Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Propuesta de Actualización del Diseño Software y Hardware de Dispositivos SmartCube

Autor: Álvaro Matellanes Fernández

Tutor/es: Bernardo Alarcos Alcázar

2021

Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo de Fin de Grado

Propuesta de Actualización Software y Hardware de Dispositivos SmartCube

Autor: Álvaro Matellanes Fernández

Tutor: Bernardo Alarcos Alcázar

TRIBUNAL:

Presidente: Antonio García Herráiz

Vocal 1º: Juan Ramón Velasco Pérez

Vocal 2º: Bernardo Alarcos Alcázar

FECHA: 21/09/2021

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Anexos

Resumen

Actualmente la electromedicina es una iniciativa que se encuentra en auge. Concretamente hay que destacar la aplicación en los más pequeños, donde la detección precoz de anomalías es de vital importancia. La pronosticación y su posterior tratamiento durante las etapas de mayor desarrollo del niño, pueden favorecer la adaptabilidad a futuras situaciones donde pueda mostrar mayores dificultades.

En este trabajo se versiona un juguete inteligente que, mediante el uso de diferentes sensores recopilará información acerca del comportamiento del niño durante su periodo de juego, tratando de buscar patrones que faciliten la detección precoz de anomalías en el desarrollo infantil.

Palabras Clave: SmartCube, ESP32, Sensores, BLE, PCB

Abstract

Electromedicine is currently a booming initiative. In particular, it is important to highlight its application in children, where the early detection of anomalies is vital. Prognosis and subsequent treatment during the stages of the child's greatest development can favour adaptability to future situations in which he or she may show greater difficulties.

In this work, a version of a smart toy is developed that, through the use of different sensors, will collect information about the child's behaviour during the play period, trying to look for patterns that facilitate the early detection of anomalies in the child's development.

Key Words: SmartCube, ESP32, Sensors, BLE, PCB

Capítulo 1:
Introducción y Objetivos

Capítulo 1: Introducción y Objetivos

Los SmartCubes son unos de los juguetes inteligentes dentro de una gama de juguetes cuyo objeto de creación, es la medición de variables físicas proporcionadas por los niños durante su periodo de juego.

En este proyecto se versionará por cuarta vez este juguete, el cual consiste en un conjunto de cubos apilables con los que el niño podrá jugar mientras se recopilan datos con ayuda de diferentes sensores.

La idea de una nueva versión de este juguete surge de la necesidad de corregir, mejorar y sustituir las funciones existentes y la implementación de nuevas. Estos requerimientos se convierten en objetivos a cumplir en esta nueva versión.

Objetivos del Proyecto

El objetivo principal del proyecto es el de la creación de un juguete inteligente, capaz de recopilar información durante los periodos de entretenimiento del niño por medio de diferentes tipos de sensores. Este objetivo a su vez forma parte de uno aún mayor, que es el de la diagnosticarían precoz de posibles irregularidades durante el desarrollo de los niños. Para ello, la información obtenida por los SmartCubes, es analizada a través de técnicas de *Machine Learning*, con el objetivo de encontrar ciertos patrones de comportamiento que permitan relacionarlo con la posibilidad de padecer este tipo de desfases.

A continuación, se exponen los objetivos parciales que deben abordarse a lo largo del proyecto para poder alcanzar el objetivo final.

- Diseño *Hardware*. Elaboración de un diseño hardware capaz de recolectar datos de los sensores, procesar la información y transmitirla a un dispositivo receptor. Las principales diferencias con versiones anteriores y por tanto objetivos a cumplir, son los siguientes:
    - Sustitución de los conectores de carga de la batería.
    - Implementación de un sistema de medición de batería.
    - Sustitución de los sensores de luminosidad (fototransistores) por sensores de fuerza (galgas extensiométricas).

  - Transmisión de datos empleando tecnología *Bluetooth Low Energy* (BLE) en lugar de por radio frecuencia como se empleaba hasta ahora.
  - Implementación de un sistema de actualización de firmware inalámbrico.
  - Diseño de las nuevas placas de circuito impreso (PCB).

  Al satisfacer estos requerimientos de *hardware* surgen otras modificaciones necesarias en el hardware como se explicará a lo largo de la memoria, entre las que destaca el cambio del microcontrolador empleado.

- Diseño *Software*. Como se ha adelantado en el punto anterior, al cambiar de microcontrolador será necesario la creación de un nuevo *software*, aunque como se verá más adelante, se mantiene el mismo entorno de programación. Los objetivos a nivel de *software* son los siguientes:
  - Creación de un código capaz de tomar correctamente información de los sensores, procesarla y transmitirla a través del BLE.
  - Implementación de modos de juego interactivos aprovechando el nuevo sistema de sensores de fuerza y los diodos Led.
  - Sistema de advertencia del agotamiento de la batería.
- Diseño 3D. Se realizarán los nuevos diseños de las carcasas de los SmartCubes y de la plataforma de carga, ya que sobre todo esta última, difiere bastante de la versión utilizada en los anteriores cubos, debido a las incorporaciones en el *hardware* de esta nueva versión, como se verá más adelante.

A lo largo de la memoria se desarrollarán los siguientes apartados: un planteamiento inicial del proyecto, la metodología seguida tanto en la memoria como en el proyecto, una propuesta de nuevo microcontrolador donde se verá el contexto histórico de los SmartCubes y los motivos por los que se realiza esta propuesta, la selección del *hardware* que se empleará en esta nueva versión, una serie de pruebas de los principales sistemas por separado, una prueba conjunta con todos los sistemas parciales, diseño de las nuevas PCB de los SmartCubes, diseño 3D de las carcasas y la plataforma de carga, el montaje de final de los SmartCube y se desarrollará un Software final de prueba, por último, se expondrán las conclusiones y trabajo a futuro del proyecto.

Capítulo 2:
Planteamiento

Capítulo 2: Planteamiento

Vistos la introducción del proyecto y sus principales objetivos a cumplir, se plantea la realización de una nueva versión de los SmartCubes. Esta versión del juguete inteligente mantendrá algunos de los elementos de la anterior versión, como son el acelerómetro, los diodos Leds, la batería y el zumbador. Entre las nuevas incorporaciones o modificaciones se encuentran, las galgas extensiométricas, sustituyendo a los fototransistores de versiones anteriores; el sistema de transmisión de datos, el método de actualización de código, el sistema de carga de batería y el propio microcontrolador. Puesto que el principal objetivo del dispositivo es la adquisición de datos, se debe plantear el tipo de datos y la forma de adquisición de estos, para ello se plantean las siguientes preguntas.

¿Qué mediciones se realizarán?

Los elementos de los que se puede adquirir información de forma directa son las galgas extensiométricas, el acelerómetro y la batería, y adicionalmente se podrá obtener información de los Leds y el zumbador empleados conjuntamente con el reloj interno del microcontrolador y las galgas extensiométricas.

Galgas Extensiométricas

Al aplicar una fuerza sobre la superficie a la que esta adherida la galga, la resistencia de la galga varía, generando a su vez una variación de tensión, que se puede asociar con un nivel concreto de fuerza.

Acelerómetro

Con los datos proporcionados por el acelerómetro, se podrán obtener medidas de la aceleración, velocidad media y agitaciones.

Leds y Zumbador

Al emitir una señal luminosa o sonora junto con el reloj interno y las galgas extensiométricas, podrá medirse el tiempo de reacción del niño ante este tipo de estímulos.

Batería

Con motivo de controlar la descarga de la batería, se tomarán medidas periódicas del estado de ésta.

Reloj Interno

Se tomarán medidas del tiempo de cada suceso relevante durante la ejecución de las pruebas.

¿Cómo se realizarán las medidas?

Las pruebas se realizarán durante el periodo de juego del niño, mientras se encuentra relajado y enfocado en jugar con el juguete inteligente. En estas pruebas el juguete puede tomar medidas de diferentes formas, según el modo en que se encuentre.

Modo Normal. Durante este modo de funcionamiento el juguete se encontrará en un estado pasivo, donde únicamente tomará medidas del acelerómetro y las galgas. También tomará la medida de tiempo de cada muestra.

Modo Interactivo. En este modo de funcionamiento el cubo emitirá señales luminosas y acústicas, e interactuará con el niño a través de las galgas extensiométricas. De esta forma, además de tomar las medidas del acelerómetro y las galgas, también se tomarán medidas del tiempo de reacción del niño ante los estímulos.

De esta forma, se cuentan con dos fuentes de obtención de datos con los que sacar posibles patrones.

Capítulo 3: Metodología

Capítulo 3: Metodología

Se muestra a continuación, la secuencia cronológica en la que se han realizado las diferentes etapas de este proyecto que, a su vez, comparte estructura con la forma en que se desarrolla esta memoria.

1) Estudio de los sistemas que conforman los SmartCubes y búsqueda de información para cumplir los objetivos propuestos.
2) Selección de *Hardware*.
3) Sistemas Parciales Principales.
    A) Estudio de funcionamiento.
    B) Esquemático.
    C) Prueba de funcionamiento.
4) Prueba de Funcionamiento del Sistema Completo.
5) Diseño SmartCubes
    A) Diseño Esquemático
    B) Diseño PCB
6) Diseños 3D
    A) Carcasa de los SmartCubes
    B) Plataforma de Carga
7) Montaje SmartCubes y Software Final

Se ha desarrollado un diagrama de Gantt donde pueden verse las diferentes etapas del proyecto junto a su duración, además se muestran periodos de inactividad debidos a diferentes factores que se explican en detalle. Dicho diagrama puede encontrase en el Anexo XIII.

Así mismo, en la sección de Anexos pueden encontrase más documentos explicativos acerca del proyecto.

Capítulo 4:
Propuesta de Cambio de Microcontrolador

4.01 Anterior Controlador, Motivos del Cambio y Búsqueda de Soluciones

4.02 Módulo ESP32

4.03 Optimización del ESP32

Capítulo 4: Propuesta de Cambio de Microcontrolador

4.01 Anterior Controlador, Motivos del Cambio y Búsqueda de Soluciones

Anterior Controlador (ATMEGA328P-AU)

En versiones anteriores, el chip encargado de realizar el control de tareas y el procesado de información ha sido el ATMEGA328P-AU, un microcontrolador simple, pero con el rendimiento necesario para esta aplicación y que, además, permite trabajar en el entorno de desarrollo IDE de Arduino. Esta herramienta de *software* facilita una programación sencilla de los microcontroladores Arduino (desarrollar, compilar y grabar código de forma simple), asimismo, Arduino cuenta con otras ventajas por las que finalmente se escogió este microcontrolador frente a otros del mercado [1] [2]:

1. Bajo Costo. Este microcontrolador, el ATMEGA328P-AU, presenta un precio bastante accesible frente a otros microcontroladores, podemos encontrarlo a precios entorno a los 2€ en grandes plataformas de venta como Aliexpress.
2. Multiplataforma. Arduino puede desarrollarse en diferentes sistemas operativos, siendo los más comunes Windows, Macintosh y Linux. Por el contrario, muchos microcontroladores únicamente están enfocados a Windows.
3. *Software* y *Hardware* de código abierto. Arduino ofrece su software con licencia libre, lo que permite ampliar o modificar el propio software. En cuanto al hardware, ofrece sus planos a cualquier persona sin necesidad de pedir permisos, lo que permite hacer las modificaciones deseadas para adaptarlo a un proyecto.
4. Facilidad. Arduino ofrece una programación sencilla y con una gran disponibilidad de librerías que simplifican el trabajo en numerosas situaciones. Esta gran variedad de librerías está directamente relacionada con el punto anterior, ya que el *software* libre habilita la programación de librerías en C++ por cualquier usuario.

Motivos del Cambio

Como en prácticamente cualquier situación, el cambio surge de la necesidad de adaptación y en este caso, se debe adaptar el dispositivo para asimilar nuevas funciones y corregir las ya existentes. A continuación, se verán los principales motivos por los que se toma la decisión de cambiar el microcontrolador:

1. Mejora en la transferencia de datos. El flujo de datos hasta ahora se ha realizado por medio de un módulo de radiofrecuencia (RF), tecnología la cuál a pesar de ser capaz de transmitir grandes cantidades de datos y presentar un coste relativamente bajo, está sujeta al acoplo de números tipos de interferencias. Por ello, se plantea la utilización de tecnología *Bluetooth Low Energy* (BLE), mucho más estable que la RF y con unos consumos muy reducidos como su propio nombre indica.
2. Carga de software inalámbrico. En las anteriores versiones de los cubos, la carga del software se realiza por medio de comunicación SPI, a través de la conexión de sus correspondientes pines. Lo que conlleva que cada vez que se requiera actualizar el software, haya que desmontar el cubo y establecer la conexión por medio de unos pines que no siempre garantizan un contacto estable. Por ello, se plantea la idea de realizar la carga del programa con tecnología *Over The Air* (OTA), en un principio empleando la tecnología BLE que se pretendía utilizar también para la transferencia de datos, pero finalmente nos decantamos por la tecnología Wifi como se explicará más adelante.
3. Mayor número de E/S y memoria Flash. Con el *software* de las anteriores versiones, la memoria Flash quedaba prácticamente llena al cargar el programa en el microcontrolador, por lo que, con la implementación de las nuevas y posibles futuras funciones del juguete, surge la necesidad de aumentar la capacidad de almacenaje de *software*. Por otro lado, con la sustitución de los fototransistores por las galgas extensiométricas, es interesante recuperar el accionamiento individual de los diodos Led ante la posibilidad de implementar juegos interactivos en un futuro, ya que en las últimas versiones la activación/desactivación del conjunto de diodos se llevaba a cabo por medio de un transistor configurado como interruptor. Por ello y ante la posibilidad de incorporar nuevos sensores y actuadores en el proyecto, surge la necesidad de aumentar el número de terminales de este tipo.

Búsqueda de Soluciones

Teniendo presente lo relatado en el anterior apartado se comenzó a investigar posibles soluciones, llegando a una primera propuesta que no requería cambio de microcontrolador, punto bastante relevante ya que, si finalmente se cambiara de controlador y este no pudiera se programado en el IDE de Arduino, habría que adaptar el código actual al lenguaje ensamblador correspondiente.

Para la transmisión de datos se utilizaría un HM-11 o cualquier otra versión más reciente de este tipo de módulos *Bluetooth LE* en formato SMD, en concreto el HM-11 utiliza Bluetooth V4.0. La transmisión de datos resultó exitosa salvo algún problema inicial en la configuración del módulo, donde el *Baud Rate* predeterminado impedía configurar correctamente el módulo.

Para solucionar el problema de los puertos de E/S se planteó la utilización de un Decodificador/Demultiplexor (SN74HCS238PWR), controlado los 6 diodos Led y el *buzzer* por medio de 4 entradas.

En cuanto al problema de la memoria Flash, se plantea la optimización del software de forma que se reduzca todo lo posible el tamaño del programa.

En cuanto a la carga inalámbrica del software, se propone la utilización de la tecnología BLE, partiendo del hecho de que existe la posibilidad de cargar un boceto en Arduino empleando un módulo bluetooth HC-05, un condensador y un par de resistencias. La razón por la que puede utilizarse el bluetooth para esta labor, es que tanto al utilizar un cable USB como la comunicación bluetooth, ambos se comportan como un puerto serie (COM en Windows) y la única diferencia entre ambos, es que el cable USB activa una señal de control DTR que carga un condensador que, a su vez permite activar la patilla *Reset*. Por lo tanto, si se consigue crear una señal DTR, con la ayuda de un condensador conectado en serie a la señal DTR y *Reset*, se podría cargar un boceto de forma inalámbrica [3]. Para realizar la programación con el módulo HC-05, primero se configura el módulo por medio de comandos AT, donde se establecerá un rol de esclavo, una velocidad de *Baud Rate* igual a la velocidad de *Bootloader* del microcontrolador que se desee programar, y se establecerá la polaridad del pin 32 o *State* para que pase de nivel alto a bajo cuando se conecte un dispositivo. Al configurar de esta forma el pin 32, lo que

se está haciendo es generar una señal DTR que, junto con el condensador ya harían posible programar el microcontrolador de forma inalámbrica, las resistencias anteriormente mencionadas se emplearían para reducir la tensión los pines TX y RX en caso de que el nivel lógico del microcontrolador sea 5V, ya que el nivel de operación del módulo es de 3.3V. Una vez todo configurado y montado, conectaremos el bluetooth del ordenador al módulo bluetooth y a continuación, en el IDE de Arduino se habrá creado un nuevo puerto COM, se seleccionará ese puerto y se cargará el programa [4].

Al tratar de trasladar este proceso a la tecnología BLE, se plantea la utilización del módulo smd HM-11, tras investigar en búsqueda de un pin capaz de generar la señal DTR, se plantea la utilización del PIO1 (pin del sistema Led), el cual a través de comandos AT es configurable para que pase de nivel bajo a alto al conectarse un dispositivo [5]. Este comportamiento es justo el contrario al de la señal DTR, por lo que al invertir la señal obtendríamos el resultado deseado (con un transistor y unas resistencias). Tras configurar el módulo, solo queda conectarnos con el ordenador para crear el puerto COM y cargar el programa, pero es aquí donde surge el problema con estos módulos. Debido a los protocolos que utiliza la tecnología BLE, no es posible conectarse de la misma forma que con el bluetooth clásico ya que éste trabaja bajo servicios personalizados, por lo que una persona con un amplio conocimiento en la materia podría crear una aplicación Cliente/Servidor capaz de establecer este tipo de comunicación [6].

Existen módulos BLE específicamente diseñados para poder realizar la actualización OTA, como es el caso del Adafruit Bluefruit LE UART Friend, donde por medio de una aplicación específica puede cargarse el nuevo boceto. No obstante, las elevadas dimensiones y precio que presentan estos módulos, los descarta como opción rentable en este proyecto [7].

Con el BLE descartado para la actualización OTA y el Bluetooth clásico, aunque sí la permite, no es acto como transmisor de información debido a sus ineficientes consumos, surge un nuevo pretendiente, el HM-13. Esta nueva alternativa es un módulo dual que permite tanto la conexión por BLE como por bluetooth clásico, de esta forma se podría realizar la transmisión de datos con la tecnología BLE y activar el bluetooth clásico para realizar las actualizaciones inalámbricas. Puesto que el HM-13 mantiene la misma estructura que el HM-11, se realiza el mismo montaje y se pasa a configurar el módulo con los comandos AT. Con el bluetooth clásico habilitado, se procede a conectar el

ordenador con el módulo y se carga el programa a través del puerto COM que se ha creado, durante dicha carga surge un fallo que impide que el programa se cargue correctamente. Se desconoce la razón por la que aparece este fallo durante la carga, aunque se mantiene la idea de que es posible realizar la carga del programa de forma inalámbrica empleando el HM-13 u otro módulo dual similar [8].

En la búsqueda de métodos alternativos al OTA con BLE, se encuentra la posibilidad de llevarlo a cabo empleando tecnología Wifi, más concretamente utilizando módulos como el ESP8266 o el ESP32. Es en este último en el que se encuentran más puntos a favor para convertirse en el nuevo controlador, ya que aparte de la tecnología Wifi, también posee BLE incorporado con el que poder realizar el intercambio de información y en general, un hardware con mayores prestaciones, el cual se tratara más en profundidad en el próximo apartado.

4.02 Módulo ESP32

El ESP32 es un módulo Wifi/Bluetooth que se aleja de los tradicionales módulos que solo proporcionan comunicación inalámbrica, incorporando un procesador de 2 núcleos cada uno de ellos pudiendo trabajar a frecuencias entre 80 y 240 MHz. El procesador permite la posibilidad de conectar a los periféricos dispositivos con las siguientes interfaces [9].

- Transmisor Receptor Asíncrono Universal (UART)
- Interfaz Periférica Serial (SPI)
- I2C
- I2S
- Ethernet
- Tarjetas SD
- Tecnología táctil y capacitiva
- Bus can

Los módulos de desarrollo ESP32 pueden integrar diferentes modelos de microcontrolador, en este proyecto se trabajará con el ESP-WROOM-32 o el ESP-WROOM-32. La razón por la que resulta indiferente un módulo de otro, es que estos dos módulos son prácticamente idénticos, la única diferencia es que el primero está basado en el chip ESP32-D0WD y el segundo en el ESP32-D0WDQ6.

Figura 1. ESP-WROOM-32

A continuación, se muestra una tabla donde se pueden apreciar las especificaciones técnicas más destacables del microcontrolador ESP-WROOM-32 junto a una comparativa con el ATMEGA328P-AU.

Microcontrolador	ESP-WROOM-32D	ATMEGA328P-AU
Rango de Tensiones de Alimentación	2.3 - 3.6 V	1.8 - 5.5 V
Tensión de Alimentación Recomendada	3.3 V	5 V
Tensión de Nivel Lógico	3.3 V	5 V
Pines de E/S Digitales	19(16 son PWM) de 40 GPIOs	14 (6 son PWM)
Pines de Entrada Analógicos	18 de 40 GPIOs	8
Corriente DC en los pines de E/S	40 mA	40 mA
Wifi	HT40	X
Bluetooth	Bluetooth 4.2 y BLE	X
Frecuencia CPU	80 - 240 MHz	0 - 20MHz
SRAM	448 KB	2 KB
Flash	520 KB	32 KB
Dimensiones	18x25.5x3.1 mm	7x7x1.05 mm

Tabla 1. Especificaciones Técnicas ESP-WROOM-32 vs ATMEGA328P-AU

El microcontrolador ESP-WROOM-32 resulta bastante superior en prácticamente todos los aspectos técnicos, el único punto negativo que tiene son sus dimensiones, las cuales le hacen un microcontrolador mucho más difícil de implementar que el ATMEGA328P-AU. No obstante, este aspecto desfavorable es totalmente asumible ya que implementado el ESP-WROOM-32 conseguimos ahorrar espacio en otras caras del cubo, véase en el caso de la transmisión de datos, donde ya no requerimos de un módulo bluetooth externo.

- Alimentación

Para alimentar el ESP32 existen dos opciones, una primera que consistiría en alimentar el módulo empleando un puerto micro-USB o bien por medio de una fuente externa conectada al terminal de 3.3V, la recomendación del fabricante en caso de alimentar por el pin de 3.3V es que sea con una fuente de 5V/1A.

- Programación

El ESP32 puede ser programado en varios lenguajes de programación, utilizando diferentes entornos de desarrollo como MicroPython, RTOS, Espruino, Mongoose OS y el más relevante para este proyecto, el IDE de Arduino, pudiendo con este último trabajar directamente sobre el código de las versiones anteriores. En ESP32 cuenta con un

bootloader cargado de serie que le permite ser programado en el IDE de Arduino sin necesidad de emplear un programador externo.

El módulo puede ser programado por medio de conector Serie-USB o en caso de poseer únicamente el microcontrolador ESP-WROOM-32, podemos emplear un convertidor Serie-USB externo como se explicará más adelante.

- Periféricos

El ESP-WROOM-32 cuenta con 48 terminales de los cuales 40 son GPIOs, cada uno de ellos con múltiples funciones y no siempre utilizables en proyectos. A continuación, se clasificarán los GPIOs según su registro de funciones [10].

1. **Pines Solo de Entrada (GPIs):** 34, 35, 36 y 39.
   Estos pines no poseen ningún tipo de resistencia interna de pull-up ni pull-down, por lo que solo pueden desempeñar la función de entradas.
2. **Pines SPI Flash (NO USAR):** 6, 7, 8, 9, 10 y 11.
   Estos pines están conectados internamente al SPI Flash que hay integrado en el ESP-WROOM-32 por lo que no es recomendable utilizarlos en el proyecto.
3. **Pines con Convertidor Analógico Digital (ADC):** 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36, 37, 38 y 39.
   Cada uno de estos 18 pines posee un ADC de 12 bits, pudiéndose configurar todos los parámetros del ADC desde el IDE de Arduino.
   Inicialmente se pretendía utilizar todos los pines de solo entrada y de esta forma optimizar al máximo la disponibilidad de los pines. Pero concretamente los GPIs 36 y 39, aunque teóricamente es posible utilizarlos como entradas analógicas y se ha comprobado que son capaces de hacerlo, existen varias plataformas de internet donde se describen fallos de funcionamiento cuando se utilizan interrupciones, debidas a errores en el hardware del módulo [11], [12]. Por este motivo, se decido un emplear estos dos pines en el diseño del proyecto.
4. **Pines con Convertidor Digital Analógico (DAC):** 25 y 26.
   Estos dos periféricos cuentan con un DAC de 8 bits.
5. **Pines GPIO RTC:** 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36 y 39.
   Los pines RTC GPIO se encuentran ligados al coprocesador, el cual a través del uso de estos pines es capaz de despertar al módulo cuando este ha entrado en el modo de ahorro de energía "*Deep Sleep*".

6. **Pines PWM:** El ESP-WROOM-32 posee 16 canales PWM independientes, que pueden ser asignados a cualquiera de los GPIOs que pueda ejercer como salida digital [13].
7. **Pines I2C:** 21 (SDA) y 22 (SCL).

   Estos dos pines pueden ser configurados como SDA o SCL, salvo que estemos trabajando en el IDE de Arduino, donde la configuración esta predeterminada como se indica arriba.
8. **Pines SPI:** 23 (VSPI MOSI), 19 (VSPI MISO), 18 (VSPI CLK), 5 (VSPI CS), 13 (HSPI MOSI), 12 (HSPI MISO), 14 (HSPI CLK) y 15 (HSPI CS).
9. **Pines UART:** 3, 1 (RX y TX de la UART0), 9, 10, 6, 11 (RX, TX, CTS y RTS de la UART1), 16, 17, 8 y 7 (RX, TX, CTS y RTS de la UART2).

   Hay que tener en cuenta que los GPIO 6, 7, 8, 9, 10 y 11 son usados internamente por la memoria Flash se ha visto en apartados anteriores, teniendo en cuenta que los pines RTS y CTS prácticamente no se utilizan en los nuevos microcontroladores debido a que las a velocidades que estos trabajan no hace falta preocuparte por si el bus es ocupado o no. Por lo que en principio solo podríamos utilizar la UART0 y la UART2, sin embargo, existe la posibilidad de cambiar los puertos TX y RX de la UART1 a otros GPIOs por medio de software.
10. **Interrupciones:** Cualquier GPIO se puede configurar como interrupción.
11. **Pines Táctiles Capacitivos:** 0, 2, 4, 12, 13, 14, 15, 27, 32 y 33.

    Estos GPIO posee sensores capacitivos capaces de diferenciar variaciones de carga eléctrica, a menudo son utilizados con sustitutos de los botones mecánicos. Además, son capaces de despertar a microcontrolador cuanto se encuentra en modo "*Deep Sleep*".
12. **Pines de Arranque:** 0, 2, 4, 5, 12 y 15.

    Hay que tener especial cuidado a la hora de tratar con estos pines ya que intervienen en el proceso de arranque o flashing, realizando la carga de programas, la secuencia de parpadeo o el restablecimiento de la placa [14].
13. **Pines a Nivel Alto durante el Arranque:** 1,3, 5, 6, 7, 8, 9, 10, 11, 14 y 15.

    Estos pueden generar PWM o cambiar a nivel alto durante el proceso de arranque, por lo que si tenemos alguno de estos pines configurado como salida puede dar lugar a comportamientos inesperados.

4.03 Optimización ESP32

Puesto que la principal limitación a la que está expuesto el proyecto es el reducido espacio del que se dispone para realizar el montaje, es necesario optimizar el esp32 eliminando los componentes innecesarios para el sistema.

Para realizar dicha optimización se partirá del modelo estructural de la placa de desarrollo ESP32 DevKitC V4 con el microcontrolador ESP-WROOM-32D [15], en la Figura 2 puede verse dicha placa de desarrollo.

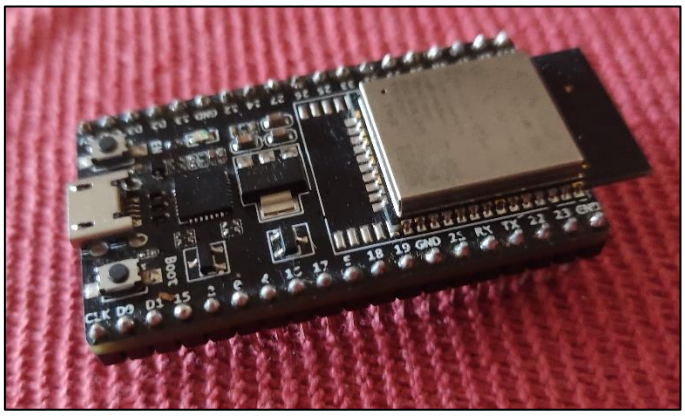


Figura 2. Placa de desarrollo ESP32 DevKitC V4

El primer componente del que se prescindirá es el diodo Led y su resistencia en serie ya que para cualquier indicación luminosa que se quiera realizar, puede ser ejecutada por cualquiera de los diodos Led que se encuentran en las caras del cubo.

El siguiente elemento a descartar es el conector Serie-USB, ya que debido a el proyecto estará alimentado por una batería y programado por medio de un convertidor Serie-USB externo, no tiene mucho sentido mantener este componente que, además, ocupa un espacio considerable en la placa.

Otro elemento que no será necesario tampoco al eliminar el conector Serie-USB es el chip conversor CP2102 o el CH340G (dependiendo de la versión lleva uno u otro), ambos son conversores USB a TTL, cuya función es la de facilitar la comunicación entre el PC y el microcontrolador por medio de protocolo USB. El conversor Serie-USB externo que se utilizará para programar el ESP-WROOM-32 ya posee un conversor USB a TTL, por lo que ya no será necesario este tipo de componente.

Puesto que uno de los objetivos del proyecto es la programación del dispositivo manipulando lo mínimo posible la estructura física del cubo en el momento de la operación, los pulsadores BOOT y EN tampoco serán necesarios.

A continuación, se describirán los componentes que si se mantienen en el diseño de los SmartCubes. Por un lado, se mantendrá un condensador de 0.1uF en paralelo con la alimentación, cuya función es la de filtrado de ruido en la señal DC y otro de 100uF (u otro valor en un rango de 10-100uF generalmente) que se encargará de estabilizar la señal de alimentación durante periodos de variación de ésta [16]. En cuanto al regulador de tensión, no se eliminará del circuito, pero sí que será intercambiado por otro con una menor caída tensión, al que le acompañaran sus condensadores de filtrado. Por último, se mantendrá el circuito de auto programación que consta de 2 transistores y 2 resistencias.

Finalmente, en la Figura 3, Figura 4 y Figura 5 se pueden observar los circuitos esquemáticos que se conservan en el proyecto.

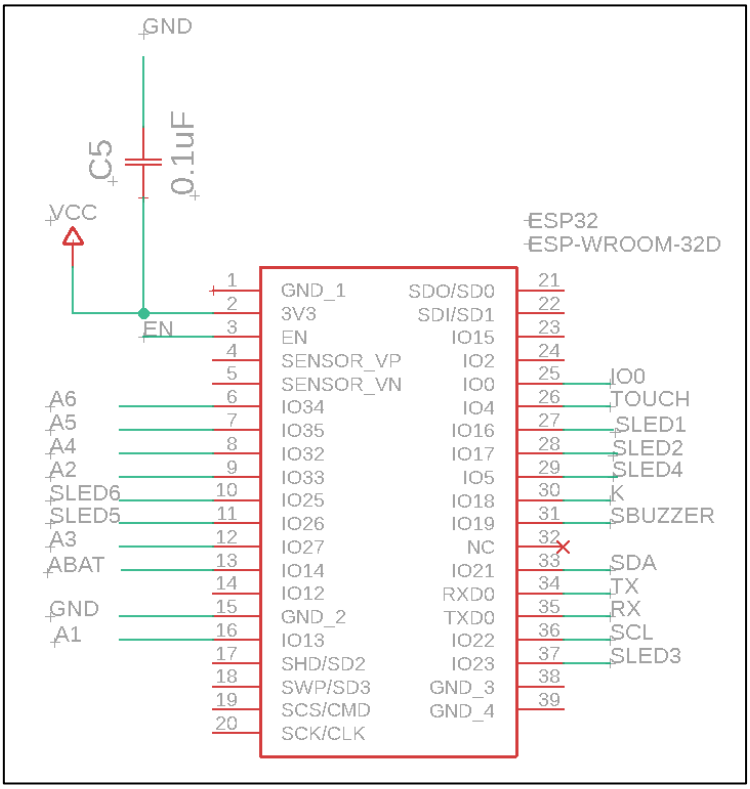


Figura 3. Esquemático del ESP-WROOM-32 con Condensador de Filtrado de Ruidos

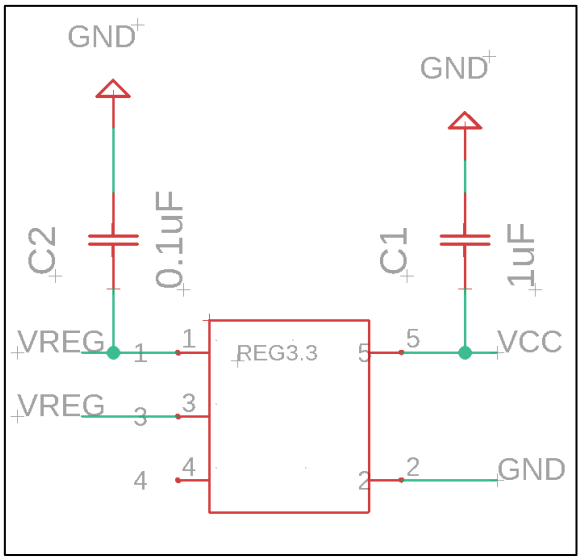


Figura 4. Esquemático del Circuito del Regulador de Tensión

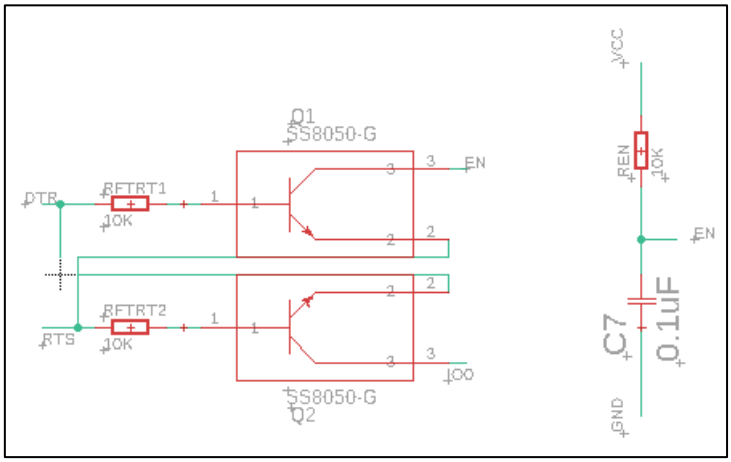


Figura 5. Esquemático del Circuito de Auto Programación

Capítulo 5:
Selección de Hardware

5.01 SmartCubes

5.02 Plataforma de Carga

5.03 Auxiliar de Programación

Capítulo 5: Selección de Hardware

En este apartado se verá el *hardware* elegido para la realización del proyecto, obviando componentes como resistencias, condensadores y transistores que estarán presentes en muchos de estos sistemas. Se diferenciará entre hardware presente en el interior de los SmartCubes, *hardware* presente en la plataforma de carga y *hardware* auxiliar para la programación del microcontrolador.

5.01 SmartCubes

Acelerómetro

Un acelerómetro como su propio nombre indica, es un dispositivo capaz de medir aceleraciones, las cuales pueden proceder de la acción de fuerzas estáticas o dinámicas, por un lado, las aceleraciones estáticas son debidas a la acción de la gravedad, mientras que las aceleraciones dinámicas son originadas por las vibraciones y el movimiento. En este proyecto utilizaremos un acelerómetro de 9 ejes, este tipo de acelerómetro se componen por un acelerómetro de 3 ejes, un magnetómetro de 3 ejes y un giroscopio de 3 ejes. El magnetómetro mide los campos magnéticos y calcula un punto de referencia fijo basado en los campos magnéticos de la Tierra, esta información junto con la del acelerómetro y el giroscopio generan una dirección del movimiento, con la que podemos saber los grados que ha cambiado la dirección, su relación con el polo norte magnético o el cabeceo, guiñada y balanceo. La combinación de sensores de permite corregir desviaciones en el trazado del movimiento, especialmente útil en aplicaciones donde la medición se realiza durante largos periodos de tiempo, lo que se traduce en resultados dinámicos más robustos.

El módulo escogido para este proyecto es el MPU-9150 [17], módulo de 9 ejes que surge de la combinación de dos chips, por un lado, tenemos el MPU-6050, que contiene un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un procesador de movimiento digital el cual es capaz de combinar las medidas de los 3 sensores y trabajar con sistemas de 9 ejes; y el otro lado tenemos el AK8975, que contiene una brújula digital o magnetómetro de 3 ejes (Figura 6).

El acelerómetro de 3 ejes permite medir aceleraciones con un rango programable de ± 2g, ± 4g, ± 8g y ± 16g, siendo g la aceleración de la fuerza de la gravedad. Las medidas son tomadas en un sistema cartesiano definido por el acelerómetro, a partir de los datos

obtenidos se pueden calcular la inclinación, la dirección y la velocidad. No obstante, que es el sistema de referencia sea el propio acelerómetro limita el sistema a ciertas condiciones de funcionamiento, concretamente, si el acelerómetro comenzara a rotar sobre uno de sus ejes, este sería incapaz de detectar cambios de velocidad. Por ello, la implementación de un giroscopio capaz de medir la aceleración y velocidad angular, es la solución a este problema [18]. El giroscopio que integra el MPU-9150 posee un rango de escala completa de ± 250, ± 500, ± 1000 y ± 2000 º/s, con una sensibilidad máxima de 131 LSBs/(º/s). Por último, la brújula digital aporta un sistema de referencia fijo midiendo los campos magnéticos con un rango de escala completa de ±1200 µT.

La comunicación con el módulo puede realizarse por medio de interfaz SPI o I2C, para este proyecto se empleará este último.

El módulo posee un rango de alimentación entre 2.4 a 3.6 V, por lo que como veremos más adelante, no supondrá ningún problema.

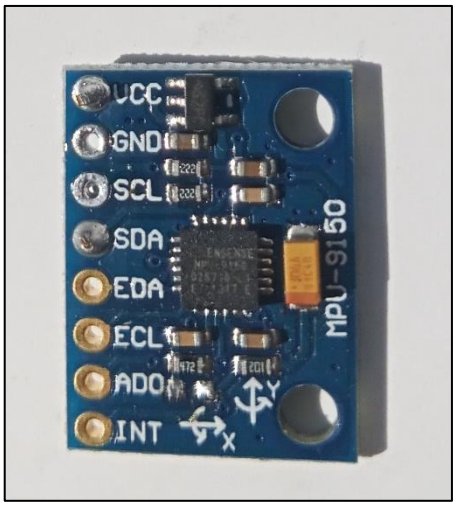


Figura 6. MPU9150

Galgas Extensiométricas

Las galgas extensiométricas son sensores que miden la deformación tras aplicar una fuerza sobre el sólido al que están adheridas, en definitiva, son resistencias cuyo valor nominal varía con la deformación. La opción elegida para este proyecto han sido las galgas extensiométricas BF350 3AA [19], unas galgas de aleación de cobre-níquel de reducidas dimensiones (7.1 x 4.8 mm) y precio muy económico (Figura 7), el objetivo de estas galgas es medir la fuerza que los niños aplican sobre las caras de los cubos, tratando de diferencias varios niveles de fuerza.

El valor nominal de las galgas es de 349.8 Ω con una tolerancia de ±1 Ω (±0.1%), el factor de galga o coeficiente de sensibilidad se encuentra en un rango de 2-2.2 con una tolerancia de ±1% y el límite de la tensión que pueden soportar estas galgas está en las 2000 μDeformaciones aproximadamente.

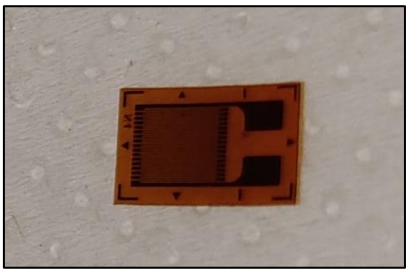


Figura 7. Galga Extensiométrica BF350 3AA

Leds

Se emplearán diodos Led en cada cara del cubo para llamar la atención del niño e incluso en un futuro para realizar algún tipo de juego interactivo aprovechando las galgas extensiométricas, también son pueden ser utilizados con señales indicativas de algún estado del SmartCube.

Buzzer

El *buzzer* o zumbador es un dispositivo capaz de emitir un sonido monótono de forma continua o intermitente. El objetivo de *buzzer* en este proyecto es la de distraer al niño mientras juega con el cubo, para esta versión conservaremos el *buzzer* empleado en anteriores versiones, el CMT-1102-SMT-TR (Figura 8) [20], un zumbador de dimensiones reducidas y un amplio rango de tensiones de alimentación que permite a emplear 3.3V.

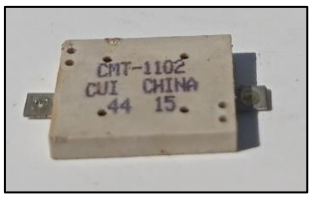


Figura 8. Buzzer

Conector Magnético

El conector magnético es un conector normal de pines tipo pogo con dos imanes en los extremos que garantizan una buena conexión (Figura 9), este conector es utilizado en los terminales de carga de la batería y tiene un conector complementario en la plataforma de carga. Es importante hablar de este conector no por la función que cumple en el conjunto, si no por la importancia que tiene su propiedad magnética en la elección del próximo componente.

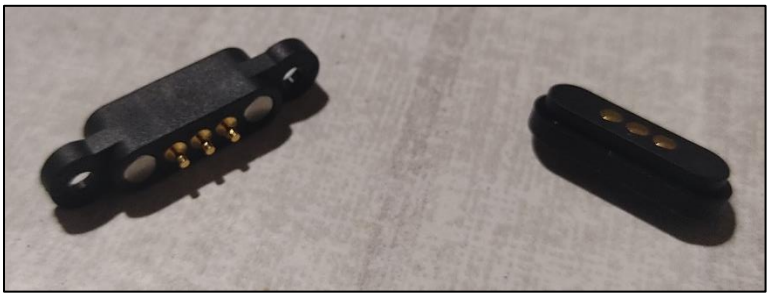


Figura 9. Conectores magnéticos

Relé de Estado Sólido

La función que tiene el relé en el sistema es la de conmutar entre el estado de carga de la batería y el de alimentación de la batería, para ello, en anteriores versiones se empleaba un relé reed, un relé que conmutaba con la presencia de un campo magnético, de ahí la importancia del conector magnético. Con la incorporación del conector magnético el relé reed permanecería siempre en estado de conmutación, por lo que se debe encontrar una alternativa que funcione en presencia de campos magnéticos. Los relés de estado sólido son una muy buena opción, son interruptores electrónicos que conmutan al aplicar una pequeña corriente entre sus terminales de control, además, una de las ventajas que tienen este tipo de relés es que, al no poseer partes mecánicas, se evitan fallos debidos al desgaste de los materiales y como consecuencia directa, el tiempo de vida útil de estos dispositivos es mayor.

El relé escogido es el LCC120S [21], un relé de estado sólido con dos polos, un polo normalmente cerrado y otro normalmente abierto (Figura 10).

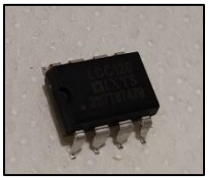


Figura 10. Relé de Estado Sólido LCC120S

Batería

En anteriores versiones se ha utilizado baterías de LiPo ya que este tipo de baterías ofrece prestaciones superiores frente a otras baterías como podrían ser las NiCd o las NiMH. Esto se debe en parte a que pueden almacenar grandes cantidades de energía sin necesitar de grandes dimensiones para ello, además poseen tasas de descargas bastante grandes, con lo que pueden generar corrientes muy elevadas si el sistema lo requiriera [22] [23].

A diferencia de las versiones previas donde alimentábamos con una batería de 3.7V 150mAh, en esta nueva versión lo hacemos con una batería de LiPo de 3.7V 250mA (Figura 11) aprovechando que vamos a aumentar las dimensiones de los cubos al incorporar el ESP-WROOM-32 [24].

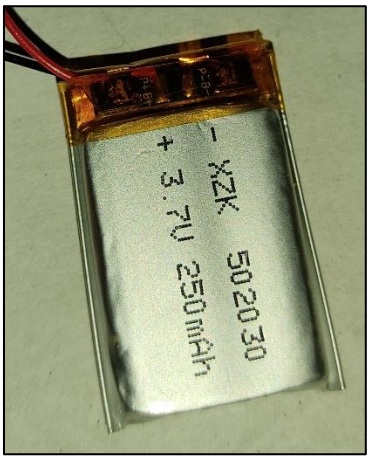


Figura 11. Batería de LiPo de 3.7V 250mAh

Pese a que el voltaje nominal de este tipo de baterías ronda los 3.7V, el voltaje final tras la carga y con el que comienzan a descargase es de 4.2V, valor a tener en cuenta a la hora de alimentar nuestros circuitos. El anterior microcontrolador, el ATMEGA328p-AU, tenía un rango de alimentación entre 1.8 y 5.5V, lo que permitía alimentar el sistema directamente con la batería, sin embargo, en el caso del ESP-WROOM-32 la tensión de alimentación que establece el fabricante debe encontrase entre los 2.3 y 3.6V, si alimentamos directamente con la batería no está garantizado que el sistema se comporte de la forma deseada, pudiéndose dañar los componentes. Por ello, es necesario introducir

un regulador de tensión que sitúe el voltaje de alimentación dentro de los valores límite, en el próximo apartado se tratará el regulador en cuestión.

El objetivo a nivel energético es conseguir un consumo lo más reducido posible, siendo la autonomía mínima requerida para este proyecto de 4-5 horas, duración habitual de una sesión de especialistas recopilando datos con niños.

A continuación, se realizará una pequeña estimación de la autonomía teniendo en cuenta los consumos promedios más remarcables del sistema.

Los componentes que más consumo generan son el ESP-WROOM-32, los diodos Leds, las galgas extensiométricas, el acelerómetro y el *buzzer*. En el caso de los diodos Leds, cada Led tiene su control independiente y en principio nunca habría más de un diodo Led encendido a la vez, por lo que solo habría que tener en cuenta el consumo promedio de uno de ellos. Las galgas extensiométricas están configuradas como un conjunto de divisores resistivos (resistencia del divisor de 10KΩ) controlados por un único transistor que actúa como interruptor, dejando pasar la corriente en el momento que se vaya a realizar la toma de medidas, siendo el consumo total la suma de los 6 consumos.

- ESP-WROOM-32 (Wifi y BT apagado, CPU a 80MHz) → 30 mA [9]
- Led → 20 mA [25]
- Galgas extensiométricas (BF350 3AA) → 2.15 mA
- Acelerómetro (MPU-9150) → 4.25 mA [17]
- *Buzzer* → 4.5 mA [20]

La peor de las situaciones sería el caso en el que todos estos consumos se dieran de forma simultánea y constante, obteniendo un consumo total de 60.9 mA durante un periodo de 4.10 horas. Esta hipotética situación no es realista ya que los Leds, las galgas y el buzzer no estarán funcionando a la vez ni de forma continua en ningún caso, pero nos sirve para asegurar una autonomía teórica del juguete superior a las 4 horas.

Además, el ESP-WROOM-32 entrará en un modo ahorro los periodos de tiempo entre tomas de medidas y envío de información. Asimismo, cuando el cubo detecte que nadie está jugando con él (no reciba datos nuevos durante un periodo de tiempo determinado) entrará en modo "*Deep Sleep*", mayor modo de ahorro que posee el ESP-WROOM-32 en el que se suspenden todas las funciones salvo el reloj en tiempo real (RTC) y un coprocesador encargado de detectar interrupciones externas que despierten al módulo, el

consumo típico en este modo es de 10 uA. En este proyecto despertaremos al módulo aprovechando los sensores capacitivos que posee, de tal forma que cuando el niño toque la superficie de la carcasa, el cubo retomará su normal funcionamiento. De esta forma, optimizamos el consumo de energía consiguiendo que la autonomía del juguete aumente considerablemente.

Regulador de Tensión

El objetivo del regulador de tensión es el de mantener un nivel de tensión constante a partir de una tensión superior mínima. Para este proyecto necesitamos un regulador de tensión para transformar la tensión de la batería en una tensión dentro del rango de alimentación del ESP-WROOM-32, entre 2.3 y 3.6 V. Otro punto a tener en cuenta es el voltaje *drop* o caída de tensión, ya que cuanto menor sea esta mayor tensión útil de la batería tendremos.

$$Vin\,mín = Vreg + Vdrop$$

La elección para el regulador de tensión ha sido el TCR2EF33 [26], un regulador de baja tensión de *drop* y respuesta transitoria rápida, este modelo concreto proporciona una tensión de salida de 3.3V que es la tensión de alimentación recomendada por el fabricante del ESP-WROOM-32.

La tensión de *drop* depende de la corriente de salida, por lo que teniendo en cuenta que el consumo promedio del sistema se encuentra por debajo de los 50 mA, se puede asegurar un voltaje de *drop* por debajo de los 50mV, podemos ver la gráfica correspondiente en el Anexo I.

Suponiendo el voltaje de *drop* máximo 50 mV, la tensión de entrada a partir de la cual el regulador será incapaz de mantener 3.3V a la salida será de 3.35V. Podría parecer que se está desperdiciando batería ya que como se ha dicho anteriormente las baterías de LiPo tienen como límite de descarga los 3V, valor a partir del cual pueden sufrir daños irreparables, pero lo cierto es que en este tipo de baterías la tasa de descarga se acelera drásticamente a partir de los 3.5-3.4 V, por lo que no existe una gran pérdida energética.

Circuito de Protección por la Descarga de la Batería

Se utilizará el mismo circuito de protección que en anteriores versiones de los SmartCube, formado por los circuitos integrados MCP112T-315E/LB y AP2281-3WG-7, el primero es un circuito integrado supervisor y el segundo un interruptor de alimentación. Este

sistema es necesario ya que las baterías de Li-Po no deben descargarse por debajo de los 3V, ya que de lo contrario de deteriorarían irreversiblemente.

El funcionamiento de este sistema es el siguiente, el circuito integrado supervisor o detector de tensión, evalúa la tensión de la batería y genera un nivel alto si la tensión supera cierto umbral, en este caso dicho umbral estaría ligeramente por encima de los 3V. A continuación, al interruptor de alimentación le llegan tanto la tensión de la batería como el nivel alto previamente generado, que actuaría como *Enable*, y permitiendo el paso de la tensión de la batería hacia el resto del sistema. Si, por el contrario, la tensión de la batería se encuentra por debajo del umbral, el detector de tensión generaría un nivel bajo que bloquearía el paso de la alimentación.

5.02 Plataforma de Carga de las Baterías

Banco de Energía Portátil

Un banco de energía o *power bank* es una batería recargable con conexión a puerto USB, capaz de proporcionar una alimentación de 5V. En este proyecto se encargará de proporcionar la energía necesaria para cargar la batería de los SmartCubes y de establecer en todo momento un nivel alto en la patilla de control de carga, lo que permitirá que el juguete deje de funcionar cuando está conectado a la plataforma de carga.

Módulo de Carga

Para cargar las baterías de LiPo es necesario utilizar un módulo de carga, ya que estas baterías requieren de ciertas condiciones para realizar la carga correctamente sin deteriorarse. El componente elegido es un módulo de carga de baterías de litio basado en el chip TP4056 [27], que garantiza una corriente de carga de 0.37C, idónea para este tipo de baterías (Figura 12).

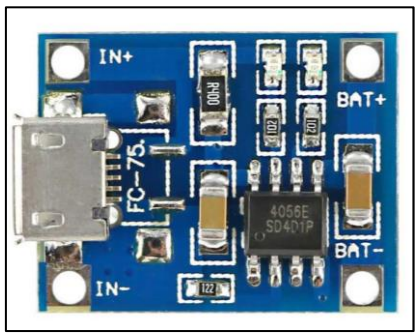


Figura 12. Módulo Cargador TP4056

5.03 Auxiliar de Programación

Convertidor Serie-USB Externo

Como se ha explicado anteriormente, para programar el ESP-WROOM-32 se empleará un convertidor Serie-USB externo, capaz de establecer una comunicación entre el PC y el microcontrolador por medio de señales TTL, las cuales establen una relación entre las interfaces USB del PC y serie (UART) del microcontrolador.

El adaptador elegido para realizar la tarea de programación, es un adaptador basado en el chip FT232RL (Figura 13).

Durante dicho proceso, se utilizarán los siguientes pines del convertidor: TX, RX, GND y 3.3V, adicionalmente, para realizar la carga automática del programa se emplearán también los pines terminales DTR (terminal de datos preparados) y RTS (terminal de solicitud de envío).

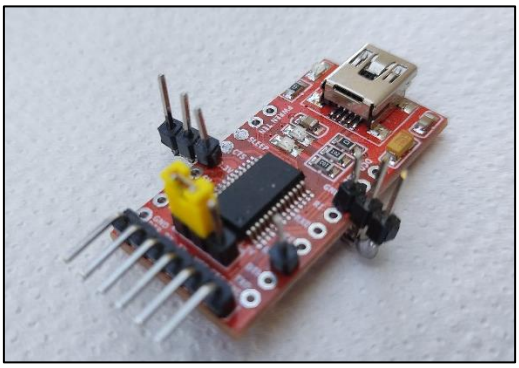


Figura 13. Convertidor USB-Serie basado en el chip FT232RL

Capítulo 6:
Sistemas Principales. Explicación, Esquemático y Prueba de Funcionamiento

6.01 Galgas Extensiométricas

6.02 Acelerómetro

6.03 Leds

6.04 Buzzer

6.05 Sistema de Carga de la Batería

6.06 Sistema de Protección por la Descarga de la Batería

6.07 Sistema de Medición del Estado de la Batería

6.08 Sistema de Programación

- 6.08.1 Programación con Convertidor Serie-USB
- 6.08.2 Programación OTA (*Over the Air*)

6.09 Transmisión de Datos (*Bluetooth Low Energy*)

6.10 Modos de Ahorro de Energía y Sistema de Detección Capacitiva

Capítulo 6: Sistemas Principales. Explicación, Esquemático y Pruebas de Funcionamiento

En este apartado, se introducirá el funcionamiento de las principales funciones que integra el SmartCube, se verá el esquema de montaje utilizado en cada uno de los sistemas y se realizará pruebas individuales de los elementos donde sea interesante verificar su funcionamiento junto nuevo microcontrolador.

6.01 Galgas Extensiométricas

La forma en la que se mide la fuerza por medio de galgas extensiométricas consiste en tomar una medida de tensión que varía conforme varía la resistencia de la galga, a la que posteriormente se le asignará un valor de fuerza gracias a una linealización con valores estándar, es decir, se realizan pruebas con calibres de pesos para los cuales se obtienen unos determinados valores de tensión y con esos valores se crean tendencias de futuros valores no medidos con el empleo de técnicas de linealización.

Existen diferentes formas y configuraciones de obtener valores de tensión empleando galgas extensiométricas, pero las más eficientes y utilizadas suelen ser empleando una configuración puente de Wheatstone junto con un amplificador. Del puente de Wheatstone obtenemos un voltaje de salida que es la resta entre los dos divisores resistivos que conforman el puente, dicho voltaje es nulo cuando el puente se encuentra equilibrado (no actúa ninguna fuerza sobre las galgas) y toma valores muy pequeños en estado de desequilibrio (fuerzas actuando sobre las galgas), este voltaje será convertido a un valor digital por medio de un convertidor analógico-digital (ADC), por ello, para trabajar con un rango más amplio de valores es necesario aumentar la señal de salida por medio de un amplificador. Existen muchas configuraciones dentro del puente de Wheatstone, utilizando una o varias galgas y variando su colocación, no obstante, las más interesantes en cuanto a términos de medición son las dos siguientes, ya que, en ellas la tensión de salida no presenta offset, es una señal lineal (sensibilidad constante) y la variación de temperatura no tiene influencia alguna.

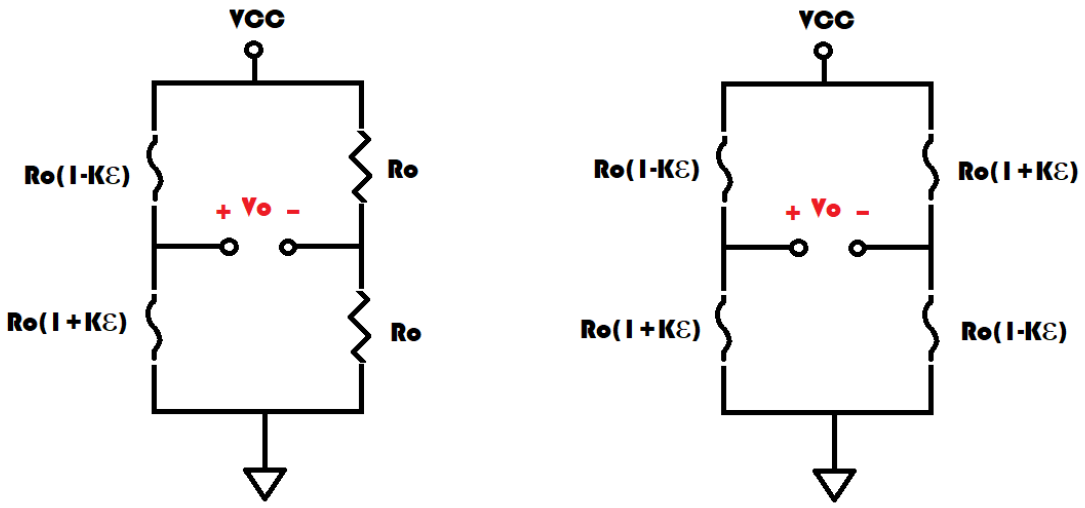


Figura 14. Configuración de Puente Wheatstone con Vo sin Offset, lineal e invariante frente a la temperatura.

Pese a que las anteriores configuraciones son las idóneas para aprovechar al máximo las galgas extensiométricas, consiguiendo mediciones de gran precisión, en este proyecto hay que tener en cuenta la limitación del espacio disponible y el requerimiento de precisión necesario para tomar medidas. Por un lado, ante un hipotético caso en el que se implementara una configuración de puente de Wheatstone, se tendrían que emplear 4 galgas o resistencias por cara del cubo, además de un amplificador operacional con sus resistencias asociadas (por lo general se utilizan 2 resistencias), en total, el sistema de galgas requería entre 12-24 galgas extensiométricas, 12-24 resistencias y 6 amplificadores operacionales; un número excesivo de componentes teniendo en cuenta el limitado espacio del que se dispone. Por otro lado, el proyecto no requiere de una precisión milimétrica a la hora de tomar las medidas, ya que el objetivo de las galgas es dotar al juguete de la capacitad de poder distinguir diferentes niveles de fuerza sin tener en cuenta en valor de la fuerza. Atendiendo a estos dos factores, la configuración en puente de Wheatstone queda descartada, teniendo que buscar una configuración que requiera de la mínima cantidad de componentes posible y permita establecer varios niveles de fuerza.

La configuración que más se ajusta a estos requerimientos es un divisor resistivo formado por una resistencia y una galga extensiométrica, y es la configuración escogida para este proyecto (Figura 15 y Figura 16).

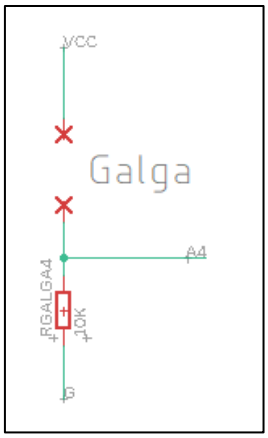


Figura 15. Configuración de las galgas como divisor resistivo

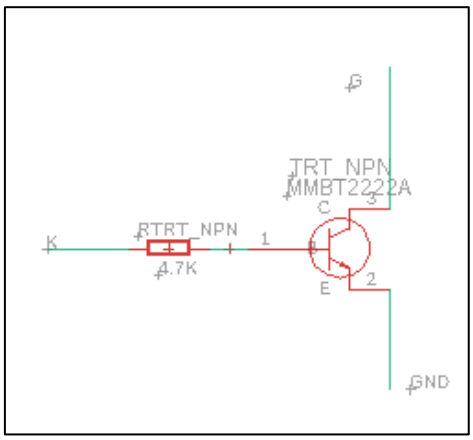


Figura 16. Transistor que habilita la medición de las galgas

Para evitar consumos innecesarios durante los periodos de tiempo en los que no se realizan mediciones, se incorpora un transistor que actuará como interruptor para habilitar y deshabilitar la circulación de corriente.

Esta configuración requiere a nivel de hardware únicamente 7 resistencias, 6 galgas y 1 transistor, una cantidad de componentes mucho más reducida que con el puente de Wheatstone, satisfaciendo así el requerimiento del espacio.

Para que la opción del divisor resistivo sea viable, no solo tiene que cumplir la limitación del espacio, por lo que se tiene que comprobar si con esta configuración es posible diferenciar varios niveles de fuerza.

A continuación, se muestran los pines del ESP-WROOM-32 que intervienen en el sistema de medición de las galgas (Tabla 2).

Función que desempeña	GPIO
Entrada analógica Galga 1, (A1)	13
Entrada analógica Galga 2, (A2)	33
Entrada analógica Galga 3, (A3)	27
Entrada analógica Galga 4, (A4)	32
Entrada analógica Galga 5, (A5)	35
Entrada analógica Galga 6 (A6)	34
Salida digital Transistor (K)	18

Tabla 2. Pines asociados al sistema de medición de las galgas extensiométricas

Pruebas de Funcionamiento

Para probar el funcionamiento de las galgas y verificar que esta configuración permite diferencias varios niveles de fuerza, se realiza el montaje del divisor resistivo y se carga un programa que leerá la tensión por medio de un de los canales del ADC1 del ESP-WROOM-32.

Antes de tomar las medidas es necesario configurar el ADC [28], el ESP-WROOM-32 cuenta con dos ADCs, el ADC1 orientado a la medición de valores analógicos externos ya que viene configurado de fábrica de tal forma, que solo tengas que establecer la precisión y la atenuación del ADC, las cuales se aplicarán directamente a todos los canales. Por otro lado, el ADC2, aunque no esta tan preparado para la toma de medidas, ya que este ADC comparte pines con la comunicación WIFI y, a diferencia del ADC1, el ADC2 es necesario configurar cada canal por separado, en lugar de configurar el ADC directamente.

- Precisión. El rango de precisión seleccionable es de 9-12 bits.
- Atenuación.
  - 0db → Rango de lectura 0-1.1 V
  - 2.5dB → Rango de lectura 0-1.5 V
  - 6dB → Rango de lectura 0-2.2 V
  - 11dB → Rango de lectura 0-Vcc V

- Valor de Calibración. Este valor se utiliza en para generar la curva característica y que tienen en cuenta el voltaje de referencia del ESP-WROOM-32. Existen tres posibles valores de calibración dependiendo del modelo del microcontrolador.
  - *Two Points*. Dos valores que se corresponden con las lecturas de cada ADC a 150 mV y 850 mV, estos dos valores son grabados en un eFuse (fusible electrónico que permite reprogramar chips a tiempo real).
  - eFuse Vref. Voltaje de referencia medido en la calibración de fábrica y grabado en el eFuse.
  - *Default Vref*. Valor proporcionado por el usuario que es utilizado en caso de que el eFuse Vref y el *Two Points* no estén disponibles. Puede tomar valores entre 1000-1200, siendo 1100 el valor predeterminado.

Para esta primera prueba establece la precisión del ADC1 en 12 bits, la atenuación en 6dB y utilizamos el valor eFuse Vref para calibrar el ADC, con esta configuración se tomarán medidas cada segundo.

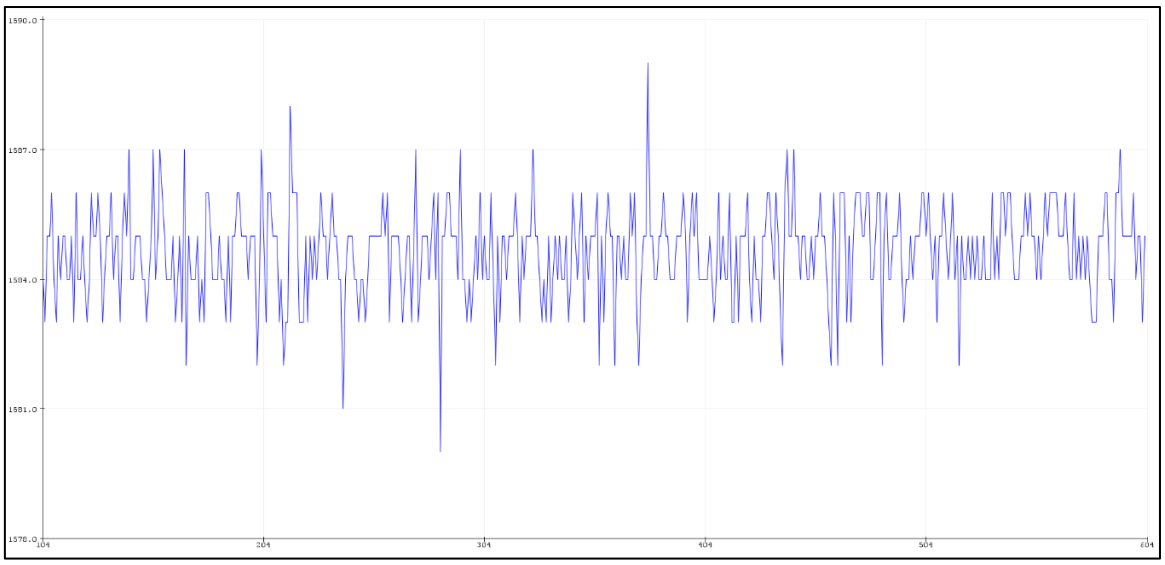


Figura 17. Muestreo medición de tensión de las galgas

Como se puede observar en la Figura 17, se obtienen valores de tensión en torno a 1584-1585 mV, la señal obtenida del muestreo presenta mucho ruido acoplado, lo que impide una lectura clara y estable.

Para eliminar parte de este ruido, presente comúnmente en este tipo de microcontroladores, se implementará por código un *multisampling*, que consiste en realizar un promediado de un conjunto de muestras por valor de tensión de la galga [29].

En la siguiente prueba hemos empleado un *multisampling* con número de muestras de 64 por medida de tensión.

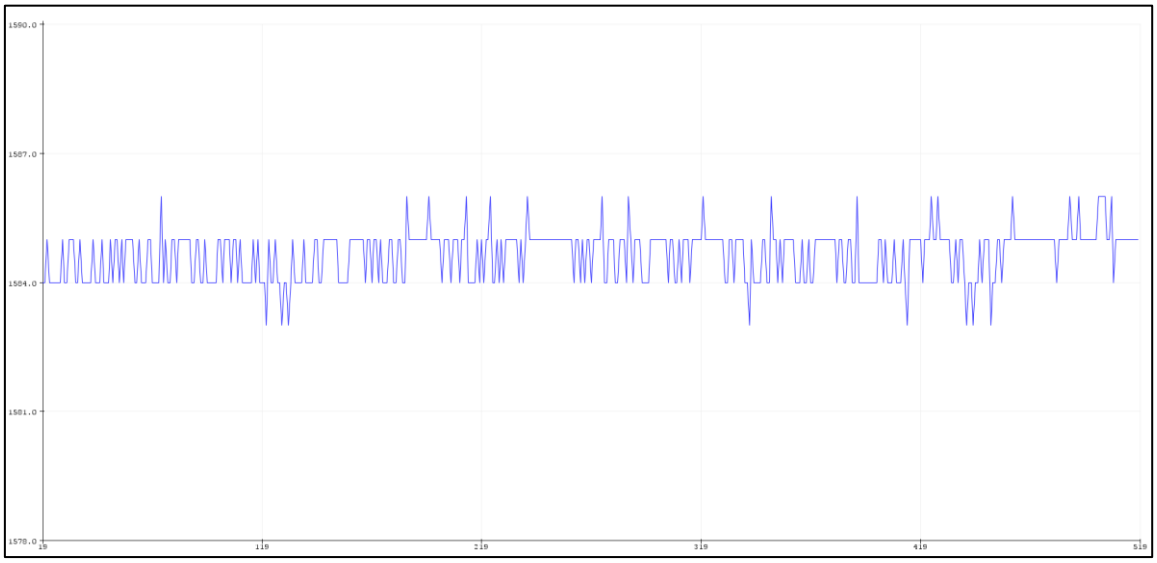


Figura 18. Gráfica de Medición de la tensión de las galgas con Multisampling

Como se puede observar en la Figura 18, se ha reducido considerablemente la presencia del ruido en la señal, agrupando las medidas entorno a los valores más concurridos. Se han implementado también otros filtros basados en el promediado de valores como el filtro de media móvil o los filtros exponenciales EMA, pero el resultado es muy similar al obtenido con el *multisampling*.

Aunque el *multisampling* consigue eliminar gran parte del ruido de la señal, es un filtro poco robusto, ya que, ante la presencia de puntos alejados la media obtenida se desvía mucho, como podemos ver en la figura anterior. Por ello, implementaremos junto con el *multisampling* un filtro de mediana móvil, que hará al sistema más estable frente a este tipo de puntos.

En este caso, se ha utilizado directamente un filtro de mediana móvil en formato de librería [30].

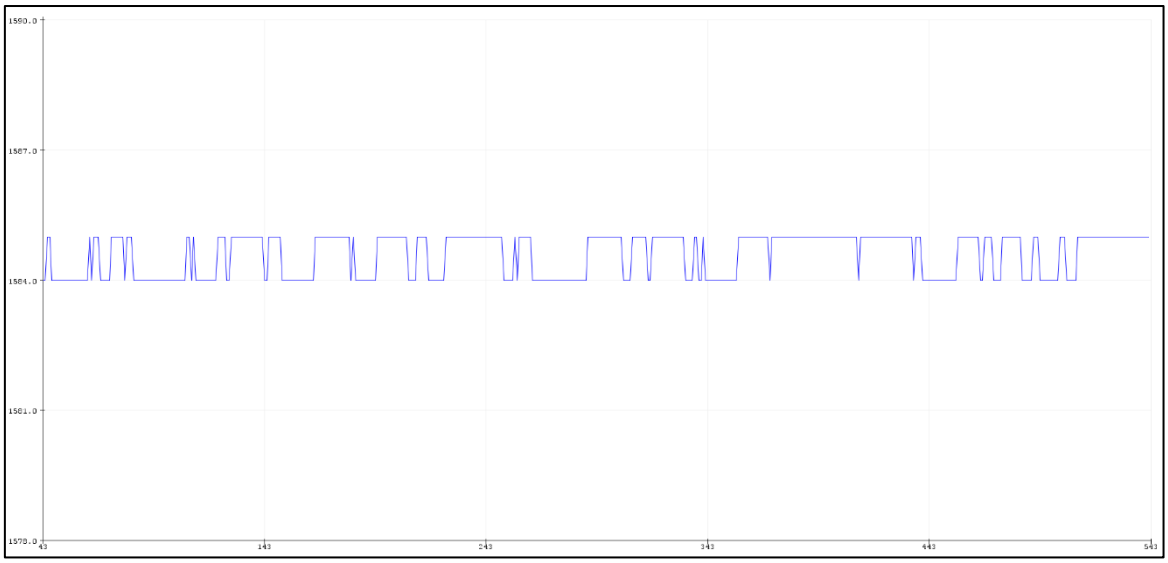


Figura 19. Gráfica de Medición de la tensión de las galgas con Multisapling y Filtro de Mediana

Con este último filtrado la señal queda mucho más limpia, oscilando únicamente entre los valores 1584 y 1585 mV (Figura 19).

A continuación, se verá el comportamiento de la señal ante la acción de una fuerza sobre la superficie a la que está adherida la galga ()Figura 20.

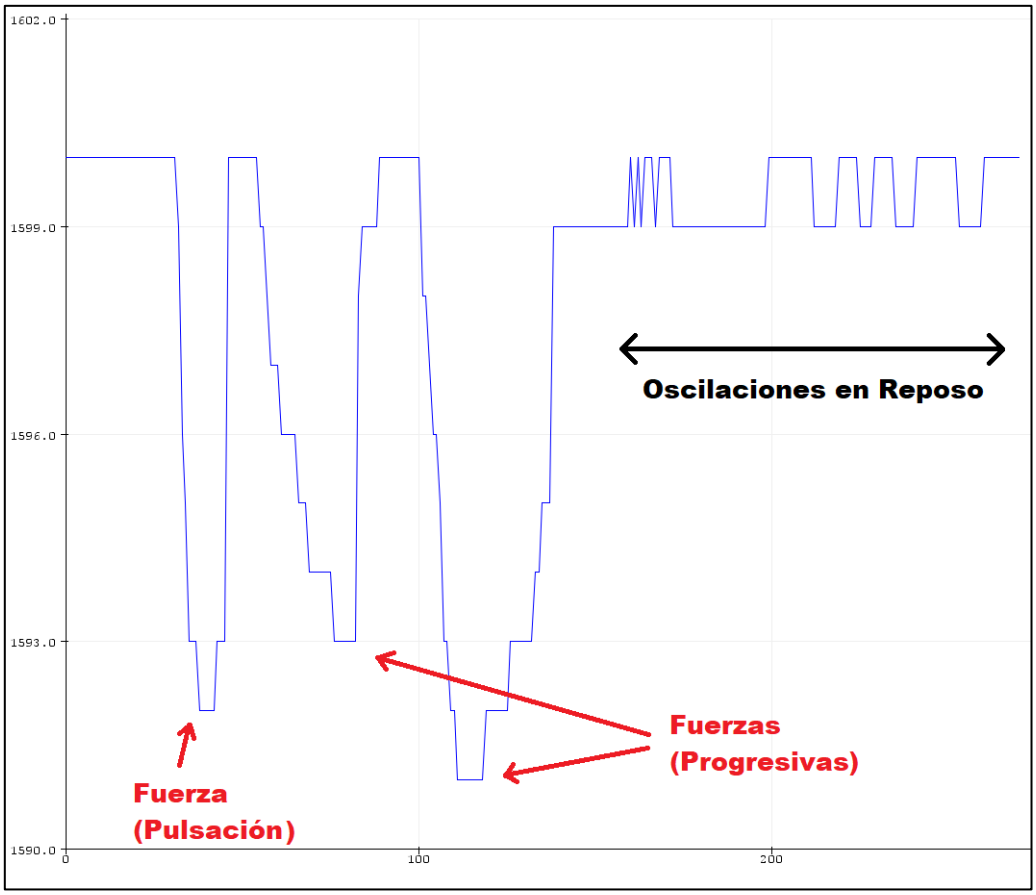


Figura 20. Variación de la tensión de la galga ante la presencia de fuerzas

Una vez comprobado que el sistema es capaz de diferenciar diferentes niveles de tensión, se creará a continuación una escala donde podrán diferenciarse varios niveles de fuerza. Para establecer el nivel 0 de fuerza, establecemos un límite superior de tensión a partir del cual se calcularán el resto de niveles de fuerza, es decir, este límite superior de tensión será el nivel 0 de fuerza y el resto de niveles de fuerza se sacarán de la resta de esta tensión con el resto de tensiones. Si el límite superior de tensión fuera 1600 mV y se aplicara una fuerza mínima que cambie el valor de tensión a 1599 mV, el nivel de fuerza obtenido sería la resta, 1600 – 1599 dando un nivel de fuerza de 1, si la fuerza aplicada generará una tensión de 1593 mV, el nivel de fuerza alcanzado sería de 7.

Previamente antes de comenzar el proceso de medición, se realizar una secuencia de estabilización, donde se tomarán valores de tensión y se establecerá el límite superior como el valor más bajo de los medidos. Este proceso de estabilización permite eliminar las oscilaciones iniciales en reposo que se verían reflejadas en la escala de fuerzas.

A continuación, en la Figura 21 se muestra una prueba donde se alcanzan diferentes niveles en la escala de fuerza.

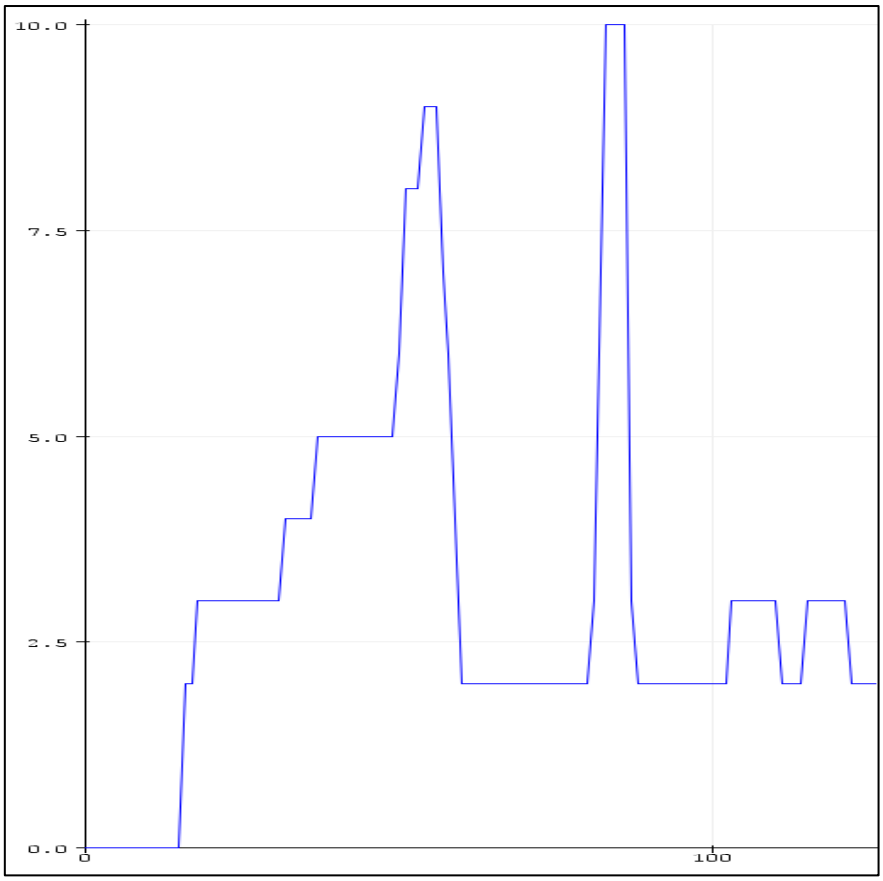


Figura 21. Prueba Niveles de Fuerza

En esta prueba se puede ver que se alcanza un nivel 10 de fuerza, aunque en otras pruebas se ha llegado a niveles superiores, no obstante, al experimentar niveles altos de fuerza puede verse que no retorna al estado de reposo, ya que la galga permanece deformada. Esto no afecta al rango de la escala de niveles de fuerza ya que, la deformación máxima de la galga y la tensión en el nivel 0 siguen siendo los mismos, pero las medidas en los niveles inferiores se ven falseadas.

Se ha comprobado que, con niveles bajos de fuerza, cuando desaparecen las fuerzas, los niveles de fuerza regresan al estado 0.

Por último, se muestra el entorno donde se han realizado dichas pruebas, ya que, en el caso de la galga, la superficie donde irá pegada influye bastante en los resultados. Por ello, se buscado una superficie lo más parecida posible a la que estará adherida en el SmartCube, siendo ésta un circuito impreso. En el caso de estas pruebas, se ha empleado un circuito impreso procedente de un teléfono móvil, ya que en este momento todavía no se disponía de los circuitos impresos de esta versión de los SmartCubes, en la Figura 22 puede observarse dicho entorno de pruebas.

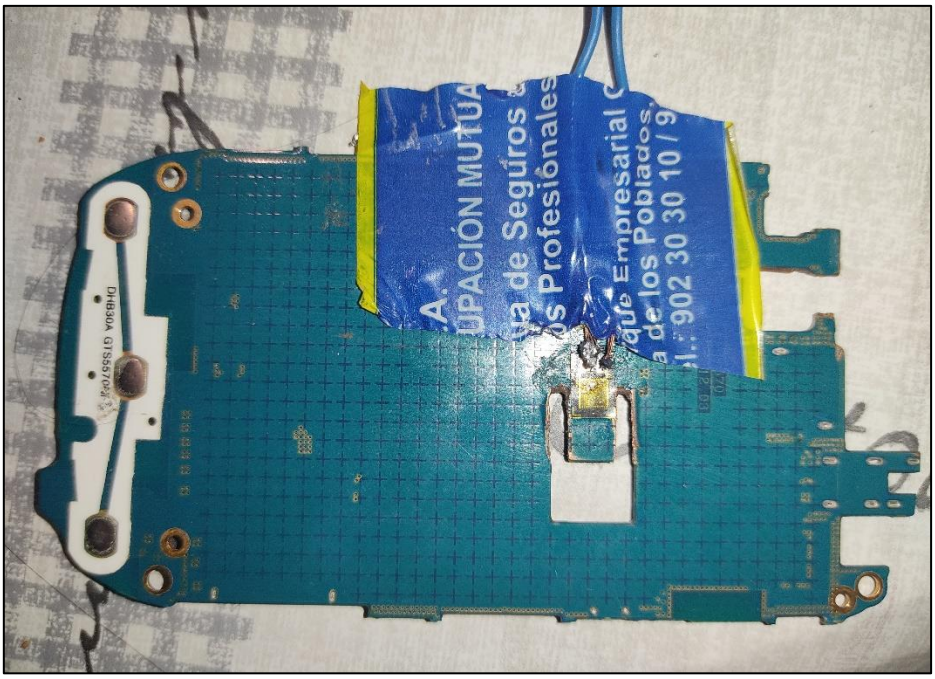


Figura 22. Entorno de prueba de la galga (PCB de un smartphone)

Puede verse el código empleado en el Anexo III.

6.02 Acelerómetro

Para el montaje del acelerómetro, se conectarán los pines SDA (GPIO 21) y SCL (GPIO 22) del ESP-WROOM-32 con los pines correspondientes del módulo, y se alimentará con 3.3V y masa. El resto de pines son innecesarios puesto que únicamente se empleará la comunicación I2C para obtener las medidas del acelerómetro (Figura 23).

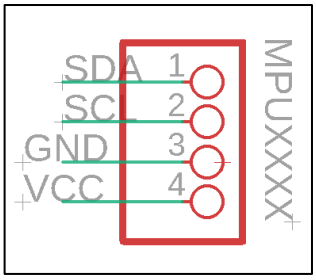


Figura 23. Esquema Montaje Acelerómetro

Con el objetivo de mejorar el rendimiento del acelerómetro se implementará un algoritmo de combinación de datos para la estimación de la orientación, dicho algoritmo será o bien el filtro de *Madgwick* o el filtro de *Mahony*. El tema de los algoritmos de predicción es muy extenso, por lo que únicamente se tratarán brevemente los filtros anteriormente mencionados y se implementará unos ya desarrollados.

Ambos filtros emplean los cuaterniones como forma de representación de la orientación, en el caso del filtro de *Madgwick*, se calcula el cuaternión de la orientación a partir de la velocidad angular (giroscopio) y del vector de observación, para la cual se alinean las direcciones de referencia proporcionadas por el sistema de coordenadas y el campo magnético; una vez obtenidas ambas orientaciones se combinan y se normaliza el cuaternión. En el caso del filtro de *Mahony*, este filtro calcula el error de orientación del instante anterior y se le aplica un controlador PI, con el compensador introducido se logra corregir la velocidad angular que a su vez modifica el cuaternión, y que al integrarla se obtiene la estimación de posición.

Pruebas de Funcionamiento

Para esta prueba de funcionamiento se utilizará una librería específica para el MPU9250, una versión de módulo superior pero compatible con el MPU9150. Esta librería proporciona directamente los valores del acelerómetro, giroscopio y magnetómetro al utilizar las correspondientes funciones. Una vez obtenidos estos valores se emplea el filtro de *Madgwick* para mejorar la precisión de las medidas del acelerómetro.

A continuación, se explicará el funcionamiento del código que se ha desarrollado para esta prueba. Cada ciclo del bucle principal, se realizará un *multisampling* de donde se sacarán varios valores, por un lado, se calculará el módulo de la aceleración de cada tanda de medidas y se sacará un valor promedio del módulo con todos los valores medidos, adicionalmente, se tomará el valor de la máxima aceleración durante el *multisampling*. Por otro lado, con estos valores de aceleración se sacarán los valores de la velocidad, realizando una media de estas velocidades al finalizar el *multisampling*. Por último, se detectará si ha habido una agitación del acelerómetro cada ciclo del programa y si lo ha habido, aumentará el valor de un contador. Se incrementará el contador de agitaciones cuando se detecte una determinada variación de aceleración, la cota mínima se ha establecido por medio de prueba y error.

En la Figura 24 se pueden observar los valores previamente explicados a través del puerto serie. Puede verse el código completo en el Anexo IV.

Figura 24. Datos Mostrados por el Puerto serie (Prueba de funcionamiento Acelerómetro)

6.03 Leds

El montaje de los Leds es tan sencillo como poner una resistencia en serie con el Led entre alimentación y masa, la función de la resistencia es la de establecer una corriente determinada que garantice el funcionamiento del Led sin que llegue a romperse.

En la Figura 25 se muestra el circuito esquemático de los diodos Leds.

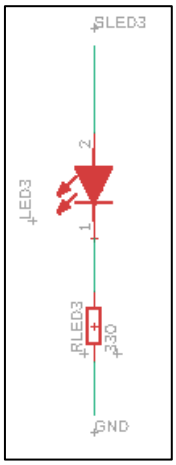


Figura 25. Esquema del Circuito de los diodos Led

Teniendo en cuenta que el voltaje de polarización directa del Led, Vd, es de 1.8V y la corriente nominal, In, es 20 mA, podemos calcular el valor mínimo de la resistencia que garantice el correcto funcionamiento del Led. [31]

$$R\ min = \frac{SLED - Vd}{In} = \frac{3.3\ V - 1.8\ V}{20\ mA} = 75\ \Omega$$

Utilizar resistencias por encima de ese valor, reducirá la luminosidad relativa del diodo Led, pero a cambio ganaríamos una mayor reducción del consumo de este. Por este motivo, el valor de las resistencias de los Leds es de 330 **Ω**, ya que lograr una gran intensidad lumínica no es un objetivo prioritario en este proyecto.

Con este valor de resistencia, la intensidad por el Led es de 4.5mA, reduciendo la intensidad luminosa a un 20% de su valor nominal.

A continuación, se muestra una tabla con terminales GPIO que se encargan del control de los diodos LED.

Led a Controlar	GPIO
Led 1 (SLED1)	16

LED 2 (SLED2)	17
LED 3 (SLED3)	23
LED 4 (SLED4)	5
LED 5 (SLED5)	26
LED 6 (SLED6)	25

Tabla 3. Pines de Control de los Diodos LED

6.04 Buzzer

El *buzzer* únicamente requiere que el voltaje de entrada este dentro del rango de operación, que en este caso es 25 Vp-p como máximo. Puesto que la tensión de operación del ESP-WROOM-32 es de 3.3 V, podemos conectar el GPIO 19 (SBUZZER) directamente al terminal positivo del *buzzer* [20].

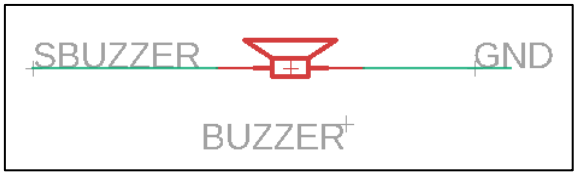


Figura 26. Esquemático del Circuito del Buzzer

6.05 Sistema de Carga de la Batería

El sistema de carga está conformado tanto por elementos externos como internos al juguete, por un lado, tenemos la base de carga que se encarga de realizar la recarga de la batería de Li-Po y, por otro lado, tenemos el relé de estado sólido en el interior del SmartCube, que es el encargado de conmutar entre el estado de consumo y carga de la batería.

A continuación, en la Figura 27 podemos observar un diagrama que muestra de forma muy visual el sistema de carga de baterías.

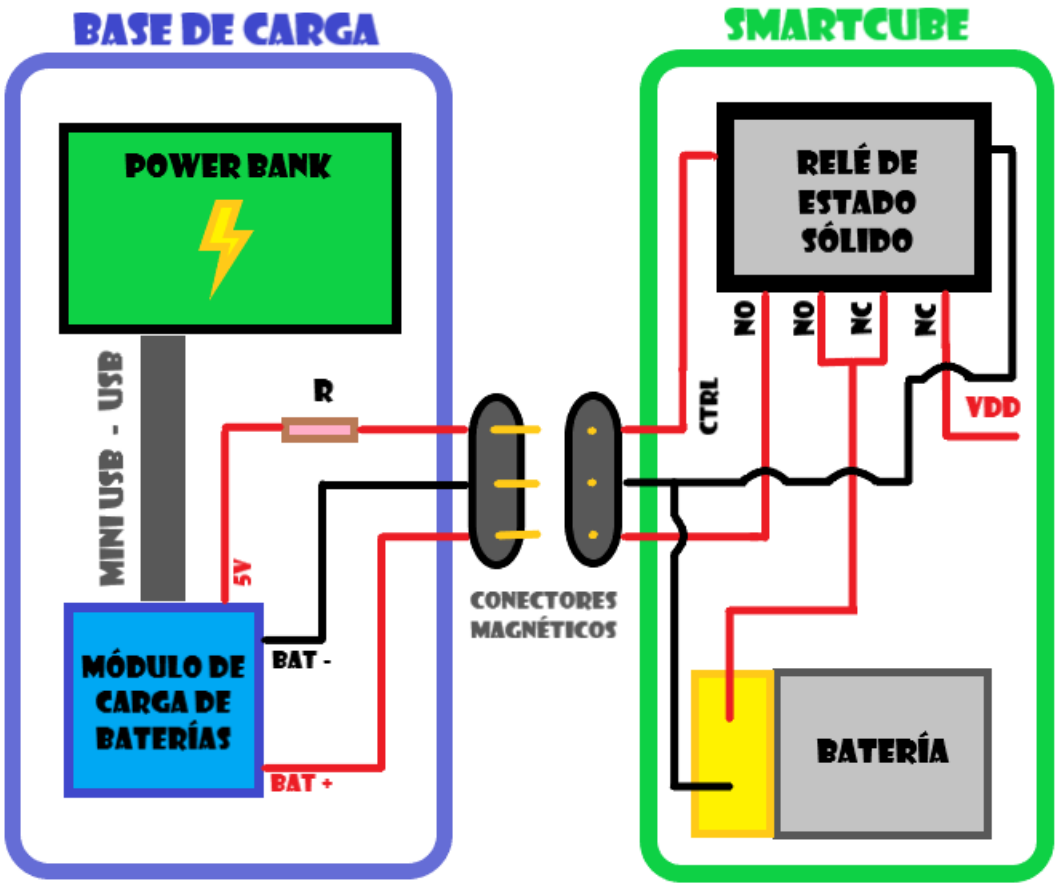


Figura 27. Diagrama del Sistema de Carga de Baterías

En la base de carga, se encuentra una *power bank* encargada de suministrar la energía necesaria para recargar las baterías durante largos periodos de tiempo, dicha recarga la realiza el módulo de carga que está conectado a la *power bank* a través de un cable USB-Mini USB (la *power bank* podrá conectarse a su vez a un cargador a través de otro cable USB-Mini USB para recargarla cuando fuese necesario).

El módulo de carga recargará la batería hasta un voltaje de 4.2 V, a través de los pines Bat+ y Bat- a una corriente de carga constante de 0.37C, en este caso puesto que la batería es de 250mAh, la corriente de carga será de 95 mA aproximadamente. Por último, se utilizarán los 5 V que proporciona la *power bank* para crear un terminal de control del relé de estado sólido, como se explicará más adelante. Finalmente, los tres terminales salientes del módulo de carga se sueldan al conector magnético que comunicará la base de carga con el SmartCube.

Una vez explicado el sistema de la base de carga, hay que pasar a ver qué es lo que sucede dentro del SmartCube, para que se realice correctamente la carga de la batería una vez depositemos el cubo en la plataforma.

A continuación, se puede ver en la Figura 28 el conexionado del relé de estado sólido.

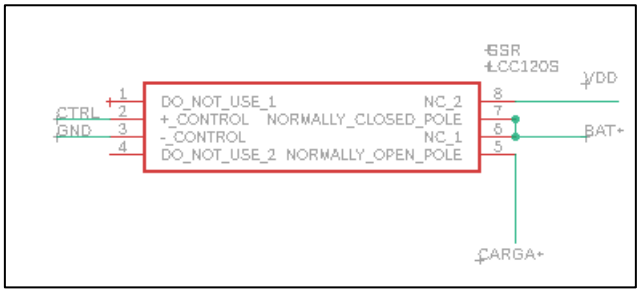


Figura 28. Esquemático del Conexionado del Relé de Estado Sólido

El relé de estado sólido cuenta con dos terminales normalmente cerrado y dos terminales normalmente abiertos, a uno de los NC está conectado el terminal positivo de la batería y el otro alimentará todo el sistema tras pasar por el sistema de protección por la descarga de la batería y por el regulador de tensión; por otro lado, uno de los terminales NO está conectado al terminal positivo de la batería y el otro al pin Bat+ del módulo de carga. De esta forma, cuando el relé está en estado de no conmutación, la batería se encontrará alimentando el SmartCube y cuando conmute pasará a cargar la batería, garantizando que en ningún momento esté realizando ambas tareas al mismo tiempo, suceso que podría acortar la vida útil de las baterías.

El relé de estado sólido conmuta cuando se hace pasar una pequeña corriente a través de sus terminales de control, lo que sucede es que esta corriente enciendo un diodo LED que a su vez activa un diodo fotosensible y la corriente de este diodo fotosensible activa un transistor mosfet que es el que conmuta de estado.

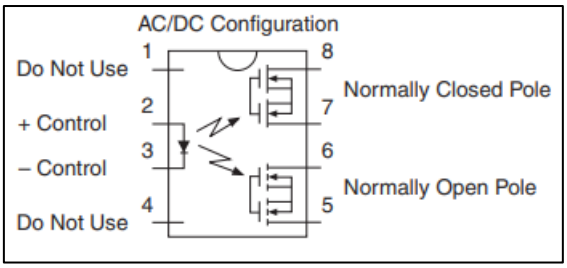


Figura 29. Configuración de Pines LCC120S. Fuente: Datasheet LCC120S [21]

La corriente mínima a la cual el relé no es capaz de conmutar, es de 0.4 mA siendo su valor típico 0.7 mA y cuyo valor máximo de tensión recomendado para conmutar el relé es 10 mA, por otro lado, la caída de tensión en el Led puede tomar valores entre 0.9 y 1.4 V, siendo su valor típico 1.2 V [21].

Para generar la corriente de activación del Led, se utiliza una resistencia en serie situada en la base de carga, como se puede ver en la Figura 27. A continuación, se calculará el rango de posibles valores de la resistencia.

$$R\,\text{min} = \frac{5V - Vd\ max}{If\ max} = \frac{5V - 1.4V}{10\ mA} = 360\ \Omega$$

$$R\,\text{max} = \frac{5V - Vd\ min}{If\ min} = \frac{5V - 0.9V}{0.7\ mA} = 5.9\ K\Omega$$

Se escogerá la resistencia más grande posible dentro del rango de valores para minimizar los consumos, aumentando así el tiempo de descarga de la *power bank*.

Pruebas de Funcionamiento

Para realizar esta prueba de funcionamiento, se realizarán los siguientes montajes de circuitos, por un lado, el propio circuito del sistema de carga, conformado por el relé, la batería, un módulo de carga conectado directamente a un cargador y los conectores magnéticos, como se puede observar en la Figura 30.

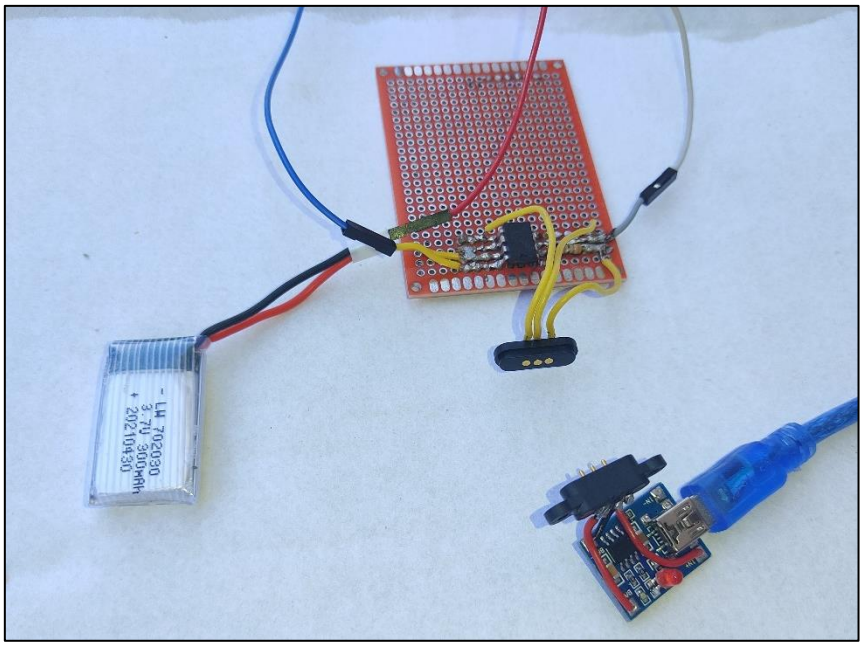


Figura 30. Montaje circuito de carga (Prueba de Funcionamiento Sistema de Carga de Batería)

Por otro lado, para comprobar el correcto funcionamiento del sistema de carga, emplearemos una placa de desarrollo casera basada en el ESP-WROOM-32 (), esta placa estará ejecutando un programa sencillo (parpadeo de un Led) y en el momento que se conecten los conectores magnéticos, el sistema dejará de estar alimentado y la batería comenzará a cargarse.

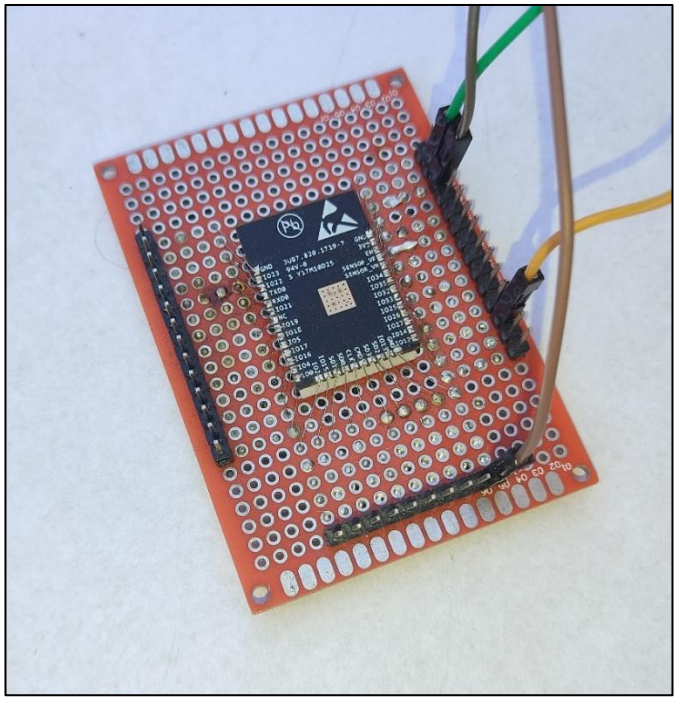


Figura 31. Placa de desarrollo casera basada en el ESP-WROOM-32

A continuación, se puede observar en la el sistema completo donde el diodo Led se encuentra parpadeando, a los elementos previamente mencionado hay que añadir el regulador de 3.3V que se incorpora para alimentar el ESP-WROOM-32 dentro del rango de tensiones recomendado.

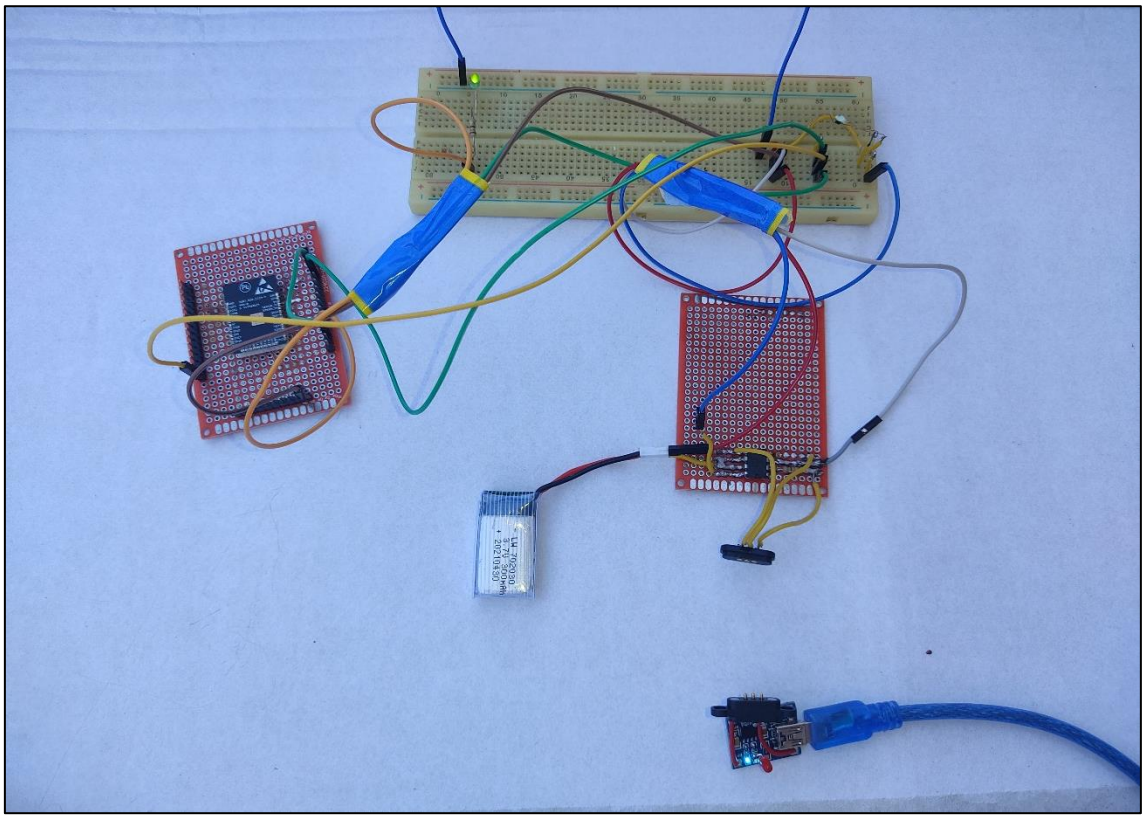


Figura 32. Sistema Completo Ejecutando Código (Prueba de Funcionamiento Sistema de Carga de Batería)

Una vez se conectan los conectores, el sistema el sistema deja de funcionar (el diodo Led deja de parpadear) y la batería comienza a cargarse (diodo Led rojo del módulo de carga se enciende), se puede ver en la Figura 33.

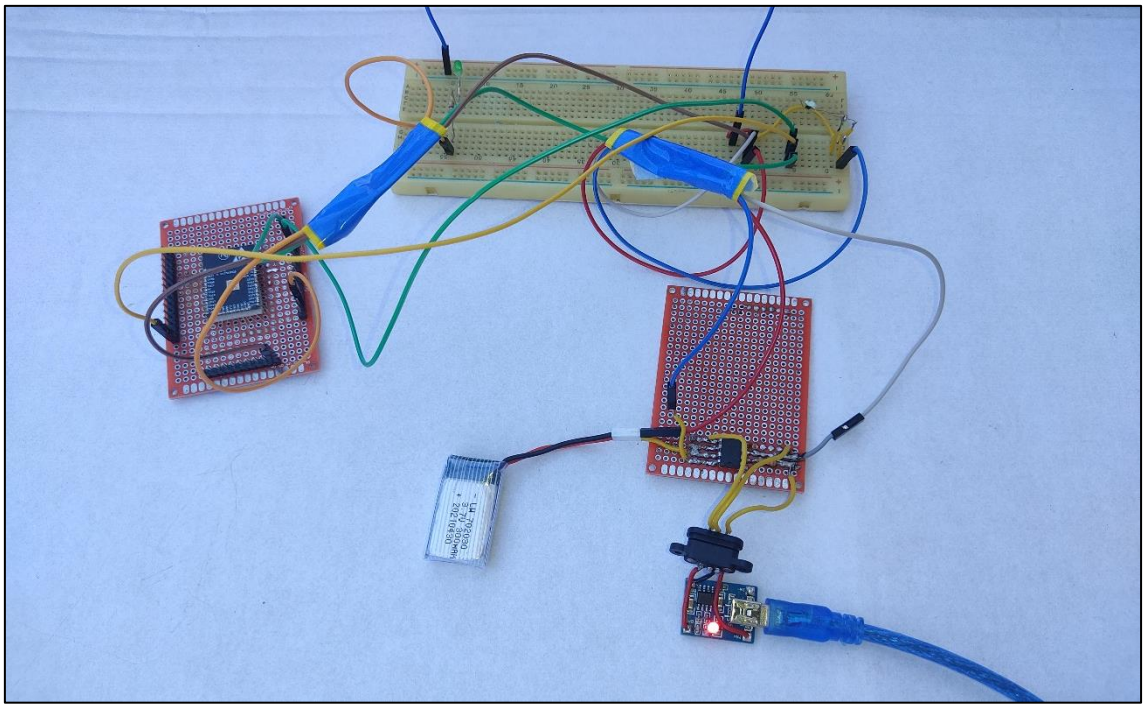


Figura 33. Sistema Completo Cargando Batería (Prueba de Funcionamiento Sistema de Carga de Batería)

6.06 Sistema de Protección por la Descarga de la Batería

Anteriormente ya se ha visto los componentes que integran el circuito de protección y su funcionamiento básico, a continuación, se verá el montaje completo de este circuito y sus valores concretos de funcionamiento [32], [33]. En la Figura 34 se puede a apreciar el montaje completo del circuito.

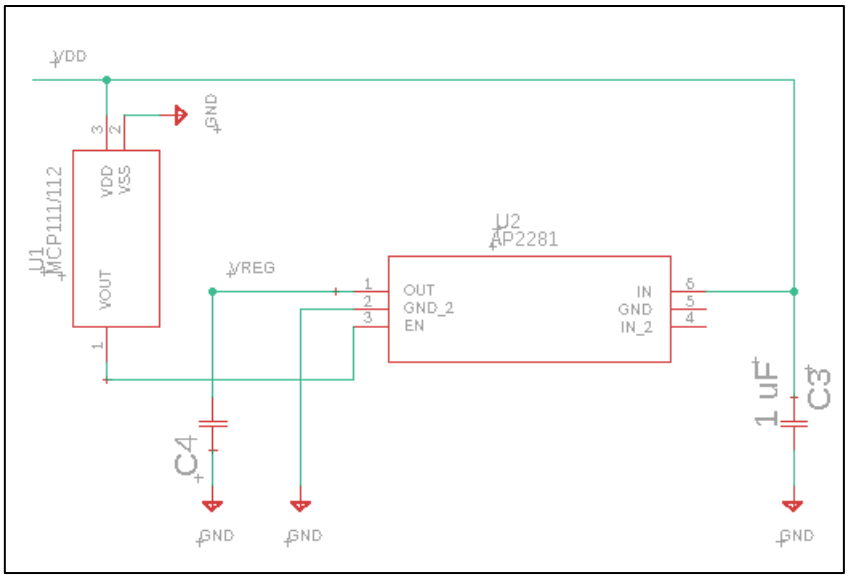


Figura 34. Esquemático del Circuito de protección por la Descarga de la Batería

La tensión VDD procedente de la batería entra por el pin VDD del detector de tensión, si esta tensión se encuentra por encima del valor umbral, que suele tener un valor típico de 3.080 V pudiendo variar entre 3.034 y 3.126 V, el detector de tensión generará un nivel alto por el pin Vout, el valor del nivel alto será como mínimo VDD-0.6V. Por el contrario, si la tensión se encontrara por debajo del umbral, Vout se pondrá a nivel bajo, con un valor máximo de 0.4V.

El valor de salida de Vout será el valor que tome el pin Enable del interruptor de alimentación, siendo el voltaje máximo para el que el pin Enable reconoce un nivel bajo de 0.4V y el valor mínimo para un nivel alto de 1.6 V (para 2.7V < Vin < 5.25V), por lo que no habría problemas de compatibilidad entre ambos módulos. Si el pin Enable se encuentra a nivel alto, el interruptor deja pasar por el pin Vout, la tensión de las baterías que entra a través del pin Vin, de lo contrario el interruptor corta la alimentación, de esta forma, se garantiza que las baterías no se descargarán por debajo de los 3V, protegiéndolas de daños irreversibles.

Por último, se colocan dos condensadores a la entrada y salida del interruptor de alimentación, un condensador de 1µF a la entrada para filtras posibles ruidos en la tensión de las baterías, y otro de 0.1µF a la salida para estabilizar la señal de alimentación y mejorar el transitorio de la carga.

La única diferencia con respecto al diseño de las anteriores versiones es que, la salida del AP2281-3WG-7 no alimenta directamente al resto del sistema, si no que pasa previamente por el regulador de tensión de 3.3V.

Con este circuito no se considera necesario realizar ninguna prueba previa ya que se ha verificado su correcto funcionamiento en anteriores versiones y la presencia del nuevo microcontrolador no interviene en la lógica del circuito.

6.07 Sistema de Medición del Estado de la Batería

Puesto que el sistema es alimentado con 3.3 V que proporciona el regulador de tensión, no es posible medir el voltaje de la batería conectando directamente a uno de los pines analógicos, ya que la tensión de la batería será siempre superior a 3.3V y estará fuera del rango de conversión del ADC. Será necesario reducir la tensión de la batería a la hora de tomar la medida, para ello, se empleará un divisor resistivo que reduzca la tensión 1/3 de su valor actual.

Para evitar que el divisor resistivo genere un consumo extra en nuestro sistema, se seleccionaran resistores de grandes valores, en torno a los mega ohmios.

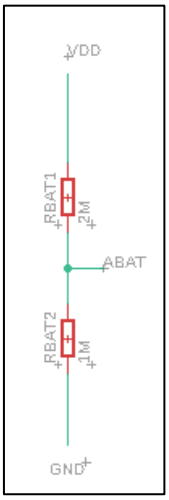


Figura 35. Esquemático Montaje del Divisor Resistivo de la Batería

Empleando resistencias 1 y 2 mega ohmios, la corriente que consume el divisor resistivo estará comprendida entre los 1.4 y 1.13 µA, corriente tan baja que prácticamente no afecta a la descarga de la batería.

Pruebas de Funcionamiento

En la prueba que se va a realizar, se medirá la tensión de una batería de Li-Po por medio del sistema previamente descrito y a su vez, empleando un multímetro, de esta forma se compararán ambos resultados y se dictaminará la validez o no de la prueba.

El divisor resistivo que se empleará en el montaje establecerá una relación de ½ entre el voltaje de la batería y de medición, ya que se utilizarán 2 resistencias de 100 KΩ debido a que no se disponía de resistencias 1y 2 MΩ.

En cuento al programa empleado para la medición de la batería, éste es casi idéntico al de la medición de la tensión en las galgas. Para no repetir el programa por completo y probar más funciones del ESP-WROOM-32, en este caso realizaremos la medición de la tensión empleando el ADC2. Puede verse el programa completo en el Anexo V.

En la Figura 36 se muestra el montaje del circuito de medición de la tensión de la batería.

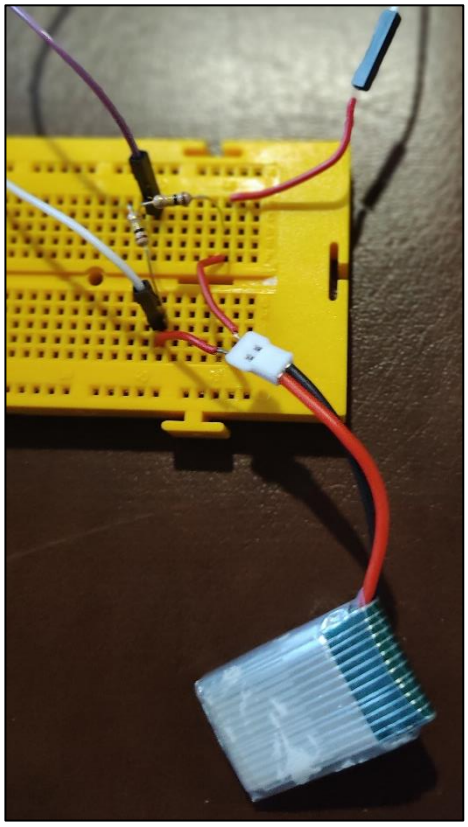


Figura 36. Montaje del circuito de medición de la tensión de batería

Al ejecutar el programa y comenzar a tomar medidas de tensión, el voltaje obtenido por el monitor serie es de aproximadamente 3.85 V, como puede verse en la Figura 37.

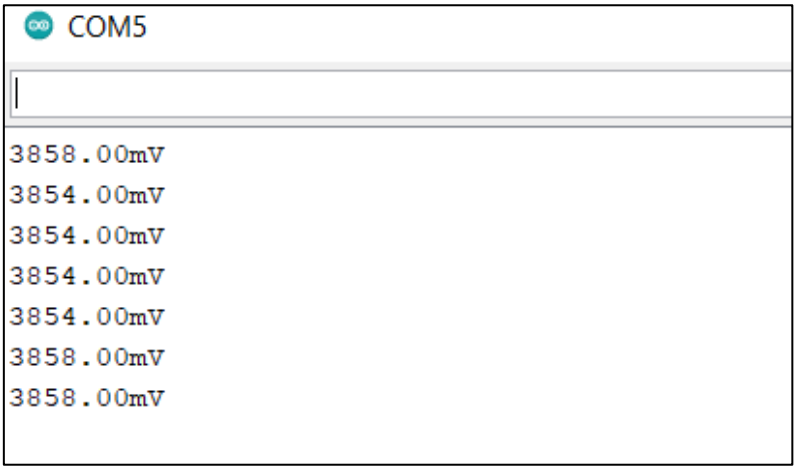


Figura 37. Medición de la tensión de la batería empleando el ESP-WROOM-32

En la Figura 38 puede verse la medición de la tensión de la batería empleando un multímetro, la diferencia entre las medidas al emplear ambos instrumentos, es de 0.04 V aproximadamente. Teniendo en cuenta que la labor de este sistema es la de indicador de la descarga de la batería y no se requiere de excesiva precisión para ello, se acepta como válido el sistema de medición.

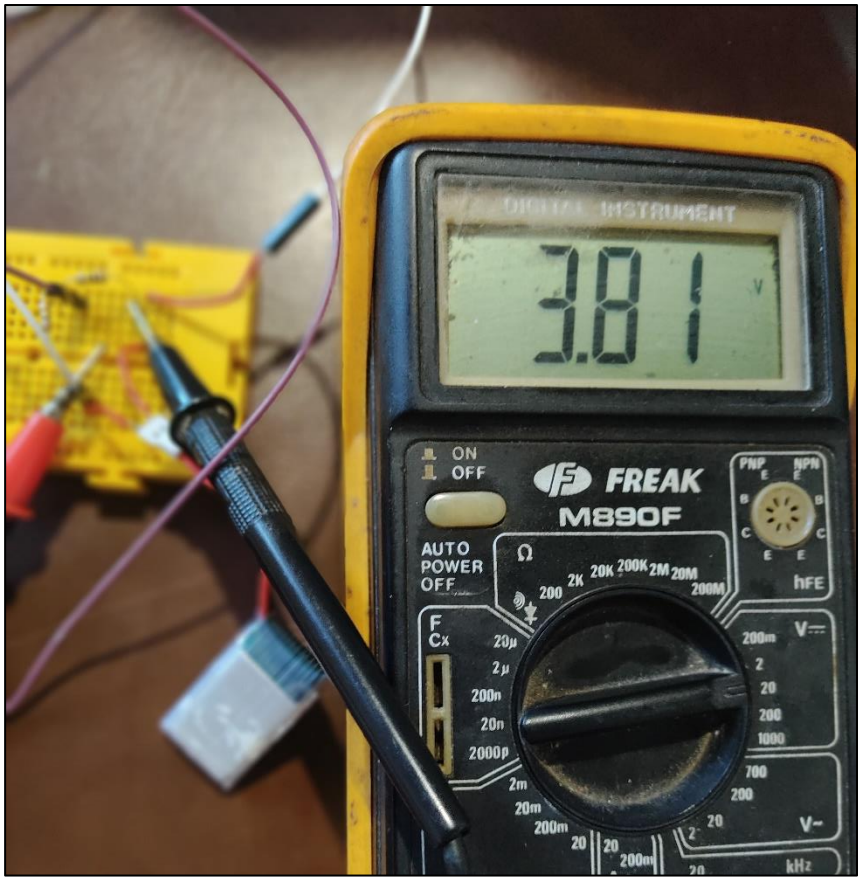


Figura 38. Medición de la tensión de la batería empleando el multímetro

6.08 Sistema de Programación

Como se ha visto anteriormente, esta nueva versión de los SmartCubes cuenta con dos métodos de programación, además de cargar el programa por medio de conexión directa con el microcontrolador, con la implementación del ESP-WROOM-32 surge la posibilidad de programar el microcontrolador de forma inalámbrica empleando tecnología Wifi.

En este apartado se verán ambos métodos de programación y se realizarán las pruebas correspondientes que verifiquen su correcto funcionamiento.

6.08.1 Programación con Convertidor Serie-USB

El elemento central de este método de programación es el convertidor Serie-USB, el cual establecerá un puente de unión entre la interfaz USB del ordenador y la interfaz Serie (UART), que permitirá programar el microcontrolador directamente. A continuación, en la Figura 39 se puede ver un diagrama ilustrativo del sistema de programación.

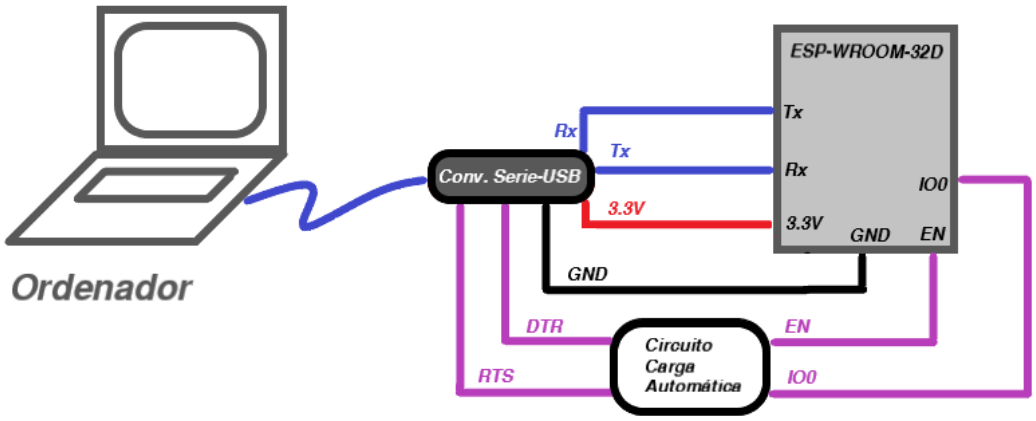


Figura 39. Diagrama Ilustrativo de la Programación con el Convertidor Serie-USB

En color azul se pueden ver los terminales de comunicación UART (Tx y Rx) se encargarán se transmitir el código del programa, en rojo y negro las conexiones de alimentación y en morado las conexiones correspondientes a los terminales de control de la carga automática del programa; para que la programación se realice correctamente hay que tener en cuenta ciertas consideraciones [34].

- Carga del Programa. Para que se posible subir el programa, durante dicho proceso, el microcontrolador debe estar alimentado entre 3.3V y masa, el módulo debe estar habilitado a través de un nivel alto en el pin *Enable* y el GPIO0 debe

permanecer a nivel bajo, este GPIO es el encargado de poner el microcontrolador en modo programación. En estas condiciones, una vez compilado el programa, puede cargarse en el microcontrolador.

- *Hard Resseting* tras la Carga. Una vez subido el programa en el microcontrolador, será necesario realizar un restablecimiento completo del microcontrolador, para ello se debe deshabilitar el microcontrolador (EN a nivel bajo) y antes de volver a habilitarlo, se debe salir del modo programación (GPIO0 a nivel alto). Una vez realizado este proceso, el microcontrolador debe comenzar a ejecutar el programa cargado.

Para evitar tener que realizar de forma manual este procedimiento, incorporaremos un circuito encargado de automatizar el proceso de programación, en la Figura 39 puede observarse integrado en el sistema completo y en la Figura 5 se muestra el circuito en detalle.

Se emplearán las señales del puerto serie, DTR (*Data Terminal Ready*) y RTS (*Request To Send*), junto con dos transistores bipolares SS8050-G, para generar las señales de *Enable* y GPIO0 que controlan el proceso de programación.

A continuación, en la Figura 40 puede observarse un cronograma con las señales DTR, RTS, *Enable* y GPIO0 durante el proceso de programación.

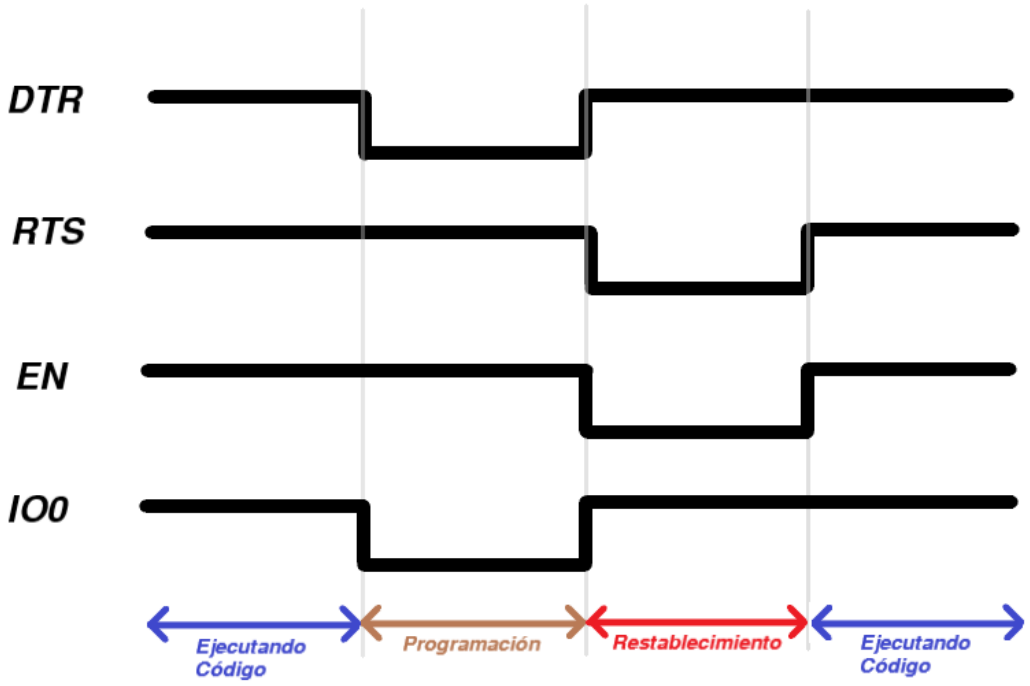


Figura 40. Cronograma con las señales DTR, RTS, EN y IO0 durante el proceso de programación

Es un cronograma orientativo del funcionamiento del circuito de automatización, ya que no se han tenido en cuenta los tiempos de establecimiento, ni los instantes concretos de activación de las señales.

Para mantener las señales a *Enable* y IO0 a nivel alto una vez se desconecte el convertidor Serie-USB, implementamos el circuito que se puede ver a la derecha en la Figura 5, si no se implementara este circuito, una vez programado el microcontrolador, permanecería inhabilitado.

Prueba de Funcionamiento

La comprobar el correcto funcionamiento de este método de programación, se realizará una pequeña prueba donde se subirá a un ESP-WROOM-32, un pequeño programa que iniciará el parpadeo de un diodo Led.

A continuación, se realiza el montaje previamente explicado, en la Figura 41 puede verse una placa casera para realizar pruebas con el ESP-WROOM-32 (abajo), el circuito encargado de la automatización de la programación (izquierda), el circuito de alimentación y carga de la batería (arriba y derecha), y el circuito del Led (arriba).

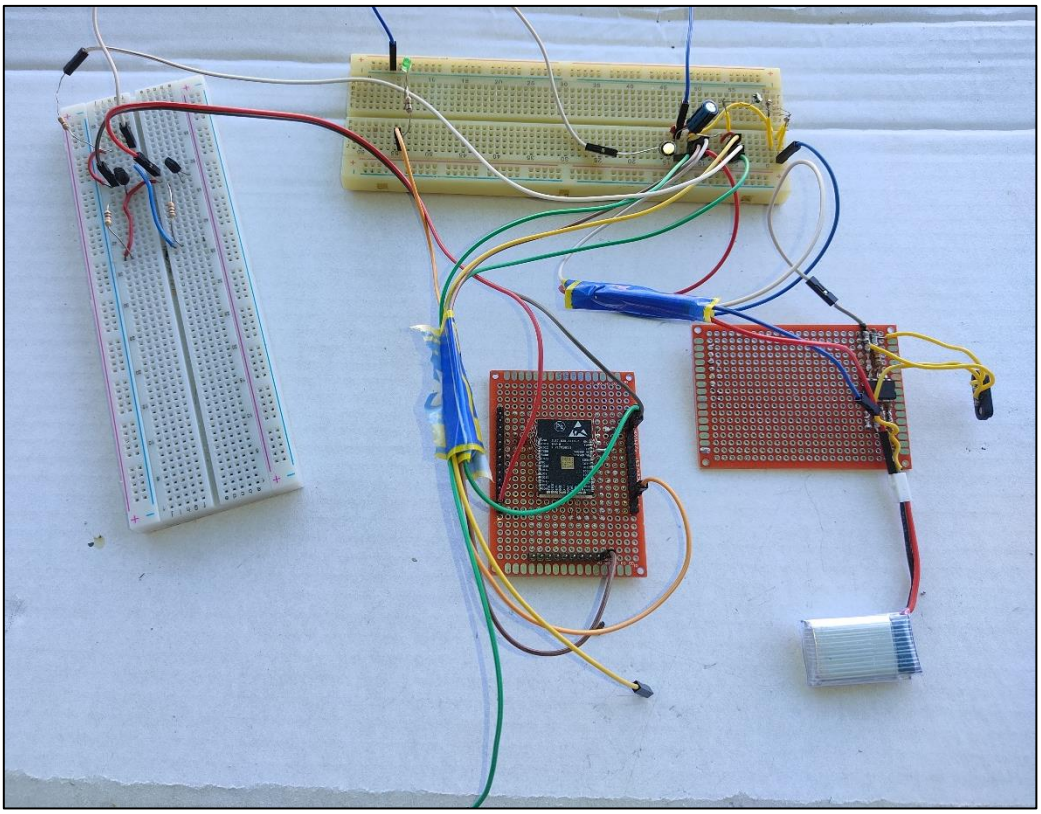


Figura 41. Montaje Carga de Programa vía USB-Serie (Prueba de Programación con Convertidor USB-Serie)

Se selecciona el puerto COM correspondiente y se carga el programa, en la Figura 42 se puede ver el proceso de carga del programa en el IDE de Arduino, tras el cual comienza a parpadear el diodo Led, como puede observarse en la Figura 43 y en la Figura 44 una vez desconectado el convertidor USB-Serie.

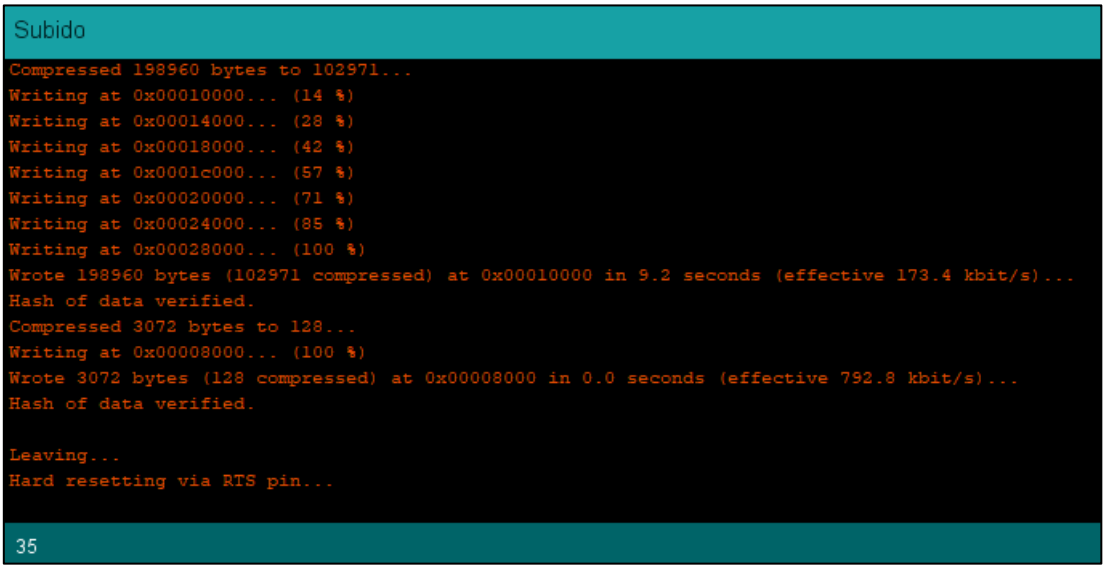


Figura 42. Carga del Programa (Prueba de Programación con Convertidor USB-Serie)

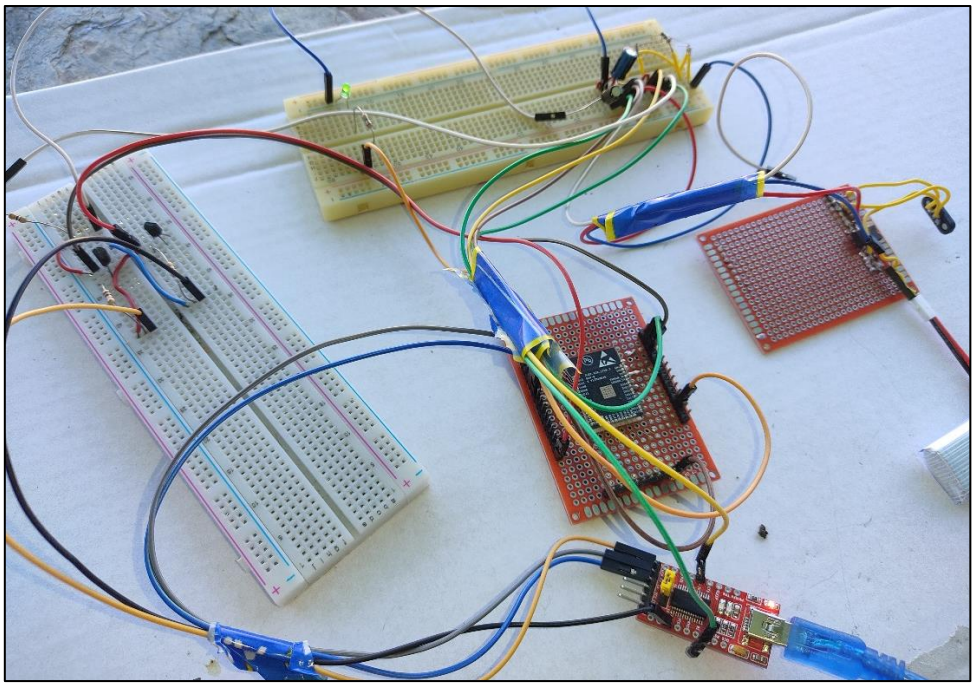


Figura 43. Programa Cargado en el microcontrolador con el Convertidor USB-Serie conectado (Prueba Programación con Convertidor USB-Serie)

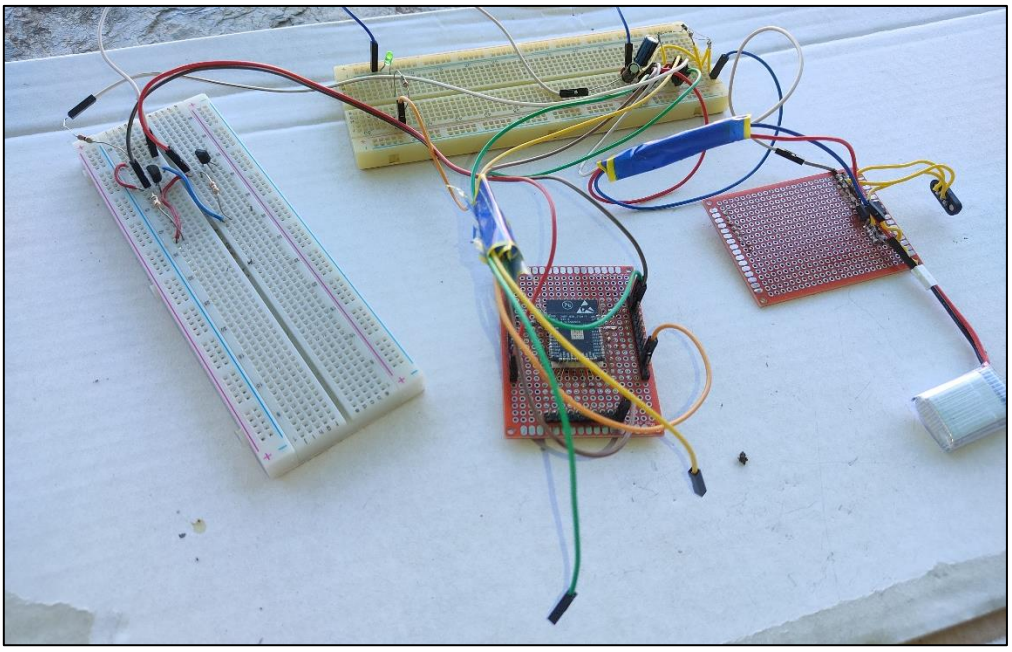


Figura 44. Programa Cargador en el microcontrolador con Convertidor USB-Serie desconectado (Prueba Programación con Convertidor USB-Serie)

6.08.2 Programación OTA (*Over The Air*)

La programación OTA consiste en una actualización del firmware del microcontrolador en cuestión a través de una conexión Wifi, sin necesidad de emplear un puerto serie. Este tipo de programación es muy útil cuando el microcontrolador no se encuentra fácilmente

accesible, es el caso de este proyecto ya que para reprogramar los SmartCubes es necesario desmontar previamente la carcasa de cada uno de ellos.

La programación OTA del ESP-WROOM-32, puede realizarse en diversos entornos de desarrollo como el IDE de Arduino, de PlatformIO, MicroPython, Visual Studio, etc. En este caso nos centraremos en la programación OTA en el IDE de Arduino [35].

Para utilizar este tipo de programación es necesario tener en cuenta los siguientes puntos:

- IDE de Arduino v1.6.7 o superior. Será necesario tener esta versión o superior ya que, de lo contrario, es muy posible que la librería Arduino OTA no esté soportada por otras versiones.
- Instalar Python v2.7. Instalar esta versión de Python es necesario ya que, el pluggin Arduino IDE para el ESP32 se apoya en Python durante la programación OTA. Otras versiones superiores como la 3.5 no funcionarían.
- Doble de la memoria que ocupa el programa disponible. Esto es un protocolo de seguridad ya que, durando el proceso de carga del programa, este se almacena en la memoria temporalmente y se traslada a la memoria Flash una vez terminada. Este punto no debería representar un problema ya que, por defecto la memoria se encuentra dividida en particiones donde ya se tiene en cuenta este aspecto, por lo que no nos permitiría desarrollar un programa que supere la limitación de almacenamiento.
- Primera programación con puerto USB. La Programación OTA no puede realizarse en la primera carga del programa ya que, para que esta sea realizable, el actual firmware cargado en el microcontrolador debe contener la parte de código dedicada a que la programación OTA sea posible. Por ello, la primera programación de los SmartCubes deberá ser empleando el convertidor Serie-USB.
- Mantener la programación OTA en futuros códigos. De la misma forma en que no es posible realizar la primera carga del programa a través de OTA porque no contamos con el código correspondiente cargado previamente en el microcontrolador, para que en un futuro se pueda seguir empleando la programación OTA, cuando se carga un programa a través de Wifi será necesario que el código cargado conserve la parte de la programación OTA.

A continuación, se verá el código necesario para realizar la programación OTA con la librería Arduino OTA en el IDE de Arduino.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
```

Lo primero que se puede observar son las 4 librerías que utiliza la programación OTA, las cuales se corresponden respectivamente con el manejo del chip Wifi, los DNS, los UDP y la funciones que emplea la programación OTA.

```
const char* ssid = ".........";
const char* password = ".........";
```

A continuación, se debe introducir la el nombre y clave de la Wifi a la que nos queremos conectar.

A partir de aquí y hasta nuevo aviso, el código que se muestre está dentro de la función *void setup ()*.

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.waitForConnectResult() != WL_CONNECTED) {
  Serial.println("Connection Failed! Rebooting...");
  delay(5000);
  ESP.restart();
}
```

Con las dos primeras sentencias se define el tipo de conexión y ordena establecer conexión respectivamente. Inmediatamente después se incluye un bucle *while*, cuya función es la de garantizar que se logra establecer conexión con la Wifi, mientras que el ESP-WROOM-32 no se conecte, se indicará la conexión fallida y se volverá a intentar hasta alcanzar su meta.

```
// Port defaults to 3232
// ArduinoOTA.setPort(3232);

// Hostname defaults to esp3232-[MAC]
// ArduinoOTA.setHostname("myesp32");

// No authentication by default
// ArduinoOTA.setPassword("admin");

// Password can be set with it's md5 value as well
// MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
// ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");
```

En letra comentada se indican una sería de valores predeterminados y las funciones con las que cambiarlos si fuese necesario, valores son el puerto (3232 por defecto), el nombre de servidor (esp3232-[Mac] por defecto) y la contraseña (que está deshabilitada por defecto).

```
ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
  });
```

Esta parte del código se corresponde con funciones *callbacks* de la programación OTA, es decir, cuando ocurre un determinado evento, en estas funciones se puede declarar la manera a proceder.

```
ArduinoOTA.begin();

Serial.println("Ready");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
```

Con las siguientes sentencias se inicializa la gestión OTA y mostramos por el terminal del puerto serie la dirección IP de la Wifi.

En este punto se cierra la función *void Setup ().*

```
ArduinoOTA.handle();
```

Dentro de la función *void loop ()* tenemos la siguiente función que se ocupa de gestionar las conexiones vía OTA durante la ejecución del programa, comprobando en cada ciclo si existe algún programador que quiera cargar un boceto.

Prueba de Funcionamiento

Por comodidad se utilizará un ESP32 devKitC v4 para la realización de la prueba, dicha prueba consistirá en carga un programa inicial al ESP32 vía USB, en el que estará incluido el código correspondiente a la programación OTA y al parpadeo de un diodo LED de color verde. Una vez cargado, se desconectará del ordenador y se alimentará el módulo a través de una batería de Li-Po, en este momento comenzará el parpadeo y se creará un puerto de comunicación vía Wifi. Puede verse el parpadeo del Led verde en la Figura 45.

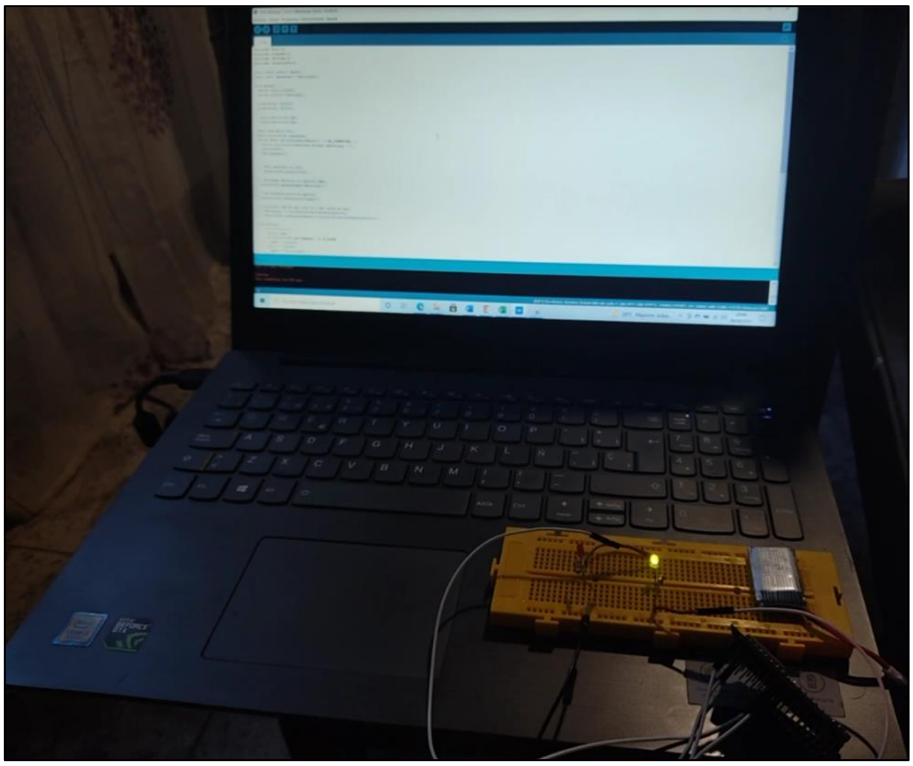


Figura 45. Programa inicial (Prueba Programación OTA)

A continuación, se modifica el programa para que ahora el que pardee sea un diodo Led de color rojo (Figura 47) y se sube el programa vía Wifi (Figura 46).

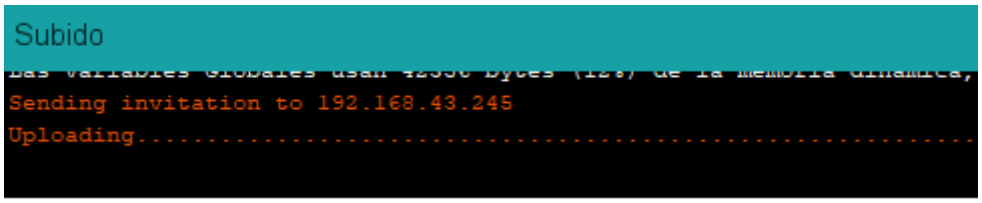


Figura 46. Carga Completa del programa vía OTA

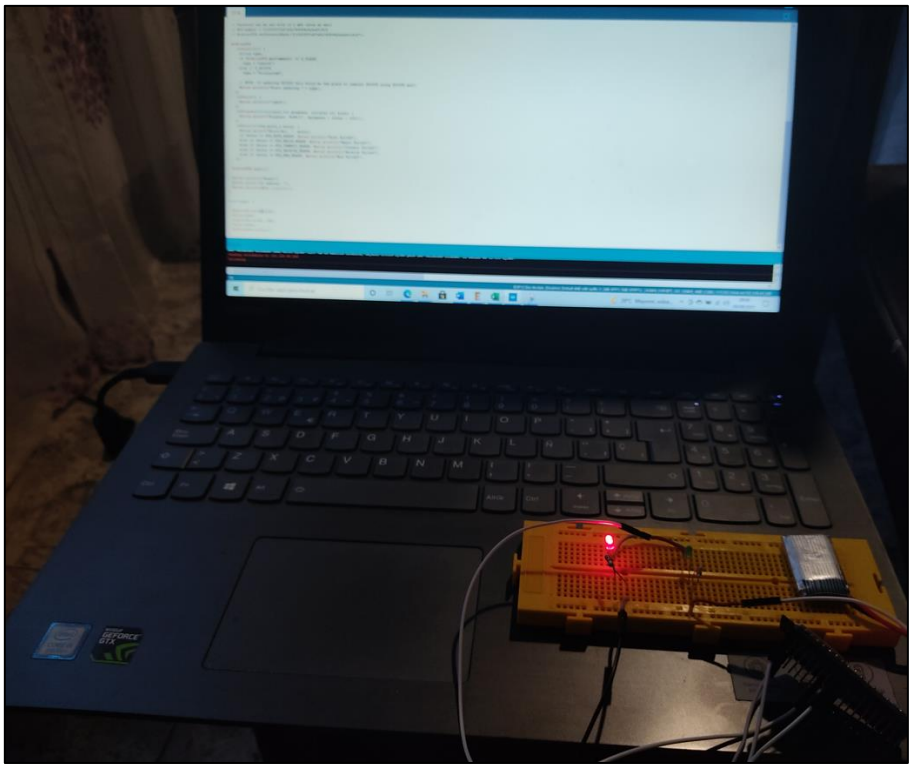


Figura 47. Programa tras la programación OTA (Prueba Programación OTA)

Puede verse le programa completo en el Anexo VI.

6.09 Transmisión de Datos (Bluetooth Low Energy)

Como se ha comentado anteriormente, la transmisión de datos se llevará a cabo a través de la tecnología BLE que incorpora el ESP-WROOM-32. La principal diferencia con el Bluetooth tradicional se encuentra en los consumos, donde el BLE consume en torno a 100 veces menos, esta gran diferencia entre consumos se debe en parte a uno de los cambios de funcionamiento que ha adquirido el BLE con respecto a Bluetooth tradicional, y es que mientras que este último se encuentra activo en todo momento, el BLE se activa cuando establece una conexión. Además, en la tecnología BLE, estos tiempos de conexión toman únicamente unos pocos milisegundos, mientras que con el Bluetooth tradicional pueden superar el centenar.

Existen dos roles que pueden adoptar los dispositivos en la tecnología BLE, el servidor y el cliente.

- El servidor. Es el dispositivo que se anuncia al resto de dispositivos clientes y que una vez conectados, posee los datos que pueden leerse.
- El cliente. Es el dispositivo que escanea los posibles servidores con los que establecer conexión y recibe los datos que estos le proporcionan.

Así mismo, la tecnología BLE cuenta con dos modos que pueden ser útiles en determinadas situaciones, son el modo difusión y el modo red de malla.

- Difusión. Permite a un único servidor, transmitir datos a varios clientes.
- Red de Malla. Permite a muchos dispositivos estar conectados todos con todos.

El funcionamiento de la tecnología BLE está fundamentada en un conjunto de protocolos, de los cuales solo se tratarán los dos que se encuentra en el nivel más alto de la estructura, los protocolos GAP y GATT.

El protocolo GAP (*Generic Acces Profile*) es el encargado de garantizar que un dispositivo sea visible por otros y define el modo en que estos interactúan entre sí (roles, modos de operación y transición, procedimiento para el establecimiento de la comunicación o los modos de seguridad). Una vez los dispositivos han superado la etapa de establecimiento de comunicación, de la cual se encarga el protocolo GAP, por medio del protocolo GATT (*Generic Attribute Profile*) se establece la forma en que los dispositivos intercambiarán información (pudiendo esta ser bidireccional).

A continuación, en la Figura 48 se puede ver la estructura jerárquica del protocolo GATT.

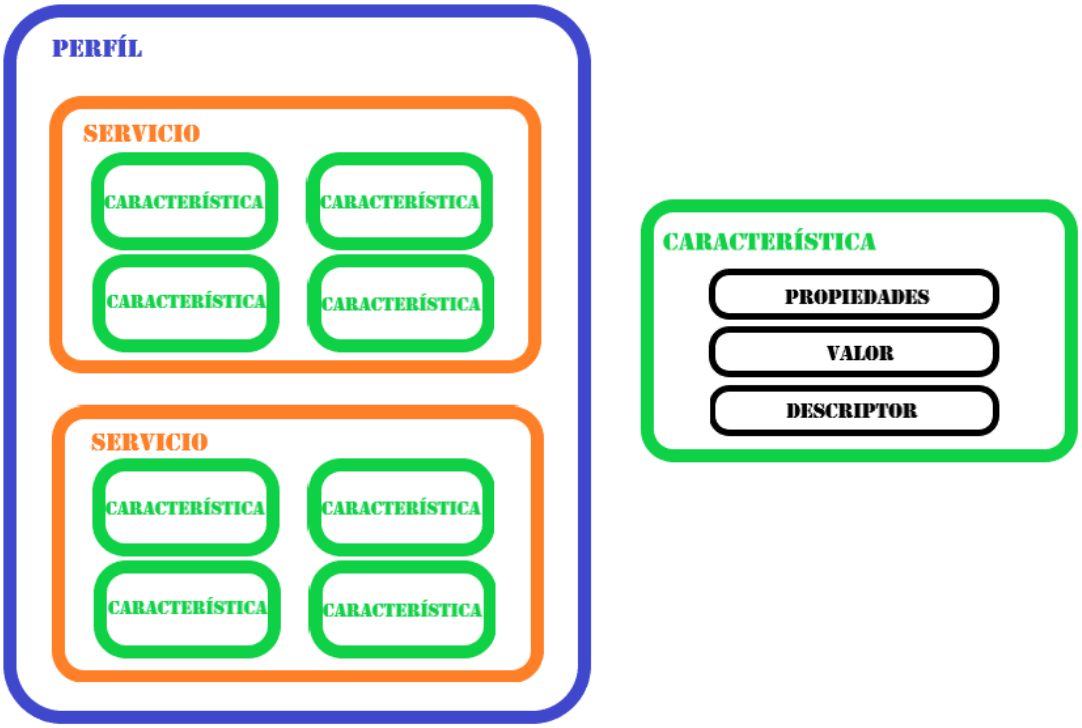


Figura 48. Estructura jerárquica protocolo GATT

El nivel más alto de la jerarquía es el perfil, el cual se puede componer de uno o más servicios, un servicio es únicamente una colección de información que hace referencia a una característica u incluso otro servicio, generalmente un perfil suele contener varios servicios, uno de los usos que se les darán a los servicios en este proyecto es como organizadores de la información de los diferentes sensores que posee el sistema. La característica es donde están localizados los datos que contienen los servicios, puede haber varias características en un único servicio y, en cada característica se pueden distinguir unas propiedades, un valor y un descriptor. En las propiedades se describe la forma en la que tiene que interactuar cada característica, como las operaciones que estas pueden realizar; en el valor se encuentra el valor real del dato a transmitir; y el descriptor únicamente aporta metadatos extra de la característica [36], [37].

Puesto que proyecto se desarrollará en el IDE de Arduino, es interesante mostrar la configuración del BLE del ESP-WROOM-32 en esta plataforma [38]. El dispositivo en este caso actuará de servidor ya que, es el rol que interpretará en el proyecto, además únicamente se tratará el procedimiento para crear un dispositivo con un perfil, un servicio y una característica.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
```

Lo primero debemos incluir son las librerías que utiliza el ESP-WROOM-32 para realizar la comunicación vía BLE.

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

Definimos los UUID de cada característica y servicio que se vaya a utilizar, la UUID es un número único de 128 bits que sirve como identificador, existen UUID predeterminados, pero siempre se pueden utilizar sitios web generadores de UUID para estos servicios y características [39].

El código que se mostrará a continuación, estará incluido en la función void setup().

```
BLEDevice::init("ESP32");
```

Con esta sentencia se crea el dispositivo BLE que emplearemos.

```
pServer = BLEDevice::createServer();
```

Creación del servidor.

```
BLEService *pService = pServer->createService(SERVICE_UUID);
```

Creación del servicio.

```
pCharacteristic = pService->createCharacteristic(
                    CHARACTERISTIC_UUID,
                    BLECharacteristic::PROPERTY_READ   |
                    BLECharacteristic::PROPERTY_WRITE  |
                    BLECharacteristic::PROPERTY_NOTIFY |
                    BLECharacteristic::PROPERTY_INDICATE
                  );
```

Se crea la característica del servicio y se definen las propiedades de esta.

```
pCharacteristic->addDescriptor(new BLE2902());
```

Se añade el descriptor de la característica, en este caso el descriptor 2902, permite al cliente activar las notificaciones cuando él quiera, de esta manera se evita el envío innecesario de datos.

```
pService->start();
```

Se inicializa el servicio.

```
pServer->getAdvertising()->start();
```

Comienza a anunciarse a otros dispositivos.

Se cierra la función *void setup().*

Aquí termina la configuración del BLE, ya en la parte del programa se especificarían las acciones deseadas.

Prueba de Funcionamiento

Se realizará una prueba de funcionamiento donde se tratará de simular una situación similar a la que se tendrá en la versión final de los SmartCubes. Se establecerán varios modos de funcionamiento donde se mostrará unos datos u otros o ambos.

Para ello, se definirán 3 servicios, uno para introducir el modo de funcionamiento deseado, y los otros dos para mostrar los datos, uno por cada servicio.

El servicio encargado de la selección del modo de funcionamiento, cuenta con una única característica con la propiedad de escribir el valor recibido del dispositivo al que se conecte.

Los servicios encargados del muestro de datos cuentan ambos con la misma configuración, una característica con la propiedad de notificar a los dispositivos que estén conectados, los valores de los datos.

En el modo 0, el programa no mostrará ningún dato, en el modo 1 mostrará los valores del Dato1, en el modo 2 mostrará los valores del Dato2 y en el modo 3 se mostrarán los valores de ambos datos.

Puede consultarse el código del programa en el Anexo VII.

Para hacer la prueba utilizaremos la aplicación BLE Scanner para conectarnos al ESP-WROOM-32 con un teléfono móvil inteligente.

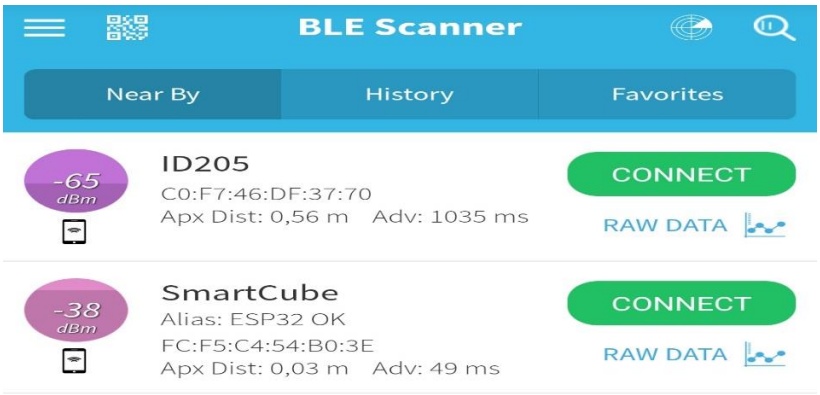


Figura 50. Escaneo de dispositivos BLE con la aplicación BLE Scanner

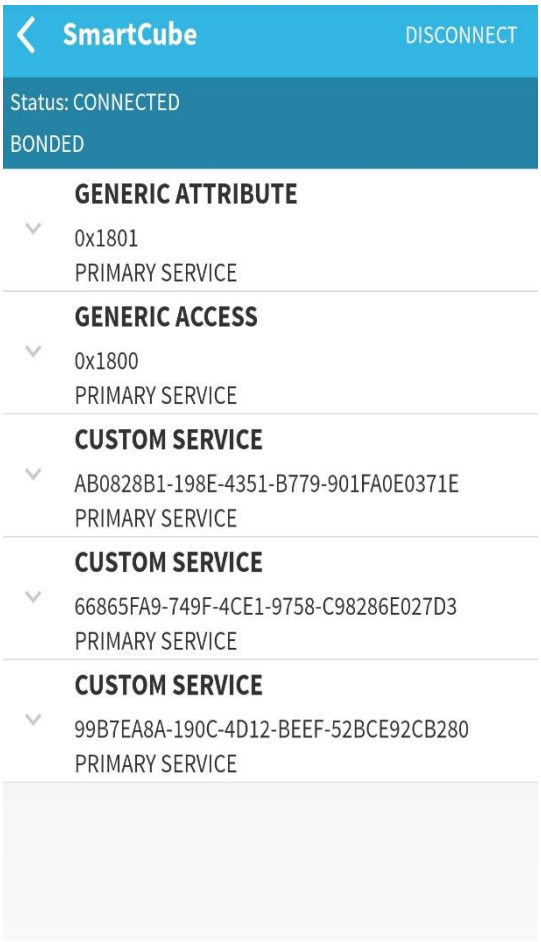


Figura 49.Pantalla de conexión con el dispositivo

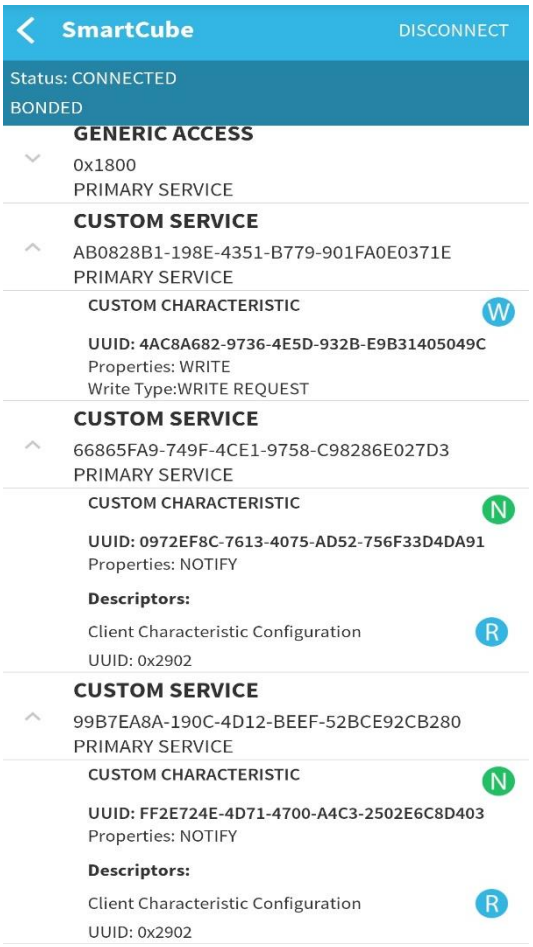


Figura 51. Se activan las notificaciones de los servicios

En la Figura 50 podemos apreciar como el ESP-WROOM-32 se anuncia al resto de dispositivos con el nombre de SmartCube, al conectarnos en la Figura 49, se pueden apreciar los 3 servicios de los que se ha hablado previamente. A continuación, en la Figura 51 se activan las notificaciones de los servicios que mostraran los datos, como podemos

ver no se muestra nada hasta que le introduzcamos un modo de funcionamiento con el primer servicio.

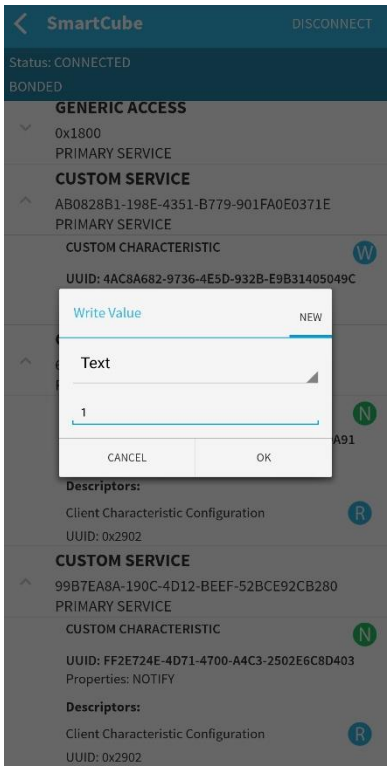


Figura 52. Se introduce el Modo 1 (Prueba BLE)

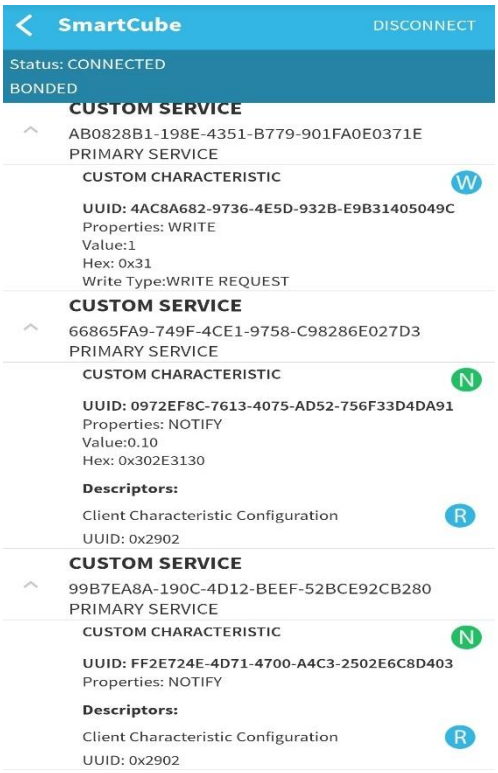


Figura 53. Modo 1 activo (Prueba BLE)

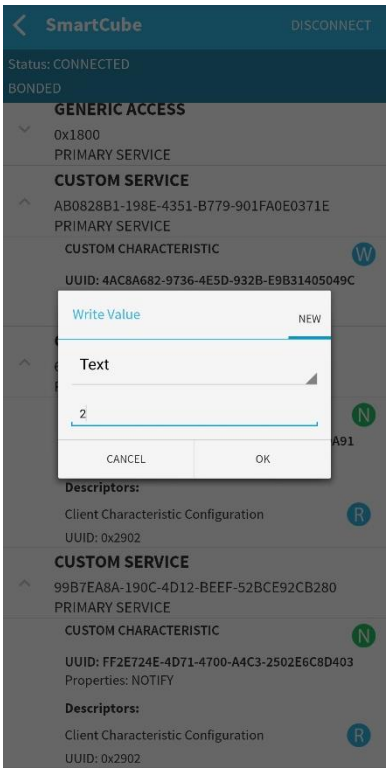


Figura 55. Se introduce el Modo 2 (Prueba BLE)

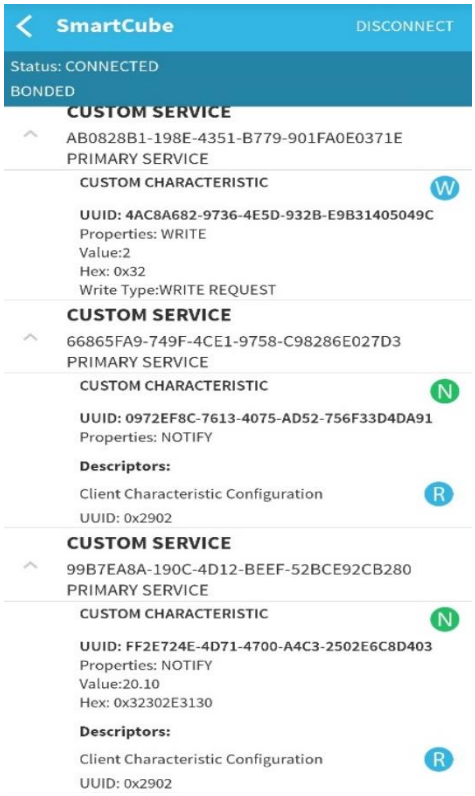


Figura 54. Modo 2 activo (Prueba BLE)

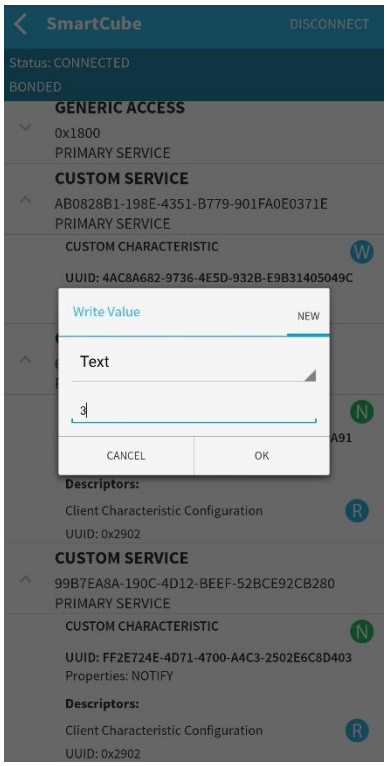


Figura 56. Se introduce el Modo 3 (Prueba BLE)

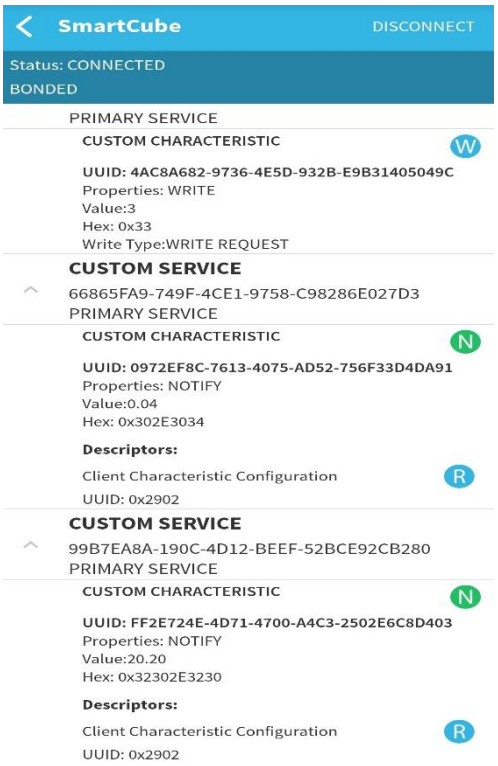


Figura 57. Modo 3 activo (Prueba BLE)

En la Figura 52 se introduce el modo de funcionamiento 1, en el que el segundo servicio comenzará a mostrar datos, como se puede ver en la Figura 53, los datos comienzan con un valor de 0 y cada segundo aumenta su valor en 0.01. Al introducir el modo 2 en la Figura 55, en la Figura 54 se puede ver como el servicio 3 comienza a mostrar datos, en este caso el valor inicial es 20 y cada segundo aumenta en 0.05. Por último, en la Figura 56 se introduce el modo 3, donde ambos servicios muestran datos, puede verse en la Figura 57.

6.10 Modos de Ahorro y Sistema de detección capacitiva

Los módulos ESP32 destacan por su alto rendimiento en términos de autonomía en comparación con otros módulos wifi, no obstante, si se mirara únicamente el consumo de éste en una aplicación que tenga activos el wifi y bluetooth, no se llegaría a esa conclusión. La razón por la que se dice que estos módulos presentan autonomías muy elevadas, es por los modos de suspensión que poseen.

Estos módulos cuentan con los siguientes modos de funcionamiento ordenados de mayor a menor consumo: Activo, *Modem Sleep, Light Sleep, Deep Sleep y Hibernation* [40], [41], [42], [43], [44].

La forma en que reducen el consumo estos modos de funcionamiento es inhabilitando algunos de los elementos que integran el chip o modificando su rendimiento.

A continuación, en la Tabla 4 podemos observar los elementos activos y el consumo medio en cada modo de funcionamiento.

	ULP Coprocessor	RTC	ESP32 Core & Memories	Wifi	Bluetooth	Radio	Peripherals	Comsuption
Active	✓	✓	✓	✓	✓	✓	✓	160-260 mA
Modem Sleep	✓	✓	✓	✗	✗	✗	✓	3-20 mA
Light Sleep	✓	✓	‖	✗	✗	✗	✗	0.8 mA
Deep Sleep	✓	✓	✗	✗	✗	✗	✗	10 µA
Hibernation	✗	✓	✗	✗	✗	✗	✗	2.5 µA

Tabla 4. Modos de funcionamiento ESP32

- Modo Activo. Es el más ineficiente de todos, especialmente cuando se utiliza Wifi y bluetooth al mismo tiempo, donde su consumo medio oscila entre los 160- 260 mA, pudiendo darse picos puntuales con valores más altos.
- *Modem Sleep*. Este primer modo de suspensión, no es un modo predefinido que pueda ser activado por medio de una función, si no que consiste principalmente en la inhabilitación de todos los circuitos relacionados con el funcionamiento de la radio, bluetooth y Wifi, también es posible reducir la velocidad de CPU, obteniendo un consumo variable entre los 3-20 mA. En este estado de funcionamiento el sistema puede continuar comunicándose con el exterior a través de los periféricos.
- *Light Sleep*. Con este modo de suspensión deshabilitamos los mismos elementos que con el modo *Modem*, pero, además, permanecen inutilizables los periféricos

digitales, la CPU y gran parte de la memoria RAM. Durante este modo de funcionamiento el consumo es de 0.8 mA y el sistema permanece suspendido, al salir del modo *Light Sleep* el sistema retoma su funcionamiento desde el punto previo a entrar en él. Este modo puede activarse empleando la siguiente función `esp_light_sleep_start()` y establecer el tiempo que tardará en despertarse con `esp_sleep_enable_timer_wakeup(SLEEP_DURATION).`

- *Deep Sleep*. En este modo de suspensión a diferencia del *Light Sleep* que solo pausábamos la CPU, con el modo Deep Sleep la apagamos, consiguiendo un consumo aproximado de 10 µA. Durante este modo, los únicos elementos que permanecen activos son el controlador, los periféricos y las memorias del RTC (*Real Time Controler*) y el coprocesador UPL (*Ultra Low Power Processor*). Para entrar en modo *Deep Sleep* se utiliza la siguiente función `esp_deep_sleep_start()`, sin embargo, existe más de una forma de despertar al sistema del modo *Deep Sleep*.
  - Temporizador. Es posible definir un temporizador, que despierte al dispositivo después de un determinado tiempo, empleando el controlador RTC. `esp_sleep_enable_timer_wakeup(time_in_us)`
  - *Touch Pad*. Otra forma de despertar al microcontrolador del sueño profundo, es por medio de los sensores capacitivos que posee el ESP-WROOM-32. `esp_sleep_enable_touchpad_wakeup()`
  - Interrupción Externa (EXT0 y EXT1). Se puede salir del modo de suspensión utilizando los pines GPIO RTC, a través de dos métodos de interrupción. Por un lado, la EXT0 que permite salir del modo *Deep Sleep* utilizando un único GPIO, y la EXT1 permite la utilización de varios GPIO. `esp_sleep_enable_ext0_wakeup(GPIO_NUM_X, level)`
  - Coprocesador UPL. Por último, en varias fuentes de información que se han empleado se menciona la posibilidad de despertar el dispositivo empleando el coprocesador UPL pero no ha sido posible encontrar un ejemplo práctico.

Aunque no es típico, es posible despertar del modo *Light Sleep* empleando todos estos métodos y no solo utilizando el temporizador.

Como se ha mencionado antes, la CPU permanece apagada durante la suspensión del dispositivo, por lo que al despertar el programa se ejecutaría como si fuera la

primera vez, reseteando todos los datos. No obstante, pueden se mantener algunos datos guardándolos en la memoria RTC.

- *Hibernation*. Este último modo, es que representa un mayor ahorro de batería ya que el consumo es de 2.5 µA, este consumo se debe a que se desactiva el oscilador interno de 8 MHz, el coprocesador UPL y la memoria RTC, los únicos elementos que permaneces activos son el reloj RTC y algunos pines GPIO RTC. Al igual que sucedía en el *Modem Sleep*, este modo no cuenta con una función que activa la suspensión, sino que es el modo *Deep Sleep* al que se le desactivan los elementos previamente nombrados.

```
void hibernation() {
 esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH,ESP_PD_OPTION_OFF);
 esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM,ESP_PD_OPTION_OFF);
 esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM,ESP_PD_OPTION_OFF);
 esp_sleep_pd_config(ESP_PD_DOMAIN_XTAL, ESP_PD_OPTION_OFF);
 deepSleep();
}
```

  Los métodos para despertar al dispositivo del modo hibernación, son los mismos que con el modo *Deep Sleep*.

Puesto que se desactiva la memoria RTC, ya no se pueden conservar datos como sucedía en el modo *Deep Sleep*, y la única manera de mantener algún dato es escribirlo directamente en la memoria EEPROM.

En este proyecto se utilizarán los modos de suspensión *Modem Sleep*, *Light Sleep* y *Deep Sleep*.

La idea original es utilizar el modo *Modem Sleep* durante el funcionamiento normal del programa, ya que se emplearán ambos ADCs (ADC1 y ADC2) para tomar medidas analógicas, y como se ha visto anteriormente, no es posible emplear el ADC2 para tomar medidas mientras se utiliza Wifi. No obstante, puesto que únicamente se utilizará el Wifi para la programación OTA, no será necesario inhabilitar el Wifi, ya que éste solo se activará cuando sea necesario actualizar el firmware. Puesto que tampoco se utilizará el bluetooth tradicional si no que se hará uso del BLE, con un consumo muy reducido, tampoco se encuentra necesario desactivar la señal bluetooth durante el modo *Modem Sleep*. Por último, sí que se reducirá la velocidad de CPU puesto que, para esta aplicación,

una frecuencia de operación de 240 MHz es innecesaria, pudiendo trabajar en perfectas condiciones a 40 MHz.

Se activará el *Light Sleep* en los periodos de tiempo en los que no sea necesario realizar ninguna operación (entre tomas de medidas), de esta forma controlaremos el tiempo que pasa y el consumo en estos instantes será muy reducido.

Por último, se suspenderá el dispositivo con el modo *Deep Sleep* cuando se detecte un periodo de inactividad, y se despertará por medio de los sensores capacitivos en el momento en el que el niño toque la superficie de la carcasa del SmartCube [45].

Estos sensores son capaces de detectar cualquier cosa que posea una carga eléctrica, como es el caso de la piel humana. Mientras no detecten carga eléctrica se podrá leer un valor numérico, el cuál descenderá en el momento en que se detecte dicha carga.

En este proyecto se utilizará el GPIO14 como detector capacitivo que despierte al sistema del modo *Deep Sleep*. Puesto que el SmartCube no estará expuesto directamente al niño, sino que se encontrará envuelto en una carcasa de plástico, será necesario comunicar el circuito integrado del SmartCube con el exterior de la carcasa, por ello, se empleará pintura conductora de la electricidad de tal forma que se garantice la conectividad entre ambos.

Se escoge la pintura conductora de *Bare Conductive* como método de interconexión, ya que presenta ciertas características favorables a este proyecto frente a otras pinturas eléctricas industriales. Principalmente al ser conductor de la electricidad, permite despertar al microcontrolador del modo de suspensión al tocar la superficie pintada. Otro punto importante sobre todo al trabajar con niños es que esta pintura no es tóxica, a diferencia de otras pinturas cuya composición se basa en partículas de metales, esta pintura se basa en una solución de agua, carbón negro y grafito, por lo que su aplicación es apta en aplicaciones con presencia de niños, aunque se recomienda la vigilancia de los más pequeños. Esta pintura se puede aplicar en cualquier superficie que no repela el agua por lo que es aplicable en la carcasa que realizará con PLA, además, se puede retirar fácilmente al limpiar con agua y jabón la superficie lo que favorece la realización de pruebas y el diseño sobre la carcasa. Por último, su grado de conductividad no están alto como el de un conductor metálico como podría ser el cobre, por lo que el acoplamiento de las señales inalámbricas con este conductor sería mucho menor, permitiendo la transmisión de información [46].

Prueba de Funcionamiento

Se implementará un código que opere con los modos de suspensión *Modem Sleep*, *Light Sleep* y *Deep Sleep*, de la misma forma en que se pretende utilizar con los SmartCubes.

El programa consistirá en un contador que irá aumentado su valor una unidad cada ciclo y lo mostrará por el puerto serie, cuando el valor del contador sea múltiplo de 10 (sin contar el 0) el sistema entrará en modo *Deep Sleep*. El método empleado para despertar al dispositivo del *Deep Sleep* será el touchpad, una vez despierto continuará el conteo, para no perder la información de la cuenta la variable de conteo se guarda en la memoria RTC.

- Funcionamiento normal (*Modem Sleep*) → Wifi no iniciado y velocidad CPU en 40 MHz (en este caso, el BLE tampoco se ha habilitado, aunque en la versión completa si estará activo). En este periodo se realizarán todas las operaciones,
- *Light Sleep* → Periodo de tiempo entre ciclos de operaciones. Se empleará el temporizador como método para despertar de este modo.
- *Deep Sleep* → Simula un periodo de inactividad, tras el cual, si es despertado retoma su funcionamiento.

Es un programa sencillo, pero con el que hay que tener en cuenta ciertas consideraciones.

- Justo antes de la iniciar el periodo de suspensión *Light Sleep* con su correspondiente función, es necesario dejar un pequeño periodo de tiempo para terminar de realizar las operaciones que estaban en ejecución, ya que al pausar la CPU la acción en ejecución se quedaría a medias. Para evitar esto, basta con emplear un pequeño *delay*.
- Puesto que se emplean varios métodos de *WakeUp* en el programa, uno para el *Light Sleep* y otro para el *Deep Sleep*, y ambos son métodos válidos para los dos modos de suspensión, se producirá un comportamiento inesperado si no se inhabilita y habilita los métodos de *WakeUp* para cada modo de suspensión. Es decir, si definimos inicialmente el temporizador para despertar del modo *Light Sleep* y posteriormente el touchpad en el *Deep Sleep*, lo que sucederá es que ambos modos se despertarán con el temporizador, por ello, es necesario inhabilitar el método de *WakeUp* del modo *Light Sleep* cuando se vaya a utilizar el *Deep Sleep* y habilitar el touchpad.

  ```
  esp_sleep_disable_wakeup_source(ESP_SLEEP_WAKEUP_TIMER);
  ```

Puede verse el programa completo en el anexo VIII.

En la Figura 58 puede observarse como el programa va mandando el valor del contador por el puerto serie, este envío se produce cada 3 segundos en los que el módulo se encuentra en suspensión por medio del modo *Light Sleep*. Al llegar a 10 en lugar de imprimir el valor por el puerto serie, aumenta su valor en uno y entra en modo *Deep Sleep*.

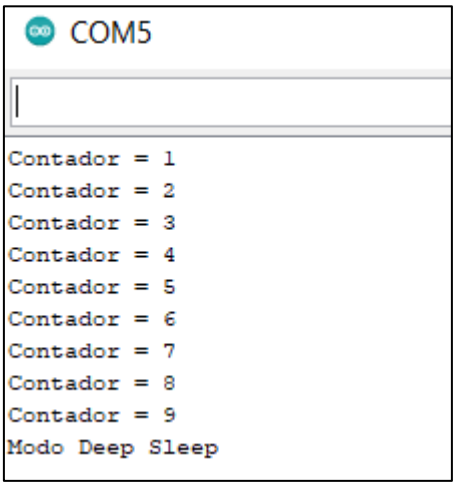


Figura 58. Programa entra en Deep Sleep (Prueba Modos de Ahorro)

Una vez dentro del modo *Deep Sleep*, el programa no se despertará hasta que detecte un cambio por el sensor capacitivo T6. En la Figura 59 se puede observar como el programa sale del modo de suspensión y continúa la cuenta hasta llegar al siguiente múltiplo de 10.

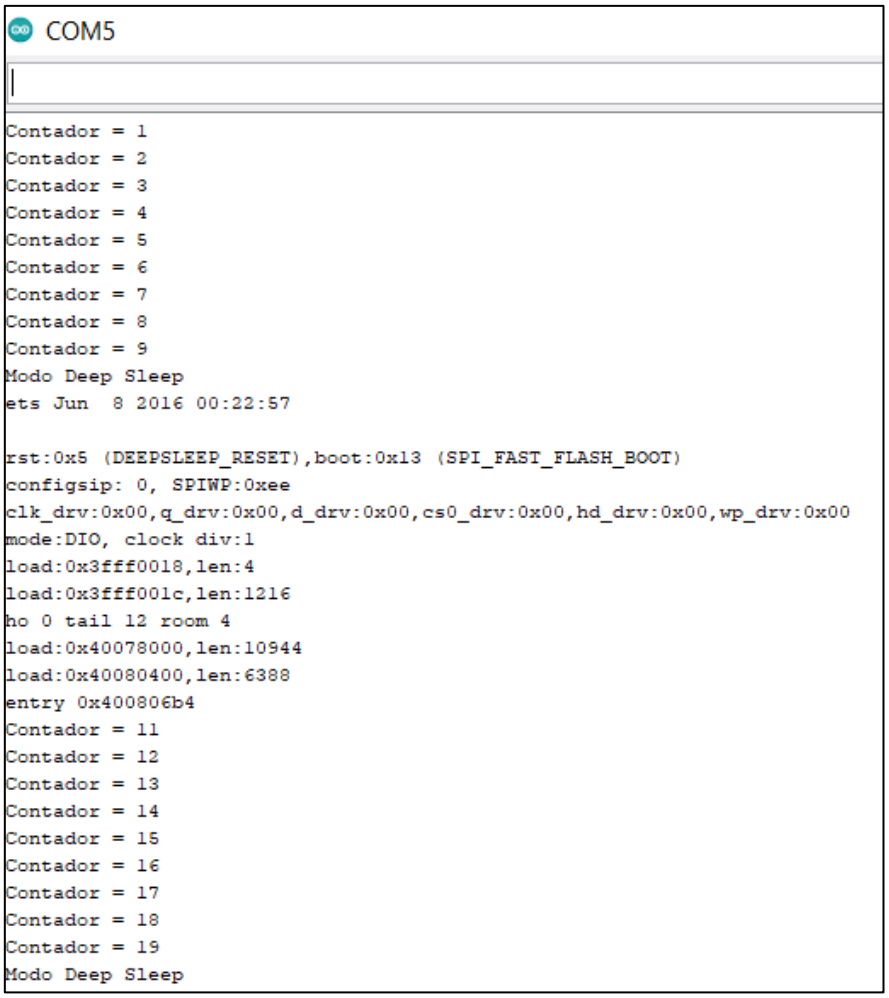


Figura 59. Programa sale del Deep Sleep y continua por donde lo dejó (Prueba Modos de Ahorro)

Capítulo 7:
Prueba de Funcionamiento del Sistema Completo

Capítulo 7: Prueba de Funcionamiento del Sistema Completo

En este apartado se integrarán todos los sistemas parciales vistos anteriormente en un solo sistema, con el objetivo de comprobar que no existen incompatibilidades de ningún tipo y que el sistema pueda ser implementado en un formato SmartCube. Por otro lado, se pondrá a prueba la durabilidad de la batería con un sistema lo más parecido posible al que se empleará en los futuros SmartCubes.

El sistema poseerá varios modos o estados de funcionamiento, inicialmente se encontrará en el modo 0, donde el sistema se encontrará en un estado de reposo a la espera de una orden. A continuación, se deberá introducir el modo 1 a través de la comunicación BLE, una vez en el modo 1, el sistema realizará una secuencia de estabilización de los sensores y una vez terminada, se regresa al modo 0. Ahora se introducirá por BLE el modo 2, ese estado es en el que se realizará la prueba de funcionamiento, el sistema cuenta con un modo de funcionamiento más, el modo 3, que únicamente se ocupa de entrar en modo de programación OTA.

Durante el modo 2 el funcionamiento del sistema será el siguiente, se tomarán medidas del acelerómetro, de las galgas y la batería cada ¼ de segundo y las mostrará por BLE. Con cada secuencia de envió de información se mandará también el instante de tiempo (en horas) desde que se inició el modo 2. Con el objetivo de mantener una buena organización de la información, tanto el sistema de galgas, acelerómetro, batería y tiempo tendrán su propio servicio BLE donde mirar las medidas, en el caso de las galgas y el acelerómetro que mandan varias medidas, cada una de ellas utilizará una característica.

Puesto que uno de los objetivos de esta prueba es la de comprobar la autonomía del sistema ante una situación “real”, se realizarán varios ensayos donde se implementarán los diferentes modos de funcionamiento anteriormente explicados. En la primera prueba se implementará únicamente el *Modem Sleep*, implementado el tiempo entre ciclos con la función *delay ()*; en la segunda prueba este tiempo entre ciclos se introducirá por medio del modo *Light Sleep*; por último, en la tercera prueba se hará uso del modo *Deep Sleep* al detectar un periodo de inactividad.

En términos de eficiencia, el programa empleado no lo es demasiado, ya que se encuentra enviando información continuamente mientras el modo 2 esté activo. En el programa final del SmartCube se tendrán en cuenta los periodos cortos de inactividad y no se realizarán

envíos de información innecesarios, pero puesto que se está evaluando la autonomía del sistema al emplear una batería, es conveniente realizar los ensayos en situaciones más desfavorables. Por ello, durante el modo 2 de funcionamiento, además de tomar medidas de los sensores y realizar el envío de información, también se encontrará encendido un diodo LED durante todo el periodo de la prueba.

Pese a que inicialmente se pretendía utilizar la placa de desarrollo casera con el ESP-WROOM-32, al cargar exitosamente el programa en la placa, el cliente BLE (smartphone) no es capaz de detectar ningún dispositivo BLE. Tras comprobar que la placa funciona correctamente con otros programas y revisar que el esquemático de la placa, donde únicamente se diferencia del de los módulos ESP32 devKitc v4 en los elementos relacionados con la programación Serie-USB que llevan integrados, se llega a la conclusión de que, el problema se debe a un problema de interferencias con la señal BLE relacionado con el montaje de la placa. Puesto que para el montaje de la placa no se contaba con una placa de circuito integrado especifica parra el montaje del ESP-WROOM-32, se realizó el montaje sobre una placa de circuito impreso genérica con orificios conductores de doble cara. La unión del microcontrolador con la placa se llevó acabo por medio de la soldadura de unos hilos conductores, los cuales se conectan con unos pines a través de unas pistas realizadas con soldadura de estaño. Todos estos elementos conductores alrededor de la antena del microcontrolador, sumado al paso de la corriente a través de los orificios conductores, pueden generar una gran cantidad de interferencias electromagnéticas que dificulten el funcionamiento del BLE.

Por este motivo y el limitado tiempo del que se dispone para la finalización de este proyecto se realizarán la prueba con el módulo ESP32 devKitc v4. Puesto que este módulo ya cuenta con un diodo Led de estado, que se mantiene encendido mientras el módulo esté alimentado, no se activará ningún diodo Led durante la prueba.

Durante las primeras interacciones del programa surge un fallo en el funcionamiento del BLE, concretamente en el servicio dedicado a la muestra de los datos recogidos de las galgas extensiométricas. En lugar de ver los valores de las 6 características cambiando cada ¼ segundo, únicamente se comportan de esta forma las 4 primeras galgas, de la 5ª galga únicamente se puede ver la característica, pero no el valor ni tampoco activar las notificaciones; y la 6ª ni siquiera se muestra en el servicio. La primera impresión que se tiene del problema es que puede tratarse de algún tipo de problema relacionado con la

memoria dedicada a cada servicio, por lo que decide dividir el servicio de las galgas en dos, 3 características en un servicio y otras 3 en otro. Al realizar este cambio, el sistema funciona correctamente, mostrando todas las características y sus valores. Tras investigar un poco acerca del problema y de las librerías que se emplean para el sistema BLE, se alcanzan una explicación y una solución para lo sucedido. Resulta que la librería BLEServer.h emplea un conjunto de variables de control llamadas *Handles*, las cuales llevan a cabo el funcionamiento de los servicios, características y descriptores, cada uno de estos elementos utiliza una variable de control. Por lo tanto, el motivo por el que el servicio de las galgas era incapaz de mostrar todas las características correctamente, se debe a que el número predeterminado de *Handles* es insuficiente para llevar a cabo todas estas funciones. No obstante, el número de *Handles* que emplea un servicio, puede definirse en el momento que se crea dicho servicio de la siguiente manera [47], [48].

```
BLEService *pService_1 = pServer->createService(BLEUUID(SERVICE_UUID), numHandles);
```

Por lo tanto, el límite de características que puede emplear un servicio lo define el número de *Handles* que este servicio emplee, y a su vez, el número máximo de variables de control que pueden definirse para un servicio queda definido por el tamaño máximo de la variable que es de 0xFFFF.

Una vez, todo funciona correctamente se lleva realiza la primera prueba de autonomía de la batería, empleando el módulo comercial, un divisor resistivo cuya corriente es equivalente a los 6 sistemas de galgas, el acelerómetro y el divisor resistivo de la batería, todos estos elementos se encuentran funcionando continuamente. Además, el reloj interno de la CPU se mantiene a 240MHz, por lo que esta sería la situación más desfavorable.

Bajo estas condiciones se obtiene la siguiente gráfica que puede observarse en la Figura 60, que muestra la descarga de la batería durante 2 horas y 45 minutos.

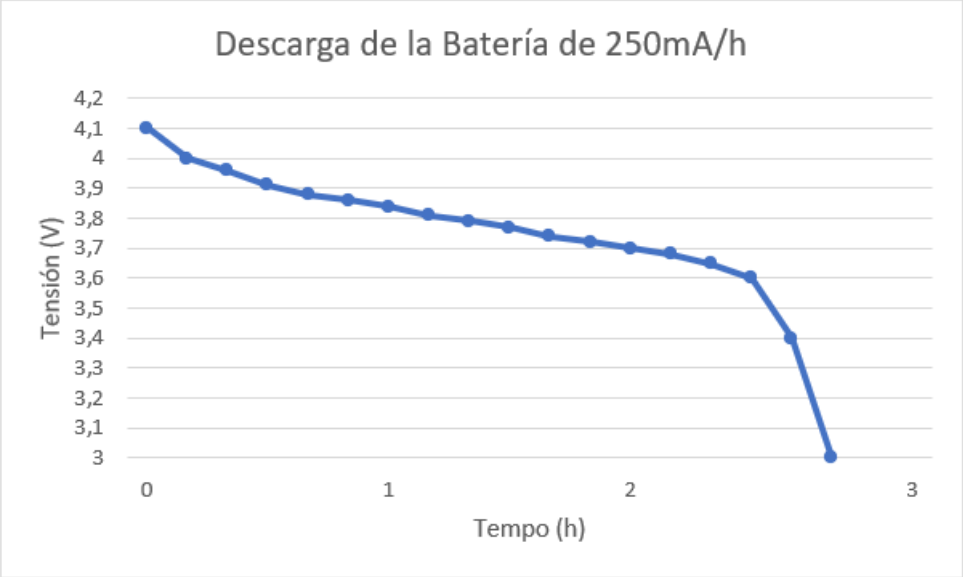


Figura 60. Descarga de la Batería (Situación más desfavorable)

Como se puede ver en la gráfica, la duración de la batería difiere mucho de la teórica que se calculó para una situación desfavorable similar en el apartado de la batería en la selección de componentes, donde la estimación de la duración de la batería era de algo más de 4 horas. Esta gran diferencia de tiempo se debe principalmente al consumo del microcontrolador ya que, en la estimación teórica, se planteó un consumo aproximado de 30mA para una velocidad de CPU de 80 MHz, mientras que el consumo real a 240 MHz seguramente ronde los 50mA.

Para la segunda prueba se introducirá el modo *Light Sleep*, sustituyendo la función *delay ()* en los periodos entre ciclos. No obstante, en este tipo de aplicaciones como el *BLE Scanner* o *nrf Connect*, surgen problemas al trabajar de esta forma con modos de suspensión que apagan el BLE (*Light Sleep y Deep Sleep* en este caso). Cuando el sistema entra en estos modos de suspensión, apaga el sistema BLE por lo que, al salir de él, hay que volver a conectar con el dispositivo BLE, y aunque algunas aplicaciones como *nrf Connect* posee sistemas de auto reconexión, hay que activar de nuevo las notificaciones de las todas características que existan. Por ello, sin una aplicación específica para este programa que reconecte automáticamente y active las notificaciones al salir de los modos de suspensión, no se ve posible realizar este tipo de pruebas de funcionamiento.

Adicionalmente, se ha encontrado un problema que imposibilita el uso del modo *Light Sleep* de la forma que se ha planteado anteriormente ya que, en periodos tan cortos de tiempo como es ¼ de segundo, en el instante en que sale de la suspensión y trata de

reconectar con el dispositivo BLE, el microcontrolador vuelve a entra en modo *Light Sleep*, imposibilitando la reconexión con el dispositivo y permaneciendo desconectado en todo momento.

De esta forma se descarta el uso del *Light Sleep* para este proyecto, ya que para largos periodos de inactividad se hará uso del *Deep Sleep* que proporciona un mayor ahorro energético.

A continuación, se realizará una última prueba para comprobar el ahorro energético que supone el uso de los modos de suspensión en el proyecto. Puesto que finalmente no se empleó el *Modem Sleep* durante la primera prueba de funcionamiento, se reducirá la velocidad de la CPU a 80 MHz. El sistema se encontrará trabajando en estas condiciones hasta pasadas 2 horas, momento en el que el sistema entrará en modo *Deep Sleep*. Pese a que el microcontrolador es capaz de reducir la velocidad de CPU por debajo de los 80 MHz, no es viable en este proyecto ya que, por debajo de los 80 MHz el microcontrolador no soporta las funciones Wifi y BLE.

En la Figura 61 se puede observar un gráfico que muestra la descarga de la batería bajo estas nuevas condiciones.

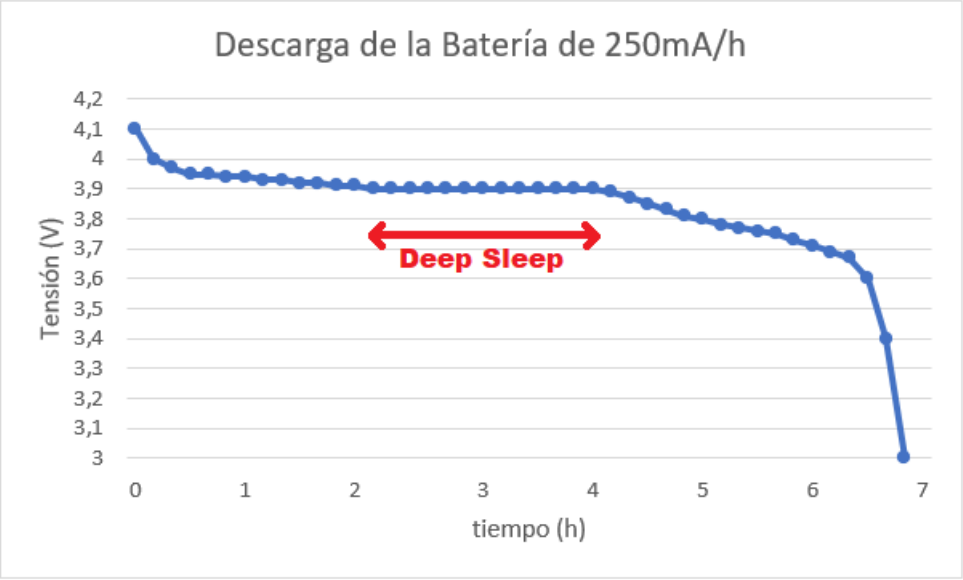


Figura 61. Descarga de la batería (Con Modos de Suspensión)

La prueba comienza con la descarga de la batería con la velocidad de CPU reducida a 80 MHz, alcanzando 3.9V al cabo de 2 horas. En este punto el microcontrolador entra en modo *Deep Sleep*, donde el consumo es tan reducido que pasadas 2 horas la tensión de la batería se mantiene en 3.9V. Tras salir del modo *Deep Sleep* transcurridas 4 horas de

prueba, se retoma el funcionamiento inicial del microcontrolador, aguantando 2 horas y 50 minutos hasta alcanzar el límite de descarga de la batería.

Con esta prueba, queda comprobado el potencial del modo *Deep Sleep* como método de ahorro de energía durante largos periodos de tiempo. Además, se garantiza la autonomía del sistema con el modo de funcionamiento normal, por más de 4 horas, incluso casi alcanzando las 5 horas.

Puede verse el código completo utilizado y la tabla de valores con la que se elaboran las gráficas en los Anexos IX y II respectivamente.

Capítulo 8:
Diseño de los SmartCubes

8.01 Diseño Esquemático

8.02 Diseño PCB

8.03 Modelos 3D

Capítulo 8: Diseño de los SmartCubes

Una vez comprobado el funcionamiento del sistema diseñado para esta nueva versión de los SmartCubes, es momento de comprimir este sistema en un diseño con forma de cubo, en este apartado se verá en detalle cada una de las caras que componen el cubo, comprendiendo la distribución y el diseño de cada una.

8.01 Diseño Esquemático

Primero se verá el esquemático de cada cara del cubo, donde se podrán ver los componentes y circuitos que albergará cada cara. La distribución se tratará de mantener lo más parecida posible a versiones anteriores, pero teniendo en cuenta los cambios de esta nueva versión.

- **Cara 1**

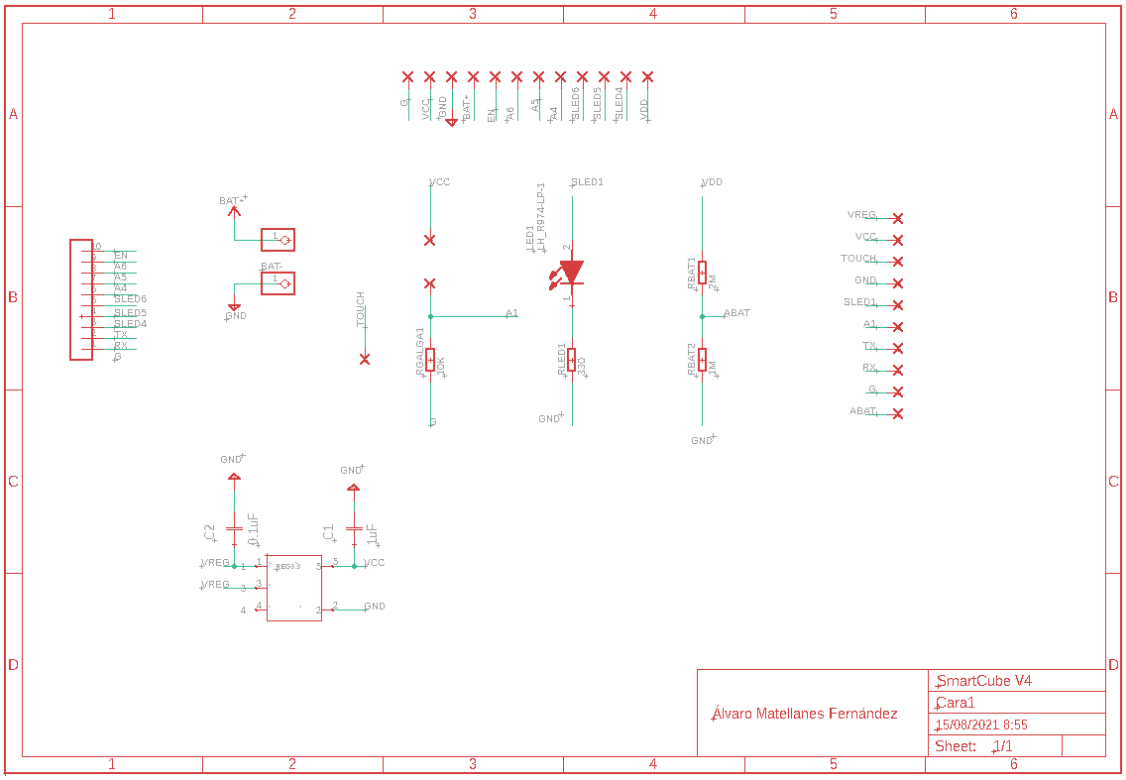


Figura 62. Esquemático Cara 1

El SmartCube tendrán tres elementos comunes en cada una de sus caras, que se corresponden con los sistemas de los diodos Leds, el de las galgas extensiométricas y el touchpad, por lo que únicamente se nombrarán esta vez.

En esta primera cara, a parte de los sistemas ya mencionados, se encuentran, además, los orificios de conexión de la batería, el divisor de tensión necesario para poder tomar las medidas de la tensión de batería, el regulador de tensión de 3.3V con sus correspondientes condensadores de desacoplo, unos conectores que encajan con otros de la cara 5, y las almohadillas de soldadura que permitirán soldar esta cara con la cara 2 y 3; los conectores y las almohadillas permiten la comunicación de las pistas con otras caras del cubo.

- **Cara 2**

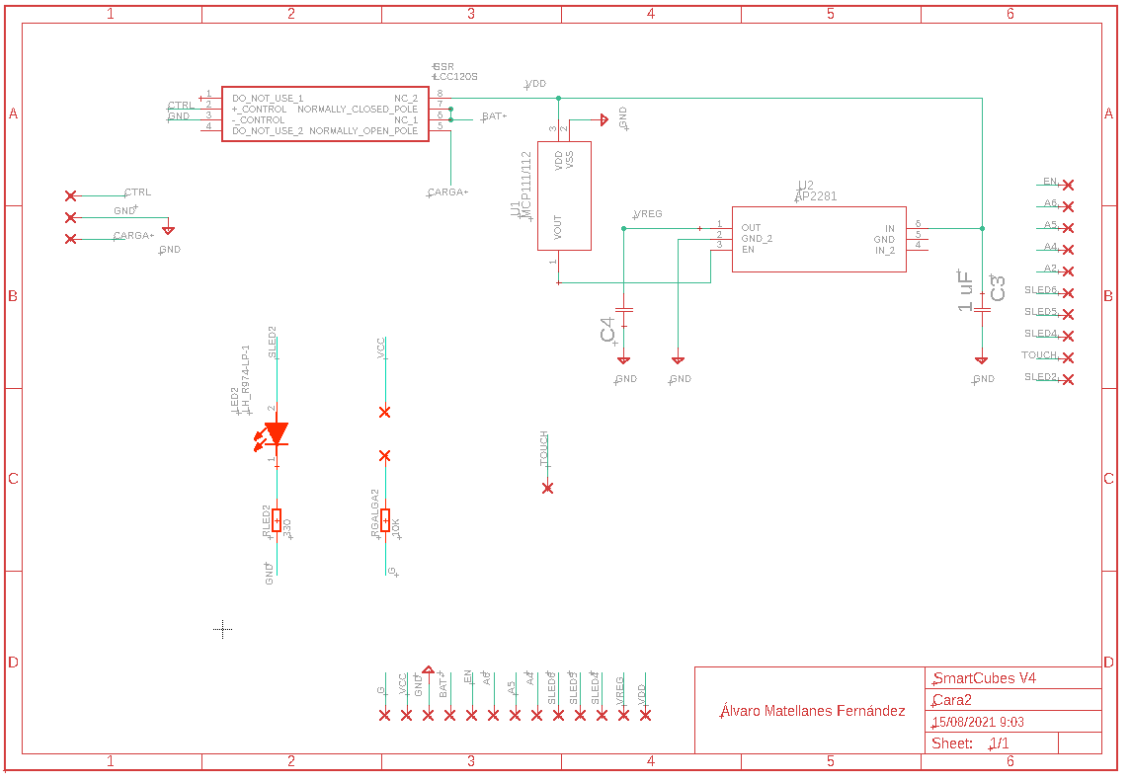


Figura 63. Esquemático Cara 2

En esta cara se encuentra el relé de estado sólido encargado de la conmutación entre estado de carga de la batería y estado en ejecución, y el sistema de protección por la descarga de la batería, conformado por el AP2281 y el MCP111/112 y sus correspondientes condensadores de desacoplo.

Esta cara comunica con las caras 1 y 3 a través de las almohadillas de soldadura, las almohadillas que se pueden ver a la izquierda no comunican con ninguna cara, sino que, conectan el conector magnético de carga con el interior del cubo.

- **Cara 3**

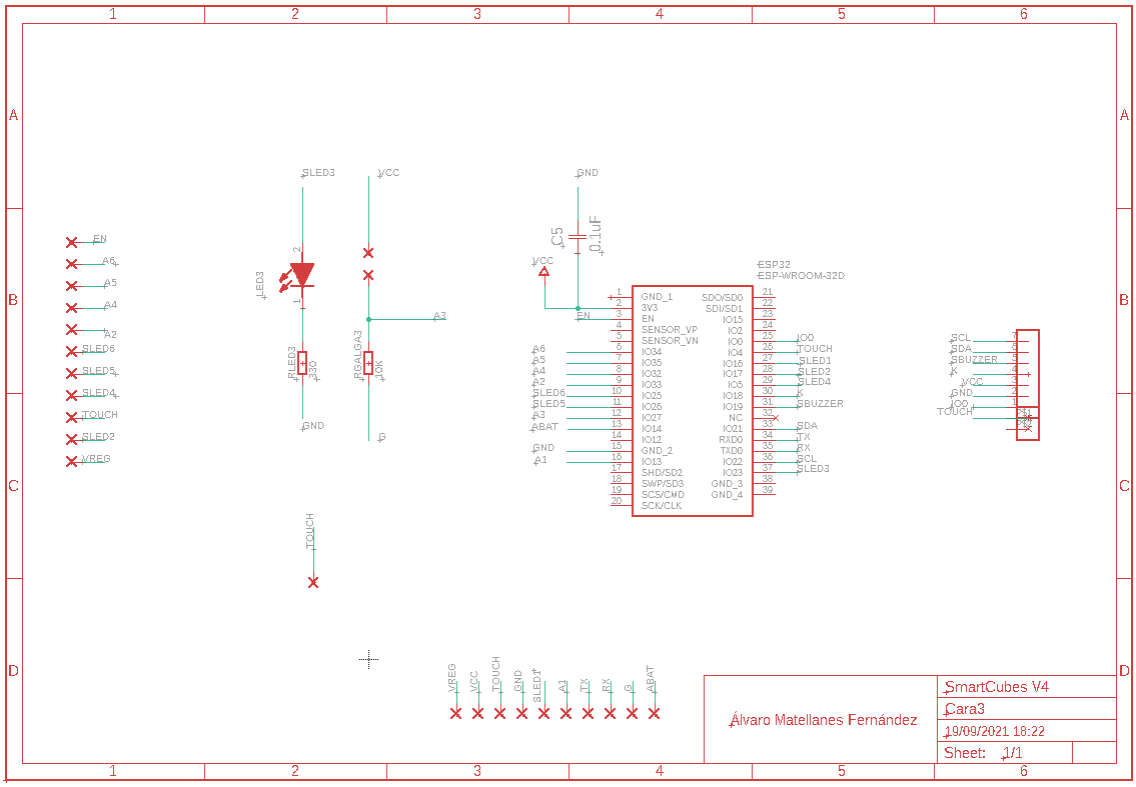


Figura 64.Esquemático Cara 3

Esta cara es con diferencias la cara con mayor ocupación de espacio, ya que, en ella se encuentra el ESP-WROOM-32, que como ya se vio anteriormente, cuenta con unas dimensiones muy elevadas teniendo en cuenta los requerimientos de este proyecto, junto con un condensador de desacoplo en paralelo con la tensión de alimentación procedente del regulador de tensión.

Las almohadillas de soldadura comunican con las caras 1 y 2, y el conector de la derecha conecta con la cara 4.

- **Cara 4**

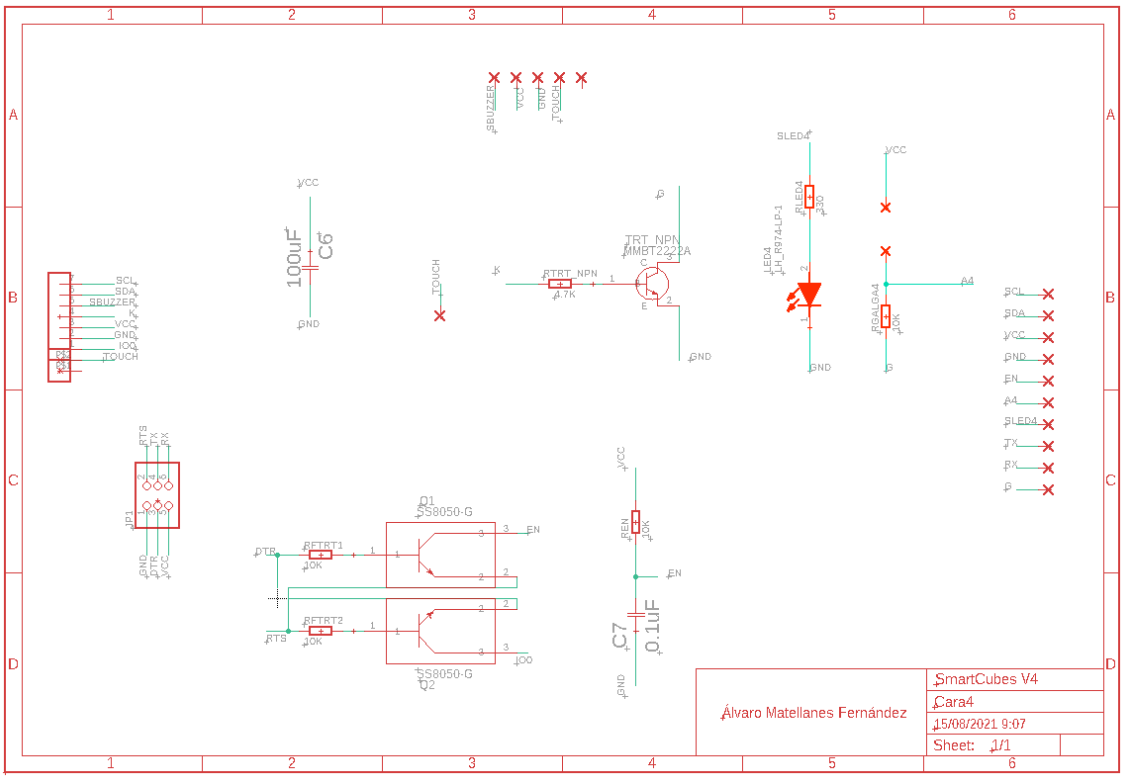


Figura 65. Esquemático Cara 4

En esta cara se encuentra el sistema de programación del microcontrolador principalmente, que se conforma de los orificios de conexión de los terminales de programación y el sistema de auto programación. También se sitúa en esta cara el sistema de activación de la medición de las galgas y un condensador bulk en paralelo con la alimentación.

Esta cara conecta con las caras 5 y 6 a través de las almohadillas de soldadura y con la cara 1 con el conector de la izquierda.

- **Cara 5**

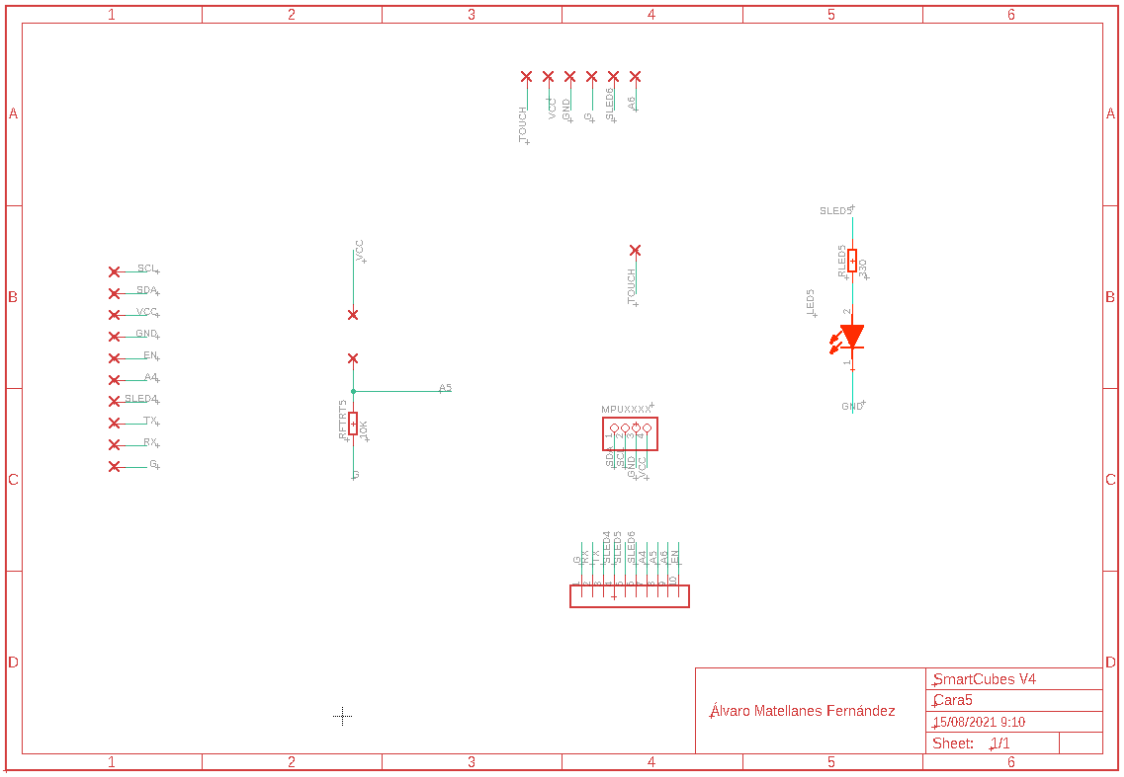


Figura 66. Esquemático Cara 5

El elemento principal que se encuentra en esta cara es el acelerómetro que, debido a sus grandes dimensiones, ocupa gran parte de la superficie de la cara.

La cara 5 conecta con las caras 4 y 6 por medio de las almohadillas de soldadura y con la cara 1 con los conectores.

- **Cara 6**

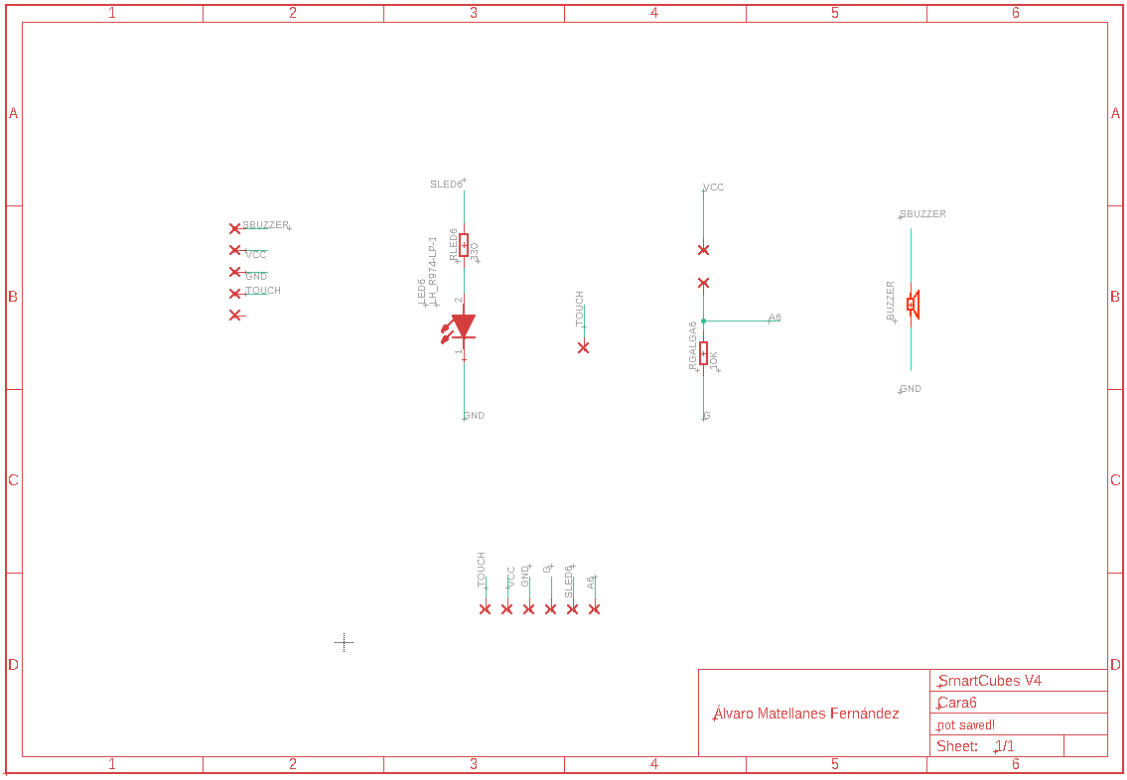


Figura 67. Esquemático Cara 6

En esta última cara, se encuentra el *buzzer* que, aunque posee unas dimensiones considerables, aún deja espacio libre en la cara. No obstante, es conveniente dejar todo el espacio libre posible ya que en el interior del cubo debe introducirse la batería.

Se conecta la cara 6 con las caras 4 y 5 por medio de las almohadillas de soldadura.

8.02 Diseño PCB

Existen una serie de reglas o pautas generales que deben tratar de seguirse a la hora de diseñar placas de circuito impreso (PCB), de lo contrario, el producto resultante podría perder en calidad e incluso acabar generando fallos eléctricos.

Otro factor a tener en cuenta a la hora de realizar el diseño, es la empresa que se encargará de la fabricación de los diseños. Cada fabricante trabaja bajo diferentes condiciones (ya sea por diferencia en la maquinaría, metodología, etc), por lo que cada empresa cuenta con sus propias directrices de diseño PCB [49].

Para este proyecto la empresa escogida para la fabricación de las PCB es JLCPCB, una empresa de origen chino líder en el sector. Las principales razones por las que se escoge esta empresa son el precio y el tiempo de entrega, en menos de 18 días realizan todos los procesos de fabricación y distribución, con un precio muy competitivo.

Pautas de Diseño Generales

A continuación, se tratarán algunas consideraciones a tener en cuenta en cualquier diseño de circuito impreso [50].

El primer aspecto a tener en cuenta será la distribución de los componentes, en concreto con este punto se tratará de evitar problemas a la hora de la soldar los componentes en la placa. Al situar los elementos, se deberá tratar de orientas los componentes similares en la misma dirección, tratando de evitar que componentes de mayores dimensiones solapen a los más pequeños. En caso de que el diseño contenga tanto componentes de montaje superficial (SMT) como pasantes (THT), conviene distribuir los componentes SMT y los pasantes en espacios claramente separados, de esta forma se facilita en proceso de ensamblaje, evitando de esta forma, situar elementos pasantes encima de elementos SMT.

La siguiente pauta de diseño abarca la colocación de las conexiones de energía, masa y señal. En caso de que la placa cuente con más de dos capas, es recomendable ubicar planos de energía y masa en las capas internas, ya que aportan rigidez a la estructura y facilitando los enrutamientos. En caso de solo poseer dos capas, lo normal es generar un plano de tierra en una de las capas, en ambos casos siempre evitando generar formas de anillos alrededor de los componentes y colocar numerosas vías al enrutar las pistas con el plano de masa. En el primer caso, los anillos de masa se comportan como inductores que,

al entrar en contacto con un campo magnético externo, podría generarse una corriente eléctrica en el plano de masa, dando lugar a interferencias electromagnéticas (EMI). En el caso de las vías, es recomendable emplear una única vía pasante por plano de tierra ya que, en caso de utilizar varias vías se generará una tensión diferencial en cada una de ellas, dando lugar a bucles de puesta a tierra [51]. En cuanto a las conexiones de señal, se deberá enrutar siguiendo el esquemático previamente realizado, tratando de que las pistas queden lo más cortas y directas posibles (sin vías), ya que de lo contrario la resistencia interna de la pista aumenta, incrementando a su vez las caídas de tensión y las pérdidas de energía. Otro parámetro importante, es el ancho de la pista que se diseñará principalmente en función de las corrientes que circulen por ellas, existen fórmulas para calcular el ancho de pista necesario en función de las especificaciones de cada circuito, aunque pueden usarse páginas web especializadas para facilitar este proceso [52], por norma general se suelen hacer más anchas las pistas de alimentación por las que circulan mayores corrientes.

Para evitar generar interferencias entre los circuitos de potencia y de control, es conveniente separar ambas zonas dentro de la PCB, evitando conexiones entre los planos de tierra de ambos circuitos.

Otro aspecto importante es el sobrecalentamiento del sistema, por ello es necesario informarse acerca de las especificaciones técnicas de cada componente y comprobar si bajo las condiciones en las que trabajará se calentará en exceso. En caso afirmativo, deberá recurrirse a disipadores de calor o alivios térmicos.

La última pauta de diseño, es el uso de los sistemas de verificación ERC (*Electric Rule Check*) y DRC (*Design Rule Check*) para comprobar que no existen fallos ni en el esquemático ni en el diseño de la placa.

Diseño de las PCBs de los SmartCubes

Cada proyecto tiene sus propias especificaciones y condiciones de trabajo, por lo que las reglas de diseño general anteriormente vistas deben ser adaptadas a cada caso concreto.

Las principales restricciones de los SmartCubes son sus reducidas dimensiones y su forma cúbica, factores que condicionan en gran medida la colocación de los componentes. Además, en esta nueva versión se debe tener en cuenta la adhesión de las galgas extensiométricas, ya que requieren de un espacio de colocación que pueda deformarse al

aplicar un mínimo de fuerza; y la inserción del conector magnético en la PCB, que conlleva un corte en la placa con la silueta del conector. Puede verse el espacio disponible en cada cara en la Figura 68 y la Figura 69.

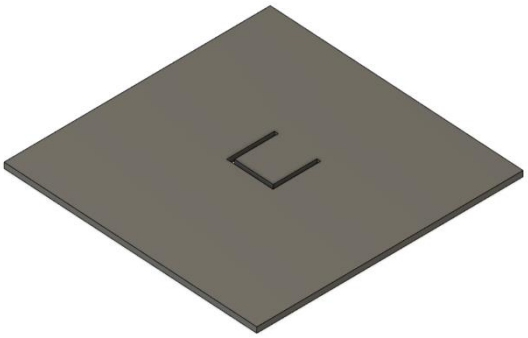


Figura 68. Espacio PCB Caras 1, 3, 4, 5 y 6

Figura 69. Espacio PCB Cara 2

Como se ha mencionado antes, un factor importante a la hora de distribuir los componentes en la placa, es su forma cúbica, la cual implica tener que conectar las pistas de las diferentes caras por medio de almohadillas de soldadura y conectores. Por lo que, se trata de asignar los pines del microcontrolador y colocar los componentes da tal forma, que se las pistas queden lo más cortas y directas posibles hacia estos puntos de conexión. El único componente electrónico pasante que se utiliza es el MPU9150, situado en la cara 5 y con el cual se realizará una excepción en cuanto a las normas de diseño. Pese a que no se recomienda nunca pasar componentes SMD por debajo de componentes pasantes, debido a que al aplicar presión sobre la superficie a la que se encuentra adherida la galga, ésta se deformará hacia el interior del cubo, por lo que el MPU no debe estar completamente pegado a la placa y por ello, se sitúan dos resistencias debajo a modo de apoyo. Este mismo problema sucede también con el ESP-WROOM-32, pero en este caso al ser éste un SMT es más complicado de solventar, por lo que, si no se alcanza una solución durante la soldadura del módulo, se prescindirá de la medición de fuerza en la cara 3 del cubo.

En cuanto a la implementación de planos de masa, pese a que se conseguiría reducir en gran medida las caídas de tensión y las impedancias de las pistas reduciendo su longitud, esta opción no es viable en este proyecto ya que, al realizar transmisiones de datos por

vía inalámbrica, si se emplearan planos de masa es muy posible que produjera un apantallamiento de la señal, impidiendo realizar la transmisión.

Como se ha dicho antes, se tratará de diseñar las pistas lo más cortas y directas posibles, pero además hay definir un ancho de pista adecuado en función de las especificaciones de nuestro circuito. Para calcular el ancho, se tomará una situación límite de tal forma que aseguremos el buen funcionamiento del sistema, se establecerá una corriente de 0.5A para garantizar el correcto funcionamiento durante la programación OTA, donde suelen aparecer picos de corriente; un espesor de placa de 0.8 mm, espesor más fino de lo normal para garantizar la deformación de la superficie de la galga; y un incremento de temperatura de 5 grados. Con estas especificaciones el ancho sugerido es de 0.3029 mil, en el diseño de la PCB se escogerá el tamaño de pista inmediatamente superior a este valor que, en este caso es 0.3048 mil. Para un diseño más eficiente, las pistas de señal deberían tener un ancho más reducido, acorde con las corrientes que circulen por ellas, pero puesto que finalmente al realizar los diseños no ha habido problemas de espacio y para evitar sobrecalentamientos se han establecido el ancho de todas las pistas en 0.3048 mil.

Otra consideración a tener en cuenta en nuestro circuito es que, hay que tener especial cuidado con la colocación de las almohadillas de soldadura que conectarán las diferentes caras de del cubo ya que, de lo contrario, aparecerán problemas en la etapa de soldadura. Al igual que con los conectores ya que, si no se cuida su colocación el cubo no podrá cerrarse. Además, en el caso de los conectores hay que tener cuidado con no colocar componentes en el espacio donde deberán conectarse con su complementario.

En cuanto a la colocación de los condensadores, se tratará de situar los condensadores de desacoplo lo más cerca posible de su chip asociado para maximizar el filtrado de la señal, aunque las limitaciones de espacio condicionarán el diseño. Por otro lado, el condensador de *bulk* de 100uF no es necesario que se encuentre cerca de ningún dispositivo, por lo que se situará en la cara 4.

Por último, puesto que este proyecto por completo trabaja a baja tensión, no es necesario diferenciar entre etapa de potencia y de control.

A continuación, se muestran los diseños PCB de todas las caras que conforman los SmartCubes.

- **Cara 1**

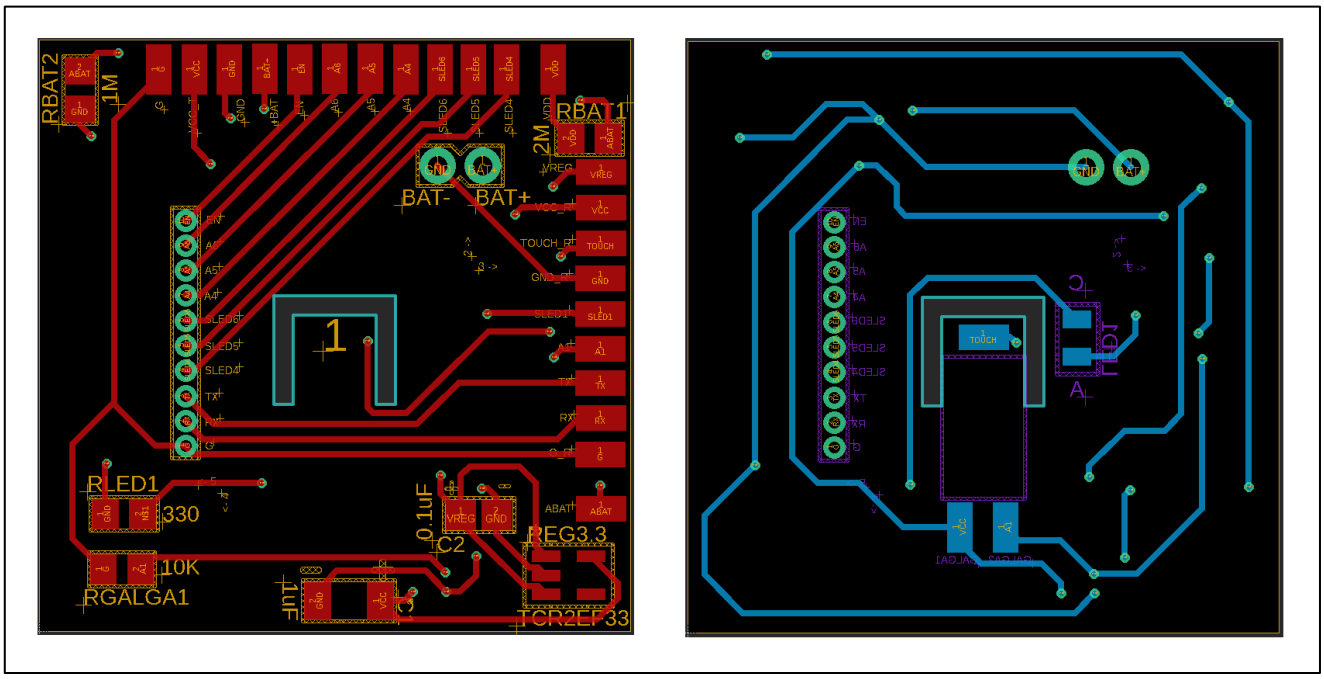


Figura 70. PCB de la Cara 1. Capa TOP y BOTTOM.

- **Cara 2**

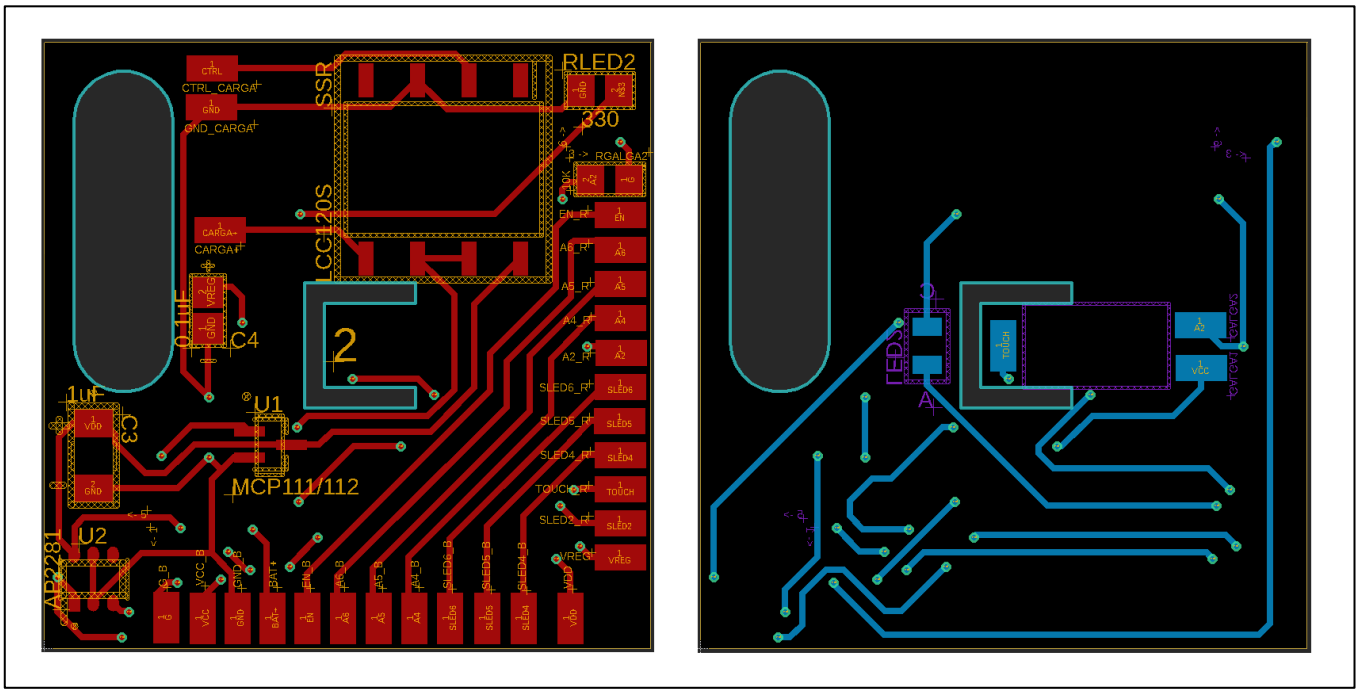


Figura 71. PCB de la Cara 2. Capas TOP y BOTTOM.

- **Cara 3**

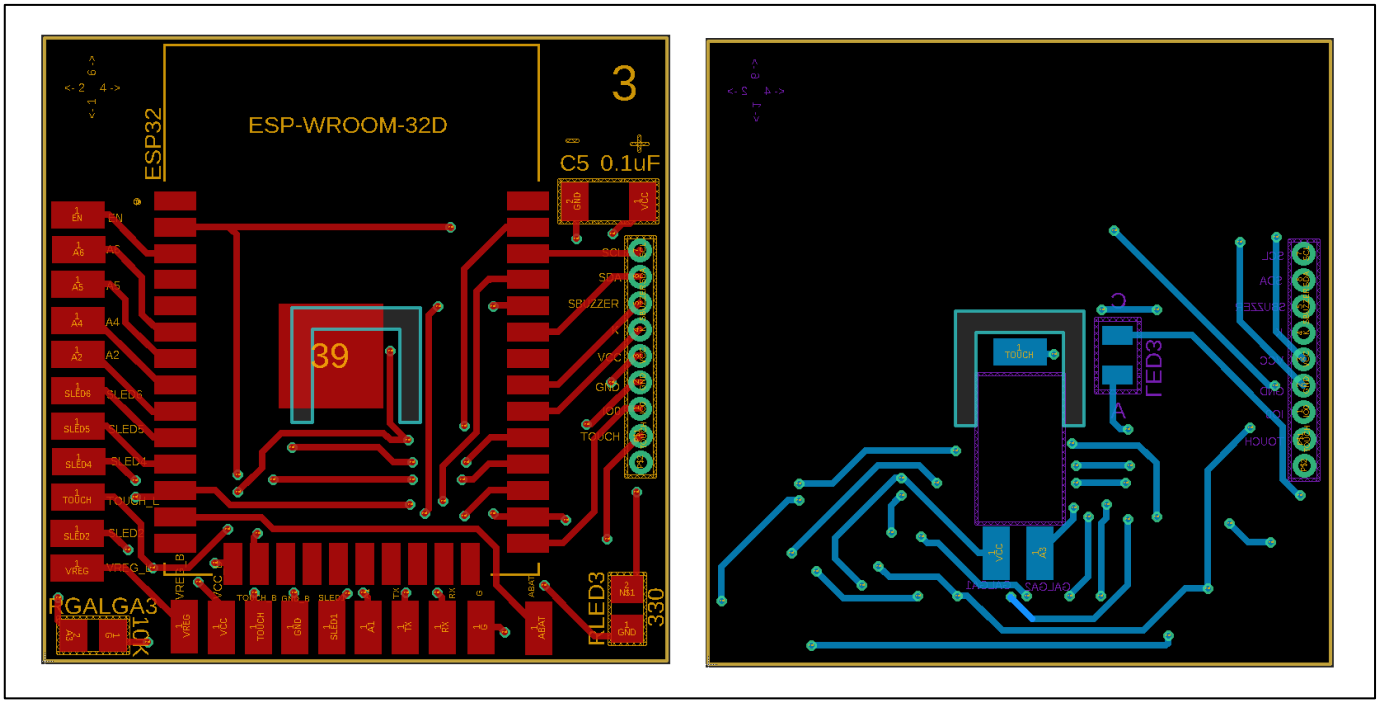


Figura 72. PCB de la Cara 3. Capas TOP y BOTTOM.

- **Cara 4**

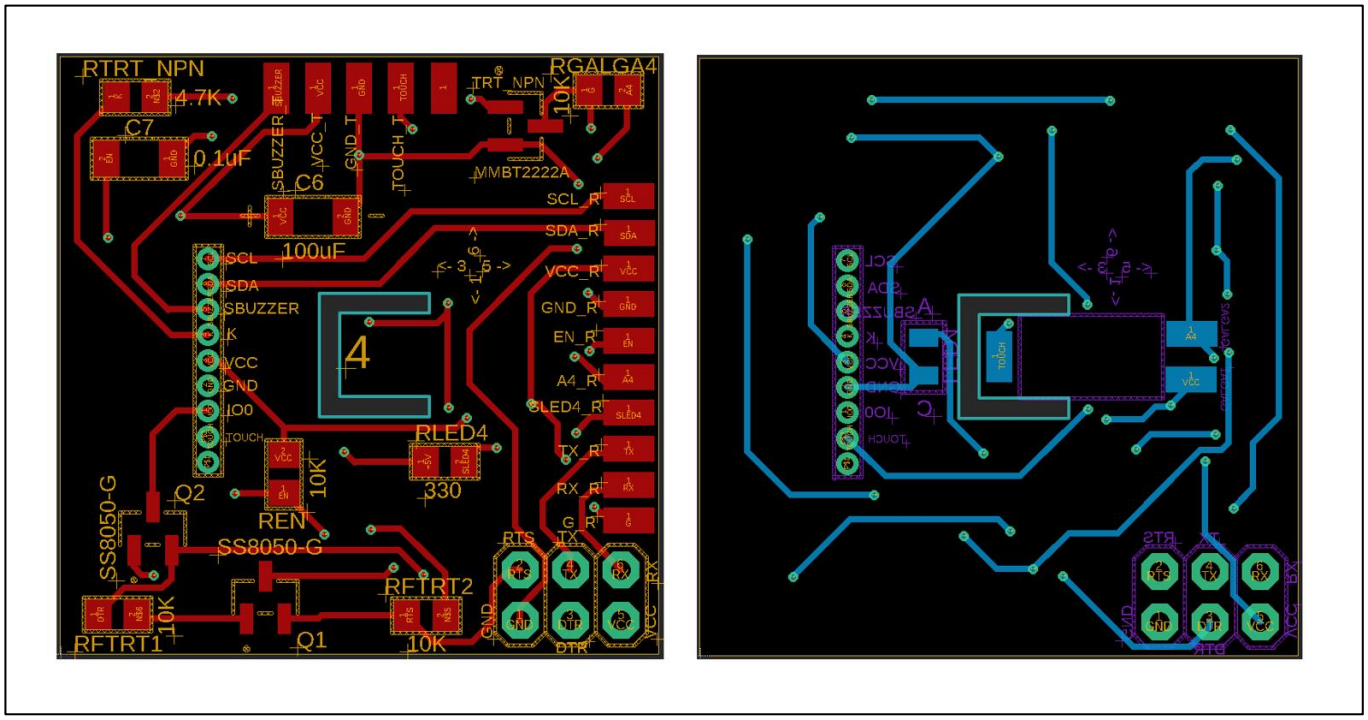


Figura 73. PCB de la Cara 4. Capas TOP y BOTTOM.

- **Cara 5**

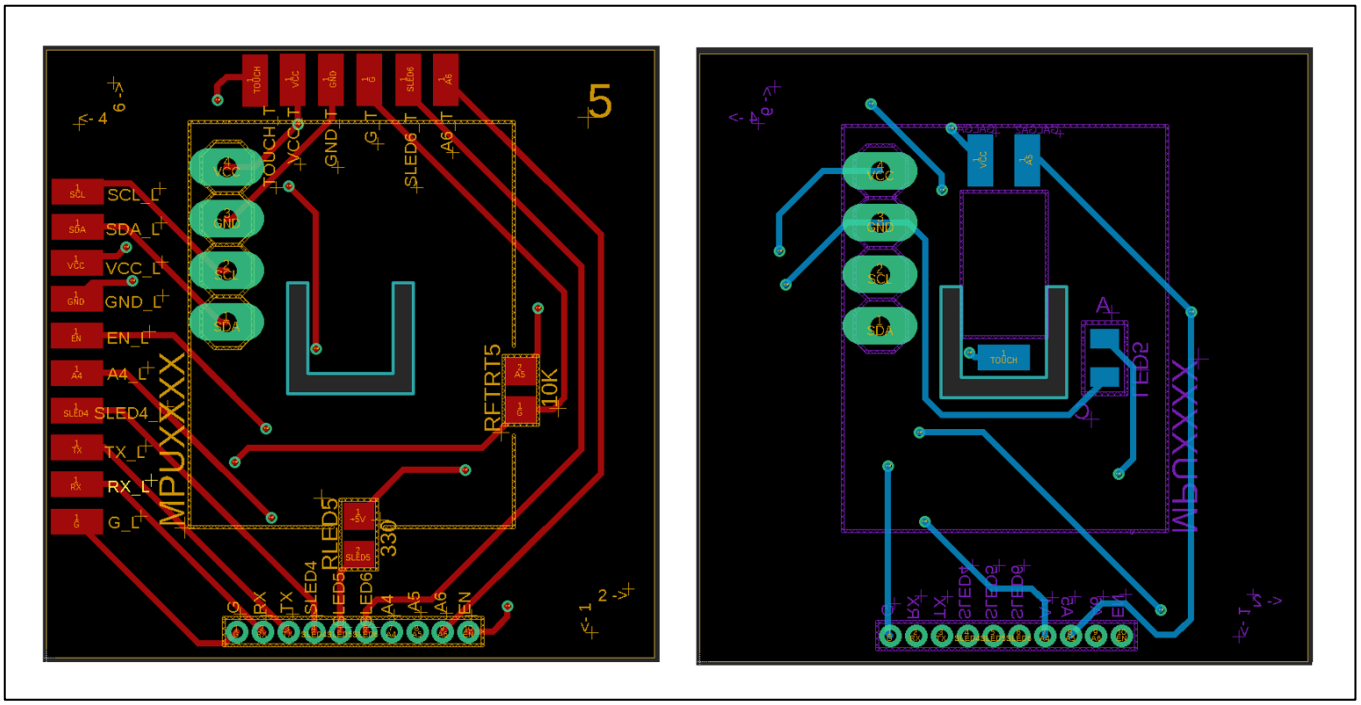


Figura 74. PCB de las Cara 5. Capas TOP y BOTTOM.

- **Cara 6**

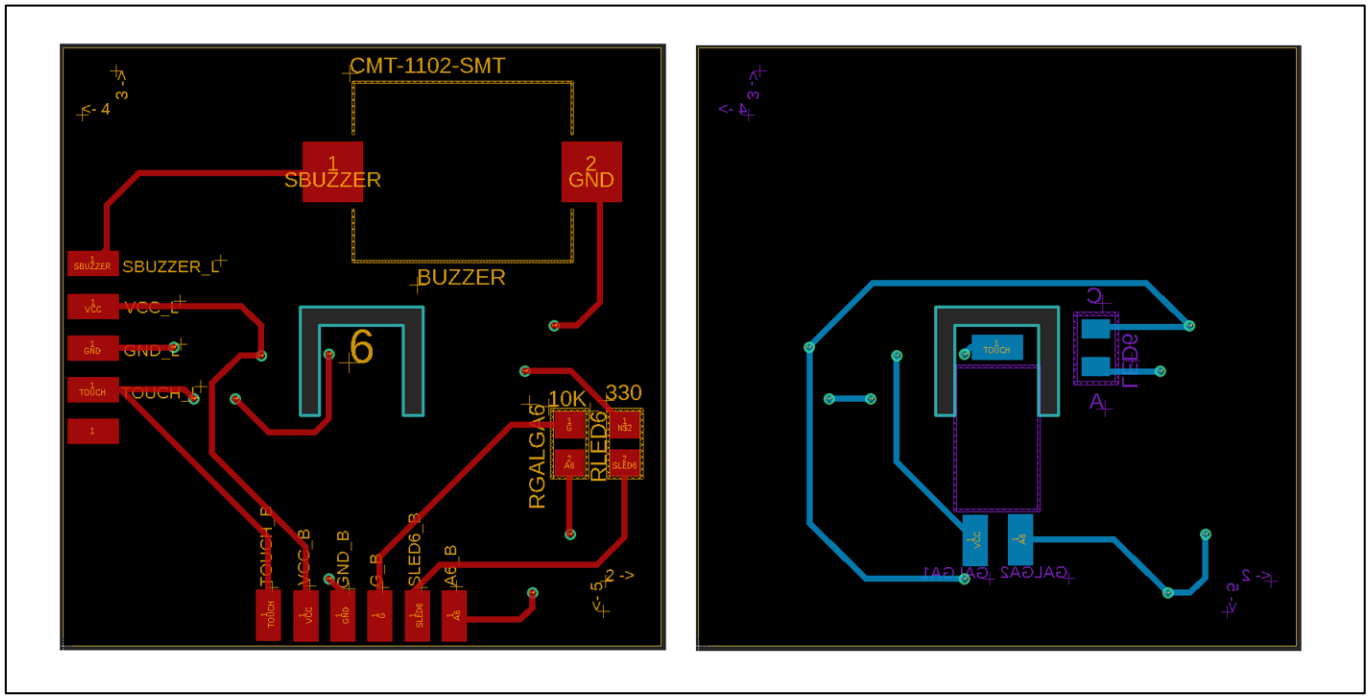


Figura 75. PCB de la Cara 6. Capas TOP y BOTTOM.

En el Anexo se muestran unos modelos 3D de las PCB diseñadas, como herramienta visual que proporciona una visión orientativa del resultado final, pudiendo observar los modelos 3D de cada una de las caras, tanto desde la perspectiva de la capa TOP como de la BOTTOM.

Capítulo 9:
Diseños 3D

9.01 Carcasa de los SmartCubes

9.02 Plataforma de Carga de los SmartCubes

Capítulo 9: Diseños 3D

En esta sección se realizarán los diseños 3D de la carcasa de los SmartCubes y la plataforma de carga. Todas las medidas de los diseños que se verán a continuación, se muestran con su valor nominal, es decir, medidas exactas sin tener en cuenta las holguras entre piezas. Posteriormente, en el software de fabricación correspondiente, se aplicará un escalado en función del método de fabricación escogido. (Algunas partes de los diseños donde no es posible realizar este escalado, presentan ya en los diseños un cierto margen que permita el encaje).

9.01 Carcasa de los SmartCubes

Es necesario diseñar una carcasa que evite que el montaje PCB este directamente expuesto al niño durante las pruebas. Para ello, se diseñará una carcasa cúbica con un orificio para el conector de carga, el cual se encontrará oculto con una tapa. Dicha tapa tendrá un espesor menor al de la carcasa para evitar que el niño puede acceder a ella, la idea es que esté fabricada de un material flexible que permita que vaya encajada en el hueco y n se salga.

Por motivo de seguridad del niño, se realizan empalmes en las aristas del cubo, evitando zonas punzantes. Puede verse los el diseño de la carcasa en la Figura 76.

Por la parte interior de las caras de la carcasa tiene unos salientes, cuya función es la de asegurar el contacto con las pestañas de las caras sobre la que están adheridas las galgas extensiométricas (Figura 77).

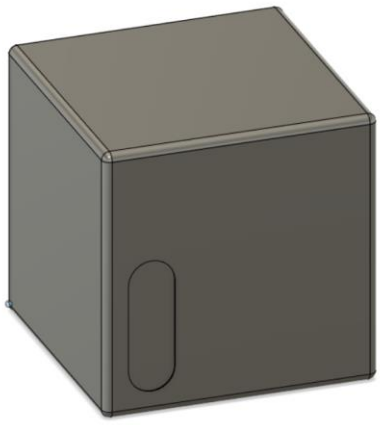


Figura 76. Diseño de la carcasa de los SmartCubes (vista al orificio del conector de carga)

Figura 77. Vista interior de la carcasa de los SmartCubes (salientes de las caras)

Para más detalle revisar los planos en el Anexo XXII.

9.02 Plataforma de Carga de los SmartCubes

En este apartado se verá en detalle el diseño de la plataforma de carga de los SmartCubes, la cual podemos ver en la Figura 78. En su interior se encontrará la parte externa del sistema de carga de las baterías como se explicó en anteriores apartados (conector magnético, módulo de carga y *power bank*).

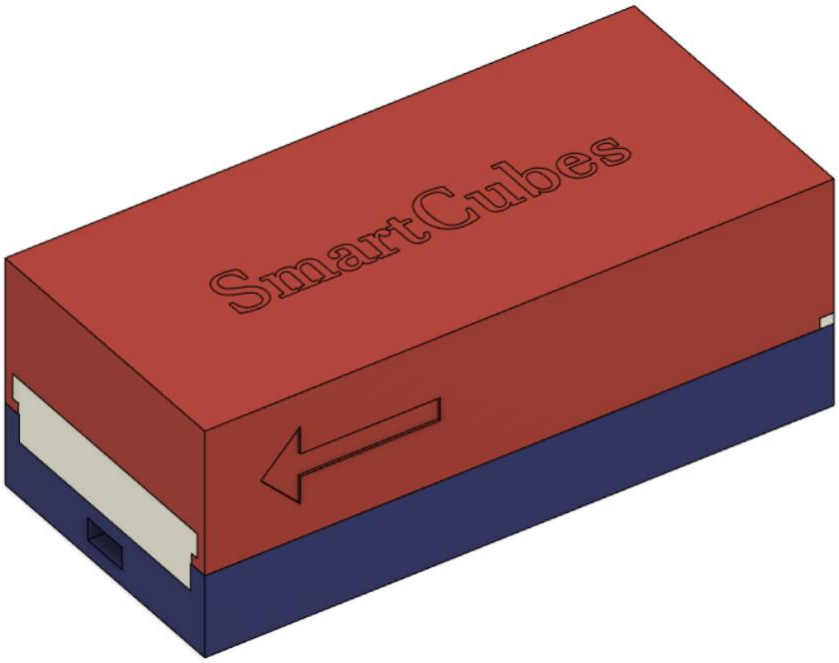


Figura 78. Diseño 3D de la Plataforma de Carga

El diseño de la plataforma está dividido en varias capas como se verá más adelante, la primera de las capas que se puede observar, es la tapa de la plataforma donde se cargarán/guardarán los SmartCubes. Dicha tapa está diseñada para retirarla en formato de corredera, como puede apreciarse en la Figura 79.

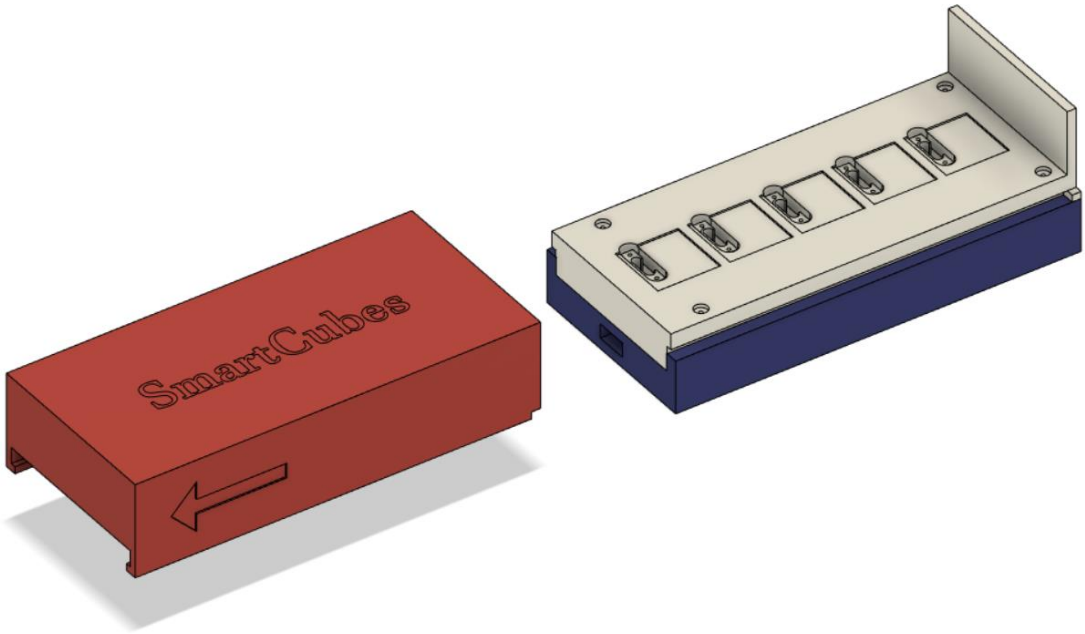


Figura 79. Diseño 3D de la plataforma de carga (Tapa retirada)

Una vez retirada la tapa, en la Figura 80 ya se puede apreciar el primer nivel de la plataforma, donde se situarán los SmartCubes. En cada uno de los espacios cuadrangulares puede apreciarse un hueco, donde se situará el conector magnético que conectará con el conector complementario dentro del SmartCube.

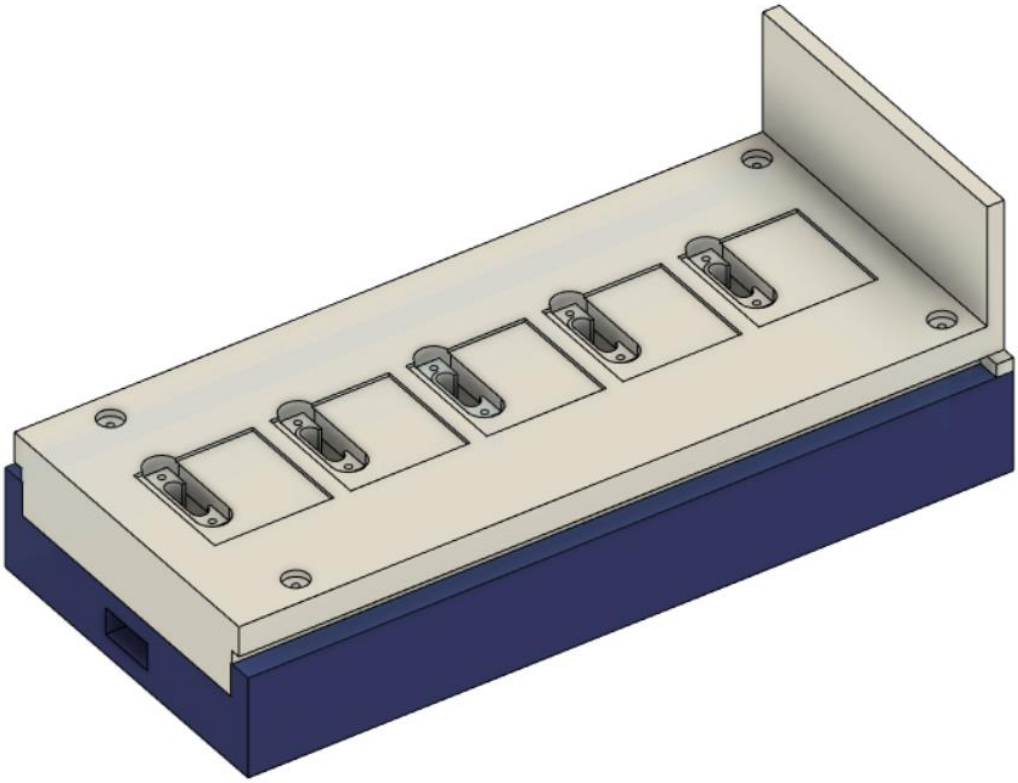


Figura 80. Diseño 3D de la Plataforma de Carga (Localización de los SmartCubes)

Justo debajo se sitúa la capa donde se encuentran los módulos de carga como puede observarse en la Figura 81, donde se puede ver el espacio reservado a estos módulos en los huecos rectangulares. Un poco encima de cada módulo, se encuentran las estructuras sobre las que, como se ha mencionado antes, irá situado el conector magnético. Por último, en la parte izquierda de esta capa, se puede observar un hueco por el que pasará el cable que alimentará los módulos de carga a través de la *power bank*.

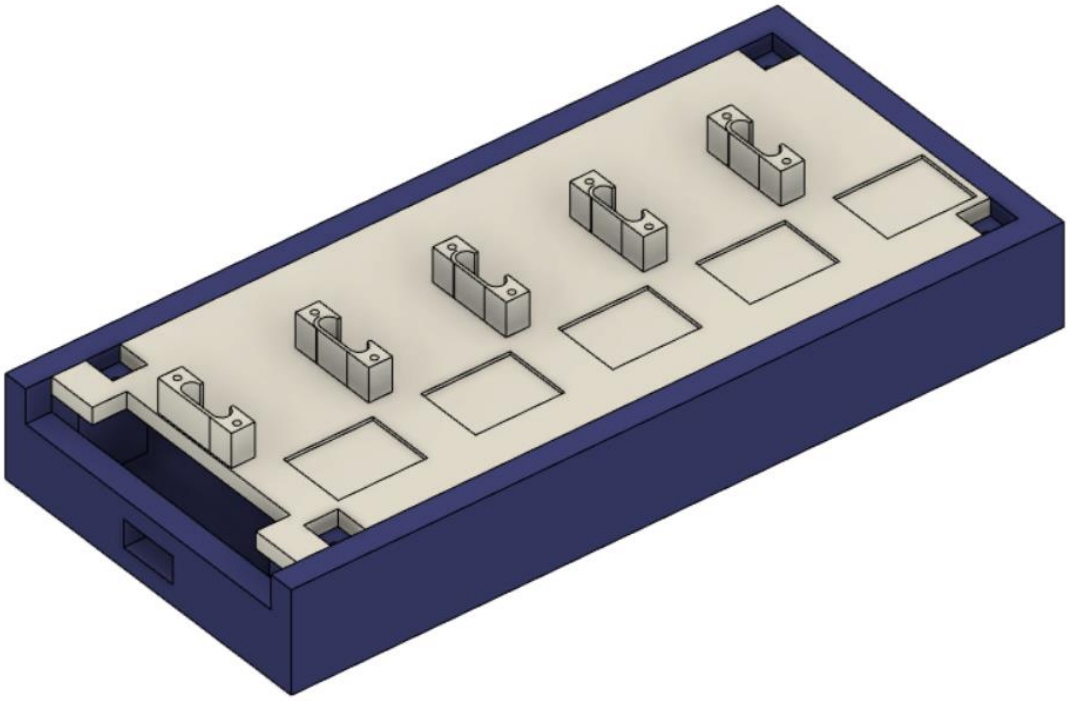


Figura 81. Diseño 3D de la Plataforma de Carga (Localización de los módulos de carga)

En la capa inferior se encuentra la *power bank* como podemos ver en la Figura 82, la *power bank* diseñada es orientativa ya que no se ha empleado ninguna medida concreta, sino que, se han elegido unas medidas basadas en las dimensiones promedio que puede llegar a alcanzar una *power bank* relativamente grande (10000mA o superior).

En la parte izquierda, se puede observar una apertura por donde puede pasar el cable USB para cargar la *power bank*.

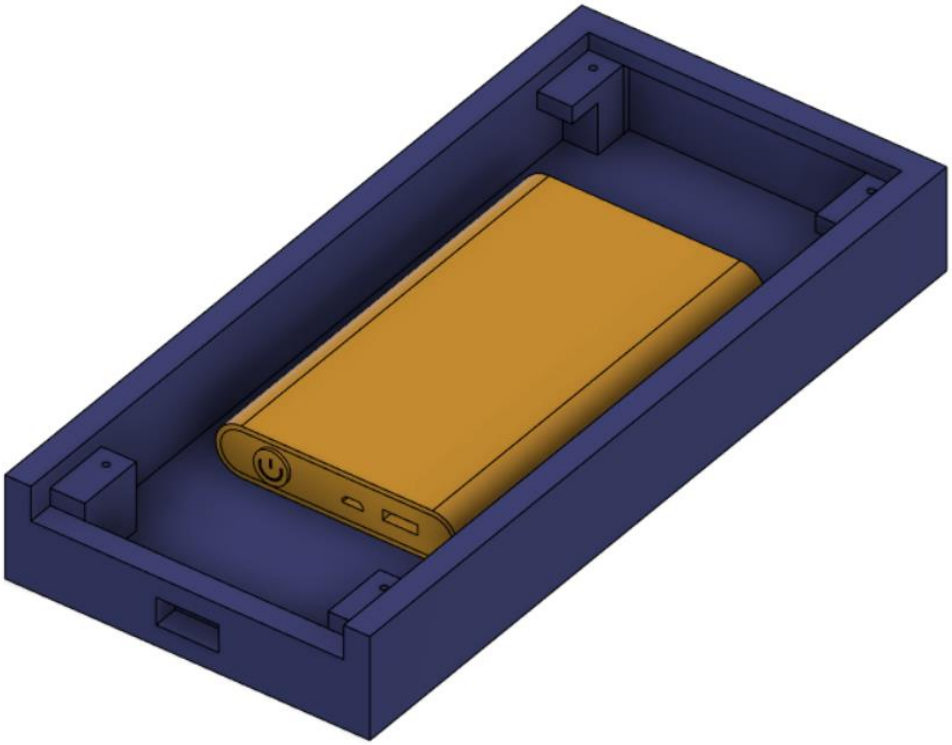


Figura 82. Diseño 3D de la Plataforma de Carga (Localización de la Power Bank)

Para garantizar la unión de todo el conjunto, se atornillan todas las capas por cuatro puntos, no obstante, para no tener que desatornillar todo si se quiere sacar la *power bank*, se ha diseñado una apertura en la parte trasera, como puede verse en la Figura 83.

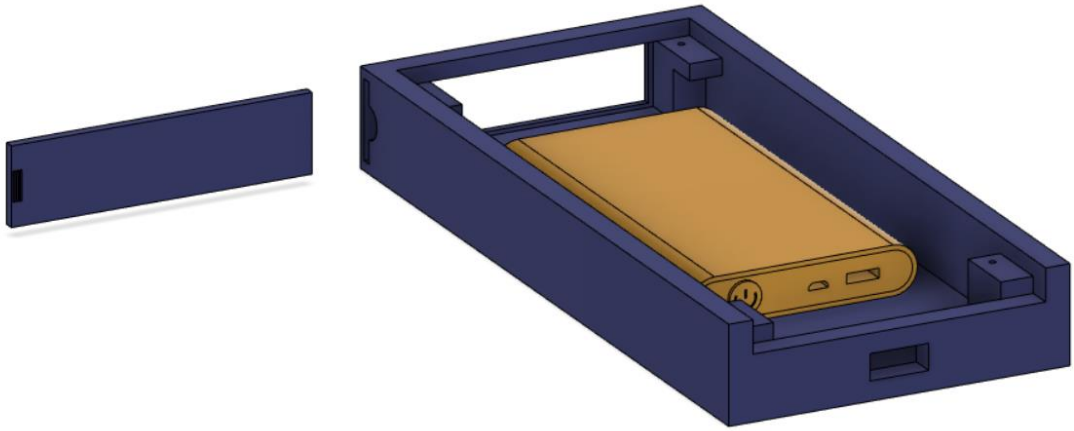


Figura 83. Diseño 3D de la Plataforma de Carga (Apertura para la batería)

Pueden verse los planos de todas las piezas de la plataforma de carga en el Anexo XXII.

Capítulo 10:
Montaje de los SmartCubes y Software Final

Capítulo 10: Montaje de los SmartCubes y Software Final

En este apartado se tratará el montaje de los SmartCubes y la realización un conjunto de pruebas para verificar su funcionamiento. Cabe destacar que en el montaje se realizará a continuación, las PCBs empleadas, no cuentan con el diseño exacto que se ha mostrado en apartados anteriores. Esta diferencia de diseño, se debe a las numerosas modificaciones efectuadas a lo largo del proyecto hasta llegar al diseño final que se muestra en el documento, y a la urgencia de la fabricación de las PCBs debido al tiempo limitado para finalización del proyecto. Por este motivo, se realizarán una serie de arreglos en el montaje para completar el diseño.

A continuación, en la Figura 84 se puede observar las PCBs con forma de cubo desglosado, apreciándose la conexión entre cubos a través de las almohadillas de soldadura y los conectores.

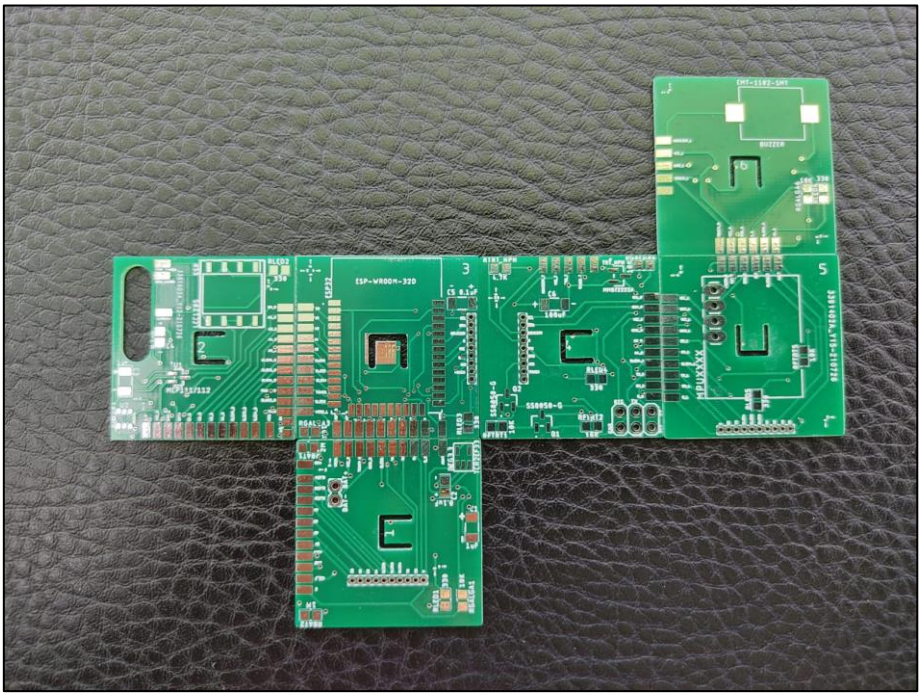


Figura 84. PCBs de un SmartCube colocadas como un cubo desplegado.

De las diferencias entre el diseño final y las PCBs empleadas en este montaje, se pueden destacar el cambio de 3 de los GPIOs del ESP-WROOM-32 (cara 3), la implementación del condensador y la resistencia que van al *Enable* en el sistema de auto programación (cara 4), el reposicionamiento de las resistencias de galga de las caras 1 y 2 ya que, junto con la resistencia de galga de cara 3 existe riesgo de cortocircuito, y el cambio del

regulador de tensión que se empleaba hasta ahora. Los motivos de estos cambios se explicarán a lo largo de este apartado.

A continuación, se sueldan los componentes a las PCBs y se monta el cubo, el resultado final puede apreciarse en la Figura 85 y Figura 86.

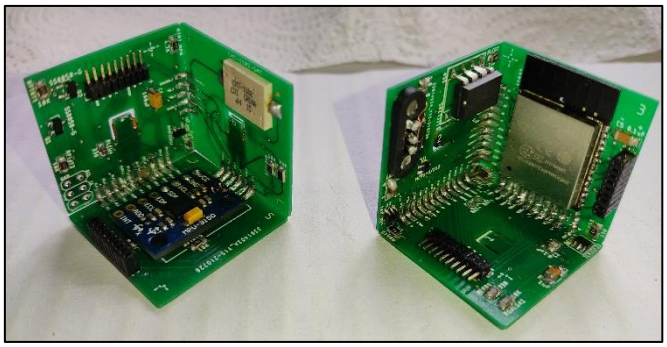


Figura 86. SmartCube (abierto)

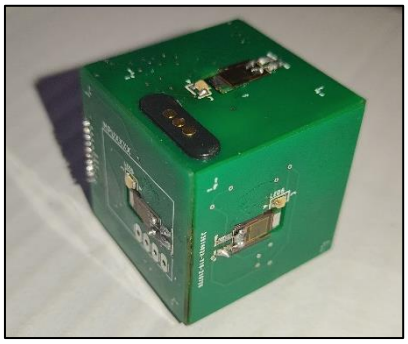


Figura 85. SmartCube (cerrado)

Al realizar el montaje de los componentes, como se ha mencionado antes, surge un problema con la proximidad entre la resistencia de la galga 2, la de la galga 3 y una de las galgas del divisor para la medición de la batería, una vez se sueldan las caras. Este motivo sumado al difícil aprovechamiento del sistema de galgas en la cara 3 debido al ESP-WROOM-32, y al error de diseño de los GPIOs mencionados anteriormente, hacen que para este primer prototipo del cubo se prescinda de los sistemas de galgas de las caras 1 y 3, y el sistema de medición de la batería. No obstante, la ausencia de estas medidas no disminuye el interés de la prueba ya que, la verificación de funcionamiento de la medición de entradas analógicas múltiples se sigue pudendo comprobar con las 4 galgas restantes.

Una vez finalizado en montaje del cubo, se procede a cargar un boceto en el microcontrolador para verificar que montaje es correcto. El resultado de esta primera carga es fallido, no es posible establecer conexión serie-usb con el dispositivo, en este momento comienza un proceso de búsqueda de errores en el montaje.

Primero ya que el problema se encuentra en la carga de código, se revisa que ninguno de los pines que intervienen en el arranque este a un nivel lógico erróneo (especialmente IO2 → LOW, IO12 → LOW, IO15 → HIGH). Una vez comprobados estos pines, se procede a comprobar las soldaduras y los posibles cortocircuitos que pudiesen existir. Puesto que tras revisar el cubo montado no se alcanza una solución, se procede a desmontar las caras y analizar la cara 3 (cara donde se sitúa el microcontrolador) por separado. Con la cara 3 totalmente vacía (a excepción del microcontrolador) se procede a cargar un boceto con

ayuda de un circuito de programación externo (Figura 87), pero el microcontrolador permanece sin poder establecer conexión, por lo que se puede deducir que, o bien el problema radica en el diseño de la placa de circuito impreso, o muy probablemente se encuentre en la soldadura del microcontrolador.

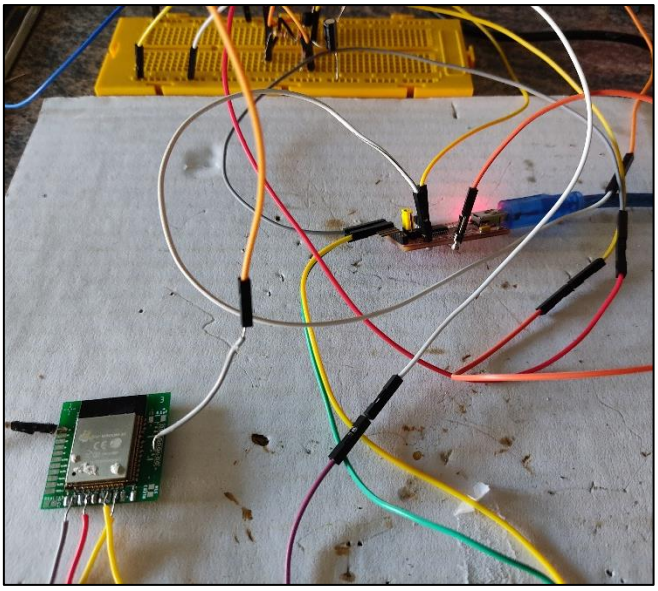


Figura 87. Testeo de la cara 3 (vacía)

Como era de esperar, el fallo se encontraba en la soldadura del ESP-WROOM-32, los terminales de este microcontrolador son especialmente difíciles de soldar, ya que a la mínima se pueden producir cortocircuitos, como en este mismo caso, donde se estaba produciendo un cortocircuito entre Vcc y masa. Una vez solucionado el problema de la soldadura, el microcontrolador fue capaz de establecer conexión y cargar un boceto.

En este punto y para continuar con el proceso de depuración paso a paso, se procede a incorporar los componentes de la cara 3 que habían sido retirados previamente y se comprueba que no hay fallos al cargar de nuevo un boceto. Posteriormente tras verificar su correcto funcionamiento, se procede a montar el cubo completo y cargar de nuevo un boceto. En este momento la carga resulta fallida de nueva, y tras un nuevo proceso de búsqueda se localiza el fallo en algunas soldaduras de los conectores que comunican las caras, que estaban provocando algunos cortocircuitos, tras eliminarlos se logra cargar un boceto correctamente.

Sin embargo, pese a que el sistema es capaz de cargar bocetos exitosamente, al subir un programa BLE o Wifi, no funciona correctamente, en este momento da inicio un proceso de validación del modelo.

Pruebas de validación

Puesto en duda el funcionamiento de la antena, habría sido interesante realizar una prueba de radiofrecuencia empleando una herramienta software que proporciona Espressif [53], pero para dicha prueba se requiere de un *Wifi Tester*, herramienta hardware con la que no se cuenta en este momento.

No siendo posible realizar esta prueba, se procede a flashear la última versión firmware proporcionada por Espressif para los módulos ESP-WROOM-32 [54], para ello se utiliza una herramienta software de flasheo proporcionada de nuevo por Espressif [53], dicha interfaz puede ver en la Figura 88.

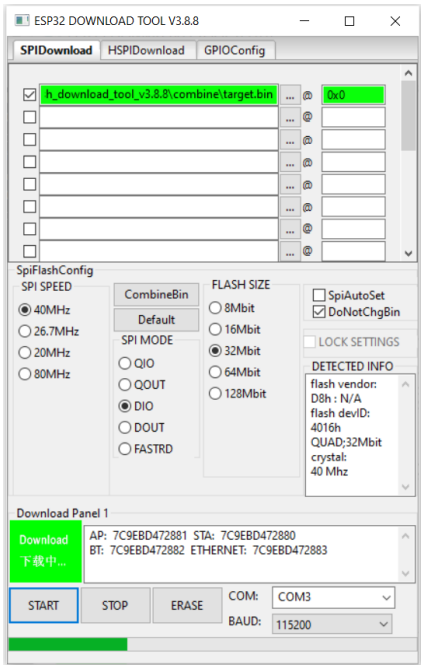


Figura 88. Herramienta software de flasheo de dispositivos Espressif

Con el firmware actualizado, se procede a cargar un boceto BLE desde el IDE de Arduino, pero no se obtiene ningún cambio de comportamiento. Este proceso se realizó con 3 microcontroladores ESP-WROOM-32 diferentes, obteniendo el mismo resultado en todas las pruebas, de esta forma se podría prácticamente descartar un posible fallo de fábrica. Así mismo, la hipótesis de las interferencias electromagnéticas que se planteó en apartado de las pruebas de conjunto queda también invalidada ya que, en este caso, se han realizado las pruebas en un entorno mucho más libre de posibles interferencias con el mismo resultado de funcionamiento.

Tras un proceso de depuración, se obtiene que lo que realmente está sucediendo es que el microcontrolador está alertando de un error y posteriormente realiza un restablecimiento, creando un buble de reseteo del que nunca sale. Dicho error es anunciado por el microcontrolador como *Brownout detector was triggered* y acto seguido se reinicia, como se puede apreciar en la Figura 89.

```
Brownout detector was triggered

ets Jun  8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
```

Figura 89. Error Brownout detector was triggered

Este error como su propio nombre indica es un detector de apagones, es decir, cuando la tensión de alimentación se vuelve inestable o experimenta una cierta variación puntual, el microcontrolador proporciona este error y se reinicia. Por lo tanto, el problema reside en la alimentación del circuito y no en la antena, tras investigar acerca de las posibles causas de este fallo se ha podido reducir todo a un único factor, la fuente de alimentación debe proporcionar un nivel de energía tensión y ser capaz de proporcionar mínimo una corriente instantánea de 500 mA.

Los motivos por los que la fuente de alimentación puede estar fallando son muy variados, puede ir desde que casos en los que la fuente de alimentación directamente no cumple los requisitos previamente nombrados, hasta fallos en el hardware como podría ser un cable de alimentación USB defectuoso o de baja calidad en el caso de que se emplee esta vía de alimentación.

Antes de centrar la atención en los posibles motivos de fallo del sistema y buscar las soluciones, es necesario comprender la raíz del problema y porque es necesaria una corriente instantánea de 500 mA. Como ya se ha visto en apartados anteriores, el ESP-WROOM-32 es un microcontrolador con unos consumos relativamente elevados en comparación con otros microcontroladores, pero es posible reducir notablemente este consumo de energía realizando ciertos ajustes de funcionamiento. No obstante, aunque se reduzca el consumo, para ciertas funciones el microcontrolador necesita picos de

corriente puntuales, como es el caso de la inicialización del BLE o el Wifi. En estos instantes, en el consumo del microcontrolador pueden observarse picos de corriente que pueden llegar hasta los 500 mA, este aumento de la corriente de alimentación genera una caída de tensión de la alimentación, y es en este momento cuando el ESP-WROOM-32 detecta la caída de tensión, muestra el error y reinicia el sistema.

Para solventar este problema, se suele emplea un condensador de bulk entre Vcc y masa, generalmente con un valor superior a los 100 uF. La función de este tipo de condensadores como se explicó en apartados anteriores, es la de evitar caídas y picos de tensión en la alimentación, en este caso cuando se produce un pico de corriente, este condensador se comporta como una reserva de energía y compensa la caída de tensión [55], [56], [57], [58] y [59].

En este caso concreto, como fuente de alimentación se utiliza una batería de Li-Po de 250 mAh, de una celda (3.7 V) y un coeficiente de descarga de 1C, lo que se traduce en una corriente de descarga máxima de 250 mAh. En principio, esta fuente de alimentación no sería apta para este sistema ya que la corriente máxima solo alcanzaría la mitad de mínima recomendada, sin embargo, se han realizado pruebas BLE con un módulo comercial ESP32 junto a esta batería, obteniendo un resultado satisfactorio. No obstante, sería muy recomendado implementar una batería con un coeficiente de descarga mínimo de 2C o ampliar la capacidad hasta los 500 mAh.

En segundo fallo en el sistema, se encontraría en el regulador de tensión escogido. Este regulador posee una corriente de salida máxima de 150 mAh, por lo que es lógico que nuestro sistema sea incapaz de inicializar las funciones de BLE o Wifi. Inicialmente al escoger un regulador de tensión para el sistema, se tuvieron en cuenta dos características principalmente, una caída de tensión pequeña alrededor del centenar de milivoltios y la corriente de salida en torno al amperio, no obstante, los reguladores elegidos únicamente se comercializaban al por mayor, por lo que al escoger una alternativa de remplazo se cometió el error de no revisar el parámetro de la corriente de salida.

Localizadas las posibles causas del problema se procede a realizar una prueba para comprobar la hipótesis, para ello, se monta el siguiente circuito: un adaptador para fuentes de alimentación que puede proporcionar una tensión entre 3 y 12 V con una corriente máxima de 1 amperio, un módulo de fuente de alimentación MB102 que proporciona una tensión regulada a 5V o 3.3V, el regulador de tensión de 3.3V AMS1117, el circuito de

auto programación, el convertidor Serie-USB, y la placa de desarrollo casera basada en el ESP-WROOM-32.

Pese a que el módulo de fuente de alimentación puede proporcionar directamente 3.3V, alimentaremos con 5V y emplearemos un regulador este regulador de tensión para simular el circuito que se emplearía en el cubo. El módulo de fuente de alimentación puede proporcionar una corriente de salida de hasta 700 mA y el regulador de tensión AMS1117 una de 800 mA, por lo que los requisitos de alimentación se cumplen.

Al realizar una primera prueba con este circuito, salta de nuevo el detector de apagones del ESP-WROOM-32, este comportamiento es el esperado ya que el módulo de fuente de alimentación empleado, no es una fuente de alimentación de buena calidad que sea capaz de controlar estas caídas de tensión transitorias. Se encuentran dos posibles soluciones a este problema, una primera solución que se podría calificar de "solución real" y una segunda a la que se podría llamar "solución parche".

La "solución real" consiste en solucionar el problema de la caída de tensión vía hardware, mediante la implementación de un condensador bulk de 220 uF, como se comentó anteriormente. En sistema completo se puede ver a continuación en la Figura 90.

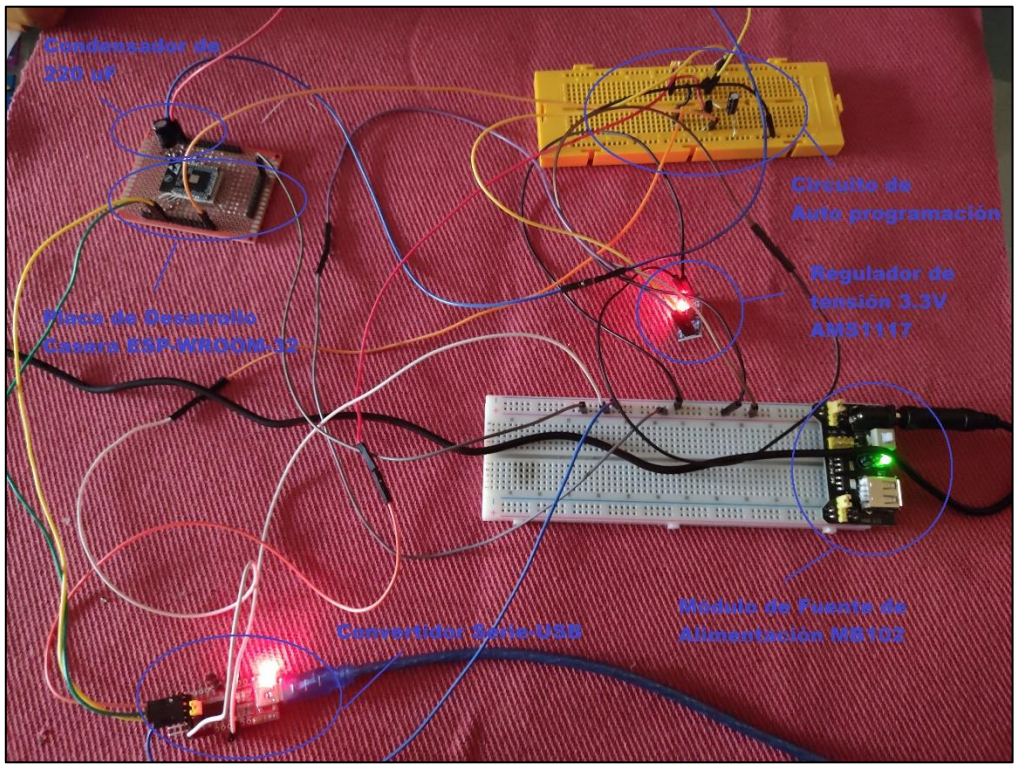


Figura 90. Prueba de fuente de alimentación estable

La “solución parche” recibe este nombre porque realmente no se compensa la caída de tensión vía hardware como en el caso anterior, si no que por medio de software se desactiva el detector de apagones en el momento en que se inicializa cualquier sistema de radio y se habilita nuevamente una vez iniciado. Puesto que el microcontrolador es capaz de funcionar con voltaje superior a 2.3 V, aunque se produzca una pequeña caída de tensión el sistema puede inicializar la radio sin problemas, una vez terminado el proceso de inicialización se reactiva el detector por si hubiera algún problema de alimentación durante la ejecución del programa.

Ambas soluciones funcionan correctamente, y al cargar un boceto BLE el programa se ejecuta con normalidad. No obstante, siempre que se pueda se optará por la solución vía hardware, ya que elimina esa caída de tensión.

A continuación, se procede a realizar el montaje equivalente en el cubo, se sustituye el regulador de tensión por el AMS1117 de 3.3V, puesto que los reguladores no poseen el mismo tipo de encapsulado ni dimensiones, se incorpora al sistema de la forma más limpia posible, aprovechando las conexiones del anterior regulador. Con este regulador y la tensión de la batería en 4V, la tensión de alimentación es 3V. Esto es debido a que este regulador presenta una caída de tensión mucho más elevada que el anterior, en un futuro habrá que incorporara un regulador de tensión de 3.3V con una caída de tensión más reducida que éste, pero para realizar las pruebas del prototipo esta alternativa improvisada es aceptable. Sin embargo, aunque el microcontrolador puede funcionar a 3V, para evitar problemas con el detector de apagones durante las pruebas, se optará por conectar la alimentación del cubo directamente a la fuente de 5V. Para asegurar una buena conexión con la fuente de alimentación, se sueldan unos cables a los orificios del cubo, y para asegurar una buena comunicación durante la programación se hace lo mismo con los terminales de programación, puede ver en la Figura 91.

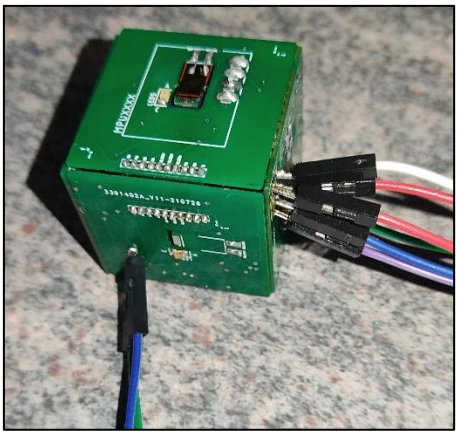


Figura 91. SmartCube con los orificios de alimentación y programación soldados

Con el cubo ya listo, se procede a cargar un boceto BLE para comprobar su correcto funcionamiento. Pero en este caso a diferencia de con la placa de desarrollo casera donde todo funcionaba correctamente, el boceto se carga, pero no funciona. Esta vez, el problema no lo genera el detector de apagones, pero resulta en un comportamiento similar, ya que el microcontrolador se encuentra atrapado en un bucle de restablecimientos, como puede observarse en la Figura 92.

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:10124
load:0x40080400,len:5828
entry 0x400806a8
ets Jun  8 2016 00:22:57

rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:10124
load:0x40080400,len:5828
entry 0x400806a8
ets Jun  8 2016 00:22:57
```

Figura 92. Error de restablecimiento continuo del microcontrolador

Para comprobar si se trata de un fallo en el hardware de programación que se encuentra en la cara 4, se separan ambas mitades del cubo y se conecta el circuito de auto programación externo con la mitad 1 (caras 1, 2 y 3). De esta forma, se probará a cargar un nuevo boceto en el microcontrolador con un circuito cuyo funcionamiento esta verificado, como puede verse en la Figura 93.

Figura 93. Mitad 1 del SmartCube conectada al circuito de auto programación externo

En dicha prueba se obtiene el mismo resultado que con sistema completo, por lo que en principio se descarta el sistema de programación del cubo como origen del problema.

Analizando más en detalle el mensaje de restablecimiento, rtc : 0x3, indica que el software reseteó el núcleo digital, los motivos de reseteo pueden observarse en la librería rtc.h [60];y el boot : 0x13 contiene la información del registro GPIO_STRAP que controla el proceso de arranque, si alguno de los pines involucrados en el arranque estuviera dando algún problema, se vería otra configuración en el registro [14].

Tras investigar acerca de este error concreto, parece ser que tiene más que ver con el entorno de desarrollo y la configuración de carga que con el microcontrolador en sí. En muchos casos, el problema se resuelve al cambiar la frecuencia de CPU de 240 MHz a 160 MHz ya que, algunos microcontroladores ESP-WROOM-32 no son capaces de trabajar a 240 MHz; en otros se resuelve al cambiar la frecuencia de flasheo de 40 MHz a 80 MHz y en otros al cambiar el modo de flasheo (DIO, QIO, DOUT o QOUT), parece ser que depende mucho del fabricante del microcontrolador y los circuitos integrados que utilice en su interior. Tras probar a realizar la carga con las diferentes configuraciones previamente indicadas, no se obtiene un resultado diferente al original. Previamente a estas pruebas, se actualizó el gestor de la placa ESP32 en el IDE de Arduino, ya que en algún caso de los leídos había sido la raíz del problema. De los 3 microcontroladores que se posee actualmente para la realización de las pruebas 1 funciona correctamente (el de la placa de desarrollo casera) y los otros 2 presentan este problema de reseteo (el del cubo y otro para pruebas), estos 2 que presentan este problema son del mismo fabricante y a su vez distinto del que funciona correctamente. Ante estos resultados, se plantea la

posibilidad de que exista algún tipo de incompatibilidad entre estos microcontroladores y el IDE de Arduino, originado por alguno de los circuitos integrados en su interior, se sospecha concretamente del SPI FLASH, ya que el otro chip de su interior es original de Espressif [61], [62] y [63].

Para comprobar la hipótesis, se intercambiará el microcontrolador del cubo con el que se encuentra en la placa de desarrollo casera y se comprobará su funcionamiento en el cubo. Al realizar el cambio y realizar una primera carga en el microcontrolador, el programa se ejecuta con normalidad. En dicho programa de prueba no se hace uso ni del BLE ni el Wifi, por lo que a continuación, se subirá un programa con BLE. Al realizar dicha carga, reaparece el error del detector de apagones pese a contar con un condensador de 22uF, por lo que se procede a incorporar una capacitancia mucho mayor (superior a los 500uF) para asegurar que el problema presenta su origen en la alimentación del sistema. Tras incorporar el nuevo condensador el programa se ejecuta correctamente. De esta forma no se puede negar la posibilidad de una incompatibilidad entre los anteriores microcontroladores ESP-WROOM-32 y el IDE de Arduino, para comprobar esta teoría habría que probar estos microcontroladores en otro entorno de programación.

Software Final

Puesto que la gran mayoría del código se mantiene igual que en la prueba del sistema completo, únicamente se verán las modificaciones con respecto a este.

El primer cambio en el código, es la activación de la medición de las galgas por medio del transistor, para ello se habilitará y deshabilitará un nivel alto en el GPIO 18, de esta forma el transistor permitirá o no el paso de corriente a través de los divisores resistivos que conforman los sistemas de galgas, ver Anexo X.

El siguiente cambio con respecto al anterior código, es la implementación de un juego interactivo como se planteó al inicio de la memoria. El juego iniciará con la iluminación aleatoria de uno de los diodos Leds, el cual permanecerá encendido hasta que se detecte una fuerza que supere el nivel 2 en esa misma cara. En éste mismo instante el *buzzer* emitirá un pitido a modo de estímulo reforzador, incentivando al niño para que juegue con el cubo pulsando sobre la cara iluminada. Tras el pitido, se apaga el diodo Led y se envía por vía BLE la información de tiempo que ha tardado el niño en presionar la cara del cubo desde el instante en que se encendió él. Este juego interactivo está definido en una única función, la cual se incluye en la función loop() dentro del modo de funcionamiento 3, repitiéndose el juego un segundo después de apagarse el diodo Led anterior. Puede verse el código del juego interactivo en el Anexo XI.

Como se ha mencionado en apartados anteriores, uno de los objetivos en el código final el aprovechar el modo *Deep Sleep* para optimizar al máximo la duración de la batería, ahorrando energía en los momentos en los que el niño no esté jugando con los SmartCubes. Por ello, se implementará en el código una forma de detectar los periodos de inactividad en ambos modos de funcionamiento, para posteriormente entrar en modo *Deep Sleep*. Primero se debe establecer el tiempo que debe pasar para considerarse periodo de inactividad, que en esta primera prueba se ha establecido en 15 minutos. Durante el modo normal de funcionamiento, la forma que se ha escogido para medir el periodo de inactividad ha sido mediante la variable *shake*, en cada ciclo se comprobará si la variable *shake* se ha incrementado, de ser así el contador de inactividad permanecerá a 0, y si no lo ha hecho el contador aumentará una unidad, puesto que cada ciclo se repite cada ¼ de segundo, el contador deberá aumentar hasta 3600 para alcanzar los 15 minutos de inactividad. Si el contador supera los 3600, la función booleana que contiene este proceso devolverá un verdadero, de esta forma en el bucle loop() se comprobará cada

ciclo si esta función es verdadera, el sistema entrará en modo *Deep Sleep*. En el caso del juego interactivo, en la función del juego, dentro del bucle *While*, se encuentra un if el cual, si el tiempo transcurrido desde que se encendió el Led supera los 15 minutos, se establecerá un nivel 30 de fuerza para salir del bucle y la variable *inactivity_interactive_game* se pondrá a 1. El cambio de valor de esta variable hará que la función booleana anteriormente mencionada devuelva un verdadero, a partir de aquí el proceso es el mismo que en el modo de funcionamiento normal. Puede ver el código correspondiente en el Anexo XII.

Tras entrar en el modo *Deep Sleep*, el sistema solo saldrá de este modo de suspensión cuando reciba un estímulo por medio del touchpad. En este momento el microcontrolador deberá despertar y retomar la actividad en las mismas condiciones en las que se encontraba antes de entrar en suspensión, y es por esto que se realizan los últimos cambios en el código. Para que el sistema pueda retomar su funcionamiento inmediatamente después de despertar, es necesario conservar cierta información guardándola en la memoria RTC. La primera variable que debe guardarse en la memoria RTC es el modo de funcionamiento ya que, de no hacerlo, el sistema permanecería inactivo (modo 0) al salir del modo *Deep Sleep*. Las siguientes variables que es necesario guardar, son las variables límite (lim_1, lim_2, lim_3, lim_4, lim_5 y lim_6) que guardan el valor de tensión de cada divisor resistivo del sistema de galgas en estado de reposo, si no se guardaran sería necesario realizar el proceso de inicialización de las galgas de nuevo. Por último, hay que guardar también la variable que *shake* ya que, de no hacerlo, se borraría en conteo total de agitaciones de la prueba.

Pruebas Finales de Funcionamiento

Vistas las principales variantes que se incluyen en el código final, se pondrá a prueba en el nuevo formato SmartCube.

Una vez subido el código en el microcontrolador y establecida una conexión vía BLE mediante un dispositivo cliente (smartphone), se podrá observar la siguiente pantalla con los servicios ofrecidos (Figura 94).

Figura 94. Servicios BLE (prueba final SmartCube)

El primer servicio se corresponde con la selección del modo de funcionamiento deseado, el segundo, muestra el nivel de fuerza en las caras del cubo durante el modo normal de funcionamiento, el tercero es el nivel de batería, el cuarto muestra las variables relacionadas con el acelerómetro, el quinto muestra el tiempo transcurrido desde que inició una de las pruebas, y por último, el sexto contiene el tiempo de reacción del niño en el juego interactivo.

Al desplegar cada servicio, se pueden ver las características de cada servicio y las posibles acciones (*Write* en el caso del modo de funcionamiento y la activación las notificaciones en el resto).

El primer aspecto a comprobar serán las galgas extensiométricas, para ello, se ejecutará el programa es el modo normal de funcionamiento. Como se ha explicado en apartado anteriores, antes de entrar en modo normal de funcionamiento (Modo 2), se debe realizar una secuencia de estabilización (Modo 1) y de esta forma mejorar la estabilidad de la señal. La principal diferencia que se encuentra con la prueba parcial y por la cual se hace referencia a la secuencia de estabilización, es que una vez realizado dicho proceso y establecido el valor de tensión en reposo, al entrar en el modo de funcionamiento normal, se produce un salto de tensión en los valores medidos por los ADCs (del orden de las decenas de mV). De esta forma, la secuencia de estabilización resulta inservible y se obtienen valores falsos de fuerza (por lo general negativos). Se desconoce la razón exacta por la que se produce esta variación en la tensión, aunque es muy posible que se deba a una variación en la tensión de alimentación, la cual se ha podido medir con un multímetro, habría sido interesante poder observar la señal de alimentación durante este proceso con un osciloscopio. Puesto que la diferencia de tensión se da entre las medidas tomadas con la función de inicialización de las galgas y la de lectura de las galgas (la forma de obtener la medida de tensión es idéntica en ambas), se opta por realizar una secuencia de estabilización dentro de la función de lectura y así evitar cambiar de función. Dicho proceso de estabilización, consiste en establecer el límite de la tensión en reposo con las 10 primeras medidas. En la Figura 95, puede observarse el resultado con esta modificación sobre el código.

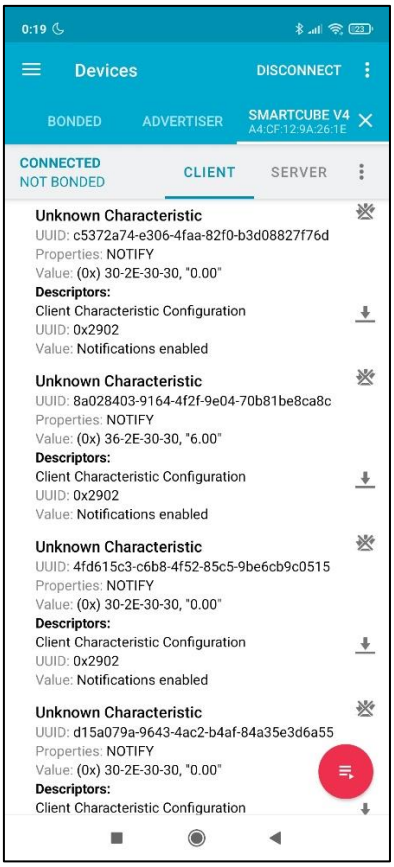


Figura 95. Resultado niveles de fuerza con nueva secuencia de estabilización (prueba final SmartCubes)

Se muestra el resultado de aplicar una fuerza sobre la cara 4 del cubo, la cual el sistema detecta como fuerza de nivel 6 mientras que el resto de caras permanecen a nivel 0. Durante los primeros instantes, el sistema es más inestable y se vuelve más robusto conforme pasa el tiempo, esto se debe al filtro de mediana móvil. En este caso el filtro tiene en cuenta las últimas 7 medidas, conforme se aumenta la capacidad de memoria del sistema se introduce un retardo en la señal, por lo que se debe buscar un equilibrio. Por lo general, la variación del nivel de fuerza en reposo no supera las 3 unidades, además se elimina por código los niveles negativos de fuerza y se establece el límite inferior en 0.

Otro aspecto a destacar es la diferencia del material de las PCBs. La PCB utilizada para realizar la prueba parcial era más fina y de un material más flexible que las PCBs utilizadas en los SmartCubes. Por lo que, la escala de niveles de fuerza obtenida en esta prueba, ha sido menor que en la prueba parcial, siendo 11 el máximo nivel de fuerza visualizado en las pruebas. Puesto que el material de las PCB más difícil de deformar, podría ser difícil percibir diferentes niveles de fuerza en una prueba con niños, por ello,

se debe valorar disminuir el espesor de la PCB en futuros diseños o cambiar el material de la PCB por uno más flexible.

A continuación, se procede a verificar el funcionamiento del acelerómetro en el cuarto servicio. Al activar las notificaciones e introducir el Modo 2 de funcionamiento, se obtienen unos resultados inesperados, como puede observarse en la Figura 96.

Figura 96. Resultados inesperados en las medidas del acelerómetro (pruebas finales SmartCubes)

Los resultados obtenidos muestran medidas de aceleración en torno a los 8.14 m/s2 y de velocidad media de 1 m/s en estado de reposo, mientras que, la variable de las agitaciones funciona correctamente ya que se calcula con una diferencia entre aceleraciones. Se sospecha que el causante de estos resultados podría ser un campo magnético generado por los imanes de neodimio del conector. Por ello, se procede a realizar un proceso de calibración del acelerómetro en el que se le introducen unos valores de offset magnético, esta calibración se realiza con un programa proporcionado por la librería utilizada. Finalizado el proceso de calibración, se prueba de nuevo el *software* final obteniendo unos resultamos mucho más razonables como puede observarse en la Figura 97.

Figura 97. Resultado de las medidas del acelerómetro tras la calibración (pruebas finales SmartCubes)

Al probar el juego interactivo (Modo 3), se puede comprobar que el código funciona correctamente, ya que, si el nivel de fuerza en la cara que se ilumina sobrepasa el umbral de fuerza establecido, se apaga el diodo led, se escucha un zumbido y acto seguido se ilumina otro diodo led. El problema durante este modo de funcionamiento, es la dificultad a la hora de calibrar tanto las medidas del ADC como el umbral límite para detectar la fuerza. Lo que se traduce en la detección de fuerzas inexistentes en las caras o la incapacidad de alcanzar el umbral de fuerza.

Capítulo 11: Conclusiones y Trabajo Futuro

Capítulo 11: Conclusiones y Trabajo Futuro

11.01 Conclusiones

Durante el desarrollo de la memoria, se ha podido observar la evolución de la propuesta de nueva versión para los SmartCubes, partiendo del modelo de versiones anteriores hasta la creación un nuevo prototipo funcional.

A parte de las dificultades generales que plantea el proyecto, existe un plus de dificultad debido a la falta de experiencia en algunos campos como, en el caso de los módulos ESP32 y los dispositivos Iot en general, o el diseño de circuitos impresos.

Entre las dificultades generales que plantea el proyecto se puede destacar el diseño *hardware* y PCB, donde además del desconocimiento previamente mencionado, el proyecto requiere de un diseño hardware optimo que pueda distribuirse en 6 caras de 30x30 mm.

Por otro lado, el diseño *software* requerido en este proyecto se centra en la recolección de datos a través de sensores, su posterior tratamiento y envío a un dispositivo receptor. Aunque puede parecer que el diseño software no representa especial dificultad debido a la cantidad de información disponible en internet, el uso del microcontrolador ESP-WROOM-32, implica la realización de un estudio y periodo de pruebas de las funciones que integra, y de esta forma, maximizar el potencial del proyecto. Además, hay que destacar los errores que se han debido tratar al probar el *software* final en el prototipo.

En términos generales, el proyecto ha logrado su propósito con la implementación de los diferentes sistemas desarrollados a lo largo de la memoria, en un único prototipo con formato cúbico. Por otro lado, hay que destacar que el prototipo no es totalmente funcional ya que, debido a los errores de diseño en el desarrollo de las PCB, no se han podido probar todas las galgas al mismo tiempo, ni la medición de la tensión de la batería. Además, los resultados obtenidos en estas primeras pruebas del prototipo no son completamente satisfactorias y tiene margen de mejora, tanto a nivel de *hardware* como *software*.

En cuanto a la planificación de los tiempos y etapas del proyecto, se valora positivamente el hecho de haber podido finalizar el proyecto con la creación del prototipo de esta nueva

versión y la realización de unas primeras pruebas. No obstante, cabe destacar que no se ha respetado la distribución de tiempos que se planteó en el anteproyecto, debido a que inicialmente no se cuenta con la aparición de tiempos muertos durante el desarrollo del proyecto (debidos a diferentes motivos como se detalla en el Anexo XIII).

Para finalizar este apartado, se ha realizado un presupuesto económico con los principales costes que ha generado el proyecto (verse el Anexo XIV).

11.02 Trabajo Futuro

Como se indica en apartados anteriores, el prototipo construido y con el que se realizan las primeras pruebas de esta nueva versión, se encuentra en fase de desarrollo y cuenta con bastante margen de mejora.

Comenzando por el sistema de galgas extensiométricas, si el sistema se comportara como en las pruebas parciales realizadas, los resultados serían válidos para identificar diferentes niveles de fuerza en las pruebas con niños. Pero los resultados en la prueba final muestran dos problemas principalmente, una superficie más difícil de deformar y una mayor inestabilidad de la señal. Para solventar estos problemas se propone la modificación del espesor, tipo de material y forma del circuito impreso utilizado. Si con estos cambios, el sistema no satisface los requerimientos de precisión, habría que plantear la incorporación de un ADC externo con mayores prestaciones que permita leer las señales analógicas con una mayor precisión, empleando conjuntamente si fuera necesario otra configuración de las galgas extensiométricas con o sin amplificador.

Una vez mejorado el sistema de las galgas, desaparecerían los problemas que impiden el correcto funcionamiento del juego interactivo, pudiéndose realizar pruebas con este modo de funcionamiento. Asimismo, se podría seguir desarrollando más juegos aprovechando la capacidad de interacción directa con el niño que proporciona el sistema de galgas.

Por último, se encuentra necesario el desarrollo de una aplicación móvil personalizada, que permita aprovechar al máximo las propiedades del microcontrolador ESP-WROOM-32 y la tecnología BLE. En términos generales, la aplicación permitiría mantener un trato personalizado con los dispositivos SmartCubes, haciendo posible entre otras cosas, una mejor gestión de los modos de ahorro energéticos del microcontrolador. Además, otra de las ventajas que se encuentra en la creación de una aplicación, es la posibilidad de poder

subir a una base de datos, la información que van proporcionando los SmartCubes a través de internet.

Una vez desarrolladas estas líneas a futuro, se considera que el juguete se encontraría en una fase de desarrollo bastante avanzada, en la que podrían comenzar las pruebas con niños.

BIBLIOGRAFÍA

Capítulo 12: Bibliografía

[1] «Portal Web de WordPress,» [En línea]. Available: https://edgardosilvi.wordpress.com/2016/02/29/acamica-ventajas-y-desventajas-de-arduino/.

[2] «Portal Web de The Green Monkey,» [En línea]. Available: https://www.thegreenmonkey.es/barriodesalamanca/ventajas-de-arduino/.

[3] «Portal Web de El Cacharreo,» [En línea]. Available: http://blog.elcacharreo.com/2015/06/04/afaq-por-que-se-resetea-arduino-cuando-nos-conectamos-al-puerto-serie/.

[4] «Portal Web de Geek Factory,» [En línea]. Available: https://www.geekfactory.mx/tutoriales/tutoriales-arduino/programacion-inalambrica-de-arduino-por-bluetooth/.

[5] «Portal Web de Elecrow,» [En línea]. Available: http://wiki.amperka.ru/_media/%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%8B:troyka-ble:hm-10_datasheet.pdf.

[6] «Portal Web de Laird Connectivity,» [En línea]. Available: https://www.lairdconnect.com/support/faqs/how-can-i-connect-bluetooth-low-energy-device-pc.

[7] «Portal Web de Adafruit,» [En línea]. Available: https://www.adafruit.com/product/2479.

[8] «Portal Web de Seeed Studio,» [En línea]. Available: https://files.seeedstudio.com/wiki/Grove-BLE-dual_model-v1.0/res/Bluetooth_HM-13_en.pdf.

[9] «Portal Web de Espressif,» [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[10] «Portal Web de RANDOM NERD TUTORIALS,» [En línea]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/.

[11] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/esp-idf/issues/1096.

[12] «Portal Web de ESP32,» [En línea]. Available: https://www.esp32.com/viewtopic.php?t=14087.

[13] «Portal Web de DeepBlue,» [En línea]. Available: https://deepbluembedded.com/esp32-pwm-tutorial-examples-analogwrite-arduino/.

[14] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/esptool/wiki/ESP32-Boot-Mode-Selection.

[15] «Portal Web de Espressif,» [En línea]. Available: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf.

[16] «Portal Web de Electrónica Gurú,» [En línea]. Available: https://electronica.guru/questions/11193/desacoplar-el-capacitor-y-el-capacitor-bulk.

[17] «Portal Web de Invensense,» [En línea]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf.

[18] «Portal Web de Safe Load Testing,» [En línea]. Available: https://www.safeloadtesting.com/3-axis-accelerometer-vs-6-axis-gyro-what-is-the-difference/.

[19] «Portal Web de Elecrow,» [En línea]. Available: https://www.elecrow.com/download/Coding%20System%20of%20Strain%20Gauges-AGS-TECH%20Version.pdf.

[20] «Portal Web de CUI Devices,» [En línea]. Available: https://www.cuidevices.com/product/resource/cmt-1102-smt-tr.pdf.

[21] «Portal web de Mouser,» [En línea]. Available: https://www.mouser.es/datasheet/2/240/Lcc120-1547908.pdf.

[22] «Portal Web de Mobus,» [En línea]. Available: https://mobus.es/blog/que-es-una-bateria-lipo/.

[23] «Potal Web de Baterías de Litio,» [En línea]. Available: https://www.bateriasdelitio.net/?p=6#:~:text=Las%20bater%C3%ADas%20recargables%20con%20base,6%20y%203%2C7%20voltios..

[24] «Portal Web de Sparkfun,» [En línea]. Available: https://www.sparkfun.com/datasheets/Batteries/UnionBattery-1000mAh.pdf.

[25] «Portal Web de DigiKey,» [En línea]. Available: https://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604.

[26] «Portal Web de Digi-Key,» [En línea]. Available: https://www.digikey.es/product-detail/es/toshiba-semiconductor-and-storage/TCR2EF33-LM-CT/TCR2EF33LM-CTCT-ND/4503328.

[27] «Portal Web de Uelectronics,» [En línea]. Available: https://uelectronics.com/producto/tp4056-cargador-de-baterias-litio-lipo-5v-1a-proteccion-dual/.

[28] «Portal Web de Espressif,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html#adc-api-reference-adc-calibration.

[29] «Portal Web de Luis Llamas,» [En línea]. Available: https://www.luisllamas.es/reducir-ruido-sensores-arduino-muestreo-multiple/.

[30] «Portal Web de Luis LLamas,» [En línea]. Available: https://www.luisllamas.es/arduino-filtro-mediana-rapido/.

[31] «Portal web de Digikey,» [En línea]. Available: https://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604.

[32] «Portal Web de Microchip,» [En línea]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/20001889F.pdf.

[33] «Portal Web de Diodes,» [En línea]. Available: https://www.diodes.com/assets/Datasheets/AP2281.pdf.

[34] «Portal Web de Tech TutorialsX,» [En línea]. Available: https://techtutorialsx.com/2017/06/05/esp-wroom-32-uploading-a-program-with-arduino-ide/.

[35] «Portal Web de Luis LLamas,» [En línea]. Available: https://www.luisllamas.es/como-programar-el-esp8266-por-wifi-con-arduino-ota/.

[36] «Portal Web de Random Nerd Tutorials,» [En línea]. Available: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/.

[37] «Portal Web de Smart Lighting,» [En línea]. Available: https://smart-lighting.es/bluetooth-low-energy-introduccion-la-tecnologia/.

[38] «Portal Web de Instructables,» [En línea]. Available: https://www.instructables.com/ESP32-Bluetooth-Low-Energy/.

[39] «Portal Web de GUID Generator,» [En línea]. Available: https://www.guidgenerator.com/online-guid-generator.aspx.

[40] «Portal web de Last Minute Engineers,» [En línea]. Available: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/.

[41] «Portal web de Mischianti,» [En línea]. Available: https://www.mischianti.org/2021/03/06/esp32-practical-power-saving-manage-wifi-and-cpu-1/.

[42] «Portal Web de Random Nerd Tutorials Lab,» [En línea]. Available: https://rntlab.com/question/how-to-use-esp32-light-sleep/.

[43] «Portal Web de Steph's µLab,» [En línea]. Available: https://m1cr0lab-esp32.github.io/sleep-modes/sleep-modes/.

[44] «Portal Web de Random Nerd Tutorials,» [En línea]. Available: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/.

[45] «Portal Web de Random Nerd Tutorials,» [En línea]. Available: https://randomnerdtutorials.com/esp32-touch-pins-arduino-ide/.

[46] «Portal Web de Bare Conductive,» [En línea]. Available: https://www.bareconductive.com/blogs/blog/what-is-electric-paint-the-composition-and-application-of-conductive-paints.

[47] «Portal Web de GitHub,» [En línea]. Available: https://github.com/nkolban/esp32-snippets/issues/114.

[48] «Portal Web de Github,» [En línea]. Available: https://github.com/nkolban/esp32-snippets/blob/master/cpp_utils/BLEServer.h#L67.

[49] «Portal Web de JLCPCB,» [En línea]. Available: https://jlcpcb.com/capabilities/Capabilities.

[50] «portal Web de Altium,» [En línea]. Available: https://resources.altium.com/es/p/top-5-pcb-design-guidelines-every-pcb-designer-needs-know.

[51] «Portal Web de Altium,» [En línea]. Available: https://resources.altium.com/es/p/understanding-ground-planes-your-two-layer-pcb.

[52] «Portal Web de DigiKey,» [En línea]. Available: https://www.digikey.es/es/resources/conversion-calculators/conversion-calculator-pcb-trace-width.

[53] «Portal Web de Espressif,» [En línea]. Available: https://www.espressif.com/en/support/download/other-tools.

[54] «Portal Web de Espressif,» [En línea]. Available: https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/ESP32_AT_binaries.html.

[55] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/arduino-esp32/issues/863.

[56] «Portal Web de GitHub,» [En línea]. Available: https://github.com/nkolban/esp32-snippets/issues/785.

[57] «Portal Web de GitHub,» [En línea]. Available: https://github.com/nkolban/esp32-snippets/issues/168.

[58] «Portal Web de GitHub,» [En línea]. Available: https://github.com/nkolban/esp32-snippets/issues/168.

[59] «Portal Web de Random Nerd Tutorials,» [En línea]. Available: https://randomnerdtutorials.com/esp32-troubleshooting-guide/.

[60] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/esp-idf/blob/release/v3.0/components/esp32/include/rom/rtc.h#L147.

[61] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/arduino-esp32/issues/3350.

[62] «Portal Web de GitHub,» [En línea]. Available: https://github.com/espressif/arduino-esp32/issues/3461.

[63] «Portal Web de AV Electronics,» [En línea]. Available: https://avelectronics.cc/esp32-rst0x3-sw_resetboot0x13-spi_fast_flash_boot/.

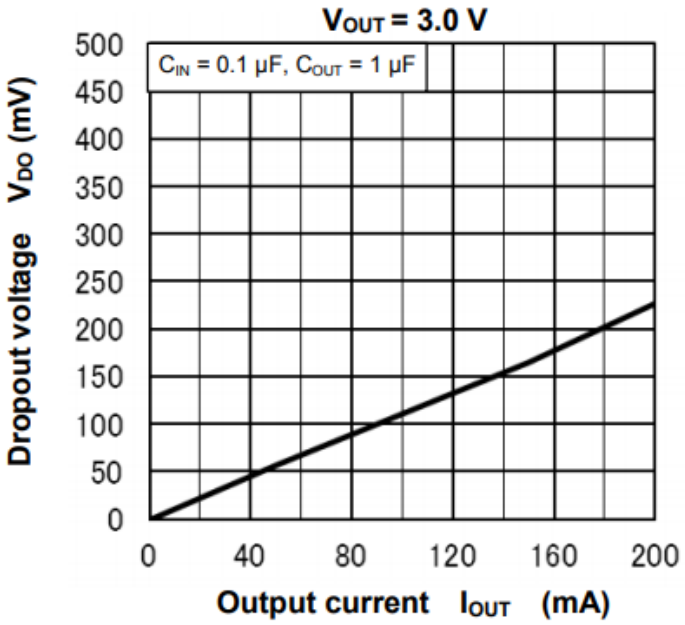
[64] «Portal Web de Jobted,» [En línea]. Available: https://www.jobted.es/salario/ingeniero-industrial#:~:text=El%20salario%20medio%20de%20un%20Ingeniero%20Industrial%20es,comparaci%C3%B3n%20con%20el%20salario%20medio%20anual%20en%20Espa%C3%B1a..

[65] «Portal web de Mouser,» [En línea]. Available: https://www.mouser.es/datasheet/2/240/Lcc120-1547908.pdf.

[66] «Portal Web de InvenSense,» [En línea]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-9150-Datasheet.pdf.

ANEXOS

Anexo I: Gráfica Vdrop (Io) del TCR2EF33 (Fuente: Datasheet)



Anexo II: Tabla de datos de la prueba de autonomía

Horas de referencia	Tiempo de la muestra (horas)	Tensión de la batería Prueba 1 (V)	Tensión de la batería Prueba 2 (V)
0	0	4,1	4,1
	0,16667	4	4
	0,33333	3,96	3,97
	0,5	3,91	3,95
	0,666665	3,88	3,95
	0,833331	3,86	3,94
1	0,999997	3,84	3,94
	1,166663	3,81	3,93
	1,333329	3,79	3,93
	1,499995	3,77	3,92
	1,666661	3,74	3,92
	1,833327	3,72	3,91
2	1,999993	3,7	3,91
	2,166659	3,68	3,9
	2,333325	3,65	3,9
	2,499991	3,6	3,9

	2,666657	3,4	3,9
	2,75	3	3,9
	2,833323		3,9
3	3		3,9
	3,166677		3,9
	3,333354		3,9
	3,500031		3,9
	3,666708		3,9
	3,833385		3,9
4	4,000062		3,9
	4,166739		3,89
	4,333416		3,87
	4,500093		3,85
	4,66677		3,83
	4,833447		3,81
5	5,000124		3,8
	5,166801		3,78
	5,333478		3,77
	5,500155		3,76
	5,666832		3,75
	5,833509		3,73
6	6,000186		3,71
	6,166863		3,69
	6,33354		3,67
	6,500217		3,6
	6,666894		3,4
	6,833571		3
7			

Tabla 5. Datos de las pruebas de autonomía

Anexo III: Código prueba de funcionamiento galgas extensiométricas

```
#include "MedianFilterLib.h"
MedianFilter<float> medianFilter(7);

#include "esp_system.h"
#include "esp_adc_cal.h"
#include "driver/adc.h"

#define ADC_BAT_PIN 34

#define NO_OF_SAMPLES   64          //Multisampling
#define REF_VOLTAGE 1100

#define LIN_COEFF_A_SCALE               65536
#define LIN_COEFF_A_ROUND               (LIN_COEFF_A_SCALE/2)
#define ADC_12_BIT_RES                  4096

uint8_t nivel;
double bat_test;
double lim;
double valor_ini;

esp_adc_cal_characteristics_t *adc_chars = new
esp_adc_cal_characteristics_t;//adc_chars = calloc(1,
sizeof(esp_adc_cal_characteristics_t));

adc1_channel_t get_adc1_chanel(uint8_t pin) {
  adc1_channel_t chan;
  switch (pin) {
    case 32:
      chan = ADC1_CHANNEL_4;
      break;
    case 33:
      chan = ADC1_CHANNEL_5;
      break;
    case 34:
      chan = ADC1_CHANNEL_6;
      break;
    case 35:
      chan = ADC1_CHANNEL_7;
      break;
    case 36:
      chan = ADC1_CHANNEL_0;
      break;
    case 37:
      chan = ADC1_CHANNEL_1;
      break;
    case 38:
      chan = ADC1_CHANNEL_2;
      break;
    case 39:
      chan = ADC1_CHANNEL_3;
      break;
  }
  return chan;
}
```

```
double read_bat_adc_idf() {
  uint32_t adc_reading = 0;

  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
  adc_reading +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_BAT_PIN));
  }
  adc_reading /= NO_OF_SAMPLES;

  //MedianFilter
   adc_reading = medianFilter.AddValue(adc_reading);


  //Convert adc_reading to voltage in mV
  double voltage = esp_adc_cal_raw_to_voltage(adc_reading,
adc_chars);

  return voltage;
}

void adc_setup() {
  adc1_config_width(ADC_WIDTH_12Bit);
  adc1_config_channel_atten(get_adc1_chanel(ADC_BAT_PIN),
ADC_ATTEN_6db );
  esp_adc_cal_value_t val_type =
esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_6db, ADC_WIDTH_12Bit,
REF_VOLTAGE, adc_chars);
}
//
void setup() {
  Serial.begin(115200);
  adc_setup();
//Gauge Stabilization Sequence
  for (int j = 0; j < 10; j++) {
  valor_ini = read_bat_adc_idf();
  if (j==0)
  {
  lim = valor_ini;
  }

  if (valor_ini < lim)
  {
    lim = valor_ini;
  }
  delay(500);
  }
  }


void loop() {
  bat_test = read_bat_adc_idf();
  nivel = lim - bat_test;
  //Serial.print(bat_test);
  //Serial.println ("mV");
  Serial.println(nivel);
  delay(1000);

}
```

Anexo IV: Código prueba de funcionamiento del acelerómetro

```
#include <MPU9250_asukiaaa.h>

#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 21
#define SCL_PIN 22
#endif

#define NO_OF_SAMPLES   64

////////////////////////// Accelerometer Variables //////////////////
MPU9250_asukiaaa mySensor;
float aX, aY, aZ, aSqrt, gX, gY, gZ, mX, mY, mZ, aSqrtMax, AveVel,
shake = 0;
float GyroMeasError = PI * (40.0f / 180.0f);   // gyroscope
measurement error in rads/s (start at 40 deg/s)
float GyroMeasDrift = PI * (0.0f  / 180.0f);   // gyroscope
measurement drift in rad/s/s (start at 0.0 deg/s/s)
float q[4] = {1.0f, 0.0f, 0.0f, 0.0f};    // vector to hold
quaternion
float beta = sqrt(3.0f / 4.0f) * GyroMeasError;   // compute beta
float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;   // compute zeta,
the other free parameter in the Madgwick scheme usually set to a
small or zero value
#define Kp 2.0f * 5.0f // these are the free parameters in the
Mahony filter and fusion scheme, Kp for proportional feedback, Ki
for integral
#define Ki 0.0f
float deltat = 0.0f;        // integration interval for both filter
schemes
float eInt[3] = {0.0f, 0.0f, 0.0f};       // vector to hold integral
error for Mahony method
//////////////////////////////////////////////////////////////////////

////////////////////////// Accelerometer System /////////////////////
void Init_Accelerometer()
{
#ifdef _ESP32_HAL_I2C_H_ // For ESP32
  Wire.begin(SDA_PIN, SCL_PIN);
  mySensor.setWire(&Wire);
#endif

  mySensor.beginAccel();
  mySensor.beginGyro();
  mySensor.beginMag();

  // You can set your own offset for mag values
  // mySensor.magXOffset = -50;
  // mySensor.magYOffset = -55;
  // mySensor.magZOffset = -10;
}
void Get_Accelerometer_Measurements()
{
  float AveVel_p, aSqrt_aux, aSqrt_aux_p;
  unsigned long time_p;
  unsigned long time_n;
  unsigned long time_i = millis();
  bool SK = false;
  int j = 0;
```

```
for (int i = 0; i < NO_OF_SAMPLES; i++) {
  if (mySensor.accelUpdate() == 0) {
    aX = mySensor.accelX();
    aY = mySensor.accelY();
    aZ = mySensor.accelZ();
  }

  if (mySensor.gyroUpdate() == 0) {
    gX = mySensor.gyroX();
    gY = mySensor.gyroY();
    gZ = mySensor.gyroZ();
  }

  if (mySensor.magUpdate() == 0) {
    mX = mySensor.magX();
    mY = mySensor.magY();
    mZ = mySensor.magZ();
  }
  MadgwickQuaternionUpdate(aX, aY, aZ, gX*PI/180.0f, gY*PI/180.0f,
gZ*PI/180.0f,  mX,  mY, mZ);
  //MahonyQuaternionUpdate(aX, aY, aZ, gX*PI/180.0f, gY*PI/180.0f,
gZ*PI/180.0f,  mX,  mY, mZ);
  aSqrt_aux = mySensor.accelSqrt();
  if (j != 0)
  {
    time_n = millis();
    aSqrt += aSqrt_aux;
    AveVel_p = (aSqrt_aux*(time_n - time_p)/1000) + AveVel_p;
    AveVel += AveVel_p;
    if (aSqrt_aux > aSqrtMax){ aSqrtMax = aSqrt_aux;}
    time_p = time_n;
    if ( (aSqrt_aux - aSqrt_aux_p) > 0.27)
    {
      SK = true;
    }
    aSqrt_aux_p = aSqrt_aux;
  }
  if (j == 0)
  {
    time_p = millis();
    aSqrt = aSqrt_aux;
    aSqrt_aux_p = aSqrt_aux;
    aSqrtMax = aSqrt;
    AveVel = aSqrt_aux*(time_p - time_i)/1000;
    AveVel_p = AveVel;
    j = 1;
  }
  }
  aSqrt /= NO_OF_SAMPLES;
  AveVel /= NO_OF_SAMPLES;
  if(SK)shake++;

Serial.println("AccSqrt: " + String(aSqrt));
Serial.println("AveVel: " + String(AveVel));
Serial.println("MaxAccSqrt: " + String(aSqrtMax));
Serial.println("Shakes: " + String(shake));
Serial.println("Time: " + String(millis()/1000) + " segundos");


}
```

```
void MadgwickQuaternionUpdate(float ax, float ay, float az, float
gx, float gy, float gz, float mx, float my, float mz)
        {
            float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   //
short name local variable for readability
            float norm;
            float hx, hy, _2bx, _2bz;
            float s1, s2, s3, s4;
            float qDot1, qDot2, qDot3, qDot4;
//            float gerrx, gerry, gerrz, gbiasx, gbiasy,
gbiasz;        // gyro bias error

            // Auxiliary variables to avoid repeated arithmetic
            float _2q1mx;
            float _2q1my;
            float _2q1mz;
            float _2q2mx;
            float _4bx;
            float _4bz;
            float _2q1 = 2.0f * q1;
            float _2q2 = 2.0f * q2;
            float _2q3 = 2.0f * q3;
            float _2q4 = 2.0f * q4;
            float _2q1q3 = 2.0f * q1 * q3;
            float _2q3q4 = 2.0f * q3 * q4;
            float q1q1 = q1 * q1;
            float q1q2 = q1 * q2;
            float q1q3 = q1 * q3;
            float q1q4 = q1 * q4;
            float q2q2 = q2 * q2;
            float q2q3 = q2 * q3;
            float q2q4 = q2 * q4;
            float q3q3 = q3 * q3;
            float q3q4 = q3 * q4;
            float q4q4 = q4 * q4;

            // Normalise accelerometer measurement
            norm = sqrt(ax * ax + ay * ay + az * az);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f/norm;
            ax *= norm;
            ay *= norm;
            az *= norm;

            // Normalise magnetometer measurement
            norm = sqrt(mx * mx + my * my + mz * mz);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f/norm;
            mx *= norm;
            my *= norm;
            mz *= norm;

            // Reference direction of Earth's magnetic field
            _2q1mx = 2.0f * q1 * mx;
            _2q1my = 2.0f * q1 * my;
            _2q1mz = 2.0f * q1 * mz;
            _2q2mx = 2.0f * q2 * mx;
            hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2
+ _2q2 * my * q3 + _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;
            hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx *
q3 - my * q2q2 + my * q3q3 + _2q3 * mz * q4 - my * q4q4;
```

```
_2bx = sqrt(hx * hx + hy * hy);
            _2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx
* q4 - mz * q2q2 + _2q3 * my * q4 - mz * q3q3 + mz * q4q4;
            _4bx = 2.0f * _2bx;
            _4bz = 2.0f * _2bz;

            // Gradient decent algorithm corrective step
            s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f
* q1q2 + _2q3q4 - ay) - _2bz * q3 * (_2bx * (0.5f - q3q3 - q4q4) +
_2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx *
(q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx *
(q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
            s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f
* q1q2 + _2q3q4 - ay) - 4.0f * q2 * (1.0f - 2.0f * q2q2 - 2.0f *
q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz *
(q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 -
q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q4 - _4bz * q2) *
(_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
            s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f
* q1q2 + _2q3q4 - ay) - 4.0f * q3 * (1.0f - 2.0f * q2q2 - 2.0f *
q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 -
q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) *
(_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q1 -
_4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) -
mz);
            s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f
* q1q2 + _2q3q4 - ay) + (-_4bx * q4 + _2bz * q2) * (_2bx * (0.5f -
q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz *
q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx *
q2 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
            norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 *
s4);    // normalise step magnitude
            norm = 1.0f/norm;
            s1 *= norm;
            s2 *= norm;
            s3 *= norm;
            s4 *= norm;
            // Compute rate of change of quaternion
            qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta *
s1;
            qDot2 = 0.5f * ( q1 * gx + q3 * gz - q4 * gy) - beta *
s2;
            qDot3 = 0.5f * ( q1 * gy - q2 * gz + q4 * gx) - beta *
s3;
            qDot4 = 0.5f * ( q1 * gz + q2 * gy - q3 * gx) - beta *
s4;

            // Integrate to yield quaternion
            q1 += qDot1 * deltat;
            q2 += qDot2 * deltat;
            q3 += qDot3 * deltat;
            q4 += qDot4 * deltat;
            norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 *
q4);    // normalise quaternion
            norm = 1.0f/norm;
            q[0] = q1 * norm;
            q[1] = q2 * norm;
            q[2] = q3 * norm;
            q[3] = q4 * norm;
        }
```

```
// Similar to Madgwick scheme but uses proportional and integral
filtering on the error between estimated reference vectors and
 // measured ones.
            void MahonyQuaternionUpdate(float ax, float ay, float
az, float gx, float gy, float gz, float mx, float my, float mz) {
            float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   //
short name local variable for readability
            float norm;
            float hx, hy, bx, bz;
            float vx, vy, vz, wx, wy, wz;
            float ex, ey, ez;
            float pa, pb, pc;

            // Auxiliary variables to avoid repeated arithmetic
            float q1q1 = q1 * q1;
            float q1q2 = q1 * q2;
            float q1q3 = q1 * q3;
            float q1q4 = q1 * q4;
            float q2q2 = q2 * q2;
            float q2q3 = q2 * q3;
            float q2q4 = q2 * q4;
            float q3q3 = q3 * q3;
            float q3q4 = q3 * q4;
            float q4q4 = q4 * q4;

            // Normalise accelerometer measurement
            norm = sqrt(ax * ax + ay * ay + az * az);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f / norm;        // use reciprocal for
division
            ax *= norm;
            ay *= norm;
            az *= norm;

            // Normalise magnetometer measurement
            norm = sqrt(mx * mx + my * my + mz * mz);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f / norm;        // use reciprocal for
division
            mx *= norm;
            my *= norm;
            mz *= norm;

            // Reference direction of Earth's magnetic field
            hx = 2.0f * mx * (0.5f - q3q3 - q4q4) + 2.0f * my *
(q2q3 - q1q4) + 2.0f * mz * (q2q4 + q1q3);
            hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f * my * (0.5f -
q2q2 - q4q4) + 2.0f * mz * (q3q4 - q1q2);
            bx = sqrt((hx * hx) + (hy * hy));
            bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f * my * (q3q4 +
q1q2) + 2.0f * mz * (0.5f - q2q2 - q3q3);
           // Estimated direction of gravity and magnetic field
            vx = 2.0f * (q2q4 - q1q3);
            vy = 2.0f * (q1q2 + q3q4);
            vz = q1q1 - q2q2 - q3q3 + q4q4;
            wx = 2.0f * bx * (0.5f - q3q3 - q4q4) + 2.0f * bz *
(q2q4 - q1q3);
            wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f * bz * (q1q2 +
q3q4);
            wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f * bz * (0.5f -
q2q2 - q3q3);
```

```
// Error is cross product between estimated direction and measured
direction of gravity
            ex = (ay * vz - az * vy) + (my * wz - mz * wy);
            ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
            ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
            if (Ki > 0.0f)
            {
                eInt[0] += ex;      // accumulate integral error
                eInt[1] += ey;
                eInt[2] += ez;
            }
            else
            {
                eInt[0] = 0.0f;     // prevent integral wind up
                eInt[1] = 0.0f;
                eInt[2] = 0.0f;
            }

            // Apply feedback terms
            gx = gx + Kp * ex + Ki * eInt[0];
            gy = gy + Kp * ey + Ki * eInt[1];
            gz = gz + Kp * ez + Ki * eInt[2];

            // Integrate rate of change of quaternion
            pa = q2;
            pb = q3;
            pc = q4;
            q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz) * (0.5f *
deltat);
            q2 = pa + (q1 * gx + pb * gz - pc * gy) * (0.5f *
deltat);
            q3 = pb + (q1 * gy - pa * gz + pc * gx) * (0.5f *
deltat);
            q4 = pc + (q1 * gz + pa * gy - pb * gx) * (0.5f *
deltat);

            // Normalise quaternion
            norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
            norm = 1.0f / norm;
            q[0] = q1 * norm;
            q[1] = q2 * norm;
            q[2] = q3 * norm;
            q[3] = q4 * norm;

        }
//////////////////////////////////////////////////////////////////////

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("started");

Init_Accelerometer();
}

void loop() {
  Get_Accelerometer_Measurements();
  delay(500);
}
```

Anexo V: Código Prueba de Medición del Estado de la Batería

```
#include "MedianFilterLib.h"
#include "esp_system.h"
#include "esp_adc_cal.h"
#include "driver/adc.h"

#define ADC_BAT_PIN 15

#define NO_OF_SAMPLES   64          //Multisampling
#define REF_VOLTAGE 1100

#define ADC_12_BIT_RES                  4096

MedianFilter<float> medianFilter(7);
double bat_test;

esp_adc_cal_characteristics_t *adc_chars = new
esp_adc_cal_characteristics_t;

adc2_channel_t get_adc2_chanel(uint8_t pin) {
  adc2_channel_t chan;
  switch (pin) {
    case 4:
      chan = ADC2_CHANNEL_0;
      break;
    case 0:
      chan = ADC2_CHANNEL_1;
      break;
    case 2:
      chan = ADC2_CHANNEL_2;
      break;
    case 15:
      chan = ADC2_CHANNEL_3;
      break;
    case 13:
      chan = ADC2_CHANNEL_4;
      break;
    case 12:
      chan = ADC2_CHANNEL_5;
      break;
    case 14:
      chan = ADC2_CHANNEL_6;
      break;
    case 27:
      chan = ADC2_CHANNEL_7;
      break;
    case 25:
      chan = ADC2_CHANNEL_8;
      break;
    case 26:
      chan = ADC2_CHANNEL_9;
      break;
  }
  return chan;
}
```

```
double read_bat_adc_idf() {
  uint32_t adc_reading = 0;
  int read_raw;
  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
        //Check that the ADC2 is not used by the Wifi
   esp_err_t r =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_BAT_PIN),
ADC_WIDTH_12Bit, &read_raw);
    if ( r == ESP_OK ) {
        adc_reading = adc_reading +
read_raw;                       //It's free. Save de value.
    } else if ( r == ESP_ERR_TIMEOUT ) {
        printf("ADC2 used by Wi-
Fi.\n");                         //Wifi is using the ADC2
    }
  }
  adc_reading /= NO_OF_SAMPLES;

  //MedianFilter
   adc_reading = medianFilter.AddValue(adc_reading);


  //Convert adc_reading to voltage in mV
  double voltage = esp_adc_cal_raw_to_voltage(adc_reading,
adc_chars);

  return voltage;
}

void adc_setup() {
  adc2_config_channel_atten(get_adc2_chanel(ADC_BAT_PIN),
ADC_ATTEN_6db );         //Set up the channel and the attenuation
  esp_adc_cal_value_t val_type =
esp_adc_cal_characterize(ADC_UNIT_2, ADC_ATTEN_6db, ADC_WIDTH_12Bit,
REF_VOLTAGE, adc_chars); //Calibration
}
//



void setup() {
  Serial.begin(115200);
  adc_setup();
  }


void loop() {
  bat_test = read_bat_adc_idf() * 2;  //Multiply by 2 due to
resistive divisor
  Serial.print(bat_test);
  Serial.println ("mV");
  delay(1000);

}
```

Anexo VI: Código Prueba de Programación OTA (*Over The Air*)

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* ssid = "MyWifi";
const char* password = "MyPassword";

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");

  pinMode(25, OUTPUT);
  pinMode(26, OUTPUT);

   digitalWrite(25,LOW);
   digitalWrite(26,LOW);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection FaiLed! Rebooting...");
    delay(5000);
    ESP.restart();
  }

  // Port defaults to 3232
  // ArduinoOTA.setPort(3232);

  // Hostname defaults to esp3232-[MAC]
   ArduinoOTA.setHostname("SmartCube");

  // No authentication by default
  //ArduinoOTA.setPassword("admin");

  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  // ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

  ArduinoOTA
    .onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";

      // NOTE: if updating SPIFFS this would be the place to unmount
SPIFFS using SPIFFS.end()
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
```

```
    .onError([](ota_error_t error) {
       Serial.printf("Error[%u]: ", error);
       if (error == OTA_AUTH_ERROR) Serial.println("Auth FaiLed");
       else if (error == OTA_BEGIN_ERROR) Serial.println("Begin
FaiLed");
       else if (error == OTA_CONNECT_ERROR) Serial.println("Connect
FaiLed");
       else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive
FaiLed");
       else if (error == OTA_END_ERROR) Serial.println("End FaiLed");
     });

  ArduinoOTA.begin();

  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {

  digitalWrite(25,HIGH);
  delay(1000);
  digitalWrite(25, LOW);
  delay(1000);
  ArduinoOTA.handle();
}
```

Anexo VII: Código Prueba de Transmisión de Datos por BLE

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer *pServer = NULL;
BLECharacteristic *characteristicTX_1; //Object to send data
BLECharacteristic *characteristicTX_2; //Object to send data

bool deviceConnected = false; //Controls if the device is connected

uint8_t MODE = 3;
float Data1 =0.00;
float Data2 =20.00;
char txString_1[8];
char txString_2[8];

#define SERVICE_UUID   "ab0828b1-198e-4351-b779-901fa0e0371e"
#define CHARACTERISTIC_UUID_RX  "4ac8a682-9736-4e5d-932b-
e9b31405049c"

#define SERVICE_UUID_1   "66865fa9-749f-4ce1-9758-c98286e027d3"
#define CHARACTERISTIC_UUID_TX_1  "0972EF8C-7613-4075-AD52-
756F33D4DA91"

#define SERVICE_UUID_2   "99b7ea8a-190c-4d12-beef-52bce92cb280"
#define CHARACTERISTIC_UUID_TX_2  "ff2e724e-4d71-4700-a4c3-
2502e6c8d403"

//callback to recibe the events from connected devices
class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
 deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
 deviceConnected = false;
    }
};

//callback for characteristic events
class CharacteristicCallbacks: public BLECharacteristicCallbacks {
     void onWrite(BLECharacteristic *characteristic) {
          //Return the pointer to save the actual characteristic
value
          std::string rxValue = characteristic->getValue();
          //Check if the value is not empty
          if (rxValue.length() > 0) {

               for (int i = 0; i < rxValue.length(); i++) {
            Serial.print(rxValue[i]);
               }
               Serial.println();
               if (rxValue.find("1") != -1) {
            MODE = 1;
               }
               else if (rxValue.find("2") != -1) {
            MODE = 2;
               }
```

```
              else if (rxValue.find("3") != -1) {
             MODE = 3;
               }
          }
     }//onWrite
};



void setup() {
    Serial.begin(115200);

    // Create the BLE Device
    BLEDevice::init("SmartCube"); // nome do dispositivo bluetooth

    // Create the BLE Server
    pServer = BLEDevice::createServer(); //cria um BLE server
    pServer->setCallbacks(new ServerCallbacks()); //seta o callback
do server

    // Create the BLE Service
    BLEService *pService = pServer->createService(SERVICE_UUID);
    BLEService *pService_1 = pServer->createService(SERVICE_UUID_1);
    BLEService *pService_2 = pServer->createService(SERVICE_UUID_2);

    // Create a BLE Characteristic to send data
    characteristicTX_1 = pService_1->createCharacteristic(
                      CHARACTERISTIC_UUID_TX_1,
                      BLECharacteristic::PROPERTY_NOTIFY
                    );

    characteristicTX_1->addDescriptor(new BLE2902());

    characteristicTX_2 = pService_2->createCharacteristic(
                      CHARACTERISTIC_UUID_TX_2,
                      BLECharacteristic::PROPERTY_NOTIFY
                    );

    characteristicTX_2->addDescriptor(new BLE2902());
// Create a BLE Characteristic to recibe data
    BLECharacteristic *characteristic = pService-
>createCharacteristic(
                                                    CHARACTERISTIC
_UUID_RX,
                                                    BLECharacteris
tic::PROPERTY_WRITE
                                                  );

    characteristic->setCallbacks(new CharacteristicCallbacks());

    // Start the service
    pService->start();
    pService_1->start();
    pService_2->start();
    // Start advertising
    pServer->getAdvertising()->start();

} //fin setup
```

```
void loop() {
    //if exist any device connected
    if (deviceConnected) {
        if (MODE == 1)
        {

            Data1 = Data1 + 0.01;

            // Convert the float values in arrays
            dtostrf(Data1, 2, 2, txString_1); // float_val,
min_width, digits_after_decimal, char_buffer

            characteristicTX_1->setValue(txString_1); //Set the
value in the characteristic
            characteristicTX_1->notify(); // // Send value to Client
Device
        }
         if (MODE == 2)
        {


            Data2 = Data2 + 0.05;

            // Convert the float values in arrays
            dtostrf(Data2, 2, 2, txString_2); // float_val,
min_width, digits_after_decimal, char_buffer

            characteristicTX_2->setValue(txString_2); //Set the
value in the characteristic
            characteristicTX_2->notify(); //// Send value to Client
Device
        }
        if (MODE == 3)
        {

            Data1 = Data1 + 0.01;
            Data2 = Data2 + 0.05;
            // Convert the float values in arrays
            dtostrf(Data1, 2, 2, txString_1);
            dtostrf(Data2, 2, 2, txString_2); // float_val,
min_width, digits_after_decimal, char_buffer
            characteristicTX_1->setValue(txString_1);  //Set the
value in the characteristic
            characteristicTX_1->notify();// Send value to Client
Device
            characteristicTX_2->setValue(txString_2); //Set the
value in the characteristic
            characteristicTX_2->notify(); // Send value to Client
Device
        }

    }
    delay(1000);
}
```

Anexo VIII: Código Prueba de los Modos de Ahorro de Energía

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#define Threshold 20 /* Greater the value, more the sensitivity */


const char* ssid = "..........";
const char* password = "..........";

RTC_DATA_ATTR int Count = 0;
touch_pad_t touchPin;


void callback(){
  //placeholder callback function
}

// ------------------------------------------------------------------
// Definition of sleep mode properties
// ------------------------------------------------------------------
//                             seconds
//                               v
const uint32_t SLEEP_DURATION = 3*1000000; // s  si* 1000000 =
microsegundos, ya que en la función debemos introducir
microsegundos.

// ------------------------------------------------------------------


int valor=0;


void setup(){
  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor

  setModemSleep();
}
// ------------------------------------------------------------------
// Deep Sleep Mode
// ------------------------------------------------------------------
void deepSleep() {
    //Setup interrupt on Touch Pad 3 (GPIO15)
    touchAttachInterrupt(T6, callback, Threshold);

    //Configure Touchpad as wakeup source
    esp_sleep_disable_wakeup_source(ESP_SLEEP_WAKEUP_TIMER);
    esp_sleep_enable_touchpad_wakeup();
    esp_deep_sleep_start();
}
```

```
// ---------------------------------------------------------------------
// Light Sleep Mode
// ---------------------------------------------------------------------

void lightSleep() {
    esp_sleep_enable_timer_wakeup(SLEEP_DURATION);
    esp_light_sleep_start();
}

//----------------------------------------------------------------------
// Modem Sleep Mode
// ---------------------------------------------------------------------

/*void disableWiFi(){
    WiFi.disconnect(true);  // Disconnect from the network
    WiFi.mode(WIFI_OFF);    // Switch WiFi off
}*/

void setModemSleep() {
    //disableWiFi();
    setCpuFrequencyMhz(40);
    // Use this if 40Mhz is not supported
    // setCpuFrequencyMhz(80);
}

/*void enableWiFi(){
    WiFi.disconnect(false);  // Reconnect the network
    WiFi.mode(WIFI_STA);    // Switch WiFi off
    delay(200);
    WiFi.begin(STA_SSID, STA_PASS);   //Start Wifi
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);                             // Wait for connect
Wifi
    }
}*/

void wakeModemSleep() {
    setCpuFrequencyMhz(240);
    //enableWiFi();
}

//----------------------------------------------------------------------
// The main loop
// ---------------------------------------------------------------------
void loop(){
 if ( Count % 10 == 0  && Count != 0)
 {
  Count++;
  Serial.println("Modo Deep Sleep");
  deepSleep();
 }
 Serial.print("Contador = ");
 Serial.println(Count);
 Count++;
 delay(10);
 lightSleep();
}
```

Anexo IX: Código Prueba del Sistema Completo

```
#include <pitches.h>
#include <Tone32.h>
#include "MedianFilterLib.h"
#include "esp_system.h"
#include "esp_adc_cal.h"
#include "driver/adc.h"
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <MPU9250_asukiaaa.h>

#define ADC_A1 2
#define ADC_A2 33
#define ADC_A3 4
#define ADC_A4 32
#define ADC_A5 35
#define ADC_A6 34
#define K 18          //transistor that active the
gauge system

#define ADC_ABAT 15

#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 21
#define SCL_PIN 22
#endif

#define SLED1 13
#define SLED2 12
#define SLED3 23
#define SLED4 27
#define SLED5 26
#define SLED6 25

#define SBUZZER 19
#define BUZZER_CHANNEL 0

#define NO_OF_SAMPLES   64          //Multisampling
#define REF_VOLTAGE 1100

#define ADC_12_BIT_RES  4096

#define Threshold 20 /* Greater the value, more the
sensitivity */

//N° of Last samples to do the median filter
MedianFilter<float> medianFilter_Galga_1(7);
MedianFilter<float> medianFilter_Galga_2(7);
MedianFilter<float> medianFilter_Galga_3(7);
MedianFilter<float> medianFilter_Galga_4(7);
MedianFilter<float> medianFilter_Galga_5(7);
MedianFilter<float> medianFilter_Galga_6(7);
MedianFilter<float> medianFilter_Bat(7);

float RealTime;
float InitTestTime;
////////////////////////////////////////////////////
////////////////////////////////

//////////////////////////Gauge and Battery
Variables////////////////////////////
uint8_t nivel;     //the level in the force scale
double lim_1;             //value to stabilize
initial gauge 1 measurements
double lim_2;             //value to stabilize
initial gauge 2 measurements
double lim_3;             //value to stabilize
initial gauge 3 measurements
double lim_4;             //value to stabilize
initial gauge 4 measurements
double lim_5;             //value to stabilize
initial gauge 5 measurements
double lim_6;             //value to stabilize
initial gauge 6 measurements
double valor_ini_1;         //auxiliary variable to
stabilize initial gauge 1 measurements
double valor_ini_2;         //auxiliary variable to
stabilize initial gauge 2 measurements
double valor_ini_3;         //auxiliary variable to
stabilize initial gauge 3 measurements
double valor_ini_4;         //auxiliary variable to
stabilize initial gauge 4 measurements
double valor_ini_5;         //auxiliary variable to
stabilize initial gauge 5 measurements
double valor_ini_6;         //auxiliary variable to
stabilize initial gauge 6 measurements
////////////////////////////////////////////////////

////////////////////////// BLE Variables ////////////
RTC_DATA_ATTR uint8_t MODE = 0;
char txGalga1[8];
char txGalga2[8];
char txGalga3[8];
char txGalga4[8];
char txGalga5[8];
char txGalga6[8];
char txBat[8];
char txAveAccSqrt[8];
char txAveVel[8];
char txShakes[8];
char txMaxAccSqrt[8];
char txTime[8];
////////////////////////////////////////////////////

////////////////////////// Accelerometer Variables ///
MPU9250_asukiaaa mySensor;
float aX, aY, aZ, aSqrt, gX, gY, gZ, mX, mY, mZ,
aSqrtMax, AveVel, shake = 0;
float GyroMeasError = PI * (40.0f / 180.0f);   //
gyroscope measurement error in rads/s (start at 40
deg/s)
float GyroMeasDrift = PI * (0.0f  / 180.0f);   //
gyroscope measurement drift in rad/s/s (start at 0.0
deg/s/s)
float q[4] = {1.0f, 0.0f, 0.0f, 0.0f};    // vector
to hold quaternion
float beta = sqrt(3.0f / 4.0f) * GyroMeasError;   //
compute beta
float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;   //
compute zeta, the other free parameter in the
Madgwick scheme usually set to a small or zero value
#define Kp 2.0f * 5.0f // these are the free
parameters in the Mahony filter and fusion scheme, Kp
for proportional feedback, Ki for integral
#define Ki 0.0f
float deltat = 0.0f;        // integration interval
for both filter schemes
float eInt[3] = {0.0f, 0.0f, 0.0f};       // vector
to hold integral error for Mahony method
////////////////////////////////////////////////////

//////////////////BLE System/////////////////////////
BLEServer *pServer = NULL;
//Objects to send data
BLECharacteristic *characteristicTX_1_Galga_1;
BLECharacteristic *characteristicTX_1_Galga_2;
BLECharacteristic *characteristicTX_1_Galga_3;
BLECharacteristic *characteristicTX_1_Galga_4;
BLECharacteristic *characteristicTX_1_Galga_5;
BLECharacteristic *characteristicTX_1_Galga_6;
```

```
BLECharacteristic *characteristicTX_2_BAT;
BLECharacteristic *characteristicTX_3_AveAccSqrt;
BLECharacteristic *characteristicTX_3_AveVel;
BLECharacteristic *characteristicTX_3_MaxAccSqrt;
BLECharacteristic *characteristicTX_3_Shakes;
BLECharacteristic *characteristicTX_4_Time;

bool deviceConnected = false; //controle de
dispositivo conectado

#define SERVICE_UUID   "ab0828b1-198e-4351-b779-
901fa0e0371e"
#define CHARACTERISTIC_UUID_RX  "4ac8a682-9736-4e5d-
932b-e9b31405049c"

#define SERVICE_UUID_1   "66865fa9-749f-4ce1-9758-
c98286e027d3"      //Gauge Service 1
#define SERVICE_UUID_1_1   "d724fa2a-974a-43ad-8364-
0be35a577d8c"      //Gauge Service 1.1
#define CHARACTERISTIC_UUID_TX_1_Galga_1  "0972EF8C-
7613-4075-AD52-756F33D4DA91"          //Gauge 1
#define CHARACTERISTIC_UUID_TX_1_Galga_2  "25bb41ac-
7d9b-4116-8188-68f72816cf95"          //Gauge 2
#define CHARACTERISTIC_UUID_TX_1_Galga_3  "c5372a74-
e306-4faa-82f0-b3d08827f76d"          //Gauge 3
#define CHARACTERISTIC_UUID_TX_1_Galga_4  "8a028403-
9164-4f2f-9e04-70b81be8ca8c"          //Gauge 4
#define CHARACTERISTIC_UUID_TX_1_Galga_5  "4fd615c3-
c6b8-4f52-85c5-9be6cb9c0515"          //Gauge 5
#define CHARACTERISTIC_UUID_TX_1_Galga_6  "d15a079a-
9643-4ac2-b4af-84a35e3d6a55"          //Gauge 6

#define SERVICE_UUID_2   "3afbeacc-92df-4a0e-94f7-
dde04f765e71"       //Battery Service
#define CHARACTERISTIC_UUID_TX_2_BAT  "8e563bc1-e849-
4728-9f19-c8d1867ba979"          //ABAT

#define SERVICE_UUID_3   "a7cc6015-8b1e-4dd2-a6d2-
e17842a02480"       //Accelerometer Service
#define
CHARACTERISTIC_UUID_TX_3_AveAccSqrt  "be8e32ab-2d0c-
4476-af11-4bcf75a7511e"          //Average
Acceleration Sqrt
#define CHARACTERISTIC_UUID_TX_3_AveVel  "f1ed081f-
5835-47ad-8d22-39ea03ee2a8e"          //Average
Acceleration
#define
CHARACTERISTIC_UUID_TX_3_MaxAccSqrt  "5eed1df9-ab31-
479f-986f-b1691808860e"          //Max. Acceleration
Sqrt
#define CHARACTERISTIC_UUID_TX_3_Shakes  "978ec928-
0cb9-4054-a7cb-d57e227740fe"          //Max.
Acceleration Sqrt


#define SERVICE_UUID_4   "5fe7cddc-4e8c-49e0-9a21-
5c26d87d52cc"       //Time Service
#define CHARACTERISTIC_UUID_TX_4_Time  "df92a0e9-
7aeb-4698-b862-d493f27e84f3"          //Time

//callback para receber os eventos de conexão de
dispositivos
class ServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
 deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
 deviceConnected = false;
    }
};
```

```
/callback para eventos das características
class CharacteristicCallbacks: public
BLECharacteristicCallbacks {
     void onWrite(BLECharacteristic *characteristic)
{
          //retorna ponteiro para o registrador
contendo o valor atual da caracteristica
          std::string rxValue = characteristic-
>getValue();
          //verifica se existe dados (tamanho maior
que zero)
          if (rxValue.length() > 0) {

               for (int i = 0; i < rxValue.length();
i++) {
              Serial.print(rxValue[i]);
                }
                Serial.println();
                if (rxValue.find("1") != -1) {
              MODE = 1;
                }
                else if (rxValue.find("2") != -1) {
              MODE = 2;
              InitTestTime = millis();
                }
                else if (rxValue.find("3") != -1) {
              MODE = 3;
                }
          }
     }//onWrite
};

void ConfigBLE()
{
  // Create the BLE Device
    BLEDevice::init("SmartCube V4"); //  BLE device's
name

    // Create the BLE Server
    pServer = BLEDevice::createServer(); //Create a
Server
    pServer->setCallbacks(new ServerCallbacks());
//Callback Server

    // Create the BLE Service
    BLEService *pService = pServer-
>createService(SERVICE_UUID);
    BLEService *pService_1 = pServer-
>createService(BLEUUID(SERVICE_UUID_1), 100);
    //BLEService *pService_1_1 = pServer-
>createService(SERVICE_UUID_1_1);
    BLEService *pService_2 = pServer-
>createService(SERVICE_UUID_2);
    BLEService *pService_3 = pServer-
>createService(SERVICE_UUID_3);
    BLEService *pService_4 = pServer-
>createService(SERVICE_UUID_4);


     // Create a BLE Characteristic para envio de
dados
    characteristicTX_1_Galga_1 = pService_1-
>createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga
_1,
                        BLECharacteristic::PROPERTY_NO
TIFY
                      );
    characteristicTX_1_Galga_1->addDescriptor(new
BLE2902());
```

```
characteristicTX_1_Galga_2 = pService_1->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga_2,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_1_Galga_2->addDescriptor(new BLE2902());


    characteristicTX_1_Galga_3 = pService_1->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga_3,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_1_Galga_3->addDescriptor(new BLE2902());


    characteristicTX_1_Galga_4 = pService_1->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga_4,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_1_Galga_4->addDescriptor(new BLE2902());


    characteristicTX_1_Galga_5 = pService_1->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga_5,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_1_Galga_5->addDescriptor(new BLE2902());


    characteristicTX_1_Galga_6 = pService_1->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_1_Galga_6,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_1_Galga_6->addDescriptor(new BLE2902());



    characteristicTX_2_BAT = pService_2->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_2_BAT,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_2_BAT->addDescriptor(new BLE2902());


    characteristicTX_3_AveAccSqrt = pService_3->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_3_AveAccSqrt,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_3_AveAccSqrt->addDescriptor(new BLE2902());
```

```
   characteristicTX_3_AveVel = pService_3->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_3_AveVel,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_3_AveVel->addDescriptor(new BLE2902());

    characteristicTX_3_MaxAccSqrt = pService_3->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_3_MaxAccSqrt,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_3_MaxAccSqrt->addDescriptor(new BLE2902());

    characteristicTX_3_Shakes = pService_3->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_3_Shakes,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_3_Shakes->addDescriptor(new BLE2902());



    characteristicTX_4_Time = pService_4->createCharacteristic(
                        CHARACTERISTIC_UUID_TX_4_Time,
                        BLECharacteristic::PROPERTY_NOTIFY
                      );
    characteristicTX_4_Time->addDescriptor(new BLE2902());

// Create a BLE Characteristic to recibe data
    BLECharacteristic *characteristic = pService->createCharacteristic(

 CHARACTERISTIC_UUID_RX,

 BLECharacteristic::PROPERTY_WRITE
                                                      )
;

    characteristic->setCallbacks(new CharacteristicCallbacks());

    // Start the service
    pService->start();
    pService_1->start();
    //pService_1_1->start();
     pService_2->start();
    pService_3->start();
    pService_4->start();
    // Start advertising
    pServer->getAdvertising()->start();
}
////////////////////////////////////////////////////

//////////////////Gauge and Battery Systems/////////
esp_adc_cal_characteristics_t *adc_chars_ADC_1 = new esp_adc_cal_characteristics_t;
esp_adc_cal_characteristics_t *adc_chars_ADC_2 = new esp_adc_cal_characteristics_t;
```

```
adc1_channel_t get_adc1_chanel(uint8_t pin) {
  adc1_channel_t chan;
  switch (pin) {
    case 32:
      chan = ADC1_CHANNEL_4;
      break;
    case 33:
      chan = ADC1_CHANNEL_5;
      break;
    case 34:
      chan = ADC1_CHANNEL_6;
      break;
    case 35:
      chan = ADC1_CHANNEL_7;
      break;
    case 36:
      chan = ADC1_CHANNEL_0;
      break;
    case 37:
      chan = ADC1_CHANNEL_1;
      break;
    case 38:
      chan = ADC1_CHANNEL_2;
      break;
    case 39:
      chan = ADC1_CHANNEL_3;
      break;
  }
  return chan;
}
adc2_channel_t get_adc2_chanel(uint8_t pin) {
  adc2_channel_t chan;
  switch (pin) {
    case 4:
      chan = ADC2_CHANNEL_0;
      break;
    case 0:
      chan = ADC2_CHANNEL_1;
      break;
    case 2:
      chan = ADC2_CHANNEL_2;
      break;
    case 15:
      chan = ADC2_CHANNEL_3;
      break;
    case 13:
      chan = ADC2_CHANNEL_4;
      break;
    case 12:
      chan = ADC2_CHANNEL_5;
      break;
    case 14:
      chan = ADC2_CHANNEL_6;
      break;
    case 27:
      chan = ADC2_CHANNEL_7;
      break;
    case 25:
      chan = ADC2_CHANNEL_8;
      break;
    case 26:
      chan = ADC2_CHANNEL_9;
      break;
  }
  return chan;
}
void read_bat() {
  float adc_reading = 0;
  int read_raw;
  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
        //Check that the ADC2 is not used by the Wifi
   esp_err_t r =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_ABAT
), ADC_WIDTH_12Bit, &read_raw);
    if ( r == ESP_OK ) {
       adc_reading = adc_reading +
read_raw;                       //It's free. Save de
value.
    }
  }
adc_reading /= NO_OF_SAMPLES;

  //MedianFilter
   adc_reading =
medianFilter_Bat.AddValue(adc_reading);


  //Convert adc_reading to voltage in mV
    dtostrf(esp_adc_cal_raw_to_voltage(adc_reading,
adc_chars_ADC_2)*3, 2, 2, txBat);

  characteristicTX_2_BAT->setValue(txBat);
  characteristicTX_2_BAT->notify();
}

double init_galgas() {
  float adc_reading_1 = 0;
  float adc_reading_2 = 0;
  float adc_reading_3 = 0;
  float adc_reading_4 = 0;
  float adc_reading_5 = 0;
  float adc_reading_6 = 0;
  int read_raw1;
  int read_raw2;

  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
  adc_reading_2 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A2))
;
  adc_reading_4 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A4))
;
  adc_reading_5 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A5))
;
  adc_reading_6 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A6))
;
  esp_err_t r1 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A1),
ADC_WIDTH_12Bit, &read_raw1);
    if ( r1 == ESP_OK ) {
       adc_reading_1 = adc_reading_1 +
read_raw1;                      //It's free. Save de
value.
    }
  esp_err_t r3 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A3),
ADC_WIDTH_12Bit, &read_raw2);
    if ( r3 == ESP_OK ) {
       adc_reading_3 = adc_reading_3 +
read_raw2;                      //It's free. Save de
value.
    }
  }
  adc_reading_1 /= NO_OF_SAMPLES;
  adc_reading_2 /= NO_OF_SAMPLES;
  adc_reading_3 /= NO_OF_SAMPLES;
  adc_reading_4 /= NO_OF_SAMPLES;
  adc_reading_5 /= NO_OF_SAMPLES;
  adc_reading_6 /= NO_OF_SAMPLES;
//MedianFilter
   adc_reading_1 =
medianFilter_Galga_1.AddValue(adc_reading_1);
   adc_reading_2 =
medianFilter_Galga_2.AddValue(adc_reading_2);
    adc_reading_3 =
medianFilter_Galga_3.AddValue(adc_reading_3);
```

```
adc_reading_4 =
medianFilter_Galga_4.AddValue(adc_reading_4);
   adc_reading_5 =
medianFilter_Galga_5.AddValue(adc_reading_5);
   adc_reading_6 =
medianFilter_Galga_6.AddValue(adc_reading_6);


  //Convert adc_reading to voltage in mV
  double voltage_1 =
esp_adc_cal_raw_to_voltage(adc_reading_1,
adc_chars_ADC_2);
  double voltage_2 =
esp_adc_cal_raw_to_voltage(adc_reading_2,
adc_chars_ADC_1);
  double voltage_3 =
esp_adc_cal_raw_to_voltage(adc_reading_3,
adc_chars_ADC_2);
  double voltage_4 =
esp_adc_cal_raw_to_voltage(adc_reading_4,
adc_chars_ADC_1);
  double voltage_5 =
esp_adc_cal_raw_to_voltage(adc_reading_5,
adc_chars_ADC_1);
  double voltage_6 =
esp_adc_cal_raw_to_voltage(adc_reading_6,
adc_chars_ADC_1);

  return voltage_1, voltage_2, voltage_3, voltage_4,
voltage_5, voltage_6;
}


void read_galgas() {
  float adc_reading_1 = 0;
  float adc_reading_2 = 0;
  float adc_reading_3 = 0;
  float adc_reading_4 = 0;
  float adc_reading_5 = 0;
  float adc_reading_6 = 0;
  int read_raw1 = 0;
  int read_raw2 = 0;

  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
  adc_reading_2 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A2))
;
  adc_reading_4 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A4))
;
  adc_reading_5 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A5))
;
  adc_reading_6 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A6))
;
  esp_err_t r1 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A1),
ADC_WIDTH_12Bit, &read_raw1);
    if ( r1 == ESP_OK ) {
        adc_reading_1 = adc_reading_1 +
read_raw1;                          //It's free. Save de
value.
    }
  esp_err_t r3 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A3),
ADC_WIDTH_12Bit, &read_raw2);
    if ( r3 == ESP_OK ) {
        adc_reading_3 = adc_reading_3 +
read_raw2;                          //It's free. Save de
value.
    }
  }
```

```
adc_reading_1 /= NO_OF_SAMPLES;
adc_reading_2 /= NO_OF_SAMPLES;
adc_reading_3 /= NO_OF_SAMPLES;
adc_reading_4 /= NO_OF_SAMPLES;
adc_reading_5 /= NO_OF_SAMPLES;
adc_reading_6 /= NO_OF_SAMPLES;

  //MedianFilter
   adc_reading_1 =
medianFilter_Galga_1.AddValue(adc_reading_1);
   adc_reading_2 =
medianFilter_Galga_2.AddValue(adc_reading_2);
   adc_reading_3 =
medianFilter_Galga_3.AddValue(adc_reading_3);
   adc_reading_4 =
medianFilter_Galga_4.AddValue(adc_reading_4);
   adc_reading_5 =
medianFilter_Galga_5.AddValue(adc_reading_5);
   adc_reading_6 =
medianFilter_Galga_6.AddValue(adc_reading_6);

 //Convert Float to Char and save in the char
variable
  dtostrf((lim_1 -
esp_adc_cal_raw_to_voltage(adc_reading_1,
adc_chars_ADC_2)), 2, 2, txGalga1);
  dtostrf((lim_2 -
esp_adc_cal_raw_to_voltage(adc_reading_2,
adc_chars_ADC_1)), 2, 2, txGalga2);
  dtostrf((lim_3 -
esp_adc_cal_raw_to_voltage(adc_reading_3,
adc_chars_ADC_2)), 2, 2, txGalga3);
  dtostrf((lim_4 -
esp_adc_cal_raw_to_voltage(adc_reading_4,
adc_chars_ADC_1)), 2, 2, txGalga4);
  dtostrf((lim_5 -
esp_adc_cal_raw_to_voltage(adc_reading_5,
adc_chars_ADC_1)), 2, 2, txGalga5);
  dtostrf((lim_6 -
esp_adc_cal_raw_to_voltage(adc_reading_6,
adc_chars_ADC_1)), 2, 2, txGalga6);

  characteristicTX_1_Galga_1->setValue(txGalga1);
//set the value in the characteristic
  characteristicTX_1_Galga_1->notify(); // Send the
value to the device
  characteristicTX_1_Galga_2->setValue(txGalga2);
  characteristicTX_1_Galga_2->notify();
  characteristicTX_1_Galga_3->setValue(txGalga3);
  characteristicTX_1_Galga_3->notify();
  characteristicTX_1_Galga_4->setValue(txGalga4);
  characteristicTX_1_Galga_4->notify();
  characteristicTX_1_Galga_5->setValue(txGalga5);
  characteristicTX_1_Galga_5->notify();
  characteristicTX_1_Galga_6->setValue(txGalga6);
  characteristicTX_1_Galga_6->notify();
}
void adc2_setup()
{
  adc2_config_channel_atten(get_adc2_chanel(ADC_ABAT)
, ADC_ATTEN_6db );          //Set up the channel and
the attenuation
  adc2_config_channel_atten(get_adc2_chanel(ADC_A1),
ADC_ATTEN_6db );          //Set up the channel and the
attenuation
  adc2_config_channel_atten(get_adc2_chanel(ADC_A3),
ADC_ATTEN_6db );
  esp_adc_cal_value_t val_type =
esp_adc_cal_characterize(ADC_UNIT_2, ADC_ATTEN_6db,
ADC_WIDTH_12Bit, REF_VOLTAGE, adc_chars_ADC_2);
//Calibration
}
void adc1_setup() {

  /* ADC1 Configuration */
   adc1_config_width(ADC_WIDTH_12Bit);
```

```
adc1_config_channel_atten(get_adc1_chanel(ADC_A2),
ADC_ATTEN_6db );
  adc1_config_channel_atten(get_adc1_chanel(ADC_A4),
ADC_ATTEN_6db );
  adc1_config_channel_atten(get_adc1_chanel(ADC_A5),
ADC_ATTEN_6db );
  adc1_config_channel_atten(get_adc1_chanel(ADC_A6),
ADC_ATTEN_6db );
  esp_adc_cal_value_t val_type =
esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_6db,
ADC_WIDTH_12Bit, REF_VOLTAGE, adc_chars_ADC_1);
}
//////////////////////////////////////////////////

//////////////////////OTA Programation///////////////
const char* ssid = "Mate";
const char* password = "minicrak6";

void OTA(){
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED)
{
    Serial.println("Connection FaiLed!
Rebooting...");
    delay(5000);
    ESP.restart();
  }

  // Port defaults to 3232
  // ArduinoOTA.setPort(3232);

  // Hostname defaults to esp3232-[MAC]
   ArduinoOTA.setHostname("SmartCube");

  // No authentication by default
  //ArduinoOTA.setPassword("admin");
/ Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4
a801fc3");

  ArduinoOTA
    .onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";

      // NOTE: if updating SPIFFS this would be the
place to unmount SPIFFS using SPIFFS.end()
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned
int total) {
      Serial.printf("Progress: %u%%\r", (progress /
(total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR)
Serial.println("Auth FaiLed");
      else if (error == OTA_BEGIN_ERROR)
Serial.println("Begin FaiLed");
      else if (error == OTA_CONNECT_ERROR)
Serial.println("Connect FaiLed");
      else if (error == OTA_RECEIVE_ERROR)
Serial.println("Receive FaiLed");
      else if (error == OTA_END_ERROR)
Serial.println("End FaiLed");
    });
 ArduinoOTA.begin();

  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
//////////////////////////////////////////////////

//////////////////////// LED System ////////////////////
void ConfigLeds()
{
  pinMode(SLED1, OUTPUT);
  pinMode(SLED2, OUTPUT);
  pinMode(SLED3, OUTPUT);
  pinMode(SLED4, OUTPUT);
  pinMode(SLED5, OUTPUT);
  pinMode(SLED6, OUTPUT);
  SetOffLED(1);
  SetOffLED(2);
  SetOffLED(3);
  SetOffLED(4);
  SetOffLED(5);
  SetOffLED(6);
}

void SetOnLED(uint8_t LED)
{
  switch (LED)
  {
    case 1:
      digitalWrite(SLED1, HIGH);
      break;
    case 2:
      digitalWrite(SLED2, HIGH);
      break;
    case 3:
      digitalWrite(SLED3, HIGH);
      break;
    case 4:
      digitalWrite(SLED4, HIGH);
      break;
    case 5:
      digitalWrite(SLED5, HIGH);
      break;
    case 6:
      digitalWrite(SLED6, HIGH);
      break;
  }
}

void SetOffLED(uint8_t LED)
{
  switch (LED)
  {
    case 1:
      digitalWrite(SLED1, LOW);
      break;
    case 2:
      digitalWrite(SLED2, LOW);
      break;
    case 3:
      digitalWrite(SLED3, LOW);
      break;
    case 4:
      digitalWrite(SLED4, LOW);
      break;
    case 5:
      digitalWrite(SLED5, LOW);
      break;
    case 6:
      digitalWrite(SLED6, LOW);
       break;
  }
}
//////////////////////////////////////////////////
```

```
////////////////////////// Buzzer System//////////////
void beeps(uint8_t count)
{
  for (int i = 0; i < count; i++)
  {
  tone(SBUZZER, NOTE_C4, 500, BUZZER_CHANNEL);
  noTone(SBUZZER, BUZZER_CHANNEL);
  }
}
////////////////////////////////////////////////////

////////////////////////// Accelerometer System //////
void Init_Accelerometer()
{
#ifdef _ESP32_HAL_I2C_H_ // For ESP32
  Wire.begin(SDA_PIN, SCL_PIN);
  mySensor.setWire(&Wire);
#endif

  mySensor.beginAccel();
  mySensor.beginGyro();
  mySensor.beginMag();

  // You can set your own offset for mag values
  // mySensor.magXOffset = -50;
  // mySensor.magYOffset = -55;
  // mySensor.magZOffset = -10;
}
void Get_Accelerometer_Measurements()
{
  float AveVel_p, aSqrt_aux, aSqrt_aux_p;
  unsigned long time_p;
  unsigned long time_n;
  unsigned long time_i = millis();
  bool SK = false;
  int j = 0;
  for (int i = 0; i < NO_OF_SAMPLES; i++) {
  if (mySensor.accelUpdate() == 0) {
    aX = mySensor.accelX();
    aY = mySensor.accelY();
    aZ = mySensor.accelZ();
  }

  if (mySensor.gyroUpdate() == 0) {
    gX = mySensor.gyroX();
    gY = mySensor.gyroY();
    gZ = mySensor.gyroZ();
  }

  if (mySensor.magUpdate() == 0) {
    mX = mySensor.magX();
    mY = mySensor.magY();
    mZ = mySensor.magZ();
   }
MadgwickQuaternionUpdate(aX, aY, aZ, gX*PI/180.0f,
gY*PI/180.0f, gZ*PI/180.0f,  mX,  mY, mZ);
  //MahonyQuaternionUpdate(aX, aY, aZ, gX*PI/180.0f,
gY*PI/180.0f, gZ*PI/180.0f,  mX,  mY, mZ);
  aSqrt_aux = mySensor.accelSqrt();
  if (j != 0)
  {
    time_n = millis();
    aSqrt += aSqrt_aux;
    AveVel_p = (aSqrt_aux*(time_n - time_p)/1000) +
AveVel_p;
    AveVel += AveVel_p;
    if (aSqrt_aux > aSqrtMax){ aSqrtMax = aSqrt_aux;}
    time_p = time_n;
     if ( (aSqrt_aux - aSqrt_aux_p) > 0.27)
    {
      SK = true;
    }
    aSqrt_aux_p = aSqrt_aux;
  }
   if (j == 0)
{
time_p = millis();
    aSqrt = aSqrt_aux;
    aSqrt_aux_p = aSqrt_aux;
    aSqrtMax = aSqrt;
    AveVel = aSqrt_aux*(time_p - time_i)/1000;
    AveVel_p = AveVel;
    j = 1;
}
 }
  aSqrt /= NO_OF_SAMPLES;
  AveVel /= NO_OF_SAMPLES;
  if(SK)shake++;
   //Convert Float to Char and save in the char
variable
  dtostrf(aSqrt, 2, 2, txAveAccSqrt);
  dtostrf(AveVel, 2, 2, txAveVel);
  dtostrf(aSqrtMax, 2, 2, txMaxAccSqrt);
  dtostrf(shake, 2, 2, txShakes);


  characteristicTX_3_AveAccSqrt-
>setValue(txAveAccSqrt); //set the value in the
characteristic
  characteristicTX_3_AveAccSqrt->notify(); // Send
the value to the device
  characteristicTX_3_MaxAccSqrt-
>setValue(txMaxAccSqrt);
  characteristicTX_3_MaxAccSqrt->notify();
  characteristicTX_3_AveVel->setValue(txAveVel);
  characteristicTX_3_AveVel->notify();
  characteristicTX_3_Shakes->setValue(txShakes);
  characteristicTX_3_Shakes->notify();

}
void MadgwickQuaternionUpdate(float ax, float ay,
float az, float gx, float gy, float gz, float mx,
float my, float mz)
        {
            float q1 = q[0], q2 = q[1], q3 = q[2], q4
= q[3];   // short name local variable for
readability
            float norm;
            float hx, hy, _2bx, _2bz;
            float s1, s2, s3, s4;
            float qDot1, qDot2, qDot3, qDot4;
//            float gerrx, gerry, gerrz, gbiasx,
gbiasy, gbiasz;        // gyro bias error

            // Auxiliary variables to avoid repeated
arithmetic
            float _2q1mx;
            float _2q1my;
            float _2q1mz;
            float _2q2mx;
            float _4bx;
            float _4bz;
            float _2q1 = 2.0f * q1;
            float _2q2 = 2.0f * q2;
            float _2q3 = 2.0f * q3;
            float _2q4 = 2.0f * q4;
            float _2q1q3 = 2.0f * q1 * q3;
            float _2q3q4 = 2.0f * q3 * q4;
            float q1q1 = q1 * q1;
            float q1q2 = q1 * q2;
            float q1q3 = q1 * q3;
            float q1q4 = q1 * q4;
            float q2q2 = q2 * q2;
            float q2q3 = q2 * q3;
            float q2q4 = q2 * q4;
            float q3q3 = q3 * q3;
            float q3q4 = q3 * q4;
            float q4q4 = q4 * q4;
```

```
            // Normalise accelerometer measurement
            norm = sqrt(ax * ax + ay * ay + az * az);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f/norm;
            ax *= norm;
            ay *= norm;
            az *= norm;

            // Normalise magnetometer measurement
            norm = sqrt(mx * mx + my * my + mz * mz);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f/norm;
            mx *= norm;
            my *= norm;
            mz *= norm;

            // Reference direction of Earth's
magnetic field
            _2q1mx = 2.0f * q1 * mx;
            _2q1my = 2.0f * q1 * my;
            _2q1mz = 2.0f * q1 * mz;
            _2q2mx = 2.0f * q2 * mx;
            hx = mx * q1q1 - _2q1my * q4 + _2q1mz *
q3 + mx * q2q2 + _2q2 * my * q3 + _2q2 * mz * q4 - mx
* q3q3 - mx * q4q4;
            hy = _2q1mx * q4 + my * q1q1 - _2q1mz *
q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 * mz
* q4 - my * q4q4;
            _2bx = sqrt(hx * hx + hy * hy);
            _2bz = -_2q1mx * q3 + _2q1my * q2 + mz *
q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 - mz
* q3q3 + mz * q4q4;
            _4bx = 2.0f * _2bx;
            _4bz = 2.0f * _2bz;

            // Gradient decent algorithm corrective
step
            s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax)
+ _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3 *
(_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) -
mx) + (-_2bx * q4 + _2bz * q2) * (_2bx * (q2q3 -
q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 *
(_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) -
mz);
            s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) +
_2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 *
(1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 *
(_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) -
mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4)
+ _2bz * (q1q2 + q3q4) - my) + (_2bx * q4 - _4bz *
q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 -
q3q3) - mz);
            s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax)
+ _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3 *
(1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3
- _2bz * q1) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz *
(q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx
* (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx
* q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz *
(0.5f - q2q2 - q3q3) - mz);
            s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) +
_2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4 +
_2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz *
(q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz * q3) *
(_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) +
_2bx * q2 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f -
q2q2 - q3q3) - mz);
            norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 +
s4 * s4);    // normalise step magnitude
            norm = 1.0f/norm;
            s1 *= norm;
            s2 *= norm;
            s3 *= norm;
            s4 *= norm;
```

```
            // Compute rate of change of quaternion
            qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 *
gz) - beta * s1;
            qDot2 = 0.5f * ( q1 * gx + q3 * gz - q4 *
gy) - beta * s2;
            qDot3 = 0.5f * ( q1 * gy - q2 * gz + q4 *
gx) - beta * s3;
            qDot4 = 0.5f * ( q1 * gz + q2 * gy - q3 *
gx) - beta * s4;

            // Integrate to yield quaternion
            q1 += qDot1 * deltat;
            q2 += qDot2 * deltat;
            q3 += qDot3 * deltat;
            q4 += qDot4 * deltat;
            norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 +
q4 * q4);    // normalise quaternion
            norm = 1.0f/norm;
            q[0] = q1 * norm;
            q[1] = q2 * norm;
            q[2] = q3 * norm;
            q[3] = q4 * norm;
        }

// Similar to Madgwick scheme but uses proportional
and integral filtering on the error between estimated
reference vectors and
 // measured ones.
            void MahonyQuaternionUpdate(float ax,
float ay, float az, float gx, float gy, float gz,
float mx, float my, float mz) {
            float q1 = q[0], q2 = q[1], q3 = q[2], q4
= q[3];   // short name local variable for
readability
            float norm;
            float hx, hy, bx, bz;
            float vx, vy, vz, wx, wy, wz;
            float ex, ey, ez;
            float pa, pb, pc;

            // Auxiliary variables to avoid repeated
arithmetic
            float q1q1 = q1 * q1;
            float q1q2 = q1 * q2;
            float q1q3 = q1 * q3;
            float q1q4 = q1 * q4;
            float q2q2 = q2 * q2;
            float q2q3 = q2 * q3;
            float q2q4 = q2 * q4;
            float q3q3 = q3 * q3;
            float q3q4 = q3 * q4;
            float q4q4 = q4 * q4;

            // Normalise accelerometer measurement
            norm = sqrt(ax * ax + ay * ay + az * az);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f / norm;        // use
reciprocal for division
            ax *= norm;
            ay *= norm;
            az *= norm;

            // Normalise magnetometer measurement
            norm = sqrt(mx * mx + my * my + mz * mz);
            if (norm == 0.0f) return; // handle NaN
            norm = 1.0f / norm;        // use
reciprocal for division
            mx *= norm;
            my *= norm;
            mz *= norm;
```

```
            // Reference direction of Earth's magnetic
field
            hx = 2.0f * mx * (0.5f - q3q3 - q4q4) +
2.0f * my * (q2q3 - q1q4) + 2.0f * mz * (q2q4 +
q1q3);
            hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f *
my * (0.5f - q2q2 - q4q4) + 2.0f * mz * (q3q4 -
q1q2);
            bx = sqrt((hx * hx) + (hy * hy));
            bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f *
my * (q3q4 + q1q2) + 2.0f * mz * (0.5f - q2q2 -
q3q3);

            // Estimated direction of gravity and
magnetic field
            vx = 2.0f * (q2q4 - q1q3);
            vy = 2.0f * (q1q2 + q3q4);
            vz = q1q1 - q2q2 - q3q3 + q4q4;
            wx = 2.0f * bx * (0.5f - q3q3 - q4q4) +
2.0f * bz * (q2q4 - q1q3);
            wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f *
bz * (q1q2 + q3q4);
            wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f *
bz * (0.5f - q2q2 - q3q3);

            // Error is cross product between
estimated direction and measured direction of gravity
            ex = (ay * vz - az * vy) + (my * wz - mz
* wy);
            ey = (az * vx - ax * vz) + (mz * wx - mx
* wz);
            ez = (ax * vy - ay * vx) + (mx * wy - my
* wx);
            if (Ki > 0.0f)
            {
                eInt[0] += ex;      // accumulate
integral error
                eInt[1] += ey;
                eInt[2] += ez;
            }
            else
            {
                eInt[0] = 0.0f;     // prevent
integral wind up
                eInt[1] = 0.0f;
                eInt[2] = 0.0f;
            }

            // Apply feedback terms
            gx = gx + Kp * ex + Ki * eInt[0];
            gy = gy + Kp * ey + Ki * eInt[1];
            gz = gz + Kp * ez + Ki * eInt[2];

            // Integrate rate of change of quaternion
            pa = q2;
            pb = q3;
            pc = q4;
            q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz)
* (0.5f * deltat);
            q2 = pa + (q1 * gx + pb * gz - pc * gy) *
(0.5f * deltat);
            q3 = pb + (q1 * gy - pa * gz + pc * gx) *
(0.5f * deltat);
            q4 = pc + (q1 * gz + pa * gy - pb * gx) *
(0.5f * deltat);

            // Normalise quaternion
            norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 +
q4 * q4);
            norm = 1.0f / norm;
            q[0] = q1 * norm;
            q[1] = q2 * norm;
            q[2] = q3 * norm;
            q[3] = q4 * norm;
        }
//////////////////////////////////////////////////
```

```
/////////////////////////// Deep Sleep ///////////////
void callbackDeepSleep(){
  //placeholder callback function
}
void deepSleep() {
    //Setup interrupt on Touch Pad 6 (GPIO15)
    touchAttachInterrupt(T6, callbackDeepSleep,
Threshold);

    //Configure Touchpad as wakeup source
    //esp_sleep_disable_wakeup_source(ESP_SLEEP_WAKEU
P_TIMER);
    esp_sleep_enable_touchpad_wakeup();
    esp_deep_sleep_start();
}
//////////////////////////////////////////////////////

////////////////////////// Light Sleep ////////////////

// Definition of sleep mode properties
//                             seconds
//                               v
const uint32_t SLEEP_DURATION = 0.25*1000000; //
s  si* 1000000 = microsegundos, ya que en la función
debemos introducir microsegundos.

void lightSleep() {
    esp_sleep_enable_timer_wakeup(SLEEP_DURATION);
    esp_light_sleep_start();
}
//////////////////////////////////////////////////////

////////////////////////// Modem Sleep ////////////////
 void setModemSleep() {
    //disableWiFi();
    //setCpuFrequencyMhz(40);
    // Use this if 40Mhz is not supported
    setCpuFrequencyMhz(80);
}

void wakeModemSleep() {
    setCpuFrequencyMhz(240);
    //enableWiFi();
}
//////////////////////////////////////////////////////

//////////////////////////// Send Time ////////////////
void send_time()
{
  RealTime = millis();
  float SampleTime = (RealTime -
InitTestTime)/1000/3600;
   //Convert Float to Char and save in the char
variable
  dtostrf(SampleTime, 2, 2, txTime);


  characteristicTX_4_Time->setValue(txTime); //set
the time value (in hours) in the characteristic
  characteristicTX_4_Time->notify(); // Send the
value to the device

}
//////////////////////////////////////////////////////

void setup() {
  Serial.begin(115200);
  adc1_setup();
  adc2_setup();
  //read_bat();
  Init_Accelerometer();
  ConfigLeds();
  setModemSleep();
  ConfigBLE();
  }
```

```
void loop() {
if (deviceConnected) {
switch (MODE)
{
  case 1:
             for (int j = 0; j < 10; j++) {

             valor_ini_1, valor_ini_2, valor_ini_3,
valor_ini_4, valor_ini_5, valor_ini_6 =
init_galgas();
             if (j==0)
             {
             lim_1 = valor_ini_1;
             lim_2 = valor_ini_2;
             lim_3 = valor_ini_3;
             lim_4 = valor_ini_4;
             lim_5 = valor_ini_5;
             lim_6 = valor_ini_6;
             }

             if (valor_ini_1 < lim_1)lim_1 =
valor_ini_1;
             if (valor_ini_2 < lim_2)lim_2 =
valor_ini_2;
             if (valor_ini_3 < lim_3)lim_3 =
valor_ini_3;
             if (valor_ini_4 < lim_4)lim_4 =
valor_ini_4;
             if (valor_ini_5 < lim_5)lim_5 =
valor_ini_5;
             if (valor_ini_6 < lim_6)lim_6 =
valor_ini_6;


             uint8_t Led_ini = 1;
             SetOnLED(Led_ini);
             delay(500);
             SetOffLED(Led_ini);
             if(Led_ini == 6)Led_ini = 0;
             Led_ini++;
             }
             beeps(1);
             MODE = 0;
  break;

  case 2:
             SetOffLED(1);
             read_galgas();
             delay(10);
             read_bat();
             delay(10);
             Get_Accelerometer_Measurements();
             delay(10);
             send_time();
             delay(250);
             //deepSleep();
             //lightSleep();

  break;

  case 3:
             ArduinoOTA.handle();
  break;
}
}
}
```

Anexo X: Código de activación del sistema de las galgas

```
 digitalWrite(K, HIGH);

  //Multisampling
 for (int i = 0; i < NO_OF_SAMPLES; i++) {
  adc_reading_2 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A2));
  adc_reading_4 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A4));
  adc_reading_5 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A5));
  adc_reading_6 +=
adc1_get_raw((adc1_channel_t)get_adc1_chanel(ADC_A6));
  esp_err_t r1 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A1),
ADC_WIDTH_12Bit, &read_raw1);
    if ( r1 == ESP_OK ) {
        adc_reading_1 = adc_reading_1 +
read_raw1;                      //It's free. Save de value.
    }
  esp_err_t r3 =
adc2_get_raw((adc2_channel_t)get_adc2_chanel(ADC_A3),
ADC_WIDTH_12Bit, &read_raw2);
    if ( r3 == ESP_OK ) {
        adc_reading_3 = adc_reading_3 +
read_raw2;                      //It's free. Save de value.
    }
  }

  digitalWrite(K, LOW);
```

Anexo XI: Código del juego interactivo

```
//////////////////////////// Interactive Game //////////////////////
void interactive_game(uint8_t num_Led)
{
  char txReactionTime[8];
  float valor_1, valor_2, valor_3, valor_4, valor_5, valor_6;
  uint8_t nivel = 0;
  SetOnLED(num_Led);
  float time_1 = millis();
  float time_3 = 0;
  while (nivel < 2)
  {
    valor_1, valor_2, valor_3, valor_4, valor_5, valor_6 =
get_value_galgas();
    if (num_Led == 1)nivel = lim_1 - valor_1;
    if (num_Led == 2)nivel = lim_2 - valor_2;
    if (num_Led == 3)nivel = lim_3 - valor_3;
    if (num_Led == 4)nivel = lim_4 - valor_4;
    if (num_Led == 5)nivel = lim_5 - valor_5;
    if (num_Led == 6)nivel = lim_6 - valor_6;
    delay(10);
    time_3 = millis();
    if((time_3 - time_1)/1000/3600 >= 0.25)
    {
      inactivity_interactive_game = 1;
      nivel = 30;
    }
  }
  beeps(1);
  SetOffLED(num_Led);
  float time_2 = millis();
  //Convert Float to Char and save in the char variable
  dtostrf((time_2 - time_1)/1000/3600, 2, 2, txReactionTime);


  characteristicTX_4_Time->setValue(txReactionTime); //set the time
value (in hours) in the characteristic
  characteristicTX_4_Time->notify(); // Send the value to the device

}
//////////////////////////////////////////////////////////////////////
```

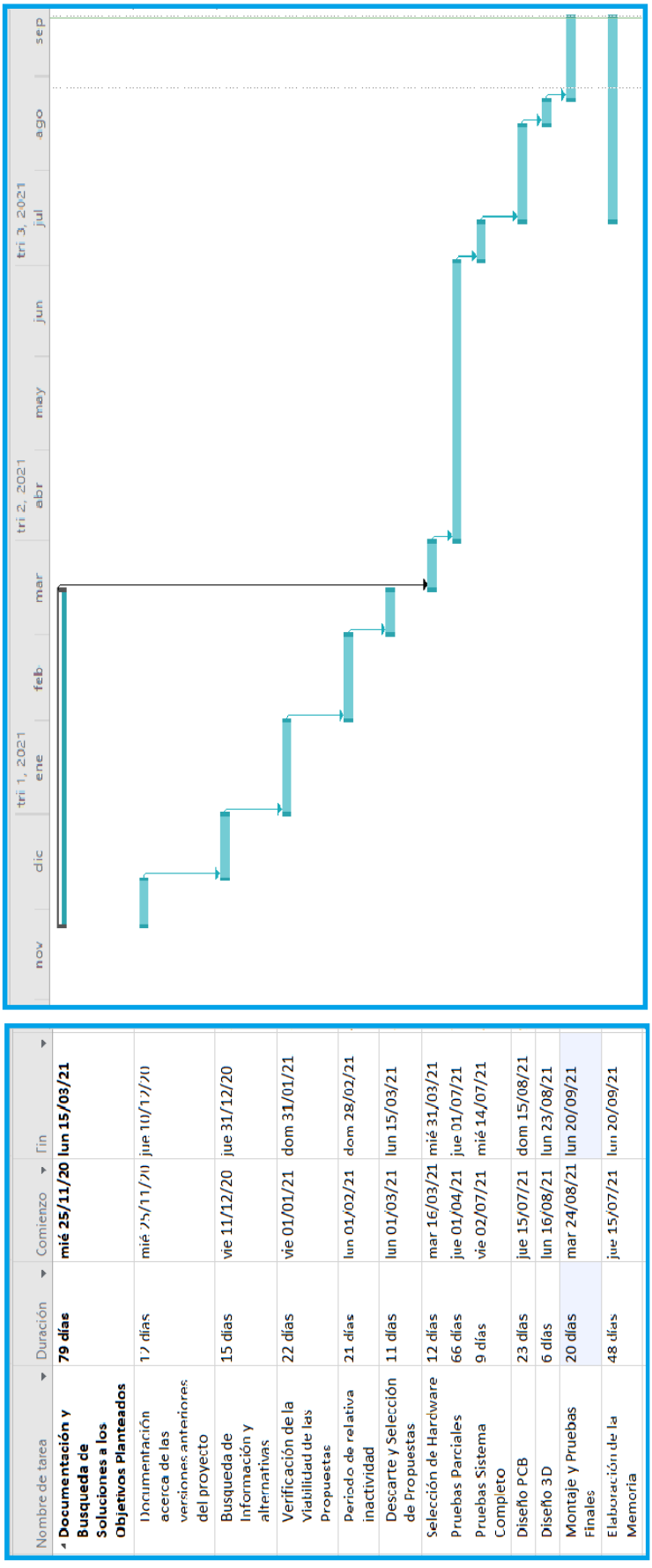
Anexo XII: Código de la función booleana de inactividad

```
////////////////////////////// Inactivity ////////////////////////
boolean inactive ()
{
  if(MODE == 2)
  {
    if( last_shake == shake)
    {
      count_inactivity++;
    }
    else
    {
      count_inactivity = 0;
      last_shake = shake;
    }

    if(count_inactivity >= 3600)return true;
    else return false;
  }

  if(MODE == 3)
  {
    if(inactivity_interactive_game == 1)return true;
    else return false;
  }
}
//////////////////////////////////////////////////////////////////////
```

Anexo XIII: Diagrama de GANTT del Proyecto



Nombre de tarea	Duración	Comienzo	Fin
Documentación y Busqueda de Soluciones a los Objetivos Planteados	**79 días**	**mié 25/11/20**	**lun 15/03/21**
Documentación acerca de las versiones anteriores del proyecto	17 días	mié 25/11/20	jue 10/12/20
Busqueda de Información y alternativas	15 días	vie 11/12/20	jue 31/12/20
Verificación de la Viabilidad de las Propuestas	22 días	vie 01/01/21	dom 31/01/21
Periodo de relativa inactividad	21 días	lun 01/02/21	dom 28/02/21
Descarte y Selección de Propuestas	11 días	lun 01/03/21	lun 15/03/21
Selección de Hardware	12 días	mar 16/03/21	mié 31/03/21
Pruebas Parciales	66 días	jue 01/04/21	jue 01/07/21
Pruebas Sistema Completo	9 días	vie 02/07/21	mié 14/07/21
Diseño PCB	23 días	jue 15/07/21	dom 15/08/21
Diseño 3D	6 días	lun 16/08/21	lun 23/08/21
Montaje y Pruebas Finales	20 días	mar 24/08/21	lun 20/09/21
Elaboración de la Memoria	48 días	jue 15/07/21	lun 20/09/21

*** Periodo de relativa inactividad. Durante este intervalo de tiempo, la carga de trabajo en otras asignaturas, hace que el tiempo dedicado al estudio del proyecto se reduzca a 1 hora diaria máxima.**

*** Pruebas Parciales. El periodo de tiempo dedicado a las pruebas parciales se alarga debido en parte a los tiempos de envio de los componentes, ya que hasta Junio-Julio no se cuenta con todos los componentes necesarios para la realización de las pruebas.**

Anexo XIV: Presupuesto Económico

A continuación, se detallarán los principales costes del presupuesto económico de este proyecto. Se tendrán en cuenta los costes de mano de obra (coste directo), los costes materiales (coste directo) y los costes generales (coste indirecto).

Costes de Mano de Obra

En la Tabla 6 se muestran las horas dedicadas a cada etapa del proyecto, a partir de una estimación en función de la duración de cada etapa y la cantidad de horas medias dedicadas en cada periodo (verse Anexo #).

Tarea	Duración (horas)
Documentación y Búsqueda de Soluciones a los Objetivos Planteados	**250**
Selección de Hardware	**15**
Pruebas Parciales	**120**
Pruebas Sistema Completo	**18**
Diseño PCB	**50**
Diseño 3D	**15**
Montaje y Pruebas Finales	**140**
Elaboración de la Memoria	**150**
Total	**758**

Tabla 6. Horas dedicadas a cada etapa

Según el portal de empleo Jobted, el salario bruto anual de un ingeniero industrial recién egresado en el año 2021en España es de 20.500€ [64]. Teniendo en cuenta que el número de horas laborales anuales son de 1760 horas, obtenemos un precio de 11,65€/hora, obtenemos el coste de mano de obra del proyecto (Tabla 7).

Horas trabajadas	Sueldo por hora	Total
758	**11,65 €**	**8.831 €**

Tabla 7. Coste de mano de obra

Costes Materiales

A continuación, se muestra la Tabla 8 con los principales costes materiales en este proyecto.

	Precio de compra	Uso en meses	Amortiz. en años	TOTAL
Ordenador personal	500,00 €	10	4	104,17 €
Software Microsoft	279,00 €	10	4	58,13 €
Software Autodesk (Suscripción anual)	2.342,00 €	10	1	1.951,67 €
BOM1 (5 SmartCubes)	15,30 €	2	10	0,26 €
BOM2 (5 SmartCubes)	36,15 €	2	10	0,60 €
BOM3 (5 SmartCubes)	23,25 €	2	10	0,39 €
ABOM4 (5 SmartCubes)	16,05 €	2	10	0,27 €
BOM5 (5 SmartCubes)	80,95 €	2	10	1,35 €
BOM6 (5 SmartCubes)	20,65 €	2	10	0,34 €
PCBs (5 SmartCubes)	30,00 €	1	10	0,25 €
Capacímetro	20,00 €	5	10	0,83 €
Multímetro	30,00 €	5	10	1,25 €
Estación de Soldadura 8786D	80,00 €	5	4	8,33 €
Elementos de soldadura	15,00 €	5	4	1,56 €
Materiales Electrónicos para Pruebas	50,00 €	5	10	2,08 €
Otros materiales de pruebas	40,00 €	5	10	1,67 €
Impresora 3D	120,00 €	5	10	5,00 €
TOTAL				**2.138,14 €**

Tabla 8. Costes Materiales del Proyecto

Otros costes

En este apartado, únicamente se han tenido en cuenta los costes de impresión, encuadernación y otros costes derivados de este proceso (Tabla 9).

MATERIAL FUNGIBLE	
Impresión y encuadernación	**100,00 €**
Otros	**30,00 €**

Tabla 9. Material fungible

Presupuesto General

Se obtiene el coste total del proyecto tras tener en cuenta los costes generales (15% de los costes directos), el beneficio industrial (6% de los costes directos e indirectos) y el Impuesto sobre el Valor Añadido (21%), como se muestra en la Tabla 10.

Desglose Presupuesto Final	
Costes mano de obra	**8.831,00 €**
Costes materiales	**2.138,14 €**
Material fungible	**130,00 €**
Costes generales	**1.645,33 €**
Beneficio Industrial	**756,85 €**
Subtotal Presupuesto	**13.501,02 €**
IVA	**2.835,21 €**
Presupuesto final	**16.336,23 €**

Tabla 10. Desglose Presupuesto Final

Anexo XV: BOM de la Cara 1 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
RLED1	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA1	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
RBAT1	RES SMD 1M OHM 0.1% 1/8W	0805 (2012 Metric)	Yageo	RT0805BRD071ML	1	https://www.mouser.es/ProductDetail/Yageo/RT0805BRD071ML?qs=l8tafrjApPBjvuUsmeHy6A%3D%3D
RBAT2	RES SMD 2M OHM 0.5% 1/4W	0805 (2012 Metric)	KOA Speer	RS73G2ATTD2004D	1	https://www.mouser.es/ProductDetail/KOA-Speer/RS73G2ATTD2004D?qs=sGAEpiMZZMtlubZbdhIBIHFjskMP2WdR5xz8vGMd8mw%3D
LED1	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
C1	CAP MLCC 1UF 50V X7R 1206 20%	1206 (3216 Metric)	KEMET	C1206F105M5RACAUTO7210	1	https://www.mouser.es/ProductDetail/KEMET/C1206F105M5RACAUTO7210?qs=ds50AKTGxA8F1MxATCT6sg%3D%3D
C2	CAP MLCC 0.1UF 250V X7R 1206 5%	1206 (3216 Metric)	KEMET	C1206X104JARECAUTO	1	https://www.mouser.es/ProductDetail/KEMET/C1206X104JARECAUTO?qs=55YtniHzbhCFPc2qUDRYlA%3D%3D
Reg3.3	LDO Voltage Regulators 3.3V	SOT-25-5	Toshiba	TCR2EF33,LM(CT	1	https://www.mouser.es/ProductDetail/Toshiba/TCR2EF33LMCT?qs=EPmvyOv1YlPmhsvS3PaWfg%3D%3D

Anexo XVI: BOM de la Cara 2 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
SSR	1-FORM-C SS RELAY	SMD DIP-8	IXYS Integrated Circuits	LCC120S	1	https://www.mouser.es/ProductDetail/IXYS-Integrated-Circuits/LCC120S?qs=IQlU%252BwB40gu1jkuBj5hvDQ%3D%3D
RLED2	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA2	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
U1	IC VOLT DET 3.08V LOW SC70-3	SC-70, SOT-323	Microchip Technology	MCP112T-315E/LB	1	http://www.digikey.es/product-detail/en/microchip-technology/MCP112T-315E%2FLB/MCP112T-315E%2FLBCT-ND/1979775
U2	IC LOAD SWITCH 100US SOT-26	SOT-23-6	Diodes Incorporated	AP2281-3WG-7	1	http://www.digikey.com/product-detail/es/diodes-incorporated/AP2281-3WG-7/AP2281-3WG-7DICT-ND/2270160
LED2	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
C3	CAP MLCC 1UF 50V X7R 1206 20%	1206 (3216 Metric)	KEMET	C1206F105M5RACAUTO7210	1	https://www.mouser.es/ProductDetail/KEMET/C1206F105M5RACAUTO7210?qs=ds50AKTGxA8F1MxATCT6sg%3D%3D
C4	CAP MLCC 0.1UF 250V X7R 1206 5%	1206 (3216 Metric)	KEMET	C1206X104JARECAUTO	1	https://www.mouser.es/ProductDetail/KEMET/C1206X104JARECAUTO?qs=55YtniHzbhCFPc2qUDRYlA%3D%3D

Anexo XVII: BOM de la Cara 3 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
ESP32	WiFi Modules - 802.11	ESPWROOM32D	Espressif Systems	ESP-WROOM-32D	1	https://www.mouser.es/ProductDetail/Espressif-Systems/ESP-WROOM-32D?qs=MLItCLRbWsyWVKgdQMcFuQ%3D%3D
RLED3	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA3	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
LED3	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
C5	CAP MLCC 0.1UF 250V X7R 1206 5%	1206 (3216 Metric)	KEMET	C1206X104JARECAUTO	1	https://www.mouser.es/ProductDetail/KEMET/C1206X104JARECAUTO?qs=55YtniHzbhCFPc2qUDRYlA%3D%3D

Anexo XVIII: BOM de la Cara 4 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
RLED4	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA4, RTRT1, RTRT2, REN	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	4	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
RTRT_NPN	RES SMD 4.7K OHM 1% 1/8W	0805 (2012 Metric)	Yageo	RC0805FR-074K7L	1	http://www.digikey.com/product-detail/en/yageo/RC0805FR-074K7L/311-4.70KCRCT-ND/730876
TRT_NPN	TRANS NPN 40V 0.6A	TO-236-3, SC-59, SOT-23-3	ON Semiconductor	MMBT2222ALT1G	1	http://www.digikey.com/product-detail/es/on-semiconductor/MMBT2222ALT1G/MMBT2222ALT1GOSCT-ND/1139806
LED4	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
Q1, Q2	TRANS NPN 40V 1.5A	TO-236-3, SC-59, SOT-23-3	Comchip Technology	SS8050-G	2	https://www.digikey.com/es/products/detail/comchip-technology/SS8050-G/6138901
C6	CAP CER 100UF 6.3V X5R 1206	1206 (3216 Metric)	KEMET	C1206C107M9PACTU	1	https://www.digikey.es/product-detail/es/kemet/C1206C107M9PACTU/399-5620-1-ND/2057834
C7	CAP MLCC 0.1UF 250V X7R 1206 5%	1206 (3216 Metric)	KEMET	C1206X104JARECAUTO	1	https://www.mouser.es/ProductDetail/KEMET/C1206X104JARECAUTO?qs=55YtniHzbhCFPc2qUDRYlA%3D%3D

Anexo XIX: BOM de la Cara 5 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
RLED5	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA5	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
LED5	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
MPU9150	Módulo Acelerometro + Giroscopio MPU-9150 9 Axis Ejes MPU 9150	THT	Redplanet	8281-20	1	https://www.amazon.es/M%C3%B3dulo-Acelerometro-Giroscopio-MPU-9150-Arduino/dp/B075XSBT3Y

Anexo XX: BOM de la Cara 6 (*Bill of Material*)

Ref	Value	Package/Case	Manufacture	Manufacture Part Number	Quantity	Specification
RLED6	RES SMD 330 OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-07330RL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-07330RL/311-330ARCT-ND/731258
RGALGA6	RES SMD 10K OHM 5% 1/8W	0805 (2012 Metric)	Yageo	RC0805JR-0710KL	1	http://www.digikey.com/product-detail/en/yageo/RC0805JR-0710KL/311-10KARCT-ND/731188
LED6	LED RED DIFFUSED 0805 SMD	0805 (2012 Metric)	OSRAM Opto Semiconductors Inc.	LH R974-LP-1	1	http://www.digikey.es/product-detail/en/osram-opto-semiconductors-inc/LH%20R974-LP-1/475-1415-1-ND/1802604
BUZZER	BUZZER PIEZO 5V 11 X 9MM SMD	CMT-1102	CUI Inc.	CMT-1102-SMT-TR	1	https://www.digikey.com/es/products/detail/cui-inc/CMT-1102-SMT-TR/610972

Anexo XXI: Modelos 3D de las PCB diseñadas

- **Cara 1**

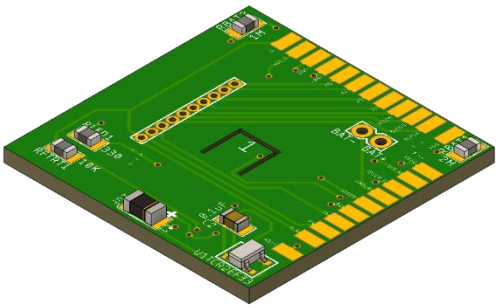


Figura 99. Modelo 3D. Cara 1. Capa TOP

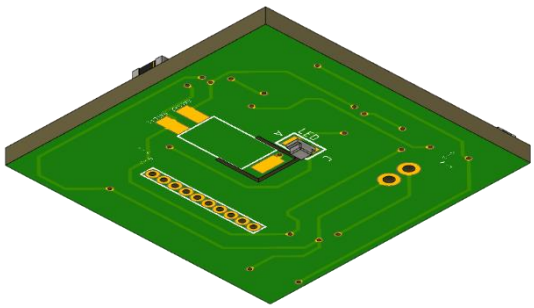


Figura 98. Modelo 3D. Cara 1. Capa BOTTOM

- **Cara 2**

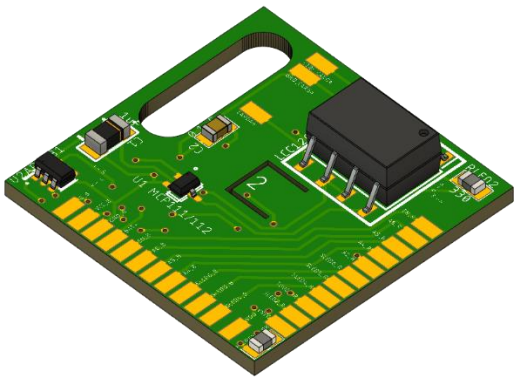


Figura 101. Modelo 3D. Cara 2. Capa TOP

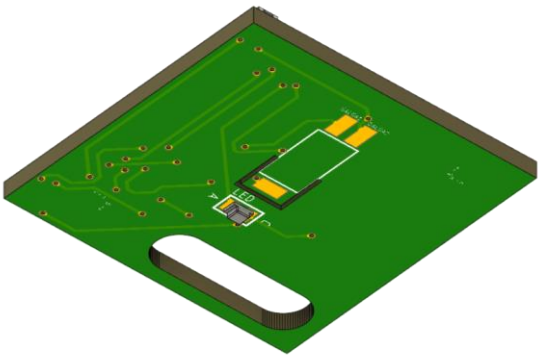


Figura 100. Modelo 3D. Cara 2. Capa BOTTOM

- **Cara 3**

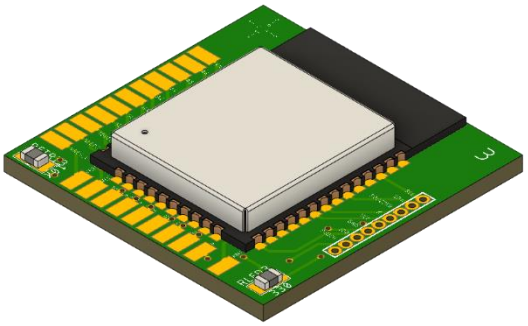


Figura 102. Modelo 3D. Cara 3. Capa TOP

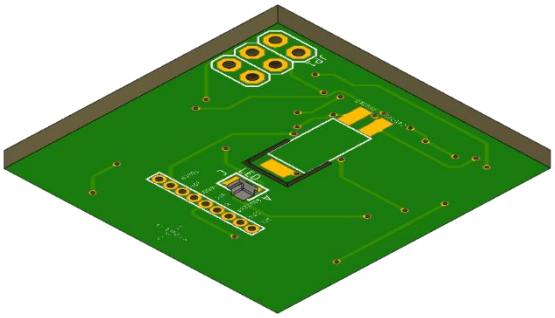


Figura 103. Modelo 3D. Cara 3. Capa BOTTOM

- **Cara 4**

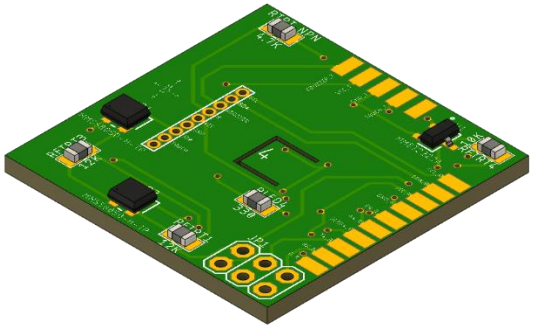


Figura 107. Modelo 3D. Cara 4. Capa TOP

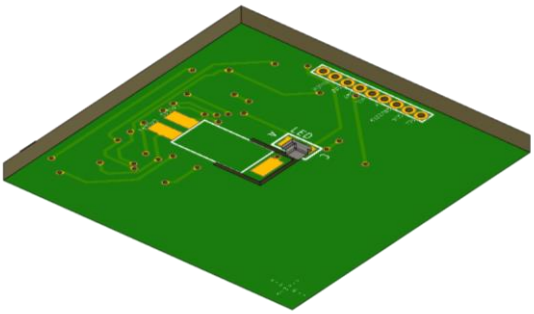


Figura 106. Modelo 3D. Cara 4. Capa BOTTOM

- **Cara 5**

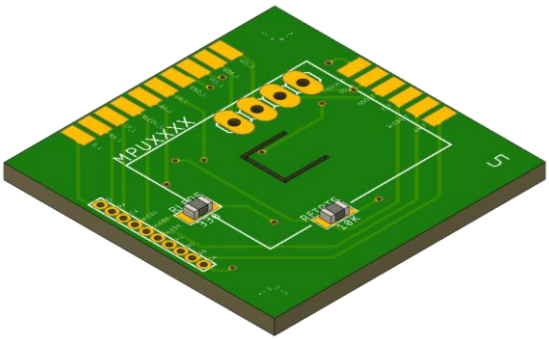


Figura 109. Modelo 3D. Cara 5. Capa TOP

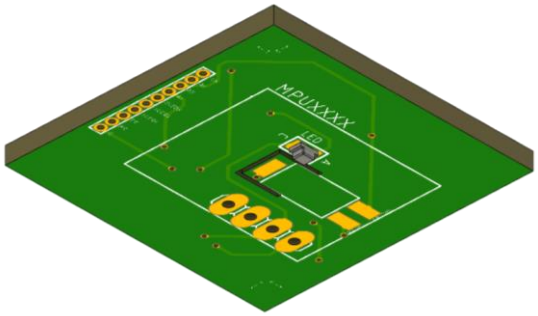


Figura 108. Modelo 3D. Cara 5. Capa BOTTOM

- **Cara 6**

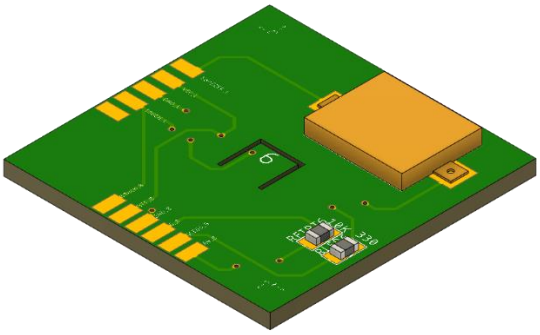


Figura 111. Modelo 3D. Cara 6. Capa TOP

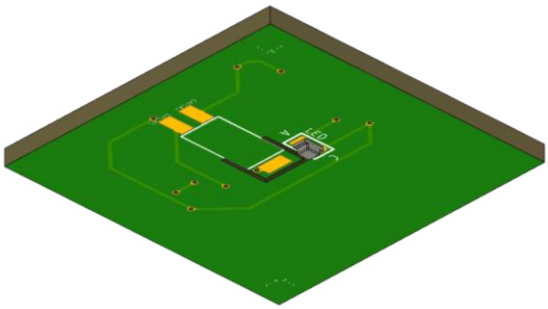


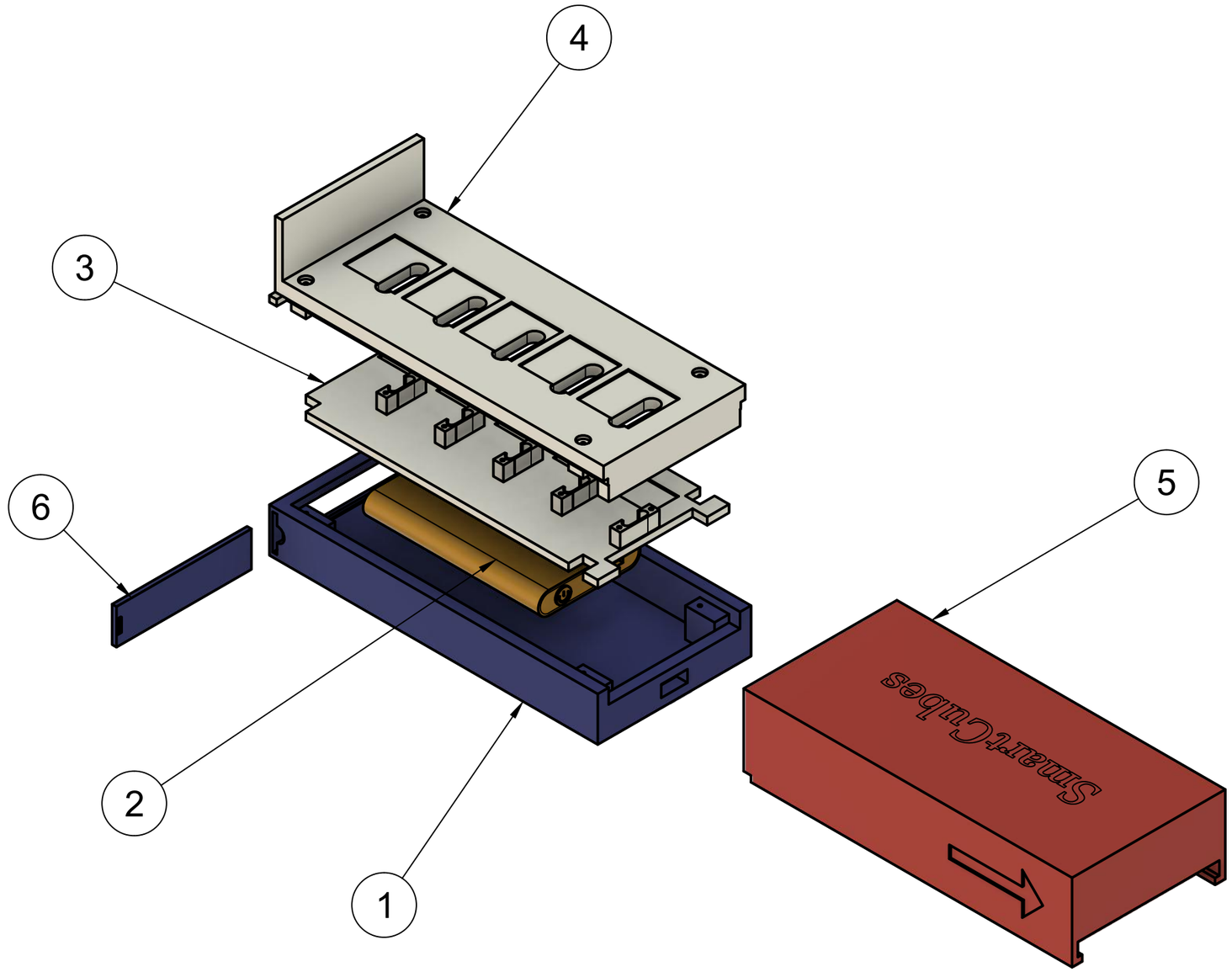
Figura 110. Modelo 3D. Cara 6. Capa BOTTOM

Anexo XXII: Planos de las piezas 3D diseñadas

Se muestra una tabla con los diferentes planos de los dos diseños 3D que se han realizado, la plataforma de carga y la carcasa de los cubos.

	Plataforma de carga
Plano 1	Explosionado del conjunto
Plano 2	Base
Plano 3	Plataforma de los módulos de carga
Plano 4	Plataforma de los SmartCubes
Plano 5	Tapa superior
Plano 6	Tapa lateral
	Carcasa
Plano 1	Explosionado del conjunto
Plano 2	Mitad 1
Plano 3	Mitad 2
Plano 4	Tapa conector

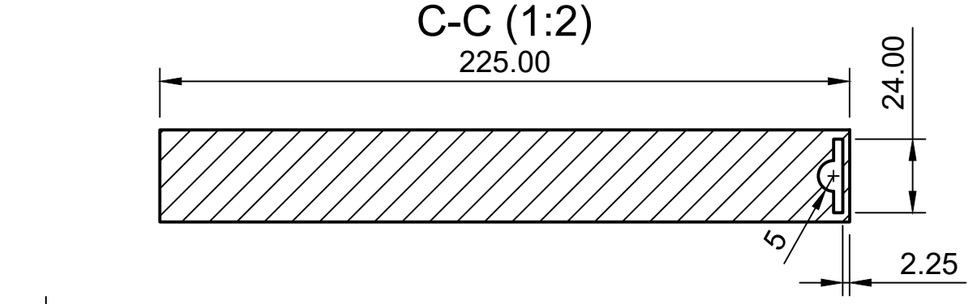
Tabla 11. Planos



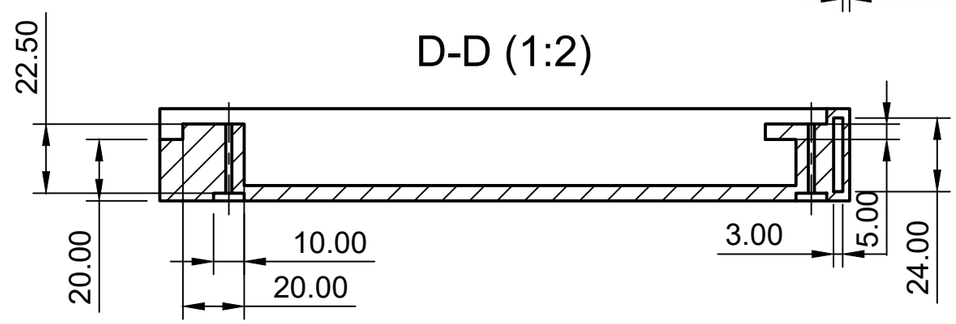
Lista de piezas	
Elemento	**Pieza**
1	Base
2	Banco de energía
3	Plataforma SmartCubes
4	Plataforma módulos
5	Tapa superior
6	Tapa lateral

Universidad de Alcalá	Grado en Ingeniería Electrónica y Automática Industrial		
Fecha: 14/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Plataforma de Carga	Plano Explosionado	Escala: 1:2.5	Hoja 1

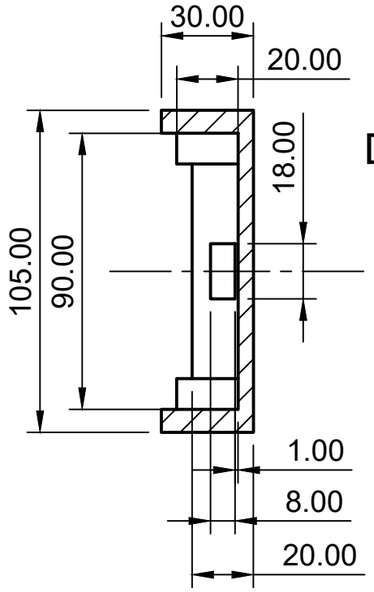
C-C (1:2)

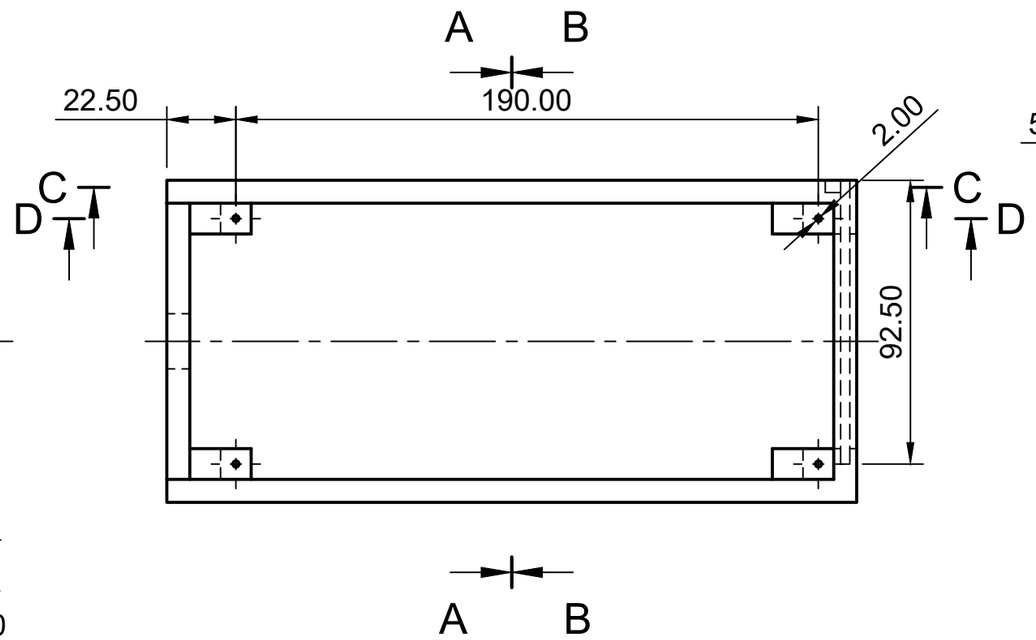


D-D (1:2)

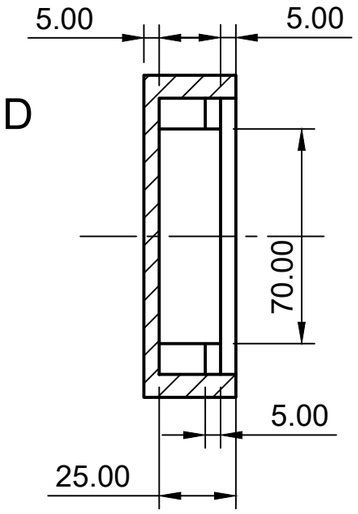


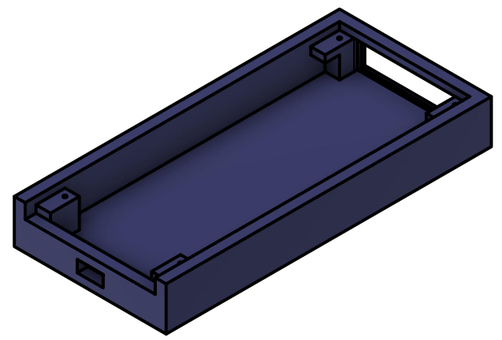
B-B (1:2)





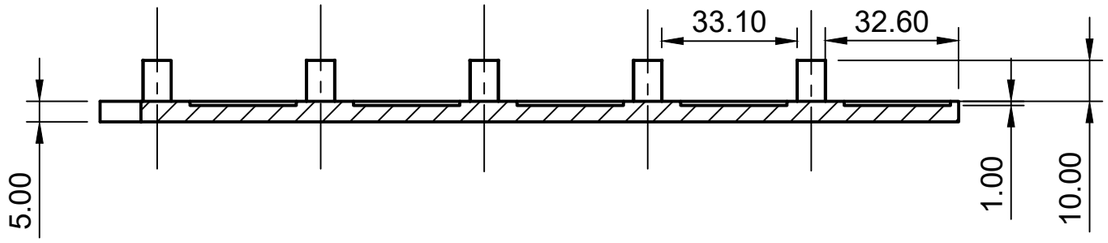
A-A (1:2)



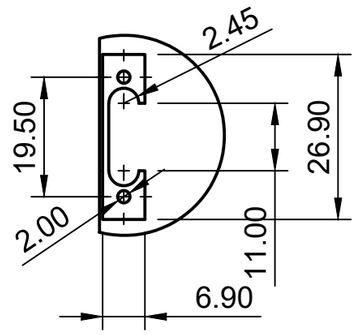


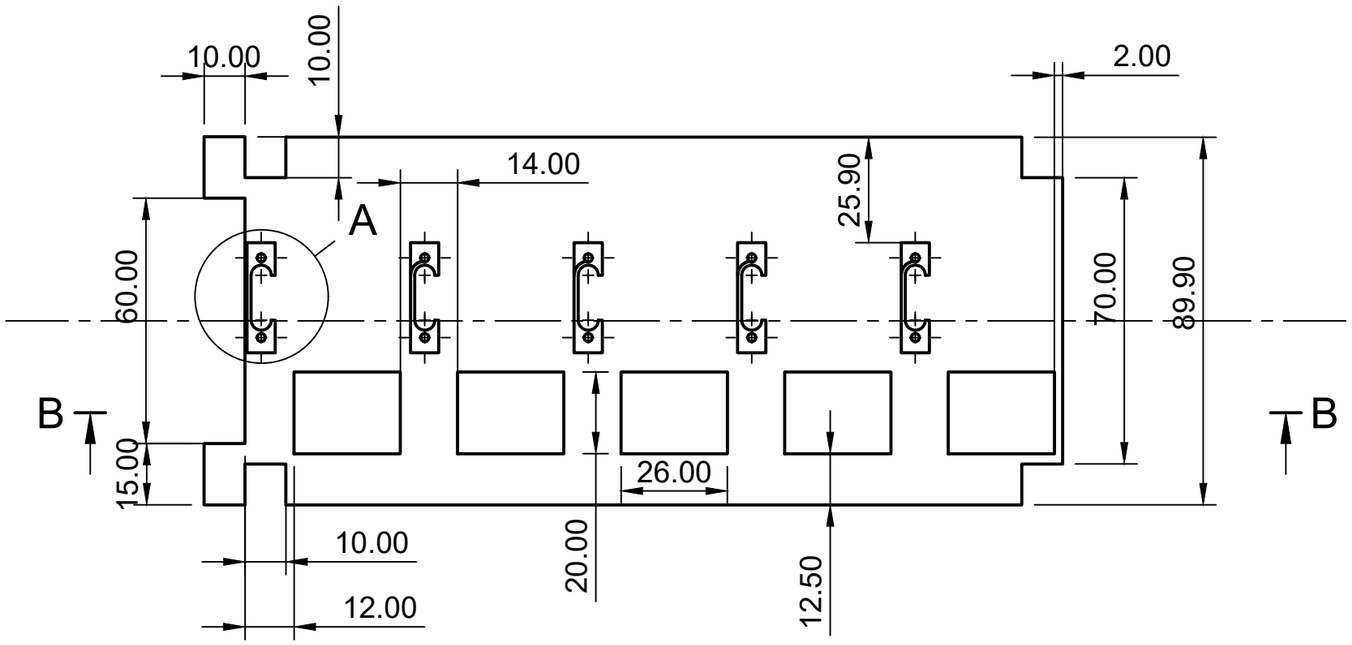
Universidad de Alcalá	Grado en Ingeniería Electrónica y Automática Industrial		
Fecha: 14/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Plataforma de Carga	Pieza 1: Base	Escala: 1:2	Hoja 2

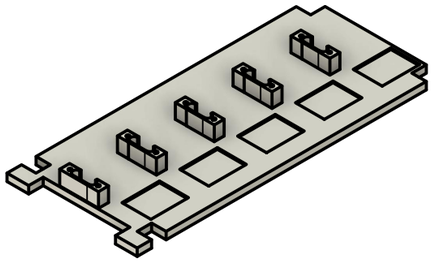
B-B (1:1.5)



A (1:1)

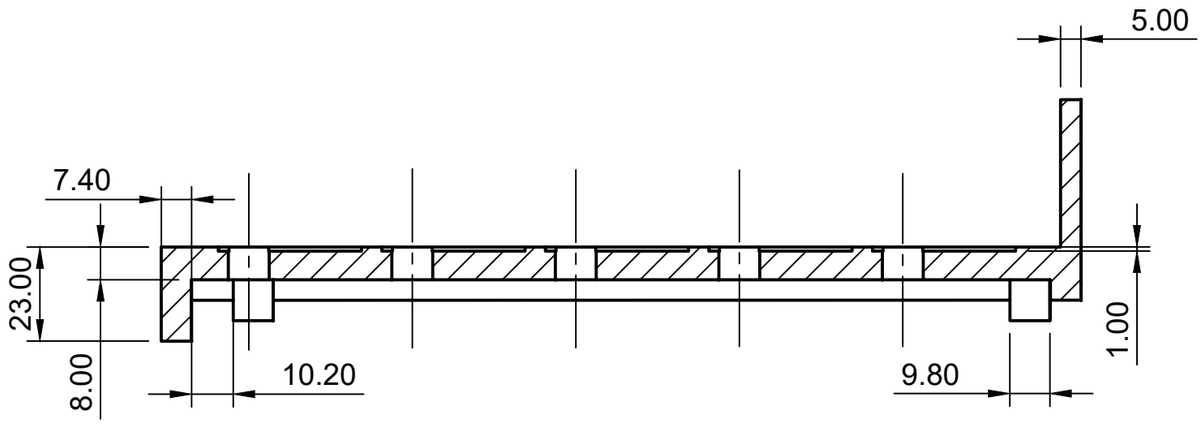




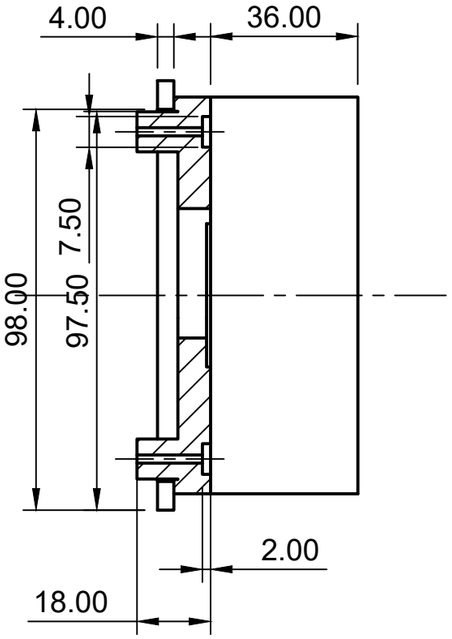


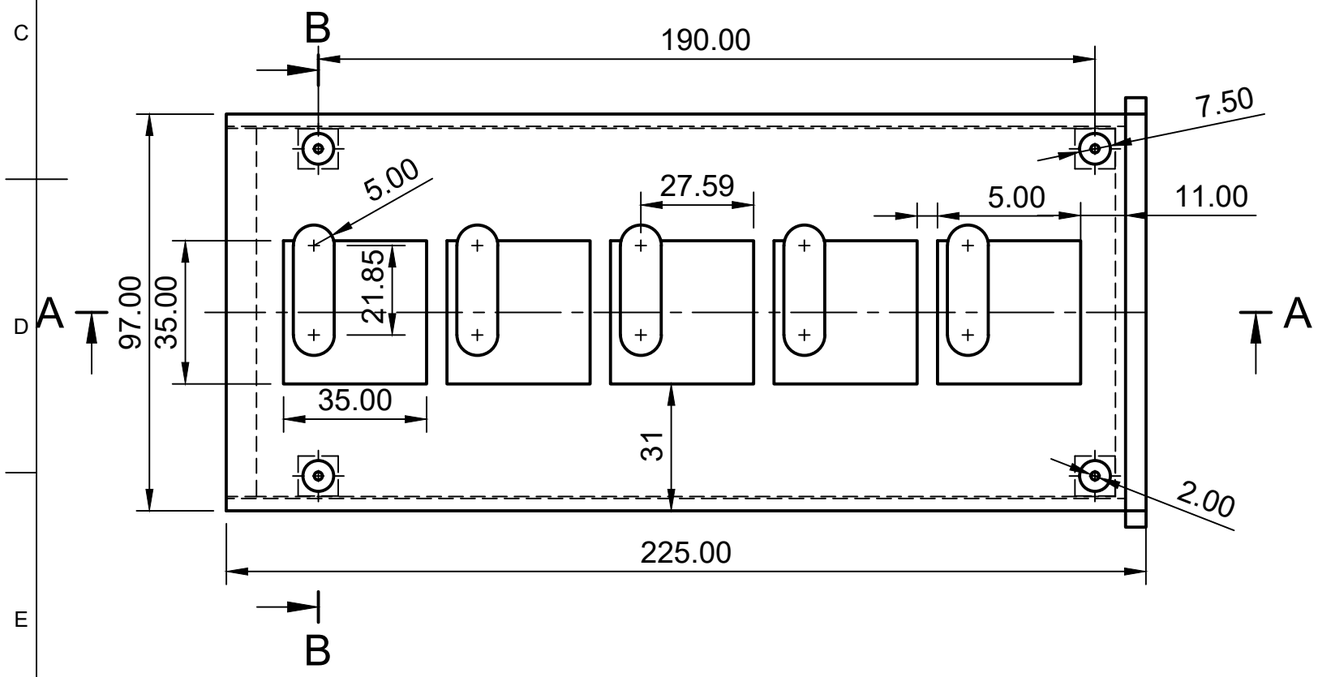
Universidad de Alcalá	Grado en Ingeniería Electrónica y Automática Industrial		
Fecha: 14/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Plataforma de Carga	Pieza 3: Plataforma de los módulos de carga	Escala: 1:1.5	Hoja 3

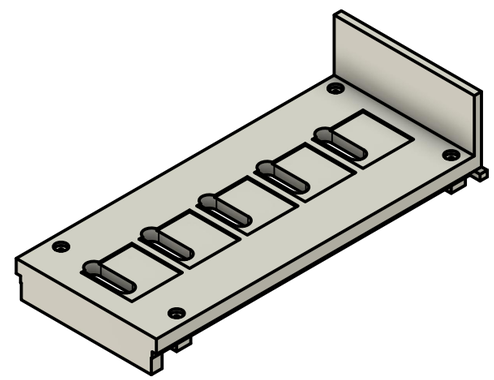
A-A (1:1.5)



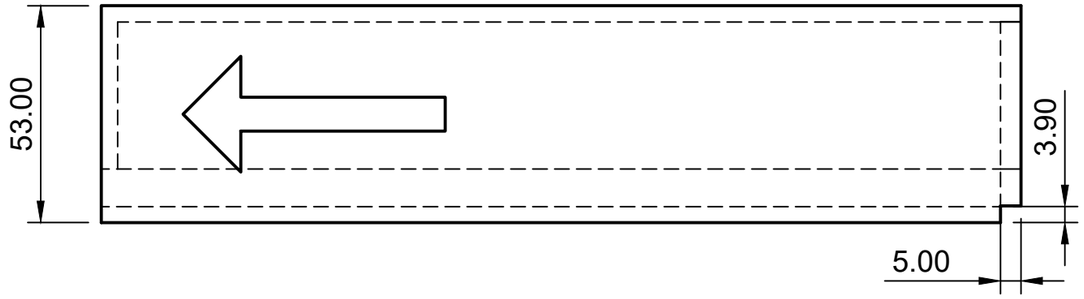
B-B (1:1.5)

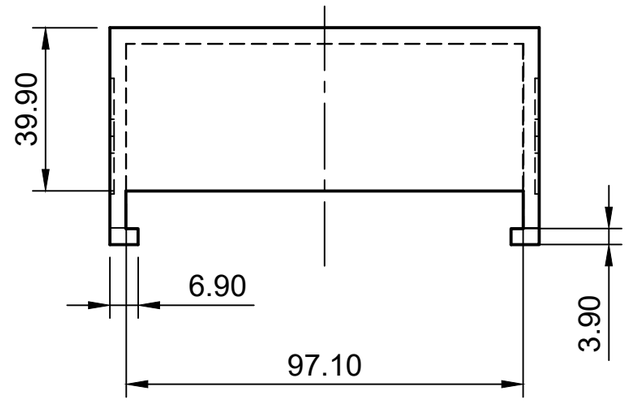


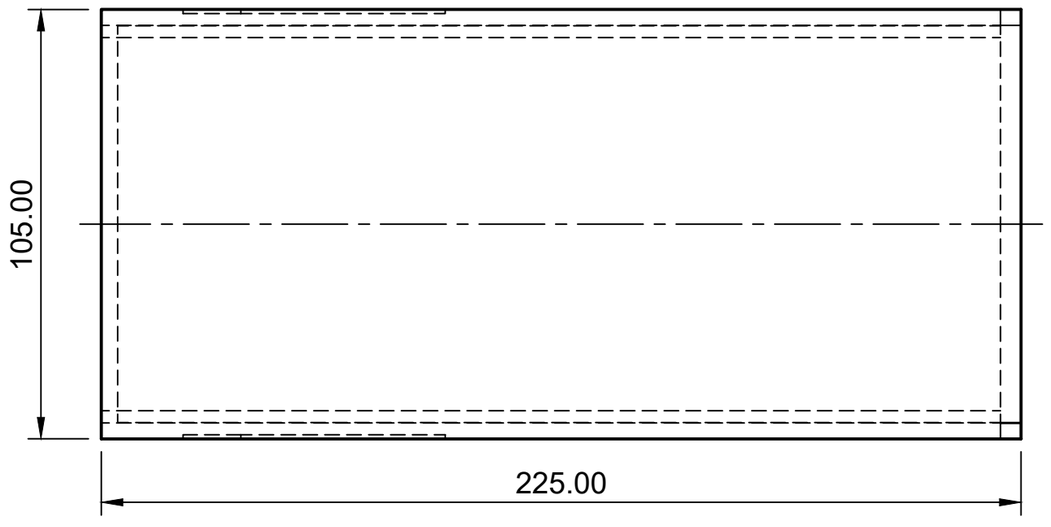


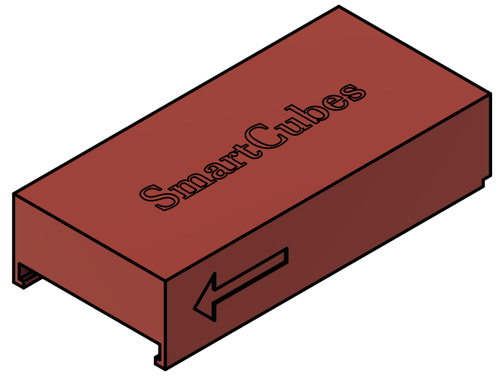


Universidad de Alcalá	Grado en IngenieríaElectrónica y Automática Industrial	
Fecha: 14/09/2021	Autor: Álvaro Matellanes Fernández	Entregado a: Bernardo Alarcos Alcázar
Plataforma de Carga	Pieza 3: Plataforma de los SmartCubes	Escala: 1:1.5 / Hoja 4



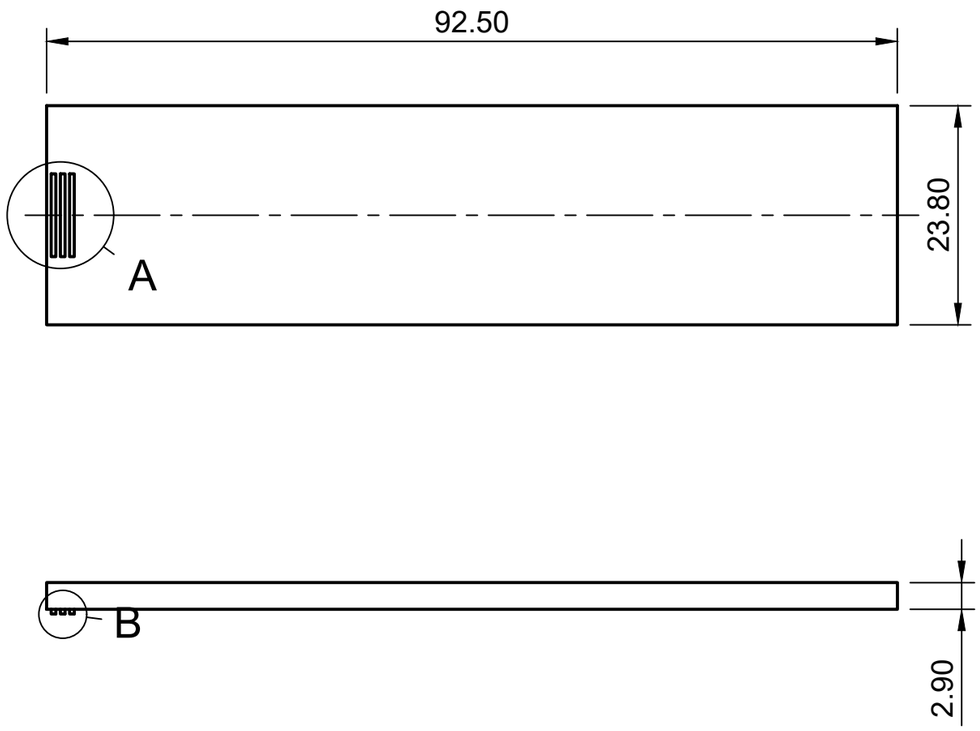




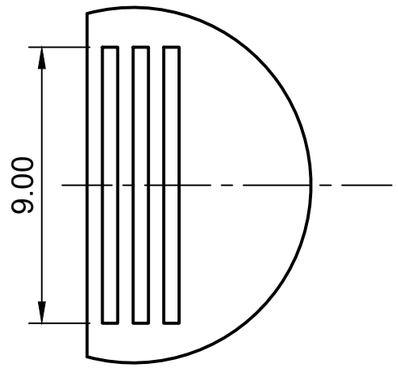


Nota:
Profundidad del repujado lateral equivale a 1 mm

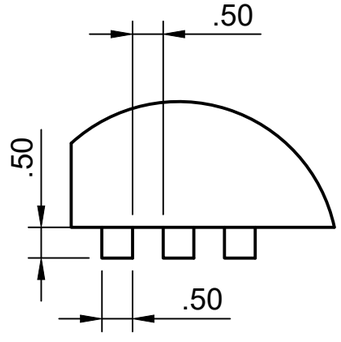
Universidad de Alcalá		Grado en Ingeniería Electrónica y Automática Industrial	
Fecha: 14/09/2021	Autor: Álvaro Matellanes Fernández	Entregado a: Bernardo Alarcos Alcázar	
Plataforma de Carga	Pieza 5: Tapa superior	Escala: 1:1.5	Hoja 5

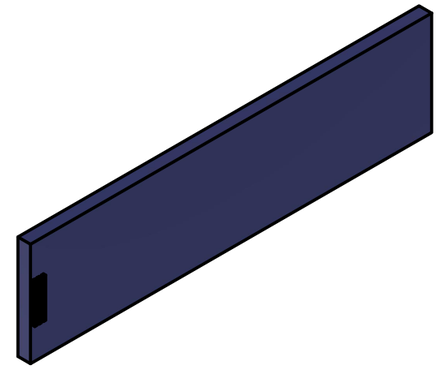


A (5:1)

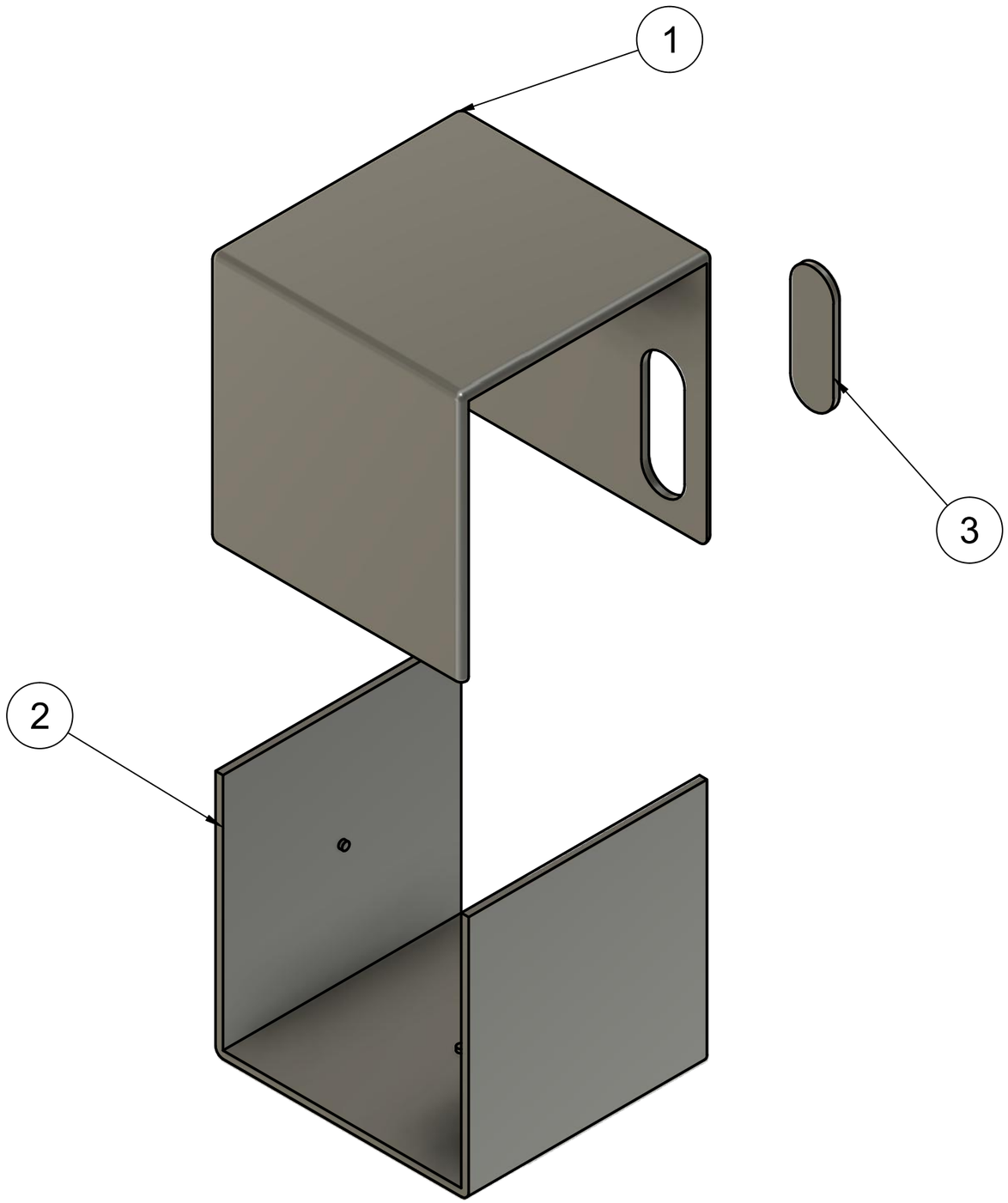


B (10:1)





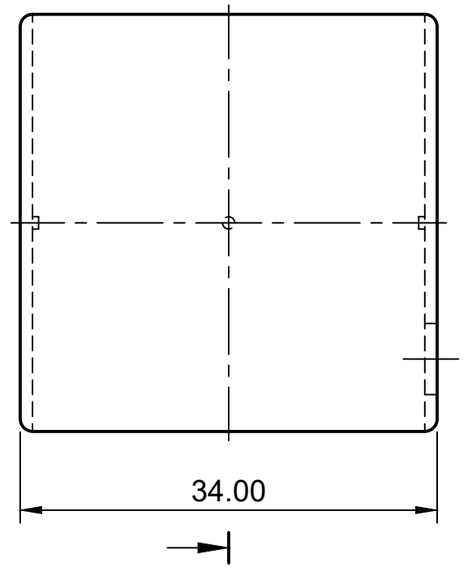
Universidad de Alcalá	Grado en Ingeniería Electrónica y Automática Industrial		
Fecha: 14/9/2021	Autor: Álvaro Matellanes Fernández	Entregado a: Bernardo Alarcos Alcázar	
Plataforma de Carga	Pieza 6: Tapa lateral	Escala: 1.5:1	Hoja 6



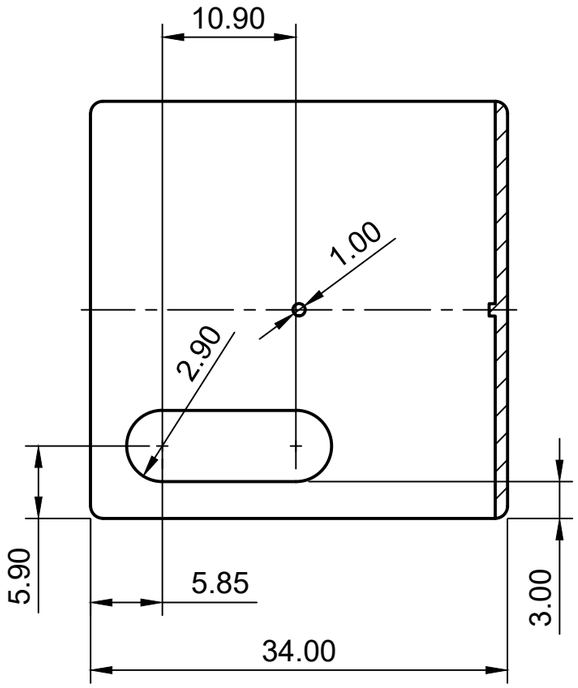
Lista de piezas	
Elemento	Pieza
1	Carcasa mitad 1
2	Carcasa mitad 2
3	Tapa Conector

Universidad de Alcalá		Grado en Ingeniería Electrónica y Automática Industrial	
Fecha : 19/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Carcasa SmartCube	Plano Explosionado	Escala: 2:1	Hoja 1

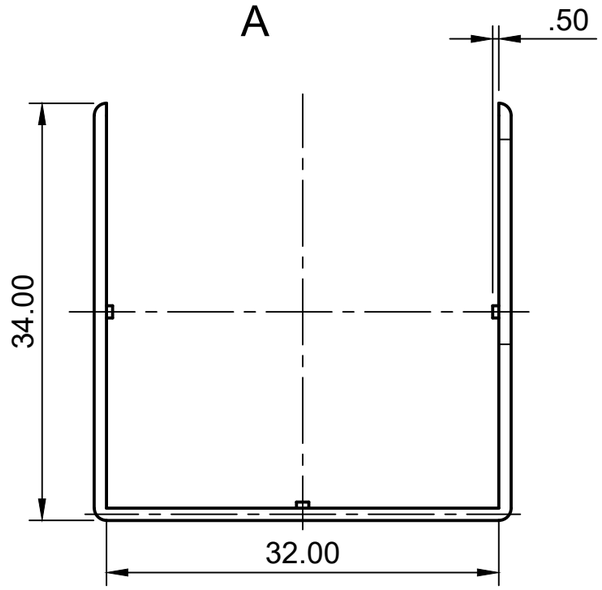
A

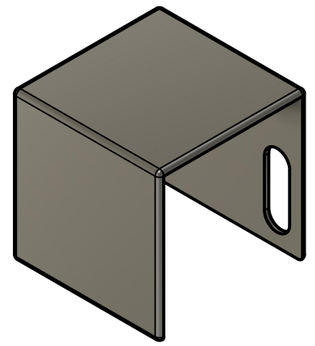


A-A (2:1)



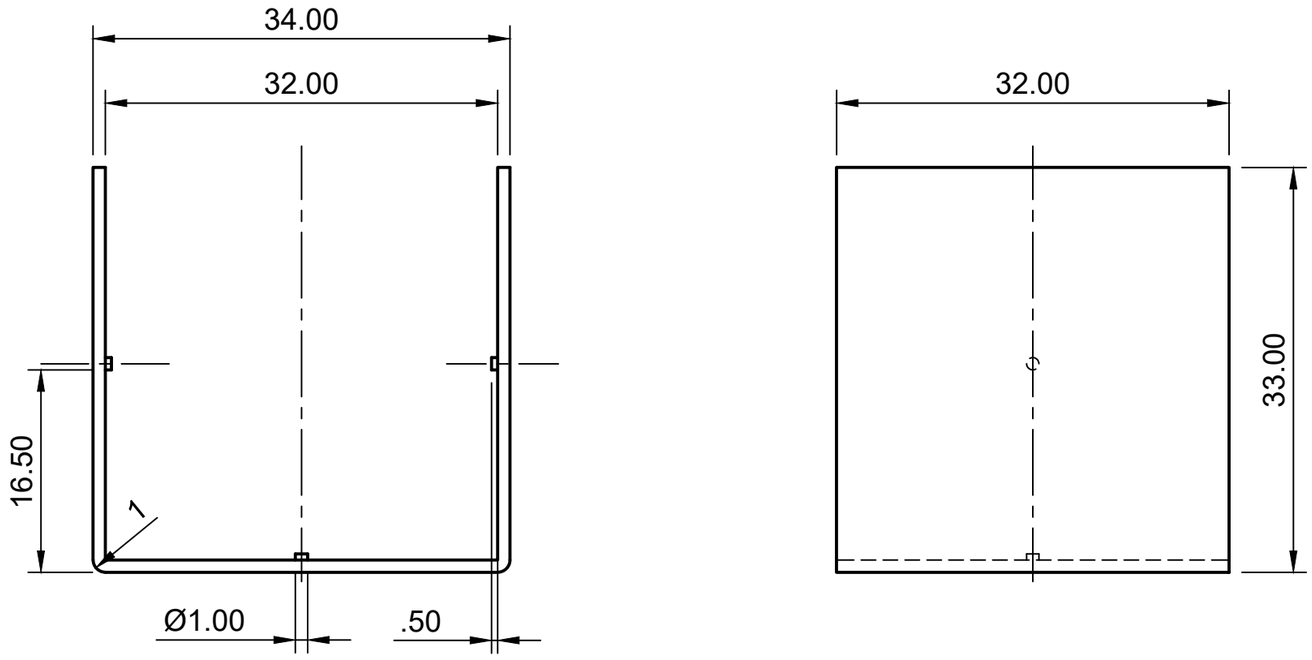
A

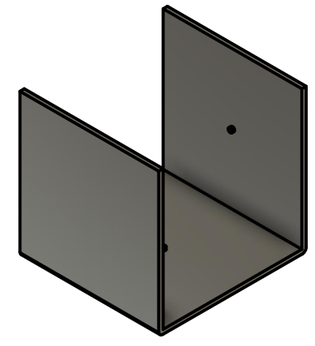




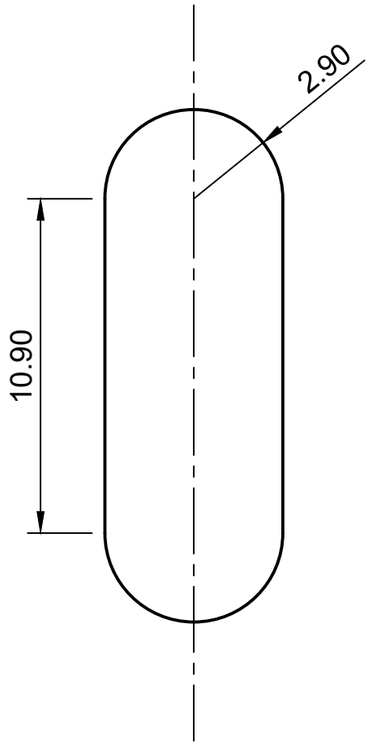
Nota:
Radio de todos los empalmes en plano equivalen a 1 mm

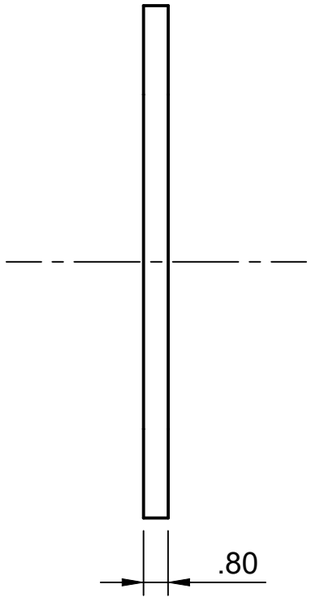
Universidad de Alcalá	Grado en Ingeniería Electrónica y automática Industrial		
Fecha: 19/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Carcasa Smartcube	Pieza 1: Midad 1	Escala: 2:1	Hoja 2

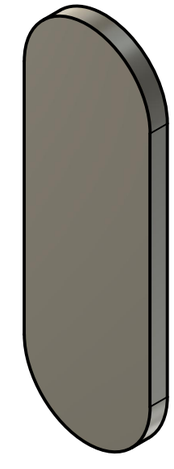




Universidad de Alcalá	Grado en Ingeniería Electrónica y Automática Industrial		
Fecha: 19/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Carcasa SmartCube	Pieza 2: Mitad 2	Escala: 2:1	Hoja 3







Universidad de Alcalá		Grado en Ingeniería Electrónica y Automática Industrial	
Fecha: 19/09/2021	Autor: Álvaro Matellanes Fernández	Presentado a: Bernardo Alarcos Alcázar	
Carcasa SmartCube	Pieza 3: Tapa Conector	Escala: 5:1	Hoja 4

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR