Grado en Ingeniería Informática

Trabajo de Final de Grado

Posicionamiento de una cabina de ascensor mediante un ESP32 y sensores

Autor:
Joaquín González Álvarez

Supervisor:
Diego Centelles Beltrán
Tutor académico:
Juan Carlos Fernández Fernández

Fecha de lectura: 21 de junio de 2021
Curso académico 2020/2021

Resumen

El proyecto desarrollado consiste en determinar la planta actual de la cabina mediante el uso de sensores y un microcontrolador. Para realizarlo utilizaremos un microcontrolador ESP-32 y un sensor de presión atmosférica con el que podremos determinar la altura de la cabina.

Para ello usaremos el microcontrolador Atom Lite programado mediante el framework Espressif IDF, y utilizaremos el lenguaje de programación "C" y también la tecnología WiFi para mandar los datos al servidor.

Esto resuelve un problema que tiene actualmente la empresa Nayar Systems, ya que para determinar la altura de la cabina dependen directamente de la maniobra del ascensorista y se tienen que adaptar a cada una de ellas.

Palabras clave

Microcontrolador, ESP32, presión atmosférica, ascensor, altitud

Keywords

Microcontroller, ESP32, atmospheric pressure, elevator, altitude

Agradecimientos

Quiero agradecer en primer lugar a todos los profesores que he tenido en mi carrera académica, desde los primeros en el colegio hasta los últimos en la universidad. A mi tutor académico Juan Carlos por la dedicación y apoyo durante todo el transcurso. También quiero agradecer a mi familia y amigos, ya que sin ellos no sería a día de hoy la persona que soy. Ellos me han apoyado incondicionalmente y me han ayudado a conseguir formarme como informático.

Pero me gustaría agradecer especialmente a mis padres y a mi hermana. Ellos fueron los que lidiaron con el pequeño y manegueta Ximo que siempre quería desmontar y toquetear todo, y lo que conseguía toquetear luego no volvía a funcionar.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Índice

La introducción recoge el contexto en el que se desarrolla el proyecto, así como la motivación que ha llevado a la empresa a su planteamiento. Además, presenta los objetivos y alcance del proyecto y las herramientas y tecnologías empleadas.

1.1. Contexto y motivación del proyecto

1.1.1. Contexto

Nayar Systems [5] es una empresa española enfocada en el sector de la ingeniería de las telecomunicaciones. La empresa está centrada en dos sectores principalmente, el de la elevación y el de Internet of Things Industrial (IoT).

La misión actual de Nayar Systems es crear soluciones IoT ágiles y competitivas y de este modo acompañar a sus clientes en su evolución tecnológica aportando su amplio conocimiento

en ascensores y sistemas industriales. Nayar Systems empezó su actividad empresarial en el año 2007 en Castellón, pero actualmente está expandiéndose por Europa y hace unos años inauguraron una sede en Shanghái.

La empresa ofrece a sus clientes una serie de productos, los cuales están explicados a continuación y podemos ver un esquema de relación entre ellos en la Figura 1.1:

1. **GSM Smart Router (GSR):** Dispositivo diseñado para las comunicaciones IoT en el sector de la elevación. Está desarrollado en su totalidad por Nayar Sytems. Tiene diferentes funcionalidades, entre ellas: conexión a la maniobra del ascensor[1], telemetría y telecontrol, conexión WiFi [7] para Advertisim, además de cumplir con la normativa EN 81-28. Cuenta con conectividad WiFi y 4G y con los puertos Ethernet, CAN BUS, RS232 y USB.
2. **net4machines:** Proveedor de servicios de IoT industrial que ofrece una VPN, una red segura y privada para cualquier dispositivo. Dentro de net4machines, se encuentra la VPN, Smart Control y Manager.
3. **72 horas:** Plataforma de comunicación M2M (Machine to Machine) que cumple con la normativa europea EN 81-28 y que ofrece una línea telefónica al ascensor por si existiera alguna emergencia.
4. **Advertisim:** Dispositivo con conectividad en tiempo real capaz de mostrar información del ascensor y contenido publicitario a elección del cliente. Muestra entre otras cosas, el piso, la dirección, señales especiales, información meteorológica, tweets, imágenes y vídeos, noticias y mensajes personalizados.

1.1.2. Motivación

La motivación de la empresa surge cuando se plantean cómo pueden conocer la ubicación del ascensor en un momento dado. Actualmente la maniobra del ascensor ofrece la ubicación del ascensor, pero esto obliga a Nayar a tener que conectarse con la maniobra de cada fabricante de ascensores. Por eso Nayar quiere conseguir una solución independiente, que pueda ofrecerles la ubicación del ascensor para ofrecérsela tanto a los usuarios del ascensor como a los técnicos del mismo para calibrar el ascensor.

Nayar Systems llevaba tiempo queriendo realizar este proyecto, y aprovechó las prácticas de la UJI para empezar el desarrollo del proyecto.

1.2. Objetivos del proyecto

El principal objetivo del proyecto es poder determinar la parada actual de la cabina de un ascensor.

[1]Una maniobra de ascensor es el sistema que controla las acciones que realiza un ascensor. En el apartado 1.4.1 lo explicamos en detalle.

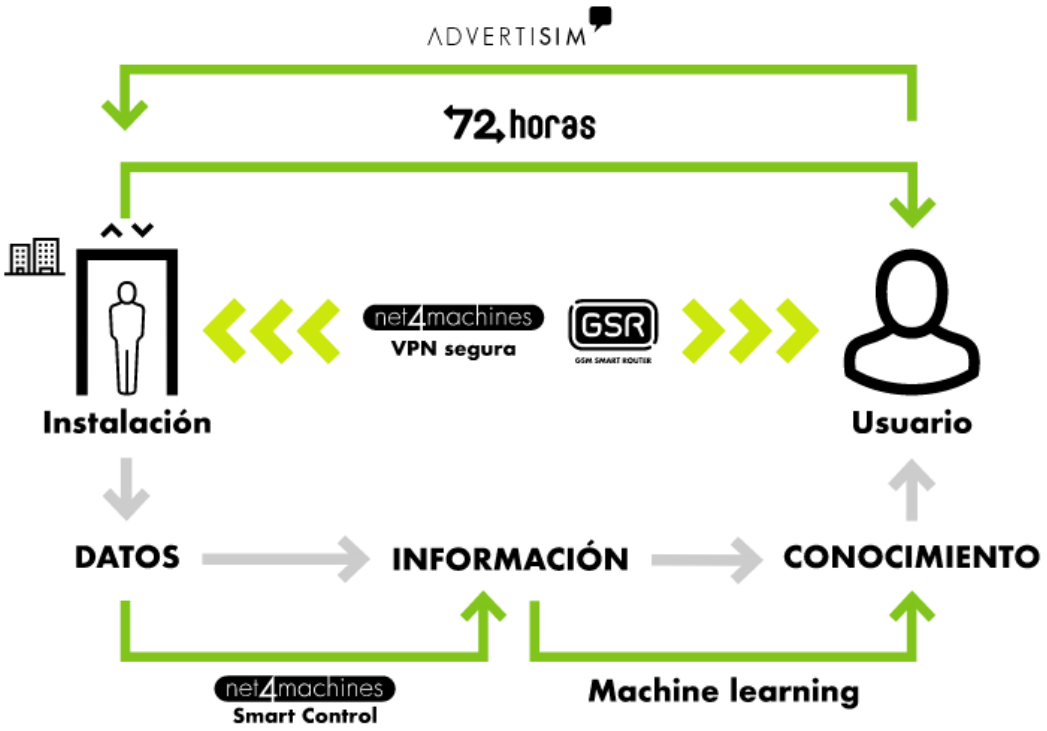

Figura 1.1: Productos y servicios de Nayar Systems

Y este objetivo lo desglosamos en los siguientes subobjetivos:

1. Familiarizarse con el entorno de desarrollo.
2. Familiarizarse con los buses de comunicación necesarios para la conexión de sensores.
3. Investigar y evaluar los distintos sensores que permiten calcular la parada actual de la cabina.
4. Conexionado de sensores.
5. Investigar el protocolo de comunicación con el sensor.
6. Lograr relacionar los datos del sensor con la posición de la cabina.
7. Añadir al desarrollo la lectura de datos de los sensores y el envío de los datos al servidor.

1.3. Alcance del proyecto

En esta sección del proyecto vemos el alcance del proyecto, el cual indica qué es lo más importante y lo que incluimos en las tareas del proyecto. Esto se hace para no tener un proyecto excesivamente grande que deberíamos de subdividir en proyectos más pequeños.

En el alcance del proyecto podemos incluir la programación y conexión del sistema empotrado que permitirá determinar la altura a la que se encuentra la cabina del ascensor y el envío de datos al servidor. Sin embargo, el desarrollo de una aplicación para ver los datos, o el desarrollo de un servidor está fuera del alcance del proyecto y podría ser un trabajo futuro a realizar.

Dentro del alcance del proyecto vemos que hay tres aspectos para definirlo de una manera más precisa, estos son el alcance funcional, el alcance organizativo y el alcance tecnológico o informático. El primero determina las funciones a alto nivel que va a resolver el proyecto. El alcance organizativo determina las áreas y departamentos que harán uso del proyecto y también las organizaciones externas que lo utilizarán. Y por último, el alcance informático o tecnológico, es el punto de vista más bajo, en el cual vemos qué otros productos o sistemas van a estar implicados con este proyecto.

A continuación explicaremos los tres alcances:

1.3.1. Alcance funcional

El alcance funcional ofrece a la empresa la posibilidad de estimar el piso actual de la cabina del ascensor con total independencia del fabricante y maniobra.

1.3.2. Alcance organizativo

El problema que pretende resolver este proyecto afecta en cierto modo a todos los departamentos de la empresa, pero fundamentalmente con los de software y hardware, ya que son los encargados de conectarse actualmente a las maniobras del ascensor. Cabe destacar que el usuario que use el ascensor también interactúa con el sistema, al poder ver el piso en el que se encuentra.

1.3.3. Alcance tecnológico

El alcance tecnológico abarca tanto las pantallas Advertisim que son las que instalan en la cabina del ascensor e indican el piso actual, como el GSR que va instalado a la maniobra del ascensor y es el encargado de comunicarse con la misma.

1.4. Descripción del proyecto

El proyecto tiene la finalidad de obtener la altura y con ella el piso actual de la cabina de un ascensor. Originalmente el proyecto estaba pensado para determinar la altura usando un sensor de presión atmosférica, pero como explicamos en los posteriores capítulos tuvimos que añadir elementos extras.

Mediante el proyecto pretendemos resolver el problema que tiene actualmente la empresa, que consiste en determinar el piso actual de la cabina, ya que actualmente necesitan conectarse a la maniobra del ascensor.

1.4.1. Tecnologías utilizadas

Maniobra del ascensor

La maniobra del ascensor es el elemento que controla todas las acciones del ascensor, ya sea la apertura de puertas, aceleración de la cabina para realizar los movimientos, estados del ascensor, velocidades, control de errores, alarmas de cabina, etc.

Nayar diseña unos dispositivos que se conectan a las maniobras y permiten su teleoperación remota por parte de un técnico de ascensor y también aportan un valor añadido permitiendo conocer el estado del ascensor en cualquier momento.

Existen maniobras de ascensor de muchos tipos, y de muchos fabricantes distintos, que funcionan de diferente manera y con protocolos distintos. La adaptación a cada maniobra resulta tedioso para la empresa, de ahí la necesidad de la realización de este proyecto. En la Figura 1.2 podemos ver una maniobra de ascensor.

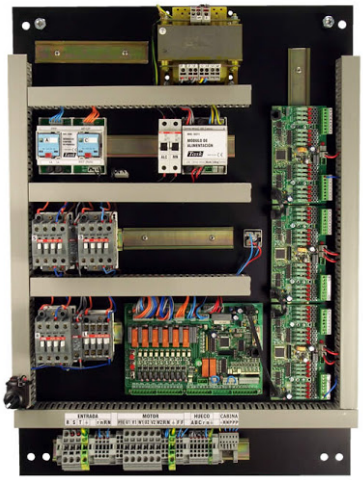

Figura 1.2: Maniobra de un ascensor

Para evitar esa dependencia, la idea consiste en colocar un microcontrolador con un sensor de presión atmosférica en la cabina del ascensor, y obtener la altura mediante la presión atmosférica en ese punto.

Presión Atmosférica

La presión atmosférica es el fenómeno por el cual vamos a estimar la altura de la cabina del ascensor. La presión atmosférica es la fuerza que ejerce el aire en un punto de la superficie, es decir es la cantidad de aire por encima de ese punto. Cuanto mayor sea la altura menor es esta presión. En la Figura 1.3 presentamos una gráfica con la relación existente entre la presión en milibares y la altura en kilómetros.

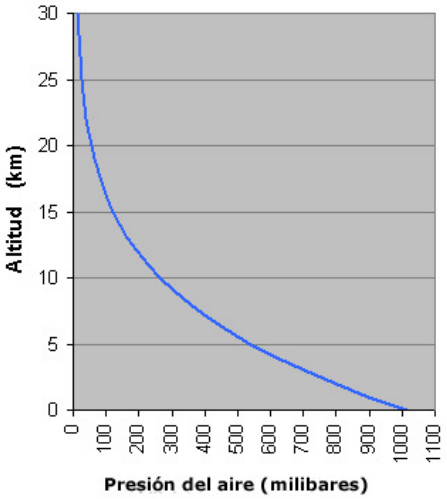

Figura 1.3: Gráfica relacional presión/altura

De esta manera, sabiendo la presión atmosférica y la altura en un punto X, y conociendo la presión atmosférica en un punto Y, podemos estimar la altitud mediante la expresión matemática que mostramos en la Figura 1.4.

```
altitude = 44330 * (1 - pow(pressure / refPressure, 0.1903));
```

Figura 1.4: Expresión de cálculo de la altura

En la expresión 1.4 vemos la fórmula matemática utilizada para calcular la altitud, donde la variable “pressure” es la presión del punto del que queremos calcular la altitud y la variable “refPressure” es la presión del punto de referencia cuya altitud conocemos.

Para conocer la presión atmosférica en un punto debemos utilizar un barómetro o sensor barométrico, antiguamente se usaban barómetros de mercurio los cuales seguían la fórmula:

$$Pa = P = pg(y2 - y1) = pgh$$

p: Densidad del mercurio
g: Fuerza gravitacional

h: Altura de la columna de mercurio

En este proyecto usaremos unos sensores barométricos, en concreto el modelo BMP280 [1] que tiene una precisión teórica de ± 1 metro.

Protocolo serie I2C

I2C [3] es un protocolo de comunicaciones en serie. Usa solamente dos cables: el cable de datos (SDA) y el cable del reloj (SCL). El protocolo soporta un rango de velocidades que va desde 100 Kbps hasta 3,4 Mbps. La velocidad de transmisión se establece al principio de la comunicación y puede ser de 100 Kbps, de 400 Kbps o de 3,4 Mbps.

Funciona mediante un esquema maestro/esclavo tal y como describimos a continuación.

1. El maestro es el primer módulo que transmite.
2. Cada esclavo tiene una dirección única de 7 bits. Con lo que conseguimos poder conectar 128 módulos a un solo bus I2C.
3. El maestro emite la dirección del esclavo, indicando si quiere realizar una lectura o una escritura.
4. El esclavo responde con una señal de reconocimiento (ACK) y posteriormente el maestro envía los datos o se prepara para recibirlos.

Las señales en I2C son a colector abierto, con esto se soluciona cualquier conflicto eléctrico. Las colisiones se evitan haciendo que todos los módulos estén escuchando el bit presente en el bus, lo que permite establecer una jerarquía de prioridades. Tienen más prioridad los módulos que tienen una dirección menor.

En la Figura 1.5 podemos ver una trama del protocolo I2C. En la parte superior aparece la señal de reloj (SCL), la cual establece la velocidad de comunicación e indica los cambios de bits. En la parte inferior vemos la señal de datos (SDA). La trama comienza con el bit de start (0 lógico), a continuación se indica la dirección de siete bits del esclavo con el que queremos comunicar. El siguiente bit indica si se quiere realizar una escritura o una lectura. A continuación se recibe el bit ACK de respuesta del esclavo. Finalmente la trama contiene los bits de datos que dependiendo de si el maestro ha solicitado lectura o escritura los recibirá el maestro o el esclavo. Los dos últimos bits indican la señal de reconocimiento negativa (NACK) de respuesta y el bit de stop (1 lógico).

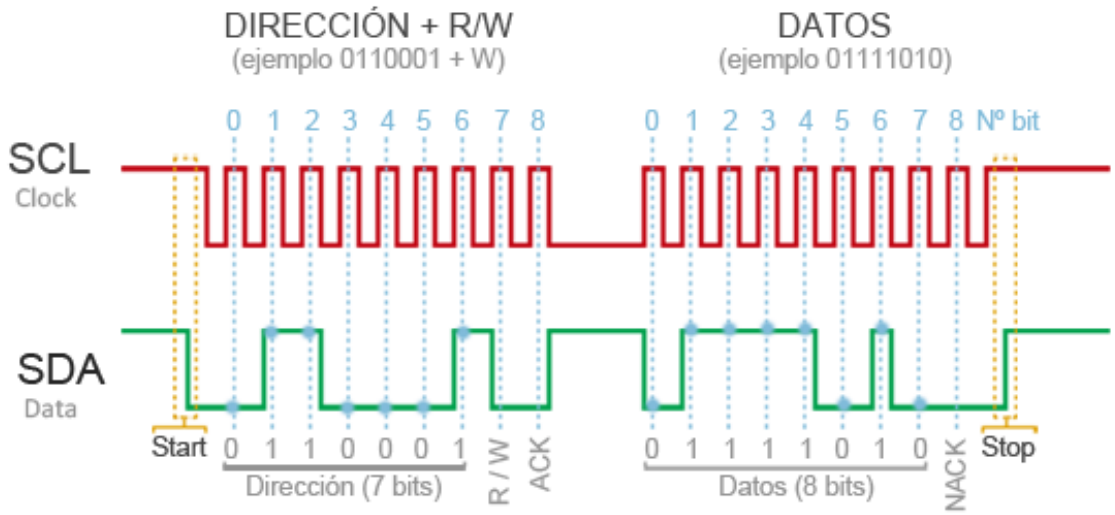

Figura 1.5: Trama del protocolo I2C

Biblioteca RMT

La biblioteca RMT (Remote Control) [9] se utiliza para enviar y recibir señales por control remoto. Sin embargo, tiene multitud de aplicaciones como WiFi y radiofrecuencia.

Su funcionamiento es muy sencillo, para utilizarla creamos un objeto indicando la duración de los pulsos y el valor (high/low) y el módulo se encarga de generar la señal. Después esta señal es la que se transmite mediante el módulo de radiofrecuencia.

En el apartado de radiofrecuencia del capítulo 4 presentamos un ejemplo del uso/funcionamiento de la biblioteca RMT.

Visual Studio Code

Visual Studio Code [12] es un editor de código fuente desarrollado por Microsoft y compatible con casi todos los sistemas operativos. Permite desarrollar software en cualquier lenguaje y además facilita esta tarea permitiéndo instalar diferentes módulos que ayudan a su programación. En nuestro caso, hemos instalado el módulo de Espressif IDF.

Espressif IDF

Espressif IDF [2] es el framework oficial del fabricante de los microcontroladores ESP, el cual es compatible con el entorno de desarrollo Visual Studio y ofrece una serie de herramientas para facilitar el desarrollo del firmware del microcontrolador.

FreeRTOS

FreeRTOS [10] es un sistema operativo de tiempo real diseñado para sistemas empotrados compatible con la gran mayoría de microcontroladores del mercado. El núcleo del sistema funciona simplemente con tres ficheros en lenguaje C y algunas funciones escritas en lenguaje ensamblador. El sistema operativo ofrece multitud de herramientas características de los sistemas operativos en tiempo real como pueden ser: hilos, mutex, semáforos, etc.

Lenguaje C

C [14] es un lenguaje de programación orientado a la implementación de sistemas operativos. Es un lenguaje débilmente tipado y estructurado. C ofrece una gran portabilidad, ya que no depende del hardware.

Control de versiones Git y Github

El control de versiones Git es una herramienta que permite tener un historial de versiones de código de forma distribuida. Funciona mediante réplicas del proyecto, las cuales se van guardando una tras otra siguiendo unas ramas. Además permite tener varias ramas o caminos paralelos. Github [11] es una plataforma que permite almacenar los proyectos que utilizan el sistema de control de versiones Git. De esta manera, permite compartir el proyecto con más gente y que haya colaboradores en el mismo proyecto.

1.4.2. Hardware utilizado

Microcontrolador ESP32

El microcontrolador ESP32 [4] es un dispositivo diseñado para proyectos empotrados que cuenta con WiFi y Bluetooth incorporado. Dispone de pines genéricos para conectar periféricos y además soporta el bus I2C para comunicarse con el sensor de presión atmosférica. En la Figura 1.6 vemos un modelo del microcontrolador llamado Dev Kit y en la Figura 1.7 podemos ver el microcontrolador Atom Lite pero con un chasis de plástico diseñado por la empresa M5Stack. Esta empresa también fabrica otros módulos en el mismo formato, lo que facilita la conexión entre ellos.

Para el proyecto utilizamos los dos microcontroladores (Dev Kit y Atom Lite). Ambos son microcontroladores de tipo ESP32. El microcontrolador Atom Lite decidimos utilizarlo, ya que en la empresa tienen multitud de sensores y microcontroladores de este tipo y es con el que realizan todos los prototipos. Por su parte, debido a que no tenían más microcontroladores de tipo Atom Lite, utilizamos el microcontrolador Dev Kit con la finalidad de no retrasar el avance del proyecto, al estar disponible.

Figura 1.6: Microcontrolador ESP32 Dev Kit

Figura 1.7: Microcontrolador ESP32 Atom Lite

Sensor BMP280

El sensor BMP280 [1] es un sensor que mide presión atmosférica, temperatura y humedad. Tiene una precisión teórica que ofrece la medición de la presión atmosférica y por tanto de la altura con un error de $\pm$ 1 metro. El sensor ofrece la posibilidad de conectarlo al microcontrolador mediante I2C y SPI. En este proyecto utilizaremos el protocolo I2C.

Tranceiver radio Fs1000a

El módulo de radio FS1000a [8] ofrece conectividad inalámbrica unidireccional y es muy sencillo de implementar. Consta de dos módulos (transmisor y receptor) que se conectan usando simplemente un pin, el de datos. Este módulo sirve para conectar los dos microcontroladores, ya que el WiFi que lleva incorporado el microcontrolador no funciona de forma adecuada en el hueco de la cabina del ascensor.

1.5. Estructura de la memoria

En esta sección explicamos la estructura de la memoria, empezando desde la introducción, y pasando por todas las etapas de desarrollo del proyecto, incluidas las conclusiones finales.

En el capítulo 2 presentamos la metodología empleada y la planificación del proyecto, además de las posibles desviaciones del desarrollo del proyecto. En el capítulo 3 desarrollamos el análisis y diseño del sistema. En el capítulo 4 detallamos la implementación del sistema y también mostramos las pruebas realizadas. Y finalmente el capítulo 5 mostraremos las conclusiones finales y futuros trabajos.

Capítulo 2

Planificación del proyecto

Índice

En este capítulo desarrollamos la metodología seguida para la realización del proyecto, estimamos los recursos y costes del proyecto, así como también describimos tanto la planificación inicial como las desviaciones respecto a esta planificación.

2.1. Metodología

La metodología seguida en este proyecto es la metodología predicativa, en el cual seguimos el modelo en cascada [15]. En este modelo tenemos la necesidad de tener una definición de los requisitos al inicio del proyecto.

En esta metodología se sigue un cauce secuencial, en la que se van realizando tareas y al acabar estas tareas van comenzando otras tareas.

2.2. Planificación inicial

Durante la fase inicial de la estancia en prácticas decidimos dedicarle las primeras horas a la planificación de las tareas a realizar. Esta fase es de las más importantes en el desarrollo de un proyecto, ya que si no se realiza de manera correcta aparecen obstáculos en el camino que dificultan el avance del proyecto.

En el Cuadro 2.1 podemos ver las correspondientes tareas y su estimación horaria. Y en la Figura 2.1 vemos el diagrama de Gantt planificado.

TAREA	ESTIMACIÓN
a) Estudio del entorno de trabajo	**60h**
a.1) Instalación y configuración del entorno de desarrollo	12h
a.2) Familiarizarse con el entorno de desarrollo	48h
b) Estudio del microcontrolador	**24h**
b.1) Investigar y elegir el microcontrolador	12h
b.2) Familiarizarse con el microcontrolador	12h
c) Estudio del sensor	**60h**
c.1) Familiarizarse con los buses de comunicación	18h
c.2) Investigar y evaluar los distintos sensores	12h
c.3) Investigar el protocolo de comunicación del sensor	12h
c.4) Conexionado y configuración de sensores	18h
d) Programación del microcontrolador y sensores	**90h**
d.1) Programación del módulo del sensor	30h
d.2) Programación del algoritmo de entrenamiento	60h
e) Pruebas de funcionamiento	**24h**
f) Relacionar datos con posición real	**12h**
g) Desarrollar el envío de los datos al servidor	**30h**

Cuadro 2.1: Estimación de la duración de las tareas

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Proyecto**	**50 días**	**lun 01/03/21**	**vie 07/05/21**
2		**Estudio del entorno de trabajo**	**10 días**	**lun 01/03/21**	**vie 12/03/21**
3		Instalación y configuración del	2 días	lun 01/03/21	mar 02/03/21
4		Familiarizarse con el entorno de desarrollo	8 días	mié 03/03/21	vie 12/03/21
5		**Estudio del microcontrolador**	**4 días**	**lun 15/03/21**	**jue 18/03/21**
6		Investigar y elegir el microcontrolador	2 días	lun 15/03/21	mar 16/03/21
7		Familiarizarse con el microcontrolador	2 días	mié 17/03/21	jue 18/03/21
8		**Estudio del sensor**	**10 días**	**vie 19/03/21**	**jue 01/04/21**
9		Familiarizarse con los buses de comunicación	3 días	vie 19/03/21	mar 23/03/21
10		Investigar y evaluar los distintos sensores	2 días	mié 24/03/21	jue 25/03/21
11		Investigar el protocolo de comunicación del	2 días	vie 26/03/21	lun 29/03/21
12		Conexionado y configuración de	3 días	mar 30/03/21	jue 01/04/21
13		**Programación del microcontrolador y sensores**	**15 días**	**vie 02/04/21**	**jue 22/04/21**
14		Programación del módulo sensor del	5 días	vie 02/04/21	jue 08/04/21
15		Programación del algoritmo de entrenamiento	10 días	vie 09/04/21	jue 22/04/21
16		Pruebas de funcionamiento	4 días	vie 23/04/21	mié 28/04/21
17		Relacionar datos del sensor con posición de la cabina	2 días	jue 29/04/21	vie 30/04/21
18		Desarrollar el envío de los datos al servidor	5 días	lun 03/05/21	vie 07/05/21

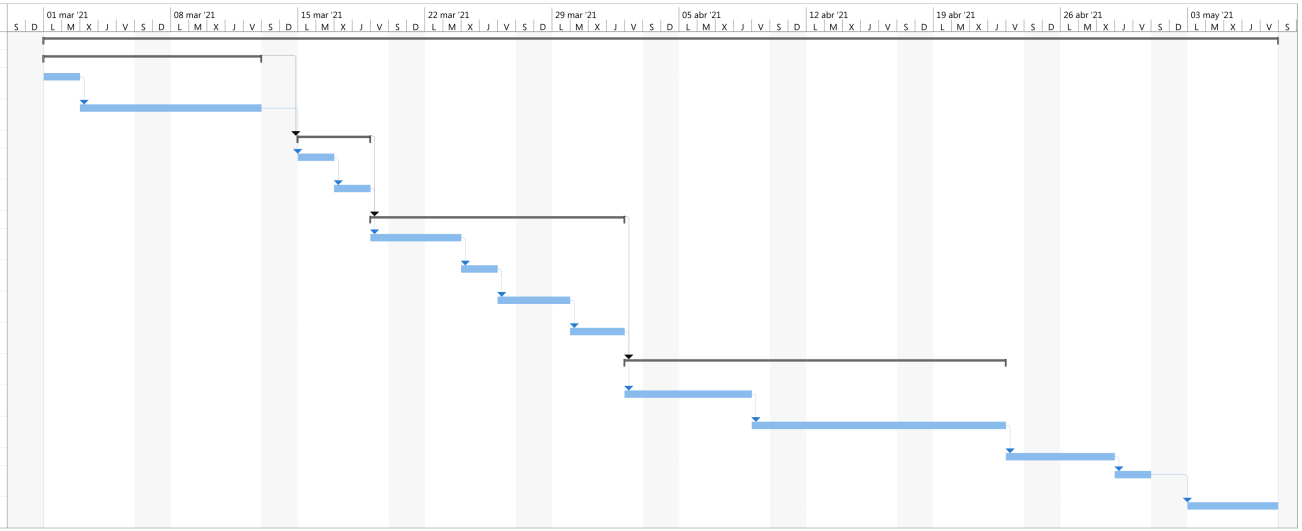

Figura 2.1: Diagrama de Gantt con la planificación inicial del proyecto

2.3. Restricciones

En esta sección mostramos las diferentes restricciones que hemos establecido para el proyecto, con el fin de conseguir lograr los objetivos y no desviarse del objetivo principal.

2.3.1. Restricciones temporales

Esta es la principal restricción del proyecto, ya que el mismo cuenta con unas horas determinadas por la universidad y que son 300 horas dedicadas a las prácticas. Por lo tanto una vez acabadas esas 300 horas debería cumplirse el objetivo final del proyecto.

2.3.2. Restricciones humanas

Las restricciones humanas aparecen debido a que el desarrollo del proyecto recae únicamente en tres personas, el alumno, el supervisor de la empresa y el tutor académico. De los cuales los dos últimos tienen la función de guiar al alumno en el desarrollo de su proyecto y el desarrollo de la memoria de prácticas, respectivamente.

2.3.3. Restricciones económicas

El proyecto a desarrollar no debe excederse en su coste, ya que de ser así la empresa no podría utilizar el producto debido a que podrían aparecer otros sistemas más económicos que resuelvan el problema.

2.3.4. Restricciones físicas

El proyecto a desarrollar debe ser colocado en una cabina de ascensor, y no debe ser ni muy voluminoso ni muy pesado. Debe tener un tamaño reducido para poder instalarlo con comodidad en cualquier cabina de ascensor sin importar el modelo ni el fabricante.

2.4. Gestión de riesgos

Esta sección está dedicada a los posibles riesgos que pueden aparecer durante el desarrollo del proyecto. Es una parte muy importante de la planificación y lo tenemos que tener en cuenta durante todo el desarrollo del proyecto.

2.4.1. Identificación de riesgos

A continuación mostramos los posibles riesgos que pueden aparecer durante el desarrollo del proyecto.

- R01 - Objetivos poco claros
- R02 - Abandono temporal de un miembro del equipo
- R03 - Falta de experiencia
- R04 - Mala estimación de la duración de las tareas

2.4.2. Análisis de riesgos

En el Cuadro 2.2 observamos el análisis de riesgos, indicando la magnitud, que indica el grado en el que el riesgo puede afectar al proyecto, la descripción y los indicadores, que es la situación que nos puede alertar de la presencia de ese riesgo.

ID	Análisis del Riesgo
R01	**Magnitud:** Medio **Descripción:** No se tienen claros los objetivos del sistema, por lo que al finalizar el proyecto no cumplirá con el sistema que se pretendía realizar. **Indicadores:** Una vez se conoce el proyecto que se quiere realizar no se indican con propiedad las funcionalidades.
R02	**Magnitud:** Bajo **Descripción:** Un miembro del equipo se da de baja temporalmente, ya sea debido a una enfermedad o a cualquier otra causa, por lo que se tiene que alargar la duración del desarrollo. **Indicadores:** Riesgo sin indicadores, difícil de predecir la causa.
R03	**Magnitud:** Alto **Descripción:** No se tiene experiencia suficiente, ya sea con el entorno de desarrollo, con el lenguaje a utilizar o con cualquier tecnología a emplear. **Indicadores:** La planificación inicial no coincide con el desarrollo real del proyecto.
R04	**Magnitud:** Medio/Alto **Descripción:** La planificación inicial no se realiza correctamente, y por tanto las tareas no tienen la duración prevista. **Indicadores:** La planificación inicial no coincide con el desarrollo real del proyecto.

Cuadro 2.2: Análisis de riesgos

2.5. Estimación de recursos y costes del proyecto

En esta sección detallaremos los diferentes costes y recursos del proyecto, tanto costes materiales como costes humanos.

2.5.1. Recursos Hardware y Software

En el Cuadro 2.3 detallamos los costes hardware necesarios para desarrollar el proyecto, incluimos tanto el material necesario para desarrollar el proyecto como también el propio material del proyecto.

Costes Hardware				
Recurso	**Cantidad**	**Coste por unidad (€)**	**Total (€)**	**Suponiendo una vida útil de 5 años (€/mes)**
Ordenador trabajo	1	300	300	5
Pantalla 22’	2	120	240	4
Ratón y teclado	1	20	20	0,4
Atom lite	2	7.5	15	0,25
Atom BMP280	2	4	8	0,14
Módulo Radio	2	2	4	0,06
Total (€)			587	9,85

Cuadro 2.3: Detalle de los recursos hardware utilizados

En cuanto a los recursos software utilizados, cabe destacar que no aparece ningún gasto, ya que todo el software utilizado es de código libre y por lo tanto gratuito. El software utilizado es el Sistema Operativo Arch Linux, el entorno de desarrollo Visual Studio Code y el Sistema Operativo FreeRTOS.

2.5.2. Recursos humanos

En este apartado calcularemos los recursos humanos utilizados para la realización del proyecto en las 300 horas de duración del mismo. Para ello tomamos de referencia el estudio realizado por la Universidad Europea [6] en el que muestra que un estudiante recién graduado obtiene un salario que ronda los 18.000 y 22.000 € anuales. Lo cual supone unos 1800 € brutos mensuales. Si suponemos la jornada laboral de 8 horas de lunes a viernes obtenemos una cantidad de unos 11,25 € por hora aproximadamente.

Tenemos que tener en cuenta también las horas desarrolladas por el supervisor de prácticas que aunque son menos, tienen una mayor importancia, ya que el salario es mayor al de un estudiante recién graduado. Si suponemos un sueldo medio mensual de unos 4000 €, hace una total de unos 25 € la hora.

En el Cuadro 2.4 vemos un desglose aproximado de las horas realizadas por cada miembro y el total en € aproximado del coste humano del proyecto.

Costes Humanos			
Recurso	**Cantidad de horas**	**Coste por hora**	**Total (€)**
Estudiante en prácticas	300	11.25	3375
Supervisor de empresa	100	25	2500
Total			5875

Cuadro 2.4: Costes humanos del proyecto

2.5.3. Gastos totales

Además de los gastos tecnológicos y humanos, aparecen otros gastos indirectos que en una situación normal sería la empresa la que se hiciese cargo de los mismos. Pero debido a la situación sanitaria del COVID-19 hemos desarrollado el proyecto mediante teletrabajo.

Estos gastos están recogidos en el Cuadro 2.5, donde también aparece el total de gastos teniendo en cuenta los humanos, tecnológicos y los indirectos. Tenemos un gasto total de unos 6552 € aproximadamente.

Costes Totales	
Recurso	**Total (€)**
Recursos hardware	587
Recursos humanos	5875
Coste electricidad	30
Coste internet	60
Total	6552

Cuadro 2.5: Costes totales proyecto

2.6. Seguimiento del proyecto

Esta sección está dedicada al seguimiento del proyecto, explicamos los retrasos o adelantos que ha habido según la planificación inicial. En el Cuadro 2.6 aparece la comparación entre la estimación inicial de las tareas y la duración real.

Como vemos en el Cuadro 2.6 no hemos podido completar la totalidad de las tareas y por tanto no hemos cumplido todos los objetivos del proyecto. Podemos observar que tres tareas han durado bastante más de lo esperado respecto a la planificación realizada inicialmente. A continuación explicamos el motivo por el que estas tres tareas son las que más retraso han provocado.

TAREA	ESTIMACIÓN	REAL	
a) Estudio del entorno de trabajo	60h	60h	0
a.1) Instalación y configuración del entorno de desarrollo	12h	12h	0
a.2) Familiarizarse con el entorno de desarrollo	48h	48h	0
b) Estudio del microcontrolador	24h	14h	-14
b.1) Investigar y elegir el microcontrolador	12h	2h	-10
b.2) Familiarizarse con el microcontrolador	12h	12h	0
c) Estudio del sensor	60h	146h	+86
c.1) Familiarizarse con los buses de comunicación	18h	8h	-10
c.2) Investigar y evaluar los distintos sensores	12h	60h	+48
c.3) Investigar el protocolo de comunicación del sensor	12h	8h	-4
c.4) Conexionado y configuración de sensores	18h	70h	+52
d) Programación del microcontrolador y sensores	90h	70h	-20
d.1) Programación del módulo del sensor	30h	70h	+40
d.2) Programación del algoritmo de entrenamiento	60h	0h	-60
e) Pruebas de funcionamiento	24h	8h	-16
f) Relacionar datos con posición real	12h	0h	-12
g) Desarrollar el envío de los datos al servidor	30h	2h	-28
		300 h	

Cuadro 2.6: Duración real de las tareas

2.6.1. Principales retrasos respecto a la planificación inicial

En este apartado detallamos los tres principales retrasos que han supuesto no poder cumplir con la totalidad de los objetivos planteados inicialmente.

Investigar y evaluar los distintos sensores (Tarea c.2)

La tarea c2, investigar y evaluar los distintos sensores, tenía una planificación estimada de 12 horas, sin embargo la duración real de la tarea fue de 60 horas aproximadamente. Lo cual supuso un retraso como indica el Cuadro 2.6 de 48 horas.

El retraso fue debido a dos circunstancias, la primera era que para el proyecto pretendíamos usar un acelerómetro como sensor principal y apoyar la medición del acelerómetro con un sensor de presión. Sin embargo por problemas logísticos no dispusimos del acelerómetro a tiempo, ya que llegó la última semana de prácticas. Así pues, el proyecto lo desarrollamos usando únicamente el sensor de presión atmosférica. Es aquí cuando apareció el segundo problema, realizando las pruebas del sensor de presión nos dimos cuenta de que tenía un error de 1 metro de precisión. No obstante si dejábamos el sensor realizando mediciones durante un día entero en un punto fijo, la altura oscilaba en 15 metros de diferencia. Esto se debe a la aparición de borrascas o anticiclones y los cambios de presión. Tuvimos que investigar qué alternativas podíamos tomar y aparecieron dos:

1. La primera tener dos sistemas, uno en un punto fijo (cota) y el otro en la cabina. Y calcular

la diferencia de presiones de un punto a otro.

2. La otra opción era detectar cuándo estaba quieta la cabina (usando el acelerómetro) y de este modo, el cambio de presión sabríamos era debido al clima. En cambio, si detectaba movimiento sabíamos que el cambio de presión era debido a la altura.

De las dos opciones, tuvimos que seleccionar la primera, ya que no disponíamos del acelerómetro y además nos parecía que podía ser interesante para el trabajo, ya que teníamos que usar una nueva tecnología para comunicar los dos sistemas. Esta decisión la tomamos junto al supervisor.

Conexionado y configuración de sensores (Tarea c.4)

Otro retraso importante fue el de la tarea c4, conexionado y configuración de sensores. Este retraso apareció debido a dos problemas principalmente. Y como podemos ver en el Cuadro 2.6, supuso un retraso de 52 horas.

1. El primero era debido a la compilación del firmware. Para desarrollar el firmware usamos Espressif IDF y el sistema operativo FreeRTOS. El problema aparece debido a la curva de aprendizaje de Espressif IDF. Utilizamos el componente “i2c” y el componente “bmp280” pero no conseguíamos hacer compilar el firmware. Este fue uno de los principales problemas.

2. El segundo problema va un poco relacionado con el primero. Fue debido al flasheo del firmware en el Atom Lite. Al intentar flashear aparecía un error constante que no dejaba escribir en la memoria del Atom Lite. Probamos a cambiar la velocidad de flasheo del Atom desde el IDE de Visual Studio, pero el problema seguía apareciendo. Después de investigar, y buscar soluciones, encontramos el problema. Existe una especie de bug el cual no detecta que le estás cambiando la velocidad de flasheo desde Visual Studio. Y conseguimos, por fin, cambiar dicha velocidad desde un fichero de configuración del propio IDE.

Programación del módulo del sensor (Tarea d.1)

En el desarrollo de esta tarea no apareció ningún problema que impidiera continuar con el proyecto, sin embargo tuvimos que dedicarle más horas de las estimadas a la programación del módulo del sensor. Y aparte de este al tener que desarrollar el software del nuevo módulo de radiofrecuencia también retrasó respecto a la planificación inicial.

2.6.2. Objetivos iniciales no cumplidos

Finalmente de los siete objetivos iniciales, no pudimos cumplir los dos últimos por falta de tiempo, estos son:

- Lograr relacionar los datos del sensor con la posición de la cabina.
- Añadir al desarrollo la lectura de datos de los sensores y el envío de los datos al servidor.

Capítulo 3

Análisis y diseño del sistema

Índice

En este capítulo explicamos los requisitos del proyecto, que son las funcionalidades que debe cumplir el sistema. También mostraremos los casos de uso del sistema. Finalmente detallaremos los componentes utilizados y la arquitectura del sistema.

3.1. Análisis del sistema

En esta sección mostraremos los requisitos y los casos de uso del sistema.

3.1.1. Requisitos

Los requisitos del sistema son las necesidades o funcionalidades del producto o servicio a desarrollar. Los requisitos establecen qué debe hacer el sistema, pero no deben indicar cómo hacerlo. Los requisitos son necesarios para poder conseguir el objetivo del proyecto de manera adecuada. Una vez finalizado el proyecto, el mismo debe cumplir con los requisitos establecidos:

- Determinar la altura de la cabina.
- Permitir a un usuario conocer la parada actual de la cabina.

- Permitir al técnico conocer la altura exacta de la cabina para realizar la calibración de la misma.
- Poder configurar el sistema para cada edificio estableciendo el número de paradas y altura inicial/final.

3.1.2. Casos de uso

El diagrama de casos de uso consiste en una descripción de las funcionalidades que el sistema debe cumplir. Los personajes o entidades que participan en el diagrama de casos de uso se denominan actores, y las actividades que estos realizan son los casos de uso. Por tanto, el diagrama de casos de uso contiene una serie de actores realizando acciones. El diagrama de casos de uso ha sido realizado mediante la herramienta MagicDraw [16].

En la Figura 3.1 podemos ver el diagrama de casos de uso del proyecto que la empresa quiere desarrollar. Más adelante explicamos los diferentes actores y los casos de uso detallados.

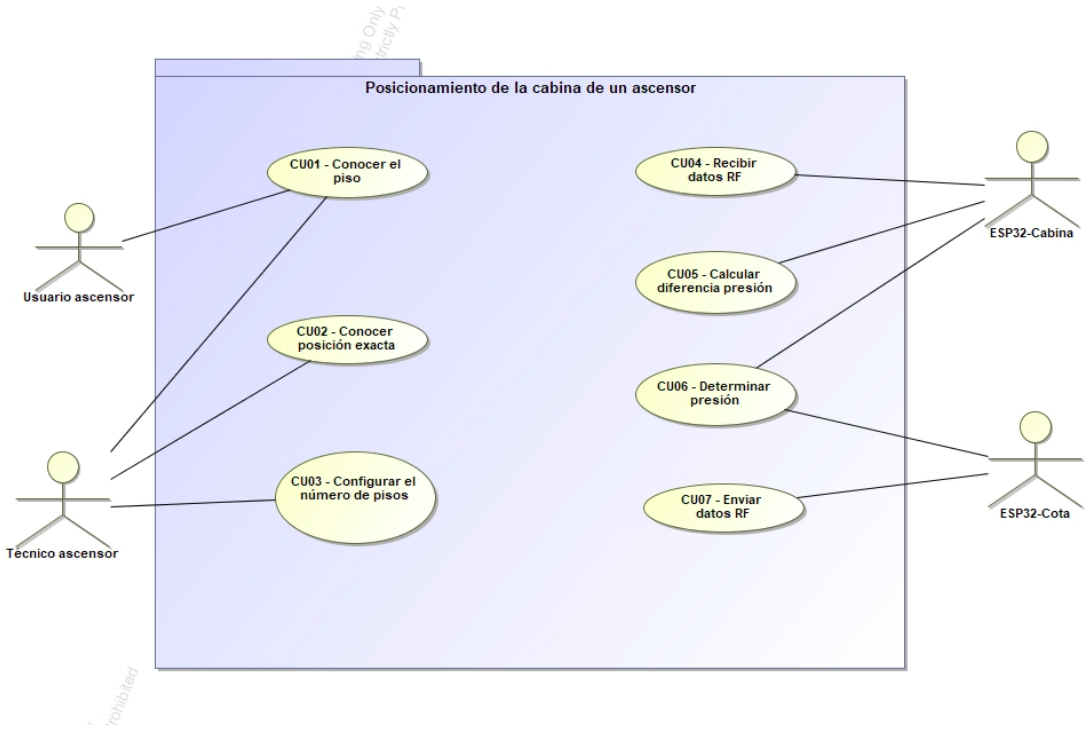

Figura 3.1: Casos de uso

Descripcion de los actores

- **Usuario ascensor:** El usuario del ascensor representa a un usuario estándar que utiliza el ascensor y quiere saber en un momento dato la planta actual de la cabina.
- **Técnico ascensor:** El técnico del ascensor representa al ascensorista que quiere saber

con exactitud la altura de la cabina para así poder calibrarla.

- **ESP32 - Cota:** El ESP32 de cota es el que se coloca en un punto de referencia conocido. De este modo sabiendo la altura y la presión atmosférica en este punto, el último actor que falta por definir, el ESP32 de cabina es capaz de determinar su altura.
- **ESP32 - Cabina:** El ESP32 de la cabina es el que recibe la presión atmosférica en la cota y de este modo utilizando también su presión atmosférica es capaz de determinar la altura de la cabina.

Casos de uso

- CU01 Conocer el piso, cuya descripción aparece en el Cuadro 3.1.
- CU02 Conocer la posición exacta, cuya descripción aparece en el Cuadro 3.2.
- CU03 Configurar el número de pisos, cuya descripción aparece en el Cuadro 3.3.
- CU04 Recibir datos mediante Radiofrecuencia (RF), cuya descripción aparece en el Cuadro 3.4.
- CU05 Calcular la diferencia de presión, cuya descripción aparece en el Cuadro 3.5.
- CU06 Determinar la presión, cuya descripción aparece en el Cuadro 3.6.
- CU07 Enviar datos mediante Radiofrecuencia (RF), cuya descripción aparece en el Cuadro 3.7.

Descripción de los casos de uso

CU01 Conocer el piso	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: El usuario del ascensor desea conocer la parada actual de la cabina del ascensor **Cómo se inicia:** El usuario llega a una planta del edificio o a la propia cabina	
Pasos del escenario 1: 1. El usuario llega al ascensor. 2. El sistema actualiza constantemente la pantalla indicadora del piso.	
Comentarios: El muestreo del piso actual de la cabina está fuera del alcance del proyecto, pero es un trabajo futuro.	

Cuadro 3.1: Caso de uso CU01 Conocer el piso

CU02 Conocer la posición exacta	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: El técnico del ascensor necesita conocer la posición exacta de la cabina del ascensor **Cómo se inicia:** El técnico del ascensor llega a la maniobra del ascensor	
Pasos del escenario 1: 1. El técnico conecta su ordenador al sistema. 2. El sistema debe facilitarle la ubicación exacta de la cabina en tiempo real.	
Comentarios: El proceso por el cual el técnico se conecta al sistema y este le ofrece la posición exacta de la cabina está fuera del alcance del proyecto, pero es un trabajo futuro.	

Cuadro 3.2: Caso de uso CU02 Conocer la posición exacta

CU03 Configurar el número de pisos	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: El técnico configura el número de pisos en el sistema, además de configurar la altura del microcontrolador situado en la cota **Cómo se inicia:** El técnico del ascensor llega a la maniobra del ascensor	
Pasos del escenario 1: 1. El técnico conecta su ordenador al sistema. 2. El técnico configura el número de pisos. 3. El técnico configura la altura de cota.	
Comentarios: La configuracion de la cabina está fuera del alcance del proyecto, pero es un trabajo futuro.	

Cuadro 3.3: Caso de uso CU03 Configurar el número de pisos

CU04 Recibir datos mediante Radiofrecuencia (RF)	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: El microcontrolador de cabina recibe la presión atmosférica leída por el microcontrolador de cota mediante el módulo de radiofrecuencia. **Cómo se inicia:** Cada ciclo de ejecución recibe los datos	
Pasos del escenario 1: 1. El microcontrolador espera recibir la presión. 2. El microcontrolador recibe la presión.	
Comentarios: En caso de que no reciba datos, se queda esperando hasta que reciba la presión.	

Cuadro 3.4: Caso de uso CU04 Recibir datos mediante RF

CU05 Calcular la diferencia de presión	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: Una vez el microcontrolador de cabina ha leído su presión atmosférica y ha recibido la presión atmosférica del microcontrolador de cota, procede a calcular la diferencia de presión, para así determinar la altura. **Cómo se inicia:** Después de recibir la presión atmosférica y leer su presión atmosférica	
Pasos del escenario 1: 1. El microcontrolador lee su presión atmosférica. 2. El microcontrolador recibe la presión atmosférica del sensor de cota. 3. El microcontrolador calcula la diferencia teniendo en cuenta la altura del sensor de cota.	
Comentarios: En el proyecto se muestra la diferencia de presión entre los dos puntos. No se llega a calcular la altura ya que no ha habido tiempo para desarrollar esa parte.	

Cuadro 3.5: Caso de uso CU05 Calcular la diferencia de presión

CU06 Determinar la presión	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: Cada microcontrolador lee los datos del sensor de presión atmosférica. **Cómo se inicia:** Se realiza de manera constante	
Pasos del escenario 1: 1. El microcontrolador espera a que el sensor esté listo para la lectura. 2. El microcontrolador realiza la lectura de la presión atmosférica.	
Comentarios: La lectura de la presión atmosférica incluye un pequeño error.	

Cuadro 3.6: Caso de uso CU06 Determinar la presión

CU07 Enviar datos mediante Radiofrecuencia (RF)	
Autor: Joaquín González Álvarez **Revisor:** Diego Centelles Beltrán	**Fecha creación:** 12/04/2021 **Fecha revisión:** 14/04/2021 **Fecha aprobación:** 15/04/2021 **Versión:** 1.0
Descripción: El microcontrolador situado en la cota envía mediante RF la presión atmosférica al microcontrolador situado en cabina. **Cómo se inicia:** Después de cada lectura de presión.	
Pasos del escenario 1: 1. El microcontrolador envía la presión obtenida mediante el sensor.	
Comentarios: El envío se realiza sin ACK por lo que el microcontrolador envía la presión constantemente sin detectar si el receptor está activo.	

Cuadro 3.7: Caso de uso CU07 Enviar datos mediante RF

3.2. Diseño de la arquitectura del sistema

El diseño de la arquitectura del sistema consiste en un esquema global de la estructura del sistema, que usamos para desarrollar los componentes. En el Apartado 3.2.1 vamos a ver con cierto detalle los componentes del sistema y en el Apartado 3.2.2 veremos las relaciones entre ellos con el esquema de la arquitectura. En el capítulo 4 vemos en más detalle la implementación y explicamos algún componente más a fondo.

3.2.1. Componentes del sistema

En este apartado describimos los componentes del sistema, no hacemos diferenciación entre software y hardware, ya que coinciden los módulos. Sin embargo en la arquitectura sí que hacemos una distinción entre el sistema "Cabina" y el sistema "Cota".

Existen los dos sistemas, ya que por diferentes problemas que explicamos en el capítulo 4 tuvimos que usar dos sistemas. Uno situado en una cota con una altura conocida, el cual le manda constantemente la presión leída al sistema situado en la cabina. El sistema de cabina también hace una lectura constante de la presión atmosférica y la compara con la recibida del sistema de cota. De este modo el sistema de cabina es capaz de calcular la diferencia de presiones y así realizar una estimación de la altura.

A continuación describimos los componentes de los dos sensores.

Sensor

El componente Sensor es el encargado de controlar el sensor de presión BMP280. Inicializa los valores de los registros del sensor con los valores por defecto. En el capítulo 4 explicamos los diferentes valores que puede tomar el registro, sin embargo los mejores resultados se obtuvieron con los valores por defecto. Este componente contiene una estructura de datos en forma de cola, realiza una lectura de la presión y la envía esa lectura por la cola.

Main

El componente Main es el componente principal, que se interconecta al resto de componentes. El componente crea diferentes colas para comunicarse con el resto de componentes y permitir el intercambio de los datos leídos.

RF (Radiofrecuencia)

El componente RF es el encargado de la comunicación por radiofrecuencia entre los dos sistemas. En el sistema de cota envía los datos por el dispositivo de radiofrecuencia y estos son recibidos por el sistema de cabina.

WiFi-TCP

El componente de WiFi-TCP es el encargado de enviar los datos al servidor. El sistema de cabina es el único que tiene activo este componente y es el que manda al servidor la diferencia de altura calculada. Cabe destacar que este componente no lo hemos podido desarrollar por falta de tiempo. Sin embargo, sí que hicimos una serie de pruebas. Para ello cogimos un ejemplo de internet para realizar las pruebas de funcionamiento.

3.2.2. Arquitectura del sistema

Mostramos la arquitectura utilizada tanto en el sistema de cota como en el sistema de cabina. En la Figura 3.2 podemos ver la arquitectura general. En cambio, en las Figuras 3.3 y 3.4 vemos la arquitectura específica de cada sistema.

Arquitectura general del sistema

En la Figura 3.2 observamos las diferentes relaciones que tienen los componentes del sistema. Partimos del componente Main que es el que contiene el programa principal y es el encargado de relacionar el resto de componentes. A la izquierda vemos el componente Sensor, que se encarga de interactuar con el sensor BMP280 y leer los datos de presión atmosférica. En la parte superior

se encuentra el módulo RF, que es el encargado de la comunicación mediante radiofrecuencia entre los dos sistemas (sistema de cota y sistema de cabina). Finalmente el módulo WiFi-TCP es el encargado de enviar los datos al servidor.

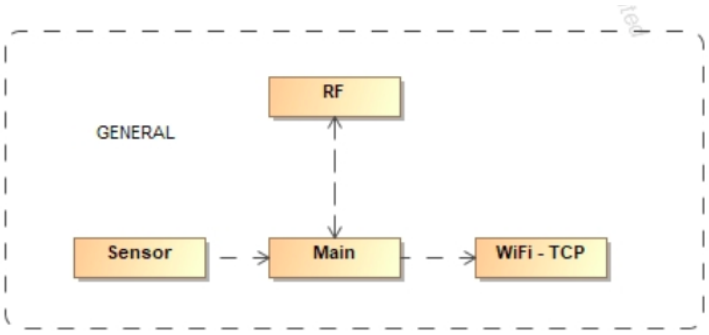

Figura 3.2: Arquitectura general del sistema

Una vez vista la arquitectura general, vamos a ver en detalle la arquitectura específica de los dos sistemas. Ambos sistemas parten del mismo esquema, pero tienen ligeras variaciones.

Arquitectura del sistema de cabina

Como observamos en la Figura 3.3, la arquitectura del sistema de cabina incluye todos los componentes. Cabe destacar que hay una diferencia en cuanto al componente de radiofrecuencia respecto al sistema de cota, ya que el flujo de información entre el componente de radiofrecuencia y Main fluye en sentido descendiente. Esto se debe a que el sistema de cabina utiliza el componente de radiofrecuencia para recibir datos. El sistema de cabina es el que utiliza el microcontrolador ESP32 Dev Kit.

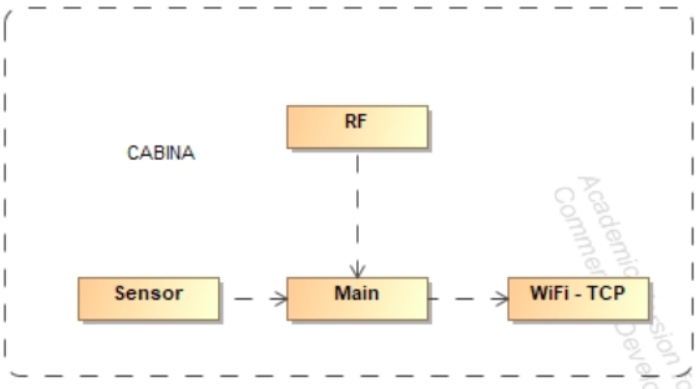

Figura 3.3: Arquitectura del sistema de cabina

Arquitectura del sistema de cota

En la Figura 3.4 vemos que no incluye el componente WiFI, ya que no hace uso de él, y el flujo de información entre Main y radiofrecuencia fluye en sentido ascendente, ya que el sistema envía datos mediante radiofrecuencia. El sistema de cota es el que utiliza el microcontrolador ESP32 Atom Lite.

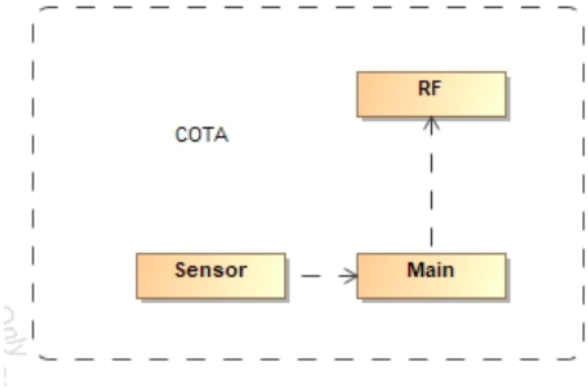

Figura 3.4: Arquitectura del sistema de cota

Capítulo 4

Implementación y pruebas

Índice

En este capítulo explicamos los detalles de implementación más importantes así como también las diferentes pruebas realizadas con los sistemas de sensores.

4.1. Detalles de implementación

En esta sección describimos, en primer lugar, el funcionamiento del componente Main, posteriormente explicamos el funcionamiento del componente del sensor BMP280 y finalmente terminamos con el componente encargado de la gestión de radiofrecuencia.

4.1.1. Componente Main

En este apartado vemos la estructura del método Main del sistema de cabina. El sistema de cota tiene un método Main parecido, sin embargo el sistema de cota realiza la tarea de enviar por radiofrecuencia en lugar de recibir.

En la Figura 4.1 podemos ver el código del método Main del sistema de cabina. Las primeras líneas sirven para configurar el microcontrolador, son funciones que vienen implementadas en

el sistema operativo FreeRTOS. Después creamos el objeto FS1000A que es el encargado de la comunicación por radiofrecuencia, en este caso el sistema de cabina solo instancia la tarea de recepción mientras que el sistema de cota instancia la tarea de envío. Creamos dos colas, la primera para la comunicación con el sensor y la siguiente para comunicarse con el módulo WiFi-TCP. Y finalmente creamos las tareas del sensor y la de recepción mediante radiofrecuencia y le pasamos los parámetros necesarios. La tarea de recepción la describimos más adelante para explicar el módulo de radiofrecuencia. Cabe destacar que no hacemos uso del módulo WiFi-TCP, ya que no hemos tenido tiempo de implementarlo y comprobar que funcione correctamente.

```
void app_main(void)
{

    ESP_ERROR_CHECK(nvs_flash_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());

    ESP_ERROR_CHECK(esp_netif_init()); // NOT WORKING
    ESP_ERROR_CHECK(example_connect()); // NOT WORKING

    FS1000A_t *ctx = fs1000a_create(TX_DATA, RX_DATA);

    QueueHandle_t queueSensor =xQueueCreate( 1, sizeof(float)*12 );
    QueueHandle_t queueTCP =xQueueCreate( 10, sizeof(float)*12 );

    struct params *par= malloc(sizeof *par);;
    par->ctx=ctx;
    par->queueTCP=queueTCP;
    par->queueSensor=queueSensor;

    xTaskCreate(tcp_client_task, "tcp_client", 8192, queueTCP, 5, NULL); // NOT WORKING
    ESP_LOGI(TAG_MAIN, "Created task tcp"); // NOT WORKING

    xTaskCreate(sensor, "sensor", 8192, queueSensor, 5, NULL);
    ESP_LOGI(TAG_MAIN, "Created task sensor");

    xTaskCreate(rx_pressure_task, "rx_pressure_task", 2048, par, tskIDLE_PRIORITY, NULL);
    ESP_LOGI(TAG_MAIN, "Created rx_pressure_task");

}
```

Figura 4.1: Código del método Main del sistema de cabina

El sistema de cota tiene un método Main prácticamente igual, simplemente en lugar de crear la tarea de recepción de mensajes, crea la tarea de envío de mensajes.

4.1.2. Componente del Sensor BMP280

El sensor BMP280 es el sensor de presión atmosférica que hemos usado en el desarrollo del proyecto. El sensor tiene unas pequeñas dimensiones y muy bajo consumo. El sensor se conecta mediante la interfaz I2C explicada anteriormente. En al Anexo podemos ver imágenes del sensor BMP280 con el encapsulado de plástico (M5Stack) y sin el encapsulado.

El sensor cuenta con tres modos de funcionamiento: el modo sleep, el modo normal y el

modo forzado. En el modo sleep no se pueden realizar mediciones mientras que en el modo normal se hacen una serie de lecturas cíclicas. Y en el modo forzado se realiza una lectura a petición. En la implementación hemos optado por realizar mediciones cíclicas, ya que cuantas más mediciones hagamos, obtenemos una mejor precisión.

En la Figura 4.2 podemos ver los ajustes óptimos para el correcto funcionamiento según la aplicación que se le quiera dar al sensor. Como vemos hay un caso específico para detectar el cambio de planta de un ascensor, en que indica el modo de funcionamiento y los diferentes filtros y registros que hay que configurar en el sensor.

Para configurar esos ajustes, hay que usar los registros de configuración del sensor. En la Figura 4.3 podemos ver el conjunto de registros del sensor. Los seis primeros registros son solo de lectura. Por ejemplo, el registro 0xF7 almacena los 8 bits más significativos de la presión mientras que el registro 0xF8 almacena los 8 bits menos significativos

Del resto de registros, los dos registros que hay que modificar para conseguir el funcionamiento adecuado son: el registro "config" (0xF5) y el registro "ctrl-meas" (0xF4).

Use case	Mode	Over-sampling setting	osrs_p	osrs_t	IIR filter coeff. (see 3.3.3)	I_{DD} [µA] (see 3.7)	ODR [Hz] (see 3.8.2)	RMS Noise [cm] (see 3.5)
handheld device low-power (e.g. Android)	Normal	Ultra high resolution	×16	×2	4	247	10.0	4.0
handheld device dynamic (e.g. Android)	Normal	Standard resolution	×4	×1	16	577	83.3	2.4
Weather monitoring (lowest power)	Forced	Ultra low power	×1	×1	Off	0.14	1/60	26.4
Elevator / floor change detection	Normal	Standard resolution	×4	×1	4	50.9	7.3	6.4
Drop detection	Normal	Low power	×2	×1	Off	509	125	20.8
Indoor navigation	Normal	Ultra high resolution	×16	×2	16	650	26.3	1.6

Figura 4.2: Filtros recomendados del sensor según el ajuste

En el registro 0xF4, los tres bits más significativos controlan el oversampling[1] deseado para la temperatura, con los siguientes tres bits se controla el oversampling de la presión y con los últimos dos bits se establece el modo de funcionamiento del sensor. Aumentando el oversampling de la presión conseguimos mejorar la resolución de la presión. Por ejemplo, con un oversampling x1 conseguimos una resolución de 16 bits, mientras que si lo aumentamos a x16 conseguimos 20 bits. Aumentando el oversampling conseguimos ser más precisos.

El registro 0xF5, con los tres bits más significativos controlamos el tiempo de standby en

[1]El oversampling es el proceso de muestreo de una señal a una frecuencia significativamente mas alta que la frecuencia de Nyquist. La frecuencia de Nyquist es el doble del ancho de banda de la señal.

el modo de funcionamiento normal, con los siguientes tres bits se establece el filtro IIR, el bit uno no se utiliza y el último bit se utiliza para otro tipo de comunicación, en nuestro caso como utilizamos la interfaz I2C tampoco lo utilizamos. El filtro IIR se utiliza para filtrar el ruido que ocurre al cerrar una puerta, o los cambios de aire, por ejemplo. Según el coeficiente del filtro, éste se aplicará con mayor o menor medida.

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x58
calib25...calib00	0xA1...0x88	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

Figura 4.3: Mapa de registros del sensor BMP280

A continuación, en la Figura 4.4 podemos ver el fragmento de código utilizado para cambiar los valores de los registros. Por ejemplo, en el registro 0xF4 asignamos el valor 55 (00110111 en binario). Vemos en la Figura 4.3 que los 3 primeros bits (001) se corresponden al oversampling de la temperatura, los 3 bits siguientes (101) al oversampling de la presión y los últimos 2 bits (11) al modo de funcionamiento del sensor. En la Figura 4.5 podemos ver que la configuración de bits 101 corresponde al oversampling x16 de la presión. En el datasheet del sensor BMP280 [1] podemos ver la configuración de cada uno de los parámetros de configuración.

```
1  reg = 0xF4;
2  dat = 55; // 8'b00110111
3  bmp.write8(reg,dat);
4
5  reg =  0xF5;
6  dat = 72; // 8'b01001010
7  bmp.write8(reg,dat);
```

Figura 4.4: Código con la configuración de registros del sensor BMP280

osrs_p[2:0]	Pressure oversampling
000	Skipped (output set to 0x80000)
001	oversampling ×1
010	oversampling ×2
011	oversampling ×4
100	oversampling ×8
101, Others	oversampling ×16

Figura 4.5: Configuración del campo osrs_p del registro 0xF4

4.1.3. Envío de mensajes por radiofrecuencia

En este apartado explicamos el protocolo de envío de mensajes usando el módulo de radiofrecuencia (RF) y la biblioteca RMT de Free RTOS.

Modulación OOK

Los módulos de radiofrecuencia funcionan mediante una onda sinusoidal que se corresponde con la onda portadora. Cuando el módulo de radiofrecuencia recibe un 1 lógico por el pin de datos, la portadora está activa, mientras que si recibe un 0 lógico, el módulo anula la portadora, por lo que el emisor no estaría emitiendo nada. A esto se le denomina modulación OOK (On Off Keying) [17], que es una variante de la modulación ASK (Amplitude-shift keying) [18] y que funciona de la manera anteriormente descrita.

En la Figura 4.6 podemos ver cómo funciona esta modulación: en el centro de la figura vemos la señal portadora sin modificar, en la parte superior vemos la trama digital de datos que queremos enviar, que es la que le pasamos al módulo. Y finalmente en la parte inferior vemos la señal portadora modificada mediante la modulación OOK, que es la señal que se transmite por el medio físico mediante los módulos de radiofrecuencia.

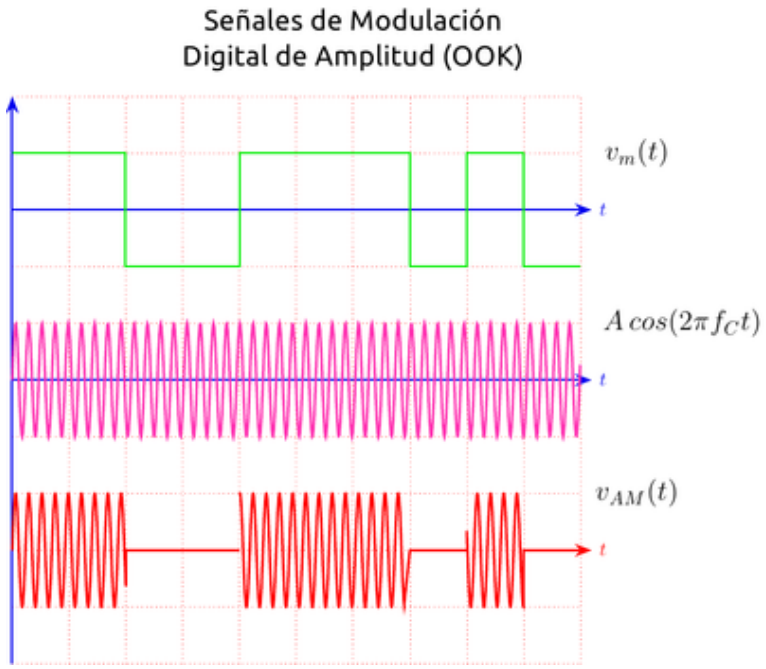

Figura 4.6: Modulación On Off Keying

Pues bien, una vez sabemos cómo funciona la modulación OOK, procederemos a explicar el funcionamiento del protocolo de envío de mensajes.

Protocolo de comunicación por radiofrecuencia

El protocolo para realizar el envío de mensajes consta de las siguientes etapas:

1. **Sincronización:** En primer lugar se realiza una etapa de sincronización, necesaria para que el receptor ajuste la velocidad de envío a la establecida por el emisor. Esta etapa realiza una batería de cambios de ceros y unos lógicos constante, de una duración de 500 microsegundos. Para ello se envían 12 unos y 12 ceros intercalados.

2. **Delimitador:** A continuación se envían dos ceros seguidos con una duración de 500 microsegundos. El delimitador se utiliza para indicar al receptor que ya ha finalizado la sincronización y empieza la transmisión de datos.

3. **Byte de tamaño:** Una vez se ha enviado el delimitador, se envía un byte que indica el tamaño del mensaje. Para enviar un byte cualquiera, tanto el byte de tamaño como cada byte de datos, se realiza una pequeña resincronización cada 4 bits. Es decir, se envía una secuencia de unos y ceros igual que en la etapa de sincronización, pero de menor duración, esta vez son 4 unos y 4 ceros intercalados. A continuación se envían los 4 bits más significativos (MSB), se repite la resincronización y se envían los 4 bits menos significativos (LSB).

4. **Bytes de datos:** Con los bytes de datos ocurre igual que con el byte de tamaño, por cada byte se realiza el mismo procedimiento. Empieza la resincronización, se envían los 4 bits más significativos, se repite la resincronización y por último se envían los 4 bits menos significativos.

5. **CRC (Comprobación Redundancia Cíclica):** Por último, se envían dos bytes de crc. Se utiliza crc16, es decir 16 bits de crc para comprobar errores en la transmisión y asegurar la integridad del paquete. Cada uno de los dos bytes de crc también van precedidos de los bits de sincronización.

Una vez explicado el funcionamiento del protocolo de comunicación por radiofrecuencia, vemos las partes del código que realizan esos pasos y analizaremos una trama analógica capturada mediante el analizador lógico de Saleae [13].

El primer método que analizamos es el método "prepare sync", que se encuentra ver en la Figura 4.7. Este método coincide con la etapa de sincronización, se crea el objeto RMT[2] con la secuencia de unos y ceros intercalado y se indica la duración de cada uno, siendo en este caso la duración de todos ellos de 500 microsegundos.

```
1  void prepare_sync(FS1000A_t *ctx, uint16_t *offset) {
2
3    rmt_item32_t sync_buf[] = {
4        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
5        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
6        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
7        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
8        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
9        {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
10       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
11       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
12       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
13       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
14       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
15       {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
16   };
17
18   uint16_t size = sizeof(sync_buf) / sizeof(rmt_item32_t);
19
20   ESP_ERROR_CHECK(rmt_fill_tx_items(RMT_TX_CHANNEL, sync_buf, size, *offset));
21
22   *offset += size;
23 }
```

Figura 4.7: Método "prepare sync"

El siguiente método a analizar es el método "prepare del", que lo podemos ver en la Figura 4.8. Este método coincide con la etapa de delimitador. Simplemente envía dos ceros seguidos con una duración total de 1 milisegundo. Indica que acaba la etapa de sincronización y comienzan a enviarse los datos.

[2]Un objeto RMT es una secuencia de pares de bits con la duracion de cada uno que utiliza la biblioteca RMT para mandar la trama lógica.

```
1  void prepare_del(FS1000A_t *ctx, uint16_t *offset) {
2
3    rmt_item32_t del_buf[] = {
4        {{{RMT_BIT_DURATION_US, 0, RMT_BIT_DURATION_US, 0}}},
5    };
6
7    uint16_t size = sizeof(del_buf) / sizeof(rmt_item32_t);
8
9    ESP_ERROR_CHECK(rmt_fill_tx_items(RMT_TX_CHANNEL, del_buf, size, *offset));
10
11   *offset += size;
12 }
```

Figura 4.8: Método “prepare del”

El método “prepare byte” que aparece en la Figura 4.9, es el método que utilizamos para transmitir un byte. Las etapas de envío del tamaño del mensaje, envío de los datos del mensaje y envío del crc usan este método, que consiste en: la resincronización con unos y ceros, el envío de los 4 bits más significativos, otra resincronización y por último, el envío de los 4 bits menos significativos.

Como podemos ver en la Figura 4.9, los 4 primeros bits (SYNC) son los de sincronización, representados por cuatro unos y cuatro ceros; y con una duración de medio microsegundo. A continuación, se envían los 4 bits más significativos (MSB), en la figura vemos que se inicializan a 0, pero es más tarde, a partir de la línea 25 cuando esos bits se intercambian por los bits que deseamos enviar. Luego se repite el patrón de sincronización (SYNC) y finalmente se añaden los 4 bits menos significativos (LSB), que al igual que con los bits más significativos, son añadidos a partir de la línea 25.

En la Figura 4.10 aparece la función que hace uso de los métodos descritos previamente y que lleva a cabo el protocolo de comunicación por radiofrecuencia. Como podemos observar el método empieza inicializando el offset a 0, este se irá incrementando según el tamaño de los datos que se vayan mandando. Posteriormente, en la línea 5 se ejecuta el método de sincronización, en la línea 6 el de delimitador y en la línea 7 se ejecuta el método “prepare byte” con el tamaño del mensaje. Luego en las líneas 9 y 10 se van enviando todos los datos del mensaje ejecutando en cada iteración del bucle el método “prepare byte”. Finalmente, calculamos la paridad CRC y enviamos los dos bytes de paridad y terminamos con otra sincronización para finalizar con la trama.

Finalmente, en la Figura 4.11 aparece la función de recepción de radiofrecuencia y que es la encargada también de calcular la diferencia de presión. Vemos en la línea 13 que emplea la función “fs1000a_recv_msg”, que funciona con el mismo protocolo de envío descrito anteriormente. Esta función, además, comprueba con los bits de crc si el mensaje ha llegado correctamente y si es así, devuelve un valor verdadero y devuelve el mensaje. Posteriormente, calcula la diferencia de presión realizando una resta entre la presión recibida por radiofrecuencia y la presión recibida por la cola, que es la presión leída por su sensor.

```
void prepare_byte(FS1000A_t *ctx, uint8_t byte, uint16_t *offset) {

  rmt_item32_t byte_buf[] = {
      // SYNC
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},

      // Msb
      {{{RMT_BIT_DURATION_US, 0, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 0, RMT_BIT_DURATION_US, 0}}},

      // SYNC
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 1, RMT_BIT_DURATION_US, 0}}},

      // Lsb
      {{{RMT_BIT_DURATION_US, 0, RMT_BIT_DURATION_US, 0}}},
      {{{RMT_BIT_DURATION_US, 0, RMT_BIT_DURATION_US, 0}}},
  };

  rmt_item32_t *msb = byte_buf + 4;
  rmt_item32_t *lsb = byte_buf + 10;
  msb->level0 = (byte & 0b10000000) > 0;
  msb->level1 = (byte & 0b01000000) > 0;
  (msb + 1)->level0 = (byte & 0b00100000) > 0;
  (msb + 1)->level1 = (byte & 0b00010000) > 0;

  lsb->level0 = (byte & 0b00001000) > 0;
  lsb->level1 = (byte & 0b00000100) > 0;
  (lsb + 1)->level0 = (byte & 0b00000010) > 0;
  (lsb + 1)->level1 = (byte & 0b00000001) > 0;

  uint16_t size = sizeof(byte_buf) / sizeof(rmt_item32_t);

  ESP_ERROR_CHECK(rmt_fill_tx_items(RMT_TX_CHANNEL, byte_buf, size, *offset));

  *offset += size;
}
```

Figura 4.9: Método “prepare byte”

Análisis de una trama de comunicacion

Una vez descritos los códigos del microcontrolador, pasamos a analizar una trama completa.

En la Figura 4.12 vemos la trama de comunicación al completo. Al principio de la trama aparece la etapa de sincronización, el delimitador de dos ceros, y a continuación se repite el patrón de resincronización y envío de medio byte. Sin embargo, vamos a ir analizando la trama paso por paso, para entender mejor la información de la misma.

En la Figura 4.13 podemos ver la etapa de sincronización, que tiene lugar desde el principio hasta la marca roja, esta marca indica el comienzo de la etapa del delimitador. En la marca verde acaba el delimitador y por tanto comienza la carga útil del mensaje, en primer lugar se manda el tamaño del paquete. Hasta la marca morada es la etapa de resincronización y desde

```
bool fs1000a_send_msg(FS1000A_t *ctx, rf_msg_t *msg) {

  uint16_t offset = 0;

  prepare_sync(ctx, &offset);
  prepare_del(ctx, &offset);
  prepare_byte(ctx, msg->size, &offset);

  for (int i = 0; i < msg->size; i++) {
    prepare_byte(ctx, msg->payload[i], &offset);
  }

  uint16_t crc = ~crc16_be((uint16_t)~0x0000, msg->payload, msg->size);
  prepare_byte(ctx, (crc >> 8) & 0xff, &offset);
  prepare_byte(ctx, crc & 0xff, &offset);
  prepare_sync(ctx, &offset);
  prepare_end(ctx, &offset);

  rmt_tx_start(RMT_TX_CHANNEL, true);
  rmt_wait_tx_done(RMT_TX_CHANNEL, 10000 / portTICK_PERIOD_MS);

  ESP_LOGD(TAG, "TX CRC %04X", crc);

  return true;
}
```

Figura 4.10: Función de envío de mensajes

la marca morada a la marca azul son los 4 bits más significativos del byte. Como vemos en la figura son cuatro ceros. Luego se repite el patrón y se ejecuta la resincronización y los 4 bits menos significativos. Si convertimos a binario esos bits vemos que el tamaño del paquete es 4. Por tanto, sabemos que los 4 bytes siguientes corresponden a bytes de datos. En el Cuadro 4.1 podemos ver el código binario del tamaño de paquete recibido.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	1	0	0
Decimal: 4							

Cuadro 4.1: Bits del tamaño del paquete recibido

En la Figura 4.14 vemos cómo se repite el patrón de resincronización y 4 bits de byte. Se reciben los 4 bytes de datos con 8 de resincronización. Como vemos el protocolo no es muy efectivo, ya que se envían demasiados bits de sincronización, pero tuvimos que programarlo de esa manera por un problema que se explicará en el apartado 4.2.1. Sin embargo, en la sección de trabajo futuro explicamos un método para resolver el problema, y poder realizar un protocolo mucho más eficiente.

Finalmente, en la Figura 4.15 vemos cómo se reciben los 2 bytes de crc, utilizando el mismo patrón y el final de la trama. Los 2 bytes de crc sirven para detectar errores y poder comprobar si el paquete ha llegado correctamente.

```
static void rx_pressure_task(void *arg)
{

  struct params *par= arg;
  FS1000A_t *ctx = par->ctx;
  QueueHandle_t xQueueSensor = par->queueSensor;

  rf_msg_t msg;
  bool ok;
  int count = 0;
  while (1)
  {
    ok = fs1000a_recv_msg(ctx, &msg);
    if (ok)
    {

      float cotaPressure = *((float *)msg.payload);
      float cabinaPressure;
      if(xQueueReceive(xQueueSensor,&cabinaPressure,10000/portTICK_RATE_MS)!=pdTRUE)
                        {
                        }

      float diffPressure=fabs(cotaPressure-cabinaPressure);
      printf("Presion cabina(sens): %f\n", cabinaPressure);
      printf("Presion cota(RX): %f\n", cotaPressure);
      printf("Diferencia presion: %f\n", diffPressure);


    }
  }
}
```

Figura 4.11: Función de recepción de mensajes

4.2. Pruebas

En esta sección mostramos algunas de las pruebas realizadas durante el desarrollo del proyecto.

4.2.1. Pruebas del módulo de radiofrecuencia

Un problema que tuvimos mientras probamos el módulo de radiofrecuencia, era que el módulo receptor cuando no está recibiendo ninguna señal de radiofrecuencia va ajustando su sensibilidad para recibir señales más lejanas hasta que encuentra una. El problema aparecía cuando teníamos que enviar muchos 0 seguidos por el transceptor, ya que como hemos visto antes en la modulación OOK un cero significa anular la portadora, por lo que el receptor no recibe nada y sube su sensibilidad hasta que le interfiere ruido de otras señales de radiofrecuencia. Nosotros lo resolvimos añadiendo una resincronización cada 4 bits para evitar transmitir muchos ceros seguidos y forzar a enviar un uno. Sin embargo hay soluciones mucho más óptimas, como podemos ver en el apartado 5.2.2.

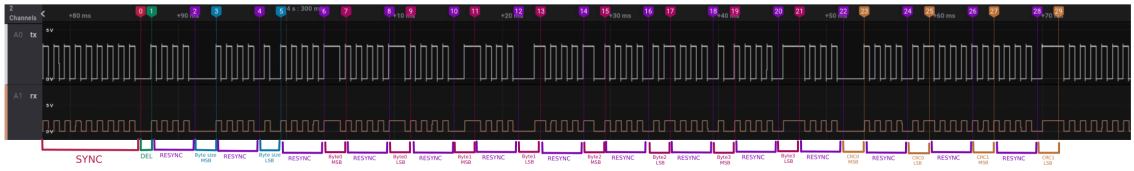

Figura 4.12: Trama de comunicación

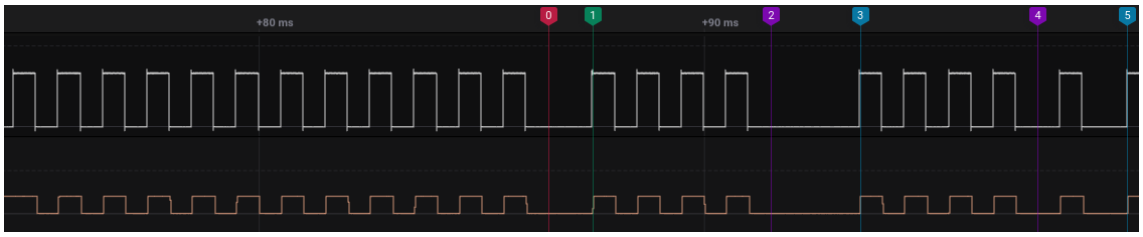

Figura 4.13: Sincronización, delimitador y tamaño

4.2.2. Pruebas con un sensor de presión durante 24 horas

En este apartado vemos las pruebas realizadas mediante el sensor de presión atmosférica colocado en un punto fijo durante un periodo aproximado de 24 horas. Como podemos ver en la Figura 4.16 la presión atmosférica va oscilando partiendo de 102000 pascales, con una altitud de 0 metros partiendo de esa referencia, la altitud llega a superar los 20 metros y los -30 metros. Estos cambios se deben a los fenómenos atmosféricos, que hacen oscilar la presión y es por ello que decidimos utilizar dos sensores para así evitar esos cambios de presión.

En el Cuadro 4.2 vemos como cambian los valores de presión y altitud respecto al tiempo. Vemos que la altitud varía desde los -28 metros hasta los 17 metros. Cabe destacar que en este experimento el sensor estaba en un punto fijo sin moverse.

Cabe destacar que hicimos pruebas cambiando los parámetros de configuración del sensor de presión, utilizando los parámetros recomendados para la aplicación de un ascensor, sin embargo estos no tuvieron efecto, ya que no estábamos obteniendo un error de precisión sino un error externo al sensor de presión.

Cuando las pruebas se realizaban en un intervalo corto de tiempo, por ejemplo en 10 minutos, obteníamos una precisión muy buena, por ejemplo subiendo el sensor de presión a +2 metros desde la referencia, obteníamos unos valores desde +1,8 a +2,2 metros.

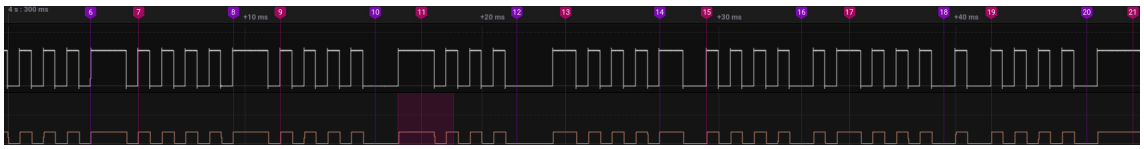

Figura 4.14: Datos del mensaje

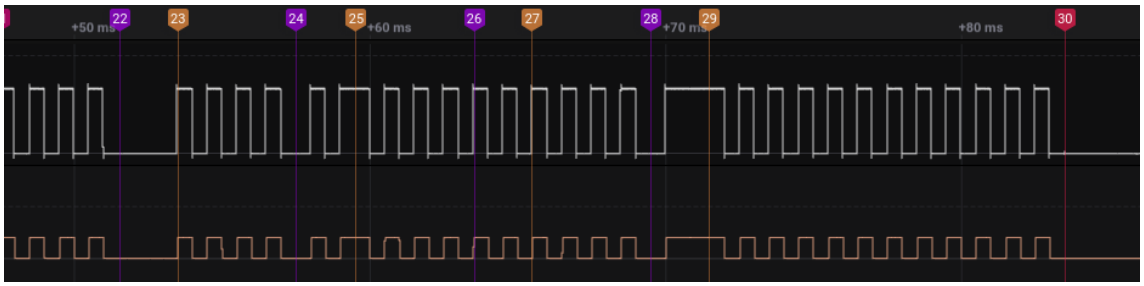

Figura 4.15: Bytes de crc y final

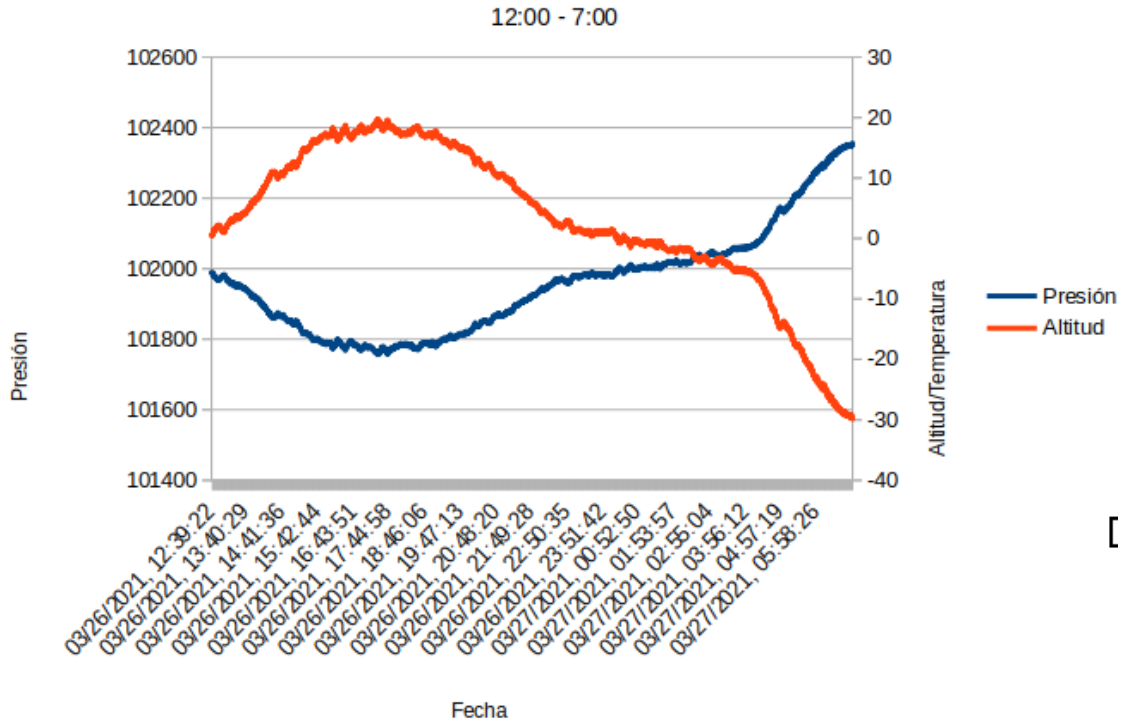

Figura 4.16: Muestreo de presión durante 24 horas

Muestra	Presión(pascales)	Altitud(m)	Temperatura(ºC)
03/26/2021, 12:37:11	101991,93	0,2	28,39
03/26/2021, 12:44:14	101973,53	1,73	28,16
03/26/2021, 13:13:27	101958,03	3,01	27,85
03/26/2021, 13:50:54	101921,66	6,02	27,85
03/26/2021, 19:00:12	101782,27	17,56	28,35
03/26/2021, 22:43:22	101965,86	2,36	28,35
03/27/2021, 06:39:45	102344,81	-28,94	28,4

Cuadro 4.2: Muestra de algunos datos leídos por el sensor

4.2.3. Pruebas con dos sensores de presión

Viendo el problema que apareció utilizando un solo sensor, decidimos utilizar dos sensores para eliminar el error del cambio de presión debido al clima. Como podemos ver en la Figura 4.17, la presión en el sensor 1 (Atom Lite) y la presión en el sensor 2 (Dev kit) van relacionadas en el tiempo y conseguimos de esta manera eliminar el error descrito. En la Figura 4.18 aparece también la relación en cuanto a la altitud de los dos sensores.

Como vemos, tanto en la Figura 4.17 como en la Figura 4.18, la presión y la altitud oscilan y no se mantienen constantes pese a que el sensor esté ubicado en un punto fijo. Haber utilizado los dos sensores nos permitió observar que el fenómeno atmosférico afecta por igual a los dos sensores, esto se ve reflejado en las figuras 4.17 y 4.18. Gracias a utilizar los dos sensores conseguimos estimar la altura de la cabina de manera precisa.

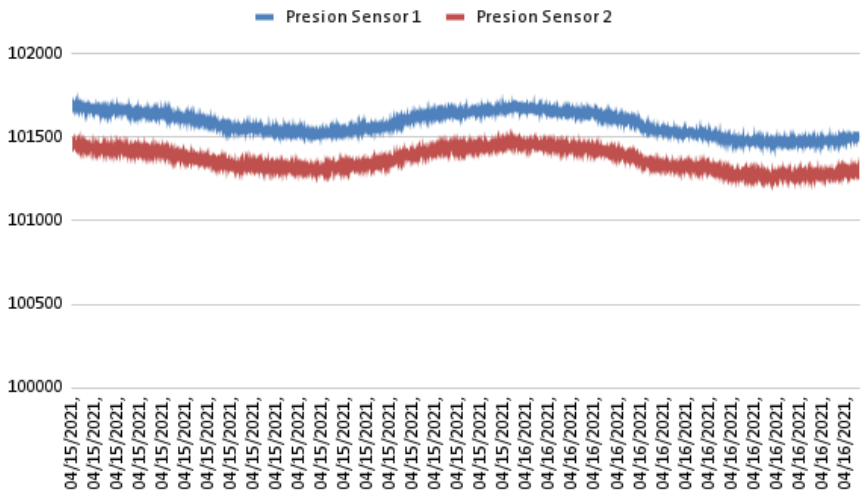

Figura 4.17: Muestreo de presión con 2 sensores

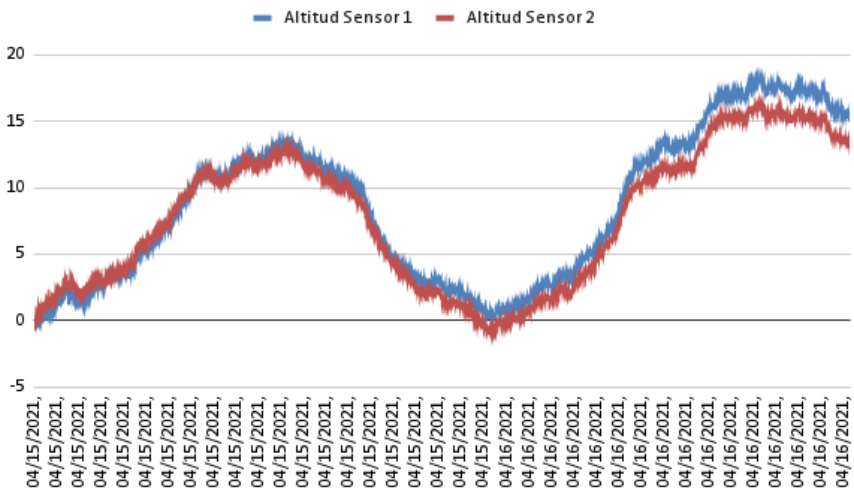

Figura 4.18: Muestreo de altitud con 2 sensores

Capítulo 5

Conclusiones y trabajo futuro

Índice

En este capítulo mostramos la conclusión del proyecto, desglosada en tres apartados; la conclusión en el ámbito formativo, la conclusión en el ámbito profesional y la conclusión en el ámbito personal. Además, incluimos una sección de trabajo futuro, en la que mostramos todo lo que querríamos haber hecho en este proyecto y no pudimos completar, y también mostramos otras mejoras que podríamos añadir al proyecto.

5.1. Conclusiones

5.1.1. Ámbito formativo

Creo que en el ámbito formativo es en el que más beneficios he obtenido. He conseguido aprender muchos conceptos tanto teóricos como prácticos. Creo que ha sido un poco difícil empezar y al principio sobre todo me costaba mucho coger el ritmo, ya que creo que en el proyecto he visto muchos conceptos y tecnologías que no había visto todavía. También he visto la importancia de realizar una buena planificación para poder cumplir los objetivos. En definitiva creo que aunque no haya cumplido alguno de los objetivos del proyecto, sí que he conseguido el objetivo formativo de la asignatura.

5.1.2. Ámbito profesional

En el ámbito profesional he conseguido adaptarme al modo de funcionamiento de la empresa, al igual que en el ámbito formativo creo que ha sido difícil adaptarme debido a la situación del COVID-19 [20] ya que las prácticas han sido totalmente mediante teletrabajo. El máximo esfuerzo que he realizado creo que ha sido en el de la investigación, ya que la mayor parte del tiempo la dedicaba a investigar los componentes y tecnologías del proyecto.

5.1.3. Ámbito personal

Finalmente en el ámbito personal creo que he aprendido mucho, y la estancia en prácticas me ha servido para enfocar un poco más mi especialización de la informática. He aprendido muchas cosas interesantes y creo que el teletrabajar me ha dado otra visión de trabajo que no conocía.

5.2. Trabajo futuro

En esta sección, abordamos las mejoras y trabajo futuro del proyecto, ya que como hemos visto en capítulos anteriores, no hemos podido cumplir la totalidad de los objetivos. Es por eso que en esta sección incluimos los pasos que se deberían seguir para continuar con el desarrollo del proyecto.

5.2.1. Acelerómetro

Creemos que si queremos determinar la altura de la cabina con cierta precisión, debemos utilizar un acelerómetro y cruzar esos datos con los obtenidos por el proyecto realizado. De esta manera obtenemos la altura con mayor precisión y de este modo cumplir con el objetivo principal del proyecto.

El acelerómetro se coloca en el sistema ubicado en la cabina, y de este modo detecta cuándo la cabina realmente se está moviendo. Las dos funcionalidades que aporta el acelerómetro son las siguientes:

Compaginar al sensor de presión

La primera funcionalidad consiste en compaginar el sensor de presión atmosférica usando el acelerómetro. Consiste en que el sensor de presión esté constantemente realizando mediciones de la presión atmosférica, si el acelerómetro no detecta movimiento y la presión va cambiando paulatinamente, sabemos que ese cambio de presión es debido al clima u otro fenómeno y no al cambio de altitud. Cuando el acelerómetro detecte una aceleración en el eje Y, entonces es

que el ascensor ha comenzado a moverse, por tanto sabemos que la presión atmosférica está cambiando debido al cambio de altitud.

A continuación mostramos un pseudocódigo que muestra el funcionamiento de esta funcionalidad.

```
class Acelerometro:
    acelX = 0
    acelY = 0
    acelZ = 0

    def __init__(self, params):
         pass

    def getAceleracion(self):
        pass
        return (self.acelX, self.acelY, self.acelZ)

class Presion:
    presionAtmosferica = 0
    presionCota = 0
    alturaCota = 0

    def __init__(self, params):
        self.presionInicialEnCota = self.getPresion()


    def getPresion(self):
        pass
        return self.presionAtmosferica

    def calculaAltura(self, presion):
        pass
        return 0

def main():

    objAcelerometro = Acelerometro(None)
    objPresion = Presion(None)

    presion, presionCota, alturaCota, movimiento = 0

    # Cota es un punto en el que conocemos la altura
    # y presion en el momento incial. Y se va calculando
    # la presion y la altura en todo momento.

    while(1):
        _, movimiento, _ = objAcelerometro.getAceleracion()
        presion = objPresion.getPresion()

        if(movimiento):
            # En el momento de empezar a moverse, los
            #   calculos de la altura se realizan
            #   mediante la ultima presion de cota
            #   cuando pare de moverse, calculamos
            #   la altura, y actualizamos tanto la
            #   presion en cota, como la altura de
            #   la cota.

        else:
            # Se va actualizando la presion en la cota
            #   todos los cambios de presion son
            #   debidos al clima, por lo que podemos
            #   ir actualizando la cota

```

Calcular altura con el acelerómetro

Utilizando un acelerómetro podemos calcular la aceleración en un momento determinado. Si integramos esa aceleración obtenemos la velocidad y si volvemos a integrar esa velocidad obtenemos la posición. Cabe destacar que al estar realizando una doble integral se añaden dos constantes, que serían la velocidad inicial y la posición inicial. En el caso de colocarlo en el ascensor, la velocidad inicial siempre sería 0, ya que siempre partimos de que el ascensor ha parado en una planta. Y la posición inicial la habríamos calculado previamente, realizando las dos estimaciones con el sensor de presión y el acelerómetro.

Sin embargo hay dos detalles que tenemos que tener en cuenta. El primero es que si queremos hacer estimaciones correctamente, deberíamos realizar una calibración nueva. Por ejemplo, si colocamos el sensor de cota en la planta 0, cada vez que la cabina llegue a ese punto, debería de calibrar su posición y la presión de referencia. El otro aspecto a tener en cuenta es que como queremos obtener una buena precisión, habría que realizar las lecturas de aceleración con una alta frecuencia para así reducir el error a lo mínimo posible.

5.2.2. Cambio de codificación

Como vimos en el capítulo de implementación, teníamos un problema con la codificación que estábamos usando para enviar los mensajes por radiofrecuencia, ya que el receptor ajustaba su sensibilidad si el transmisor enviaba muchos 0 seguidos, ya que un 0 como vimos en OOK significa anular la portadora.

La idea consiste en cambiar por ejemplo la codificación. Los unos lógicos los codificamos como un uno y un cero, y los ceros lógicos los codificamos como un uno y dos ceros. De esta manera forzamos al transmisor a que envíe mínimo un uno cada 3 bits. Y de esta manera tan solo tendríamos que realizar la etapa de sincronización una vez, y a partir de ahí transmitir la trama con la codificación nueva.

Otra codificación que podríamos aplicar es la codificación Manchester [19], que asegura el cambio de flanco y además incluye la sincronización en la misma codificación por lo que las ventajas son bastantes mayores que en la codificación explicada anteriormente.

En la Figura 5.1 podemos ver el funcionamiento la codificación Manchester con un ejemplo de transmisión.

En la parte superior de la Figura vemos la señal del reloj. En la parte central podemos ver la trama de datos que deseamos codificar. Finalmente, en la parte inferior vemos la trama codificada.

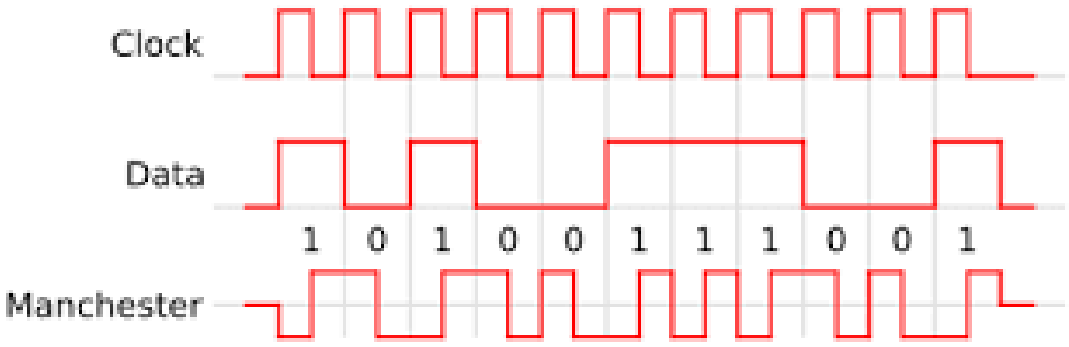

Figura 5.1: Ejemplo de codificación Manchester

Bibliografía

[1] Bmp280 datasheet. https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf. [Internet; descargado 18-mayo-2021].

[2] Espressif idf. https://docs.espressif.com/projects/esp-idf/en/latest/esp32. [Internet; descargado 20-mayo-2021].

[3] I2c - bus comunicación en serie. https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/. [Internet; descargado 19-mayo-2021].

[4] Microcontrolador esp32. https://www.espressif.com/en/products/socs/esp32. [Internet; descargado 17-mayo-2021].

[5] Nayar systems. https://www.nayarsystems.com/. [Internet; descargado 17-mayo-2021].

[6] Salarios ingeniería informática. https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico. [Internet; descargado 14-mayo-2021].

[7] Wireless fidelity. https://es.wikipedia.org/wiki/Wifi. [Internet; descargado 10-mayo-2021].

[8] Fs1000a transciver. https://www.luisllamas.es/comunicacion-inalambrica-en-arduino-con-modul 2021. [Internet; descargado 24-mayo-2021].

[9] Espressif. Rmt - remote control. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/rmt.html, 2020. [Internet; descargado 30-mayo-2021].

[10] FreeRTOS. Sistema operativo en tiempo real freertos. https://www.freertos.org/, 2021. [Internet; descargado 24-mayo-2021].

[11] Github. Control de versiones en línea. https://github.com/, 2021. [Internet; descargado 24-mayo-2021].

[12] Microsoft. Visual studio code. https://code.visualstudio.com/, 2021. [Internet; descargado 24-mayo-2021].

[13] Saleae. Analizador lógico. https://www.saleae.com/es/downloads/. [Internet; descargado 20-mayo-2021].

[14] Wikipedia. C (lenguaje de programación) — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=C_(lenguaje_de_programaci%C3%B3n)&oldid=135092282. [Internet; descargado 21-mayo-2021].

[15] Wikipedia. Desarrollo en cascada — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Desarrollo_en_cascada&oldid=134267691. [Internet; descargado 19-mayo-2021].

[16] Wikipedia. Magicdraw uml — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=MagicDraw_UML&oldid=118940107. [Internet; descargado 18-mayo-2021].

[17] Wikipedia. Modulación digital de amplitud — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Modulaci%C3%B3n_Digital_de_Amplitud&oldid=123456429, 2020. [Internet; descargado 29-mayo-2021].

[18] Wikipedia. Modulación por desplazamiento de amplitud — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Modulaci%C3%B3n_por_desplazamiento_de_amplitud&oldid=122921174, 2020. [Internet; descargado 29-mayo-2021].

[19] Wikipedia. Codificación manchester — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Codificaci%C3%B3n_Manchester&oldid=135050543, 2021. [Internet; descargado 21-mayo-2021].

[20] Wikipedia. Covid-19 — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=COVID-19&oldid=135852017, 2021. [Internet; descargado 27-mayo-2021].

Anexo A

Imágenes del montaje

En este Anexo mostramos las figuras del montaje final realizado en una protoboard.

En las Figuras A.1, A.2 y A.3 podemos ver los dos sistemas montados en la misma protoboard donde los dos cables que se ven en vertical son las antenas del módulo de radiofrecuencia.

En las Figuras A.4, A.5, A.6 y A.7 podemos ver en detalle las conexiones del montaje,en concreto, vemos el módulo de radiofrecuencia y el sensor de presión BMP280.

Finalmente en la Figura A.8 vemos el montaje en el que hemos utilizado el microcontrolador Atom Lite.

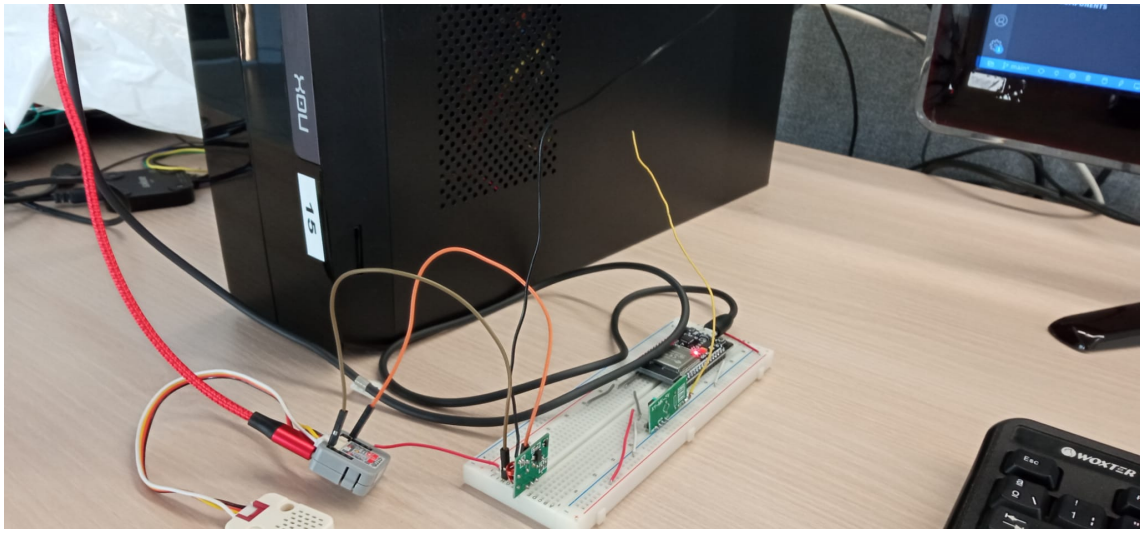

Figura A.1: Montaje del sistema de radiofrecuencia

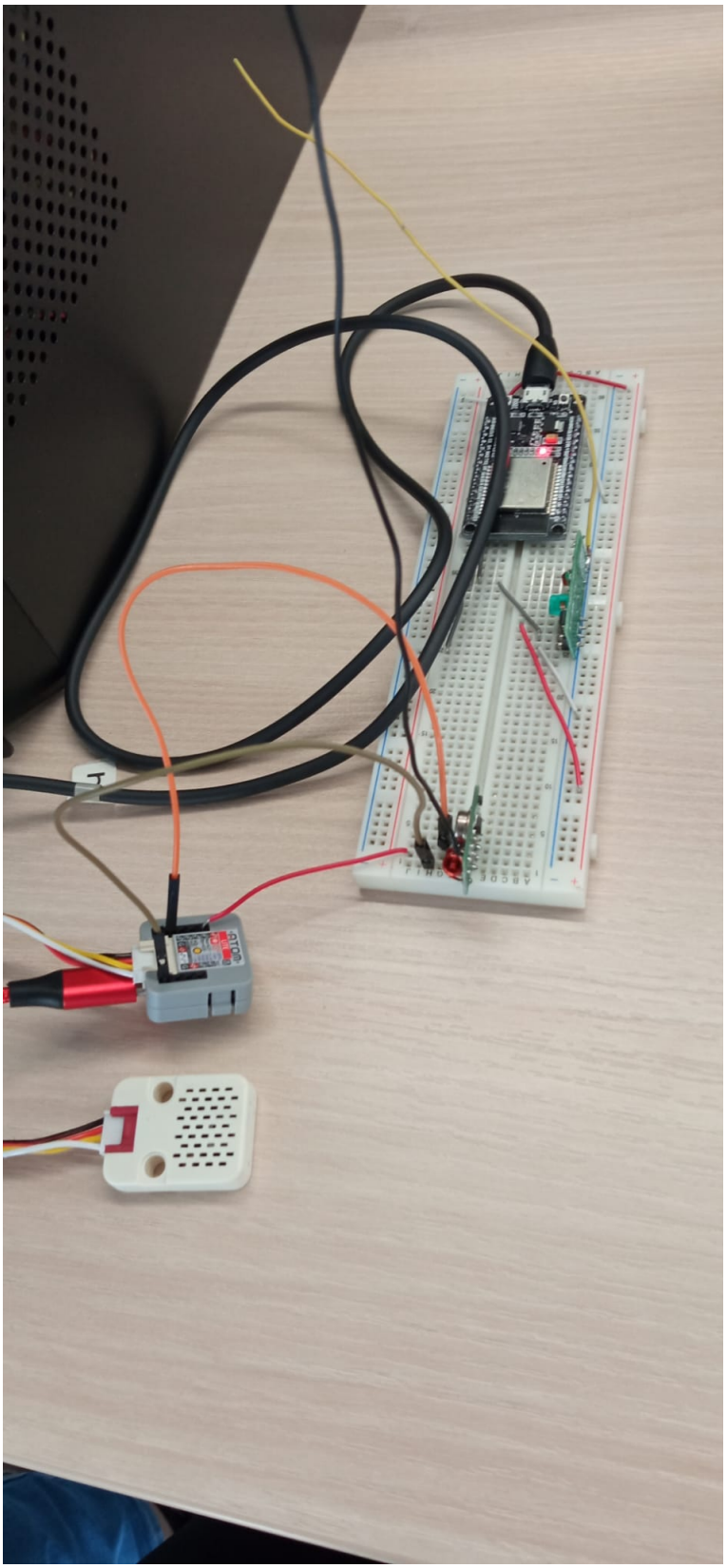

Figura A.2: Montaje del sistema de radiofrecuencia

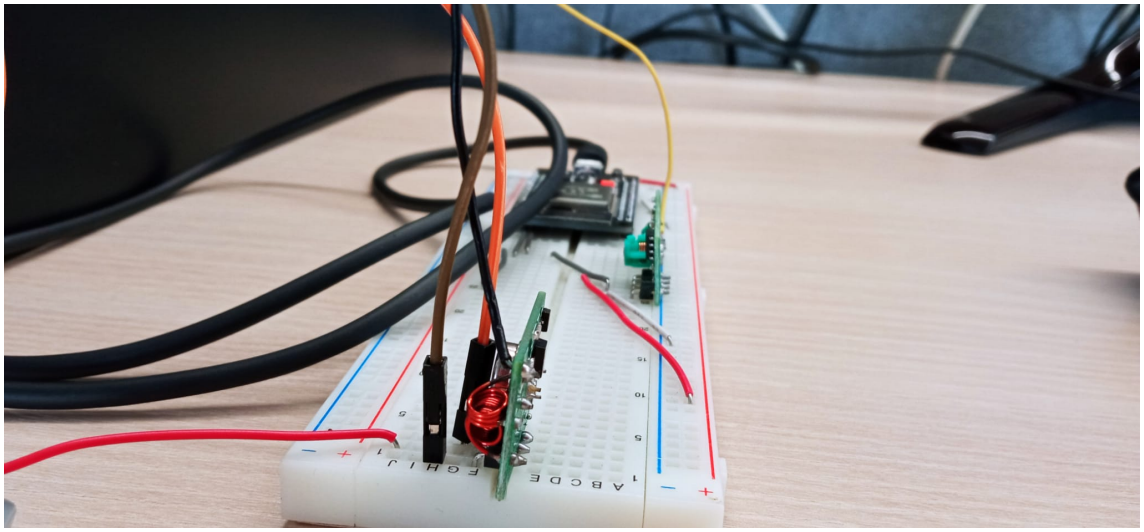

Figura A.3: Montaje del sistema de radiofrecuencia

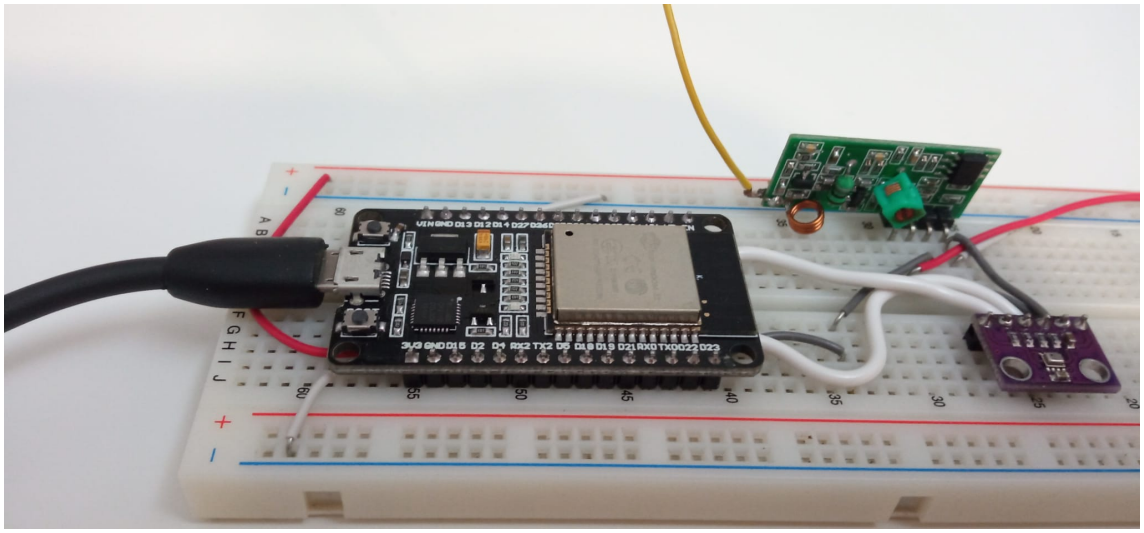

Figura A.4: Montaje del sistema de radiofrecuencia y el sensor de presión

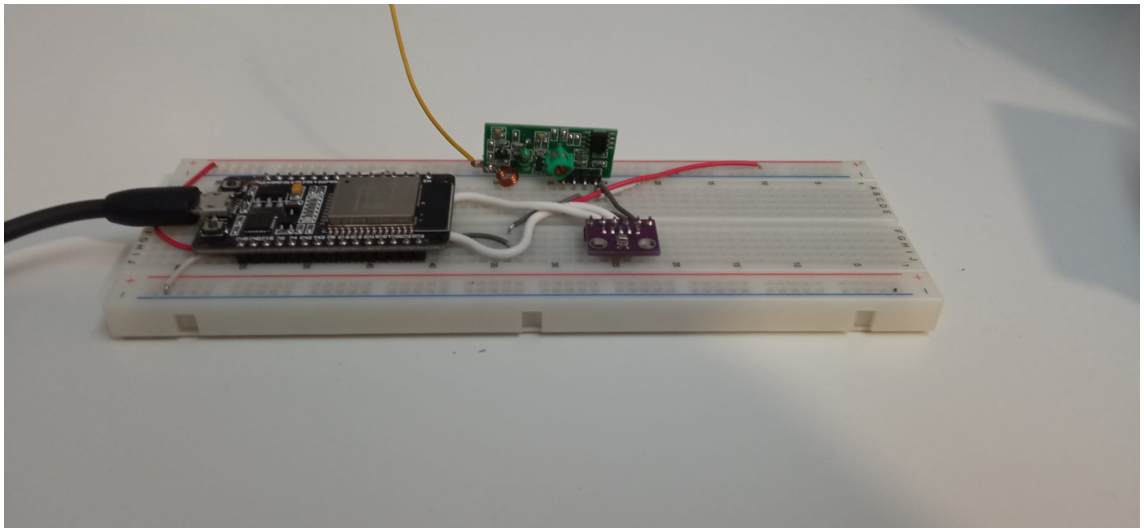

Figura A.5: Montaje del sistema de radiofrecuencia y el sensor de presión

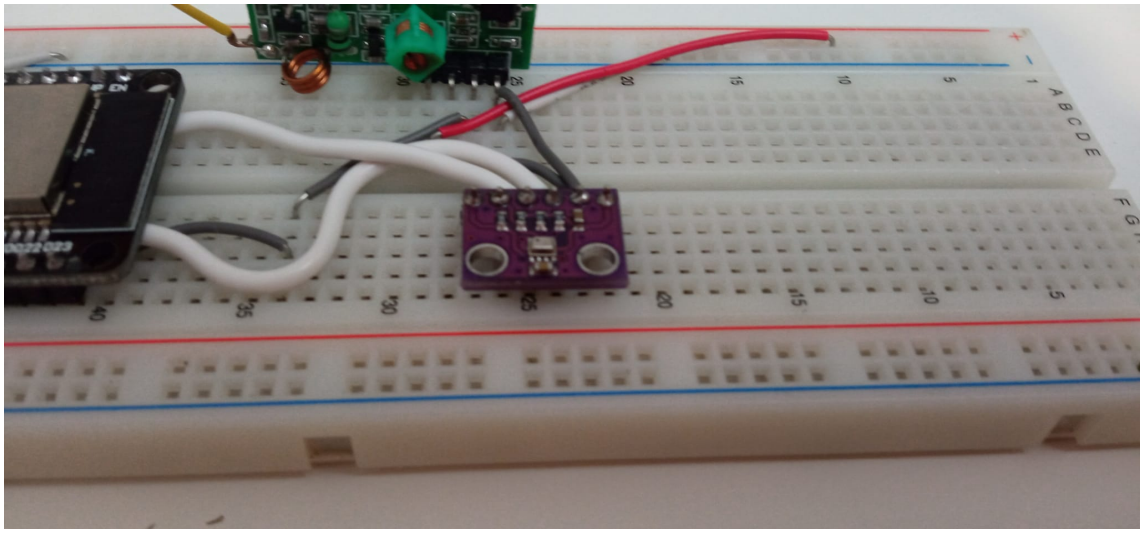

Figura A.6: Montaje del sistema de radiofrecuencia y el sensor de presión

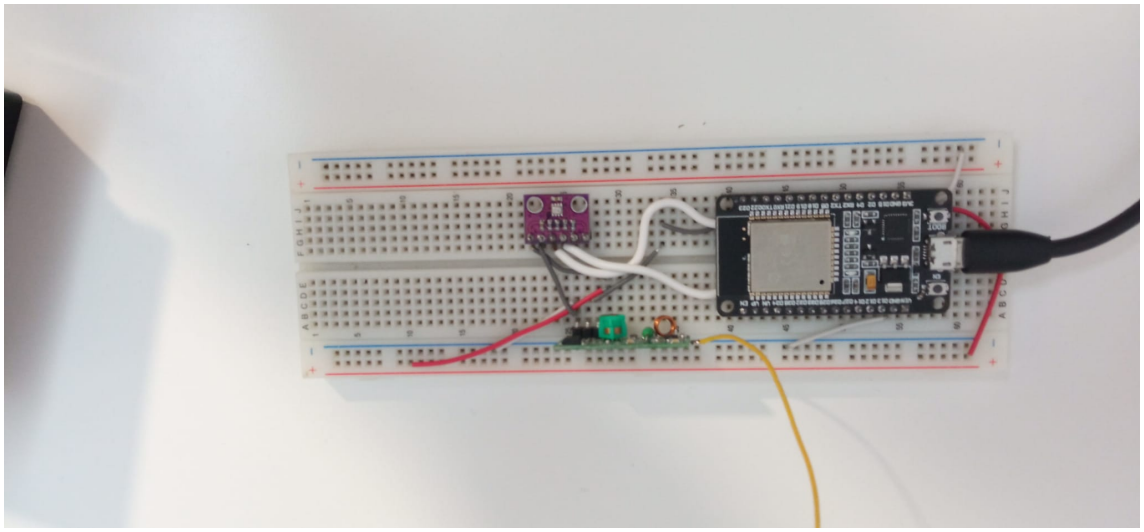

Figura A.7: Montaje del sistema de radiofrecuencia y el sensor de presión

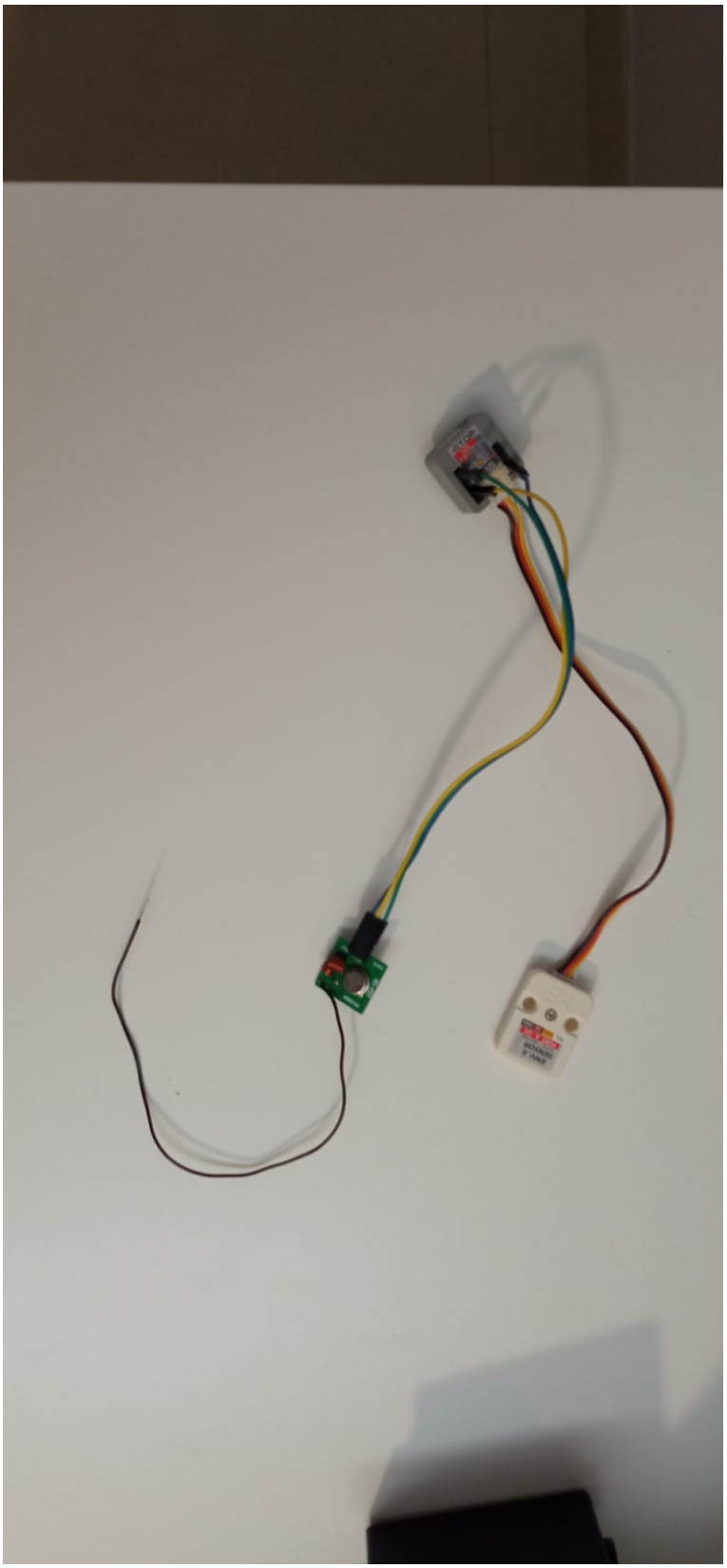

Figura A.8: Montaje del sistema donde se emplea el microcontrolador Atom Lite