlinePlus" [66], y la revista de salud y bienestar "Webconsultas" [67], la temperatura corporal puede ser medida en lugares como la boca, recto, axila, oído, o frente, entre otros. Cada uno de los lugares nos ofrece mejor o peor precisión en el cálculo, siendo la axila el lugar más usado por la mayoría de personas, debido a que se trata de un lugar de fácil alcance y a que los termómetros que se emplean en esta zona suelen ser económicos. Aun así, la axila es la zona en la que se requiere un mayor tiempo a la hora de tomar la medida, y puede ser de 0.3ºC a 0.6ºC más baja que la temperatura tomada en la zona de la boca, y a su vez, la temperatura tomada en la boca puede ser alterada debido a la saliva, que no es buen conductor del calor.

Al igual que existen diferentes partes del cuerpo en las que se puede realizar la medición, también existen distintos tipos de termómetros. Tenemos los termómetros de mercurio (retirados del mercado en 2009 por el riesgo de toxicidad que suponen) y de galistán (mezcla de galio, indio y estaño), termómetros de oído (tecnología de infrarrojos), digitales (más usado por su precio reducido), de pistola (tecnología de infrarrojos), de chupete para bebés, e incluso se puede medir la temperatura usando un dispositivo móvil (resulta poco fiable) [67].

Nosotros en este trabajo nos vamos a centrar en el termómetro de pistola, el cual ha incrementado su uso en este último año a causa de la pandemia causada por el virus SARS-CoV-2. Para obtener una medida correcta de la temperatura usando este tipo de termómetros debemos colocar el plano de la lente del termómetro lo más paralelo posible a la frente de la persona. De esta forma podemos garantizar la perpendicularidad de la radiación incidente, como se muestra en la Figura 5-1 [68]. Así obtenemos de manera rápida y precisa el valor de la temperatura corporal de la persona. En este caso, el plano del objeto a medir sería el plano de la frente de la persona que se somete al análisis.

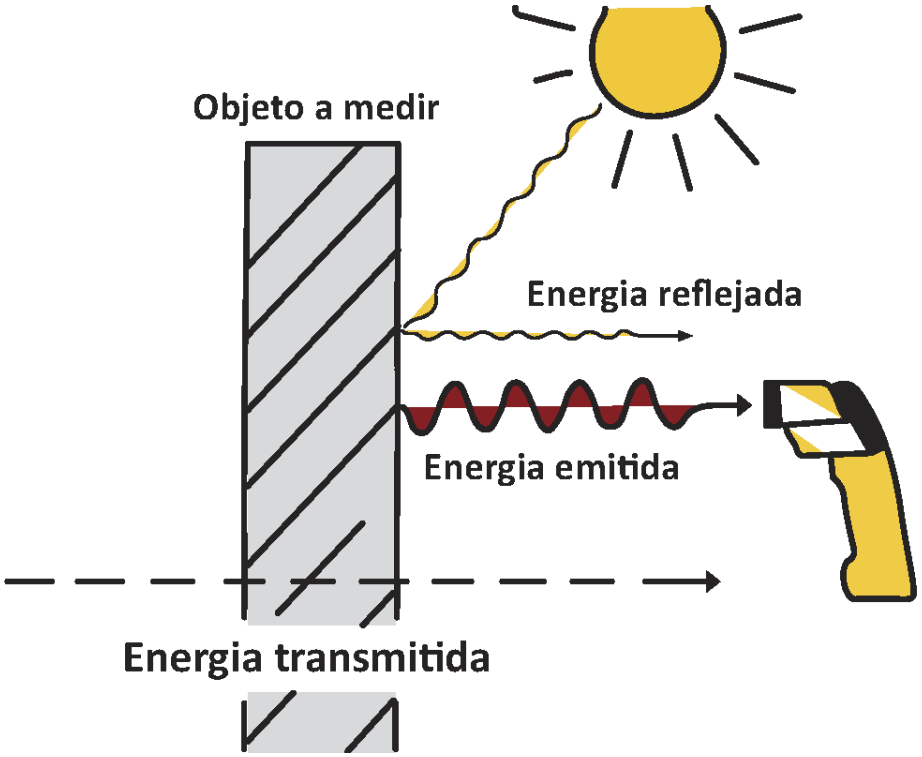


Figura 5-1: Esquema de uso de termómetro de pistola.

De forma muy parecida, en nuestro caso obtendremos la temperatura corporal con el uso del sensor infrarrojos implementado en el sistema de cámaras. La idea, de manera general será:

1) Obtener ROI de la imagen donde aparezca rostro.
2) Aplicar el método de Otsu [69] a la ROI para quedarnos solamente con la cara (eliminando el pelo de la persona y el resto de la imagen).
3) Calcular el valor medio de temperatura de la cara a partir de los valores de los píxeles que la forman y de unos parámetros previamente definidos.

El primer paso en el que obtenemos la región de interés de la imagen en la que aparece rostro ya lo hacíamos en los apartados Método 1: Creación y entrenamiento de un detector Haar Cascade y Método 2: Mapeo entre los píxeles de la cámara visible y los de la infrarroja., donde, aplicando los distintos métodos propuestos de detección de rostros obtenemos el Bounding-Box que separa la región de rostro del resto de la imagen.

A pesar de ser el método empleado en prácticamente la totalidad de pruebas rápidas que se realizan para detectar fiebre causada por la enfermedad COVID-19, la técnica de medición de la temperatura corporal en la zona de la frente tiene varias limitaciones y podría llegar a ser, en algunos casos, no representativa de la verdadera temperatura corporal. Un ejemplo sencillo de esto es el que comenta el profesor y director del Centro de Investigación en Ciencias del Movimiento Humano (Universidad de Costa Rica), Luis Fernando Aragón-Vargas, en la revista *"Pensar en Movimiento"* [68]:

"Así, una persona que se ejercita en el calor puede tener una temperatura central alta de 38°C o 39°C, mientras que la temperatura de su piel está mucho más baja, por la evaporación del sudor. Otra persona en reposo al aire libre en un día soleado se verá expuesta al calor por radiación, con lo cual su temperatura cutánea se elevará antes de que la temperatura central comience a subir. Por lo tanto, usar la temperatura de la frente para decidir si una persona tiene fiebre o no podría no ser un método suficientemente riguroso."

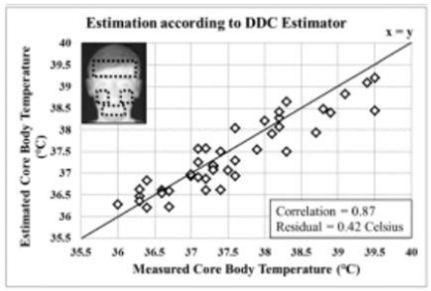
5.1. Procesado de la región de interés

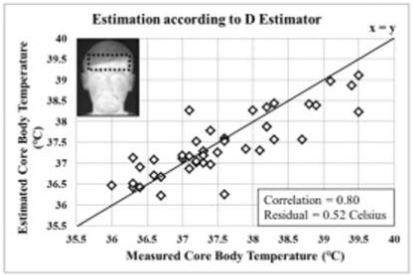
Una vez se haya obtenido la región de interés de la imagen en la que se detecta rostro, debemos analizar y procesar esa ROI. La ROI será en nuestro código un objeto de la clase *"Rect()"* de OpenCV [61]. Como podemos ver en la documentación correspondiente a la clase, para crear este objeto que contendrá las coordenadas del Bounding-Box que encuadra la cara, necesitamos aportarle dichas coordenadas como argumentos.

En la línea 369 se muestra la línea de código en la que se crea el objeto que será la región a analizar para el cálculo de la temperatura corporal. Los argumentos de la función son: *Rect(Esquina superior izquierda X, Esquina superior izquierda Y, Ancho, Alto)*. En nuestro caso, *"x1", "y1", "x2" e "y2"* son las coordenadas calculadas de la forma que se explica en el apartado 4.2.2 Bounding-Box en la imagen de infrarrojos. Este objeto de nombre ROI será el que, a su vez, se le pasará como argumento a la función que se encargue de calcular la temperatura de la persona.

La precisión de la medida de la temperatura corporal realizada únicamente en la región de la frente es baja. En Silawan et al. [70] se realizan diversas pruebas con distintas regiones de interés, comparando los resultados con el termómetro de oído Braun ThermoScan PRO 4000. En la Figura 5-2 se muestran los resultados obtenidos usando la ROI de la frente (D), de la frente junto con la zona de los cachetes (DC), y la formada por la frente, cachetes y boca la ROI (DDC). Esta última es la opción que presenta una mayor precisión. Basándonos en estos resultados, en este proyecto tomaremos como región de interés todo el rostro de la persona. Al fin y al cabo, utilizar todo el rostro de la persona se puede considerar una aproximación del método descrito, en el que se utiliza la frente, cachetes y boca.

Estimator	*Sensitivity*	*Specificity*
D *(Conventional)*	78.9	87.0
DC	78.9	87.0
DDC *(Proposed)*	84.2	91.3





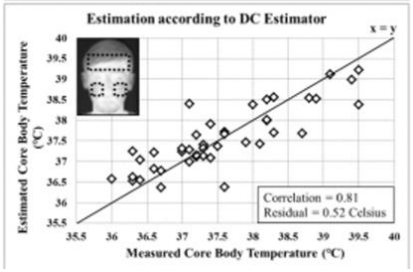


Figura 5-2: Resultados de la estimación de la temperatura usando distintas ROI, y tabla de resultados de sensibilidad y especificidad de cada método.

El primer paso en el proceso de medición de la temperatura corporal en nuestro sistema consistirá en un filtrado de la ROI para eliminar ruido y la eliminación de zonas en la ROI candidatas a distorsionar la medida. El filtrado de ruido se realiza empleando un filtro Gaussiano. A continuación, se implementará el método de Otsu [69] con objetivo de eliminar regiones de la imagen donde el valor de sus píxeles sean muy inferior al resto de píxeles que la rodean y puedan alterar el valor de la medida. Un ejemplo de este tipo de regiones son los cristales de unas gafas de sol. Los cristales son opacos a la radiación infrarroja, ya que absorben toda la radiación de este rango y resultarían en zonas de píxeles completamente oscuros. Cualquier objeto que se encuentre detrás del cristal, aunque tenga una temperatura elevada, no se verá en una imagen termográfica [71].

5.1.1. Filtrado Gaussiano de la imagen

El primer paso será filtrar la imagen aplicando un filtro de Gauss. Con esto conseguiremos eliminar de la imagen el ruido Gaussiano que pudiera aparecer y que afecta al valor de los píxeles. Este tipo de ruido es el que se puede producir en circuitos electrónicos debido al ruido de los sensores, falta de iluminación o altas temperaturas [72]. La función Gaussiana es la indicada en la ecuación 5-1:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (5\text{-}1)$$

En la ecuación 5-1, "σ" se corresponde con la desviación estándar y de su valor dependerá el grado de suavizado de la imagen. A mayor valor de "σ", mayor cantidad de ruido conseguimos eliminar, pero también se difuminará más la imagen. Los valores de "x" e "y" son las coordenadas del pixel dentro de la máscara Gaussiana. En nuestro caso se utilizará una máscara Gaussiana de 3x3 píxeles, en la que se da mayor ponderación a los píxeles más cercanos al píxel central. La Figura 5-3 muestra la máscara Gaussiana empleada, en la que se aprecia la mayor importancia que se le da a los píxeles cercanos al pixel central.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 5-3: Kernel Gaussiano 3x3.

Para obtener el valor del pixel suavizado se realiza una operación de convolución de la imagen original y la máscara que se esté usando [73]. La ecuación 5-2 representa la operación de convolución.

$$g(i,j) = \sum_{-a}^{a}\sum_{-b}^{b} w(s,t) \cdot f\,(i+s, j+t) \qquad (5\text{-}2)$$

Donde:

- "a" y "b" se refieren al radio (número de columnas que definen la mitad del ancho de la matriz) de la máscara Gaussiana empleada. Una máscara de 3x3 como la que se usará para el filtrado en nuestro caso tendrá radio 1, mientras que una máscara 5x5 tendrá un radio de 2 píxeles (número de columnas o filas que rodean al pixel central). El pixel central de la máscara corresponde a los valores de a=0 y b=0 [73].
- w(s,t) es la máscara Gaussiana.
- f(i,j) es el pixel correspondiente en la imagen original.
- g(i,j) será el valor del pixel resultado de la convolución.

Para eliminar el ruido en la imagen disponemos de varios tipos de filtros, como pueden ser: filtro de la media, filtro de la mediana, o filtro Gaussiano, entre otros muchos. El filtro de la mediana es muy útil para eliminar el ruido de tipo sal y pimienta (puntos de nivel de intensidad muy diferentes a los valores de los puntos de alrededor), que puede producirse por un mal funcionamiento de los sensores de cámara o errores durante la digitalización de la imagen, pero en nuestro caso, este tipo de ruido es prácticamente nulo (ver Figura 4-13). Trataremos entonces de suavizar la imagen reduciendo el ruido Gaussiano, que simula el ruido generado por dispositivos electrónicos durante la adquisición de imágenes [74]. Para reducir el ruido Gaussiano podremos utilizar los otros dos tipos de filtros nombrados, el filtro de la media y el filtro Gaussiano.

Usaremos el filtro Gaussiano en lugar del filtro de la media porque a pesar de que ambos tienen un comportamiento similar (disminución de la nitidez, aumento de borrosidad y pérdida de detalles), el filtro Gaussiano produce un suavizado más uniforme que el filtro de la media [72]. Podemos ver el resultado del filtrado de la imagen en la Figura 5-4.

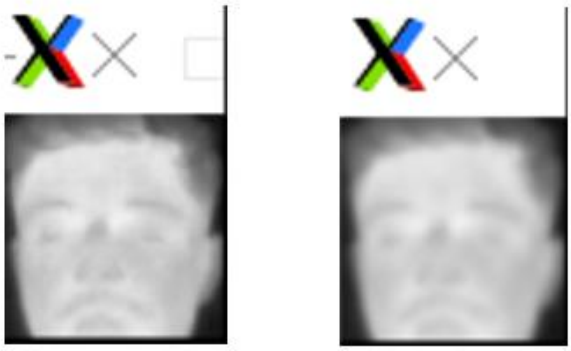


Figura 5-4: ROI sin aplicar filtro de Gauss (izquierda); ROI aplicando filtro de Gauss (derecha).

Todo este procedimiento descrito está programado en lenguaje C++ en el Temperature.cpp. Concretamente la función "GaussianBlur", en la *línea 43*, es la encargada de realizar el filtrado de la imagen. Podemos ver la información correspondiente a esta función y su funcionamiento en la web de OpenCV [75]. Como primer argumento recibe la imagen a la que se le aplicará el filtrado, que en este caso será la ROI (variable *ImgROI*); el segundo argumento hace referencia a la variable en la que guardaremos el resultado de imagen filtrada (*ImgROIBlur*) y como tercer argumento indicaremos el tamaño de la máscara empleada en el suavizado (tamaño 3x3).

Una vez tenemos la imagen filtrada y sin ruido podemos pasar a la segunda etapa del procesado de la imagen previa al cálculo del valor de temperatura corporal. Este segundo paso consistirá en segmentar la imagen aplicando el método de Otsu.

5.1.2. Segmentación de la imagen aplicando el método de Otsu.

El método de Otsu es un método que nos ayuda a segmentar imágenes. Se suele utilizar este método para separar objetos del fondo de la imagen, es decir, calcula de forma automática el umbral (o valor de nivel de gris) en una imagen en escala de grises óptimo para separar los objetos del fondo de la imagen. Dicho de otra forma, el umbral que produzca la mejor separación entre clases, o que maximice la varianza entre dichas clases, ese será el umbral óptimo para la segmentación [69], entendiéndose por clases el conjunto de píxeles de valores cercanos entre ellos. El método de Otsu debe utilizarse con imágenes bimodales para obtener segmentaciones más precisas. Una imagen es bimodal cuando en su histograma podemos ver dos picos dominantes [76]. Vamos a analizar el histograma de una de las imágenes captadas por el sensor a la que ya se le haya aplicado el filtro Gaussiano, y aplicando Otsu veremos qué valor umbral es el óptimo para la segmentación de la imagen. Para el cálculo del histograma utilizaremos MATLAB.

Figura 5-5:Imagen de la que obtendremos el histograma.

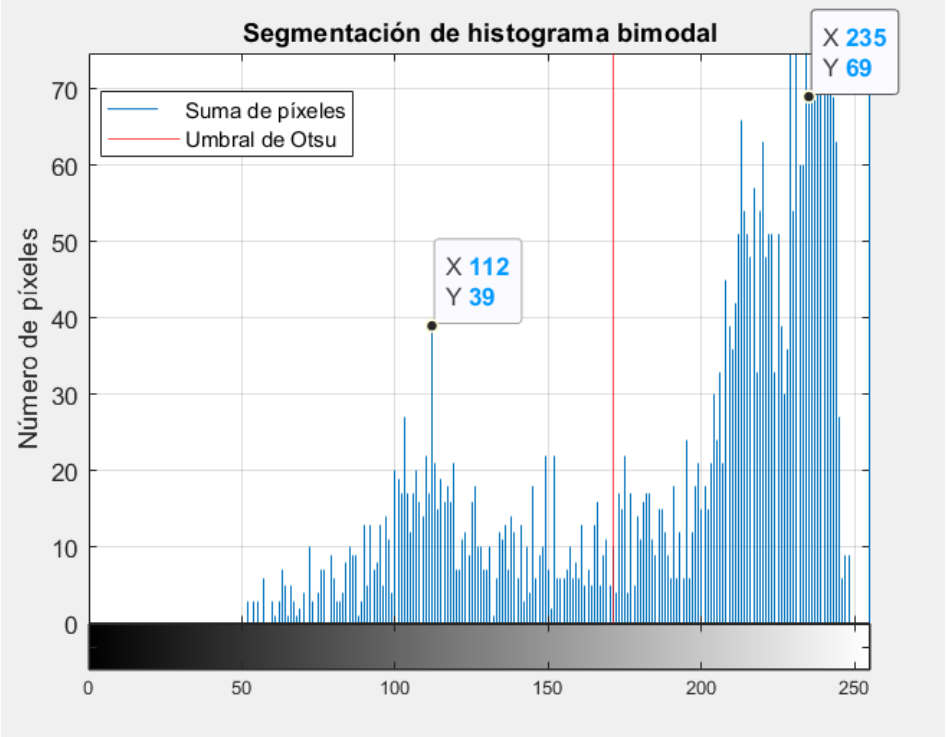


Figura 5-6: Histograma de la imagen suavizada y umbral de segmentación (marcado en rojo).

En la Figura 5-6 podemos ver el histograma bimodal del que se hablaba líneas arriba y que pertenece a la Figura 5-5. Su principal característica es la aparición de dos picos diferenciables en torno a dos valores de nivel de gris. En este caso, el método de Otsu nos ayudará a obtener el valor adecuado para segmentar la imagen (valor umbral), quedándonos únicamente con los píxeles cuyo valor está más cercano al pico de valor más alto, es decir, aquellos píxeles que superen el valor umbral.

Con esto eliminaremos de la imagen las zonas frías que pudieran afectar al dato de la temperatura que se calculará luego. El resultado de la segmentación podemos verlo en la Figura 5-7.

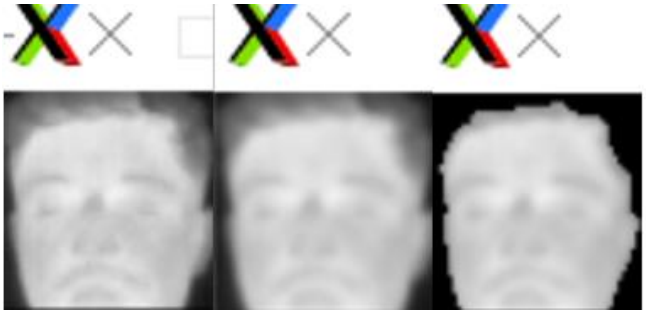


Figura 5-7:ROI detectada (izquierda); Imagen suavizada(centro); Imagen segmentada (derecha).

Hemos conseguido eliminar la zona del pelo (aunque no llega a eliminar el pelo de las cejas, por ejemplo), que pueden alterar la medición del valor de temperatura. También se eliminarán las zonas más oscuras (frías), como el fondo de la imagen, ya que son zonas que no interesan a la hora de dicho cálculo.

Para aplicar el método de Otsu, implementado en la función *"ApplySegmentation"* (*ver Temperature.cpp*), usamos con la función *"threshold" (ver línea 97)* de OpenCV [77]. El primer argumento que recibe será la imagen sobre la que queremos aplicar dicha segmentación (*"ImgROIBlur"*), y el segundo argumento es la variable en la que guardamos el resultado de la segmentación, es decir, la imagen segmentada (*"ImgSegmented"*). El tercer y cuarto argumento no los usaremos y le daremos valor 0, ya que con el quinto de los argumentos tendremos suficiente para indicar el tipo de segmentación que queremos hacer. Con el argumento "*THRESH_TOZERO | THRESH_OTSU"* indicamos que, utilizando el valor umbral (resultado del método de Otsu), vamos a segmentar la imagen manteniendo el valor de aquellos píxeles que superen el umbral. El resto de píxeles que no superen dicho valor pasarán a ser 0 y así tendremos la imagen segmentada [77].

5.2. Calibración térmica del sensor infrarrojo

Una vez hayamos procesado la imagen formada por la ROI, podemos centrarnos en el siguiente paso, en el que obtendremos valores de temperatura corporal. Para ello, lo que haremos será calibrar el sensor infrarrojo para convertir sus salidas en valores absolutos de temperatura. Es decir, pasaremos de tener valores de radiación IR a valores de temperatura.

Para calibrar este sensor nos basaremos en el método seguido en uno de los trabajos realizados recientemente en el Instituto de Microelectrónica de Sevilla por Juan Antonio Leñero Bardallo y Rafael de la Rosa Vidal, entre otros autores [78]. Se basa en un sistema para la adquisición y procesamiento de imágenes médicas en las bandas visible e infrarroja. Para llevar a cabo dicha calibración usaron un termómetro IR, el Melexis MLX90614D. Con él, se midió la temperatura en dos zonas distintas del cuerpo.

Según FLIR, hay una dependencia lineal entre el valor de radiación de un pixel y su valor de temperatura. Por lo que, conociendo el valor de temperatura de dos puntos a una misma distancia del sensor (T1 y T2), y sus respectivos valores de radiación (R1 y R2) en la banda del infrarrojo de onda larga (LWIR), podemos conocer dicha relación lineal.

Resolviendo el sistema de ecuaciones 5-3 se obtienen los valores de *a* y *b.*

$$T_1 = a \cdot IR_1 + b$$
$$T_2 = a \cdot IR_2 + b \qquad (5\text{-}3)$$

Se puede decir entonces que la relación existente entre valores de radiación y temperatura vine dada por la ecuación 5-4. Aquí, el valor de IR será el valor medio de todos los píxeles que forman la ROI, calculado como podemos ver en la *línea 51* del *Temperature.cpp.*

$$T = a \cdot IR + b \qquad (5\text{-}4)$$

En esta serie de pruebas también estudiaron el error cometido en la medida en función de la distancia

al objeto a analizar. Este error, entre otros factores, se debe a la caída cuadrática de radiación IR con la distancia, y el tamaño del objeto. Para cuantizar este error se comenzó a medir la temperatura a una distancia inicial de 40 cm. En la Figura 5-8 podemos ver el error absoluto de temperatura cometido con respecto a la medida realizada a la distancia inicial. De esta forma, conocemos el error que cometemos en la medida en función de la distancia a la que se encuentre el objeto de estudio, en nuestro caso la persona. Para corregir estas variaciones del valor de temperatura, se dará una solución con la que podremos calcular la distancia a la ROI, pero que en este caso, no se ha llegado a implementar. Se dejará como posible línea de trabajo futura (ver Capítulo 7).

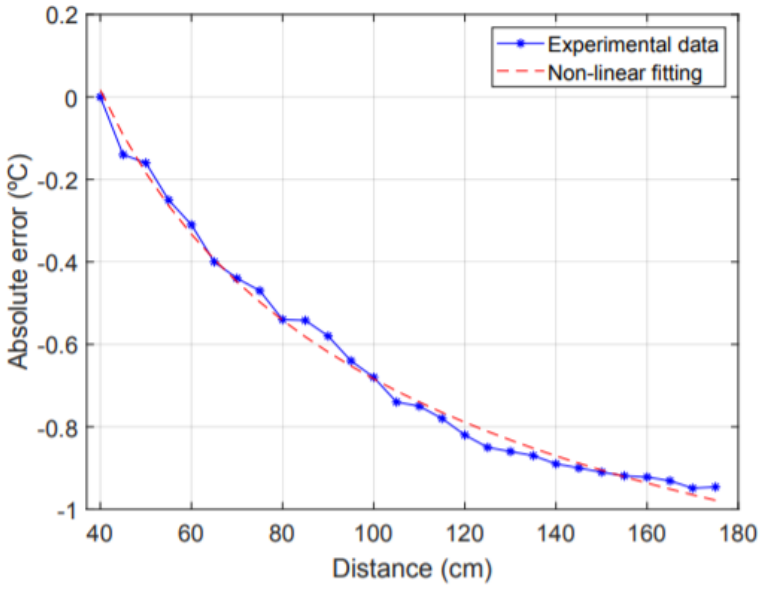


Figura 5-8: Error absoluto de temperatura frente a distancia entre objeto y sensor. Curva azul: resultados experimentales. Curva roja: curva de mejor ajuste.

Como el sensor que estamos usando para este proyecto es el mismo que usaron en el proyecto citado (el sensor FLIR Lepton 3.5 [46]), nos servirán los mismos datos para la calibración. En las pruebas realizadas en el IMSE obtuvieron los resultados de $T_1 = 36.0$ (temperatura corporal tomada en el primer punto del cuerpo) y $T_2 = 36.04$ (temperatura corporal tomada en el segundo punto), para los valores de radiación $R_1 = 8250$ y $R_2 = 8400$ (valores en formato de 14 bits), resolviendo el sistema de ecuaciones 5-3.

Tal y como podemos ver en la función *"IRtoTempComversion"* del apartado de código correspondiente al Anexo B, esta función nos devuelve un valor de temperatura resolviendo la ecuación 5-4 y aplicando un parámetro de conversión. Este parámetro (línea 79) es necesario para convertir el valor de radiación de 14 bits a un valor de 8 bits, ya que el sensor se ha calibrado con valores de radiación de 14 bits y, en este caso, nosotros trabajamos con datos de 8 bits.

En la Figura 5-9 vemos el resultado de la calibración del sensor. Se muestra el resultado de dos medidas de temperatura corporal a diferentes distancias. La primera de ellas (izquierda) a una distancia de unos 40 cm, nos da como resultado una temperatura de 36.5ºC; en la segunda medida realizada (derecha) a unos 180 cm, se ha obtenido 35.3ºC. Esto supone una diferencia de 1.2ºC. A una distancia de 180 cm, el error cometido en la medida está en muy cercano a 1ºC en las condiciones en las que se realizó la calibración del sensor, por lo que sigue un comportamiento muy similar al mostrado en la Figura 5-8.

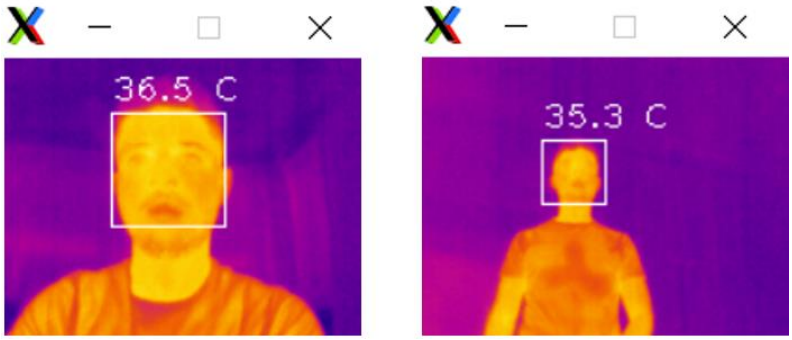


Figura 5-9: Medida de temperatura corporal a 40 cm de la persona (izquierda). Medida de temperatura corporal a 180 cm de la persona (derecha).

Cuando en algún caso se detecte una temperatura corporal superior a los 38ºC, el sistema debe avisar a la persona que se encuentre supervisando la tarea del mismo, como puede ser el conserje de la recepción de alguna empresa en la que se realice esta medición de manera rutinaria. El aviso consistirá en mantener "congelada" la imagen y el sistema con el rostro de la persona que ha superado dicho valor de temperatura. Será sólo en este momento cuando aparezca la imagen recogida por el sensor, ya que, de no detectar valores elevados de temperatura, la imagen no se mostrará para reducir el coste computacional que supone mostrar el vídeo en tiempo real.

Capítulo 6

Diseño y control de la plataforma móvil

Una de las funcionalidades que se añaden al sistema de cámaras es la capacidad de seguimiento de las caras detectadas, como hacen las cámaras de tipo PTZ *(pan-tilt-zoom)* [79], las cuales son capaces de rotar en el plano horizontal (*panning*), en el plano vertical (*tilt*) y alejarse o acercarse a través del *zoom*. En nuestro caso, el sistema no contará con la posibilidad de acercar la imagen mediante zoom, pero sí podemos dotarlo de la habilidad de rotar en los planos vertical y horizontal. Así podremos hacer un seguimiento automático de los rostros detectados en la imagen.

Para dotar al sistema de cámaras con la capacidad de seguimiento de rostros necesitaremos instalar un par de servomotores. Uno de ellos se encargará del movimiento en el plano horizontal, y el segundo realizará el movimiento en el plano vertical. Así cuando se detecte alguna cara en la imagen, se intentará orientar el sistema para que la cara quede en el centro de la imagen.

6.1. Diseño de la plataforma.

Como acople a los motores y a las cámaras se ha propuesto un diseño mecánico en el que se fusiona la apariencia de una cámara de vigilancia con un sistema gimbal de dos ejes, y no de tres ejes [80, 81] como los que se usan para la estabilización de teléfonos móviles o cámaras profesionales, por ejemplo. El diseño de las diferentes piezas se ha llevado a cabo en el programa de diseño Catia V5 .

En la Figura 6-1 se muestra el diseño propuesto para anclar la placa de Raspberry Pi 3B y ambos sensores de cámara.

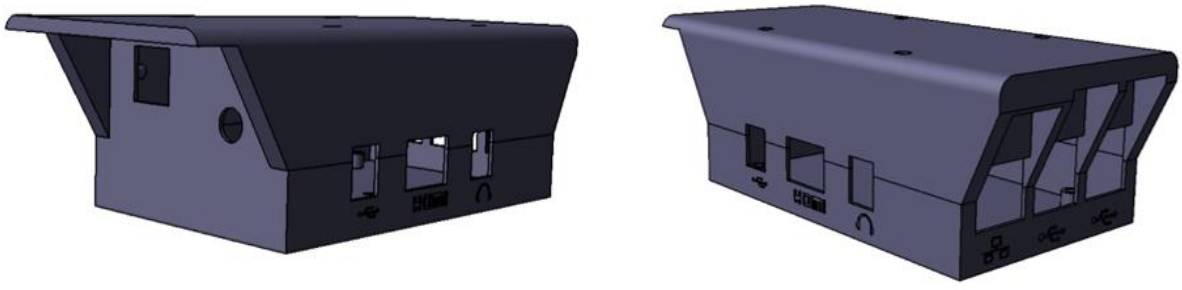


Figura 6-1: Vista de plataforma de cámaras y Raspberry Pi 3B.

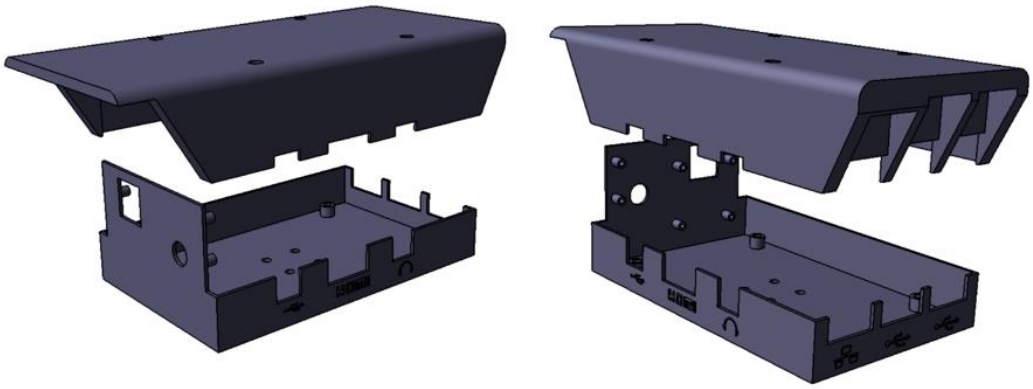


Figura 6-2: Vista explosionada de plataforma de cámaras y Raspberry Pi 3 B.

Figura 6-3: Vista de pieza real de plataforma de cámaras y Raspberry Pi 3 B.

Para la plataforma que soportará el sistema de cámaras junto con los motores se ha diseñado el conjunto de piezas mostrado en la Figura 6-4 y la Figura 6-5. El objetivo es, como se muestra en la Figura 6-6, es crear una plataforma sobre la que colocar el sistema de cámaras de la Figura 6-3 permitiendo modificar su pitch y su yaw. Estos ángulos representan las rotaciones en los ejes Y, Z, respectivamente. Para dotar de movimiento a la estructura se usarán dos servomotores.

Figura 6-4: Soporte base y anclaje del motor 1.

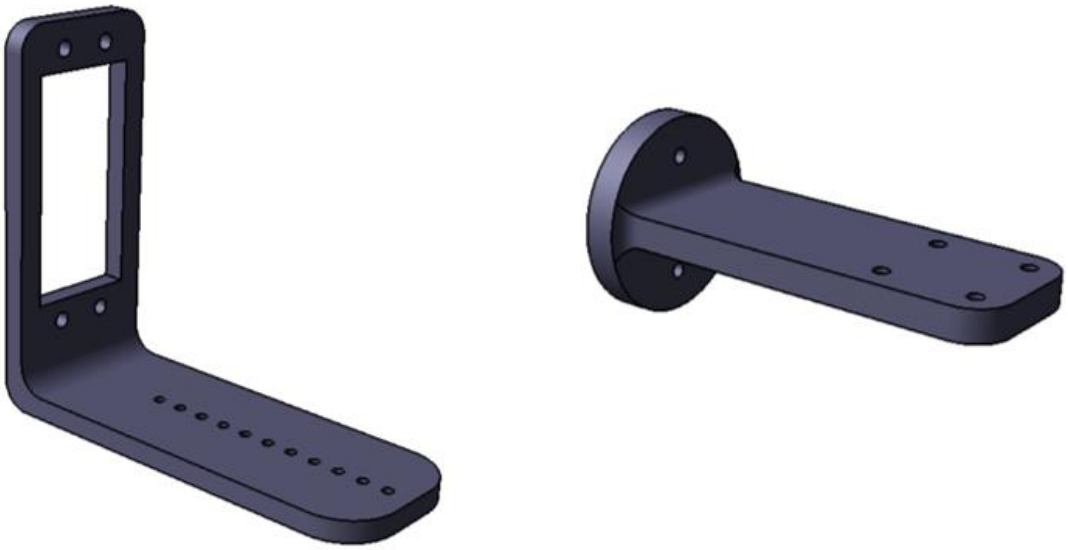


Figura 6-5: Anclaje del motor 2 (izquierda) y soporte del sistema de cámaras y Raspbery Pi 3B (derecha).

Podemos ver en la Figura 6-5 (izquierda) una serie de orificios alineados. Estos orificios serán el lugar de anclaje de esta pieza con el motor del eje de yaw. Todos estos orificios alinados y separados entre ellos una distancia de 5 milímetros nos servirá a la hora del montaje final, donde tendremos que alinear la estructura superior e inferior de la plataforma para hacer que el centro de masas de la plataforma coincida con el centro geométrico de la misma. Con estos orificios tendremos la opción de colocar la plataforma en varias posiciones, para ajustar ese balance de pesos. A esto se le denomina "Balancear la estructura". En la Figura 6-6 se puede ver dónde deberían coincidir el centro de gravedad y el centro geométrico de la carga a colocar, que en este caso será la pieza que contiene la Raspberry

Pi 3 B junto con las cámaras (Figura 6-1):

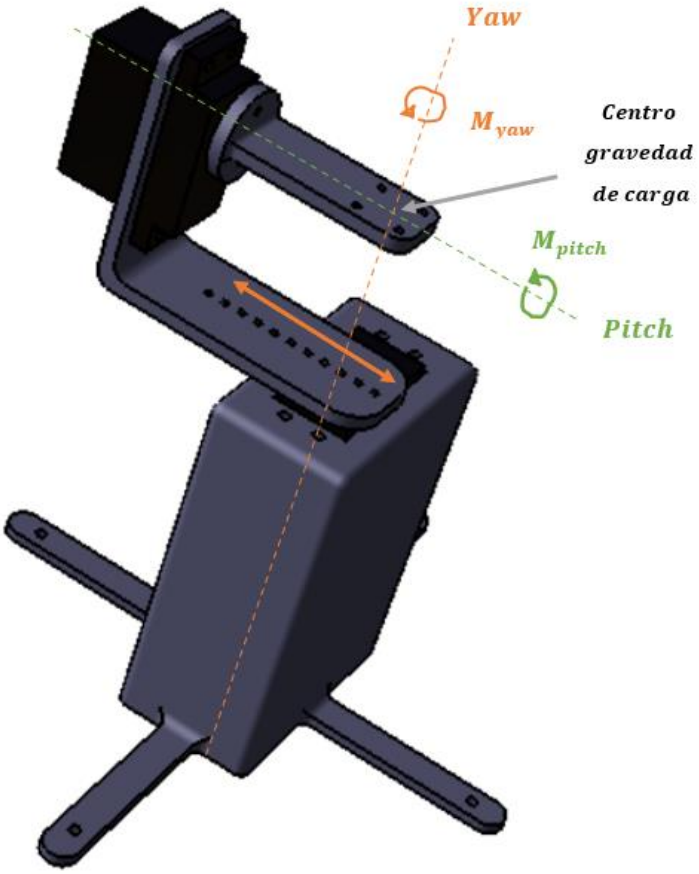


Figura 6-6: Diseño inicial de la estructura en la que se han marcado los diferentes ejes de relevancia (Pitch y Yaw) y los momentos que se van a crear (MPitch y MYaw).

En esta imagen (Figura 6-6), la flecha marcada en color naranja se corresponde con la línea de orificios de la pieza. Para centrar correctamente la carga en la plataforma se debe hacer coincidir, de manera lo más precisa posible, el centro del anclaje de la carga con el punto de corte de los ejes Pitch y Yaw. Para ellose ve que debería desplazarse la pieza soporte del motor superior sobre el raíl naranja hasta que el centro del anclaje de la carga (carga situada en la pieza soporte superior, Figura 6-5 derecha) quede lo más cerca posible del corte entre ambos ejes.

Algo que se ha aplicado a la estructura que se ha diseñado es la sencilla idea, o forma de comprender el comportamiento físico de una estructura, que nos muestra el profesor *'Miguel de Vicente López'* en su breve presentación *"Estabilidad y Centro de Gravedad"* [82]. Se hace hincapié en las distintas formas de aumentar la estabilidad de una plataforma para que esta no vuelque como por ejemplo: aumentar el peso de la base para bajar el centro de gravedad; haciendo la base más grande; o anclando el objeto al suelo. En nuestro caso, se han llevado a cabo las dos últimas ideas, proporcionando a la base una especie de "patas" o "brazos" a cada una de las caras de la pieza, que consiguen aumentar de tamaño esta base, y además, se pueden anclar a través de los orificios en los extremos de cada una de estas "patas", quedando la estructura completamente fija y estable.

Figura 6-7: Montaje de la plataforma incluyendo modelos de los motores.

Figura 6-8: Montaje de la plataforma incluyendo modelo de los motores y sistema de cámaras.

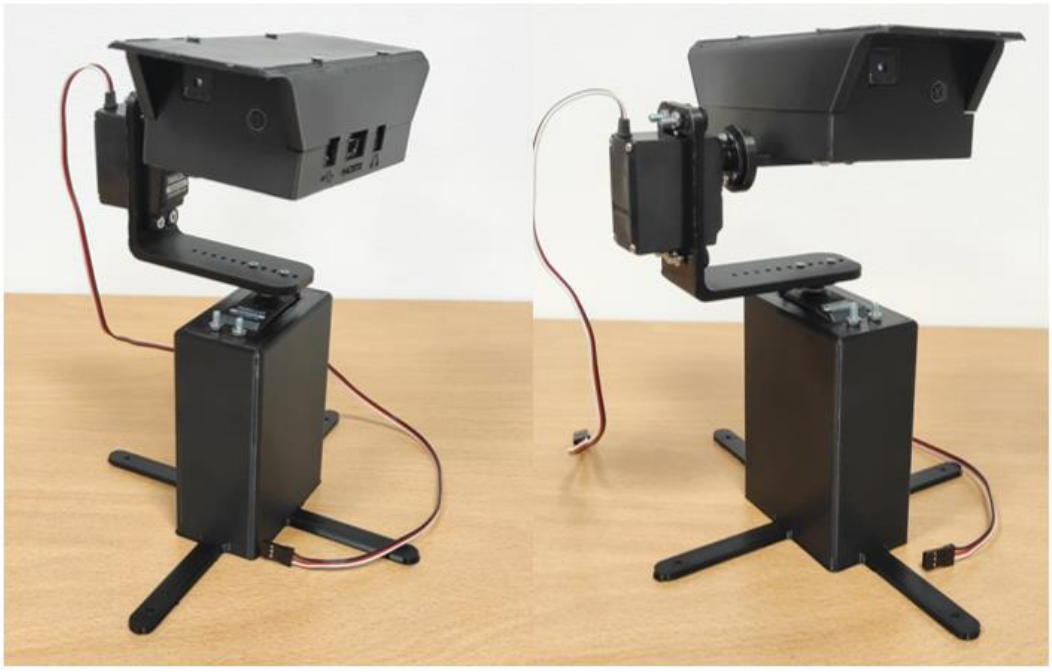


Figura 6-9: Vista del resultado final del montaje completo.

6.2. Control de la plataforma.

Para controlar la posición de la plataforma móvil, es decir, los valores de Pitch y Yaw, aplicaremos un control PI a cada uno de los servomotores. El servomotor empleado es el Parallax Standard Servo, mostrado en la Figura 6-10 [83]. El sistema deberá alinearse con la ROI, formada por el rostro de la persona detectada, de manera que esta quede en el centro de la imagen. Para ello, tomaremos como referencia el pixel central de la imagen.

Figura 6-10: Parallax Standard Servo.

El objetivo es reducir al mínimo posible la diferencia entre ambos puntos centrales: el punto central de la imagen (referencia) y el punto central de la ROI (punto que debe seguir la referencia). Para realizar este control se realiza una implementación software de un controlador PID (ver Anexo C), del cual sólo se usará la acción proporcional e integral, es decir, el control será de tipo PI. Este tipo de control ha sido suficiente para alcanzar las características dinámicas y de estabilización requeridas por la aplicación.

6.2.1. Control PI

En el Anexo C podemos ver los dos ficheros (PID.h y PID.cpp) creados para implementar la clase PID encargada de controlar los servomotores de Pitch y de Yaw. En PID.h se definen los distintos métodos y atributos de la clase, y su constructor. En PID.cpp se inicializan dichos métodos. Mediante el método Reference indicamos el valor de la referencia a seguir y el que debe tomar como último valor de la señal de control (LastOutput). El primer valor de LastOutput será aquel con el que el sistema llegó a la posición inicial. Posición en la que se establece el sistema al iniciar su funcionamiento.

El método Update nos servirá para definir el comportamiento del controlador. Lo primero que haremos será comprobar si se ha sobrepasado el rango de error que consideraremos como válido. Este rango de error admisible es necesario debido a que la precisión de los servomotores no es muy elevada y cada mínimo cambio de posición del motor supone una gran variación de píxeles (dependiendo también de la distancia a la que se encuentre la ROI del sensor). En este caso, tras un ajuste fino del valor, se ha considerado un offset (error en píxeles admitido) de 50 píxeles. Entonces el rango de error viene definido por [-offset, +offset].

De esta forma, sólo se calculará una nueva señal de control cuando el error cometido tenga un valor absoluto de 50 (ver línea 44). Mientras tengamos un error menor, no se ejecutará una acción de control y la plataforma se mantendrá en la misma posición.

Al valor de la señal de control calculado, le aplicaremos un filtro (ver línea 49), con el que conseguiremos eliminar ruido de alta frecuencia, evitando cambios bruscos en la posición del motor. Aunque esto perjudica a la dinámica del sistema, evita situaciones en las que la ROI sale del plano de la imagen y se pierde su seguimiento. El filtro aplicado es el filtro exponencial EMA [84] (Exponential Moving Average), muy usado en electrónica digital por sus buenos resultados e implementación sencilla mediante la ecuación 6-1.

$$A_n = \alpha M + (1 - \alpha) A_{n-1} \tag{6-1}$$

Donde A_n es el valor filtrado, A_{n-1} el valor filtrado anterior, M es el valor de la señal sin filtrar y α es un coeficiente entre 0 y 1. Del valor de este coeficiente dependerá el comportamiento del filtro exponencial. Dando como resultado, para α igual a 0, un valor nulo a la salida del filtro, y para α igual a 1, obtendremos la señal sin filtrar. El valor de este coeficiente dependerá de las características del sistema, siendo lo más habitual, valores entre 0.2 y 0.6. En nuestro caso, tras un proceso empírico, se ha obtenido un comportamiento adecuado con un valor de 0.4.

Una vez calculada la señal de control a aplicar a cada servo (llamada EMA_LP en el código) comprobaremos ahora que se encuentra entre los límites en los que puede trabajar dicho servomotor (ver líneas 52 y 58). Para el servomotor de Yaw se tiene disponible todo el rango de operación que este es capaz de dar, es decir, un giro de 180º, siendo el valor de 0º el correspondiente a enviar un pulso de 0.4ms, y el valor de 180º correspondiente a un pulso de 2.2ms (valores obtenidos mediante ensayo del servomotor), configurado para un intervalo de 20ms entre cada pulso. El servo de Pitch, siendo el mismo modelo, se ha limitado más el rango debido a restricciones mecánicas y a que no se necesita tanto rango de movimiento, ya que nunca necesitaremos que la plataforma apunte, por ejemplo, directamente al suelo , a 0º. Por esto el rango operativo para el servo de Pitch comprende pulsos que van de 1.3 a 2.0ms. Si se sobrepasa algunos de estos umbrales, el valor de la señal de control pasará a ser el valor límite que se ha superado.

En un controlador PI, la acción integradora decimos que tiene memoria, por el hecho de guardar datos de error anteriores. Esto provoca que la señal integradora se sobrecargue cuando se sobrepasan los límites del actuador (límites de los servomotores). Cuando esto ocurre, se debe esperar a que se descargue la acción integral antes de que el control pueda actuar. Para evitar esta sobrecarga, y que la señal de control continue creciendo, aplicaremos una estrategia Anti-Windup. De esta forma, la señal

integradora no saturará y el control podrá actuar de forma inmediata cuando desaparece la saturación. La técnica seguida consiste en dejar de actualizar la acción integral con valores de error pasados cuando se satura el sistema (ver desde línea 52 hasta línea 66).

En la secuencia de control, primero se calcula el valor de salida del controlador PI de Yaw y luego el de Pitch (líneas 210 y 211), y se aplican a los servos en el mismo orden (líneas 223 y 238). Para que no se produzcan fallos de temporización, es decir, que la señal de control se aplique a los servomotores antes de volver a procesar un nuevo frame, se deben introducir las esperas y retardos adecuados. Aquí nos podemos encontrar con el caso en el que se detecta rostro, y aquel en el que no se encuentre rostro, en el que el tiempo de procesamiento será menor. Midiendo los tiempos de procesamiento para ambos casos se ha comprobado que cuando se detecta rostro, el peor de los tiempos obtenidos suele estar en torno a los 120ms, y cuando no se detecta rostro, el tiempo disminuye hasta los 80ms. En el caso en el que se detecten muchos rostros en una misma imagen (más de 5),el tiempo de procesamiento puede llegar hasta los 150ms, aunque para nuestra aplicación, este caso no será habitual. Para cubrir todas las posibilidades, introducimos una espera que sea suficiente para el peor de los casos (cuando se produce detección de rostro). Definiremos esta espera como el tiempo de muestreo del control PI de los servomotores (definido a 200ms) menos el tiempo de procesamiento de la imagen. Por lo que realmente la espera será de entre 80ms como mínimo y 120ms como máximo.

Cabe destacar que cuando se dé el caso en el que se detecte más de un rostro en una misma imagen (línea 346), el sistema se quedará en la posición en la que permanecía anteriormente, es decir, no se realiza una selección de rostro a la que seguir, ya que para la aplicación que se ha pensado (medición de temperatura en alguna recepción) no sería del todo necesario. Además, de esta forma podremos evitar que el sistema se dirija a una posición errónea cuando se detecte un falso positivo a la vez que se está analizando un rostro verdadero.

Una vez explicado el funcionamiento, veamos el diagrama de bloques del control implementado. En la Figura 6-11 aparece el diagrama general del control, en el que se puede ver un primer bloque, *"Frame k-1"*, que se corresponde con el frame anterior al actual. El segundo bloque, *"Face Detection"*, hace referencia al procesamiento del frame actual en busca de rostros. En la primera iteración, "*centerX[K-1]* y *centerY[K-1]"* es el punto central de la imagen, mientras que *"centerX[K]"* y *"centerY[K]"* se corresponden con el centro de la ROI detectada. La diferencia entre estas salidas será el error de posición cometido, que lo recibe el tercer bloque como entrada, *"PI controller"*, y se obtiene como resultado una señal de posición para el servomotor (pulsos en milisegundos), *"salidaX y salidaY"*. El cuarto bloque es el controlador interno del servomotor, el cuál recibe las órdenes de posición del servo y aplica al motor DC de su interior el voltaje necesario para llevarlo a dicha posición. Al enviar las órdenes de movimiento al motor, debemos tener en cuenta el tiempo que le puede llevar al servo ir de una posición a otra, por lo que se añade el bloque *"Delay"* en el que se realiza dicha espera para esperar el posicionamiento de la plataforma. Una vez estabilizada la plataforma en una nueva posición, se analiza el siguiente frame y se repite el ciclo.

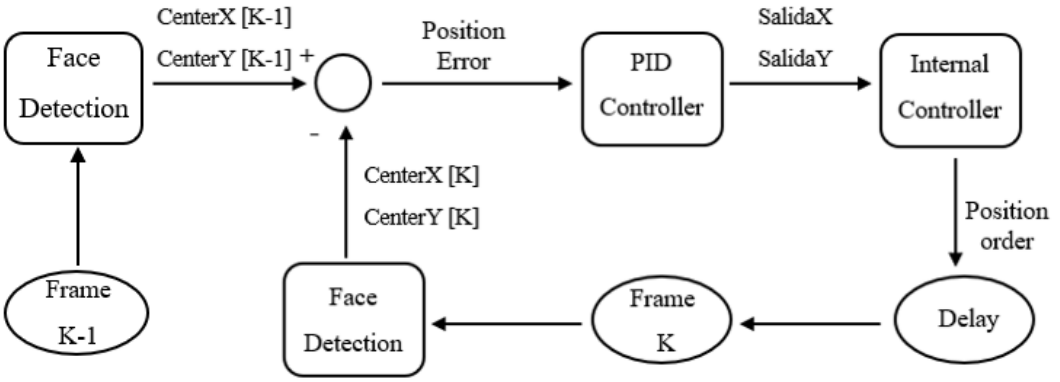


Figura 6-11: Diagrama de bloques general.

En la Figura 6-12 podemos ver el diagrama interno del bloque *"PI controller"* de la Figura 4-11. Este es el diagrama correspondiente al controlador PI conocido por todos. Para ajustar los parámetros K_p y K_I se han seguido los siguientes pasos:

- Para el servomotor de Yaw, y teniendo bloqueado el de Pitch, aumentar lenta progresivamente desde cero el valor de K_p hasta que notemos que la plataforma sigue el rostro sin pasarse, es decir, sin que se produzca sobreoscilación.
- Una vez encontrado el valor correcto de K_p, hacer lo mismo con la constante K_I. Esto ajustará un poco mejor la posición del servo, pero debemos tener en cuenta que se produzcan las menores oscilaciones posibles.

Hacer lo mismo para el servomotor de Pitch, dejando bloqueado ahora el de Yaw para que sea más sencillo.

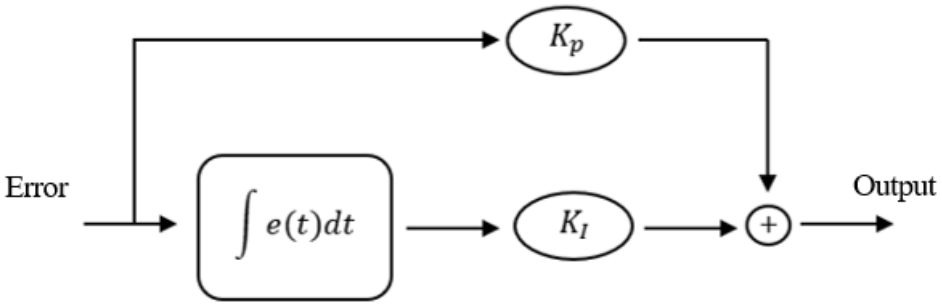


Figura 6-12: Diagrama de bloques del control PI.

Capítulo 7

Conclusiones y líneas de trabajo futuras

Durante la realización de este trabajo se han llevado a cabo diferentes tareas para cumplir el objetivo de construir un sistema de seguimiento de personas y medición de su temperatura corporal en tiempo real. Se han realizado tareas de programación, tareas de diseño hardware para la implementación de dos sensores de imagen en una Raspberry Pi, se han realizado tareas de diseño mecánico y fabricación aditiva (haciendo uso de las impresoras 3D del IMSE) de las distintas piezas que forman la plataforma y, finalmente, se han integrado los servomotores que otorgan de movilidad al sistema.

Las tareas de programación de alto nivel en lenguaje C++ han consistido en llevar a cabo las tareas de reconocimiento de rostros y control de la plataforma móvil. En el caso de la detección de rostros, se ha usado el código propuesto en la web de OpenCV como modelo a seguir. Respecto al control de la plataforma, ha sido necesario añadir las instrucciones relacionadas con la generación de señales PWM para el uso de los servomotores; la implementación de una clase PID, y la integración del método de cálculo de puntos singulares (coordenadas del Bounding-Box) en la imagen IR a partir de puntos en la imagen de espectro visible (visto en el apartado 4.2). También en alto nivel, en lenguaje Python, utilizando como guía el código de Gabriela Solano se ha podido crear una base de datos con la que entrenar nuestra propia red neuronal (apartado 4.1) en el infrarrojo, aunque finalmente no ha sido el método elegido para la detección de rostros.

Se ha estudiado la forma más eficiente de crear una plataforma para esta aplicación, desarrollando distintos diseños 3D en Catia V5. Finalmente se ha optado por un diseño simple con piezas pequeñas en las que se han evitado los ángulos rectos para evitar crear puntos débiles y que además no implicasen un aumento de peso considerable al sistema, lo que provocaría un funcionamiento inadecuado de los servomotores (ver apartado 6.1).

Como resultado tenemos una plataforma capaz de realizar un seguimiento adecuado de objetos (rostros en este caso) a velocidad moderada. La distancia a la que es capaz de detectar estos rostros dependerá mucho de la exhaustividad en el entrenamiento de la red neuronal empleada. En nuestro caso, la red de OpenCV muestra un rendimiento aceptable a distancias de entre 1,5 a 2 metros desde el objetivo a detectar. Es necesario ajustar bien los parámetros de la función *"detectMultiScale"*, con la que realizamos la detección de rostros en la imagen, para que no se produzcan falsas detecciones.

Durante el desarrollo de este trabajo han surgido diferentes ideas que podrían servir como futuras líneas de trabajo:

- Sustitución del método de detección de rostros. En este trabajo se ha utilizado el método de mapeo entre los píxeles de las imágenes infrarroja y visual (descrito en el apartado 4.2) usando una red neuronal para el especto visible. Se propone utilizar el método descrito en el apartado 4.1, en el que se debe crear una base de datos de imágenes en el espectro infrarrojo para luego entrenar una red neuronal. Así conseguiríamos una red neuronal específica para su uso en el espectro infrarrojo. Para ello necesitaríamos repetir el proceso del apartado 4.1 pero creando una base de datos mucho más extensa y con más personas que participasen en la toma de imágenes, para tener así un mayor número de casos (rostros de distintas características).

Como resultado tendríamos un sistema más eficiente, ya que nos ahorraríamos la integración de un sensor de imagen en el espectro visible, lo que tendría como consecuencia una disminución del coste energético y computacional.

- Para conocer la distancia a la ROI detectada se podría usar un sensor de ultrasonidos, como el HC-SR04. Este tipo de dispositivos disponen de un terminal que emite un rayo de ultrasonidos y otro terminal que lo recibe. Conociendo la velocidad a la que se propaga el rayo y el tiempo transcurrido entre el instante en el que se envió y en el que se recibe, podemos calcular la distancia que nos separa del objeto al que apunta. Es un sensor de bajo coste y alta precisión. Mirando su datasheet, vemos que este sensor tiene un alcance de 4 metros. Para usarlo, debemos asegurarnos de que el terminal que emite el rayo apunta al centro de la imagen, que será donde tenemos colocada la ROI. Conociendo la distancia a la ROI podemos tener en cuenta el error cometido en la medida de la temperatura (Figura 5-8) y corregirlo mediante software.
- Implementación hardware de la tarea de detección de rostros. La implementación hardware tiene ventajas sobre la implementación software. La más importante es la capacidad de poder dividir el algoritmo en distintas partes para poder ejecutarlas al mismo tiempo (paralelismo). A mayor complejidad del algoritmo a implementar, más tiempo de ejecución ahorraremos utilizando la implementación hardware, que al fin y al cabo es lo más importante en este tipo de sistemas, ya que tendría un funcionamiento más fluido. El inconveniente es que la implementación hardware resulta más caro que una implementación software en un procesador de propósito general, como es el de la Raspberry Pi.
- Sustitución de servomotores por motores DC brushless (motores de corriente continua sin escobillas). Podrían usarse dos motores Fityle 2208 90KV, con los que conseguiremos movimientos más suaves que mejorarán la calidad de la estabilización, gracias a que permiten el control de su velocidad y aceleración. Así podremos centrar la ROI de forma más exacta a como lo hacemos con los servomotores. Esto es necesario, como se ha comentado anteriormente, para poder tener una buena medición de la distancia a la ROI, asegurándonos que esta se encuentra lo más cerca del pixel central de la imagen, para que el rayo emitido por el sensor de ultrasonidos rebote en el rostro de la persona. El inconveniente es que los motores DC brushless necesitan de un controlador específico, como el *"BaseCam SimpleBGC de 32 bits"*, utilizados para controlar la posición de estos motores

Capítulo 8

Bibliografía

[1] «who.int,» 7 Octubre 2020. [En línea]. Available: https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public. [Último acceso: 25 Agosto 2021].

[2] «espanol.cdc.gov,» 13 Agosto 2021. [En línea]. Available: https://espanol.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/prevention.html. [Último acceso: 20 Agosto 2021].

[3] L. B. A. Caballero y E. E. M. Herrera, «La fiebre. Conceptos básicos,» *Revista Cubana de Pediatría,* vol. 70, nº 2, 1998.

[4] J. D. Matich, «Redes Neuronales: Conceptos básicos y aplicaciones,» *Universidad Tecnológica Nacional, México,* vol. 41, 2001.

[5] P. P. G. García, Reconocimiento de imágenes utilizando redes neuronales artificiales, 2013.

[6] L. Rouhiainen, Inteligencia artificial, Madrid: Alienta Editorial, 2018.

[7] M. A. Boden, Inteligencia artificial, Turner, 2017.

[8] H. T. Quispe, «Inteligencia Artificial: Historia de un muñeco,» *Revista de Información, Tecnología y Sociedad,* p. 186, 2008.

[9] R. Benítez, G. Escudero, S. Kanaan y D. M. Rodó, Inteligencia artificial avanzada, Editorial UOC, 2014.

[10] F. O. P. Ramírez y H. F. Castaño, «Las redes neuronales y la evaluación del riesgo de crédito,» *Revista Ingenierías Universidad de Medellín,* vol. 6, nº 10, pp. 77-91, 2007.

[11] E. R. C. Barriga, Aplicación práctica de la visión artificial para el reconocimiento de rostros en una imagen, utilizando redes neuronales y algoritmos de reconocimiento de objetos de la biblioteca opencv, 2017.

[12] M. R. S. Maila, Evaluación De Algoritmos Aplicados A La Extracción De Caracteristicas Para El Reconocimiento Facial En Base A La Iso/Iec 25010, Ecuador: PUCESE-Escuela de

Sistemas y Computación, 2021.

[13] C. M. T. Gómez, Análisis comparativo de los algoritmos Fisherfaces y LBPH para el reconocimiento facial en diferentes condiciones de iluminación y pose, Tacna – 2015, Universidad Nacional Jorge Basadre Grohmann, 2016.

[14] S. D. Pavón, Reconocimiento facial mediante el Análisis de Componentes Principales (PCA), Sevilla: Universidad de Sevilla, 2017.

[15] P. Viola y M. Jones, «Detección rápida de objetos mediante una cascada mejorada de funciones simples,» de *Actas de la Conferencia de la Sociedad de Computación IEEE de 2001 sobre Visión por Computadora y Reconocimiento de Patrones. CVPR 2001*, Kauai, HI, EE. UU, IEEE, 2001, p. II.

[16] J. A. C. Drews, Registro multimodal de imágenes VIS, SWIR y LWIR en hardware dedicado, 2018.

[17] R. d. r. latinoamericanos, «repositorioslatinoamericanos.uchile.cl,» [En línea]. Available: http://repositorioslatinoamericanos.uchile.cl/handle/2250/3292332. [Último acceso: 15 Agosto 2021].

[18] «vcipl-okstate.org,» [En línea]. Available: http://vcipl-okstate.org/pbvs/bench/index.html. [Último acceso: 20 Agosto 2021].

[19] G. Friedrich y Y. Yeshurun, Seeing people in the dark: Face recognition in infrared images. En International Workshop on Biologically Motivated Computer Vision, Berlin, Heidelberg: Springer, 2002.

[20] C.-L. Chen y B.-L. Jian, «Infrared thermal facial image sequence registration analysis and verification,» *Infrared Physics & Technology,* vol. 69, pp. 1-6, 2015.

[21] K. Reese, Y. Zheng y A. Elmaghraby, A comparison of face detection algorithms in visible and thermal spectrums, Int'l Conf. on Advances in Computer Science and Application, 2012.

[22] S. J. Krotosky, S. Y. Cheng y M. M. Trivedi, Face detection and head tracking using stereo and thermal infrared cameras for "smart" airbags: a comparative analysis, Washington, WA, USA: Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749), 2004.

[23] M. Vergara, A. Wolf y M. Figueroa, «A texture-based architecture for face detection in IR images on an FPGA.,» *Electro-Optical and Infrared Systems: Technology and Applications XI. Electro-Optical and Infrared Systems: Technology and Applications XI. International Society for Optics and Photonics,* vol. 9249, 2014.

[24] J. E. S. Salcedo, Detección y reconocimiento de rostros en imágenes infrarrojas sobre hardware digital dedicado, 2016.

[25] A. A. Rafiq, W. N. Rohman y S. D. Riyanto, «Development of a Simple and Low-cost Smartphone Gimbal with MPU-6050 Sensor,» *Journal of Robotics and Control (JRC),* vol. 1, nº 4, pp. 136-140, 2020.

[26] X. Lu, B. Benes, J. Zhang, H. Li y H. Kang, «FlyCam: Multitouch Gesture Controlled Drone

Gimbal Photography,» *IEEE Robotics and Automation Letters,* vol. 3, nº 4, pp. 3717-3724, 2018.

[27] H.-C. Park, S.-W. Lee y H. Jeong, «Image-Based Gimbal Control in a Drone for Centering Photovoltaic Modules in a Thermal Image,» *Applied Sciences,* vol. 10, nº 13, p. 4646, 2020.

[28] S. R. e. al, «JMM Gimbal Stabilizer,» *INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND TECHNOLOGY,* vol. 3, nº 1, pp. 31-40, 2019.

[29] Raspberry (consumos), «raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/documentation/faqs/. [Último acceso: 29 Junio 2021].

[30] Raspberry Pi, «raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/about/. [Último acceso: 28 Abril 2021].

[31] E. L. Aldea, Raspberry Pi Fundamentos y aplicaciones, Ra-Ma, 2018.

[32] Raspberry Pi (3B), «raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Último acceso: 28 Abril 2021].

[33] Raspberry Pi (4B), «raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. [Último acceso: 28 Abril 2021].

[34] JEGX, «geeks3d.com,» 30 Septiembre 2019. [En línea]. Available: https://www.geeks3d.com/20190930/raspberry-pi-4-vs-raspberry-pi-3-cpu-and-gpu-benchmarks/. [Último acceso: 28 Abril 2021].

[35] A. Kopytov, «yumpu.com,» [En línea]. Available: https://www.yumpu.com/en/document/read/17130663/sysbench-manualpdf. [Último acceso: 3 Agosto 2021].

[36] JEGX, «geeks3d.com,» 25 Septiembre 2019. [En línea]. Available: https://www.geeks3d.com/hacklab/20190925/geexlab-0-29-2-released-for-all-platforms-alpha-to-coverage-added-fmod-and-wiringpi-support-updated/. [Último acceso: 30 Julio 2021].

[37] B. Balon y M. Simic, Using Raspberry Pi Computers in Education, Opatija, Croatia: IEEE, 2019.

[38] F. M. Fernández, «franciscomoya.gitbooks.io,» 17 Enero 2017. [En línea]. Available: https://franciscomoya.gitbooks.io/taller-de-raspberry-pi/content/es/elems/gpio.html. [Último acceso: 24 Mayo 2021].

[39] M. Barr, «Pulse With Modulation,» *Beginner's Corner,* pp. 103-104, 2001.

[40] I. a. c. y. m. PWM, «solectroshop.com,» 26 Agosto 2020. [En línea]. Available: https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38. [Último acceso: 30 Abril 2021].

[41] M. Vollmer, K.-P. Möllmann y R. R. Pastor, Termografía infrarroja, Universitat politècnica de valència, 2013.

[42] Testo AG, Termografía, guía de bolsillo, Testo AG, 2008.

[43] D. J. Pérez, Introducción a los sensores remotos - Aplicaciones en Geología, Buenos Aires, 207.

[44] J. Hashagens, «photonics.com,» Septiembre 2014. [En línea]. Available: https://www.photonics.com/Articles/SWIR_Applications_and_Challenges_A_Primer/a56646. [Último acceso: 1 Mayo 2021].

[45] A. F. Ros Gómez, Desarrollo e Integración de Sistema de medida de temperatura foliar en plataforma Linux sobre Raspberry Pi, Cartagena: Universidad Politécnica de Cartagena, 2019.

[46] GroupGets (FLIR Lepton 3.5), «groupgets.com,» [En línea]. Available: https://groupgets.com/manufacturers/flir/products/lepton-3-5. [Último acceso: 1 Mayo 2021].

[47] A. Calle y P. Salvador, «Revisando el concepto de resolución en teledetección,» *Revista de teledetección,* nº 37, p. 78, 2012.

[48] J. D. G. Parrilla y A. H. López, más allá del selfie: un nuevo paradigma en prroducciones audiovisuales, Madrid: McGraw-Hill Interamericana de España, 2020.

[49] Alex, «raspi.tv,» 25 Abril 2016. [En línea]. Available: https://raspi.tv/2016/new-8-megapixel-raspberry-pi-camera-2-1-launches. [Último acceso: 3 Mayo 2021].

[50] Raspberry Pi Cam, «raspberrypi.org,» [En línea]. Available: https://www.raspberrypi.org/products/pi-noir-camera-v2/. [Último acceso: 24 Mayo 2021].

[51] Gus, «youtube.com,» Pi My Life Up, 11 Marzo 2015. [En línea]. Available: https://www.youtube.com/watch?v=lP-QQGxm2Y0&ab_channel=PiMyLifeUp. [Último acceso: 15 Abril 20021].

[52] OpenCV (Cascade Classifier), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. [Último acceso: 27 Abril 2021].

[53] L. Yuan-Hsiang, L. Jia-Wei y L. Ming-Hung, A Thermal Camera Based Continuous Body Temperature Measurement System, Seoul, Korea: IEEE, 2019.

[54] A. Alekhin, «github.com,» 12 Abril 2020. [En línea]. Available: https://github.com/opencv/opencv/tree/master/data/haarcascades. [Último acceso: 27 Abril 2021].

[55] D. A. C. Véliz, Implementación de un sistema de identificación de patentes en RPI, Valparaíso, 2018.

[56] G. Solano, «omes-va.com,» 29 Julio 2020. [En línea]. Available: https://omes-va.com/como-crear-tu-propio-detector-de-objetos-con-haar-cascade-python-y-opencv/. [Último acceso: 17 abril 2021].

[57] A. Ahmadi, «amin-ahmadi.com,» 5 Julio 2017. [En línea]. Available: https://amin-ahmadi.com/cascade-trainer-gui/. [Último acceso: 20 Abril 2021].

[58] OpenCV (Library), «opencv.org,» 16 Octubre 2020. [En línea]. Available: https://opencv.org/opencv-4-5-0/. [Último acceso: 27 Abril 2021].

[59] OpenCV (facedetect.pp), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/3.4/d4/d26/samples_2cpp_2facedetect_8cpp-example.html. [Último acceso: 27 Abril 2021].

[60] OpenCV (detectMultiScale), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html#accf96d130d9f3cf4c58bf445b7861c19. [Último acceso: 27 Abril 2021].

[61] OpenCV (cv::Rect_< _Tp > Class Template Reference), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/4.5.0/d2/d44/classcv_1_1Rect__.html#a5a41149f4b012b9f323b5913454375a1. [Último acceso: 11 Mayo 2021].

[62] W. R. Guerrero Albán, «Detección y descripción de puntos característicos en imágenes multiespectrales,» *Revista Tecnológica ESPOL – RTE,* vol. 27, nº 1, pp. 70-79, 2014.

[63] OpenCV (Trackbar), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/3.4/d7/dfc/group__highgui.html#gaf78d2155d30b728fc413803745b67a9b. [Último acceso: 27 Abril 2021].

[64] OpenCV (Resize), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/master/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d. [Último acceso: 27 Abril 2021].

[65] S. B. Melo, TRANSFORMACIONES GEOMÉTRICAS SOBRE IMÁGENES DIGITALES, Universidad Distrital Francisco José de Caldas.

[66] MedlinePlus (Medición de la temperatura), «medlineplus.gov,» 4 Mayo 2021. [En línea]. Available: https://medlineplus.gov/spanish/ency/article/003400.htm. [Último acceso: 8 Mayo 2021].

[67] N. Castejón, «webconsultas.com,» 20 Abril 2020. [En línea]. Available: https://www.webconsultas.com/belleza-y-bienestar/habitos-saludables/tipos-de-termometros-corporales-pros-y-contras. [Último acceso: 8 Mayo 2021].

[68] L. F. Aragón-Vargas, «Limitaciones de la lectura de la temperatura temporal (en la frente) como método de tamizaje para el Covid-19,» *Pensar en Movimiento,* vol. 18, nº 1, 2020.

[69] B. M. L.-P. Vigil, R. J. M. Alonso y M. E. I. Martínez, «Implementación del Algoritmo de Otsu sobre FPGA,» *Revista Cubana de Ciencias Informáticas,* vol. 10, nº 3, 2016.

[70] N. Silawan, K. Kusukame, K. J. Kek y W. S. Kuan, «A Novel Environment-Invariant Core Body Temperature Estimation for High Sensitivity and Specificity Fever Screening,» de *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, 2018.

[71] S. F. Larsen y M. Hongn, «Termografía infrarroja en la edificación: aplicaciones cualitativas,» *Avances En Energias Renovables y Medio Ambiente,* vol. 16, 2012.

[72] Universidad de Sevilla, «Tema 2: Procesamiento en el dominio espacial (parte 2),» pp. 1-23.

[73] P. Turmero, «monografias.com,» 1 Abril 2016. [En línea]. Available: https://www.monografias.com/trabajos108/filtros-y-convoluciones/filtros-y-convoluciones.shtml. [Último acceso: 20 Mayo 2021].

[74] U. d. Córdoba, «uco.es,» [En línea]. Available: http://www.uco.es/users/ma1fegan/2015-2016/vision/Temas/ruido.pdf. [Último acceso: 28 Julio 2021].

[75] OpenCV (GaussianBlur), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html. [Último acceso: 14 Junio 2021].

[76] S. M. Valencia, Medidor de área foliar utilizando una cámara digital, Pereira, Colombia, 2019.

[77] OpenCV (threshold), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/4.5.0/d7/d4d/tutorial_py_thresholding.html. [Último acceso: 14 Junio 2021].

[78] J. A. Leñero-Bardallo, R. D. l. Rosa-Vidal, R. Padial-Allué, J. Ceballos-Cáceres, Á. Rodríguez-Vázquez y J. Bernabéu-Wittel, «A Customizable Thermographic Imaging System for Medical Image Acquisition and Processing,» *EEE Sensors Journal,* 2021.

[79] C.-S. a. L. C.-S. Fahn, «A high-definition human face tracking system using the fusion of omni-directional and PTZ cameras mounted on a mobile robot,» de *2010 5th IEEE Conference on Industrial Electronics and Applications*, Taichung,Taiwan, IEEE, 2010, pp. 6-11.

[80] M. T. Kristjan Tiimus, Camera gimbal control system for unmanned platforms, Pärnu, Estonia, 2010.

[81] R. J. Rajesh y P. Kavitha, Camera gimbal stabilization using conventional PID controller and evolutionary algorithms, Indore, India: IEEE, 2015.

[82] M. d. V. López, «es.slideshare.net,» 22 Enero 2015. [En línea]. Available: https://es.slideshare.net/tecnoparador/estabilidad-y-centro-de-gravedad. [Último acceso: 2021 Abril 27].

[83] P. INC, «docs.rs-online.com,» 24 Octubre 2011. [En línea]. Available: https://docs.rs-online.com/0e85/0900766b8123f8d7.pdf. [Último acceso: 23 Junio 2021].

[84] C. Rosero y J. Enrique, Estudio de un sistema de posicionamiento para interiores, Universitat Politècnica de València, 2018.

[85] Dejan, «howtomecatronics.com,» 9 abril 2019. [En línea]. Available: https://howtomechatronics.com/projects/diy-arduino-gimbal-self-stabilizing-platform/. [Último acceso: 23 abril 2021].

[86] C. Y. O. C. H. C. DETECCIÓN DE ROSTROS, «unipython.com,» 5 Abril 2018. [En línea].

Available: https://unipython.com/deteccion-rostros-caras-ojos-haar-cascad/. [Último acceso: 27 4 2021].

[87] M. Ángel, «mbrobotics.es,» 2 Mayo 2019. [En línea]. Available: http://mbrobotics.es/blog/raspberry-pi-gpios/. [Último acceso: 2021 Abril 29].

[88] N. Castejón, «webconsultas.com,» 17 Abril 2020. [En línea]. Available: https://www.webconsultas.com/belleza-y-bienestar/habitos-saludables/donde-colocar-el-termometro-para-medir-la-temperatura. [Último acceso: 8 Mayo 2021].

[89] Gesa, «termometros.com,» [En línea]. Available: https://www.termometros.com/Como-funciona-un-termometro. [Último acceso: 8 Mayo 2021].

[91] OpenCV (bitwise_and), «docs.opencv.org,» [En línea]. Available: https://docs.opencv.org/master/d2/de8/group__core__array.html#ga60b4d04b251ba5eb1392c34425497e14. [Último acceso: 14 Junio 2021].

[92] Benewake, «sparkfun.com,» [En línea]. Available: https://cdn.sparkfun.com/assets/1/4/2/1/9/TFmini_Plus_A02_Product_Manual_EN.pdf. [Último acceso: 28 Julio 2021].

[93] «andorra.desertcart.com,» [En línea]. Available: https://andorra.desertcart.com/products/100007042-fityle-2208-90-kv-3-mm-shaft-gimbal-brushless-motor-for-100-200-g-gopro-3-rc-quadcopter. [Último acceso: 29 Julio 2021].

[94] «tutorialspoint.com,» [En línea]. Available: https://www.tutorialspoint.com/arduino/arduino_ultrasonic_sensor.htm. [Último acceso: 25 Agosto 2021].

[95] «datasheet.es,» [En línea]. Available: http://www.datasheet.es/PDF/779948/HC-SR04-pdf.html. [Último acceso: 25 Agosto 2021].

[96] «basecamelectronics.com,» [En línea]. Available: https://www.basecamelectronics.com/simplebgc32bit/. [Último acceso: 29 Agosto 2021].

Capítulo 9

Anexos

Anexo A
IRFacesDetector.py

El fichero *"IRFacesDetector.py"*, que se encuentra en la carpeta que creamos exclusivamente para el entrenamiento de la red neuronal ("*IRFacesCascade"*) se usará para la creación de una base de datos formada por imágenes en el espectro de infrarrojos y su funcionamiento es el siguiente:

Primero, se crean ambas carpetas "n" y "p" e iniciamos la grabación de imágenes con el sensor infrarrojos (tiene índice 1 en nuestro caso).

Iniciada la grabación, se copia el frame y se dibuja aproximadamente en el centro de la imagen un rectángulo que delimitará una región de interés.

Finalmente, cuando queramos guardar como imagen positiva o negativa la imagen formada por la ROI, presionamos la letra *'p'* (si es imagen positiva) o *'n'* (si es imagen negativa).

```
1   import cv2
2   import numpy as np
3   import imutils
4   import os
5
6   Positivas = 'p'                 # Llamaremos p a la carpeta de imágenes positivas
7   Negativas = 'n'                 # Llamaremos n a la carpeta de imágenes negativas
8
9   if not os.path.exists(Positivas):       # Si no existe, se crea carpeta p
10      print('Carpeta creada: ',Positivas)
11      os.makedirs(Positivas)
12
13  if not os.path.exists(Negativas):       # Si no existe, se crea carpeta n
14      print('Carpeta creada: ',Negativas)
15      os.makedirs(Negativas)
16
17  cap = cv2.VideoCapture(1)               # Capturamos imagen con cámara IR, índice 1
18
19  #Dimensiones de rectángulo que contendrá la ROI
20  x1, y1 = 45, 20
21  x2, y2 = 110, 100
22
23  #contador de imágenes guardadas positivas y negativas
24  countP = 0
25  countN = 0
```

```
26
27 while True:
28     ret, frame = cap.read()
29     if ret == False: break
30     imAux = frame.copy()                                    # Copiamos el frame
31     cv2.rectangle(frame,(x1,y1),(x2,y2),(255,0,0),2)     # Dibujamos el rectángulo
32
33     objeto = imAux[y1:y2, x1:x2]                  # Imagen delimitada por rectángulo
34     objeto = imutils.resize(objeto, width=38)     # Redimensionamos "objeto" para
35 hacerlo más pequeño
36     k = cv2.waitKey(1)
37     if k == 27:
38         break
39
40     cv2.imshow('Frame',frame)                     # Muestra imagen
41     cv2.imshow('Objeto',objeto)                   # Muestra ROI de la imagen
42
43     # Si presionamos "p" guardaremos la imagen formada por objeto en la carpeta "p"
44     if k == ord('p'):
45         cv2.imwrite(Positivas+'/objeto_{}.jpg'.format(countP),objeto)
46         print('Imagen almacenada:','objeto_{}.jpg'.format(countP))
47         countP = countP + 1
48
49     # Si presionamos "n" guardaremos la imagen formada por objeto en la carpeta "n"
50     if k == ord('n'):
51         cv2.imwrite(Negativas+'/objeto_{}.jpg'.format(countN),objeto)
52         print('Imagen almacenada:','objeto_{}.jpg'.format(countN))
53         countN = countN + 1
54
55 cap.release()
56 cv2.destroyAllWindows()
```

Anexo B
Temperature.cpp

El fichero *"Temperature.cpp"* implementa lo descrito en el apartado ***"¡Error! No se encuentra el origen de la referencia.. ¡Error! No se encuentra el origen de la referencia."***. Este fichero contiene las funciones necesarias para el procesado de la imagen y estimación de la temperatura corporal de la persona que aparece en imagen.

```
1   /**
2    * @file      Temperature.cpp
3    * @author    Daniel Clavijo Calvo
4    * @brief     Segmentación por método de Otsu y estimación de temperatura
5    * @version   0.1
6    * @date      2021-04-14
7    *
8    * @copyright Copyright (c) 2021
9    *
10   */
11
12  #include <opencv2/imgproc.hpp>     // Librería para el procesado de imagen
13
14  using namespace cv;
15
16  double IRtoTempComversion(const double& IR);
17  double ApplySegmentation(const Mat& Image, Mat& result);
18
19  /**
20   * @brief   Función principal para la estimación de temperatura
21   *
22   * @param   Image: frame en el que se detecta rostro
23   * @param   ROI: región de interés del frame
24   * @param   applySegmentation: booleano para aplicar, o no, el método de Otsu
25   * @param   otsuThreshold: valor de segmentación mediante Otsu
26   * @return  double Temperature: valor de temperatura de la región de interés
27   */
28
29  double GetTempFromIRImage(const Mat& Image, const Rect& ROI, bool
30  applySegmentation, double* otsuThreshold)
31  {
32      double IR = 0, temperature = 0, threshold;
33      Mat ImgROI = Image(ROI);
34
35
```

```
36
37
38      if(applySegmentation)
39      {
40          Mat result, ImgROIBlur, ImgSegmented;   // Matriz que usaremos de máscara
41          double numPixel = 0, pixelValue = 0;    // Nº total, valor de píxeles
42
43          GaussianBlur(ImgROI,ImgROIBlur,Size(3,3),0);          // Filtro de Gauss
44          threshold = ApplySegmentation(ImgROIBlur, ImgSegmented);  // Aplica Otsu.
45
46
47          //imshow("ROI",ImgROI);                 // ROI sin aplicar Otsu
48          //imshow("filtrado",ImgROIBlur);        // Imagen suavizada con Gauss
49          //imshow("mascara",ImgSegmented);       // Imagen segmentada
50
51          IR = mean(final)[0];     //valor medio de los píxeles de la imagen
52
53      }
54      else
55      {
56          IR = mean(ImgROI)[0];                 // Media de la imagen sin segmentar.
57      }
58
59      temperature = IRtoTempComversion(IR);   // Pasa valor IR medio a Tª media.
60
61      if(otsuThreshold != 0)
62      {
63          *otsuThreshold = threshold;         // Devuelve el valor umbral utilizado
64  por referencia.
65      }
66      return temperature;                     // Devuelve valor de Tª media.
67  }
68
69  /**
70   * @brief   Cálculo de temperatura de la región de interés
71   *
72   * @param   IR: valor medio de todos los píxeles de la región de interés
73   * @return  double: valor de temperatura
74   */
75  double IRtoTempComversion(const double& IR)
76  {
77      double T1 = 36.0, T2 = 36.04;           // Parámetros de calibración de la Tª
78      double R1 = 8250, R2 = 8400;
79      double IRSensorTo8Bit = 16383.0/255;    // Conversión de 14 a 8 bits
80      double a, b;
81
82      a = (T2-T1)/(R2-R1);
83      b = T2 - a*R2;
84
85      return IR*IRSensorTo8Bit*a + b;         // Cálculo de Tª
86  }
87
88  /**
89   * @brief  Función que aplica Otsu a la Imagen
90   *
91   * @param  Image: Imagen a la que se le applica Otsu formada por ROI
92   * @param  result:Resultado de la segmentación
```

```
93   * @return double: valor de segmentación de Otsu
94   */
95  double ApplySegmentation(const Mat& Image, Mat& result)
96  {
97      return threshold(Image,result,0, 0, THRESH_TOZERO | THRESH_OTSU);
98  }
```

Anexo C
Clase PID

a) PID.h

Definición de la clase PID. Aquí se definen los métodos, atributos y constructor de la clase.

```
1
2  // PID class
3  #ifndef PID_h
4  #define PID_h
5
6  class PID{
7      private:          // Atributos
8          double reference;
9          double LastOutput;
10         double LastError;
11         double AccumErr;
12         double kp, ki, kd;
13         double MinSat, MaxSat;
14         double alpha;
15         double EMA_LP;
16         double output;
17         double offset;
18
19     public:           // Métodos y constructor
20           PID(double kp, double ki, double kd, double alpha, double MinSat, double
21   MaxSat, double offset);
22         int Reference(double reference, double LastOutput);
23         double Update (double input, double incT);
24 };
   #endif
```

b) PID.cpp

```
1    /**
2     * @file    PID.cpp
3     * @author  Daniel Clavijo Calvo
4     * @brief   Clase PID
5     * @version 0.1
6     * @date    2021-06-24
7     *
8     * @copyright Copyright (c) 2021
9     *
10    */
11
12   #include "PID.h"
13   #include <iostream>
14
15   PID::PID(double kp, double ki, double kd, double alpha, double MinSat, double
16    MaxSat,double offset){
17       this-> kp = kp;                 // Ganancia proporcional
18       this-> ki = ki;                 // Ganancia integral
19       this-> kd = kd;                 // Ganancia derivativa
```

```
20      this-> MinSat = MinSat;          // Límite inferior de la señal de control
21      this-> MaxSat = MaxSat;          // Límite superior de la señal de control
22      this-> offset = offset;          // Rango de error que consideramos admisible
23      this-> alpha = alpha;    // Factor del que dependerá el suavizado de la señal
24  }
25  int PID::Reference(double reference, double LastOutput){
26      this-> reference = reference;   // Valor referencia (centro imagen)
27      this-> LastOutput = LastOutput; // Valor anterior de la señal de control
28      LastError = 0;                  // Último valor de error (en píxeles)
29      EMA_LP = 0;       // Valor filtrado de la señal de control (filtro pasa baja)
30      output = 0;       // Señal de control sin filtrar
31
32      return reference;
33  }
34
35  double PID::Update(double input,double incT){
36
37      double Tm = incT;
38      double Error = reference - input;
39      double new_AccumErr;
40      bool act_AccumErr;     //Indicará si se debe actualizar el error integral o no
41      printf("Error=%f\r\n",Error);
42      new_AccumErr = AccumErr + Error*Tm;     // error integral
43
44      if (Error < -offset || Error > offset)  // Rango de error a superar
45      {
46          output = LastOutput +  kp * Error + ki * new_AccumErr + kd * (Error-
47   LastError)/Tm;
48          // Filtro paso bajo
49          EMA_LP = alpha * output + (1 - alpha) * LastOutput;     // Filtrado paso
50   baja de la señal de control
51          // Anti-Windup
52          if (EMA_LP >= MaxSat) // Si valor de salida es mayor al valor máximo
53          {
54              EMA_LP = MaxSat;
55              if (Error > 0)                  // Si error es positivo
56                  {act_AccumErr = false;}     // No actualizaremos error integral
57          }
58          else if (EMA_LP <= MinSat) // Si valor de salida es menor al valor mínimo
59          {
60              EMA_LP = MinSat;
61              if (Error < 0)                  // Si el error es negativo
62                  {act_AccumErr = false;}     // No actualizaremos error integral
63          }
64
65          if (act_AccumErr)
66              {AccumErr = new_AccumErr;}      // Actualización del error integral
67
68          LastOutput = EMA_LP;   // actualizamos último valor de control
69          LastError = Error;    // actualización del error
70      }
71
72      printf("PID EJECUTADO\n");
73      printf("EMA_LP=%f\n\n",EMA_LP);
74      return EMA_LP;
75  }
```

Anexo D
Rutina principal: peopledetect.cpp

```
/**
 * @file        peopledetect.cpp
 * @author      OpenCV
 * @brief       Programa principal
 * @version     0.1
 * @date        2021-04-14
 *
 * @copyright   Copyright (c) 2021
 *
 */

#include "opencv2/objdetect.hpp"        //Incluye la clase cv::CascadeClassifier
#include "opencv2/highgui.hpp"          //Incluye funciones imshow, createTrackbar
#include "opencv2/imgproc.hpp"          //Incluye funciones cvtColor, resize…
#include "opencv2/videoio.hpp"          //Incluye la clase cv::VideoCapture
#include <iostream>                     //Entradas/salidas (cin...,cout...)
#include <wiringPi.h>                   //Librería que permite controlar la GPIO
#include <softPwm.h>                    //Librería para el uso de señales PWM
#include "PID.h"

// Pin de señal de motores
#define P_PIN12 1                       // motor de eje X
#define P_PIN33 23                      // motor de eje Y
// Dimensiones de imagen IR.
#define IRROWS 120                      // filas
#define IRCOLS 160                      // columnas
// Dimensiones de imagen visible.
#define VROWS 480                       // filas
#define VCOLS 640                       // columnas
// Proporción de imagen visible entre imagen IR.
#define PROP 4
// Margen añadido a la ROI obtenida en IR.
#define MARGIN 5

using namespace std;
using namespace cv;

/**
 * @brief Help function
 *
 * @param argv
```

```
42  */
43 static void help(const char** argv)
44 {
45     cout << "\nThis program demonstrates the use of cv::CascadeClassifier class to
46 detect objects (Face + eyes). You can use Haar or LBP features.\n"
47             "This classifier can recognize many kinds of rigid objects, once the
48 appropriate classifier is trained.\n"
49             "It's most known use is for faces.\n"
50             "Usage:\n"
51         <<  argv[0]
52         <<  "    [--cascade=<cascade_path> this is the primary trained classifier
53 such as frontal face]\n"
54             "    [--nested-cascade[=nested_cascade_path this an optional secondary
55 classifier such as eyes]]\n"
56             "    [--scale=<image scale greater or equal to 1, try 1.3 for
57 example>]\n"
58             "    [--try-flip]\n"
59             "    [filename|camera_index]\n\n"
60             "example:\n"
61         <<  argv[0]
62         <<  " --cascade=\"data/haarcascades/haarcascade_frontalface_alt.xml\" --
63 nested-cascade=\"data/haarcascades/haarcascade_eye_tree_eyeglasses.xml\" --
64 scale=1.3\n\n"
65             "During execution:\n\tHit any key to quit.\n"
66             "\tUsing OpenCV version " << CV_VERSION << "\n" << endl;
67 }
68
69 void detectAndDraw( Mat& img, CascadeClassifier& cascade,CascadeClassifier&
70 nestedCascade,double scale, bool tryflip, int& centerX, int& centerY, int
71 iSliderValue2,int iSliderValue1, Mat IRMat);
72
73 double GetTempFromIRImage(const Mat& Image, const Rect& ROI, bool
74 applySegmentation, double* otsuThreshold);
75
76 string cascadeName;
77 string nestedCascadeName;
78 bool detect;                    // Indicará si se ha producido detección de rostro
79 bool multidetect;               // Indicará si se ha detectado más de un rostro
80
81 int main( int argc, const char** argv )
82 {
83     string inputName;
84     bool tryflip;
85     CascadeClassifier cascade, nestedCascade;
86     PID PIDX(0.0025,0.00001,0,0.4,4,22,50);          // Crea PID de Yaw
87     PID PIDY(0.004,0.00001,0,0.4,13,20,50);          // Crea PID de Pitch
88     double scale;
89
90     int CentralPointX = 246;     // punto central de imagen en eje X escala 1.3
91     int CentralPointY = 123;     // punto central de imagen en eje Y escala 1.3
92     int centerX = 246;           // punto central de ROI (eje X)
93     int centerY = 123;           // punto central de ROI (eje Y)
94     int pos_iniX = 13, pos_iniY = 17;         // posiciones iniciales del motor
95     double salidaPIDX, salidaPIDY;            // valor de señal de control
96     double new_salidaPIDX, new_salidaPIDY;    // actualización de posición de servo
97     double tini, tfin, t_proc;                // medidas de tiempo
98     double Tm=200;                            // tiempo de muestreo del control
```

```
99    int reset=0;              // contador de iteraciones sin detección de rostro
100
101   PIDX.Reference(CentralPointX, pos_iniX); //Referencia de control, pos. inicial
102   PIDY.Reference(CentralPointY, pos_iniY);
103
104   wiringPiSetup();                    // configurar GPIO como pines PWM
105   pinMode(P_PIN12, PWM_OUTPUT);
106   pinMode(P_PIN33, PWM_OUTPUT);
107   digitalWrite(P_PIN12,0);            // se inicia a 0 el valor de cada pin
108   digitalWrite(P_PIN33,0);
109
110   softPwmCreate(P_PIN12,0,200); //Crea señal PWM, inicializado a 0 y frec. 20ms
111   softPwmCreate(P_PIN33,0,200);
112
113   softPwmWrite(P_PIN12,13);     // motor de eje X en posición central (90°)
114   softPwmWrite(P_PIN33,17);
115   delay(500);                   // tiempo de movimiento del motor
116   softPwmWrite(P_PIN12,0);      // valor de duty-cycle 0 para evitar vibraciones
117   softPwmWrite(P_PIN33,0);
118
119   Mat VMat, IRMat, image;       // Matrices de imagen visual e infrarroja
120   int X_OFFSET = 0;             // Offset de puntos entre VMat e IRMat.
121   int Y_OFFSET = 0;
122
123   VideoCapture capture;         // Iniciamos grabación visual
124   VideoCapture IR(1);           // Iniciamos grabación infrarrojos
125
126   // Crea ventana y trackbars para modificar offset en tiempo real.
127   /*namedWindow("Offset", 1);
128
129   int iSliderValue1 = 50 + X_OFFSET;
130   createTrackbar("Offset X", "Offset", &iSliderValue1, 100); // Offset en X.
131
132   int iSliderValue2 = 50 + Y_OFFSET;
133   createTrackbar("Offset Y", "Offset", &iSliderValue2, 100); // Offset en Y.*/
134
135   int iSliderValue1 = 40;   // Valores finales de sliders. Factor de escala 1.3
136   int iSliderValue2 = 70;
137
138   cv::CommandLineParser parser(argc, argv,
139
140       "{help h||}"
141       "{cascade|data/haarcascades/haarcascade_frontalface_alt.xml|}"
142       "{nested-cascade|data/haarcascades/haarcascade_eye_tree_eyeglasses.xml|}"
143       "{scale|1|}{try-flip||}{@filename||}"
144   );
145   /*if (parser.has("help"))
146   {
147       help(argv);
148       return 0;
149   }*/
150   cascadeName = parser.get<string>("cascade");  // Lee "cascade" empleada
151   nestedCascadeName = parser.get<string>("nested-cascade");//Lee "nestedcascade"
152   scale = parser.get<double>("scale");          // Lee factor de escala
153   if (scale < 1)
154       scale = 1;
155   inputName = parser.get<string>("@filename");     // Valor de puerto de cámara
```

```
156     if (!parser.check())      // Comprueba errores que haya producido parser.get
157     {
158         parser.printErrors(); // Lista de errores
159         return 0;
160     }
161     if (!nestedCascade.load(samples::findFileOrKeep(nestedCascadeName)))
162         cerr << "WARNING: Could not load classifier cascade for nested objects" << endl;
163     if (!cascade.load(samples::findFile(cascadeName))) //Si no encuentra "cascade"
164     {
165         cerr << "ERROR: Could not load classifier cascade" << endl;
166         help(argv);
167         return -1;
168     }
169     if( inputName.empty() || (isdigit(inputName[0]) && inputName.size() == 1) )
170     {
171         int camera = inputName.empty() ? 0 : inputName[0] - '0';
172         if(!capture.open(camera))
173         {
174             cout << "Capture from camera #" <<  camera << " didn't work" << endl;
175             return 1;
176         }
177              // Disminución de resolución de cámara
178              capture.set(CAP_PROP_FRAME_WIDTH, 640);
179              capture.set(CAP_PROP_FRAME_HEIGHT,320);
180     }
181
182     if( capture.isOpened() && IR.isOpened())  //Espera imagen de ambos sensores
183     {
184         cout << "Video capturing has been started ..." << endl;
185
186         for(;;)
187         {
188             tini = (double)getTickCount();
189             capture >> VMat;                // Guarda frame visual en matriz VMat
190             IR >> IRMat;                    // Guarda frame IR en matriz IRMat
191             detect = false;                 // Aún no hay detección de rostro
192             multidetect = false;
193
194             if( VMat.empty() )
195                 break;
196
197             detectAndDraw( VMat, cascade, nestedCascade, scale, tryflip, centerX,
198 centerY, iSliderValue2, iSliderValue1, IRMat);
199
200
201             reset++;              // contador de iteraciones sin detección
202             tfin = ((double)getTickCount() - tini);  // Tiempo de procesamiento
203             t_proc = tfin*1000/getTickFrequency();
204
205             delay(Tm-t_proc);    // Espera que finalice el procesado de la imagen
206
207             if (detect)          // Si se detecta rostro orientamos el sistema
208             {
209                 reset = 0;
210                 salidaPIDX = PIDX.Update(centerX,Tm);  // señal de control Yaw
211                 salidaPIDY = PIDY.Update(centerY,Tm);  // señal de control Pitch
212
```

```
213                 // MOTOR EJE X
214                 if (salidaPIDX == new_salidaPIDX || multidetect)// si señal de
215 control no cambia, no mover motor
216                 {
217                     softPwmWrite(P_PIN12,0); // pin a 0 para evitar vibraciones
218                     delay(100);
219                 }
220                 else  // si el valor de posición cambia, enviamos señal al motor
221                 {
222                     new_salidaPIDX = salidaPIDX;
223                     softPwmWrite(P_PIN12,new_salidaPIDX);       // enviamos nuevo
224 valor de posición al motor
225                     delay(100);         // espera posicionamiento del motor
226                     softPwmWrite(P_PIN12,0);    // pin a 0 para evitar vibraciones
227                 }
228 
229                 // MOTOR EJE Y
230                 if (salidaPIDY == new_salidaPIDY || multidetect)
231                 {
232                     softPwmWrite(P_PIN33,0);
233                     delay(100);
234                 }
235                 else
236                 {
237                     new_salidaPIDY = salidaPIDY;
238                     softPwmWrite(P_PIN33,new_salidaPIDY);
239                     delay(100);
240                     softPwmWrite(P_PIN33,0);
241                 }
242             }
243 
244             if (reset >= 30)     // Si no hay detección de cara en 30 iteraciones
245 el sistema vuelve a la posición inicial
246             {
247                 new_salidaPIDX = 13;         // posición inicial del sistema
248                 new_salidaPIDY = 17;
249                 softPwmWrite(P_PIN12,new_salidaPIDX);
250                 softPwmWrite(P_PIN33,new_salidaPIDY);
251                 delay(450);                  // espera posicionamiento del motor
252                 softPwmWrite(P_PIN12,0);     // pin a valor bajo
253                 softPwmWrite(P_PIN33,0);
254                 reset = 0;                   // reiniciamos índice
255                 PIDX.Reference(CentralPointX, pos_iniX);   // reiniciar valores
256                 PIDY.Reference(CentralPointY, pos_iniY);
257             }
258 
259             // Presionar q o ESC para abortar programa
260             char c = (char)waitKey(10);
261             if( c == 27 || c == 'q' || c == 'Q' )
262                 break;
263         }
264     }
265 
266     return 0;
267 }
268 
269 /**
```

```
270  * @brief Función encargada de la detección y cálculo de Bounding-Box
271  *
272  * @param img
273  * @param cascade
274  * @param nestedCascade
275  * @param scale
276  * @param tryflip
277  * @param centerX
278  * @param centerY
279  * @param iSliderValue2
280  * @param iSliderValue1
281  * @param IRMat
282  */
283 void detectAndDraw( Mat& img, CascadeClassifier& cascade,
284                     CascadeClassifier& nestedCascade,
285                     double scale, bool tryflip, int& centerX, int& centerY,
286                     int iSliderValue2,int iSliderValue1, Mat IRMat )
287 {
288     double t = 0;
289     double Temperature=0;
290     char Text[2];
291     Mat gray, smallImg;
292     vector<Rect> faces, faces2;
293     const static Scalar colors[] =
294     {
295         Scalar(255,0,0),
296         Scalar(255,128,0),
297         Scalar(255,255,0),
298         Scalar(0,255,0),
299         Scalar(0,128,255),
300         Scalar(0,255,255),
301         Scalar(0,0,255),
302         Scalar(255,0,255)
303     };
304
305     // Convierte el valor de las trackbars a offset real.
306     int X_OFFSET = iSliderValue1 - 50;
307     int Y_OFFSET = iSliderValue2 - 50;
308
309     cvtColor( img, gray, COLOR_BGR2GRAY );   // Convierte img a escala de grises
310     double fx = 1 / scale;                   // Factor de reducción de imagen
311     resize( gray, smallImg, Size(), fx, fx, INTER_LINEAR_EXACT );   // Reduce gray
312 un factor fx. Interpolacion bilineal
313     equalizeHist( smallImg, smallImg );      // Histograma plano de smallImg
314
315     cascade.detectMultiScale(smallImg, faces,
316         1.1, 5, 0,                 // Factor de escala; número de vecinos; flags
317         Size(30, 30) );            //tamaño mínimo para ser detectado
318
319     for ( size_t i = 0; i < faces.size(); i++ )
320     {
321         Rect r = faces[i];               // Variable que almacena ROIs
322         Mat smallImgROI;                 // ROI cara detectada
323         vector<Rect> nestedObjects;
324         Scalar color = colors[i%8];
325         Point center;
326         int radius;
```

```
327         Mat IRMatBW;   // Matriz para almacenar imagen IR en escala de grises
328         double otsuThreshold = 0;          // Variable que almacena umbral de Otsu.
329 
330         double aspect_ratio = (double)r.width/r.height;
331         if( 0.75 < aspect_ratio && aspect_ratio < 1.3 )
332         {
333             center.x = cvRound((r.x + r.width*0.5)*scale);
334             center.y = cvRound((r.y + r.height*0.5)*scale);
335             rectangle( img, Point(cvRound(r.x*scale), cvRound(r.y*scale)),
336                        Point(cvRound((r.x + r.width-1)*scale), cvRound((r.y +
337 r.height-1)*scale)),Scalar(255, 255, 255), 1, 8, 0);
338              circle (img, center, 1 ,Scalar(255,255,255), 3,8,0); //Centro de ROI
339             detect = true;
340         }
341         else
342             rectangle( img, Point(cvRound(r.x*scale), cvRound(r.y*scale)),
343                        Point(cvRound((r.x + r.width-1)*scale), cvRound((r.y +
344 r.height-1)*scale)),color, 3, 8, 0);
345 
346         if (faces.size() > 1)       // Si detecta más de un rostro en la imagen
347             multidetect = true;
348 
349         // Calcula el rectángulo con offset correspondiente en la imagen IR.
350         int x1 = (r.x*scale)/PROP + X_OFFSET - MARGIN;
351         int y1 = (r.y*scale)/PROP + Y_OFFSET - MARGIN;
352         int x2 = ((r.x + r.width-1)*scale)/PROP + X_OFFSET + MARGIN;
353         int y2 = ((r.y + r.height-1)*scale)/PROP + Y_OFFSET + MARGIN;
354 
355         // Comprueba que los valores son aceptables. (FIX?)
356         if(x1 < 0)
357             x1 = 0;
358         if(y1 < 0)
359             y1 = 0;
360         if(x2 > IRCOLS - 1)
361             x2 = IRCOLS - 1;
362         if(y2 > IRROWS - 1)
363             y2 = IRROWS - 1;
364 
365         // Convierte la imagen IR a blanco y negro para poder binarizar.
366         cvtColor(IRMat, IRMatBW, COLOR_BGR2GRAY);
367 
368         // Coloca un rectángulo en la cara detectada en la imagen IR.
369         Rect ROI = Rect(x1, y1, x2-x1, (y2-y1));             // ROI
370         rectangle(IRMat, ROI, Scalar(255,255,255), 1, 1, 0); //Dibujar BoundingBox
371 
372 
373         // Calcula la temperatura media de la cara.
374         double Temperature = GetTempFromIRImage(IRMatBW,ROI,true,&otsuThreshold);
375          sprintf(Text,"%.1f C" , Temperature); // Convierte double a string
376          putText(IRMat,Text,Point(x1,y1-5), FONT_HERSHEY_PLAIN,1, Scalar(255, 255,
377 255), 0.8, LINE_8);
378 
379         // Comprobar si la temperatura pasa de un valor determinado
380         if (Temperature >= 38){
381             int pausa = 1;
382             imshow("IR",IRMat);     // Muestra ROI con temperatura elevada
383             delay(100);
```

```
384             while(pausa == 1){      // Ejecución pausada. Pulsar 'p' para reanudar
385                char a = (char)waitKey(10);
386                 if( a == 27 || a == 'p' || a == 'P' )
387                     {pausa = 0;
388                     destroyWindow("IR");
389                     }
390             }
391         }
392     }
393
394     //Muestra imágenes
395     imshow("IR",IRMat);
396 }
```