



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2021-2022

Trabajo fin de grado

Sistema de Monitorización de animales de laboratorio
bajo plataforma de bajo coste

Tutor: Julio Vega Pérez

Autor: Isabel Cebollada Gracia



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

Desde hace siglos, la experimentación con animales se ha llevado a cabo para saber qué les sucede a los humanos y al mundo que les rodea. A día de hoy, aunque de una manera distinta debido tanto al avance de la humanidad como de la tecnología, esta experimentación sigue desarrollándose en diferentes ámbitos; para el estudio de los comportamientos de animales o el uso de éstos como modelo de investigación en diferentes campos, desde el testado de productos hasta la sanidad humana.

Los estudios con animales en la medicina han tenido gran importancia en el desarrollo de vacunas modernas como la tuberculosis o la meningitis ya que los animales, más concretamente los ratones, sufren enfermedades similares a los humanos. De esta manera, los ratones se han convertido en uno de los principales animales para estudiar su comportamiento y el efecto que éste puede tener en la detección de enfermedades neurológicas, como el autismo, el parkinson o el alzheimer.

La observación de estos animales en todo momento es una necesidad para analizar y estudiar su comportamiento, así como tener la información de las condiciones del entorno a las que se encuentran, haciendo necesario que haya trabajadores analizando las condiciones y grabaciones que tienen que hacerse a los ratones diariamente. Para evitar esto, el fin de este trabajo es crear un sistema dotando con los sensores necesarios al entorno de los roedores para obtener la información de forma automatizada y traducirlo en una interfaz comprensible para cualquier usuario, así como el control por vídeo a través de algoritmos de visión artificial para detectar los movimientos de los animales bajo una plataforma económicamente accesible para todo el mundo.

Durante todo el desarrollo del trabajo han surgido problemas en conexiones, seguridad, instalación, en el uso de algunos sensores y el uso de herramientas no usadas hasta el momento. Tras el estudio y la comprensión de los errores encontrados, así como con la ayuda de internet y de las experiencias de otros usuarios, tanto con problemas similares como con posibles soluciones, todos los errores han podido solucionarse.

Acrónimos

IA *Inteligencia Artificial*

VA *Visión Artificial*

DL *Deep Learning*

ML *Machine Learning*

SO *Sistema Operativo*

IU *Interfaz de Usuario*

2FA *Factor de doble Autenticación*

Índice general

1. Introducción	1
1.1. La robótica	2
1.2. Inteligencia Artificial	5
1.2.1. Visión Artificial	7
1.3. Machine Learning	8
1.4. Deep Learning	9
1.5. Sistemas multisensoriales	12
2. Objetivos	15
2.1. Descripción del problema	15
2.2. Requisitos	16
2.3. Metodología	16
3. Plataforma de desarrollo	18
3.1. Plataforma hardware	18
3.2. Infraestructura software	22
3.2.1. Lenguaje Python	23
3.2.2. TensorFlow Lite	24
3.2.3. OpenCV	25
3.3. Flask	26
3.3.1. Node-Red	27
4. Sistema multisensorial	30
4.1. Desarrollo hardware	31
4.2. Desarrollo software	33
4.2.1. Lectura sensorial con Python	35
4.2.2. Creación de la interfaz de usuario	35
4.2.3. Integración de las cámaras en la UI	38
4.2.4. Seguridad	41

4.2.5. Autoarranque	42
5. Conclusiones	46
5.1. Conclusiones	46
5.2. Corrector ortográfico	47
Bibliografía	48

Índice de figuras

1.1.	Producción en serie en la fábrica de coches Ford.	2
1.2.	Robot Da Vinci	4
1.3.	Usos de la impresión 3D en medicina.	4
1.4.	Ejemplos de la robótica en diferentes campos.	5
1.5.	Ejemplos de la robótica en diferentes campos.	5
1.6.	Detección de fracturas en la radiografía mediante IA	6
1.7.	Algunos usos en la visión artificial.	8
1.8.	Esquema de relaciones entre IA, ML y DL.	10
1.9.	Simple ejemplo de una red neuronal en el algoritmo de Deep Learning.	11
1.10.	Decreto sobre la Acrópolis de Atenas reconstruido con Ithaca.	11
1.11.	Reconocimiento facial con técnica de Deep Learning.	11
1.12.	Patrón de movimientos e imagen grabada del grupo de cerdos.	13
1.13.	Esquema del desarrollo del sistema mediante sensores.	13
3.1.	Sistemas empotrados.	19
3.2.	Sensor BME680.	19
3.3.	Sensor DS18B20.	20
3.4.	Sensores térmicos utilizados en el trabajo.	20
3.5.	Pi Camera.	21
3.6.	Sensor de nivel de agua.	21
3.7.	Sensor MQ-135.	22
3.8.	Imagen de Raspberry Pi OS.	22
3.9.	24
3.10.	Detección de ratones en una imagen usando TensorFlow Lite.	25
3.11.	Ejemplo de detección de bordes con OpenCV.	26
3.12.	Blog creado con Flask.	27
3.13.	Distintos flujos hechos en Node-Red	28
3.14.	IU de un sistema de control de un garaje.	29

4.1.	Raspberry Pi 4B utilizada para el desarrollo del presente TFG.	31
4.2.	Imagen generada después de la conversión de los datos numéricos.	32
4.3.	Esquema de conexiones.	33
4.4.	Diagrama de casos de uso del proyecto.	34
4.5.	Esquema de clases de cada sensor hecho en UMLet.	34
4.6.	Proceso de indicación del nivel de agua.	37
4.7.	IU sin los sensores cámaras.	38
4.8.	Incorporación de la fecha y hora en la visualización de la PiCam.	39
4.9.	Proceso de registro en el servidor de Flask de la PiCam.	40
4.10.	Visualización de la interfaz de usuario.	42
4.11.	Acceso al flujo de nodos en Node-Red.	43
4.12.	Acceso a la interfaz de usuario desde distintos dispositivos.	44
4.13.	Proceso de autoarranque del sistema.	45

Listado de códigos

3.1. Código para generar un modelo de detección de ratones en Python	23
4.1. Función para crear un Thread por sensor y obtener su lectura.	36
4.2. Ejemplo de salida del fichero Python	36
4.3. Código para incorporar la fecha en la esquina superior izquierda.	39
4.4. Código simple que crea un servidor web en el puerto 8000 y muestra el contenido de index.html.	39

Listado de ecuaciones

Índice de cuadros

3.1. Cuadro de características del sensor BME680.	19
3.2. Cuadro de características del sensor DS18B20.	20

Capítulo 1

Introducción

El desarrollo tecnológico ha provocado el avance en la robótica, un campo de investigación muy amplio en la actualidad que está facilitando la vida a los humanos. Cualquier robot está formado por tres componentes principales: un sistema de control, sensores y actuadores. Dentro de los numerosos sensores que pueden tener los robots, uno de los que está adquiriendo mayor relevancia en los últimos años es el sensor de visión.

Los sensores de visión aportan mucha información y son baratos, pero se requiere de un proceso arduo para extraer la información útil en tiempo real. Por este motivo se busca la eficiencia en resultados y tiempo, lo que significa que el proceso de extracción de la información debe ser rápido. Es por ello que en los últimos años, en el campo de la visión artificial moderna, están adquiriendo gran importancia los algoritmos de Deep Learning (DL) o aprendizaje profundo, usados para adquirir esta información.

Los algoritmos más utilizados hoy en día para obtener la información de los sensores de Visión Artificial (VA) se obtienen bajo la técnica del Machine Learning (ML) o aprendizaje automático. Estos algoritmos son mejores que los que se tenían anteriormente, y se han podido desarrollar gracias al avance de la capacidad de cómputo. El objetivo del ML es que las máquinas sean capaces de aprender sin tener que ser programadas explícitamente.

Un subcampo del ML es el DL, que aunque este segundo utiliza algoritmos del primero, sus algoritmos se basan en estructuras similares al modelo del cerebro humano, emulando redes neuronales. A medida que el algoritmo de DL obtiene más información, aumenta su precisión. Debido a su gran capacidad de abstracción, es el más usado en

VA.

A continuación se describen estos conceptos y el contexto en el que se enmarca el presente trabajo. Asimismo se presentan sistemas actuales que existen en el mercado con una idea similar a la de este trabajo.

1.1. La robótica

La robótica, aunque no siempre bajo ese nombre, se lleva utilizando desde hace mucho tiempo con el propósito de desarrollar herramientas automatizadas. Hasta mediados del siglo pasado, esta disciplina se centraba en el desarrollo de máquinas que realizaban un trabajo que resultaba tedioso, debido a su repetitividad, para el ser humano. Esta concepción se refleja con Henry Ford y la producción en serie de su fábrica de coches inaugurada en 1901 (Figura 1.1).



Figura 1.1: Producción en serie en la fábrica de coches Ford.

Sin embargo, la palabra *robo* —origen de la palabra robot que significa trabajo forzado— aparece por primera vez en 1921 en una obra de teatro de un autor checo llamado Karel Čapek. Empezando a concebir esta palabra a principios de siglo pasado, comenzó a nacer el desarrollo de la idea de robótica que existe hoy en día y que no se centra exclusivamente en el desarrollo de herramientas automatizadas, sino que se concibe como una industria interdisciplinaria que surge de la intersección de la ciencia, la ingeniería y la tecnología. Esta nueva concepción une el conocimiento científico,

computacional e informático con diversas ramas de la ingeniería, ya que no solo implica el estudio de robots, sino también de su diseño, programación y aplicación. Así, la robótica incluye disciplinas como la IA, la informática, la VA, la programación o el álgebra, entre otras muchas.

Tras la primera aparición de la palabra *robota*, la definición de robot ha ido formándose hasta la que hay en la actualidad: cualquier máquina que opera de forma automática y autónoma, y que sustituye a los seres humanos en determinadas tareas, especialmente las peligrosas, aburridas o pesadas. Está compuesto por el sistema de control, los sensores y los actuadores.

La analogía de un robot respecto a un ser humano es la siguiente. Los sensores son sus sentidos, que transmiten la información percibida, bien sea del propio robot o del entorno al sistema de control, que se asemeja al cerebro humano. El sistema de control toma decisiones adaptándose a la información recibida, que modifica bien el entorno o bien la manera de actuar internamente del propio robot. Y finalmente los actuadores, que permiten la modificación del entorno; su equivalente en el cuerpo humano serían las extremidades.

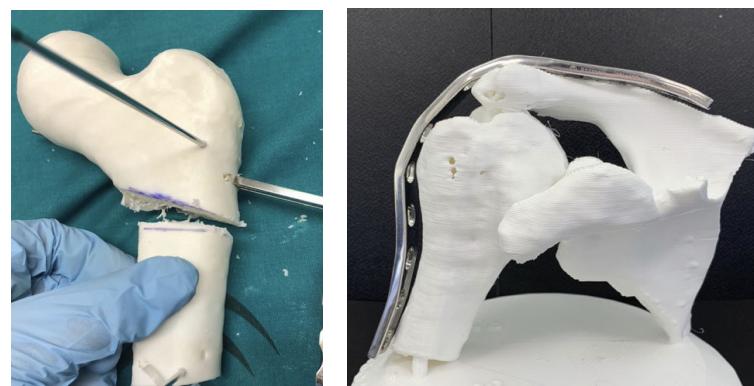
El objetivo de la robótica es ayudar al ser humano en todos los ámbitos, tanto en la vida cotidiana como en la vida profesional, evitando los trabajos perjudiciales, e incluso haciendo trabajos que el ser humano no sería capaz de hacer por sí mismo en distintos sectores.

Uno de estos sectores es la medicina. En este ámbito cabe resaltar el robot Da Vinci (Figura 1.2), uno de sus robots más famosos que permite a un cirujano comandar órdenes a través de una consola, permitiéndole realizar cirugías de alta precisión eliminando los temblores nerviosos que un humano pueda tener y permitiendo al cirujano una mayor visión. Otro ejemplo en el campo de la medicina se encuentra en la impresión 3D. En [Andrés-Cano et al., 2021], la impresión 3D de biomodelos de partes del cuerpo (por ejemplo huesos) adquiere un papel útil en fracturas complejas, displasias de cadera o en tumores óseos principalmente. Con estos biomodelos, no solo disminuye tanto el tiempo quirúrgico como los costes de intervención, sino que además se reduce la dosis intraoperatoria de radiación. También sirven para la planificación preoperatoria (Figura 1.3-a) o el premodelado de placas (Figura 1.3-b). Con la impresión 3D también se han creado férulas para los pacientes (Figura 1.3-c), ya que ofrecen una mayor

adaptación a la anatomía del paciente que las férulas de yeso o plástico que se acostumbran a llevar.



Figura 1.2: Robot Da Vinci



(a) Preoperatoria

(b) Premodelado de placas



(c) Férulas con impresión 3D

Figura 1.3: Usos de la impresión 3D en medicina.

Otro de los campos más importantes de la robótica es la manipulación en entornos inaccesibles u hostiles para el ser humano. Un ejemplo de ello lo encontramos en la carrera espacial. Uno de estos robots es el Curiosity (Figura 1.4-a), cuya misión era

evaluar la habitabilidad en Marte recogiendo información del entorno. Otro ejemplo de entornos hostiles se encuentra en el lugar que quedó tras el accidente nuclear que ocurrió en Fukushima en 2011, causado por un terremoto y el consecuente tsunami. Gracias a los robots (Figura 1.4-b) se pudo acceder a la zona para poder limpiarla.

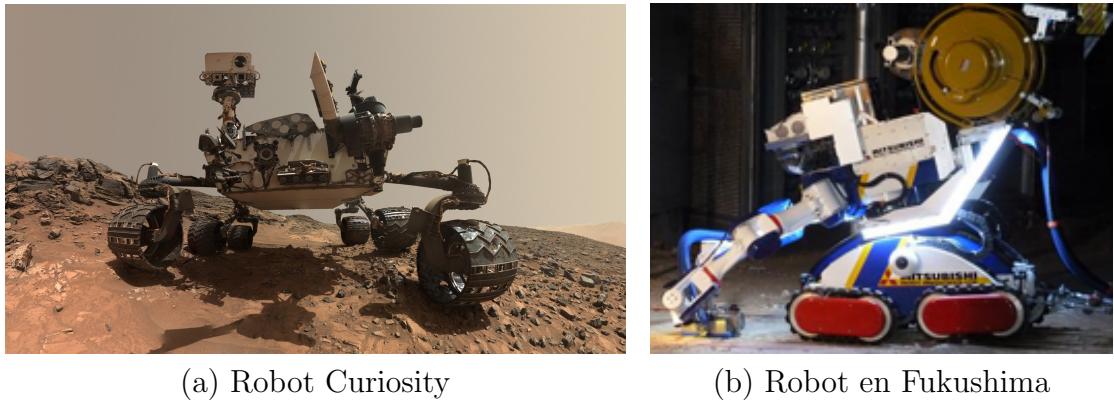


Figura 1.4: Ejemplos de la robótica en diferentes campos.

La industria del automóvil, con los coches autónomos (Figura 1.5-a), es otro de los grandes frentes de la robótica. El objetivo es que un coche sea capaz de realizar todas las tareas de conducción de forma autónoma. Como medio autónomo, es capaz de percibir el medio que le rodea y navegar en consecuencia.



Figura 1.5: Ejemplos de la robótica en diferentes campos.

1.2. Inteligencia Artificial

La robótica engloba un conjunto de diferentes disciplinas que permiten abordar todos los campos que esta comprende. Uno de los campos más importantes es

la Inteligencia Artificial (IA), que consiste en replicar los mecanismos del cerebro humano mediante algoritmos que emplean unidades equivalentes a las neuronas. Estos algoritmos van aprendiendo a medida que realizan sus tareas en base a la experiencia que van adquiriendo durante su ejecución.

Numerosos son los estudios existentes en la actualidad sobre la IA. Uno de ellos es la mejora del rendimiento y la eficiencia del reconocimiento de fracturas radiográficas a través de la IA [Guermazi et al., 2021]. Este algoritmo es capaz de detectar las fracturas en las radiografías con un menor rango de fallo y en menor tiempo que un humano. En el caso de la Figura 1.6, nueve especialistas no detectaron la fractura mientras que el algoritmo sí.



Figura 1.6: Detección de fracturas en la radiografía mediante IA

Hay dos tipos de inteligencia artificial:

- IA fuerte, compuesta por la IA general y la superinteligencia artificial. La IA general está inspirada en la teoría de que una máquina adquiera inteligencia humana, siendo capaz de resolver cualquier problema por sí misma. Por otro lado, la superinteligencia artificial supera la capacidad del cerebro humano. La IA fuerte es todavía teórica y no hay ejemplos reales de su uso.
- IA débil, entrenada para realizar tareas específicas que operan dentro de un rango previamente definido. Los sistemas dotados con esta inteligencia parecen inteligentes, pero están limitados al contexto en el que trabajan. Un ejemplo de ello es el asistente de voz Siri de Apple.

La IA abarca diferentes áreas, como la comprensión, el reconocimiento o el aprendizaje. Una de las líneas de investigación dentro de la IA es la VA, que se detalla

a continuación.

1.2.1. Visión Artificial

La visión artificial es uno de los campos más importantes de la IA cuyo objetivo es que un sistema inteligente obtenga la información en tiempo real más relevante de las imágenes que percibe. Así como la IA trata de emular un cerebro humano, la VA hace lo mismo con respecto a la visión humana, con la diferencia de que esta segunda cuenta con las experiencias aprendidas para, entre otras cosas, diferenciar los elementos que le rodean, su movimiento, distancia o tamaño. Así pues, la VA trata de asemejar el funcionamiento del ojo humano y su posterior procesamiento con el cerebro mediante una cámara y el posterior procesamiento de los datos que esta vierte.

Tiene diferentes usos, entre ellos los más fundamentales son:

- *Clasificación de imágenes (Figura 1.7-a)*. Consiste en asignar imágenes a una serie de categorías predefinidas a través de un algoritmo. Es necesario contar con una gran cantidad de datos para poder tener suficiente entrenamiento y así fallar lo mínimo posible.
- *Detección de objetos (Figura 1.7-b)*. Consiste en analizar partes de la imagen para localizar objetos señalándolos mediante cuadros limitadores. Esto se consigue mediante el entrenamiento de una gran cantidad de imágenes en las que se etiquetan diferentes tipos de objetos, obteniendo así los datos que se usarán para la detección en una imagen.
- *Segmentación de imágenes (Figura 1.7-c)*. Consiste en el enmascaramiento exacto de los píxeles que representan a los objetos en una imagen. Es un paso más en la detección de objetos, siendo capaz de separar el objeto concreto del resto de la imagen.

Un sistema dotado de VA funciona siguiendo tres niveles de operación. En primer lugar, la obtención de imágenes o vídeos mediante una cámara, las cuales son transferidas al sistema. En segundo lugar, el procesamiento de estas imágenes para poder representar correctamente los datos de interés. Este proceso consiste en un trabajo automático del sistema en el que se elimina el ruido, se reescalan las imágenes o se ajusta el contraste, entre otros, para adaptar las imágenes. Finalmente, se realiza la comprensión de imágenes, que es el paso clave para que el sistema pueda llamarse

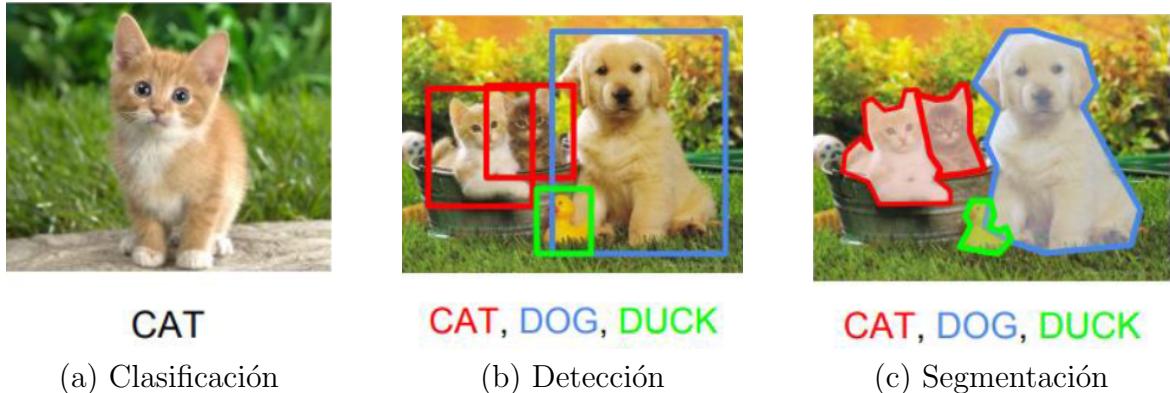


Figura 1.7: Algunos usos en la visión artificial.

inteligente, donde a través de un modelo de aprendizaje, es capaz de llevar a cabo su propósito, como puede ser detectar un determinado objeto utilizando datos aprendidos en el pasado.

Un ejemplo muy extendido de uso de la VA son los coches autónomos (Figura 1.5-b). En este caso, el funcionamiento de la VA ha de ser impecable debido a las graves consecuencias que supondría el más mínimo fallo. Con el uso de cámaras y sensores, el ordenador del coche es capaz de detectar los objetos de las imágenes que recibe, como señales o semáforos —entre otros— y actuar en consecuencia según el estado de estos. Así, el coche es capaz de tener una vista de 360º y de reconocer todos los elementos de su entorno para ser capaz de actuar como lo haría un humano.

Para que la VA funcione correctamente es necesario disponer de algoritmos precisos que sean fiables. Entre los distintos métodos que existen para entrenar estos algoritmos, uno de los más utilizados es el ML, que se detalla a continuación.

1.3. Machine Learning

Para que la IA pueda simular al cerebro humano, debe ser capaz también de aprender. Este aprendizaje se lleva a cabo a través de algoritmos. Dentro de la IA se encuentra una categoría denominada Machine Learning o aprendizaje automático, cuyo fin es que los algoritmos descubran patrones en conjuntos de datos que les hacen aprender y mejorar respecto a la experiencia anterior pudiendo así tener un aprendizaje autónomo para poder realizar una tarea sin ayuda externa.

En el sistema de aprendizaje de un algoritmo de ML hay tres partes principales. En primer lugar, un proceso de decisión, en el que una vez recibidos los datos de entrada el algoritmo genera una estimación en los datos partiendo de un patrón. Los datos de entrada pueden estar o no etiquetados; si lo están, indican al modelo lo que debe identificar. La segunda parte es una función de error, que sirve para evaluar la precisión del modelo sobre la decisión tomada. La tercera y última parte es un proceso de optimización de modelos donde las ponderaciones se ajustan en el caso de que el modelo se pueda adaptar mejor al conjunto de datos de entrenamiento, repitiendo este proceso hasta cumplir un umbral de precisión.

Hay diferentes tipos de *Machine Learning*:

- *Aprendizaje supervisado*, donde los conjuntos de datos están etiquetados. Estos conjuntos de datos sirven para entrenar los datos que servirán como experiencia y base al algoritmo a la hora de hacer una clasificación con un nuevo dato. Por tanto, usa un conjunto de datos entrenados iniciales a la hora de recibir un nuevo dato.
- *Aprendizaje no supervisado*, donde los conjuntos de datos están sin etiquetar, es decir, no cuenta con un conjunto entrenado previamente.
- *Aprendizaje semisupervisado*, que es el término medio entre los dos anteriores. En este caso, en el entrenamiento se usa un conjunto de datos más pequeño que en el aprendizaje supervisado y usa un grupo más grande de datos sin etiquetar.

1.4. Deep Learning

De la misma forma que el ML está dentro del amplio campo de la IA, el ML también tiene diferentes campos. Entre ellos está el DL (Figura 1.8). Mientras en el ML el autómata está preprogramado por un humano, en el DL el autómata aprende por sí mismo, siendo un ápice más de cómo el DL trata de simular el cerebro humano. Lo hace mediante unidades equivalentes a las neuronas.

Mientras el ML necesitaba un humano para definir previamente las características de un conjunto de datos para su clasificación, el Deep Learning no necesita intervención humana. Este aprendizaje profundo se asemeja mucho más al aprendizaje humano por tener un funcionamiento similar al de las neuronas. Este funcionamiento se denomina

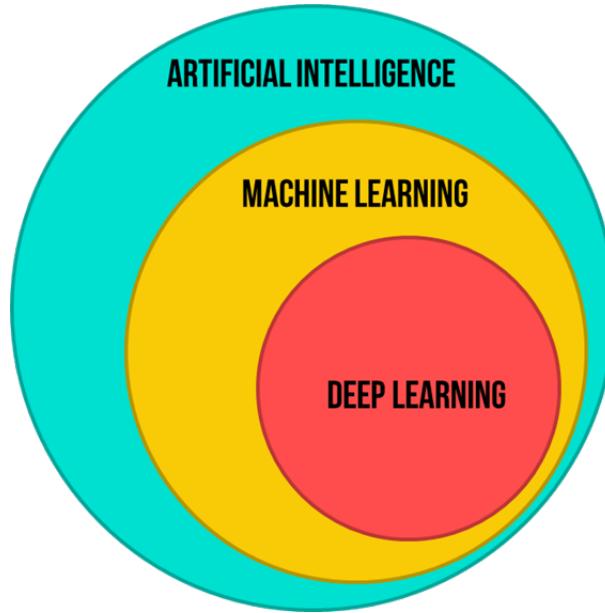


Figura 1.8: Esquema de relaciones entre IA, ML y DL.

red neuronal.

Las redes neuronales (Figura 1.9) están formadas por distintas capas de nodos interconectados, donde cada nodo se basa en su capa anterior para refinar y optimizar la precisión. Está compuesto por la capa visible, que es la primera capa, donde el modelo recibe los nuevos datos de entrada. Las otras son las capas ocultas o *hidden layers*, que son las que realizan todo el proceso de optimización hasta llegar al resultado final.

Existen multitud de estudios sobre el Deep Learning. Uno de ellos ha sido la restauración de textos antiguos con el uso de una red neuronal profunda llamada Ithaca [Assael et al., 2022] (Figura 1.10). Con ella se ha conseguido reparar textos dañados con una precisión del 62 % por sí sola. Está pensada para utilizarla con los historiadores, que han mejorado su precisión del 25 % al 72 % con esta nueva herramienta.

También existen aplicaciones de Deep Learning que se utilizan en la vida cotidiana, como los asistentes virtuales. Este es el caso de los dispositivos Siri o Alexa —entre otros— que, mediante algoritmos de Deep Learning, ayudan a los usuarios a realizar distintas tareas, aprendiendo continuamente de la información que reciben.

Otro ejemplo donde se emplean algoritmos de DL es el reconocimiento de imágenes.

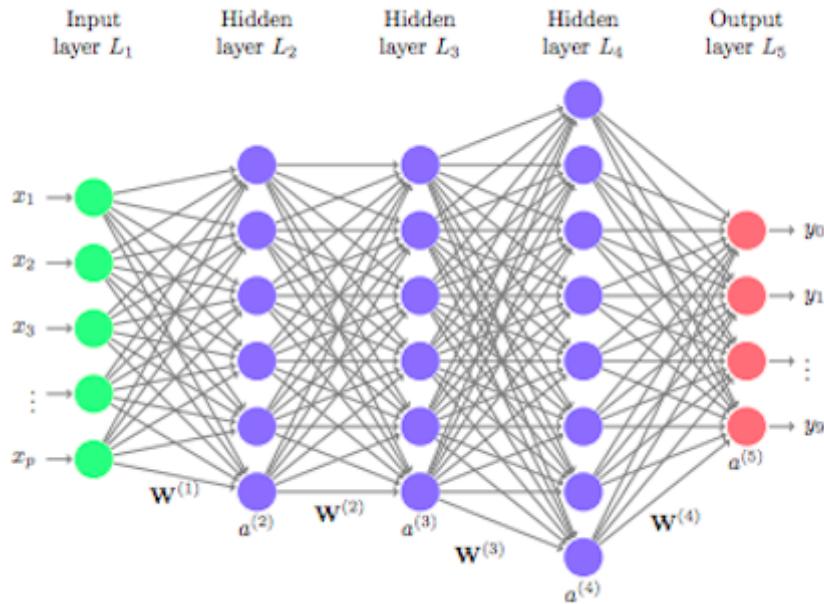


Figura 1.9: Simple ejemplo de una red neuronal en el algoritmo de Deep Learning.



Figura 1.10: Decreto sobre la Acrópolis de Atenas reconstruido con Ithaca.

Aquí se pueden encontrar numerosas aplicaciones prácticas, como por ejemplo el enfoque automático en las caras al realizar fotos (Figura 1.11).

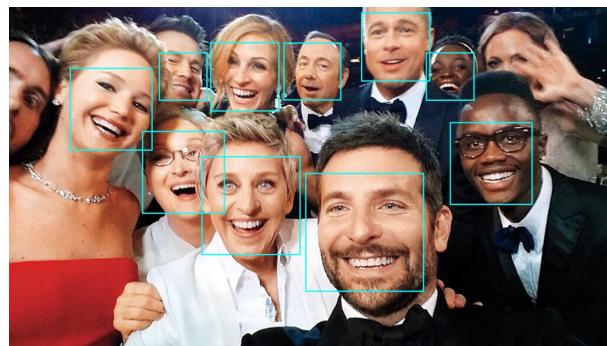


Figura 1.11: Reconocimiento facial con técnica de Deep Learning.

1.5. Sistemas multisensoriales

Como se ha explicado en las secciones anteriores, el desarrollo del DL es muy importante. Lo es especialmente en el campo de la VA, pues permite crear aplicaciones sofisticadas que simulan la visión humana. Pero si además del sensor de visión se incorporan otros tipos de sensores a un robot, el resultado de la actuación de este mejora, ya que, además de percibir la imagen del entorno, está percibiendo más información que puede ser útil a la hora de tomar decisiones. Retomando la analogía con el ser humano, si este, además de ver, es capaz de sentir, oler o tocar, recibe más información de su entorno, por lo que puede conocer mejor la situación en la que se encuentra y, con ello, actuar más inteligentemente.

Numerosas son las aplicaciones de un sistema multisensorial, por ejemplo para los sistemas propioceptivos. Estos sistemas necesitan conocer con exactitud las variables del entorno. Otra aplicación se encuentra en los robots móviles. Al disponer de múltiples sensores, estos robots reducen el error en sus movimientos o en su autolocalización. Otro ejemplo del uso de sistemas multisensoriales es el seguimiento continuado de animales. Algunos ejemplos de estos sistemas se describen a continuación.

En [UCM, 2017], de la Universidad Complutense de Madrid se describe un sistema para detectar de forma temprana las enfermedades o infecciones en animales de granja, más concretamente en cerdos. Este sistema parte de la instalación de microchips y otro tipo de sensores en los animales para poder controlar las condiciones, como su temperatura o el consumo de agua de los bebederos, de forma que un ordenador muestra esta información de manera gráfica. También se incluye una monitorización grupal mediante grabaciones diarias para que el sistema detecte cualquier tipo de actividad no común (Figura 1.12).

En [Arce et al., 2009], la Facultad de Zootecnia e Ingeniería de la Universidad de São Paulo presenta un sistema de sensores con el fin de obtener datos fisiológicos de los animales (en este caso vacas) y obtener las condiciones en las que estos tienen menos perturbaciones en su comportamiento natural, pues las condiciones climatológicas en cada región pueden afectar de diferentes maneras: estrés, pérdidas productivas o incluso la muerte. Este sistema se creó con la instalación de sensores en cada una de las vacas creando una red de sensores (Figura 1.13) que se comunican con una estación para obtener todos los datos del conjunto de animales.

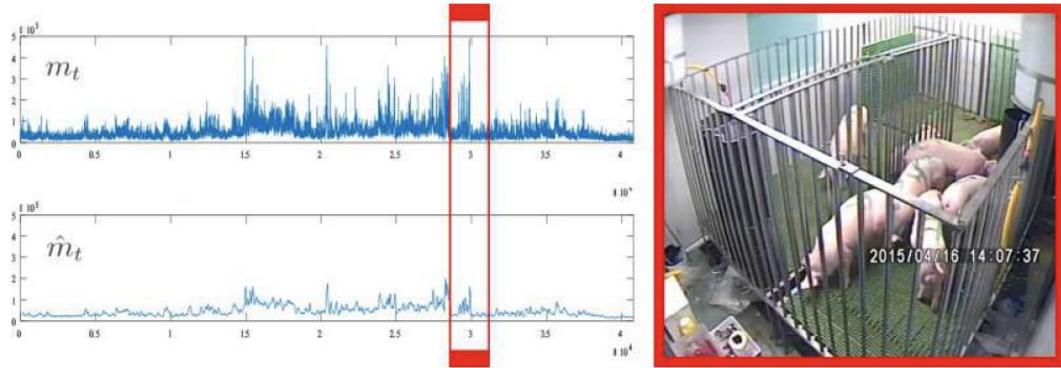


Figura 1.12: Patrón de movimientos e imagen grabada del grupo de cerdos.

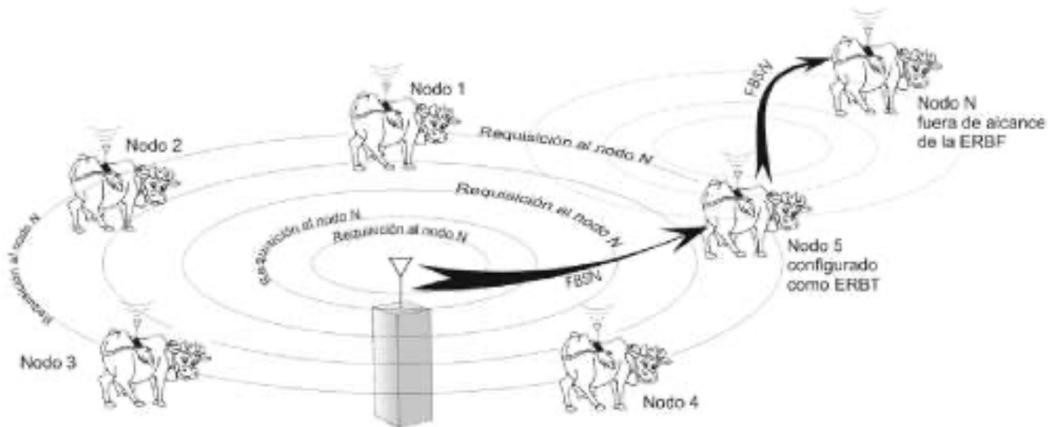


Figura 1.13: Esquema del desarrollo del sistema mediante sensores.

El comportamiento que tienen los animales puede ofrecer mucha información acerca de su salud si se mantiene una observación constante, pero también tienen una gran importancia las condiciones del entorno en la que se encuentran, ya que pueden ser decisivas para detectar el motivo de su comportamiento. Por ello, es importante tener un seguimiento constante y automatizado de estas características, permitiendo al humano encargarse únicamente de controlar los datos que el sistema registra.

El presente trabajo se enmarca en el contexto de los sistemas multisensoriales, concretamente en los sistemas multisensoriales destinados al bienestar animal y al análisis de comportamiento, en el cual juega un papel fundamental las técnicas de

DL.

En los próximos capítulos se describe el sistema desarrollado. En el Capítulo 2 se comentan los objetivos, requisitos y metodología del trabajo. En el Capítulo 3 se explican las plataformas de desarrollo que se han utilizado para desarrollar el trabajo. En el Capítulo 4 se describe el proceso exhaustivo que ha llevado el trabajo y finalmente, en el Capítulo 5 se escriben las conclusiones.

Capítulo 2

Objetivos

En el capítulo anterior se ha dado el contexto del trabajo; en este, se presenta el plan de trabajo, definiendo los objetivos tanto generales como específicos que se han marcado para desarrollarlo y los requisitos que debe respetar el proyecto. Posteriormente se explica la metodología utilizada para cumplir con los objetivos.

2.1. Descripción del problema

El objetivo general de este proyecto es crear un sistema multisensorial para la monitorización y seguimiento de animales de laboratorio, que además sea económicamente accesible para cualquier usuario. Este sistema incluirá diferentes sensores para medir las condiciones del entorno en la que se encuentran los animales en un laboratorio —en concreto, ratones— y mostrarlo en una interfaz gráfica en tiempo real que sea fácilmente manejable por cualquier usuario. Por otro lado, el sistema también deberá ser capaz de registrar el tiempo en el que los animales realizan sus diferentes tareas.

Actualmente, el control de estas características se suele realizar en la mayoría de casos por humanos, sin ningún sistema automático de apoyo; entre otras razones, por el elevado coste de los sistemas existentes en el mercado equivalentes al que se pretende desarrollar en este trabajo.

Para cumplir el objetivo general marcado; es necesario establecer los siguientes objetivos específicos:

1. Recoger la lectura de los sensores en un mismo fichero. Cada sensor necesita de las librerías pertinentes para su correcto funcionamiento, por lo que se creará una

clase por cada sensor, de manera que sea más fácil unificar las distintas lecturas.

Para ejecutar concurrentemente la lectura de todos los sensores, se utilizará la librería *Thread* que permite la ejecución y el control de distintos hilos lanzados en paralelo.

2. Crear un servidor web para los dos sensores que recogen imágenes. Para la cámara térmica, la única función del servidor será mostrar la imagen que graba la cámara. Sin embargo, para la cámara normal, además de mostrar las imágenes, se deberá incorporar un botón que permita iniciar y parar la grabación cuando el usuario quiera, guardando el vídeo automáticamente en el sistema. Estos servidores estarán protegidos para aportar seguridad.
3. Detectar los diferentes ratones mediante un algoritmo de reconocimiento de objetos a través de TensorFlowLite, para así poder determinar el tiempo que pasan los animales haciendo las diferentes actividades.

2.2. Requisitos

El trabajo cumplirá la siguiente serie de requisitos:

- El sistema deberá poder ejecutar en tiempo real sobre la plataforma Raspberry Pi 4B, resultando así un sistema de bajo coste.
- El lenguaje de programación será Python, pues este lenguaje es sencillo y ofrece una gran variedad de librerías útiles para este trabajo y está soportado completamente por el sistema operativo oficial de la placa Raspberry.
- La interfaz de usuario (IU) será creada con Node-Red, ya que es una herramienta muy visual y está creada para sistemas embebidos como Raspberry. Además permite el acceso a la interfaz desde distintos dispositivos.

2.3. Metodología

Para la satisfacción de los objetivos y el cumplimiento de los requisitos mencionados anteriormente, se han usado diferentes herramientas para el correcto control, seguimiento y desarrollo del proyecto.

Para el seguimiento del trabajo se han llevado a cabo reuniones semanales con el tutor del trabajo a través de la plataforma Microsoft Teams, donde se compartían los

problemas surgidos durante la semana junto a posibles soluciones que evaluar, además del control y evaluación de los objetivos marcados la semana anterior. También se establecían los nuevos objetivos para la próxima semana. Asimismo, se ha utilizado el correo electrónico para comentar y aclarar problemas que surgían a lo largo de la semana.

Paralelamente, se ha contactado con los investigadores del Laboratorio del Bienestar e Investigación Animal¹ de la Universidad de Alcalá de Henares tanto por videoconferencia como por correo electrónico, para conocer la situación y la prioridad de los problemas reales que tienen y que podrían solventar con el sistema planteado.

Todo el código desarrollado, así como los distintos recursos empleados y la presente memoria, se encuentran alojados en el repositorio de GitHub² dedicado a este trabajo. Por otro lado, en la wiki³ se ha ido escribiendo el proceso que se ha llevado a cabo para realizar el trabajo, junto a los problemas que han ido surgiendo acompañados de imágenes o vídeos, así como las soluciones que han servido para solventarlos. También se han aportado fotos y vídeos mostrando el funcionamiento de las distintas partes del sistema, así como el funcionamiento entero de este.

¹<https://www.uah.es/es/investigacion/unidades-de-investigacion/grupos-de-investigacion/Bienestar-en-Investigacion-Animal-Welfare-on-Animal-Research./>

²<https://github.com/jmvega/tfg-icebollada>

³<https://github.com/jmvega/tfg-icebollada/wiki>

Capítulo 3

Plataforma de desarrollo

En este capítulo se definen tanto la infraestructura utilizada como los métodos, tanto a nivel software como hardware, que se han utilizado para desarrollar este trabajo.

3.1. Plataforma hardware

Para satisfacer el objetivo de crear un sistema multisensorial con las características definidas en el capítulo anterior y que además sea *lowcost*, la infraestructura hardware en este trabajo se ha centrado en sistemas embebidos empotrados existentes en el mercado. Dos de los sistemas más importantes son Raspberry (Figura 3.1-a) y Arduino (Figura 3.1-b). La decisión de trabajar con Raspberry en lugar de Arduino ha estado motivada por diferentes motivos. Aunque ambas placas disponen de pines GPIO para la conexión de sensores o actuadores, Arduino no permite la conexión de cámaras, mientras que en Raspberry no solo se puede acoplar su cámara (PiCam) sino que permite conectar una webcam a través de uno de los puertos USB que tiene la placa. Arduino es un microprocesador que cuenta con su IDE, mientras que Raspberry es un microordenador completo que cuenta con su propio sistema operativo Raspbian. De esta forma el usuario final puede visualizar el resultado con la conexión de la Raspberry a una pantalla a través de su puerto HDMI.

A la placa Raspberry se han conectado una serie de sensores para obtener distintos valores necesarios para la monitorización del sistema que se presentan a continuación.

Para la obtención de la temperatura se han utilizado el sensor BME680 (Figura 3.2) y el sensor DS18B20 (Figura 3.3). Este segundo sensor es resistente al agua, por lo que permite recoger la temperatura en superficies húmedas o mojadas. El rango en el que trabaja el BME680 así como su resolución se encuentra en el cuadro 3.1 extraído de la

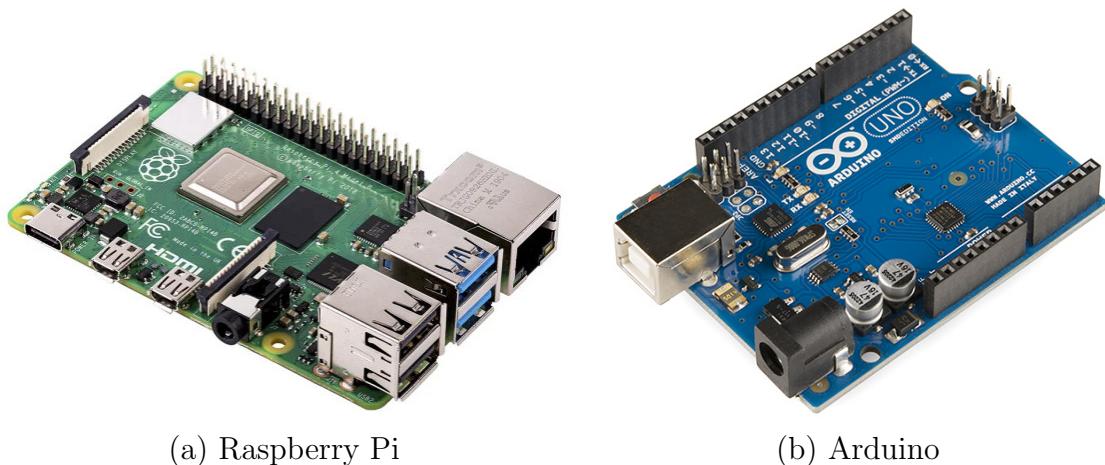


Figura 3.1: Sistemas empotrados.

ficha de datos del sensor. Las características del DS18B20 se encuentran en el cuadro 3.2.



Figura 3.2: Sensor BME680.

Parametros	Min	Tipo	Max	Unidad
Rango de temperatura de funcionamiento	-40		80	°C
Resolución de salida temperatura		0.01		°C
Rango de presión de funcionamiento	300		1100	hPa
Resolución de salida presión		0.18		hPa
Rango de humedad de funcionamiento	0		100	% r.H.
Resolución de salida humedad		0.005		% r.H.

Cuadro 3.1: Cuadro de características del sensor BME680.



Figura 3.3: Sensor DS18B20.

Parámetros	Min	Max	Unidad
Rango de temperatura de funcionamiento	-55	125	°C

Cuadro 3.2: Cuadro de características del sensor DS18B20.

Para la lectura de la humedad, la presión y la calidad del aire se ha utilizado el sensor BME680 (Figura 3.2), pues es capaz de medir más de un parámetro. La tabla de características se encuentra en el cuadro 3.1.

Para el registro de imágenes térmicas, se han utilizado dos sensores. Uno de ellos es el sensor AMG8833 (Figura 3.4-a) que ofrece una matriz de valores de temperatura. El segundo es la cámara Seek Thermal (Figura 3.4-b) que se ha conectado a la Raspberry a través de su puerto USB por medio de un adaptador.



(a) Sensor AMG8833.



(b) Cámara Seek Thermal

Figura 3.4: Sensores térmicos utilizados en el trabajo.

Para el registro de imágenes se ha utilizado una de las cámaras de Raspberry, la Pi

Camera (Figura 3.5). Ofrece una resolución de 8 megapíxeles, siendo una cámara de alta definición que ofrece poco ruido y alta sensibilidad.



Figura 3.5: Pi Camera.

Otro sensor utilizado ha sido el sensor de nivel de agua (Figura 3.6), que permite detectar la presencia de agua. También es posible obtener una idea de la cantidad de agua presente, aunque no de forma precisa.



Figura 3.6: Sensor de nivel de agua.

El último sensor integrado en el trabajo ha sido el MQ-135 (Figura 3.7), que permite detectar la concentración de diferentes gases como el alcohol, benceno, humo, dióxido de carbono o amoníaco. En concreto, este sensor se ha utilizado para la detección de amoníaco.

Para conseguir el correcto funcionamiento de los sensores en la placa Raspberry, se ha utilizado la infraestructura software descrita a continuación.



Figura 3.7: Sensor MQ-135.

3.2. Infraestructura software

Para dar soporte a la placa Raspberry se ha usado su sistema operativo oficial Raspberry Pi OS, también llamado Raspbian (Figura 3.8). La elección de usar este Sistema Operativo (SO), entre otras, es que está optimizado para funcionar en procesadores ARM, que es el que tiene Raspberry. Además, Raspberry Pi OS es el SO que mejor rendimiento ofrece para la placa.

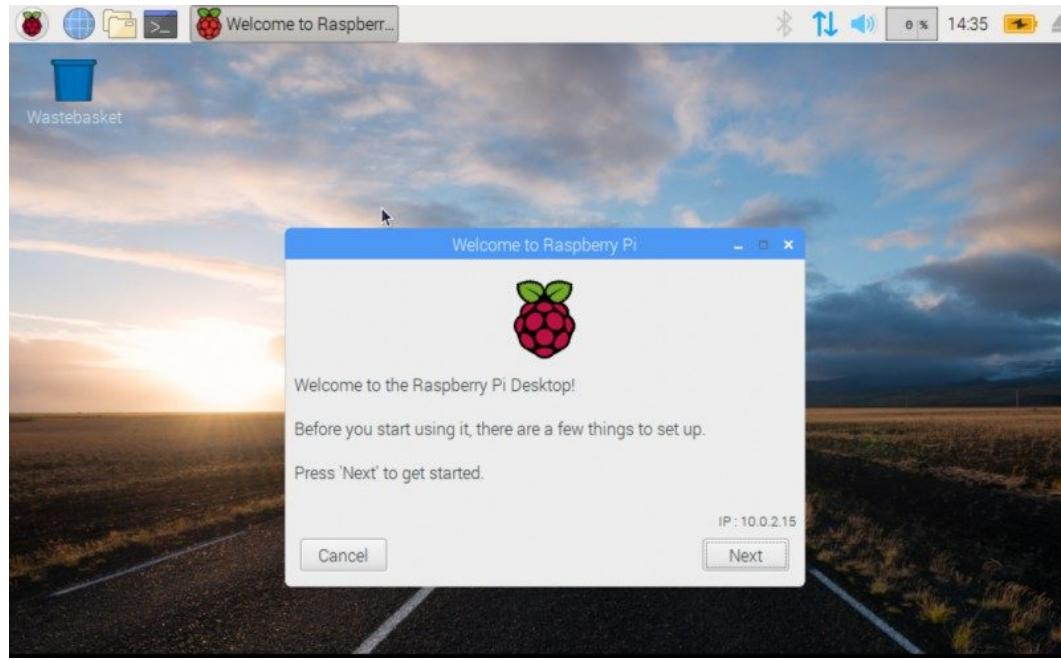


Figura 3.8: Imagen de Raspberry Pi OS.

A continuación se presentan las herramientas utilizadas para el desarrollo y funcionamiento del presente trabajo.

3.2.1. Lenguaje Python

Python es un lenguaje de alto nivel de programación, interpretado y orientado a objetos cuya filosofía hace hincapié en la legibilidad de su código y, por ello, su sintaxis es sencilla. Ofrece numerosos módulos y librerías para las distintas aplicaciones. Algunas de estas librerías son: TensorFlow para aplicaciones en ML (Código 3.1), Pandas para análisis de datos, Flask para aplicaciones web o SQLAlchemy para comunicación entre bases de datos y programas. El uso de estos módulos y librerías facilita la reusabilidad de código y la programación modulada. No requiere del proceso de compilación, lo que lo convierte en un lenguaje muy rápido en el ciclo de edición, prueba y depuración.

```

train_data =
    object_detector.DataLoader.from_pascal_voc('split_train_validate/train',
                                                'split_train_validate/train', ['mouse'])
validation_data =
    object_detector.DataLoader.from_pascal_voc('split_train_validate/validate',
                                                'split_train_validate/validate',
                                                ['mouse'])
model = object_detector.create(train_data, model_spec=spec, epochs=20,
                               batch_size=8, train_whole_model=True, validation_data=validation_data)

```

Código 3.1: Código para generar un modelo de detección de ratones en Python

Actualmente es el lenguaje de programación más usado en el mundo según la calificación de la empresa TIOBE¹, por lo que cuenta con una gran comunidad. Además, también es el más popular en el ámbito del Machine Learning. Algunas de las aplicaciones que usan Python son Google, Netflix, Dropbox o Spotify.

La decisión de usar Python para el desarrollo de este TFG ha sido que es uno de los lenguajes de programación que mejor funciona y está soportado de forma nativa en el SO utilizado Raspbian, además de las numerosas librerías útiles para este proyecto adaptadas a Raspberry que Python posee.

En este trabajo, Python se utiliza para la lectura de los sensores de forma concurrente, para la creación de los dos servidores web que tiene tanto la PiCamera como la cámara térmica con Flask y para el reconocimiento de los ratones a través de la librería TensorFlow Lite.

¹<https://www.tiobe.com/tiobe-index/>

3.2.2. TensorFlow Lite

TensorFlow Lite (Figura 3.9-a) es una variación de TensorFlow (Figura 3.9-b) más ligera adaptada a dispositivos como teléfonos móviles, microcontroladores o dispositivos embebidos como Raspberry. Es una plataforma de código abierto multiplataforma que permite entrenar un modelo para su posterior uso. Algunos de estos usos pueden ser la clasificación o el reconocimiento de imágenes, entre otros.



(a) Logo de TensorFlow Lite.

(b) Logo de TensorFlow

Figura 3.9

Las características que hacen que pueda utilizarse en este tipo de dispositivos son las siguientes:

- Ligereza, ya que estos dispositivos tienen límite tanto en el almacenamiento como en la capacidad de cómputo.
- Baja latencia, ya que las inferencias se realizan en el dispositivo y no en un servidor externo.
- Seguridad, debido a que el modelo viene implementado en el propio dispositivo.
- Consumo de energía óptimo, dado a que no es necesario que el dispositivo esté conectado a la red, que consume mucha energía.

TensorFlow Lite se ha usado, en primer lugar, para la creación y entrenamiento de un modelo capaz de detectar ratones en una imagen (Figura 3.10), basándose en un dataset de 60 imágenes. En segundo lugar, para la detección de ratones en cualquier imagen o vídeo, siendo capaz de detectar el número de ratones y el lugar en el que se encuentra cada uno de ellos. Para conseguir esta detección en tiempo real del vídeo grabado por la cámara se requiere el uso de otra librería llamada OpenCV, que es

descrita a continuación.



Figura 3.10: Detección de ratones en una imagen usando TensorFlow Lite.

3.2.3. OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de software de visión computacional y machine learning de código abierto. Es ampliamente usada en todo el mundo debido al soporte multiplataforma que ofrece para Windows, Linux, MacOS y Android, además de ofrecer interfaz en diferentes lenguajes como Python, Java, C++ y MATLAB.

Compuesto por más de 2500 algoritmos, OpenCV está especializado en la visión computacional y en algoritmos de aprendizaje, permitiendo reconocer rostros, identificar o clasificar objetos, extraer modelos 3D, mejorar la calidad de las imágenes o detectar bordes (Figura 3.11), entre otros.

OpenCV se ha usado en este trabajo para manipular en tiempo real los vídeos grabados por las cámaras, permitiendo —entre otras cosas— integrar un *timestamp* en estos. También ha permitido ofrecer la funcionalidad de guardar los vídeos cuando el usuario lo solicite a través del interfaz. Por último, y dada su fácil adaptación a otras aplicaciones, ha sido posible mostrar los vídeos de ambas cámaras (térmica y RGB) en el servidor web de Flask sin problema. Esta librería se detalla a continuación.

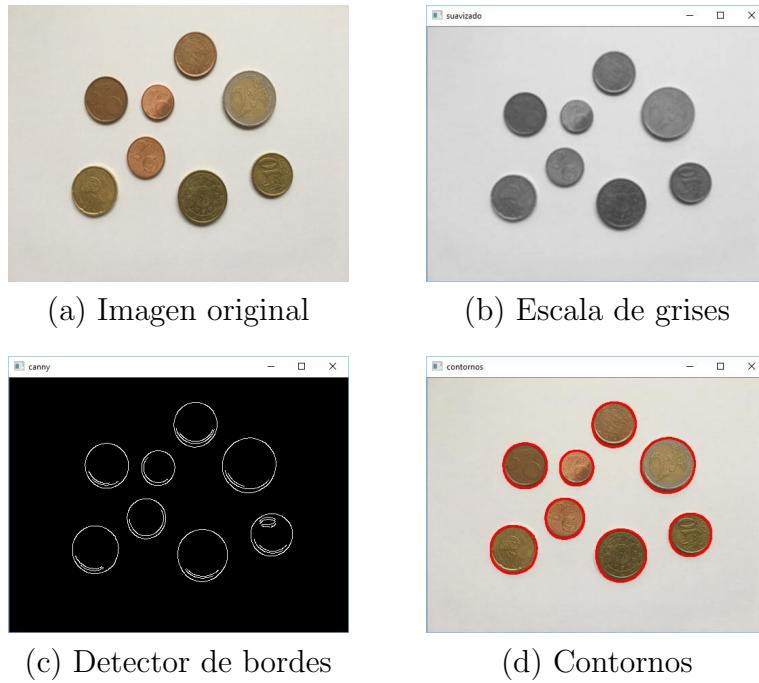


Figura 3.11: Ejemplo de detección de bordes con OpenCV.

3.3. Flask

Flask es un marco de aplicación web minimalista —en inglés, *microframework*— escrito en Python. Esto significa que ofrece al usuario herramientas y librerías para crear una aplicación web, y al ser minimalista, no requiere de otras dependencias para realizar las cosas básicas. Ofrece una serie de extensiones que permiten dotar de más características a la aplicación, como autenticación o manejo a través de comandos, entre otros.

Se compone de dos partes: las plantillas —en inglés, *templates*—, que están escritas en lenguaje HTML e indican la organización y diseño que tendrá cada página al ser visualizada, y el código en Python, que indica las rutas de cada página y las funciones de cada ruta. Este es el fichero que se ejecuta para crear el servidor.

En este proyecto se ha utilizado Flask para la creación de los dos servidores web de las cámaras. Se ha preferido Flask frente a Django, que es el más conocido para el uso con Python, debido a que el primero es más sencillo de utilizar y de aprender. Entre otras, este servidor permite realizar funciones como loguearse y registrarse en él o acceder y controlar la visualización de las cámaras. En la Figura 3.12 se puede ver un ejemplo de un trabajo realizado con Flask.

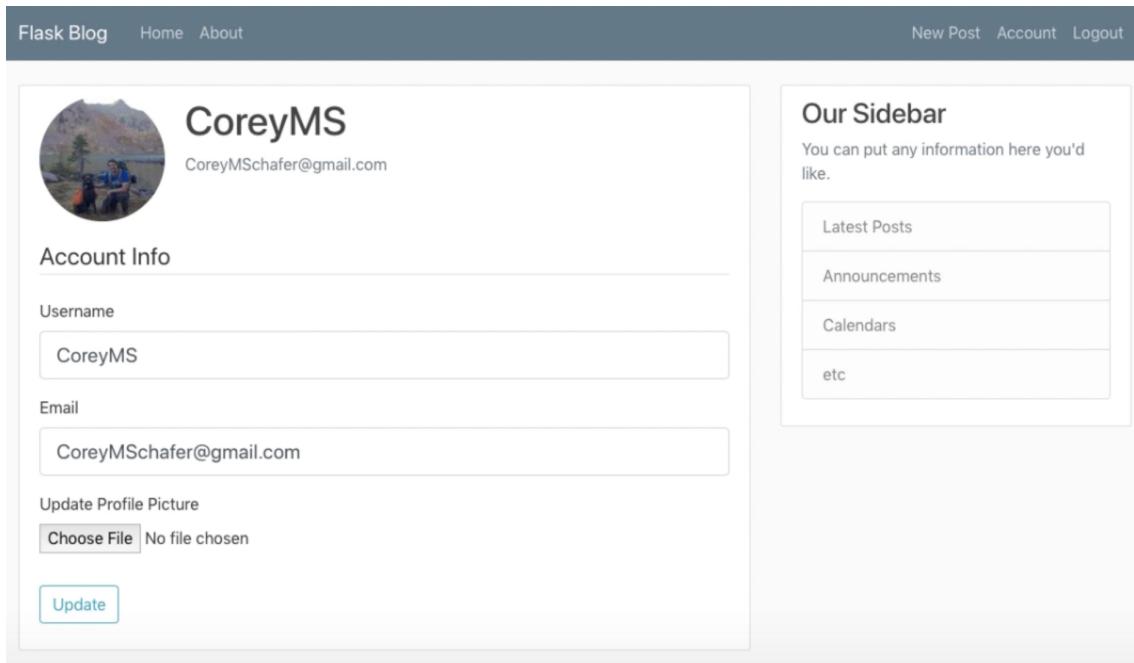


Figura 3.12: Blog creado con Flask.

3.3.1. Node-Red

Node-Red es una herramienta de programación creada por IBM y escrita en lenguaje JavaScript que permite conectar dispositivos hardware y servicios en línea. Muestra de manera visual las conexiones de los nodos en un panel, mostrando gráficamente el flujo de la información. Además, es un entorno de ejecución ligero, lo que lo hace ideal para ejecutarse en hardware de bajo coste como la Raspberry.

Está compuesto por dos partes principales. Una de ellas es la edición de flujo desde navegador, donde muestra los nodos disponibles en el margen izquierdo así como la disposición y conexión del flujo creado por el usuario. La otra parte es el cuadro de mando, más comúnmente *dashboard*, que muestra la interfaz de usuario en base al flujo de los nodos de la parte previamente comentada.

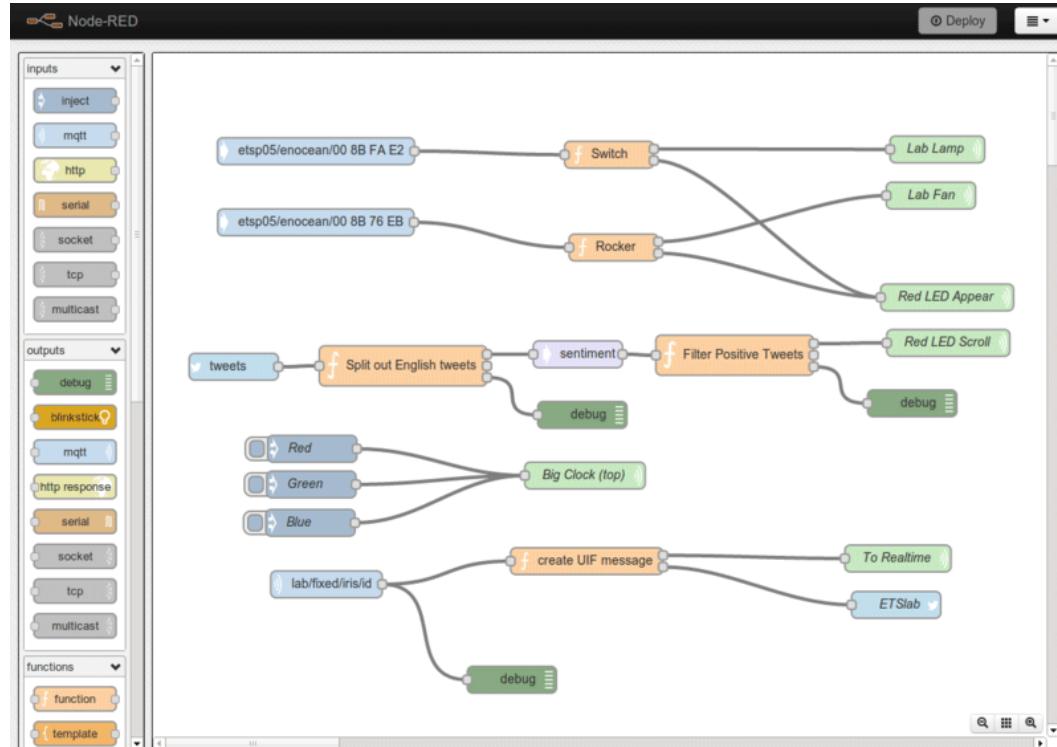
Node-Red tiene una amplia variedad de nodos. Algunos de ellos permiten distintos tipos de conexiones, la interacción directa con los pines de Raspberry, la conexión con twitter o el correo electrónico. También permite la creación de ficheros de distintos tipos o la ejecución de ficheros ya existentes en la computadora.

Node-Red también permite el modo *scripting*; esto es, la ejecución directa de código escrito en lenguaje JavaScript. Pueden crearse nuevos nodos para agregar nuevas capacidades gracias a los más de 225.000 módulos que hay en el repositorio de paquetes.

Los flujos de datos se pueden importar o exportar con ficheros en formato estándar JSON, permitiendo compartirlos con cualquier persona. La página de Node-Red tiene una librería de flujos que han creado otros usuarios, permitiendo el acceso a ellos para utilizarlos como base en otros proyectos (Figura 3.13). En la figura 3.14 se puede ver un ejemplo de un trabajo realizado en Node-Red.



(a) Flujo de Hola Mundo en Node-Red



(b) Flujo del proyecto de creación de un sistema de alarma

Figura 3.13: Distintos flujos hechos en Node-Red

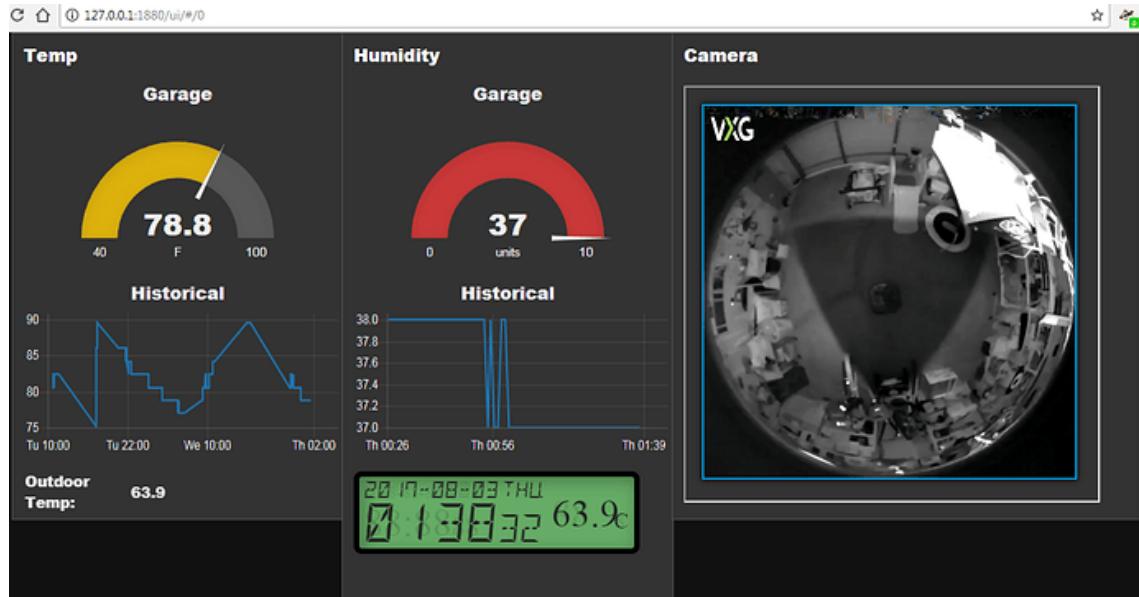


Figura 3.14: IU de un sistema de control de un garaje.

Para el desarrollo de este TFG, Node-Red ha tenido un papel muy importante. Además de tener una gran comunidad que permite la resolución de la mayoría de dudas, ha permitido la creación de la interfaz de usuario donde se muestran las mediciones de cada sensor, los dos servidores web de las cámaras y los botones e interruptores necesarios que permiten al usuario interactuar con la interfaz de una manera sencilla para obtener toda la información.

Capítulo 4

Sistema multisensorial

Este capítulo consta de la descripción detallada del proceso que se ha llevado a cabo para crear el sistema multisensorial para la monitorización de animales de laboratorio. Como se ha introducido previamente, la base idea de este trabajo ha sido pensada para un laboratorio de animales. Debido al desconocimiento de robótica de los trabajadores de este sector, puede haber trabajo que se realice en el laboratorio que podría mejorarse e incluso automatizarse.

Para obtener más información, se ha contactado con el Laboratorio del Bienestar e Investigación Animal de la Universidad de Alcalá de Henares¹ para conocer la situación en la que trabajan. Se han apreciado bastantes cosas mejorables y se ha enfocado el objetivo en la monitorización de los ratones del laboratorio, que requieren una observación constante así como del entorno en el que se encuentran debido a que deben estar bajo unas condiciones determinadas.

Asimismo, para facilitar la comprensión de los datos, se ofrece una interfaz gráfica donde los valores numéricos obtenidos por los sensores, así como cualquier interacción necesaria en la IU se traducen en widgets —pequeñas imágenes que proveen información visual— comprensibles para cualquier usuario de una manera intuitiva y sencilla.

Dos son las partes esenciales para el desarrollo: hardware y software. En la primera sección se presenta el desarrollo hardware y en la sección posterior el desarrollo software que se ha realizado para tener un funcionamiento correcto del sistema.

¹<https://www.uah.es/es/investigacion/unidades-de-investigacion/grupos-de-investigacion/Bienestar-en-Investigacion-Animal-Welfare-on-Animal-Research./>

4.1. Desarrollo hardware

La placa que se ha utilizado para el proyecto es la Raspberry Pi 4B (Figura 4.1), el último modelo de Raspberry y muy utilizada para el desarrollo de proyectos debido a su bajo coste. Este modelo es el más rápido y potente, ofreciendo de dos a tres veces el rendimiento del procesador de su modelo predecesor. El sistema operativo utilizado para el desarrollo del proyecto ha sido Raspbian.



Figura 4.1: Raspberry Pi 4B utilizada para el desarrollo del presente TFG.

Está dotada de una serie de pines y puertos que permiten su conexión con distintos sensores o actuadores. Para este trabajo se han utilizado los sensores presentados en el capítulo 3.1. A continuación se explican con más detalle estos sensores:

- Sensor BME680 (Figura 3.2). Este sensor es capaz de medir hasta cuatro parámetros: el gas, la temperatura, la presión y la humedad.
- Sensor AMG8833 (Figura 3.4-a). Este sensor es una cámara térmica que permite recibir la información del entorno y mostrarla a través de un rango de colores. Los colores más cálidos —como el rojo— indican la presencia de mayor temperatura, mientras que los colores fríos —como el azul— indican una temperatura inferior. Los valores que ofrece este sensor es una matriz de 8x8 valores de temperatura, que deben ser traducidos para poder visualizar la imagen de temperaturas (Figura 4.2).
- Sensor DS18B20 (Figura 3.3). Este sensor permite medir la temperatura en superficies húmedas, ya que es resistente al agua.

- MQ-135 (Figura 3.7). Este sensor permite medir la calidad de distintos gases, entre ellos el de amoníaco. Es muy importante tener controlada la cantidad de amoníaco en el entorno de los ratones. Este sensor también realiza lecturas analógicas, por lo que es necesario el uso de un conversor analógico a digital.
- Sensor nivel de agua (Figura 3.6). Con este sensor se puede medir la presencia de agua, así como la cantidad de agua que hay. Este sensor tiene lecturas analógicas, por lo que es necesario el uso de un conversor analógico a digital.
- Seek thermal (Figura 3.4-b). Este sensor es una cámara térmica. Permite obtener imágenes representando las temperaturas con colores de la misma forma que el sensor 3.4-a, pero con mucha más calidad y precisión.
- PiCamera (Figura 3.5). Este sensor es una de las cámaras oficiales de Raspberry, que permite obtener imágenes del entorno y mostrarlas en pantalla.

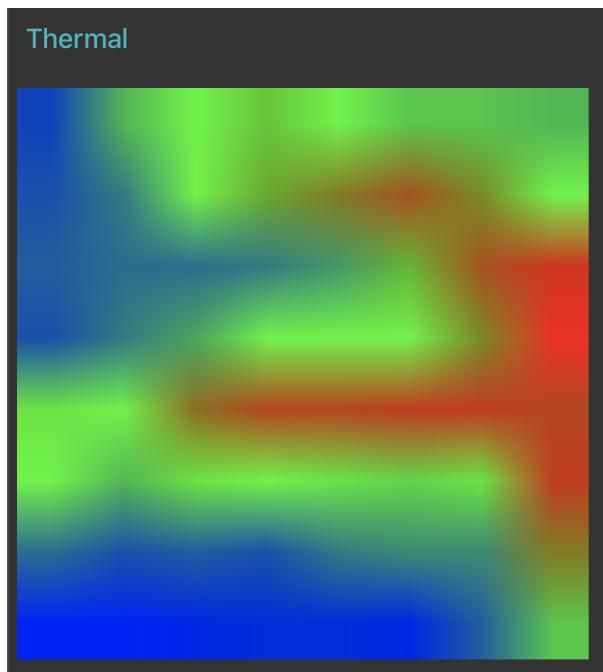


Figura 4.2: Imagen generada después de la conversión de los datos numéricos.

Raspberry no tiene lecturas analógicas, por lo que ha sido necesario el uso de un conversor analógico a digital (el MCP3008) para la lectura del sensor MQ-135 3.7 y el sensor de nivel de agua 3.6. Además, se han utilizado otros componentes como son resistencias o leds para el correcto funcionamiento de los sensores y proporcionar al usuario mayor comprensión del funcionamiento del sistema.

En la figura 4.3 se puede ver un esquema de los sensores utilizados integrados en la Raspberry.

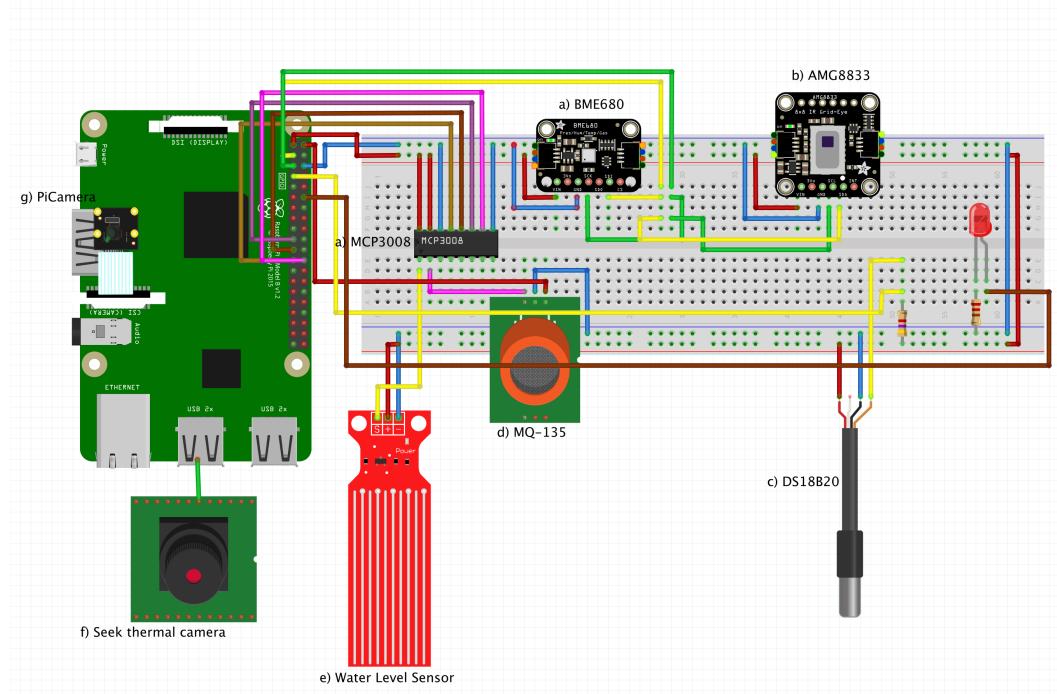


Figura 4.3: Esquema de conexiones.

4.2. Desarrollo software

Tras disponer de la parte hardware, el equipo está preparado. Para que este equipo funcione, es necesario crear las instrucciones y reglas a través de programas o aplicaciones, esto es, la parte software.

Como se ha comentado previamente, se han mantenido reuniones con el laboratorio así como reuniones semanales con el tutor. De estas reuniones se ha obtenido la información necesaria para realizar el proyecto. Además, ha permitido extraer el diagrama de casos de uso (Figura 4.4).

En base al diagrama de casos de uso, se debe crear un fichero que permita obtener todas las lecturas de los distintos sensores. Para proporcionar abstracción y mejorar el código, por cada sensor debe crearse una clase. Esto ha permitido crear el diagrama de clases visible en la Figura 4.5 para ser implementado.

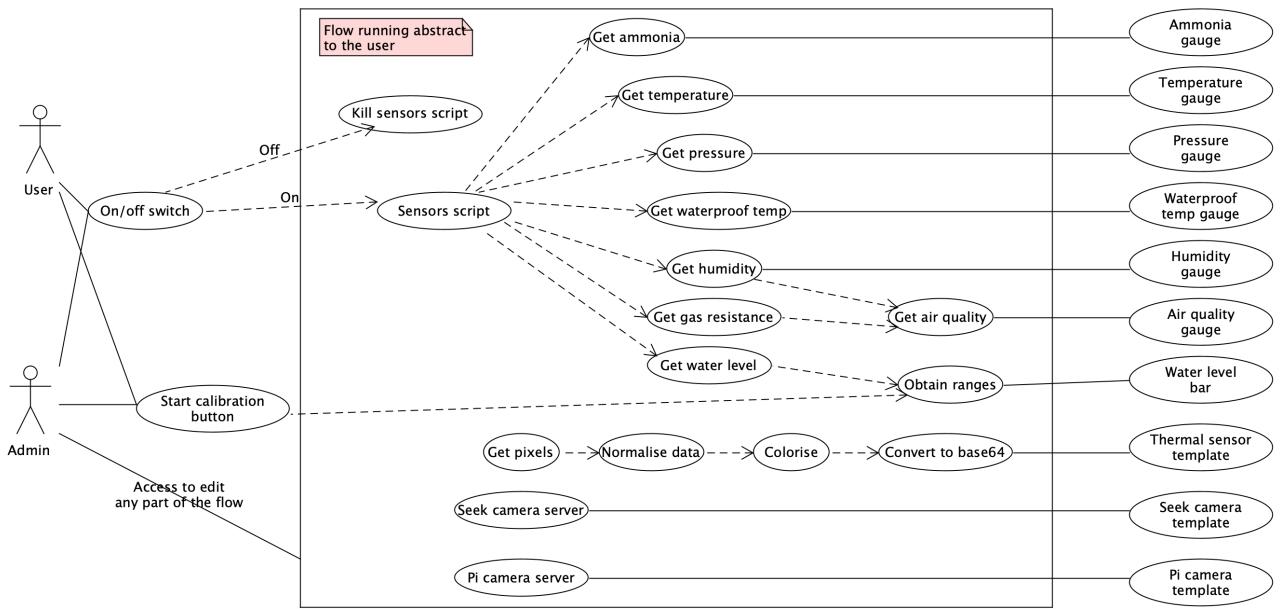


Figura 4.4: Diagrama de casos de uso del proyecto.

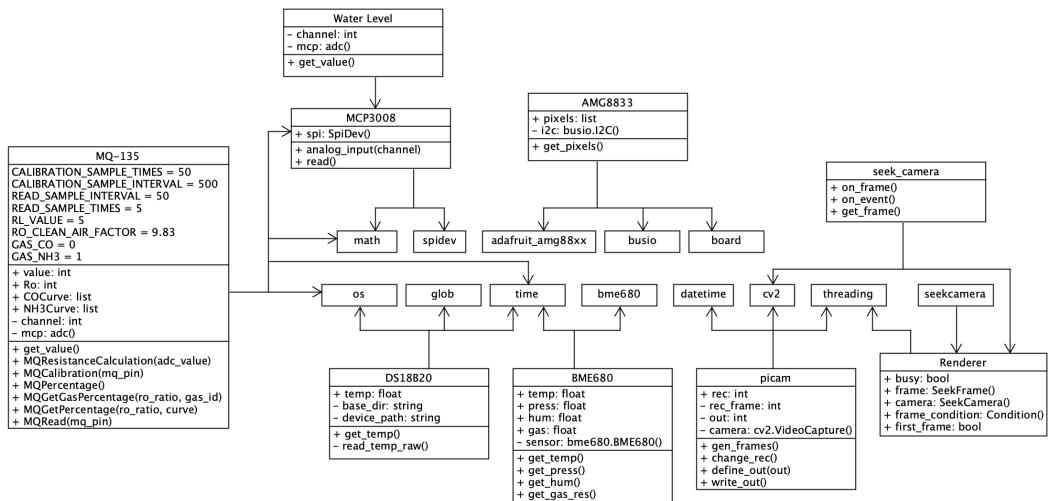


Figura 4.5: Esquema de clases de cada sensor hecho en UMLet.

En primer lugar ha sido necesaria la instalación de las librerías necesarias para el funcionamiento de todos los sensores. Después de la instalación, han surgido varios problemas al intentar leer de los distintos sensores. Uno de los problemas encontrado en dos sensores —los que funcionaban con conexión i2c— ha sido que la Raspberry no los detectaba. Se pueden encontrar los errores que han surgido en la instalación y las respectivas soluciones para hacer funcionar todos los sensores en la wiki ² que se ha creado para realizar el proyecto.

²<https://github.com/jmvega/tfg-icebollada/wiki/2.February-progress>

Durante este proceso ha habido dos sensores que se han intentado utilizar sin éxito debido a que no tienen compatibilidad con Raspberry: CCS811, que mide la calidad del aire y AM2315, que mide la temperatura y humedad. Aun así, esto no ha sido un problema ya que son datos que se han podido obtener con otros sensores disponibles.

4.2.1. Lectura sensorial con Python

Se ha considerado que el usuario no tiene conocimiento para comprender los datos en crudo que ofrece un sensor, por tanto, ha sido necesario realizar diferentes cambios para ofrecer una comprensión fácil al usuario. Se ha utilizado el programa Node-Red para ofrecer una interfaz al usuario, de forma que este sea capaz de ver la información del estado del sistema representada a través de widgets. Así ha sido posible crear un interfaz ameno e intuitivo basado en el diagrama de casos de uso presentado anteriormente (Figura 4.4), permitiendo que el usuario comprenda la información a través de algo más visual que valores numéricos.

Para poder mostrar todos los sensores en tiempo real en Node-Red, ha sido necesario disponer de un fichero de Python que cada vez que se ejecute lea una sola vez de todos ellos. No debe leer continuamente ya que es la configuración de Node-Red la que se encarga de ello. En este fichero se procesan los sensores menos el AMG8833 3.4-a, la PiCam 3.5 y la cámara térmica 3.4-b. Para crear este fichero se ha creado una clase por cada sensor, tal y como se estableció en el diagrama de clases (Figura 4.5).

Este fichero de Python utiliza la librería Threads para poder leer de manera concurrente todos los sensores, ahorrando tiempo y ganando eficacia. El código 4.1 muestra cómo se crea un hilo —o thread— por cada sensor.

Además, este programa guarda las lecturas en un fichero CSV por si fuese necesaria la comparación o estudio de datos a lo largo del tiempo. El fichero devuelve una cadena con la lectura de cada sensor (Código 4.2), para poder procesar la salida en Node-Red.

4.2.2. Creación de la interfaz de usuario

Una vez creado este fichero, se ha procedido a crear la interfaz en Node-Red. Este programa funciona a través de diferentes tipos de nodos, que deben ser combinados

```

threads = []

#Lista de funciones que devuelven la lectura del sensor respectivo
for func in [x.ammo, x.bme, x.lev, x.water_t]:
    threads.append(Thread(target=func))
    threads[-1].start()

for thread in threads:
    thread.join()

```

Código 4.1: Función para crear un Thread por sensor y obtener su lectura.

```

Ammonia: 0.91 Temperature: 28.24 Pressure: 1002.1 Humidity: 52.2 Gas:
4005.9 Water level: 2 Waterproof temp: 28.312

```

Código 4.2: Ejemplo de salida del fichero Python

para obtener el resultado deseado.

En primer lugar, a través del nodo de ejecución, se pueden ejecutar ficheros que están en la máquina local. Este nodo es el que se ha utilizado para integrar en la aplicación el fichero presentado en la sección anterior 4.2.1. Para obtener individualmente el valor de cada sensor, se ha utilizado otro nodo —llamado nodo función— que permite incluir código directamente en Node-Red. Este nodo se ha utilizado para extraer de la salida del nodo ejecución el valor de interés en cada caso. Este proceso corresponde a las diferentes elipses salientes de *sensors script* del diagrama de casos (Figura 4.4). Con estos dos nodos algunos sensores han estado listos para mostrar los valores en la interfaz de usuario (IU), por lo que solo ha sido necesario añadir el widget más adecuado para cada caso a través del nodo pertinente. Este ha sido el caso de los sensores MQ-135 3.7, BME680 3.2 (en el caso de temperatura, presión y humedad) y DS18B20 3.3.

En el caso de los otros sensores han sido necesarios otros cambios para obtener el resultado adecuado:

- El sensor BME680 3.2 mide la resistencia del gas, pero se ha considerado que no es un parámetro intuitivo y no proporciona conocimiento al usuario. Por ello se ha utilizado otro nodo función que primero realiza la media de 50 lecturas de resistencia de gas y posteriormente establece la calidad del aire en función a la humedad y resistencia del aire comparándolo con la media calculada.
- Durante diferentes pruebas, se ha detectado que el sensor que mide el nivel

de agua 3.6 no es preciso a la hora de detectar la cantidad de agua presente, solamente es preciso detectando la presencia de esta. Por tanto, se ha optado por dividir en 4 rangos el nivel de agua (Figura 4.6), de tal manera que el usuario tenga una idea sobre la cantidad de agua presente. Aún así, el sensor no es suficientemente preciso durante la realización de diferentes pruebas y esta opción tampoco ha sido válida. Como ajuste que dota de más precisión a este sensor, se ha incorporado un botón en la IU para poder calibrar el sensor cuando el recipiente esté lleno de agua. Todo el desarrollo y las pruebas que se han llevado a cabo con este sensor pueden ser encontradas en la wiki ³ del proyecto.

- El sensor AMG8833 3.4-a devuelve en su lectura una matriz de 8x8 valores. Estos valores han sido convertidos a imagen ya que este sensor es una cámara térmica. Para este proceso ha sido necesario enlazar una serie de nodos hasta poder obtener la imagen más nítida, ya que la imagen obtenida originalmente ha sido una imagen de 8x8 píxeles que no alcanzaba a diferenciar objetos. De nuevo, tanto este progreso como las diferentes soluciones obtenidas se pueden encontrar en la wiki.

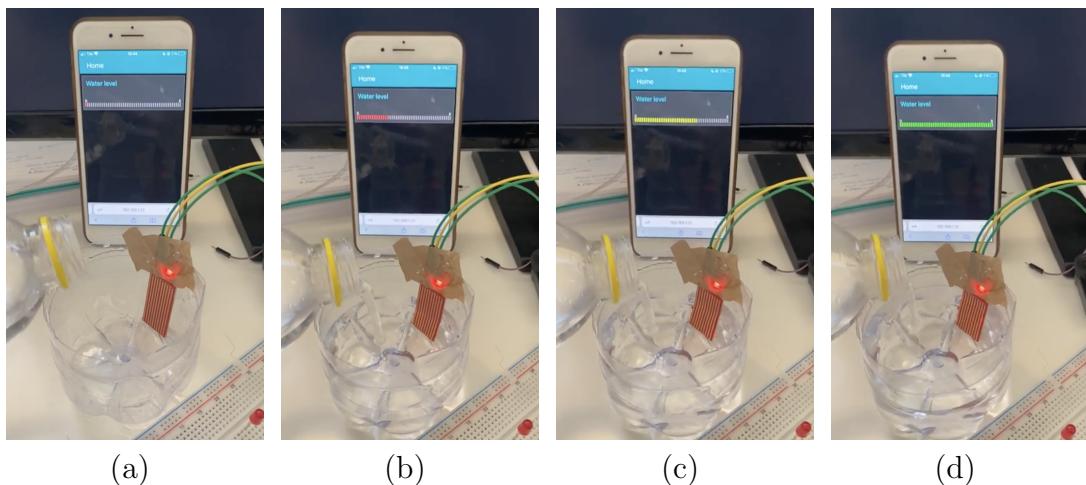


Figura 4.6: Proceso de indicación del nivel de agua.

Con estos ajustes realizados, se ha obtenido parte de la IU. Para hacerla más completa se ha incorporado un interruptor de forma que el usuario puede decidir si apagarla o bien mantenerla encendida, así como un nodo que regula la repetitividad del sistema cada 3 segundos. La interfaz formada por los sensores comentados previamente se puede ver en la Figura 4.7.

³<https://github.com/jmvega/tfg-icebollada/wiki/3.March-progress>

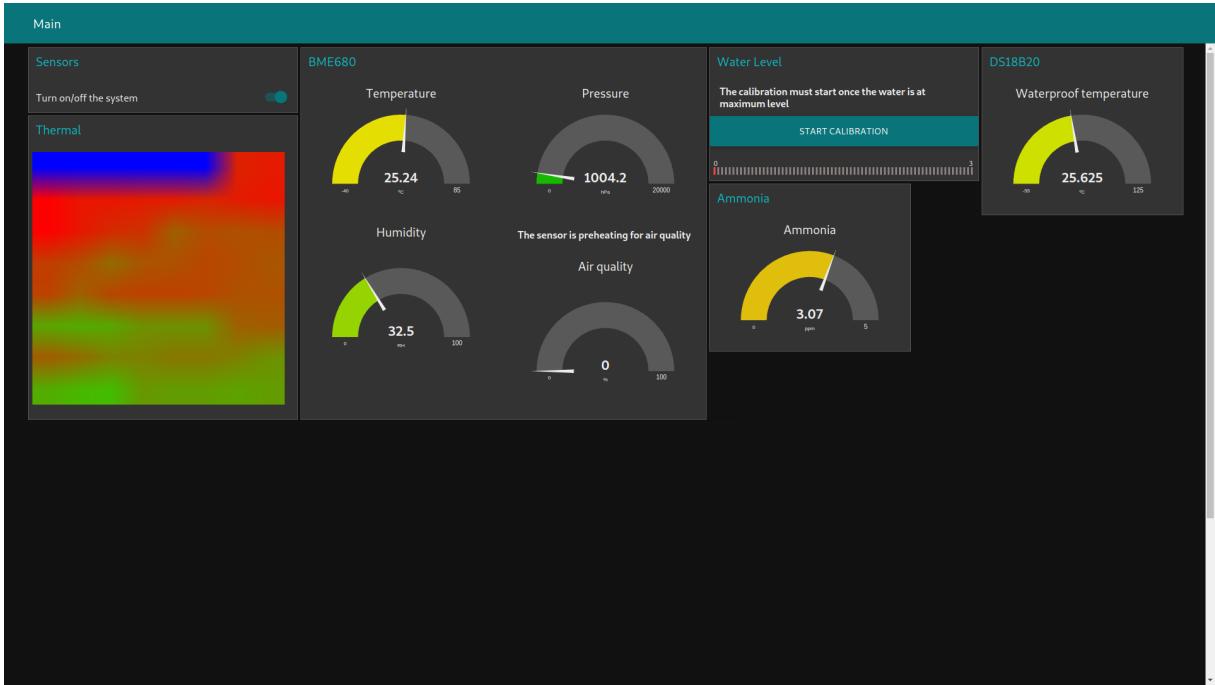


Figura 4.7: IU sin los sensores cámaras.

4.2.3. Integración de las cámaras en la UI

La creación de la interfaz en la subsección anterior no es completa, ya que faltan dos de los sensores que se utilizan en este trabajo: las cámaras. Para poder visualizar las cámaras desde Node-Red debe ser a través del nodo template (que significa plantilla), que permite visualizar una dirección web. Por tanto ha sido necesario crear un servidor que permita visualizar las imágenes de las dos cámaras y por tanto poder incluirlo en la IU. Para crear este servidor se ha utilizado Flask.

Se ha decidido crear un servidor por cada cámara. De esta manera se puede acceder solamente a una visualización o bien a las dos, según el interés del usuario. Para el funcionamiento de la cámara térmica Seek Thermal 3.4-b se ha utilizado la librería que tiene seek, que permite su visualización a través de OpenCV. Para el funcionamiento de la PiCam 3.5 se ha utilizado OpenCV tanto para su visualización como para la adición de la fecha y la hora en la visualización (Figura 4.8). El código 4.3 muestra como se consigue.

Una vez obtenidas las imágenes de las cámaras desde la Raspberry, se ha incorporado al código el uso de Flask para poder crear el servidor. Es necesario disponer de dos elementos: el código para mostrar la imagen de la cámara y una carpeta donde



Figura 4.8: Incorporación de la fecha y hora en la visualización de la PiCam.

```
ret, frame = self.__camera.read()
frame = cv2.rectangle(frame, (2,2), (275,35), (255,255,255), -1)
frame = cv2.putText(frame,
    str(datetime.datetime.now().replace(microsecond=0)), (10,25),
    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,0), 1, cv2.LINE_AA)
```

Código 4.3: Código para incorporar la fecha en la esquina superior izquierda.

estás las plantillas. Estas plantillas son ficheros escritos en html que muestran en el servidor la página que interesa en cada momento. La estructura que sigue un programa con Flask es la representada en el código 4.4.

```
from flask import *
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)
```

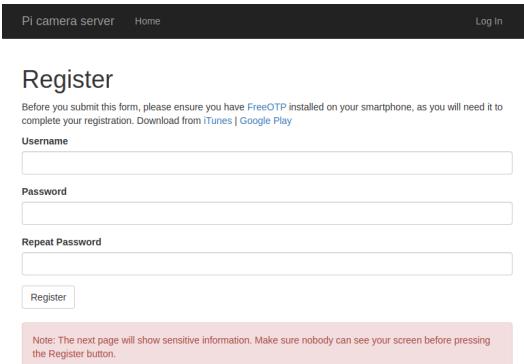
Código 4.4: Código simple que crea un servidor web y muestra el contenido de index.html.

Después de incorporar el código del funcionamiento de cada cámara en los respectivos servidores, se ha añadido un botón para iniciar o parar la grabación del vídeo en la cámara normal. Para diferenciar cuándo se está grabando de cuándo no, se ha añadido un mensaje en pantalla indicando que se está grabando. El vídeo se guarda

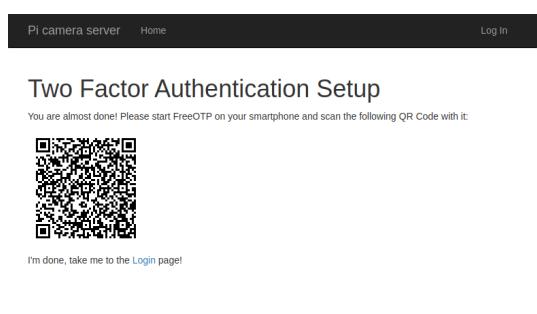
en la Raspberry de forma local y en formato .avi. En la Figura 4.9 se puede ver el proceso de registro en el servidor para acceder a la visualización de la PiCamera.



(a) Pantalla inicial del servidor



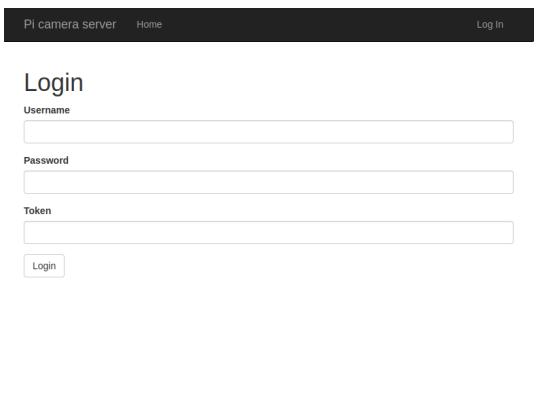
(b) Pantalla de registro



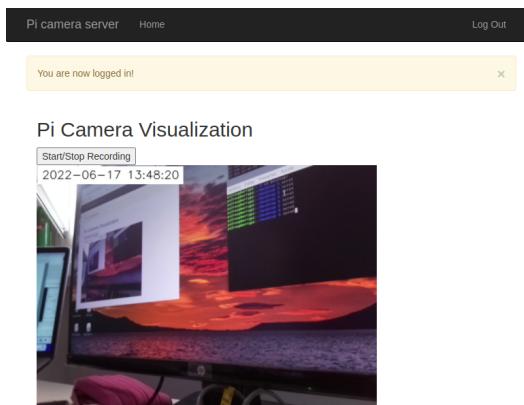
(c) Código de doble autenticación



(d) Escaneo de código



(e) Pantalla de login



(f) Visualización de la cámara

Figura 4.9: Proceso de registro en el servidor de Flask de la PiCam.

Una vez se ha obtenido la visualización de las cámaras en la IU junto al resto de

sensores, la interfaz está completada siguiendo la estructura presentada en el diagrama de casos de uso (Figura 4.4).

4.2.4. Seguridad

Debido a que se está trabajando en un servidor web, es importante dotar de cierta seguridad al servidor para evitar que sólo las personas autorizadas puedan acceder a las imágenes. Para ello, se ha decidido hacerlo a través del factor de doble autenticación (del inglés two factor authentication, 2FA). 2FA es el proceso de autenticación donde se añade un segundo paso en el proceso de identificación frente a un servicio. En este caso, además de que un usuario introduzca su contraseña, es necesario un código que se genera cada 30 segundos y caduca pasado ese tiempo. La generación de este código se hace a través de google authenticator⁴. Esta extensión de google genera los códigos de 30 segundos a partir de el escaneo de un código QR (Figura ??-d) o por la inserción manual de un código inicial.

Para que estos servidores funcionen con 2FA, ha sido necesario incorporar otras librerías así como otros módulos de Flask como Flask_login, Flask_bootstrap, Flask_sqlalchemy o Flask_wtf. Durante el desarrollo del servidor con 2FA, han surgido determinados problemas que pueden encontrarse junto a la solución en la wiki del proyecto⁵.

Una vez los servidores web han funcionado, se han integrado en Node-Red a través del nodo template. Un servidor está en el puerto 5000 y el otro en el 8000 para que así puedan estar ambas cámaras disponibles a la vez. Después de añadirlos, la interfaz de usuario (Figura 4.10) se ha completado.

También se ha tenido en cuenta que Node-Red funciona sobre una dirección a la que puede acceder cualquier usuario de la red sobre el puerto 1880. Por tanto, es imprescindible que sean solo usuarios autorizados los que puedan acceder a esta plataforma, sobre todo en la parte del flujo de nodos. Por tanto se ha optado por dividir este acceso en dos partes: Una de ellas la parte de administrador (Figura 4.11), donde se encuentra el flujo de nodos y se crea toda la interfaz de usuario. Únicamente un administrador debe poder acceder a esta parte, ya que es importante que una

⁴<https://chrome.google.com/webstore/detail/authenticator/bhghoamapcdpbophigooaddinpkbai?hl=es>

⁵<https://github.com/jmvega/tfg-icebollada/wiki/5.May-progress>

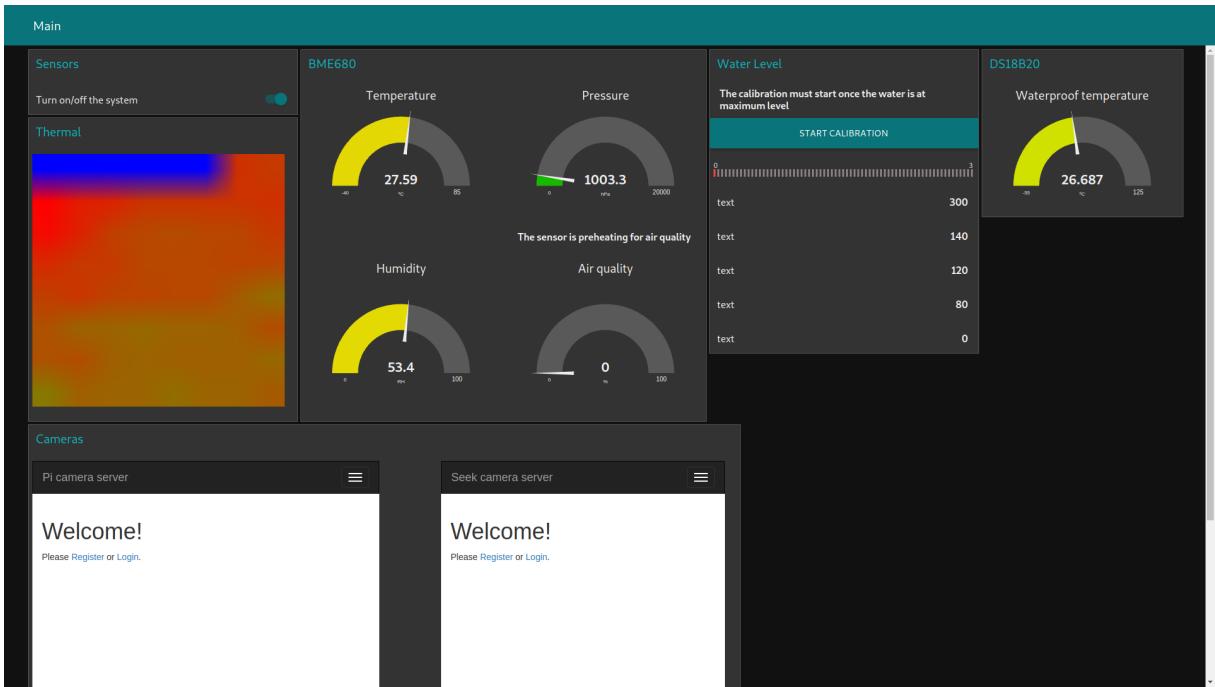


Figura 4.10: Visualización de la interfaz de usuario.

persona que no tiene conocimientos acerca de su funcionamiento no pueda realizar modificaciones. La otra parte corresponde a la interfaz de usuario (Figura 4.12) que, aunque ahí no pueden realizarse modificaciones, solo un usuario registrado conocedor de la contraseña sea capaz de acceder para evitar que personas ajenas puedan acceder a la información disponible.

Por otro lado, tanto los servidores de Flask como Node-Red se encontraban bajo una URL de HTTP (Hypertext Transfer Protocol). Por tanto, un cambio necesario ha sido añadir seguridad de tal forma que sea HTTPS (Hypertext Transfer Protocol Secure), para impedir que otros usuarios puedan interceptar la información confidencial que se transfiere entre el cliente y los servidores web a través de Internet. Este cambio se ha realizado añadiendo certificados autofirmados creados por el autor.

4.2.5. Autoarranque

Recordando que este sistema está pensado para personas que no están familiarizadas con la programación o los ordenadores, se ha considerado añadir una opción que facilite el arranque del sistema.

Con el encendido de la Raspberry, la interfaz de usuario se lanza de forma

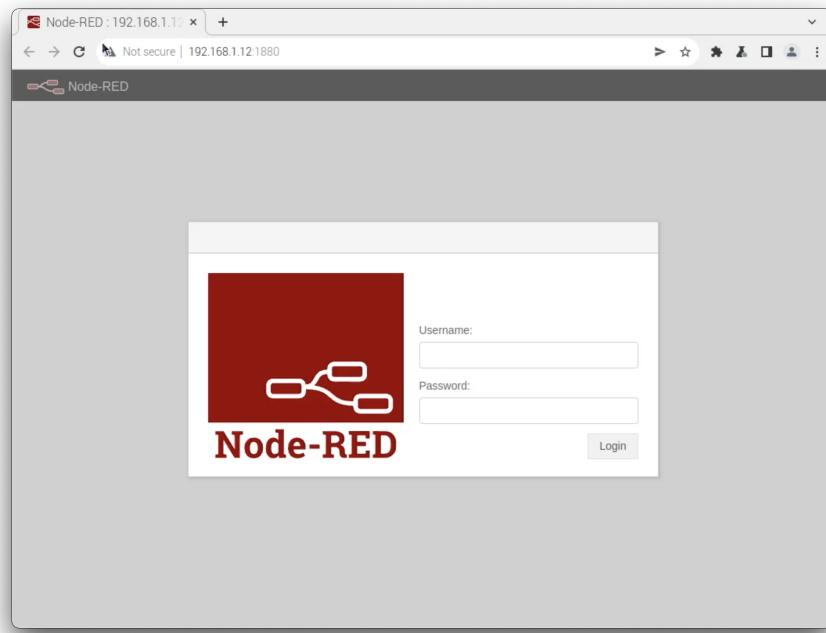
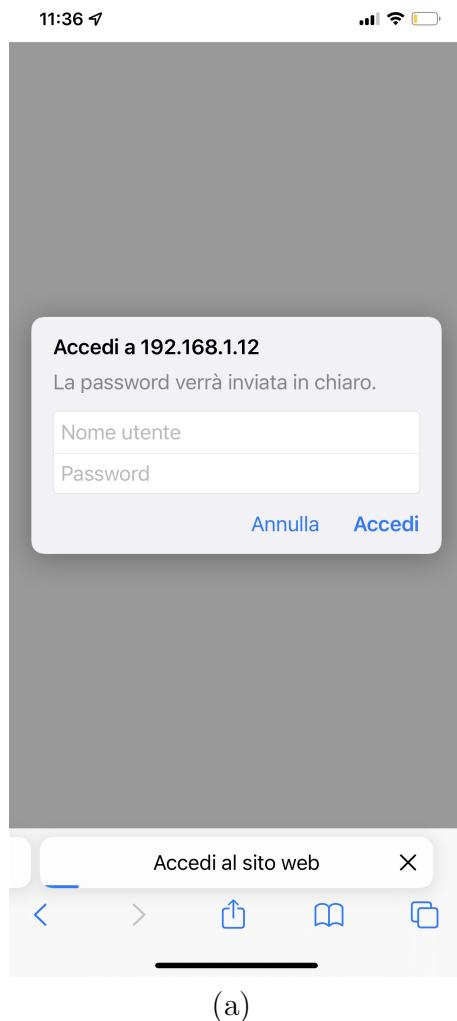


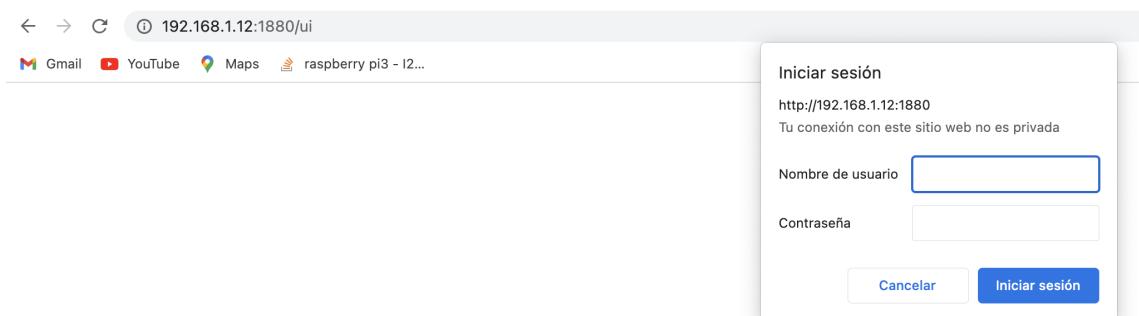
Figura 4.11: Acceso al flujo de nodos en Node-Red.

automática —así como los dos servidores— y se enciende solicitando el login del usuario para acceder a la IU. Además, se ha incorporado un led verde que indica cuando el sistema está listo para ser usado. El funcionamiento del autoarranque, tanto de Node-Red como de los servidores se encuentra detallado en la wiki ⁶. El proceso de autoarranque es visible en la Figura 4.13.

⁶<https://github.com/jmvega/tfg-icebollada/wiki/5.May-progress>



(a)



(b)

Figura 4.12: Acceso a la interfaz de usuario desde distintos dispositivos.

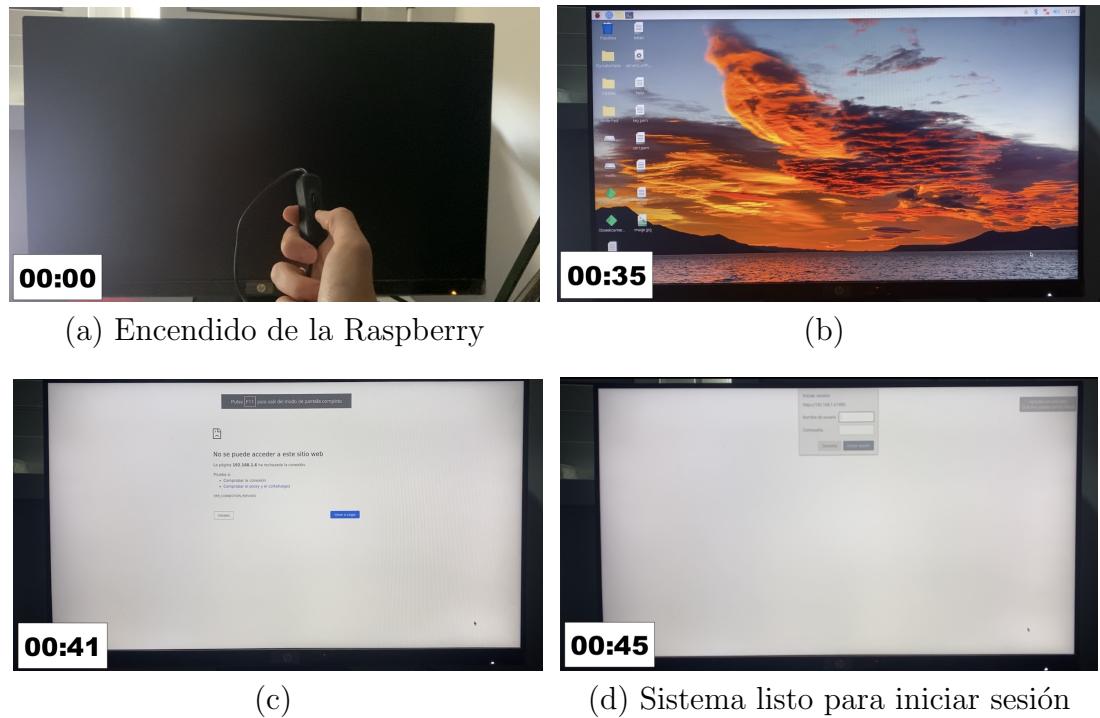


Figura 4.13: Proceso de autoarranque del sistema.

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

- [Andrés-Cano et al., 2021] Andrés-Cano, P., Calvo-Haro, J., Fillat-Gomà, F., Andrés-Cano, I., and Perez-Mañanes, R. (2021). Papel del cirujano ortopédico y traumatólogo en la impresión 3d: aplicaciones actuales y aspectos legales para una medicina personalizada. *Revista Española de Cirugía Ortopédica y Traumatología*, 65(2):138–151.
- [Arce et al., 2009] Arce, A., Tech, A., Silva, A., and Costa, E. (2009). Wireless sensor networks for bovine herd monitoring. *Archivos de Zootecnia*, 58:253 – 263.
- [Assael et al., 2022] Assael, Y., Sommerschield, T., Shillingford, B., Bordbar, M., Pavlopoulos, J., Chatzipanagiotou, M., Androutsopoulos, I., Prag, J., and de Freitas, N. (2022). Restoring and attributing ancient texts using deep neural networks — nature.
- [Guermazi et al., 2021] Guermazi, A., Tannoury, C., Kompel, A. J., Murakami, A. M., Ducarouge, A., Gillibert, A., Li, X., Tournier, A., Lahoud, Y., Jarraza, M., Lacave, E., Rahimi, H., Pourchot, A., Parisien, R. L., Merritt, A. C., Comeau, D., Regnard, N.-E., and Hayashi, D. (2021). Improving radiographic fracture recognition performance and efficiency using artificial intelligence — radiology. Radiological Society of North America.
- [UCM, 2017] UCM, V. (2017). Diseñan un sistema para monitorizar en tiempo real la salud animal. *Universidad Complutense de Madrid*.