Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DISEÑO DE UN SISTEMA DE SEGURIDAD EN EL HOGAR BASADO EN IoT Y CREACIÓN DE PROTOTIPO

Alumno: **Eduardo Fúnez Fernández**

Tutor: Prof. D. Ildefonso Ruano Ruano
Depto.: Ingeniería de Telecomunicación

Febrero, 2022

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Departamento de Ingeniería de Telecomunicación

Don ILDEFONSO RUANO RUANO como tutor del Trabajo Fin de Grado titulado: DISEÑO DE UN SISTEMA DE SEGURIDAD EN EL HOGAR BASADO EN IOT Y CREACIÓN DE PROTOTIPO; autoriza su presentación para defensa y evaluación en el Campus Científico – Tecnológico de Linares.

Linares, Febrero de 2022.

El alumno: El tutor:

Eduardo Fúnez Fernández. Ildefonso Ruano Ruano.

Índice

1. Resumen 1
1.1. Resumen 1
1.2. Abstract 1
2. Introducción 3
2.1. Internet of Things (IoT) en la actualidad 5
2.2. IoT en líneas generales 6
2.3. Funcionamiento de IoT 6
2.4. Inteligencia Artificial (IA) 7
3. Objetivos 9
4. Estudio Previo y Justificación 10
4.1. Sensores de presencia 10
4.2. Controlador y Pasarela (Controller and Gateway) 13
4.2.1. ASUS Tinker Board S 13
4.2.2. Arduino UNO R3 14
4.2.3. Raspberry Pi 4B 16
4.2.4. Decisión 17
4.3. Cámaras 17
4.4. Altavoz y luz 19
4.5. Cerradura 20
4.6. Comunicaciones 21
4.6.1. MQTT (Message Queuing Telemetry Transport) 21
4.6.2. HTTP (Hypertext Transfer Protocol) 21
4.6.3. AMQP (Advanced Message Queuing Protocol) 21
4.6.4. DDS (Data Distribution Service) 21
4.6.5. Decisión 22
5. Materiales y métodos 23
5.1. Materiales 23
5.2. Métodos 25

5.2.1. Bot de Telegram 25
5.2.2. MongoDB 28
5.2.3. Cerradura de solenoide 31
5.2.4. Detección de movimiento (sensores PIR) 34
5.2.5. Simulación de presencia 37
5.2.6. Detección de movimiento (cámara de seguridad) 41
5.2.7. Inteligencia Artificial (IA) 45
5.2.8. Transmisión en directo del módulo de cámara 55
5.2.9. Registro 57
5.3. Presupuesto 59
5.3.1. Materiales 59
5.3.2. Mano de obra 59
5.3.3. Presupuesto total 60
6. Resultado y discusión 61
7. Conclusión y líneas futuras 62
8. Bibliografía IX
Anexos 63
Anexo I: Guía de instalación 64
I.I. Introducción 65
I.II. Preparación inicial 65
I.III. Instalación de OpenCV 67
I.IV. Instalación de dependencias 69
I.V. Configuraciones varias 70
Anexo II: Manual de uso 73
II.I. Introducción 74
II.II. Comando */start* 74
II.III. Comando */secret* 75
II.IV. Comando */chsecret* 76
II.V. Comando */admin* 76

II.VI. Comando */chadmin* 76
II.VII. Comando */rmuser* 77
II.VIII. Comando */stream* 78
II.IX. Comando */lock* 79
II.X. Comando */pres* 79
II.XI. Comando */detmo* 81
II.XII. Comando */pir* 82
II.XIII. Enviar imagen 83
II.XIV. Comando */rmfile* 83
II.XV. Comando */rmfolder* 85
II.XVI. Comando */train* 86
II.XVII. Reconocimiento facial 87
II.XVIII. Comando */log* 88
II.XIX. Comando */reboot* 90
Anexo III: Planos 91

Índice de figuras

Figura 1: Esquema general del sistema de seguridad 4
Figura 2: Vista de planta de la maqueta 4
Figura 3: Perspectiva isométrica de la maqueta 4
Figura 4: Pronóstico de gasto en soluciones IoT en todo el mundo de 2017 a 2025 5
Figura 5: Arquitectura IoT 7
Figura 6: Funcionamiento de un sensor por microondas 10
Figura 7: Diagrama del funcionamiento de un sensor PIR 11
Figura 8: A la izquierda, sensor PIR y a su derecha, un sensor piroeléctrico 12
Figura 9: Patillaje de un sensor PIR 12
Figura 10: ASUS Tinker Board S 13
Figura 11: Arduino UNO R3 14
Figura 12: Raspberry Pi 4B 16
Figura 13: Módulo de cámara de Raspberry Pi 18
Figura 14: Cerradura de solenoide 20
Figura 15: Esquema de las comunicaciones del sistema de seguridad 22
Figura 16: Esquema general de los componentes electrónicos del sistema de seguridad 24
Figura 17: Conversación bot de Telegram 26
Figura 18: Recepción de comandos en el bot 27
Figura 19: Código QR para acceder al bot de Telegram 27
Figura 20: Flujograma del comando /start 28
Figura 21: Resultado comando /start cuando el usuario no está en la base de datos 28
Figura 22: Usuario se añade a la base de datos y gana acceso a comandos básicos 29
Figura 23: Usuario se añade como administrador y gana acceso completo a las funciones del bot 29
Figura 24: Usuario administrador en base de datos MongoDB 30
Figura 25: Inicialización de la conexión con la base de datos 30
Figura 26: Flujograma del comando /lock 31
Figura 27: Configuración del GPIO 23 32
Figura 28: Cambio de estado de la cerradura de solenoide 32
Figura 29: Menú de la cerradura en el bot de Telegram 32
Figura 30: Gestión respuesta del usuario (menú cerradura) 33

Figura 31: Flujograma del comando /pir 34
Figura 32: Configuración de los GPIO de los sensores PIR 34
Figura 33: Gestión respuesta del usuario (menú sensores PIR) 35
Figura 34: Controlador de la función de detección de presencia mediante sensores PIR 36
Figura 35: Función de detección de movimiento del sensor PIR S1 36
Figura 36: Flujograma de la simulación de presencia aleatoria 37
Figura 37: Función de simulación de presencia aleatoria 38
Figura 38: Función sig_handler 39
Figura 39: Flujograma de la simulación de presencia con alarma 39
Figura 40: Función de simulación de presencia con alarma 39
Figura 41: Función del sensor PIR 1 para la simulación de presencia 40
Figura 42: Función de simulación de presencia manual 40
Figura 43: Flujograma del comando /detmo 41
Figura 44: Función detmo_cont 42
Figura 45: Inicialización de variables en la función detmo 42
Figura 46: Tratamiento de imagen de la función detmo 43
Figura 47: Función de corrección gamma 43
Figura 48: Detección de movimiento de la función detmo 44
Figura 49: Función para la gestión de imágenes recibidas 45
Figura 50: Función para el listado de /rmfolder 46
Figura 51: Inicialización del detector facial 47
Figura 52: Inicialización del modelo Torch de incrustación visual 47
Figura 53: Carga del dataset de entrenamiento 48
Figura 54: Cuantificación de las caras (blob) 48
Figura 55: Detección facial y extracción de región de interés 49
Figura 56: Extracción del vector 128-D de la cara 49
Figura 57: Serialización de las codificaciones en un fichero pickle 49
Figura 58: Codificación de las etiquetas 50
Figura 59: Finalización del entrenamiento 50
Figura 60: Flujograma para el reconocimiento facial 51
Figura 61: Inicialización del detector facial y el modelo de incrustación facial 52
Figura 62: Inicialización de los modelos de reconocimiento facial y el codificador de etiquetas 52
Figura 63: Captura de imagen de entrada para reconocimiento facial 53
Figura 64: Extracción región de interés para el reconocimiento facial 53
Figura 65: Elaboración de la predicción del reconocimiento facial 54

Figura 66: Notificación al usuario del reconocimiento facial 54
Figura 67: Flujograma del comando /stream 55
Figura 68: Función webmain 56
Figura 69: Pantalla de sitio web peligroso 57
Figura 70: Aviso dentro de la página web 57
Figura 71: Función log 57
Figura 72: Raspberry Pi Imager 65
Figura 73: Configuración para el módulo de cámara 66
Figura 74: "Your Authtoken" en la página web de ngrok 70
Figura 75: Comando Authtoken 70
Figura 76: Código de “startup.sh” 71
Figura 77: /boot/config.txt 71
Figura 78: Archivo autostart 72
Figura 79: Resultado del comando /start 74
Figura 80: Resultado del comando /secret 75
Figura 81: Resultado del comando /chsecret 76
Figura 82: Resultado del comando /admin 76
Figura 83: Resultado del comando /chadmin 76
Figura 84: Primer menú del comando /rmuser 77
Figura 85: Segundo menú del comando /rmuser 77
Figura 86: Mensaje de confirmación del comando /rmuser 77
Figura 87: Resultado del comando /stream 78
Figura 88: Página web del comando /stream 78
Figura 89: Resultado del comando /lock 79
Figura 90: Mensaje de confirmación del comando /lock 79
Figura 91: Menú principal para activar la simulación de presencia 79
Figura 92: Menú para la frecuencia de la simulación de presencia aleatoria 80
Figura 93: Menú de confirmación para la simulación de presencia con alarma 80
Figura 94: Menú de confirmación para la simulación de presencia manual 80
Figura 95: Menú para desactivar la simulación de presencia 81
Figura 96: Menú para el comando /detmo 81
Figura 97: GIF de la detección de movimiento captada por la cámara de seguridad 82
Figura 98: Menú del comando /pir 82
Figura 99: Mensaje de detección de movimiento del comando /pir 83
Figura 100: Mensaje al enviar imagen al bot 83
Figura 101: Menú carpeta de el comando /rmfile 83

Figura 102: Menú imagen del comando /rmfile 84
Figura 103: Menú de confirmación del comando /rmfile 84
Figura 104: Menú carpeta del comando /rmfolder 85
Figura 105: Menú de confirmación del comando /rmfolder 85
Figura 106: Mensajes del comando /train 86
Figura 107: Mensaje entrenamiento finalizado del comando /train 86
Figura 108: Resultado del reconocimiento facial 87
Figura 109: Menú año del comando /log 88
Figura 110: Menú mes del comando /log 88
Figura 111: Menú día del comando /log 88
Figura 112: Menú de confirmación del comando /log 89
Figura 113: Envío de archivo de registro del comando /log 89
Figura 114: Archivo de registro 89
Figura 115: Menú de confirmación del comando /reboot 90
Figura 116: Mensaje de reinicio del comando /reboot 90

Índice de tablas

Tabla 1: Presupuesto para la compra de los materiales 59
Tabla 2: Presupuesto para la mano de obra 59
Tabla 3: Presupuesto total 60
Tabla 4: Presupuesto total para implementaciones futuras 60

1. Resumen

1.1. Resumen

En este Trabajo de Fin de Grado (TFG), se ha realizado el diseño de un sistema de seguridad con elementos *Internet of Things* (IoT) basado en Raspberry Pi y se ha creado un prototipo funcional del mismo. Para la realización de dicho sistema de seguridad, se llevó a cabo un estudio previo de las comunicaciones, tecnologías y dispositivos necesarios para su funcionamiento. Además, se implementó un bot de Telegram para la gestión a distancia del sistema de seguridad por parte del usuario.

El sistema de seguridad incluye sensores y actuadores de distintos tipos. Entre los sensores existen detectores de presencia mediante infrarrojos pasivos y cámaras, mientras que los actuadores utilizados son una cerradura automática, un pulsador, luces y un altavoz.

Este sistema realiza detección de movimiento, grabación y reproducción remota de imágenes gracias a los sensores. Su estado se podrá notificar a los usuarios, quienes podrán configurar el modo de funcionamiento del mismo a través de Internet.

La cerradura accionada por el sistema de seguridad puede ser controlada a distancia mediante el bot de Telegram. El usuario puede, además, controlar un simulador de presencia para disuadir posibles intentos de robo basada en la reproducción de audio y el control de iluminación del recinto.

Adicionalmente, se ha implementado una inteligencia artificial (IA) usando la cámara de la entrada, que se puede activar cada vez que se acciona un pulsador y realiza un reconocimiento facial de la imagen captada. El resultado de dicho reconocimiento facial se puede enviar al usuario mediante el bot de Telegram, así como una transmisión en directo de las imágenes captadas.

1.2. Abstract

In this Thesis, the design of a Raspberry Pi based security system with *Internet of Things* (IoT) elements and the creation of a functional prototype of said system has been carried out. For the development of said security system, a preliminary study of the communications, technologies and devices needed for its operation was conducted. In

addition, a Telegram bot was implemented for the user to manage the security system remotely.

The security system includes sensors and actuators of different types. Among the sensors there are passive infrared presence detectors and cameras, while the actuators used are an automatic lock, a push button, lights and a speaker.

This system performs motion detection, recording and remote reproduction of images thanks to the sensors. Its status can be notified to users, who can configure its operating mode through the Internet.

The lock activated by the security system can be controlled remotely through the Telegram bot. The user can also control a presence simulator to deter potential theft attempts based on audio playback and lighting control of the premises.

Additionally, an artificial intelligence (AI) has been implemented using the entrance camera, which can be activated every time a button is pressed and performs a facial recognition of the captured image. The result of said facial recognition can be sent to the user through the Telegram bot, as well as a livestream of the captured images.

2. Introducción

El objeto del presente TFG es la implementación práctica de las tecnologías IoT en un sistema de seguridad. La motivación académica para la realización de dicho TFG es la exploración de un ejemplo de aplicación práctica del IoT. Este está cada vez más presente en elementos cotidianos de la sociedad y, por triviales que puedan parecer sus aplicaciones, hacen más fácil y eficiente la realización de distintas tareas.

La realización de este TFG se ha basado en el uso de la Raspberry Pi como nexo para controlar todos los procesos que se ejecutan en el sistema y hacerlos accesibles al usuario final actuando como controlador y gateway del sistema, para ello se ha usado el lenguaje de programación Python. La Raspberry Pi se controla mediante un bot de Telegram, el cual hace de interfaz de usuario, permitiendo que los usuarios configuren el funcionamiento del sistema a su gusto a la vez que actúa como centro de notificaciones. Mediante el bot de Telegram, se puede notificar al usuario en tiempo real los eventos que se producen en función de la configuración del sistema, dándole un margen de acción necesario en el caso de que haya que actuar ante una situación adversa (un acceso al recinto, por ejemplo).

Un elemento esencial de un sistema basado en IoT es la captación de datos del entorno para su procesado y posterior notificación al usuario. La captación de datos se hace mediante sensores. En este TFG se hace uso de varios sensores de presencia infrarrojos pasivos y dos cámaras de seguridad, que se verán en profundidad en capítulos posteriores.

Para permitir la interacción del usuario en el sistema, se ha incluido en el mismo una serie de actuadores que pueden ser accionados a conveniencia del usuario. Los actuadores del presente TFG son una cerradura electrónica, luces y un altavoz. Además, el sistema incluye un pulsador que activa un algoritmo de inteligencia artificial para realizar un reconocimiento facial utilizando una cámara situada en la entrada de la vivienda.

Una vez presentados a rasgos generales los elementos incluidos en el sistema, la Figura 1 muestra un esquema general del esquema de seguridad que puede ayudar a obtener una mejor comprensión de todos los elementos descritos hasta el momento que son utilizados el mismo:

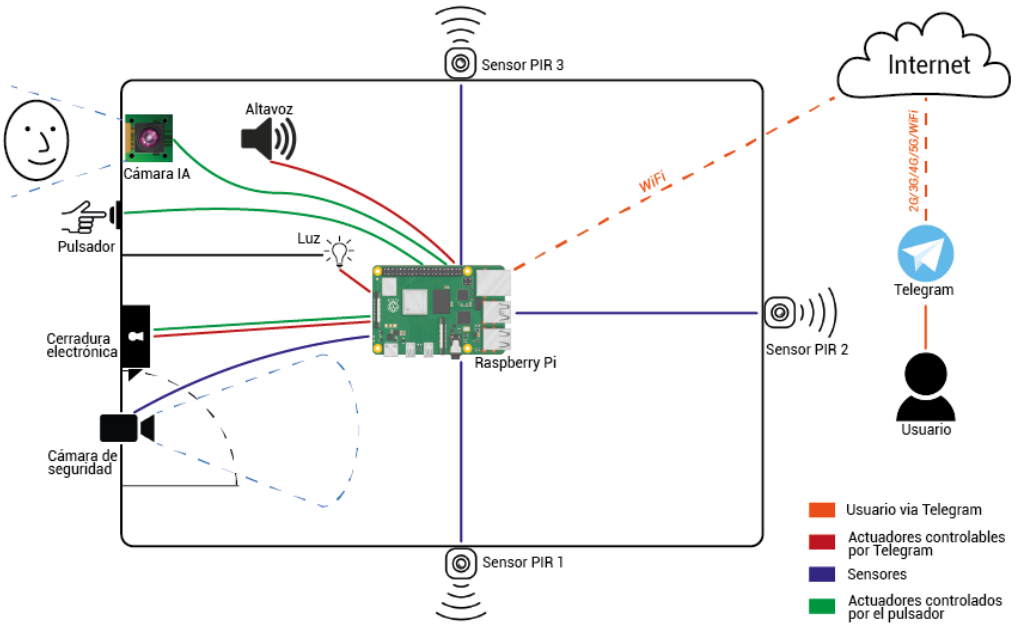


Figura 1: Esquema general del sistema de seguridad

Teniendo en cuenta el esquema de la Figura 1, a continuación, se muestran, en las Figuras 2 y 3, un par de fotografías de la maqueta creada en las que se pueden distinguir algunos de los componentes antes descritos:

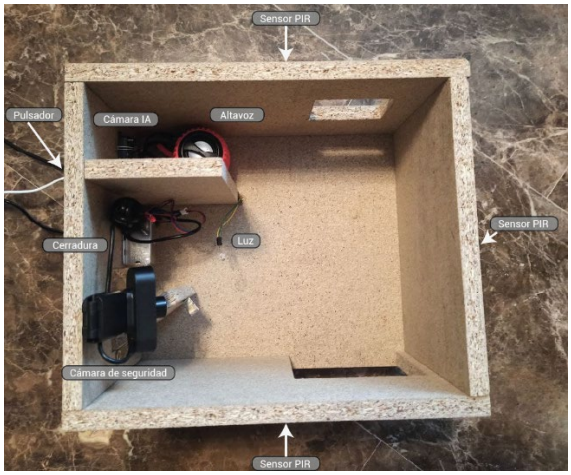


Figura 2: Vista de planta de la maqueta

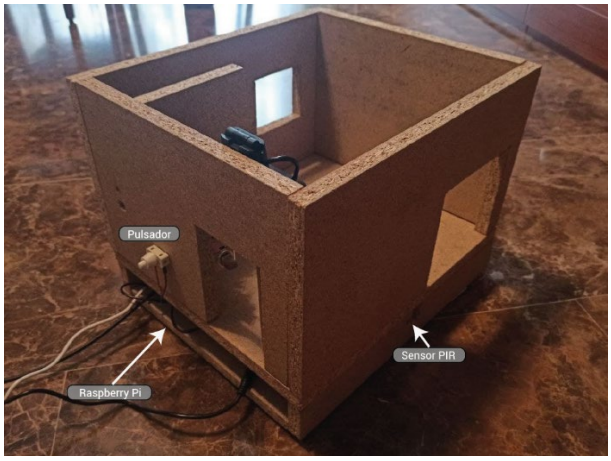


Figura 3: Perspectiva isométrica de la maqueta

Todas las conexiones electrónicas entre los distintos componentes utilizados en la maqueta, tanto sensores como actuadores, y la Raspberry Pi están ubicados en un falso suelo situado debajo del suelo visto de la maqueta.

A continuación, en los siguientes subapartados, se van a explicar brevemente algunos conceptos que pueden ayudar a la comprensión del presente TFG.

2.1. Internet of Things (IoT) en la actualidad

El concepto de las tecnologías IoT se ha popularizado en los últimos años. Si bien el concepto de *Internet of Things* como tal ha existido desde 1999, cuando Kevin Ashton (directivo de Procter & Gamble) utilizó el término por primera vez, los avances tecnológicos de los últimos años han propiciado su crecimiento en popularidad [1].

Las limitaciones que tenían las tecnologías IoT se han reducido en estos años, haciendo que su aplicación práctica sea asequible y accesible en diferentes áreas. Desde el ocio hasta la medicina, pasando por la seguridad doméstica, por nombrar algunas áreas. IoT tiene cada vez más potencial y el pronóstico para el gasto en soluciones IoT es creciente en años venideros.

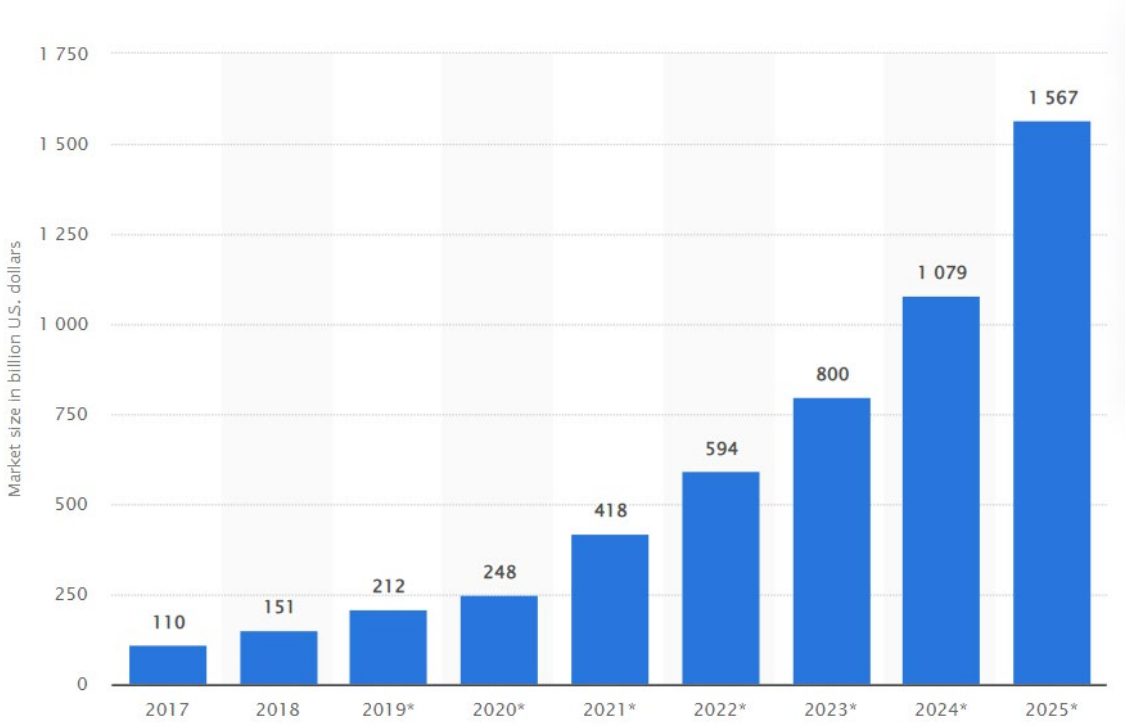


Figura 4: Pronóstico de gasto en soluciones IoT en todo el mundo de 2017 a 2025

Fuente: https://www.statista.com/statistics/976313/global-iot-market-size/

2.2. IoT en líneas generales

Un ecosistema IoT consiste en dispositivos inteligentes conectados a internet que usan procesadores, sensores, actuadores y hardware de comunicación, para recoger o enviar datos e incluso actuar en función a los mismos. Estos dispositivos funcionan sin necesidad de intervención humana, aunque en algunos casos el usuario puede interactuar con ellos para darles instrucciones o acceder a los datos recogidos [2].

Uno de los puntos fuertes de IoT es el fácil acceso a la información por parte del usuario. Se puede acceder desde cualquier dispositivo en cualquier momento, siempre y cuando se disponga de acceso a Internet.

Lo anterior, combinado con la autonomía que por naturaleza presentan los sistemas IoT, su sensibilidad a los cambios en su entorno gracias a los sensores, su asequibilidad y su escalabilidad, hacen de las tecnologías IoT una solución rentable y cómoda en escenarios que requieran de un monitoreo constante.

2.3. Funcionamiento de IoT

Como se ha mencionado anteriormente en el subapartado *2.2. IoT en líneas generales*, uno de los pilares esenciales de un ecosistema IoT es la capa de sensores. Esta es la capa más baja en la arquitectura IoT y se compone, precisamente, de sensores que captan datos en tiempo real para ser procesados y/o almacenados. Los sensores tienen la capacidad de tomar diferentes mediciones: temperatura, humedad, calidad del aire o presión, por poner algunos ejemplos. Los sensores se pueden agrupar de acuerdo a su propósito específico: sensores de entorno, corporales, domésticos, etc. La interconexión entre los sensores se puede realizar mediante protocolos orientados a comunicaciones Machine-to-Machine (M2M) como ZigBee, Bluetooth, MQTT o LoraWAN, aunque también se puede hacer por cable si las circunstancias lo permiten.

La información captada en la capa anterior, se transmitirá al elemento Gateway del ecosistema. Este elemento actúa de frontera entre la capa de sensores y la red. Se encarga generalmente de recibir los datos y procesarlos para su posterior envío o guardado en un sistema de almacenamiento, como podría ser una base de datos. Algunos ejemplos de Gateways son los microcontroladores, los procesadores de señales y los módulos SIM.

Desde los elementos Gateway, los datos se enviarán mediante red (WI-FI, GSM/GPRS, Ethernet, …) a la capa de gestión de servicio. Esta capa se encarga de analizar la información procesada para proveer información en forma de eventos o datos

contextuales al usuario. Además, en esta capa se le otorga al usuario acceso y control sobre los datos. En esta capa también se ha de proteger la privacidad de los datos mediante técnicas de autenticación, anonimización de datos o cifrado [3].

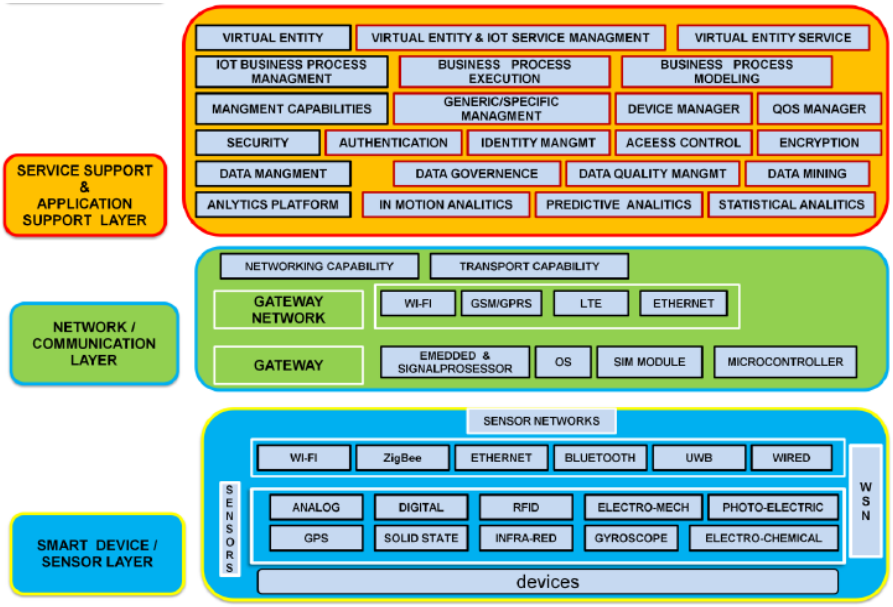


Figura 5: Arquitectura IoT

Fuente: Patel K, Patel S, "Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges".

2.4. Inteligencia Artificial (IA)

En el contenido de este TFG también se estudiará la implementación de una inteligencia artificial, por lo que es conveniente definir brevemente lo que comprende este concepto.

La inteligencia artificial no se puede definir como una tecnología en específico, sino como una combinación de varias tecnologías y técnicas diferentes de recolección, análisis y preparación de datos [4].

La rama que más prevalencia tiene bajo el paraguas de la IA es el *Machine Learning* o Aprendizaje Automático (AA). En el AA, una vez los datos han sido procesados, se usan para entrenar un modelo, que servirá para clasificar, predecir o conocer características desconocidas de datos.

Para el entrenamiento del modelo, se ha usado un algoritmo de aprendizaje que se adecúa al objetivo para el que se quiere usar dicho modelo. A grandes rasgos, se pueden distinguir tres tipos de algoritmos [5]:

- **Aprendizaje Supervisado**: Este tipo de algoritmo consiste en el entrenamiento del modelo a partir de un conjunto de datos que contienen tanto datos predictores (entradas) como datos objetivo (salidas). Usando este conjunto de datos, se genera una función que relaciona los datos de entrada con los datos de salida deseados. El modelo se entrena hasta que se llega al nivel de precisión deseado. Algunos ejemplos de este tipo de algoritmo son SVM, KNN, árbol de decisión y regresión.
- **Aprendizaje no supervisado**: En este tipo de algoritmo, no se tiene ninguna variable a predecir. Simplemente se introduce un conjunto de datos con entradas y se espera que el algoritmo encuentre estructuras y patrones en los datos para agruparlos. Algunos ejemplos de este tipo de algoritmo son K-Means y *clustering* jerárquico.
- **Aprendizaje por refuerzo**: En este tipo de algoritmo, el modelo se expone a un entorno en el que se entrena por prueba y error. El modelo aprende de intentos anteriores e intenta conseguir mejores resultados por medio de un sistema de recompensas y castigos. Un ejemplo de este tipo de algoritmo son los procesos de decisión de Markov.

En el entrenamiento del modelo, la elección de un conjunto de datos adecuado es tan importante como la elección del algoritmo de entrenamiento. Puede significar la diferencia entre un modelo adecuado y un modelo inservible. Además de cómo de adecuado sea el conjunto de datos, otro factor a tener en cuenta es la cantidad de datos en el conjunto. Cuanto mayor sea el número de datos en el conjunto, más claras serán las relaciones y los patrones a distinguir por el modelo y, por lo tanto, se obtendrán mayor precisión y mejores resultados.

3. Objetivos

El objetivo principal de este TFG es el diseño y creación del prototipo de un sistema de seguridad doméstico basado en Raspberry Pi. Este sistema se ha diseñado teniendo en cuenta las necesidades que pueda requerir el usuario final, como pueden ser el monitoreo de la seguridad de la vivienda mediante detección de movimiento o la disuasión de potenciales intentos de robo mediante simulación de presencia. Además, cabe mencionar que en dicho sistema de seguridad se ha implementado un bot de Telegram para facilitar la gestión del sistema por parte del usuario final.

El logro de este objetivo principal ha dependido de los siguientes objetivos parciales:

- Se ha llevado a cabo un estudio previo de las tecnologías y dispositivos a usar en el diseño del sistema de seguridad, teniendo en cuenta las necesidades de un hipotético usuario final.
- Posteriormente al estudio previo, se realizó la elección justificada de las comunicaciones, sensores, actuadores y dispositivos de procesamiento de acuerdo con los requisitos necesarios para la realización del TFG, cumpliendo con los límites de presupuesto marcados por el tutor.
- Se ha obtenido el prototipo del sistema de seguridad. Esto se ha conseguido mediante:
    - El diseño del prototipo del sistema de seguridad con los elementos elegidos en el objetivo parcial anterior.
    - La creación del prototipo de acuerdo con el diseño.
    - La prueba y depuración del prototipo creado y de las comunicaciones usadas.
- Con el prototipo terminado, se han hecho los análisis y estudios pertinentes de los resultados para la elaboración de la memoria.
- Se ha procedido a la elaboración de la memoria, documentos y códigos a entregar del presente TFG.

4. Estudio Previo y Justificación

Para la elección de los elementos que componen el sistema de seguridad, se ha realizado un estudio previo de cada una de las alternativas y se ha justificado dicha elección. El resumen de los resultados de este estudio queda plasmado a continuación.

4.1. Sensores de presencia

Los dos tipos de sensores de presencia más relevantes son los sensores PIR y los sensores por microondas. Se diferencian en la forma por la que detectan movimiento y por este motivo cada uno de ellos es adecuado para su instalación en un entorno determinado.

Los detectores de movimiento por microondas emiten señales de microondas y miden el tiempo de retorno de la señal al sensor continuamente. Estos sensores son adecuados para su instalación en un entorno cerrado, ya que cuando alguien entra en la zona de detección, se provoca una alteración en el haz de microondas que modifica el tiempo de retorno y se detecta movimiento [6].

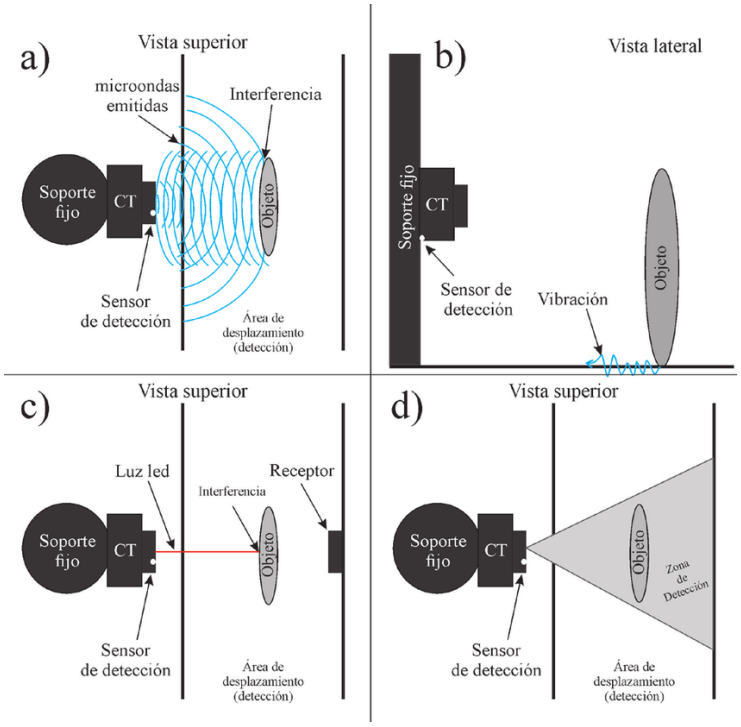


Figura 6: Funcionamiento de un sensor por microondas

Fuente: García-Grajales J, Buenrostro A, "Las cámaras trampa y su avance tecnológico en favor de la conservación. Ciencia y Mar XXII (65): 53-61."

Los sensores PIR, en cambio, son un tipo de sensores que reaccionan ante determinadas fuentes de energía tales como el calor humano. Los sensores PIR son mejores que los de microondas para instalaciones exteriores, ya que, al ser un entorno abierto, es posible que el sensor de microondas se active por error.

El sensor recibe la variación de las radiaciones infrarrojas del área que cubre mediante su componente principal: los sensores piroeléctricos.

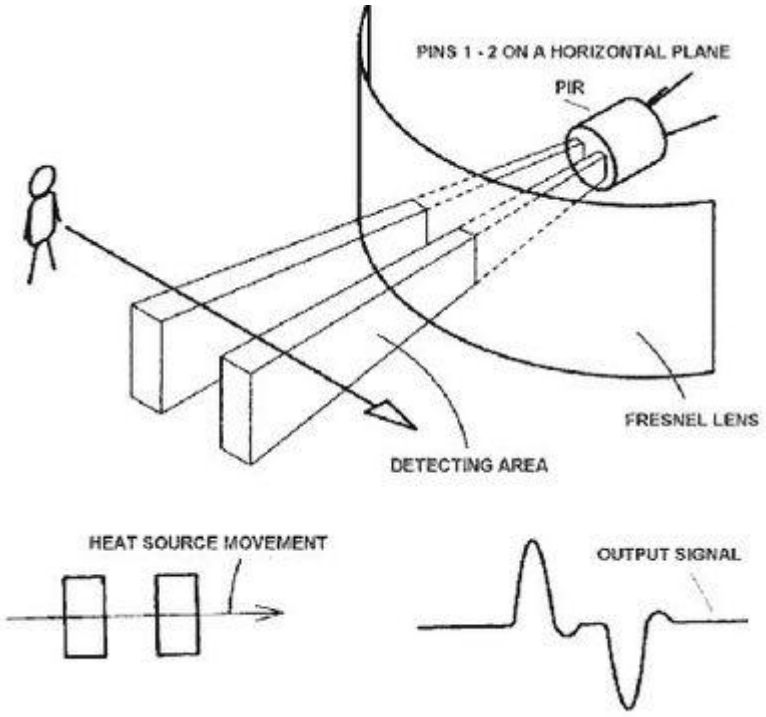


Figura 7: Diagrama del funcionamiento de un sensor PIR

Fuente: https://www.prometec.net/sensor-pir/

Para entender el funcionamiento de los sensores piroeléctricos, se debe entender el concepto de piroelectricidad. La piroelectricidad se puede definir como la aparición de cargas superficiales en una dirección determinada cuando experimenta un cambio de temperatura, causando así un cambio en su polarización eléctrica. Para la detección de radiación térmica, se disponen dos electrodos metálicos en dirección perpendicular a la de polarización, formando un condensador que actúa como sensor térmico [7].

Figura 8: A la izquierda, sensor PIR y a su derecha, un sensor piroeléctrico

Fuente: https://www.zonamaker.com/arduino/modulos-sensores-y-shields/sensor-pir-para-la-deteccion-de-presencia

En un sensor PIR, el sensor piroeléctrico va detrás de una lente de Fresnel de plástico para ampliar su ángulo de detección. En la Figura 8, se pueden ver también los controladores del temporizador y la sensibilidad del sensor PIR (en naranja, de izquierda a derecha). En la Figura 9 se pueden ver con más claridad los diferentes elementos de un sensor PIR.

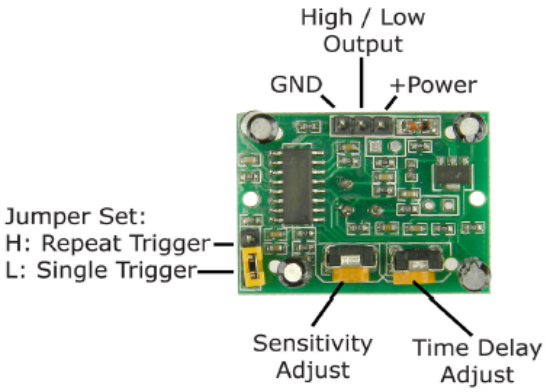


Figura 9: Patillaje de un sensor PIR

Fuente: https://www.mpja.com/download/31227sc.pdf

Dado que el objetivo con los sensores de movimiento es captar movimiento en el perímetro exterior de la vivienda y, por lo tanto, en un espacio abierto, se utilizarán para el diseño del sistema de seguridad sensores PIR, en concreto tres sensores del modelo HC-SR501 [8].

4.2. Controlador y Pasarela (Controller and Gateway)

Para el presente TFG se usará un SBC (Single-Board Computer) para actuar de controlador y pasarela en el sistema de seguridad. Este tipo de ordenadores se caracterizan por ser muy básicos, asequibles y potentes, además de ser de un tamaño reducido. Son capaces de hacer cosas que cualquier ordenador convencional podría hacer, pero además son capaces de interactuar con su entorno. Por esta razón se usan para una variedad de proyectos diferentes, desde estaciones meteorológicas hasta servidores de archivos.

A continuación, se muestran brevemente algunas de las alternativas posibles para cubrir este rol.

4.2.1. ASUS Tinker Board S

La ASUS Tinker Board S es un SBC con un factor de forma pequeño que ofrece un rendimiento muy alto comparado con otros SBC similares.

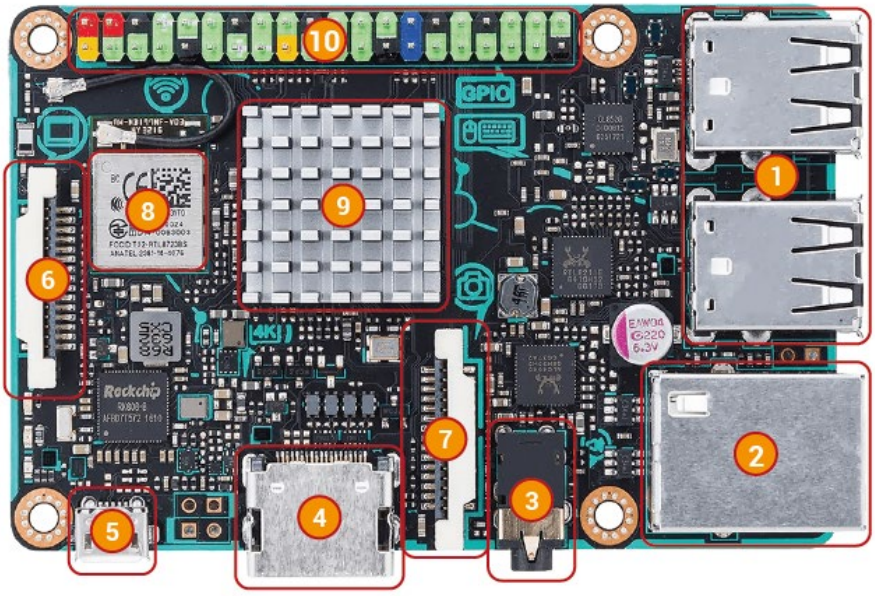


Figura 10: ASUS Tinker Board S

Está compuesta por [9]:

1. **Puertos USB**: Dispone de 4 puertos USB 2.0.
2. **Puerto Ethernet:** La Tinker Board S tiene un puerto Gigabit Ethernet.

3. **A/V:** Ofrece un conector para entrada de voz y salida de audio que soporta audio de 24 bits y 192 kHz.
4. **Puerto HDMI:** Tiene un puerto HDMI al que se puede conectar un monitor que soporta resoluciones HD y UHD.
5. **Entrada de alimentación por Micro USB**
6. **Puerto MIPI DSI**: Es un puerto con el que su puede conectar un monitor.
7. **Puerto MIPI CSI:** Es el puerto que se usará para conectar una cámara MIPI.
8. **Conectividad inalámbrica:** Dispone de WLAN 802.11b/g/n y Bluetooth 4.0.
9. **Procesador:** La Tinker Board S está equipada con un procesador de cuatro núcleos basado en ARM, el Rockchip RK3288.
10. **Pines GPIO:** Se dispone de una cabecera de 40 pines GPIO (General-Purpose Input/Output) en los que se podrán conectar diversos componentes electrónicos.

4.2.2. Arduino UNO R3

La placa Arduino Uno R3 es una placa electrónica basada en el microcontrolador de Atmel, ATmega328.

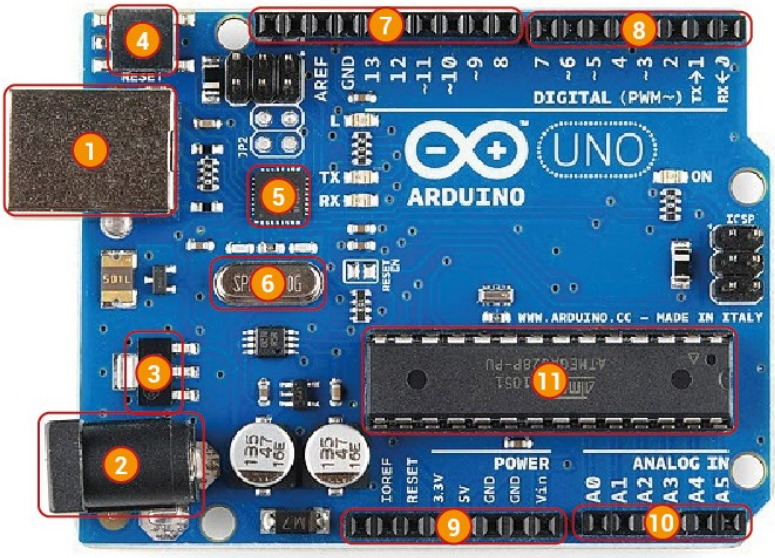


Figura 11: Arduino UNO R3

El Arduino UNO R3 está compuesto por [10]:

1. **Puerto USB**: Utilizado tanto para conectar con un ordenador y transferir o cargar los programas al microcontrolador como para dar electricidad al Arduino.
2. **Puerto de corriente continua**: Este puerto es el que se usa para darle electricidad a la placa si no se usa alimentación USB.
3. **Regulador de tensión**: Este sirve para controlar la cantidad de electricidad que se envía a los pines, con lo que asegura que no se estropee lo que se conecte a dichos pines.
4. **Botón de reset**: Sirve para inicializar nuevamente el programa cargado en el microcontrolador de la placa.
5. **Chip de interfaz USB**: Es el encargado de controlar la comunicación con el puerto USB.
6. **Reloj oscilador**: Es el elemento que hace que el Arduino vaya ejecutando las instrucciones. Es el encargado de marcar el ritmo al cual se debe ejecutar cada instrucción del programa.
7. **Pines GPIO de entrada**: Aquí se conectarán los sensores de los que el Arduino recibirá información del entorno.
8. **Pines GPIO de salida**: Aquí se conectarán los actuadores que el Arduino controlará.
9. **Zócalo de tensión**: Aquí estarán los pines GPIO con los que se alimentará el circuito.
10. **Entradas analógicas**: Zócalo con distintos pines GPIO de entrada analógica que permiten leer entradas analógicas.
11. **Microcontrolador:** Es el procesador que se encarga de ejecutar las instrucciones de los programas. En el Arduino UNO R3 se usa el chip ATmega328.

4.2.3. Raspberry Pi 4B

La Raspberry Pi 4B (RPi 4B) es un SBC muy potente y asequible, razones por las cuales se usa en muchos proyectos relacionados con la robótica y la automatización.

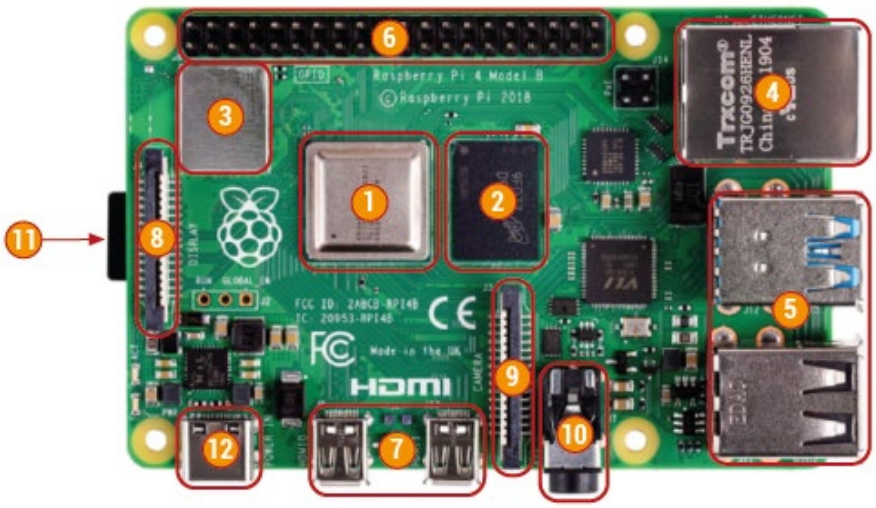


Figura 12: Raspberry Pi 4B

La RPi 4B está compuesta por varios elementos [11]:

1. **Procesador**: Broadcom BCM2711, de cuatro núcleos ARM Cortex-A72 capaces de funcionar hasta a 1,5 GHz. El BCM2711 dispone de una GPU 3D VideoCore VI que funciona a 500MHz.
2. **Memoria RAM**: De 1GB, 2GB o 4GB LPDDR4 dependiendo del modelo.
3. **Conectividad inalámbrica**: En cuanto a conectividad inalámbrica, la RPi 4B dispone de WLAN 802.11b/g/n/ac a 2,4 GHz y 5,0 GHz. Además, tiene Bluetooth 5.0.
4. **Puerto Ethernet:** Dispone de un puerto Gigabit Ethernet
5. **Puertos USB**: La RPI tiene 2 puertos de USB 2.0 y otros 2 de USB 3.0.
6. **Pines GPIO**: Se dispone de una cabecera de 40 pines GPIO (General-Purpose Input/Output) en los que se podrán conectar diversos componentes electrónicos.
7. **HDMI**: Se tienen dos puertos micro HDMI que soportan hasta una resolución de 4K a 60fps.
8. **Puerto MIPI DSI**: Es un puerto con el que su puede conectar un monitor.
9. **Puerto MIPI CSI**: Es el puerto que se usará para conectar el módulo de cámara a la RPi.

10. **A/V**: Puerto Jack hembra de 3.5mm para salida de audio y entrada de voz.
11. **Almacenamiento**: La RPI dispone de un slot para tarjetas micro SD para el sistema operativo y almacenamiento de datos.
12. **Alimentación**: Dispone de un puerto USB-C para alimentación a 5V DC. Adicionalmente, se puede alimentar mediante GPIO y mediante PoE (Power over Ethernet).

4.2.4. Decisión

En el proceso de selección se ha descartado el Arduino UNO R3, ya que se necesitan al menos dos puertos USB para poder conectar las cámaras necesarias en el sistema de seguridad. Además, la inteligencia artificial requiere una capacidad de procesamiento que no puede ser cubierta por el Arduino.

Así, la decisión queda entre la Raspberry Pi 4B o la Tinker Board S de ASUS. Viendo los componentes de cada una, se puede ver que en cuanto a rendimiento y conectividad son similares, ambas supliendo las necesidades de diseño del sistema de seguridad, pero difiriendo bastante en el precio. Aquí la Raspberry Pi 4B, es la opción más asequible de las dos al ser bastante más económica.

Por estos motivos, para la realización del presente TFG, se determinó usar como elemento procesador del sistema de seguridad una Raspberry Pi 4B.

4.3. Cámaras

Para las cámaras del sistema de seguridad hay dos alternativas posibles: módulo de cámara de Raspberry Pi o cámara web USB. Dado que se necesitan dos cámaras, se evaluará si usar dos cámaras web USB o usar un módulo de cámara con una cámara web USB.

El módulo de cámara de Raspberry Pi es una cámara ligera portable que se conecta al puerto CSI (Camera Serial Interface) de la Raspberry Pi mediante un cable tipo Ribbon. Esta conexión usa el protocolo de interfaz serial de cámara MIPI (MIPI CSI).

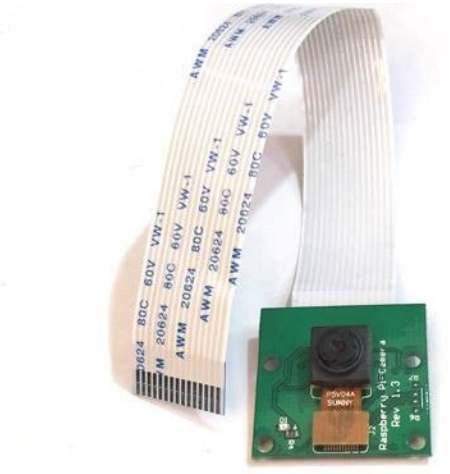


Figura 13: Módulo de cámara de Raspberry Pi

Fuente: https://components101.com/misc/pi-camera-module

Este módulo ofrece una buena calidad de imagen, además de algunas ventajas con respecto a una cámara web USB convencional [12]:

- **Menor carga en la CPU**: El protocolo USB que usan las cámaras web convencionales, tiene un sobrecoste computacional relativamente alto por su complejidad. El protocolo MIPI CSI, en cambio, tiene un coste computacional reducido. Esto, en un ordenador, no es algo relevante, dado que hoy en día los ordenadores tienen la suficiente potencia como para que este coste computacional no tenga un impacto en el rendimiento. Este hecho cambia cuando se pasa a trabajar con una Raspberry Pi, más limitada en capacidad de procesamiento, comparativamente. Para ahorrar poder computacional en este aspecto, es conveniente usar un módulo de cámara siempre que sea posible.
- **Carga de bus**: Los puertos USB y el de Ethernet comparten el mismo ancho de banda en el bus de la Raspberry Pi. Si se quiere conectar la Raspberry Pi a Ethernet porque se requiere una conexión más rápida, no es conveniente tener una cámara web consumiendo ancho de banda y ralentizando la conexión. Esto se evita con el módulo de cámara, ya que va en un puerto independiente.
- **Soporte hardware**: Si bien es cierto que el módulo de cámara no provee codificación hardware por sí solo, se conecta directamente a la GPU (Graphics Processing Unit) que sí lo hace. Si se considera la alternativa con cámara web USB, se ve que, si se quiere codificación hardware, la cámara

ha de hacerlo internamente. Por lo tanto, si se desea aceleración hardware, comparando costes, se ve que el módulo de cámara es más barato que una cámara web.

- **Factor de forma**: El módulo de cámara es más beneficioso para diseños compactos que la cámara web, por su tamaño y peso.

Se puede concluir que, siempre que se requiera y el diseño del sistema lo permita, se ha de usar el módulo de cámara. Dado que en el sistema de seguridad se necesitarán dos cámaras, se usará un módulo de cámara de Raspberry Pi, modelo NoIR v1.3, y una cámara web USB, modelo Kaulery HD 1080p USB webcam.

4.4. Altavoz y luz

El altavoz y la luz se utilizarán para la simulación de presencia. La simulación de presencia consiste en hacer que desde el exterior parezca que la vivienda está habitada cuando no se está en casa. Esto se consigue mediante la automatización y control remoto de elementos de una casa domótica, como son las persianas, las luces o la música. Así, se disuade de intentos de ocupación y robo [13].

En el caso del diseño del presente sistema de seguridad, se han usado un altavoz y una luz LED que, a demanda del usuario, se pueden encender o apagar en intervalos aleatorios de tiempo, en función a la detección de movimiento o manualmente.

4.5. Cerradura

La cerradura será otro de los actuadores controlables por el usuario para abrir o cerrar la puerta de la vivienda.

La cerradura simple de solenoide es la opción más viable en el caso del presente TFG, ya que las otras alternativas constan de software propio, que sólo complicaría el desarrollo del prototipo.

La cerradura de solenoide funciona gracias a una bobina de alambre de cobre con un lingote de metal ferromagnético en el medio. Este lingote metálico está conectado al pestillo y cuando pasa corriente por la bobina, se mueve el lingote de metal, retirando el pestillo y por lo tanto abriendo la cerradura [14].

Se utilizará una cerradura simple de solenoide para el diseño del sistema de seguridad por la razón anteriormente mencionada y por su bajo coste.

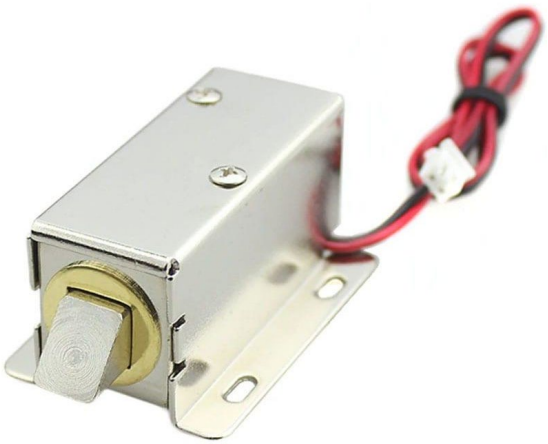


Figura 14: Cerradura de solenoide

Fuente: https://uelectronics.com/producto/cerradura-electrica-solenoide/

4.6. Comunicaciones

Para la transmisión de datos en el sistema de seguridad, las conexiones entre la Raspberry Pi y los sensores y actuadores son cableadas siguiendo las características propias de cada dispositivo.

La conexión de la Raspberry Pi con el bot de Telegram se produce a través de Internet; es inalámbrica mediante WiFi para conectar la Raspberry Pi y desde Internet a Telegram depende de la conexión del dispositivo en el que se esté ejecutando Telegram, ya que existe una versión de escritorio y una aplicación para dispositivos móviles en diferentes sistemas operativos. Para esta conexión hay varios protocolos que se podrían usar. Se verán algunos de ellos brevemente [15]:

4.6.1. MQTT (Message Queuing Telemetry Transport)

MQTT es un protocolo usado para comunicaciones de poco ancho de banda entre dispositivos en el que se sigue un esquema de publicación/suscripción. Se usa especialmente en situaciones en las que los dispositivos requieren un uso eficiente de la batería y el ancho de banda.

4.6.2. HTTP (Hypertext Transfer Protocol)

Es un protocolo con un esquema cliente/servidor, efectivo a la hora de enviar grandes cantidades de información en un ecosistema IoT. Esto lo hace muy recomendable cuando el nodo o gateway ha de enviar imágenes, vídeos o archivos [16].

4.6.3. AMQP (Advanced Message Queuing Protocol)

AMQP es un protocolo abierto que sigue un esquema de publicación/suscripción. Incluye funcionalidad para entregar los mensajes de manera fiable, representar los datos a través de diferentes formatos, flexibilidad para definir los datos, preparado para escalabilidad y capacidad de definir varias topologías en un mismo sistema [17].

4.6.4. DDS (Data Distribution Service)

Es un protocolo de tipo publicación/suscripción orientado a la comunicación M2M (machine-to-machine). Sigue una estructura descentralizada en la que los sensores, dispositivos y aplicaciones están interconectados [18].

4.6.5. Decisión

Para el presente TFG, se utilizará HTTP por la flexibilidad que ofrece en cuanto al volumen de datos que se puede transmitir con respecto al resto de los protocolos existentes. Se necesitará HTTP, por ejemplo, para poder realizar la transmisión en directo desde la cámara de la mirilla que se acciona con el timbre o para poder enviar y recibir imágenes a través del bot de Telegram.

A continuación, se muestra un esquema para ilustrar las comunicaciones del sistema de seguridad:

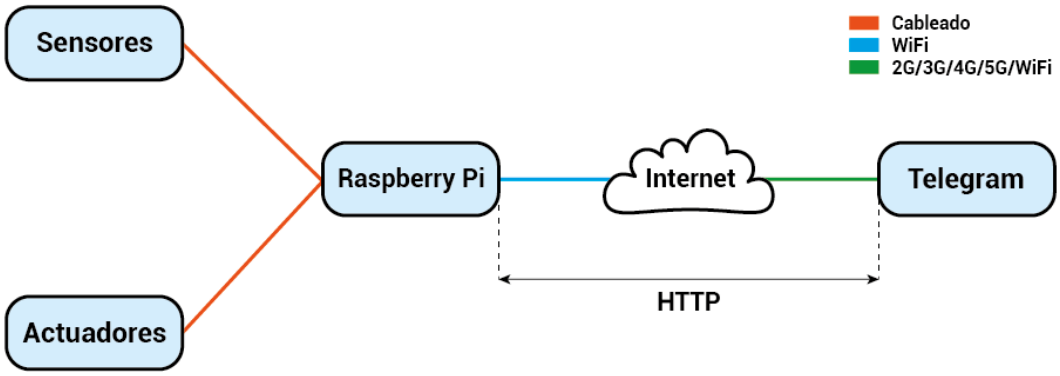


Figura 15: Esquema de las comunicaciones del sistema de seguridad

5. Materiales y métodos

5.1. Materiales

Se han clasificado los elementos en los que se basa IoT en tres categorías: dispositivos finales (sensores o actuadores), procesadores y comunicaciones. Con esta clasificación en mente, la agrupación de los elementos usados en este TFG es la siguiente:

- **Dispositivos finales**:
  - Sensores de presencia infrarrojos pasivos para captar movimiento en el exterior de la vivienda.
  - Un pulsador que actuará de timbre.
  - Módulo de cámara de Raspberry Pi NoIR v1.3, para realizar reconocimiento facial cuando se accione el pulsador. Con la cámara, se activará un flash para mejorar las condiciones de luminosidad y facilitar la labor de la IA en el reconocimiento facial. Bajo demanda del usuario, se podrá activar la transmisión en directo de lo que capta este módulo de cámara.
  - Cámara web USB para captar movimiento en el interior de la vivienda.
  - Un altavoz que actuará en tándem con la luz del interior de la casa para la simulación de presencia.
  - Una cerradura de solenoide controlable, tanto manualmente por el usuario, como automáticamente por la IA de reconocimiento facial.
- **Procesadores**:
  - Raspberry Pi Modelo 4B. La programación del software de control que se ejecuta en este dispositivo se ha realizado utilizando el lenguaje de programación Python y diversas librerías que se describirán en apartados posteriores.
- **Comunicaciones**:
  - Se han basado en el protocolo HTTP, a través del cual el sistema de seguridad, se conecta al bot de Telegram desde el que el usuario podrá controlarlo y recibir notificaciones.

A continuación, se puede ver un esquema general con los componentes electrónicos del sistema de seguridad:

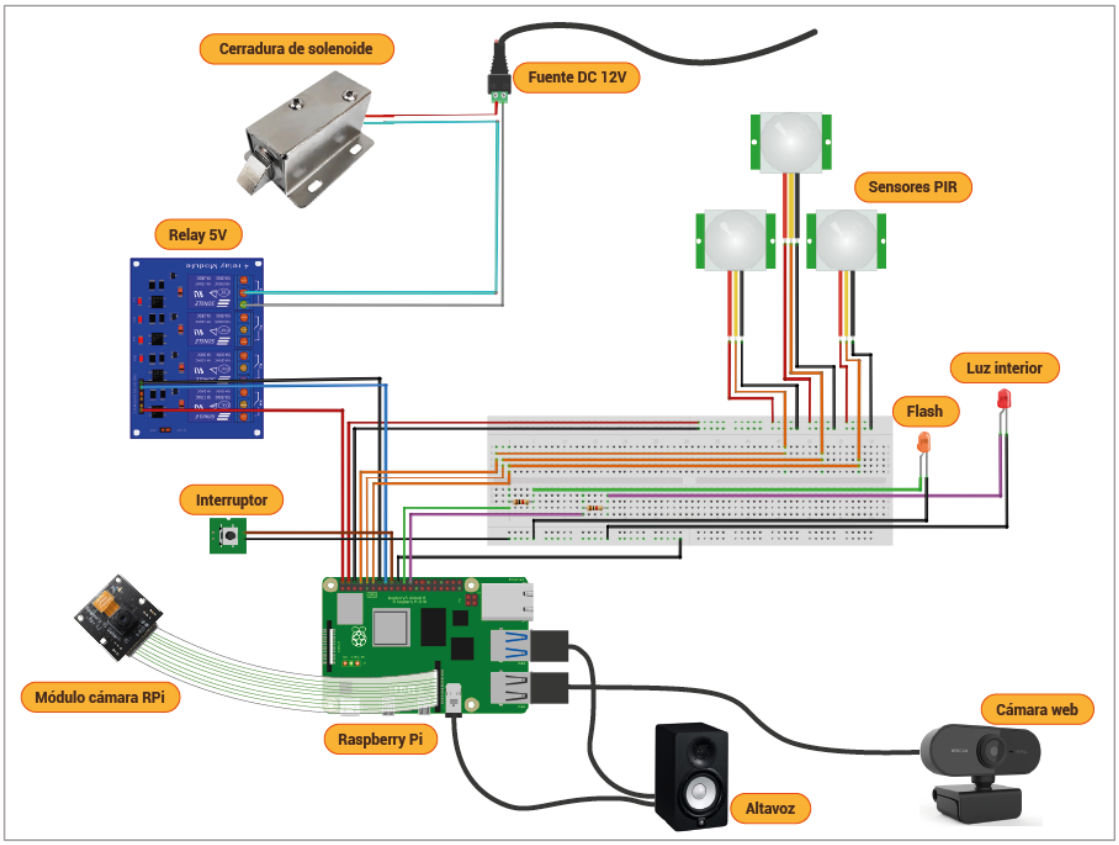


Figura 16: Esquema general de los componentes electrónicos del sistema de seguridad

5.2. Métodos

En primer lugar, se ha de preparar la Raspberry Pi como se indica en el Anexo I: Guía de instalación.

Una vez el sistema operativo y las librerías necesarias para el funcionamiento del código están instaladas, se puede iniciar la Raspberry Pi. La Raspberry Pi se ha preparado para que, al iniciarse, se ponga en funcionamiento el sistema de seguridad. Al ponerse en funcionamiento, se inicializan los componentes asociados a las diferentes funcionalidades del sistema de seguridad. Estas funcionalidades son:

- Control del sistema mediante el bot de Telegram
- Apertura y cierre de la puerta
- Detección de movimiento mediante sensores PIR
- Simulación de presencia
- Detección de movimiento mediante cámara de seguridad
- Reconocimiento facial
- Transmisión en directo de la cámara a la entrada de la vivienda
- Registro de los eventos que suceden mientras el sistema de seguridad está activo

A continuación, se explicarán los componentes del sistema de seguridad individualmente, acompañados de su implementación en el código y su funcionamiento.

5.2.1. Bot de Telegram

El bot de Telegram es una herramienta útil y accesible que Telegram proporciona. Se dice que es accesible porque para hacer uso de un bot de Telegram, el único requisito previo es descargar Telegram en un dispositivo compatible, crear una cuenta y abrir la conversación con el bot. Esto se puede hacer desde un dispositivo de escritorio usando un simple navegador web o desde un móvil en el que se use la aplicación de Telegram. En el presente TFG, se hace uso de comandos desde los que se controlarán las funciones del sistema de seguridad.

Para la creación del bot de Telegram, en primer lugar, se tendrá que usar el bot *BotFather*. La utilidad de este bot es la de crear bots nuevos cuando el usuario lo solicita. En la Figura 17, se puede ver los comandos usados para crear el bot del sistema de seguridad.

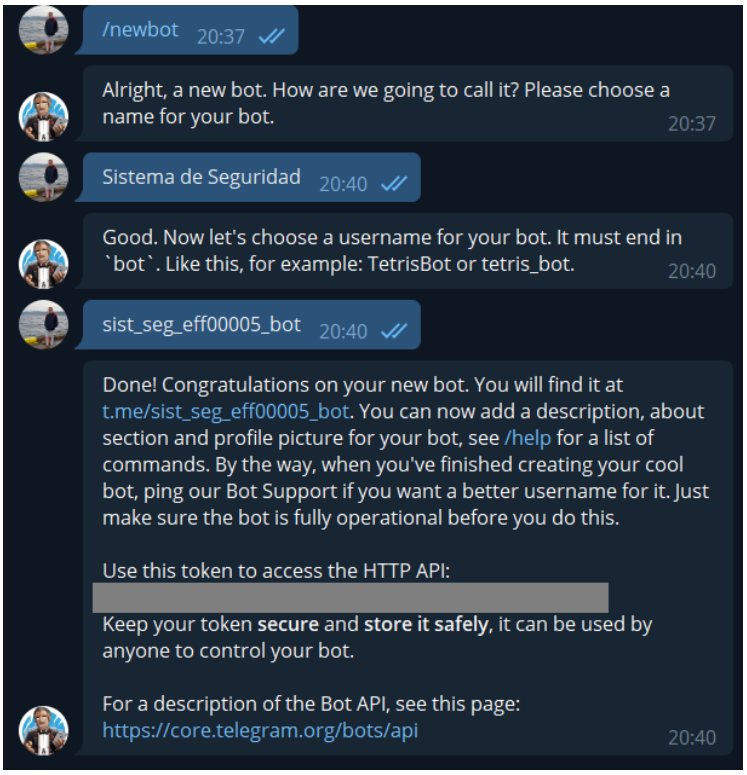


Figura 17: Conversación bot de Telegram

Una vez creado el bot que se va a usar para el sistema de seguridad, *BotFather* devuelve un token de acceso. Este token servirá para controlar el bot creado y habrá que implementarlo en el código del mismo. Es importante anotar que el token es privado, ya que, de saberlo, cualquier persona podría tener acceso a él.

Una parte fundamental del bot de Telegram son los comandos. Los comandos permiten al usuario interactuar con el bot y, en el caso de este TFG, realizar las diferentes funciones disponibles. Estos comandos se gestionan en el bot como se muestra en la Figura 18:

```
#iniciamos el bot y lo dejamos a la espera de comandos
dp.add_handler(CommandHandler('start', start))
dp.add_handler(CommandHandler('stream', stream))
dp.add_handler(CommandHandler('lock', lock))
dp.add_handler(MessageHandler(Filters.photo, img_handler))
dp.add_handler(CommandHandler('rmfolder', frai_folder_rm))
dp.add_handler(CommandHandler('rmfile', frai_file_rm))
dp.add_handler(CommandHandler('rmuser', user_rm))
dp.add_handler(CommandHandler('train', frai_train))
dp.add_handler(CommandHandler('pres', mode_com))
dp.add_handler(CommandHandler('pir', pir))
dp.add_handler(CommandHandler('detmo', detmo_com))
dp.add_handler(CommandHandler('log', log_com))
dp.add_handler(CommandHandler('reboot', reboot_com))
dp.add_handler(CallbackQueryHandler(button))
dp.add_handler(CommandHandler('secret', secret, pass_args=True))
dp.add_handler(CommandHandler('chsecret', chng_secret, pass_args=True))
dp.add_handler(CommandHandler('admin', admin, pass_args=True))
dp.add_handler(CommandHandler('chadmin', chng_admin, pass_args=True))

try:
    updater.start_polling()
    updater.idle()

except:
    pass
```

Figura 18: Recepción de comandos en el bot

Para acceder al bot del sistema de seguridad desde un dispositivo móvil con cámara se ha generado un código QR que facilita el acceso:

Figura 19: Código QR para acceder al bot de Telegram

5.2.2. MongoDB

MongoDB es la base de datos elegida para almacenar las ID de las conversaciones del bot con los usuarios y a la que el bot recurrirá para saber qué permisos tiene el usuario. Los tipos de usuario son usuario no registrado, usuario base y administrador.

Al iniciar la conversación con el bot, el usuario envía el comando */start*. Cuando el bot recibe este comando, sigue el proceso que se muestra en el siguiente flujograma:

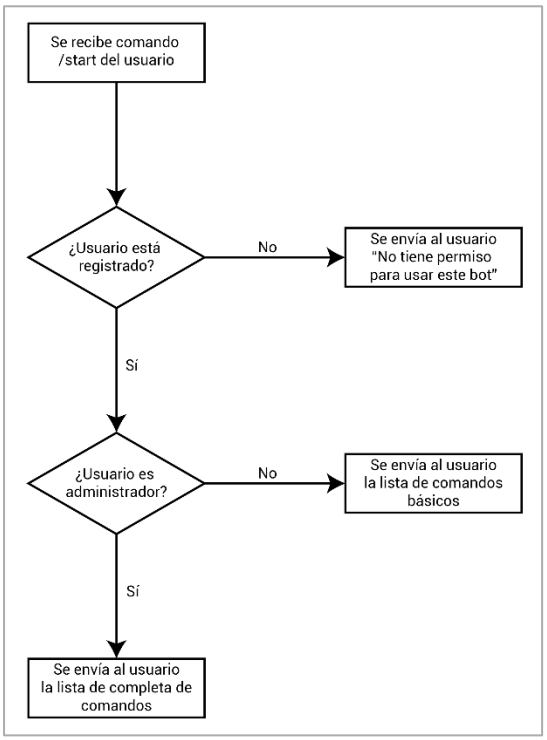


Figura 20: Flujograma del comando /start

A continuación, se explica más detalladamente lo visto en la Figura 20:

1. El usuario envía el comando */start* al bot y el bot comprueba si la ID del usuario está en la base de datos. En el caso de existir en la base de datos, devolverá la lista de comandos a la que el usuario tiene acceso. En caso contrario, devolverá el mensaje “No tiene permiso para usar este bot”.

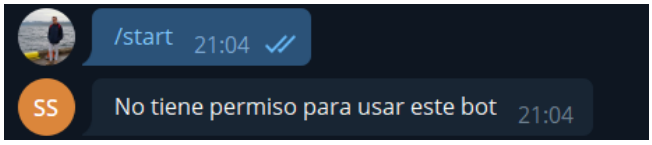


Figura 21: Resultado comando /start cuando el usuario no está en la base de datos

2. En el momento en el que el usuario le manda el comando */secret* acompañado de la contraseña correcta al bot, el bot guarda la ID de ese usuario en la base de datos. En este momento, el usuario tiene acceso a las funciones básicas del bot, además de a las alertas que el bot pueda enviar.

Figura 22: Usuario se añade a la base de datos y gana acceso a comandos básicos

3. Cuando el usuario envíe el comando */admin* acompañado de la contraseña adecuada, el usuario pasará a ser de tipo administrador. Esta información se actualizará en la base de datos y el usuario tendrá acceso a las funciones de administrador del bot.

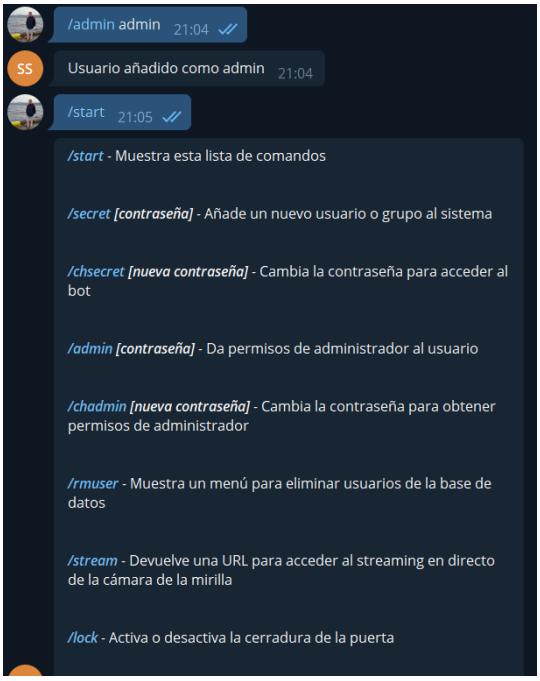


Figura 23: Usuario se añade como administrador

En la base de datos, el usuario aparecerá de la siguiente manera:

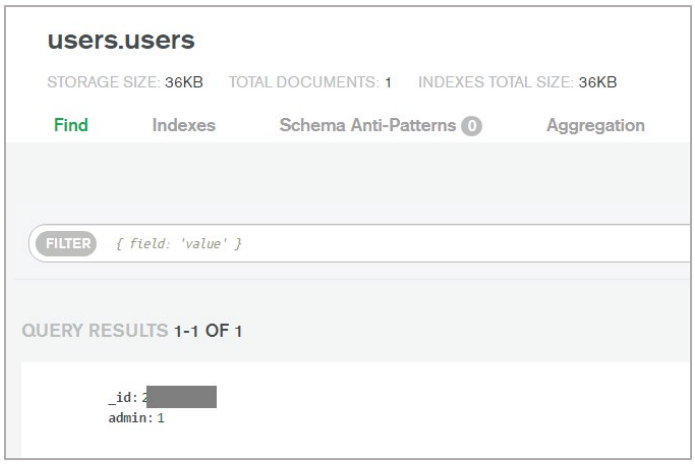


Figura 24: Usuario administrador en base de datos MongoDB

Para el acceso a la base de datos desde el sistema de seguridad, se usa la librería *Pymongo*. Esta librería proporciona las herramientas necesarias para poder trabajar con MongoDB en Python [19]. Para poder inicializar la conexión con la base de datos, se utiliza la clase *MongoClient* de dicha librería con la URL que MongoDB proporciona cuando se crea la base de datos.

```
cluster = MongoClient(
    "mongodb+srv://                @cluster0.gnxwa.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"
)
db = cluster["users"]
collection = db["users"]
```

Figura 25: Inicialización de la conexión con la base de datos

También, los usuarios administradores disponen de los comandos */rmuser*, */chsecret* y */chadmin*:

- ***/rmuser*** permite borrar usuarios de la base de datos. El usuario envía el comando y el bot le devolverá un menú desde el que podrá seleccionar el usuario que quiere eliminar del sistema.
- ***/chsecret*** permite cambiar la contraseña de acceso al bot.
- ***/chadmin*** permite cambiar la contraseña para otorgar permisos de administrador a los usuarios.

5.2.3. Cerradura de solenoide

La cerradura de solenoide es uno de los actuadores del sistema de seguridad. Esta será accionada con una señal en el GPIO 23, que deberá pasar por un módulo de relé. La cerradura de solenoide tiene un rango de operación de 9V a 12V y, por lo tanto, en el prototipo se conecta a una fuente de voltaje de 12V, como se puede ver en el esquema de la Figura 16 [20]. La función del módulo de relé es la de completar el circuito entre la fuente de voltaje de 12V y la cerradura de solenoide cuando el GPIO 23 de la RPi esté a voltaje bajo, es decir a 0.

Cuando el bot recibe el comando */lock*, se sigue la lógica de funcionamiento que se muestra a continuación:

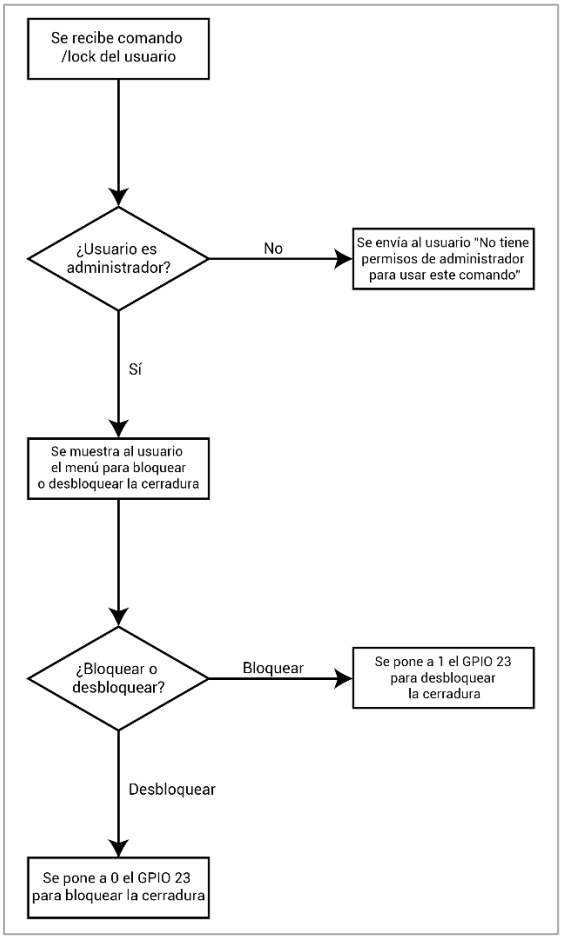


Figura 26: Flujograma del comando /lock

En el código, se usa la clase *GPIO* de la librería *RPi* para controlar los estados y los modos de los GPIO de la RPi. Como se puede ver en la Figura 27, el GPIO 23 se configura como salida y se inicializa a 1 (voltaje alto), para que la cerradura esté inactiva.

```
relay = 23;
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(relay, GPIO.OUT)
GPIO.output(relay, 1)
```

Figura 27: Configuración del GPIO 23

La función para poder cambiar el estado del GPIO recibirá la información sobre el estado deseado desde el bot de Telegram y es como se muestra en la Figura 28:

```
def lock(state):
    state = int(state)
    if state == 1:
        GPIO.output(relay, 1)

    elif state == 0:
        GPIO.output(relay, 0)
```

Figura 28: Cambio de estado de la cerradura de solenoide

El bot de Telegram se encargará de proporcionarle un menú al usuario cuando envíe el comando */lock*, en el que podrá elegir el estado de la cerradura:

```
def lock(update, context):
    userid = str(update.message.chat_id)
    results = collection.find_one({"_id": int(userid)})
    if results is None:
        dp.bot.sendMessage(
            chat_id=update.message.chat_id, text="No tiene permiso para usar este bot"
        )
    else:
        if results["admin"] == 1:
            update.message.reply_text(
                "¿Bloquear o desbloquear cerradura?", reply_markup=lock_menu()
            )
        elif results["admin"] == 0:
            dp.bot.sendMessage(
                chat_id=update.message.chat_id,
                text="No tiene permisos de administrador para usar este comando",
            )


def lock_menu():
    keyboard = [
        [InlineKeyboardButton("Bloquear", callback_data="lock:1")],
        [InlineKeyboardButton("Desbloquear", callback_data="lock:0")],
    ]
    return InlineKeyboardMarkup(keyboard)
```

Figura 29: Menú de la cerradura en el bot de Telegram

Como se ha mencionado en la subapartado *5.2.2. MongoDB*, hay funciones básicas que corresponden a usuarios base y hay funciones que corresponden a usuarios administradores. La función para controlar la cerradura es una de las que componen el último grupo.

Como se puede ver en la Figura 29, en la línea 233, primero se comprueba que el usuario está en la base de datos como usuario del bot de Telegram y, en la línea 239, si el usuario es administrador se devuelve el menú para controlar la cerradura.

En el menú, cada botón tiene asignado un *callback_data* para saber qué acción desea realizar el usuario. Este *callback_data* se gestionará de la siguiente manera

```
965         elif query.data.startswith("lock:"):
966             if sel == "1":
967                 query.edit_message_text(text="Ha bloqueado la cerradura")
968             elif sel == "0":
969                 query.edit_message_text(text="Ha desbloqueado la cerradura")
970             main.lock(sel)
```

Figura 30: Gestión respuesta del usuario (menú cerradura)

Todas las respuestas del usuario son gestionadas mediante una función que recogerá los *callback_data* de los menús que proporcione el bot.

En el caso de la cerradura, el *callback_data* empezará con “lock:” (línea 965) y dependiendo de *sel* (la segunda parte del *callback_data*, después de “:”), se enviará un mensaje de confirmación al usuario. Acto seguido, se llamará a la función de la Figura 28 para cambiar el estado del GPIO 23 y bloquear o desbloquear la cerradura.

5.2.4. Detección de movimiento (sensores PIR)

La detección de movimiento por sensores PIR se activa mediante el comando */pir* y su funcionamiento se puede ver en la siguiente figura:

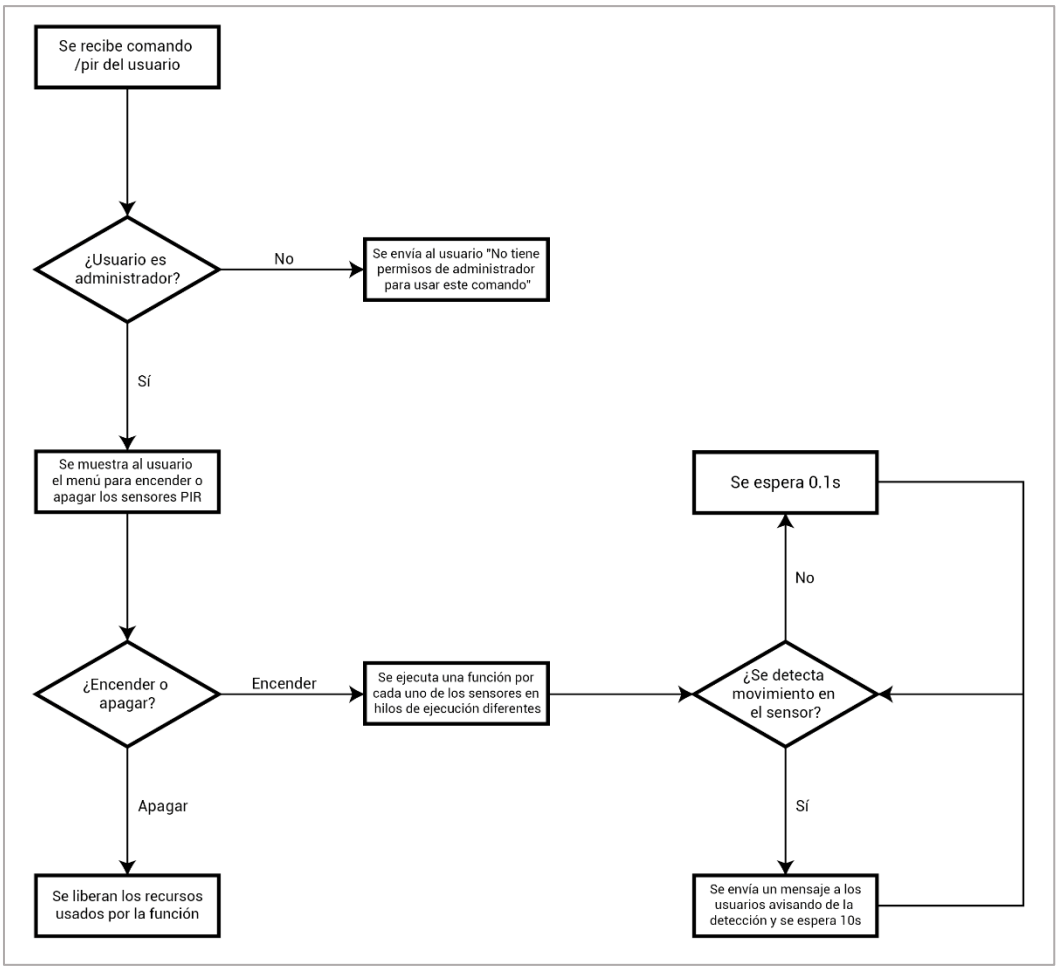


Figura 31: Flujograma del comando /pir

Para poder recibir datos de los sensores PIR, los pines GPIO correspondientes a cada sensor tienen que ser configurados como entradas:

```
28  GPIO.setmode(GPIO.BCM)
29  sensor = {
30      "S1": {"name": "Sensor 1", "pin": 14},
31      "S2": {"name": "Sensor 2", "pin": 15},
32      "S3": {"name": "Sensor 3", "pin": 18},
33  }
34  GPIO.setup(sensor["S1"]["pin"], GPIO.IN)
35  GPIO.setup(sensor["S2"]["pin"], GPIO.IN)
36  GPIO.setup(sensor["S3"]["pin"], GPIO.IN)
```

Figura 32: Configuración de los GPIO de los sensores PIR

De manera similar a la función de la cerradura (Figura 29), cuando el usuario envíe el comando */pir*, el bot le proporcionará un menú desde el que podrá decir si desea encender o apagar la detección de movimiento mediante los sensores PIR. El bot sólo les proporcionará el menú a los usuarios administradores.

Una de las diferencias notables con respecto a la función de la cerradura es la manera de la que se gestiona la respuesta del usuario:

```
1070        elif query.data.startswith("pir:"):
1071            if pir_pres_state is False:
1072                if sel == "1":
1073                    if pir_state is False:
1074                        pir_cont(sel)
1075                        pir_state = True
1076                        query.edit_message_text(
1077                            text="Ha encendido los sensores de movimiento"
1078                        )
1079                    elif pir_state is True:
1080                        query.edit_message_text(
1081                            text="Los sensores de movimiento ya están encendidos"
1082                        )
1083                elif sel == "0":
1084                    if pir_state is True:
1085                        pir_cont(sel)
1086                        pir_state = False
1087                        query.edit_message_text(
1088                            text="Ha apagado los sensores de movimiento"
1089                        )
1090                    elif pir_state is False:
1091                        query.edit_message_text(
1092                            text="Los sensores de movimiento ya están apagados"
1093                        )
1094            if pir_pres_state is True:
1095                query.edit_message_text(
1096                    text="No puede usar esta función mientras la simulación de presencia con PIR está activa"
1097                )
1098
```

Figura 33: Gestión respuesta del usuario (menú sensores PIR)

Como se puede ver en la Figura 33, se sigue un sistema de *flags* en el que se comprueban dos variables:

- ***pir_pres_state***: Informa sobre el estado de ejecución de la función de simulación de presencia mediante sensores PIR (explicada más adelante).
- ***pir_state***: Informa sobre el estado de ejecución de la propia función de detección de movimiento mediante sensores PIR

Ambas variables se comprueban, ya que de ejecutarse una instancia de la función mientras se está ejecutando la simulación de presencia o la propia función de detección de movimiento, se pueden provocar errores o duplicación de mensajes de aviso (ambas funciones avisan al usuario si se detecta movimiento).

La función que se llama desde el gestor de respuestas de la Figura 33 es un controlador:

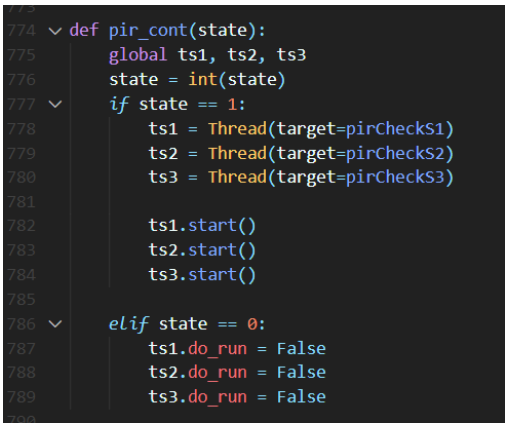


Figura 34: Controlador de la función de detección de presencia mediante sensores PIR

Desde este controlador se ejecutan tres funciones (una para cada sensor PIR) simultáneamente en hilos de ejecución diferentes (esto se hace mediante la clase *Thread* de la librería *threading* [21]). Cada una de estas funciones comprobará si la alarma de alguno de los sensores PIR se activa. A continuación, se muestra una de estas funciones:

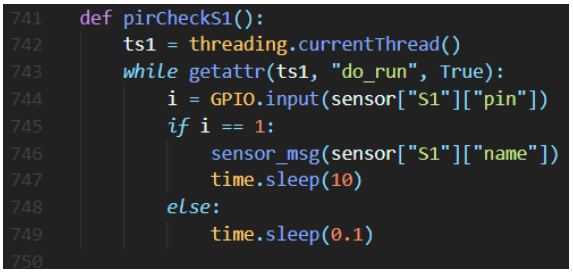


Figura 35: Función de detección de movimiento del sensor PIR S1

En la Figura 35 se puede ver el motivo por el cual se debe ejecutar las funciones en hilos de ejecución separados: un bucle. Este bucle, de no ejecutarse en un hilo de ejecución, provocaría el bloqueo del bot y de las demás funciones.

Para poder parar la función desde fuera del bucle, se usa el atributo *do_run*, al que el controlador puede asignarle el valor de False para parar el bucle cuando el usuario lo desee (línea 786 en la Figura 34).

En la línea 745 de la Figura 35, se puede ver que, si el sensor PIR se activa, se llama a la función *sensor_msg*, que les enviará a los usuarios registrados en la base de datos un mensaje notificando en qué sensor se ha detectado movimiento. Después de hacer esto, se esperará 10 segundos, para evitar superar el límite de *flooding* impuesto por Telegram para los bots (un máximo de 30 mensajes/minuto en chats con usuarios y de 20 mensajes/minuto en grupos [22]).

5.2.5. Simulación de presencia

La simulación de presencia consiste en la disuasión de potenciales intentos de robo u ocupación mediante distintos actuadores, que hacen aparentar que la vivienda está habitada. Estos actuadores pueden ser las luces de las habitaciones, las persianas o altavoces en el interior de la casa, por poner algunos ejemplos. En el prototipo del sistema de seguridad, la simulación de presencia se hará con un altavoz y un diodo LED (simulando la luz de una habitación).

La función de simulación de presencia funciona de manera similar a las funciones vistas anteriormente, el usuario envía el comando */pres* y el bot le proporciona un menú al usuario si es administrador. Desde el menú de esta función, el usuario puede escoger el modo en el que quiere activar la simulación de presencia: aleatoria, con alarma o manual. Estos modos se verán a continuación.

5.2.5.1. Simulación de presencia aleatoria

El objetivo de la simulación de presencia aleatoria es el de aparentar que la vivienda está habitada cuando no lo esté. Esto se logra mediante el encendido o apagado de los actuadores siguiendo un patrón pseudoaleatorio. Para ilustrar esta opción de simulación de presencia, se muestra el siguiente flujograma:

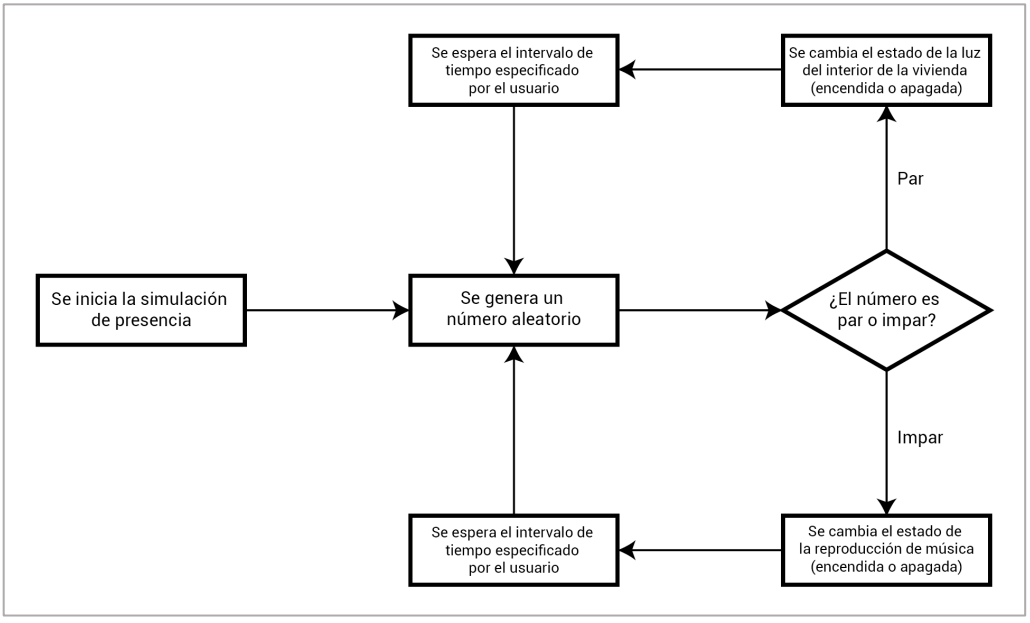


Figura 36: Flujograma de la simulación de presencia aleatoria

El usuario puede elegir la frecuencia con la que quiere que los eventos sucedan. El bot recibirá esta información e iniciará la función de simulación de presencia en un subproceso (esto se hace con la clase *Process* de la librería *multiprocessing* [23]). La función de simulación de presencia funciona de la siguiente manera:

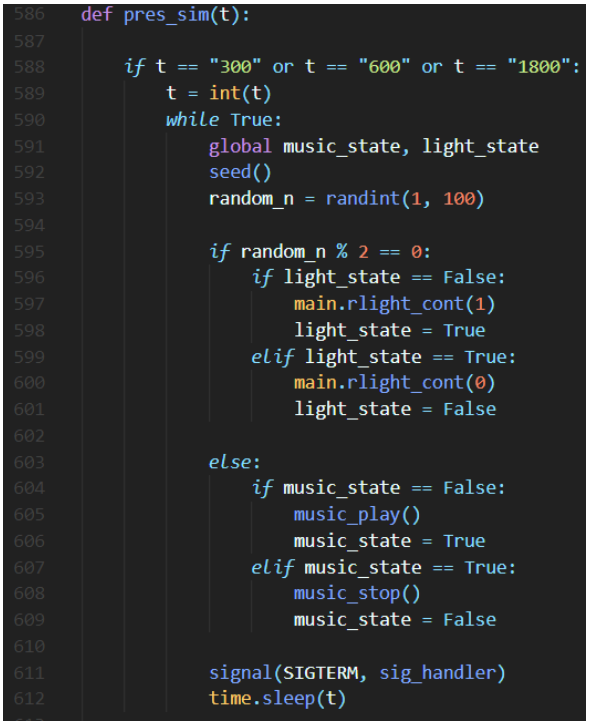


Figura 37: Función de simulación de presencia aleatoria

En la Figura 37 se puede apreciar cómo, en la línea 588, se comprueba si *t* es 300 (5 minutos), 600 (10 minutos) o 1800 (30 minutos). Cabe decir que se hace esta comprobación previa porque la función *pres_sim* es la función que se llamará para todos los modos de simulación de presencia.

En la línea 591 de la Figura 37, se llama a dos variables globales que van a hacer de *flags*, *music_state* y *light_state*. Estas se comprobarán después para poder alternar correctamente entre el encendido y el apagado de la luz y la música. Para decidir si se enciende o apaga la luz o la música, se comprueba la paridad de un número pseudoaleatorio generado en la línea 593.

En la línea 611, se define un controlador de señal. Este controlador de señal servirá para interceptar la señal que envía la función *terminate* de la librería *multiprocessing*. Al interceptar esta señal, el controlador de señal ejecutará la función *sig_handler* para finalizar el subproceso:

```
633  def sig_handler(*args):
634      global music_state
635      main.rlight_cont(0)
636      if music_state is True:
637          music_stop()
638      sys.exit(0)
```

Figura 38: Función sig_handler

5.2.5.2. Simulación de presencia con alarma

El objetivo de este modo es el de alertar a los usuarios que estén en la vivienda cuando se detecte movimiento en los sensores PIR. Para ello, se activará una alarma y se encenderá la luz de la casa, ahuyentando así a los asaltantes. Para comprender mejor el funcionamiento de esta función, se muestra el siguiente flujograma:

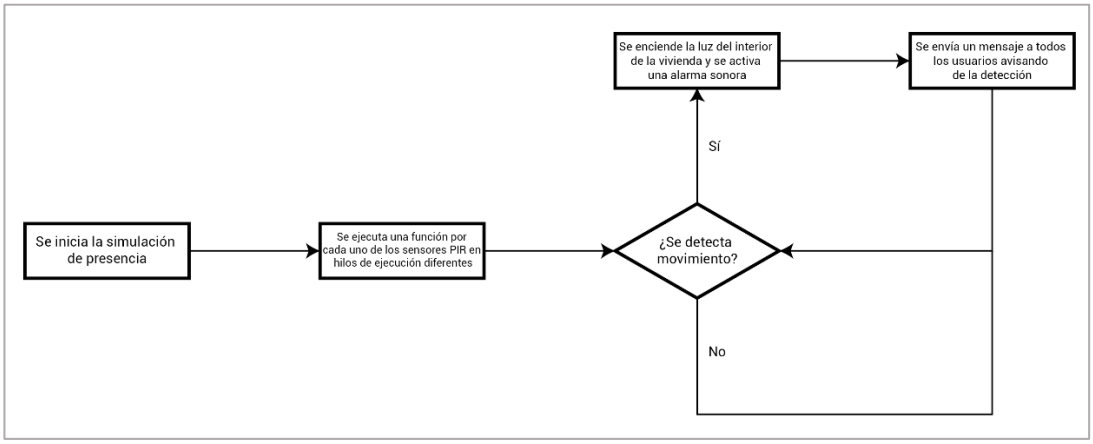


Figura 39: Flujograma de la simulación de presencia con alarma

Para conseguir lo anterior, se seguirá un procedimiento similar al de la función de simulación de presencia aleatoria. Cuando el usuario elige el modo de simulación de presencia con alarma, desde el gestor de respuestas se ejecutará la función *pres_sim* (Figura 37) en un subproceso con el argumento *t = "pir"*. Desde *pres_sim* se llamará a la función *pir_pres_cont*:

```
614      elif t == "pir":
615          pir_pres_cont(1)
616          while True:
617              time.sleep(1)
618              signal(SIGTERM, pir_pres_sh)
```

Figura 40: Función de simulación de presencia con alarma

Como se puede ver en la Figura 40, también se define un controlador de señal para poder finalizar el subproceso.

La función *pir_pres_cont* funciona de manera similar a la función *pir_cont* (Figura 34). Desde esta función se llama a tres funciones que comprobarán si la alarma de alguno de los sensores PIR se activa, pero en vez de solamente enviar un mensaje al usuario, se activarán las luces y una alarma:

```
663  def pirPresS1():
664      global pres_busy
665      tps1 = threading.currentThread()
666      while getattr(tps1, "do_run", True):
667          i = GPIO.input(sensor["S1"]["pin"])
668          if i == 1:
669              if pres_busy is False:
670                  pres_busy = True
671                  main.rlight_cont(1)
672                  pres_play()
673                  time.sleep(5)
674                  main.rlight_cont(0)
675                  pres_stop()
676                  pres_busy = False
677          else:
678              time.sleep(0.1)
679
```

Figura 41: Función del sensor PIR 1 para la simulación de presencia

Desde *main.rlight_cont* se encenderá o apagará la luz, y desde *pres_play* y *pres_stop* se encenderá y se parará un sonido de alarma, respectivamente.

5.2.5.3. Simulación de presencia manual

Mediante esta función el usuario puede encender o apagar la luz y la música manualmente.

Esta función también parte de la función *pres_sim* (Figura 37), que se ejecutará en un subproceso con el argumento *t = "manual"*. A continuación, se muestra lo que hará la función en este caso:

```
620      elif t == "manual":
621          main.rlight_cont(1)
622          music_play()
623          signal(SIGTERM, sig_handler)
624          while True:
625              time.sleep(0.1)
```

Figura 42: Función de simulación de presencia manual

En la Figura 42, se puede ver como se enciende la luz (línea 621) y se enciende la música (línea 622). Luego, se define un controlador de señal que llamará a la función *sig_handler* (Figura 38) para finalizar el subproceso.

5.2.6. Detección de movimiento (cámara de seguridad)

Cuando un usuario administrador le envíe al bot el comando */detmo*, el bot le proporcionará un menú de confirmación para activar la detección de movimiento mediante una cámara de seguridad situada en el interior de la vivienda. Para el prototipo del sistema de seguridad, se usará una cámara web.

Se puede ver a continuación un flujograma que ilustra el proceso de detección de movimiento:

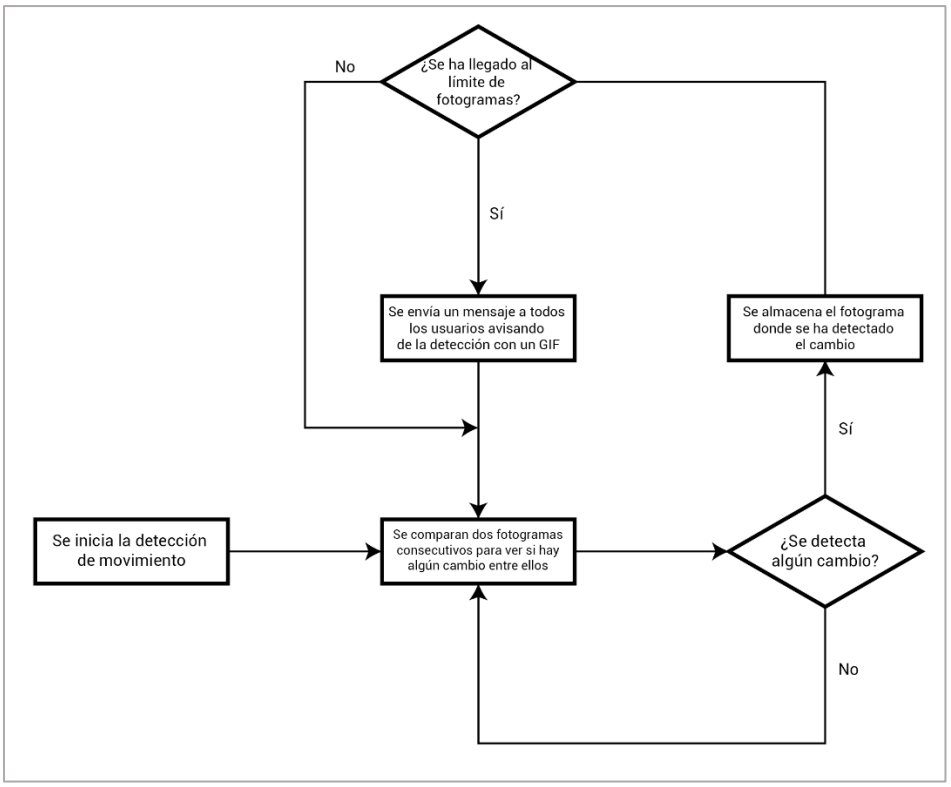


Figura 43: Flujograma del comando /detmo

Una vez el usuario confirme que quiere activar la detección de movimiento, se llamará a la función *detmo_cont*:

```
148   td = None
149
150   def detmo_cont(state):
151       global td
152       state = int(state)
153
154       if state == 1:
155           td = Thread(target=detmo)
156           td.start()
157
158       elif state == 0:
159           td.do_run = False
160
```

Figura 44: Función detmo_cont

Esta función servirá para controlar la ejecución o finalización de la función *detmo*, que se ejecutará mediante un hilo de ejecución:

```
195 def detmo(): #detección de movimiento
196     #Inicializamos las variables a usar
197     td = threading.currentThread()
198     feed = cv2.VideoCapture(0)
199     camName = "WebCam"
200     clean_png(camName)
201     time.sleep(2.0)
202     img = None #image a enviar a intruChecker en caso de intrusión
203     intru = 0 #intrusión
204     init = time.time() #tiempo de inicio de la detección de movimiento
205
206     #inicializamos dos variables que contendrán dos fotogramas, el actual y el anterior
207     #se usarán para detectar diferencias entre ellos
208     ret, frame1 = feed.read()
209     ret, frame2 = feed.read()
```

Figura 45: Inicialización de variables en la función detmo

En la Figura 45, se puede apreciar cómo se inicializan las diferentes variables a utilizar en la función. Algunas variables a destacar son:

- ***td***: Servirá para controlar la ejecución de la función.
- ***feed***: Corresponde a la imagen que capta la cámara web.
- ***frame1***: Corresponde al fotograma anterior al actual.
- ***frame2***: Corresponde al fotograma actual.

A continuación de la inicialización de variables, se inicia un bucle en el que se realizará un tratamiento de imagen a *frame1* y *frame2*. Este tratamiento de imagen y el enfoque para la detección de movimiento están basados en un proyecto de detección de movimiento basado en la librería OpenCV (Open Computer Vision) [24]:

```
212     while getattr(td, "do_run", True):
213         intru = 0
214         diff = cv2.absdiff(frame1, frame2)
215         gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
216         cgray = gammaCorrection(gray, 2)
217         blur = cv2.medianBlur(cgray, 5)
```

Figura 46: Tratamiento de imagen de la función detmo

En primer lugar, se inicia un bucle con la variable *td* para poderlo interrumpir en cualquier momento. Luego, se obtiene la imagen diferencial entre *frame1* y *frame2* (línea 214). Después, se pasa las imagen resultante a escala de grises (línea 215) ya que, en la detección de movimiento, el color de la imagen no influye. Una vez hecho esto, para ajustar la luminancia de las imágenes, se aplica una corrección gamma y un difuminado a la imagen. El difuminado se hace para para reducir el ruido en la imagen, disminuyendo así la posibilidad de falsos positivos, y la corrección gamma, para mejorar el funcionamiento de la detección de movimiento en condiciones de poca luminosidad [25]:

```
190 def gammaCorrection(src, gamma): #aumentamos la luminancia de
191     invGamma = 1/ gamma
192     table = [((i/255) ** invGamma) * 255 for i in range(256)]
193     table = np.array(table, np.uint8)
194     return cv2.LUT(src, table)
```

Figura 47: Función de corrección gamma

Como se puede ver en la línea 191 de la Figura 47, el factor gamma con el que se corrige la imagen es fraccional para aumentar la luminancia de la imagen. De lo contrario, si se usase un factor gamma entero, la imagen se oscurecería.

Cuando la imagen está preparada, se realiza la detección de movimiento como tal:

```
        _, thresh = cv2.threshold(blur, 90, 255, cv2.THRESH_BINARY)
        dilated = cv2.dilate(thresh, None, iterations=3)
        contours = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        contours = imutils.grab_contours(contours)


        #cv2.drawContours(frame1, contours, -1, (0, 255, 0), 1) #dibujamos los contornos en el f

        for i in contours: #se comprueba si hay un contorno en el fotograma y, si lo hay, se por
            if cv2.contourArea(i) > 0:
                intru = 1
                img = frame1
        #cv2.imshow('Feed', frame1) #mostramos fotogramas por pantalla
        #asignamos el fotograma actual al anterior y seguimos capturando

        frame1 = frame2
        ret, frame2 = feed.read()
        intruChecker(intru, init, camName, img) #Le pasamos las variables a intruChecker
```

Figura 48: Detección de movimiento de la función detmo

El funcionamiento de la detección de movimiento mostrada en la Figura 48 funciona de la siguiente manera:

1. A partir de la imagen difuminada, se crea una imagen binaria, que es una imagen con píxeles blancos o negros, siendo 1 los píxeles blancos y 0 los píxeles negros.
2. Después, se dilatan los píxeles blancos de la imagen binaria obtenida para compensar por el difuminado que se ha aplicado anteriormente en el tratamiento de imagen.
3. Una vez se ha dilatado la imagen, solo queda encontrar los contornos de los píxeles blancos.
4. En el caso de que existan contornos, se asignará el fotograma en el que se han encontrado los contornos a la variable *img*, que se pasará como argumento en la llamada a la función *intruChecker*.
5. Antes de llamar a la función *intruChecker*, se asigna el fotograma de *frame2* a *frame1* y, a su vez, *frame2* se asigna al nuevo fotograma que se capte en la cámara web.

La función *intruChecker*, que se llama al final del bucle de la función *detmo*, se encarga de almacenar los fotogramas en los que se detecte movimiento. Cuando el número de fotogramas almacenados alcance el límite especificado (en el caso del prototipo, 10 fotogramas), se generará un GIF con los fotogramas y se enviará a los usuarios base y administradores del bot.

5.2.7. Inteligencia Artificial (IA)

Para el prototipo del sistema de seguridad se ha usado una IA que efectúa reconocimiento facial cuando se acciona un interruptor en la entrada de la vivienda.

Para el proceso de entrenamiento y reconocimiento de esta IA se ha adaptado de un proyecto de autoría de Adrian Rosebrock [26], al que se le ha añadido la posibilidad de detección múltiple en una imagen y el código necesario para poder ser gestionada mediante el bot de Telegram. Desde el bot de Telegram se puede administrar el conjunto de datos (dataset) con el que se entrena la IA, iniciar el proceso de entrenamiento y recibir resultados de los reconocimientos faciales que se hagan.

Para la gestión del dataset, el usuario dispone de los comandos */rmfolder* y */rmfile*. Además, puede añadir imágenes nuevas al dataset simplemente enviándole imágenes al bot.

5.2.7.1. Función para el almacenamiento de imágenes

Las imágenes enviadas por los usuarios son almacenadas y clasificadas en el sistema de seguridad con los nombres y apellidos de los usuarios que las envían (se usa el nombre y el apellido que se haya configurado al crear la cuenta en Telegram). En la Figura 49 se puede ver parte de la función que se encarga de almacenar las imágenes:

```
name = update.message.from_user.first_name
last_name = update.message.from_user.last_name
if name is None:
    name = ""
if last_name is None:
    last_name = ""
path = "/home/pi/Desktop/code/project/facial_recog/dataset/{} {}".format(
    name, last_name
)
if not os.path.exists(path):
    os.makedirs(path)
file = dp.bot.getFile(update.message.photo[-1].file_id)
print("file_id: " + str(update.message.photo[-1].file_id))
n = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
file.download(
    "/home/pi/Desktop/code/project/facial_recog/dataset/{} {}/{}.jpg".format(
        name, last_name, n
    )
)
```

Figura 49: Función para la gestión de imágenes recibidas

Las imágenes entonces son guardadas en carpetas con el nombre y el apellido del usuario que las envía.

5.2.7.2. Función /rmfolder

Para poder eliminar las diferentes carpetas donde se almacenan las imágenes, se usa el comando */rmfolder*. El bot, cuando recibe este comando, devuelve un menú en el que se muestra un listado de las carpetas existentes en el sistema. Desde este menú, el usuario puede seleccionar la carpeta a eliminar. Para poder mostrar el listado se usa la siguiente función:

```
354  def folder_menu():
355      ktext = "["
356      path = "/home/pi/Desktop/code/project/facial_recog/dataset/"
357      files = os.listdir(path)
358      for f in files:
359          if f != "Unknown":
360              ktext += "[InlineKeyboardButton('{}', callback_data='folder:{}')], ".format(
361                  f, f
362              )
363      ktext += "[InlineKeyboardButton('Salir', callback_data='salir')]]"
364      keyboard = eval(ktext)
365      return InlineKeyboardMarkup(keyboard)
```

Figura 50: Función para el listado de /rmfolder

Se puede observar cómo se recogen todas las carpetas dentro del dataset de entrenamiento menos la carpeta *Unknown* (línea 359). Esto es porque la carpeta *Unknown* contiene 100 imágenes de caras aleatorias obtenidas del dataset "Flickr-Faces-HQ Dataset (FFHQ)" [27]. Así, esta carpeta servirá para entrenar la IA para clasificar como *Unknown* las caras que no identifique entre las caras enviadas por los usuarios.

5.2.7.3. Función /rmfile

Además de */rmfolder*, el usuario dispone también del comando */rmfile* para eliminar imágenes del dataset individualmente. Cuando el usuario le envíe al bot este comando, devolverá un listado con las carpetas del dataset de entrenamiento. Ahí el usuario escogerá la carpeta que desee editar. El bot entonces devolverá un listado con las imágenes dentro de la carpeta y el usuario podrá escoger cuál de ellas eliminar. Cuando el usuario escoja la imagen a eliminar, el bot le mostrará una previsualización de la imagen que va a eliminar, acompañada de un menú de confirmación.

5.2.7.4. Función /train

Con las funciones explicadas en los anteriores subapartados, el usuario ya puede gestionar cómodamente el dataset de entrenamiento de la IA. Sólo queda poder poner en marcha el propio entrenamiento. Esto se hace con el comando */train*, que recogerá todas las carpetas del dataset e iniciará el entrenamiento de la IA con ellas. A lo largo del proceso de entrenamiento, se le irá notificando en tiempo real el progreso al usuario mediante el bot.

La función que lleva a cabo el entrenamiento se explicará por partes a continuación:

1. En primer lugar, se inicializa el detector facial. Este detector está basado en Caffe, que es un marco de aprendizaje profundo, y se utiliza para detectar caras dentro de las imágenes:

```
def train(update, context):

    text = "[INFO] cargando detector facial..."
    print(text)
    bot.frai_train_prog(update, context, text)
    protoPath = os.path.sep.join(
        [
            "/home/pi/Desktop/code/project/facial_recog/face_detection_model",
            "deploy.prototxt",
        ]
    )
    modelPath = os.path.sep.join(
        [
            "/home/pi/Desktop/code/project/facial_recog/face_detection_model",
            "res10_300x300_ssd_iter_140000.caffemodel",
        ]
    )
    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
```

Figura 51: Inicialización del detector facial

2. Luego, se inicializa el modelo Torch de incrustación visual que extraerá vectores de incrustación 128-D de las caras detectadas en las imágenes:

```
text = "[INFO] cargando reconocimiento facial..."
print(text)
bot.frai_train_prog(update, context, text)
embedder = cv2.dnn.readNetFromTorch(
    "/home/pi/Desktop/code/project/facial_recog/openface.nn4.small2.v1.t7"
)
```

Figura 52: Inicialización del modelo Torch de incrustación visual

3. Después, se carga el dataset de entrenamiento para iniciar la cuantificación de las caras:

```
text = "[INFO] cuantificando caras..."
print(text)
bot.frai_train_prog(update, context, text)
imagePaths = list(
    paths.list_images("/home/pi/Desktop/code/project/facial_recog/dataset")
)
```

Figura 53: Carga del dataset de entrenamiento

4. Ahora, se inicia la cuantificación de las caras en las imágenes del dataset. Para ello se recorrerán todas las imágenes con un bucle y se irán guardando los nombres de las carpetas en la variable *name* para identificar la cara con ese nombre. Luego se hace un *blob* (Binary Large OBject) de la imagen, es decir, una imagen en la que se distingue un grupo de píxeles con valores similares y los que lo rodean. Así se aísla la cara del fondo.

```
for (i, imagePath) in enumerate(imagePaths):
    text = "[INFO] procesando imagen {}/{}".format(i + 1, len(imagePaths))
    print(text)
    bot.frai_train_prog(update, context, text)
    name = imagePath.split(os.path.sep)[-2]

    image = cv2.imread(imagePath)
    image = imutils.resize(image, width=600)
    (h, w) = image.shape[:2]

    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(image, (300, 300)),
        1.0,
        (300, 300),
        (104.0, 177.0, 123.0),
        swapRB=False,
        crop=False,
    )
```

Figura 54: Cuantificación de las caras (blob)

5. Una vez se ha aislado la cara del fondo, se carga el blob como input para el detector facial. Si se detecta al menos una cara (se asume que en cada imagen hay una sola cara), se aísla la cara con el mayor factor de confianza, descartando las detecciones faciales por debajo del factor de confianza, 50% en el caso del prototipo. Si se supera este factor de confianza, se extrae la región de interés de la cara.

```
detector.setInput(imageBlob)
detections = detector.forward()

if len(detections) > 0:
    i = np.argmax(detections[0, 0, :, 2])
    confidence = detections[0, 0, i, 2]

    if confidence > 0.5:
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        face = image[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]

        if fW < 20 or fH < 20:
            continue
```

Figura 55: Detección facial y extracción de región de interés

6. Teniendo esto, se hace un blob de la región de interés de la cara y se pasa por el modelo de incrustación. Este genera un vector 128-D que describe la cara. Se añade también el nombre de identificación a la lista de nombres conocidos.

```
faceBlob = cv2.dnn.blobFromImage(
    face, 1.0 / 255, (96, 96), (0, 0, 0), swapRB=True, crop=False
)
embedder.setInput(faceBlob)
vec = embedder.forward()

knownNames.append(name)
knownEmbeddings.append(vec.flatten())
total += 1
```

Figura 56: Extracción del vector 128-D de la cara

7. Una vez el bucle termina de recorrer todas las imágenes, se serializan los datos en un fichero *pickle*, que convierte la jerarquía del objeto *data* en una cadena de bytes almacenarlo fácilmente en memoria.

```
text = "[INFO] serializando {} codificaciones...".format(total)
print(text)
bot.frai_train_prog(update, context, text)
data = {"embeddings": knownEmbeddings, "names": knownNames}
f = open(
    "/home/pi/Desktop/code/project/facial_recog/output/embeddings.pickle", "wb"
)
f.write(pickle.dumps(data))
f.close()
```

Figura 57: Serialización de las codificaciones en un fichero pickle

8. El fichero *pickle* se carga y se le codifican las etiquetas con los nombres de las incrustaciones en la variable *le*.

```
text = "[INFO] cargando incrustaciones faciales..."
print(text)
bot.frai_train_prog(update, context, text)
data = pickle.loads(
    open(
        "/home/pi/Desktop/code/project/facial_recog/output/embeddings.pickle", "rb"
    ).read()
)

text = "[INFO] codificando etiquetas..."
print(text)
bot.frai_train_prog(update, context, text)
le = LabelEncoder()
labels = le.fit_transform(data["names"])
```

Figura 58: Codificación de las etiquetas

9. Se entrena el modelo que se va a usar para el reconocimiento facial. En el prototipo del sistema de seguridad se usa un kernel lineal. Una vez finalizado el entrenamiento, se guardan tanto el modelo de reconocimiento facial, *recognizer*, como el codificador de etiquetas, *le*.

```
text = "[INFO] entrenando modelo..."
print(text)
bot.frai_train_prog(update, context, text)
recognizer = SVC(C=1.0, kernel="linear", probability=True)
recognizer.fit(data["embeddings"], labels)

f = open(
    "/home/pi/Desktop/code/project/facial_recog/output/recognizer.pickle", "wb"
)
f.write(pickle.dumps(recognizer))
f.close()

f = open("/home/pi/Desktop/code/project/facial_recog/output/le.pickle", "wb")
f.write(pickle.dumps(le))
f.close()

text = "[INFO] entrenamiento finalizado"
print(text)
bot.frai_train_prog(update, context, text)
```

Figura 59: Finalización del entrenamiento

5.2.7.5. Función para efectuar el reconocimiento facial

Con las cuatro funciones explicadas anteriormente, el usuario puede cómodamente gestionar la IA de reconocimiento facial. Esta, como se ha mencionado al principio del subapartado, se acciona mediante un interruptor en la entrada de la vivienda.

El proceso que se sigue para el reconocimiento facial se puede ver en el siguiente flujograma:

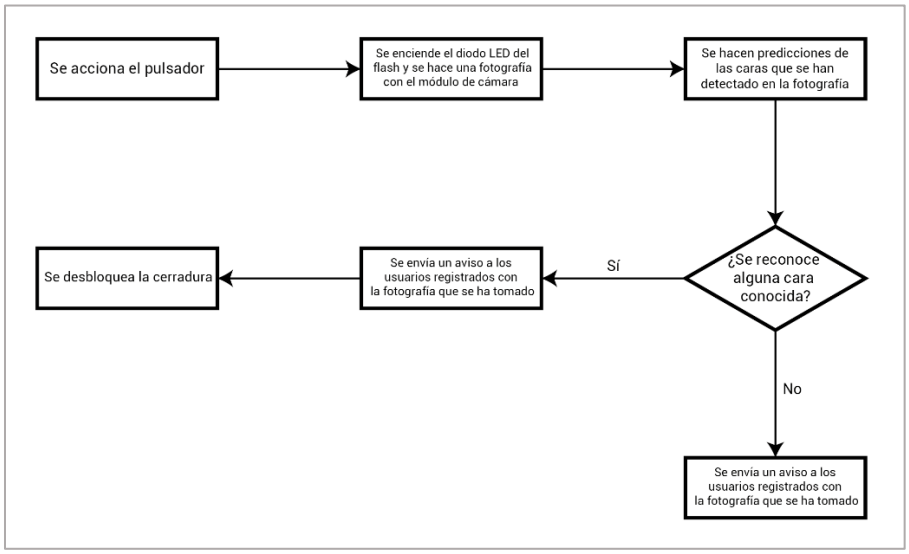


Figura 60: Flujograma para el reconocimiento facial

Su funcionamiento se muestra a continuación:

1. Cuando el interruptor es pulsado, se enciende un diodo LED que hace de flash y se llama la función de reconocimiento facial. La función de reconocimiento facial comienza inicializando el detector facial y el modelo Torch de incrustación facial, de la misma manera que la función de entrenamiento.

```
def recognize(input_img):
    protoPath = os.path.sep.join(
        [
            "/home/pi/Desktop/code/project/facial_recog/face_detection_model",
            "deploy.prototxt",
        ]
    )
    modelPath = os.path.sep.join(
        [
            "/home/pi/Desktop/code/project/facial_recog/face_detection_model",
            "res10_300x300_ssd_iter_140000.caffemodel",
        ]
    )
    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

    embedder = cv2.dnn.readNetFromTorch(
        "/home/pi/Desktop/code/project/facial_recog/openface.nn4.small2.v1.t7"
    )
```

Figura 61: Inicialización del detector facial y el modelo de incrustación facial

2. Después, se cargan los archivos *pickle* del modelo de reconocimiento facial y el codificador de etiquetas que se guardaron al final del entrenamiento de la IA.

```
recognizer = pickle.loads(
    open(
        "/home/pi/Desktop/code/project/facial_recog/output/recognizer.pickle", "rb"
    ).read()
)
le = pickle.loads(
    open("/home/pi/Desktop/code/project/facial_recog/output/le.pickle", "rb").read()
)
```

Figura 62: Inicialización de los modelos de reconocimiento facial y el codificador de etiquetas

3. Se comprueba si se ha pasado una imagen de entrada como argumento en la función. En caso negativo, se hará una foto mediante el comando *raspistill* por el módulo de cámara instalado en la entrada de la vivienda y se guardará. En caso positivo, se usará la imagen de entrada y se guardará. Esto es para evitar colisiones entre esta función y la función de transmisión en directo de la cámara, ya que se usan los mismos recursos para ambas. Así, en caso de activarse el reconocimiento facial mientras está activa la transmisión en directo, se pasará el fotograma que se esté captando en ese momento para la transmisión en directo y se pasará como imagen de entrada a la función de reconocimiento facial.

```
img_dir = "/home/pi/Desktop/code/project/facial_recog/input/input.jpg"
if input_img is None:
    cmd = "raspistill -o {}".format(img_dir)
    subprocess.call(cmd, shell=True)
elif input_img is not None:
    cv2.imwrite(img_dir, input_img)
image = cv2.imread(img_dir)
image = imutils.resize(image, width=600)
(h, w) = image.shape[:2]
```

Figura 63: Captura de imagen de entrada para reconocimiento facial

4. A continuación, se crea un *blob* de la imagen de entrada y se asigna como entrada del detector facial.

5. Ahora, se recorren con un bucle todas las detecciones que haga el detector facial. De cada una se extrae su región de interés.

```
for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        face = image[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]
        if fW < 20 or fH < 20:
            continue
```

Figura 64: Extracción región de interés para el reconocimiento facial

6. Se hace un *blob* de la región de interés y se extrae el vector 128-D que describe la cara. Este vector se pasará por el modelo de reconocimiento facial que se ha entrenado previamente, que hará una predicción sobre de quién es la cara que se ha detectado. Luego, se cogerá la etiqueta correspondiente a la predicción que se ha hecho, que se usará para notificar al usuario sobre quién ha accionado el pulsador.

```
faceBlob = cv2.dnn.blobFromImage(
    face, 1.0 / 255, (96, 96), (0, 0, 0), swapRB=True, crop=False
)
embedder.setInput(faceBlob)
vec = embedder.forward()
preds = recognizer.predict_proba(vec)[0]
j = np.argmax(preds)
proba = preds[j]
name = le.classes_[j]

text = "{} ({:.2f}%)".format(name, proba * 100)
y = startY - 10 if startY - 10 > 10 else startY + 10
cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)
cv2.putText(
    image, text, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2
)
names.append(name)
cont += 1
```

Figura 65: Elaboración de la predicción del reconocimiento facial

7. Una vez se obtiene la predicción del reconocimiento facial, se le notifica al usuario. Hay tres casos posibles: no se detecta ninguna cara, hay detecciones de caras desconocidas o hay detecciones de caras conocidas. En el último caso, se desbloquea la cerradura durante tres segundos y se vuelve a bloquear.

```
if cont == 0:
    msg_text = "Alguien ha tocado al timbre, pero no se ha reconocido ninguna cara"
    cv2.imwrite("recog_img.png", image)
    bot.recog_msg(msg_text)

else:
    known = 0
    for n in names:
        if n != "Unknown":
            known += 1

    if known == 0:
        join_names = ", ".join(names)
        msg_text = "Alguien ha tocado al timbre"
        cv2.imwrite("recog_img.png", image)
        bot.recog_msg(msg_text)

    else:
        join_names = ", ".join(names)
        msg_text = "{} ha(n) tocado al timbre".format(join_names)
        cv2.imwrite("recog_img.png", image)
        bot.recog_msg(msg_text)
        main.lock(0)
        time.sleep(3)
        main.lock(1)
```

Figura 66: Notificación al usuario del reconocimiento facial

5.2.8. Transmisión en directo del módulo de cámara

A demanda del usuario, se puede iniciar un servicio de transmisión en directo del vídeo del módulo de cámara de RPi situada a la entrada de la vivienda. El usuario, para esto debe enviar el comando */stream*, al que el bot le responderá con una URL a través de la cual podrá acceder a la transmisión en directo. Este proceso se puede entender mejor con la siguiente figura:

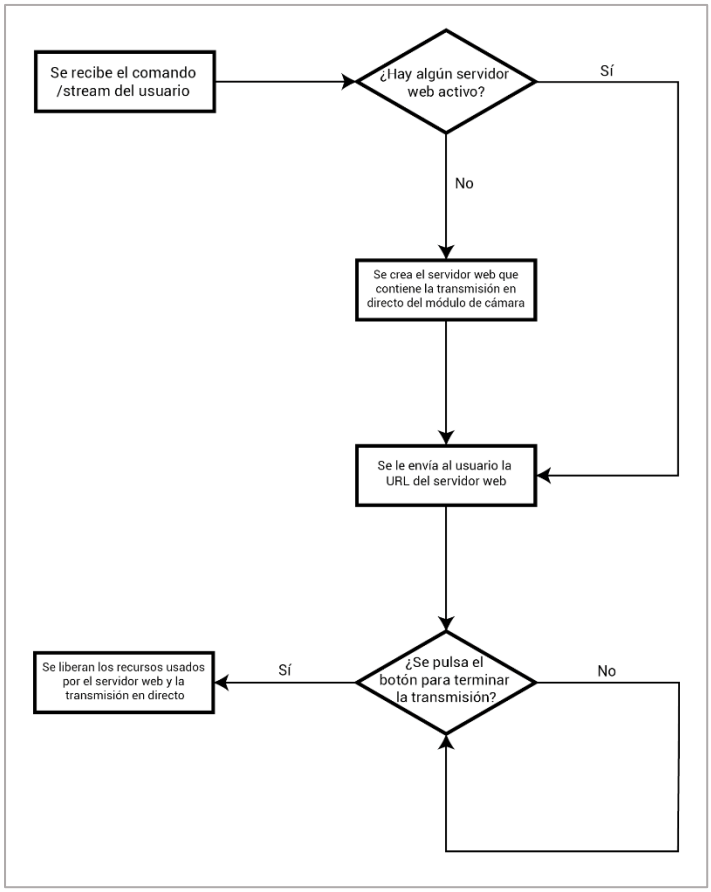


Figura 67: Flujograma del comando /stream

Para el propósito experimental del presente TFG, esta transmisión en directo se aloja en un servidor *Ad-Hoc* local. El usuario accede a este servidor mediante *ngrok*, para que pueda acceder desde fuera de la red local.

Ngrok se encarga de exponer el servidor local a internet mediante túneles introspectivos, haciendo así de proxy inverso (redireccionando las peticiones de la URL de *ngrok* al servidor local) [28]. Para la implementación de *ngrok* en el código se ha usado la librería *pyngrok* [29]. Para la servidor local se usa la librería *Flask*, que permite la administración del servidor mediante Python [30].

Para poder hacer que el vídeo del módulo de cámara transmita en directo al servidor local, se ha adaptado código de un proyecto basado en OpenCV [31]. A continuación, se explican fragmentos del código que permite el funcionamiento del servidor.

Como se ha mencionado, cuando el usuario le envía al bot el comando */stream* el bot llama a la función *webmain* para iniciar el servidor. Cuando esto sucede, la función comienza abriendo un túnel con *ngrok* para abrir el servidor local a internet público. Luego, la dirección pública que *ngrok* crea, se envía al usuario mediante el bot. Después de esto se ejecuta la función *streaming* que capturará el vídeo del módulo de cámara y *web.run* para iniciar el servidor web donde se aloja la transmisión en directo. Si el servidor web ya está ejecutándose en una instancia, se envía la URL del servidor ya creado, sin crear uno nuevo.

```
ts = None
tw = None
def webmain(update):
    global web_active, pubUrl, ts, tw
    if web_active is False:
        https_tunnel = ngrok.connect(5000, bind_tls=True)
        pubUrl = https_tunnel.public_url
        print(pubUrl)
        bot.stream_msg(update, pubUrl)

        web_active = True
        ts = Thread(target = streaming)
        ts.daemon = True
        ts.start()
        tw = Thread(target = web.run)
        tw.daemon = True
        tw.start()

    elif web_active is True:
        bot.stream_msg(update, pubUrl)
```

Figura 68: Función webmain

Una vez el usuario entra a la página web cuya URL que ha recibido del bot, podrá ver la retransmisión en directo del módulo de cámara y un botón para terminar la retransmisión en directo. Este botón liberará los recursos usados por el servidor.

Es importante anotar que, dado que el servidor web se apoya en *ngrok* para poder ser accesible desde cualquier ubicación sin tener que estar en la red de la RPi, está marcado como sitio web peligroso por los navegadores. Esto es porque *ngrok* es una herramienta usada para diferentes ataques de ingeniería social por hackers [32]. Por lo tanto, al acceder a la URL aparecerá un aviso de sitio web peligroso como el que se ve en la Figura 69, donde el usuario tendrá que pulsar el botón “Detalles” y después “acceder a este sitio no seguro”.

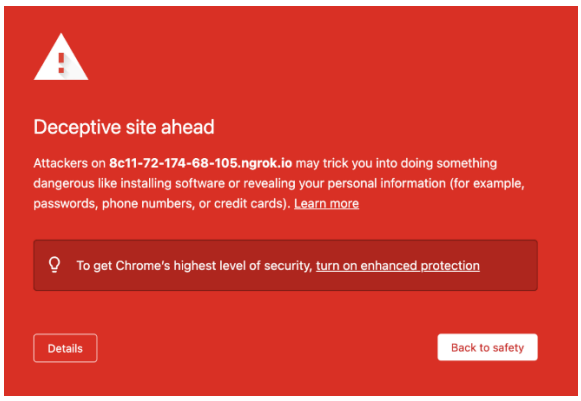


Figura 69: Pantalla de sitio web peligroso

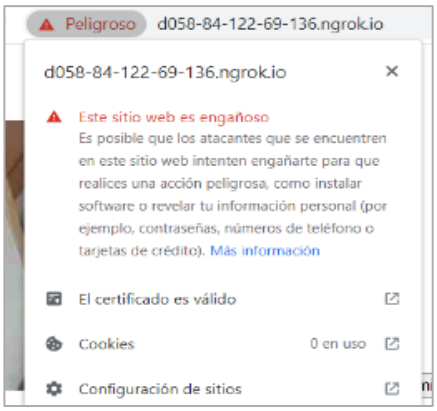


Figura 70: Aviso dentro de la página web

5.2.9. Registro

Cada uno de los eventos que suceden en el sistema queda recogido en un sistema de registros al que los usuarios pueden acceder desde el bot de Telegram mediante el comando */log*. A continuación, se muestra la función que creará y escribirá los ficheros de registro:

```
def log(logtxt):
    log_path = "/home/pi/Desktop/code/project/log/"
    y = datetime.now().strftime("%Y")
    ypath = "/home/pi/Desktop/code/project/log/{}/".format(y)
    m = datetime.now().strftime("%m")
    mpath = "/home/pi/Desktop/code/project/log/{}/{}/".format(y, m)
    d = datetime.now().strftime("%d")
    dpath = "/home/pi/Desktop/code/project/log/{}/{}/{}.txt".format(y, m, d)

    if not os.path.exists(ypath):
        os.makedirs(ypath)

    if not os.path.exists(mpath):
        os.makedirs(mpath)

    if not os.path.isfile(dpath):
        timestamp = datetime.now().strftime("%H:%M:%S")
        log_file = open(dpath, "w")
        log_file.write("[{}] {}".format(timestamp, logtxt))
        log_file.close()

    elif os.path.isfile(dpath):
        timestamp = datetime.now().strftime("%H:%M:%S")
        log_file = open(dpath, "a")
        log_file.write("\n[{}] {}".format(timestamp, logtxt))
        log_file.close()
```

Figura 71: Función log

Como se puede ver, la función crea un registro organizado por años, meses y días, para mayor comodidad del usuario final. Esta función será llamada al final de las diferentes funciones del bot, registrando los diferentes eventos.

Cuando el usuario envía el comando */log* al bot de Telegram, se le muestra en primer lugar un menú para escoger el año del registro que desea descargar. Una vez escoja el año, se le muestra un menú para escoger el mes y después el día. Cuando escoja el día, se le envía al usuario el archivo de texto correspondiente al registro del día que ha escogido.

El objetivo de este sistema de registro es el de complementar al historial de mensajes del chat con el bot. Por ejemplo, si alguna persona accionase el pulsador de la entrada, se enviaría una imagen de lo que capte la cámara a la conversación de Telegram y además quedaría recogido en el registro.

5.3. Presupuesto

En este subapartado se detalla el presupuesto para la realización del prototipo del sistema de seguridad del presente TFG.

5.3.1. Materiales

Ud.	Concepto	P. Unitario	Subtotal
1	Cables DuPont Macho - Macho (20 cm / 40 unidades)	1,60 €	1,60 €
1	Cables DuPont Hembra - Hembra (20 cm /40 unidades)	1,60 €	1,60 €
1	Cables DuPont Macho - Hembra (20 cm /40 unidades)	1,60 €	1,60 €
3	Módulo sensor PIR HC-SR501	1,85 €	5,55 €
1	Módulo 4 relés 5V	3,30 €	3,30 €
1	Memoria MicroSD 32GB (Clase 10)	7,95 €	7,95 €
1	Raspberry Pi 4 Model B (4 GB RAM)	79,50 €	79,50 €
1	Fuente Alimentación Raspberry Pi 4, USB-C, 5.1V 3A	8,20 €	8,20 €
1	DC 12V Cerradura magnética puerta eléctrica bloqueo de solenoide	12,95 €	12,95 €
1	Cámara Web PC HD 1080p	17,95 €	17,95 €
1	Transformador 240V a 12V 5A	14,75 €	14,75 €
1	Cable Micro HDMI a HDMI	7,85 €	7,85 €
1	Teclado y ratón inalámbricos	17,30 €	17,30 €
1	Placa de prototipo 16x5cm (830 puntos)	2,80 €	2,80 €
1	Cámara Raspberry Pi NoIR v1.3 (5MP 1080p)	22,47 €	22,47 €
1	Cable Ribbon 15cm para cámara Raspberry Pi	2,40 €	2,40 €
1	Tablero aglomerado	9,10 €	9,10 €
		TOTAL (SIN I.V.A.)	**216,87 €**

Tabla 1: Presupuesto para la compra de los materiales

El coste total de la compra de los materiales usados para el presente TFG es de DOSCIENTOS DIECISÉIS EUROS CON OCHENTA Y SIETE CÉNTIMOS sin I.V.A.

5.3.2. Mano de obra

Horas	Concepto	Precio/Hora	Subtotal
4	Especificación de los requisitos	50,00 €	200,00 €
12	Diseño	50,00 €	600,00 €
28	Implementación	50,00 €	1.400,00 €
6	Pruebas	50,00 €	300,00 €
		TOTAL (SIN I.V.A.)	2.500,00 €

Tabla 2: Presupuesto para la mano de obra

El coste de la mano de obra asciende a un total de DOS MIL QUINIENTOS EUROS sin I.V.A.

5.3.3. Presupuesto total

Capítulo	Importe
Materiales	216,87 €
Mano de obra	2.500,00 €
TOTAL SIN I.V.A.	2.716,87 €
I.V.A. (21%)	570,54 €
TOTAL	**3.287,41 €**

Tabla 3: Presupuesto total

Asciende el presupuesto total del presente TFG a TRES MIL DOSCIENTOS OCHENTA Y SIETE EUROS CON CUARENTA Y UN CÉNTIMOS.

Cabe anotar que el anterior presupuesto se ha realizado con respecto a la primera implementación del prototipo de sistema de seguridad. En el presupuesto para siguientes implementaciones del prototipo, no constarían las horas de especificación de requisitos ni de diseño y, además, las horas de implementación y pruebas se reducirían. Por lo tanto, el presupuesto total para implementaciones futuras asciende a MIL NOVECIENTOS CINCUENTA Y SEIS EUROS CON CUARENTA Y UN CÉNTIMOS:

Capítulo	Importe
Materiales	216,87 €
Mano de obra	1.400,00 €
TOTAL SIN I.V.A.	1.616,87 €
I.V.A. (21%)	339,54 €
TOTAL	**1.956,41 €**

Tabla 4: Presupuesto total para implementaciones futuras

6. Resultado y discusión

El resultado del presente TFG ha sido la implementación de las tecnologías IoT en el caso práctico de un sistema de seguridad con elementos domóticos, además de la incorporación de un bot de Telegram para la gestión del mismo.

Se han introducido las tecnologías usadas en el prototipo del sistema de seguridad, para pasar a evaluar las diferentes alternativas en cuanto a componentes del ecosistema IoT. Una vez hecho esto, se han escogido los elementos que componen el sistema de seguridad.

Luego, se han detallado las funciones de cada elemento en el sistema y su implementación dentro del código. La implementación en el código de los distintos elementos ha sido hecha de manera que las diferentes funciones puedan ejecutarse simultáneamente sin colisiones entre las mismas.

El bot de Telegram que se ha logrado, ha sido programado de manera que proporcione una experiencia cómoda al usuario final, pudiendo acceder a las diferentes funciones desde cualquier lugar, siempre y cuando cuente con conexión a internet.

Para simular la instalación de este prototipo de sistema de seguridad en una vivienda, se ha construido una maqueta en la que se han colocado los componentes.

Como se ha mencionado anteriormente, el resultado del presente TFG es un prototipo. De aplicarse el sistema de seguridad en un caso real, como su instalación en una vivienda, se deberían de añadir más actuadores y sensores. Además, el esquema electrónico cambiaría por completo, para dar cabida a los distintos elementos domóticos disponibles.

7. Conclusión y líneas futuras

Se puede concluir, a la luz de los resultados, que las tecnologías IoT son capaces de aportar flexibilidad y versatilidad a un sistema de seguridad sencillo a la hora de implementarlo en la práctica.

El hecho de que el prototipo del sistema de seguridad esté programado íntegramente en Python, demuestra la potencia y relativa facilidad que caracteriza este lenguaje. Librerías tan potentes como *OpenCV*, usada para proyectos de visión artificial, o *scikit-learn*, usada para el aprendizaje automático en el área de la inteligencia artificial, se pueden implementar fácilmente en Python. Esta facilidad, en parte, viene de las comunidades activas que hay detrás del lenguaje y de muchas de estas librerías, haciendo que la búsqueda de información sea más rápida y eficiente.

Como propuestas de mejora en líneas futuras se propone:

- La implementación de un servidor dedicado para la transmisión en directo del módulo de cámara de RPi, en vez del servidor web *Ad-Hoc* que se ha implementado.
- La adición de otros elementos domóticos al sistema, como podrían ser persianas motorizadas para mejorar la simulación de presencia.
- En la línea del anterior punto, mejorar la función de simulación de presencia con un patrón aleatorio que simule mejor un patrón de presencia humana en la vivienda.
- La implementación de una cámara de visión nocturna para la detección de movimiento en el interior de la vivienda.
- La creación de una app dedicada, en lugar o complementando el bot de Telegram.
- El uso de otros modelos para el entrenamiento de la Inteligencia Artificial, en vez del SVM (Support Vector Machine) Linear que se usa en el prototipo.
- La introducción de parámetros configurables por el usuario, como podría ser la longitud del GIF enviado por la función *detmo* o el ajuste de luminancia de las cámaras.
- La implementación de sensores PIR con parámetros configurables por software, para poder cambiar la sensibilidad y el intervalo de tiempo entre alertas.

8. Bibliografía

[1] K. Ashton, “That ‘Internet of Things’ Thing,” *RFID Journal*, Jun. 22, 2009. https://www.rfidjournal.com/that-internet-of-things-thing (accessed Jan. 20, 2022).

[2] A. Gillis, “What is internet of things (IoT)?,” *IoT Agenda*, Aug. 2021. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (accessed Jan. 20, 2022).

[3] K. K. Patel and S. M. Patel, “Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges,” *International Journal of Engineering Science and Computing*, 2016, doi: 10.4010/2016.1482.

[4] S. A. Gbadegeshin *et al.*, “WHAT IS AN ARTIFICIAL INTELLIGENCE (AI): A SIMPLE BUZZWORD OR A WORTHWHILE INEVITABILITY?,” in *ICERI2021 Proceedings*, Nov. 2021, vol. 1, pp. 468–479. doi: 10.21125/iceri.2021.0171.

[5] S. Ray, “Commonly used Machine Learning Algorithms (with Python and R Codes),” *Analitics Vidhya*, Sep. 09, 2017. https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ (accessed Jan. 21, 2022).

[6] R. Pallás Areny, *Sensores y Acondicionadores de Señal*, 4th ed. S.A. Marcombo, 2005.

[7] “SENSOR POR MICROONDAS (MWS) EN COMPARACIÓN CON EL SENSOR PIR (INFRARROJO PASIVO),” *Ansell Lighting*. https://ansell-lighting.es/news/Sensor-Por-Microondas (accessed Jan. 21, 2022).

[8] “HC-SR501 PIR MOTION DETECTOR.” Accessed: Feb. 05, 2022. [Online]. Available: https://www.mpja.com/download/31227sc.pdf

[9] “Tinker Board S,” *ASUS*. https://www.asus.com/es/Networking-IoT-Servers/AIoT-Industrial-Solutions/All-series/Tinker-Board-S/ (accessed Jan. 27, 2022).

[10] “Arduino Uno, partes, componentes, para qué sirve y donde comprar,” *DescubreArduino*, Jul. 03, 2021. https://descubrearduino.com/arduino-uno/ (accessed Jan. 27, 2022).

[11] “Raspberry Pi 4 Computer Model B,” 2019. [Online]. Available: www.raspberrypi.org

[12] “Raspberry Pi Camera, what’s the big deal?,” *StackExchange*, Apr. 20, 2016. https://raspberrypi.stackexchange.com/questions/19379/raspberry-pi-camera-whats-the-big-deal (accessed Jan. 21, 2022).

[13] “Aumenta la seguridad de tu hogar gracias a la simulación de presencia,” *Simon Electric*, Feb. 2021. https://www.simonelectric.com/blog/aumenta-la-seguridad-de-tu-hogar-gracias-la-simulacion-de-presencia (accessed Jan. 22, 2022).

[14] “Como hacer una cerradura electrica casera por tu propia cuenta,” *Curso Cerrajería*. http://www.cursodecerrajeria.info/blog/como-hacer-una-cerradura-electrica/ (accessed Jan. 22, 2022).

[15] I. Porro, “IoT: protocolos de comunicación, ataques y recomendaciones,” *INCIBE-CERT*, Feb. 07, 2019. https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones (accessed Jan. 28, 2022).

[16] “MQTT vs HTTP – Qué elegir para tu proyecto IoT,” *IoT Consulting*, Jul. 04, 2019. https://iotconsulting.tech/mqtt-vs-http-que-elegir-para-tu-proyecto-iot/ (accessed Jan. 28, 2022).

[17] A. Bassi, “Introducción a AMQP,” *Goto IoT*, Jul. 22, 2021. https://www.gotoiot.com/pages/articles/amqp_intro/index.html (accessed Feb. 05, 2022).

[18] D. Barnett, “MQTT and DDS: Machine to Machine Communication in IoT,” *RTI*, May 08, 2013. https://www.rti.com/blog/mqtt-dds-m2m-protocol-internet-of-things/ (accessed Jan. 28, 2022).

[19] “PyMongo 4.0.1 Documentation,” *Read the Docs*. https://pymongo.readthedocs.io/en/stable/ (accessed Jan. 06, 2022).

[20] “IoT Based Solenoid Door Lock using Raspberry Pi 4,” Feb. 18, 2020. https://iotdesignpro.com/projects/iot-based-solenoid-door-lock-using-raspberry-pi-4?page=1 (accessed Dec. 12, 2021).

[21] “threading — Thread-based parallelism,” *Python docs*. https://docs.python.org/3/library/threading.html (accessed Jan. 31, 2022).

[22] “Bots FAQ,” *Telegram*. https://core.telegram.org/bots/faq#my-bot-is-hitting-limits-how-do-i-avoid-this (accessed Jan. 30, 2022).

[23] “multiprocessing — Process-based parallelism,” *Python docs*. https://docs.python.org/3/library/multiprocessing.html (accessed Jan. 31, 2022).

[24] A. Rosebrock, “Basic motion detection and tracking with Python and OpenCV,” *pyimagesearch*, May 25, 2015. https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/ (accessed Feb. 01, 2022).

[25] “Apply Gamma Correction to an Image using OpenCV,” Jul. 24, 2021. https://lindevs.com/apply-gamma-correction-to-an-image-using-opencv/ (accessed Nov. 13, 2021).

[26] A. Rosebrock, “OpenCV Face Recognition,” Sep. 24, 2018. https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/ (accessed Dec. 26, 2021).

[27] J. Hellsten and T. Karras, “Flickr-Faces-HQ Dataset (FFHQ),” Feb. 04, 2019. https://github.com/NVlabs/ffhq-dataset (accessed Dec. 28, 2021).

[28] J. A. Ponce, “Ngrok: una herramienta con la que hacer público tu localhost de forma fácil y rápida,” *SDOS*, Sep. 01, 2020. https://www.sdos.es/blog/ngrok-una-herramienta-con-la-que-hacer-publico-tu-localhost-de-forma-facil-y-rapida (accessed Feb. 04, 2022).

[29] A. Laird, “Pyngrok (a Python wrapper for ngrok) Documentation,” *Read the Docs*, 2020. https://pyngrok.readthedocs.io/en/latest/index.html (accessed Feb. 04, 2022).

[30] “Flask Documentation,” *Pallets Projects*. https://flask.palletsprojects.com/en/2.0.x/ (accessed Feb. 04, 2022).

[31] A. Rosebrock, “OpenCV – Stream video to web browser/HTML page,” Sep. 02, 2019. https://www.pyimagesearch.com/2019/09/02/opencv-stream-video-to-web-browser-html-page/ (accessed Feb. 05, 2022).

[32] “Ngrok Platform Abused By Hackers To Deliver A New Wave Of Phishing Attacks,” *Cyble*, Feb. 15, 2021. https://blog.cyble.com/2021/02/15/ngrok-platform-abused-by-hackers-to-deliver-a-new-wave-of-phishing-attacks/ (accessed Feb. 05, 2022).

[33] “Setting up your Raspberry Pi.” https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/0 (accessed Nov. 02, 2021).

[34] A. Rosebrock, “Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster,” Sep. 16, 2019. https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/ (accessed Nov. 04, 2021).

Anexos

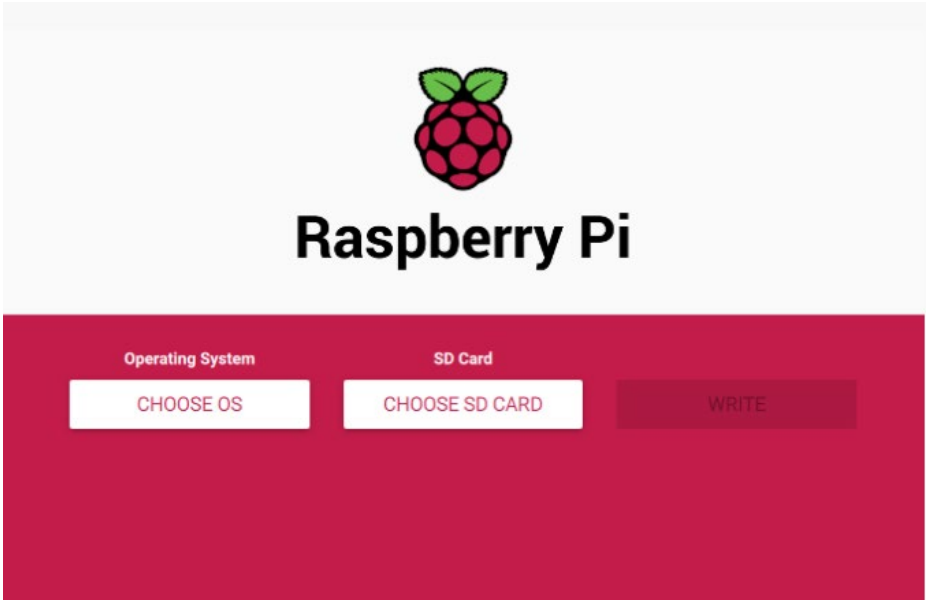
Anexo I: Guía de instalación

I.I. Introducción

En este anexo se mostrará paso a paso la instalación y configuración de las diferentes dependencias necesarias para el correcto funcionamiento del sistema de seguridad.

I.II. Preparación inicial

Para el sistema de seguridad se ha usado una Raspberry Pi 4B con el sistema operativo Raspbian 10 (Buster). Para su instalación en la Raspberry Pi, se debe usar un programa para el formateo e instalación del sistema operativo en una tarjeta micro SD, como puede ser *Raspberry Pi Imager* [33]:

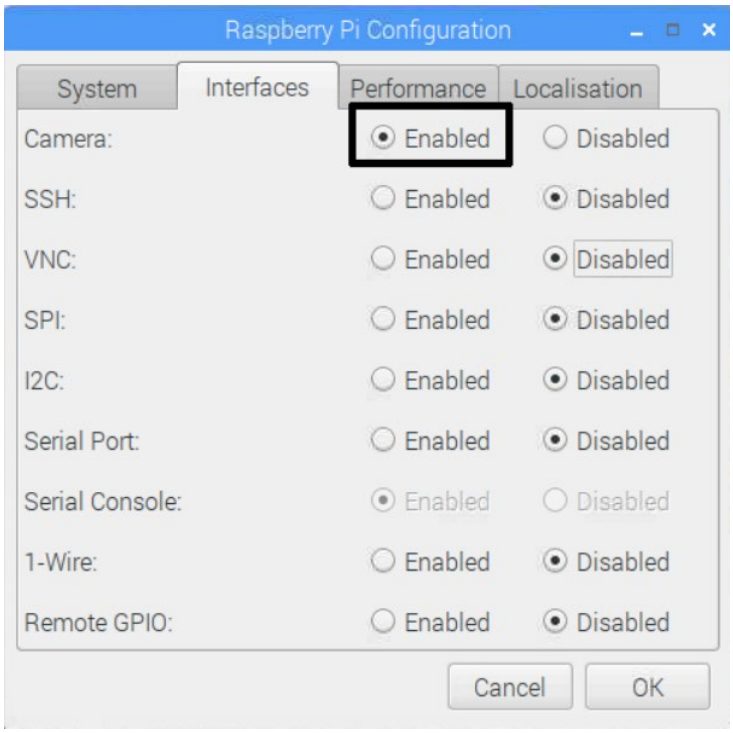


Una vez instalado, se puede poner la tarjeta micro SD en la ranura de la Raspberry Pi y conectar la alimentación de la Raspberry Pi para encenderla. Se puede conectar un monitor, un teclado y un ratón para más comodidad.

Hecho esto, se accede a la Raspberry Pi. En primer lugar, se abre un terminal y se ejecuta el siguiente comando para actualizar los paquetes instalados:

sudo apt update && sudo apt upgrade -y

Luego, hay que configurar la Raspberry Pi para poder conectar el módulo de cámara. Para ello, se abre ***Preferences > Raspberry Pi Configuration*** y, en la pestaña ***Interfaces***, se marca la siguiente casilla:



Hecho esto, se instala IDLE3 y Python3 para poder ejecutar el programa del sistema de seguridad, ya que está escrito en Python. Se instala con el siguiente comando:

sudo apt install python3 idle3

I.III. Instalación de OpenCV

Para las funciones relacionadas con imagen y vídeo, por ejemplo, la función de detección de movimiento, se ha usado la librería OpenCV. Esta librería es conocida por la variedad de utilidades que proporciona. Para su instalación completa, se han de seguir los siguientes pasos [34] (*Nota: cada ❖ indica el inicio de un comando nuevo*).

En primer lugar, se instalan las dependencias necesarias con el siguiente comando:

- *sudo apt install build-essential cmake pkg-config libjpeg-dev *
  *libtiff5-dev libjasper-dev libpng-dev libavcodec-dev libavformat-dev *
  *libswscale-dev libv4l-dev libxvidcore-dev libx264-dev *
  *libfontconfig1-dev libcairo2-dev libgdk-pixbuf2.0-dev *
  *libpango1.0-dev libgtk2.0-dev libgtk-3-dev libatlas-base-dev *
  *gfortran libhdf5-dev libhdf5-serial-dev libhdf5-103 *
  libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5 python3-dev -y

Habiendo instalado las dependencias, hay que instalar *pip*. *Pip* es la herramienta que hace posible la creación del entorno virtual en el que se trabajará con OpenCV. Para instalarlo se usan los siguientes comandos:

- *wget https://bootstrap.pypa.io/get-pip.py*
- *sudo python get-pip.py*
- *sudo python3 get-pip.py*
- *sudo rm -rf ~/.cache/pip*

Teniendo *pip* instalado, se ejecuta el siguiente comando para instalar *virtualenv*:

- *sudo pip install virtualenv virtualenvwrapper*

Hecho esto, se tiene que abrir el fichero *~/.bashrc* con un editor de texto (por ejemplo *nano*) y añadir las siguientes líneas al final del fichero:

- *export WORKON_HOME=$HOME/.virtualenvs*
- *export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3*
- *source /usr/local/bin/virtualenvwrapper.sh*

Después de esto, se recarga el fichero *~/.bashrc* con el comando que sigue:

- *source ~/.bashrc*

Luego, se crea el entorno virtual:

- *mkvirtualenv cv -p python3*

También se tiene que instalar la API de *PiCamera* para el módulo de cámara que se va a conectar:

- *pip install "picamera[array]"*

Ahora, se descarga OpenCV con los siguientes comandos:

- *cd ~*
- *wget -O opencv.zip https://github.com/opencv/opencv/archive/4.1.1.zip*
- *wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.1.1.zip*
- *unzip opencv.zip*
- *unzip opencv_contrib.zip*
- *mv opencv-4.1.1 opencv*
- *mv opencv_contrib-4.1.1 opencv_contrib*

Hecho esto, se instala OpenCV en el entorno virtual:

- *workon cv*
- *pip install numpy*
- *cd ~/opencv*
- *mkdir build*
- *cd build*
- *cmake -D CMAKE_BUILD_TYPE=RELEASE *
  *-D CMAKE_INSTALL_PREFIX=/usr/local *
  *-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules *
  *-D ENABLE_NEON=ON *
  *-D ENABLE_VFPV3=ON *
  *-D BUILD_TESTS=OFF *

*-D INSTALL_PYTHON_EXAMPLES=OFF *

*-D OPENCV_ENABLE_NONFREE=ON *

*-D CMAKE_SHARED_LINKER_FLAGS=-latomic *

-D BUILD_EXAMPLES=OFF ..

- *sudo make install*

Después de haber ejecutado estos comandos, el proceso de compilación de OpenCV empezará. Cuando finalice, se tienen que configurar los enlaces simbólicos:

- *cd /usr/local/lib/python3.7/site-packages/cv2/python-3.7*
- *sudo mv cv2.cpython-37m-arm-linux-gnueabihf.so cv2.so*
- *cd ~/.virtualenvs/cv/lib/python3.7/site-packages/*
- *ln -s /usr/local/lib/python3.7/site-packages/cv2/python-3.7/cv2.so cv2.so*

Con esto, OpenCV queda instalado en el entorno virtual y está listo para usarse.

I.IV. Instalación de dependencias

El código del sistema de seguridad usa varias librerías que no vienen en los paquetes instalados por defecto en Raspbian. Para poder hacer uso de estas librerías, hay que instalar sus paquetes mediante *pip* en el entorno virtual que se ha creado al instalar OpenCV en el subapartado *I.III. Instalación de OpenCV*. Para ello, se hace uso de los siguientes comandos:

- *workon cv*
- *pip install numpy imutils scipy imageio flask pyngrok Pillow RPi.GPIO gpiozero pygame dlib fase-recognition scikit-learn pymongo pymongo[srv]*
- *pip3 install python-telegram-bot*

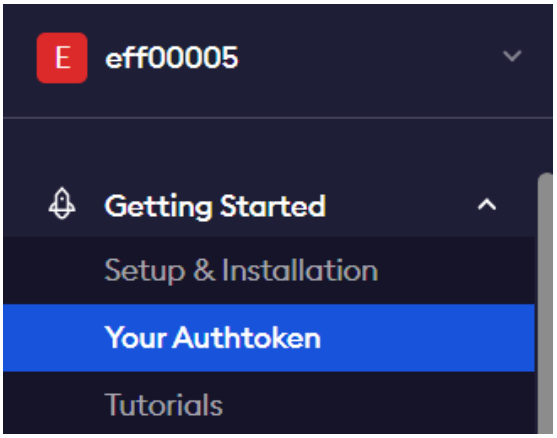
Así, quedan instaladas las dependencias usadas en el código.

I.V. Configuraciones varias

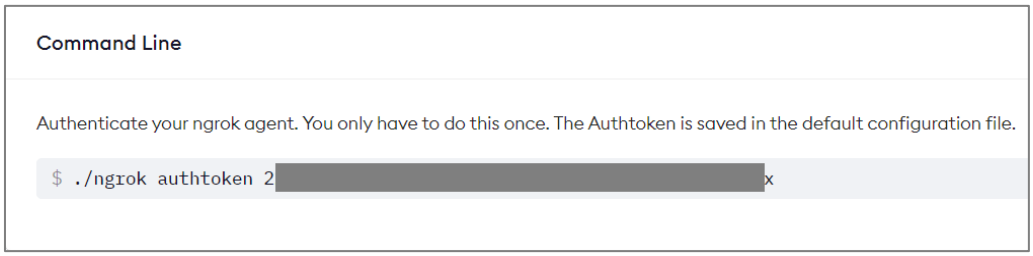
I.V.I. Ngrok

Para completar la instalación, sólo queda realizar algunas configuraciones. La primera que hace falta hacer es la de *ngrok*. Como se ha mencionado en apartados anteriores de la memoria, *ngrok* le permite al usuario poder acceder desde cualquier lugar a la transmisión en directo del módulo de cámara instalado a la entrada de la vivienda. Para esto, hace falta registrarse en la página web de *ngrok* y obtener el *Authtoken*.

Este *Authtoken* es lo que le permitirá a *ngrok* crear nuevos túneles para hacer público el servidor local del sistema de seguridad. Para obtenerlo, dentro de la página web de *ngrok*, en el panel de la izquierda se accede a “Your Authtoken”:



Ahí se podrá ver el siguiente apartado:



El comando de la Figura 75 es el que se tendrá que ejecutar en el directorio de instalación de *ngrok*. Para ello, se ejecutan los siguientes comandos en un terminal:

- *cd ./.virtualenvs/cv/bin/*

- *./ngrok authtoken [Authtoken]*

Con esto, *ngrok* quedará listo para usarse.

I.V.II. Inicio automático

Para que el sistema de seguridad se inicie cuando se encienda la Raspberry Pi, se ha de configurar el fichero *autostart*. Este se configura para ejecutar un archivo *bash* llamado "startup.sh" que contiene el siguiente código:

```
1  !#/bin/sh
2  sleep 20
3  source /home/pi/.virtualenvs/cv/bin/activate
4  sudo /home/pi/.virtualenvs/cv/bin/python /home/pi/Desktop/code/project/bot.py
```

En la línea 2 se le da tiempo a la Raspberry Pi para iniciarse por completo y, en las líneas siguientes, se activa el entorno virtual en el que se han instalado todas las dependencias y se ejecuta el programa principal *bot.py*.

Ahora, se modifica el archivo */boot/config.txt* quitando el comentario en la línea que se marca en la siguiente figura:

```
# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=800

# use edid_file in boot folder
#hdmi_edid_file=1

# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (this will force VGA)
hdmi_group=2
hdmi_mode=28
```

Esto hará que el programa del sistema de seguridad se pueda ejecutar, tanto si hay un monitor conectado a la Raspberry Pi, como si no lo hay. Esto se hace porque la Raspberry Pi, al iniciarse comprueba si hay un monitor conectado y, en función del resultado de esa comprobación, se ejecutará *autostart* o no. La razón por la que lo anterior explicado es así, es que *autostart* es dependiente del entorno del escritorio, que no se puede iniciar sin un monitor.

Ahora se muestra la modificación que hay que hacer en el fichero */etc/xdg/lxsession/LXDE-pi/autostart*:

```
@lxpanel --profile LXDE-pi
@pcmanfm --desktop --profile LXDE-pi
@screensaver -no-splash
@bash /home/pi/startup.sh
```

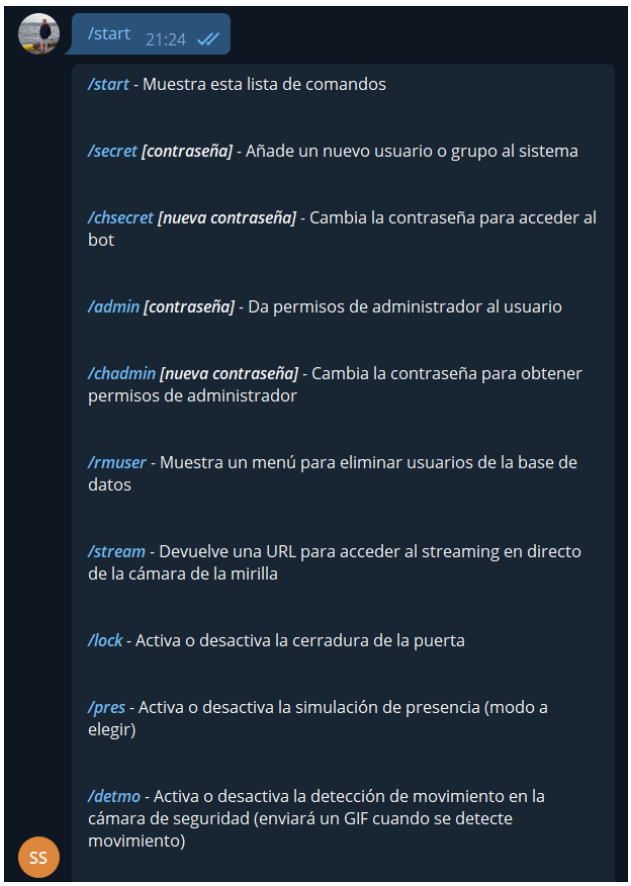
Anexo II: Manual de uso

II.I. Introducción

El presente anexo tiene el objetivo de servir de guía para el manejo del sistema de seguridad por parte del usuario. A continuación, se muestra el uso de las diferentes funcionalidades a disposición del usuario.

II.II. Comando */start*

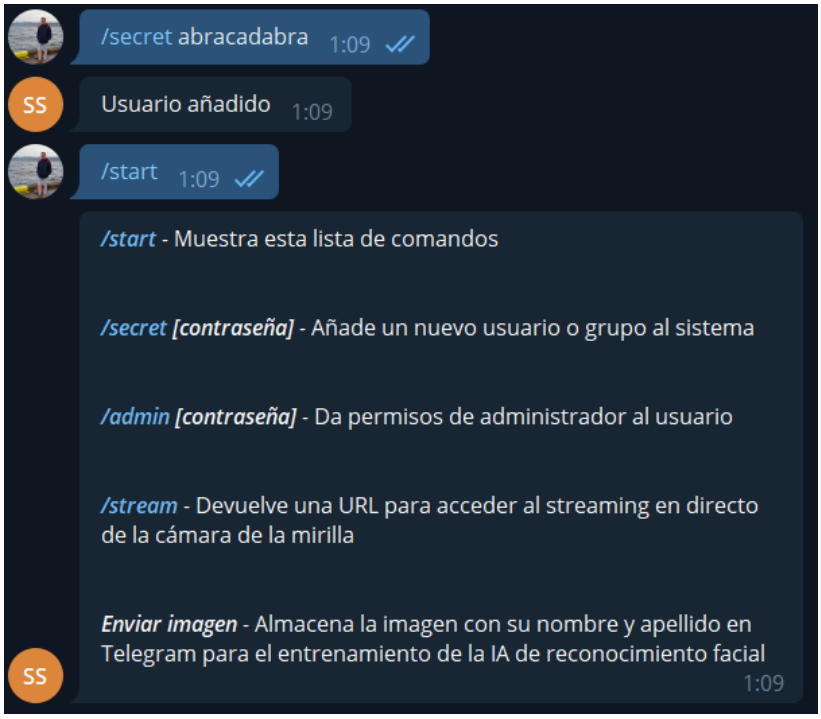
Cuando el usuario envía el comando */start*, el bot devuelve el listado de todos los comandos disponibles. Desde este listado se puede ver los comandos acompañados de sus funcionalidades:



II.III. Comando */secret*

Con el comando */secret* se registran usuarios nuevos en la base de datos. Registrándose, obtienen permisos de usuarios base y, por lo tanto, acceso a funciones básicas y a las alertas del sistema de seguridad.

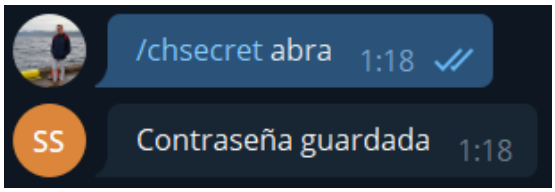
Para que un usuario se pueda registrar, tiene que introducir la contraseña adecuada junto con el comando. A continuación, se muestra una demostración del uso de este comando:



En la Figura 80 se puede ver el listado de funciones básicas a las que tiene acceso el usuario base.

II.IV. Comando */chsecret*

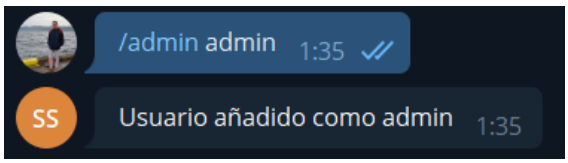
Este comando permite a los usuarios administradores cambiar la contraseña usada para registrar usuarios base con el comando */secret*. Para ello, se ha de enviar el comando */chsecret* acompañado de la nueva contraseña que se quiere configurar. En la siguiente figura se ve un ejemplo de uso de este comando:



Hecho esto, la contraseña de registro queda configurada como “abra”. Entonces si un usuario quisiese registrarse tendría que usar el comando */secret abra*.

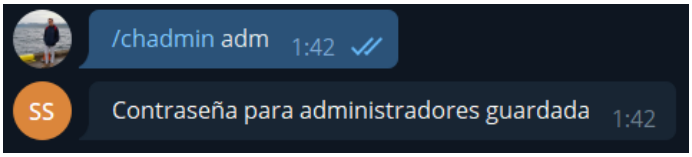
II.V. Comando */admin*

Con este comando, el usuario puede obtener permisos de administrador y ganar acceso a todas las funcionalidades del sistema. La forma de usar este comando es igual a la del comando */secret*. A continuación, se muestra un ejemplo de uso:



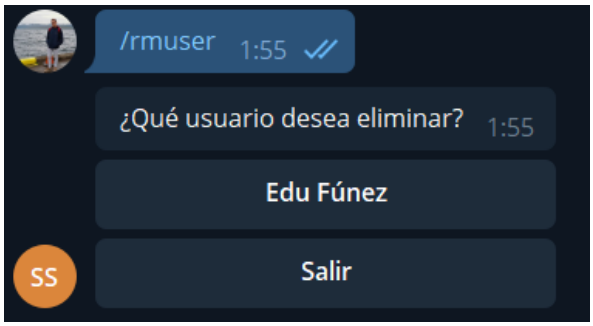
II.VI. Comando */chadmin*

La funcionalidad de este comando es similar a la del comando */chsecret*, con la única diferencia de que, en este caso, se cambia la contraseña para el comando */admin*:

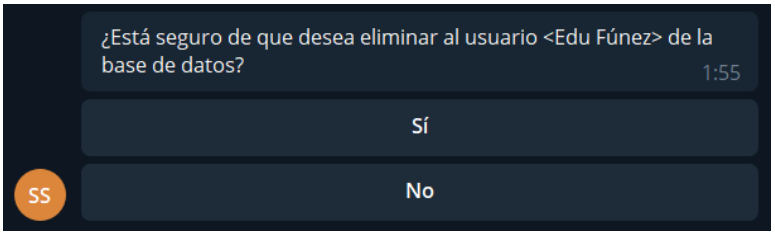


II.VII. Comando */rmuser*

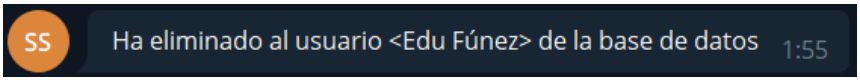
El comando */rmuser* permite a los usuarios administradores eliminar usuarios de la base de datos. Para ello, en primer lugar, se introduce el comando:



Como se puede ver en la Figura 84, el bot devuelve un menú desde el que el usuario puede seleccionar el usuario a eliminar. Cuando se selecciona el usuario a eliminar, se muestra el siguiente menú de confirmación:

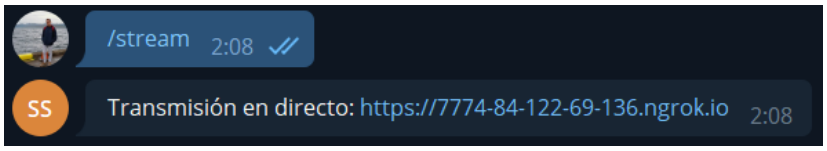


Cuando se pulsa la opción “Sí”, se elimina el usuario de la base de datos y el bot devuelve el siguiente mensaje confirmando la eliminación:



II.VIII. Comando */stream*

Este comando tiene la función de iniciar el servidor web donde se aloja la transmisión en directo de la cámara a la entrada de la vivienda. Cuando el usuario envía este comando, el bot le devuelve la URL para acceder a la página web donde está la transmisión en directo:

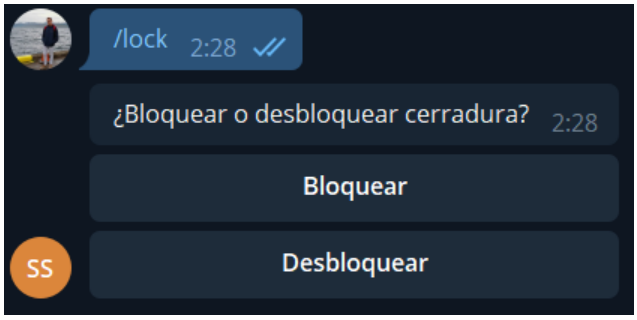


Cuando se accede a la página web, se ve lo siguiente:

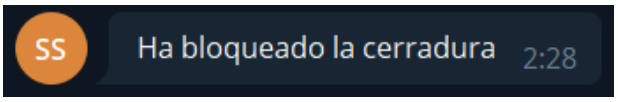
Como se puede observar en la Figura 88, en el centro de la página web, se encuentra la imagen captada por la cámara, debajo de la cual, se encuentra el botón “Terminar transmisión”. Este último se pulsa para liberar los recursos usados para la transmisión y cerrar el servidor web.

II.IX. Comando */lock*

Con este comando, el usuario puede controlar la cerradura de solenoide que hay instalada en la puerta de la vivienda. Cuando se envía el comando al bot, se muestra el siguiente menú:

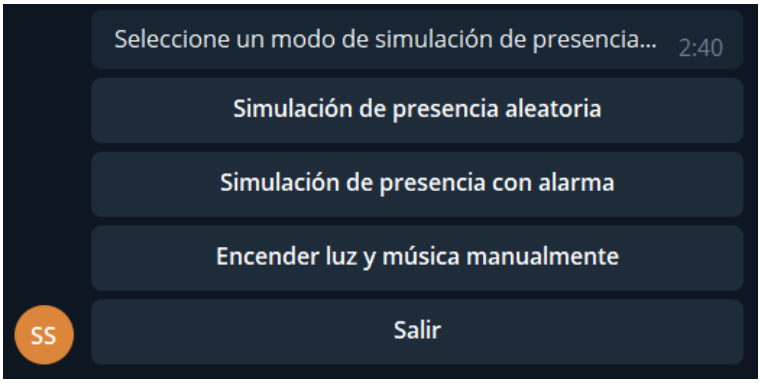


Desde el menú mostrado en la Figura 89, el usuario puede elegir si bloquear o desbloquear la cerradura. Una vez se selecciona una de las opciones, se envía un mensaje de confirmación al usuario:



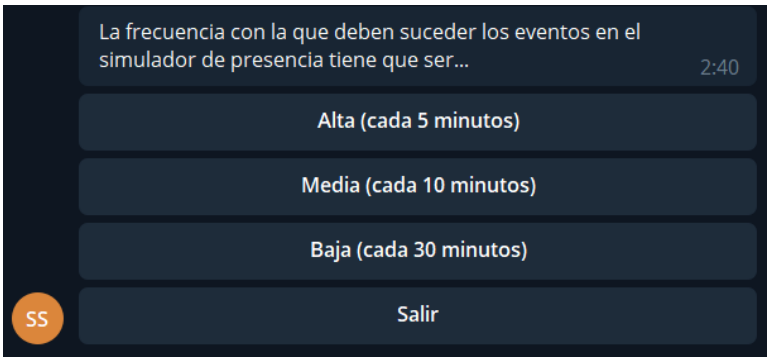
II.X. Comando */pres*

Con este comando, el usuario puede activar o desactivar la simulación de presencia. Cuando se envía el comando, el bot muestra el siguiente menú desde donde se podrá escoger el modo de la simulación de presencia:

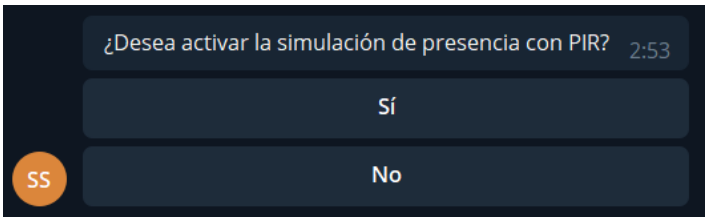


Las opciones que se muestran en el menú son:

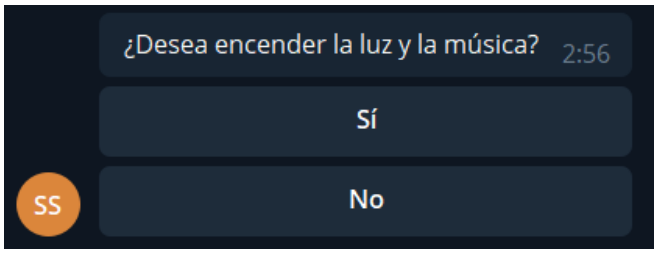
- Simulación de presencia aleatoria: Si se selecciona este modo, se encenderán o apagarán las luces o la música en intervalos aleatorios. Cuando se selecciona esta opción, se muestra el siguiente menú para elegir la frecuencia con la que sucederán los eventos:



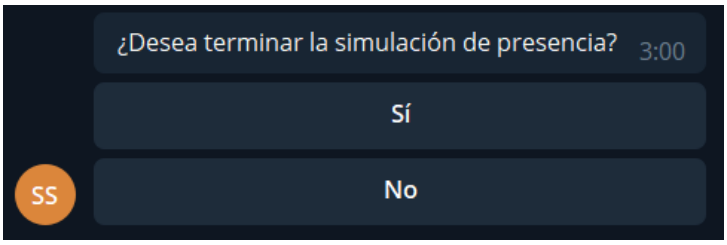
- Simulación de presencia con alarma: Si se selecciona este modo, cuando se detecte movimiento en alguno de los sensores PIR, se encienden las luces y se activa una alarma sonora. Cuando se selecciona este modo se muestra el siguiente menú de confirmación:



- Encender luz y música manualmente: Cuando se seleccione este modo, se enciende la luz y la música. Tras seleccionar el modo, aparece el siguiente menú de confirmación:

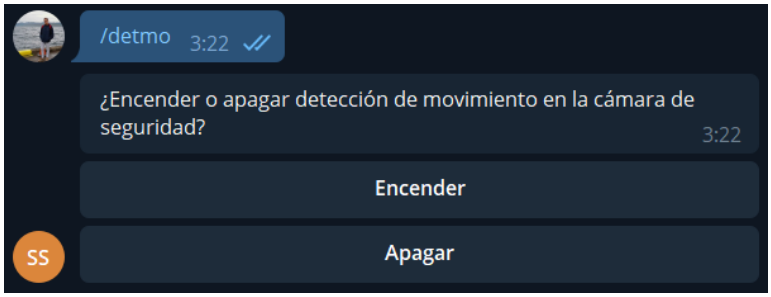


Cuando se desea desactivar la simulación de presencia, se tiene que enviar otra vez el comando */pres*, que devolverá el siguiente menú para desactivar la simulación de presencia:

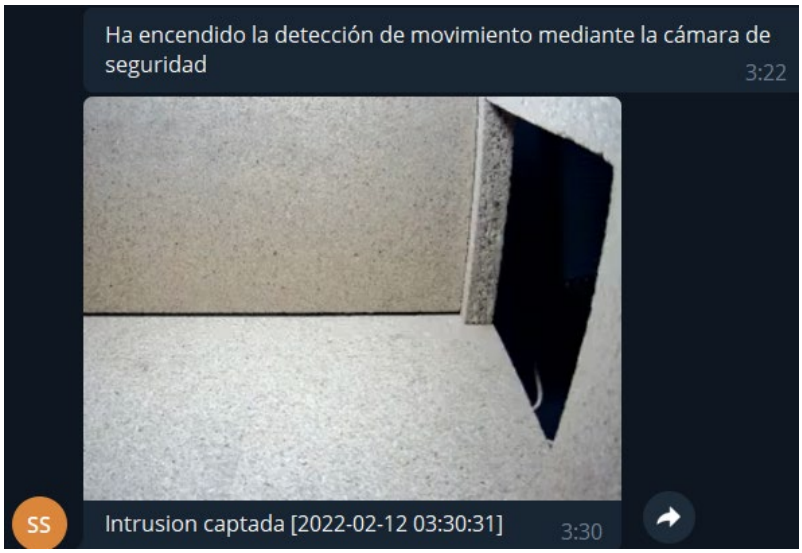


II.XI. Comando */detmo*

Este comando activa la detección de movimiento mediante la cámara de seguridad. Cuando se detecte algún movimiento, se envía un GIF a los usuarios registrados donde se podrá ver los fotogramas donde se ha detectado movimiento. Cuando el usuario envía el comando */detmo*, se muestra el siguiente menú al usuario:



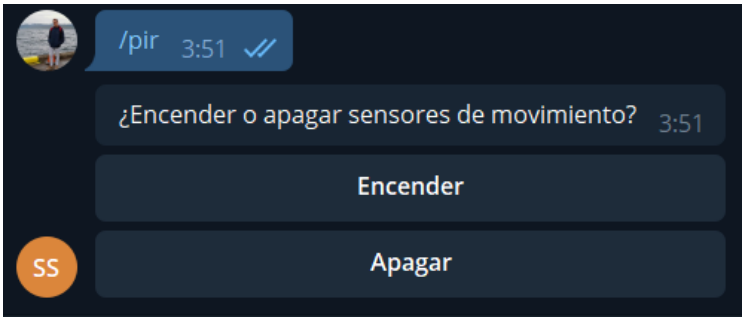
Cuando se enciende, la cámara de seguridad se inicializa y, tras unos segundos, empieza a captar vídeo. Si detecta algún movimiento enviará un aviso con un GIF de la detección a los usuarios registrados, como se ve en la siguiente figura:



Si se envía el comando */detmo* otra vez, se puede apagar la detección de movimiento mediante el menú de la Figura 96.

II.XII. Comando */pir*

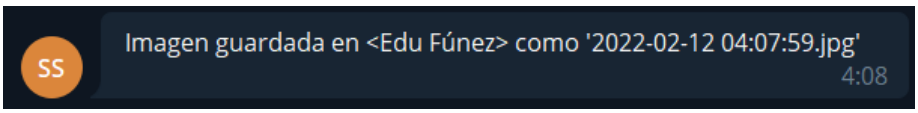
Con el comando */pir*, el usuario puede encender o apagar la detección de movimiento mediante sensores PIR. Cuando los sensores detecten movimiento se les enviará un aviso a todos los usuarios registrados. Si el usuario envía el comando */pir*, se le muestra el siguiente menú:



El aviso que se le enviará a los usuarios cuando se detecte movimiento será como el que se muestra a continuación:

II.XIII. Enviar imagen

Cuando el usuario envía una imagen, el bot almacenará y clasificará la imagen para usarla en el próximo entrenamiento que se haga de la IA de reconocimiento facial. Luego enviará el siguiente mensaje al usuario:

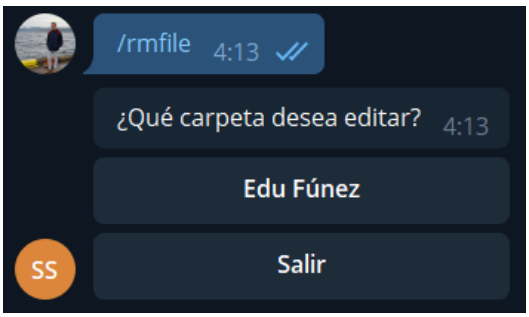


Como se puede ver en la Figura 100, el bot guarda la imagen en una carpeta con el nombre del usuario que la envía. Todas las imágenes que ese usuario envíe se almacenarán en esa carpeta.

Para el entrenamiento de la IA, se recomienda que cada usuario envíe entre 10 y 20 imágenes al bot.

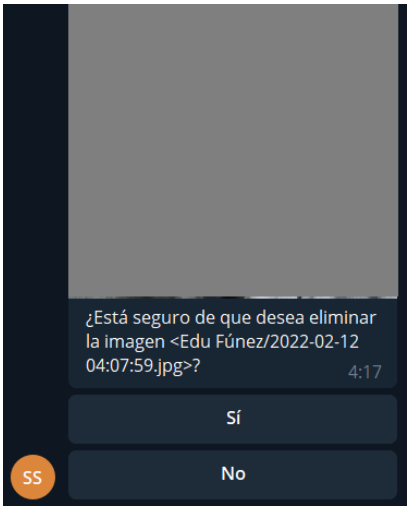
II.XIV. Comando */rmfile*

Este comando permite al usuario eliminar una imagen en concreto de alguna de las carpetas que se usan para el entrenamiento de la IA de reconocimiento facial. Cuando se envía el comando al bot, se muestra el siguiente menú para escoger la carpeta de la que se quiere eliminar la imagen:



Cuando se selecciona la carpeta, aparece el siguiente menú para seleccionar la imagen a eliminar:

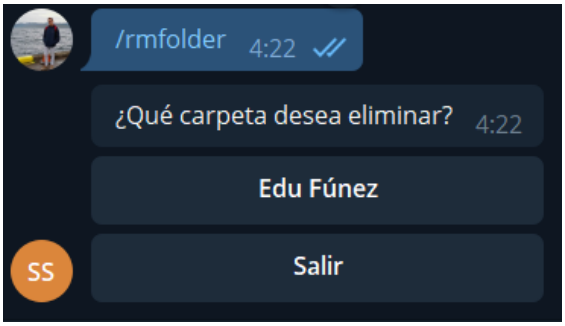
Cuando se selecciona la imagen a eliminar, el bot envía una previsualización de la imagen seleccionada acompañada de un menú de confirmación:



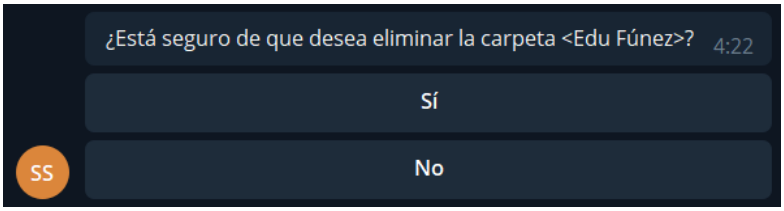
Si se selecciona la opción “Sí”, se elimina la imagen seleccionada.

II.XV. Comando */rmfolder*

El comando */rmfolder* le permite al usuario eliminar del sistema de seguridad la carpeta correspondiente a las imágenes enviadas por un usuario en concreto. Cuando se envía el comando */rmfolder*, se muestra el siguiente menú:



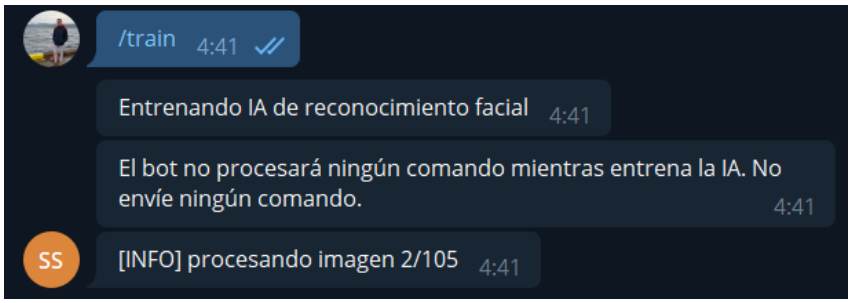
Cuando se selecciona la carpeta a eliminar, se devuelve el siguiente menú de confirmación:



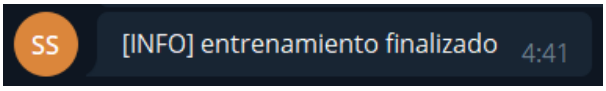
Si se selecciona la opción “Sí”, se elimina la carpeta del sistema de seguridad.

II.XVI. Comando */train*

Este comando permite al usuario iniciar el entrenamiento de la IA de reconocimiento facial. Este entrenamiento usará todas las imágenes enviadas por los usuarios, además de un conjunto de imágenes para la clasificación de caras desconocidas. Cuando el usuario envía el comando */train*, se devuelven los siguientes mensajes:

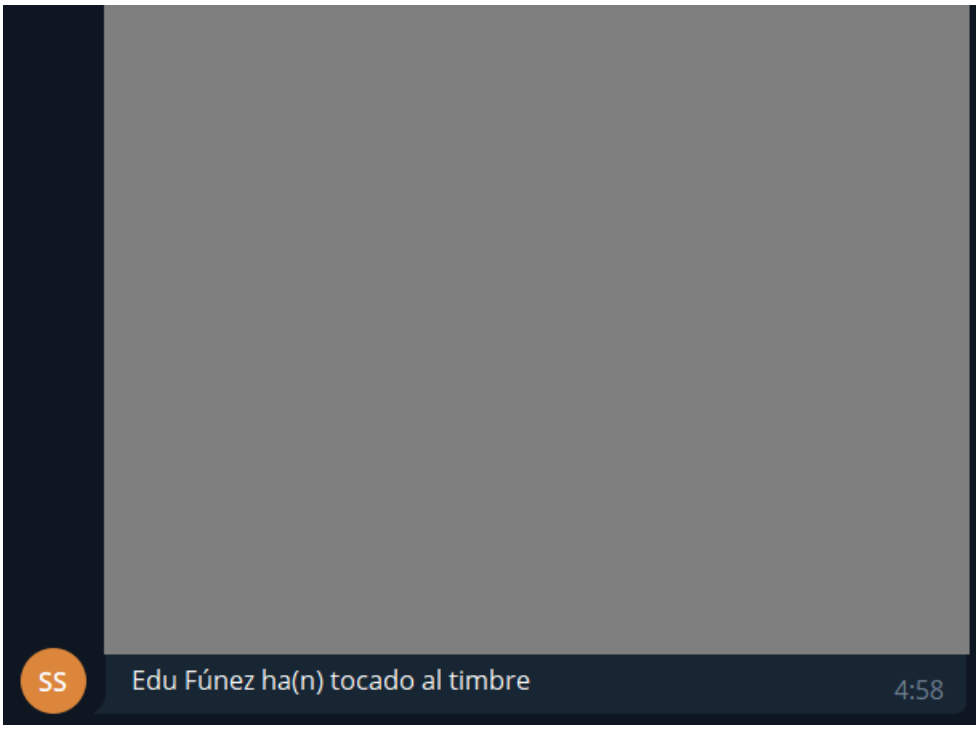


El último mensaje que se ve en la Figura 106 se irá actualizando conforme se avance en el proceso de entrenamiento. Hasta que el entrenamiento no finalice, el bot queda bloqueado sin poder procesar ningún comando entrante. Cuando el entrenamiento finaliza se muestra el siguiente mensaje:



II.XVII. Reconocimiento facial

El reconocimiento facial se activa cuando alguien acciona el pulsador a la entrada de la vivienda. Entonces, se toma una fotografía mediante la cámara de la entrada y se realiza un reconocimiento facial con la IA que se ha entrenado previamente con el comando */train*:

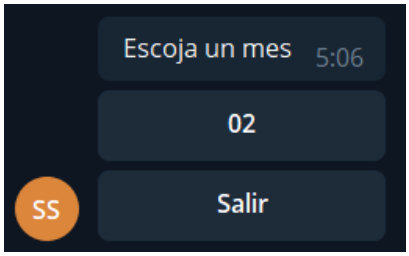


Sea cual sea el resultado del reconocimiento facial, ya sea nulo (no se detecta ninguna cara), positivo (se detecta una cara conocida) o negativo (se detecta una desconocida), se envía un aviso a todos los usuarios registrados con la fotografía captada y el resultado del reconocimiento facial. Si el resultado del reconocimiento facial es positivo, se desbloquea la cerradura durante unos segundos, tras los que se vuelve a bloquear.

II.XVIII. Comando */log*

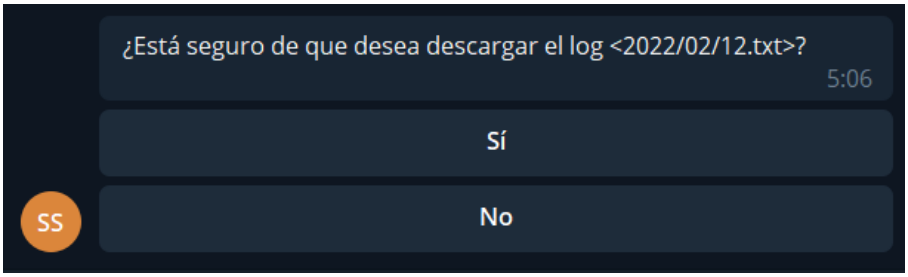
Este comando le permite al usuario acceder a los registros del sistema de seguridad. Estos registros recogen diariamente todos los eventos que suceden en el sistema de seguridad en archivos de texto. Cuando se envía el comando */log*, se muestra el siguiente menú:

Tras escoger un año se muestra otro menú:

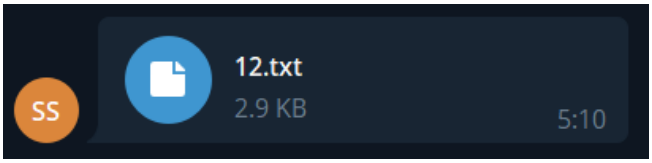


Cuando se escoge el mes, se muestra el siguiente menú:

Cuando se escoge una de las opciones, se muestra el siguiente menú de confirmación:



Si se escoge la opción “Sí”, se envía al usuario el archivo de texto correspondiente al registro del día seleccionado:

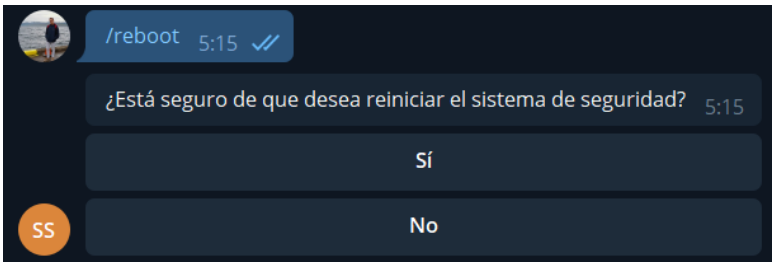


Dentro del registro aparecen todos los eventos registrados con su correspondiente marca de tiempo:

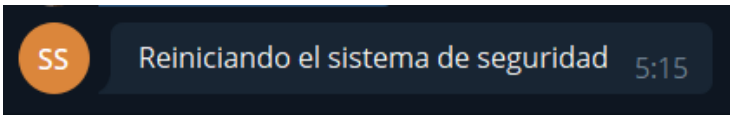
```
[01:09:26] Usuario <Edu Fúnez> eliminado
[01:09:39] Usuario añadido: Edu Fúnez
[01:11:10] Usuario añadido como admin: Edu Fúnez
[01:11:16] Usuario <Edu Fúnez> eliminado
[01:18:00] Usuario añadido: Edu Fúnez
[01:18:15] Usuario añadido como admin: Edu Fúnez
[01:18:19] Contraseña guardada: abra
[01:22:16] Contraseña guardada: abracadabra
[01:25:46] Usuario <Edu Fúnez> eliminado
[01:35:13] Usuario añadido: Edu Fúnez
[01:35:22] Usuario añadido como admin: Edu Fúnez
[01:42:48] Contraseña para administradores guardada:adm
[01:59:02] Usuario <Edu Fúnez> eliminado
[02:07:42] Usuario añadido: Edu Fúnez
[02:07:50] Usuario añadido como admin: Edu Fúnez
[02:07:58] Contraseña para administradores guardada:admin
[02:31:08] Cerradura bloqueada
[02:46:01] Simulación de presencia en frecuencia alta activada
[02:53:09] Simulación de presencia desactivada
[03:00:49] Simulación de presencia manual activada
[03:01:34] Simulación de presencia desactivada
[03:28:21] Detección de movimiento mediante la cámara de seguridad activada
[03:30:33] Intrusion captada [2022-02-12 03:30:31]
[03:33:02] Detección de movimiento mediante la cámara de seguridad desactivada
[03:57:51] Sensores de movimiento PIR encendidos
[03:57:52] Movimiento detectado en Sensor 2
[03:57:57] Sensores de movimiento PIR apagados
[04:06:19] Imagen guardada en <Edu Fúnez> como '2022-02-12 04:06:19.jpg'
[04:06:28] Imagen guardada en <Edu Fúnez> como '2022-02-12 04:06:28.jpg'
[04:06:40] Imagen guardada en <Edu Fúnez> como '2022-02-12 04:06:40.jpg'
[04:07:59] Imagen guardada en <Edu Fúnez> como '2022-02-12 04:07:59.jpg'
[04:41:33] Entrenamiento de la IA de reconocimiento facial iniciado
```

II.XIX. Comando */reboot*

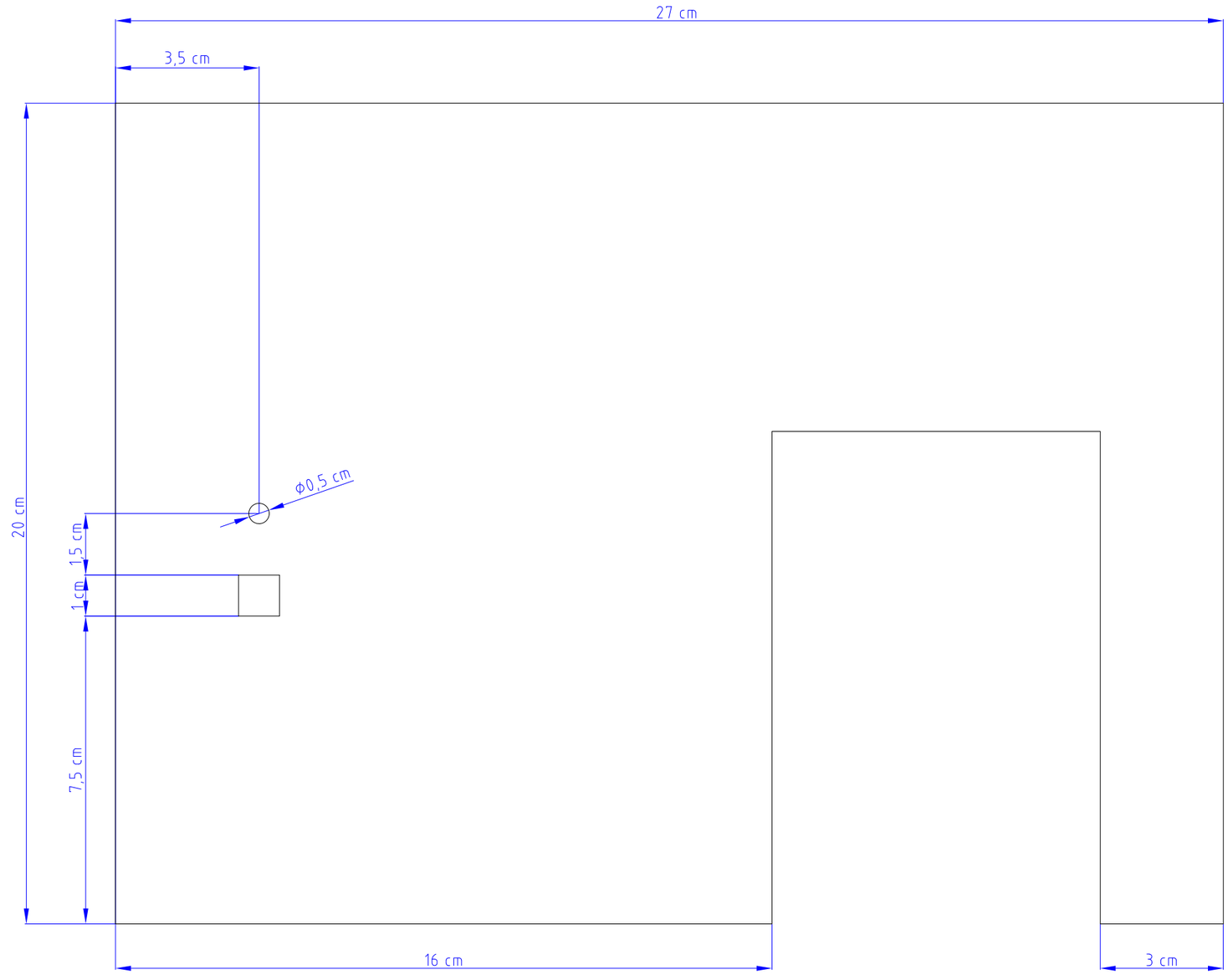
Este comando tiene la funcionalidad de reiniciar el sistema de seguridad. Cuando el usuario envía el comando */reboot*, se muestra el siguiente menú de confirmación:



Si se selecciona la opción “Sí”, se envía el siguiente mensaje, tras lo que se reinicia el sistema:

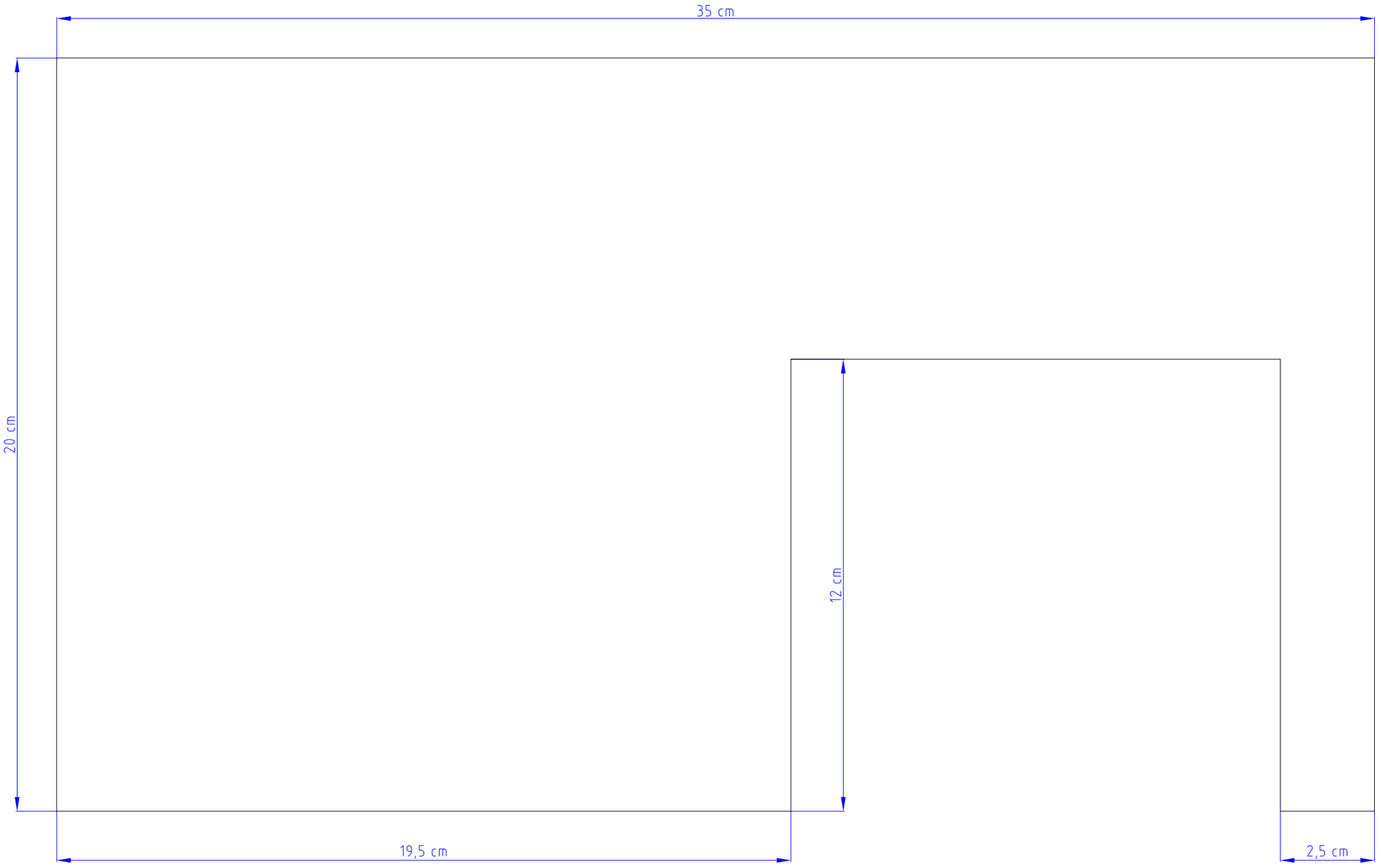


Anexo III: Planos



	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Pared 1	Nº PLANO 1 de 31 SUSTITUYE A: SUSTITUIDO POR:

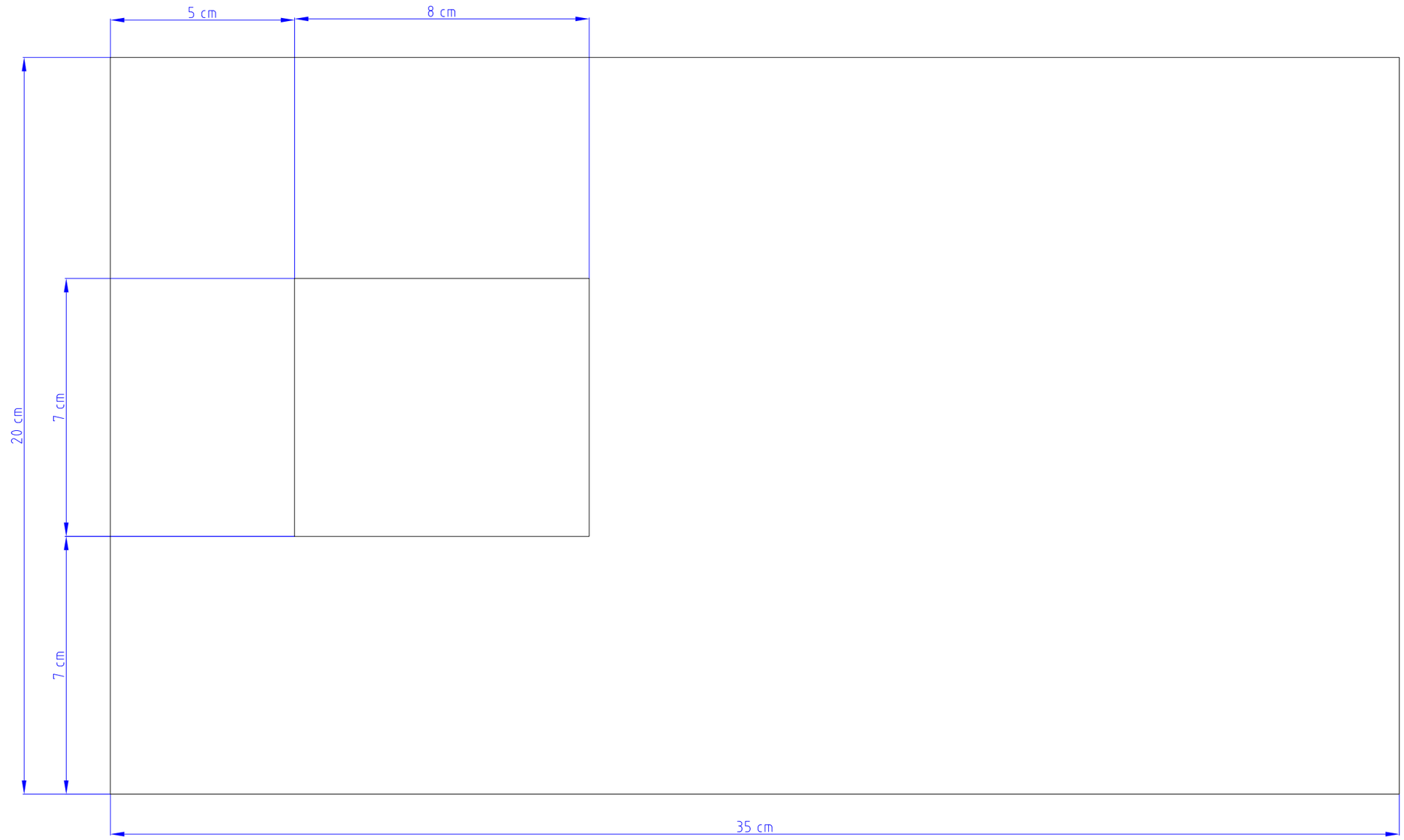


	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Pared 2	Nº PLANO 2 de 31 SUSTITUYE A: SUSTITUIDO POR:

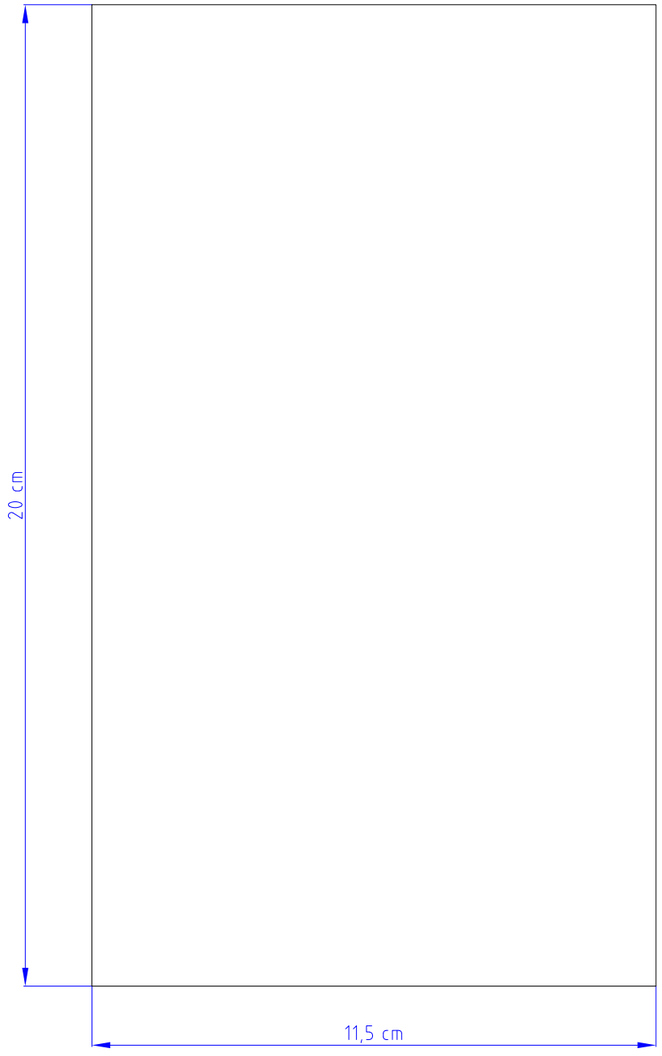
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Pared 3	Nº PLANO 3 de 31 SUSTITUYE A: SUSTITUIDO POR:



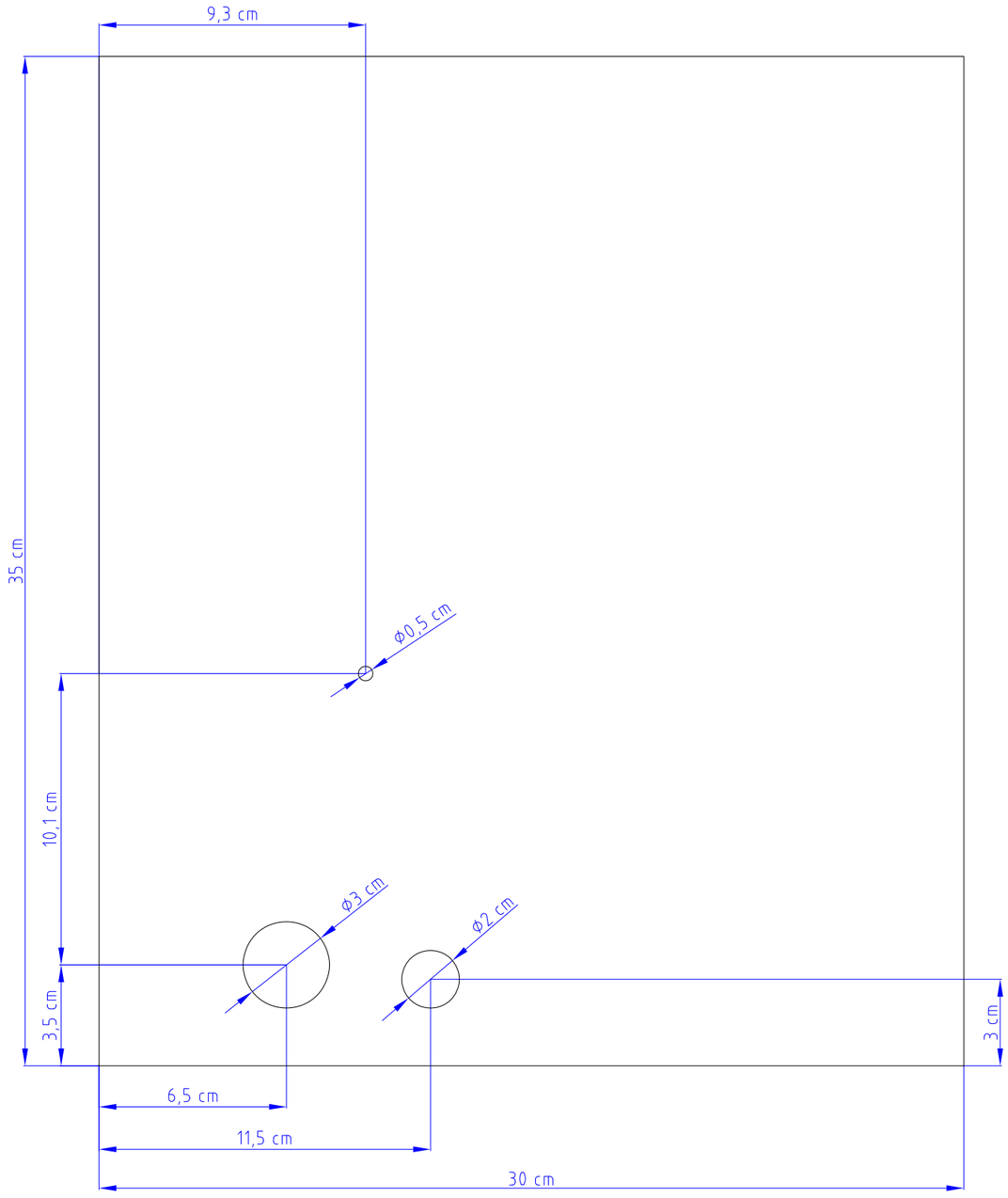
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Pared 4	Nº PLANO 4 de 31 SUSTITUYE A: SUSTITUIDO POR:



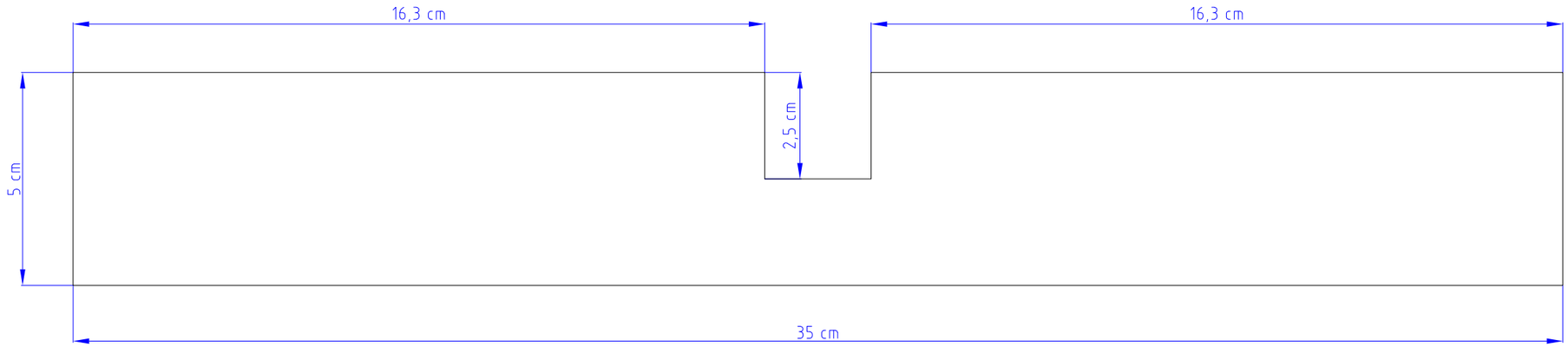
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Pared Interior	Nº PLANO 5 de 31 SUSTITUYE A: SUSTITUIDO POR:



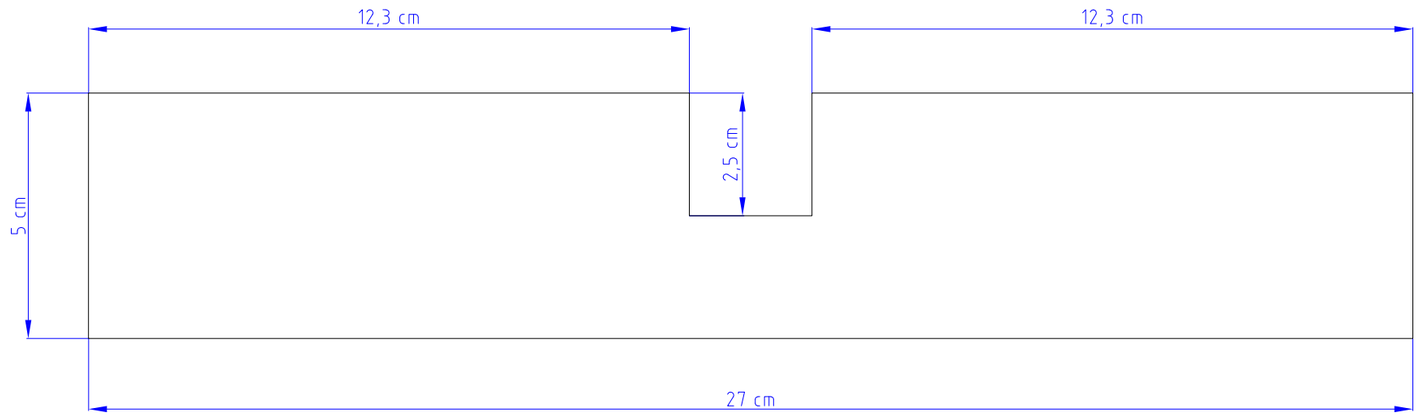
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 5:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Suelo	Nº PLANO 6 de 31 SUSTITUYE A: SUSTITUIDO POR:



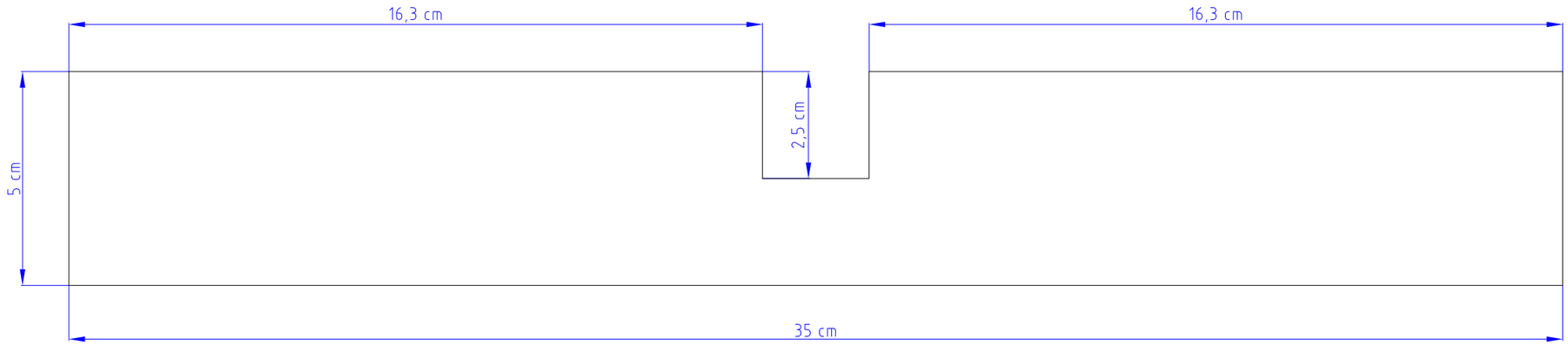
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Subpared 1	Nº PLANO 7 de 31
		SUSTITUYE A:
		SUSTITUIDO POR:



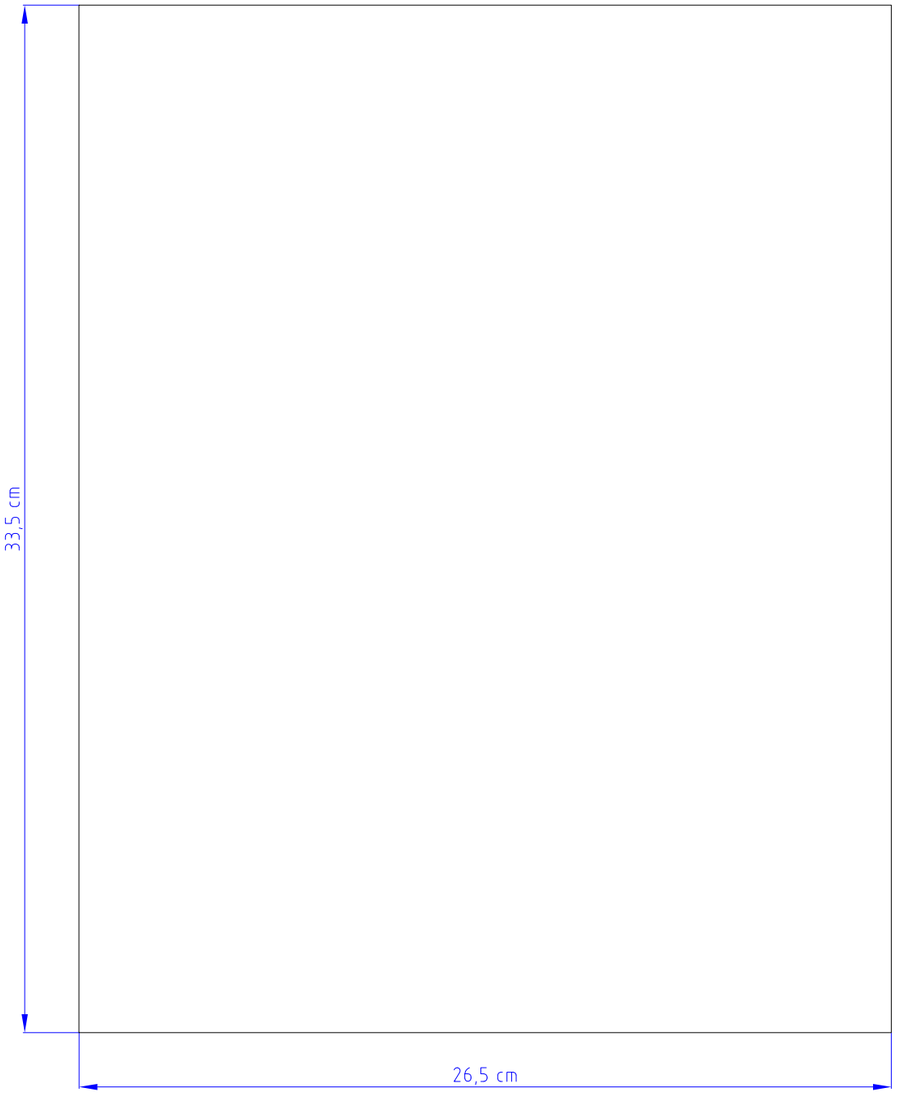
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Subpared 2	Nº PLANO 8 de 31 SUSTITUYE A: SUSTITUIDO POR:



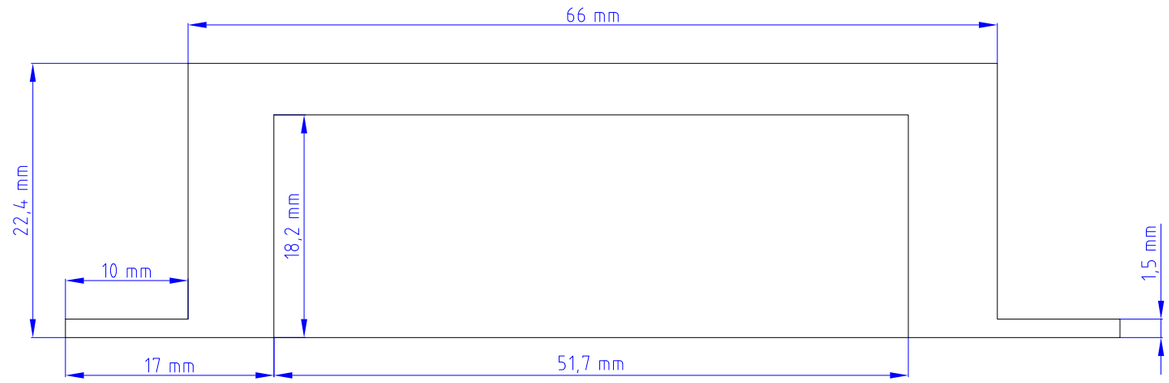
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 8:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Subpared 3	Nº PLANO 9 de 31 SUSTITUYE A: SUSTITUIDO POR:



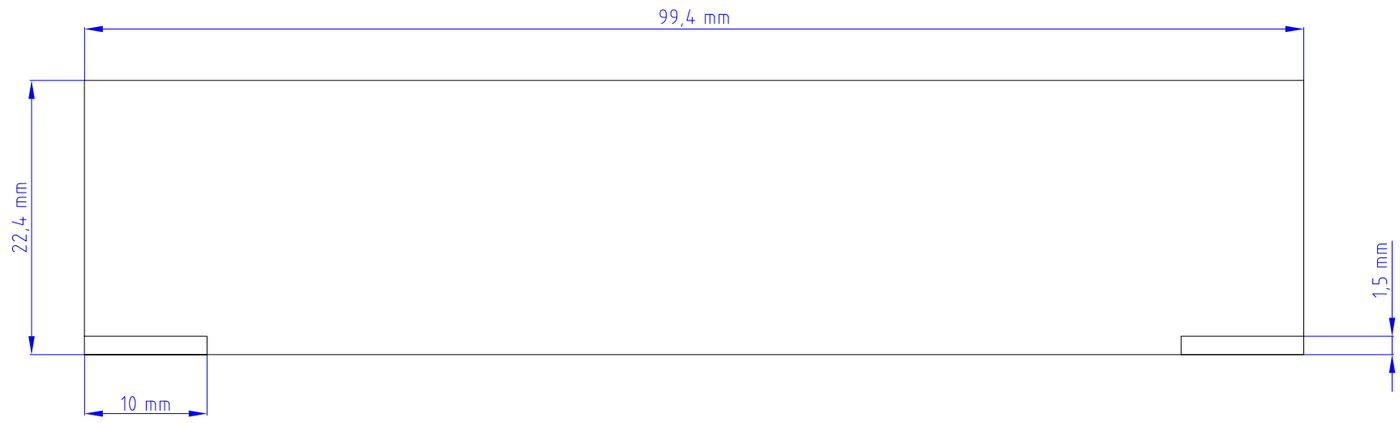
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 5:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Subsuelo	Nº PLANO 10 de 31 SUSTITUYE A: SUSTITUIDO POR:



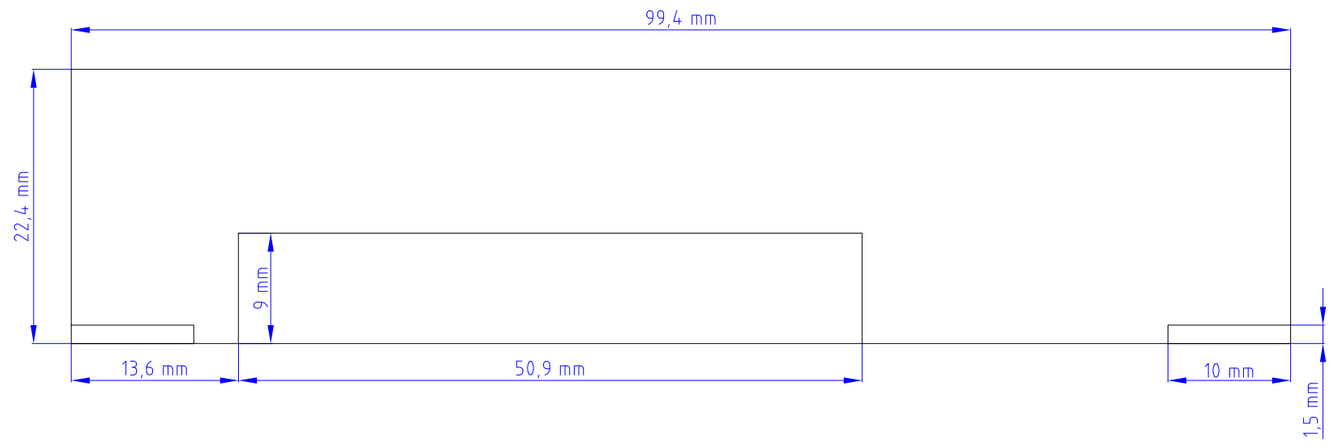
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín RPi: Alzado	Nº PLANO 11 de 31 SUSTITUYE A: SUSTITUIDO POR:



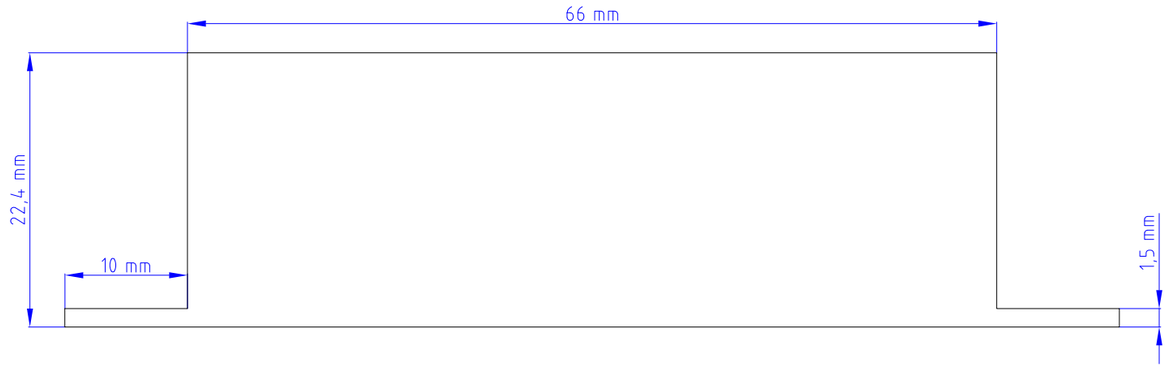
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín RPi: Perfil Derecho	Nº PLANO 12 de 31 SUSTITUYE A: SUSTITUIDO POR:



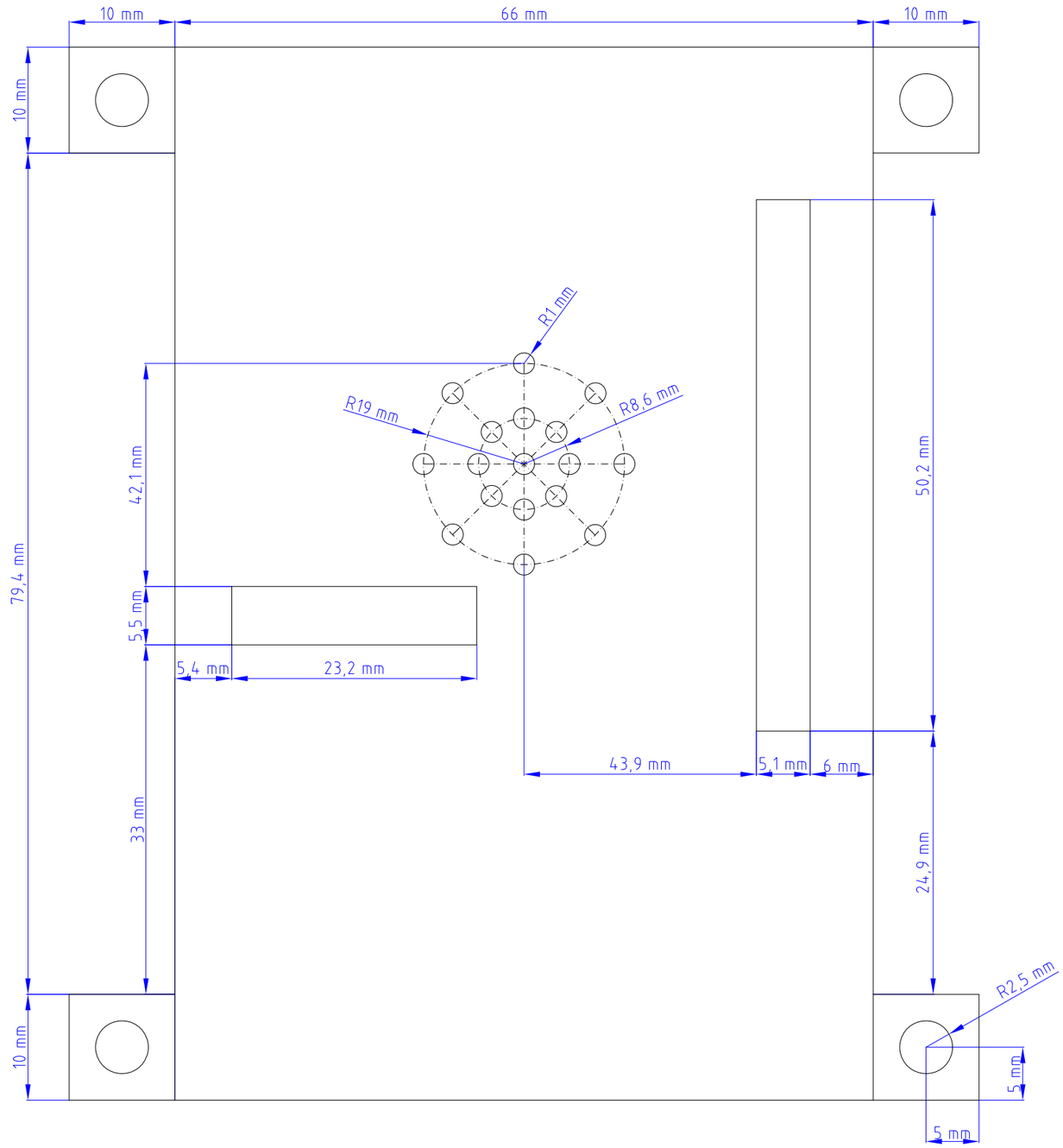
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín RPi: Perfil Izquierdo	Nº PLANO 13 de 31 SUSTITUYE A: SUSTITUIDO POR:



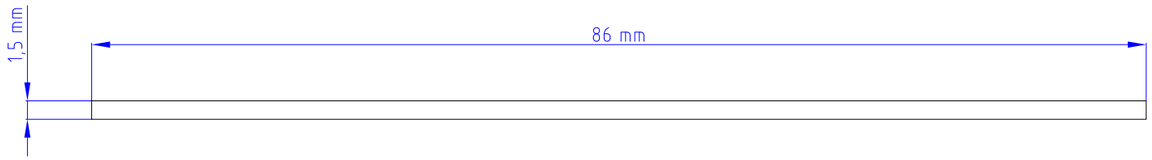
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín RPi: Vista Posterior	Nº PLANO 14 de 31 SUSTITUYE A: SUSTITUIDO POR:



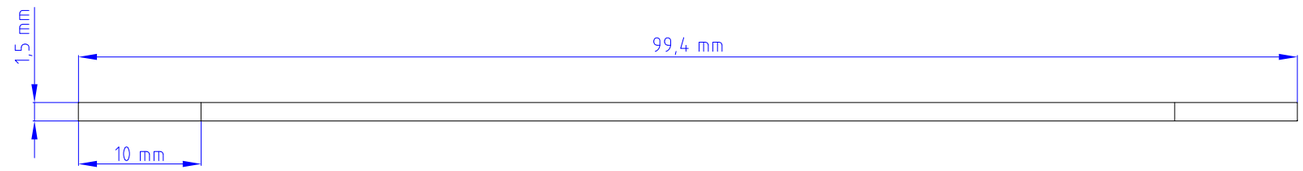
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín RPi: Planta	Nº PLANO 15 de 31 SUSTITUYE A: SUSTITUIDO POR:



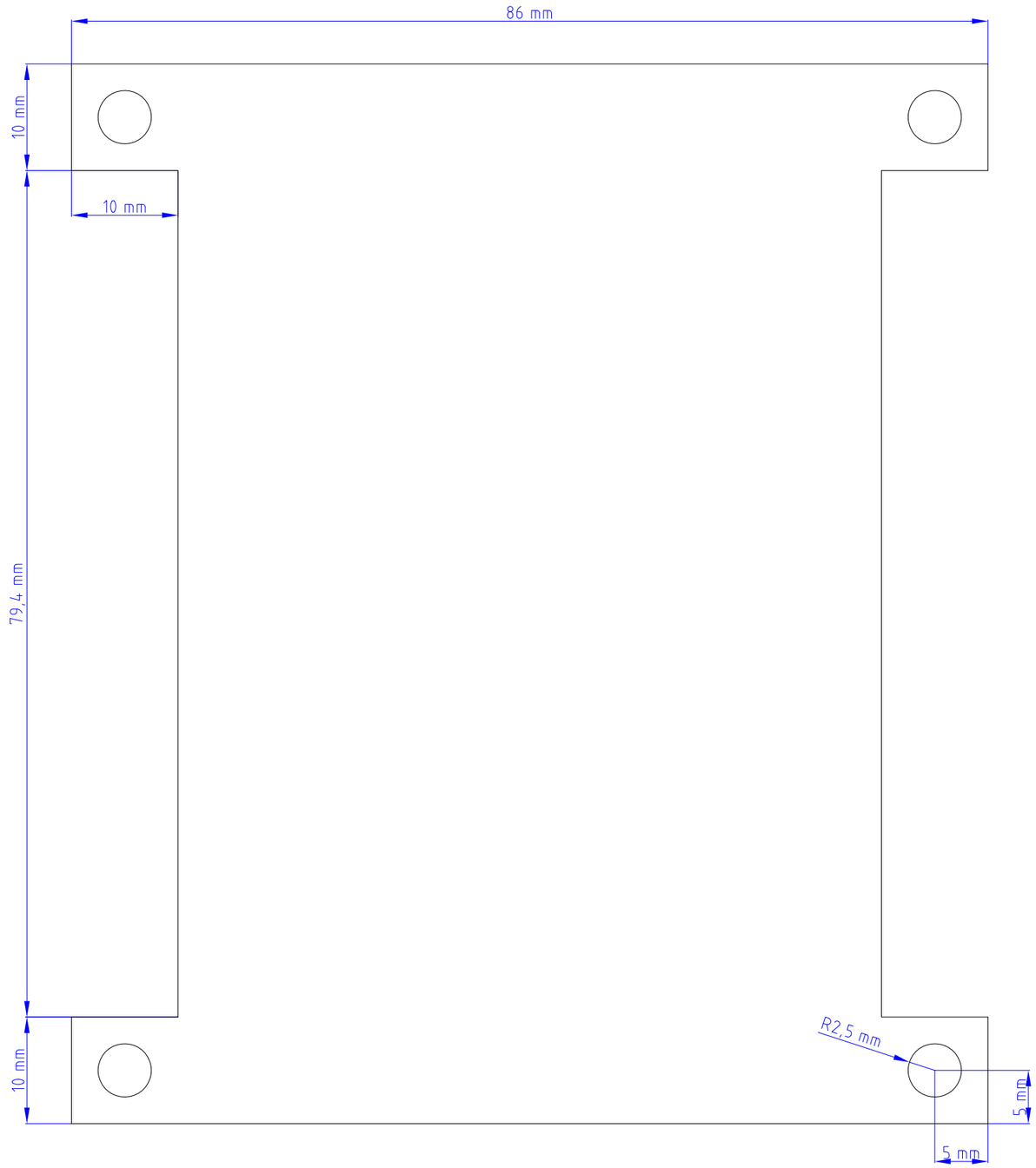
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín RPi: Alzado	Nº PLANO 16 de 31 SUSTITUYE A: SUSTITUIDO POR:



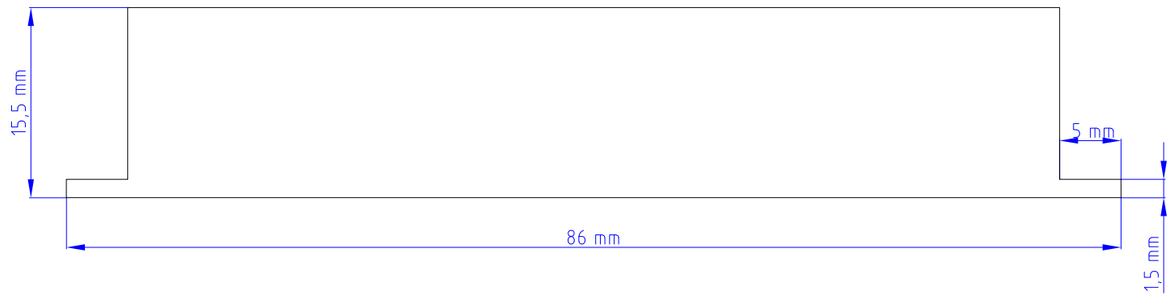
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín RPi: Perfil	Nº PLANO 17 de 31 SUSTITUYE A: SUSTITUIDO POR:



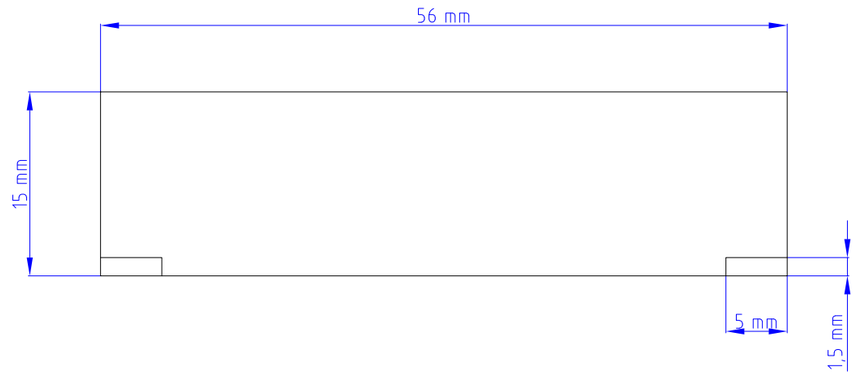
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín RPi: Planta	Nº PLANO 18 de 31 SUSTITUYE A: SUSTITUIDO POR:



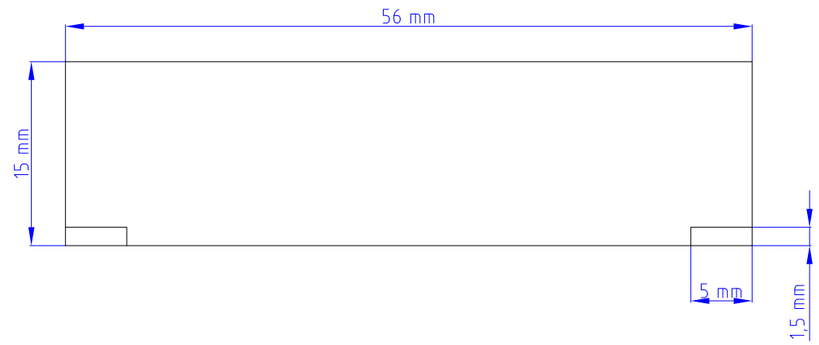
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín Relay: Alzado	Nº PLANO 19 de 31 SUSTITUYE A: SUSTITUIDO POR:



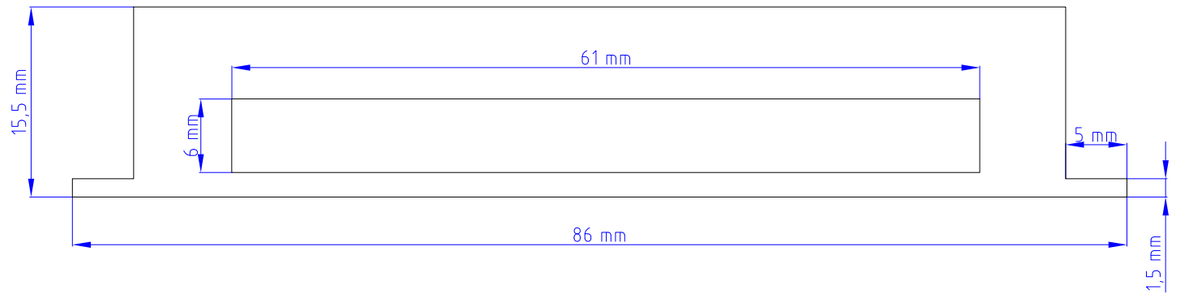
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín Relay: Perfil Derecho	Nº PLANO 20 de 31 SUSTITUYE A: SUSTITUIDO POR:



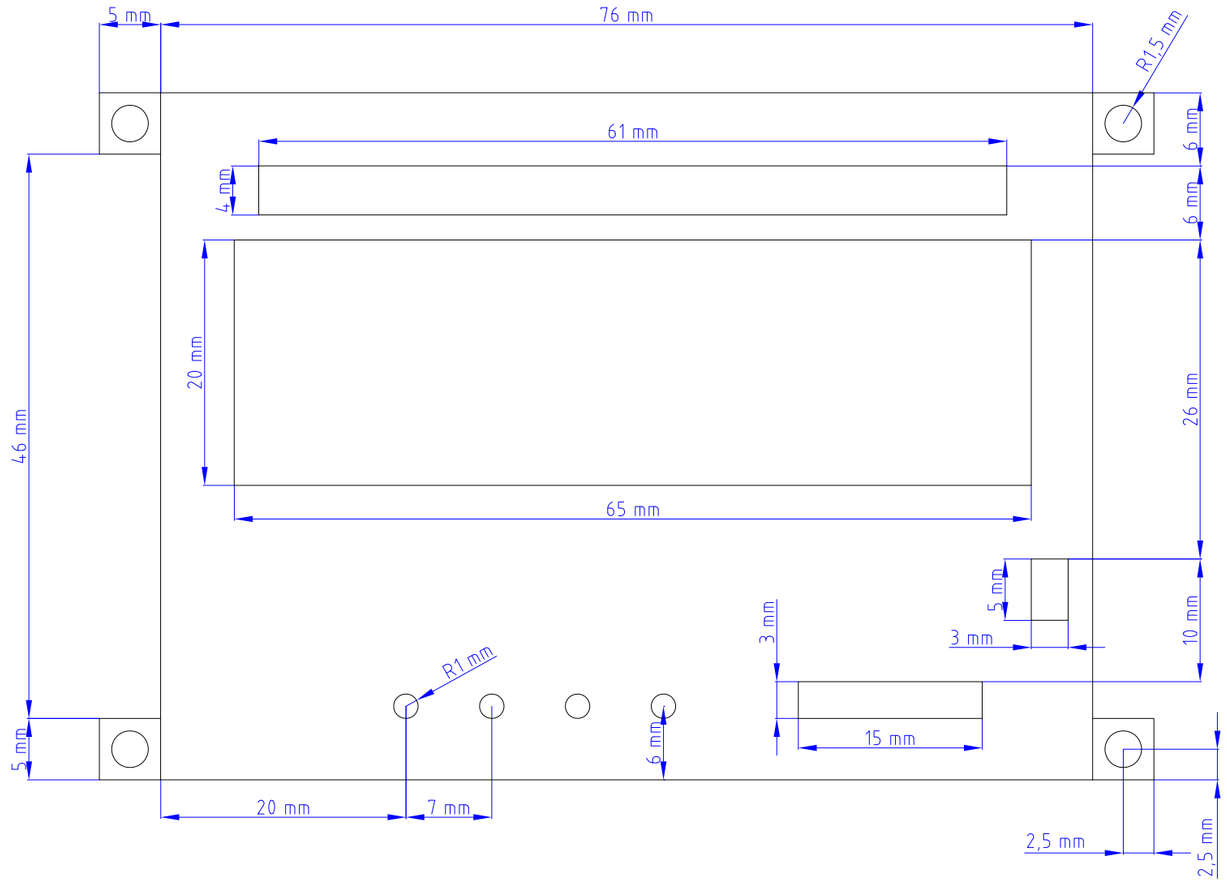
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín Relay: Perfil Izquierdo	Nº PLANO 21 de 31 SUSTITUYE A: SUSTITUIDO POR:



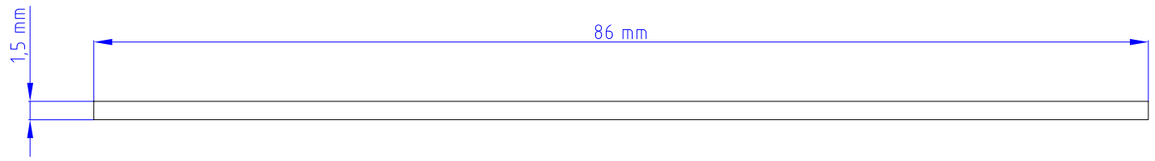
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín Relay: Vista Posterior	Nº PLANO 22 de 31 SUSTITUYE A: SUSTITUIDO POR:



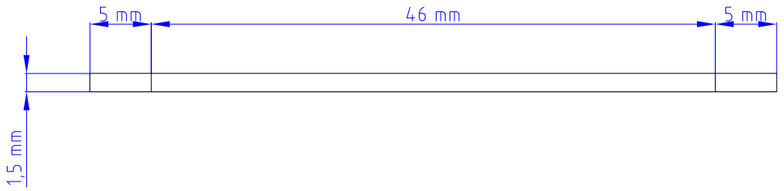
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Tapa Cajetín Relay: Planta	Nº PLANO 23 de 31 SUSTITUYE A: SUSTITUIDO POR:



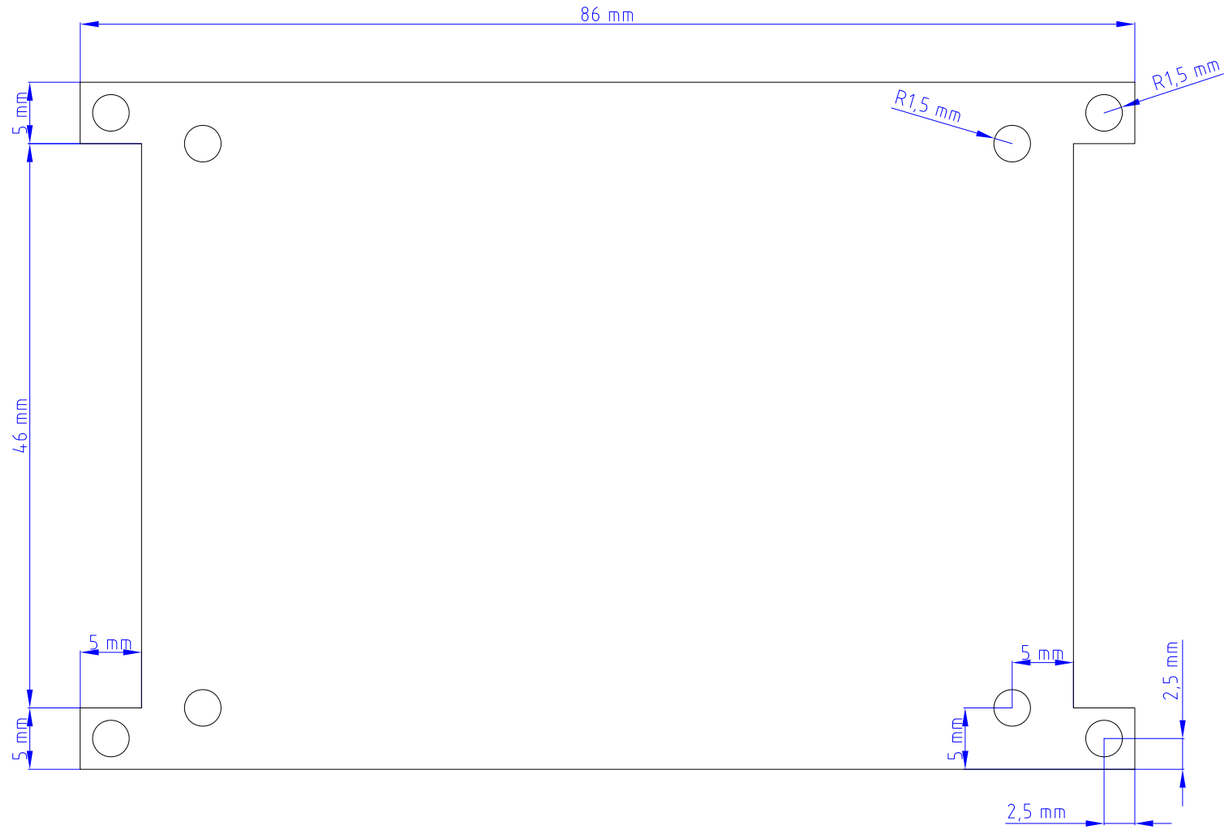
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín Relay: Alzado	Nº PLANO 24 de 31 SUSTITUYE A: SUSTITUIDO POR:



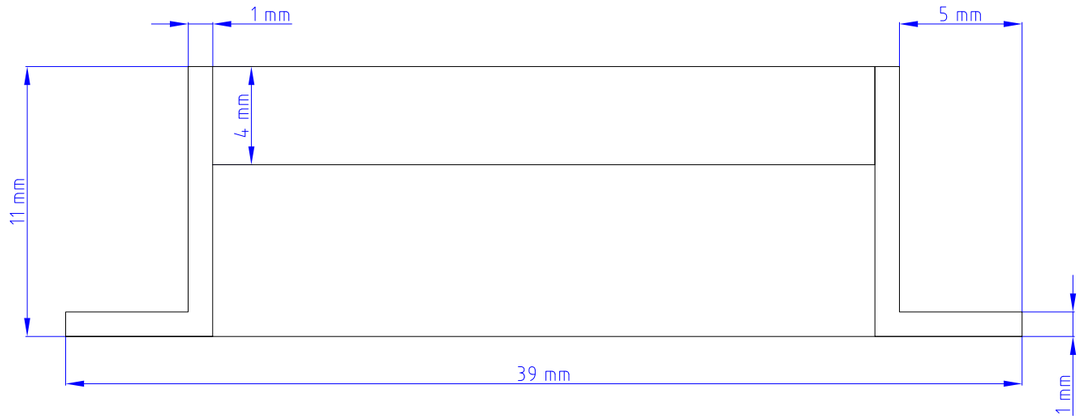
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín Relay: Perfil	Nº PLANO 25 de 31 SUSTITUYE A: SUSTITUIDO POR:



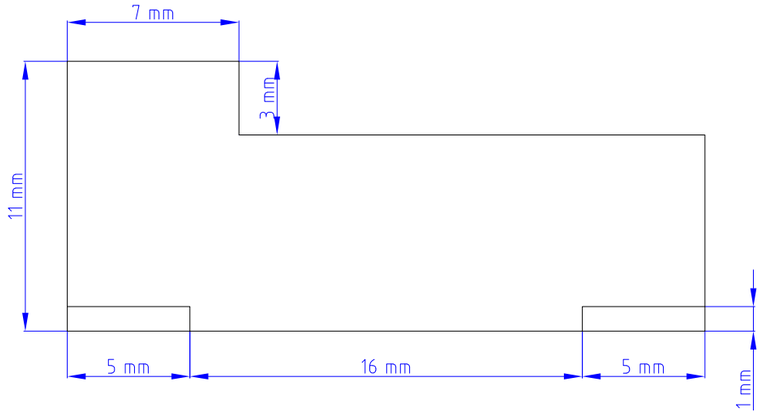
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 2:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Base Cajetín Relay: Planta	Nº PLANO 26 de 31 SUSTITUYE A: SUSTITUIDO POR:



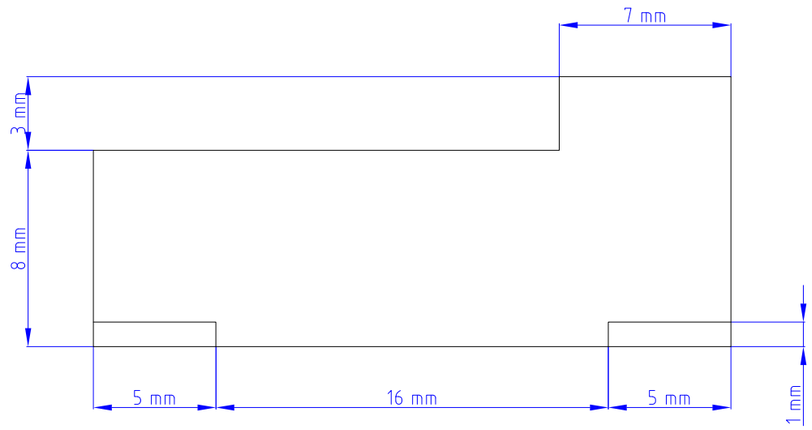
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 4:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Cajetín Módulo Cámara RPi: Alzado	Nº PLANO 27 de 31 SUSTITUYE A: SUSTITUIDO POR:



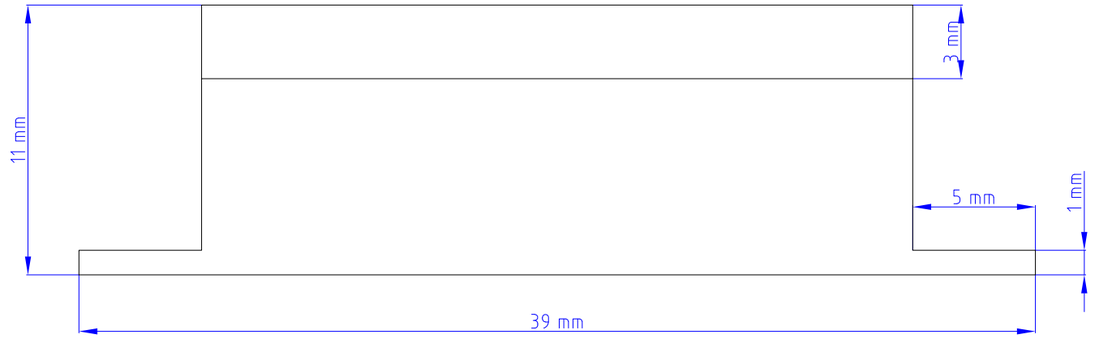
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 4:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Cajetín Módulo Cámara RPi: Perfil Derecho	Nº PLANO 28 de 31 SUSTITUYE A: SUSTITUIDO POR:



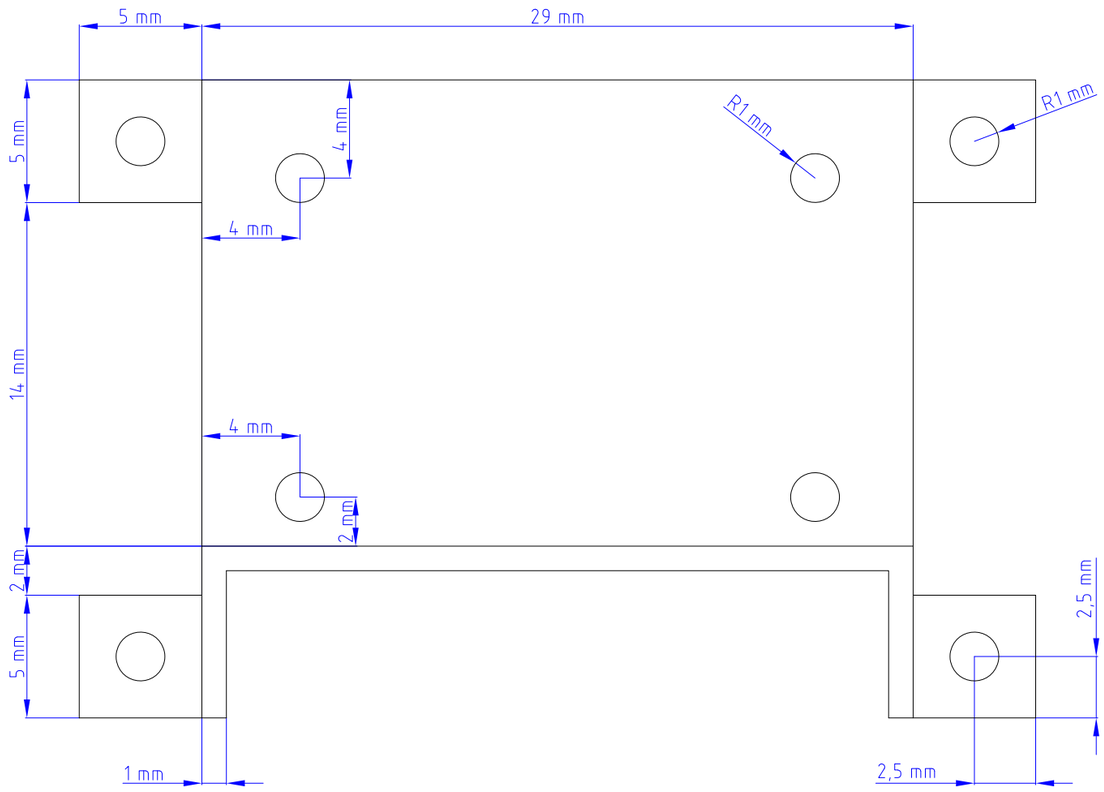
	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 4:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Cajetín Módulo Cámara RPi: Perfil Izquierdo	Nº PLANO 29 de 31 SUSTITUYE A: SUSTITUIDO POR:



	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 4:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Cajetín Módulo Cámara RPi: Vista Posterior	Nº PLANO 30 de 31 SUSTITUYE A: SUSTITUIDO POR:



	FECHA	NOMBRE	FIRMA	ESCUELA POLITÉCNICA SUPERIOR LINARES
DIBUJADO	07/02/2022	Eduardo Fúnez Fernández		
COMPROBADO				

ESCALA: 4:1	Diseño de un sistema de seguridad en el hogar basado en IoT y creación de prototipo Cajetín Módulo Cámara RPi: Planta	Nº PLANO 31 de 31 SUSTITUYE A: SUSTITUIDO POR: