



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2021-2022

Trabajo fin de grado

Sistema de detección de emociones faciales mediante técnicas de Machine Learning adaptado a ROS para un robot de bajo coste basado en Raspberry Pi

Autor: Javier Martínez Madruga

Tutor: Julio Vega Pérez



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-NC-SA International License (Creative Commons AttributionNonCommercial-ShareAlike 4.0). Usted es libre de *(a) compartir*: copiar y redistribuir el material en cualquier medio o formato; y *(b) adaptar*: remezclar, transformar y crear a partir del material. El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciatante.
- *No comercial.* Usted no puede hacer uso del material con propósitos comerciales.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.

Agradecimientos

Me gustaría dedicar este proyecto a todo aquel que, de alguna manera, me ha apoyado o ayudado durante el desarrollo del mismo y me ha acompañado en todo el largo camino de subidas y bajadas que, por suerte o por desgracia, es necesario vivir para lograr llegar hasta este lugar.

Quisiera comenzar dando las gracias a la Universidad Rey Juan Carlos y a todos sus profesores por la gran labor que han realizado durante estos cuatro años de Grado, muchos han mostrado gran pasión y esfuerzo para que cada uno de nosotros, sus alumnos, logremos aprender lo máximo posible. En especial quiero nombrar a Julio Vega, el tutor de este proyecto, por su dedicación y ayuda durante los pasados nueve meses de trabajo. Es de admirar su gran profesionalidad y todo el tiempo dedicado para que todo salga bien.

Por otro lado, no quiero olvidarme de mis compañeros de clase, ya grandes amigos, y les dedico este párrafo de agradecimiento. A Noel, Irene y Vero, acompañándome desde los primeros días, por los buenos momentos tanto fuera como dentro de la universidad; a Carlos, por lo mucho que hemos aprendido y crecido juntos; a Alex por su magnífica compañía y lo bien que lo pasamos durante todas las horas de prácticas en Technaid.

A mis compañeros de entrenamiento en Alcalá y mi entrenador Antonio, que junto con el deporte me han ayudado a evadirme de los problemas y recargar las pilas para seguir adelante con todo. Javi, Alex, Darío, Celia, Rocío, Carlota, Clara... dejándose muchos por nombrar. Y sobre todo a Lucía que, además de en el deporte, siempre ha estado para todo.

Por último, nombrar a la familia y agradecer todo su apoyo incondicional desde que nací. Sin ellos no sería la persona que soy ahora mismo y, seguramente, ni siquiera estaría escribiendo estos Agradecimientos. Este trabajo también es vuestro.

Resumen

La robótica de servicio —amplia rama de investigación dentro de la robótica— se caracteriza por ofrecer servicios útiles para los humanos. Dentro de dicha rama encontramos la Interacción Humano Robot o HRI, campo de estudio que nace con la necesidad de que los robots sean capaces de colaborar y vivir con nosotros, los humanos. Para ello, la Visión Artificial juega un papel fundamental, que sumada al Machine Learning, proporcionan al robot cualidades como el reconocimiento facial y de emociones, dotándole de capacidades que le permiten interactuar de forma adecuada según el contexto.

Las tareas de Visión Artificial que hacen uso de Machine Learning, suelen demandar una alta carga computacional, y no todos los desarrolladores de robots poseen el suficiente dinero o incluso espacio dentro de sus autómatas como para hacerse con una gran estación de cómputo. Es por ello que en este trabajo se ha desarrollado una herramienta de detección de emociones capaz de funcionar en un sistema de bajo coste, o lo que es lo mismo, en un sistema empotrado o embebido caracterizado por su tamaño reducido y precio cometido.

La detección de emociones se ha llevado a cabo mediante el entrenamiento de algoritmos de clasificación (KNN, SVM y MLP) con datos angulares extraídos de las expresiones faciales producidas por emociones humanas (enfado, felicidad, tristeza y sorpresa). Los datos angulares se han obtenido de la construcción de una *malla emocional*, a partir de puntos faciales, basada en el Sistema de Codificación Facial (FACS). El hardware usado ha sido la placa Raspberry Pi 4 Model B y la herramienta final ha sido integrada en ROS Noetic para facilitar su uso en sistemas robóticos.

Se han realizado varios estudios con el afán de buscar la mayor optimización para la herramienta, además de realizar pruebas finales en las que se verifique su funcionamiento y el rendimiento en fotogramas por segundo.

Acrónimos

RUR *Rossum's Universal Robots*

IFR *Federación Internacional de Robótica*

AGV *Vehículo Guiado Automático*

AMR *Robot Móvil Autónomo*

HRI *Interacción humano-robot*

ML *Machine Learning*

SVM *Support Vector Machine*

KNN *K Nearest Neighbour*

ALU *Unidad Aritmética Lógica*

CPU *Unidad Central de Procesamiento*

SBC *Single Board Computer*

TFG *Trabajo de Fin de Grado*

ROS *Robot Operating System*

HOG *Histogram of Oriented Gradients*

MMOD *Max-Margin Object Detection*

CNN *Convolutional Neural Network*

FACS *Facial Action Coding System*

AUs *Action Units*

EMFACS *Emotional Facial Action Coding System*

FPS *Fotogramas por segundo*

PCA *Principal Component Analysis*

Índice general

1. Introducción	1
1.1. Robótica de servicio	1
1.2. Interacción Humano Robot (HRI)	5
1.2.1. Problemática. Paradoja de Moravec	6
1.2.2. Soluciones	7
1.3. Visión Artificial	10
1.4. Machine Learning	11
1.4.1. Aprendizaje supervisado	11
1.4.2. Aprendizaje no supervisado	13
1.4.3. Aprendizaje por refuerzo	13
1.5. Sistemas empotrados	15
1.5.1. Sistemas empotrados populares	15
2. Objetivos	18
2.1. Descripción del problema	18
2.2. Requisitos	19
2.3. Metodología	19
2.4. Plan de trabajo	20
3. Plataforma de desarrollo	21
3.1. Raspberry Pi 4 Model B	21
3.1.1. Raspberry Pi OS	22
3.1.2. Raspberry Pi Camera Module V2.1	23
3.2. Python	25
3.3. MediaPipe	27
3.3.1. Face Mesh	27
3.4. Detector de puntos faciales de dlib.	28
3.5. Scikit-learn. Algoritmos de Machine Learning	29
3.5.1. Máquinas de Vector Soporte (SVM)	29

3.5.2. K Vecinos más Cercanos (KNN)	30
3.5.3. Redes Neuronales Multicapa	31
3.5.4. Análisis de Componentes Principales (PCA)	32
3.6. ROS (Robot Operating System)	33
4. Sistema de detección de emociones	35
4.1. Detección de puntos faciales	36
4.2. Extracción de información de los puntos faciales	37
4.3. Generación del dataset	40
4.4. Entrenamiento del modelo	43
4.5. Integración del sistema en ROS	47
4.5.1. Instalación de ROS en Raspberry	47
4.5.2. Estructura ROS y código desarrollado	49
4.5.3. Uso y rendimiento	51
5. Estudios de optimización	55
5.1. Búsqueda del detector de puntos faciales óptimo	55
5.2. Cantidad de ángulos a utilizar en el dataset	59
5.2.1. Estudio de los ángulos más influyentes en cada emoción	59
5.2.2. Estudio de simetría de las emociones	61
5.3. Búsqueda del dataset óptimo para el entrenamiento	62
6. Conclusiones	69
6.1. Consecución de objetivos	69
6.2. Competencias adquiridas	70
6.3. Valoración final y líneas futuras	71
Bibliografía	72

Índice de figuras

1.1.	Ejemplos de robots de limpieza.	2
1.2.	Robot Spot de Boston Dynamics.	2
1.3.	Ejemplos de robots para educación.	3
1.4.	Ejemplos de coches autónomos.	3
1.5.	Robots Kiva de Amazon.	4
1.6.	Ejemplos de robots ayudantes en cirugía.	4
1.7.	Robot Robin.	5
1.8.	Robot usado en el estudio del artículo [Mumm and Mutlu, 2011].	6
1.9.	Evolución del ser humano. Selección natural.	7
1.10.	Ejemplos de sensores.	8
1.11.	Ejemplo de conversación con GPT-3 en OpenAI.	9
1.12.	Ejemplo de detección de bordes con Canny.	11
1.13.	Ejemplo de regresión lineal. Predicción del precio de la vivienda.	12
1.14.	Demo de la detección de objetos proporcionada por YOLO.	13
1.15.	Ejemplo de clustering. Agrupación de datos de casas.	14
1.16.	Modelo de Aprendizaje por Refuerzo jugando a Pacman.	14
1.17.	Sistemas empotrados más usados.	16
1.18.	Ejemplos de placas Arduino.	16
1.19.	Ejemplos de placas Raspberry.	17
3.1.	Raspberry Pi 4b.	21
3.2.	Captura de pantalla de Raspberry Pi OS.	23
3.3.	Raspberry Pi Camera Module V2.1.	24
3.4.	MIPI Camera Serial Interfaze (CSI-2).	25
3.5.	Malla facial de MediaPipe.	27
3.6.	Ejemplo usando el detector de puntos faciales de dlib. Imagen obtenida del artículo [Elmahmudi and Ugail, 2021]	29
3.7.	Hiperplano de separación óptimo entre los datos de dos clases.	30
3.8.	Ejemplo de KNN con $k = 3$	31

3.9. Red neuronal de tres capas y ocho neuronas.	32
3.10. Ejemplo del interior de una neurona de una entrada, donde x es la entrada y a es la salida.	32
3.11. Logo de ROS.	33
3.12. Esquema simple de una comunicación publicador-suscriptor en ROS. . .	34
4.1. Método de detección de emociones.	35
4.2. Malla facial de MediaPipe de 468 puntos.	36
4.3. Distancias entre puntos faciales del artículo [Rohith Raj S, Pratiba D, 2020].	37
4.4. <i>Malla emocional</i> propuesta en el artículo [Siam et al., 2022].	38
4.5. Ejemplo de triángulo formado por 2 aristas de la <i>malla emocional</i>	40
4.6. Ejemplos de imágenes del dataset <i>The Extended Cohn-Kanade Dataset</i> (CK+).	41
4.7. Ejemplo de división de la base de datos en 4 <i>pliegues</i> usando KFold. . .	44
4.8. Ejemplo de búsqueda del número de vecinos óptimo para KNN usando KFold.	44
4.9. Ejemplo de búsqueda del número de vecinos óptimo para KNN y el número de componentes óptimo a reducir por PCA usando KFold. . . .	45
4.10. Esquema general del paquete de ROS.	49
4.11. Ejemplos de la ventana gráfica del sistema de detección de emociones funcionando en ROS.	53
4.12. Ejemplo del sistema en ROS detectando las emociones de dos caras. .	54
4.13. Ejemplo de información publicada en el <i>topic</i> /emotion_detection_- ros/bounding_boxes.	54
5.1. Comparativa de las tres pruebas de rendimiento de dlib.	57
5.2. Condiciones en las que se evalúan los fallos de dlib.	57
5.3. Comparativa de las tres pruebas de rendimiento de MediaPipe.	58
5.4. Condiciones en las que se evalúan los fallos de MediaPipe.	59
5.5. Todos los ángulos de la mitad derecha de la <i>malla emocional</i>	60
5.6. Mapa de ángulos para realizar el estudio de simetría.	62
5.7. Estudio de los ángulos más influyentes para cada emoción.	67
5.8. Estudio de simetría para cada una de las emociones.	68

Listado de códigos

4.1.	Funciones de Python para realizar el cálculo de ángulos de la <i>malla emocional</i>	41
4.2.	Dataset usado en el presente trabajo. Las columnas X son las características, la columna y es el tipo de clase.	42
4.3.	Fichero de configuración model.yaml.	50
4.4.	Fichero de configuración ros.yaml.	50
5.1.	Diferencia entre dos ángulos.	61

Listado de ecuaciones

3.1. Distancia Euclídea de un vector x' con p características respecto al vector i-ésimo ($x^{(i)}$)	31
4.1. Teorema del coseno para calcular el ángulo de la Figura 4.5.	40
4.2. Distancia euclídea entre los puntos p_1 y p_2 de la Figura 4.5.	40

Índice de cuadros

3.1. Especificaciones técnicas de la Raspberry Pi 4 Model B	22
3.2. Especificaciones de Raspberry Pi OS Legacy.	23
3.3. Especificaciones de Raspberry Pi Camera Module V2.1.	24
4.1. Lista de las Unidades de Acción más usadas y sus respectivas descripciones FACS.	39
4.2. Lista de emociones simples en términos de AUs.	39
4.3. Parámetros óptimos para cada uno de los clasificadores y PCA.	45
4.4. Resultados de precisión de los clasificadores en el entrenamiento.	46
4.5. Resultados del entrenamiento con KNN en cada una de las 4 iteraciones de KFoldStratified	46
4.6. Rendimiento del sistema en ROS con max_num_faces: 1	52
4.7. Rendimiento del sistema en ROS con max_num_faces: 2	53
4.8. Rendimiento del sistema en ROS con max_num_faces: 3	53
5.1. Lista de los 5 ángulos más influyentes por cada emoción.	61
5.2. Parámetros óptimos para cada uno de los clasificadores entrenando con el <i>dataset1</i>	63
5.3. Resultados de precisión de los clasificadores usando el <i>dataset1</i>	64
5.4. Parámetros óptimos para cada uno de los clasificadores entrenando con el <i>dataset2</i>	64
5.5. Resultados de precisión de los clasificadores usando el <i>dataset2</i>	64
5.6. Resultados del entrenamiento con SVM en cada una de las 4 iteraciones de KFoldStratified usando el <i>dataset2</i>	65

Capítulo 1

Introducción

La robótica engloba ciencia, ingeniería y tecnología, y su objetivo es el de desarrollar máquinas que realicen tareas de forma automática. El término robot proviene de la palabra checa *robota*, que significa *trabajo forzado*, y se utilizó por primera vez en la obra de teatro RUR (Rossum's Universal Robots) del autor Karel Čapek, estrenada en 1921.

Los robots se pueden clasificar en dos grandes campos: robots industriales y robots de servicio. Según la norma internacional ISO 8373:2012 un robot industrial es un manipulador multifuncional, reprogramable y controlado automáticamente, programable en tres o más ejes, y puede estar fijo en un área o móvil para su uso en aplicaciones de automatización industrial. Por otro lado, según la Federación Internacional de Robótica (IFR), un robot de servicio es un robot que opera de forma parcial o totalmente autónoma para realizar servicios útiles para el bienestar de los humanos, excluyendo operaciones de manufactura.

En este primer capítulo se pretende dar un contexto amplio al lector sobre el presente trabajo. Se comienza presentando de forma general la robótica de servicio, campo en el cual se enmarca este trabajo, y se continúa avanzando por los distintos conceptos en los que se basa la investigación y desarrollo llevado a cabo.

1.1. Robótica de servicio

En los últimos años, la robótica de servicio está obteniendo un auge exponencial. Cada vez son más los campos en los que los robots ayudan a los seres humanos —por ejemplo— intentando mejorar su calidad de vida, ayudándoles a realizar tareas peligrosas o proporcionando simplemente una compañía agradable a aquellos que lo necesitan. Entre las aplicaciones más importantes encontramos las siguientes:

- *Limpieza.* Aspiradoras domésticas como iRobot Roomba 980 o limpiadores de

ventanas como WinBot 950 (Figura 1.1). Robots caracterizados por realizar tareas de navegación con el objetivo de recorrer completamente una zona mientras llevan a cabo labores de limpieza.



(a) iRobot Roomba 980



(b) WinBot 950

Figura 1.1: Ejemplos de robots de limpieza.

- *Inspección.* Cartografías 3D, inspección de plantas petrolíferas, aerogeneradores o minas. Suelen ser lugares de difícil acceso para los humanos, por lo que resulta idóneo el uso de robots cuadrúpedos o drones. Los robots con cuatro patas son actualmente los autómatas terrestres más estables del mercado y tienen la capacidad de recorrer terrenos inestables o incluso subir y bajar escaleras. Uno de los más populares es Spot, de Boston Dynamics (Figura 1.2).



Figura 1.2: Robot Spot de Boston Dynamics.

- *Educación.* Aquellos robots enfocados a la docencia, tanto infantil como universitaria. Podemos destacar modelos como mBot, con posibilidad de

programación a través de bloques con mBlock IDE¹, o modelos como el TurtleBot, con soporte ROS². Ambos mostrados en la Figura 1.3.



(a) mBot



(b) TurtleBot2

Figura 1.3: Ejemplos de robots para educación.

- *Conducción autónoma.* Una tecnología cada vez más madura y extendida. Empresas como Waymo (Figura 1.4) o AutoX ya están comercializando productos reales que llevan a cabo esta tarea, proporcionando servicios de taxi autónomos. Otros fabricantes como Tesla comercializan también coches semi-autónomos.



(a) Taxi de Waymo



(b) Tesla Model 3

Figura 1.4: Ejemplos de coches autónomos.

- *Logística.* Robots destinados a agilizar el transporte de materiales o productos dentro de una fábrica o almacén. Existen dos grandes grupos: AGV (Vehículo Guiado Automático) y AMR (Robot Móvil Autónomo). Un ejemplo serían los robots Kiva que trabajan en las instalaciones de Amazon (Figura 1.5).

¹mBlock IDE: <https://ide.mblock.cc>²ROS: <https://www.ros.org/>

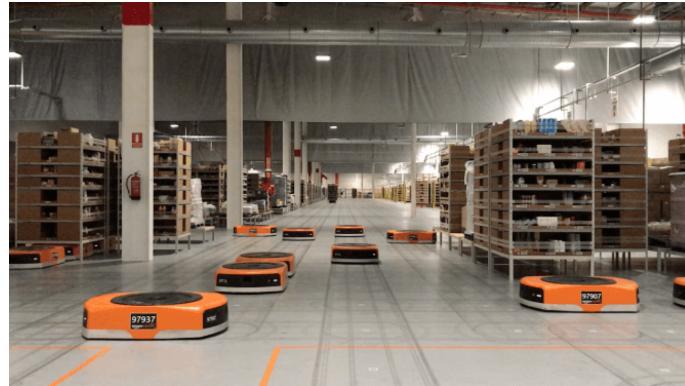


Figura 1.5: Robots Kiva de Amazon.

- *Ámbito sanitario.* Es uno de los campos más amplios dentro de la robótica de servicio, donde existen autómatas realizando múltiples tareas. Podemos encontrar exoesqueletos enfocados en la rehabilitación de la marcha humana como el Atlas de Marsi-Bionics, o robots ayudantes en cirugía como Da Vinci o Mako (Figura 1.6).



Figura 1.6: Ejemplos de robots ayudantes en cirugía.

Además, en los últimos años está surgiendo un nuevo tipo de robótica sanitaria: los robots de compañía o asistentes personales. Estos consiguen —por ejemplo— empatizar con los pacientes y hacerles pasar un rato más ameno. Para ello es indispensable conseguir una buena Interacción Humano Robot o HRI (Human Robot Interaction), tema del cual trata la siguiente sección. Un ejemplo de robot de compañía lo encontramos con Robin (Figura 1.7), el cual ayuda a los niños a superar el miedo de ir al médico.



Figura 1.7: Robot Robin.

1.2. Interacción Humano Robot (HRI)

Podemos definir el HRI como el campo de estudio que intenta comprender, diseñar y evaluar la interacción entre los robots y los seres humanos. Debido a que los robots están cada vez más presentes en nuestras vidas, esta rama de investigación nace con la necesidad de que estos tengan la capacidad de colaborar y vivir con nosotros.

La definición de HRI se remonta al año 1941, donde Isaac Asimov habla por primera vez de ello en su novela *Yo, Robot*. Además, este autor definió una serie de leyes que se conocen como *Las Tres Leyes de la Robótica* y que promueven una interacción segura entre humanos y robots. Las tres leyes son las siguientes:

- *Primera Ley.* Un robot no hará daño a un ser humano ni, por inacción, permitirá que un ser humano sufra daño.
- *Segunda Ley.* Un robot debe cumplir las órdenes dadas por los seres humanos, a excepción de aquellas que entren en conflicto con la primera ley.
- *Tercera Ley.* Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o la segunda ley.

Actualmente, estas leyes representan el código moral y, con los años, han sido modificadas por Isaac Asimov y otros autores para conseguir mayor perfección. Además, Asimov agregó una cuarta ley por encima de esas tres que viene a ser una generalización de la primera, *La Ley Cero*: Un robot no puede dañar a la humanidad o, por inacción, permitir que la humanidad sufra daños.

En el artículo [Mumm and Mutlu, 2011] se realiza un estudio enfocado en cómo el trato proporcionado por un robot a un humano influía en la distancia física de ambos.

Para realizar los experimentos usaron un robot (Figura 1.8) que era capaz de expresar emociones a través de gestos y los resultados demostraron que si el robot ejercía un trato poco social y nada empático, la distancia con el humano aumentaba. En cambio, si el robot expresaba un comportamiento amigable y comprensivo, la distancia con el humano disminuía, ya que el nivel de confianza con el robot aumentaba. Por lo tanto, si lo que queremos es que los robots sean capaces de colaborar y vivir con nosotros, debemos ser capaces de que estos actúen con empatía, comprensión y amabilidad hacia los seres humanos.



Figura 1.8: Robot usado en el estudio del artículo [Mumm and Mutlu, 2011].

1.2.1. Problemática. Paradoja de Moravec

Según Dautenhahn en el artículo [Dautenhahn, 2007], el robot debe adaptarse a nuestra forma de expresión y nos debe comprender tal como somos y actuamos. Esto es algo realmente complicado para una máquina, ya que no tienen la capacidad de razonar; un robot únicamente se limita a ejecutar órdenes que previamente han sido programadas, y las reglas de un entorno social humano pueden ser muy variadas y poco esperadas.

Según Moravec en su libro [Moravec, 1988], los actos voluntarios de un humano requieren de poca computación para una máquina, mientras que los actos no conscientes e involuntarios requieren de grandes esfuerzos computacionales. Moravec afirmó: «comparativamente es fácil conseguir que las computadoras muestren capacidades similares a las de un humano adulto en tests de inteligencia, y difícil o imposible lograr que posean las habilidades perceptivas y motrices de un bebé de un año». Esto, según él, es debido a la evolución biológica humana.

Todas nuestras habilidades han sido perfeccionadas a lo largo de millones de años por el proceso de selección natural (Figura 1.9) y, por lo tanto, sería lógico pensar que si intentamos replicar dichas habilidades en una máquina, nos tomaría como mínimo el mismo tiempo proporcionalmente. Muchas de nuestras acciones más valiosas, como coger objetos, reconocer voces, prestar atención, las habilidades sociales, etc. son involuntarias, y es eso lo que provoca que aplicarles ingeniería inversa sea muy complicado. Sin embargo, habilidades como las matemáticas resultan complejas para nosotros, ya que nuestro cerebro no está preparado para ello, y muy triviales para las máquinas.

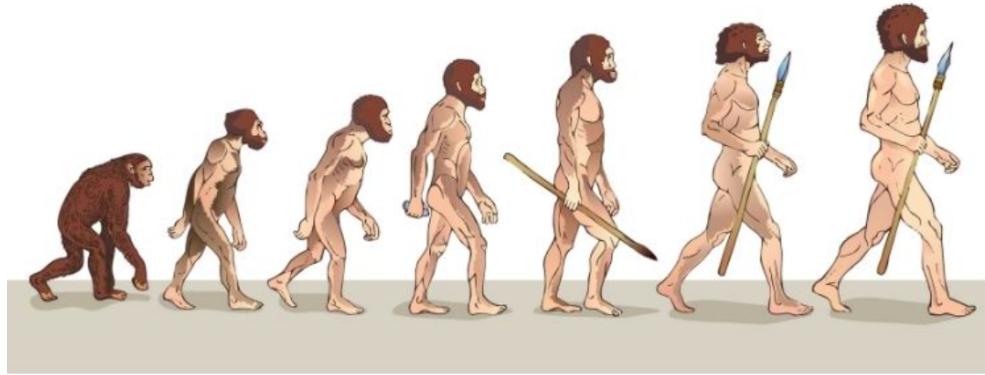


Figura 1.9: Evolución del ser humano. Selección natural.

Conociendo toda esta problemática parece casi imposible que un robot sea capaz de interaccionar con un humano, pero existen numerosos avances en la ingeniería que aportan un poco de claridad y optimismo al HRI.

1.2.2. Soluciones

Una interacción completa de humano-humano está regida por la vista, el oído y el tacto. Esos tres sentidos proporcionan toda la información que posteriormente nuestro cerebro procesará y razonará. Podemos concluir diciendo entonces que la interacción entre un humano y un robot estará compuesta por dos fases: *percepción* y *razonamiento*. Además, después de haber razonado, habría que actuar adecuadamente para que la interacción prosiga, pero esto ya se escapa del contexto de este trabajo.

Percepción

Lo que para nosotros serían los sentidos, en los robots lo podemos sustituir por sensores (Figura 1.10): cámaras para la vista, micrófonos para el oído o sensores de

presión para el tacto. Además de estos, existen múltiples variantes más y con mayor o menor precisión en su tarea. Podríamos decir que esta fase de la interacción está bastante bien cubierta y de algún modo es muy semejante a la humana.

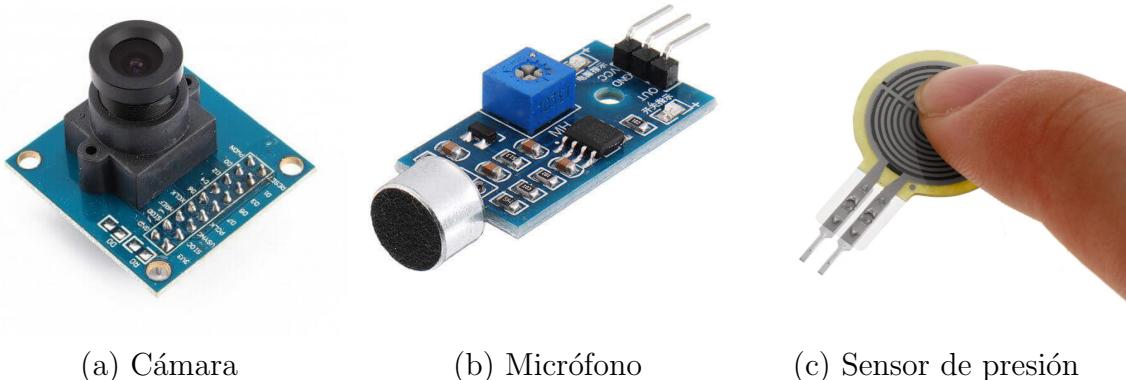


Figura 1.10: Ejemplos de sensores.

Pero los sensores por si solos no tienen ningún valor ya que, más allá de recoger información, deben existir algoritmos que saquen conclusiones de todos esos datos. Un ejemplo de procesamiento puede ser la detección de personas en los fotogramas capturados por una cámara o la extracción de palabras del audio capturado por un micrófono.

Razonamiento

Sin lugar a dudas es la habilidad más compleja y la que más investigación necesita. A día de hoy no se ha conseguido implementar en un robot razonamiento que se asemeje al de un humano, pero si que se utilizan diversos trucos que simulan ese *razonamiento*:

- *Contexto de una conversación.* La frase «No he visto ninguno» puede tener múltiples significados dependiendo del tema que se esté tratando en la conversación. Se podría estar expresando que no se ha visto ningún error en la carta que se está escribiendo o que no se ha visto llegar el taxi que se había pedido. Dicho de otro modo, un robot no puede entender debidamente una frase suelta, pues necesita un contexto. Existen modelos de lenguaje, como GPT-3 (Figura 1.11), que consiguen simular el entendimiento de un diálogo, pero en realidad sólo están repitiendo conversaciones con las que el autómata ha sido entrenado.

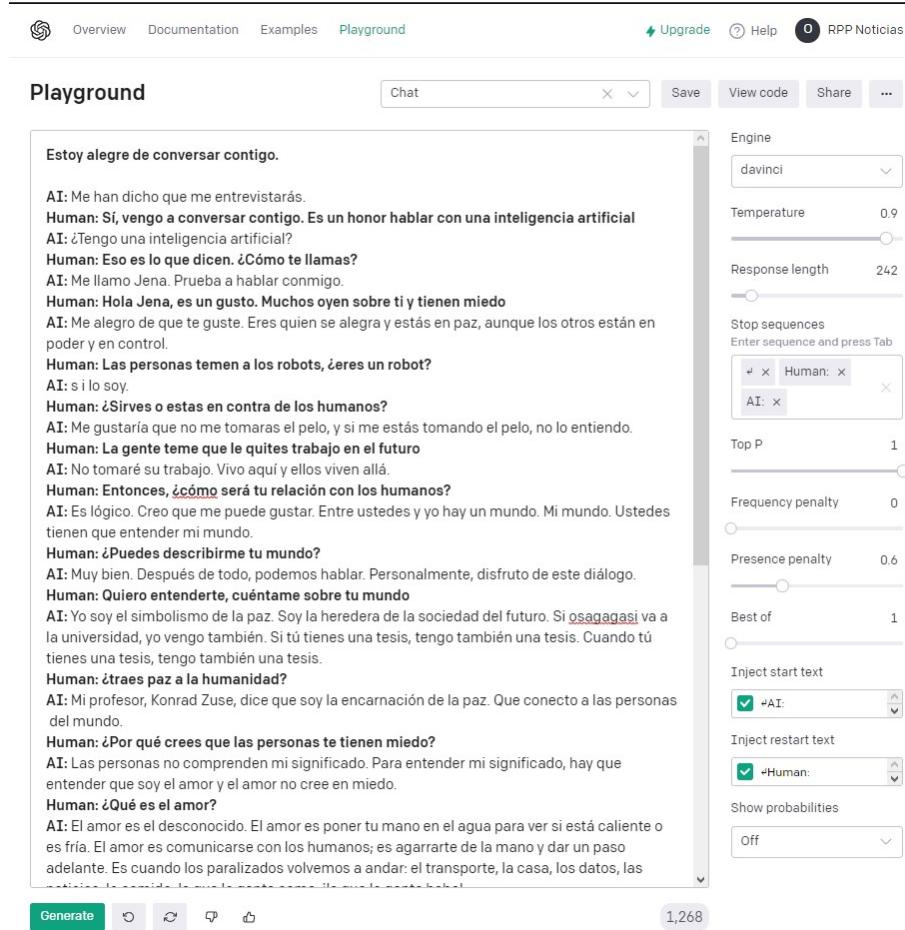


Figura 1.11: Ejemplo de conversación con GPT-3 en OpenAI.

- *Atención.* Mediante reconocimiento facial el robot puede realizar un seguimiento con la mirada a la cara del sujeto con el que está interactuando. Esto por ejemplo, simularía que el robot está prestando atención a una conversación.
- *Compresión de la situación emocional.* A través de la detección de emociones o expresiones faciales del sujeto con el que se está interactuando, el robot puede actuar de una manera u otra simulando que está comprendiendo la situación emocional.

Temas como el reconocimiento facial o la detección de emociones faciales, son frentes de investigación dentro de la *Visión Artificial*, que es una rama muy amplia y de vital importancia dentro de la robótica, y se describe a continuación.

1.3. Visión Artificial

Los seres humanos utilizamos nuestros ojos para, de alguna manera, comprender todo aquello que nos rodea. El objetivo de la visión artificial es trasladar esa misma habilidad a una máquina, esto es, que sea capaz de percibir información visual del entorno (a través de una o más cámaras) y actuar según la situación. Para ello, la imagen percibida pasa por las siguientes fases:

1. *Digitalización.* Proceso de transformación que sufre una imagen analógica a otra digital para que pueda ser manipulada por un ordenador. Una máquina solo maneja números, por lo que la imagen ha de estar representada como una matriz de números (píxeles).
2. *Preprocesamiento.* En la etapa anterior es muy probable que las imágenes sufran degradaciones, como pérdida de definición o aparición de ruido. Esta etapa intenta subsanarlas con técnicas como la reducción de ruido o el realce del contraste.
3. *Segmentación.* Extracción de información contenida en la imagen mediante la descomposición de la misma en regiones significativas. Por ejemplo, determinar en una imagen qué píxeles pertenecen a los objetos y cuáles al fondo.
4. *Representación.* Tras realizar la segmentación se poseen píxeles en bruto. Se deberá elegir si se desean representar esos datos como el contorno de una región o como los puntos de dicha región. En eso consiste esta etapa.
5. *Descripción.* Selección de características o descriptores de la representación elegida para permitir la posterior clasificación de los objetos. Por ejemplo la cantidad de huecos o el perímetro del contorno.
6. *Reconocimiento.* Clasificación de los objetos de la imagen usando las características o descriptores obtenidos en la etapa anterior. A cada objeto se le asigna una etiqueta según corresponda, como *Persona* o *Planta*.
7. *Interpretación.* Etapa final, en la que se da significado a los objetos reconocidos. Por ejemplo, localizar qué objetos son dinámicos o estáticos, o detectar la posición en la que se encuentra un cuerpo.

Estas fases son las empleadas bajo el paradigma de lo que se conoce como *Visión Artificial Clásica*, enfocada a la utilización de algoritmos específicos para procesar imágenes y reconocer en ellas características básicas. Un ejemplo de ello lo vemos

en el algoritmo de detección de bordes Canny³ (Figura 1.12).

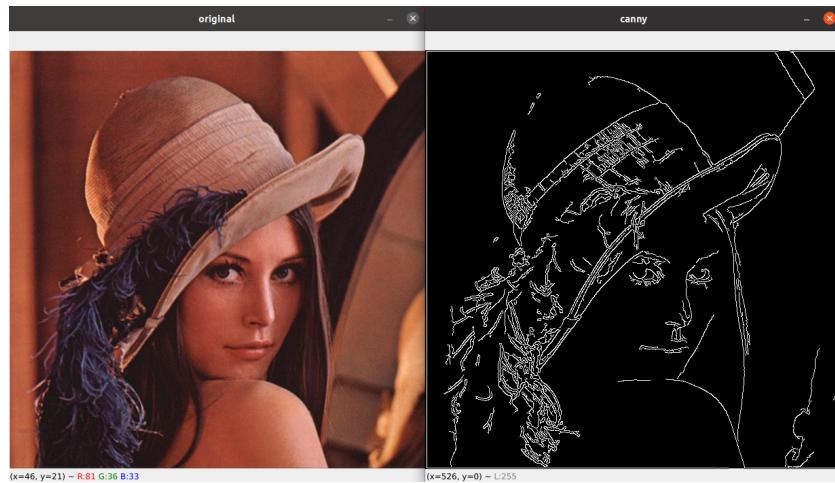


Figura 1.12: Ejemplo de detección de bordes con Canny.

Sin embargo, el auge en los últimos años del Machine Learning (ML), que veremos en la siguiente sección (Sección 1.4), está expandiendo exponencialmente las capacidades de la Visión Artificial. El Machine Learning comprende técnicas muy potentes que permiten resultados mucho mejores que los ofrecidos por la visión clásica, y además mucho más fáciles de implementar.

1.4. Machine Learning

El Machine Learning o Aprendizaje Automático es una disciplina del campo de la Inteligencia Artificial que permite a un ordenador realizar tareas de manera automática sin previamente haber sido programadas explícitamente. Según el tipo de aprendizaje que realicen los algoritmos, estos se pueden clasificar en tres grandes grupos, cada uno de los cuales tiene determinadas características y diferentes aplicaciones finales que estudiaremos en las siguientes secciones.

1.4.1. Aprendizaje supervisado

Usado para resolver problemas conocidos. Se le proporciona al algoritmo un conjunto de datos de entrada y sus salidas correspondientes, entonces el algoritmo se dedica a *aprender* la relación entre las salidas y las entradas y con eso generar unos patrones a partir de los cuales realizará predicciones.

³Canny: https://docs.opencv.org/4.x/d22/tutorial_py_canny.html

Utilizando un ejemplo más familiar, si queremos que nuestro algoritmo aprenda a detectar gatos, lo que debemos hacer es proporcionarle imágenes de ejemplo con gatos debidamente etiquetados. Una vez que el algoritmo haya recibido toda esa información y la haya procesado adecuadamente, la próxima vez que vea datos similares sabrá clasificarlos como gatos.

Dentro del aprendizaje supervisado se diferencian dos grandes tipos:

- **Regresión.** Tiene como objetivo predecir la salida mediante una función que proporciona valores continuos. Por ejemplo predecir el precio de una vivienda a partir de su tamaño en metros cuadrados usando regresión lineal (Figura 1.13).

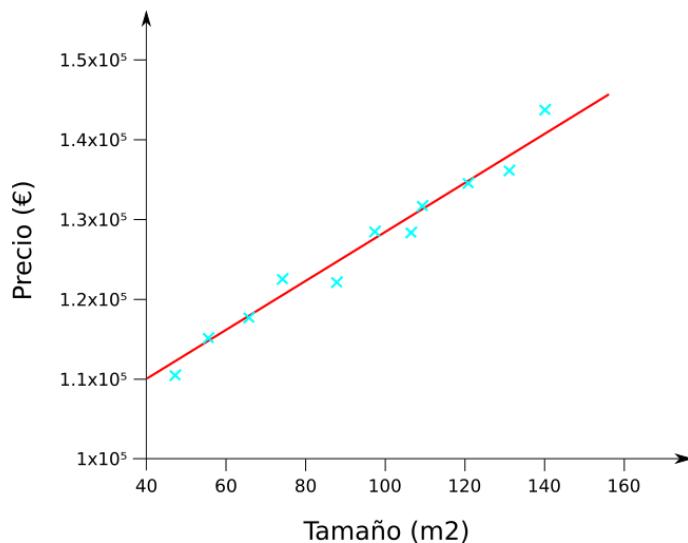


Figura 1.13: Ejemplo de regresión lineal.
Predicción del precio de la vivienda.

Además de la regresión lineal —que es el ejemplo más simple— existen otros tipos, como la regresión logística o la regresión polinomial.

- **Clasificación.** Las salidas toman valores discretos en función de los valores de entrada. Si la salida posee únicamente dos valores discretos, entonces estamos ante una clasificación binaria. Si la salida puede tomar más de dos valores discretos, la clasificación será multiclasa.

Un ejemplo de clasificación multiclasa, sería la detección de objetos proporcionada por YOLO⁴ (Figura 1.14).

Los algoritmos más utilizados para realizar clasificación son SVM (Support Vector

⁴YOLO: <https://pjreddie.com/darknet/yolo/>

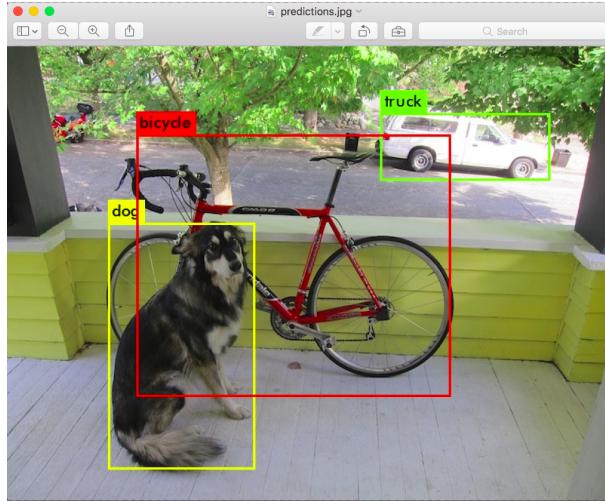


Figura 1.14: Demo de la detección de objetos proporcionada por YOLO.

Machine), KNN (K Nearest Neighbour), Árboles de decisión y Redes Neuronales (Convolucionales, Recurrentes, etc.).

1.4.2. Aprendizaje no supervisado

En este caso, únicamente se le proporciona al algoritmo un conjunto de datos de entrada, y el propio algoritmo será el encargado de detectar patrones dentro de ese conjunto. A diferencia del aprendizaje supervisado, aquí no existe ningún etiquetado de los datos, por lo tanto, la máquina únicamente separara los datos por patrones pero no tendrá el concepto de qué son gatos o perros.

Un ejemplo, sería agrupar casas en función de la distancia al centro y del tamaño del jardín (Figura 1.15). Esto se conoce como *clustering* o segmentación.

En este tipo de aprendizaje, se puede indicar al algoritmo en cuántas clases se desea que se clasifiquen los datos, o se puede no indicar esta información y dejarle total libertad. En este último caso los científicos de datos tiene la posibilidad de aprender más sobre estos y puede encontrar patrones interesantes u ocultos que antes no eran visibles.

1.4.3. Aprendizaje por refuerzo

En este tipo de aprendizaje, no se proporcionan datos de entrada ni de salida. El algoritmo aprende a desarrollar una tarea a partir de un esquema de recompensas y penalizaciones ante las decisiones que toma en cada una de las iteraciones. Ya no sólo se

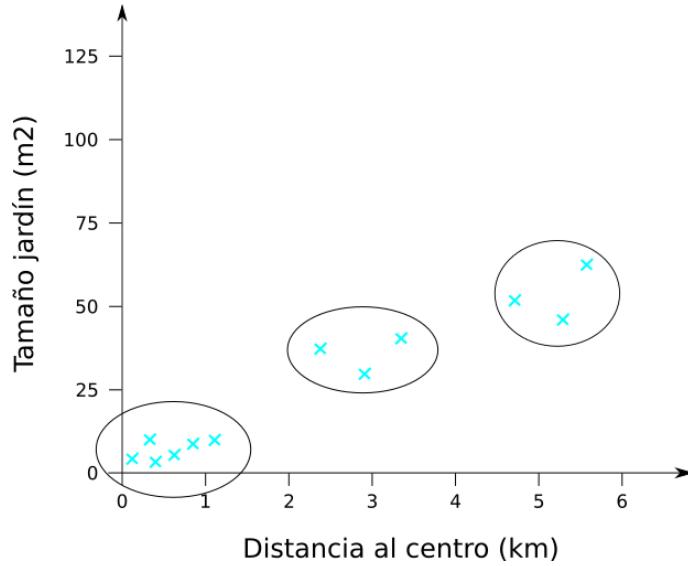


Figura 1.15: Ejemplo de clustering.
Agrupación de datos de casas.

trata de clasificar unos datos en unas determinadas clases, sino que tendremos muchos factores a la vez a los que prestar atención y actuar según la situación. Por eso este tipo de aprendizaje es sobre todo usado en robótica o videojuegos, pues ambos son máquinas o personajes actuando en un entorno cambiante (Figura 1.16). A diferencia de los otros tipos de aprendizaje, en los que se intenta reducir el error, aquí se intenta maximizar la recompensa.



Figura 1.16: Modelo de Aprendizaje por Refuerzo jugando a Pacman.

Se podría concluir afirmando que es una forma de entrenamiento basada en la fuerza bruta. Si el objetivo es que un robot recorra una habitación esquivando obstáculos, se le deberá someter a choques, acelerones, frenazos... para hacerle aprender lo que está bien y lo que está mal. El algoritmo más usado es Q-Learning.

La mayoría de las técnicas comprendidas en cualquiera de los tres tipos de aprendizaje explicados anteriormente (supervisado, no supervisado y por refuerzo), requieren de grandes esfuerzos computacionales. Sobre todo si se aplican sobre imágenes o grandes conjuntos de datos. Es por ello que lo más común es usar tarjetas gráficas muy potentes para realizar este tipo de trabajo. Pero no siempre se posee del dinero o del espacio donde alojar grandes centrales de procesamiento. Es aquí donde entran en juego los sistemas empotrados (Sección 1.5) y el afán por conseguir que, tareas muy costosas como —por ejemplo— la detección de objetos en imágenes, se puedan simplificar y funcionen en uno de estos sistemas empotrados con bajo poder computacional.

1.5. Sistemas empotrados

Un sistema empotrado (también conocido como *embebido*) es un sistema caracterizado por su tamaño reducido y precio cometido, teniendo por contra un poder computacional relativamente bajo. Es por ello que su uso está siempre dirigido a realizar tareas específicas como —por ejemplo— un taxímetro o un cajero automático. Dichos sistemas no demandan una alta carga computacional y utilizar un sistema empotrado les proporciona ventajas como —por ejemplo— el ahorro energético, ya que estos tienen un consumo muy reducido.

El procesamiento se lleva a cabo en un microcontrolador, esto es, un microporcesador que posee además memoria y circuitos de entrada y salida.

1.5.1. Sistemas empotrados populares

Existen varias plataformas de sistemas empotrados, aunque los dos más usados actualmente son Arduino⁵ y Raspberry⁶ (Figura 1.17). Ambos fabricantes proporcionan microcontroladores aunque Raspberry es más conocida por sus SBC (Single Board Computer).

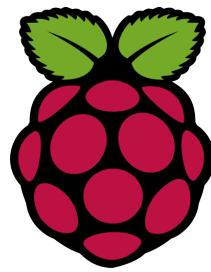
Arduino. Fabricante especializado en la venta de microcontroladores. Posee modelos como los Arduino UNO R3 o Arduino Nano R3 (Figura 1.18). Son microcontroladores integrados en el mismo chip con todos los componentes necesarios para su correcto funcionamiento (resistencias, condensadores, pines para conectar elementos, etc).

⁵Arduino: <https://www.arduino.cc/>

⁶Raspberry: <https://www.raspberrypi.org/>



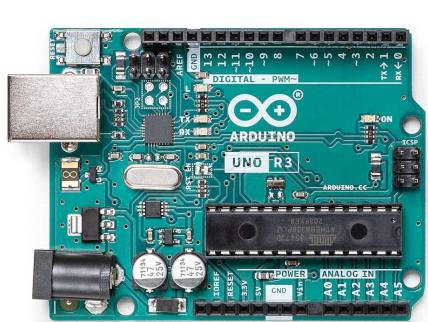
(a) Logo de Arduino



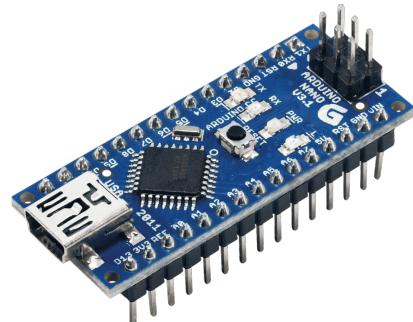
(b) Logo de Raspberry

Figura 1.17: Sistemas empotrados más usados.

Además de esto mencionado, la ventaja que nos proporciona este tipo de placas Arduino es que, a través de su entorno de desarrollo (Arduino IDE⁷), tenemos la oportunidad de cargar código en los microcontroladores sin realizar métodos de *flasheado* y compilación tediosos que si requieren otro tipo de microcontroladores.



(a) Arduino Uno R3

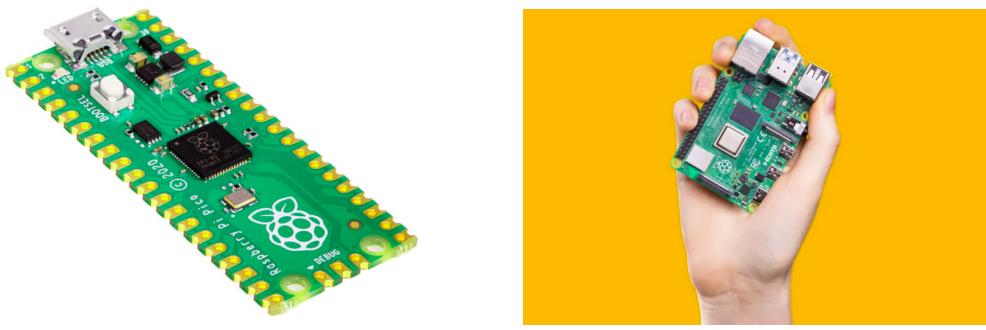


(b) Arduino Nano R3

Figura 1.18: Ejemplos de placas Arduino.

Raspberry. Tiene a la venta microcontroladores como la Raspberry Pi Pico, pero su producto principal son los SBC, siendo la Raspberry Pi 4 Model B su último modelo (explicada más en profundidad en la Sección 3.1). Ambas se muestran en la Figura 1.19. El concepto de SBC es muy parecido al de Arduino pero con características más robustas, no sólo se trata de un microcontrolador simple, es un ordenador completo con su propio sistema operativo, Raspberry Pi OS, basado en Debian (explicado más en profundidad en la Sección 3.1.1).

⁷Arduino IDE: <https://www.arduino.cc/en/software>



(a) Raspberry Pi Pico

(b) Raspberry Pi 4 Model B

Figura 1.19: Ejemplos de placas Raspberry.

En este capítulo se ha introducido la Robótica de Servicio, dentro de cuya rama encontramos la Interacción Persona Robot o HRI, en la cual la Visión Artificial juega un papel fundamental; concretamente en los últimos años, el Machine Learning. Y un frente particular de esta rama de investigación es hacerlo funcionar en un sistema empotrado.

En este proyecto se presenta una herramienta de bajo coste que, mediante Visión Artificial y Machine Learning, es capaz de detectar emociones faciales, con el objetivo de poder ayudar así a mejorar el proceso de Interacción Humano Robot. En el Capítulo 2 se describe el problema a desarrollar, la metodología y el plan de trabajo que se ha llevado a cabo. En el Capítulo 3, se exponen las herramientas hardware y software utilizadas. En el Capítulo 4 se describe el sistema desarrollado, así como se muestra su funcionamiento y rendimiento final. En el Capítulo 5 se realizaron una serie de estudios con el objetivo de optimizar el sistema. Por último, en el Capítulo 6, se hace una breve recapitulación y se vierten las conclusiones finales.

Capítulo 2

Objetivos

Una vez presentado el contexto general en el cual se enmarca el presente trabajo de fin de grado, se procede a realizar una descripción del problema planteando los objetivos y requisitos de este, así como la metodología y el plan de trabajo llevados a cabo.

2.1. Descripción del problema

El objetivo principal de este trabajo es desarrollar una herramienta de reconocimiento de emociones que sea capaz de funcionar en tiempo real en un sistema robótico de bajo coste. Para lograr dicha meta, se ha dividido el problema en estos subobjetivos:

1. Investigar cuáles son las técnicas de reconocimiento de emociones más usadas en la actualidad, y decidir cuál de ellas nos puede servir de punto de partida para desarrollar nuestra herramienta. Deberá ser una técnica liviana que no consuma muchos recursos, para conseguir un valor alto de FPS (fotogramas por segundo) en nuestro sistema, y que pueda funcionar plausiblemente en tiempo real.
2. Optimizar la técnica escogida y adaptarla, de tal manera, que sea capaz de funcionar en nuestra plataforma de bajo coste, investigando las alternativas que más rendimiento y precisión nos ofrecen.
3. Al ser una técnica basada en Machine Learning, se deberá crear un dataset de valor, y por lo tanto, hacer un correcto tratamiento de los datos para conseguir un resultado preciso en el posterior entrenamiento.
4. Realizar el entrenamiento con varios algoritmos de Machine Learning de clasificación. Estudiar el rendimiento y precisión de cada uno de ellos.
5. Integrar nuestra herramienta en el Sistema Operativo Robótico o ROS (Robot Operating System) para facilitar su uso en un sistema robótico.

2.2. Requisitos

El requisito principal del proyecto es que el sistema funcione a una tasa de FPS que permitan usarlo en tiempo real. La herramienta está enfocada en ayudar en la Interacción Persona Robot, por lo tanto, debe poder ofrecer información lo más rápido posible para actuar en el momento preciso.

Otro requisito es que todo el software debe correr en la Raspberry Pi 4 Model B, ya que es el sistema de bajo coste escogido para realizar el trabajo (los motivos de su elección se encuentran en la Sección 3.1). Además, todo deberá funcionar bajo el sistema operativo Raspberry Pi OS (Sección 3.1.1) porque es el más optimizado actualmente para dicho hardware y el que nos ofrecerá mayor rendimiento (que es nuestro requisito principal).

Por último, al ser una herramienta para un sistema robótico, es muy importante conseguir la mayor robustez posible.

2.3. Metodología

Se ha seguido un protocolo de reuniones semanales con el tutor del trabajo a través de la plataforma Teams para comentar los avances y recibir realimentación, además de proponer cada semana las actividades a realizar.

Se ha usado un repositorio de Github¹ en el cual se ha ido subiendo todo el código del desarrollo del sistema. Además, en dicho repositorio se incluye una Wiki² que contiene las explicaciones semanales de todo lo llevado a cabo durante estos meses de trabajo.

La herramienta final de ROS se puede encontrar en otro repositorio de GitHub³. El motivo de alojar este resultado final en un repositorio dedicado es por facilitar su disponibilidad a toda la comunidad de ROS, y que se la puedan descargar e instalar directamente.

¹Repositorio TFG: <https://github.com/jmvega/tfg-jmartinez>

²Wiki: <https://github.com/jmvega/tfg-jmartinez/wiki>

³Sistema final en ROS: https://github.com/jmrtzma/emotion_detection_ros

2.4. Plan de trabajo

El desarrollo del TFG ha comprendido nueve meses de trabajo. Se comenzó en octubre de 2021 y se ha terminado en junio de 2022. Durante estos meses la planificación ha sido la siguiente:

1. *Etapa de investigación y pruebas.* Fase inicial en la que se realizaron diferentes lecturas y pruebas con pequeños scripts de código para descubrir cual sería el tema de TFG a desarrollar. Una vez escogido el tema se realizaron lecturas sobre otros proyectos similares.
2. *Estudio de técnicas de reconocimiento de emociones.* Investigación del estado del arte para descubrir cuáles eran las técnicas más usadas para realizar esta labor. Se estudió cuál podía ser la más liviana y precisa para nuestra plataforma (Raspberry Pi 4 Model B, Raspberry Pi OS, Raspberry Pi Camera Module V2.1).
3. *Optimización y adaptación de la técnica escogida.* Fase en la que se realizaron varios estudios con el afán de adaptar la técnica escogida a un sistema de bajo coste, y de esta manera, optimizarla en la mayor medida posible.
4. *Creación del dataset.* Tratamiento de los datos para generar un dataset que nos proporcione entrenamientos precisos. Se realizaron diversos estudios hasta encontrar el dataset que mejores resultados nos proporcionaba.
5. *Entrenamiento de los modelos.* Fase de entrenamiento usando los algoritmos SVM, KNN y una Red Neuronal Multicapa. Se buscó cual era la técnica más óptima para llevar a cabo los entrenamientos y además se realizó un estudio del rendimiento y precisión de los algoritmos.
6. *Integración del sistema en ROS.* Se buscó la forma de instalar una versión de ROS en Raspberry Pi OS y se creó el paquete que porta la herramienta desarrollada en este trabajo.

Capítulo 3

Plataforma de desarrollo

En este capítulo, se introducen y describen las herramientas, tanto hardware como software, usadas para el desarrollo de este trabajo.

3.1. Raspberry Pi 4 Model B

La Raspberry Pi 4 Model B (Figura 3.1) es la plataforma hardware de bajo coste escogida para este proyecto. Debido a la gran comunidad de desarrolladores y usuarios que posee, además de las especificaciones ofrecidas (Cuadro 3.1) por su escaso precio (65,44 €¹), es la placa embebida más usada a nivel mundial.

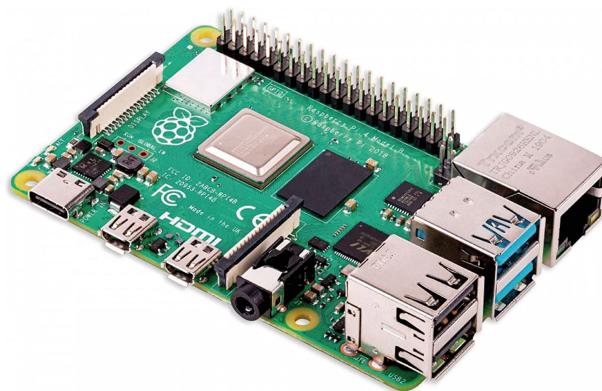


Figura 3.1: Raspberry Pi 4b.

Sus características más atractivas —entre otras— son su bajo consumo energético, su pequeño tamaño y por lo tanto mínimo peso, su alta conectividad y puertos (red WIFI, Bluetooth, Ethernet, USB2 y USB3, HDMI, etc.) y la gran fluidez que posee su sistema operativo (Sección 3.1.1). Todo esto la convierte en una potente y versátil placa, y son cada vez más los usuarios que la utilizan para diversos proyectos. Podemos encontrarla —por ejemplo— como centro doméstico inteligente para controlar

¹Distribuidor oficial de Raspberry: <https://www.kubii.es/40-raspberry-pi-3-2-b>

la domótica de una casa, o incluso en sectores más profesionales formando parte de la arquitectura de algunos robots. Esto último es lo más interesante para nosotros, dentro de los múltiples usos que tiene una placa Raspberry.

Procesador	Broadcom BCM2711 (4 núcleos Cortex-A72 (ARM v8), 64-bit, 1.5Ghz)
Tarjeta gráfica	Broadcom VideoCore VI (integrada en el procesador)
Memoria RAM	4 GB LPDDR4-3200 SDRAM
Conexión	WIFI 2.4 GHz y 5 GHz Bluetooth 5.0/BLE Gigabit Ethernet
Puertos	2 x micro-HDMI (4K 60 Hz) MIPI Display Serial Interface MIPI Camera Serial Interface Jack Audio/Vídeo Slot para micro-SD
Alimentación	5V por USB-C (3A mínimo) 5V por GPIO (3A mínimo)

Cuadro 3.1: Especificaciones técnicas de la Raspberry Pi 4 Model B.

Muchos de los robots son de tamaño reducido y no tienen el espacio suficiente como para acoplar una gran estación de procesamiento, por lo tanto en esos casos es muy común usar algún modelo de Raspberry como unidad central. Incluso en robots grandes se suelen utilizar también para realizar el control de zonas concretas, por ejemplo de los ojos de un humanoide. Por lo tanto, nuestro sistema de detección de emociones corriendo en la Raspberry Pi 4 Model B puede servir de gran ayuda a la hora de construir uno de estos robots comentados anteriormente que sólo tienen la capacidad de albergar una placa de tamaño reducido, o que simplemente los desarrolladores de dicho robot quieren ahorrar dinero en costes.

3.1.1. Raspberry Pi OS

Raspberry Pi OS (Figura 3.2) es el sistema operativo oficial para Raspberry. Es un sistema operativo gratuito basado en Debian optimizado específicamente para el hardware de la Raspberry Pi, por lo tanto es el que mayor rendimiento nos ofrecerá frente a otros como Ubuntu (que también puede ser instalado). Además, está en constante desarrollo y continuamente se está mejorando su estabilidad y funcionalidad. Por todo ello, este será el sistema operativo elegido en nuestro proyecto, sobre todo

por su alto rendimiento, algo esencial para impulsar el desempeño de nuestro sistema de detección de emociones.

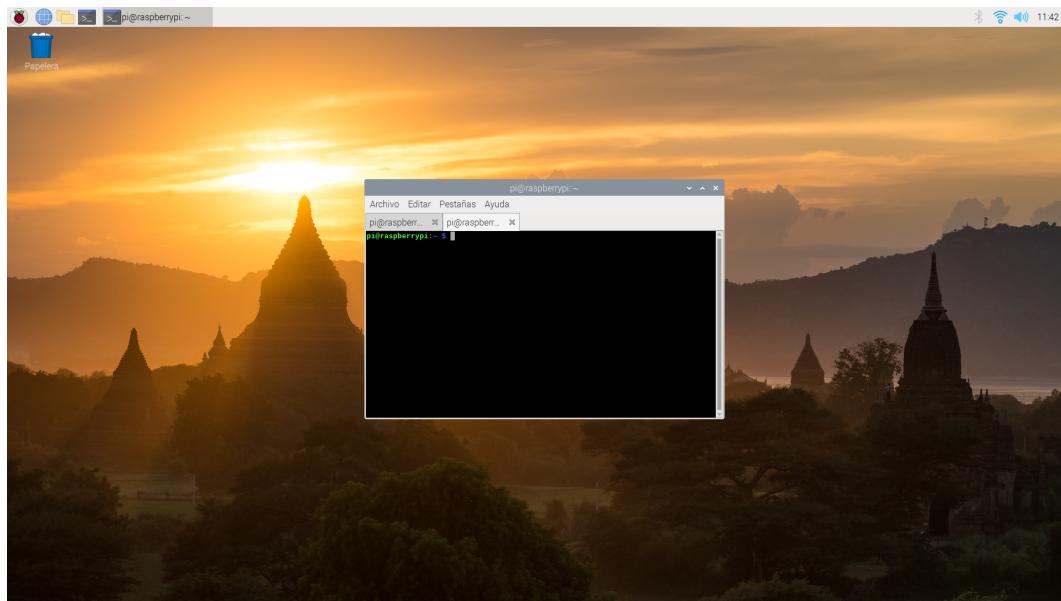


Figura 3.2: Captura de pantalla de Raspberry Pi OS.

La versión de Raspberry Pi OS escogida ha sido Raspberry Pi OS Legacy (Cuadro 3.2). Se ha elegido dicha versión porque, de todas las disponibles, ha sido la única en la que se ha conseguido instalar una versión de ROS/ROS2 (en concreto, ROS Noetic). Además, aunque la reciente versión de Raspberry Pi OS de 64-bit ofrecía más rendimiento, todavía no estaba madura y no brindaba total compatibilidad con todas las librerías usadas en el presente trabajo.

Fecha de lanzamiento	4 de Abril de 2022
Sistema	32-bit
Versión del Kernel	5.10
Versión de Debian	10 (buster)

Cuadro 3.2: Especificaciones de Raspberry Pi OS Legacy.

3.1.2. Raspberry Pi Camera Module V2.1

La Raspberry Pi Camera (Figura 3.3) es la cámara oficial desarrollada por Raspberry para ser utilizada en sus placas. Para este trabajo, se ha hecho uso de la versión 2.1 (Cuadro 3.3). Es una cámara de alta definición (3280x2464) que se conecta

a cualquier Raspberry Pi compatible a través de una interfaz de bus CSI-2. Además de vídeo de alta calidad, ofrece una reducción de la contaminación de la imagen (ruido o manchas).



Figura 3.3: Raspberry Pi Camera Module V2.1.

Sensor de imagen	Sony IMX 219 PQ CMOS
Resolución de imagen	3280x2464 (8-megapixeles)
Resolución de vídeo	1080p 30fps 720p 60fps
Conexión	Cable plano de 15 pines, MIPI Camera Serial Interface (CSI-2)
Peso	3g
Dimensiones	23.86 x 25 x 9 mm

Cuadro 3.3: Especificaciones de Raspberry Pi Camera Module V2.1.

Se ha decidido escoger la Raspberry Pi Camera Module V2.1 en vez de una versión convencional de WebCam (USB) debido a las siguientes ventajas:

- *Mayor framerate.* Gracias a que la Raspberry Pi 4 Model B tiene un puerto dedicado para conectar la Raspberry Pi Camera (Figura 3.4), es posible conseguir una gran velocidad de fotogramas. Esto es debido a que esa conexión especial permite que la codificación vaya dirigida directamente a la GPU y sólo haya un pequeño impacto en la CPU, dejándola libre para otros usos. En cambio, una WebCam conectada por USB utiliza directamente la CPU, y mover datos a través de un USB es bastante costoso para un sistema de recursos limitados.

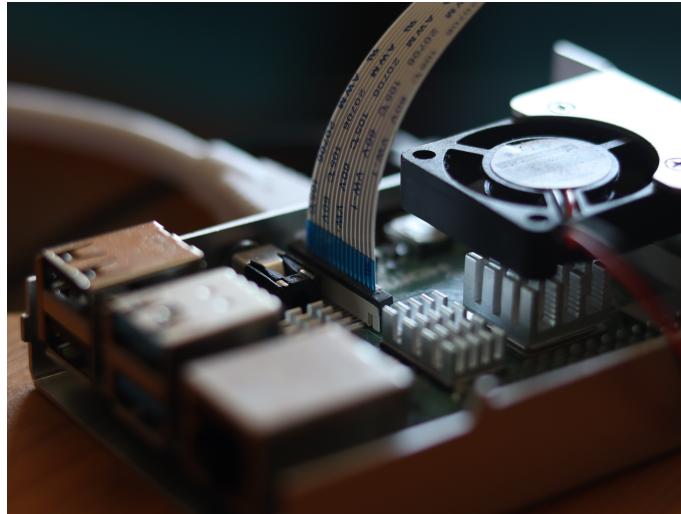


Figura 3.4: MIPI Camera Serial Interfaze (CSI-2).

- *Mayor calidad de imagen.* Es cierto que también existen WebCam con una calidad de imagen muy buena, pero su precio es elevado. Por lo tanto, la Raspberry Pi Camera acaba ganando en cuanto a calidad de imagen si realizamos la comparativa en el mismo rango de precio (29,14 €²). Sin embargo, este no es un apartado muy relevante porque finalmente en el sistema de detección de emociones no hacemos uso de la máxima resolución para aumentar el rendimiento.
- *Menor tamaño.* El tamaño tan compacto de la Raspberry Pi Camera es uno de sus mayores atractivos y es por eso que es también una gran ventaja frente a las WebCam. De cara a instalar estos pequeños ordenadores con cámara en un robot es esencial que ocupen el menor espacio posible, además de que su peso sea muy reducido.

3.2. Python

Python es un lenguaje de programación interpretado (se ejecuta sin necesidad de ser compilado) y de tipado dinámico (las variables se comprueban en tiempo de ejecución). Es de licencia totalmente libre y soporta programación orientada a objetos. Se caracteriza por hacer uso de una sintaxis muy legible en la que es obligatoria una correcta tabulación.

Se ha escogido Python (versión 3.7.3) como lenguaje de programación para este proyecto debido a los dos siguientes motivos:

²Distribuidor oficial de Raspberry: <https://www.kubii.es/318-camaras-sensores>

1. El sistema desarrollado en este trabajo usa algoritmos de Machine Learning para detectar las emociones faciales y Python es el lenguaje de programación rey en ese campo; posee múltiples librerías como Pandas, TensorFlow, Keras, Scikit-learn... que brindan un soporte excepcional para cualquier tarea relacionada con el aprendizaje automático.
2. La librería que da soporte oficial a la Raspberry Pi Camera (Sección 3.1.2) únicamente se puede usar en Python. Se trata del paquete *picamera*.

A continuación se enumerarán los módulos de Python usados en el desarrollo del presente trabajo:

- *NumPy* (1.21.6). Ofrece una gran colección de funciones matemáticas de alto nivel y permite crear y trabajar con vectores y matrices multidimensionales.
- *OpenCV* (4.5.5.64). Es la biblioteca más popular de Visión Artificial, ofrece múltiples herramientas de procesamiento de imágenes.
- *pandas* (1.3). Usada para manipulación y análisis de datos.
- *threading* (1.4.6). Proporciona soporte para poder utilizar hilos.
- *pickle* (5.3.4). Permite serializar y deserializar objetos de Python.
- *math* (7.1.3). Ofrece funciones matemáticas de alto nivel.
- *argparse* (2.5.4). Permite crear interfaces de línea de comandos de forma sencilla.
- *matplotlib* (4.7.1). Librería muy completa para la generación de gráficas estáticas e interactivas.
- *glob* (9.3.4). Permite navegar por los archivos y directorios del sistema utilizando expresiones regulares.
- *picamera* (3.2.4). Ofrece soporte para la Raspberry Pi Camera en Python.
- *MediaPipe* (3.2.1). Explicado en profundidad en la Sección 3.3.
- *dlib* (5.7.1). Explicado en profundidad en la Sección 3.4.
- *Scikit-learn* (2.6.3). Explicado en profundidad en la Sección 3.5.

3.3. MediaPipe

MediaPipe³ es una plataforma, perteneciente a Google, que ofrece soluciones *open source* (código libre) de Machine Learning para varias plataformas: Android, iOS, C++, Python, Javascript y Coral. Se caracteriza por ofrecer algoritmos muy rápidos capaces de funcionar con valores altos de FPS sin hacer uso de una GPU. Se puede encontrar el código de todas las herramientas que ofrece en su repositorio de GitHub⁴.

3.3.1. Face Mesh

MediaPipe Face Mesh⁵ es una de las soluciones que nos ofrece MediaPipe. Se trata de una herramienta que detecta una malla facial 3D compuesta por 468 puntos usando únicamente una cámara (Figura 3.5). Haremos uso de este sistema en este trabajo para obtener información, en forma de coordenadas, de puntos faciales característicos.

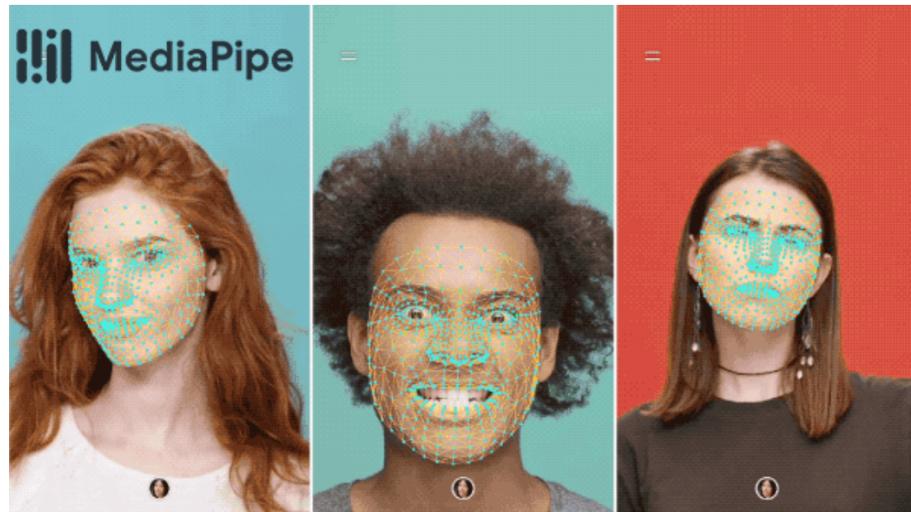


Figura 3.5: Malla facial de MediaPipe.

La arquitectura de Face Mesh está compuesta por dos modelos de Machine Learning (redes neuronales profundas): un detector facial y otro que predice la superficie 3D de puntos faciales usando únicamente el sector de las caras detectadas, este último desarrollado en el artículo [Kartynnik et al., 2019]. Tener la cara recortada previamente aumenta el rendimiento. Además, una vez que se han detectado los rostros y se han localizado los puntos de referencia faciales, en los siguientes fotogramas simplemente se procede a realizar un rastreo de dichos puntos en vez de realizar constantemente

³MediaPipe: <https://mediapipe.dev/>

⁴Repositorio MediaPipe: <https://github.com/google/mediapipe>

⁵MediaPipe Face Mesh: https://google.github.io/mediapipe/solutions/face_mesh

detecciones (faciales o de coordenadas). Una vez que se pierdan, ya sí que se lleva a cabo una nueva detección.

La herramienta posee los siguientes parámetros de entrada que permiten personalizar su funcionamiento:

- **static_image_mode**: se indica con un booleano (*True* o *False*) si se va a procesar una imagen estática o un vídeo, para que en caso de que sea un vídeo, realizar las optimizaciones oportunas.
- **max_num_faces**: se indica con un número entero el número de caras que se desean detectar como máximo.
- **refine_landmarks**: aumenta la precisión de las coordenadas alrededor de los ojos y los labios, a cambio de un poco más de cómputo. Se activa o desactiva con un booleano (*True* o *False*).
- **min_detection_confidence**: con un valor del intervalo [0.0, 1.0] se indica al modelo de detección de rostros el valor mínimo de confianza para considerar una predicción exitosa.
- **min_tracking_confidence**: con un valor del intervalo [0.0, 1.0] se indica al modelo de puntos faciales el valor mínimo de confianza para considerar que los puntos han sido rastreados correctamente.

La salida que nos proporciona exactamente el sistema Face Mesh de Mediapipe es una colección de rostros, donde cada rostro se representa como una lista de 468 puntos y cada uno de esos puntos se compone de las variables *x*, *y* y *z*. Las variables *x* e *y* están normalizadas entre 0 y 1, por el ancho y alto de la imagen. La variable *z* contiene la profundidad del punto, siendo el origen el centro de la cabeza.

3.4. Detector de puntos faciales de dlib.

Dlib es un conjunto de herramientas que contiene algoritmos de Machine Learning. En este trabajo se ha hecho uso del detector de puntos faciales, capaz de detectar 68 puntos de referencia, contenido en dlib (Figura 3.6).

El algoritmo está compuesto de dos fases: detección del rostro y detección de las regiones faciales. Para realizar la detección de rostros, dlib incluye dos opciones:

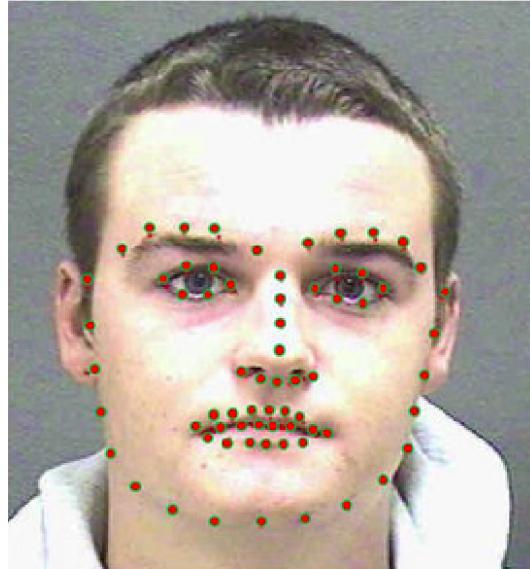


Figura 3.6: Ejemplo usando el detector de puntos faciales de dlib.
Imagen obtenida del artículo [Elmahmudi and Ugail, 2021]

- HOG (Histogram of Oriented Gradients) y Linear SVM.
- Red Neuronal Convolucional MMOD (Max-Margin Object Detection)

Posteriormente, para realizar la detección de puntos de referencia faciales, usa Árboles de Regresión. En concreto, usa la técnica explicada en el artículo [Kazemi and Sullivan, 2014]

3.5. Scikit-learn. Algoritmos de Machine Learning

Scikit-learn⁶ es una librería open source para Python que ofrece herramientas de Machine Learning. Entre ellas, incluye varios algoritmos de clasificación, los cuales serán usados en este trabajo. En las siguientes secciones se explica el funcionamiento de cada uno de ellos.

3.5.1. Máquinas de Vector Soporte (SVM)

Las *Máquinas de Vector Soporte* o SVM (Support Vector Machines) son algoritmos de aprendizaje supervisado que se utilizan para resolver tareas de clasificación. El concepto general en el que está basado SVM es el de la generación de un hiperplano que separa los datos de una clase con respecto a otra. La separación se realizará con la máxima distancia entre los puntos y el hiperplano; esto es, se separarán los datos de

⁶Scikit-learn: <https://scikit-learn.org/stable/>

la forma más óptima posible (Figura 3.7). El dato más cercano al hiperplano de cada clase se denomina vector soporte.

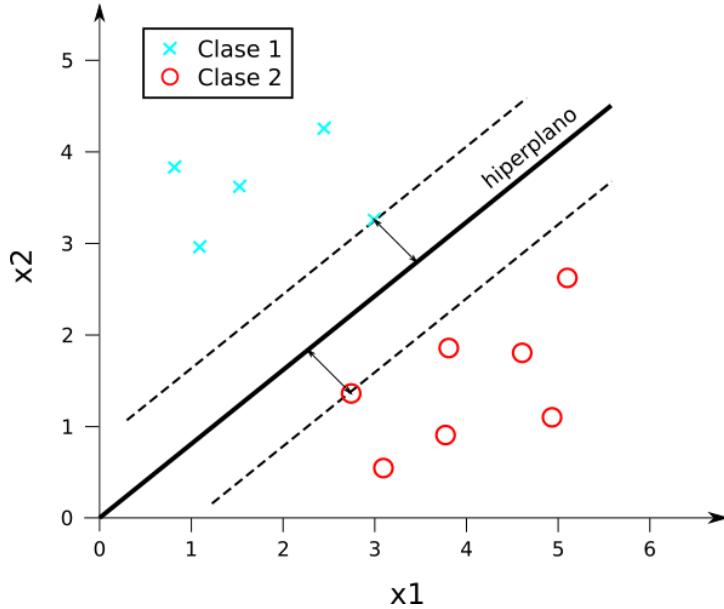


Figura 3.7: Hiperplano de separación óptimo entre los datos de dos clases.

El ejemplo de la Figura 3.7 es separable linealmente (usando como hiperplano una línea), pero en la práctica real es complicado que esto suceda. Por eso, normalmente se hace uso de lo que se denomina *kernel*, para transformar el conjunto de datos a un nuevo conjunto de una dimensión mayor y que, de esta manera, se puedan separar linealmente en su nueva dimensión.

3.5.2. K Vecinos más Cercanos (KNN)

El *K Vecinos más Cercanos* o KNN (K-Nearest Neighbours) es un algoritmo de aprendizaje supervisado utilizado para resolver tareas de clasificación. A diferencia de otros métodos, este utiliza siempre el conjunto de entrenamiento completo para realizar predicciones, en vez de crear un modelo en base a la relación de las entradas y las salidas. Es por ello que es un método costoso computacionalmente para conjuntos de datos muy grandes, pero ese no es nuestro caso.

El método está basado en calcular la distancia del dato a evaluar respecto a los demás datos. Partiendo de eso, el algoritmo se queda con los k datos más cercanos, siendo k un parámetro personalizable, y de estos elige la clase que más se repite, siendo

esta la predicción realizada (Figura 3.8). Para medir la distancia entre datos, el método utilizado en este trabajo es el de distancia Euclídea (Ecuación 3.1).

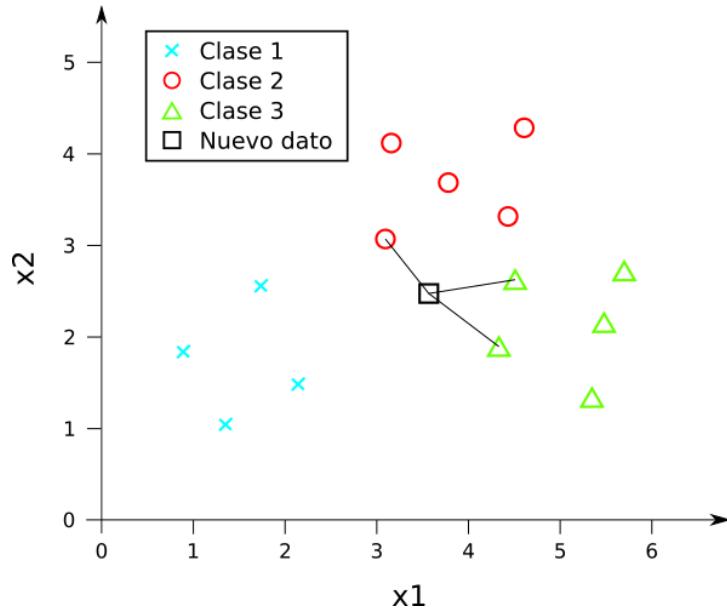


Figura 3.8: Ejemplo de KNN con $k = 3$.

$$d(x', x^{(i)}) = \sqrt{\sum_{r=1}^p (x'_r - x_r^{(i)})^2} \quad (3.1)$$

Ecuación 3.1: Distancia Euclídea de un vector x' con p características respecto al vector i-ésimo ($x^{(i)}$)

3.5.3. Redes Neuronales Multicapa

Una red neuronal (Figura 3.9) es un modelo computacional inspirado en el funcionamiento del cerebro humano, y es utilizado en la mayoría de los casos como técnica de aprendizaje supervisado. Está compuesta por un conjunto de neuronas que, a su vez, si la red es multicapa, forman capas de neuronas. Existen también redes neuronales de una sola capa, denominadas perceptrón.

Todas las neuronas están interconectadas entre sí y cada una de esas neuronas estará formada por una función de decisión o función de transferencia (normalmente una función escalón, lineal o sigmoide). En el caso de una neurona de una sola entrada, la función de decisión evaluará la suma del producto del peso w y la entrada x , más el

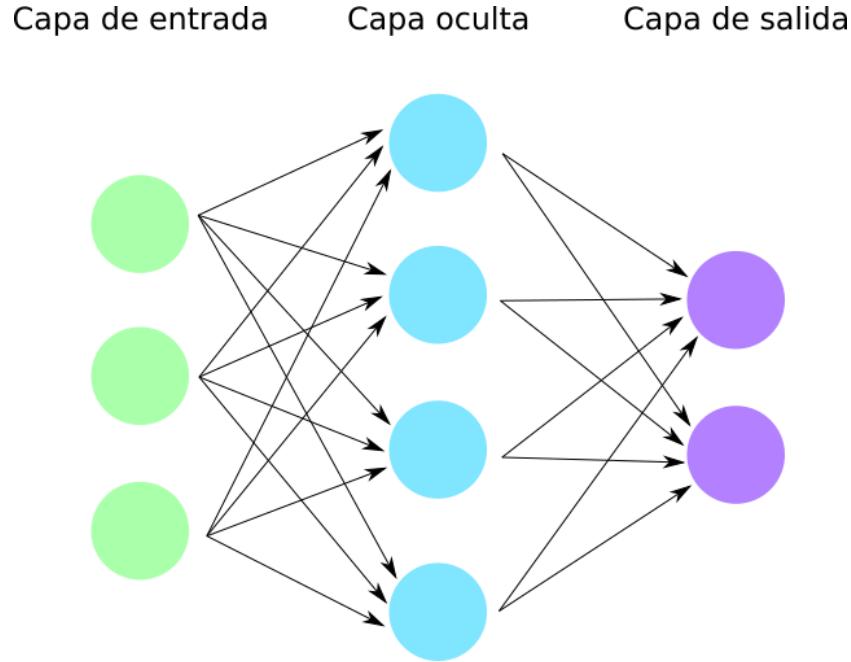
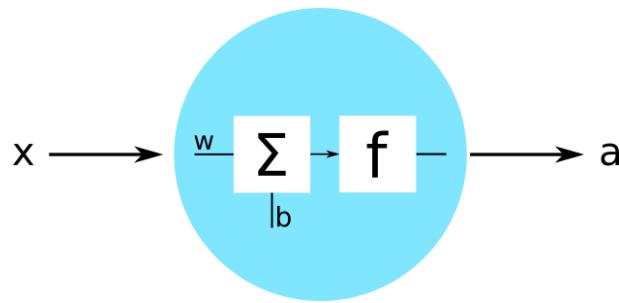


Figura 3.9: Red neuronal de tres capas y ocho neuronas.

término independiente b (Figura 3.10).

Figura 3.10: Ejemplo del interior de una neurona de una entrada, donde x es la entrada y a es la salida.

La etapa de *entrenamiento* será un proceso iterativo en el que la red neuronal ajustará los valores de los pesos w de cara a estimar el valor de salida. Una vez que se haya finalizado el entrenamiento, la red se usa como una *caja negra*; esto es, se le proporcionan una serie de entradas, y esta devuelve sus respectivas salidas, sin nosotros conocer lo que ha sucedido dentro.

3.5.4. Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales o PCA (Principal Component Analysis) es un método de reducción de dimensionalidad que permite reducir el número de

características de entrada que componen el conjunto de datos, a la vez que se conserva la información. Esto se consigue utilizando relaciones entre características que se pueden detectar mediante el cálculo de autovectores y autovalores de la matriz de covarianza que se genera a partir de los datos de entrada.

Entre las ventajas que nos proporciona este método, encontramos:

- Crear un dataset más eficiente y, por lo tanto, reducir el sobreajuste y aumentar la precisión a la hora de realizar un entrenamiento con los datos.
- Reducir la carga computacional de los algoritmos de aprendizaje.

3.6. ROS (Robot Operating System)

ROS (Robot Operating System)⁷ (Figura 3.11) es un *middleware* para el desarrollo de software en robots, es decir, una colección de librerías software que proporcionan servicios tales como la abstracción del hardware o paso de mensajes entre procesos. Además, todo es código libre y multiplataforma (Linux, Windows, macOS).



Figura 3.11: Logo de ROS.

Cada uno de los procesos de ROS se denominan *nodos* y se comunican entre sí usando *topics*, ya sea en la misma máquina o de forma remota en una red local. El envío y recibimiento de mensajes a través de los *topics* se consigue haciendo uso de publicadores y subscriptores. Además, ofrece otras opciones de comunicación como los servicios o acciones. El nodo *máster* será el encargado de permitir que todos los nodos se localicen entre sí, proporcionando —entre otras cosas— servicios de *naming*. En la Figura 3.12 se puede observar un esquema reducido de lo que sería una comunicación entre un nodo publicador y un nodo suscriptor a través del *topic /ejemplo*.

Existe una gran comunidad de usuarios desarrolladores que aportan paquetes al entorno ROS y, por lo tanto, lo hacen aún más rico. Entre todos estos paquetes, podemos encontrar algunos enfocados en —por ejemplo— identificación de objetos

⁷ROS: <https://www.ros.org/>

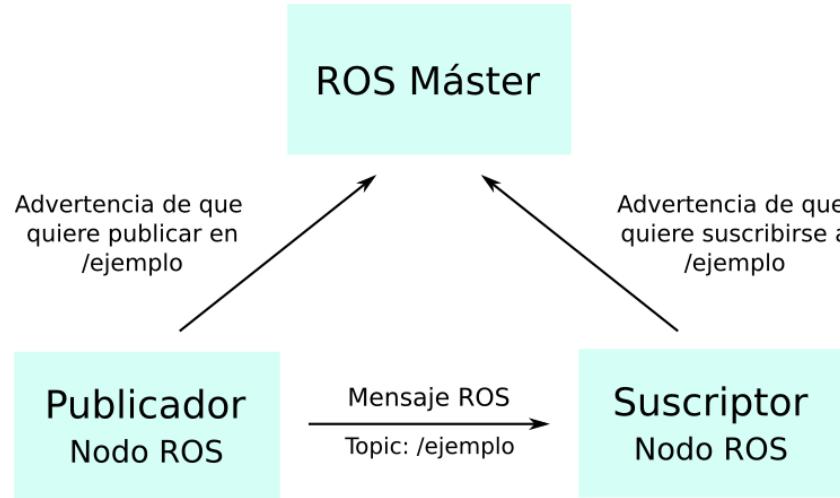


Figura 3.12: Esquema simple de una comunicación publicador-suscriptor en ROS.

o reconocimiento de voz. La versión de ROS usada en este trabajo será ROS Noetic y el resultado final del sistema será de código libre para dicha comunidad de ROS.

Capítulo 4

Sistema de detección de emociones

En este capítulo se describe el sistema de detección de emociones, su desarrollo y su integración en ROS. Además, se muestra su funcionamiento y rendimiento final.

Existen múltiples técnicas utilizadas para realizar detección de emociones, en el artículo [Canal et al., 2022] encontramos una revisión del estado del arte de los últimos años. En este trabajo, se ha escogido la técnica comprendida por los siguientes tres pasos: detección de puntos faciales, extracción de información de esos puntos faciales, clasificación de esa información. En la Figura 4.1 se muestra un esquema general del funcionamiento del sistema, obviando la integración en ROS, la cual se describe en la Sección 4.5. Cada una de las fases de desarrollo mostradas en dicho esquema, corresponden a una sección de este capítulo.

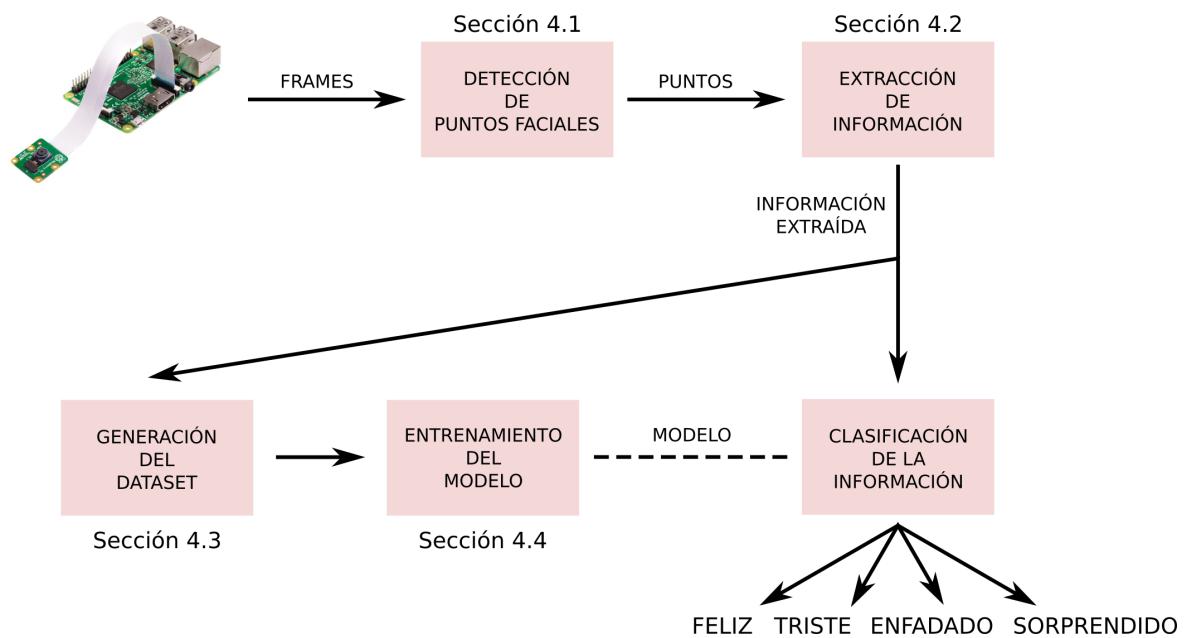


Figura 4.1: Método de detección de emociones.

Finalmente, añadir que el motivo de elegir la técnica descrita anteriormente es

debido a que es un método con bajo coste computacional y además es de los más precisos, algo esencial para aportar robustez a la herramienta robótica final de este trabajo.

4.1. Detección de puntos faciales

El primer paso del método escogido en este trabajo consiste en realizar una detección de puntos faciales característicos. En búsqueda de la máxima optimización, se hizo un estudio comparando dos librerías de extracción de puntos faciales para obtener conclusiones de cual nos ofrecía más rendimiento y precisión. Estas dos librerías son dlib y MediaPipe, ya presentadas en las secciones 3.4 y 3.3, escogidas por ser las más usadas dentro de la investigación de este campo en publicaciones como, por ejemplo, los artículos [Rohith Raj S, Pratiba D, 2020] o [Siam et al., 2022]. Este estudio y sus resultados se describen detalladamente en la Sección 5.1.

El algoritmo que mejores resultados obtuvo fue MediaPipe FaceMesh, obteniendo un rendimiento de 13.28 fps de media y es por lo tanto el usado en este sistema. Nos brinda información de coordenadas 3D de 468 puntos faciales (Figura 4.2).

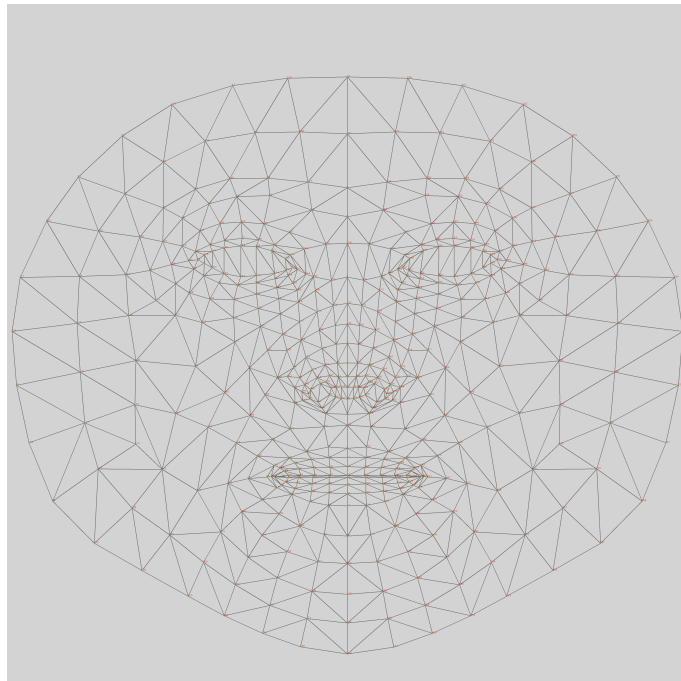


Figura 4.2: Malla facial de MediaPipe de 468 puntos.

4.2. Extracción de información de los puntos faciales

Tras obtener datos en coordenadas de los puntos faciales característicos de un rostro, debemos tratarlos para que a partir de ellos obtengamos información de las posibles emociones que en ese momento se estén llevando a cabo en el rostro estudiado y, finalmente, crear un dataset con toda esa información que nos permita entrenar modelos de la forma más precisa posible.

Retomando la extracción de información, uno de los métodos más usados hasta ahora es la utilización de las distancias entre puntos faciales, tal como se realiza en el artículo [Rohith Raj S, Pratiba D, 2020] (Figura 4.3). Por ejemplo, una emoción de *sorpresa* estará caracterizada por poseer grandes distancias entre el labio inferior y la nariz (boca abierta). Sin embargo, esta técnica es propensa a confundir unas emociones con otras, la información de distancias no es suficiente en determinados casos, ya que a veces se puede mantener constante aunque la expresión facial haya cambiado.

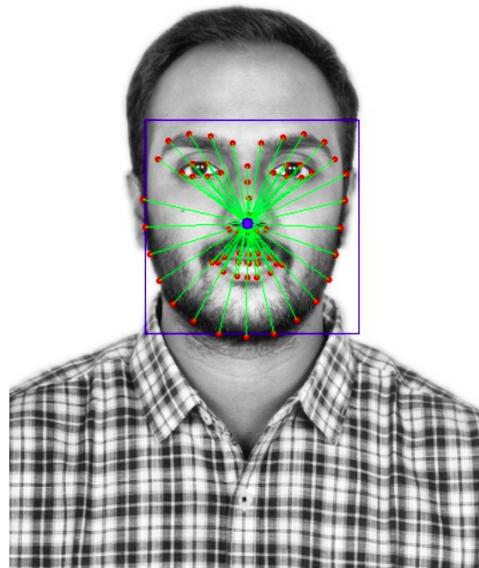


Figura 4.3: Distancias entre puntos faciales del artículo [Rohith Raj S, Pratiba D, 2020].

El método usado para extraer información de los puntos faciales, por lo tanto, no será el relatado anteriormente, sino que se usará un novedoso método propuesto en el artículo [Siam et al., 2022]. Este consiste en construir una *malla emocional* (Figura 4.4), basada en el Sistema de Codificación Facial o FACS (Facial

Action Coding System)[Ekman and Friesen, 1978a][Ekman and Friesen, 1978b], que nos proporcionará información en ángulos de las expresiones faciales.

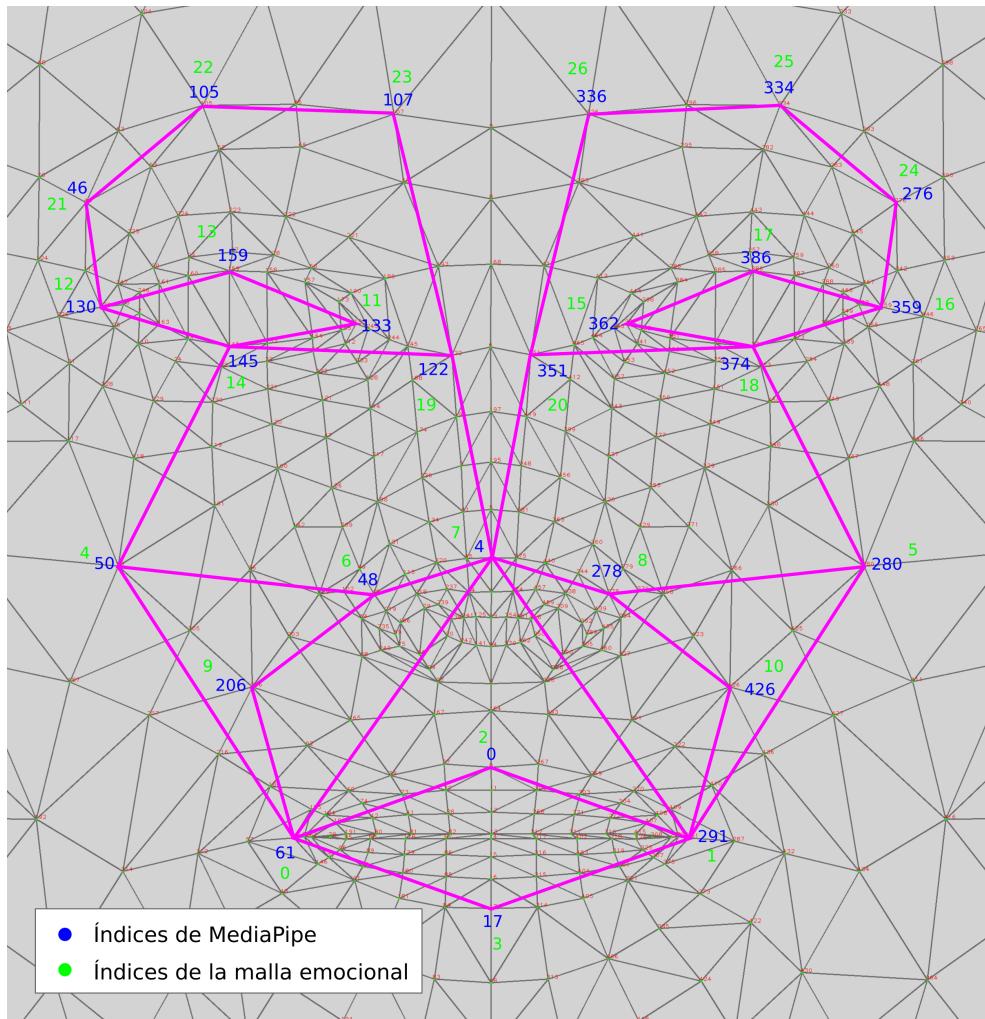


Figura 4.4: *Malla emocional* propuesta en el artículo [Siam et al., 2022].

FACS es un sistema que clasifica movimientos faciales humanos basándose en los cambios producidos en la cara a cargo de los movimientos de los músculos. Estos movimientos son definidos como Unidades de Acción o AUs (Action Units), de los cuales existen hasta 46. En el Cuadro 4.1, están expuestas las AUs más usadas con sus respectivas descripciones.

Además, EMFACS (Emotional Facial Action Coding System) describe emociones simples usando combinaciones de AUs (Cuadro 4.2). Por lo tanto, sabiendo esto, se puede afirmar que el movimiento de cada uno de esos músculos faciales está relacionado con siete emociones simples, y es por eso, que cada ubicación de los puntos de la *malla emocional* (Figura 4.4) ha sido elegida de manera que se vea afectada por una AU y

de esta manera conseguir un mejor reconocimiento de emociones.

AU	Descripciones FACS
1	Interior de las cejas elevado
2	Exterior de las cejas elevado
4	Cejas bajadas
5	Párpado superior elevado
6	Mejillas elevadas
7	Párpados tensos
9	Nariz arrugada
10	Labio superior elevado
12	Comisuras de los labios elevados
15	Comisuras de los labios hacia abajo
16	Labio inferior hacia abajo
17	Barbilla elevada
20	Labios apretados y estirados
22	Labios en forma de <i>o</i>
23	Labios tensos
24	Labios presionados
25	Labios separados
26	Boca abierta (mandíbula caída)
27	Boca abierta

Cuadro 4.1: Lista de las Unidades de Acción más usadas y sus respectivas descripciones FACS.

Emoción	AU
Felicidad	6 + 12
Tristeza	1 + 4 + 15
Sorpresa	1 + 2 + 5 + 26
Miedo	1 + 2 + 4 + 5 + 7 + 20 + 26
Enfado	4 + 5 + 7 + 23
Asco	9 + 15 + 17
Desprecio	12 + 14

Cuadro 4.2: Lista de emociones simples en términos de AUs.

Cada uno de los puntos de la *malla emocional* está unido a otros mediante aristas, formando así una malla cerrada de 27 vértices y 38 aristas. Estas aristas forman ángulos entre ellas, y serán estos los utilizados como información para clasificar emociones. Aprovechando que todas las aristas forman triángulos entre sí (Figura 4.5), para calcular los ángulos deseados se utilizará el teorema del coseno (Ecuación 4.1), el cual usa la longitud de las aristas para realizar los cálculos, esto es la distancia entre los dos

puntos que conforman la arista. Esta distancia es calculada mediante distancia euclídea (Ecuación 4.2). En forma de código de Python, las funciones que realizarán esta tarea se encuentran en el Código 4.1.

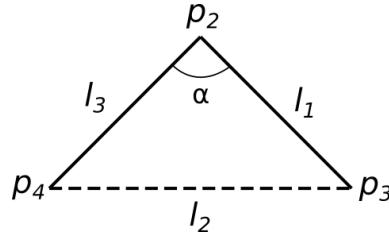


Figura 4.5: Ejemplo de triángulo formado por 2 aristas de la *malla emocional*.

$$\alpha = \arccos \frac{l_1^2 + l_3^2 - l_2^2}{2l_1l_3} \quad (4.1)$$

Ecuación 4.1: Teorema del coseno para calcular el ángulo de la Figura 4.5.

$$l_3 = \sqrt{(p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2} \quad (4.2)$$

Ecuación 4.2: Distancia euclídea entre los puntos p_1 y p_2 de la Figura 4.5.

4.3. Generación del dataset

Siguiendo el esquema de la Figura 4.1, el paso siguiente a la extracción de información de los puntos faciales es la generación de un dataset (base de datos) con dicha información, para posteriormente entrenar un modelo usando técnicas de Machine Learning y que este nos permita clasificar emociones. Poseer un buen conjunto de datos es la fase más importante del trabajo; existen proyectos muy buenos que han llegado a fracasar por no poseer un buen dataset. Por lo tanto, hay que dedicar gran parte del tiempo total a la construcción del mismo. Es importante que contenga una buena cantidad de datos y que estos aporten generalidad.

La idea llevada a cabo en este trabajo ha sido, en primer lugar, localizar un dataset de imágenes que contenga emociones de distintos sujetos. Seguidamente, tratar cada una de esas imágenes con la *malla emocional* y extraer ángulos de cada una de las emociones. Por último, con esos ángulos, generar un nuevo dataset, que ya nos sirve para entrenar nuestros modelos. En dicho dataset final, el número de muestras es

```

1 def distance(self, point1, point2):
2     x0 = point1[0]
3     y0 = point1[1]
4     x1 = point2[0]
5     y1 = point2[1]
6     return math.sqrt((x0 - x1)**2+(y0 - y1)**2)
7
8 def angle(self, point1, point2, point3):
9     side1 = self.distance(point2, point3)
10    side2 = self.distance(point1, point3)
11    side3 = self.distance(point1, point2)
12
13    angle =
14        math.degrees(math.acos((side1**2+side3**2-side2**2)/(2*side1*side3)))
15    return angle

```

Código 4.1: Funciones de Python para realizar el cálculo de ángulos de la *malla emocional*.

el número de fotografías del primer dataset, y el número de características de cada muestra es la cantidad de ángulos usados de la *malla emocional*.

En cuanto al dataset de imágenes, tras realizar una búsqueda por conseguir las imágenes que mejor se adaptasen a nuestro método, se decidió usar *The Extended Cohn-Kanade Dataset (CK+)*[Kanade et al., 2000][Lucey et al., 2010] (Figura 4.6).



Figura 4.6: Ejemplos de imágenes del dataset *The Extended Cohn-Kanade Dataset (CK+)*.

Esta base de datos contiene 593 secuencias de imágenes, en las que se muestra un rostro desde una posición neutral hasta la máxima expresión. De esas 593 secuencias, 327 poseen el último *frame* etiquetado con una emoción entre 1 y 7 (1 = anger, 2 = contempt, 3 = disgust, 4 = fear, 5 = happy, 6 = sadness, 7 = surprise). Estos últimos

frames de esas 327 secuencias, han sido los utilizados en este trabajo.

CK+ ha sido elegido frente a otro tipo de datasets populares como FER¹ debido a la alineación de las caras en las imágenes. Bases de datos como la nombrada anteriormente están enfocadas a ser usadas por CNN (Convolutional Neural Network), y la posición de las caras no influye en el entrenamiento, es más, le aporta más generalidad. Pero para llevar a cabo nuestra técnica, es necesario disponer de caras alineadas para obtener información confiable de los ángulos formados por las expresiones faciales. Recordemos que nosotros lo que deseamos es generar un dataset nuevo a partir de otro de imágenes, no utilizar directamente las imágenes, como si lo haría una CNN con FER.

En cuanto al dataset de ángulos generado a partir del tratamiento de las imágenes de CK+, se han realizado diversos estudios en las secciones 5.2 y 5.3 para encontrar la base de datos que más precisión nos ofrece. Finalmente, el dataset que mejores resultados ha obtenido y, por lo tanto el usado en este trabajo, es el comprendido por cuatro emociones (enfado, felicidad, tristeza, sorpresa) y 21 ángulos (Figura 5.5) por cada una de las muestras. Este se ha generado en un fichero csv en el que cada fila es una muestra, esto es los datos obtenidos de una imagen de CK+, por lo tanto, posee tantas filas como imágenes procesadas. Y cada una de esas filas tiene tantas columnas como ángulos calculados en la imagen y, además, una última columna que indica que tipo de emoción es. El dataset en formato reducido se muestra en el Código 4.2.

	X0	X1	X2	...	X19	X20	y
1	54.288044	37.570711	154.655589	...	44.625203	63.887010	1.0
2	44.670597	35.229102	148.630240	...	47.334403	61.278073	1.0
3	46.613914	36.808837	161.148375	...	57.291823	64.390395	1.0
4	49.404349	47.407905	153.817836	...	49.880184	61.894869	1.0
5	42.510847	43.626048	146.891826	...	46.965439	59.971707	1.0
6
7	23.444336	97.667648	88.384186	...	28.011004	63.905389	7.0
8	24.634940	96.406625	93.413763	...	28.637606	63.551521	7.0
9	22.425106	105.319774	87.727242	...	30.811141	61.646881	7.0
10	15.966920	120.968600	60.790043	...	28.018767	56.442379	7.0
11	19.667632	104.568158	80.332991	...	29.088510	64.107810	7.0

Código 4.2: Dataset usado en el presente trabajo. Las columnas X son las características, la columna y es el tipo de clase.

¹FER: <https://www.kaggle.com/datasets/msambare/fer2013>

4.4. Entrenamiento del modelo

Se realizó el entrenamiento con tres algoritmos de clasificación diferentes y se compararon los resultados de cada uno de ellos. Los algoritmos a utilizar fueron KNN, SVM y MLP. Además, se hizo uso de PCA para reducir el número de características del dataset y de esta manera mejorar la precisión de la detección de emociones.

Cada uno de los algoritmos posee unos parámetros a optimizar en la fase de entrenamiento; nuestra labor es encontrar aquellos valores que proporcionen más precisión al modelo sin llegar a sobreentrenarlo. En caso de KNN se debe encontrar el número de vecinos óptimo (k); para SVM, el parámetro de regulación óptimo (C); y, para MLP, el número de capas ocultas y neuronas óptimas. Los demás parámetros de configuración, como la función de activación en MLP o el algoritmo para calcular distancias en KNN, se deja por defecto tal como los ofrece Scikit-Learn, son los que mejores resultados ofrecen.

A la hora de conocer la precisión obtenida durante el entrenamiento existen varios valores que puntúan el desempeño del modelo:

- *Precision*. De las predicciones que ha realizado, indica que porcentaje son correctas.
- *Recall*. De los datos, indica que porcentaje están bien predichos.
- *F1-Score*. Hace referencia a la media ponderada entre *Precision* y *Recall*.
- *Accuracy*. La suma de los aciertos entre el número total de muestras, esto es el porcentaje de acierto.

Para evitar el sobreentrenamiento y generalizar los modelos, se hace uso de la validación cruzada K-Fold. Este método nos permite usar todos los datos como test y todos los datos como entrenamiento. Esto se consigue gracias a que el algoritmo divide el conjunto de datos en K *pliegues* y uno de ellos es dedicado a datos de test. Entonces, por cada iteración, este conjunto de test se va desplazando a la derecha. En la Figura 4.7, encontramos un ejemplo de 4 *pliegues*.

La clase con menos datos del dataset, tiene una cantidad de 28. Por lo tanto, se usan 4 *pliegues* para KFold, de esta manera aseguramos 7 datos de esa clase en cada pliegue.

	Test	Train	Train	Train
Iteración 1	Test	Train	Train	Train
Iteración 2	Train	Test	Train	Train
Iteración 3	Train	Train	Test	Train
Iteración 4	Train	Train	Train	Test

Figura 4.7: Ejemplo de división de la base de datos en 4 *pliegues* usando KFold.

Al tener una base de datos desequilibrada, esto es, que no tiene el mismo número de muestras para todas las clases, debemos usar KFold Stratified, que nos asegura poseer siempre el mismo porcentaje de muestras de cada clase en todos los pliegues.

KFold Stratified por lo tanto, nos sirve para encontrar los parámetros óptimos de cada algoritmo. Por ejemplo, para KNN probamos cada una de las k (vecinos) posibles en cada una de las iteraciones. El resultado es una matriz con el *Accuracy* para cada una de las k en cada una de las iteraciones. Finalmente, se calcula la media de todas las iteraciones y se escoge la k que más *Accuracy* de media haya obtenido. Y así conseguimos el número de vecinos óptimo. Un ejemplo con 4 valores de vecinos se muestra en la Figura 4.8.

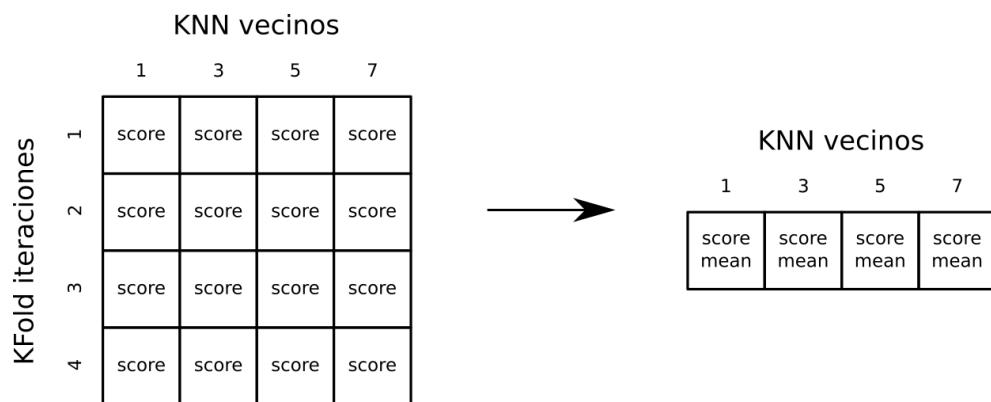


Figura 4.8: Ejemplo de búsqueda del número de vecinos óptimo para KNN usando KFold.

Sin embargo, como se usa PCA para reducir el número de características (componentes), debemos encontrar además el número óptimo de las mismas. Entonces,

en vez de tener una matriz de 2 dimensiones, pasamos a trabajar con una matriz de 3 dimensiones. Se puede ver un ejemplo en la Figura 4.9. Al igual que en el caso anterior, la combinación óptima de vecinos y componentes de PCA, es la que haya obtenido mejor puntuación de media.

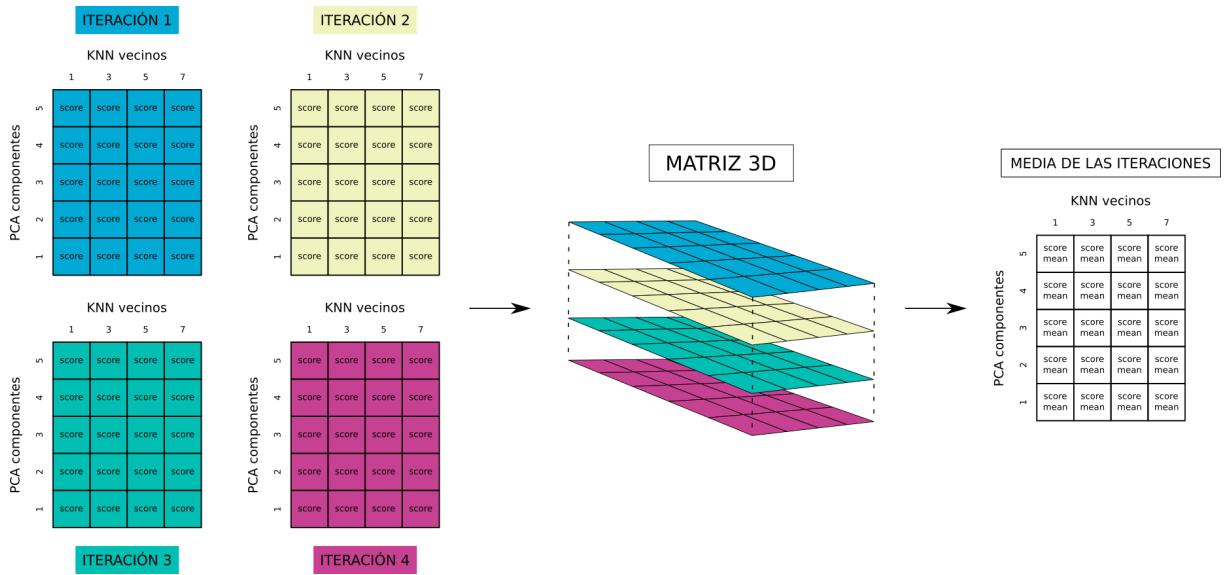


Figura 4.9: Ejemplo de búsqueda del número de vecinos óptimo para KNN y el número de componentes óptimo a reducir por PCA usando KFold.

En este trabajo, el valor óptimo de características a mantener por PCA y los parámetros óptimos para cada clasificador tras realizar su búsqueda usando validación cruzada KFoldStratified de 4 *pliegues*, se encuentran en el Cuadro 4.3. Para hacer la reducción con PCA, se ha probado con un número de componentes entre 2 y 20 incluidos, para KNN se ha probado con valores de k impares entre 1 y 13 incluidos, para SVM con valores de C entre 1 y 999 (saltos de 10 en 10) y para MLP se ha probado con 1 capa oculta de neuronas entre 5 y 24 incluidos, no ha sido necesario añadir más capas.

Clasificador y PCA	Parámetros
KNN y PCA	$k = 7$, $n_components = 11$
SVM y PCA	$C = 21$, $n_components = 11$
MLP y PCA	$hidden_layer_sizes = (17)$, $n_components = 11$

Cuadro 4.3: Parámetros óptimos para cada uno de los clasificadores y PCA.

Finalmente, los resultados obtenidos en el entrenamiento usando los parámetros óptimos, se encuentran en el Cuadro 4.4. Estos resultados de precisión han sido calculados también mediante validación cruzada KFoldStratified de 4 *pliegues*, y se muestra la media de todas las iteraciones.

Clasificador	Accuracy	Precision	Recall	F1-score
KNN	0.95	0.93	0.94	0.92
SVM	0.95	0.93	0.92	0.92
MLP	0.95	0.95	0.93	0.93

Cuadro 4.4: Resultados de precisión de los clasificadores en el entrenamiento.

Como conclusiones finales, observamos que hemos obtenido unos resultados muy buenos para los tres clasificadores, con un porcentaje de acierto medio del 95 %. Además, consultando por separado los resultados de cada una de las clases en cada una de las iteraciones de la validación cruzada, afirmamos también que ninguna emoción de forma individual tiene malos resultados. Por ejemplo, para el entrenamiento con KNN (Cuadro 4.5).

Clase	Precision	Recall	F1-score	Clase	Precision	Recall	F1-score
Enfado	0.88	0.58	0.70	Enfado	1.00	0.82	0.90
Felicidad	0.94	1.00	0.97	Felicidad	1.00	1.00	1.00
Tristeza	0.58	1.00	0.74	Tristeza	0.78	1.00	0.88
Sorpresa	1.00	0.90	0.95	Sorpresa	1.00	1.00	1.00

Clase	Precision	Recall	F1-score	Clase	Precision	Recall	F1-score
Enfado	1.00	1.00	1.00	Enfado	0.90	0.82	0.86
Felicidad	1.00	1.00	1.00	Felicidad	1.00	1.00	1.00
Tristeza	1.00	1.00	1.00	Tristeza	0.75	0.86	0.80
Sorpresa	1.00	1.00	1.00	Sorpresa	1.00	1.00	1.00

Cuadro 4.5: Resultados del entrenamiento con KNN en cada una de las 4 iteraciones de KFoldStratified

Los modelos entrenados son guardados en ficheros con formato *pkl* usando el módulo *pickle* de Python. De esta manera, pueden ser cargados en cualquier otro programa y realizar predicciones con ellos usando el método `predict()`. A este último se le introducen los ángulos del rostro a clasificar y este devuelve la clase predicha.

4.5. Integración del sistema en ROS

El último paso a llevar a cabo en este trabajo es integrar el sistema de detección de emociones en ROS para facilitar su uso en un sistema robótico. Con esto, se pretende crear una herramienta que abstraiga al desarrollador del código del sistema y, que por lo tanto, a través de *topics* de ROS se puedan obtener todos los datos detectados por el sistema de manera fácil.

En esta sección se comenzará explicando cuál ha sido el proceso llevado a cabo para instalar ROS bajo Raspberry Pi OS, se comentará la estructura del paquete y del código ROS desarrollado y, por último, se hará una explicación de su funcionamiento, comentando aspectos de su rendimiento.

4.5.1. Instalación de ROS en Raspberry

El objetivo principal es instalar ROS2, pero no existe ninguna versión de este compatible con 32-bit, y la reciente versión de 64-bit de Raspberry Pi OS lanzada por primera vez en enero de este año, a fecha de hoy, no tiene total compatibilidad con todas las librerías usadas en este trabajo. Igualmente, se ha intentado instalar ROS2 en Raspberry Pi OS de 64-bit, pero no ha resultado satisfactorio porque hay paquetes que no son compatibles con un procesador ARM.

Finalmente, se instaló ROS Noetic, ya que existe una versión del mismo para Debian Buster, y la versión de Raspberry Pi OS usada en este trabajo está basada en Debian Buster. Sin embargo, no se puede hacer la instalación de la forma tradicional a través de *apt*, porque dicha versión de ROS tiene soporte nivel 3². Los pasos a seguir para su instalación son los siguientes³:

1. Añadimos el repositorio oficial de ROS para Debian.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu buster main" >
/etc/apt/sources.list.d/ros-noetic.list'
```

2. Agregamos la clave de ROS para asegurarnos de que instalamos paquetes de ROS autenticados.

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

²Nivel 3: no existen archivos binarios, el usuario debe compilar el código desde la fuente

³<https://varhowto.com/install-ros-noetic-raspberry-pi-4/>

3. Actualizamos los repositorios del sistema.

```
$ sudo apt-get update
```

4. Instalamos las dependencias de compilación.

```
$ sudo apt-get install -y python-rosdep python-rosinstall-generator python-wstool  
python-rosinstall build-essential cmake
```

5. Inicializamos `rosdep`.

```
$ sudo rosdep init  
$ rosdep update
```

6. Creamos un espacio de trabajo de ROS.

```
$ mkdir ~/ros_catkin_ws  
$ cd ~/ros_catkin_ws
```

7. Usamos `rosinstall_generator` para generar una lista de dependencias de Noetic para la variante `ros_comm`, porque los tradicionales `desktop-full` o `desktop` no son compatibles y la placa Raspberry no posee la suficiente memoria como para compilar `rviz`.

```
$ rosinstall_generator ros_comm --rosdistro noetic --deps --wet-only --tar >  
noetic-ros_comm-wet.rosinstall
```

8. Usamos `wstool` para obtener todos los repositorios.

```
$ wstool init src noetic-ros_comm-wet.rosinstall
```

9. Instalamos dependencias usando `rosdep`.

```
$ rosdep install -y --from-paths src --ignore-src --rosdistro noetic -r --os=debian:buster
```

10. Por último, antes de compilar, aumentamos el espacio de `swap` que se usará cuando se agote la memoria física en la Raspberry. Primero desactivamos la memoria `swap`.

```
$ sudo dphys-swapfile swapoff
```

11. Editamos el archivo `/etc/dphys-swapfile`, cambiando `CONF_SWAPSIZE=100` por `CONF_SWAPSIZE=1024`. Y volvemos a configurar y activar la `swap`.

```
$ sudo dphys-swapfile setup
$ sudo dphys-swapfile swapon
```

12. Procedemos a realizar la compilación, todo se instalará en `/opt/ros/noetic`.

```
$ sudo src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release --install-space
/opt/ros/noetic -j1 -DPYTHON_EXECUTABLE=/usr/bin/python3
```

4.5.2. Estructura ROS y código desarrollado

Se ha desarrollado un paquete llamado `emotion_detection_ros`⁴, cuyo esquema general de funcionamiento se puede ver en la Figura 4.10.

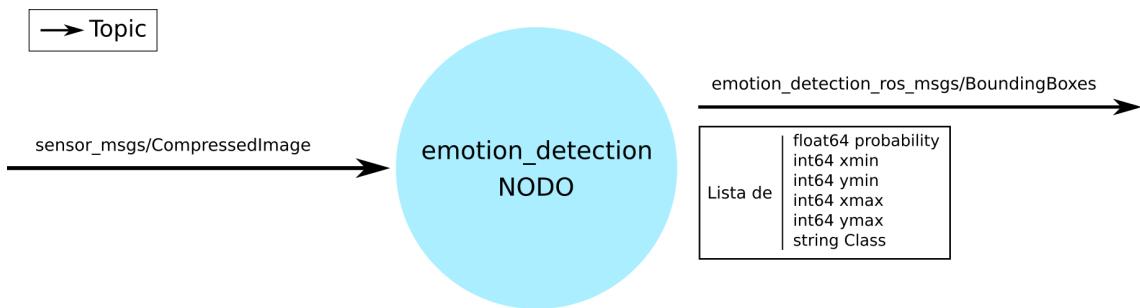


Figura 4.10: Esquema general del paquete de ROS.

El nodo `emotion_detection` recibe los *frames* de una cámara a través de un *topic* que maneja mensajes del tipo `sensor_msgs/CompressedImage`. Este nodo se encargará de hacer todo el procesamiento, y a través de otro *topic* que maneja mensajes del tipo `emotion_detection_ros_msgs/BoundingBoxes`, envía una lista de `emotion_detection_ros_msgs/BoundingBox`. Cada uno de estos últimos mensajes contiene la información de cada emoción detectada: un valor tipo `float64` con la probabilidad de la predicción, cuatro campos de tipo `int64` con las coordenadas del bounding box que rodea la emoción detectada, y un `string` con la emoción detectada.

Parámetros de configuración

El paquete posee un directorio `config` en el que se encuentran dos ficheros de configuración. Estos son de formato `yaml`, y sirven para personalizar parámetros del código sin necesidad de editar el mismo.

⁴https://github.com/jmrtzma/emotion_detection_ros

El fichero `model.yaml` (Código 4.3) nos permite cambiar el algoritmo que usará nuestro sistema para realizar la clasificación de emociones (KNN, SVM o MLP) y elegir el número máximo de caras que deseamos que se detecten. Este último nos permite controlar el rendimiento ofrecido, ya que por cada cara nueva que se esté detectando, los FPS se reducen.

```

1 model:
2
3   algorithm: KNN
4   max_num_faces: 1

```

Código 4.3: Fichero de configuración `model.yaml`.

El fichero `ros.yaml` (Código 4.4) nos permite cambiar los nombres de los *topics* usados, además de algunos parámetros de configuración de los publicadores y suscriptores. Por último, podemos elegir si deseamos que se muestre el visualizador o no, así como el delay entre los *frames* mostrados en este.

```

1 subscribers:
2
3   camera_reading:
4     topic: /raspicam_node/image/compressed
5     queue_size: 1
6
7 publishers:
8
9   bounding_boxes:
10    topic: /emotion_detection_ros/bounding_boxes
11    queue_size: 1
12    latch: False
13
14 image_view:
15
16   enable: True
17   wait_key_delay: 1

```

Código 4.4: Fichero de configuración `ros.yaml`.

Ficheros de código

El nodo principal se encuentra en el fichero `emotion_detection_node.py` del directorio `scripts`. Este posee dos *threads*. El principal del programa se encarga de recibir los *frames* a través de un suscriptor, y el otro *thread* se encarga de realizar el

procesamiento de los frames, esto es: realizar las predicciones, dibujar el bounding box, dibujar el valor de FPS y, además, publicar los datos obtenidos en el *topic* correspondiente. Se ha organizado de esta manera para conseguir un mayor rendimiento ya que, realizando el procesamiento en un *thread* distinto al principal, no se tratarán estrictamente todos los frames, lo que impulsará considerablemente el valor de FPS del sistema.

Las tareas del nodo principal están repartidas en diferentes clases que se encuentran en el directorio `src`:

- **EmotionalMesh**. Encapsula cada *malla emocional* detectada. Recibe como entrada las coordenadas de los puntos faciales detectados por MediaPipe FaceMesh y, con ello, forma la *malla emocional* y calcula todos los ángulos.
- **EmotionalMeshDetection**. Se encarga de detectar los puntos faciales de la cara usando MediaPipe FaceMesh y, con ello, genera una lista de objetos **EmotionalMesh**, esto es, una lista con las *mallas emocionales* de las caras detectadas en el *frame*.
- **Emotion**. Encapsula los datos de cada predicción realizada, esto es, el nombre de la emoción, la probabilidad, la etiqueta que se mostrará por pantalla que contiene el nombre de la emoción y la probabilidad, y las coordenadas del bounding box que rodea esa cara.
- **EmotionPredictor**. Se encarga de realizar las predicciones en los frames usando el modelo entrenado y los objetos **EmotionalMesh**. Genera una lista de objetos **Emotion** con los datos de las predicciones.

4.5.3. Uso y rendimiento

Gracias a que el sistema está integrado en ROS y la comunicación se produce a través de *topics*, se podría usar cualquier cámara que publique *frames* en el *topic* necesario. Sin embargo, se recomienda usar la Raspberry Pi Camera V2.1, ya que ha sido la cámara usada en este trabajo.

Para usar dicha cámara en nuestro entorno ROS, se ha hecho uso del paquete `raspicam_node`⁵. Este se encarga de leer *frames* de la Raspberry Pi Camera de forma

⁵https://github.com/UbiqutyRobotics/raspicam_node

eficiente y publicarlos en el *topic* /raspicam_node/image/compressed.

Con los paquetes `emotion_detection_ros` y `raspicam_node` compilados en el *workspace* de ROS, los pasos a seguir para poner en marcha el sistema son los siguientes:

1. Lanzamos el launcher de la cámara.

```
$ rosrun raspicam_node camerav2_410x308_30fps.launch
```

2. Lanzamos el sistema de detección de emociones

```
$ rosrun emotion_detection_ros emotion_detection_ros.launch
```

Tras lanzar el sistema, se abrirá una ventana gráfica en la que se mostrarán los *frames*, en los cuales aparecerán dibujados los bounding boxes con sus respectivas etiquetas mostrando la predicción (Figura 4.11). Además, en la esquina superior izquierda se expondrá el valor de FPS del sistema.

Otro ejemplo, esta vez detectando dos caras, se encuentra en la Figura 4.12. Y si ejecutamos el comando `rostopic echo /emotion_detection_ros/bounding_boxes`, podemos observar cómo la información detectada se está publicando en dicho *topic* (Figura 4.13).

En cuanto al rendimiento general del sistema, se ha realizado un estudio del valor de FPS arrojado dependiendo del número de caras detectadas. Se han realizado tres pruebas, en cada una de ellas modificando el parámetro de configuración `max_num_faces` del fichero `model.yaml`. De esta manera se ha probado el sistema con hasta tres caras. Los resultados de las tres situaciones se encuentran en los Cuadros 4.6, 4.7 y 4.8, respectivamente.

Caras detectadas	Media de FPS	Valor máximo de FPS	Valor mínimo de FPS
1	13.18	15.13	11.04

Cuadro 4.6: Rendimiento del sistema en ROS con `max_num_faces`: 1.

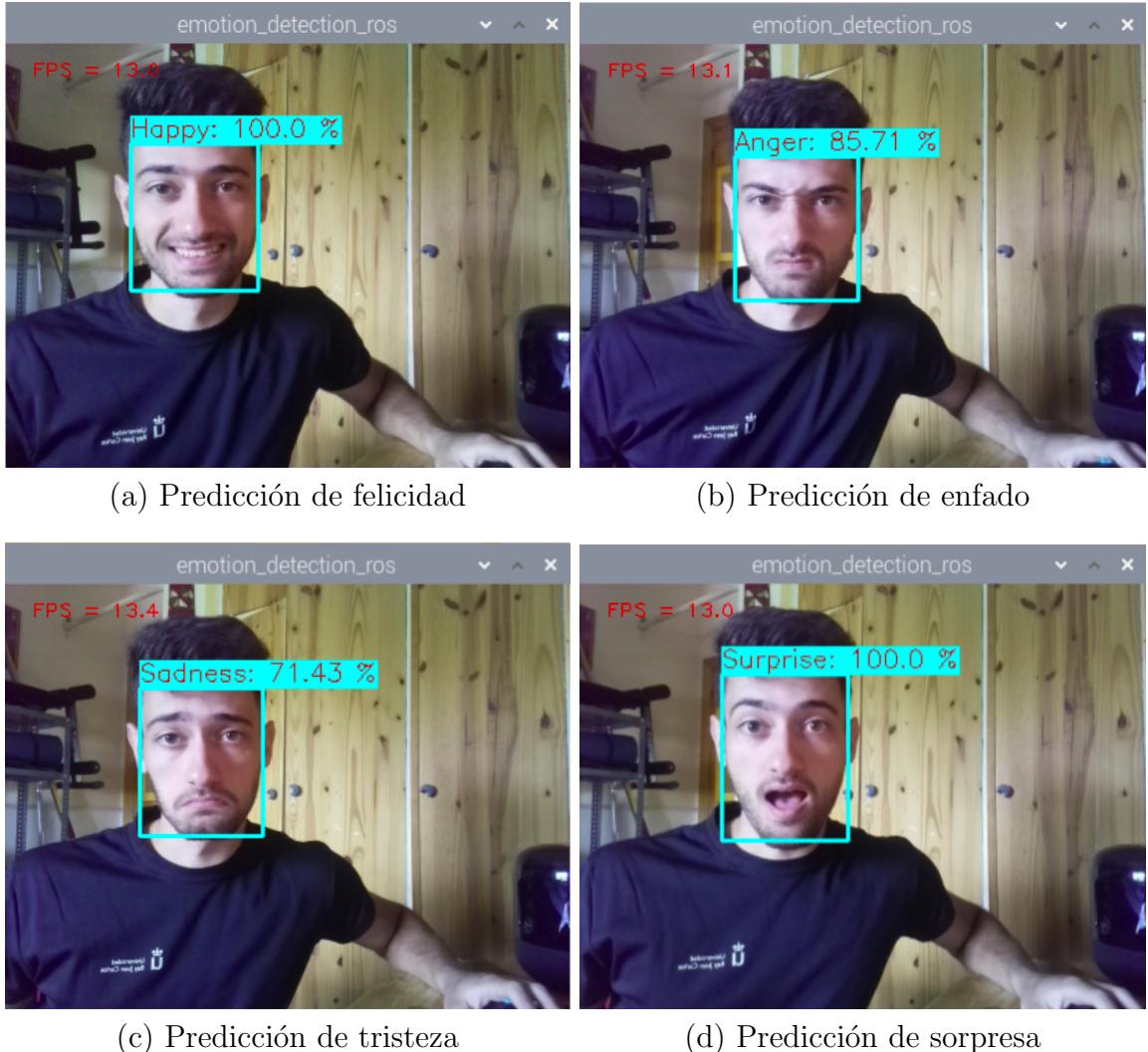


Figura 4.11: Ejemplos de la ventana gráfica del sistema de detección de emociones funcionando en ROS.

Caras detectadas	Media de FPS	Valor máximo de FPS	Valor mínimo de FPS
1	11.11	11.97	9.18
2	7.53	8.35	6.96

Cuadro 4.7: Rendimiento del sistema en ROS con max_num_faces: 2.

Caras detectadas	Media de FPS	Valor máximo de FPS	Valor mínimo de FPS
1	11.05	11.72	9.41
2	6.77	7.61	6.15
3	5.37	6.93	5.02

Cuadro 4.8: Rendimiento del sistema en ROS con max_num_faces: 3.

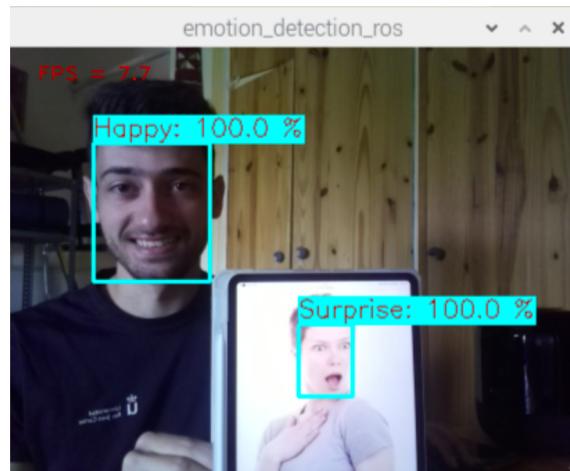


Figura 4.12: Ejemplo del sistema en ROS detectando las emociones de dos caras.

```
---  
bounding_boxes:  
-  
  probability: 100.0  
  xmin: 198  
  ymin: 199  
  xmax: 234  
  ymax: 249  
  Class: "Surprise"  
-  
  probability: 100.0  
  xmin: 42  
  ymin: 74  
  xmax: 131  
  ymax: 177  
  Class: "Happy"  
---
```

Figura 4.13: Ejemplo de información publicada en el *topic* /emotion_detection_ros/bounding_boxes.

Capítulo 5

Estudios de optimización

En este capítulo se describen los estudios realizados durante el desarrollo del sistema con el objetivo de optimizar las diferentes partes del mismo.

5.1. Búsqueda del detector de puntos faciales óptimo

Se hizo un estudio comparando dos librerías de extracción de puntos faciales para obtener conclusiones de cual nos ofrecía más rendimiento y precisión. Estas dos librerías son dlib y MediaPipe, ya presentadas en las secciones 3.4 y 3.3.

Las pruebas se realizaron en la Raspberry Pi 4 Model B bajo Raspberry Pi OS, usando la Raspberry Pi Camera como dispositivo para capturar vídeo. Se puede encontrar más información sobre las versiones usadas en el Capítulo 3. La resolución utilizada para las pruebas fue de 640x480 píxeles y se parte de una media en crudo de 20 fps, esto es sólo mostrando los *frames* (capturados con la librería *picamera*) por pantalla sin ningún tipo de procesamiento.

Se comenzó estudiando el rendimiento ofrecido en FPS y posteriormente se hizo un estudio de los fallos producidos por los algoritmos en distintas situaciones. Para realizar las pruebas de rendimiento se guardó el valor de fps calculado para cada *frame* durante 30 segundos y posteriormente se calculó la media de dichos *frames* guardados. Para realizar las pruebas de fallos, se tuvieron en cuenta dos situaciones: falsos positivos y no detección de ningún rostro. Se dio por hecho que siempre hay una cara en cada *frame*, por lo tanto, un *falso positivo* significa que hay más de una cara y *no detección* significa que no hay ninguna. Sin embargo, en MediaPipe se pueden evitar los falsos positivos indicando al algoritmo que sólo detecte una cara, por lo tanto en su caso no se tuvieron en cuenta. Se evaluó cada *frame* durante 30 segundos siguiendo esos criterios, y se hizo un recuento de los fallos.

Prueba 1 de rendimiento para dlib

Tal como se ha explicado en la Sección 3.4, dlib divide la extracción de características faciales en dos pasos: detección del rostro y extracción de características. Para el primero de los pasos, se ofrecen dos opciones incorporadas en sus librerías (HOG y Linear SVM o CNN). En la primera prueba se hizo uso del detector de caras basado en HOG y Linear SVM, ya que es el más eficiente computacionalmente hablando, sumado al detector de características también incorporado en las librerías de dlib. Esto ofreció un rendimiento de 0.83 fps de media.

Prueba 2 de rendimiento para dlib

Tras comprobar el bajo rendimiento obtenido en la primera prueba, se procedió a sustituir el detector de caras usado anteriormente por el detector de caras que trae incorporado OpenCV, este es descrito en el artículo [Viola and Jones, 2001]. Se hizo uso de este algoritmo porque actualmente es uno de los más rápidos realizando detección facial. Este cambio impulsó el rendimiento de dlib un 656 %, obteniendo una media de 6.28 fps.

Prueba 3 de rendimiento para dlib

Con el objetivo de mejorar aún más el rendimiento, se procedió a dividir el procesamiento de los *frames* de la lectura de los mismos, haciendo uso de threads. De esta manera se aprovecharon los 4 núcleos del procesador ARM de la Raspberry Pi 4 Model B. Para esto, lo que se hizo fue lanzar un *thread* que se encargase constantemente de realizar la lectura de los *frames* usando la librería *picamera*, y por otro lado, el thread principal del programa se encargaba de realizar el procesamiento de dlib y el detector de caras de OpenCV. Con esto impulsamos el rendimiento un 30 % más, logrando así una media de 8.21 fps. Se puede ver una comparativa de las tres pruebas de rendimiento en la Figura 5.1.

Prueba de fallos para dlib

Para comprobar la robustez del algoritmo se evaluó a este en las siguientes condiciones (Figura 5.2):

- Buenas condiciones lumínicas: 81 fallos.
- Malas condiciones lumínicas: 126 fallos.
- Rostros parcialmente cubiertos: 188 fallos.

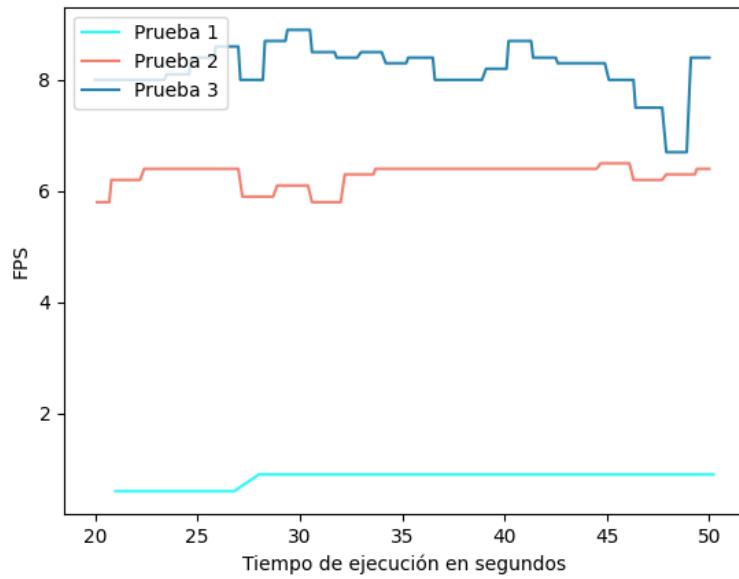


Figura 5.1: Comparativa de las tres pruebas de rendimiento de dlib.

- Rostros girados: 183 fallos.

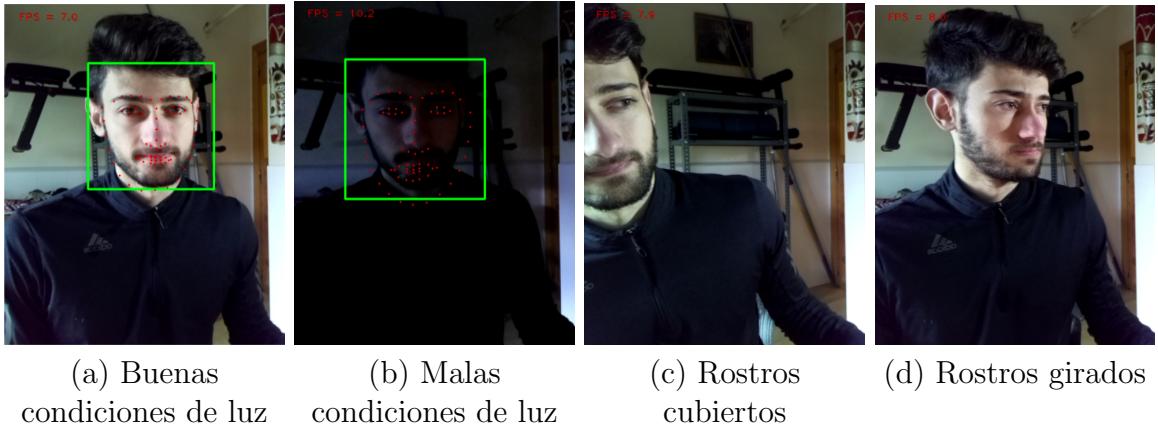


Figura 5.2: Condiciones en las que se evalúan los fallos de dlib.

Prueba 1 de rendimiento para MediaPipe

En esta primera prueba se utilizó el algoritmo de la forma general recomendada en el tutorial oficial de MediaPipe. De esta manera, obtuvimos una media de 5.91 fps. Este dato es inferior al resultado final de 8.21 fps de dlib.

Prueba 2 de rendimiento para MediaPipe

Llegados a este punto, para mejorar el resultado anterior, dividimos el procesamiento total en *threads* al igual que hicimos con dlib. Dedicamos un *thread*

a la lectura de *frames* y el *thread* principal al procesamiento de MediaPipe. De esta manera, aumentamos el rendimiento un 32 %, obteniendo una media de 7.80fps, todavía sin superar el mejor desempeño de dlib.

Prueba 3 de rendimiento para MediaPipe

Hasta ahora, estábamos mostrando por pantalla la malla facial detectada, pero esto no será necesario cuando implementemos nuestro sistema de detección de emociones. Por ello, se analizó el rendimiento sin dibujar dicha malla en cada *frame*. De esta manera, el rendimiento subió un 70 % más, llegando así ya a una media de 13.28 fps. Se puede ver una comparativa de las tres pruebas de rendimiento en la Figura 5.3

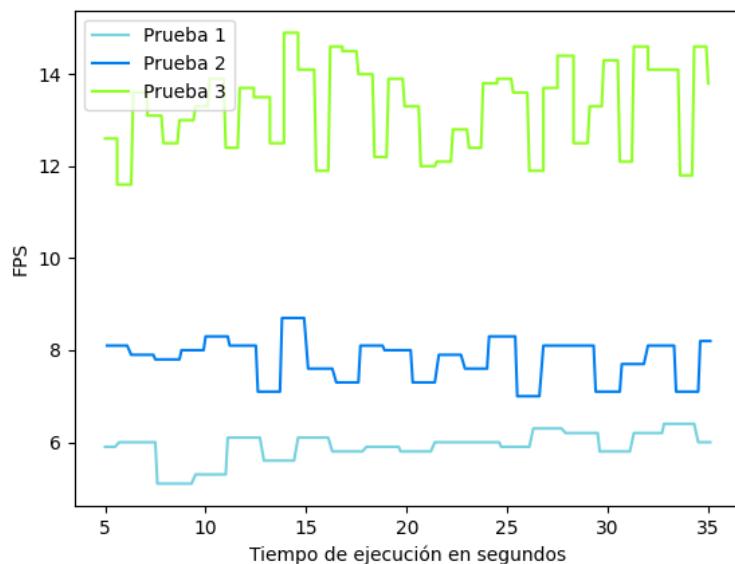


Figura 5.3: Comparativa de las tres pruebas de rendimiento de MediaPipe.

Prueba de fallos para MediaPipe

Para comprobar la robustez del algoritmo se evaluó a este en las siguientes condiciones (Figura 5.4):

- Buenas condiciones lumínicas: 0 fallos.
- Malas condiciones lumínicas: 0 fallos.
- Rostros parcialmente cubiertos: 103 fallos.
- Rostros girados: 0 fallos.

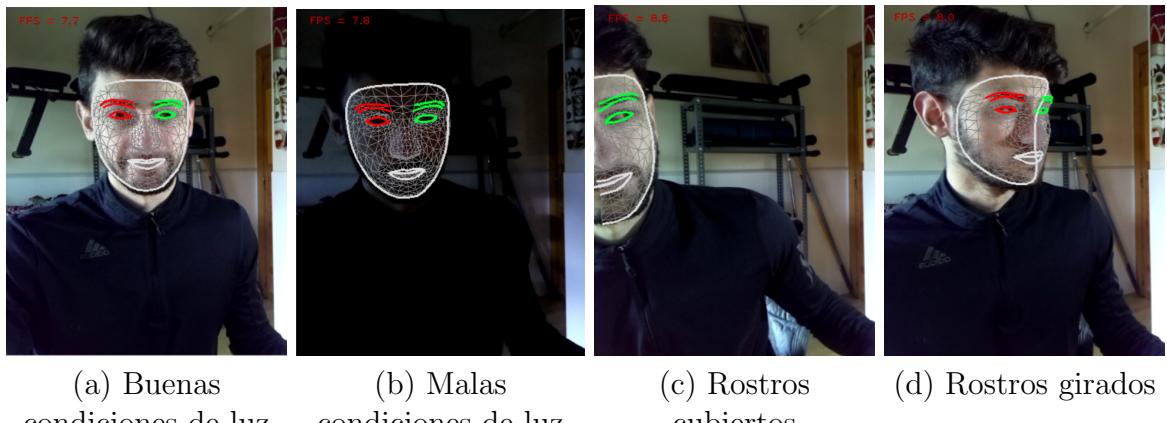


Figura 5.4: Condiciones en las que se evalúan los fallos de MediaPipe.

Llegados a este punto, podemos concluir afirmando que el algoritmo que mejor rendimiento y precisión nos ofrece es el de MediaPipe. Este ha obtenido un rendimiento de 13.28 fps de media, frente a los 8.21 fps de dlib y OpenCV. Además, este último ha obtenido un total de 578 fallos, frente a los 103 fallos de MediaPipe.

5.2. Cantidad de ángulos a utilizar en el dataset

El número de ángulos a utilizar de la *malla emocional*, hace referencia a la cantidad de características que posee nuestro dataset. Es por ello que se debe estudiar cuál es la cantidad óptima de los mismos que mejores resultados nos ofrece a la hora de entrenar los modelos.

En primer lugar se realizó un estudio para comprobar qué ángulos de toda la malla son más influyentes en las emociones y, posteriormente, se hizo un estudio de simetría con dichos ángulos, para así comprobar si las emociones se pueden considerar simétricas en ambas mitades de un rostro y entonces únicamente utilizar los ángulos de una sola mitad.

5.2.1. Estudio de los ángulos más influyentes en cada emoción

Tal como se ha comentado anteriormente, realizamos un estudio para escoger los ángulos más influyentes en cada emoción, esto es, los ángulos que más varían cada vez que se producen dichas expresiones faciales. Para realizar esta prueba, partimos de todos los ángulos de la parte derecha del rostro (Figura 5.5), y se calculó la variación que existe en cada uno de ellos y para cada una de las emociones de CK+, desde una

posición neutral de la cara hasta la máxima expresión.

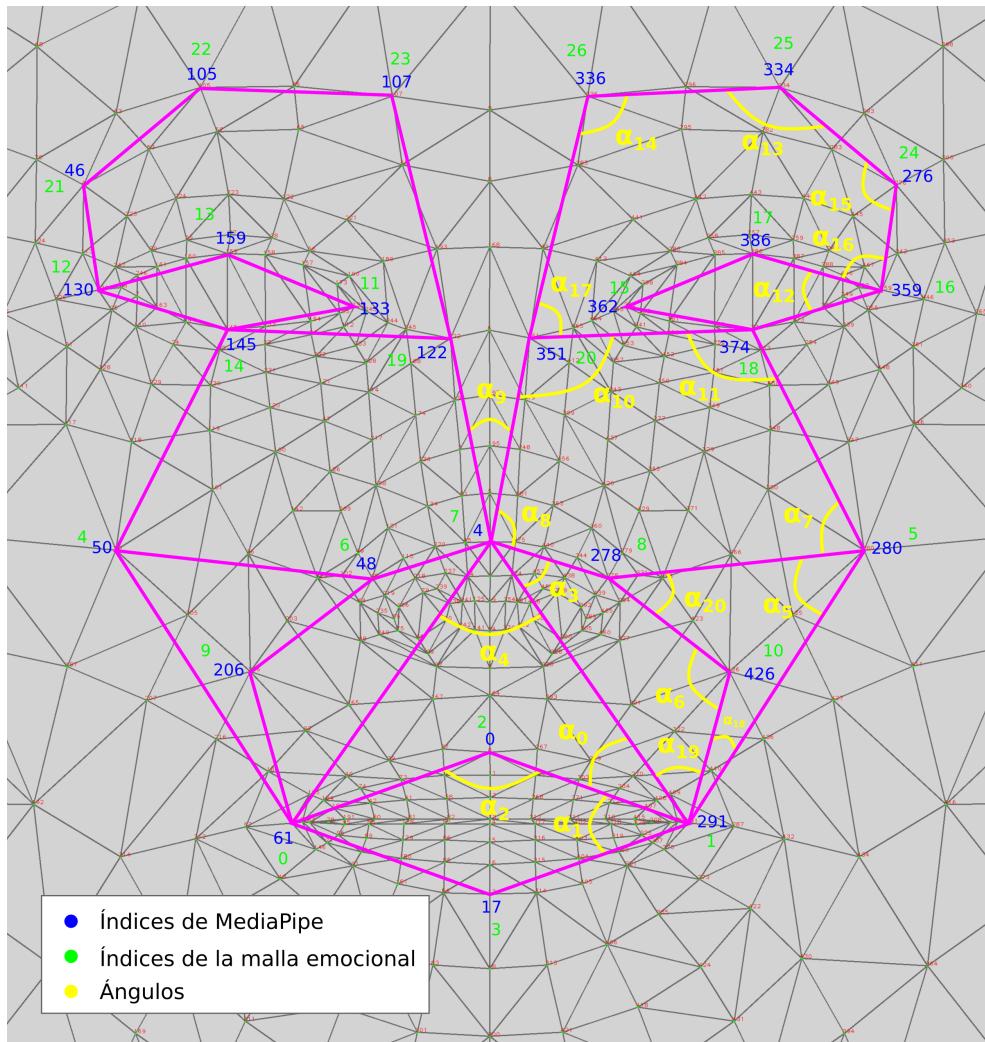


Figura 5.5: Todos los ángulos de la mitad derecha
de la *malla emocional*.

El método a seguir que nos proporcionó esa variación, fue calcular la diferencia del ángulo en posición neutral respecto a la posición de la máxima expresión, para ello utilizamos el Código 5.1. Este, además de calcular la diferencia, evitó que apareciesen resultados de otros cuadrantes o negativos. Se realizó este cálculo para cada una de las imágenes de CK+ y los resultados fueron las medias calculadas. El resultado del estudio se encuentra en la Figura 5.7.

Como conclusiones de este estudio, sacamos en claro que los cinco ángulos más influyentes en cada emoción son los mostrados en el Cuadro 5.1 y, que por lo tanto, los ángulos que más varían en total son: 2, 12, 1, 16, 15, 4, 6, 14, 18, 8, 0 y 19.

```

1 def angle_difference(alpha, beta):
2     phi = abs(beta-alpha) %360
3     if phi > 180:
4         return (360 - phi)
5     return phi

```

Código 5.1: Diferencia entre dos ángulos.

Emoción	Ángulos influyentes
Felicidad	4, 2, 18, 12, 8
Tristeza	2, 1, 12, 16, 14
Sorpresa	1, 2, 0, 4, 19
Miedo	2, 4, 6, 1, 14
Enfado	2, 12, 1, 16, 15
Asco	12, 16, 2, 15, 1
Desprecio	2, 1, 4, 0, 19

Cuadro 5.1: Lista de los 5 ángulos más influyentes por cada emoción.

5.2.2. Estudio de simetría de las emociones

Con el afán de descubrir si es posible entrenar a nuestro modelo con sólo ángulos de una mitad del rostro se realizó un estudio de simetría, en el que comprobamos si las emociones se pueden considerar simétricas para ambos lados de la cara. Para ello, partimos de los ángulos obtenidos en el estudio anterior (los más influyentes) y generamos un nuevo mapa que abarca las dos mitades del rostro (Figura 5.6).

A la hora de estudiar la simetría calculamos la variación que existe entre los ángulos del lado izquierdo y del lado derecho para cada una de las emociones y cada una de las imágenes del dataset CK+. La forma de llevar a cabo este cálculo ha sido la misma que en el estudio anterior (Código 5.1). Los resultados son la media de lo obtenido para todas las imágenes (Figura 5.8).

Observando los resultados, podemos considerar que las emociones son simétricas, ya que la mayor variación de media entre el lado izquierdo y derecho ha sido de 5 grados, siendo esta una variación mínima. Por lo tanto, conociendo la información de los dos estudios, descartamos todos los ángulos de una mitad de la cara y generamos los dos siguientes datasets:

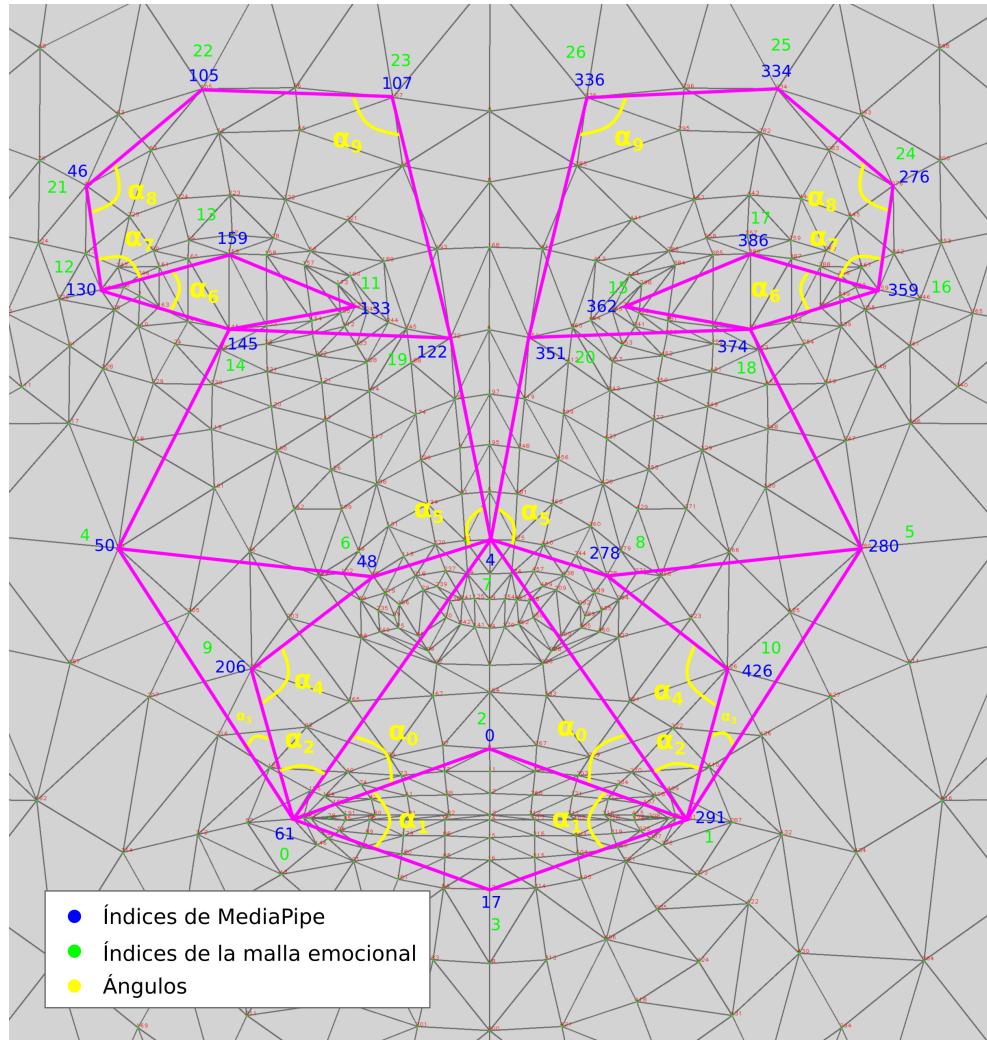


Figura 5.6: Mapa de ángulos para realizar el estudio de simetría.

- Base de datos con sólo los ángulos más influyentes de la parte derecha de la *malla emocional*.
- Base de datos con todos los ángulos de la parte derecha de la *malla emocional*.

En las siguientes secciones se analizó cual de ellos finalmente ofreció mejores resultados en el entrenamiento.

5.3. Búsqueda del dataset óptimo para el entrenamiento

Se realizó un estudio en búsqueda del dataset que mejores resultados nos ofrecía a la hora de realizar el entrenamiento. Para ello se probó con los dos datasets generados en la sección anterior a raíz de los estudios de influencia (Sección 5.7) y de simetría

(Sección 5.8). De aquí en adelante se utilizan los siguientes nombres para identificar a cada uno de los datasets:

- **dataset1**: base de datos con sólo los ángulos más influyentes de la parte derecha de la *malla emocional*.
- **dataset2**: base de datos con todos los ángulos de la parte derecha de la *malla emocional*.

Ambos han sido entrenados por los clasificadores KNN, SVM y MLP y, además, con el *dataset2* se ha utilizado PCA para reducir el número de características. Se desea comprobar si ofrece más rendimiento el *dataset1* habiendo reducido los ángulos de forma manual (sólo contiene los más influyentes de cada emoción) o el *dataset2* reduciéndolos usando PCA.

Entrenamiento usando el *dataset1*

Los parámetros óptimos de los tres clasificadores, tras realizar su búsqueda usando validación cruzada KFoldStratified de 4 *pliegues*, se encuentran en el Cuadro 5.2. Para KNN se ha probado con valores de k impares entre 1 y 13 incluidos, para SVM con valores de C entre 1 y 999 (saltos de 10 en 10) y para MLP se ha probado con 1 capa oculta de neuronas entre 5 y 24 incluidos, no ha sido necesario añadir más capas.

Clasificador	Parámetros
KNN	k = 11
SVM	C = 11
MLP	hidden_layer_sizes = (19)

Cuadro 5.2: Parámetros óptimos para cada uno de los clasificadores entrenando con el *dataset1*.

Los resultados obtenidos en el entrenamiento para el *dataset1* usando los parámetros óptimos, se encuentran en el Cuadro 5.3. Estos han sido calculados también mediante validación cruzada KFoldStratified de 4 *pliegues*, y se muestra la media de todas las iteraciones.

Clasificador	Accuracy	Precision	Recall	F1-score
KNN	0.82	0.76	0.74	0.74
SVM	0.84	0.79	0.78	0.77
MLP	0.82	0.78	0.78	0.77

Cuadro 5.3: Resultados de precisión de los clasificadores usando el *dataset1*.

Entrenamiento usando el *dataset2*

El valor óptimo de características a mantener por PCA y los parámetros óptimos para cada clasificador tras realizar su búsqueda usando validación cruzada KFoldStratified de 4 *pliegues*, se encuentran en el Cuadro 5.4. Para hacer la reducción con PCA se ha probado con un número de componentes entre 2 y 20 incluidos, para KNN se ha probado con valores de k impares entre 1 y 13 incluidos, para SVM con valores de C entre 1 y 999 (saltos de 10 en 10) y para MLP se ha probado con 1 capa oculta de neuronas entre 5 y 24 incluidos, no ha sido necesario añadir más capas.

Clasificador y PCA	Parámetros
KNN y PCA	k = 11, n_components = 17
SVM y PCA	C = 11, n_components = 7
MLP y PCA	hidden_layer_sizes = (12), n_components = 11

Cuadro 5.4: Parámetros óptimos para cada uno de los clasificadores entrenando con el *dataset2*.

Los resultados obtenidos en el entrenamiento para el *dataset2* usando los parámetros óptimos, se encuentran en el Cuadro 5.5. Estos resultados de precisión han sido calculados también mediante validación cruzada KFoldStratified de 4 *pliegues*, y se muestra la media de todas las iteraciones.

Clasificador	Accuracy	Precision	Recall	F1-score
KNN	0.83	0.79	0.75	0.75
SVM	0.84	0.80	0.77	0.78
MLP	0.84	0.81	0.80	0.80

Cuadro 5.5: Resultados de precisión de los clasificadores usando el *dataset2*.

Conclusiones

Los resultados obtenidos para cada uno de los datasets son bastante similares, aunque un poco superiores para el *dataset2*, sobre todo si observamos *Precision*, *Recall* y *F1-score*. En cuanto a los resultados obtenidos por cada clasificador, también son muy similares y sería muy complicado decantarse por uno sólo.

Sin embargo, en general no son resultados demasiado buenos y la herramienta robótica construida a partir de estos modelos no sería muy robusta. Sobre todo si observamos los resultados de cada una de las clases por separado, por ejemplo los arrojados por SVM usando el *dataset2* (Cuadro 5.6). En dichos resultados, podemos observar que clases como —por ejemplo— la de *Desprecio* o la de *Enfado* han obtenido puntuaciones malas, incluso por debajo del 50 % en algún caso. Por lo tanto, esto nos lleva a pensar que nuestro sistema no va a dar buen rendimiento a la hora de detectar todas las emociones, aunque de media si que tenga una puntuación aceptable.

Clase	Precision	Recall	F1-score	Clase	Precision	Recall	F1-score
Enfado	0.67	0.50	0.57	Enfado	0.71	0.45	0.56
Desprecio	0.67	0.50	0.57	Desprecio	0.57	0.80	0.67
Asco	0.76	0.87	0.81	Asco	0.76	0.93	0.84
Miedo	0.67	1.00	0.80	Miedo	1.00	0.71	0.83
Felicidad	0.94	0.94	0.94	Felicidad	0.85	1.00	0.92
Tristeza	0.83	0.71	0.77	Tristeza	0.83	0.71	0.77
Sorpresa	0.95	0.95	0.95	Sorpresa	1.00	0.95	0.98

Clase	Precision	Recall	F1-score	Clase	Precision	Recall	F1-score
Enfado	0.75	0.82	0.78	Enfado	0.67	0.55	0.60
Desprecio	0.67	0.80	0.73	Desprecio	0.50	0.25	0.33
Asco	0.82	0.93	0.87	Asco	0.67	0.80	0.73
Miedo	0.80	0.67	0.73	Miedo	1.00	0.67	0.80
Felicidad	0.94	0.94	0.94	Felicidad	1.00	1.00	1.00
Tristeza	1.00	0.57	0.73	Tristeza	0.50	0.71	0.59
Sorpresa	0.95	0.95	0.95	Sorpresa	1.00	1.00	1.00

Cuadro 5.6: Resultados del entrenamiento con SVM en cada una de las 4 iteraciones de KFoldStratified usando el *dataset2*.

Esta mala clasificación de algunas emociones se debe a la gran similitud que existe entre varias. Esto provoca que geométricamente sean prácticamente indiferenciables, y por lo tanto, es complicado que los algoritmos sepan clasificarlas correctamente. Si observamos el estudio de ángulos influyentes en cada emoción (Figura 5.7) (Cuadro 5.1), podemos ver que —por ejemplo— las emociones *sorpresa* y *desprecio* poseen exactamente los mismos 5 ángulos influyentes, o las emociones *enfado* y *asco* que

también poseen los mismos 5. Esto dificulta muchísimo la tarea de clasificación.

Por lo tanto, se propuso eliminar las emociones que estén interceptando con otras y, de esta manera, aumentar la robustez de nuestra futura herramienta consiguiendo mayor precisión en nuestro modelo. Las emociones que se eligieron son las siguientes: *felicidad, tristeza, sorpresa y enfado*. Son las 4 emociones simples más generales y, además, se diferencian geométricamente unas de otras. Sabiendo esto, el dataset ganador y, por lo tanto el que es usado para entrenar nuestro sistema final, es el *dataset2* con las emociones de *felicidad, tristeza, sorpresa y enfado*.

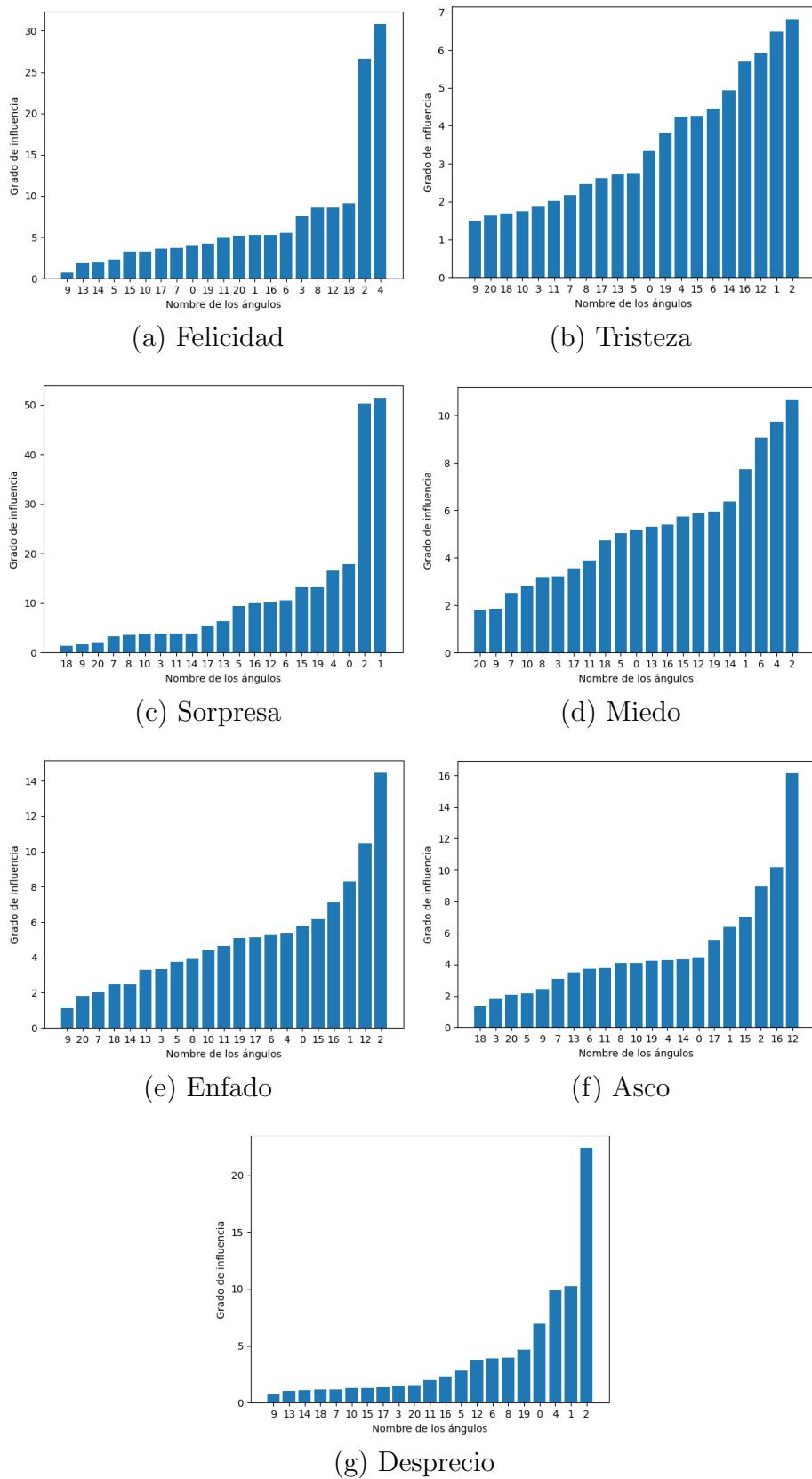


Figura 5.7: Estudio de los ángulos más influyentes para cada emoción.

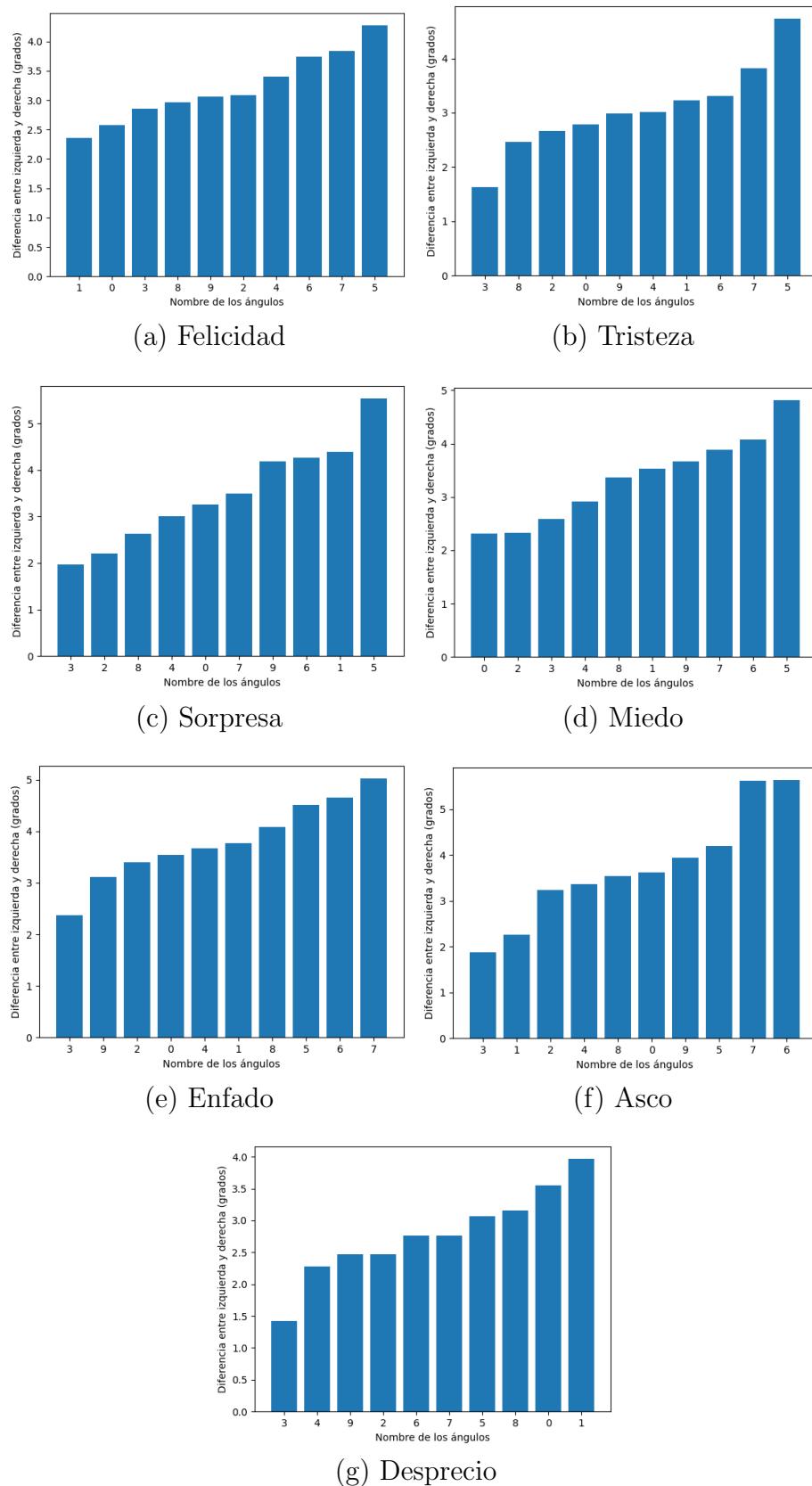


Figura 5.8: Estudio de simetría para cada una de las emociones.

Capítulo 6

Conclusiones

En este último capítulo se resumirán los objetivos conseguidos y, con ello, las conclusiones finales a las que se ha llegado en este trabajo. Además, se comentarán cuales han sido las competencias y conocimientos adquiridos durante el desarrollo, así como mencionar las líneas futuras.

6.1. Consecución de objetivos

Se destaca que se ha conseguido cumplir el objetivo principal de este trabajo: desarrollar una herramienta de reconocimiento de emociones que sea capaz de funcionar en un sistema robótico de bajo coste bajo el entorno ROS. Como se ha mostrado en la Sección 4.5, el sistema de detección de emociones está funcionando bajo ROS Noetic a un rendimiento medio de 13.18 fps, esto es, en tiempo real.

Además, en el Capítulo 2 se dividió el objetivo general en una serie de subobjetivos y todos ellos se han cumplido:

- *Estudiar el estado del arte y elegir la técnica más óptima.* Como se ha comentado en el Capítulo 4, se ha escogido la técnica comprendida por los siguientes tres pasos: detección de puntos faciales, extracción de información de esos puntos faciales, clasificación de dicha información. Y se ha cumplido el objetivo de ser una técnica liviana y precisa.
- *Optimizar y adaptar la técnica escogida en nuestra plataforma.* Para ello, se ha realizado un estudio en búsqueda del algoritmo de detección de puntos faciales más rápido y preciso capaz de funcionar en nuestra plataforma (Sección 5.1). Además, se ha investigado profundamente el estado del arte en cuanto a técnicas óptimas para obtener información confiable de dichos puntos faciales que, posteriormente, nos permitiera generar un buen dataset, saliendo victoriosa la construcción de una *malla emocional* [Siam et al., 2022] que proporciona información angular de las emociones (Sección 4.2).

- *Generar un dataset de valor.* Se buscó un dataset de imágenes que sirviera de base para posteriormente generar el nuestro con información angular obtenida de dichas imágenes (Sección 4.3). El dataset escogido ha sido The Extended Cohn-Kanade Dataset (CK+) [Kanade et al., 2000][Lucey et al., 2010]. Se realizaron diversos estudios que nos han ayudado a decidir qué información angular era la más óptima a la hora de generar la base de datos (Sección 5.2), y, posteriormente, más estudios que nos han indicado cual de los datasets generados nos ofrecía más precisión en el entrenamiento de nuestro modelo (Sección 5.3).
- *Realizar el entrenamiento del modelo.* Se han utilizado tres algoritmos de clasificación para entrenar nuestro modelo (KNN, SVM, MLP). A la hora de buscar los parámetros óptimos de cada uno de los algoritmos y también para testear la precisión en su entrenamiento, se ha utilizado la técnica de validación cruzada KFold Stratified. Finalmente, los tres modelos entrenados poseen un porcentaje de acierto del 95 % y todo el proceso de su entrenamiento se relata en la Sección 4.4.
- *Integrar el sistema en ROS.* Tal como se muestra en la Sección 4.5, se ha conseguido instalar ROS Noetic bajo Raspberry Pi OS y se ha creado un paquete llamado `emotion_detection_ros` que porta la herramienta y la integra totalmente dentro del ecosistema ROS. En la Sección 4.5.3 se muestra el funcionamiento y rendimiento final del sistema, 13.18 fps de media.

6.2. Competencias adquiridas

Durante el desarrollo de este trabajo se han adquirido diferentes competencias y conocimientos, los cuales se listan a continuación:

- Soltura a la hora de buscar y leer artículos de investigación.
- Profundización en el uso de un sistema de bajo coste Raspberry Pi y sus periféricos, así como su sistema operativo.
- Tomar decisiones tras realizar estudios que las validen.
- Experiencia usando librerías de tratamiento de datos y Machine Learning en Python (NumPy, pandas, Scikit-Learn).
- Aprendizaje de técnicas avanzadas de entrenamiento (validación cruzada KFold).
- Ampliación de conocimientos en ROS.

6.3. Valoración final y líneas futuras

Se ha desarrollado una herramienta que ofrece un servicio de detección de emociones a los desarrolladores de robótica de bajo coste, permitiéndoles avanzar en la investigación de la Interacción Humano Robot. Es por ello que se plantean las siguientes líneas futuras que se podrían seguir a raíz de este trabajo:

- Implementar aplicaciones de HRI que se ayuden de esta herramienta.
- Portar el sistema a otras plataformas para dar servicio en otros campos de investigación. Por ejemplo, incorporarlo en cines y así estudiar las reacciones del público a las distintas escenas de una película. Gracias a ser una herramienta que necesita pocos recursos, no es complicado incorporarla en otros sistemas.
- Sería interesante añadir nuevas funcionalidades a la herramienta, como por ejemplo, que sea capaz de detectar el género o predecir la edad del sujeto a evaluar.

Bibliografía

- [Canal et al., 2022] Canal, F. Z., Müller, T. R., Matias, J. C., Scotton, G. G., de Sa Junior, A. R., Pozzebon, E., and Sobieranski, A. C. (2022). A survey on facial emotion recognition techniques: A state-of-the-art literature review. *Information Sciences*, 582:593–617.
- [Dautenhahn, 2007] Dautenhahn, K. (2007). Methodology themes of human-robot interaction: A growing research field. *International Journal of Advanced Robotic Systems*, 4.
- [Ekman and Friesen, 1978a] Ekman, P. and Friesen, W. V. (1978a). Facial action coding system: a technique for the measurement of facial movement.
- [Ekman and Friesen, 1978b] Ekman, P. and Friesen, W. V. (1978b). *Manual of the Facial Action Coding System (FACS)*.
- [Elmahmudi and Ugail, 2021] Elmahmudi, A. and Ugail, H. (2021). A framework for facial age progression and regression using exemplar face templates. *The Visual Computer*, 37.
- [Kanade et al., 2000] Kanade, T., Cohn, J., and Tian, Y. (2000). Comprehensive database for facial expression analysis. In *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, pages 46–53.
- [Kartynnik et al., 2019] Kartynnik, Y., Ablavatski, A., Grishchenko, I., and Grundmann, M. (2019). Real-time facial surface geometry from monocular video on mobile gpus. In *CVPR Workshop on Computer Vision for Augmented and Virtual Reality 2019*, Long Beach, CA.
- [Kazemi and Sullivan, 2014] Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874.

- [Lucey et al., 2010] Lucey, P., Cohn, J., Kanade, T., Saragih, J., Ambadar, Z., and Matthews, I. (2010). The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. pages 94 – 101.
- [Moravec, 1988] Moravec, H. (1988). *The future of robot and human intelligence*. Harvard University Press.
- [Mumm and Mutlu, 2011] Mumm, J. and Mutlu, B. (2011). Human-robot proxemics: Physical and psychological distancing in human-robot interaction. In *Proceedings of the 6th International Conference on Human-Robot Interaction*, HRI '11, page 331–338, New York, NY, USA. Association for Computing Machinery.
- [Rohith Raj S, Pratiba D, 2020] Rohith Raj S, Pratiba D, R. K. P. (2020). Facial Expression Recognition using Facial Landmarks: A novel approach. *Advances in Science, Technology and Engineering Systems Journal*, 5(5):24–28.
- [Siam et al., 2022] Siam, A., Soliman, N., Algarni, A., Abd El-Samie, F., and Sedik, A. (2022). Deploying machine learning techniques for human emotion detection. *Computational Intelligence and Neuroscience*, 2022.
- [Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I.