UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Desarrollo de un prototipo de comedero automático con visión artificial

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Pérez Sánchez, Víctor Antoni

Tutor/a: Salido Gregorio, Miguel Angel

Director/a Experimental: PEREZ BERNAL, CHRISTIAN

CURSO ACADÉMICO: 2022/2023

Índice

Memoria

Memoria

1. Objeto y alcance del proyecto

El objetivo de este proyecto es el desarrollo e implementación de un sistema automático mediante visión artificial para comederos de gallinas para poder darles de comer sin necesidad de interacción humana. También se va a diseñar un prototipo de comedero integrando todos los componentes necesarios para el correcto funcionamiento del sistema.

La principal necesidad que se busca satisfacer es reducir la atención necesaria que supone tener un gallinero. Un comedero automático eliminaría la necesidad de tener que alimentar a las gallinas manualmente, pudiendo incluso estar ausente de la granja durante varios días y cumpliendo el cometido. La única atención que requeriría este sistema, más allá de alguna posible avería, sería rellenar el comedero cuando hiciera falta.

El sistema detectará si hay movimiento cerca de él y, seguidamente, detectará si el animal que lo ha causado es una gallina o algún otro animal. Según el animal que detecte, el sistema permitirá al animal acceder al pienso, si es una gallina, o no lo permitirá, si es otro animal.

El desarrollo de la aplicación que controlará el funcionamiento del sistema se realizará en Python debido a que, principalmente, el lenguaje posee librerías de captura y procesamiento de imagen y por ser compatible con la placa que se va a usar en el proyecto.

En cuanto a la implementación física, el sistema estará constituido por componentes sencillos y fiables, como servomotores y sensores de propósito general aptos para una gran variedad de situaciones. Esto permite obtener una implementación relativamente sencilla y altamente fiable.

El encargado de controlar todo el sistema será una placa Raspberry Pi. El uso de este tipo de placas nos otorga la posibilidad de leer y controlar los diferentes componentes del sistema al mismo tiempo que permite realizar el procesado de las imágenes obtenidas.

El sistema desarrollado permitirá que las gallinas sean alimentadas siempre que lo necesiten mientras que impide que animales callejeros o salvajes coman de él, lo cual supondría una pérdida de dinero para el dueño de las gallinas. Además, el sistema requeriría una atención humana prácticamente nula, siendo sólo necesario rellenar el comedero ocasionalmente.

La reducida atención que requiere este sistema supondrá, principalmente, un ahorro de tiempo sustancial, permitiendo al dueño de las gallinas atender otras necesidades agrícolas o personales, o incluso no estar presente en la granja durante largos periodos de tiempo, además de el ahorro de dinero que supone el no desperdiciar comida.

1.1 Relación con los ODS

La globalización y la industrialización ha golpeado duramente al mundo en el que vivimos. En los últimos años, hemos visto un aumento significativo en la contaminación del agua, la huella de carbono, la pérdida de diferentes especies animales y vegetales y la pobreza generalizada, el sexismo y la guerra. Esto ha llevado a los líderes mundiales a desarrollar los Objetivos de Desarrollo Sostenible (ODS) [7], los cuales deben ser seguidos por todos los miembros y organizaciones de la sociedad actual.

Este proyecto contribuye a alcanzar dos de los objetivos. El primero es el ODS número 9: industria, innovación e infraestructura. Nuestro sistema es un claro ejemplo de innovación a pequeña escala, ya que supone una mejora en eficiencia y comodidad en un proceso concreto que lleva existiendo muchísimos años.

En segundo lugar, también contribuye indirectamente a alcanzar el objetivo número 12: producción y consumo responsables. Según la ONU, *"El consumo y la producción sostenibles consisten en hacer más y mejor con menos"*. Nuestro proyecto supone una mejora que, aunque no muy grande, agiliza y optimiza el proceso de alimentación de gallinas, permite usar el pienso de manera más eficiente y el tiempo que se ahorre en esta tarea se podría utilizar para otros fines, aumentando así la eficiencia del trabajador.

Figura 1. Logos de los ODS 9 y 12

2. Antecedentes

La creación de este proyecto nace de la búsqueda de una forma más moderna y eficiente de alimentar a gallinas en pequeñas granjas en las que los animales disfrutan de cierta libertad. A raíz de ello, otros animales de granja pueden compartir espacio con las gallinas o incluso animales salvajes pueden invadir este espacio. Por ello, un sistema de visión artificial que discrimine los animales que se le acercan implicaría una forma autónoma de alimentar a las gallinas.

En esta sección se va a comentar el estado del arte, haciendo énfasis en las ventajas e inconvenientes de cada alternativa.Los comederos que se listan a continuación han tenido cierta influencia a la hora de diseñar este proyecto.

Actualmente, la mayoría de comederos para aves presentes en las granjas son diseños sencillos sin ningún tipo de automatización [4]. Este tipo de comederos (figura 2) funcionan meramente como recipientes con alguna ventaja adicional, como que presentan líneas separadoras o un recipiente grande para almacenar el pienso. Cumplen su función como comedero, pero sin ningún tipo de automatización.

Figura 2. Comederos para gallinas de Finca Casarejo y RVUEM

La empresa china Zhengyuan [22] ofrece un comedero para aves con un pequeño nivel de automatización (figura 3). Este comedero almacena el pienso en el silo de la parte superior y lo distribuye a la parte inferior cuando detecta movimiento cerca de él. Si bien consigue optimizar ligeramente la distribución del alimento, no consigue prevenir que otros tipos de animales se aprovechen de su diseño.

Figura 3. Comedero automático para aves de Zhengyuan

Existen otros casos como, por ejemplo, el comedero casero de Lumnah Acres [1] en YouTube (figura 4) aunque existen otros modelos similares. Este tipo de comederos pueden funcionar sin la intervención de ningún humano, siempre que haya comida dentro de él. Su funcionamiento es sencillo: al presionar la placa de presión del comedero se acciona un mecanismo de palancas que abre la puerta que permite acceder al interior del mismo.

En este caso, también se consigue automatizar el proceso, impidiendo además que animales livianos como pájaros puedan acceder a su interior. Sin embargo, cualquier animal suficientemente pesado que presione la placa de presión obtendría acceso al comedero, cosa que se quiere evitar.

Figura 4. Comedero automático de Lumnah Acres activado con un mecanismo físico

El comedero de Lumnah Acres es de fabricación casera, aunque existen marcas, como Safeed o UISEBRT [4], que comercializan comederos muy similares, con las mismas ventajas e inconvenientes.

Por último, existe otro tipo de comedero que cumple una función similar aunque el funcionamiento es diferente. Este es el caso del comedero para gatos que comercializa la empresa inglesa Sure PetCare [5] (figura 5). En este caso, el comedero detecta un sensor de proximidad presente en el collar del animal. Cuando el gato se acerca al comedero, éste se abre permitiéndole comer.

Este sistema es efectivo, pero sólo permite alimentar un único animal particular y, además, requiere que el animal porte un accesorio, condición trivial en el caso de animales caseros como perros y gatos pero no tanto con animales de granja.

Figura 5. Comedero para gatos automático de Sure PetCare

Los modelos descritos previamente no satisfacen completamente las necesidades de este proyecto, sin embargo, han servido como base e inspiración para la creación y diseño de nuestra solución.

3. Estudio de necesidades: limitaciones y condicionantes

En este apartado se va a realizar un estudio de necesidades con el que se pretende tanto concretar las características que deberá presentar el proyecto como qué factores influirán y limitarán la libertad creativa a la hora de tomar decisiones.

3.1 Necesidades

En primer lugar se va a analizar las necesidades esenciales que deberá satisfacer el proyecto.

En primer lugar, se pretende diseñar un sistema basado en visión artificial para discriminar qué animales podrán obtener acceso a la comida del comedero. Por ello, se necesitará una larga colección de imágenes de gallinas y de otros animales para la creación de este sistema.

Por otro lado, también se desea crear un sistema físico que reacciona a las respuestas que proporcione el sistema de visión artificial. Para ello será necesario obtener una serie de componentes, principalmente una cámara, algún tipo de motor que accione un mecanismo conectado a una compuerta y un microprocesador capaz de realizar procesado y análisis de imágenes y de comunicarse con el resto de componentes.

3.2 Limitaciones

A continuación se listan las restricciones que se han tenido en cuenta y que han influido en el desarrollo del sistema.

Como se ha nombrado previamente, se pretende diseñar un sistema basado en visión artificial. Este tipo de sistemas requieren de una enorme cantidad de imágenes para que funcionen de manera correcta. El número de imágenes puede incluso llegar a ser infinitamente grande en algunos casos. En el caso de este proyecto, ante la imposibilidad de poder realizar un recolección propia de imágenes, se recurrirá a la obtención de las imágenes a través de bases de datos en internet.

Por otro lado, al tener que procesar muchas imágenes, los sistemas de visión artificial requieren de una alta capacidad computacional o, en su defecto, de mucho tiempo. Debido a esta limitación, nuestro sistema sólo será alimentado con animales de granja, animales domésticos, como perros y gatos, pequeños animales salvajes como zorros y conejos, roedores e imágenes de objetos presentes en granjas. Esto puede llevar a que el sistema tenga dificultades para distinguir objetos en otras situaciones que no sean un entorno agrícola.

También se debe tener en cuenta que el sistema físico ha de ser factible. Se deberá comprobar debidamente que la placa y los componentes utilizados sean compatibles y que la fuente de alimentación utilizada puede suministrar la energía necesaria a todos los componentes.

Por último, se va a intentar, en la medida de lo posible, mantener los costes lo más bajos posibles. Con esto se pretende que el precio final del sistema resulte algo más atractivo a los posibles compradores.

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

4.1 Soluciones software alternativas

4.1.1 Datasets de COCO

Existen varias formas de crear sistemas de clasificación de imágenes. Una opción es hacer uso de los datasets de COCO (Common Objects in Context) (figura 6). Sin embargo, pese a tener datasets de varios animales, COCO no tiene un clasificador de gallinas. Aunque aún así podría resultar útil.

Figura 6. Logo de COCO

Para poder hacer uso de esta herramienta se debería operar de la siguiente manera: en primer lugar, se deberán preparar las imágenes positivas (imágenes de aquello que se quiere identificar) mediante una herramienta de anotación como CVAT (figura 7). Con este tipo de herramientas se pretende delimitar la zona de la imagen que contiene el objeto de interés, en nuestro caso, las gallinas. Además, esta aplicación se puede utilizar para fines más complejos, ya que ofrece la posibilidad de anotar diversos objetos simultáneamente, a la vez que proporciona herramientas que permiten realizar anotaciones más precisas, como pueden ser herramientas de anotación inteligente o filtros de imagen.

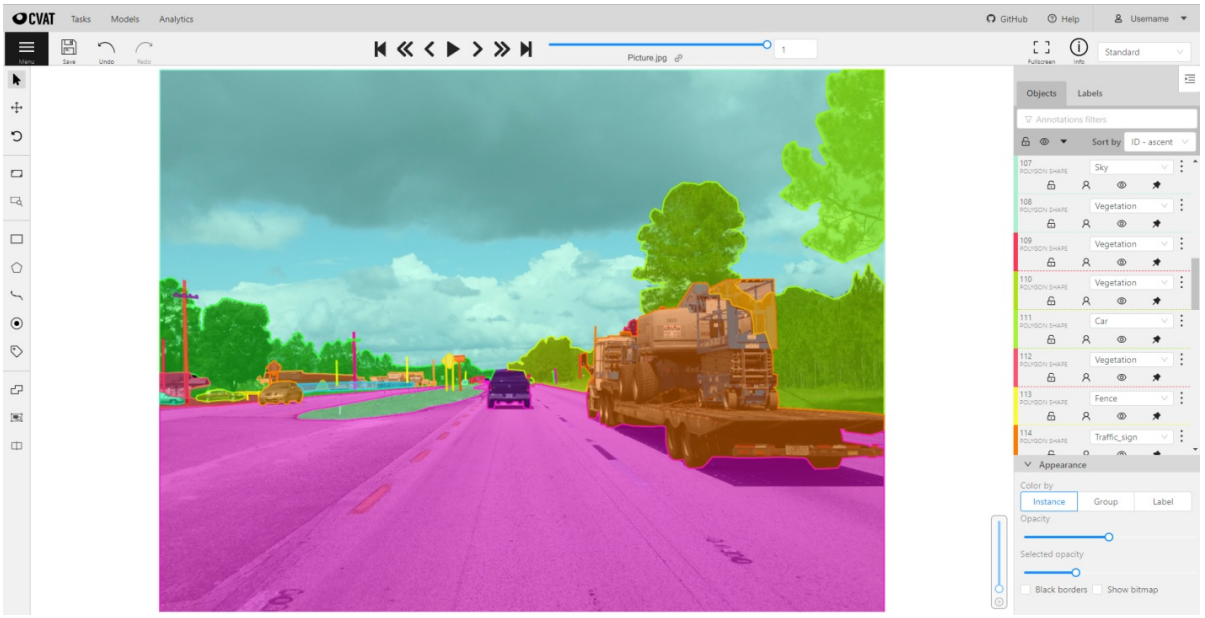

Figura 7. Interfaz de CVAT

A continuación se generaría un archivo JavaScript Object Notation (.json) con la información de éstas anotaciones. Finalmente, se incluiría este archivo al resto de datasets de COCO y con ello se obtendría un clasificador.

Esta opción fue descartada debido a la alta complejidad técnica que suponía este último paso y se consideró que dicha complejidad quedaba fuera del alcance de un trabajo de final de Grado de Ingeniería Electrónica Industrial y Automática.

4.1.2 Cámara Pixy2

Otra alternativa que se estudió fue usar la cámara Pixy2 (figura 8) para realizar la identificación de imágenes. La principal ventaja que ésta presenta es la simplicidad de su funcionamiento, ya que la Pixy2 tiene implementados algoritmos de detección de objetos en base a su color y forma y únicamente requiere de una fotografía para empezar a realizar la detección de objetos.

No obstante, la simplicidad de estos algoritmos es de tal magnitud que la cámara no es capaz de distinguir gallinas de otros animales u objetos del mismo color. De hecho, los creadores de la Pixy2 recomiendan usar la cámara para detectar objetos simples y monocromos, como pelotas de plástico de colores o piezas de Lego.

Otra ventaja que presenta esta alternativa es su fácil implementación a un sistema físico con una placa Arduino. Sin embargo, esta ventaja resulta completamente irrelevante para justificar el uso de la Pixy2 para el proyecto, debido a su inhabilidad para detectar objetos complejos. Finalmente, esta opción fue descartada.

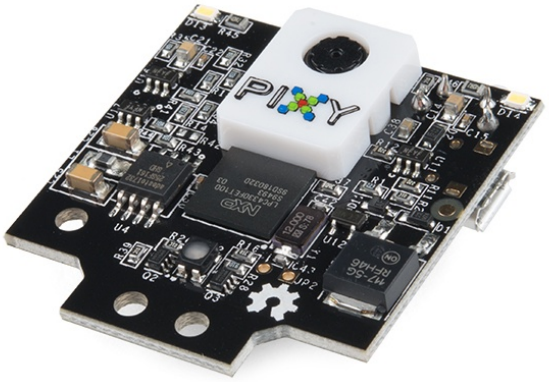

Figura 8. Cámara Pixy2

4.1.3 Cascade Trainer GUI

Cascade Trainer GUI (figura 9) es un programa que se encarga de crear distintos tipos de clasificadores, entre ellos el clasificador que se va a utilizar en este proyecto. La forma de trabajar con esta aplicación es muy similar a la de la solución por la que se ha optado. Por un lado, no necesita que se creen los ficheros de texto y el fichero vec (más información en el apartado 5) que se necesitan para el entrenamiento. Sin embargo, no permite crear las muestras positivas necesarias para el entrenamiento, y requiere que se tomen de un modo alternativo, ya sea con otra aplicación o con un script de Python.

Pese a que esta solución es muy similar a la que se ha desarrollado, este pequeño inconveniente a la hora de trabajar hizo que no se tomase esta opción, aunque el resultado final sería el mismo.

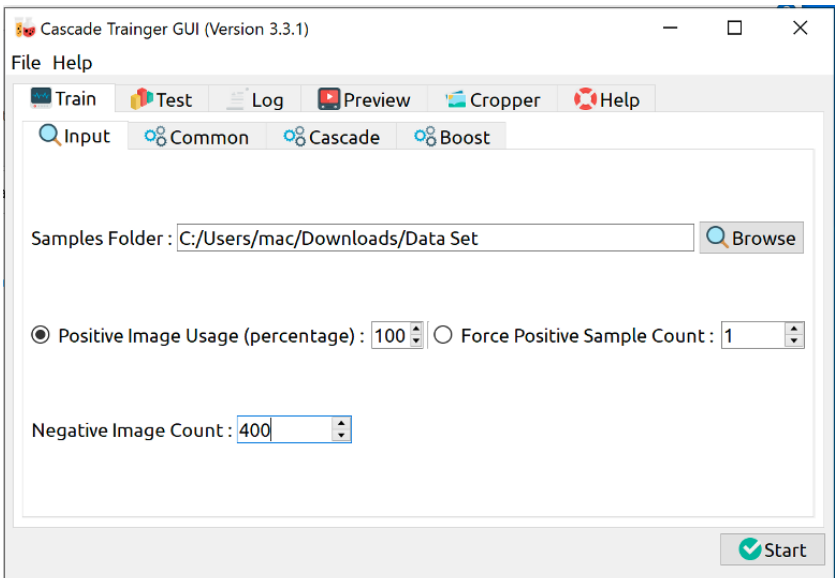

Figura 9. Interfaz de la aplicación Cascade Trainer GUI

4.2 Componentes descartados

4.2.1 LDR

En cuanto a hardware, se consideró la posibilidad de incluir en el sistema un sensor fotosensible (figura 10). Su finalidad sería reducir el tiempo de actividad del sistema a solamente durante el día, ya que durante la noche las gallinas no se acercarán al comedero.

Finalmente se optó por no usar este componente. Esto se debe a que en el sistema final se incluye en su lugar un sensor de movimiento. Con él se consigue también reducir el tiempo de actividad del sistema incluso más que con el LDR. El único inconveniente que podría darse es que el sistema se active por la noche debido a la presencia de algún animal salvaje. Sin embargo, debido a la oscuridad de la noche, la cámara no podrá reconocer que se le ha acercado y el comedero no se abrirá.

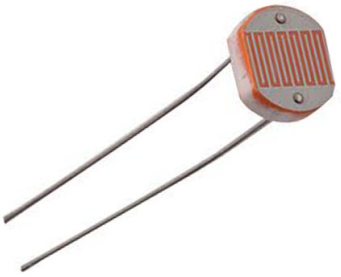

Figura 10. LDR GL5516

4.2.2 Placa Arduino

En cuanto a hardware, se tuvo en cuenta la posibilidad de usar una placa Arduino (figura 11), pero esta opción fue descartada rápidamente debido a que ésta no es suficientemente potente para controlar un sistema con visión artificial en tiempo real. Sin embargo, se consideró la posibilidad de usar la placa Arduino para realizar el control de los sensores y actuadores, mientras que un ordenador se encargaría de procesar y clasificar las imágenes.

Finalmente, no se realizó de esta manera debido a que se ha decidido priorizar que el sistema funcione de manera autónoma sin la necesidad de tener un ordenador conectado a él.

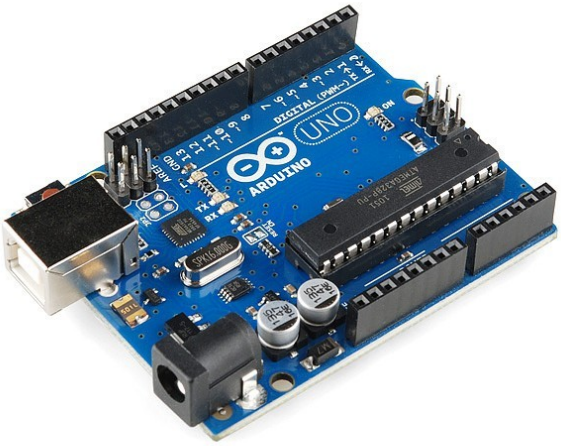

Figura 11. Placa Arduino Uno

4.2.3 Raspberry Pi 4

La Raspberry Pi 4 (figura 12) ofrece más potencia que la 3 y la 3 B + (modelo que se ha utilizado en el proyecto). Sin embargo, aunque esta potencia extra podría resultar útil para tener una respuesta algo más rápida, el programa que controla el sistema no es demasiado exigente, por lo que la Raspberry Pi 3 B + cumple su cometido a la perfección.

Por otro lado, la crisis de componentes electrónicos ha derivado en una escasez de esta placa. Los distribuidores oficiales como Kubii y Tiendatec en España se encuentran sin stock, mientras que unidades vendidas por terceros o de segunda mano alcanzan precios muy elevados, alrededor de 180€ en algunos casos.

Estos dos factores han resultado en que no se utilice la Raspberry Pi 4 para el proyecto y se opte por el modelo 3 B+.

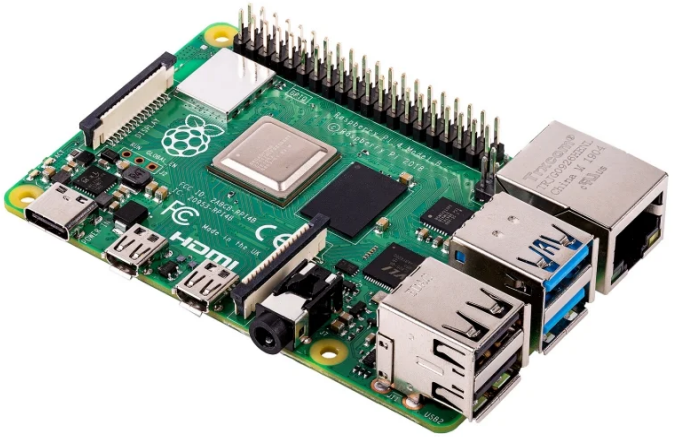

Figura 12. Placa Raspberry Pi 4

4.2.4 Adaptador de corriente 5V 2A

El sistema necesita una fuente de alimentación para suministrar energía a sus componentes. Para ello, se pretende usar la red eléctrica, sin embargo, se necesitará usar un adaptador de corriente para transformar los 230V en 5V, ya que la Raspberry y los actuadores deben ser alimentados con este voltaje.

En el apartado 6.1 de este documento se demuestra que un adaptador de corriente de 2A satisfaría las necesidades del proyecto. Seguidamente se contrastaron los precios de los transformadores de 5V 2A con los de 5V 3A. Los precios del primero oscilan entre 5€ y 8€, mientras que los precios del segundo oscilan entre 6€ y 11€. Esta pequeña diferencia de precio se ha considerado que compensa la seguridad que brinda tener algo más de amperaje, por lo finalmente se usará un adaptador de corriente de 5.1V 3A.

4.3 Justificación de la solución adoptada

4.3.1 Solución software

En este apartado se va a enumerar y detallar las decisiones finales que se han tomado para desarrollar e implementar el sistema.

En cuanto a la parte software del proyecto, finalmente se decidió usar un clasificador en cascada Haar [14][23] para realizar el reconocimiento de imágenes. Esto es un método de machine learning en el que se utiliza la información recopilada en la salida de un clasificador como información adicional para el próximo clasificador en la cascada.

Las principales ventajas que presenta este método es que es muy eficiente computacionalmente para la gran cantidad de información que maneja y que se puede desarrollar e implementar fácilmente en Python gracias a las librerías e información que existe sobre el tema.

Para que el clasificador funcione necesita una gran cantidad de imágenes positivas (imágenes de aquello que se quiere identificar) y de imágenes negativas (imágenes de otras cosas). A partir de estas imágenes se identifican unas características comunes, conocidas como los rasgos Haar (figura 13). Las características son contrastes entre zonas claras y oscuras de la imagen, y son representadas por un valor único obtenido al restar la suma de píxeles contenidos en el rectángulo blanco y la suma de píxeles contenidos en el rectángulo negro al colocar una de estos rasgos en la imagen.

Figura 13. Rasgos Haar rectangulares básicos

Los rasgos Haar se pueden agrupar dependiendo de si son de borde, de línea o centrados (figura 14). Además, cada uno de ellos puede estar rotado 45º o 90º.

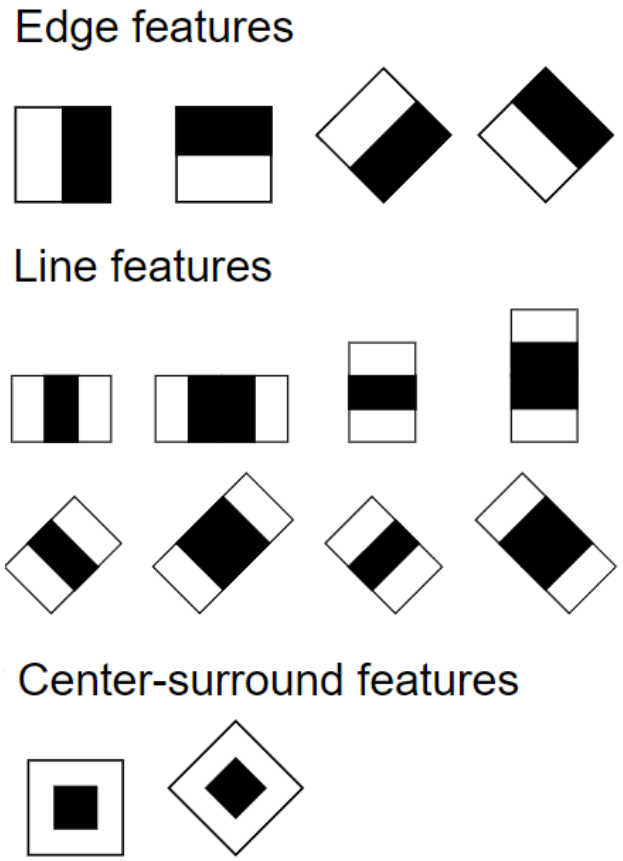

Figura 14. Rasgos Haar clasificados

Tras identificar las características, éstas se agrupan en diferentes etapas y se aplican una por una o en pequeños grupos. Normalmente, las primeras etapas contendrán menos características y más generales, aumentando en especificidad al avanzar la clasificación. De esta forma, si una imagen no satisface las exigencias de una de las primeras etapas, se descarta en ese momento y no es analizada por las siguientes etapas.

Esta solución presenta dos grandes inconvenientes. El primero es la gran cantidad de imágenes que hay que obtener y procesar para la creación del clasificador. El segundo es el tiempo necesario para el cómputo del propio clasificador, pudiendo llegar a durar desde unas horas hasta varios días. Pese a ello, las ventajas superan a los inconvenientes y se ha optado por esta alternativa.

4.3.2 Solución hardware

Una vez obtenido el clasificador y desarrollado un programa que lo aplique como se requiere, éste se debe implementar en una placa de desarrollo. Ésta debe ser capaz de soportar una cámara para la captura de imágenes y debe tener compatibilidad con Python, lenguaje de programación en el que se ha desarrollado el clasificador.

Descartando la Raspberry Pi 4 por los motivos comentados anteriormente, la mejor opción y por la que se ha optado es la Raspberry Pi 3 B+ (figura 15).

Figura 15. Raspberry Pi 3 B+

Además, la Raspberry posee una serie de pines I/O que nos permitirán controlar los sensores y actuadores. De esta forma no se requeriría de ningún componente adicional para hacer funcionar el sistema.

Para el mecanismo físico se va a recurrir únicamente a un servomotor, encargado de abrir o cerrar la puerta del comedero según resulte oportuno. Además, se va a incluir un sensor de movimiento con el fin de facilitar la tarea del clasificador y que sólo entre en funcionamiento cuando se detecte movimiento. Estos dos componentes tienen tareas ciertamente poco exigentes, por lo que se ha decidido usar componentes de propósito general, dado su bajo coste, fácil obtención y simplicidad de implementación.

Por último, el sistema necesita una cámara para funcionar. Dado el contexto en el que se pretende utilizar, no es necesario que la cámara sea de última generación. Por otro lado, debe ser compatible con la Raspberry. Afortunadamente, la placa dispone de varios puertos USB, lo que facilita la búsqueda de una cámara adecuada. Cualquier cámara o webcam HD con conectividad USB satisfaría nuestras necesidades.

El sistema completo se ha decidido diseñar pretendiendo usar un número reducido de componentes y manteniendo los costes bajos, con la finalidad de que el proyecto sea accesible y pueda ser instalado con facilidad en la mayor cantidad de situaciones posibles.

5. Descripción detallada de la solución adoptada

5.1 Solución software

5.1.1 Clasificador en cascada Haar

Como ya se ha comentado, la mejor manera de abordar el problema sería usando la librería OpenCV de Python para el procesado de imágenes, debido a su popularidad, ser un software libre y recibir gran cantidad de actualizaciones y mejoras constantemente. Para realizar la identificación de las gallinas, se va a programar un clasificador en cascada Haar. Para diseñarlo se han seguido los siguientes pasos [12][13][15]:

5.1.1.1 Recolección de imágenes y data augmentation

En primer lugar, se recolectaron una serie de imágenes de dos tipos: positivas (gallinas) (figura 16) y negativas (otros animales, pasto, hierba, vallas, graneros,...) (figura 17). Es importante que en las imágenes positivas se distinga correctamente la gallina y sus características, principalmente la cresta y la barbilla. Por otro lado, las imágenes negativas deben de ser de elementos que puedan entrar en el campo de visión del sistema. Por ello se ha seleccionado una colección de animales de granja, animales salvajes o callejeros, vallas u otros tipos de confinamientos e imágenes que se podrían considerar como fondo (background), como hierba, paja o palos.

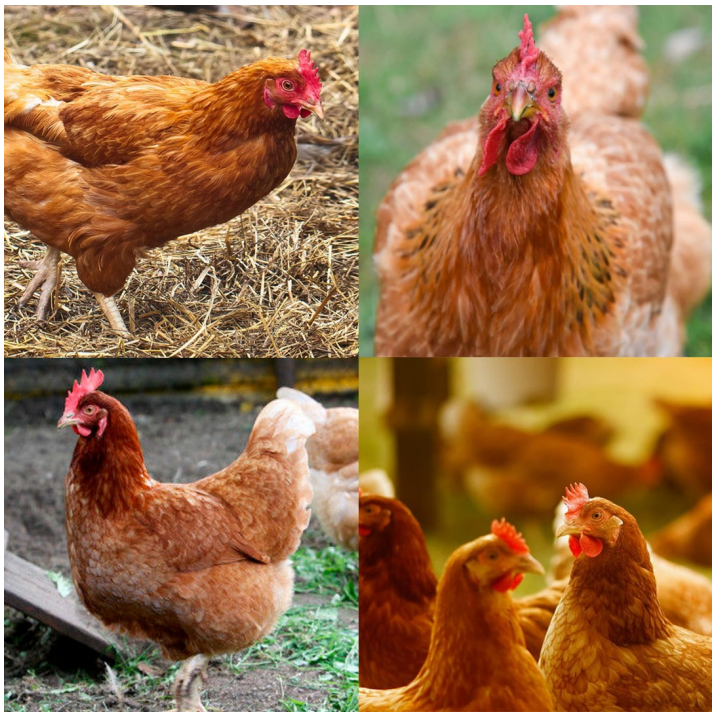

Figura 16. Ejemplos de imágenes positivas

Figura 17. Ejemplos de imágenes negativas

Este tipo de clasificadores requieren una gran cantidad de imágenes, generalmente en el rango de las altas centenas o millares, dependiendo de la complejidad del objeto a identificar. Sin embargo, encontrar las suficientes imágenes de gallinas de buena calidad y únicas no es tan sencillo como parece, dado que la cantidad de imágenes disponibles es limitada.

De bases de datos de imágenes como Unsplash [21] y Adobe Stock [2] se recolectaron alrededor de 280 imágenes distintas de gallinas. Siendo estas no suficientes, las imágenes fueron sometidas a un proceso conocido como “data augmentation” [8], con el fin de aumentar el número de imágenes con las que trabajar.

Ese proceso consiste en lo siguiente: las imágenes serán volteadas y, posteriormente, se les añadirá ruido gaussiano a las imágenes volteadas y a las originales. De esta forma, se consigue aumentar el número de imágenes de entrenamiento ya que, pese a que al ojo humano éstas le pueden resultar muy similares, éstas imágenes poseen características que un algoritmo de visión artificial es capaz de identificar y asimilar como un nuevo subconjunto.

De esta manera se cuadruplicó el número de imágenes positivas y se pasó de tener 280 ejemplares a casi 1200. La función de la figura 18 fue la utilizada para voltear las imágenes, siendo “positive” el directorio con las imágenes positivas. Con ella, simplemente se obtienen todos los archivos de un directorio con la función “listdir” de la librería “os”. Estos archivos se almacenan temporalmente en una variable (image) y se someten a un volteo mediante la función “flip” de OpenCV. Finalmente, las imágenes volteadas se añaden al directorio con el nombre de la imagen original añadiendo “flipped_” al principio.

```
def flip_images(PATH='./positive/'):
    for filename in os.listdir(PATH):
        image = cv2.imread(PATH+filename)
        flipped_image = cv2.flip(image, 1)
        cv2.imwrite(PATH+"flipped_"+filename, flipped_image)
```

Figura 18. Función para voltear imágenes horizontalmente

El segundo filtro utilizado es un filtro gaussiano. Este permite añadir ruido a las imágenes para aumentar aún más nuestra colección de imágenes positivas. En las siguientes figuras (19 y 20) se muestra cómo calcular distribuciones normales para posteriormente poder crear una función para añadir ruido gaussiano a las imágenes [11].

$$P(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

Figura 19. Fórmula de la función de distribución de probabilidad gaussiana

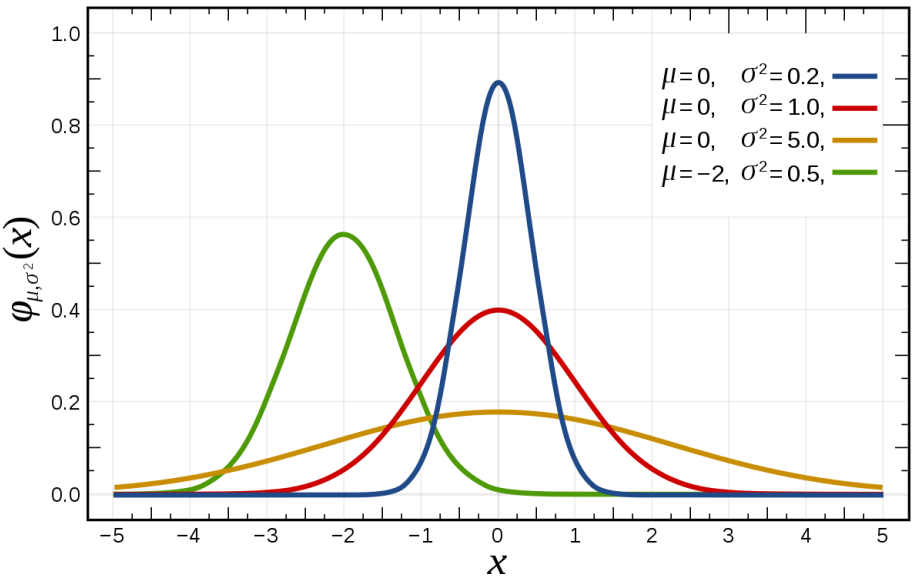

Figura 20. Efecto de mean y sigma en la distribución gaussiana

A continuación se describe el funcionamiento de la función creada. Las imágenes se guardan en la variable “image” temporalmente igual que con la función de voltear las imágenes. Posteriormente se definen las variables “mean” (μ) y “sigma” (σ) (a través de “var”) que son las que controlan el nivel de ruido gaussiano que se va a añadir. “Mean” se mantiene a 0 para tener una distribución de sonido uniforme y con “var” se decide cuánto sonido se quiere añadir, habiendo más sonido conforme mayor sea la variable “var”. En este proyecto se ha usado un valor de 25 para “var”.

El siguiente paso es usar unas funciones de la librería numpy para crear las imágenes con ruido. Esta biblioteca permite crear matrices numéricas y trabajar con ellas. Esto nos será de utilidad debido a que las imágenes se definen como una matriz de píxeles con distintos valores RGB para cada píxel.

El primer paso para calcular la distribución gaussiana consiste en usar la función "np.random.normal" y los valores definidos anteriormente. Seguidamente, se crea la variable "noisy_image" para ser usada como base para la imagen que se va a crear. Finalmente, se crea la imagen con el ruido añadido. Si la imagen tiene dos dimensiones, simplemente se suman los valores de la imagen original y la variable "gaussian" creada anteriormente. Si la imagen tiene 3 dimensiones, se debe sumar cada una independientemente, ya que la variable gaussian sólo tiene 2. Tras esto, se normaliza la imagen y se guarda en la carpeta deseada añadiendo "gaussian_" a su ruta. La figura 21 muestra la función definida.

```
def gaussian_noise(PATH='./positive/'):
    for filename in os.listdir(PATH):
        image = cv2.imread(PATH+filename)
        mean = 0
        var = 25
        sigma = var ** 0.5
        gaussian = np.random.normal(mean, sigma,
(image.shape[0],image.shape[1]))

        noisy_image = np.zeros(image.shape, np.float32)

        if len(image.shape) == 2:
                noisy_image = image + gaussian
        else:
            noisy_image[:, :, 0] = image[:, :, 0] + gaussian
            noisy_image[:, :, 1] = image[:, :, 1] + gaussian
            noisy_image[:, :, 2] = image[:, :, 2] + gaussian

            cv2.normalize(noisy_image, noisy_image, 0, 255,
cv2.NORM_MINMAX, dtype=-1)
        noisy_image = noisy_image.astype(np.uint8)

        cv2.imwrite(PATH+"gaussian_"+filename, noisy_image)
```

Figura 21. Función para añadir ruido gaussiano

La obtención de imágenes negativas es mucho más sencilla, ya que en este caso no se nos limita a un solo animal. Por ello, se puede simplemente acceder a Unsplash [21] y descargar las imágenes que sean necesarias. Para este proyecto se han obtenido unas 2200 imágenes negativas, aproximadamente el doble que de imágenes positivas.

5.1.1.2 Creación de los ficheros necesarios para el clasificador

Con las imágenes descargadas y clasificadas en dos carpetas, se deben crear los archivos necesarios para poder proceder a programar el clasificador. Para los siguientes pasos se ha usado una serie de programas obtenidos de SourceForge [3] para facilitar el proceso de creación del clasificador.

En primer lugar se deben crear dos ficheros de texto con las direcciones de las imágenes positivas y negativas. Para las imágenes negativas simplemente se usó la función de la figura 22. Con ella se abre un fichero de texto (neg.txt) en modo de escritura (w). Con la función "os.listdir" se analizan todos los archivos de la carpeta "negative" y se escribe en el fichero de texto su ruta, incluyendo la carpeta en la que se encuentran. La función descrita se muestra a continuación:

```
def generate_negative_description_file():
    # Open a new file
    with open('neg.txt', 'w', encoding='utf-8') as f:
        # Loop through all files
        for filename in os.listdir('negative'):
            f.write('negative/' + filename + '\n')
```

Figura 22. Función para crear el fichero de las imágenes negativas

Las imágenes positivas han de ser anotadas. Para ello se usará el programa "opencv_annotation.exe". Tras descomprimir la carpeta y añadirla al proyecto, se ejecutará el siguiente comando (figura 23) en la terminal del entorno de desarrollo, donde 'pos.txt' es el fichero de texto de destino y 'positive' es la carpeta donde se encuentran las imágenes positivas.

```
opencv\build\x64\vc15\bin\opencv_annotation.exe
 --annotations=pos.txt --images=positive/
```

Figura 23. Comando para lanzar el programa de anotación

Al hacerlo, se abrirá la herramienta de anotación (figura 24), donde con dos clicks se deben dibujar rectángulos alrededor de las regiones de interés [18]. Tras esto, se debe pulsar la tecla "c" para confirmar la selección. En caso de una anotación errónea, se puede dibujar otro rectángulo sin confirmar el anterior, de forma que éste último se borrará. En caso de querer borrar una anotación confirmada, se debe pulsar la tecla "d" para borrar la última selección.

Figura 24. Interfaz del programa de anotación

Concretamente, en este proyecto se ha decidido considerar como región de interés la cabeza de las gallinas, en lugar de la gallina entera. Esto se debe a que se ha considerado que, al obligar al clasificador a centrarse en sólo las cabezas, le resultará más fácil reconocer lo que se le pide y reducir el número de falsos positivos.

Tras pasar por todas las imágenes, se generará un fichero de texto con las direcciones de las imágenes positivas y las coordenadas de los rectángulos que encarcelan las regiones de interés.

El siguiente paso es generar el fichero .vec de las muestras positivas. Este fichero contiene la información recogida en el archivo pos.txt tras someterse a un proceso de preparación necesario para que el clasificador pueda entrenarse. Para hacer esto se usará el programa "opencv_createsamples.exe". Simplemente se ha de ejecutar la siguiente instrucción (figura 25) en la terminal del entorno de desarrollo.

```
opencv\build\x64\vc15\bin\opencv_createsamples.exe -info pos.txt
-w 30 -h 30 -num 1500 -vec pos.vec
```

Figura 25. Comando para crear el fichero .vec

Los parámetros proporcionados son los siguientes:

- info: el fichero de texto generado en el paso anterior (pos.txt).
- w: ancho que tomarán las muestras positivas tras ser redimensionadas. A mayor valor, mayor detalle tendrá el entrenamiento a costa de tardar más en crearse el

clasificador. Recomendablemente, este valor debe ser entre 20 y 35. En este proyecto se ha usado un valor de 30.

- h: altura que tomarán las muestras positivas tras ser redimensionadas. A mayor valor, mayor detalle tendrá el entrenamiento a costa de tardar más en crearse el clasificador. Recomendablemente, este valor debe ser también entre 20 y 35. En este proyecto se ha usado también un valor de 30.
- num: número máximo de imágenes positivas a procesar. Cualquier número mayor que el número total de imágenes positivas servirá.
- vec: nombre del archivo que contendrá las muestras positivas tras ser redimensionadas (pos.vec).

5.1.1.3 Entrenamiento del clasificador

Finalmente, ya se puede proceder a crear el clasificador. Para ello se va a usar el programa “opencv_traincascade.exe”. Se ha de ejecutar la instrucción de la figura 26 en la terminal del entorno de desarrollo.

```
opencv\build\x64\vc15\bin\opencv_traincascade.exe -data cascade/
-vec pos.vec -bg neg.txt -w 30 -h 30 -precalcValBufSize 6800
-precalcIdxBufSize 6800 -numPos 900 -numNeg 2100 -numStages 22
```

Figura 26. Comando para comenzar el entrenamiento del clasificador

Los parámetros que se deben proporcionar son los siguientes:

- data: carpeta en la que se guardarán los archivos generados.
- vec: fichero .vec generado en el apartado anterior (pos.vec).
- bg: fichero de texto con las direcciones de las imágenes negativas (neg.txt).
- w: ancho de las muestras positivas. Este valor ha de ser el mismo que el que se haya usado en el apartado anterior.
- h: altura de las muestras positivas. Este valor ha de ser el mismo que el que se haya usado en el apartado anterior.
- numPos: número de muestras positivas utilizadas en el entrenamiento en cada etapa del clasificador. Este número ha de oscilar entre el 68% y el 85% del número total de muestras positivas, ya que el entrenamiento consumirá en cada etapa algunas muestras adicionales. El valor seleccionado es 900, siendo este aproximadamente el 83% de las 1080 muestras positivas generadas.
- numNeg: número de imágenes negativas utilizadas en el entrenamiento en cada etapa del clasificador. Este número ha de oscilar entre el 90% y 98% del total de imágenes negativas. El valor seleccionado es 2100, siendo este aproximadamente el 94% de las 2240 muestras negativas que se tenían.
- numStages: número de etapas durante las que se entrenará el clasificador. Este valor cambiará dependiendo de la complejidad de las imágenes a detectar y de la precisión que se quiera obtener. Generalmente oscilará entre 20 y 60. Tras realizar pruebas con diferentes valores, el que presentaba mejores resultados era 22. Esto se detalla en el siguiente apartado.

Otros parámetros opcionales que pueden resultar útiles para entrenar modelos son los siguientes:

- precalcValBufSize: tamaño del búfer para valores precalculados de características. Este valor combinado con el siguiente no debe exceder el total de memoria RAM disponible en el dispositivo. Para mayor eficiencia temporal, se recomienda que la suma de estos dos sea sobre el 90% de memoria RAM disponible.
- precalcIdxBufSize: tamaño del búfer para valores precalculados de características. Este valor combinado con el anterior no debe exceder el total de memoria RAM disponible en el dispositivo. Para mayor eficiencia temporal, se recomienda que la suma de estos dos sea sobre el 90% de memoria RAM disponible.
- minHitRate: tasa de acierto mínima deseada para cada etapa del clasificador. Para diseñar un buen clasificador este valor debe oscilar entre 0.995 y 0.999. En este proyecto se ha usado el valor predeterminado (0.995).
- maxFalseAlarmRate: tasa máxima deseada de falsos positivos para cada etapa del clasificador. En este proyecto se ha dejado este valor sin especificar y se ha relegado esta tarea al propio programa.
- acceptanceRatioBreakValue: este valor se usa para determinar cuando el modelo debe seguir aprendiendo y cuándo detenerse. Una buena pauta es no entrenar más allá de un valor de 10^{-5}, para garantizar que el modelo no se sobreentrene. En este proyecto este parámetro no se ha usado y se ha decidido cuándo parar de entrenar manualmente, comprobando la efectividad del clasificador tras un cierto número de etapas. Esto se detalla en el siguiente apartado.

Cuando el programa haya terminado, se habrán generado una colección de archivos .xml en la carpeta que se haya indicado, siendo “cascade.xml” el clasificador. Se puede continuar entrenando el clasificador volviendo a lanzar la misma instrucción aumentando el parámetro numStages. El programa retomará el entrenamiento desde donde lo hubiera dejado y no necesitará volver a empezar de cero. Si se quisiera crear otro clasificador con parámetros completamente distintos se debe vaciar la carpeta de “data”, pudiéndose guardar el archivo .xml en otra carpeta para su uso futuro.

5.1.1.3.1 Criterio para seleccionar el clasificador

Para hacer una primera criba, se ha observado el valor de “acceptanceRatio” que se muestra en cada etapa del entrenamiento del clasificador. Un valor aceptable para este dato debe oscilar en el rango de las cienmilésimas. Este valor se obtenía entre las etapas 20 y 25 de entrenamiento. Por ello, se han seleccionado 3 clasificadores en este intervalo (20, 22 y 25 etapas) y se han sometido a unas pruebas usando distintos valores de los parámetros scaleFactor y minNeighbours. Estos valores se explican detalladamente en el siguiente apartado, sin embargo, nótese que aumentando el valor de minNeigh o bien, reduciendo el valor de scaleFactor, se obtendrá un clasificador más estricto, lo cual implicaría una reducción en la tasa de falsos positivos pero aumentaría la tasa de falsos negativos, por lo que se debe llegar a un compromiso.

En estas pruebas, se ha buscado conseguir una tasa de falsos positivos inferior al 10% y una tasa de falsos negativos inferior al 20%. Se es más estricto con los falsos positivos ya que, que un animal que se acerque al sistema causará que éste entre en funcionamiento varias veces, tantas como veces se detecte su presencia con el PIR.

En primer lugar, se toma el clasificador de 20 etapas y se le aplican unos valores estándar para scaleFactor y minNeigh, siendo estos 1.05 y 3, respectivamente. Habiendo obtenido una tasa de falsos positivos demasiado elevada, se procede a, por un lado, aumentar el valor de minNeigh y por otro, a disminuir el valor de scaleFactor, pretendiendo conseguir un clasificador más estricto por dos vías distintas. En ambos casos, se obtiene la tasa de falsos positivos deseada a costa de una tasa de falsos negativos demasiado alta. Todos estos resultados se pueden ver reflejados en la figura 27.

20 stages			
Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	45	5	0,100
Muestras negativas	19	31	
False positive rate	0,380		
Scalefactor=1,05			
MinNeigh=7	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	39	11	0,220
Muestras negativas	10	40	
False positive rate	0,200		
Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	39	11	0,220
Muestras negativas	9	41	
False positive rate	0,180		

Figura 27. Resultados de las pruebas del clasificador de 20 etapas.

A continuación se realizará un proceso similar con el clasificador de 22 etapas. El primer clasificador estudiado muestra unos resultados bastante positivos, siendo estos un 12% y 20% de tasa de falsos positivos y negativos respectivamente. Sin embargo, se realizaron otras pruebas modificando ligeramente estos valores en caso de que se pudieran obtener mejores resultados.

Con valores cercanos a 1.05 y 3 de scaleFactor y minNeigh se obtienen peores resultados. Distanciándose de estos valores aún más se obtendrán resultados aún peores, por lo que no será necesario realizar más pruebas con este clasificador. Los resultados de estas pruebas se pueden ver en la figura 28.

22 stages			
Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	40	10	0,200
Muestras negativas	6	44	
False positive rate	0,120		
Scalefactor=1,05			
MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	42	8	0,160
Muestras negativas	13	37	
False positive rate	0,260		
Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	41	9	0,180
Muestras negativas	17	33	
False positive rate	0,340		
Scalefactor=1,05			
MinNeigh=5	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	34	16	0,320
Muestras negativas	3	47	
False positive rate	0,060		

Figura 28. Resultados de las pruebas con el clasificador de 22 etapas

Por último, se someterá al clasificador de 25 etapas al mismo proceso. Al aplicar los valores estándar de scaleFactor y minNeigh(1.05 y 3) se obtiene un clasificador extremadamente estricto, con un 0% de falsos positivos y un 46% de falsos negativos. Por ello, se probaron distintas combinaciones de valores en busca de un mejor clasificador, sin embargo, los resultados no superan a los obtenidos con el clasificador anterior. Los resultados se muestran en la figura 29.

25 stages			
Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	27	23	0,460
Muestras negativas	0	50	
False positive rate	0,000		
Scalefactor=1,05			
MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	31	19	0,380
Muestras negativas	1	49	
False positive rate	0,020		
Scalefactor=1,05			
MinNeigh=1	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	35	15	0,300
Muestras negativas	8	42	
False positive rate	0,160		

Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	34	16	0,320
Muestras negativas	3	43	
False positive rate	0,065		
Scalefactor=1,03			
MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	33	17	0,340
Muestras negativas	5	45	
False positive rate	0,100		
Scalefactor=1,03			
MinNeigh=1	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	38	12	0,240
Muestras negativas	15	35	
False positive rate	0,300		

Figura 29. Resultados de las pruebas con el clasificador de 25 etapas

Tras observar los resultados de todos los clasificadores con diferentes parámetros, el que muestra mejores resultados es el de 22 etapas con unos valores de 1.05 y 3 para

scaleFactor y minNeigh, con un un 12% de tasa de falsos positivos y un 20% de tasa de falsos negativos, valores muy cercanos a los 10% y 20% deseados inicialmente. Estos valores se consideran aceptables y se implementarán en el programa final.

5.1.1.4 Implementación del clasificador

Para implementar el clasificador en un programa se debe usar la función CascadeClassifier de OpenCV. Para usarla se usará la función detectMultiScale o detectMultiScale2, dependiendo de la finalidad. Estas funciones deben recibir la imagen que se quiera procesar en escala de grises junto a otros parámetros para ajustar las detecciones. Estos son:

- scaleFactor: especifica cuánto se reduce el tamaño de la imagen en cada escala de imagen. Cuanto más pequeño, más estricto será el algoritmo con los posibles candidatos. Se ha usado 1.05 como se ha indicado anteriormente.
- minNeighbors: especifica cuántos vecinos debe tener cada rectángulo candidato para conservarlo. Este parámetro afectará la calidad de los objetos detectados: un valor más alto da como resultado menos detecciones pero de mayor calidad. Se ha usado un valor de 3, como se ha indicado anteriormente.
- minSize: menor tamaño posible del objeto detectado. Se ha usado (30, 30) para evitar posibles falsos positivos de pequeño tamaño.
- maxSize: mayor tamaño posible del objeto detectado. Este parámetro no se ha especificado, de forma que por defecto el tamaño máximo es la totalidad de la imágen.

La función detectMultiscale devuelve las coordenadas del rectángulo que contiene el objeto detectado. Esto es útil si se quiere visualizar la efectividad del clasificador. La función detectMultiscale2, además del rectángulo también devuelve el número de detecciones. Esto será útil en el programa final.

El siguiente código (figura 30) es un programa para comprobar la efectividad del clasificador cargando un video con la función “VideoCapture” de OpenCV. Seguidamente se carga el clasificador en cascada en una variable con la función “CascadeClassifier” de OpenCV. El siguiente paso es pasar la imagen a escala de grises con la función COLOR_BGR2GRAY ya que el próximo paso requiere que las imágenes sean de este tipo. A la función detectMultiscale se le pasan la imagen anterior ya transformada a escala de grises y los parámetros especificados anteriormente, y se guarda su output (las coordenadas de los rectángulos que encierran los objetos detectados) en una variable. Finalmente, se usan estas coordenadas para dibujar rectángulos alrededor de los objetos detectados y se muestra la imagen.

```
import cv2

# Load video
cap = cv2.VideoCapture(0)
# Load classifier
cascade_chicken = cv2.CascadeClassifier('cascade/cascade.xml')
```

```
while True:
    # Obtain video images
    success, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Chicken detection
    detections = cascade_chicken.detectMultiScale(gray,
scaleFactor=1.05, minNeighbors=3, minSize=(10, 10))

    # Draw the rectangles
    for (x, y, w, h) in detections:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0),
2)
        cv2.putText(img, 'Chicken', (x, y - 10), 2, 0.7, (0, 255,
0), 2, cv2.LINE_AA)

    cv2.imshow('Video test', img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        break
```

Figura 30. Programa para comprobar la efectividad del clasificador

En las figuras 31 y 32 se puede ver el resultado de unas imágenes al ser sometidas al clasificador.

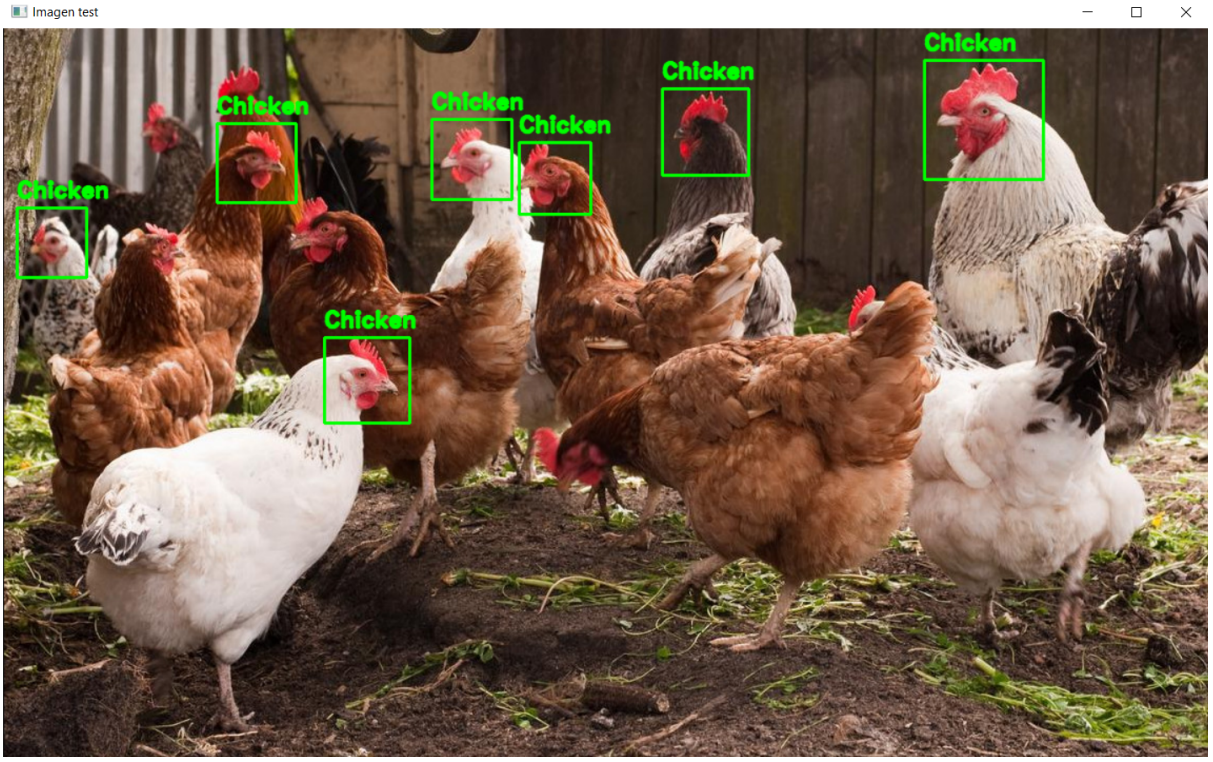

Figura 31. Programa en funcionamiento

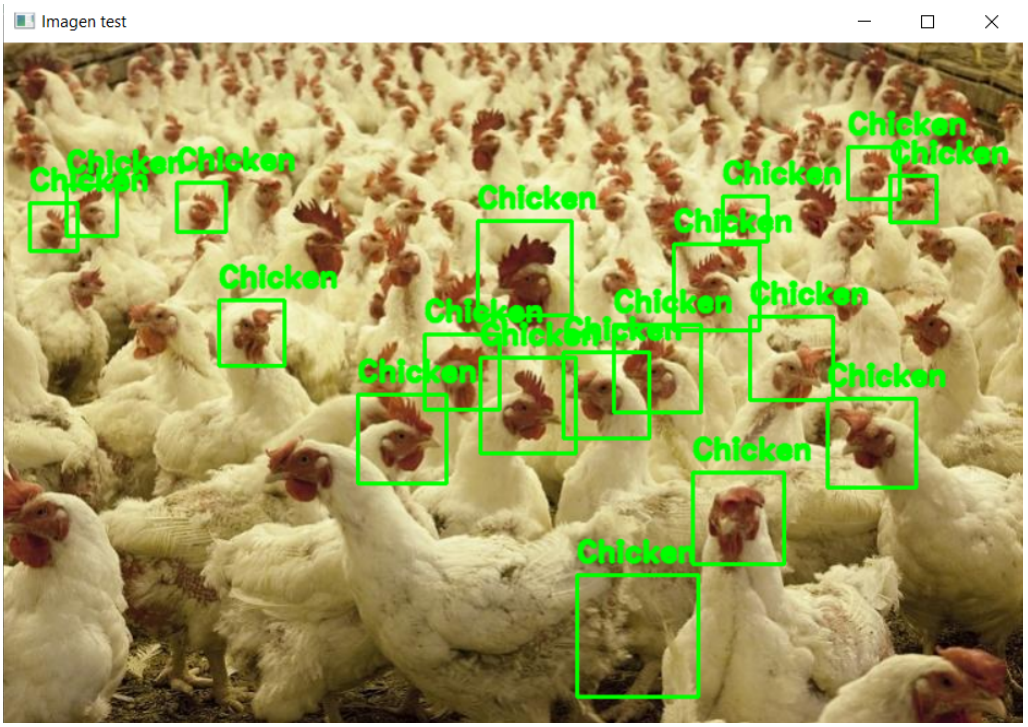

Figura 32. Programa en funcionamiento

5.2 Control de los sensores y actuadores

5.2.1 PIR

Para nuestra solución se va a utilizar como sensor de movimiento un sensor PIR HC SR501. Su control se realiza mediante la librería RPi.GPIO, librería instalada por defecto en Raspberry OS, el sistema operativo de la placa.

Para hacerlo funcionar [10][20], en primer lugar se debe declarar el modo en el que se van a usar los pines. En este caso se ha usado el sistema de numeración de la placa, es decir, enumerando todos los pines en orden de izquierda a derecha y de arriba a abajo. Tras esto, se ha configurado uno de los pines IO para ser usado como input, en este caso el pin 12. Esto se hace con la función “GPIO.setup”. Finalmente, se puede leer su estado con la función “GPIO.input” y usar este valor (1 si detecta movimiento o 0 si no lo detecta) para hacer funcionar el sistema. En la figura 33 se muestra el código usado para verificar su funcionamiento.

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN)       #Read output from PIR motion sensor

while True:
    PIR=GPIO.input(12)
```

```
    if PIR==0:                  #If output from motion sensor is LOW
        print("No intruders",PIR)
        time.sleep(0.1)
    elif PIR==1:                #If output from motion sensor is
HIGH
        print("Intruder detected",PIR)
        time.sleep(0.1)
```

Figura 33. Programa para comprobar el funcionamiento del PIR

5.2.2 Servomotor

El servomotor usado en el proyecto es un SG 90. Su control se realiza también mediante la librería RPi.GPIO. Para hacerlo funcionar [6], en primer lugar se selecciona el modo de numeración de la placa y se configura un pin IO para ser usado como output y otro como una señal PWM (señal que determinará cuánto debe girar el servomotor), en este caso, los pines 13 y 11 respectivamente. Tras asignar el pin a una variable, en este caso “servo”, se puede usar la función “servo.ChangeDutyCycle” para cambiar el ciclo de trabajo del PWM y hacer girar el servo al ángulo que se desee.

El ciclo de trabajo es lo que determina la longitud del pulso de la señal PWM y a su vez, éste determina el ángulo en el que se posiciona el servomotor. Realizando pruebas, se ha podido observar fácilmente que el valor del ciclo de trabajo que posiciona el servo en 0º es 2, y el que lo coloca en 180º es 12. Con estos valores, partiendo de la ecuación de la pendiente de una recta y teniendo en cuenta que el término independiente es 2, se puede definir la ecuación de la recta que relaciona estos valores de esta forma:

$$m = \frac{(y2-y1)}{(x2-x1)} = \frac{(12-2)}{(180-0)} = 1/18$$

$$Duty\ cycle = \frac{ángulo}{18} + 2$$

Sabiendo esto, ya se puede configurar el sistema. Tras el PIR detectar movimiento, el sistema toma una captura de la webcam. Si en ésta hay una gallina, el sistema abre la puerta ajustando el ciclo de trabajo del servomotor a 7 (90/18 + 2). Inmediatamente después, el sistema se congela durante 60 segundos con la función “sleep” de la librería “time” para dar tiempo a los animales para comer. Por otro lado, en caso de no detectar ninguna gallina, la puerta se cierra ajustando el ciclo de trabajo del servomotor a 2 (0/18 + 2).

Después de cualquiera de estos dos escenarios, se sale del bucle y el sistema vuelve a estar a la espera de detectar movimiento con el PIR.

El sistema implementa un pequeño detalle para mejorar la robustez de los movimientos del servo. Tras moverse a una posición, se ajusta el pulso del PWM a 0, de forma que el brazo del servo se queda fijo en la nueva posición. A continuación (figura 34) se muestra el código descrito:

```
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)

# Set pin 13 as an output, and set servo1 as pin 11 as PWM
GPIO.setup(13,GPIO.OUT)
servo1 = GPIO.PWM(13,50) # Note 13 is pin, 50 = 50Hz pulse

# start PWM running, but with value of 0 (pulse off)
servo1.start(0)


# Turn back to 90 degrees
print ("Turning back to 90 degrees for 2 seconds")
servo1.ChangeDutyCycle(5)
time.sleep(0.5)
servo1.ChangeDutyCycle(0)
time.sleep(1.5)

# Turn back to 0 degrees
print ("Turning back to 0 degrees")
servo1.ChangeDutyCycle(2)
time.sleep(0.5)
servo1.ChangeDutyCycle(0)

# Clean things up
servo1.stop()
GPIO.cleanup()
print ("Goodbye")
```

Figura 34. Programa para comprobar el funcionamiento del servomotor

5.3 Programa final

El programa final se muestra a continuación (figura 36). En primer lugar se deben configurar el modo de numeración de la placa, seguido de la inicialización del servomotor, la inicialización de la webcam y la carga del clasificador.

Tras esto, se encuentra el bucle infinito con el que se hará funcionar el sistema. El siguiente diagrama de flujo (figura 35) permite visualizar de manera sencilla su funcionamiento.

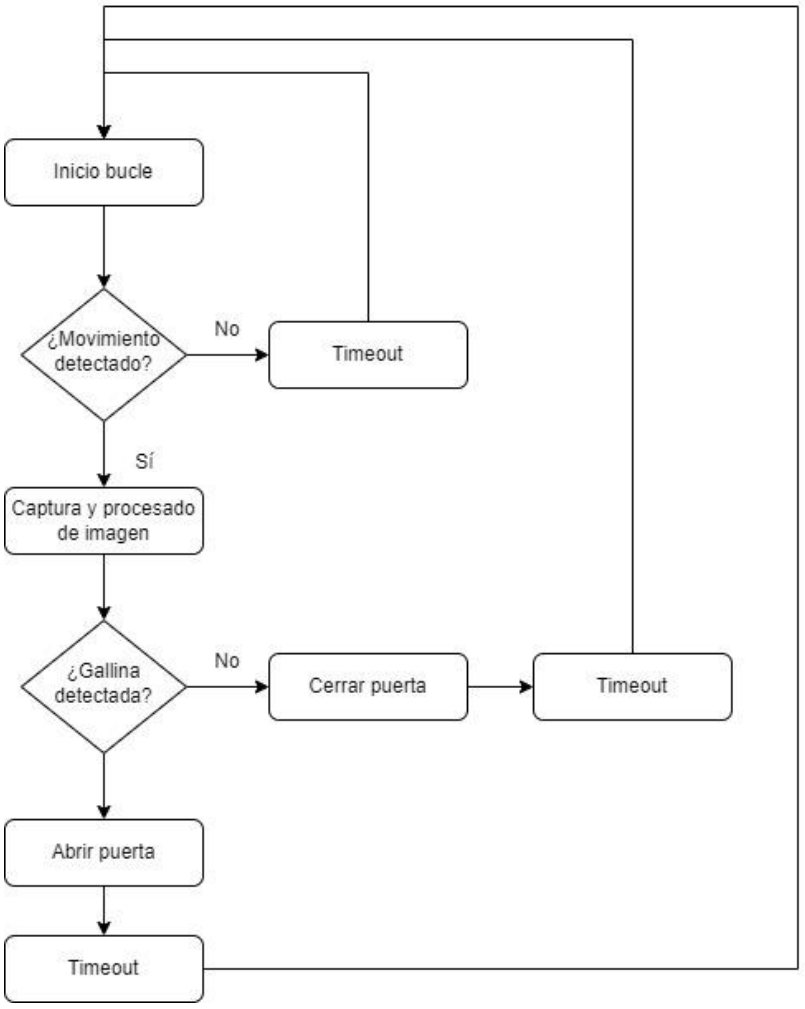

Figura 35. Diagrama de flujo que describe el funcionamiento del programa

En el sistema, si el output del PIR es 1, la imagen de la webcam en ese momento es analizada por el clasificador en cascada. Si éste detecta alguna gallina, la variable detections será igual o mayor que 1. En caso de que lo sea, se posiciona el servomotor en 90º, permitiendo al animal acceder a la comida.

Tras 60 segundos, el sistema vuelve a comprobar la imagen, cerrando o no la puerta dependiendo de si el animal se ha marchado. Si la imagen no detecta ninguna gallina, el servomotor se mantiene en 0º, manteniendo la puerta cerrada y dejando al sistema a la espera de movimiento de nuevo.

```
import cv2
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN)      # PIR config
PIR = GPIO.input(12)         # PIR variable
```

```
GPIO.setup(13,GPIO.OUT)      # Servo config
servo1 = GPIO.PWM(13,50)     # Servo variable. Note 13 is pin, 50
= 50Hz pulse

# Preparing the servo
servo1.start(0)
duty = 2   # 0°

# Waiting for raspberry to boot
time.sleep(60)

# Capturing webcam
webcam = cv2.VideoCapture(0)

# Loading the classifier
cascade_chicken = cv2.CascadeClassifier('cascade.xml')


while True:
    PIR = GPIO.input(12)
    # When movement detected
    if PIR == 1:
        # Capture a frame
        success, img = webcam.read()

        # Convert image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Chicken detection
        rectangles, detections =
cascade_chicken.detectMultiScale2(gray, scaleFactor=1.05,
minNeighbors=3, minSize=(30, 30), maxSize=(500, 500))

        # Activate outputs
        if len(detections) >= 1:
            servo1.ChangeDutyCycle(5)   # 90°
            time.sleep(0.5)
            servo1.ChangeDutyCycle(0)
            time.sleep(10)

        else:
            servo1.ChangeDutyCycle(2)   # 0°
            time.sleep(0.5)
            servo1.ChangeDutyCycle(0)
            time.sleep(10)
        time.sleep(0.1)
```

Figura 36. Programa final

5.4 Implementación en la Raspberry

Con el programa finalizado y listo para ser funcionado, el último paso es preparar la Raspberry para que pueda usarlo. Para ello, se debe conectar una pantalla y un teclado a la Raspberry y ejecutar el programa desde alguna aplicación compatible como Putty. Se debe tener en cuenta que el archivo .xml debe estar en la misma carpeta que el programa principal.

Tras esto, los periféricos se podrán retirar y el sistema se encontraría operativo mientras que reciba alimentación.

5.5 Prototipo

Se ha diseñado mediante la aplicación Tinkercad un prototipo en el que poder integrar todos los componentes y que pueda cumplir la función de este proyecto.

El prototipo en cuestión presenta forma de recipiente. La finalidad de esta decisión es doble. Servirá para proteger a los componentes del exterior al mismo tiempo que almacena el pienso de los animales. Como se puede observar en la figura 37, en la parte frontal del comedero hay tres cavidades. En el hueco superior se colocará la lente de la cámara. El hueco de la derecha corresponde al sensor de movimiento. Por último, el hueco grande central es la conexión con el interior del comedero.

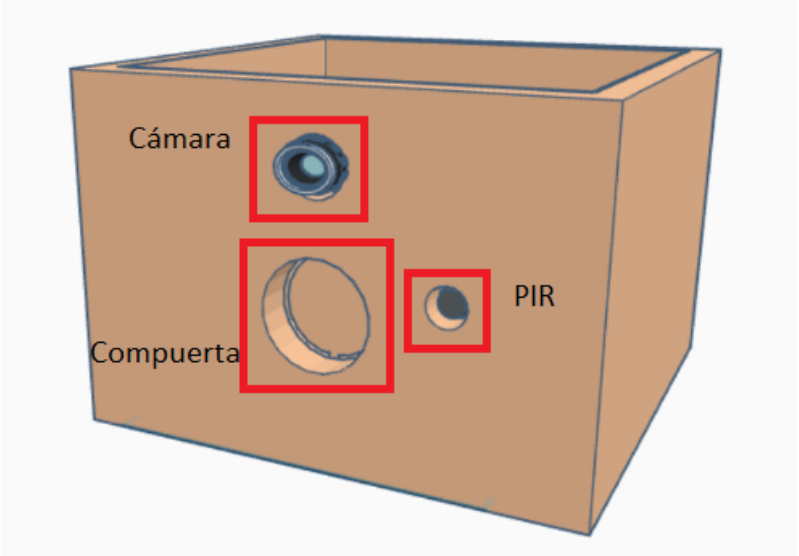

Figura 37. Vista frontal del prototipo diseñado

En la figura 38 se puede observar una vista superior descubierta del prototipo. En ella se puede apreciar la localización del servomotor y de la compuerta anexionada a él. Esta compuerta tiene forma circular con un pequeño brazo, para poder cubrir el orificio del comedero y para adherirse al servomotor. La figura también muestra el movimiento que realizará la compuerta.

Por otro lado, en una de las esquinas, elevada a cierta altura se encontraría una pequeña plataforma. Sobre esta descansará la Raspberry con todos los componentes

conectados a ella. Por último, la fuente de alimentación se encontrará en el exterior del comedero, aunque también debe conectarse a la Raspberry.

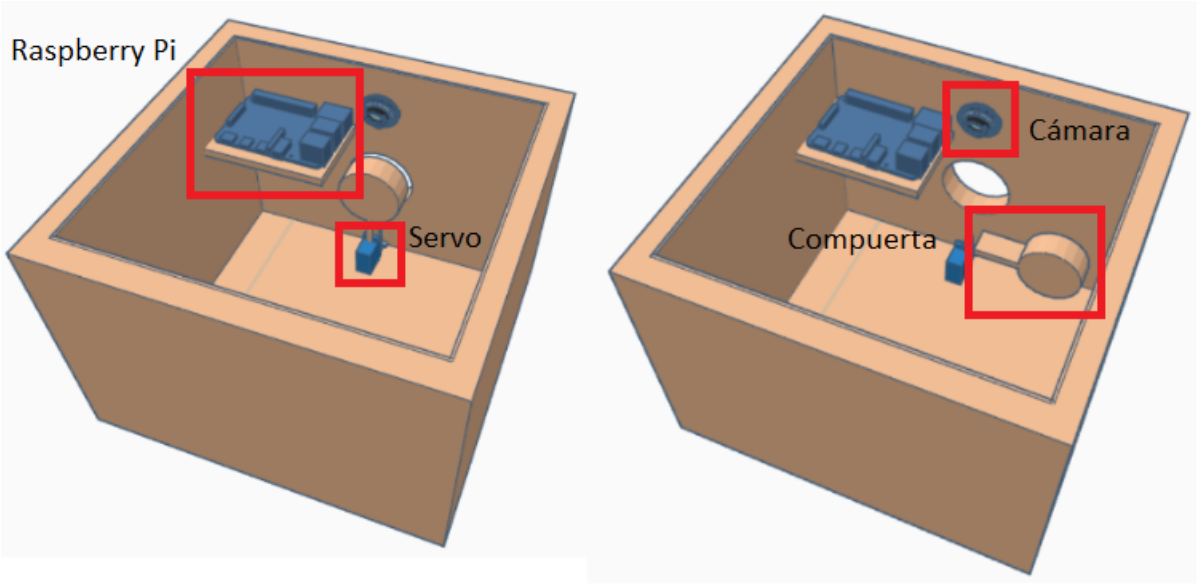

Figura 38. Vista superior descubierta del prototipo diseñado

6. Montaje e implementación del sistema

6.1 Cálculos de consumo energético

Se pretende usar una webcam, un servomotor y un sensor PIR en el sistema. Tras consultar sus fichas técnicas [16][17][19], se puede calcular el consumo energético total del sistema:

- Raspberry Pi 3 B+ en estado de funcionamiento: 900 mA
- Webcam USB 2.0: 500 mA
- PIR: 65 mA
- Servomotor (máximo): 360 mA
- Consumo total: 900 + 500 + 65 + 360 = 1825 mA = 1.825 A

Dado que el sistema no usa muchos componentes, el consumo energético no es demasiado elevado. Al ser inferior a 2A, se puede alimentar todos los componentes desde la placa usando un adaptador de corriente 5V 2A, redistribuyendo la alimentación que ésta obtiene. De esta forma, se simplifica la implementación del sistema a la vez que se reduce su coste. Se usará un adaptador de 5.1V 3A por precaución y por tener un precio muy similar al de 2A.

6.2 Cableado y conexiones

6.2.1 Raspberry Pi

La Raspberry Pi 3 B+ presenta cuatro puertos USB 2.0 y una serie de 40 pines que se usarán para el montaje del sistema. El pinout de la placa se detalla en la figura 39.

Desde estos pines se podrá alimentar los componentes, controlarlos mediante los pines I/O y conectarlos a tierra.

	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Figura 39. Pinout de la Raspberry Pi 3 B +

6.2.2 Webcam

La webcam se conecta a la Raspberry Pi por el puerto USB. Este periférico no requiere de ninguna configuración previa.

6.2.3 PIR

En la figura 40 se puede apreciar la estructura del PIR [9][20]. Éste tiene tres pines. Estos son, de izquierda a derecha, alimentación, output y tierra. El primero se debe conectar a uno de los pines de alimentación de 5V de la Raspberry, en este caso el pin 2. El segundo se debe conectar a uno de los pines GPIO. En este caso se ha usado el pin 12, pero cualquier pin GPIO valdría. Finalmente, el pin de tierra se debe conectar a uno de los pines de tierra de la Raspberry, en este caso el pin 6.

Cabe destacar que hay 3 parámetros que se pueden ajustar físicamente en el sensor. Dos de ellos son dos resistores variables que permiten, respectivamente, ajustar la sensibilidad con la el sensor devuelve un output positivo y ajustar el tiempo del flanco en el que el output se mantiene positivo tras detectar movimiento. Ambos parámetros se reducen con un giro contrario a las agujas del reloj y se aumentan con un giro en sentido de las agujas del reloj.

Para obtener una buena precisión, se ha ajustado la sensibilidad a un 50% y el tiempo de flanco a unos 15 segundos, para que el sistema analice la imagen 3 veces cada vez que se detecte movimiento, en caso de que en la primera vez la gallina pueda estar fuera de la región capturada por la cámara.

El tercer parámetro que puede seleccionarse es el modo de disparo (trigger mode). Se puede seleccionar uno de los dos modos disponibles o no seleccionar ninguno, ya que este parámetro no es imprescindible para el funcionamiento del sensor. Los modos disponibles son “L”, en el que el output se pone en valor lógico alto una sola vez durante el periodo en el que el objeto es detectado, y “H”, en el que el output alcanza el valor lógico alto tantas veces como se detecte movimiento. En este proyecto se ha usado el modo “L”, aunque se podría usar cualquiera ya que este matiz no tiene efecto en el funcionamiento del sistema.

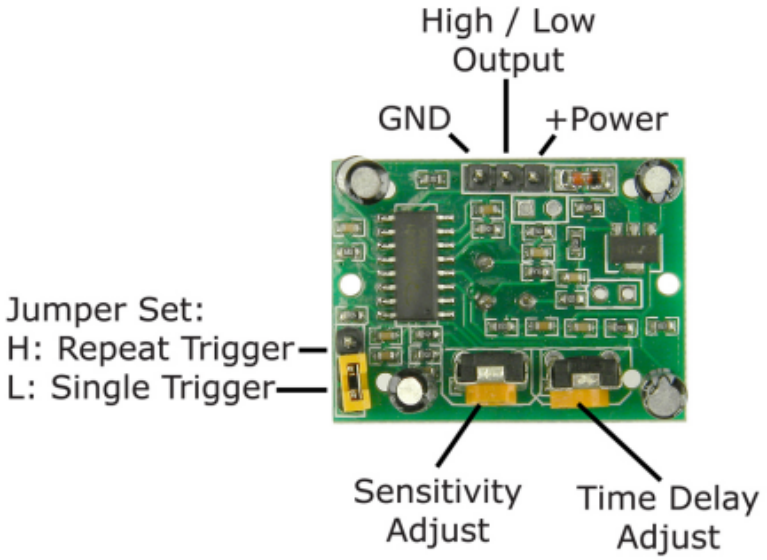

Figura 40. Pines y resistores ajustables del PIR

6.2.4 Servomotor

El servomotor tiene tres cables. Estos son tierra (marrón), alimentación (rojo) y la señal PWM (naranja). El cable de alimentación se debe conectar a uno de los pines de alimentación de 5V de la Raspberry, en este caso el pin 4. El segundo se debe conectar a uno de los pines GPIO. En este caso se ha usado el pin 11, pero cualquier pin GPIO valdría. Finalmente, el pin de tierra se debe conectar a la tierra de la Raspberry. En este caso se ha usado el pin 14.

6.3 Montaje

Para una completa implementación del sistema se recomienda consultar los diagramas proporcionados en el apartado 5.5 de este documento, así como las especificaciones

detalladas en el pliego de condiciones. A continuación se incluye una breve explicación de cómo montar el sistema.

En primer lugar, se debe anexionar la pieza de la puerta del comedero al brazo del servomotor. Tras esto, el servomotor se deberá atornillar con dos tornillos a la base del comedero, alineado con el agujero central del mismo. El PIR y la cámara deberán atornillarse en sus huecos asignados en la cara frontal del comedero.

Por último, la Raspberry se encontrará en el saliente preparado para ella, con todos los componentes conectados a ella.

7. Justificación detallada de los elementos de la solución adoptada

7.1 Raspberry Pi 3 B+

Las placas Raspberry son una opción muy popular para este tipo de proyectos. Las principales ventajas que presentan son su compatibilidad con Python y el apoyo constante por parte de los desarrolladores. Al mismo tiempo, tanto Python y sus librerías como la Raspberry tienen disponibles una gran cantidad de documentación con actualizaciones constantes, puesto que tiene un público tan amplio.

En particular, se ha usado una Raspberry Pi 3 B+ (figura 41). Los motivos detrás de esto son principalmente la dificultad de conseguir la Raspberry Pi 4 por la escasez de componentes y que las pocas que hay disponibles presentan precios un tanto desorbitados. Sin embargo, la Raspberry Pi 3 B+ no tendrá ningún problema en hacer funcionar este proyecto.

Figura 41. Raspberry Pi 3 B+

7.2 Webcam

Dado que el sistema es inmóvil y las gallinas no son animales particularmente veloces, la cámara que se vaya a usar no debe ser demasiado potente. Además, las imágenes al

ser procesadas por nuestro sistema perderán calidad. Sin embargo, debe tener USB para poder conectarse a la Raspberry. Por todo esto, se usará una webcam HD de 30fps (figura 42) con la intención de no encarecer el proyecto.

Figura 42. Cámara usada en el desarrollo de este proyecto

7.3 Servomotor SG 90

La finalidad del servomotor es muy simple. Sólamente ha de abrir y cerrar una compuerta. Por ello, cualquier servomotor de propósito general debería satisfacer esta necesidad. Se usará un servo SG 90 (figura 43) por su popularidad y reducido precio.

Figura 43. Servomotor SG 90

7.4 PIR HC SR501

En el sistema se usa un sensor de movimiento PIR (Passive Infrared) para detectar cuando algo pasa por delante del comedero y reducir el consumo energético a sólo estos momentos.

Para este proyecto se ha usado un HC SR501 (figura 44). Las principales ventajas que presenta son su bajo consumo y el poder ajustar la sensibilidad y el tiempo de respuesta mediante los potenciómetros que se encuentran en su lateral. Además, su

lente esférica permite realizar detecciones en todas las direcciones, lo que aumenta su eficiencia.

Figura 44. Sensor PIR HC SR501

7.5 Fuente de alimentación

Como se ha comentado previamente en el apartado 6.1, se pretende alimentar el sistema usando la red eléctrica. Esto es con el fin de simplificar su implementación y reducir su coste. Concretamente, se usará un adaptador de corriente de 5.1V 3A (figura 45) para poder suministrar corriente a todos los componentes del sistema sin usar cableado o componentes adicionales.

Figura 45. Fuente de alimentación de Raspberry

8. Resultados y conclusiones

Tras concluir el desarrollo del proyecto, se puede afirmar que los resultados obtenidos son satisfactorios. En cuanto al procesado de las imágenes, se ha obtenido un resultado lejos de óptimo. El principal problema que se ha presentado en este aspecto ha sido la recolección de imágenes. Se ha creado el clasificador con 1200 imágenes positivas, cuando para el entrenamiento de clasificadores en cascada se recomienda usar hasta 10.000 imágenes en algunos casos.

Pese a haber usado un número reducido de imágenes, observando los resultados obtenidos en las pruebas del apartado 5.1.1.3.1 y en la tabla de la figura 46, se puede afirmar que el sistema se muestra responsivo e identifica las gallinas correctamente en la mayoría de los casos. No obstante, cabe mencionar que al presentar al clasificador con imágenes cotidianas, o bien, fuera del ámbito agrícola, su efectividad y precisión se ve reducida considerablemente, ya que no ha sido entrenado para ello. Sin embargo, esto no es un problema debido a que el comedero en uso no se verá en esas situaciones.

Clasificador final	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	40	10	0,200
Muestras negativas	6	44	
False positive rate	0,120		

Figura 46. Efectividad del clasificador final

En los anexos nº8 se muestran más detalladamente los resultados obtenidos en las pruebas del clasificador.

En cuanto a la implementación física, tanto el PIR como el servomotor han sido programados perfectamente y responden correctamente en el 100% de los casos. La programación de la Raspberry y su interacción con el resto de componentes también ha resultado exitosa.

Los dos objetivos principales de este proyecto eran la creación de un sistema de visión artificial capaz de distinguir gallinas de otros animales y su posterior implementación en un sistema físico que controle el acceso al interior de un comedero en función de la respuesta del primero. Ambos objetivos se han cumplido satisfactoriamente.

8.1 Trabajos futuros

Como en todo proyecto, especialmente aquellos relacionados con la inteligencia artificial, siempre hay lugar para mejoras. A continuación se detallan aspectos que pueden mejorar el resultado del desarrollo de este proyecto:

- El principal aspecto con margen de mejora es el clasificador en cascada. Con un número de imágenes mayor y con mayor capacidad computacional se podría crear un clasificador mucho más eficiente y preciso. El margen de mejora es casi infinito, siempre que se puedan obtener más y más imágenes.
- Actualmente el programa se debe poner en marcha manualmente conectando una pantalla y un teclado a la Raspberry. A modo de mejora se podría configurar la placa para que ejecute el programa automáticamente cuando ésta se encienda.

- A costa de aumentar el precio del proyecto, se podría optar por usar componentes más caros o una Raspberry de mayor gama. Esto dotaría al sistema de una mejora en rendimiento.
- Actualmente, el sistema funciona alimentado por la red eléctrica. Se puede mejorar la portabilidad del sistema sustituyendo la fuente de alimentación actual por una batería portátil, siempre que tenga un puerto micro-USB y suministre 2 A como mínimo.
- Siguiendo la línea de la propuesta anterior, se podría alimentar el sistema mediante una fuente de alimentación solar. Esta alternativa, además de ser una fuente de energía renovable, resulta idónea para el contexto en el que se va a encontrar el sistema: al aire libre y durante el día.
- Tanto el diseño del prototipo como su llevada a cabo pueden optimizarse sustancialmente invirtiendo los recursos adecuados en ello.

9. Bibliografía

[1] Acres, L. (2017, October 11). *Diy Automatic Chicken Feeder (easiest way to feed your chickens with a 5~gallon Bucket)*. YouTube. Retrieved August 22, 2022, from https://www.youtube.com/watch?v=__gpaK3dVSo

[2] *Adobe Stock*. (n.d.). Stock photos, royalty-free images, graphics, vectors & videos. Retrieved June 12, 2022, from https://stock.adobe.com/

[3] alalek, garybradski, kirillkornyakov, mshabunin, & relrotciv. (n.d.). *OpenCV Files*. SourceForge. Retrieved May 28, 2022, from https://sourceforge.net/projects/opencvlibrary/files/3.4.14/

[4] *Amazon - Comedero para gallinas*. (2022, January 17). Amazon. Retrieved August 22, 2022, from https://www.amazon.es/s?k=comedero+gallinas&i=lawngarden&__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=3C93VIT93PJXW&sprefix=comedero+gallinas%2Clawngarden%2C88&ref=nb_sb_noss_1

[5] *Comedero Automático Surefeed para perros y gatos*. (n.d.). Sure Petcare. Retrieved August 22, 2022, from https://www.surepetcare.com/es-es/comederos/comedero-con-microchip

[6] ExplainingComputers. (2020, January 12). *Raspberry Pi Servo Motor Control*. YouTube. Retrieved July 6, 2022, from https://www.youtube.com/watch?v=xHDT4CwjUQE

[7] Gamez, M. J. (2022, May 24). Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible. Retrieved February 6, 2023, from https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[8] Gandhi, A. (2021, May 19). *Data Augmentation | How to use Deep Learning when you have Limited Data*. Nanonets. Retrieved June 14, 2022, from https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[9] *HC SR501 PIR Motion Sensor Module Pinout, Datasheet & Details*. (2020, April 25). Components Info. Retrieved July 13, 2022, from https://www.componentsinfo.com/hc-sr501-module-pinout-datasheet/

[10] *How to Interface a PIR Motion Sensor With Raspberry Pi GPIO | Raspberry Pi*. (2018, March 23). Maker Pro. Retrieved July 6, 2022, from https://maker.pro/raspberry-pi/tutorial/how-to-interface-a-pir-motion-sensor-with-raspberry-pi-gpio

[11] *How To Make Gaussian Noise On Image - C# Guide*. (2021, January 9). Epoch Abuse. Retrieved July 19, 2022, from https://epochabuse.com/gaussian-noise/

[12] Learn Code By Gaming. (2020, August 22). *Training a Cascade Classifier - OpenCV Object Detection in Games #8*. YouTube. Retrieved May 26, 2022, from https://www.youtube.com/watch?v=XrCAvs9AePM

[13] OMES. (2020, July 28). *Como crear tu propio DETECTOR DE OBJETOS con Haar Cascade | Python y OpenCV*. YouTube. Retrieved May 26, 2022, from https://www.youtube.com/watch?v=v_cwOq06g9E&t=811s

[14] *OpenCV 3 Object Detection : Face Detection using Haar Cascade Classfiers - 2020*. (n.d.). BogoToBogo. Retrieved June 10, 2022, from https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php

[15] *OpenCV: Cascade Classifier Training*. (n.d.). OpenCV documentation. Retrieved May 26, 2022, from https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html

[16] *Power Consumption Benchmarks*. (n.d.). Raspberry Pi Dramble. Retrieved July 15, 2022, from https://www.pidramble.com/wiki/benchmarks/power-consumption

[17] *Raspberry Pi 3 Pinout, Features, Specifications & Datasheet*. (2018, April 26). Components101. Retrieved July 10, 2022, from https://components101.com/microcontrollers/raspberry-pi-3-pinout-features-datasheet

[18] Roboflow. (2020, December 21). *The Best Way to Annotate Images for Object Detection*. YouTube. Retrieved June 2, 2022, from https://www.youtube.com/watch?v=pJaM06FG-wQ

[19] *Servo Motor Micro SG90*. (n.d.). ProtoSupplies. Retrieved July 18, 2022, from https://protosupplies.com/product/servo-motor-micro-sg90/

[20] Tech With Tim. (2019, June 12). *How to Use a PIR Motion Sensor with Raspberry Pi*. YouTube. Retrieved July 6, 2022, from https://www.youtube.com/watch?v=Tw0mG4YtsZk

[21] *Unsplash. La fuente de Internet de imágenes de uso libre.* (n.d.). Unsplash. Retrieved June 10, 2022, from https://unsplash.com/es

[22] *Zhengyuan Poultry Equipment*. (n.d.). Cangzhou Zhengyuan Poultry Equipment CO.,LTD|chicken|Equipment. Retrieved August 14, 2022, from https://www.poultryequipment.cn/

[23] *Cascade Classifier*. (n.d.). OpenCV. Retrieved June 10, 2022, from https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

Pliego de condiciones

1. Objeto

El objeto del pliego de condiciones de este proyecto tiene la finalidad de indicar las condiciones de los materiales empleados para el funcionamiento y las condiciones en la que se debe ejecutar el proyecto. Se va a proceder a detallar las especificaciones técnicas de los materiales y componentes utilizados, como se deben actuar para ponerlos en funcionamiento y cómo realizar un correcto control de calidad.

2. Condiciones de los materiales

- Se debe usar una placa Raspberry Pi 3B + o, en su defecto, una de mayor potencia. En caso de utilizar otra, asegurarse de que el consumo energético total del sistema no supere los 3 amperios.
- La cámara usada ha de tener cable USB y funcionar a 720p y 30fps como mínimo.
- El servomotor usado ha de ser un SG90 o de condiciones similares. En caso de utilizar otro, asegurarse de que el consumo energético total del sistema no supere los 3 amperios.
- El sensor de infrarrojos usado ha de ser un HC SR501 o uno de condiciones similares. En caso de utilizar otro, asegurarse de que el consumo energético total del sistema no supere los 3 amperios.
- Para fijar la Raspberry se deben utilizar tornillos de 2.5mm. Para el PIR se deben usar tornillos de 1mm. En caso de usar otro sensor de infrarrojos, se deben usar los tornillos que éste necesite.

2.1 Control de calidad

El control de calidad de los componentes electrónicos debe ser verificado por el proveedor de cada uno. En caso de obtener un componente defectuoso contactar con el proveedor. Se puede comprobar el funcionamiento tanto de la cámara como del servo y del PIR con los códigos anexos (nº 4, 5 y 6) a este documento.

3. Condiciones de ejecución

La ejecución del sistema se debe llevar a cabo en dos fases. La primera de ellas es el montaje de los diferentes componentes electrónicos conforme se ha descrito previamente en el apartado 6.

La segunda fase consiste en la integración de los componentes electrónicos en el comedero. En el apartado 5.5 se incluye un esquema de la disposición de los componentes en el comedero. Tanto el sensor PIR como la cámara deben colocarse en

sus huecos designados en la parte frontal del comedero. El servo, con la compuerta anexionada a él, se debe colocar en la parte central de la base del comedero, alineado con el hueco por donde los animales accederán al pienso. La placa se colocará en la plataforma destinada a sostenerla.

Tras la instalación del sistema, se debe verificar que todos los componentes han sido incorporados correctamente, asegurándose de que están sujetos firmemente tanto entre ellos como al comedero.

3.1 Control de calidad

Para la correcta evaluación de la ejecución del sistema y sus fases se realiza un procedimiento escalonado de verificación del dispositivo de monitorización. Se debe realizar el montaje de los componentes electrónicos del sistema, habiendo comprobado previamente su correcto funcionamiento.

Seguidamente, se debe comprobar el correcto montaje de los componentes en la estructura general del sistema, ya que una mala colocación o fijación puede causar que el sistema no funcione correctamente. Se debe realizar la verificación de funcionamiento del sistema mediante una prueba controlada. Esto se puede realizar usando una gallina o, en su defecto, una imagen de una.

Con el sistema evaluado en las diferentes fases designadas ya es posible ejecutar la integración del dispositivo en un entorno real, es decir, un recinto avícola con gallinas. Esta ejecución supone la finalización completa del proyecto.

4. Pruebas y ajustes finales

Una vez integrado y comprobado el correcto funcionamiento del sistema, se evalúa su funcionamiento en una aplicación real como es una granja, donde pueden actuar agentes externos como el sol, el viento o los propios animales.

El sistema instalado en diferentes lugares con diferentes condiciones pueden provocar la necesidad de ajustar del dispositivo. En caso de mostrarse no responsivo, se puede ajustar ligeramente la sensibilidad del sensor PIR con el potenciómetro indicado. Más allá de esto, no se recomienda modificar ningún aspecto del sistema sin consultar con el personal adecuado.

Presupuesto

1. Precios unitarios

1. Precios unitarios			
Referencia	Uds.	Descripción	Precio (€)
1.1 Materiales			
m1	u.	Raspberry Pi 3B+	43,99
m2	u.	Tarjeta SD 32GB	4,99
m3	u.	Webcam HD 30fps	14,99
m4	u.	Servomotor SG90	3,99
m5	u.	PIR HC SR501	2,29
m6	u.	Set cables electrónica	1,20
m7	u.	Tornillo 2.5mm	0,40
m8	u.	Tuerca 2.5mm	0,40
m9	u.	Tornillo 1mm	0,44
m10	u.	Tuerca 1mm	0,44
m11	u.	Transformador Raspberry 5.1V 3A	7,19
m12	u.	Black box	4,00
1.2 Mano de obra			
h1	h	Ingeniero industrial	40,00
h2	h	Técnico de programación	35,00

2. Desglose de precios unitarios

2. Desglose de precios unitarios					
Referencia	Uds.	Descripción	Precio (€)	Cantidad	Total
d1	u.	Sistema de visión artificial			
Materiales					
m1	u.	Raspberry Pi 3B+	43,99	1	43,99
m2	u.	Tarjeta SD 32GB	4,99	1	4,99
m3	u.	Webcam HD 30fps	14,99	1	14,99
m4	u.	Servomotor SG90	3,99	1	3,99
m5	u.	PIR HC SR501	2,29	1	2,29
m6	u.	Set cables electrónica	1,20	1	1,2
m7	u.	Tornillo 2.5mm	0,40	4	1,6
m8	u.	Tuerca 2.5mm	0,40	4	1,6
m9	u.	Tornillo 1mm	0,44	2	0,88
m10	u.	Tuerca 1mm	0,44	2	0,88
m11	u.	Transformador Raspberry 5.1V 3A	7,19	2	14,38
m12	u.	Black box	4,00	1	4,00
Mano de obra					
h1	h	Ingeniero industrial	40,00	12	480,00
h2	h	Técnico de programación	35,00	12	420,00
Gastos generales de fabricación					
	%	Costes de producción sobre el coste directo	10	994,79	99,48
Beneficios industriales (5-10%)					
	%	Beneficio industrial	7	994,79	69,64

IVA (21%)					
	%	IVA sobre el coste directo	21	994,79	208,91
Coste total de ejecución					1372,81

3. Cantidades

3. Cantidades			
Referencia	Uds.	Descripción	Cantidad
d1	u.	Sistema de visión artificial	1

4. Coste total

4. Coste total					
Referencia	Uds.	Descripción	Precio (€)	Cantidad	Total
d1	u.	Sistema de visión artificial	1372,81	1	1372,81
Coste total del proyecto					1372,81

El presupuesto final que supone la totalidad del proyecto es de mil trescientos setenta y dos con ochenta y un euros.

Anexos

1. Funciones para realizar data augmentation

```
import os
import cv2
import numpy as np

# Flip images horizontally
def flip_images(PATH='./positive/'):
    for filename in os.listdir(PATH):
        image = cv2.imread(PATH+filename)
        flipped_image = cv2.flip(image, 1)
        cv2.imwrite(PATH+"flipped_"+filename, flipped_image)


# Add gaussian noise
def gaussian_noise(PATH='./positive/'):
    for filename in os.listdir(PATH):
        image = cv2.imread(PATH+filename)
        mean = 0
        var = 25
        sigma = var ** 0.5
        gaussian = np.random.normal(mean, sigma,
(image.shape[0],image.shape[1]))

        noisy_image = np.zeros(image.shape, np.float32)

        if len(image.shape) == 2:
                noisy_image = image + gaussian
        else:
            noisy_image[:, :, 0] = image[:, :, 0] + gaussian
            noisy_image[:, :, 1] = image[:, :, 1] + gaussian
            noisy_image[:, :, 2] = image[:, :, 2] + gaussian

            cv2.normalize(noisy_image, noisy_image, 0, 255,
cv2.NORM_MINMAX, dtype=-1)
        noisy_image = noisy_image.astype(np.uint8)

        cv2.imwrite(PATH+"gaussian_"+filename, noisy_image)
```

2. Funciones para la creación de los ficheros de texto

```
import os

# Create .txt for positive images
def generate_negative_description_file():
    # abre y sobreescribe el archivo
    with open('neg.txt', 'w', encoding='utf-8') as f:
        # loopea todos los nombres de archivo
        for filename in os.listdir('negative'):
            f.write('negative/' + filename + '\n')


# Functions to launch programs

# Create .txt for positive images
# opencv\build\x64\vc15\bin\opencv_annotation.exe
--annotations=pos.txt --images=positive/


# Create .vec for positive images
# opencv\build\x64\vc15\bin\opencv_createsamples.exe -info pos.txt
-w 30 -h 30 -num 1500 -vec pos.vec

# Create.xml
# opencv\build\x64\vc15\bin\opencv_traincascade.exe -data cascade/
-vec pos.vec -bg neg.txt -w 30 -h 30 -precalcValBufSize 6800
-precalcIdxBufSize 6800 -numPos 900 -numNeg 2100 -numStages 22
```

3. Prueba del funcionamiento del clasificador en cascada

```
import cv2

# Load video
cap = cv2.VideoCapture(0)

# Load classifier
cascade_chicken = cv2.CascadeClassifier('cascade/cascade.xml')

while True:
    # Obtain video images
    success, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    # Chicken detection
    detections = cascade_chicken.detectMultiScale(gray,
scaleFactor=1.05, minNeighbors=3, minSize=(10, 10))

    # Draw the rectangles
    for (x, y, w, h) in detections:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(img, 'Chicken', (x, y - 10), 2, 0.7, (0, 255,
0), 2, cv2.LINE_AA)

    cv2.imshow('Video test', img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        break
```

4. Prueba de funcionamiento de la cámara

```
import cv2

# Load video
cap = cv2.VideoCapture(0)

while True:
    # Obtain video images
    success, img = cap.read()

    cv2.imshow('Video test', img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        break
```

5. Prueba del funcionamiento del servomotor

```
import RPi.GPIO as GPIO
import time

# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)

# Set pin 13 as an output, and set servo1 as pin 11 as PWM
GPIO.setup(13,GPIO.OUT)
servo1 = GPIO.PWM(13,50) # Note 13 is pin, 50 = 50Hz pulse

# start PWM running, but with value of 0 (pulse off)
servo1.start(0)
```

```
# Turn back to 90 degrees
print ("Turning back to 90 degrees for 2 seconds")
servo1.ChangeDutyCycle(5)
time.sleep(0.5)
servo1.ChangeDutyCycle(0)
time.sleep(1.5)

# Turn back to 0 degrees
print ("Turning back to 0 degrees")
servo1.ChangeDutyCycle(2)
time.sleep(0.5)
servo1.ChangeDutyCycle(0)

# Clean things up
servo1.stop()
GPIO.cleanup()
print ("Goodbye")
```

6. Prueba del funcionamiento del sensor PIR

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN)          #Read output from PIR motion
sensor

while True:
    PIR=GPIO.input(12)
    if PIR==0:                  #When output from motion sensor is
LOW
        print("No intruders",PIR)
        time.sleep(0.1)
    elif PIR==1:                #When output from motion sensor is
HIGH
        print("Intruder detected",PIR)
        time.sleep(0.1)
```

7. Programa principal

```
import cv2
import RPi.GPIO as GPIO
import time
```

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN)      # PIR config
PIR = GPIO.input(12)         # Variable para el PIR
GPIO.setup(13,GPIO.OUT)      # Servo config
servo1 = GPIO.PWM(13,50)     # Variable para el servo. Note 13 is
pin, 50 = 50Hz pulse

# Prepare the servo
servo1.start(0)
duty = 2   # 0°

# Boot the raspberry
time.sleep(60)

# Capture video
webcam = cv2.VideoCapture(0)

# Load the classifier
cascade_chicken = cv2.CascadeClassifier('cascade.xml')


while True:
    PIR = GPIO.input(12)
    # When movement detected
    if PIR == 1:
        # Capture a frame
        success, img = webcam.read()

        # Convert image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Chicken detection
        rectangles, detections =
cascade_chicken.detectMultiScale2(gray, scaleFactor=1.05,
minNeighbors=3, minSize=(30, 30), maxSize=(500, 500))


        # Activate outputs
        if len(detections) >= 1:
            servo1.ChangeDutyCycle(5)   # 90°
            time.sleep(0.5)
            servo1.ChangeDutyCycle(0)
            time.sleep(10)

        else:
            servo1.ChangeDutyCycle(2)   # 0°
            time.sleep(0.5)
```

```
            servo1.ChangeDutyCycle(0)
            time.sleep(10)
        time.sleep(0.1)
```

8. Eficiencia de los clasificadores en cascada

8.1 Clasificador 20 stages

Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	45	5	0,100
Muestras negativas	19	31	
False positive rate	0,380		
Scalefactor=1,05			
MinNeigh=7	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	39	11	0,220
Muestras negativas	10	40	
False positive rate	0,200		
Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	39	11	0,220
Muestras negativas	9	41	
False positive rate	0,180		

8.2 Clasificador 22 stages

Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	40	10	**0,200**
Muestras negativas	6	44	
False positive rate	**0,120**		
Scalefactor=1,05			
MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	42	8	0,160
Muestras negativas	13	37	
False positive rate	0,260		
Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	41	9	0,180
Muestras negativas	17	33	
False positive rate	0,340		
Scalefactor=1,05			
MinNeigh=5	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	34	16	0,320
Muestras negativas	3	47	
False positive rate	0,060		

8.3 Clasificador 25 stages

Scalefactor=1,05			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	27	23	0,460
Muestras negativas	0	50	
False positive rate	0,000		
Scalefactor=1,05			
MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	31	19	0,380
Muestras negativas	1	49	
False positive rate	0,020		
Scalefactor=1,05			
MinNeigh=1	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	35	15	0,300
Muestras negativas	8	42	
False positive rate	0,160		
Scalefactor=1,03			
MinNeigh=3	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	34	16	0,320
Muestras negativas	3	43	
False positive rate	0,065		
Scalefactor=1,03			

MinNeigh=2	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	33	17	0,340
Muestras negativas	5	45	
False positive rate	0,100		
Scalefactor=1,03			
MinNeigh=1	Resultado positivo	Resultado negativo	False negative rate
Muestras positivas	38	12	0,240
Muestras negativas	15	35	
False positive rate	0,300		