

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Universidad
Politécnica
de Cartagena

MIEMBRO DE



EUROPEAN
UNIVERSITY OF
TECHNOLOGY

**Trabajo Fin de Grado
GRADO EN INGENIERÍA TELEMÁTICA**

**Sistema de Bajo Coste
para Contabilizar el
Aforo en una Biblioteca**



AUTOR: Bruno Moral Delgado

DIRECTOR: Juan Carlos Aarnoutse Sánchez

Septiembre / 2024



Autor	Bruno Moral Delgado
E-mail del autor	brunobajista6@gmail.com
Director	Juan Carlos Aarnoutse Sánchez / juanc.sanchez@upct.es
Título de TFG	Sistema de Bajo Coste para Contabilizar el Aforo en una Biblioteca
Resumen	<p>Este trabajo fin de grado nace a raíz de la necesidad de tener una solución automatizada de conteo y monitorización del aforo en tiempo real de la biblioteca de Antigones y disponer de informes del histórico de ocupación de la misma. En este proyecto se ha optado por una solución de bajo coste implementando sensores de proximidad infrarrojos que se coloca en los arcos de entrada de la biblioteca.</p> <p>Para realizar dicha tarea, estos sensores se conectan a unas placas de desarrollo ESP32(S3) que se han programado en el entorno de Arduino. El algoritmo creado para los ESP32 recogerá los datos de entrada o salida de personas y, posteriormente, enviará un mensaje por Wi-Fi a un servidor. Este servidor (creado en Python) procesa los mensajes en función de su tipo de mensaje. Por último, el servidor contiene una base de datos en la que se almacenan los diferentes datos correspondientes que, a su vez, se pueden visualizar en el front-end de Grafana. También crea informes del histórico de aforo en formato CSV para su posterior procesamiento.</p>
Titulación	Grado en Ingeniería Telemática
Departamento	Tecnología de la Información y las Comunicaciones
Fecha de presentación	Septiembre - 2024

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor Juan Carlos Aarnoutse Sánchez por darme la oportunidad de realizar este proyecto. Su amplia experiencia, sus valiosas sugerencias, el apoyo que me ha brindado durante todos estos meses de desarrollo y su implicación han sido imprescindibles para que este trabajo pudiera salir adelante.

También me gustaría agradecer su paciencia y generosidad al personal de la biblioteca de Antigones durante la realización de las pruebas.

Y, por último, pero no menos importante, me gustaría agradecer a mi familia todo el apoyo que me han dado desde siempre. Teniendo unos padres que no pudieron tener estudios superiores por carencias económicas, ellos siempre desearon que su hijo si los tuviera y se han esforzado siempre porque hoy esté redactando esta memoria de trabajo final de grado.

Contenido

Agradecimientos	3
Contenido	4
Índice de figuras	6
Capítulo 1. Introducción y objetivos	9
1.1 Introducción y objetivos.....	9
1.2 Estructura del sistema.....	10
1.3 Metodología de trabajo.....	12
1.4 Cronograma del proyecto	13
1.5 Organización del proyecto.....	14
Capítulo 2. Tecnologías usadas	15
2.1 Sensores de proximidad infrarrojos E18-D80NK	15
2.2 Placas de desarrollo ESP32	16
2.3 IDE Arduino	17
2.4 Python y Pycharm	18
2.5 Base de datos SQLite	19
2.6 Grafana.....	20
2.7 Archivo CSV	20
Capítulo 3. Configuración de los sensores y programación de las placas de desarrollo ESP32 .	21
3.1 Funcionamiento de los sensores	21
3.2 Configuración de los sensores.....	23
3.3 Cableado de los sensores.....	24
3.4 Encapsulado de los sensores y las placas ESP32	27
3.5 Programación de las placas de desarrollo ESP32.....	28
3.5.1 Algoritmo de conteo	28
3.5.2 Conexión de los ESP32 con el servidor	33
3.5.3 Configuración LED RGB del ESP32-S3.....	36
3.6 Problemas surgidos	37
Capítulo 4. Red y servidor Python	40
4.1 Router y configuración de la red Wi-Fi.....	40
4.2 Servidor Python.....	43
4.2.1 Creación de la base de datos.....	44
4.2.2 Inicialización del servidor	45
4.2.3 Aceptación de las conexiones.....	46
4.2.4 Recepción de mensajes.....	47
4.2.5 Acciones a realizar tras la recepción	49

4.2.6 Monitorización de los clientes.....	51
4.2.7 Actualizaciones de la base de datos.....	53
4.3 Problemas surgidos	54
Capítulo 5. Presentación de los datos.....	56
5.1 Plugin SQLite para Grafana y fuente de datos	56
5.2 Paneles de Grafana	58
5.3 Archivos CSV.....	61
Capítulo 6. Pruebas y resultados	62
6.1 Pruebas preliminares	62
6.2 Pruebas de testeo y validación del sistema.....	62
6.3 Análisis de los resultados	66
Capítulo 7. Conclusiones y trabajos futuros.....	67
7.1 Conclusiones	67
7.2 Grado de consecución de los objetivos y formación adquirida.....	67
7.3 Trabajos futuros.....	69
Bibliografía	70
Anexos.....	71
Datasheet del sensor E18-D80NK.....	71
Datasheet SoC ESP32-S3.....	72

Índice de figuras

Figura 1. Plano de planta de la biblioteca de Antigones indicando la entrada	9
Figura 2. Entrada de la biblioteca con los arcos antihurto.	10
Figura 3. Esquema de funcionamiento del sistema	11
Figura 4. Comparativa del método científico y el método ingeniero	12
Figura 5. Cronograma del proyecto.....	13
Figura 6. Sensor E18-D80NK	15
Figura 7. ESP32-DevKitC.....	16
Figura 8. ESP32-S3-DevKitC-1	17
Figura 9. Logo IDE Arduino	17
Figura 10. Librerías ESP32	18
Figura 11. Librería Adafruit Neopixel.....	18
Figura 12. Logo de Python.....	18
Figura 13. Logo de PyCharm	19
Figura 14. Logo SQLite	19
Figura 15. Logo de Grafana	20
Figura 16. Archivo CSV.....	20
Figura 17. Circuito con BJT de colector abierto	21
Figura 18 Circuito Sensor	22
Figura 19. Diagrama transmisor y receptor del sensor E18-D80NK	22
Figura 20. Imagen en detalle de la parte posterior del sensor E18-D80NK.....	23
Figura 21. Detalle de los cables del sensor	24
Figura 22. Pin-layout del ESP32-S3-DevKitC-1	25
Figura 23. Pin-layout del ESP32-DevKitC	25
Figura 24. Cableado de los sensores a la placa ESP32-S3-DevKitC-1	26
Figura 25. Esquema del cableado de los sensores con la placa ESP32-S3-DevKitC-1	26
Figura 26. Parte frontal de la caja con sus dos sensores.....	27
Figura 27. Localización de la caja en los arcos.....	28
Figura 28. Distancia entre los sensores	29
Figura 29. Máquina de estados finita	29
Figura 30. Máquina de estados finita completa.....	30
Figura 31. Esquema del algoritmo de conteo	32
Figura 32. Algoritmo de conexión de los ESP32 al servidor	33
Figura 33. Esquema del envío de datos de los ESP32 al servidor.....	35
Figura 34. Esquema de la configuración del LED RGB	36

Figura 35. Vista desde arriba de la segunda estructura de pruebas	37
Figura 36. Vista frontal de la segunda estructura de pruebas	38
Figura 37. Captura de los drivers del puente USB a UART de los ESP32	38
Figura 38. Cables usados en primera instancia	39
Figura 39. Router Linksys WRT54LG	40
Figura 40. SSID y canal de la red inalámbrica.....	41
Figura 41. Seguridad de la red inalámbrica.....	41
Figura 42. Configuración IP del router.....	42
Figura 43. Asociación de la MAC del servidor a la primera IP del rango.....	42
Figura 44. Diagrama de bloques del servidor.....	43
Figura 45. Captura de la BB.DD.	44
Figura 46. Inicialización del servidor.....	45
Figura 47. Captura del hilo principal del servidor.....	46
Figura 48. Establecimiento de la conexión TCP de los clientes con el servidor	46
Figura 49. Formato de los mensajes (campos)	47
Figura 50. Formato de dos mensajes juntos	47
Figura 51. Gráfico relacional entre el nº de campos y el nº de mensajes	47
Figura 52. Ecuación que relaciona el nº de campos y el nº de mensajes.....	48
Figura 53. Código correspondiente a la recepción de mensajes.....	48
Figura 54. Detalle captura Wireshark presentación.....	49
Figura 55. Detalle captura Wireshark incrementa	49
Figura 56. Detalle captura Wireshark decrementa	49
Figura 57. Detalle captura Wireshark nada	49
Figura 58. Diagrama de bloques de las acciones a realizar una vez recibido un mensaje	50
Figura 59. Captura del hilo de acciones realizar tras la recepción	51
Figura 60. Diagrama de bloques del monitor de conexiones	52
Figura 61. Implementación del monitor de conexiones	52
Figura 62. Implementación para actualizar la tabla aforo	53
Figura 63. Implementación para actualizar la tabla conexiones.....	53
Figura 64. Versión prematura del código de recepción de los mensajes	54
Figura 65. Error registrado en el archivo aforo_log.txt.....	55
Figura 66. Lugar en el que se encontraba el error provocado por \n.....	55
Figura 67. Captura del plugin SQLite	56
Figura 68. Fuente de datos de Grafana.....	57
Figura 69. Consulta SQL gráfico aforo	58
Figura 70. Consulta SQL tabla conexiones	58

Figura 71. Consulta SQL panel aforo.....	59
Figura 72. Consulta SQL panel estado dispositivo 1	59
Figura 73. Consulta SQL panel estado dispositivo 2	59
Figura 74. Vista de todos los paneles de Grafana.....	60
Figura 75. Captura del archivo CSV	61
Figura 76. Lugar en el que se encuentran los sensores (por delante)	63
Figura 77. Lugar en el que se encuentran los sensores (por detrás)	63
Figura 78. Imagen del PC donde se ha ejecutado nuestro servidor.....	64
Figura 79. Tabla de datos recogidos	65
Figura 80. Tabla de porcentajes de error	66
Figura 81. Ecuación del porcentaje de error de entradas	66
Figura 82. Ecuación del porcentaje de error de salidas	66
Figura 83. Ecuación del error total.....	66
Figura 84. Especificaciones sensor E18-D80NK	71
Figura 85. Esquema de conexión del sensor E18-D80NK	71
Figura 86. Funcionamiento del sensor E18-D80NK	72
Figura 87. Diagrama SoC ESP32-S3	72

Capítulo 1. Introducción y objetivos

1.1 Introducción y objetivos

Indudablemente, conocer la cantidad de personas que entran y salen, así como el aforo en tiempo real, es una prioridad para cualquier biblioteca pública. Este proyecto se suma a otros proyectos realizados en esta línea para contabilizar el aforo de las bibliotecas de la UPCT. En este TFG, solamente se ha probado en la biblioteca de Antigones, pero sería realmente sencillo escalar este proyecto para contabilizar el aforo en las bibliotecas de Alfonso XIII y CIM, tanto individualmente, como en conjunto de todas.

Tradicionalmente, hasta hace no mucho, el aforo de cada biblioteca se registraba manualmente, es decir, el personal contabilizaba el aforo de cada biblioteca en una libreta a las 12:00 y 18:00 horas. Posteriormente, los datos anotados se pasaban a un Excel compartido y se unificaban. Evidentemente, este proceso es bastante engorroso, ineficiente y no permite tener una visión de la evolución del aforo a lo largo del día, por lo que es recomendable optar por una automatización del registro de entradas, salidas, así como del aforo en tiempo real.

Para llevar a cabo dicha automatización se decidió crear un sistema de bajo coste basado en unos sensores de proximidad infrarrojos, que fuera posible colocar en los arcos antihurto de libros que se sitúan en la entrada de la puerta. Como es lógico, la biblioteca sólo cuenta con un acceso, por lo que ese lugar es el óptimo para contabilizar los flujos de entrada y salida de personas. En la siguiente figura se muestra un plano de la planta de biblioteca de Antigones.

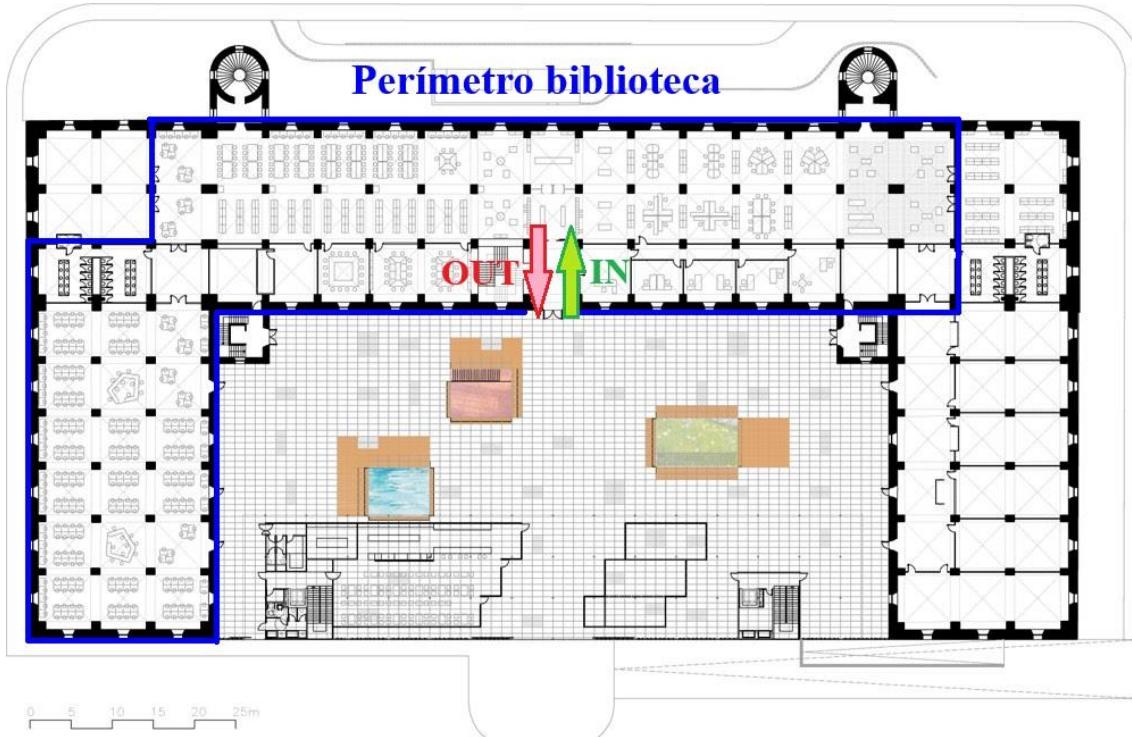


Figura 1. Plano de planta de la biblioteca de Antigones indicando la entrada

El objetivo de este Trabajo Fin de Grado (TFG) ha sido el de diseñar, implementar y validar un sistema basado en sensores de proximidad infrarrojos que sea capaz de detectar con precisión las entradas y salidas de personas en los dos arcos de la puerta de la biblioteca de Antigones (adaptable a cualquier número de arcos de entrada/salida). Además, como en todo proyecto de ingeniería, el sistema tiene que ser totalmente funcional, pero con el coste más bajo posible. Este sistema tiene que ser capaz de contabilizar en tiempo real los flujos de entrada y salida de personas, el aforo instantáneo y generar informes detallados para el servicio CRAI.



Figura 2. Entrada de la biblioteca con los arcos antihurto.

En la Figura 2 se puede observar la entrada de la biblioteca donde se van a colocar los sensores. Cabe decir que, como los sensores únicamente detectan objetos a cierta distancia, es decir, no se usan cámaras de ningún tipo, se está cumpliendo con el Reglamento General de Protección de Datos (GDPR) puesto que no se obtienen ningún tipo de datos biométricos de ninguna persona.

1.2 Estructura del sistema

La Figura 3 muestra el esquema del sistema completo y su interacción que se explicará brevemente en esta subsección. En primer lugar, los sensores de proximidad infrarrojos recogen los datos del flujo de personas. Estos sensores están conectados a los pines de las placas de desarrollo ESP32, de manera que el microcontrolador de estas placas ejecuta el código con el algoritmo que se ha creado para contabilizar personas. Una vez los datos son procesados, son enviados al servidor a través de la red Wi-Fi. El servidor recibe los datos y hace lo que corresponda con ellos, para más tarde guardarlos en una base de datos, y mostrarlos gráficamente en el servicio de Grafana donde por último se crean informes periódicos del aforo en la biblioteca.

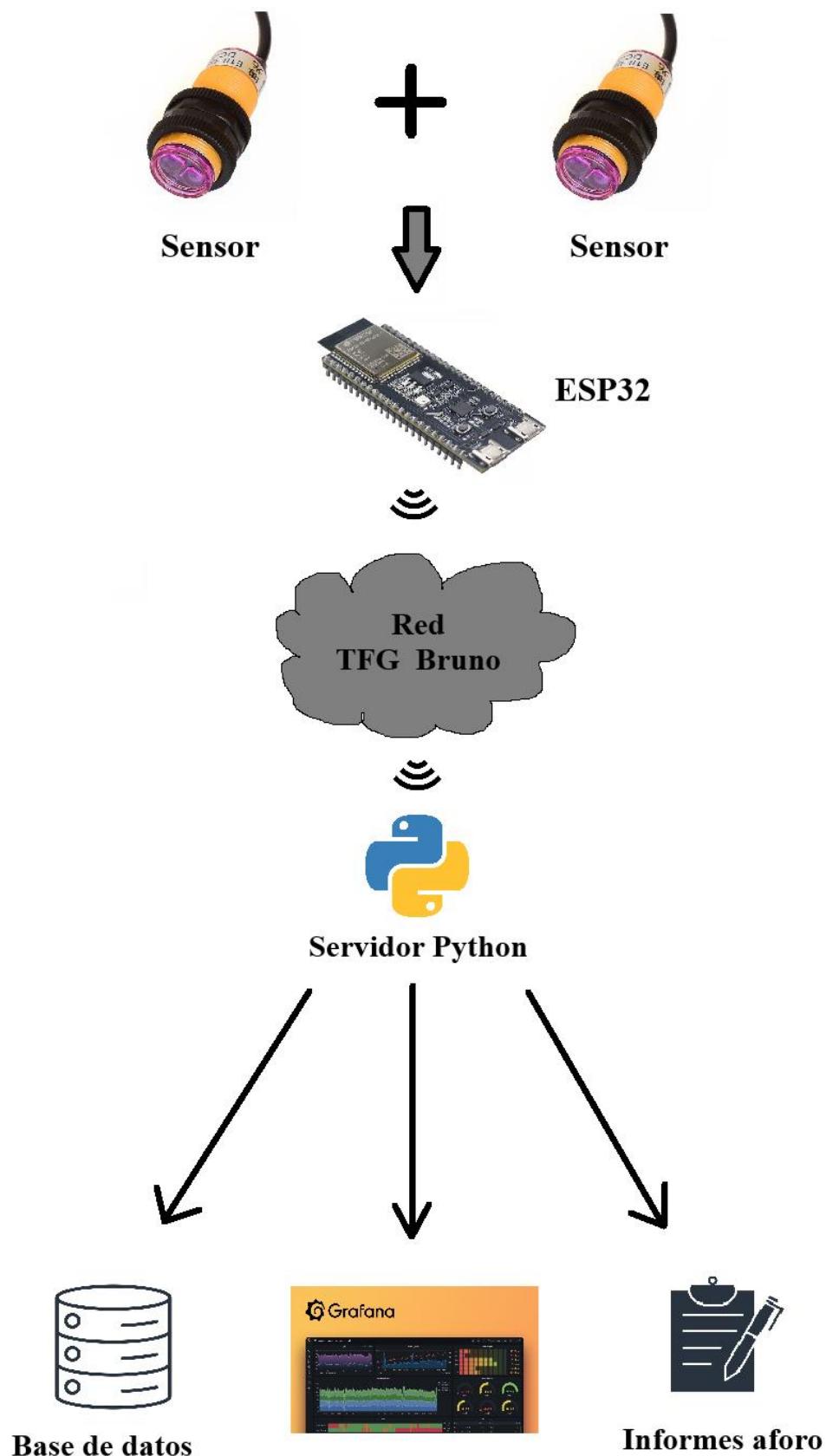


Figura 3. Esquema de funcionamiento del sistema

1.3 Metodología de trabajo

La metodología de trabajo de este proyecto se ha basado en dos metodologías de trabajo complementarias: la metodología ágil y la metodología de iteración y experimentación.

- La **metodología ágil** de trabajo es un enfoque colaborativo y flexible para la gestión de proyectos, especialmente en el desarrollo de software. Se basa en ciclos iterativos cortos denominados sprints o iteraciones, en los cuales se desarrolla y entrega un producto funcional de manera incremental. En lugar de planificar y diseñar todo el proyecto desde el principio, como en los métodos tradicionales, la metodología ágil permite adaptarse a los cambios y responder a las necesidades del cliente a medida que avanza el proyecto. Los equipos ágiles trabajan de manera colaborativa y autogestionada, manteniendo una comunicación constante y priorizando la entrega continua de valor.
- La **metodología de trabajo de iteración y experimentación** se basa en la creación y validación continua de ideas mediante ciclos repetitivos y pruebas. Este enfoque es común en áreas como la innovación, el diseño y el desarrollo de productos, donde la incertidumbre es alta y las soluciones finales no están claramente definidas desde el inicio. En lugar de intentar desarrollar una solución completa de una sola vez, los equipos construyen prototipos o versiones iniciales, los prueban en el mundo real, recopilan retroalimentación y luego ajustan o rediseñan el producto con base en los resultados obtenidos. Este ciclo de experimentar, medir y ajustar se repite tantas veces como sea necesario para llegar a una solución óptima.

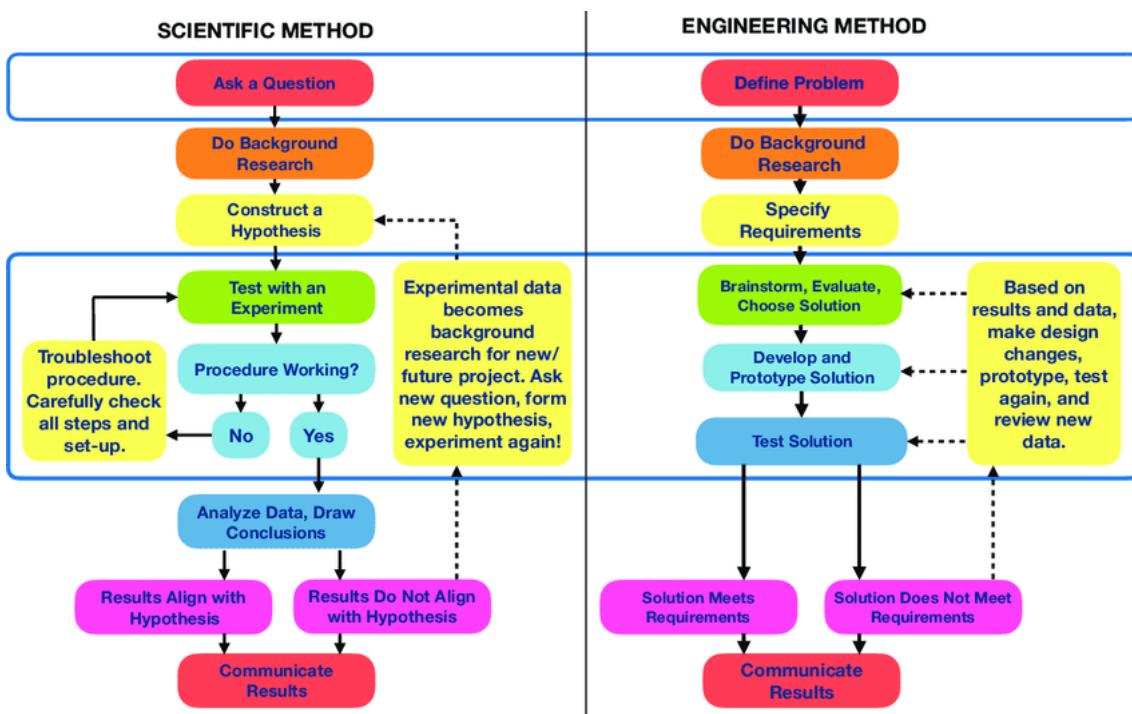


Figura 4. Comparativa del método científico y el método ingeniero

1.4 Cronograma del proyecto

El presente proyecto se ha realizado en el periodo que transcurre desde noviembre de 2023 hasta septiembre de 2024. El trabajo realizado se puede estructurar en las siguientes fases:

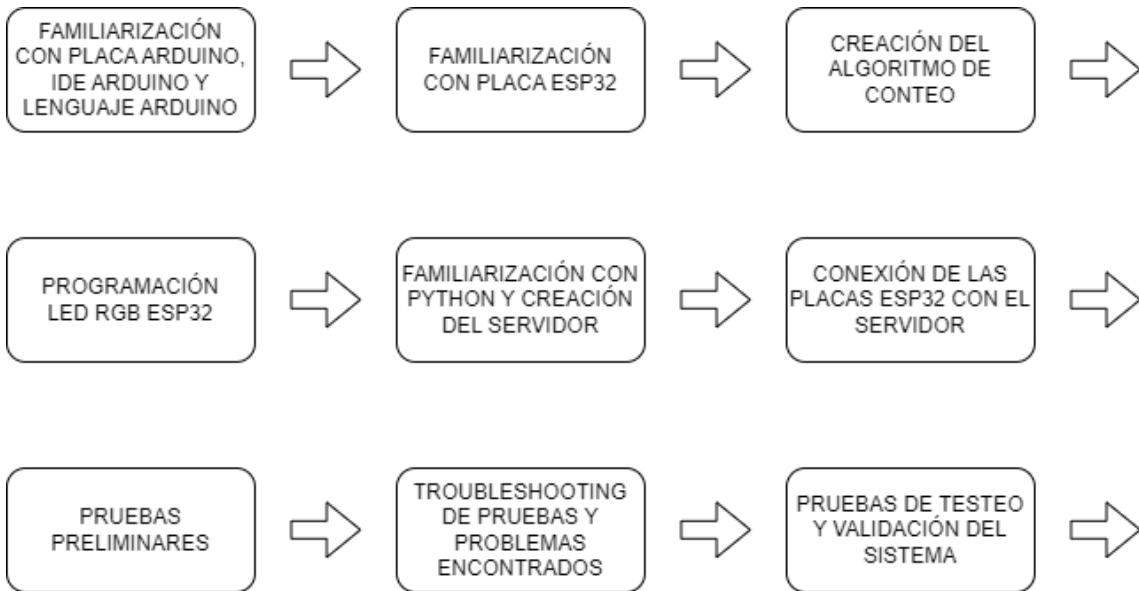


Figura 5. Cronograma del proyecto

Que una fase esté antes que otra no significa que haya que terminar por completo para empezar la siguiente, como bien hemos explicado en el apartado [1.3 Metodología del proyecto](#), los incrementos de los diferentes componentes funcionales del sistema han sido poco a poco y continuos.

Los bloques pertenecientes a la familiarización se deben a que a lo largo de la carrera no había estudiado dichas materias, aunque sí parecidas. Por ejemplo, nunca había trabajado con placas Arduino o su entorno de desarrollo, pero si había trabajado con el microcontrolador PIC en la asignatura de Sistemas Digitales Basados en Microprocesadores. De la misma manera, nunca había trabajado con el lenguaje Python, pero sí había visto la creación de servidores a través de sockets en Java en Sistemas Distribuidos.

También cabe resaltar que hay que diferenciar entre las pruebas preliminares y las pruebas finales de testeo y validación del sistema ya que no son lo mismo. Las primeras se refieren a las primeras pruebas que se hacen para ver cómo funciona el sistema en un principio y ver qué cambios hay que realizar para que nuestro sistema funcione como nosotros queremos. Una vez resolvemos todos los problemas que encontramos, entonces sí que podemos realizar las pruebas definitivas que van a darnos las conclusiones que podemos sacar del sistema que hemos creado.

1.5 Organización del proyecto

En esta sección vamos a narrar las distintas partes que nuestro proyecto va a tener sin tener en cuenta este mismo primer capítulo.

- **Tecnologías usadas.** En este capítulo se van a describir y explicar las tecnologías que hemos usado para llevar a cabo nuestro proyecto. Todo ello conlleva hablar de los sensores de proximidad infrarrojos, de las placas de desarrollo Arduino que hemos programado usando el entorno integrado de desarrollo (IDE) de Arduino, de las librerías que hemos usado para que el IDE reconozca nuestra placa ESP32, también vamos a hablar del lenguaje de programación de Python y el IDE que hemos usado para programar y depurar nuestro código, etc. Por último, también hablaremos de la creación de una base de datos para que Grafana pueda recolectar esos datos y exponerlos y la creación de informes periódicos del aforo.
- **Configuración de los sensores y programación de las placas de desarrollo ESP32.** En este capítulo se describirá ampliamente la configuración de los sensores y se explicará el algoritmo que hemos creado para contabilizar el flujo de personas que se encuentra en el código que hemos guardado en los ESP32, además de cómo se realiza la conexión de éstos con el servidor. También hablaremos de el funcionamiento del LED RGB del ESP32-S3-DevkitC-1.
- **Red y servidor Python.** En este capítulo se explicarán tanto la infraestructura de red como la manera en la que se ha implementado el servidor que recibe los diferentes mensajes que le envían los ESP32, cómo los procesa, cómo los guarda y cómo monitoriza la conexión con los clientes para saber si están en funcionamiento o si, por el contrario, por alguna causa, se han desconectado.
- **Presentación de los datos.** En este capítulo se describirá cómo se presentan los datos a través de Grafana mediante gráficos, paneles y tablas, y de cómo se generan los informes periódicos del aforo.
- **Pruebas y resultados.** Como su propio nombre indica, en este capítulo se hablará de las diferentes pruebas que se han realizado, de los resultados que hemos obtenido y analizaremos dichos resultados.
- **Conclusiones y trabajos futuros.** Este capítulo presenta las conclusiones del proyecto y alguna idea para posibles futuros proyectos similares o de ampliación a éste.
- **Bibliografía:** aquí se expondrán todos los enlaces de los que se ha sacado la información necesaria para la realización de este TFG.
- **Anexos**

Capítulo 2. Tecnologías usadas

En este capítulo se realiza una descripción en detalle de las diferentes tecnologías, tanto de hardware como de software que hemos usado (y aprendido a usar) en el proyecto. Se comentarán las características más importantes y por qué se ha optado por ellas. Las tecnologías y dispositivos que hemos usado son tanto los sensores de proximidad infrarrojos, como las placas de desarrollo ESP32, el IDE de Arduino para programar dichas placas, el IDE Pycharm para poder implementar nuestro servidor, la creación y uso de una base de datos, la interfaz de Grafana, y la generación de informes CSV del aforo.

2.1 Sensores de proximidad infrarrojos E18-D80NK

El sensor de proximidad infrarrojos E18-D80NK [1] es un dispositivo electrónico que se utiliza para detectar la presencia de objetos o personas dentro de un rango predefinido. Este sensor utiliza un emisor de luz infrarroja (IR) y un receptor que detecta la luz reflejada por los objetos. Al detectar un objeto, el sensor emite una señal de salida que puede ser utilizada por microcontroladores, como Arduino o Raspberry Pi, para tomar decisiones en sistemas automatizados.

Entre las características principales del E18-D80NK se incluyen su capacidad de ajuste de distancia, lo que permite configurar el rango de detección entre 3 cm y 80 cm, y su bajo consumo de energía, que lo hace adecuado para aplicaciones de bajo consumo. El sensor es resistente a interferencias ambientales, como la luz solar directa, y está encapsulado en una carcasa robusta, lo que le otorga mayor durabilidad en entornos industriales o en exteriores. Además, tiene una salida de tipo NPN normalmente abierto (NO), lo que facilita su integración con otros circuitos electrónicos.

En cuanto al precio, el sensor E18-D80NK es bastante accesible, en la fecha de elaboración de este trabajo, suele rondar entre 5 y 10 euros, dependiendo del distribuidor y las características adicionales que puedan ofrecerse, como cables o conectores. Para conocer más a fondo sus especificaciones recomendamos se vea el anexo.



Figura 6. Sensor E18-D80NK

2.2 Placas de desarrollo ESP32

En este proyecto hemos usado dos placas de desarrollo, tanto el ESP32-DevkitC V4 [\[2\]](#), como el ESP32-S3-DevkitC-1 [\[3\]](#).

Por una parte, el ESP32-DevKitC V4 es una placa de desarrollo basada en el popular microcontrolador ESP32, fabricado por Espressif Systems. Este módulo es conocido por su alta integración y capacidades inalámbricas, lo que lo convierte en una excelente opción para proyectos de IoT (Internet of Things). La placa DevKitC incluye todos los componentes esenciales para facilitar el trabajo con el ESP32, como reguladores de voltaje, un puerto USB para alimentación y programación, y un chip de interfaz serial para facilitar la conexión con una computadora.

Entre las características principales del ESP32-DevKitC, destaca su procesador dual-core Xtensa LX6 con una velocidad de hasta 240 MHz, acompañado de 520 KB de SRAM. Cuenta con soporte para Wi-Fi de 2.4 GHz y Bluetooth BLE (Bluetooth Low Energy) y clásico, lo que lo hace extremadamente versátil para una amplia variedad de aplicaciones. Además, el DevKitC ofrece múltiples pines de entrada y salida (GPIO), compatibilidad con diversos periféricos como UART, SPI, I2C y PWM, y la posibilidad de emplear conexiones analógicas y digitales. Todo esto lo convierte en una opción atractiva para proyectos que van desde domótica hasta sistemas de monitorización o automatización avanzada.



Figura 7. ESP32-DevKitC

Por otro lado, el ESP32-S3-DevkitC-1 es una versión más avanzada que introduce mejoras significativas en comparación con el ESP32 tradicional como el soporte para conexión Bluetooth LE 5.0 y Wi-Fi de doble banda, además de una RAM más amplia.

En cuanto al precio ambos suelen rondar los 10€, lo que los convierten en una opción atractiva muy por debajo del precio de una placa Arduino.

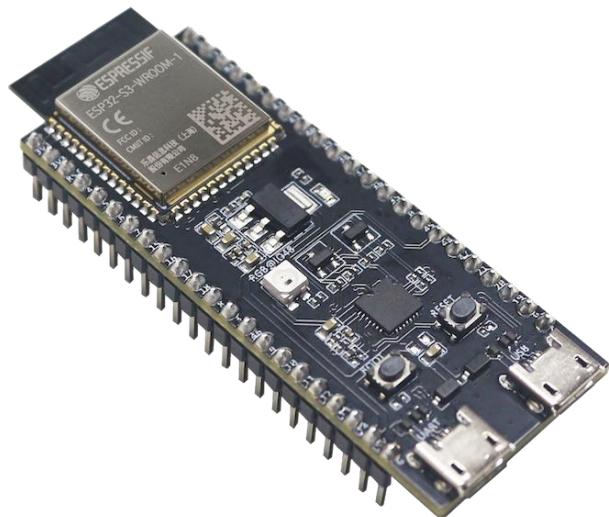


Figura 8. ESP32-S3-DevKitC-1

2.3 IDE Arduino

El entorno de desarrollo integrado (IDE por sus siglas en inglés) de Arduino [4] es una plataforma de software diseñada para programar y cargar código en microcontroladores de la familia Arduino y otros compatibles. Es un entorno de desarrollo fácil de usar, que permite escribir código en el lenguaje de programación Arduino (basado en C++) [5], compilarlo y cargarlo directamente en la placa mediante una conexión USB. El IDE incluye una interfaz sencilla con un editor de texto, opciones para verificar y compilar el código, y una serie de herramientas para monitorizar el comportamiento del dispositivo en tiempo real. Es compatible con una amplia gama de bibliotecas y ofrece soporte para una gran comunidad de desarrolladores, lo que facilita el desarrollo de proyectos electrónicos y de IoT.



Figura 9. Logo IDE Arduino

Por defecto, el IDE de Arduino incluye las librerías básicas para trabajar con las propias placas de Arduino como la Arduino UNO, pero no incluye las librerías necesarias para otras placas, por eso debemos de instalar las librerías correspondientes, en nuestro caso necesitamos dos, una para que funcionen nuestros ESP32 [6] y otra porque el ESP32-S3 DevkitC-1 tiene un LED RGB Adafruit Neopixel y necesitamos instalar la librería correspondiente [7] para hacerlo funcionar, ya que vamos a utilizar este LED para comprobar el correcto funcionamiento en el conteo de personas.

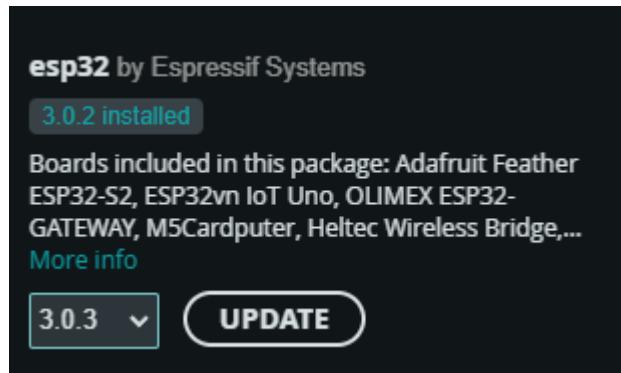


Figura 10. Librerías ESP32

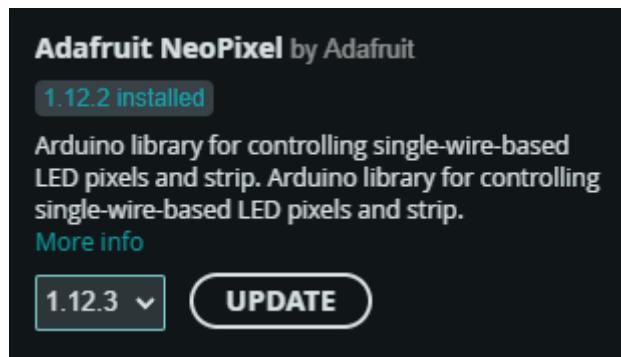


Figura 11. Librería Adafruit Neopixel

2.4 Python y Pycharm

Python [8] es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su simplicidad y legibilidad. Es ampliamente utilizado en diversas áreas, como desarrollo web, ciencia de datos, automatización, inteligencia artificial y aprendizaje automático. Python es popular debido a su sintaxis limpia y fácil de aprender, así como por su gran cantidad de bibliotecas y frameworks que simplifican tareas complejas. Su enfoque en la legibilidad del código y la productividad lo convierte en una excelente opción tanto para principiantes como para desarrolladores experimentados.



Figura 12. Logo de Python

PyCharm [9] es un IDE especializado en Python, desarrollado por JetBrains. PyCharm ofrece herramientas avanzadas para la escritura, depuración y prueba de código Python, como autocompletado de código, análisis de errores en tiempo real, integración con sistemas de control de versiones, depurador gráfico, y soporte para frameworks populares como Django y Flask. Es ideal tanto para desarrolladores principiantes como avanzados, proporcionando un entorno optimizado para trabajar eficientemente en proyectos Python de cualquier escala.



Figura 13. Logo de PyCharm

2.5 Base de datos SQLite

Las bases de datos SQLite [10] son ligeras, portables, embebidas y basadas en archivos, lo que significa que no requieren de un servidor para funcionar; todo se almacena en un único archivo de base de datos que se crea y gestiona automáticamente, lo que las hace ideales para aplicaciones con bajos requerimientos de recursos o que necesitan una base de datos simple y autónoma.

La librería SQLite3 [11] de Python es un módulo estándar que permite interactuar con bases de datos SQLite desde código Python. Con SQLite3, los desarrolladores pueden ejecutar comandos SQL (como consultas, inserciones y actualizaciones) directamente desde Python, crear tablas, manejar transacciones y administrar datos de manera eficiente.



Figura 14. Logo SQLite

2.6 Grafana

Grafana [12] es una plataforma de código abierto para la visualización y presentación de datos en tiempo real. Permite crear paneles interactivos y gráficos a partir de diversas fuentes de datos, lo que lo convierte en una herramienta poderosa para monitorear sistemas, infraestructura y aplicaciones. Grafana soporta múltiples tipos de bases de datos y formatos, como Prometheus, InfluxDB, MySQL y muchas otras, lo que lo hace altamente versátil para diferentes proyectos de análisis y monitoreo.



Figura 15. Logo de Grafana

2.7 Archivo CSV

Un archivo CSV [13] (Comma Separated Values) es un formato de archivo que almacena datos en forma de texto plano, donde cada línea representa un registro y las columnas están separadas por comas (u otro delimitador como punto y coma, dependiendo de la configuración regional). Es comúnmente utilizado para almacenar grandes conjuntos de datos de manera simple y eficiente.



Figura 16. Archivo CSV

Capítulo 3. Configuración de los sensores y programación de las placas de desarrollo ESP32

3.1 Funcionamiento de los sensores

La lógica de la salida del sensor E18-D80NK basada en un transistor bipolar NPN es clave para entender cómo este sensor puede controlar otros dispositivos o interactuar con un microcontrolador como, en nuestro caso, el ESP32.

El E18-D80NK es un sensor de proximidad infrarrojo que detecta objetos dentro de un rango de distancia y produce una señal de salida basada en la detección. El sensor tiene un transistor NPN en su circuito de salida, lo que significa que su configuración es "de colector abierto" [14].

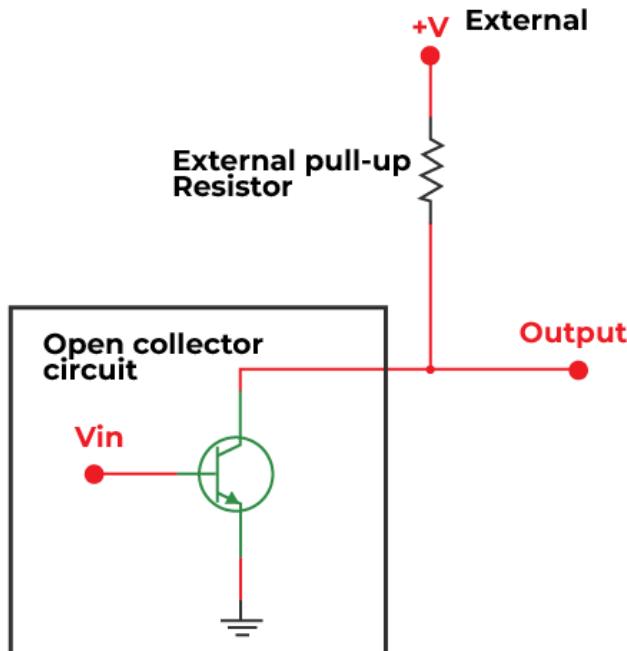


Figura 17. Circuito con BJT de colector abierto

Si el sensor no detecta ningún objeto en su rango de proximidad, el transistor NPN se encuentra en estado cortado (no conduce corriente entre colector y emisor). En esta situación, la salida del sensor está prácticamente desconectada, y el pin de salida no está conectado a tierra (GND). En un circuito con una resistencia de pull-up externa, la salida se encontrará en un estado alto (5V en nuestro caso).

Cuando el sensor detecta un objeto dentro de su rango, el transistor NPN se satura, lo que significa que ahora conduce corriente entre el colector y el emisor. Esto conecta el pin de salida del sensor a tierra (GND), y la salida del sensor se encontrará en un estado bajo (0V).

Es decir, cuando el sensor no detecta ningún objeto, el transistor se encuentra en estado cortado, por lo tanto, se comporta como un interruptor abierto; no hay corriente entre colector y emisor, por lo que la salida del sensor es alta (5V). Mientras que, si el sensor detecta un objeto, el transistor se encuentra en estado saturado, de manera que actúa como un interruptor cerrado; conecta la salida del sensor a GND, poniendo la salida en bajo (0V). Por esta razón, en el datasheet del sensor, vemos que el transistor se muestra meramente como un interruptor, tal y como se puede apreciar en la Figura 18:

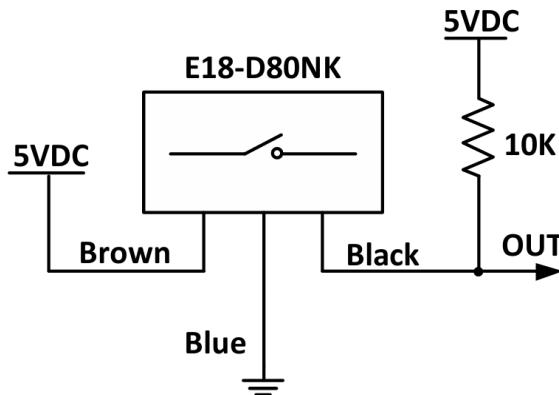


Figura 18 Circuito Sensor

La resistencia pull-up que se encuentra en las figuras 17 y 18 no es necesaria en nuestro diseño puesto que en el código hemos dicho que las entradas sean INPUT_PULLUP, de manera que la placa ESP32 ya lo tiene en cuenta, si la entrada la configurásemos como INPUT sí que sería necesario colocar esa resistencia en nuestro circuito.

En la Figura 19 se puede observar el diagrama del transmisor y receptor de los sensores E18-D80NK:

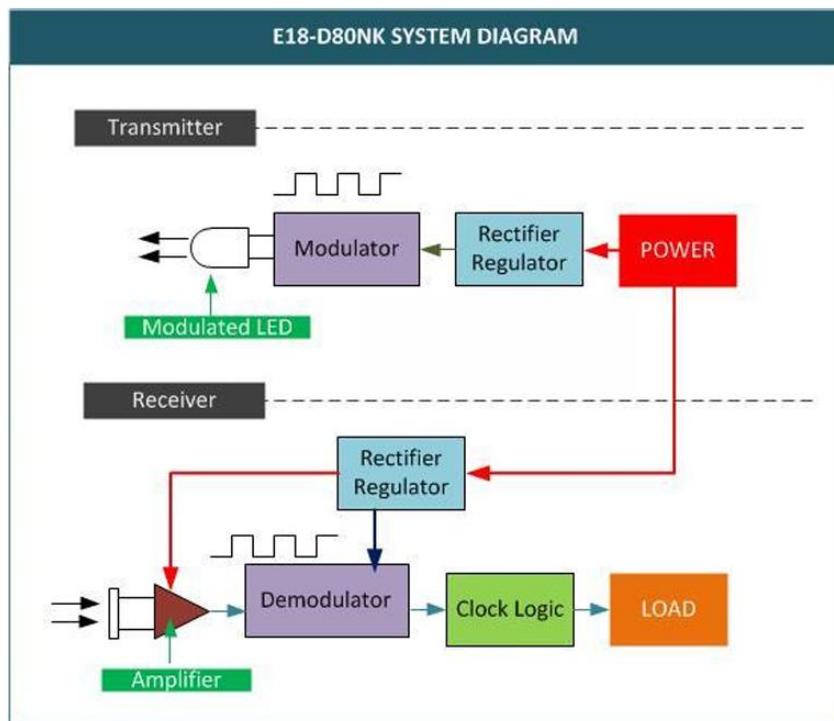


Figura 19. Diagrama transmisor y receptor del sensor E18-D80NK

3.2 Configuración de los sensores

La configuración de los sensores de proximidad infrarrojos es bastante sencilla, al menos en teoría, puesto que ya veremos en el apartado de problemas de esta sección del trabajo todos los contratiempos con los que nos hemos enfrentado para configurar los sensores para que funcionen correctamente.

Los sensores E18-D80NK traen en su parte posterior un LED que nos indica si están o no detectando algo. Si el LED se enciende es porque el sensor está detectando algo y cuando está apagado es porque no está detectando nada. También traen un tornillo conectado a un potenciómetro que nos permite configurar la sensibilidad del sensor, es decir, la distancia a la que detecta o no una persona u objeto. Para ajustar el tornillo necesitaremos un destornillador plano. Si giramos el tornillo en el sentido de las agujas del reloj incrementaremos la distancia a la que el sensor detecta y, por el contrario, si lo giramos en el sentido contrario de las agujas estaremos reduciendo la distancia a la que detecta. Cabe recordar que el sensor tiene un rango de detección de entre 3 cm y 80 cm de longitud.

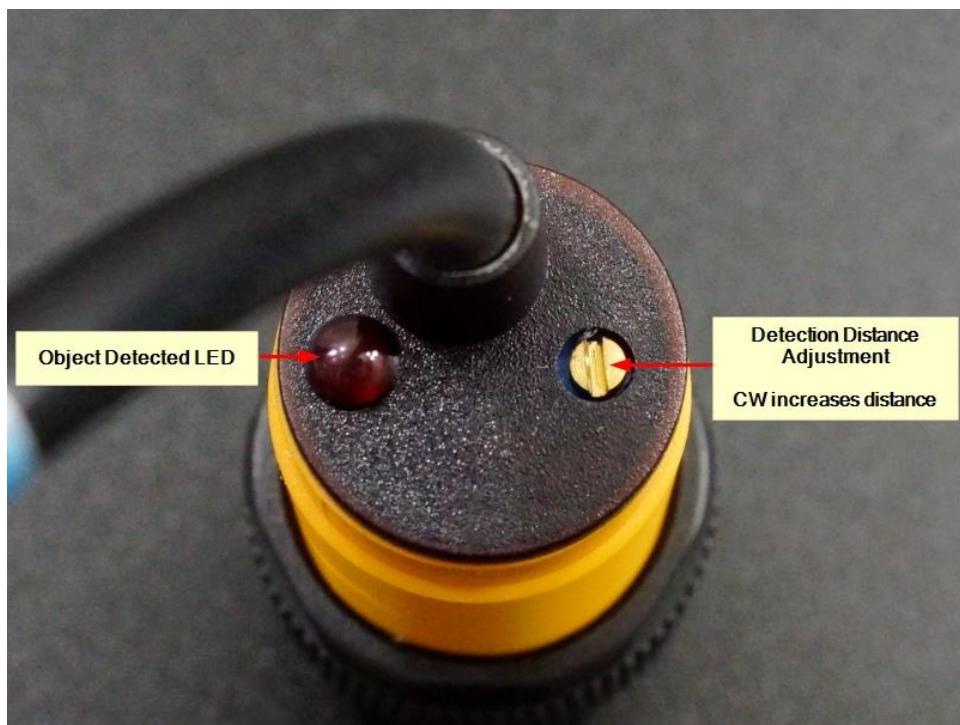


Figura 20. Imagen en detalle de la parte posterior del sensor E18-D80NK

Se recomienda usar una cinta métrica y colocar el sensor enfrente de una pared a la distancia a la que lo queramos configurar. Si el LED está iluminado, lo que tenemos que hacer es girar el tornillo en dirección contraria a la de las agujas del reloj hasta que el LED se apague. En caso contrario, si el LED inicialmente está apagado, tendremos que girar el tornillo en la dirección de las agujas del reloj hasta que el LED se encienda (véase el datasheet de los sensores del [anexo](#)). En nuestro caso, como vamos a colocar los sensores en los arcos antihurto de la entrada de la biblioteca de Antigones, tenemos que configurarlos con la máxima distancia de 80 cm puesto que la distancia entre los arcos es de 97 cm. Nos quedan 17 cm sin cubrir que no importan puesto que las dimensiones de una persona son superiores a esa.

3.3 Cableado de los sensores

Los sensores traen tres cables que se pueden apreciar en la Figura 21. Uno de los cables es de color marrón y es el correspondiente a Vcc a 5V, otro es de color azul y es el que corresponde con GND (0V) y, por último, tenemos el de color negro que corresponde a la salida del sensor. Esta combinación de colores puede resultar algo confusa puesto que cabría pensar que el cable que corresponde a los 5V fuera de color rojo y el que corresponde a GND fuera de color negro, siendo el tercer cable de la salida del sensor de cualquier otro color, pero usar el cable azul para los 5V, el marrón para GND y el negro como la salida del sensor es el estándar que se utiliza en sensores industriales. Esa es la razón por la que tenemos esta combinación de colores.



Figura 21. Detalle de los cables del sensor

Una vez conocemos la configuración de los cables, toca conectarlos al sensor. Hemos usado dos sensores por cada ESP32 y hemos usado dos ESP32, es decir, que en total hemos usado cuatro sensores. En el capítulo referente a la programación de las placas de desarrollo explicaremos por qué hemos usado cuatro sensores y dos ESP32. Para conectar los sensores correctamente a las placas hay que conocer el pin-layout de las mismas. En las figuras 22 y 23 se muestra el pin-layout de éstas y en la Figura 24 se muestra la conexión de los cables en la protoboard del ESP32-S3-DevkitC-1.

ESP32-S3-DevKitC-1

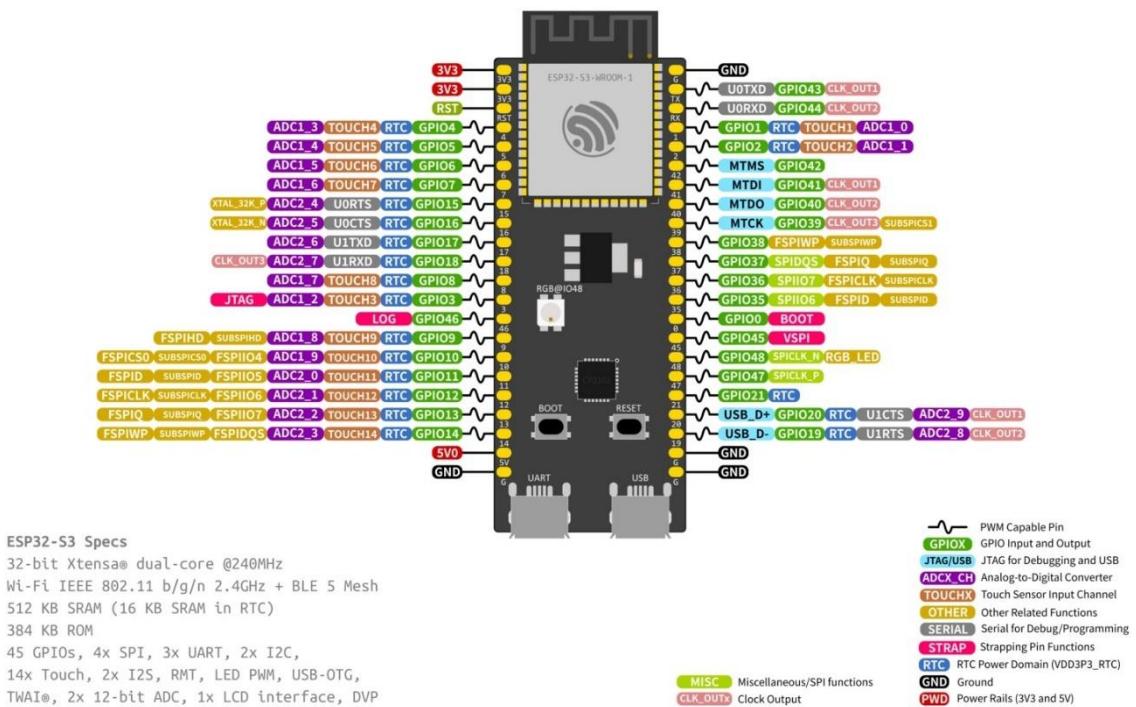


Figura 22. Pin-layout del ESP32-S3-DevKitC-1

ESP32-DevKitC

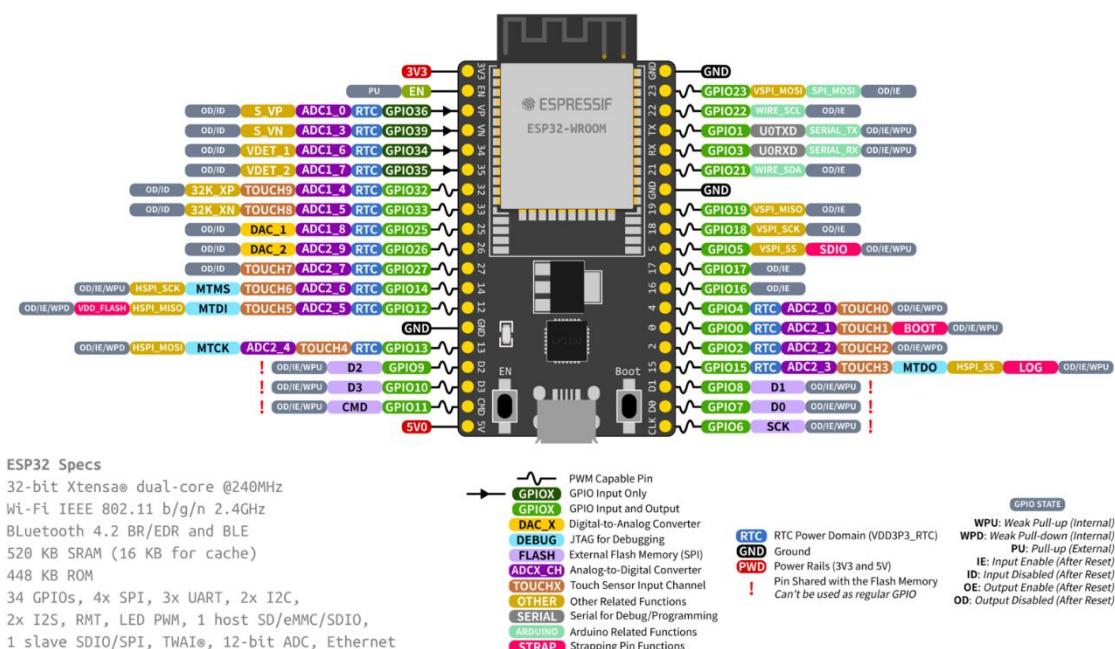


Figura 23. Pin-layout del ESP32-DevKitC

Como podemos observar, ambas placas ESP32 únicamente tienen un pin de 5V, por lo que necesitaremos de una protoboard para puentear ese pin y alimentar a los dos sensores. También puentearemos el mismo pin GND para ambos sensores.

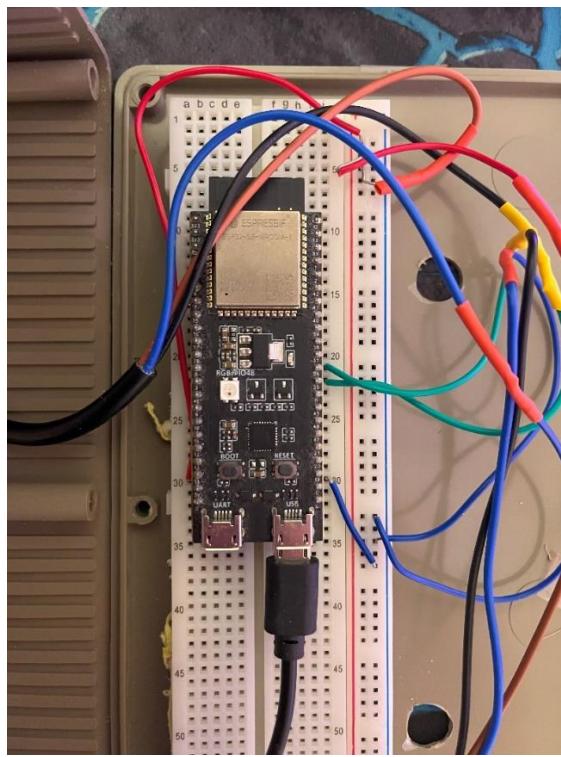


Figura 24. Cableado de los sensores a la placa ESP32-S3-DevKitC-1

En la Figura 25 se muestra un esquema en el que se puede observar mejor la conexión de los cables:

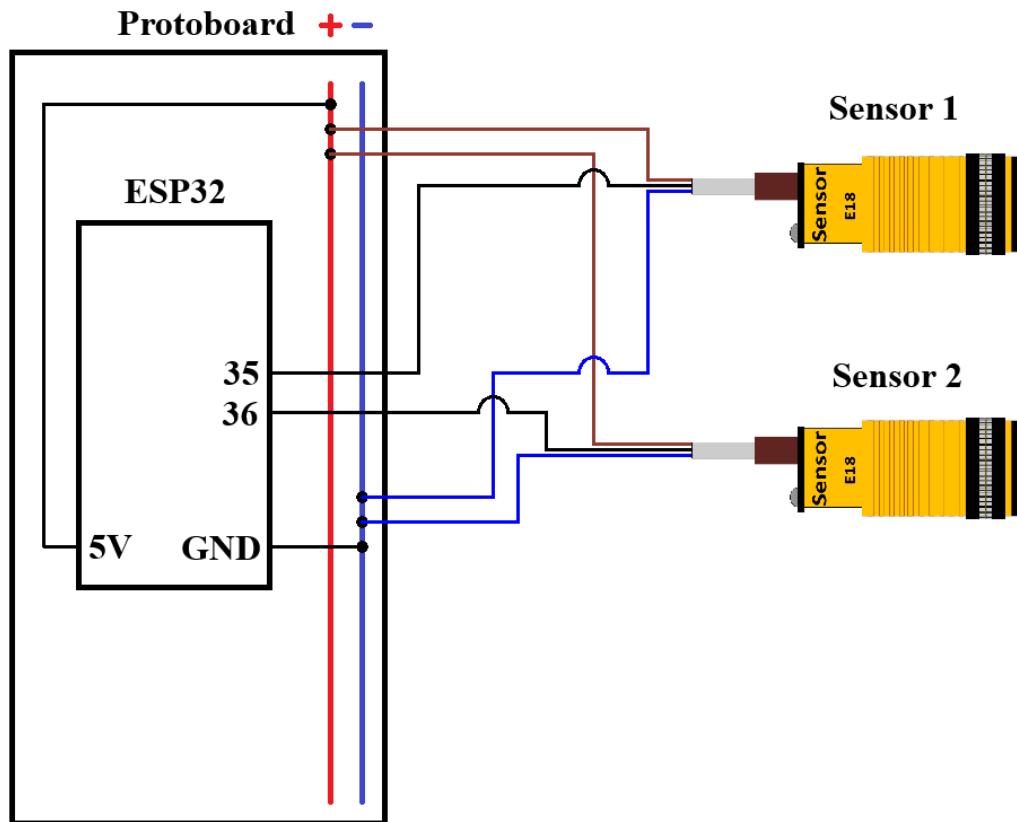


Figura 25. Esquema del cableado de los sensores con la placa ESP32-S3-DevKitC-1

Una vez hemos explicado el cableado de alimentación (cables marrón y azul) vamos a proceder con el de la salida del sensor, es decir, el cable negro. Para el ESP32-DevkitC V4 hemos optado por conectar el sensor 1 y el sensor 2 a los pines configurables de entrada y salida 25 y 26, respectivamente. Para el ESP32-S3 DevkitC-1 hemos optado por conectar el sensor 1 y el sensor 2 a los pines configurables de entrada y salida 35 y 36, respectivamente.

3.4 Encapsulado de los sensores y las placas ESP32

Para que los sensores y la placa no estuvieran expuestos se ha optado por encapsularlos en una pequeña caja de prototipado. En la Figura 26 se puede apreciar dicha caja por fuera, y en la Figura 24 ya se vio la parte interior de la caja con la protoboard, la placa ESP32-S3 DevkitC-1 y el cableado:



Figura 26. Parte frontal de la caja con sus dos sensores

Además, a esta caja se le practicaron un par de aperturas en la parte posterior a la altura de los LEDs de los sensores para comprobar si estos funcionan correctamente. Por el lateral tiene otra apertura para pasar el cable USB para su correspondiente alimentación.

Hay que tener en cuenta que es una caja de prototipado, para una versión de producción se encapsularía todo en una caja más pequeña con una apariencia lo más cercana posible a los arcos de la entrada de la biblioteca para pase lo más desapercibida posible.

3.5 Programación de las placas de desarrollo ESP32

3.5.1 Algoritmo de conteo

Habiendo explicado el funcionamiento, cómo se configuran y cómo se conectan los sensores, procedemos a programar las placas ESP32. Para ello usaremos el IDE de Arduino puesto que, para familiarizarme con los sensores, primeramente, he usado una placa Arduino UNO y resulta, que este software de programación también es compatible con las placas ESP32. Así que, como ya hemos dicho anteriormente, el lenguaje de programación que vamos a usar es Arduino.

Cabe destacar que una de las ventajas que nos brindan los sensores E18-D80NK es que son sensores digitales, no analógicos, y por lo tanto nos dicen si detectan o no alguna persona u objeto y eso nos facilita bastante trabajar con ellos a la hora de programar las placas de desarrollo ESP32.

Antes de proceder a explicar el algoritmo que hemos creado para contabilizar el flujo de personas hay que mencionar unas consideraciones que hemos asumido.

Para empezar, puesto que hay tres arcos antihurto, hay dos huecos para entrar o salir de la biblioteca, de manera que tenemos que controlar los flujos de esos dos huecos. Por esta razón vamos a colocar dos cajas como la que hemos visto en el apartado [3.4 Encapsulado de los sensores y las placas ESP32](#) donde cada una de ellas contiene una protoboard con un ESP32 y dos sensores E18-D80NK. Estas cajas irán en los huecos de los arcos como se muestra la Figura 27:



Figura 27. Localización de la caja en los arcos

La siguiente consideración que debemos tener en cuenta, es que los sensores tienen que estar lo suficientemente cerca el uno del otro como para que se activen a la vez cuando pasa una persona. En nuestro caso se ha optado por que los sensores estén a 10 cm el uno del otro. Además, el dispositivo debe ser capaz de detectar si la persona está entrando o saliendo.

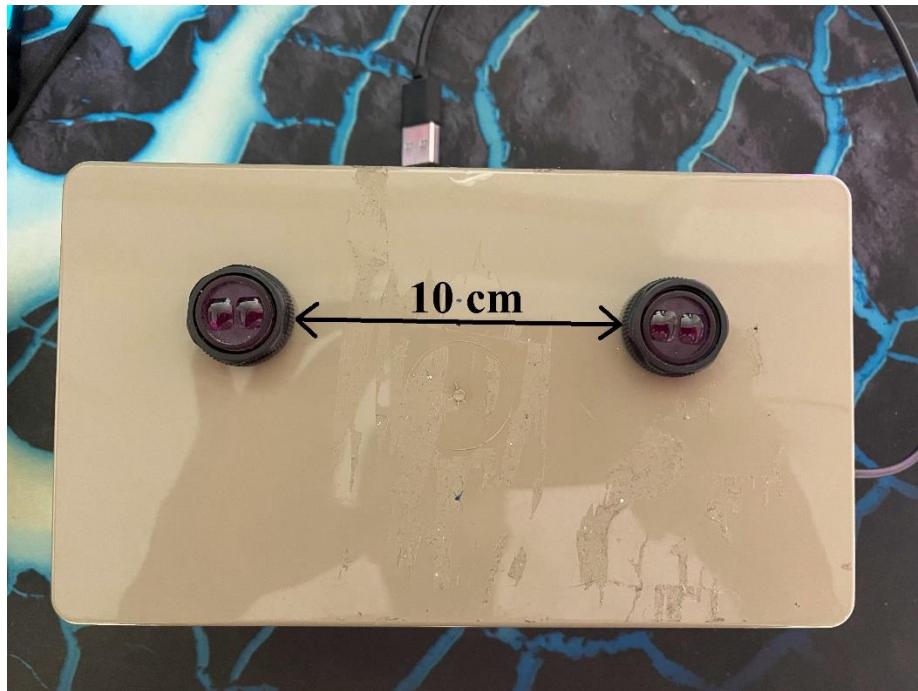


Figura 28. Distancia entre los sensores

Uno de los conceptos fundamentales para ejecutar nuestro algoritmo es el de las máquinas de estado finitas. En nuestro caso se ha optado por una máquina de estado Mealy para los sensores cuyos estados son NADIE y PERSONA_DETECTADA como se puede observar en la Figura 29:

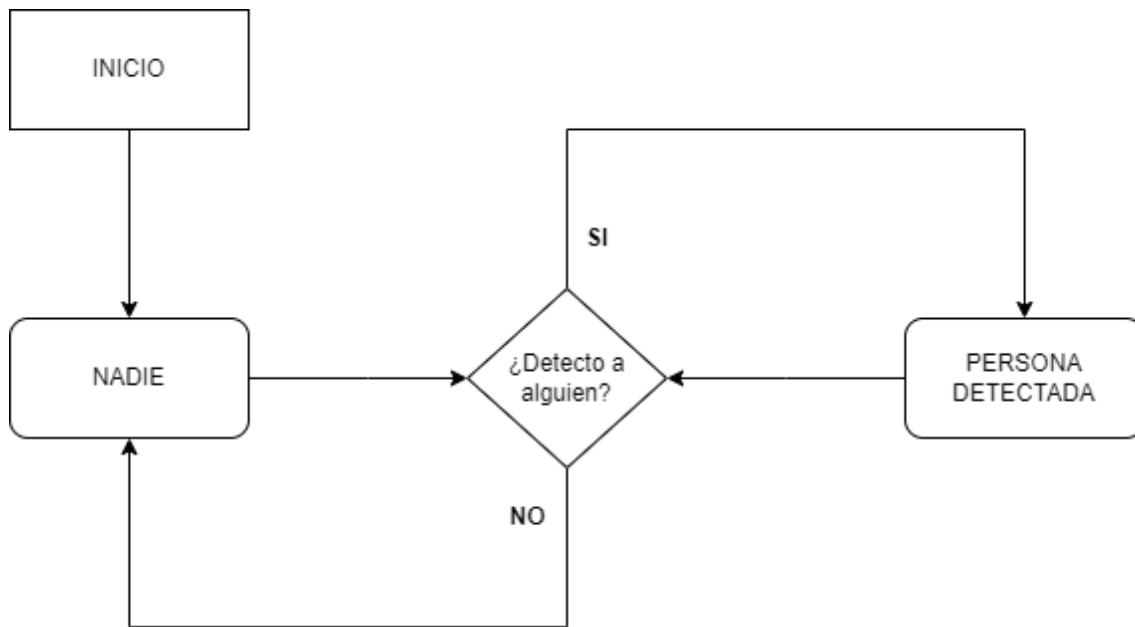


Figura 29. Máquina de estados finita

Los esquemas que se están presentando en este apartado están realizados con la web draw.io [15].

Esta máquina de estados finita únicamente puede tener dos entradas, si el sensor detecta o si no detecta a alguien. La máquina empieza en el estado NADIE y si no detecta a alguien sigue en ese mismo estado, pero, cuando detecta a alguien, transita hacia el estado PERSONA_DETECTADA. Cuando esta se encuentra en el estado PERSONA_DETECTADA, si sigue detectando asume que es la misma persona y se queda en ese mismo estado, y si deja de detectar vuelve al estado NADIE. Cuando empieza a detectar a alguien guarda en la variable **tiempo_empiezaDetectar** el tiempo en el que ha comenzado a detectar y cuando deja de detectar guarda en la variable **tiempo_dejaDetectar** el tiempo en el que deja de detectar (estas serían las salidas de la máquina).

Realmente la máquina de estados finita que se muestra en la Figura 29 es una simplificación porque no tiene en cuenta las salidas. Si tuviera en cuenta las salidas tendría la siguiente forma donde en rojo se pueden apreciar las salidas:

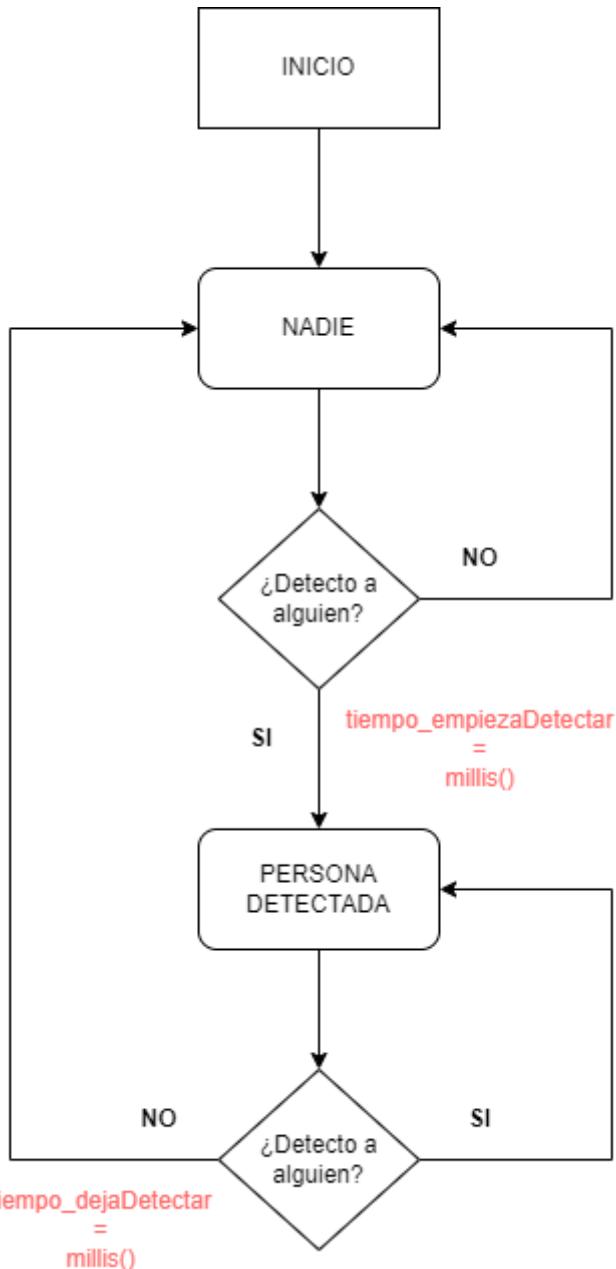


Figura 30. Máquina de estados finita completa

Esta máquina de estados finita funciona para cada sensor de manera independiente y la clave del algoritmo es la secuencia en la que los sensores detectan y dejar de detectar a alguien. Como bien vemos en la Figura 27, el sensor 1 será el que esté siempre más hacia afuera de la biblioteca, y el sensor 2 el que esté más hacia adentro. De manera que, si una persona entra, primero le detectará el sensor 1 y después el 2 y le dejará de detectar primero el sensor 1 y después el 2. Por el contrario, si una persona sale, le empezará a detectar primero el sensor 2 y después el 1 y le dejará de detectar primero el sensor 2 y después el 1.

Esta es la base de nuestro algoritmo. Pues bien, cuando ambos sensores estén detectando a alguien se compararán las variables **tiempo_empiezaDetectar** del sensor 1 y del sensor 2, de manera que, si la del sensor 2 es mayor que la del sensor 1, tendremos una posible entrada. Por el contrario, si la variable **tiempo_empiezaDetectar** del sensor 1 es mayor que la del sensor 2 tendremos una posible salida.

Cuando hayamos tenido una posible entrada o una posible salida, y cuando ambos sensores dejen de detectar a alguien sabremos si alguien ha entrado o no o si ha salido o no. ¿Cómo sabemos si realmente alguien ha entrado o ha salido? Pues, si tenemos una posible entrada, cuando ambos sensores dejen de detectar a alguien tendremos que comparar las variables **tiempo_dejaDetectar** del sensor 1 y del sensor 2, si la del sensor 2 es mayor que la del sensor 1, es decir, si el sensor 2 le ha dejado de detectar después que el sensor 1, entonces es que alguien realmente habrá entrado, si no, es que alguien se ha dado media vuelta y no ha entrado. De la misma manera, si tenemos una posible salida, cuando ambos sensores dejen de detectar a alguien compararemos las variables **tiempo_dejaDetectar** de ambos sensores; si la del sensor 1 es mayor que la del sensor 2, es decir, si el sensor 1 ha dejado de detectarle después que el sensor 2, es que alguien ha salido, si no, es que alguien se ha dado media vuelta y no ha salido. Esto se puede ver en el esquema de la Figura 31:

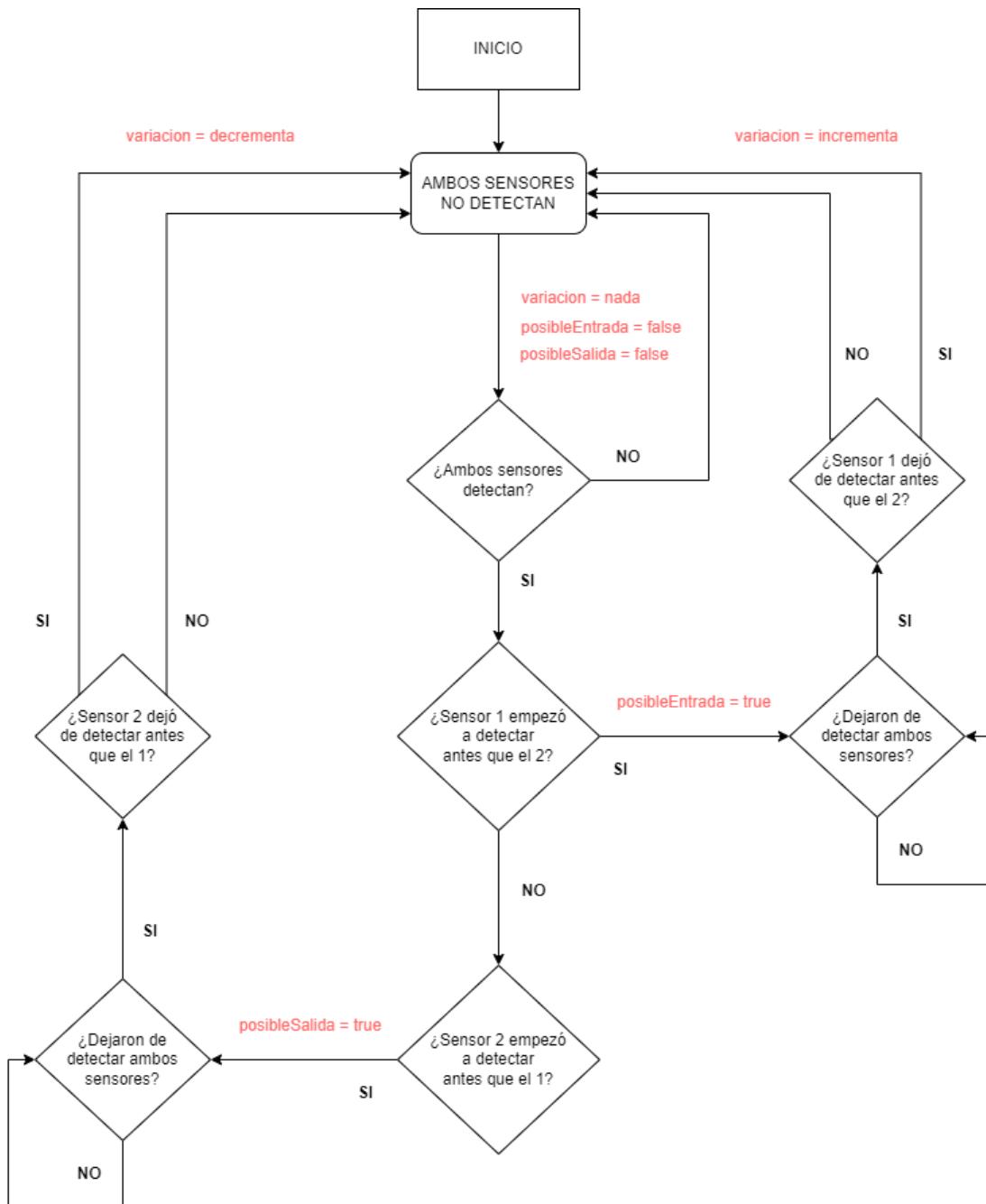


Figura 31. Esquema del algoritmo de conteo

Tenemos una variable que se llama variación que va a mandar al servidor si alguien ha entrado o salido, es decir, si tiene que incrementar o decrementar el aforo. Hemos decidido que sea así porque tenemos dos ESP32, así que es mejor que el aforo lo cuente el servidor, ya que si se guardara en los ESP32 habría incoherencias en el conteo. Además, de esta forma evitamos tener que trabajar con variables almacenadas persistentemente en la memoria (en el caso de que un ESP32 se reinicie), y tener que distinguir si se ha reiniciado, o si se ha apagado de noche para evitar un consumo innecesario. Por último, este diseño también permite ampliar el sistema a cualquier número de sensores de entrada/salida, por ejemplo, si la entrada dispusiera de más arcos de entrada/salida, o si tuviera diferentes puertas de entrada/salida. Esto último no es habitual en una biblioteca, pero sería igualmente posible contabilizar el aforo con esta centralización del recuento de personas.

3.5.2 Conexión de los ESP32 con el servidor

Para conectar los ESP32 con el servidor hemos creado una red Wi-Fi cuyo SSID es “TFG Bruno”. El ESP32 intenta conectarse a esta red y cuando lo logra intenta conectarse al servidor varias veces hasta que lo consigue. Inicialmente intenta conectarse al servidor cada 750 ms y una vez lo ha intentado 5 veces lo intenta cada 2 minutos. Cuando ha logrado conectarse entra en el bucle correspondiente al algoritmo de conteo de personas que hemos explicado con detalle en el apartado [3.5.1 Algoritmo de conteo](#) para finalmente mandar los datos correspondientes al servidor. En la Figura 32 se muestra el esquema que acabamos de describir:

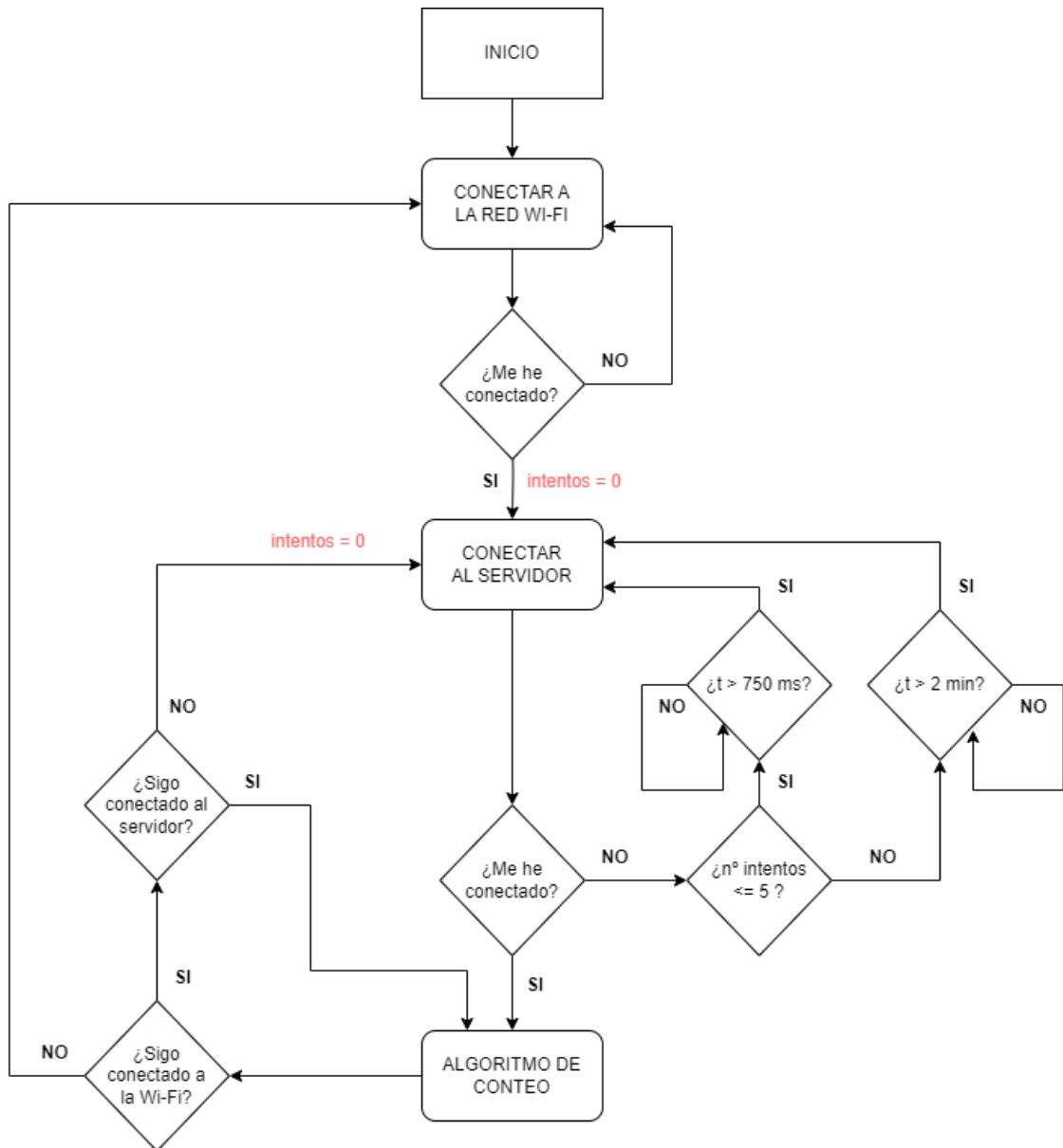


Figura 32. Algoritmo de conexión de los ESP32 al servidor

El bloque “Algoritmo de conteo” sería todo el esquema de la Figura 31, pero se ha optado por explicar el funcionamiento en varios esquemas de manera separada para facilitar la comprensión de estos.

Ahora explicaremos cómo mandan los datos los ESP32 al servidor. Cuando se conectan al servidor envían un mensaje de presentación con los siguientes campos:

- **ID de dispositivo.** Es el número que identifica a cada placa de desarrollo, en nuestro caso, el ESP32-S3 DevkitC-1 tiene el número 1 y el ESP32-DevKitC V4 tiene el número 2.
- **Número de secuencia.** Este campo nos ayudará a observar si los mensajes que se mandan al servidor llegan en orden. Inicialmente empieza en 1 y va incrementando con cada mensaje que se manda.
- **Presentación.** Cadena de caracteres que dice “presentacion” para que el servidor sepa que ese cliente está activo.

Una vez se ha conectado con el servidor y ha mandado ese primer mensaje de presentación, entonces mandará mensajes con los siguientes campos cuando se detecte que alguien ha entrado o salido. También se envían mensajes cada minuto, aunque no se detecte la presencia de ninguna entrada o salida. Esto se realiza a modo de “keep-alive” para que el servidor sepa que el ESP32 no se ha desconectado.

- **ID de dispositivo.** Es el número que identifica a cada placa de desarrollo, en nuestro caso, el ESP32-S3 DevkitC-1 tiene el número 1 y el ESP32-DevKitC V4 tiene el número 2.
- **Número de secuencia.** Este campo nos ayudará a observar si los mensajes que se mandan al servidor llegan en orden. Inicialmente empieza en 1 y va incrementando con cada mensaje que se manda.
- **Variación.** Cadena de caracteres que dice “incrementa” o “decrementa” para que el servidor aumente o decremente su variable aforo.

Todos los campos se envían en forma de String. A continuación, se muestra un esquema en el que se muestra el envío de mensajes. Este esquema está combinado con el de la Figura 31 y 32:

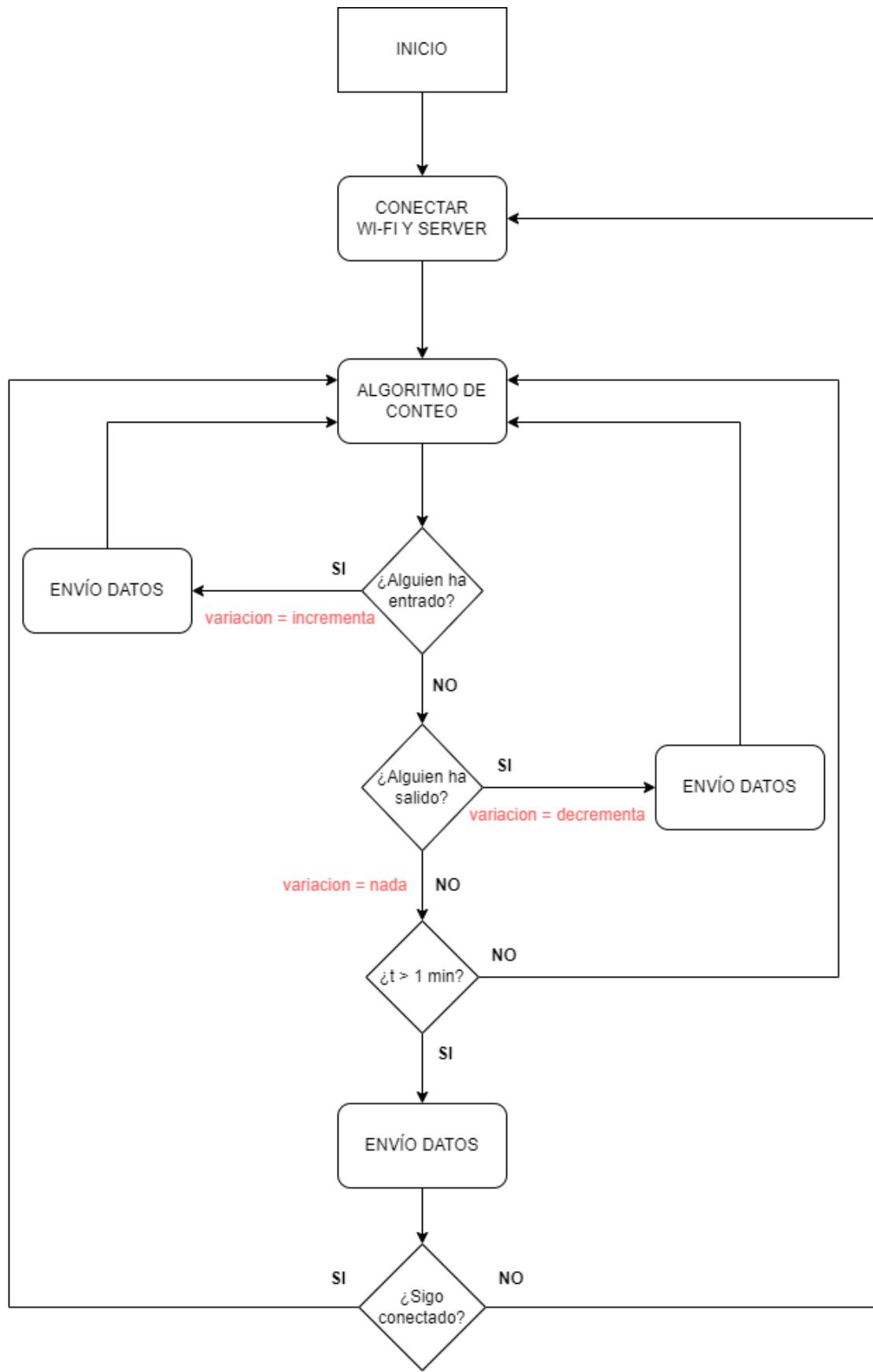


Figura 33. Esquema del envío de datos de los ESP32 al servidor

La variable t (en el código de Arduino ultima_actualizacion) que controla que al menos cada minuto se envíe un mensaje al servidor vuelve a estar a 0 cada vez que se manda un mensaje.

3.5.3 Configuración LED RGB del ESP32-S3

El ESP32-DevKitC V4 no tiene ningún LED RGB integrado, pero el ESP32-S3 DevkitC-1 sí que posee un LED RGB Adafruit Neopixel así que vamos a configurar ese LED para que nos sirva de ayuda a la hora de ver si nuestro ESP32 está funcionando correctamente ya que, gracias a las aperturas que hemos hecho en la parte posterior de la caja, vamos a poder ver el color que tiene éste. Vamos a usar la siguiente configuración en el LED:

- **Color rojo:** significa que el ESP32 se encuentra desconectado de nuestro servidor
- **Color azul:** significa que el ESP32 se encuentra conectado a nuestro servidor.
- **Color verde:** significa que una persona ha entrado. Este color se mantendrá únicamente durante un segundo antes de volver al azul.
- **Color amarillo:** significa que una persona ha salido. Este color se mantendrá únicamente durante un segundo antes de volver al azul.

A continuación, se muestra el esquema correspondiente a la configuración de colores del LED:

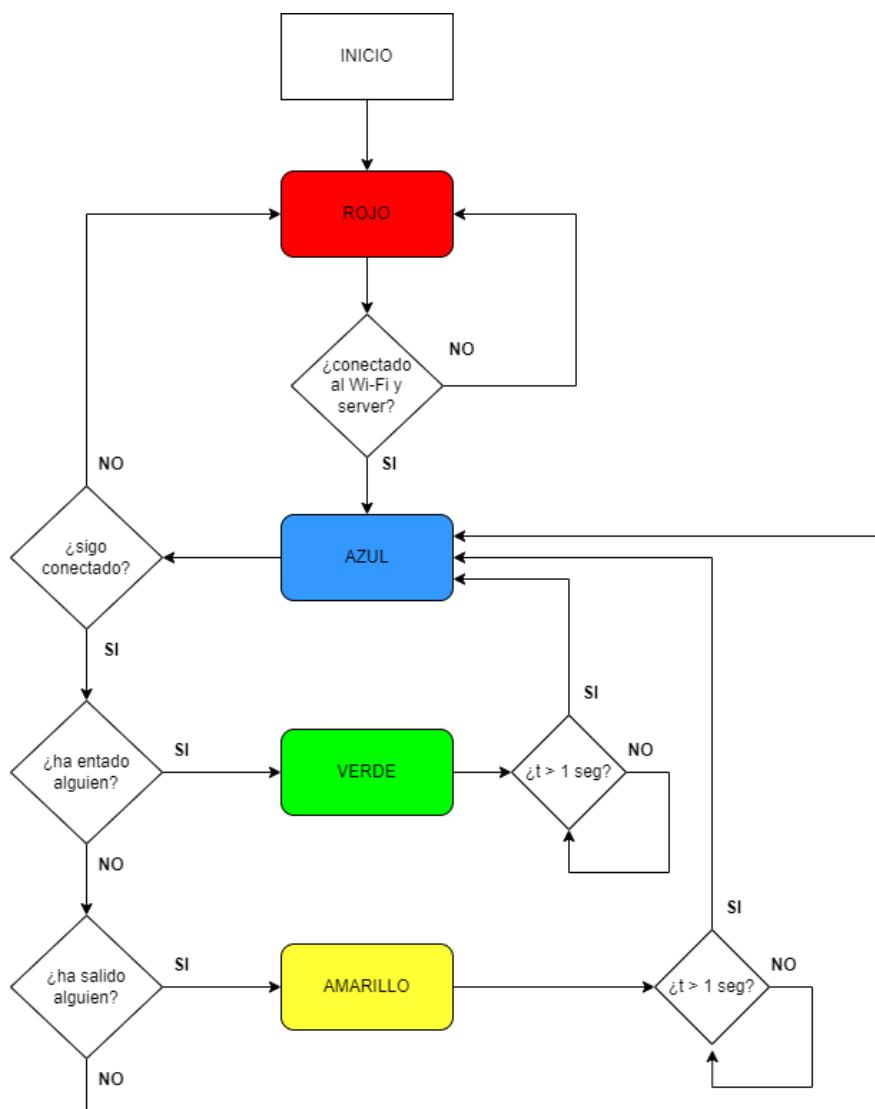


Figura 34. Esquema de la configuración del LED RGB

3.6 Problemas surgidos

Problemas con los sensores

El encapsulado que se muestra en la Figura 26 de la sección [3.4 Encapsulado de los sensores y las placas ESP32](#) sería el encapsulado que se usaría en cada placa, pero para agilizar la tarea y poder realizar pruebas, para la segunda placa hemos optado por algo más sencillo que se muestra en la Figura 35 y 36:

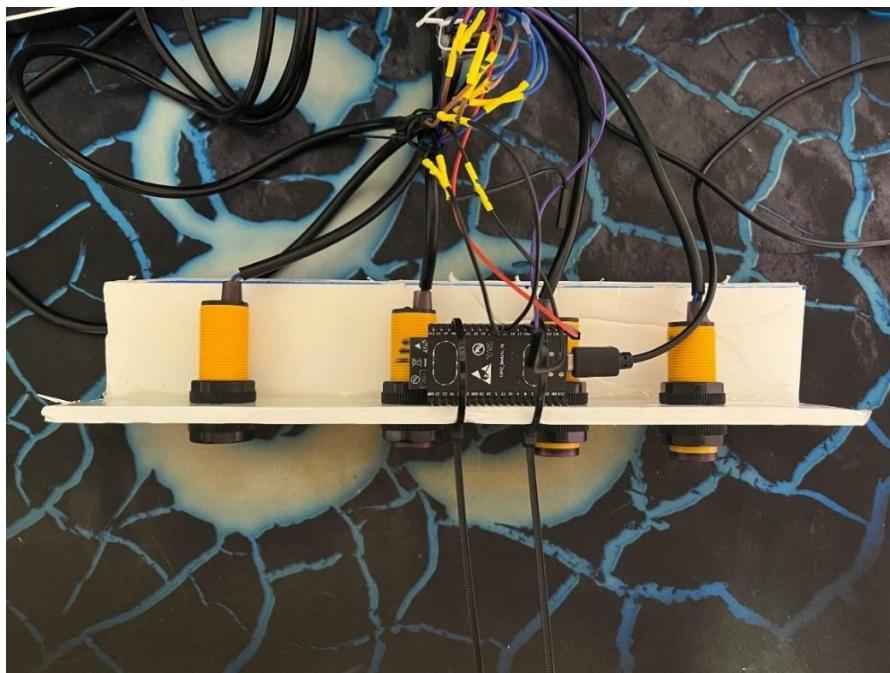


Figura 35. Vista desde arriba de la segunda estructura de pruebas

Hemos usado esta segunda estructura para los sensores porque así podíamos probar diferentes distancias entre los sensores, el problema es que estos sensores son de otro distribuidor y no funcionan correctamente, son de demasiada poca calidad, los primeros, que son del distribuidor Lefircko sí que funcionan correctamente.

¿Cómo sabemos que no funcionan correctamente estos sensores? Pues bien, como ya hemos dicho los sensores que teníamos inicialmente son del distribuidor Lefircko. Teníamos tres de estos sensores y dos de ellos han sido usados para el primer encapsulado conectados al ESP32-S3-DevkitC-1 mientras que el tercero se usó primeramente sin encapsulado y conectado al ESP32-DevkitC V4 con un código simplificado, ya que solo teníamos un sensor, y, cuando éste detectaba a alguien, de manera aleatoria, mandaba un mensaje diciendo si tenía que incrementar, decrementar o dejar el aforo como estaba. Se realizaron pruebas y funcionaba como era esperado.

Los cuatro sensores que se muestran en la Figura 35 y 36 se han adquirido después y se han calibrado igual que los primeros, incluso con un multímetro y, ni aun así funcionaban correctamente, de manera que se recomienda comprar otros sensores de otro distribuidor como Lefircko que sí parece más fiable. También hay que comentar que la lógica de los sensores no era la misma en todos. Mientras que el de más a la derecha era igual que los de Lefircko como hemos explicado en el apartado [3.1 Funcionamiento de los sensores](#), en los otros tres era inversa. Para aclarar, aunque haya cuatro sensores en la estructura solamente se conectaban dos a la vez al ESP32-DevkitC V4.

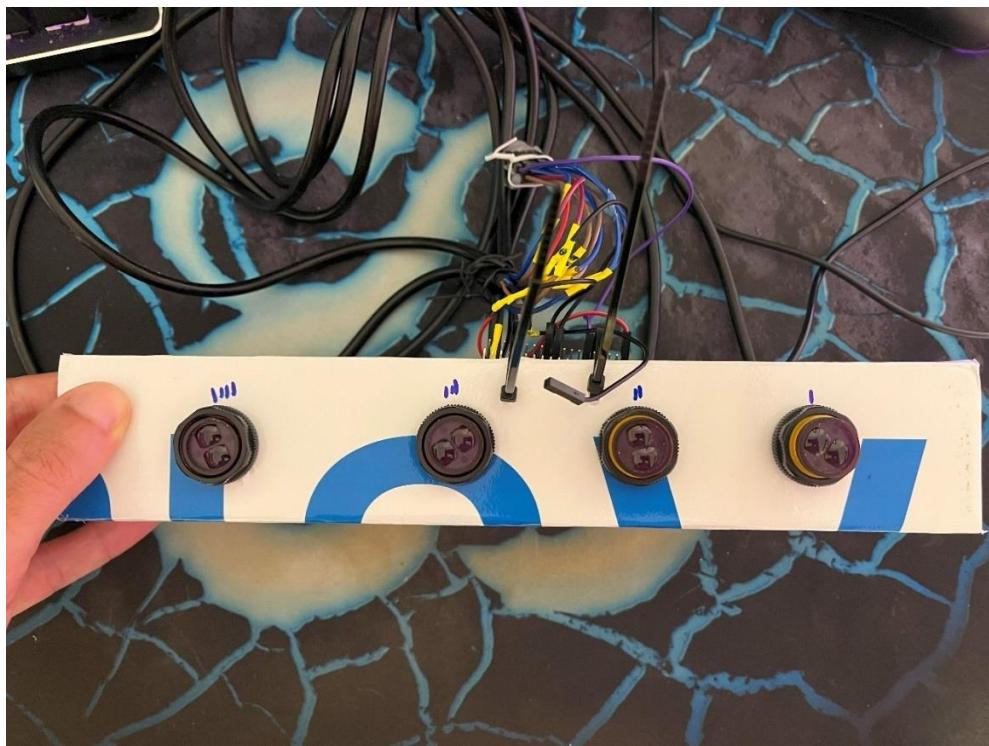


Figura 36. Vista frontal de la segunda estructura de pruebas

Problemas con el driver del ESP32 en Windows

Otro de los problemas que hemos tenido es que es necesario instalar los controladores del puente USB a la UART que hay en la placa del ESP32 [16], de lo contrario, Windows no te reconoce la placa de desarrollo y no puedes cargar el código en ella a través del IDE de Arduino.

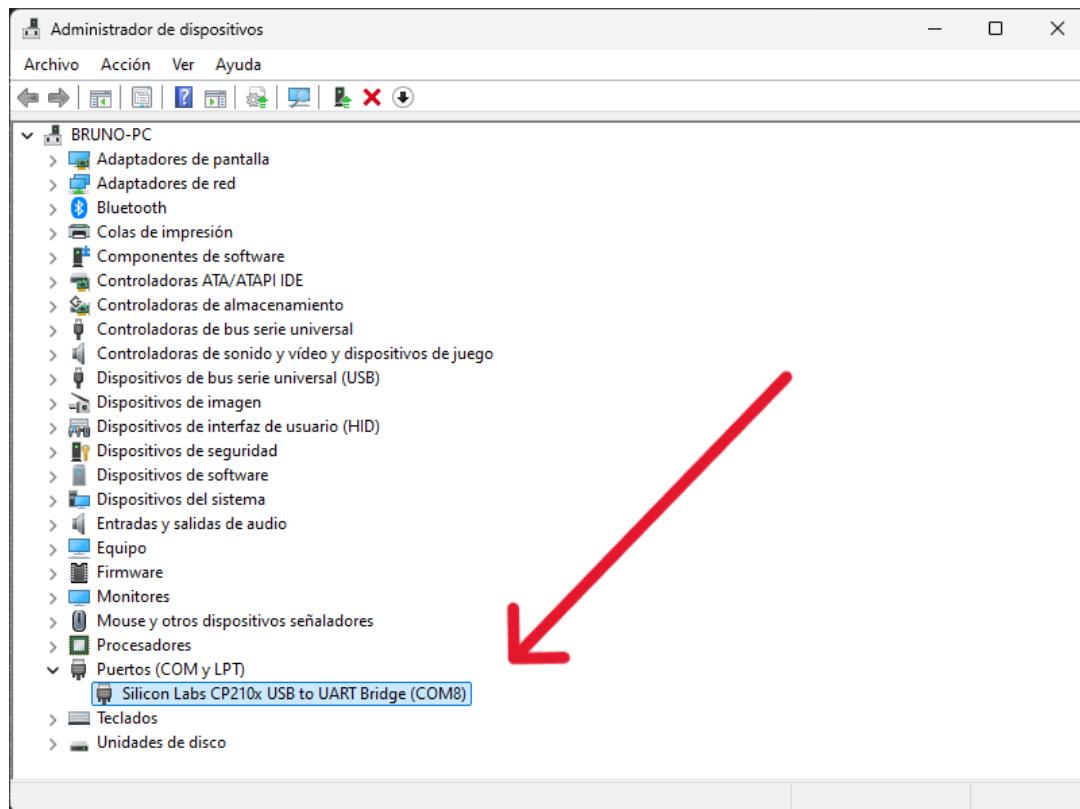


Figura 37. Captura de los drivers del puente USB a UART de los ESP32

Problemas con los cables de prototipado

El último problema que nos ha surgido en este apartado es que los cables no hacían buen contacto o directamente se desconectaban cuando se cerraba el encapsulado. Esto ocurría por la presión que la tapa de la caja ejercía sobre los cables que eran del estilo de la Figura 38:



Figura 38. Cables usados en primera instancia

Para solucionar esto decidimos usar otro tipo de cables que son los que se aprecian en la Figura 24 y soldarlos a los cables de los sensores.

Capítulo 4. Red y servidor Python

En este capítulo se va a describir la infraestructura de red que hemos usado, la arquitectura del servidor edge-computing para gestionar la recepción de mensajes que le mandan los ESP32 y cómo gestiona esas recepciones, es decir, qué acciones realiza cuando le llegan dichos mensajes. Por último, también se verá qué problemas nos han surgido a lo largo del desarrollo.

4.1 Router y configuración de la red Wi-Fi

Para montar nuestra red Wi-Fi hemos utilizado un viejo router del laboratorio, en concreto el Linksys WRT54GL [17], lanzado al mercado en el año 2005 y cuya alta popularidad viene dada por su estabilidad, durabilidad y la posibilidad de personalizar su firmware con software de terceros. Este modelo específico cuenta con un switch de 4 puertos Ethernet de 10/100 Mbps y un punto de acceso con el estándar Wi-Fi 802.11b/g de hasta 54Mbps.



Figura 39. Router Linksys WRT54LG

La configuración que hemos usado para nuestra red inalámbrica es la siguiente:

- **SSID:** TFG Bruno
- **Wireless Channel:** Auto
- **Tipo de contraseña:** WP2 Personal (TKIP + AES)
- **Nombre IP de la red:** 192.168.1.0
- **Máscara de la red:** 255.255.255.0
- **IP del router:** 192.168.1.254
- **Rango de IPs del servidor DHCP:** 192.168.1.1 - 192.168.1.3

En las siguientes figuras se puede ver la configuración que acabamos de ver en el firmware del router:

Wireless Physical Interface wlo

Physical Interface wlo - SSID [TFG Bruno] HWAddr [20:AA:4B:3D:7C:9C]

Wireless Mode	AP	
Wireless Network Mode	Mixed	
Wireless Network Name (SSID)	TFG Bruno	
Wireless Channel	Auto	
Wireless SSID Broadcast	<input checked="" type="radio"/> Enable <input type="radio"/> Disable	
Sensitivity Range (ACK Timing)	2000	(Default: 2000 meters)
Network Configuration	<input type="radio"/> Unbridged <input checked="" type="radio"/> Bridged	

Virtual Interfaces

Add

Save Apply Settings Cancel Changes

Figura 40. SSID y canal de la red inalámbrica

Physical Interface wlo SSID [TFG Bruno] HWAddr [20:AA:4B:3D:7C:9C]

Security Mode	WPA2 Personal	
WPA Algorithms	TKIP+AES	
WPA Shared Key	*****	
Key Renewal Interval (in seconds)	3600	(Default: 3600, Range: 1 - 99999)

Save Apply Settings

Figura 41. Seguridad de la red inalámbrica

Figura 42. Configuración IP del router

MAC Address	Host Name	IP Address
E0:51:D8:12:3F:9D	RAFA-PC	192.168.1.1

Figura 43. Asociación de la MAC del servidor a la primera IP del rango

Como bien podemos ver en la configuración IP del router dejamos la IP 192.168.1.254 para la tarjeta de red del router. Por su parte, el servidor DHCP únicamente da tres IPs, desde la 192.168.1.1 hasta la 19.2.168.1.3, de las cuales vamos a usar la primera para nuestro servidor y las otras dos para los ESP32. La primera la asociamos con la MAC del router para que solo dé esa IP a esa MAC porque en el código de Arduino de las placas hay que especificar la IP del servidor.

4.2 Servidor Python

Este es uno de los apartados más importantes de este TFG. En él vamos a explicar cómo funcionan las diferentes partes del código que tiene nuestro servidor. Estas secciones están separadas en funciones para facilitar su comprensión y mantenimiento. Además, veremos que, tanto el manejo de las conexiones TCP, como la monitorización de los clientes, se realizan en hilos separados para mantener el servidor funcionando de forma fluida y no bloquear el proceso principal.

En la Figura 44 se puede observar un diagrama de bloques que facilita la comprensión del funcionamiento de nuestro servidor:

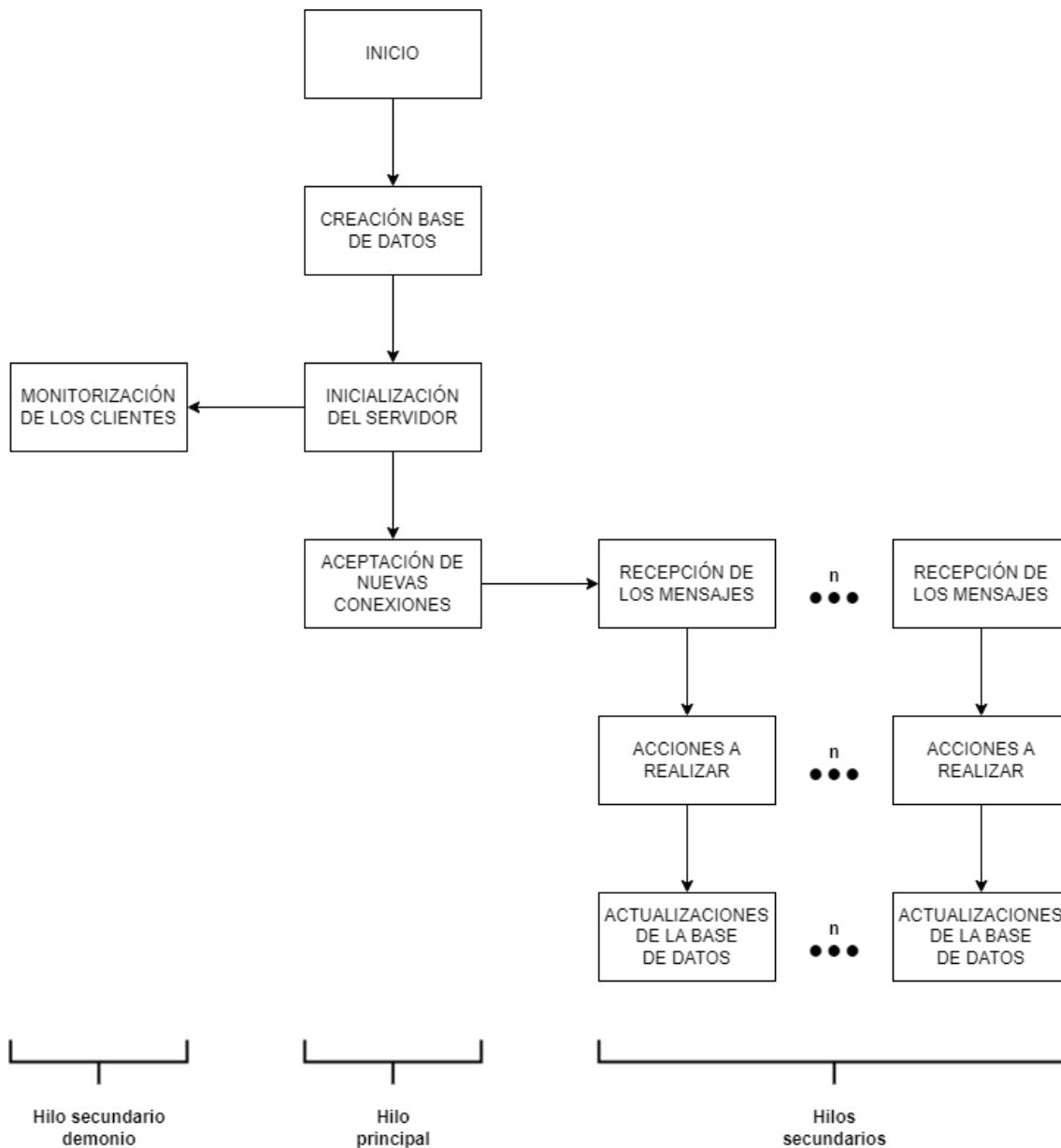


Figura 44. Diagrama de bloques del servidor

4.2.1 Creación de la base de datos

Primeramente, tendremos que crear, si no lo está ya, una base de datos que guarde la información que nos va a importar para después visualizarla en la herramienta de Grafana. Para ello vamos a utilizar la librería SQLite3.

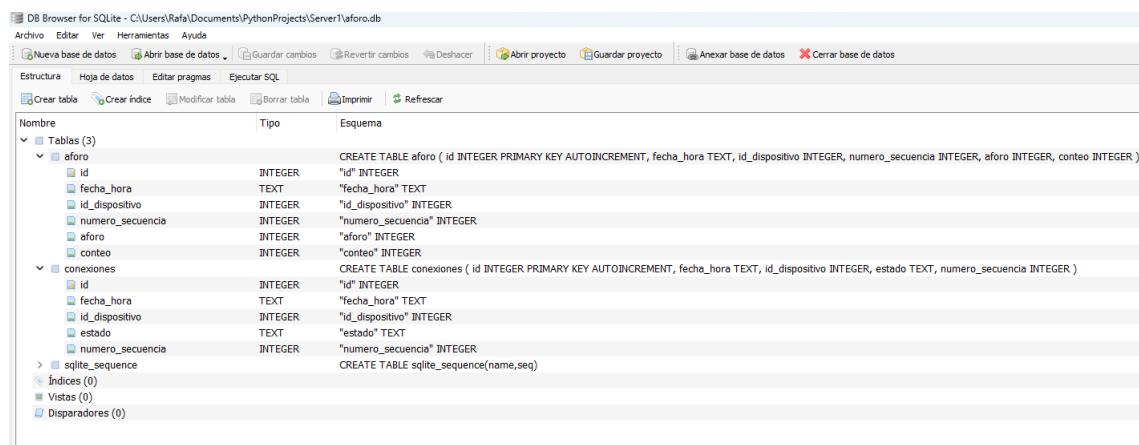
La base de datos que vamos a crear contiene dos tablas, la tabla aforo y la tabla conexiones. La tabla aforo contiene los siguientes campos:

- **Fecha hora**
- **Id dispositivo**
- **Número de secuencia**
- **Aforo**

Mientras que la tabla conexiones tiene los siguiente campos:

- **Fecha hora**
- **Id dispositivo**
- **Estado** (activo o no activo)
- **Número de secuencia**

Como podemos observar en la Figura 45, hemos utilizado el software DB Browser for SQLite [18] para visualizar la base de datos que hemos creado:



The screenshot shows the DB Browser for SQLite interface with the following details:

- Toolbar:** Archivo, Editar, Ver, Herramientas, Ayuda, Nueva base de datos, Abrir base de datos, Guardar cambios, Revertir cambios, Deshacer, Abrir proyecto, Guardar proyecto, Anexar base de datos, Cerrar base de datos.
- Menu Bar:** Estructura, Hoja de datos, Editar pragmas, Ejecutar SQL.
- Table List:** Tablas (3) - aforo, conexiones, sqlite_sequence.
- Table Details:**
 - aforo:** CREATE TABLE aforo (id INTEGER PRIMARY KEY AUTOINCREMENT, fecha_hora TEXT, id_dispositivo INTEGER, numero_secuencia INTEGER, aforo INTEGER, conteo INTEGER)
 - id: INTEGER
 - fecha_hora: TEXT
 - id_dispositivo: INTEGER
 - numero_secuencia: INTEGER
 - aforo: INTEGER
 - conteo: INTEGER
 - conexiones:** CREATE TABLE conexiones (id INTEGER PRIMARY KEY AUTOINCREMENT, fecha_hora TEXT, id_dispositivo INTEGER, estado TEXT, numero_secuencia INTEGER)
 - id: INTEGER
 - fecha_hora: TEXT
 - id_dispositivo: INTEGER
 - estado: TEXT
 - numero_secuencia: INTEGER
- Sequence:** sqlite_sequence (CREATE TABLE sqlite_sequence(name,seq))
- Indices:** (0)
- Vistas:** (0)
- Disparadores:** (0)

Figura 45. Captura de la BB.DD.

4.2.2 Inicialización del servidor

Para inicializar nuestro servidor tendremos que crear una serie de variables que nos van a servir para llevar la cuenta de ciertos parámetros que nos interesan. Esas variables son:

- **Aforo:** se trata de la variable maestra de nuestro sistema. Como su propio nombre indica, nos dice la cantidad de gente que hay dentro de la biblioteca de Antigones. Esta variable se inicializa mediante una función que nos devuelve el último aforo que se ha guardado en el archivo aforo_log.txt.
- **Clientes_conectados:** se trata de un diccionario que asocia en su índice el id_dispositivo (cliente, explicado en el apartado [3.5.2. Conexión de los ESP32 con el servidor](#)) y en su valor si está o no activo.
- **Hora_ultimo_mensaje:** se trata de otro diccionario que asocia en su índice el id_dispositivo (cliente) y en su valor el momento del último mensaje que ha recibido de dicho cliente.
- **Id_disp_num_sec:** se trata de otro diccionario más que asocia en su índice el id_dispositivo y en su valor el último número de secuencia que ha recibido de dicho cliente (el número de secuencia también se encuentra explicado en el apartado [3.5.2. Conexión de los ESP32 con el servidor](#))

Una vez hemos inicializado estas variables, creamos el socket que vamos a poner en escucha, indicándole que se va a tratar de un socket TCP/IPv4 cuya IP va a ser la del PC en el que se ejecute nuestro servidor y el puerto 12345. A partir de ahí, iniciamos el monitor de conexiones (se explica en el apartado [4.2.6 Monitorización de los clientes](#)) y la función que corresponde a la aceptación de conexiones de posibles nuevos clientes.

```
169 #Creamos la base de datos y la tabla aforo si no existe
170 crear_base_de_datos()
171
172 # Inicializamos la variable aforo
173 aforo = obtener_ultimo_aforo()
174 print(f"Aforo inicial: {aforo}")
175 clientes_conectados = {}
176 hora_ultimo_mensaje = {}
177 id_disp_num_sec = {}
178
179 # Inicializamos la IP y puerto del server
180 server_IP = socket.gethostname()
181 print(f"IP del server: {server_IP}")
182 PORT = 12345
183
184 # Creamos el socket del server y lo ponemos en escucha
185 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
186 server_socket.bind((server_IP, PORT))
187 server_socket.listen()
188
189 print("Servidor TCP iniciado.")
190
191 monitor_thread = threading.Thread(target=monitor_conexiones, daemon=True, args=(hora_ultimo_mensaje, clientes_conectados))
192 monitor_thread.start()
193
194 aceptar_conexionesserver_socket)
```

Figura 46. Inicialización del servidor

4.2.3 Aceptación de las conexiones

Como hemos visto anteriormente en la Figura 44 de diagrama de bloques del servidor, necesitamos que varias tareas se ejecuten de manera concurrente. Un hilo (thread) es la unidad más pequeña de procesamiento que puede ser programada y ejecutada. En un programa con varios hilos, cada uno representa una secuencia de instrucciones que puede ejecutarse independientemente de otros hilos. El módulo threading de Python proporciona una API de alto nivel para trabajar con hilos de manera concurrente (realmente no hay concurrencia real, pero si apariencia de que varias tareas se están ejecutando al mismo tiempo).

En nuestro caso, la aceptación de las conexiones de posibles nuevos clientes es el hilo principal del servidor, ya que está siempre pendiente de nuevas posibles conexiones que se puedan realizar. Por cada nueva conexión TCP que se establezca, se crea un nuevo hilo secundario para no entorpecerse así mismo. Estos hilos se explican en el siguiente apartado [4.2.4 Recepción de los mensajes \[19\]](#).

```
163 def aceptar_conexiones(server_socket):
164     while True:
165         client_socket, addr = server_socket.accept()
166
167         client_thread = threading.Thread(target=handle_client, args=(client_socket, addr))
168         client_thread.start()
169
```

Figura 47. Captura del hilo principal del servidor

En la siguiente captura se puede observar el intercambio de mensaje TCP mediante el programa Wireshark [\[20\]](#):

No.	Time	Source	Destination	Protocol	Info
12	4.434337	192.168.1.3	192.168.1.1	TCP	56134 → 12345 [SYN] Seq=0 Win=5760 Len=0 MSS=1436
13	4.434769	192.168.1.1	192.168.1.3	TCP	12345 → 56134 [SYN, ACK] Seq=0 Ack=1 Win=64620 Len=0 MSS=1460
14	4.501022	192.168.1.3	192.168.1.1	TCP	56134 → 12345 [ACK] Seq=1 Ack=1 Win=5760 Len=0
15	4.542171	192.168.1.3	192.168.1.1	TCP	56134 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=5760 Len=16
16	4.590346	192.168.1.1	192.168.1.3	TCP	12345 → 56134 [ACK] Seq=1 Ack=17 Win=64604 Len=0
19	4.750579	192.168.1.2	192.168.1.1	TCP	56574 → 12345 [SYN] Seq=0 Win=5760 Len=0 MSS=1436
20	4.750842	192.168.1.1	192.168.1.2	TCP	12345 → 56574 [SYN, ACK] Seq=0 Ack=1 Win=64620 Len=0 MSS=1460
21	4.758800	192.168.1.2	192.168.1.1	TCP	56574 → 12345 [ACK] Seq=1 Ack=1 Win=5760 Len=0
22	4.760085	192.168.1.2	192.168.1.1	TCP	56574 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=5760 Len=16
23	4.811395	192.168.1.1	192.168.1.2	TCP	12345 → 56574 [ACK] Seq=1 Ack=17 Win=64604 Len=0

Figura 48. Establecimiento de la conexión TCP de los clientes con el servidor

Hemos decidido usar TCP sobre UDP porque es un protocolo de comunicación fiable orientado a conexión que garantiza la entrega correcta y en orden de los datos entre dos puntos de una red. Así mismo, este protocolo también proporciona control de flujo y corrección de errores.

4.2.4 Recepción de mensajes

Como bien hemos dicho en el apartado anterior, se creará un hilo por cada cliente que se conecte a nuestro socket TCP/IP. Este hilo estará a la espera de recibir mensajes del cliente que le corresponda, los descompondrá en sus 3 campos y llamará a la función de contabilizar() (explicada en el siguiente apartado [4.2.5 Acciones a realizar tras la recepción](#)) para cada mensaje que le llegue.

Los ESP32 están programados para que envíen mensajes de uno en uno, pero como explicaremos en el apartado [4.3 Problemas surgidos](#), nos dimos cuenta de que, a veces, nuestro servidor recibe más de un mensaje a la vez, de manera que... ¿cómo podemos saber cuántos mensajes nos llegan? Pues bien, el formato de los mensajes que nos llegan es el siguiente:

ID dispositivo + , + numero de secuencia + , + presentación/variación

Figura 49. Formato de los mensajes (campos)

De manera que tenemos 3 campos, separados por comas, por cada mensaje. Pero si nos llegan más de un mensaje a la vez, no tenemos una coma de separación entre el último campo del mensaje anterior y el primer campo del mensaje siguiente, es decir, se solapan. En el caso de que nos llegaran dos mensajes juntos quedarían tal que así:

ID dispositivo + , + numero de secuencia + , + presentación/variaciónID dispositivo + , + numero de secuencia + , + presentación/variación

Figura 50. Formato de dos mensajes juntos

Si nos damos cuenta, tenemos dos campos extra por cada mensaje de más que nos llega. Por lo tanto, tendríamos el siguiente gráfico:

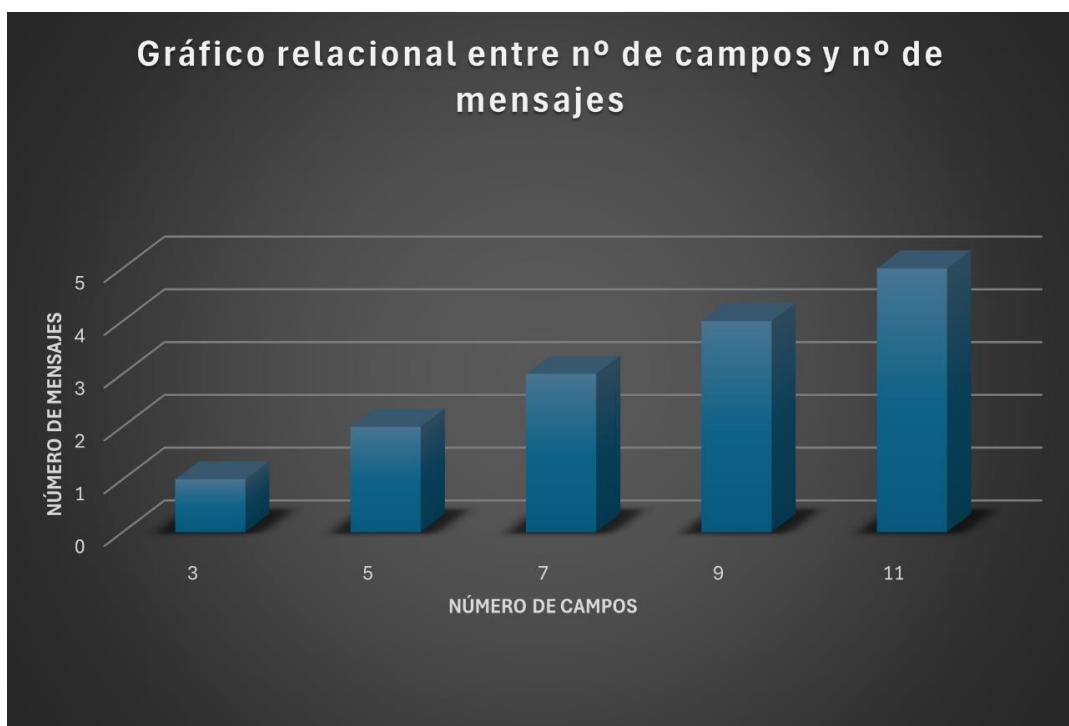


Figura 51. Gráfico relacional entre el nº de campos y el nº de mensajes

Teniendo el gráfico, podemos calcular la ecuación, que tendría la siguiente expresión para $Nº\ de\ campos \geq 3 \in \mathbb{N}$:

$$Nº\ de\ mensajes = \frac{Nº\ de\ campos}{2} - 0,5$$

Figura 52. Ecuación que relaciona el nº de campos y el nº de mensajes

Como es lógico, el $Nº\ de\ mensajes \geq 1 \in \mathbb{N}$.

De esta manera ya sabemos cómo podemos separar varios mensajes si es que nos llegan más de uno juntos. En la siguiente captura se puede apreciar la implementación del código de lo que se ha explicado en este apartado:

```

76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
1 usage
def handle_client(client_socket, addr):
    while True:
        try:
            # Recibimos datos del cliente
            fecha_hora = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            data = client_socket.recv(1024).decode()
            print(f"{fecha_hora} - {data}")

            numero_campos = len(data.split(','))
            numero_mensajes = math.trunc(numero_campos / 2)

            for i in range(numero_mensajes):
                id_dispositivo_prov = data.split(',')[0 + 2 * i]
                cadena1 = ""
                for char in id_dispositivo_prov:
                    if char.isdigit():
                        cadena1 += char
                id_dispositivo = int(cadena1)
                numero_secuencia_prov = data.split(',')[1 + 2 * i]
                cadena2 = ""
                for char in numero_secuencia_prov:
                    if char.isdigit():
                        cadena2 += char
                numero_secuencia = int(cadena2)
                presentacion_variacion_prov = data.split(',')[2 + 2 * i]
                cadena3 = ""
                for char in presentacion_variacion_prov:
                    if char.isalpha():
                        cadena3 += char
                presentacion_variacion = str(cadena3)

                contabilizar(id_dispositivo, numero_secuencia, presentacion_variacion, addr)

        except ValueError as ve:
            print(f"Datos inválidos recibidos. Error: {ve}")
            break

```

Figura 53. Código correspondiente a la recepción de mensajes

Además, como podemos observar hay manejo de errores en el caso de que no lleguen los datos en su correspondiente formato sin que el servidor se bloquee.

4.2.5 Acciones a realizar tras la recepción

Una vez que hemos explicado la recepción de los mensajes, pasamos a describir qué acciones realiza el servidor con los datos que acaba de recibir.

Para empezar, lo primero que hace es actualizar la variable diccionario hora_ultimo_mensaje al momento en el que ha recibido el mensaje y la variable diccionario clientes_conectados (explicadas en el apartado [4.2.2 Inicialización del servidor](#)). Posteriormente, se actualiza la tabla conexiones de la base de datos indicando que el cliente con cierta id_dispositivo está activo, pues acaba de recibir un mensaje de él. También se muestra el número de secuencia de este mensaje.

Más tarde es donde se distingue si el mensaje es de presentación o si bien se trata de un mensaje debido al flujo de una persona por los sensores o un “keep-alive”. Si se trata de un mensaje de presentación (como se muestra en la captura de wireshark de la Figura 54), el campo presentación/variación será igual a “presentación” y se mostrará en pantalla que el cliente con cierta id_dispositivo se acaba de conectar.

0000	e0 51 d8 12 3f 9d 7c df a1 e1 63 d8 08 00 45 00	Q ? c E
0010	00 38 00 06 00 00 40 06 f7 66 c0 a8 01 02 c0 a8	8 @ f
0020	01 01 ca af 30 39 70 93 b0 61 ff 1e 5f 3a 50 18	09p a _;P
0030	16 80 b6 cb 00 00 31 2c 31 2c 70 72 65 73 65 6e	1, 1,presen
0040	74 61 63 69 6f 6e	tacion

Figura 54. Detalle captura Wireshark presentación

En cambio, si se trata de un mensaje debido al flujo de una persona o a un “keep-alive” (ya que los ESP32 mandan mensajes cada minuto, aunque no haya pasado nadie por los sensores) el campo presentación/variación será igual a “incrementa”, “decrementa” o “nada”, dependiendo de lo que haya ocurrido. A su vez, se actualiza o no la variable aforo y el archivo aforo_log.txt y la tabla aforo de la base de datos. Las Figuras 55, 56 y 57 muestran capturas en Wireshark de mensajes correspondientes a “incrementa”, “decrementa” y “nada”, respectivamente.

0000	e0 51 d8 12 3f 9d 7c df a1 e1 63 d8 08 00 45 00	Q ? c E
0010	00 36 00 09 00 00 40 06 f7 65 c0 a8 01 02 c0 a8	6 @ e
0020	01 01 ca af 30 39 70 93 b0 81 ff 1e 5f 3a 50 18	09p;P
0030	16 80 2a 1d 00 00 31 2c 34 2c 69 6e 63 72 65 6d	1, 4,increm
0040	65 6e 74 61	enta

Figura 55. Detalle captura Wireshark incrementa

0000	e0 51 d8 12 3f 9d 7c df a1 e1 63 d8 08 00 45 00	Q ? c E
0010	00 36 00 15 00 00 40 06 f7 59 c0 a8 01 02 c0 a8	6 @ Y
0020	01 01 ca b0 30 39 9f 37 75 a6 fc 04 c7 77 50 18	09 7 u wP
0030	16 80 d1 38 00 00 31 2c 39 2c 64 65 63 72 65 6d	1, 9,decrem
0040	65 6e 74 61	enta

Figura 56. Detalle captura Wireshark decrementa

0000	e0 51 d8 12 3f 9d b8 f0 09 93 d2 e4 08 00 45 00	Q ? c E
0010	00 30 00 13 00 00 40 06 f7 60 c0 a8 01 03 c0 a8	0 @ ^
0020	01 01 c0 dd 30 39 19 9c 3d 75 af c5 f8 8e 50 18	09 =u P
0030	16 80 e8 57 00 00 32 2c 38 2c 6e 61 64 61	W 2, 8,nada

Figura 57. Detalle captura Wireshark nada

Por último, se comprueba si es la primera vez que se ha conectado el cliente cuyo mensaje acabamos de recibir. Si no lo es, se comprueba entonces el número de secuencia del último mensaje que habíamos recibido de éste con el del mensaje actual, si el del mensaje actual es inferior entonces habremos detectado un reinicio en el ESP32, que puede ser debido a numerosos factores, como por ejemplo un corte de la alimentación.

En Figura 58 se puede apreciar el diagrama de bloques de las acciones que realiza el servidor una vez ha recibido un mensaje:

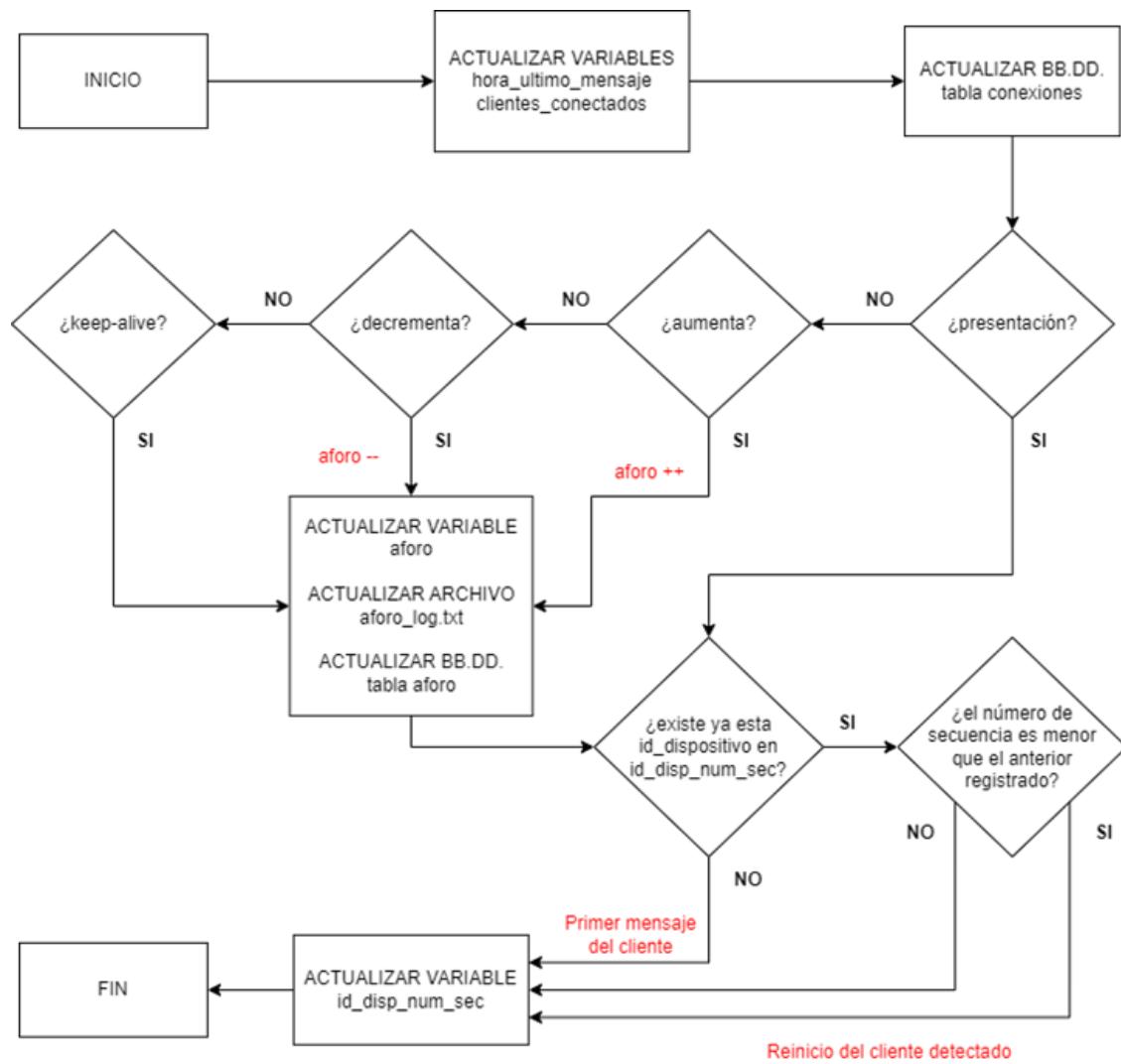


Figura 58. Diagrama de bloques de las acciones a realizar una vez recibido un mensaje

En la siguiente figura se muestra la implementación del código de este apartado:

```

1 usage
110 def contabilizar(id_dispositivo, numero_secuencia, presentacion_variacion, addr):
111     global aforo
112
113     fecha_hora = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
114     hora_ultimo_mensaje[id_dispositivo] = time.time()
115     # print(f"{current_time} - {hora_ultimo_mensaje}")
116     clientes_conectados[id_dispositivo] = "Activo"
117
118     actualizar_conexiones_bd(fecha_hora, id_dispositivo, estado="Activo", numero_secuencia)
119
120     if presentacion_variacion == "presentacion":
121         print(f'{fecha_hora} - Conexión establecida con el cliente {id_dispositivo} con IP y puerto: {addr}. ACTIVE THREADS: {threading.active_count() - 1}')
122         print(f'{fecha_hora} - {clientes_conectados}')
123
124     # Escribimos en el archivo el aforo y la hora a la que llega
125     elif presentacion_variacion in ["incrementa", "decrementa", "nada"]:
126         if presentacion_variacion == "incrementa":
127             aforo += 1
128         elif presentacion_variacion == "decrementa" and aforo > 0:
129             aforo -= 1
130         with open("aforo_log.txt", "a") as archivo:
131             archivo.write(f'{fecha_hora} - Dispositivo: {id_dispositivo}, Secuencia: {numero_secuencia}, Aforo: {aforo}\n')
132
133         actualizar_aforo_bd(fecha_hora, id_dispositivo, numero_secuencia, aforo)
134
135         # Mostrar estado actual del aforo con hora a la que llega
136         print(f'{fecha_hora} - Dispositivo: {id_dispositivo}, Secuencia: {numero_secuencia}, Aforo: {aforo}')
137         # print(f'{current_time} - {clientes_conectados}')
138
139     if id_dispositivo in id_disp_num_sec:
140         if numero_secuencia < id_disp_num_sec[id_dispositivo]:
141             print(f'{fecha_hora} - Reinicio detectado en el cliente {id_dispositivo}')
142         else:
143             print(f'{fecha_hora} - Primer mensaje del cliente {id_dispositivo}')
144
145     id_disp_num_sec[id_dispositivo] = numero_secuencia
146     # print(f'{current_time} - {id_disp_num_sec}')
147     # print(f'{current_time} - {id_disp_num_sec[id_dispositivo]}')



```

Figura 59. Captura del hilo de acciones realizar tras la recepción

Las variables hora_ultimo_mensaje y clientes_conectados serán clave a la hora de monitorizar el estado de las conexiones por parte de los clientes como bien veremos en el siguiente apartado:

4.2.6 Monitorización de los clientes

El monitor de conexiones se encarga de comprobar si los clientes siguen conectados cada 30 segundos. Éste muestra la lista de clientes conectados y su estado (si están o no activos). También muestra el tiempo transcurrido desde el último mensaje que hemos recibido de cada cliente activo.

Posteriormente, también se ocupa de comprobar si los clientes siguen o no activos. ¿Cómo sabe si los clientes están o no activos? Pues bien, ya que los ESP32 están programados para enviar un mensaje cada vez que pase alguien o cada minuto, aunque no pase nadie, si han transcurrido más de 65 segundos (a modo de timeout), se entiende que la conexión con el ESP32 se ha perdido por la circunstancia que sea y este cliente pasa a figurar como no activo.

Por último, pero no menos importante, se actualiza la tabla de conexiones de la base de datos. Este funcionamiento se ve reflejado en la siguiente figura de diagrama de bloques:

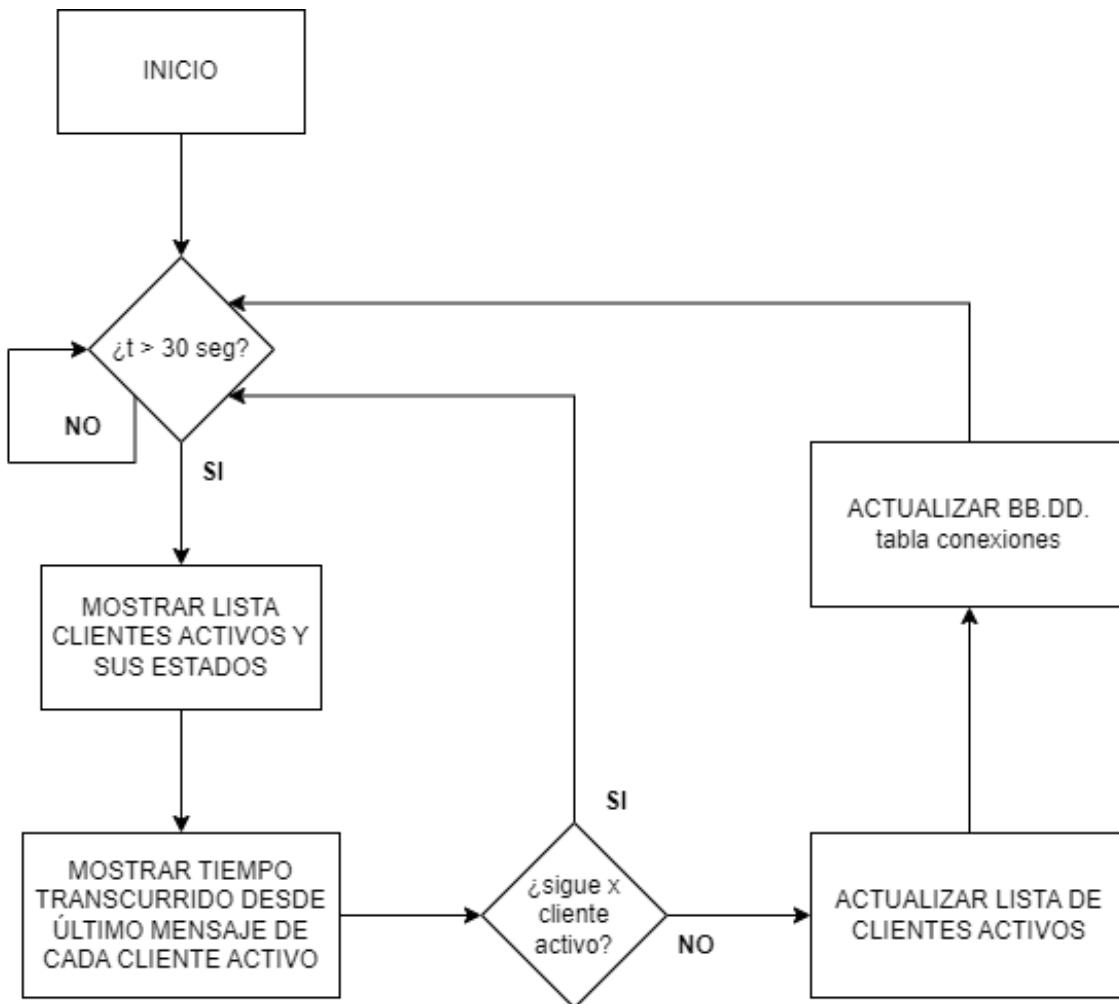


Figura 60. Diagrama de bloques del monitor de conexiones

El monitor de conexiones se trata de un hilo secundario demonio (o Daemon). Es decir, si el hilo principal del programa termina, el programa se cerrará sin importar si los hilos demonio siguen activos o no. Se ha diseñado así porque pensamos que el monitor de conexiones se trata de una tarea secundaria que no es crucial para la finalización del programa.

```

1 usage
149 def monitor_conexiones(hora_ultimo_mensaje, clientes_conectados):
150     while True:
151         time.sleep(30)
152         fecha_hora = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
153         print(f'{fecha_hora} - {clientes_conectados}')
154         for id_dispositivo, tiempo in hora_ultimo_mensaje.items():
155             if clientes_conectados.get(id_dispositivo) == "Activo":
156                 print(f'{fecha_hora} - {time.time() - tiempo} TIEMPO DESDE EL ÚLTIMO MENSAJE DE DISPOSITIVO: {id_dispositivo}')
157                 if time.time() - tiempo > 65:
158                     print(f'{fecha_hora} - Cliente {id_dispositivo} desconectado por tiempo de espera.')
159                     clientes_conectados[id_dispositivo] = "No activo"
160                     actualizar_conexiones_bd(fecha_hora, id_dispositivo, estado="No Activo", id_disp_num_sec[id_dispositivo])
161                     print(f"Estado de las conexiones: {clientes_conectados}")
162
  
```

Figura 61. Implementación del monitor de conexiones

4.2.7 Actualizaciones de la base de datos

Estas funciones se encargan de actualizar las diferentes variables en nuestra base de datos tanto para la tabla “aforo” como para la tabla “conexiones” para que así se puedan ver reflejadas en los diferentes gráficos, tablas y paneles de Grafana. La implementación de ambas se muestra en las siguientes Figuras 62 y 63:

```
1 usage
32  def actualizar_aforo_bd(fecha_hora, id_dispositivo, numero_secuencia, aforo):
33      conn = sqlite3.connect('aforo.db')
34      cursor = conn.cursor()
35      cursor.execute(_sql: '''
36          INSERT INTO aforo (fecha_hora, id_dispositivo, numero_secuencia, aforo)
37          VALUES (?, ?, ?, ?)
38      ''', _parameters: (fecha_hora, id_dispositivo, numero_secuencia, aforo))
39      conn.commit()
40      conn.close()
41
```

Figura 62. Implementación para actualizar la tabla aforo

```
2 usages
42  def actualizar_conexiones_bd(fecha_hora, id_dispositivo, estado, numero_secuencia):
43      conn = sqlite3.connect('aforo.db')
44      cursor = conn.cursor()
45      cursor.execute(_sql: '''
46          INSERT INTO conexiones (fecha_hora, id_dispositivo, estado, numero_secuencia)
47          VALUES (?, ?, ?, ?)
48      ''', _parameters: (fecha_hora, id_dispositivo, estado, numero_secuencia))
49      conn.commit()
50      conn.close()
51
```

Figura 63. Implementación para actualizar la tabla conexiones

4.3 Problemas surgidos

Durante el desarrollo de este apartado nos hemos encontrado con dos problemas principales.

Problemas con la recepción de más de un mensaje simultáneamente

El primero de ellos tiene que ver con que, en una versión prematura de nuestro servidor (se puede observar el código en la Figura 64), el apartado [4.2.4 Recepción de los mensajes](#) y [4.2.5 Acciones a realizar tras la recepción](#) se encontraban en una sola función, ya que el servidor estaba pensado para que, únicamente se recibieran mensajes de uno en uno. Pero nos dimos cuenta de que, en las pruebas preliminares, en ciertas ocasiones, bastantes esporádicas, se recibían más de un mensaje a la vez, entendimos que, a consecuencia de algún fallo de procesamiento de los ESP32. De manera que, cuando recibíamos más de un mensaje a la vez, nuestro servidor no era capaz de descomponer los diferentes campos y contabilizar correctamente las entradas o salidas de personas de la biblioteca, por lo tanto, se optó por dividir la función en dos.

```
20 def handle_client(client_socket, addr):
21
22     global aforo
23     while True:
24         try:
25             #Recibimos datos del cliente
26             current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
27             data = client_socket.recv(1024).decode()
28             print(f"{current_time} - {data}")
29
30             #Analizamos datos recibidos
31             id_dispositivo, numero_secuencia, presentacion_variacion = data.split(',')
32             id_dispositivo = int(id_dispositivo)
33             numero_secuencia = int(numero_secuencia)
```

Figura 64. Versión prematura del código de recepción de los mensajes

La primera de ellas, la del apartado [4.2.4 Recepción de los mensajes](#) se encargaría de recibir uno o varios mensajes, de separar estos mensajes y de descomponer los campos correspondientes de cada mensaje. Y, una vez los campos de cada mensaje están descompuestos, entonces se llama a la segunda función. Esta segunda función se encarga de actualizar el aforo y el estado de las conexiones de los clientes como bien se ha explicado en el apartado [4.2.5 Acciones a realizar tras la recepción](#).

```
2024-06-25 13:26:47 - Dispositivo: 1, Secuencia: 244, Aforo: 3
2024-06-25 13:26:51 - Dispositivo: 1, Secuencia: 245, Aforo: 2
2024-06-25 13:26:53 - Dispositivo: 1, Secuencia: 246, Aforo: 3
2024-06-25 13:26:57 - Dispositivo: 1, Secuencia: 247, Aforo: 2
2024-06-25 13:27:00 - Dispositivo: 1, Secuencia: 248, Aforo: 3
2024-06-25 13:27:03 - Dispositivo: 1, Secuencia: 249, Aforo: 2
2024-06-25 13:27:05 - Dispositivo: 1, Secuencia: 250, Aforo: 3
2024-06-25 13:27:18 - Dispositivo: 1, Secuencia: 255, Aforo: 2
2024-06-25 13:27:19 - Dispositivo: 1, Secuencia: 256, Aforo: 3
2024-06-25 13:27:21 - Dispositivo: 1, Secuencia: 257, Aforo: 2
2024-06-25 13:27:23 - Dispositivo: 1, Secuencia: 258, Aforo: 3
2024-06-25 13:27:26 - Dispositivo: 1, Secuencia: 259, Aforo: 2
2024-06-25 13:27:28 - Dispositivo: 1, Secuencia: 260, Aforo: 3
2024-06-25 13:27:30 - Dispositivo: 1, Secuencia: 261, Aforo: 2
```

Figura 65. Error registrado en el archivo aforo_log.txt

Como bien vemos en la Figura 65, en el archivo aforo_log.txt faltan los mensajes con número de secuencia 251 al 254, esto es debido a que se recibieron tres mensajes a la vez. El 251, 252 y 253 corresponderían a el número de secuencia de estos mensajes, mientras que el 254 correspondería al mensaje de presentación que no se registra en el archivo aforo_log.txt.

Problema a la hora de recibir y descomponer un mensaje debido a un \n

El segundo problema que tuvimos en el servidor fue a la hora recibir los mensajes, ya que nuestro código saltaba a la excepción de que los datos recibidos eran inválidos. Este problema nos llevó varios dolores de cabeza, ya que no averiguábamos a qué se debía y tardamos varias semanas en descubrirlo. El programa funcionaba, pero saltaba a la excepción y mostraba un error cada vez que se recibía un mensaje. Esto era debido a que cada vez que recibía un mensaje en realidad recibía dos, el primer mensaje era correcto, pero el segundo aparecía un espacio en blanco y por eso saltaba a la excepción, pero, no sabíamos qué estaba recibiendo.

Finalmente, depurando el código nos dimos cuenta de que ese espacio en blanco se trataba de un \n. Nos dimos cuenta de ello porque cambiamos las estructuras de datos de tuplas a listas en Python, y de esa manera, sí que nos aparecía en consola ese \n, en lugar de un espacio en blanco.

En realidad, el error, no estaba en el servidor, sino en el código de Arduino, que habíamos puesto println (que te incluye ese \n) en lugar de únicamente print.

```
203 void enviarDatos(){
204
205     numero_secuencia++;
206
207     String mensaje = String(id_dispositivo) + "," + String(numero_secuencia) + "," + String(variacion);
208
209     Serial.print("Enviando datos al servidor: ");
210     Serial.println(mensaje);
211
212     client.print(mensaje);
213 }
```

Figura 66. Lugar en el que se encontraba el error provocado por \n

Capítulo 5. Presentación de los datos

En este capítulo se van a describir las herramientas que hemos usado para presentar mediante una interfaz gráfica de usuario, ya sean gráficos, tablas o paneles, los datos de importancia relevante de nuestro sistemas de conteo de personas. Para ello vamos a usar la plataforma de Grafana.

5.1 Plugin SQLite para Grafana y fuente de datos

Una vez tenemos instalado Grafana en nuestro equipo, procedemos a instalar el plugin de SQLite para Grafana, ya que por defecto no hay soporte nativo. De esta manera nuestra base de datos podrá ser la fuente de datos de nuestros paneles.

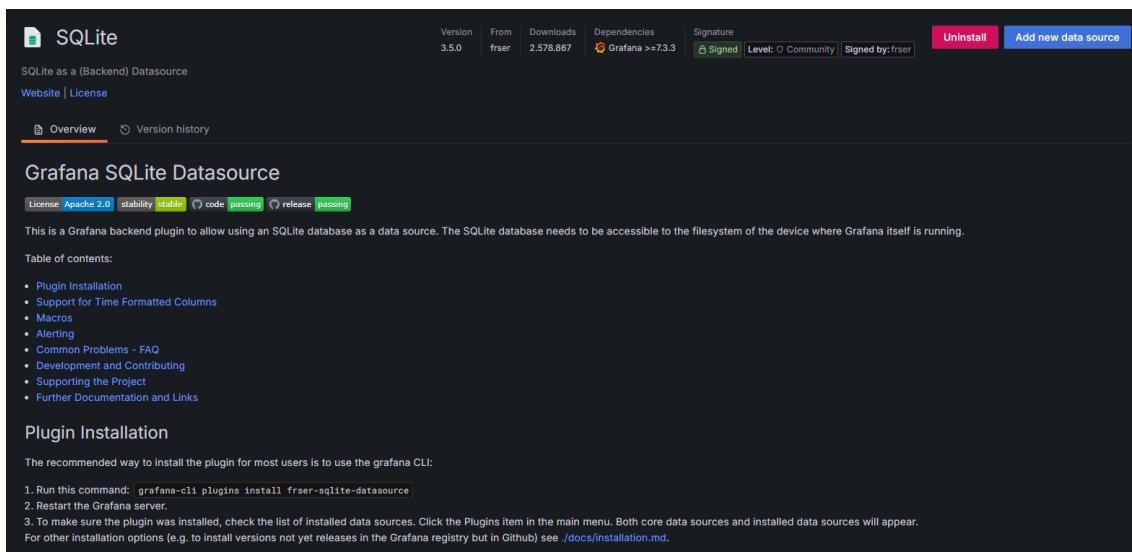


Figura 67. Captura del plugin SQLite

Con el plugin instalado, ahora añadiremos nuestra base de datos como fuente de los datos de los paneles, gráficos y tablas que configuraremos más adelante. Tendremos que seleccionar el archivo de nuestra base de datos para ello.

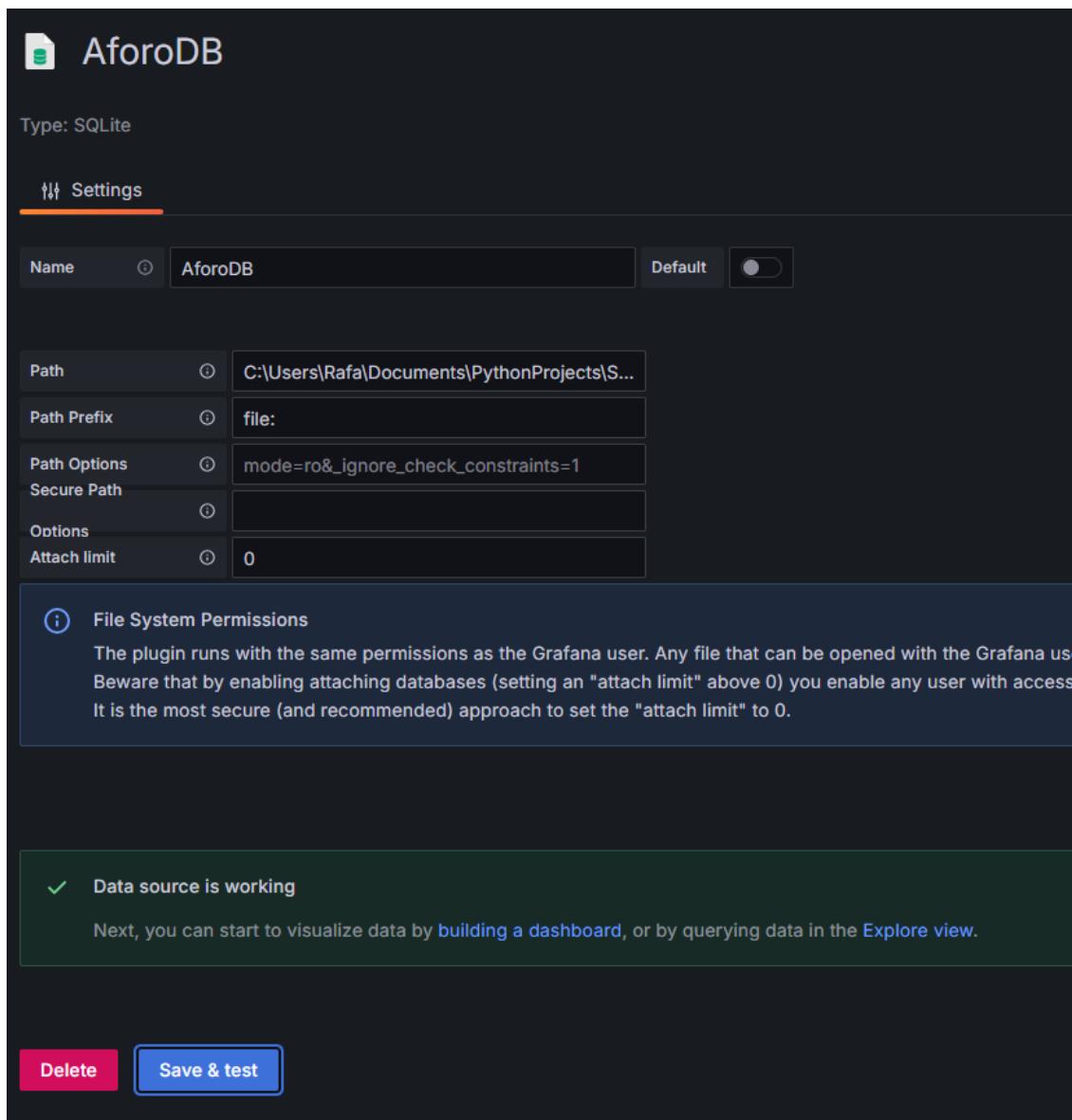


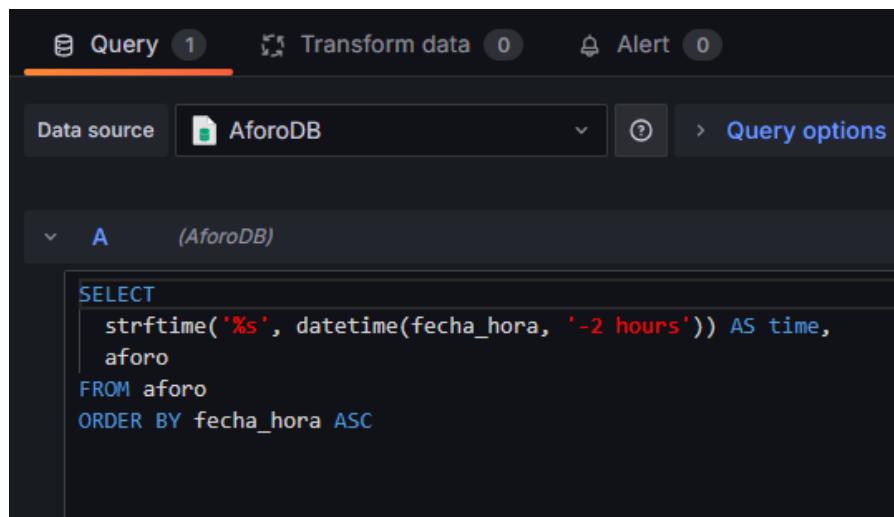
Figura 68. Fuente de datos de Grafana

5.2 Paneles de Grafana

Una vez tenemos el plugin de SQLite y nuestra fuente de datos, podremos crear los paneles que van a mostrar la información que queramos casi en tiempo real (casi porque el intervalo de actualización más rápido de los paneles que nos permite Grafana son 5 segundos).

Ahora vamos a explicar cada uno de los paneles para finalmente mostrar la visualización que tendríamos de todos ellos.

El primer panel corresponde con el gráfico del aforo. La consulta SQL que necesitamos para que funcione correctamente se muestra en la siguiente figura:

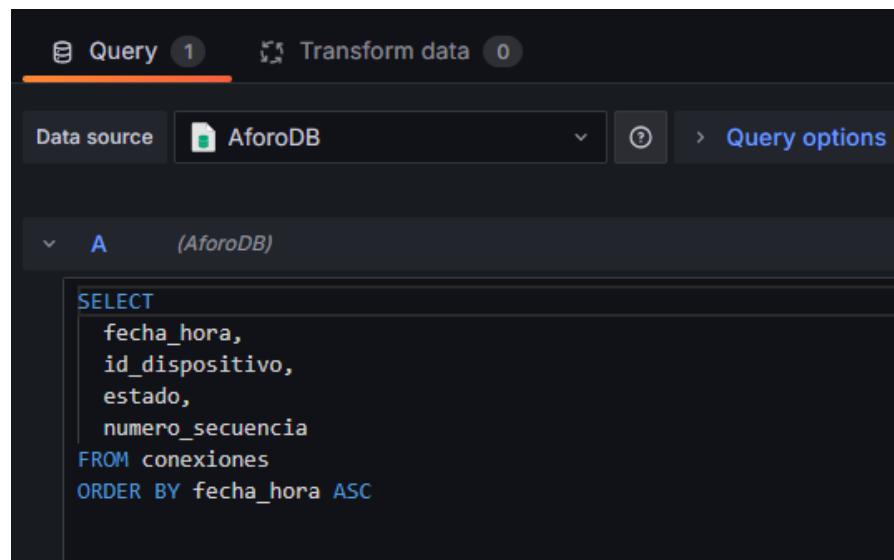


The screenshot shows the Grafana Query editor interface. At the top, there are three tabs: 'Query' (which is selected, indicated by an orange underline), 'Transform data', and 'Alert'. Below these are buttons for 'Data source' (set to 'AforoDB'), 'Query options', and a help icon. The main area contains a query labeled 'A (AforoDB)'. The SQL code is:

```
SELECT
    strftime('%s', datetime(fecha_hora, '-2 hours')) AS time,
    aforo
FROM aforo
ORDER BY fecha_hora ASC
```

Figura 69. Consulta SQL gráfico aforo

El segundo panel se trata una tabla llamada Conexiones en la que se muestran el estado y el número de secuencia de cada dispositivo en cierto momento. Su consulta SQL se muestra en la siguiente figura:

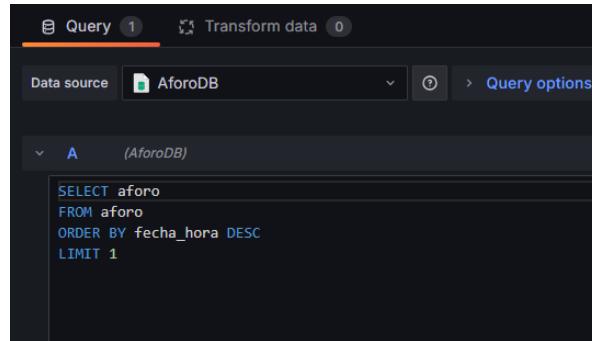


The screenshot shows the Grafana Query editor interface. At the top, there are three tabs: 'Query' (selected), 'Transform data', and 'Alert'. Below these are buttons for 'Data source' (set to 'AforoDB'), 'Query options', and a help icon. The main area contains a query labeled 'A (AforoDB)'. The SQL code is:

```
SELECT
    fecha_hora,
    id_dispositivo,
    estado,
    numero_secuencia
FROM conexiones
ORDER BY fecha_hora ASC
```

Figura 70. Consulta SQL tabla conexiones

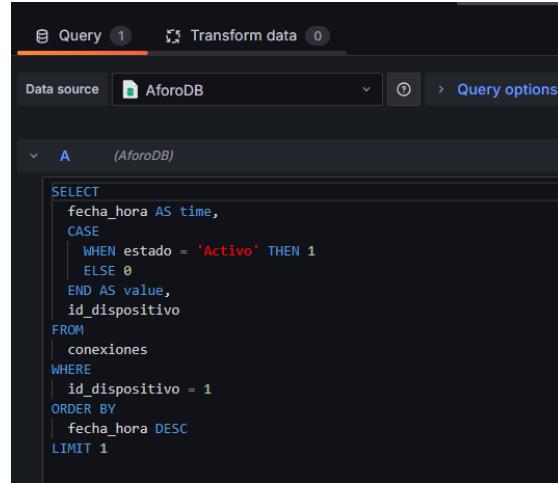
El siguiente panel muestra en un número en grande el aforo, se trata de un panel redundante, puesto que el aforo ya lo podemos ver en el gráfico correspondiente, pero lo hemos añadido para que sea más fácil de visualizarlo. Su consulta SQL se muestra en la siguiente figura:



```
Query 1 Transform data 0
Data source AforoDB
A (AforoDB)
SELECT aforo
FROM aforo
ORDER BY fecha_hora DESC
LIMIT 1
```

Figura 71. Consulta SQL panel aforo

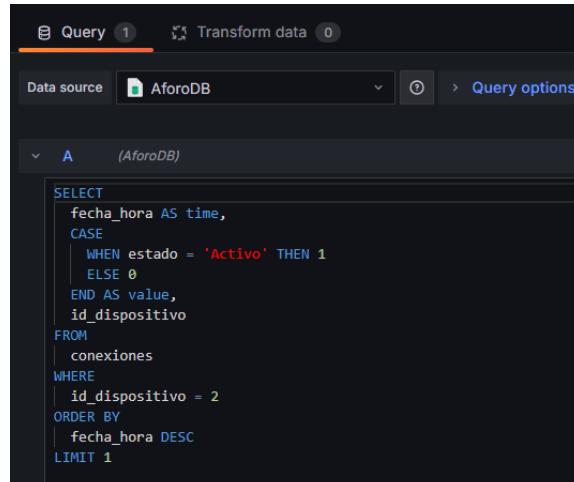
El cuarto panel muestra si el primer dispositivo se encuentra o no activo. Su consulta SQL se muestra en la siguiente figura:



```
Query 1 Transform data 0
Data source AforoDB
A (AforoDB)
SELECT
    fecha_hora AS time,
    CASE
        WHEN estado = 'Activo' THEN 1
        ELSE 0
    END AS value,
    id_dispositivo
FROM
    conexiones
WHERE
    id_dispositivo = 1
ORDER BY
    fecha_hora DESC
LIMIT 1
```

Figura 72. Consulta SQL panel estado dispositivo 1

El quinto y último panel muestra si el segundo dispositivo se encuentra o no activo. Su consulta SQL se muestra en la siguiente figura:



```
Query 1 Transform data 0
Data source AforoDB
A (AforoDB)
SELECT
    fecha_hora AS time,
    CASE
        WHEN estado = 'Activo' THEN 1
        ELSE 0
    END AS value,
    id_dispositivo
FROM
    conexiones
WHERE
    id_dispositivo = 2
ORDER BY
    fecha_hora DESC
LIMIT 1
```

Figura 73. Consulta SQL panel estado dispositivo 2

Finalmente, la vista de todos los paneles quedaría como se muestra en la siguiente figura. De esta manera se puede visualizar rápidamente el número de personas que hay en la biblioteca de Antigones o detectar si hay algún dispositivo que se ha desconectado.



Figura 74. Vista de todos los paneles de Grafana

También cabría mencionar que la configuración de los paneles se puede cambiar a gusto de cada uno gracias a las múltiples opciones que nos ofrece la interfaz de Grafana. Además, también podemos movernos en el eje del tiempo y que los gráficos se muestren desde los últimos 5, 15 o 30 minutos y desde varias horas, días o desde el inicio de la fuente de datos, en nuestro caso la base de datos.

5.3 Archivos CSV

Grafana también nos ofrece la descarga de archivos CSV que pueden ser compatibles con la aplicación Excel donde a partir de la cual podremos calcular las diferentes estadísticas que nos interesen como la media de personas que hay en la biblioteca a lo largo de un día o periodo de tiempo en concreto o, por ejemplo, el porcentaje de tiempo que los clientes se encuentran conectados o desconectados al servidor.

	A	B
1	time	aforo
2	01/08/2024 14:12	1
3	01/08/2024 14:12	2
4	01/08/2024 14:13	2
5	01/08/2024 14:13	3
6	01/08/2024 14:13	4
7	01/08/2024 14:13	3
8	01/08/2024 14:14	3
9	01/08/2024 14:15	3
10	01/08/2024 14:15	4
11	01/08/2024 14:15	4
12	01/08/2024 14:15	3
13	01/08/2024 14:15	5
14	01/08/2024 14:15	4
15	01/08/2024 14:15	3
16	01/08/2024 14:15	4
17	01/08/2024 14:15	5
18	01/08/2024 14:16	6
19	01/08/2024 14:16	7
20	01/08/2024 14:16	8
21	01/08/2024 14:16	7
22	01/08/2024 14:16	8
23	01/08/2024 14:16	7
24	01/08/2024 14:16	6
25	01/08/2024 14:16	7
26	01/08/2024 14:17	8
27	01/08/2024 14:17	9
28	01/08/2024 14:17	8
29	01/08/2024 14:17	9
30	01/08/2024 14:17	10
31	01/08/2024 14:17	9
32	01/08/2024 14:17	8

Figura 75. Captura del archivo CSV

Capítulo 6. Pruebas y resultados

En este capítulo se va a diferenciar entre dos tipos de pruebas: las pruebas preliminares y las pruebas definitivas de testeo y validación del sistema.

6.1 Pruebas preliminares

Con respecto a las pruebas preliminares, son aquellas que son necesarias en muchos de los pasos del desarrollo del proyecto para poder continuar con las siguientes fases. De estas pruebas no se han recogido datos puesto que únicamente se han realizado para comprobar el correcto funcionamiento de alguna parte del proyecto.

Estas pruebas abarcan desde los primeros testeos con un único sensor con la placa Arduino Uno, hasta las pruebas de testeo que se han realizado sin recoger datos, pasando por las pruebas que se han realizado con dos sensores tanto en la placa Arduino Uno como en los ESP32 para comprobar que la conexión de los sensores con las placas fuera satisfactoria. También hay que añadir las pruebas que se han realizado para comprobar el correcto funcionamiento del algoritmo de conteo y del LED RGB de la placa ESP32-S3-DevkitC-1, además de la conexión de ambos ESP32 con el servidor, de la comprobación de la correcta recepción de datos en el servidor y de las acciones que toma en base a los mensajes que recibe. Por último, en esta parte también entran las pruebas que se han realizado para que la interfaz gráfica de Grafana funcione correctamente con los datos que iba recogiendo de nuestra base de datos.

6.2 Pruebas de testeo y validación del sistema

Antes de presentar los datos con sus respectivas estadísticas, mencionaremos la metodología que hemos llevado a cabo para la realización de las pruebas de testeo y validación del sistema.

Hemos colocado nuestra caja en los arcos antihurto de la biblioteca tal y como se muestra en la Figura 76 y 77. Los sensores están configurados para detectar hasta 80 cm, ya que la distancia entre los arcos es de 97 cm. No hay que preocuparse por esos 17 cm que nos quedan sin cubrir puesto que una persona es más ancha y los sensores siempre detectarían a una persona que pase entre los arcos. Con respecto a la altura, están situados aproximadamente a unos 130 cm del suelo (ya que ambos sensores no están a la misma altura puesto que la caja se encuentra un poquito inclinada). Nos parece una altura adecuada, ya que la gran mayoría de personas miden más de la altura a la que están situados nuestros sensores.

También se puso en marcha nuestro router, necesario para que nuestra red Wi-Fi funcione y haya comunicación entre los ESP32 y el servidor.

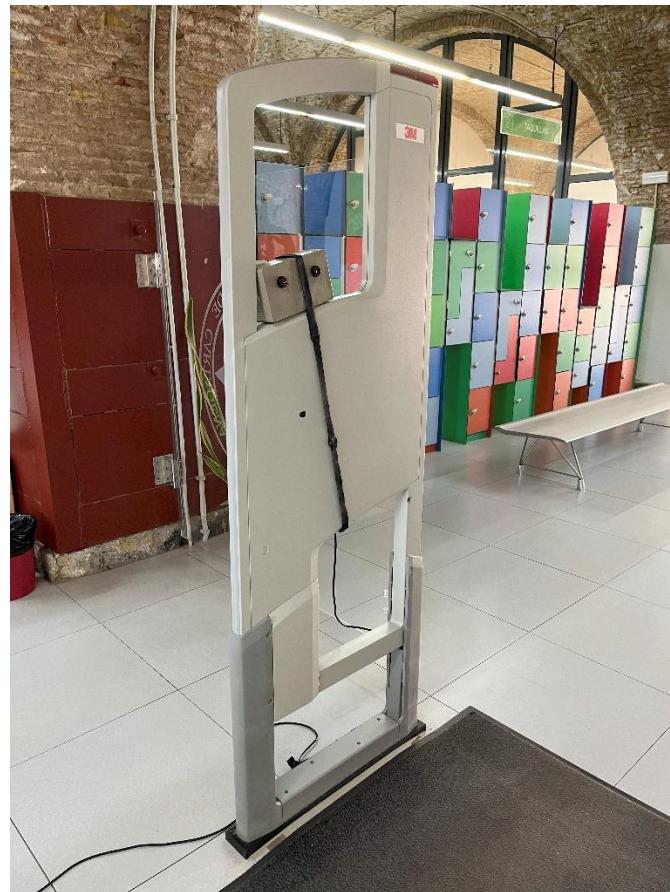


Figura 76. Lugar en el que se encuentran los sensores (por delante)



Figura 77. Lugar en el que se encuentran los sensores (por detrás)

Nuestro servidor se ejecutó en un mini PC Leotec (se muestra en la figura 78) con un Intel N100 (4 núcleos eficientes de arquitectura Gracemont 12^a gen. 6W TDP) con 12GB de RAM DDR5 y 256 GB de SSD.



Figura 78. Imagen del PC donde se ha ejecutado nuestro servidor

La realización de las pruebas consiste en comparar la realidad con lo que el sistema reconoce, es decir, se va a contar a mano todas y cada una de las personas que entran y salen y se va a comparar con lo que nuestro sistema detecta.

Para ello utilizaremos los siguientes parámetros:

- **Entradas reales:** se trata del número de personas que realmente han entrado a la biblioteca.
- **Entradas detectadas:** se trata del número de personas que el sistema ha detectado que han entrado a la biblioteca.
- **Entradas no detectadas:** se trata del número de personas que, a pesar de haber entrado, el sistema no ha detectado.
- **Falsas entradas:** se trata del número de personas que el sistema ha detectado que han entrado, pero que realmente no han entrado. Suele ocurrir cuando entra una persona, pero se cuenta como si fueran dos.
- **Salidas reales:** se trata del número de personas que realmente han salido de la biblioteca.
- **Salidas detectadas:** se trata del número de personas que el sistema ha detectado que han salido de la biblioteca.
- **Salidas no detectadas:** se trata del número de personas que, a pesar de haber salido, el sistema no ha detectado.
- **Falsas salidas:** se trata del número de personas que el sistema ha detectado que han salido, pero que realmente no han salido. Suele ocurrir cuando sale una persona, pero se cuenta como si fueran dos.

Los datos recogidos a lo largo de varios días de testeo son los siguientes:

	Entradas	Salidas
Reales	800	800
Detectadas	788	776
No detectadas	12	24
Falsas	0	4

Figura 79. Tabla de datos recogidos

De manera que los porcentajes de error serían los siguientes:

	Entradas	Salidas	Total
Porcentaje error	1,5 %	3,5 %	2,5 %

Figura 80. Tabla de porcentajes de error

El porcentaje de error de entrada está calculado sumando el número de entradas no detectadas al número de falsas entradas y dividiéndolo entre el número de entradas reales. De la misma manera, el porcentaje de error de salidas está calculado sumando el número de salidas no detectadas al número de falsas salidas y dividiéndolo entre en el número de salidas reales. Por último, el porcentaje de error total está calculado haciendo la media entre el porcentaje de error de entradas y el porcentaje de error de salidas.

$$\% \text{ Error entradas} = \frac{\text{Entradas no detectadas} + \text{Falsas entradas}}{\text{Entradas reales}} * 100$$

Figura 81. Ecuación del porcentaje de error de entradas

$$\% \text{ Error salidas} = \frac{\text{Salidas no detectadas} + \text{Falsas salidas}}{\text{Salidas reales}} * 100$$

Figura 82. Ecuación del porcentaje de error de salidas

$$\% \text{ Error total} = \frac{\% \text{ Error entradas} + \% \text{ Error salidas}}{2}$$

Figura 83. Ecuación del error total

6.3 Análisis de los resultados

Con los datos recopilados en las pruebas de testeo y validación del sistema podemos asegurar que estamos ante un sistema bastante robusto con un error total de cálculo del conteo de personas de únicamente un 2,5%, lo que nos otorga una fiabilidad del sistema del 97,5%.

Creemos que se trata de unos datos realmente positivos para el coste realmente económico que tiene todo el sistema, en el que, además, hemos reutilizado un router antiguo para reducir todavía más el presupuesto.

Capítulo 7. Conclusiones y trabajos futuros

7.1 Conclusiones

Uno de los pilares de este proyecto son las placas de desarrollo ESP32, que nos ofrecen un equipo SoC (system-on-a-chip) muy competitivo por el precio que tienen, ya que son perfectas para cualquier proyecto IoT gracias a su versatilidad y conectividad Wi-Fi y Bluetooth. Gracias a esto, podemos encontrar gran cantidad de información en internet acerca de cómo aprender a usarlas y programarlas.

Se ha diseñado e implementado un sistema de conteo de personas en la biblioteca de Antigones basándonos en estas placas ESP32, programadas en Arduino, para obtener los flujos de personas y que un servidor, programado en Python, los gestione. La solución que se ha implementado es una solución con un coste realmente bajo, ya que estaríamos hablando de unos 60€, que serían unos 10€ por cada placa ESP32 (necesitamos dos), más 10€ por cada sensor E18-D80NK (necesitamos cuatro).

Los resultados de las pruebas de testeo creemos que son especialmente buenos para el bajo coste que tiene el sistema completo, ya que estaríamos hablando de un 97,5% de fiabilidad de todo el sistema.

7.2 Grado de consecución de los objetivos y formación adquirida

Objetivos logrados

Los objetivos logrados en este proyecto han sido los siguientes:

- Aprendizaje del entorno de desarrollo de Arduino con las placas ESP32 y del funcionamiento de estas con los sensores de proximidad infrarrojos E18-D80NK.
- Implementación de un programa con un algoritmo de conteo de personas y el uso del LED RGB de la placa ESP32-S3-DevkitC-1.
- Aprendizaje de la conexión Wi-Fi de un ESP32 con un servidor para mandarle mensajes.
- Formación en el lenguaje de programación Python y programación de un servidor que sea capaz de recibir mensajes de los ESP32 y de tomar decisiones en base a los mensajes que recibe, además de monitorizar el estado de la conexión de los clientes.
- Creación de una base de datos que se actualice con los mensajes que recibe el servidor de los ESP32.
- Presentación de los datos en la herramienta de Grafana

Formación adquirida

A lo largo del desarrollo de este proyecto, se ha adquirido una formación significativa en varias áreas:

- **IDE Arduino y placas ESP32.** Se han adquirido conocimientos del entorno de desarrollo de Arduino y su lenguaje de programación, con placas ESP32 y la conexión de estas con un servidor para mandarle mensajes.
- **Lenguaje de programación Python.** Se han adquirido conocimientos en el lenguaje de programación Python ya que nunca lo había utilizado.
- **Procesamiento de datos en tiempo real.** Se ha ganado experiencia en el diseño e implementación de sistemas que procesan y analizan datos en tiempo real.
- **Herramienta Grafana.** Se ha obtenido una formación en el uso de la herramienta Grafana para presentar datos gráficamente.

En conclusión, el proyecto no solo ha cumplido con todos los objetivos establecidos, sino que también ha proporcionado una valiosa formación en diversas áreas tecnológicas y de gestión de proyectos.

Dificultades en el desarrollo del proyecto

El desarrollo de este proyecto no estuvo exento de dificultades, las cuales ofrecieron valiosas oportunidades de aprendizaje:

- **Aprendizaje desde cero:** se enfrentaron desafíos significativos al no haber utilizado antes entornos de desarrollo como el de Arduino o Pycharm o el lenguaje de programación de Arduino o Python. También se presentó dificultad a la hora de usar herramientas como Grafana. Cada una de estas áreas requirió una formación y estudio para poder comprender e implementar correctamente las diferentes soluciones del proyecto.
- **Detección y solución de problemas:** a lo largo del proyecto surgieron varios problemas que fueron detectados y solucionados como se explicó en los apartados [3.6](#) y [4.3](#).
- **Implementación de una interfaz gráfica.** Al principio se pensó en usar las interfaces gráficas ofrecidas por Python mediante la librería tkinter, pero finalmente nos dimos cuenta de que la herramienta de Grafana se trataba de una mejor solución, ya que se encarga de mostrar de manera gráfica los datos almacenados en nuestra base de datos.

En conclusión, aunque el desarrollo del proyecto presentó diversas dificultades, cada una de ellas contribuyó a un aprendizaje significativo y a la mejora del sistema. El enfrentamiento y la resolución de estos desafíos han fortalecido el resultado en este proyecto.

7.3 Trabajos futuros

Se proponen las siguientes ideas para ampliar el actual proyecto:

- El uso de Bluetooth Low Energy (BLE) para comparar si el envío de mensajes es mejor que por Wi-Fi en cuanto a varios parámetros, entre ellos el de la congestión de red y el del consumo de energía de los ESP32, etc.
- La utilización de más sensores, 4 por ejemplo, por cada puerta, para obtener datos y entrenar una inteligencia artificial mediante Machine Learning para saber si entra solo una persona o dos muy juntas, etc. Esto se podría realizar en el ESP32-S3-DevkitC-1, ya que su arquitectura está optimizada para ello con instrucciones específicas para acelerar algoritmos de IA, además de una memoria RAM más grande que el ESP32-DevKitC V4. También dispone de un soporte mejorado para periféricos como cámaras y pantallas, lo que lo hace ideal para aplicaciones avanzadas en domótica, visión por computadora y sistemas embebidos inteligentes.
- Reducir el consumo energético de los ESP32 mediante el uso del modo Deep Sleep. En nuestro proyecto no está contemplado que los ESP32 vayan a funcionar con batería, pero, si fuera el caso, sería interesante la utilización del modo Deep Sleep del ESP32 para la optimización del consumo energético y la prolongación de la autonomía de la batería.

Bibliografía

- [1] Vídeo explicativo del funcionamiento del sensor E18-D80NK:
<https://youtu.be/MrYsmAwzfrM?si=WeGgR4jYfHy1ame4>
- [2] Página web del ESP32-DevKitC V4: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/hw-reference/esp32/get-started-devkitc.html>
- [3] Página web del ESP32-S3-DevKitC-1: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1-v1.0.html>
- [4] IDE Arduino: <https://www.arduino.cc/en/software>
- [5] Documentación de Arduino: <https://www.arduino.cc/reference/en/>
- [6] Librerías ESP32 para IDE Arduino: <https://github.com/espressif/arduino-esp32>
- [7] Librerías LED RGB Adafruit Neopixel: https://github.com/adafruit/Adafruit_NeoPixel
- [8] Python: <https://www.python.org/>
- [9] Pycharm: <https://www.jetbrains.com/es-es/pycharm/>
- [10] SQLite: <https://www.sqlite.org/>
- [11] SQLite3: <https://docs.python.org/3/library/sqlite3.html>
- [12] Grafana: <https://grafana.com/>
- [13] CSV: https://es.wikipedia.org/wiki/Valores_separados_por_comas
- [14] Vídeo explicativo del circuito de salida del sensor E18-D80NK:
https://youtu.be/wb9AwY4qdts?si=NTQ29S_dv0bHQi8o
- [15] Diagramas realizados en: <https://app.diagrams.net/>
- [16] Controladores del puente USB a la UART: <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>
- [17] Router Linksys WRT54GL: <https://www.xataka.com/perifericos/linksys-fabrico-router-que-revoluciono-mundo-routers-ingrediente-secreto-era-open-source>
- [18] DB Browser for SQLite: <https://sqlitebrowser.org/>
- [19] Vídeo explicativo de un servidor en Python: <https://youtu.be/3QiPPX-KeSc?si=9A92T-gAfOH4khub>
- [20] Wireshark: <https://www.wireshark.org/download.html>

Anexos

Datasheet del sensor E18-D80NK

Device Characteristics

- Power Supply: 5VDC
- Supply current DC <25mA
- Maximum load current 100mA (Open-collector NPN pulldown output)
- Response time <2ms
- Diameter: 17MM
- Pointing angle: $\leq 15^\circ$, effective from 3-80CM Adjustable
- Detection of objects: transparent or opaque
- Working environment temperature: -25°C+55°C
- Case Material: Plastic
- Lead Length: 45CM

Figura 84. Especificaciones sensor E18-D80NK

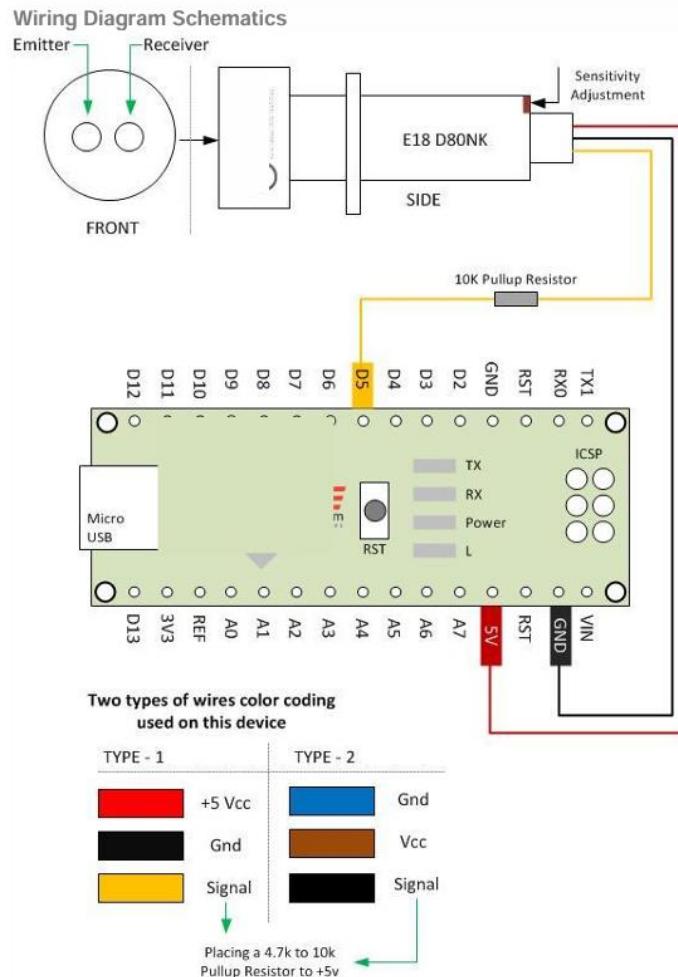
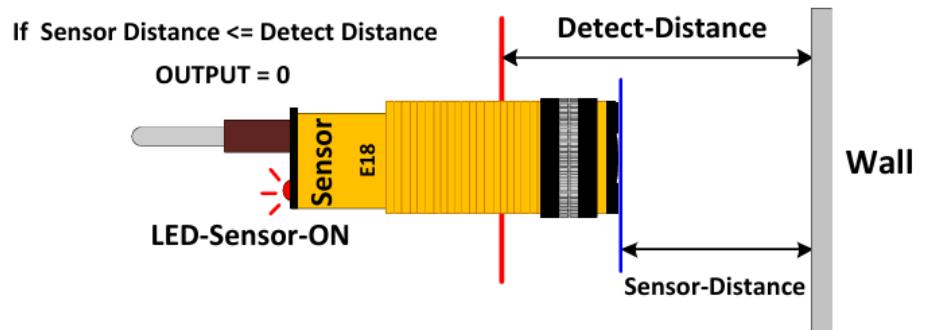
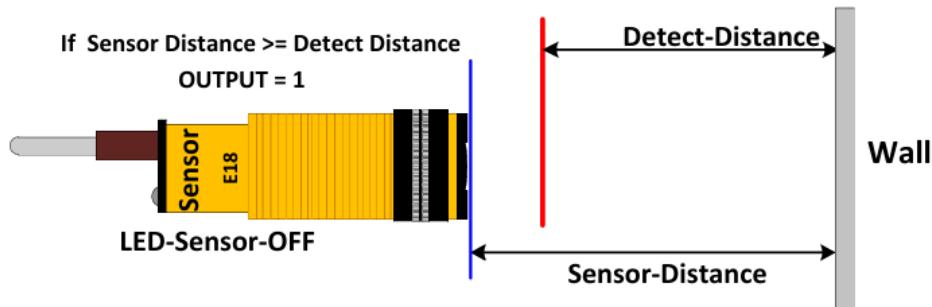


Figura 85. Esquema de conexión del sensor E18-D80NK



When distance of Sensor <= the specified distance detection, LED Status is ON and OUTPUT = 0



When distance of Sensor >= the specified distance detection, LED Status is OFF and OUTPUT = 1

Figura 86. Funcionamiento del sensor E18-D80NK

Datasheet SoC ESP32-S3

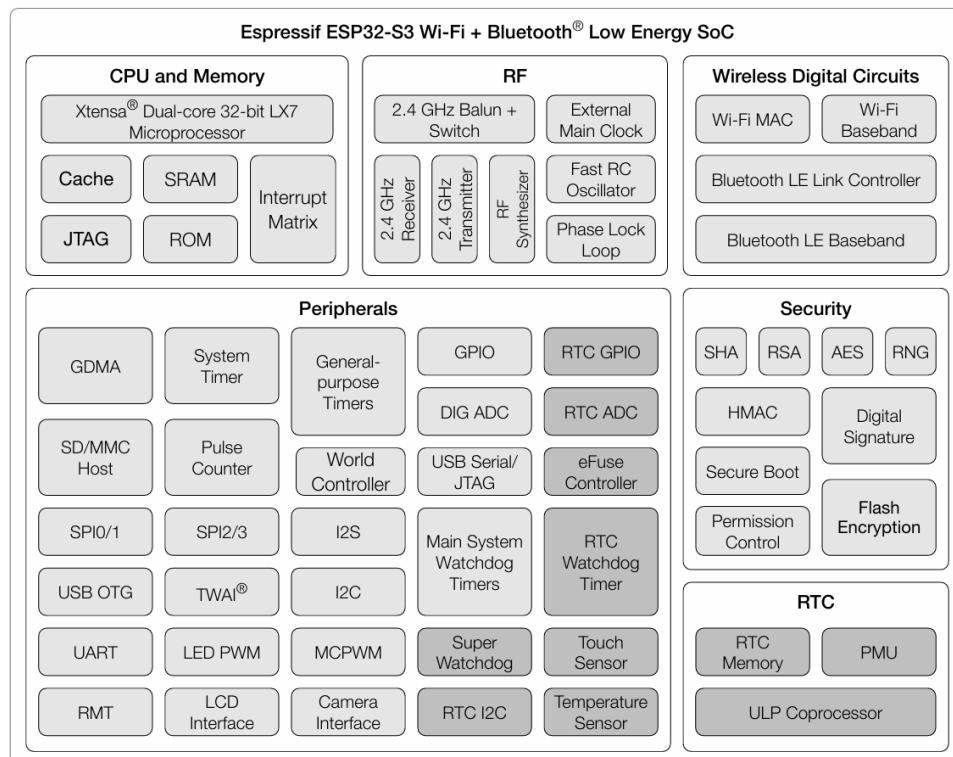


Figura 87. Diagrama SoC ESP32-S3