



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 para
un robot de bajo coste basado en ROS

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

Unas bonitas palabras...

Quizás un segundo párrafo esté bien. No te olvides de nadie.

Un tercero tampoco viene mal para contar alguna anécdota...

¿Alguien más? Aunque sean *actores* secundarios.

Un quinto párrafo como colofón.

*A alguien especial;
si no, tampoco pasa nada*

Madrid, xx de xxxxxx de 20xx

Tu nombre

Resumen

Escribe aquí el resumen del trabajo. Un primer párrafo para dar contexto sobre la temática que rodea al trabajo.

Un segundo párrafo concretando el contexto del problema abordado.

En el tercer párrafo, comenta cómo has resuelto la problemática descrita en el anterior párrafo.

Por último, en este cuarto párrafo, describe cómo han ido los experimentos.

Acrónimos

RUR *Rossum's Universal Robots*

IFR *Federación Internacional de Robótica*

AGV *Vehículo Guiado Automático*

AMR *Robot Móvil Autónomo*

HRI *Interacción humano-robot*

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*

Índice general

1. Introducción	1
1.1. Robótica de servicio	1
1.2. Interacción Humano Robot (HRI)	4
1.2.1. Problemática. Paradoja de Moravec	5
1.2.2. Soluciones	6
1.3. Visión Artificial	7
1.4. Machine Learning	9
1.4.1. Aprendizaje supervisado	9
1.4.2. Aprendizaje no supervisado	11
1.4.3. Aprendizaje por refuerzo	12
1.5. Sistemas empotrados	14
1.5.1. Sistemas empotrados populares	14
2. Objetivos	17
2.1. Descripción del problema	17
2.2. Requisitos	18
2.3. Metodología	18
2.4. Plan de trabajo	19
3. Plataforma de desarrollo	20
3.1. Raspberry Pi 4 Model B	20
3.1.1. Raspberry Pi OS	21
3.1.2. Raspberry Pi Camera Module V2.1	22
3.2. Python	24
3.3. MediaPipe	25
3.3.1. Face Mesh	25
3.4. Detector de puntos faciales. Dlib.	27
3.5. Scikit-learn. Algoritmos de Machine Learning	27
3.5.1. Máquinas de Vector Soporte (SVM)	28

3.5.2. K Vecinos más Cercanos (KNN)	28
3.5.3. Redes Neuronales Multicapa	29
3.6. ROS (Robot Operating System)	31
4. Diseño	33
4.1. Snippets	33
4.2. Verbatim	33
4.3. Ecuaciones	34
4.4. Tablas o cuadros	34
5. Conclusiones	36
5.1. Conclusiones	36
5.2. Corrector ortográfico	37
Bibliografía	38

Índice de figuras

1.1.	iRobot Roomba 980 y WinBot 950.	2
1.2.	Robot Spot de Boston Dynamics.	2
1.3.	Taxi autónomo de Waymo.	3
1.4.	Robot DaVinci y Robot Mako.	4
1.5.	Robot Robin.	4
1.6.	Ejemplo de detección de bordes con Canny.	9
1.7.	Demo de la detección de objetos proporcionada por YOLO.	9
1.8.	Ejemplo de regresión lineal. Predicción del precio de la vivienda.	10
1.9.	Ejemplo de clasificación binaria. Predicción del tipo de casa.	11
1.10.	Ejemplo de clustering. Agrupación de datos de casas.	12
1.11.	Arduino UNO R3.	15
1.12.	Raspberry Pi Pico.	15
3.1.	Raspberry Pi 4b.	20
3.2.	Captura de pantalla de Raspberry Pi OS.	22
3.3.	Raspberry Pi Camera Module V2.1.	23
3.4.	MIPI Camera Serial Interfaze (CSI-2).	24
3.5.	Malla facial de MediaPipe.	26
3.6.	Ejemplo usando el detector de puntos faciales de dlib. Imagen obtenida del artículo [Elmahmudi and Ugail, 2021]	27
3.7.	Hiperplano de separación óptimo entre los datos de dos clases.	28
3.8.	Ejemplo de KNN con $k = 3$	29
3.9.	Red neuronal de tres capas y ocho neuronas.	30
3.10.	Ejemplo del interior de una neurona de una entrada, donde x es la entrada y a es la salida.	30
3.11.	Logo de ROS.	31
3.12.	Esquema simple de una comunicación publicador-suscriptor en ROS. . .	31

Listado de códigos

4.1. Función para buscar elementos 3D en la imagen	33
4.2. Cómo usar un Slider	34

Listado de ecuaciones

3.1. Distancia Euclídea de un vector x' con p características respecto al vector i-ésimo ($x^{(i)}$)	29
4.1. Ejemplo de ecuación con fracciones	34
4.2. Ejemplo de ecuación con array y letras y símbolos especiales	34

Índice de cuadros

3.1. Especificaciones técnicas de la Raspberry Pi 4 Model B	21
3.2. Especificaciones de Raspberry Pi OS Legacy.	22
3.3. Especificaciones de Raspberry Pi Camera Module V2.1.	23
4.1. Parámetros intrínsecos de la cámara	35

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*; esta 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.

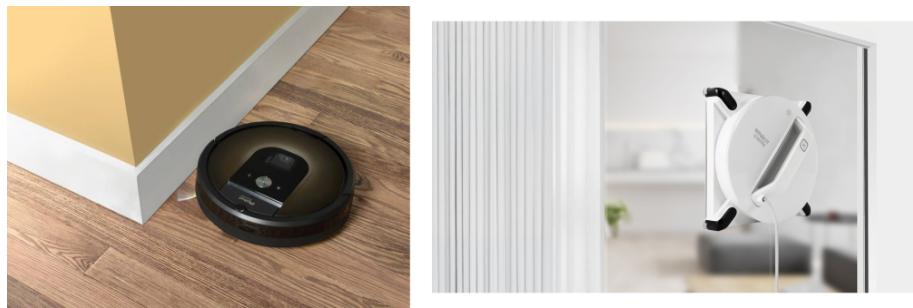


Figura 1.1: iRobot Roomba 980 y WinBot 950.

- *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.

- *Conducción autónoma.* Una tecnología cada vez más madura y extendida. Empresas como Waymo (Figura 1.3) 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-automáticos.



Figura 1.3: Taxi autónomo de Waymo.

- *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).
- *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 1.4).

Además, en los últimos años está surgiendo un nuevo tipo de robótica sanitaria, los robots de compañía o asistentes personales. Estos consiguen empatizar con los pacientes y hacerles pasar un rato más ameno o ayudarles a que no se sientan solos. 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 1.5), el cual ayuda a los niños a superar el miedo de ir al médico.

¹mBlock IDE: <https://ide.mblock.cc>

²ROS: <https://www.ros.org/>



Figura 1.4: Robot DaVinci y Robot Mako.



Figura 1.5: 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, los humanos.

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 actualmente 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 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*:

- *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 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.

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 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 cuerpo y cerebro no está preparado para ello, y muy triviales para las máquinas.

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 no es un tema relevante en el contexto de este trabajo.

Percepción

Lo que para nosotros serían los sentidos, en los robots lo podemos sustituir por sensores. 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.

Estos sensores por si sólo no tienen ningún valor, sino que detrás de todos esos datos capturados deben existir algoritmos que saquen conclusiones de todos ellos. Por ejemplo, se puede realizar detección de personas en los fotogramas capturados por una cámara o extraer 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 un real razonamiento humano dentro

de un robot, 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 el cual se esté tratando en la conversación. Se podría estar expresando que no se ha visto ningún error en la carta que se escribiendo o que no se ha visto llegar el taxi se había pedido. Entonces los robots no se pueden dedicar a detectar únicamente frases sueltas y actuar ante ellas. Para ello existen modelos de lenguaje como GPT-3 que consiguen simular que están entendiendo el diálogo, pero en realidad sólo están repitiendo conversaciones con las que han sido entrenados.
- *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. El sistema de detección de emociones desarrollado en este trabajo estaría presente en este apartado y el anterior de *Atención*.

En las secciones 1.2.2 y 1.2.2 se han tratado temas como el reconocimiento facial o la detección de personas, ambas son conseguidas a través de lo que se conoce como *Visión Artificial*, técnica la cual se introduce 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 y actuar según la situación. Para ello, se realiza lo que se conoce como procesamiento de imagen, que se puede dividir en las siguientes fases o etapas:

- *Digitalización.* Proceso de transformación que sufre una imagen analógica a otra digital para que pueda ser manipulada por un ordenador. Una máquina sólo sabe comprender números, por lo tanto la imagen estará representada como una matriz de números (píxeles).

- *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 realce del contraste.
- *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.
- *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.
- *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.
- *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 como *Persona* o *Planta*.
- *Interpretación.* Etapa final, la cual se encarga de dar significado a los objetos reconocidos. Por ejemplo localizar que objetos son dinámicos o estáticos, o detectar la posición en la que se encuentra un cuerpo.

Lo explicado y desarrollado anteriormente se puede enmarcar dentro de lo que se conoce como la *Visión Artificial Clásica*, enfocada en la utilización de algoritmos específicos para procesar imágenes y reconocer en ellas características básicas. Por ejemplo, el algoritmo de detección de bordes Canny³ (Figura 1.6).

El auge del Machine Learning, que es introducido en la siguiente sección (Sección 1.4), está expandiendo exponencialmente las capacidades de la Visión Artificial. Son técnicas muy potentes que permiten resultados mucho mejores que los ofrecidos por la visión clásica y además mucho más fáciles de implementar. Dentro de estas nuevas capacidades —por ejemplo— encontramos la detección de objetos proporcionada por YOLO⁴ (Figura 1.7), sistema desarrollado con técnicas de Machine Learning.

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

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

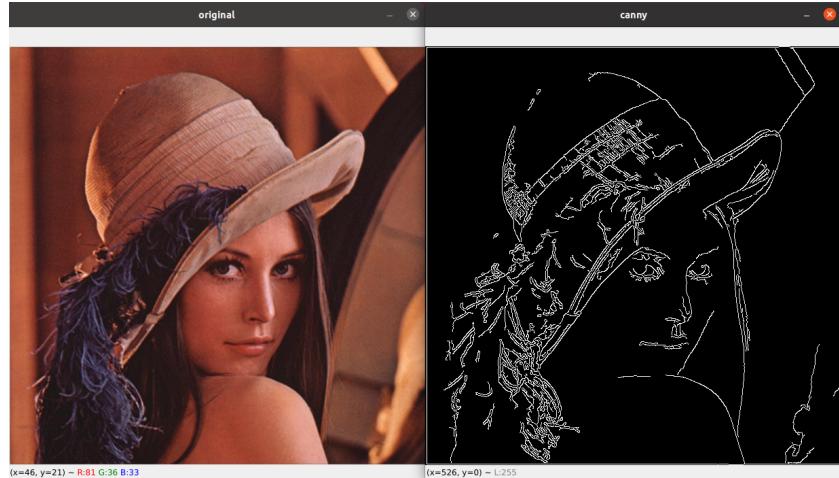


Figura 1.6: Ejemplo de detección de bordes con Canny.

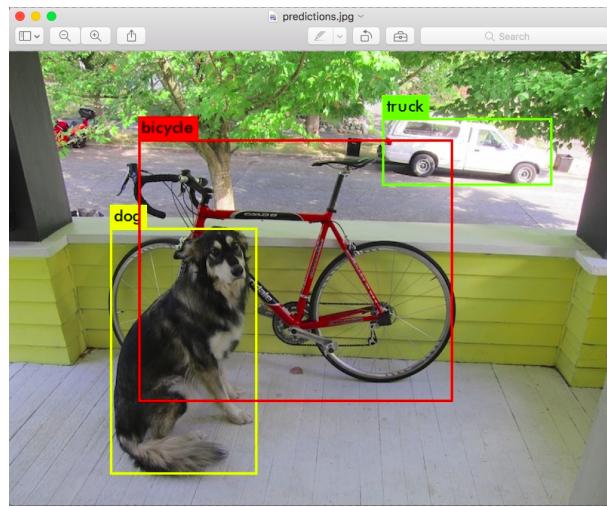


Figura 1.7: Demo de la detección de objetos proporcionada por YOLO.

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 programados explícitamente para dichos casos. Según el tipo de aprendizaje que realicen los algoritmos podemos clasificar estos 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.

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 1.8).

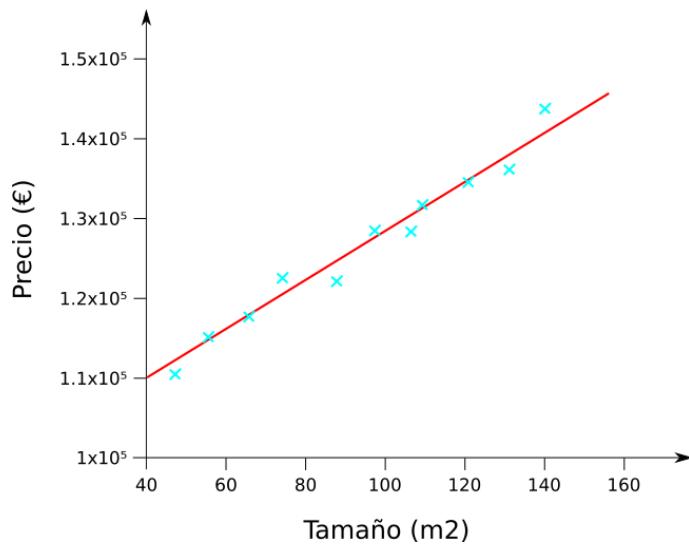


Figura 1.8: Ejemplo de regresión lineal.
Predicción del precio de la vivienda.

En el ejemplo de la Figura 1.8 los datos usados como entrada serían los tamaños de las viviendas y los datos de salida los precios. Las cruces azules representan estos valores de entrada y salida y la línea roja representa la relación obtenida entre ellos. A partir de esa función lineal, el algoritmo ya sería capaz de realizar cualquier predicción para una pareja de datos de tamaño y precio.

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 únicamente posee 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 binaria sería la predicción del tipo de casa (piso o chalet) en función de la distancia al centro de la ciudad y el tamaño del jardín (Figura 1.9).

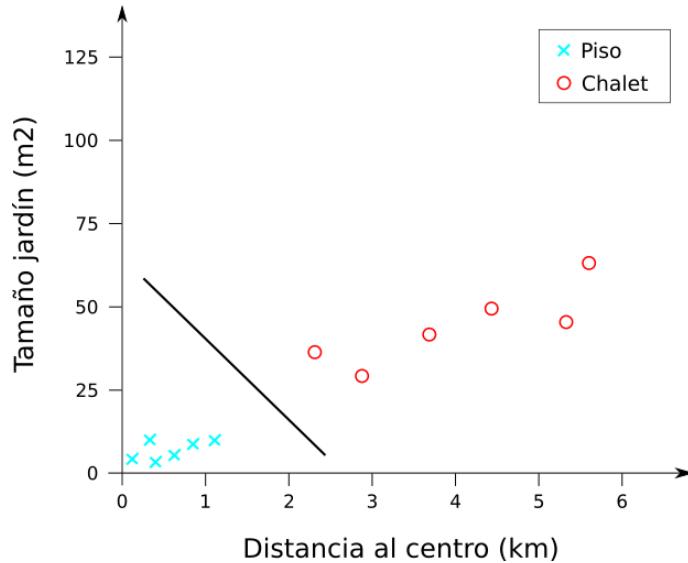


Figura 1.9: Ejemplo de clasificación binaria.
Predicción del tipo de casa.

En el ejemplo de la Figura 1.9 los datos usados como entrada serían el tamaño del jardín y la distancia al centro y los datos de salida el tipo de casa. Se puede observar como el algoritmo de clasificación ha separado los datos en dos conjuntos diferenciados, cualquier dato nuevo que caiga en uno de esos conjuntos será clasificado con su etiqueta correspondiente. Es a eso a lo que se le llama predicción.

Los algoritmos más utilizados para realizar clasificación son SVM (Support Vector Machine), KNN (K Nearest Neighbour), Árboles de decisión y Redes Neuronales (Convolucionales, Recurrentes, etc.).

1.4.2. Aprendizaje no supervisado

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 que son gatos o perros.

Un ejemplo, siguiendo con la temática de las viviendas, sería agrupar las casas en función de la distancia al centro y del tamaño del jardín (Figura 1.10). Esto se conoce como clustering o segmentación.

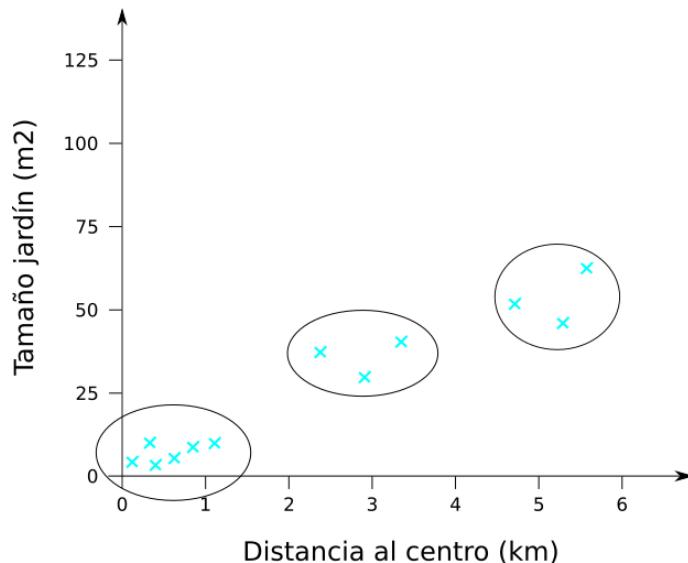


Figura 1.10: Ejemplo de clustering.
Agrupación de datos de casas.

En el ejemplo de la Figura 1.10 los datos de entrada serían el tamaño del jardín y la distancia al centro. Además al algoritmo, en este tipo de aprendizaje, se le suele indicar en cuántas clases se desea que se clasifiquen los datos. También se puede no indicar esta información y dejarle total libertad a la máquina. 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

No se le 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 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, ambos son máquinas o personajes actuando en un entorno cambiante.

A diferencia de los otros tipos de aprendizaje que se intenta reducir el error, aquí se intenta maximizar la recompensa. Los componentes del Aprendizaje Reforzado son:

- **Agente.** Modelo que se pretende entrenar para que aprenda a tomar decisiones.
- **Ambiente.** Entorno donde interactúa el agente, el cual contiene las reglas posibles en cada momento.

Además la relación que se realimenta entre los componentes mencionados anteriormente cuenta con los siguientes nexos:

- **Acción.** Posibles acciones que puede tomar el Agente en un momento determinado.
- **Estado (del ambiente).** Indicadores del ambiente de cómo están los elementos que lo componen en ese momento.
- **Recompensas o penalizaciones.** Por cada acción tomada por el Agente, se obtendrá un premio o una penalización que orientará al Agente en si lo está haciendo bien o mal.

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 ver tarjetas gráficas muy potentes realizando este tipo de trabajo, pero no siempre se posee del dinero o del espacio donde alojar dichas 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. Esta tarea es una de las que se llevan a cabo en este trabajo.

1.5. Sistemas empotrados

Un sistema empotrado (también conocido como *embedido*) es un sistema caracterizado por su tamaño reducido y mínimo precio, teniendo por contra un poder computacional 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 prácticamente nulo.

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 múltiples fabricantes de sistemas empotrados, en este capítulo de introducción se nombraran los dos más grandes actualmente: Arduino⁵ y Raspberry⁶. Ambos 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 (Figura 1.11) o Arduino Nano. Son microcontroladores integrados en el mismo chip con todos los componentes necesarios para su correcto funcionamiento (resistencias, condensadores, pines para conectar elementos, etc). Además de esto mencionado, la ventaja que nos proporciona este tipo de placas y 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.

Raspberry. Tiene a la venta también microcontroladores como la Raspberry Pi Pico (Figura 1.12) pero su producto principal son los SBC, su último modelo la Raspberry Pi 4 Model B (explicada más en profundidad en la Sección 3.1). El concepto 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 en una sola placa. Este mencionado sistema operativo es Raspberry Pi OS basado en

⁵Arduino: <https://www.arduino.cc/>

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

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

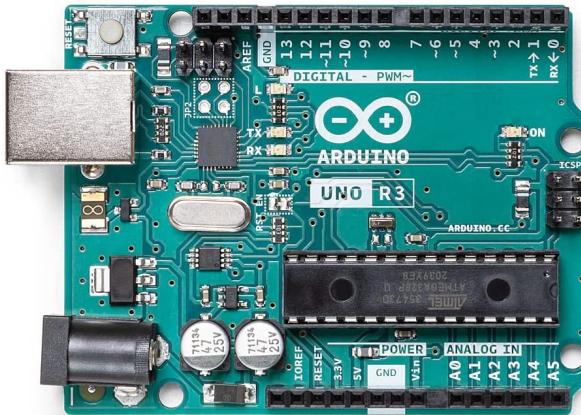


Figura 1.11: Arduino UNO R3.

Debian (explicado más en profundidad en la Sección 3.1.1).

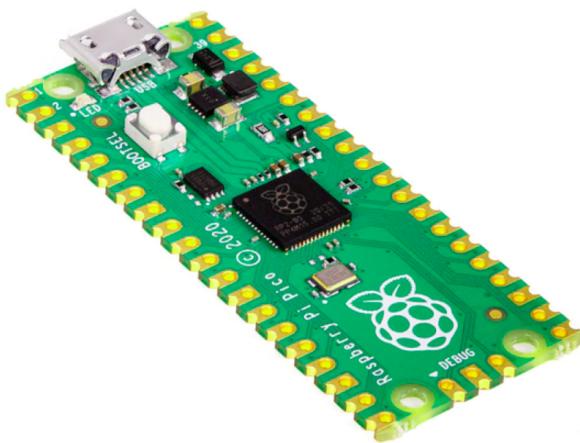


Figura 1.12: Raspberry Pi Pico.

Tras poner en contexto de manera general a la Robótica de Servicio y entender que entendemos por HRI, Visión Artificial, Machine Learning y Sistemas Empotrados, con este proyecto se pretende desarrollar una herramienta que mediante Visión Artificial y Machine Learning sea capaz de detectar emociones faciales para ayudar en una interacción entre humano y robot. Además, esta herramienta deberá funcionar en un sistema empotrado.

En el siguiente capítulo (Capítulo 2) se realizará una descripción del problema a desarrollar y la metodología y el plan de trabajo que se ha llevado a cabo. Prosiguiendo con el capítulo 3, el autor expondrá las herramientas hardware y software utilizadas. En el capítulo 4 se mostrará todo el desarrollo del trabajo y los experimentos realizados.

Por último, en el capítulo 5 se podrán encontrar las conclusiones finales.

Capítulo 2

Objetivos

Una vez presentado el contexto general en el cual se enmarca nuestro trabajo de fin de grado, se procederá a realizar una descripción del problema planteando los objetivos finales de este, requisitos, metodología y plan de trabajo para llegar a resolverlo.

2.1. Descripción del problema

El objetivo principal de este TFG 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. Estudiar cuál será la técnica más óptima en cuanto al reconocimiento de emociones. 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. Tras decidir que la extracción de puntos característicos faciales y el posterior entrenamiento con el tratamiento de ellos será la técnica más óptima y liviana para nuestro hardware, el siguiente paso será estudiar qué método para extraer esos puntos faciales es el más rápido y preciso.
3. Crear un dataset de valor con los puntos característicos faciales. Se deberá 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 nuestro sistema en ROS para facilitar su uso en cualquier 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 entre personas y robots, 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 4b, 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 nuestro hardware y el que por lo tanto 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 TFG 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 repositorio¹ de Github en el cual se ha ido subiendo todo el código del desarrollo del sistema de este trabajo. Además, este repositorio posee una Wiki² que contiene explicaciones semanales de todo lo llevado a cabo durante estos meses de trabajo.

La herramienta final de ROS se puede encontrar en un repositorio³ de GitHub a parte. Se ha hecho de esta manera para que esté disponible para toda la comunidad de ROS y 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 entrenamiento.* 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 de TFG se realizaron lecturas sobre otros proyectos similares.
2. *Estudio de técnicas de reconocimiento de emociones.* Investigación 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 4b + Raspberry Pi OS + Raspberry Pi Camera Module V2).
3. *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.
4. *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.
5. *Búsqueda de robustez.* El sistema ya estaba desarrollado pero no poseía la suficiente robustez como para ser usado en un robot. Esta fase se encargó de aumentar dicha robustez a nuestro sistema consiguiendo más fiabilidad en la detección de las emociones en un entorno real y práctico.
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 TFG.

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 plataforma más usada por la mayoría del público.

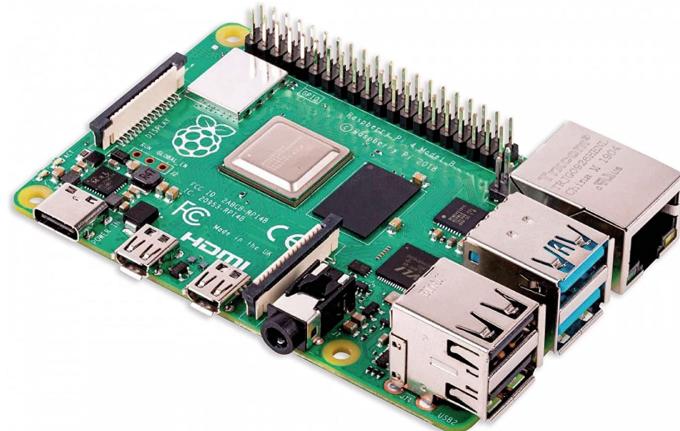


Figura 3.1: Raspberry Pi 4b.

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 auténtica joya y son

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

cada vez más usuarios los que la utilizan en diversos proyectos. Podemos encontrarla —por ejemplo— como centro doméstico inteligente para controlar 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 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. Es por todo ello que 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).

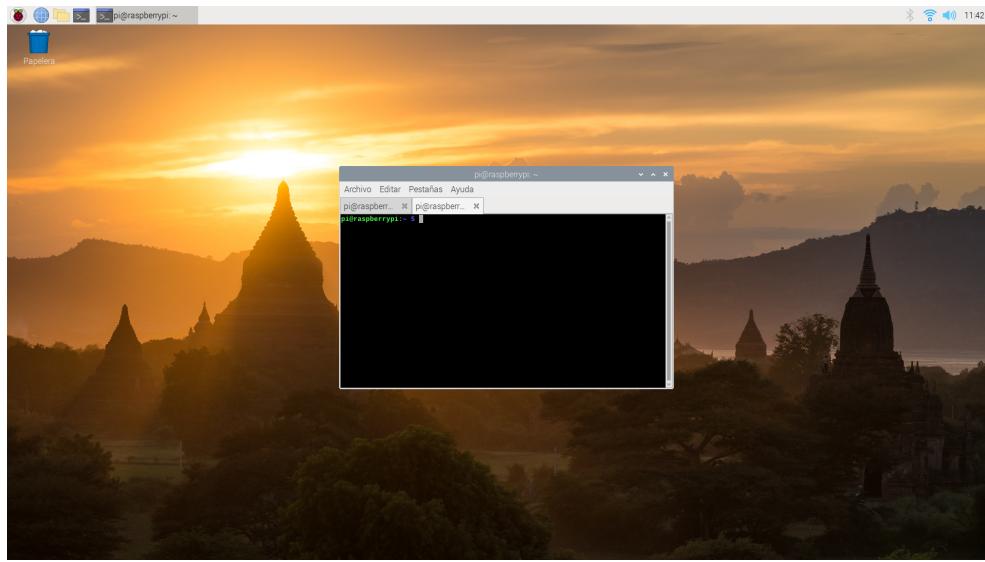


Figura 3.2: Captura de pantalla de Raspberry Pi OS.

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 las versiones de Raspberry Pi OS de 64-bit ofrecían más rendimiento, todavía no estaban maduras y no ofrecían 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.

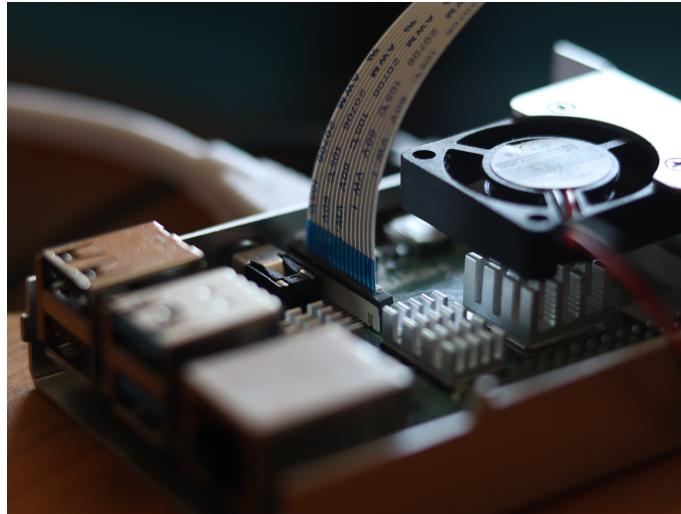


Figura 3.4: MIPI Camera Serial Interfaze (CSI-2).

- *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*.

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 repositorio⁴ de GitHub.

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.

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

³MediaPipe: <https://mediapipe.dev/>

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

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

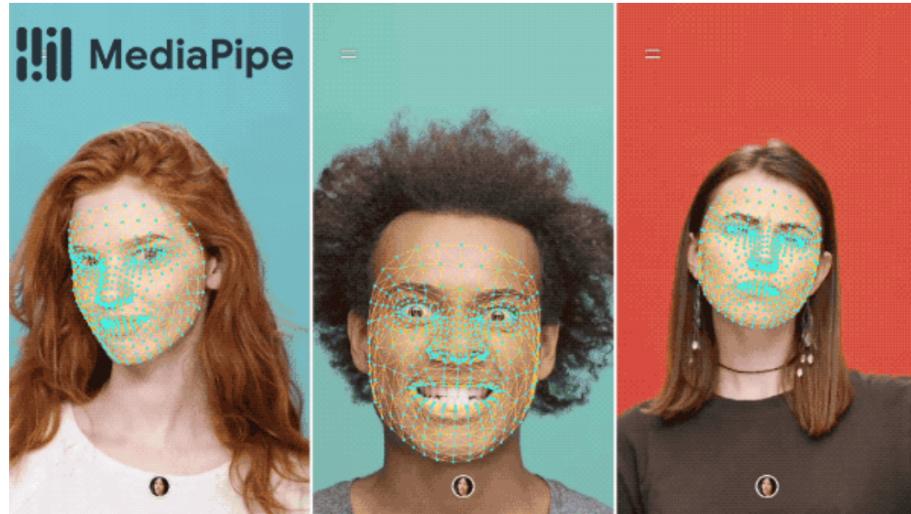


Figura 3.5: Malla facial de MediaPipe.

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. 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).

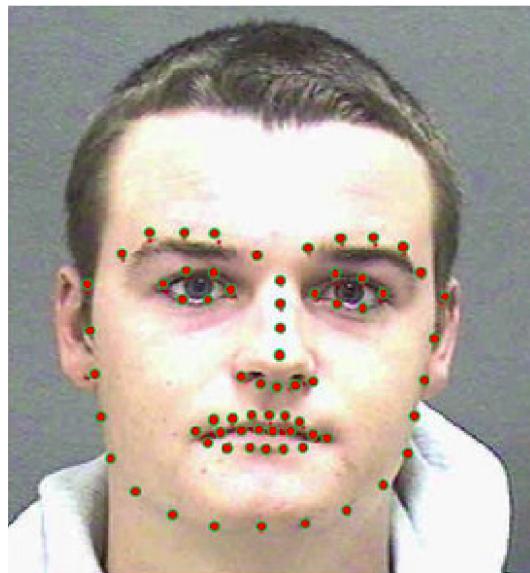


Figura 3.6: Ejemplo usando el detector de puntos faciales de dlib.
Imagen obtenida del artículo [Elmahmudi and Ugail, 2021]

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:

- 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.

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

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

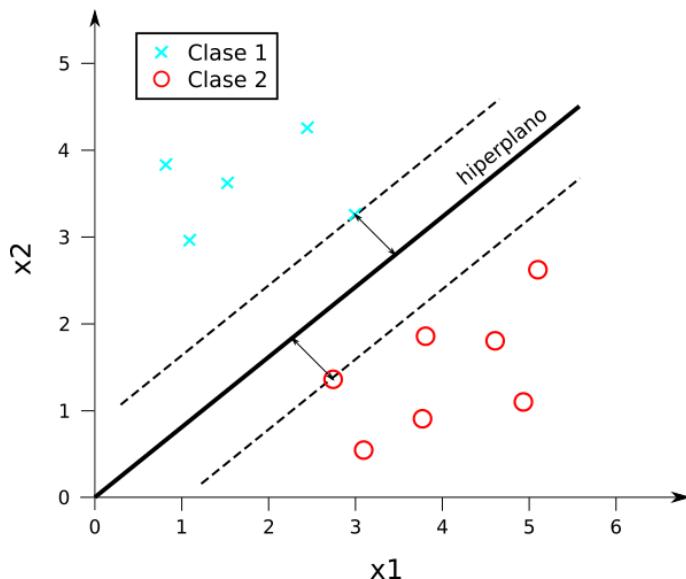


Figura 3.7: Hiperplano de separación óptimo entre los datos de dos clases.

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. Entonces, 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).

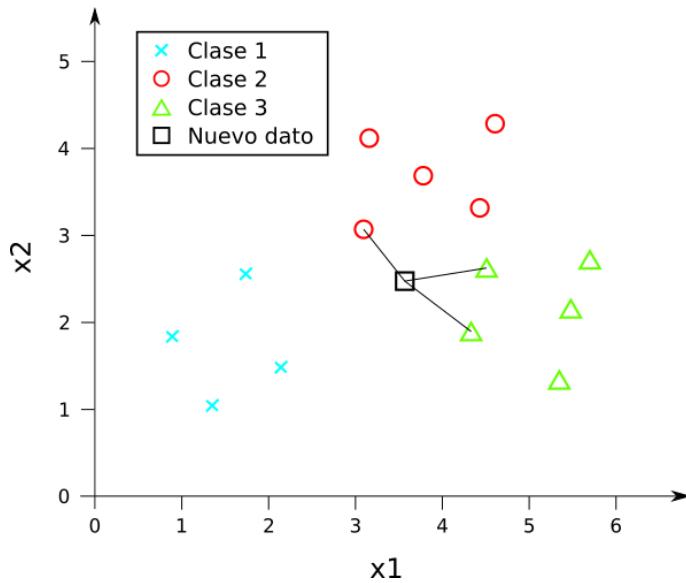


Figura 3.8: Ejemplo de KNN con $k = 3$.

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

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 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. Realizo la excepción anterior porque existen también redes neuronales de una sola capa, denominadas perceptrón.

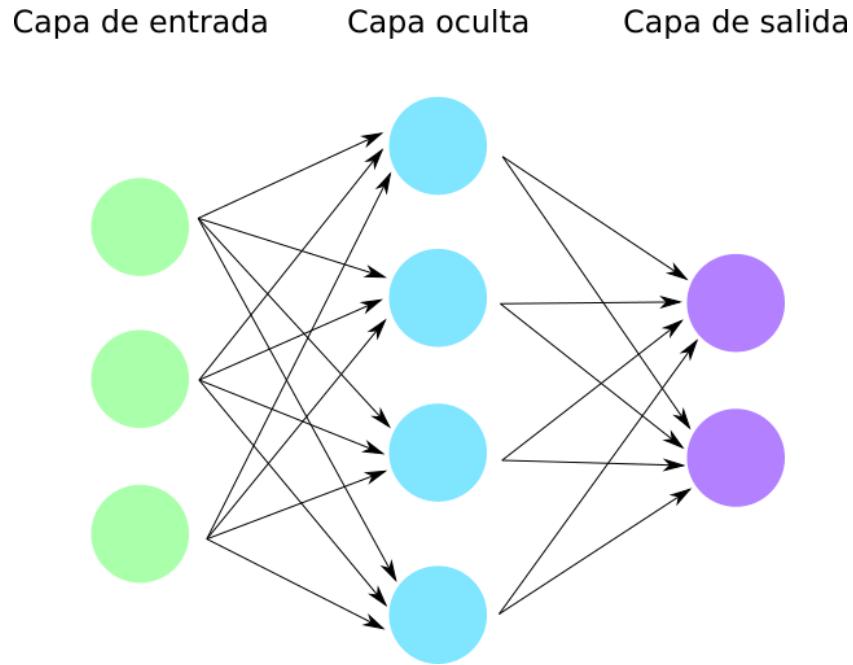


Figura 3.9: Red neuronal de tres capas y ocho neuronas.

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 término independiente b (Figura 3.10).

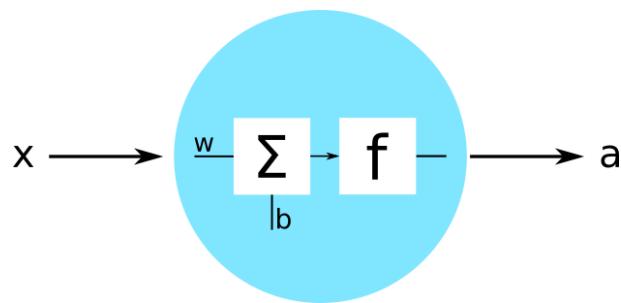


Figura 3.10: Ejemplo del interior de una neurona de una entrada, donde x es la entrada y a es la salida.

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 te devuelve unas respectivas salidas, sin nosotros conocer lo que ha sucedido dentro.

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.

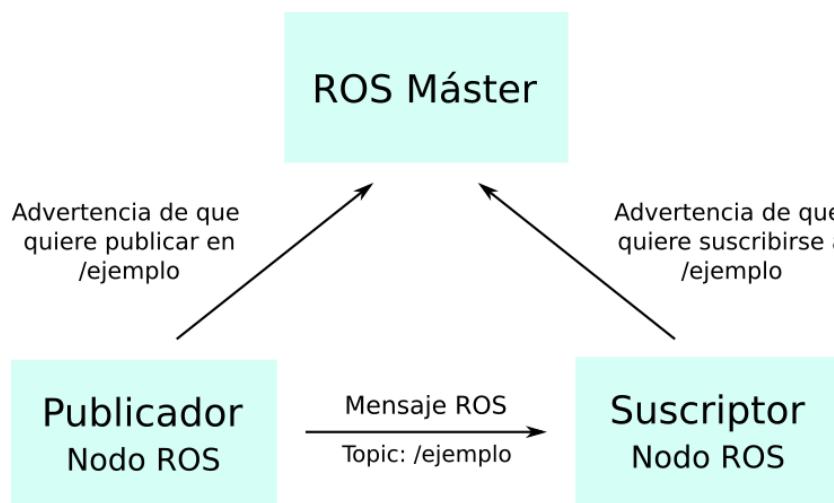


Figura 3.12: Esquema simple de una comunicación publicador-suscriptor en ROS.

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,

⁷ROS: <https://www.ros.org/>

podemos encontrar algunos enfocados en —por ejemplo— identificación de objetos 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

Diseño

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo. En este capítulo (y quizás alguno más) es donde, por fin, describes detalladamente qué has hecho y qué experimentos has llevado a cabo para validar tus desarrollos.

4.1. Snippets

Puede resultar interesante, para clarificar la descripción, mostrar fragmentos de código (o *snippets*) ilustrativos. En el Código 4.1 vemos un ejemplo escrito en C++.

```
void Memory::hypothesizeParallelograms () {
    for(it1 = this->controller->segmentMemory.begin(); it1++) {
        squareFound = false; it2 = it1; it2++;
        while ((it2 != this->controller->segmentMemory.end()) && (!squareFound))
        {
            if (geometry::haveACommonVertex((*it1), (*it2), &square)) {
                dist1 = geometry::distanceBetweenPoints3D ((*it1).start, (*it1).end);
                dist2 = geometry::distanceBetweenPoints3D ((*it2).start, (*it2).end);
            }
        // [...]
    }
}
```

Código 4.1: Función para buscar elementos 3D en la imagen

En el Código 4.2 vemos un ejemplo escrito en Python.

4.2. Verbatim

Para mencionar identificadores usados en el código —como nombres de funciones o variables— en el texto, usa el entorno literal o verbatim

```

def mostrarValores():
    print (w1.get(), w2.get())

master = Tk()
w1 = Scale(master, from_=0, to=42)
w1.pack()
w2 = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w2.pack()
Button(master, text='Show', command=mostrarValores).pack()

mainloop()

```

Código 4.2: Cómo usar un Slider

`hypothesizeParallelograms()`. También se puede usar este entorno para varias líneas, como se ve a continuación:

```

void Memory::hypothesizeParallelograms () {
    // add your code here
}

```

4.3. Ecuaciones

Si necesitas insertar alguna ecuación, puedes hacerlo. Al igual que las figuras, no te olvides de referenciarlas. A continuación se exponen algunas ecuaciones de ejemplo: Ecuación 4.1 y Ecuación 4.2.

$$H = 1 - \frac{\sum_{i=0}^N \frac{(\frac{d_{js} + d_{je}}{2})}{N}}{M} \quad (4.1)$$

Ecuación 4.1: Ejemplo de ecuación con fracciones

$$v(\text{entrada}) = \begin{cases} 0 & \text{if } \epsilon_t < 0,1 \\ K_p \cdot (T_t - T) & \text{if } 0,1 \leq \epsilon_t < M_t \\ K_p \cdot M_t & \text{if } M_t < \epsilon_t \end{cases} \quad (4.2)$$

Ecuación 4.2: Ejemplo de ecuación con array y letras y símbolos especiales

4.4. Tablas o cuadros

Si necesitas insertar una tabla, hazlo dignamente usando las propias tablas de L^AT_EX, no usando pantallazos e insertándolas como figuras... En el Cuadro 4.1 vemos

un ejemplo.

Parámetros	Valores
Tipo de sensor	Sony IMX219PQ[7] CMOS 8-Mpx
Tamaño del sensor	3.674 x 2.760 mm (1/4"format)
Número de pixels	3280 x 2464 (active pixels)
Tamaño de pixel	1.12 x 1.12 um
Lente	f=3.04 mm, f/2.0
Ángulo de visión	62.2 x 48.8 degrees
Lente SLR equivalente	29 mm

Cuadro 4.1: Parámetros intrínsecos de la cámara

Capítulo 5

Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

5.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

5.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

Bibliografía

- [Dautenhahn, 2007] Dautenhahn, K. (2007). Methodology themes of human-robot interaction: A growing research field. *International Journal of Advanced Robotic Systems*, 4.
- [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.
- [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.
- [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.