UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE UN SISTEMA DE DETECCIÓN DE ANOMALÍAS OPERACIONALES EN PROCESOS INDUSTRIALES BASADOS EN SENSORES IOT

POR

Moisés Alejandro Barraza Riquelme

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero(a) Civil Electrónico(a)

Profesor Guía
Pablo Esteban Aqueveque Navarro

Agosto 2022
Concepción (Chile)

A mis padres Jeny y Moisés, por guiarme a ser la persona que soy y por haberme dado la oportunidad de estudiar. A mi hermana Rosa, por su compañía y risas infinitas.

Agradecimientos

Primeramente doy gracias a mis profesores de la Facultad de Ingeniería, por motivarme a cuestionarme y analizar siempre los cómo y los porqué de las cosas. A mis compañeros y amigos que conocí en la universidad por la buena compañía, por las subidas de ánimo en momentos oportunos, por las largas jornadas de estudio e inolvidables tardes recreativas.

Agradecer también a mi polola Fernanda, por estar siempre conmigo y levantarme el ánimo en los momentos más difíciles. Por su cariño, comprensión y paciencia infinita.

Finalmente agradecer a mis compañeros del Laboratorio de Instrumentación y Desarrollo de la Facultad de Ingeniería por su buena disposición a resolver dudas y hacer de este proceso algo mucho mas ameno. A Luciano, Francisco y Martín por sus constantes comentarios, consejos e ideas durante el desarrollo del trabajo.

Nuevamente gracias a mi familia, ya que sin ellos nada de esto hubiese sido posible. Infinitas gracias.

Sumario

El trabajo desarrollado en esta memoria presenta un estudio del arte actual de las tecnologías IoT con fines industriales, se discuten sus capacidades, beneficios y desafíos que se presentan a la hora de abrir el camino a la revolución de la Industria 4.0.

Se propone la identificación del estado de salud de motores eléctricos a partir del análisis de sus vibraciones de operación. Para esto se desarrolla un sistema electrónico embebido, compuesto principalmente por un microprocesador y un sensor inercial, capaz de conectarse a una red inalámbrica por la cual poder enviar un mensaje de alerta en el caso de que se compruebe, por medio de su vibración, que un motor eléctrico está operando fuera de las condiciones esperadas.

El trabajo involucra el diseño de una red inalámbrica de características industriales, el diseño y desarrollo de una placa electrónica capaz de funcionar por sí sola, y el software necesario para el control del hardware, el análisis de la vibración y la determinación de fallas operacionales.

Se desarrollan dos algoritmos de identificación de fallas que se ejecutan en el microprocesador del dispositivo embebido, los cuales por medio de operaciones matemáticas realizadas en una ventana de muestras de acelerómetro pueden describir el comportamiento de las vibraciones en el dominio del tiempo y en el de la frecuencia. Finalmente comparan estos resultados con parámetros de referencia y envía un mensaje de alerta en caso de haber diferencias relevantes.

Se implementa el sistema en laboratorio y se realizan pruebas que permiten concluir con el cumplimiento de todas las características establecidas en su diseño. Se comprueba el funcionamiento esperado del dispositivo embebido, sus algoritmos y la red propuesta. Se presentan además las diferencias entre los resultados del dispositivo embebido y los que se obtienen en un software matemático avanzado.

Summary

The work developed in this memory presents a review and state of art of IoT technologies for industrial purposes, discussing their capabilities, benefits and challenges that arise when opening the way to the Industry 4.0 revolution.

The identification of the state of health of electric motors is proposed from the analysis of their operating vibrations. For this, an embedded electronic system is developed, composed mainly of a microprocessor and an inertial sensor, capable of connecting to a wireless network through which to send an alert message if it is verified, by means of its vibration, that an electric motor is operating outside the expected conditions.

The work involves the design of a wireless network with industrial characteristics, the design and development of an electronic board capable of operating on its own, and the necessary software for hardware control, vibration analysis and determination of operational faults.

Two fault identification algorithms are developed that are executed in the microprocessor of the embedded device, which by means of mathematical operations carried out in an accelerometer sample window can describe the behavior of vibrations in the time domain and in the frequency domain. Finally, they compare these results with reference parameters and send an alert message in case there are relevant differences.

The system is implemented in the laboratory and tests are carried out that allow concluding with the fulfillment of all the characteristics established in its design. The expected performance of the embedded device, its algorithms and the proposed network are checked. The differences between the results of the embedded device and those obtained in advanced mathematical software are also presented.

Índice

Índice de Figuras

CAPÍTULO 1

Introducción

1.1. Introducción General

En este informe se presenta el trabajo desarrollado para el diseño de un sistema IoT en el que un dispositivo embebido será capaz de enviar mensajes de alarma de forma inalámbrica cuando se detecten anomalías relacionadas a las vibraciones de operación de un motor eléctrico. Estas anomalías se identificarán mediante el desarrollo de algoritmos matemáticos, computados en un microprocesador, a partir de la información obtenida de un sensor inercial montado sobre el motor a analizar. El dispositivo embebido, además de ejecutar los algoritmos, será capaz de crear y gestionar su propia red inalámbrica, permitiendo la comunicación entre varias unidades del mismo. Esta red de dispositivos se diseñará para futura implementación en plantas con numerosas máquinas eléctricas operativas.

El proyecto se desarrollará para uso industrial, con el objetivo de implementar una red de dispositivos IoT que sean capaces de trabajar de forma independiente, tomando lectura de los movimientos y sacudidas en distintos puntos estratégicos de una planta. La implementación del sistema permitirá tomar decisiones con respecto a la operación y la mantención técnica de las partes móviles asociadas al funcionamiento de motores eléctricos. Si por algún motivo un sensor detecta un movimiento errático fuera del esperado en el funcionamiento de un motor, el sistema embebido enviará un mensaje de alerta a un servidor, lo que permitirá que se realicen los ajustes técnicos en terreno a la máquina en cuestión. El sistema de monitoreo de fallas permitirá una mantención preventivas, pudiéndose identificar fallas en los equipos en etapas tempranas, evitando daños mecánicos, paradas de emergencia y detención de procesos.

1.1.1. IoT: Internet de las Cosas

Gracias al desarrollo de la electrónica se han diseñado un sin número de dispositivos que destacan por su tamaño reducido y por su capacidad de realizar operaciones matemáticas con un bajo consumo de energía, esto se traduce en dispositivos portátiles capaces de computar información obtenida de la realidad por medio de sensores sensibles a una magnitud o variable física en particular, para luego tomar decisiones y actuar por su propia cuenta sin la intervención humana, es por esto que también son llamados “inteligentes”. Pero sin duda lo más interesante de estos desarrollos es que estos dispositivos son capaces de conectarse en red y comunicarse con sus pares, pudiendo incluso enviar información a servidores web o a una nube. Esta característica

sumado al creciente uso de redes inalámbricas con acceso a internet ha permitido el poder conectar dispositivos, también conocidos como "cosas", a internet, con aplicaciones industriales, domésticas o personales. Esta tecnología es conocida como Internet de las Cosas o IoT: "Internet of Things", descrita como una red de objetos físicos (cosas) complementados con sensores, actuadores y software que les permiten conectarse e intercambiar información con otros dispositivos o sistemas a través de internet.

Una de las aplicaciones más interesantes de esta tecnología es la implementación de sistemas IoT en hogares, pudiendo encender y apagar electrodomésticos, controlar luces, medir la temperatura, humedad e incluso detectar fugas de gas desde un smartphone o cualquier otro dispositivo conectado a la red. Esto nos abre el camino a los Hogares inteligentes o *Smart homes*, en donde una serie de dispositivos domésticos se conectan a internet brindando la posibilidad de controlarlos por medio de cualquier dispositivo conectado a la red, siendo esta la ventaja de la tecnología IoT frente a la domótica tradicional [1]. Estas tecnologías llevadas a mayor escala permiten, por ejemplo, el poder controlar y medir el consumo de energía eléctrica de cada casa de una comunidad con la finalidad de aplicar técnicas de optimización en la red que podrían resultar en una reducción del consumo máximo de energía y de las cuentas de luz particulares [2]. Sin ir más lejos, en el año 2018 la Empresa de Servicios Sanitarios del Bio-Bío S.A. implementó un plan piloto que contempló la conexión de alrededor de 6000 hogares bajo una red de área amplia de baja potencia, conocida como Narrow Band IoT o NB-IoT, que permitió a los usuarios conocer el consumo diario de agua, facilitar pagos y facturaciones e incluso el poder detectar fugas, al empresa tuvo como beneficio el ahorros en gastos operacionales y la detección de filtraciones en la red, disminuyendo las de pérdidas de agua potable. Una prueba realizada por Huawei determinó que se puede dar cobertura de red para esta implementación al 80 % de los clientes de ESSBIO con solo una antena ubicada en el centro de la ciudad [3].

En [4] el autor presenta una solución IoT para un problema relacionado al área de la salud en donde se trata a pacientes en estado crítico. El trabajo contempla el monitoreo del nivel de una botella de solución salina y un aviso de forma inalámbrica al personal de salud respectivo en caso de que el nivel se encuentre fuera del margen deseado. Si bien la solución a este problema pareciera ser trivial, al estar involucrado con la salud de las personas se deben asegurar todas las condiciones que permitan al sistema funcionar de la mejor manera posible y se encuentre siempre disponible. Es por eso que el autor se enfoca enérgicamente en el protocolo de red que utilizará para transmitir la información, proponiendo una solución con bajo consumo eléctrico mientras se minimiza el uso de ancho de banda de la red, y en donde se garantiza la entrega, la veracidad y la privacidad de los mensajes de alerta para el personal médico, incluso para los no conectados, ya que la arquitectura propuesta es capaz de almacenar la información y enviarla apenas el receptor se encuentre disponible.

Luego se pueden citar estudios de interés como la monitorización de frecuencia cardíaca [5], monitoreo de sistema fotovoltaico [6] y control de sistema de bombeo de agua solar [7], las cuales coinciden en el uso del módulo ESP32 para la lectura de sensores, el análisis de los datos obtenidos y el envío de información relevante a la red. Los autores destacan el bajo coste, el bajo consumo energético, la versatilidad inalámbrica, la posibilidad de ver los datos de los procesos desde una página web o aplicación móvil y por sobre todo la viabilidad de todos los proyectos.

1.1.2. IIoT: Internet de las Cosas Industriales

El IoT por lo general está desarrollado pensando en el ser humano como usuario final, disponer de una serie de dispositivos de electrónica de consumo inteligentes interconectados entre sí pueden ayudar al usuario a comprender y controlar su entorno de mejor manera, ahorrando tiempo y dinero. Las comunicaciones de IoT se suelen caracterizar como de máquina a humano, enfocando su atención a que las "cosas" intercambien datos y puedan interactuar entre ellas, con requisitos de conectividad, confiabilidad y tiempo mas bien flexibles, sin embargo, las aplicaciones de esta tecnología pueden demandar otro tipo de requisitos específicos dependiendo del objetivo final, por ejemplo en un escenario industrial la norma general es establecer una comunicación inteligente entre máquina y máquina, permitiendo que, gracias a un sinnúmero de dispositivos conectadas con distintas máquinas a la vez, una planta pueda tomar decisiones por sí sola como un sistema completo. Aquí es donde podemos empezar a hablar de otra tecnología, conocida como IIoT o "Internet de las Cosas Industriales", cuyo objetivo es integrar la tecnología de operación con las tecnologías de la información, es decir, se pretende conectar las máquinas y sistemas de control industriales con los sistemas de información, ofreciendo soluciones analíticas y operaciones industriales óptimas, priorizando la estabilidad de la red, la eficiencia energética, la latencia, la coexistencia e interoperabilidad, la seguridad y la privacidad [8], dando así el paso a la Cuarta revolución industrial, también llamada Industria 4.0.

	IoT	**IIoT**
Principal diferencia	Centrado en un consumidor	Orientado a las máquinas
Objetivos generales	Permitir la comunicación de distintos dispositivos, sensores y actuadores en red para ofrecer una mejor experiencia al usuario	Obtener información y/o controlar maquinarias con el propósito de analizar y optimizar procesos industriales
Principal desarrollo	Creación de nuevos estándares de comunicación que facilitan la integración de nuevos dispositivos cada vez más sofisticados	Disminución de consumo energético de implementación y maquinarias. Mejoras en tiempo de respuesta y uso de ancho de banda
Conectividad	Ad-hoc (se pueden comunicar con cualquier otro nodo de la red)	Estructurado (funciona con nodos fijos, por seguridad no se permiten nuevos nodos sin autorización)
Precisión, exactitud y fiabilidad	En algunas aplicaciones se suelen aceptar márgenes para abaratar costos	Se prioriza enérgicamente, las fallas pueden significar una pérdida de eficiencia, actividad e ingresos
Volumen de datos	Medio a Alto	Alto a Muy Alto

Fig. 1.1: Internet de las cosas vs Internet de las cosas industriales.

Por lo general si hablamos de conectividad la tecnología IoT tiende a ser más flexible que el IIoT, permitiendo implementaciones en redes Ad Hoc en donde cada dispositivo es capaz de comunicarse con otro directamente, con independencia de redes externas, aunque por lo general se suele implementar en redes locales con conexión a internet con la intención de comunicarse con todos los dispositivos disponibles que se encuentren dentro del alcance de la red y a la vez

conectarse a servicios en la nube. En cambio en IIoT se suele implementar en redes estructuradas, en donde se pueda controlar exactamente la comunicación, puntualidad, confiabilidad y seguridad entre nodo y nodo, esto permite crear sistemas de redes específicos para cada aplicación. Las aplicaciones IIoT suelen estar orientadas al análisis y control de máquinas eléctricas, en donde generalmente, mientras mayor información obtenida por unidad de tiempo se tenga del proceso, mayor será la posibilidad de obtener resultados precisos, esto conlleva la generación de enormes cantidades de datos nuevos que son actualizados constantemente [9].

El mayor interés que despierta la implementación de IIoT es el aumento de productividad y eficiencia por medio de una gestión inteligente de los procesos industriales involucrados. Sin embargo, uno de los mayores desafíos tiene relación con la implementación en plantas industriales actualmente operativas, como la dependencia de sistemas y protocolos existentes y el alto riesgo de interrupción asociado a la implementación, además de limitaciones presupuestarias, disponibilidad de tecnología y la presencia de mano de obra calificada [10].

En este ámbito industrial podemos mencionar trabajos como el monitoreo inteligente de nivel de solución salina [11], sistema de control de supervisión y adquisición de datos SCADA [12] y el sistema de monitoreo en interiores IPS [13]. Estos autores comparten los comentarios finales de los trabajos mencionados anteriormente pero las aplicaciones desarrolladas en esta oportunidad tienen un carácter industrial, por lo que la seguridad y la robustez de los intercambios de información están asegurados. Es de destacar también las amplias posibilidades que ofrece un sistema SCADA para la monitorización y control de todos los datos de una planta, si bien este proyecto no propone el diseño de un sistema de estas características el embebido en su conjunto podría ser integrado a uno en una aplicación real.

1.1.3. Anomalías operacionales en procesos industriales

Uno de los grandes desafíos que tienen las plantas industriales es el poder contar con los equipos y máquinas eléctricas disponibles y funcionando de forma óptima la mayor cantidad de tiempo posible, para esto es sumamente importante la correcta implementación y operación de los equipos y por sobre todo la debida mantención de cada uno de ellos. Una mantención débil de las máquinas desencadena inevitablemente en fallas eléctricas y/o mecánicas, provocando deficiencias, ralentización o la detención completa de un proceso, lo que se traduce en importantes pérdidas económicas. Una correcta mantención en cambio permite asegurar la disponibilidad de los equipos y poder aprovechar todo el ciclo de vida de las máquinas.

La implementación de este proyecto permitirá realizar una mantención preventiva de motores eléctricas analizando sus vibraciones asociadas a la operación de los mismos. Una vibración consiste en un movimiento oscilante o movimiento repetitivo de una partícula o cuerpo alrededor de una posición de equilibrio. La vibración puede provocar daños a las estructuras de las máquinas, lo que puede resultar en una interrupción del funcionamiento, un desgaste excesivo e incluso el colapso, por lo que poder observar y/o predecir vibraciones anormales es de particular interés para la industria.

Haciendo uso de un sensor electrónico podemos registrar las aceleraciones asociadas a las vibraciones de un motor y analizarlas de forma digital por medio de un sistema computacional. Al analizar el comportamiento de un motor en funcionamiento se puede concluir que un motor

dañado tendrá una vibración notoriamente diferente a la del mismo motor sano [14] [15], mientras en el motor sano podemos esperar una vibración por lo general periódica en el tiempo y ajustada dentro de unos márgenes reducidos, en un motor con fallas esperamos ver una vibración mas bien errática con picos de aceleración fuera de los márgenes normales, situación que empeorará si se hace trabajar el motor a mayor velocidad.

Hay evidencia que nos permite concluir el análisis de vibraciones en motores de inducción permite detectar daños mecánicos, al detectar vibraciones excesivas en la maquinaria, y problemas eléctricos en motores que funcionan con variadores de frecuencia [16] [17] [18] [19]. Con servomotores ocurre una situación similar, las principales fallas mecánicas ocurren debido a altas temperaturas y a los intensos campos magnéticos asociados al funcionamiento del motor. Un campo magnético desbalanceado genera armónicos que inducen ruido en la red eléctrica y vibraciones fuera de lo normal en la máquina [20].

Contar con un análisis de las vibraciones de un motor nos permitirá entonces identificar fallas apenas estas ocurran y así poder realizar los ajustes del caso antes de que la falla provoque un daño mayor [14].

En el trabajo realizado por el Dr. Pedro Saavedra en [21] podemos ver un completo estudio del arte de la identificación de fallas operacionales por medio de las vibraciones de un motor. Comenta los estándares para el monitoreo operacional de las máquinas tomando como referencia estándares ISO, siendo el más importante para el caso la ISO 10816: "Vibración Mecánica. Evaluación de la vibración de máquinas en base a su medición en partes no–rotatorias de ella", en donde se clasifica el estado de la máquina de acuerdo al tipo de máquina, potencia o altura del eje. Además el trabajo comenta el uso de valores RMS, en vez del valor pico de la señal, y el estudio de los espectros de frecuencias para el análisis de las vibraciones. Un análisis similar se realiza en [22], en donde se discute la clasificación por nivel de criticidad de las máquinas, esencial para poder realizar análisis preventivo de fallas, y además se propone una aplicación de los conceptos para el análisis de vibraciones de un motor eléctrico, en donde también se utiliza el valor RMS de la vibración y el análisis en frecuencias para determinar la salud del motor.

El valor RMS de la señal de acelerómetro es una buena forma de determinar fallas operacionales ya que ya que su valor refleja la energía de la vibración del motor, por lo que si sabemos cuál es el comportamiento deseado de una máquina por medio de su valor RMS de vibración podemos determinar con certeza si la vibración actual es o no la esperada, por lo que nos sirve como primer indicio de que algo no favorable está pasando con el motor.

El análisis del espectro en frecuencias de la vibración de un motor también es una buena forma de determinar si el comportamiento de la máquina eléctrica es el adecuado o no. Para este análisis lo que se hace es descomponer la señal de las aceleraciones relacionadas a la vibración en sus componentes por frecuencias, pudiendo identificar las frecuencias con más relevancia involucradas en la vibración. De esta forma podemos determinar la salud del motor, como en los trabajos en [23], [24], [25] y [26], en donde se le aplica la Transformada rápida de Fourier FFT a la señal de acelerómetro para realizar el análisis. Las conclusiones de estos trabajos son bastante favorables para el uso de esta técnica, incluso hay estudios que han clasificado el comportamiento de las señales en frecuencia para determinar ciertos tipos de fallas operacionales en particular como en

[27] y [28], y otros incluso que han implementado soluciones con aplicación de redes neuronales [29], cosa que ya se escapa de los márgenes de este trabajo.

Para el desarrollo del dispositivo consideramos el uso del microprocesador ESP32, que incluye en su conjunto un microprocesador que puede ser programado en lenguaje C y que integra Wifi 802.11 b/g/n. Destacando de sobremanera en su versatilidad, robustez e integración de capacidades de red para aplicaciones industriales. Trabajos relacionados con este microprocesador y detección de fallas operacionales por medio de sensores inerciales podemos ver en [30] y [31], en donde se demuestra que este tipo de proyectos son viables actualmente.

El dispositivo será capaz de conectarse a una red inalámbrica compuesta por una serie de unidades y será capaz de monitorear el comportamiento inercial de máquinas eléctricas, permitiendo el cuidado de distintas partes del sistema. Cabe destacar que el dispositivo será embebido y será escalable en número de dispositivos conectados en simultáneo. El algoritmo de detección de fallas se centrará en la comparación del valor RMS de la vibración y de su análisis en el espectro de frecuencias.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar red de dispositivos IoT para la detección de anomalías operacionales en procesos industriales basados en sensores inerciales.

1.2.2. Objetivos Específicos

- Estudiar las topologías de red de sistemas IoT con sensores industriales
- Diseñar sistema embebido capaz de tomar lecturas de un sensor incercial y de comunicarse a través de una red inalámbrica.
- Diseño de algoritmo de detección de fallas por medio del dispositivo embebido del objetivo 1.
- Implementación del sistema embebido en una red de tipo industrial
- Analizar los resultados de los datos obtenidos para determinar posibles anomalías en procesos industriales.

1.3. Discusión

Hoy en día estamos frente a una revolución en el mundo de las IoT debido al masivo interés que ha despertado en la industria, propiciando la intensificación del desarrollo de hardware, software e infraestructuras de red para soluciones e implementaciones cada vez más específicas y sofisticadas.

Empresas como Intel, AMD, Huawei y Microsoft han implementado esta tecnología en soluciones relacionadas a procesos industriales, en Chile empresas como Entel, Telefónica Chile y Gtd han estado impulsando el desarrollo de infraestructuras de red para brindar soluciones IoT en la nube.

Los avances de la electrónica y las telecomunicaciones nos ha permitido poder implementar un microprocesador y una tarjeta de red en un mismo circuito, cambiando la forma en como se implementan las soluciones electrónicas. Previamente el sistema para realizar control automático en un proceso industrial consistía en conectar todos los sensores y actuadores a un computador central, por medio de redes físicas, para luego tomar decisiones. En la actualidad cada sensor y actuador puede incorporar un chip programado para que las decisiones de control se tomen directamente en el dispositivo y que solo se ocupe la red para comunicar información de interés en particular, ahorrando ancho de banda del sistema y recursos informáticos.

Un elemento clave entonces para el desarrollo de la solución propuesta es el microprocesador con el cual se trabajará. Actualmente existe una creciente variedad de SoCs que nos ofrecen sistemas que incluyen un microprocesador, junto con componentes de memoria y conectividad que se ajustan a las necesidades del proyecto. Uno de los primeros candidatos son los del fabricante Espressif Systems, el ESP8266 y el ESP32 debido a los recursos computacionales que ofrecen sus microprocesadores sumado a robustas capacidades de redes inalámbricas, además de ofrecer placas de desarrollo de fácil implementación y de ser una solución de bajo costo. Tenemos otros fabricantes como STMicroelectronics con su familia de microcontroladores STM32 wireless, y CEVA con su familia de microprocesadores CEVA-X1 que incluyen tecnologías de comunicación inalámbricas con microprocesadores de bajo consumo energético y limitado en recursos como cabría esperar, pero lamentablemente su tecnología se sustenta por software y ecosistemas propietarios, por lo que los ESP32 se convierten en la mejor opción si queremos tener total libertad en la implementación del proyecto.

Es bastante interesante ver el interés y el soporte que el fabricante Espressif Systems ha puesto en sus dispositivos ESP, apostando por ofrecer una amplia variedad de módulos con el fin de trabajar con el que más se acomode a nuestras necesidades. Además, sus módulos son bastante amables con los desarrolladores facilitando hojas técnicas detalladas de cada módulo y microprocesador, software y framework dedicados, guías de uso básico y avanzado, y un foro de soporte abierto.

Módulo	ESP-WROOM-02D	ESP32-WROOM-32D	ESP32-S2-WROOM
Chip embebido	ESP8266EX	ESP32-D0WD	ESP32-S2
Microprocesador	Tensilica L106 (32-bit)	Xtensa® single/dual-core 32-bit LX6	Xtensa® single core 32-bit LX7
Frecuencia de reloj	80/160 MHz	80/240 MHz	80/240 MHz
Voltaje de operación	3V - 3.6V	3V – 3.6V	3V – 3.6V
Consumo de corriente con Wifi activo	170 mA (17 dBm)	240 mA (19.5 dBm)	310 mA (19.5 dBm)
Consumo de corriente en Deep Sleep	20 µA	10 µA	20 µA
Entradas/Salidas de propósito general	9	34	43
Entradas ADC	1	18	20
SPI/I2C/I2S/UART	1/1/1/1	4/2/2/2	4/2/1/2
Memoria Flash	4 MB	4/8/16MB	4 MB
SRAM	50 kB	520 kB	320 kB
WIFI	802.11 b/g/n 2.4GHz	802.11 b/g/n 2.4GHz	802.11 b/g/n 2.4GHz
Bluetooth	No	Bluetooth v4.2 BR/EDR	No
USB OTG	No	No	Sí
Sensor de temperatura	No	Sí	Sí
Sensor efecto Hall	No	Sí	No
Criptografía	AES	AES/Hash/RSA/ECC/RNG	AES/Hash/RSA/RNG/HMAC
Precio	$3 USD ~ $2500 CLP	$4.08 USD ~ $3500 CLP	$4.13 USD ~ $3500 CLP

Fig. 1.2: Microcontroladores ESP para el desarrollo IoT.

En la Figura 1.2 podemos ver una comparación de los módulos ESP que se ajustan a las necesidades del proyecto, que son básicamente el bajo consumo de energía, un microprocesador programable en lenguaje C, comunicación WiFi 802.11 y una entrada I^2C para la conexión con el sensor inercial. Como podemos ver los 3 módulos seleccionados cumplen con los requisitos, sin embargo, el ESP8266EX es bastante limitado en las opciones de criptografía por hardware, que si bien no trabajaremos con ellas directamente en esta oportunidad, es de particular interés contar con las mejores opciones de seguridad posibles para una futura implementación en un ambiente industrial. Luego podemos notar que los dos candidatos restantes son bastante similares entre sí, sin embargo, tenemos dos diferencias que discutir. La primera característica que los diferencia es que el ESP32-S2 nos ofrece USB OTG directamente desde su encapsulado, esto quiere decir que permite la conexión directa por USB desde sus pines hasta un computador para su programación, a diferencia del ESP32-WROOM-32D en el que contamos sólo con conexión UART, por lo que se debe contar con una interfaz UART-USB para conectar el módulo al computador por USB. La otra diferencia y la más importante es que el microprocesador del ESP32-WROOM-32D es doble núcleo, eso quiere decir que podemos tener dos programas ejecutándose de forma paralela e independiente en cada uno de ellos, en cambio en el ESP32-S2 el microprocesador es de un solo núcleo. Considerando la naturaleza del proyecto y en los algoritmos que lo dominarán podemos afirmar que contar con dos núcleos de procesamiento sería de bastante ayuda, ya que se puede dejar corriendo el algoritmo de lectura del sensor en un núcleo y el programa que controlará las funciones de red en el otro, por lo que finalmente se realizará el trabajo con un ESP32-WROOM-32D.

Categories	Items	Specifications
Certification	RF Certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi Certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green Certification	REACH/RoHS
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I²C, LED PWM, Motor PWM, I²S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®, compatible with ISO11898-1)
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash [1]	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range [2]	–40 °C ~ +85 °C
	Moisture sensitivity level (MSL)	Level 3

Fig. 1.3: Especificaciones ESP32-WROOM-32D

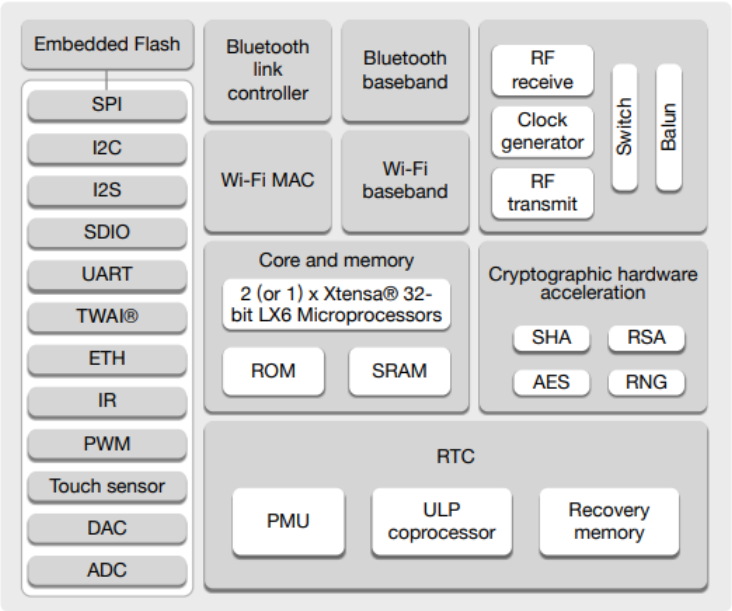


Fig. 1.4: Diagrama de bloques ESP32-D0WD [32].

Si bien los ESP32 presentan debilidades relacionadas a conversiones análogo digital [33], para el caso no es de preocupar debido a que la comunicación con el sensor se realizará de forma digital. También es necesario comentar el trabajo de Ionescu *et al.* en [34] que nos presenta una comparativa entre el lenguaje MicroPython y C ejecutados en el ESP32, entregando interesantes conclusiones al respecto y es que las implementaciones en C pueden llegar a ser un 45 % más rápidas que con MicroPython dependiendo de la complejidad de la tarea y de las asignaciones en memoria.

Con respecto a la seguridad, el autor en [35] destaca lo esencial que es establecer el protocolo correcto demostrando lo vulnerable que puede ser la comunicación WiFi de un ESP32 si se deja este apartado de lado, concluye con la recomendación de usar el protocolo TCP en vez del UDP para reducir la probabilidad de ataques a la red. Es por eso que los autores de [11], [12] y [13] usaron el protocolo Message Queuing Telemetry Transport o MQTT que funciona sobre TCP/IP y que nos permite el transporte de mensajes Publicador/Suscriptor inmensamente liviano. MQTT se puede implementar en para configurar la comunicación entre el módulo ESP32 y un servidor.

Ahora centrándonos en los sensores inerciales disponibles también podemos comentar que estamos frente a un avance tecnológico importante debido por sobretodo a la madurez de la electrónica digital, pasando de sensores basados en principios mecánicos por lo general costosos y con un sinnúmero de problemas asociados a descalibración y desgaste de los equipos, a pequeños dispositivos electrónicos integrados capaces de hacer las mismas mediciones con mucha más resolución y confiabilidad, creciendo junto a tecnologías digitales cada vez más cotidianas como teléfonos inteligentes y drones, entre otros.

Los sensores inerciales son también conocidos como IMU por sus siglas en inglés Inertial Measurement Unit y son dispositivos capaces de medir la velocidad angular, orientación y fuerzas gravitacionales de un objeto, usando una combinación multieje de giroscopios de precisión, acelerómetros, magnetómetros y sensores de presión. En nuestro caso para detectar vibraciones solo es de interés la información de acelerómetro. En la actualidad podemos hablar de dos grandes fabricantes, TDK (antes InvenSense) de donde destacan las familias MPU e ICM, y Bosch con sus sensores BMI.

En esta discusión tomaremos los mejores representantes de cada uno, en particular las familias del fabricante TDK ofrecen una numerosa cantidad de variaciones de cada uno y se eligieron los que más se acercaban a las necesidades del proyecto. Por otro lado la empresa Bosch trabaja principalmente con dos modelos, el BMI160 y el BMI270. Si bien los 4 IMU presentados se encuentran en proceso de producción, los fabricantes no recomiendan el uso de MPU-6500 y BMI160 para nuevos proyectos ya que se espera una descontinuación de estos modelos en favor de las familias ICM y del BMI270 que son dispositivos más nuevos y sofisticados. A pesar de esto, se incluyeron en la tabla para tener una mirada más amplia de los IMU actualmente disponibles.

IMU	MPU-6500	ICM-20602	BMI160	BMI270
Fabricante	TDK InvenSense	TDK InvenSense	Bosch Sensortec	Bosch Sensortec
Ejes	6 ejes	6 ejes	6 ejes	6 ejes
Acelerómetro	Sí, 16 bits	Sí, 16 bits	Sí, 16 bits	Sí, 16 bits
Rangos Acelerómetro	+-2, +-4, +-8, +-16 g	+-2, +-4, +-8, +-16 g	+-2, +-4, +-8, +-16 g	+-2, +-4, +-8, +-16 g
Sensibilidad Acelerómetro	16384 LSB/g	16384 LSB/g	16384 LSB/g	16384 LSB/g
Giroscopio	Sí, 16 bits	Sí, 16 bits	Sí, 16 bits	Sí, 16 bits
Rangos Giroscopio	+-250, +-500, +-1000, +-2000 dps	+-250, +-500, +-1000, +-2000 dps	+-125, +-250, +-500, +-1000, +-2000 dps	+-125, +-250, +-500, +-1000, +-2000 dps
Sensibilidad Giroscopio	131 LSB/dps	131 LSB/dps	262.4 LSB/dps	262.144 LSB/dps
Frecuencia de Muestreo (ODR)	Acc: 4 – 4000 Hz Gyro: 4 – 8000 Hz	Acc: 3.91 – 4000 Hz Gyro: 3.91 – 8000 Hz	Acc: 12.5 – 1600 Hz Gyro: 25 – 3200 Hz	Acc: 0.78 – 1600 Hz Gyro: 25 – 6400 Hz
Filtro pasa-bajo	Acc: 5 – 260 Hz Gyro: 5 – 250 Hz	Acc: 5 – 218 Hz Gyro: 5 – 250 Hz	Acc: 5.06 – 684 Hz Gyro: 10.7 – 890 Hz	Acc: 5.5 – 740 Hz Gyro: 11 – 751 Hz
Interfaz de salida	I2C, SPI	I2C, SPI	I2C, SPI	I2C, SPI
Voltaje de alimentación Mín. – Max.	1.8 V – 3.3 V	1.71 V – 3.45 V	1.2 V – 3.6 V	1.2 V – 3.6 V
Consumo de corriente (Giroscopio y acelerómetro)	3.4 mA (Normal mode)	2.79 mA (low-noise mode)	925 µA (full operation)	685 µA (normal mode)
Consumo de corriente (Solo acelerómetro)	450 µA (max. ODR)	321 µA (low-noise mode)	180 µA (full operation)	210 µA (max. ODR)
Precio	$4.72 USD ~ $4000 CLP	$5.08 USD ~ $4400 CLP	$5.73 USD ~ $4900 CLP	$6.72 USD ~ $5800 CLP

Fig. 1.5: Sensores inerciales para el desarrollo IoT.

Al ver la Figura 1.5 podemos notar que los cuatro dispositivos cumplen con los requisitos del proyecto, contar con acelerómetro y que se pueda comunicar por medio del protocolo I^2C para poder obtener sus datos desde el microprocesador. Si bien los candidatos parecen ser bastante similares, podemos ver claras diferencias en la frecuencia de muestreo, en particular del acelerómetro que es lo que nos interesa. Sin embargo, no estamos interesados en tener el acelerómetro con el mayor ODR posible, si no más bien con uno que cumpla con el objetivo, y, analizando también el consumo de corriente y la disponibilidad de los chips podemos decir que el mejor candidato es el BMI270.

Por las razones discutidas anteriormente la implementación y desarrollo del proyecto se realizará en un microcontrolador ESP32 junto a un sensor inercial, el cual será capaz de comunicarse a través del protocolo MQTT con un servidor local. De forma específica el microcontrolador será el ESP32-WROOM-32D [36] y el sensor el BMI270 [37], esto porque cumplen las expectativas técnicas del proyecto y por la disponibilidad de estos chips.

1.4. Alcances y Limitaciones

- El microcontrolador del sistema embebido será programado en lenguaje C en Windows y será capaz de hacer lecturas del sensor de inercia.
- Cada sistema embebido será capaz de medir vibraciones por medio de lecturas de acelerómetro (sensor de inercia).
- Cada sistema embebido será capaz de analizar estos datos en tiempo real y de forma independiente.

- Al identificar un patrón anormal de las lecturas de acelerómetro el embebido enviará un mensaje a modo de alerta a través de una conexión local via WiFi a un servidor mediante protocolos IIoT (Industrial Internet of Things).

1.5. Metodología

El trabajo se desarrollará en mayor parte en el domicilio del estudiante, con un computador con Windows 10 y placas de desarrollo facilitadas el profesor guía. Las pruebas se realizarán en el laboratorio de Instrumentación y desarrollo de la Facultad de Ingeniería de la Universidad de Concepción.

La programación del módulo ESP32 se realizará en el software IDE Visual Studio Code, un editor de código desarrollado por Microsoft que nos permitirá escribir el programa para el ESP32 en lenguaje C. Posteriormente será compilado y escrito en el módulo utilizando el software ESP-IDF.

La metodología del trabajo consistirá primero en la invención del dispositivo embebido, partiendo por el diseño del circuito electrónico y posterior programación. Se programará primero la lectura de las variables de interés del sensor por medio del microprocesador, luego se configurarán sus capacidades de red y se establecerá la topología de red a utilizar, para finalmente programar el algoritmo de detección de fallas. Posteriormente se realizarán pruebas en laboratorio, donde se harán pruebas en un motor industrial adaptado para poner a prueba la identificación de fallas y la comunicación inalámbrica.

CAPÍTULO 2

Redes Industriales en IoT

2.1. Introducción

Cuando hablamos de dispositivos IoT nos referimos a dispositivos que pueden comunicarse entre sí, pueden ser identificados, localizables, direccionables o controlables a través de una red. Esta red puede ser física, inalámbrica y puede estar conectada o no a internet. Una red de carácter industrial, a diferencia de redes con otros fines, se enfoca principalmente en la conexión de sensores y actuadores de máquinas industriales a una red en donde se prioriza el rendimiento, escalabilidad (que se pueda agregar más dispositivos sin afectar al sistema global), consumo de energía y seguridad de la información comprometida para cada aplicación en específico. Estas redes, como comentamos en el Capítulo 1, están desarrolladas para la comunicación entre máquinas, y de máquinas a sistemas. Una de las cosas más interesantes de estas redes es poder obtener información de decenas, cientos o miles de dispositivos y poder realizar un control automático de forma independiente con cada uno de ellos, trayendo consigo una mejora en la eficiencia de equipos involucrados en un proceso industrial [38].

2.2. Protocolos de red industriales

Para cumplir con estos objetivos se han desarrollado diversos protocolos de mensajería, como lo son Open Platform Comumnication Unified Architecture (OPC UA), Message Queuing Telemetry Transport (MQTT) y Message Queuing Protocol (AMQP), y su elección estará vinculada directamente de la naturaleza del sistema de IoT y sus requisitos de mensajería.

- OPC Unified Architecture (OPC UA):

  Es un estándar multiplataforma de código abierto desarrollado en 2006 para la comunicación en el campo del control y supervisión de procesos industriales. La comunicación entre sistemas y dispositivos puede realizarse bajo el modelo de solicitud / respuesta entre clientes y servidores o bajo el modelo de publicación / suscripción. OPC UA se desarrolló con el fin de poder comunicar equipos que trabajaban en base a estándares propietarios incompatibles entre fabricantes distintos estableciendo una forma común de comunicación, permitiendo el intercambio de datos de todos ellos con un solo cliente, o incluso entre los propios dispositivos. Permite el cifrado de la información con AES-128 y AES-256 bits e implementaciones bajo

SSL y HTTPS. Además de protocolo de comunicación, OPC UA se puede implementar como arquitectura, permitiendo servicios como modelado de datos, espacio de direcciones, gestión de alarmas y eventos, historial de variables, control de acceso, entre otros.

- Message Queuing Telemetry Transport (MQTT):

  Es uno de los protocolos de comunicación M2M más antiguos, presentado en 1999. Es un protocolo de mensajería de tipo publicación / suscripción diseñado para comunicaciones M2M ligeras. Los clientes publican mensajes a un servidor conocido como Broker, quien está suscrito a otros clientes. Cada mensaje es publicado con una dirección llamada Tópico. Los clientes pueden suscribirse a varios Tópicos y recibir cada mensaje publicado en cada uno de ellos. Trabaja bajo TCP como capa de transporte y permite TLS/SSL en cuanto a seguridad.

  MQTT también ofrece una función llamada Calidad de servicio o QoS (Quality of Service), que nos permite definir la seguridad de entrega de cada mensaje. Son tres, el QoS 0: *at most one*, que envía el mensaje sin garantías de recepción, se utiliza cuando la información no es crítica y la comunicación es estable, el QoS 1: *at least once*, que envía el mensaje las veces que sean necesarias hasta que el Broker confirme su recepción, y el QoS 2: *exactly once*, que nos asegura exactamente la entrega del mensaje, permitiendo incluso que el publicador guarde el mensaje hasta que el Broker no confirme su envío a través de la red, esta última opción es la más lenta de todas.[39]

- Message Queuing Protocol (AMQP):

  Desarrollado el 2003, este protocolo admite la arquitectura tanto de solicitud / respuesta como de publicación / suscripción. Es un protocolo de mensajería binario diseñado para admitir de forma eficaz una amplia gama de aplicaciones de mensajería y patrones de comunicación. Permite la interoperabilidad de sus clientes. Una de sus principales características es el sistema de Colas de mensajes *(Queue)*. Cada Cola se utiliza para 1 Suscriptor. Cuando el Publicador publica mensajes en el Broker no se envían directamente al Suscriptor como el Broker MQTT, si no que se envían a un intercambiador *(Exchange)*. El intercambiador será responsable de enrutar los mensajes en las colas correspondientes. El mensaje se almacenará en la cola hasta que el suscriptor lo elimine. Se ejecuta sobre el protocolo de transporte TCP y utiliza TLS / SSL y SASL para la seguridad. AMQP admite dos niveles de QoS para la entrega de mensajes: *Unsettle Format* (no confiable y al menos una vez) y *Settle Format* (confiable y como máximo una vez).

Para escoger el protocolo a utilizar debemos tener muy presente las características de nuestro proyecto, ya que cada protocolo tiene sus ventajas y desventajas desde el punto de vista de la aplicación en la que se usarán [40]. En primer lugar debemos considerar que se busca diseñar un sistema embebido tal que permita realizar las funciones requeridas consumiendo la menor cantidad de energía posible, esto ya que al ser un sistema escalable, un gran número de dispositivos con un alto consumo de energía por unidad es un resultado altamente indeseado.

	OPC UA	MQTT	AMQP
Año	2006	1999	2003
Arquitectura	Cliente / Servidor	Cliente / Broker	Cliente / Servidor o Cliente / Broker
Tamaño del header	8 bytes	2 Byte	8 Byte
Calidad del servicio	Depende de los dispositivos conectados	QoS 0: *At most once* QoS 1: *At least once* QoS 2: *Exactly once*	Settle Format (Similar a *At most once*) Unsettle Format (Similar a *At least once*)
Protocolo de transporte	TCP, HTTP	TCP	TCP, SCTP
Finalidad	Unificar los protocolos propietarios en entornos industriales	Comunicación para dispositivos de bajos recursos, minimizando el ancho de banda y tamaño del mensaje	Protocolo de propósito general. Incluye cola y enrutador de mensajes, soporta múltiples tipos de mensajes y de comunicación con el servidor

Fig. 2.1: Comparación redes industriales IoT.

Por la naturaleza del proyecto podemos descartar la implementación con OPC UA, ya que su fuerte es la comunicación de distintos dispositivos y máquinas entre sí, inclusive si cada uno trabaja bajo protocolos de comunicación distintos, y este no es el caso, ya que el sistema que se está desarrollando necesita la comunicación entre dispositivos conocidos y programados justamente para comunicarse entre ellos. Otro punto fuerte de esta tecnología es que permite la implementación de una arquitectura completa basada en este estándar, abriendo un abanico de posibilidades si se requiere implementar una red IoT en toda una planta, incluso se puede implementar OPC UA en conjunto con dispositivos conectados bajo MQTT [41] [42]. Ahora bien, si queremos establecer OPC UA exclusivamente para el sistema de este proyecto tenemos una debilidad crítica en comparación a MQTT y es que este último necesita menos ancho de banda y logra tiempos de transmisión menores para enviar el mismo mensaje [43]. En conclusión OPC UA es un protocolo robusto que abarca más de lo necesario y se escapa de los requisitos del proyecto, por lo que los verdaderons candidatos son MQTT y AMQP.

Comparando MQTT y AMQP el protocolo MQTT es el que tiene el mejor rendimiento, ocupando menos ancho de banda y recursos del servidor [44], además es el que menos energía consume en comparación a AMQP [45]. Otro punto a considerar es el uso de CPU del microprocesador, como deseamos diseñar un sistema embebido pequeño y de bajo consumo mientras menos recursos de CPU utilicemos mejor, en este sentido MQTT la opción más ligera [46].

Podemos ver además que MQTT es el protocolo más adecuado para utilizarse en grandes redes compuestos por dispositivos pequeños con pocos recursos, y que además se puede utilizar para mensajería de dispositivo a dispositivo y en mensajes de tipo multicast para enviar datos a varios dispositivos a la vez [47]. Requiere menos recursos para implementar que el protocolo AMQP, debido a su simplicidad. Su uso ideal es cuando los paquetes de datos no se envían de forma continua, como en nuestro caso [39].

Luego de todo lo comentado anteriormente se considera el protocolo MQTT como el que más se adapta con nuestros objetivos por su ventajas en eficiencia y uso de ancho de banda.

2.3. Protocolo MQTT

Message Queue Telemetry Transport o MQTT es un protocolo usado para la comunicación M2M Machine-To-Machine en dispositivos IoT. Se basa en la mensajería del tipo Publicador/Suscriptor en donde el emisor envía el mensaje con un tema en particular al servidor y si el receptor está suscrito a ese tema entonces el servidor le enviará el mensaje, a diferencia del protocolo HTTP que utiliza mensajes del tipo Solicitud/Respuesta en donde el receptor pide el mensaje y el emisor lo envía. Un buen uso de este protocolo es el referente a la comunicación de sensores y dispositivos IoT, ya que requiere pocos recursos de parte de los microcontroladores y poco ancho de banda para el intercambio de mensajes. El protocolo MQTT se ha convertido en uno de los principales pilares del IoT por su sencillez y ligereza.

MQTT utiliza una topología en estrella, por lo que todos los clientes se conectan directamente a un punto central que hace de servidor, comúnmente llamado Broker.

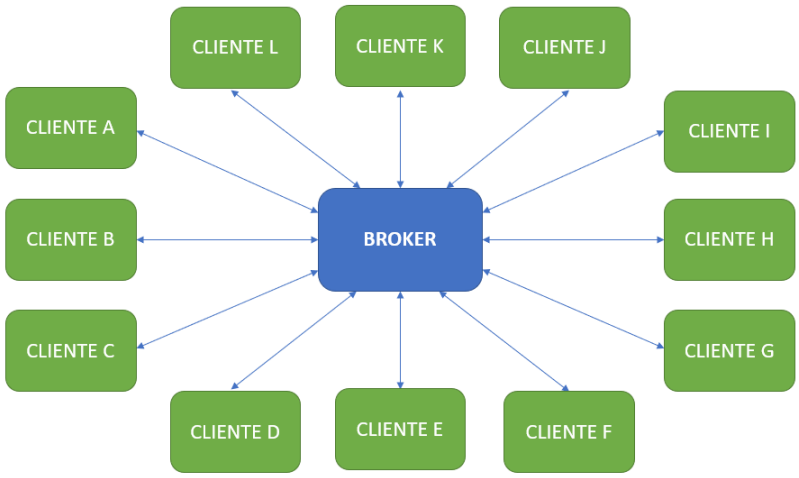


Fig. 2.2: Topología MQTT.

Cada mensaje que emite un Publicador se envía a los receptores que se hayan suscrito a un tópico en concreto (Suscriptores). El publicador no decide a quién enviarle el mensaje, pero publica cada mensaje en un Tópico, el Broker es el encargado de distribuir los mensajes a los Suscriptores que estén suscritos a ese tópico en específico. El emisor no sabe a quién va dirigido dicho mensaje, sólo el Broker lo sabe y es el encargado de distribuir los mensajes a los receptores. Cuando llega un nuevo mensaje se lo envía a aquellos que se han suscrito al tópico en específico.

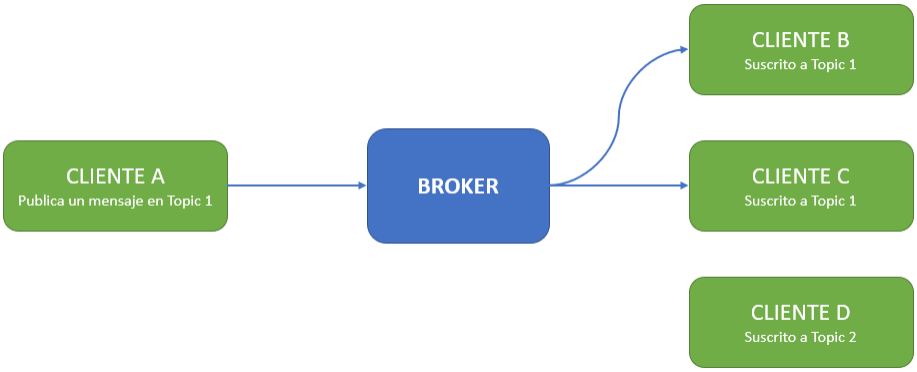


Fig. 2.3: Ejemplo Publicación/Suscripción MQTT.

Por ejemplo en la Figura 2.3 podemos ver que los Clientes B y C están suscritos al Tópico 1, entonces cuando un Cliente A publica un mensaje en el Tópico 1 el Broker se encarga se enviar este mensaje a los Clientes B y C. El cliente D no recibe la información porque está suscrito al Tópico 2 y no hay mensajes enviados con ese tópico.

Finalmente comentar que el diseño de nuestra red puede ser descrito en el modelo de capas representado en la Figura 2.4, en donde se puede ver que las capas física y de enlace (1 y 2) estarán dominadas por el estándar IEEE 802.11, en la capa de red (3) se utilizará el protocolo IPv4, en la capa de transporte (4) usaremos el protocolo TCP y las capas de sesión, presentación y aplicación (5, 6 y 7) quedará regido por el protocolo MQTT.

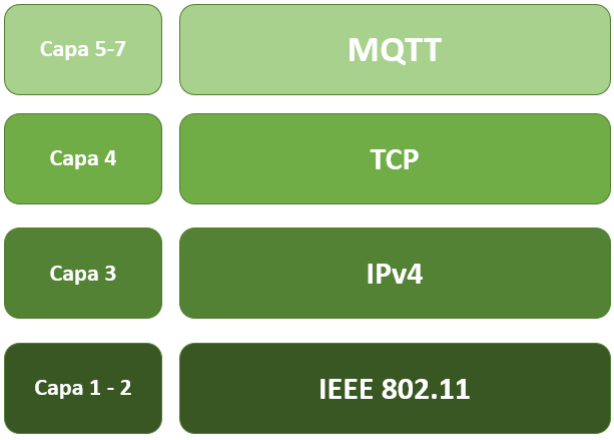


Fig. 2.4: Capas de red MQTT según modelo OIS.

CAPÍTULO 3

Dispositivos de IoT: Microcontrolador y sensores

3.1. Introducción

Una de las tecnologías que ha estado creciendo bastante últimamente de la mano de las IIoT es el Edge Computing o Computación de frontera (o de borde), idea que propone acercar el procesamiento y el almacenamiento de datos al lugar físico desde donde se están generando. La principal solución que ofrece esta tecnología es la de alivianar la carga de la red y disminuir la carga computacional del servidor principal. La tecnología IIoT busca conectar los sensores y actuadores a una red o nube, lugar al que llegan todos los datos generados y donde se realizan los cálculos necesarios para tomar decisiones y luego realizar las acciones. Esto puede tornarse caótico si se aumenta la cantidad de sensores ya que se aumenta también la cantidad de datos con las que trabajar. En cambio, con el Edge computing lo que se hace es realizar los cálculos directamente en el sensor, con la intención de tomar decisiones *in situ* y enviar solo la información importante a la red.

Evidemente, el edge computing está muy relacionado con el IoT porque lo que pretende es procesar la información en los propios objetos conectados en lugar de hacerlo en un servidor. Al menos, gran parte de ellos. Es el futuro del procesado de datos. El sistema propuesto tiene características de Edge computing ya que las decisiones son tomadas por cada dispositivo en el mismo lugar donde se están tomando las lecturas de acelerómetro, sin intervenciones ni permisos externos.

Se propone entonces un dispositivo embebido que consiste en un circuito con la electrónica necesaria para el funcionamiento y puesta en marcha del sistema en cuestión. El principal desarrollo del sistema se realizará en el microcontrolador y el sensor inercial, en su comunicación y en sus requisitos de energía. Los demás bloques se integrarán con los circuitos adecuados para permitir la independencia portátil por medio de baterías. A continuación se presenta un esquema de bloques que representa el funcionamiento del sistema y las relaciones entre sus principales componentes.

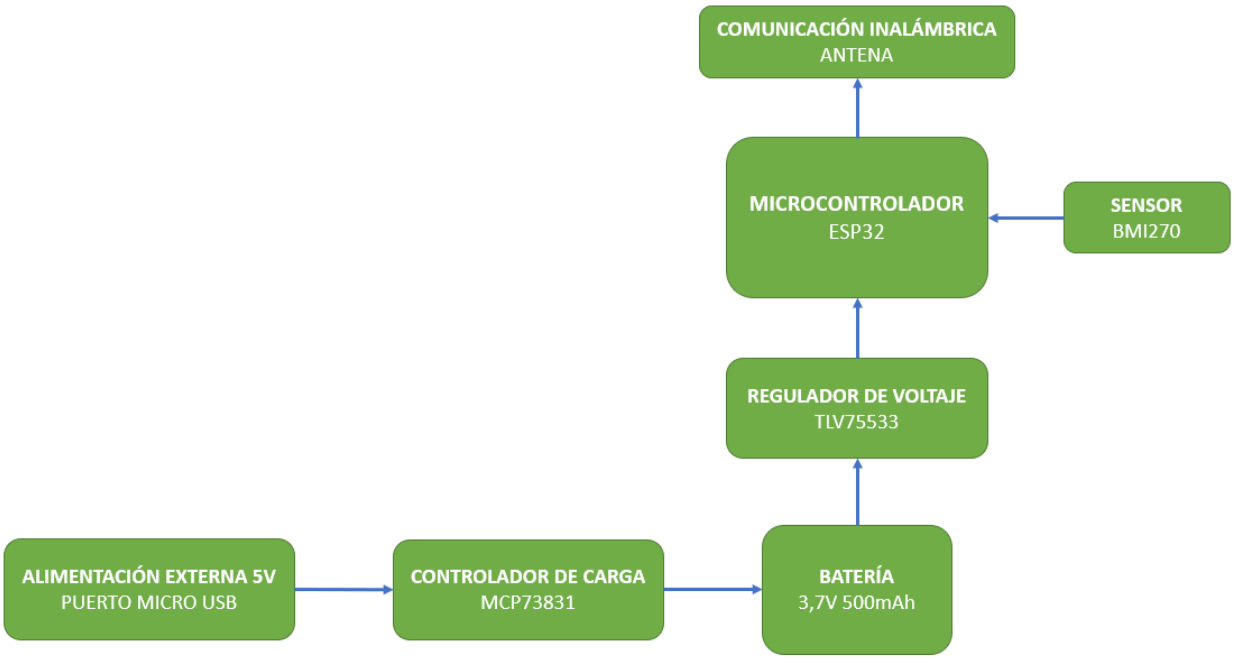


Fig. 3.1: Diagrama de bloques sistema embebido propuesto.

3.2. ESP32

La programación del ESP32-WROOM-32D se realizará bajo el framework ofrecido por el fabricante llamado ESP-IDF [48], el cual contiene las herramientas de software necesario para la compilación y construcción de los programas para cargar en el ESP32.

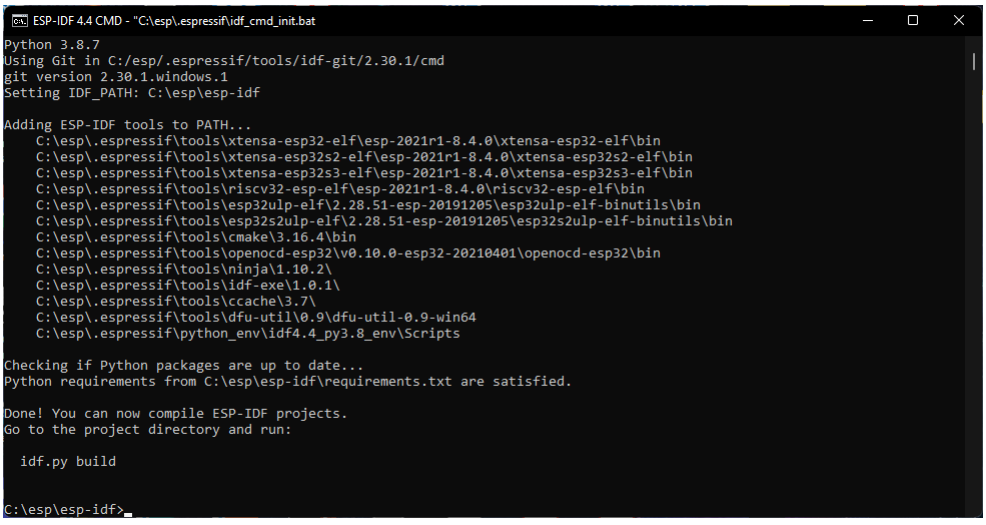


Fig. 3.2: Ventana de inicio ESP-IDF.

Además del software ESP-IDF también haremos uso del framework ESP-MDF [49], un entorno de trabajo que nos permite crear redes de tipo ESP-WIFI-MESH, esto nos facilitará la implementación de una red inalámbrica de estándares industriales de dispositivos ESP32 de forma tal que la

integración de cada nuevo nodo se realizará después de encontrar el mejor enrutamiento posible según los valores de intensidad de señal entre todos los pares cercanos.

La electrónica involucrada en el SoC ESP32 corresponde a condensadores utilizados como filtro pasa bajos en la entrada de alimentación del chip, estos nos aseguran disponer de 3.3V con características DC deseables [36]. Se incluyen además pulsadores y la disponibilidad de pines de tipo IO para la comunicación con el sensor y pines de tipo RX y TX para la futura programación en la placa electrónica.

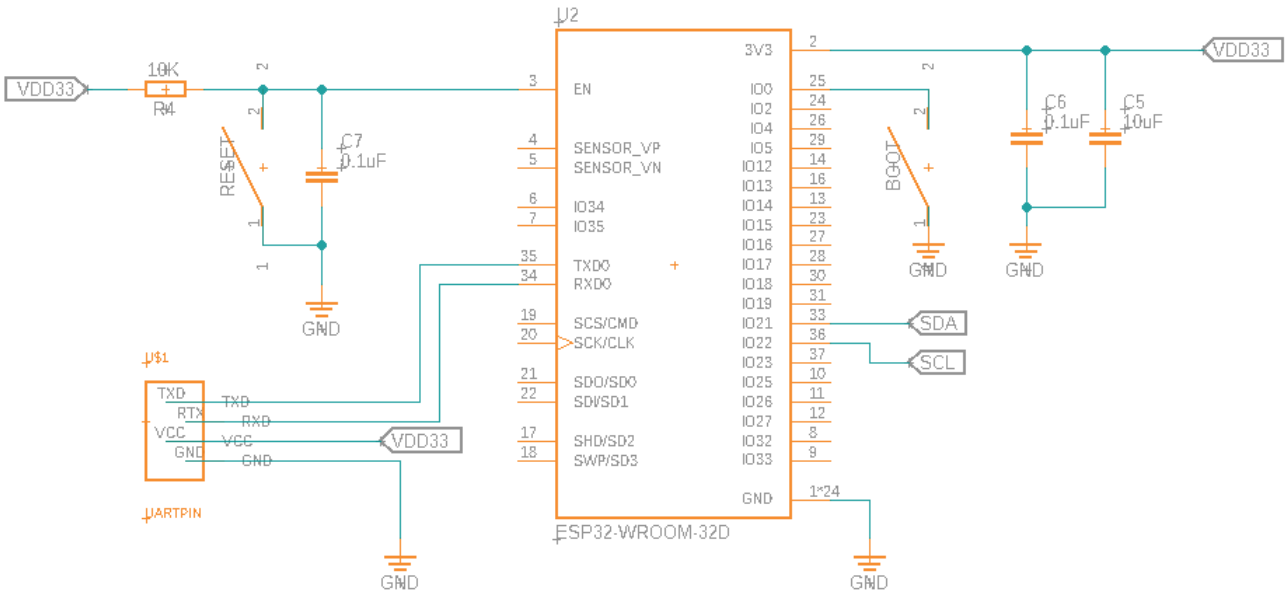


Fig. 3.3: Circuito esquemático ESP32.

3.3. BMI270

El circuito para la alimentación y la comunicación del BMI también viene dada por una configuración de filtros que nos permite trabajar con las señales bajo el protocolo I2C de la forma recomendada por el fabricante [37].

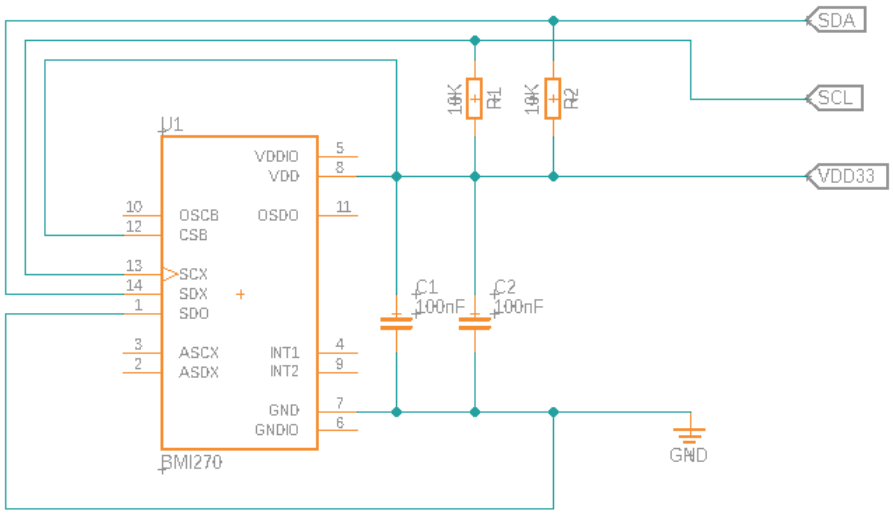


Fig. 3.4: Circuito esquemático BMI270.

3.3.1. Comunicación I2C

Para realizar una lectura de los datos obtenidos por el sensor BMI270 con el microcontrolador ESP32 haremos uso del protocolo I^2C, el cual nos permite establecer una comunicación entre estos dos dispositivos. La comunicación nos permite leer o escribir en registros en específicos del sensor.

I^2C (Inter-Integrated Circuit) es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre 2 o más dispositivos digitales. A diferencia del puerto Serial tradicional, los mensajes que se envían mediante un puerto I^2C, incluye además del byte de información, una dirección tanto del registro como del sensor. Para la información que se envía siempre existe una confirmación de recepción por parte del dispositivo.

La conexión de los dispositivos se realiza por medio de dos cables, llamados SDA y SCL. El SDA o Serial Data corresponde al bus de datos del protocolo y el SCL o Serial Clock corresponde a la señal de reloj que domina la comunicación. Los dispositivos pueden conectarse como Maestro o como Esclavo. El maestro se encarga de generar la señal de reloj y de iniciar y detener la comunicación, en cambio el esclavo suministra la información de interés al maestro. El sistema es escalable para varios dispositivos Esclavos conectados a un Maestro en el mismo bus I^2C.

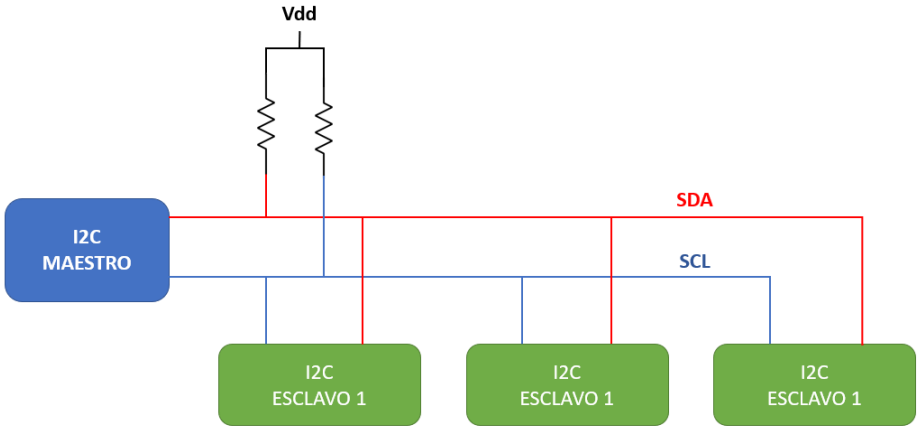


Fig. 3.5: Bus de datos I^2C.

La transmisión de datos se inicia con un bit de inicio (START) y termina con un bit de finalización (STOP). El bit de START se reconoce porque la señal SDA pasa de un estado lógico alto a un estado lógico bajo mientras la señal SCL está en nivel alto. Por el contrario, el STOP ocurre cuando el SDA pasa de un estado lógico bajo a uno alto cuando la señal SCL está en un nivel alto. Existe una tercera señal importante conocida como ACKS o Acknowledge (reconocimiento) que nos indica que cada byte haya sido recibido.

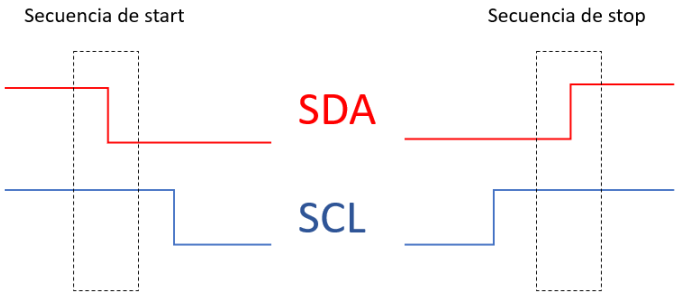


Fig. 3.6: Start y Stop en la comunicación I^2C.

3.4. Diseño de la electrónica

En el circuito electrónico final se incluyeron dos circuitos integrados que son importantes para la independencia del sistema, el MCP73831 [50] y el TLV75533 [51], un controlador de gestión de carga lineal de baterías Li-Po para obtener portabilidad y un regulador lineal de tensión de 3.3V para alimentar el circuito de forma correcta. Cabe destacar que el MCP73831 está configurado para cargar baterías de 500mAh. Sus circuitos esquemáticos se pueden ver en Anexo.

Para finalizar, presentamos en la Figura 3.7 el diseño del circuito impreso de la placa, diseñado en el software Autodesk Fusion 360. En este proceso se tomaron en cuenta las recomendaciones

de las Hojas técnicas de todos los componentes electrónicos, con tal de fabricar una placa con las condiciones óptimas para el correcto funcionamiento del circuito [36] [37] [50] [51]. En primer lugar el ESP32 se instalará en la parte superior de la placa, con la PCB de la antena fuera de la superficie de la placa para reducir interferencias con la placa misma [52]. Luego el BMI270 se instalará en un lugar en donde no se presente tensión mecánica en la placa debido a sus cuatro puntos de sujeción [53]. La placa electrónica es de dos capas y su tamaño es de 35mm x 45mm. En la Fig. 3.7 podemos ver el diseño en computador de la placa y el montaje de la placa funcionado a modo standalone.

Para determinar el grosor de las pistas primero debemos notar que la máxima corriente que existirá en el circuito es 500mA, que corresponden a la corriente máxima del controlador de carga de baterías y del regulador de voltaje. Comentar que el circuito exigirá menos corriente que la máxima, ya que se estima que el microcontrolador consuma alrededor de 240mA el sensor alrededor de 685 μA. Además, como el circuito no cuenta con partes de alta potencia o partes análogas, se decide fabricar las pistas de 16 mil (0.4064 mm) de grosor para uso general y de 12 mil (0.3048 mm) para las pistas del BMI270, lo cual que nos permite trabajar con corrientes superiores a 1 A sin presencia de calentamientos en las pistas, ideal para la naturaleza del proyecto ya que el circuito completo no consumirá más de 0.3 A [54].

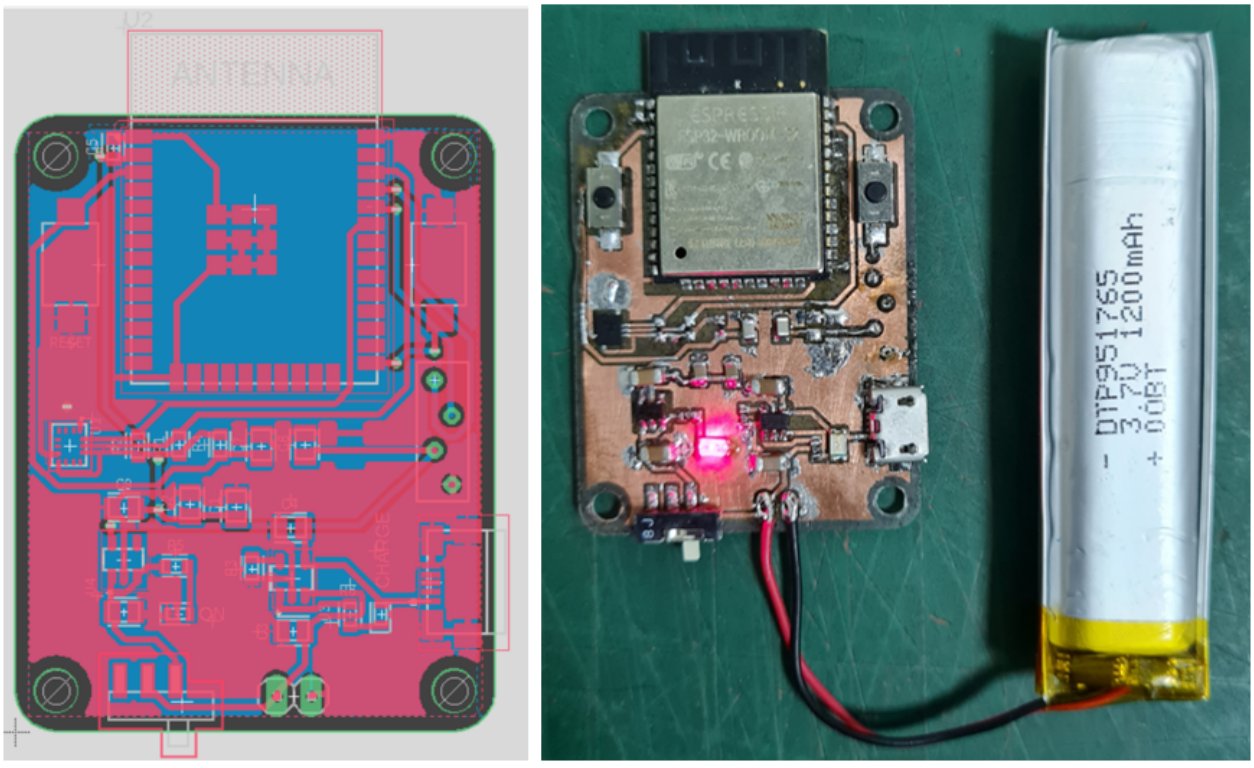


Fig. 3.7: Sistema embebido: Esquema circuito impreso y circuito en placa.

CAPÍTULO 4

Diseño de red de sensores IoT

4.1. Introducción

La red que se busca diseñar debe permitir una conexión inalámbrica de pocos a muchos dispositivos IoT a la vez, para asegurar la escalabilidad del sistema. Estos dispositivos estarán por lo general estarán dispuestos físicamente en líneas rectas, requiriendo cobertura en largas distancias. Esta conexión deberá permitir enviar un mensaje desde cualquier dispositivo a un servidor local y se le exigirá estabilidad y disponibilidad en toda su área de alcance.

4.2. Red Mesh Inalámbrica

Las Redes Mesh Inalámbricas (WMN) son redes que se comunican sobre la infraestructura del protocolo WiFi (IEEE 802.11) y que pueden ser entendidas como muchas redes WiFi individuales en una sola red WLAN, facilitando y asegurando la escalabilidad de la red. El protocolo WiFi permite conectar un nodo (dispositivo o sistema conectado a la red) a un solo Punto de acceso, mientras que cada Punto de acceso puede estar conectado a varios nodos a la vez, esta forma de conexión es conocida como topología tipo estrella, en donde el área de cobertura de la red queda limitado por la potencia de transmisión y recepción de las antenas del Punto de acceso y de los nodos.

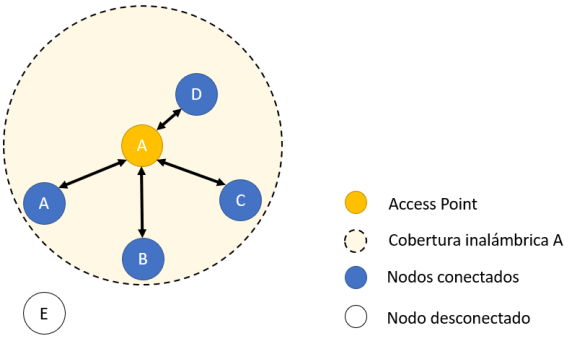


Fig. 4.1: Topología estrella.

En la Figura 4.1 podemos ver una topología de WiFi clásica, en donde el área de red queda definida por el área de alcance del Access Point por lo que los nodos A, B, C y D se conectan a la

red, mientras que E queda excluido. Una red Mesh permite que los nodos se comporten como nodo y Punto de acceso. Una de sus características principales es que los nodos no están conectados a un nodo central, si no que lo hacen entre ellos formando capas de conexiones. Cada nodo se encarga de retransmitir la información a una capa siguiente hasta llegar al servidor. Esta topología nos permite establecer una red inalámbrica de muy alto alcance, ya que este está determinado por el alcance de cada nodo.

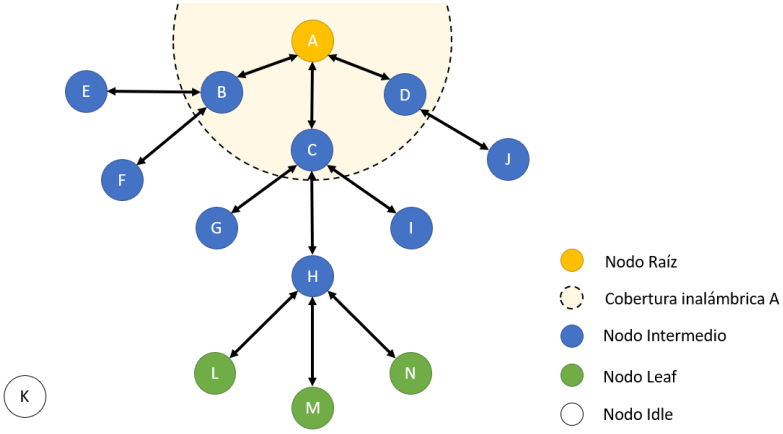


Fig. 4.2: Topología Mesh para red de 4 capas.

Como podemos ver en la Figura 4.2 las conexiones se ordenan siguiendo una topología tipo árbol con una jerarquía bien establecida:

- *Nodo Raíz*: Corresponde al nodo de mayor nivel y es el único que puede conectarse a una red exterior. Solo se permite establecer un único nodo raíz.
- *Nodo Leaf*: Corresponde al nodo de menor nivel y no permite que otros nodos se conecten a él, por lo que este tipo de nodo solo puede transmitir datos. Si un nodo se conecta en la última capa permitida por la red entonces será asignado como nodo leaf. Los nodos L, M y N fueron asignados en la última capa permitida por lo que funcionan como nodos leafs.
- *Nodo intermedio*: Son nodos que no son ni raíz ni leaf y que pueden estar conectados al nodo raíz o a otro nodo intermedio. Además permite que otros nodos se conecten a él. Los nodos de la B a la J son nodos intermedios. Notar que los nodos E, F, G, I y J no son nodos leaf ya que estos permiten que otros nodos se conecten a ellos.
- *Nodo Idle o Nodo Inactivo*: Corresponden a nodos que están fuera de la red y que intentarán conectarse con un nodo intermedio. El nodo K es un nodo inactivo.

Para construir la red primero se debe seleccionar el nodo raíz y desde ahí crear las conexiones hacia abajo en la jerarquía. Primero se buscan y conectan los nodos intermedios más cercanos para construir la segunda capa, luego se irán formando las capas siguientes hasta llegar al límite de capas de la red. Los nodos de la última capa serán considerados como nodos leafs y no permitirá que otros nodos se conecten a ellos. El número de capas se establece en el nodo raíz.

En nuestro caso dispondremos de una serie de ESP32 que estarán tomando lecturas inerciales de forma individual e independiente, se conectará a una red WiFi Mesh y será capaz de enviar

información usando el protocolo MQTT a una Raspberry Pi que hará el trabajo de servidor Broker, que puede estar o no conectado a internet o a otro servidor local. Entonces cada vez que un ESP32 deba enviar un mensaje lo enviará a los nodos intermedios de las capas siguientes hasta llegar al nodo raíz y este a su vez lo enviará al Broker en la Raspberry Pi. En su implementación en una planta real la Raspberry podría convertir el mensaje recibido a uno compatible con el servidor de la planta en cuestión o conectarse a un Broker ya existente.

4.3. Diseño de red

El servidor Broker estará programado en un Raspberry Pi 3, un computador pequeño de bajo consumo de energía que ofrece hardware de red inalámbrica y permite ejecutar programas en el sistema operativo GNU/Linux. La Raspberry Pi estará configurada como Access Point inalámbrico, es decir, que será la Raspberry la encargada de crear una red WiFi con su propio nombre (SSID) y contraseña, red a la cual se conectará el nodo Root de la red Mesh, esto lo implementamos haciendo uso de los programas *Hostapd* y *Dnsmasq*, el primero nos permite usar el WiFi como Access Point y el segundo nos permite implementar servicios de DHCP que, entre otras cosas, nos permite asignar direcciones IP de forma automática a cada dispositivo conectado. Cabe mencionar que todos los dispositivos cuentan con una MAC única y conocida con la cual podemos identificarlos de manera singular, y que además cada uno tendrá un identificador único asignado por software, que corresponde a un número entero que comienza desde el 1 para el nodo root y continúa de forma consecutiva hasta el último dispositivo.

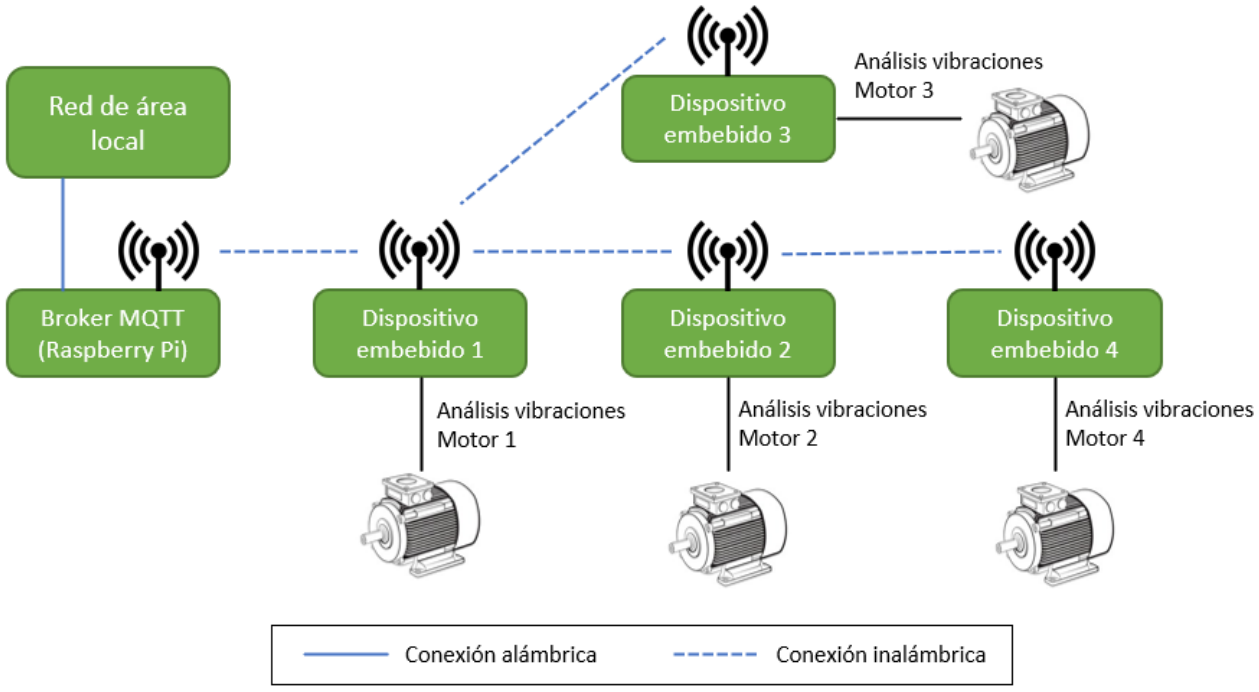


Fig. 4.3: Esquema de red inalámbrica propuesta.

Entonces, apoyándonos en la topología Mesh de la Fig. 4.2, podemos representar el diseño de la red como el esquema de la Fig. 4.3, en donde cada nodo representa a un dispositivo embebido

realizando el análisis de fallas operacionales por medio de vibraciones para cada motor en particular. Cabe destacar que seguimos bajo la topología Mesh, por lo que se permiten conexiones como el Dispositivo embebido 3 por ejemplo. Además mencionar que es una topología escalable, por lo que se puede ampliar la red para más dispositivos.

Para establecer la red en el protocolo MQTT haremos uso del software *Eclipse Mosquitto*, un Broker MQTT de código abierto que configuraremos en la Raspberry Pi para poder suscribirnos a los tópicos de interés a usar, y estará configurada como QoS 0 por la estabilidad de la red y porque en caso de haber un problema con una máquina el sistema lo reconocerá cada vez que se analice el sistema por lo que se seguirán enviando mensajes de alerta.

Los tópicos a utilizar, y a los que está suscrito el Broker, son:

- meshIMUs/MAC_ROOT/alerta:

  En este tópico se publicarán los mensajes de alerta relacionadas a anomalías operacionales, contendrán información relevante en cada caso.

- meshIMUs/MAC_ROOT/IMUs:

  En este tópico se publicarán las direcciones MAC de los dispositivos conectados, incluyendo el nodo Root en la primera posición de la lista. Cada vez que ingrese o salga un dispositivo, se publicará un mensaje con la lista actualizada en este tópico.

La dirección MAC_ROOT de los tópicos nos sirve para tener siempre identificado el dispositivo que corresponde al nodo Root. En caso de dudas nos podemos suscribir al tópico "meshIMUs/+/IMUs", en donde podremos recibir los mensajes de cualquier dispositivo Root que envíe un mensaje dentro del tópico meshIMUs y como ruta final IMUs. Como en esta ruta se publicará la lista de todos los dispositivos conectados podremos identificar fácilmente la dirección MAC del nodo Root. De forma análoga también nos podemos suscribir al tópico "meshIMUs/+/alerta", y recibir los mensajes de alerta.

CAPÍTULO 5

Programación microcontrolador: Configuración inicial y lectura de datos del sensor.

5.1. Introducción

Habiendo ya diseñado la electrónica y definido el comportamiento en red del sistema procedemos al desarrollo del código que dominará el hardware de cada dispositivo. Este desarrollo lo dividiremos en 3 etapas: la configuración del hardware, la matemática necesaria para implementar los algoritmos de detección de fallas y el establecimiento de la red inalámbrica tipo Mesh. La configuración del hardware consiste en la puesta en marcha del microcontrolador y la comunicación, configuración y lectura de datos del IMU.

La programación del microcontrolador se realizará bajo el software ESP-IDF o Espressif IoT Development Framework, un entorno de desarrollo puesto a disposición por el fabricante. ESP-IDF incluye las herramientas de software necesarias para crear, configurar, compilar y escribir proyectos para ESP32. Incluye la implementación de *freeRTOS*, un sistema operativo de tiempo real que nos permite trabajar con multiprocesos simétricos (SMP), es decir, nos permite ejecutar algoritmos utilizando los dos núcleos disponibles en el ESP32. *freeRTOS* ejecuta los procesos dependiendo la prioridad y los recursos disponibles del microprocesador, ordenando y ejecutando las tareas en un núcleo o en otro sin interrumpir otra tarea en marcha. Además, nos permite establecer en qué núcleo se ejecutará una tarea en particular, en nuestro caso haremos uso de esta función para tener un núcleo trabajando en la lectura constante y continua del IMU y el otro núcleo para hacer los cálculos matemáticos relacionados con los algoritmos de detección de fallas. Esto nos permite una obtención de datos de acelerómetro de forma ininterrumpida.

El algoritmo general del sistema queda descrito en la Fig. 5.1, en donde podemos destacar la jerarquía de las funciones a realizar. En primer lugar se debe inicializar el ESP32 de forma que podamos hacer uso de las características de hardware necesarias para la implementación del sistema, y luego de forma paralela, inicializar la red y el sensor IMU BMI270, para finalmente poder realizar las acciones requeridas por el sistema.

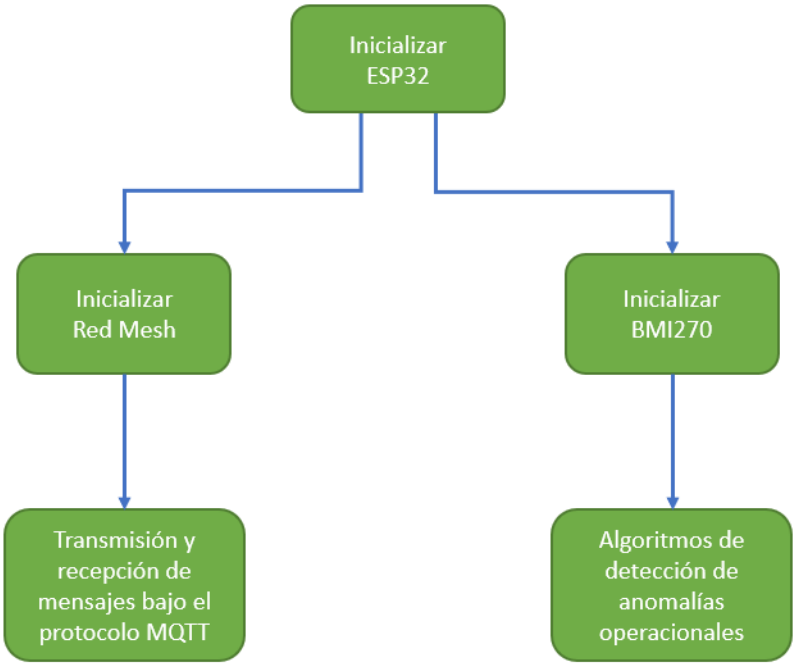


Fig. 5.1: Algoritmo general propuesto.

5.2. Puesta en marcha microcontrolador e IMU

La inicialización del ESP32 corresponde a la configuración de parámetros que establecen las características de hardware a usar del SoC. Para esto, haremos uso de la interfaz de configuración implementada por ESP-IDF, que además de ofrecer la configuración de los componentes del ESP32 también podemos configurar parámetros asociados a la compilación y escritura de programas. En esta oportunidad la configuración por defecto nos ofrece todas las características que son exigidas por el proyecto. Entre las opciones más destacadas está la habilitación del WiFi, la desactivación del Bluetooth, y la habilitación de *freeRTOS*. En la Fig. 5.2 podemos ver las opciones generales del microprocesador, dentro de lo más destacado es la opción de definir la frecuencia de reloj del CPU establecido en 160 MHz, suficiente para correr todo el sistema. Cabe destacar que todas estas opciones las podemos definir directamente en código, pero por comodidad aprovechamos las bondades del entorno de desarrollo.

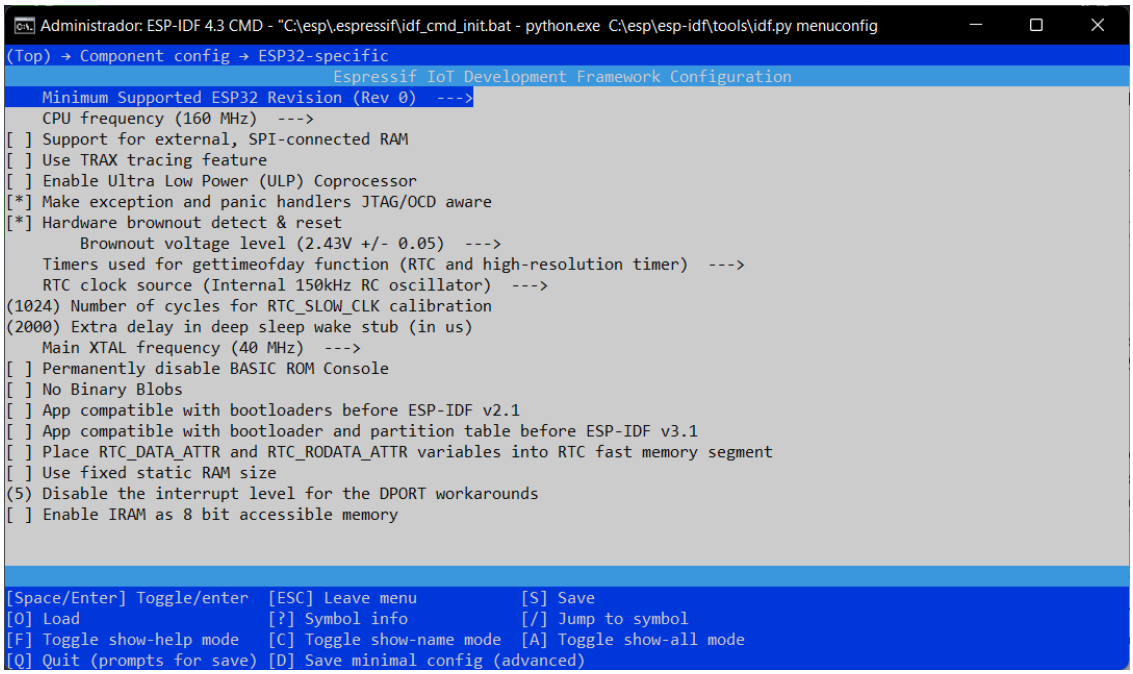


Fig. 5.2: Interfaz visual de configuración ESP-IDF.

La programación inicial del sistema consiste en la definición de parámetros, la inicialización de la comunicación I^2C del ESP32 con el BMI270, y luego la inicialización de este último.

Para la configuración de la comunicación I^2C, primero se establece que el ESP32 hará uso de su primer puerto I^2C y lo hará como maestro. Además se definen los pines para las señales SDA y SCL (21 y 22 respectivamente) y la frecuencia de la comunicación, que en este caso se configura al estándar fast mode I^2C de 400kHz, compatible con el BMI270. La dirección de esclavo y los bits de escritura, lectura y acknowledge del BMI son establecidos por el fabricante y presentados en su datasheet [37].

La configuración del BMI270 se realiza leyendo y escribiendo registros en específico de su memoria, el código para esto lo podemos ver en Anexo. Para realizar estas acciones bajo el protocolo I^2C se debe seguir un algoritmo detallado en su Datasheet [37] que consiste en una serie de pasos que incluye el uso de los bits START, STOP y ACKS, comentados en el Capítulo 3.3.1, para preparar al chip para escribir datos en registros o para poner a disposición un registro para su lectura.

Posterior a esto procedemos a inicializar el BMI270, proceso que consiste en la carga de un arreglo llamado *bmi270_config_file*, un vector de datos de configuración de 8kB puesto a disposición por el fabricante, para luego configurar los modos de control de energía, en donde habilitamos el funcionamiento del acelerómetro y desactivamos el giroscopio, sensor de temperatura y la función de sensor auxiliar.

Los parámetros disponibles para la configuración del acelerómetro tienen relación con la frecuencia de muestro y el filtro pasa bajos disponible en el chip. Para definir esto debemos considerar la naturaleza del proyecto, que en nuestro caso consiste en la lectura de vibración de un motor eléctrico de corriente alterna, en donde podemos considerar como uso frecuente una velocidad de funcionamiento de unos 3000 - 3600 RPM, es decir, de unos ciclos de 50 - 60 Hz, por lo que nos

interesa capturar los datos a una frecuencia mayor a los 120 Hz para obtener una representación completa de la señal según el teorema de muestreo de Nyquist-Shannon. Además, como queremos hacer un estudio en frecuencia de la vibración, en donde nos interesa su comportamiento en su frecuencia fundamental y en sus armónicos siguientes, establecemos la frecuencia de muestreo en 800 Hz, lo que nos ofrece un espectro de frecuencias de 0 a 400 Hz (aplicando una FFT), en nuestro caso suficientes para el estudio de la frecuencia fundamental de la vibración y de sus primeras seis armónicas. El filtro pasa bajos incluido en el chip lo dejamos configurado en el modo normal, el cual tiene una frecuencia de 3dB, lo que a esta frecuencia de muestreo correspondería a una frecuencia de 343 Hz, por lo que solo tendremos un espectro de frecuencias de la vibración de las primeras cinco armónicas. El rango de aceleraciones quedará por defecto en +/-8g, más que suficiente para registrar el funcionamiento normal y golpes o fallas considerables.

Finalmente los datos del acelerómetro se consiguen leyendo el registro ACC_DATA. Para asegurarnos una lectura de datos sincronizada con la frecuencia de muestreo del sensor haremos uso de un flag disponible en el registro STATUS en su octavo bit, un bit que será 1 cuando un dato de acelerómetro se encuentre disponible y que se hará 0 cuando el dato sea leído. Con esto logramos obtener la información del acelerómetro de forma correcta.

El algoritmo para la inicialización del BMI270 queda descrito en la Fig. 5.3.

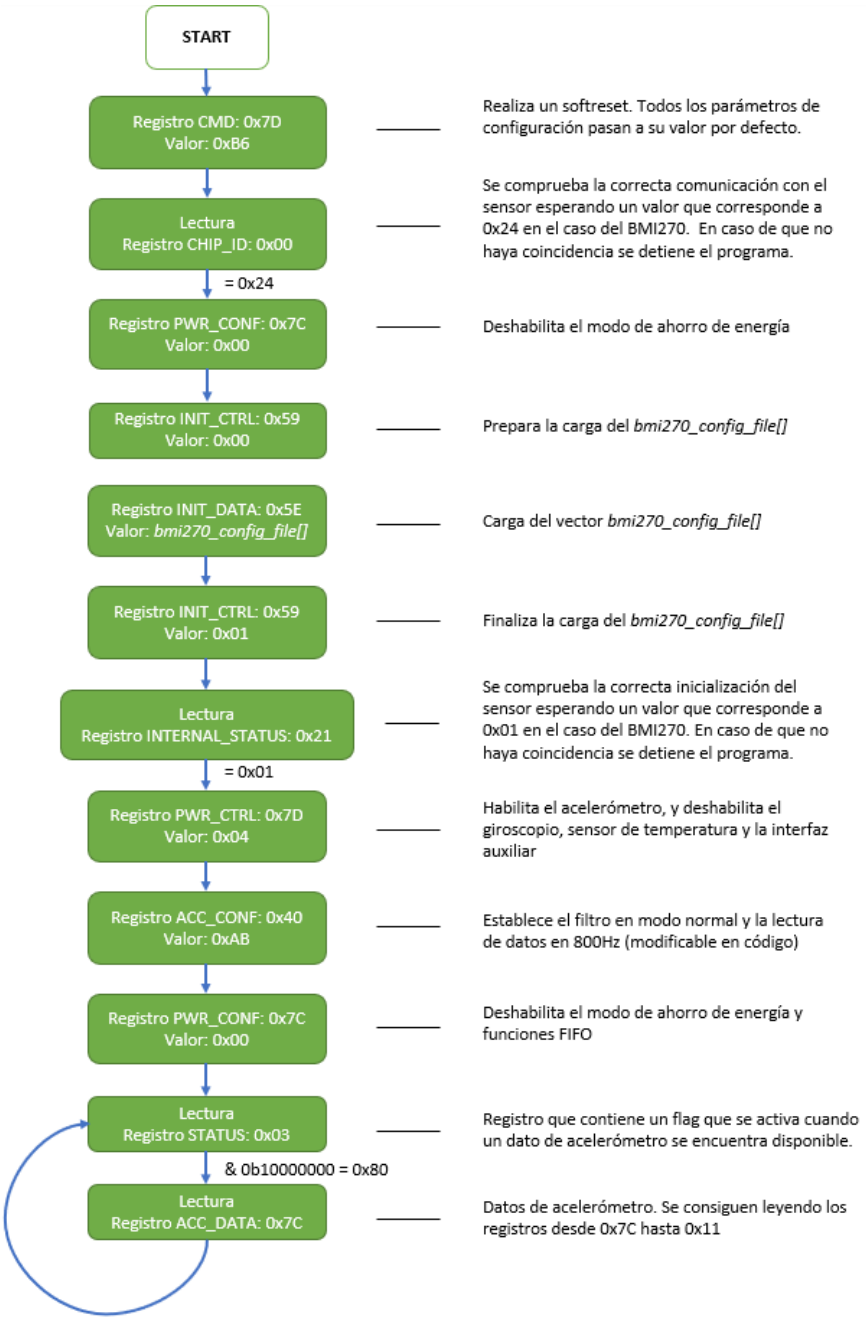


Fig. 5.3: Inicialización BMI270.

5.3. Lectura de acelerómetro

Luego de haber inicializado el sensor tendremos acceso a los registros del IMU que contienen los datos de cada lectura realizada. Estamos interesados en obtener los datos de aceleración en $[m/s^2]$ por comodidad al momento de analizar y comparar vibraciones. El sensor entregará la información de cada eje en un arreglo de 16bytes. Con 16bytes podemos contar desde el 0 hasta el 65536 por lo que, considerando que el acelerómetro está configurado para lecturas en un rango que va desde el -8 [g] hasta 8 [g] y que 1 [g] = 9.80665 $[m/s^2]$, podemos convertir la información en bytes a un dato

en $[m/s^2]$ dada la relación (78.4532/32768). El sensor entonces será capaz de medir amplitudes de vibración en el rango de los +/- 78.4532 $[m/s^2]$.

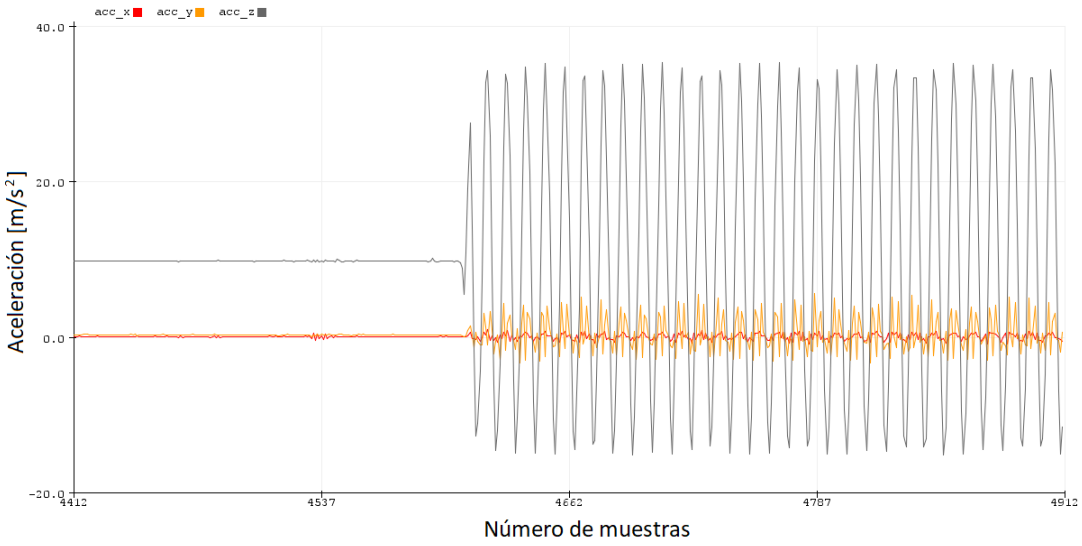


Fig. 5.4: Lectura de acelerómetro.

Las lecturas representadas en la Fig. 5.4 fueron obtenidas con el dispositivo montado en un instrumento de pruebas llamado Mini SmartShaker K2007E, un excitador electrodinámico que agitará un imán permanente dependiendo de una señal de entrada. Esta implementación queda descrita en la Fig. 5.6 y nos permitirá realizar pruebas con vibraciones. En este caso la señal de entrada es obtenida desde el generador de funciones GW INSTEK GFG-8216A, en donde se configuró para una señal análoga de salida sinusoidal de 20 Hz. En la Fig. 5.4 se pueden ver las lecturas de aceleración con el Shaker apagado (sin vibración) y luego vibrando. Podemos notar además que en la posición actual del acelerómetro los ejes x e y marcan 0 $[m/s^2]$, en cambio el eje z se encuentra en un valor cercano a los 9.8 $[m/s^2]$ en reposo, resultado esperado debido a la posición del sensor. En la Fig. 5.5 podemos ver la distribución de los ejes del acelerómetro del BMI270, lo que corrobora lo anteriormente mencionado. Finalmente comentar que el eje z, en esta aplicación, es el más sensible a la vibración y el que más información nos ofrece a la hora de identificar fallas operacionales en un motor, sin embargo, todos los algoritmos implementados utilizarán la información de los tres ejes del acelerómetro para tomar decisiones.

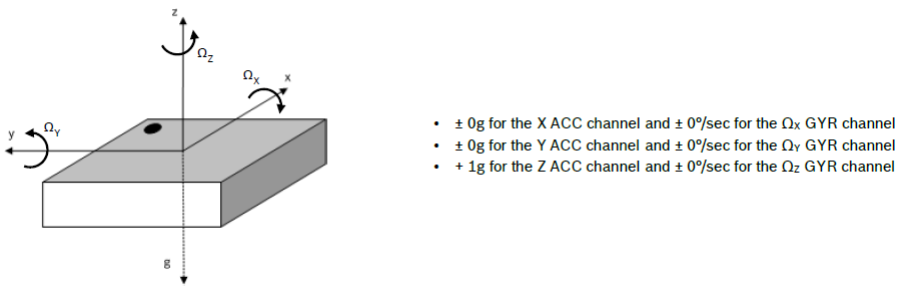


Fig. 5.5: Definición de orientación de los ejes de detección del BMI270 [37].

Fig. 5.6: Entorno de pruebas con Mini SmartShaker K2007E, GW INSTEK GFG-8216A y la placa embebida.

A continuación, en las Figuras de la 5.7 a la 5.9, podemos ver las lecturas de acelerómetro del eje z para una vibración de 20 Hz medida a distintas frecuencias de muestro. Podemos comentar que al usar una frecuencia de muestreo de 200Hz obtenemos formas puntiagudas en los picos de la onda, lo cual se aleja de lo que podríamos esperar de la representación de la realidad. Una medición fidedigna se puede obtener en este caso con una frecuencia de muestreo de 400Hz o superior.

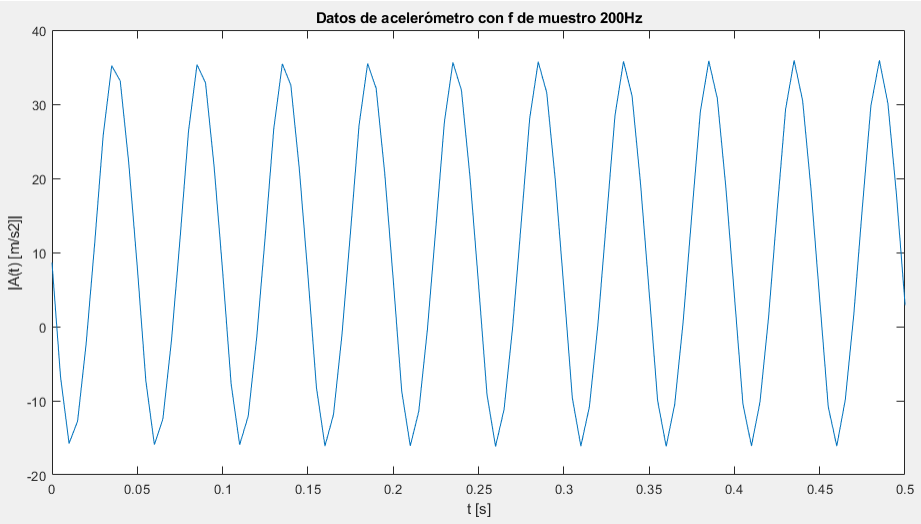


Fig. 5.7: Lectura de acelerómetro a 200 Hz.

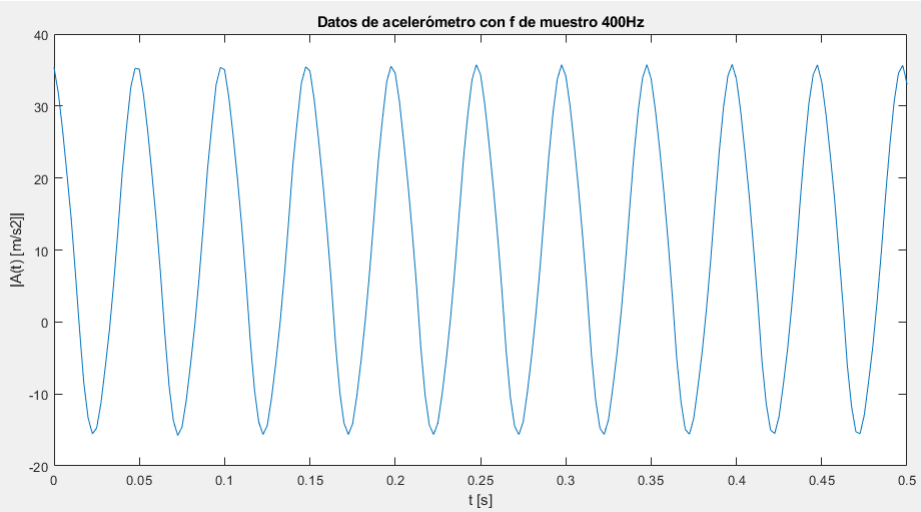


Fig. 5.8: Lectura de acelerómetro a 400 Hz.

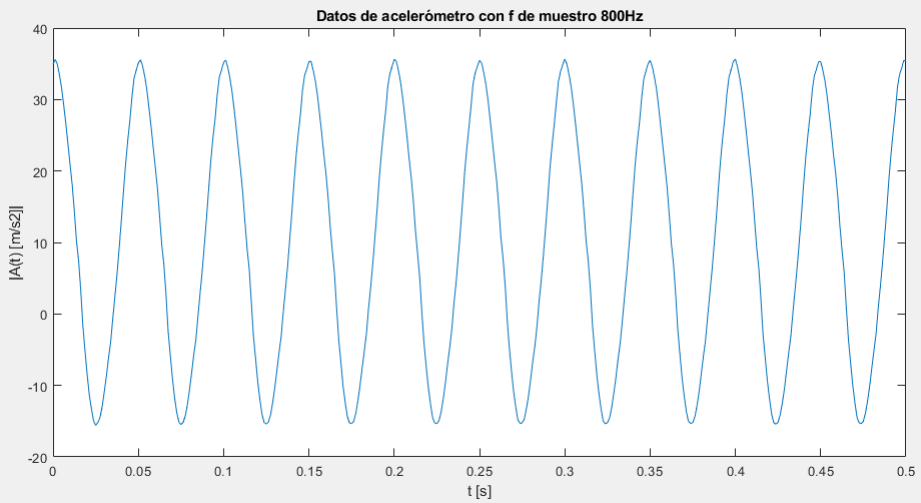


Fig. 5.9: Lectura de acelerómetro a 800 Hz.

CAPÍTULO 6

Algoritmo de detección de eventos

6.1. Introducción

Como hemos discutido en el capítulo 1.1.3, es posible detectar anomalías operacionales de un motor conociendo y analizando su vibración. Para este proyecto hemos desarrollado dos algoritmos que nos permitirán detectar estos problemas, uno en el dominio del tiempo y otro en el dominio de la frecuencia. En el primer algoritmo lo que haremos será calcular el valor eficaz o RMS de la señal de vibración, con el fin de contar con un indicador que nos permita identificar un comportamiento general de la vibración, permitiendo identificar una vibración de características indeseadas en el motor. Es el algoritmo que se ejecuta más veces. El segundo algoritmo realiza un análisis del espectro de frecuencias de la señal, en donde se le realiza una transformada de Fourier a la señal para determinar su composición en frecuencias y así poder identificar comportamientos indeseados con respecto al espectro de frecuencias de referencia y sus armónicos.

Estos algoritmos realizan cálculos matemáticos con los datos recientemente obtenidos por el acelerómetro y guardan los resultados en variables, que luego son comparadas con variables de referencia de un comportamiento deseado de la vibración. Para esto es necesario realizar un estudio preliminar a la implementación con el fin de poder establecer las variables que describen el escenario ideal de un motor sano y es información que se debe tener a disposición para determinar la magnitud del problema.

Los algoritmos se ejecutarán tomando una ventana de datos obtenidos del acelerómetro con la finalidad de poder realizar los cálculos con un paquete de datos y no con los valores de uno en uno mientras se actualizan. Esta muestra de tiempo del comportamiento del motor corresponden a vectores que son usados como buffer de datos, que son llenados con los datos actualizados en tiempo real y son reescritos con valores nuevos después de haber completado su tamaño.

Como los algoritmos se ejecutan al momento siguiente de haber completado el vector buffer existe un momento en que se comienza a sobreescribir el inicio del vector antes y mientras se ejecutan los cálculos, alterando completamente los resultados. Es por eso que resulta necesario contar con dos buffers que puedan ir alternando su llenado con la finalidad de ejecutar los algoritmos con una muestra de datos sin alteración mientras se siguen guardando datos nuevos en el otro buffer.

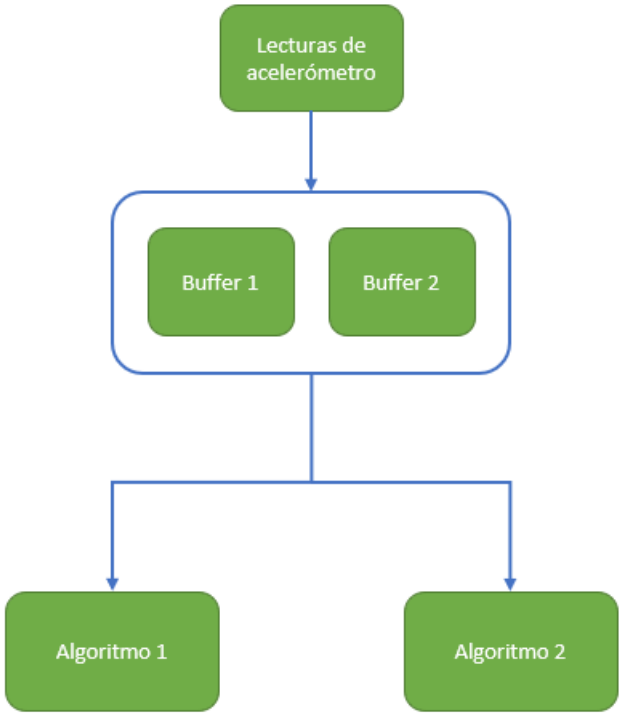


Fig. 6.1: Algoritmo Buffer de lectura.

En la Fig. 6.1 podemos ver que las lecturas de acelerómetro son guardadas en los buffers, y que los algoritmos toman los datos desde allí. Los buffers están diseñados para guardar la información de los tres ejes de acelerómetro disponibles. Comentar además que las lecturas de acelerómetro se realizan en el Nucleo 0 del ESP32 y que los algoritmos se ejecutan en el Núcleo 1. Con esto nos aseguramos de estar guardando todas las lecturas del sensor de forma interrumpida, sin pausas debido a los cálculos realizados por los algoritmos. Podemos determinal el núcleo con el que se ejecutan las tareas usando la función `xTaskCreatePinnedToCore()`, disponible gracias a *freeRTOS*.

Destacar que estos cálculos son realizados en un microprocesador, que a diferencia de un sistema computacional complejo, se alimenta solo con 3.3V y tiene un consumo del orden de las unidades de miliamperios.

6.2. Algoritmo 1: RMS

Este algoritmo calcula el promedio de los valores RMS de cada una de las muestras de la ventana de datos. El valor RMS nos permite representar la energía contenida en una señal sinusoidal periódica en un solo dato positivo, por lo que en la práctica nos permitirá comparar una ventana de datos actuales con una ventana de datos de referencia de forma sencilla y eficaz por medio de una sola varible que contiene toda la información.

Uno de los requisitos para implementar esta solución es que las señales se encuentren centradas en el origen, y como pudimos ver en el Capítulo 5.2 Fig. 5.5 por disposición de los ejes del acelerómetro siempre existirá un eje con valores distintos a 0, por lo que es necesario trasladar

las señales desde su posición de equilibrio hacia uno que sea igual a 0. Para esto se implementó una función en el código que obtiene la posición de equilibrio dada por la media aritmética de las señales de la primera ventana de datos, luego estos valores serán restados de las señales de las ventanas siguientes para que el cálculo RMS tenga sentido.

El resultado de este procedimiento es un valor RMS representativo de la ventana de tiempo actual que luego se compara con el valor RMS de referencia para determinar si el comportamiento del motor es el deseado o no. En caso de que no se corresponda el algoritmo envía un mensaje con información relevante alertando de la falla al servidor braker. El mensaje queda estructurado como JSON y contiene como primera información la fuente de la falla detectada, el número identidicador del dispositivo (que corresponde además a la identificación de un motor en particular) y luego el valor RMS medido. Como comentamos anteriormente el valor RMS de referencia es un valor conocido y no se incluye en el mensaje con la finalidad de minimizar el tamaño del mensaje.

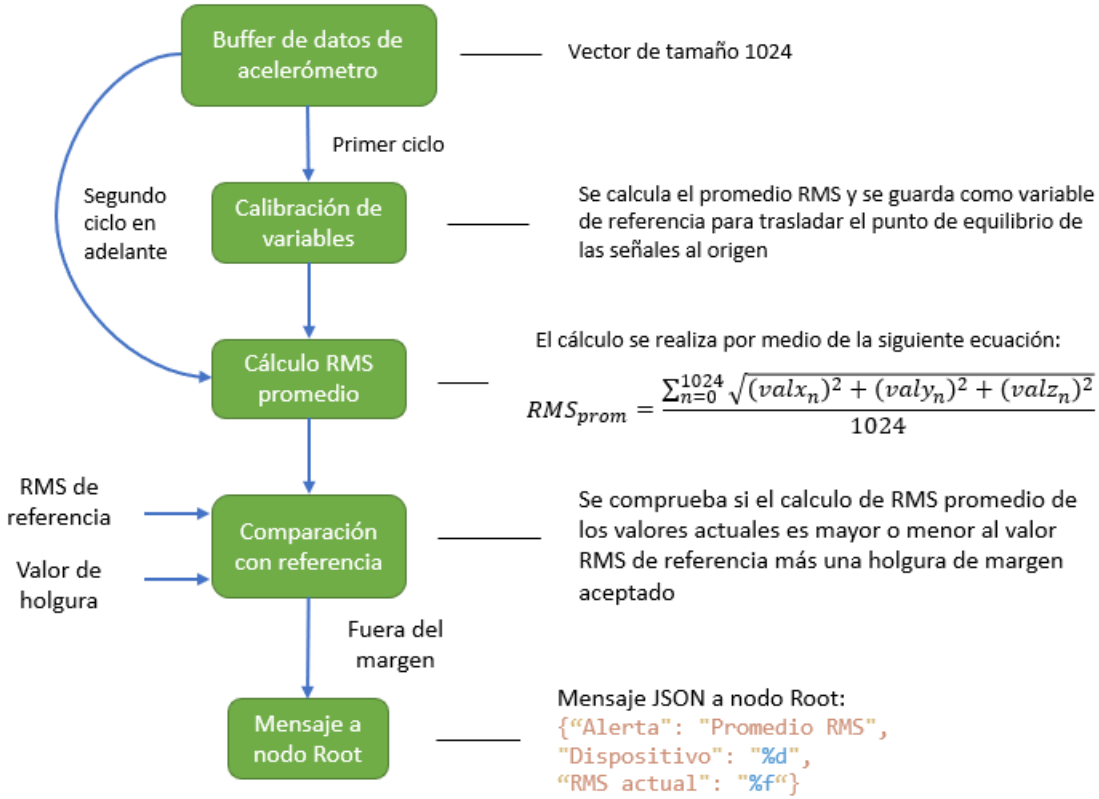


Fig. 6.2: Algoritmo 1: RMS.

En la Fig. 6.3 podemos ver el comportamiento de los valores RMS dependiendo del comportamiento del motor. Estas pruebas se realizaron con el sistema montado en el Mini SmartShaker K2007E mientras se fueron cambiando los parámetros de frecuencia y amplitud de la señal del generador de funciones. La lectura comienza con el shacker vibrando a 10Hz sin ninguna intervención, en este caso podemos esperar que los promedios RMS de la señal se mantenga en un valor constante, que es justamente lo que pasa en los primeros 7 puntos del gráfico. Luego se empezó a subir la frecuencia de la señal de entrada del shacker, desde los 10Hz hasta los 30Hz y luego se fue bajando hasta llegar a los 20Hz, esto lo podemos ver desde la muestra número 8 hasta la 48, en donde nos

mantenemos sin alteración por un instante de tiempo, comportamiento reflejado en las muestras número 49 al 52 en donde podemos ver una serie de puntos con valores de RMS cercanos entre sí. Además podemos comprobar que existe un aumento del promedio RMS debido al aumento de la frecuencia de la vibración, donde el valor RMS relacionado a la frecuencia de 30Hz se encuentra en una vecindad de la muestra 37 y marca un valor RMS de 23.7 m/s^2, superior al valor RMS de la vibración a 10 Hz que corresponde a 6.2 m/s^2. Luego desde la muestra 52 hasta la 59 aumentamos la amplitud de la señal de entrada al shacker, en los mismos 20Hz, lo que se traduce en una vibración con mayor recorrido. Esto lo podemos ver con un acelerómetro ya que como la vibración presenta un mayor recorrido, quiere decir que la amplitud máxima de las aceleraciones también aumenta, es por eso que en estos puntos de la gráfica podemos ver un aumento considerable en los valores de los promedios RMS. Finalizamos la prueba disminuyendo la amplitud de la señal de entrada al valor anterior y podemos ver que volvemos a obtener los mismos resultados anteriores para 20 Hz, tal como lo esperado.

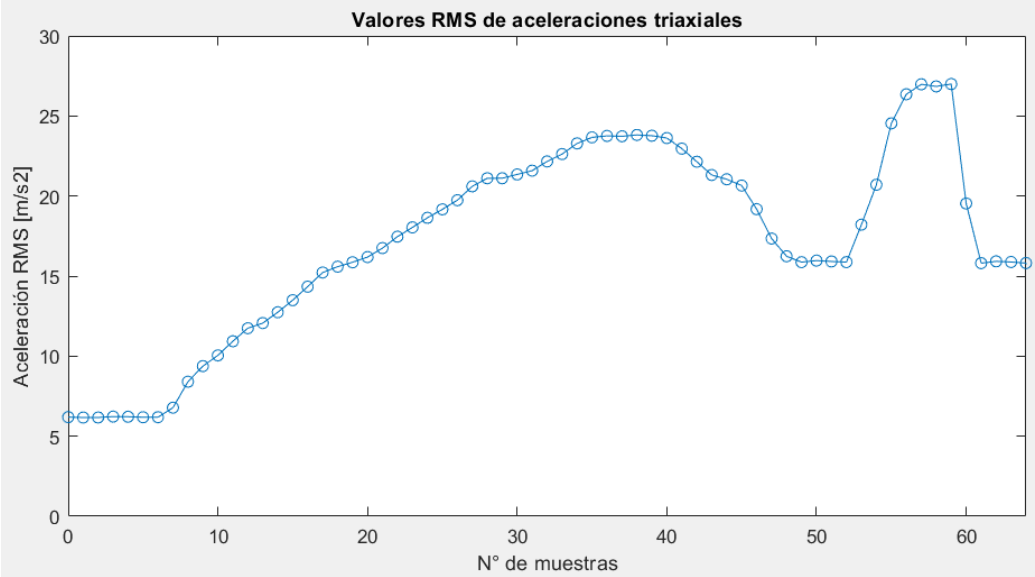


Fig. 6.3: Valores RMS.

Con esta prueba pudimos demostrar que los cambios en la vibración del motor sí se ven reflejados en los cálculos realizados en el ESP32. Lo interesante de esta aplicación es que como tomamos en consideración los 3 ejes del acelerómetro al momento de calcular el promedio RMS, podemos tener un dato que concentra toda la potencia disipada por la vibración. Además, al realizar los cálculos con ventanas que superan el segundo de tiempo también nos protegemos de cambios instantáneos de alta frecuencia ya que podemos ver este cálculo matemático como un filtro para la señal de acelerómetro.

6.3. Algoritmo 2: FFT

Para realizar un análisis en el dominio de la frecuencia lo que hacemos es calcular la transformada de Fourier de la señal de vibración, transformación matemática que nos permite transformar señales en el dominio del tiempo al dominio de la frecuencia, permitiéndonos descomponer la señal en

sus componentes por frecuencia. Esta técnica nos permite observar el espectro de frecuencias de la señal, de donde podemos identificar la frecuencia principal de la vibración, conocida como frecuencia fundamental, y si la señal tiene o no componentes relacionadas a frecuencias armónicas. Además de evidenciar la frecuencia también podemos conocer la magnitud de cada una de estas componentes, pudiendo determinar el grado de relevancia de esa frecuencia en la señal en cuestión.

La transformada de Fourier es una transformada para señales continuas por lo que, por la naturaleza de la electrónica digital, no podemos realizarla en un sistema computacional. Sin embargo, existe una técnica llamada Transformada discreta de Fourier o DFT, lo cual nos permite obtener la transformada de un muestreo discreto de una señal periódica. Lamentablemente este proceso requiere de hardware mas bien avanzado y potente para ejecutarlo en cortos períodos de tiempo, es por eso que nos enfocamos en la Transformada rápida de Fourier o FFT, un algoritmo mucho más eficiente para calcular DFT por medios computacionales. El tamaño del espectro resultante de una FFT depende de la frecuencia de muestreo de la señal y de la cantidad de muestras tomadas para el análisis, el cual debe ser obligatoriamente una potencia de dos, es por esto que el tamaño de la ventana a analizar corresponde a 1024 muestras. La frecuencia de muestreo del sensor es de 800 Hz, por lo que el espectro de frecuencias de la FFT en este caso nos permitirá analizar hasta una frecuencia de 400 Hz.

Para el calculo de la FFT haremos uso de la librería llamada `esp32-fft` [55], un código desarrollado en lenguaje C que permite realizar cálculos FFT pensando en el hardware de un ESP32. Esta librería utiliza el algoritmo Cooley-Tukey, que permite dividir una DFT en varias DFT más pequeñas con el fin de optimizar la velocidad de computación. La librería permite realizar el cálculo por medio de una sola función asociada, a la cual se le entregan como parámetros el tamaño de la ventana, si se trata de una transformación real o compleja, si es una transformación directa o inversa y dos vectores a modo de buffer de entrada y de salida. En nuestro caso el tamaño de la ventana es 1024, es una transformara real debido a que los datos de acelerómetro son reales, y es una transformación directa, es decir, del dominio del tiempo al dominio de la frecuencia.

```
Input  : [ x[0], x[1], x[2], ..., x[NFFT-1] ]
Output : [ X[0], X[NFFT/2], Re(X[1]), Im(X[1]), ..., Re(X[NFFT/2-1]), Im(X[NFFT/2-1]) ]
```

Fig. 6.4: Tamaño de buffers y organización de los datos librería `esp32-fft` [55].

La Fig. 6.4 nos muestra el orden de los vectores de entrada y salida para un vector de entrada de tamaño NFFT. Lo primero que podemos notar es que el tamaño del vector de salida tiene la mitad del tamaño que la entrada y es de esperar por el tipo de algoritmo que se está aplicando. Otra cosa interesante es analizar el orden del vector de salida, y es que su primer componente corresponde a la magnitud de la frecuencia 0, o frecuencia DC, el segundo componente corresponde al último valor del resultado, y los valores siguientes irán presentando los resultados imaginarios restantes, primero su componente real y en el siguiente espacio su componente imaginaria. Para realizar un análisis correcto de estos resultados debemos reordenar el vector de salida a nuestra conveniencia, en donde el primer componente del vector ordenado corresponda al primer componente del vector de salida, el último componente del vector ordenado corresponda al segundo componente del vector de salida, y el resto de componentes contengan el doble del valor absoluto de las componentes reales

e imaginarias por cada uno de los resultados. Gracias a este reordenamiento podemos presentar los resultados de forma gráfica, mejorando su comprensión.

Luego de obtener un vector resultado que se corresponda con la FFT de la señal original ejecutamos un pequeño algoritmo que buscará los peaks en las magnitudes de los resultados, obteniendo además la frecuencia a la que corresponden estos datos. Estos peaks de la FFT corresponden a las magnitudes de interés para el análisis de frecuencia, de donde el peak con las importancia es el peak con mayor magnitud y estará presente en la frecuencia fundamental. Los peaks siguientes estarán ordenados en valores cercanos a múltiplos de esta frecuencia y corresponderán a las componentes armónicas de la señal.

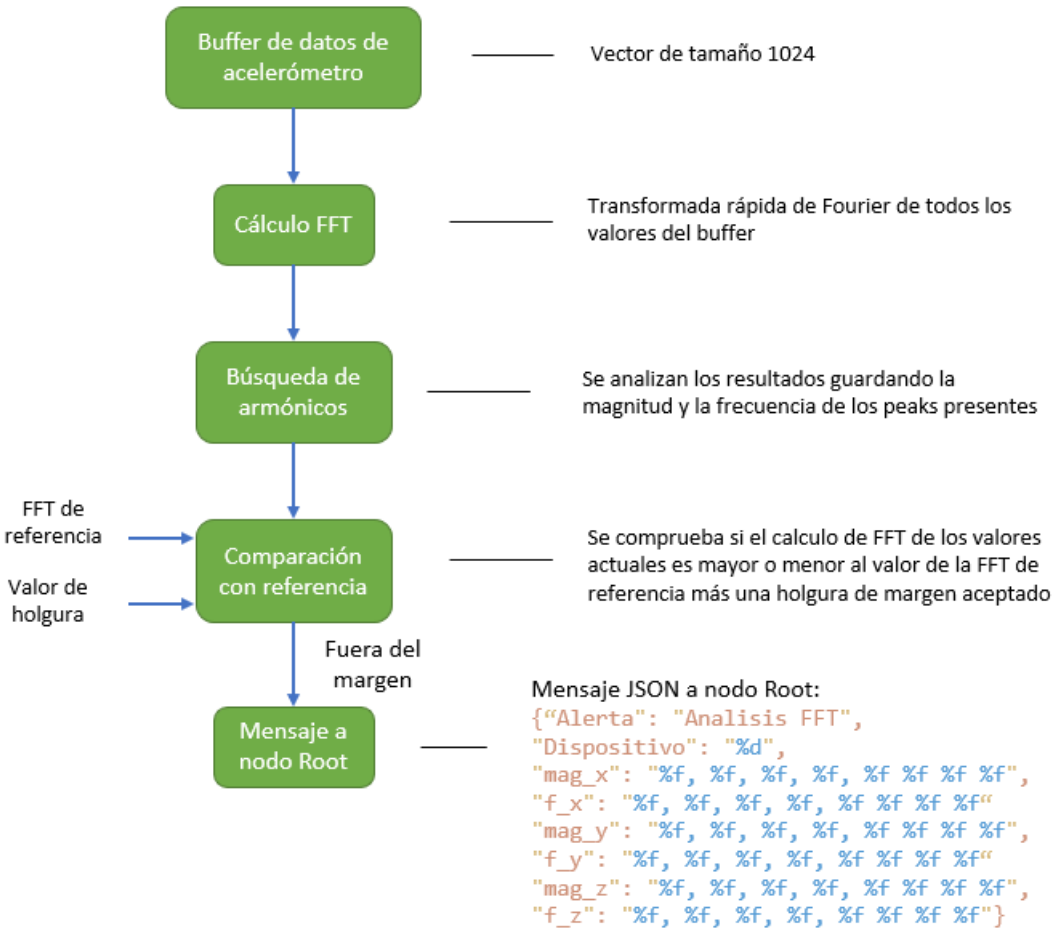


Fig. 6.5: Algoritmo 2: FFT.

Para comprobar la fidelidad de los resultados obtenidos por el microcontrolador haremos una comparación con una fft calculada en un sistema computacional complejo en el software Matlab. Los cálculos son realizados con los mismos datos de la ventana usada en el ESP32 y se ejecutan usando el comando `fft` para los 3 ejes.

En la Figura 6.6 podemos ver los cálculos obtenidos por el microcontrolador del sistema embebido, realizando mediciones en el shacker de pruebas vibrando a una frecuencia determinada, misma implementación que en la Fig. 5.6. En la Fig. 6.7 podemos ver una comparación gráfica entre

los cálculos de fft obtenidos en el sistema embebido y los resultados de Matlab usando la misma ventana de datos de acelerómetro, de donde podemos concluir que los algoritmos implementados en el ESP32 tienen unos resultados bastante cercanos a un algoritmo de un software sofisticado ejecutado en un procesador complejo. Además, en la Fig. 6.8 podemos ver la correcta identificación de armónicos por el código del algoritmo.

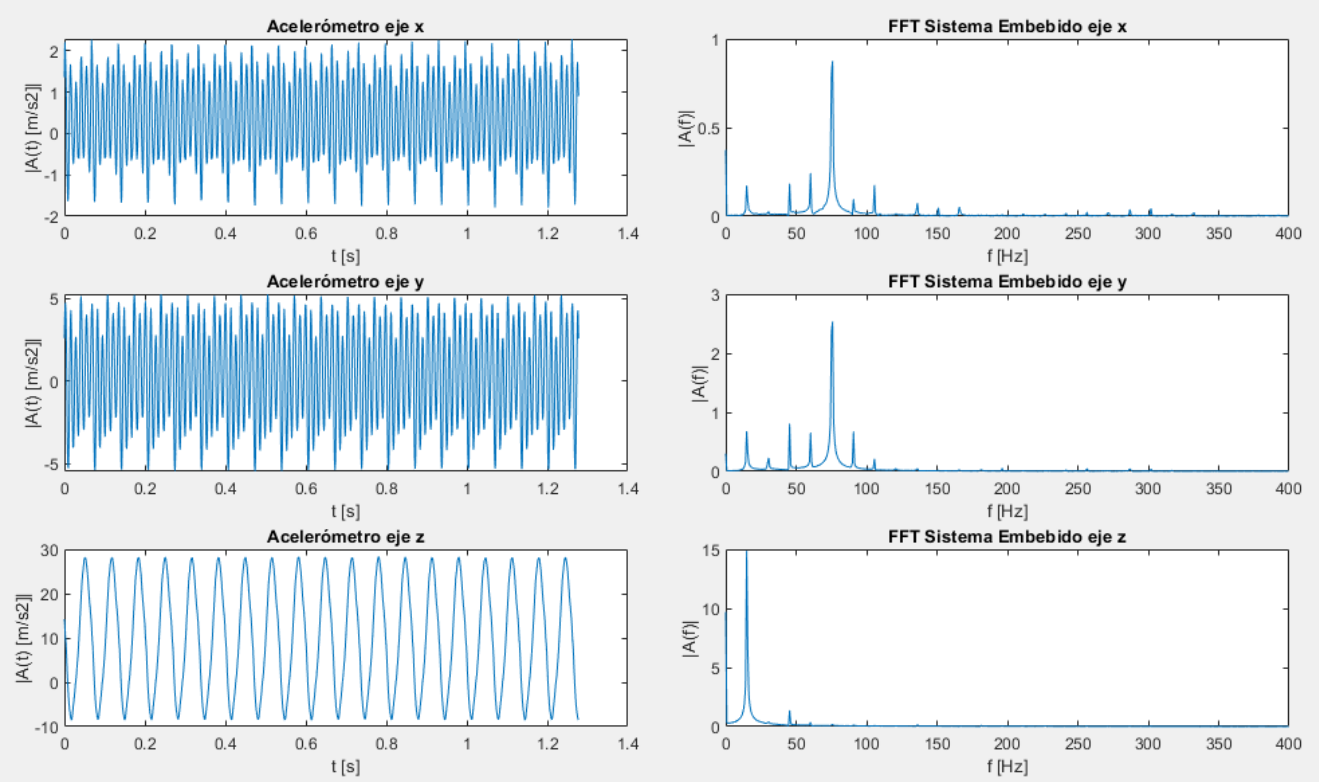


Fig. 6.6: Gráfico señal de vibración obtenidas por el acelerómetro y sus espectros de frecuencias calculada en el ESP32.

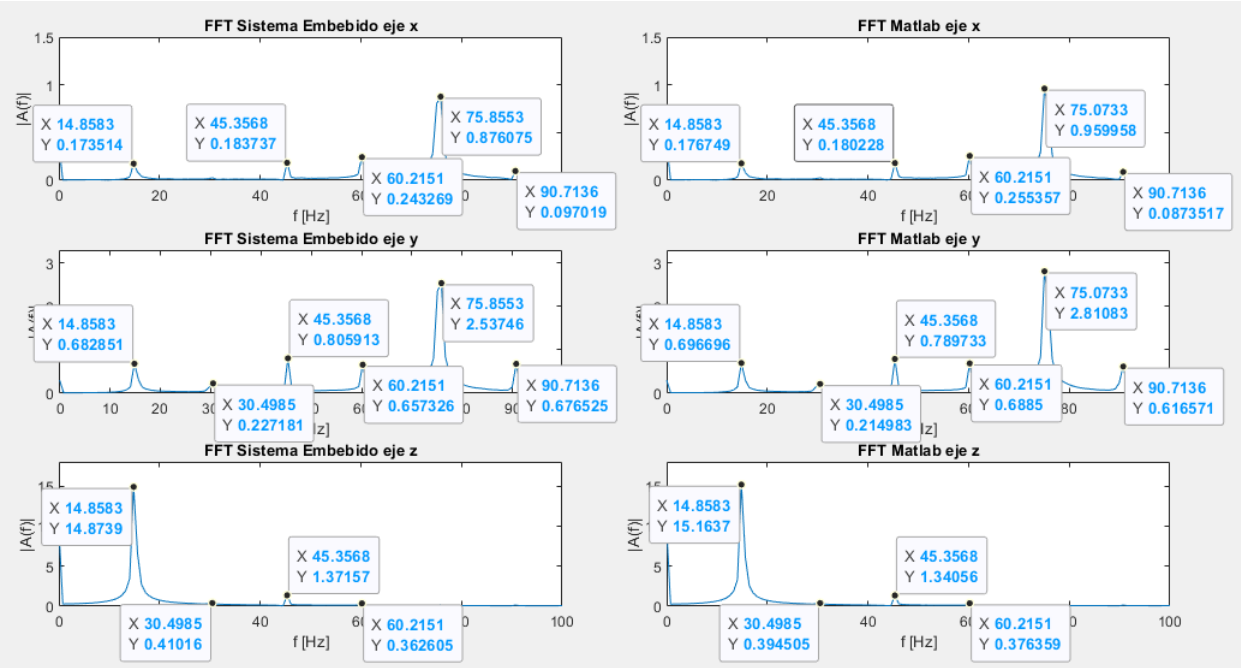


Fig. 6.7: Comparación de espectros de frecuencia calculado en el ESP32 v/s Matlab.

```
EJE X
farmonico | marmonico
14.843750 | 0.173147
45.312500 | 0.184307
60.156250 | 0.243782
75.781250 | 0.875505
90.625000 | 0.096471
105.468750 | 0.175942
0.000000 | 0.000000
0.000000 | 0.000000

EJE Y
farmonico | marmonico
14.843750 | 0.682602
30.468750 | 0.227118
45.312500 | 0.806367
60.156250 | 0.657556
75.781250 | 2.536999
90.625000 | 0.676350
105.468750 | 0.210426
0.000000 | 0.000000

EJE Z
farmonico | marmonico
14.843750 | 15.169125
30.468750 | 0.396149
45.312500 | 1.371993
60.156250 | 0.364215
75.781250 | 0.207752
90.625000 | 0.153198
105.468750 | 0.114898
135.937500 | 0.162753
```

Fig. 6.8: Armónicos identificados por el algoritmo.

Si bien el instrumento de prueba Mini SmartShaker K2007E genera una vibración mas bien limpia a partir de una señal de entrada con muy poca presencia de ruido, cosa que podemos evidenciar en particular en el ejes z de la Figura 6.6 en donde las armónicas distintas a la

fundamental son exageradamente de menor proporción de magnitud que esta frecuencia, sí podemos confirmar que el algoritmo es capaz de identificar estos picos y que será capaz de identificarlos cada vez que existan, como por ejemplo los cálculos realizados en los ejes x e y. Además, si no se está conforme con la selección de armónicos siempre se pueden ajustar los parámetros de holgura para tener los resultados deseados.

Finalmente comentar que a modo de desarrollo se probaron otras librerías compatibles con programas escritos en lenguaje C para calcular una FFT, entre ellas `arduinoFFT` [40], `kissfft` [56] y `minfft` [57]. Se prefirió finalmente el uso de `esp32-fft` por su comodidad a la hora de implementar el código y por su rendimiento, logrando realizar los mismos cálculos en menor cantidad de tiempo y usando menos memoria que sus competidores.

6.4. Tiempos de ejecución

Para conocer el tiempo que le toma al microprocesador en realizar los cálculos de los algoritmos nos apoyamos en una caracterísica del ESP_IDF que nos permite imprimir mensajes en el monitor serial indicando el tiempo transcurrido en ms desde que se inició el ESP32, esto nos permitirá conocer los tiempos de ejecución en el orden de los microsegundos. En la Fig. 6.9 podemos ver los promedios de tiempo que tarda el ESP32 en realizar las tareas, se consideraron 10 pruebas para cada una de ellas con un tamaño de ventana de 1024 valores. En la Fig. 6.10 podemos ver a modo de ejemplo la forma en que se obtuvieron los datos, el valor entre paréntesis corresponde al marcador de tiempo en donde se realiza la acción, en nuestro caso pusimos marcadores al inicio y al final de los algoritmos para determinar el tiempo que tarda en completar su tarea.

	Tiempo
Llenado de buffer	1280 [ms]
Algoritmo RMS	10 [ms]
Algoritmo FFT	30 [ms]

Fig. 6.9: Promedio de tiempo de ejecución de algoritmos para ventanas de 1024 valores.

```
I (82566) lectura: Buffer 1 sobreescribiendo. core = 0
I (83846) lectura: Buffer 1 completado. core = 0
I (83846) lectura: Buffer 2 sobreescribiendo. core = 0
I (85126) lectura: Buffer 2 completado. core = 0
```

```
I (17128) algoritmo1: Algoritmo RMS iniciado. core = 1
I (17138) algoritmo1: Algoritmo RMS finalizado. core = 1
I (18408) algoritmo1: Algoritmo RMS iniciado. core = 1
I (18418) algoritmo1: Algoritmo RMS finalizado. core = 1
```

```
I (224881) algoritmo2: Algoritmo FFT iniciado. core = 1
I (224911) algoritmo2: Algoritmo FFT finalizado. core = 1
I (226161) algoritmo2: Algoritmo FFT iniciado. core = 1
I (226191) algoritmo2: Algoritmo FFT finalizado. core = 1
```

Fig. 6.10: Marcadores de tiempo de ejecución de algoritmos para ventanas de 1024 valores. De arriba a abajo: Llenado de buffers, Ejecución algoritmo RMS y Ejecución Algoritmo 2.

Como era de esperar, el buffer tarda 1.28 s en llenar un vector de 1024 datos con una frecuencia de muestreo de 800 Hz. Notar además que con la prueba de la Fig. 6.10 también verificamos que los algoritmos se ejecutan de forma paralela utilizando los dos núcleos del microprocesador, en este caso usamos el Núcleo 0 exclusivamente la lectura y el Núcleo 1 para realizar los cálculos de los algoritmos. En el esquema de la Fig. 11 podemos ver una aproximación gráfica de la distribución de tareas en los núcleos, de donde se puede concluir que el margen de 1280 ms para llenado de los buffers es tiempo suficiente para realizar los cálculos de los dos algoritmos, 40 ms en total, sin que se sobreescriban los datos.

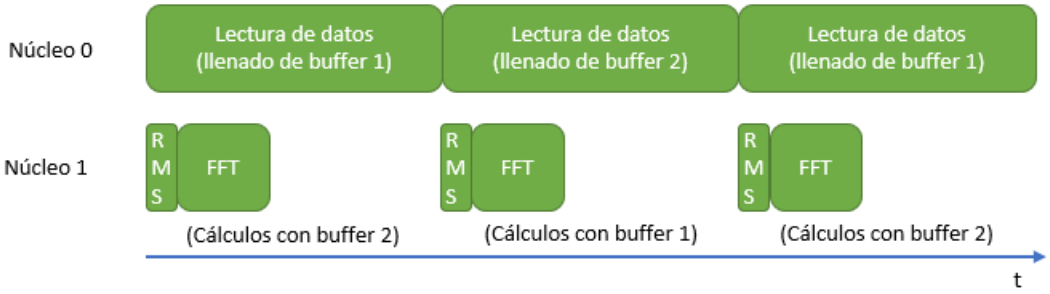


Fig. 6.11: Esquema de ejecución del programa en el microprocesador de dos núcleos.

6.5. Implementación de red MESH MQTT

Para finalizar el programa lo que haremos será implementar el código que permite la configuración y establecimiento de la red tipo Mesh, con mensajes de características MQTT. Para esto nos apoyaremos en el trabajo realizado en los algoritmos de detección de eventos ya fueron desarrollados con la finalidad de resolver todo el análisis en un mensaje que será construido y enviado sólo en el caso que sea necesario, por lo que solo falta implementar el envío del mensaje. Además de esto último también debemos configurar la red inalámbrica Mesh que cumpla con las características establecidas en el Capítulo 4.

Para establecer esta red usando los ESP32 haremos uso del entorno de desarrollo llamado ESP-MDF, que de forma similar el entorno ESP-IDF nos brinda un abanico de herramientas y

librerías ofrecidas por el fabricante, esta vez orientadas a la creación y administración de redes tipo Mesh. El complemento más importante de ESP-MDF es la API Mwifi (MESH Wifi) que nos permite configurar los microcontroladores a usar de forma individual, pudiendo definir de forma manual el tipo de nodo que será cada uno y con cuál se debe conectar, o dejando que el software tome las decisiones de forma automática, configurándolos para que al conectarse la red definan el tipo de nodo que mejor se relaciona con la red existente. Para este proyecto consideramos que es mejor utilizar la última opción, ya que así será más fácil a la hora de implementar el agregar o quitar dispositivos del sistema en general.

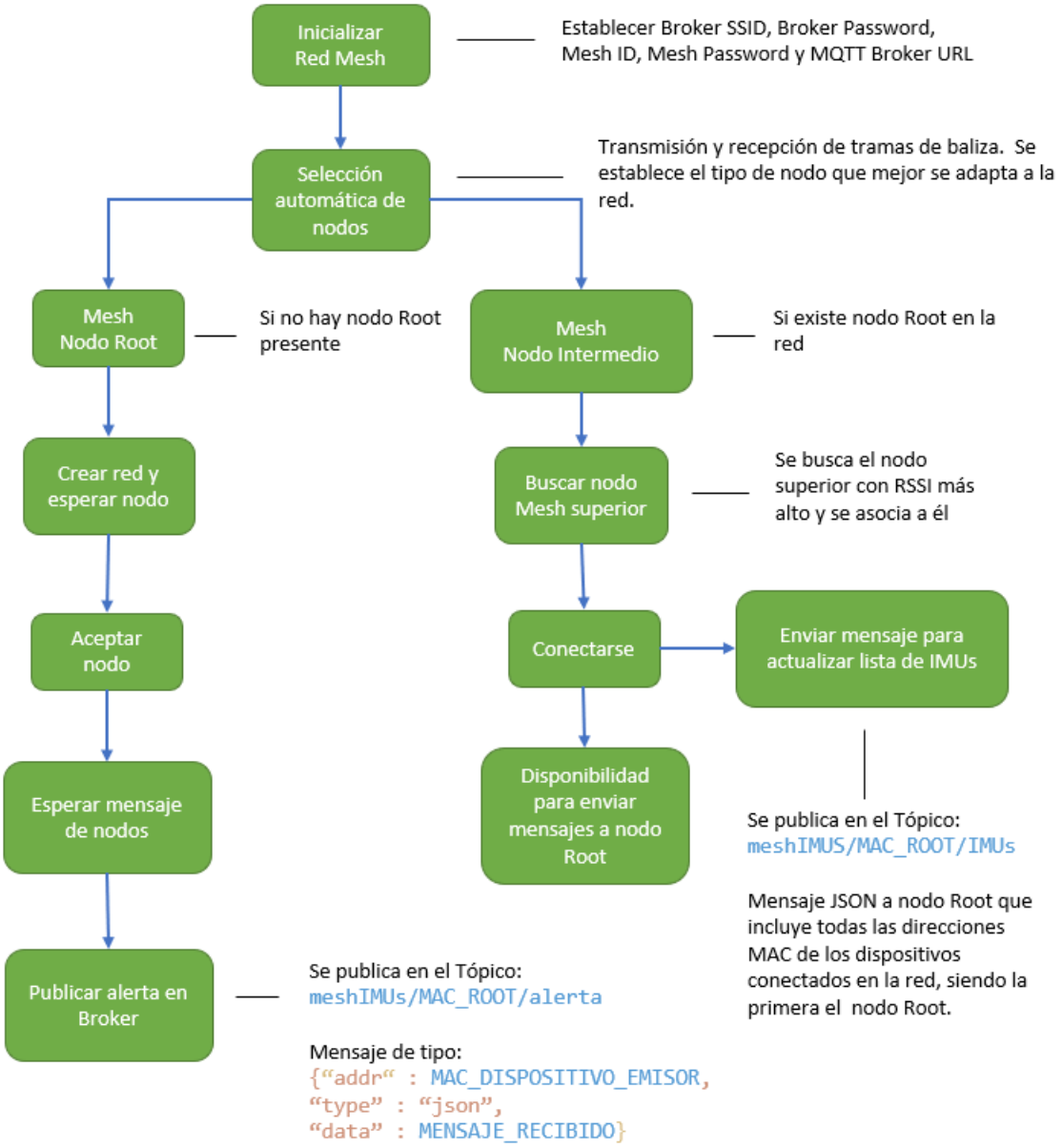


Fig. 6.12: Algoritmo red Mesh MQTT.

En la Fig. 6.12 podemos ver el algoritmo que domina la parte del red del sistema. De donde

se puede destacar primero la elección de establecer el dispositivo en cuestión como nodo red o como nodo intermedio. El nodo red será el primer dispositivo que se conecte a la red, y será el encargado de publicar los mensajes recibido por los otros dispositivos en el servidor Breaker. El nodo intermedio se encargará de establecer y permitir la inclusión de nuevos dispositivos en la red. Al conectarse exitosamente enviará un mensaje al tópico correspondiente con la finalidad de tener certeza de la conexión desde el servidor y para tenerlo identificado. El nodo root y los nodos intermedios están desarrollados para implementar y ejecutar los algoritmos relacionados al IMU.

Los tópicos de la configuración MQTT están configurados por medio de código. Los mensajes se prepararon para enviar la información necesaria para poder identificar el problema, el dispositivo en cuestión y el detalle del análisis con problema.

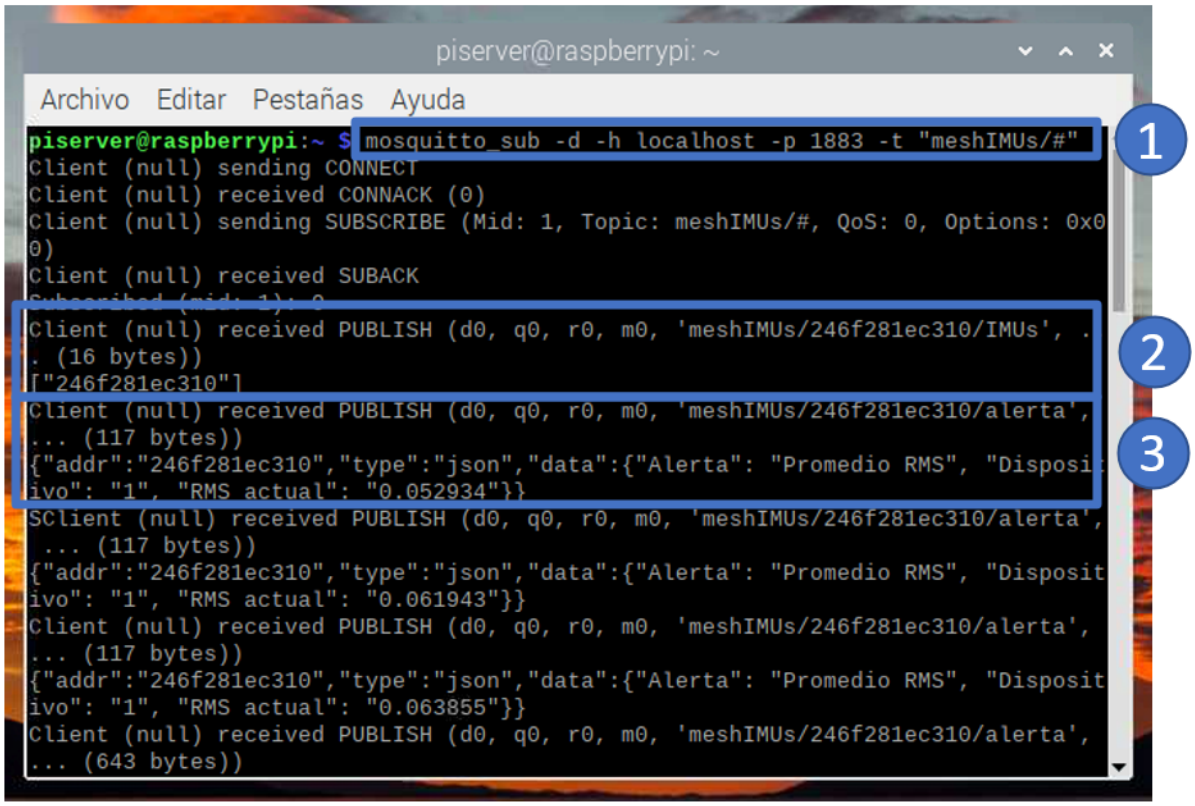


Fig. 6.13: Broker MQTT en Raspberry. 1. Suscripción a los tópicos de interés. 2. Publicación en tópico IMUs, el mensaje contiene las MAC de los dispositivos conectados. 3. Mensaje de alerta publicado por el dispositivo "1".

6.6. Implementación en laboratorio

Las pruebas fueron realizadas en un sistema harnero instalado en un laboratorio de la Facultad de Ingeniería. El sistema está compuesto por un motor eléctrico con una masa en su eje con la finalidad de que la base del harnero se agite con la ayuda de resortes montados en su base. La prueba consiste en poner en marcha el sistema harnero con el dispositivo atornillado sobre la carcaza del motor, ver Fig. 6.14. El dispositivo identificará la vibración del sistema motor - harnero, y luego

de hacer el análisis de sus vibración podremos definir los parámetros de la condición ideal - sana del sistema.

Fig. 6.14: Sistema harnero.

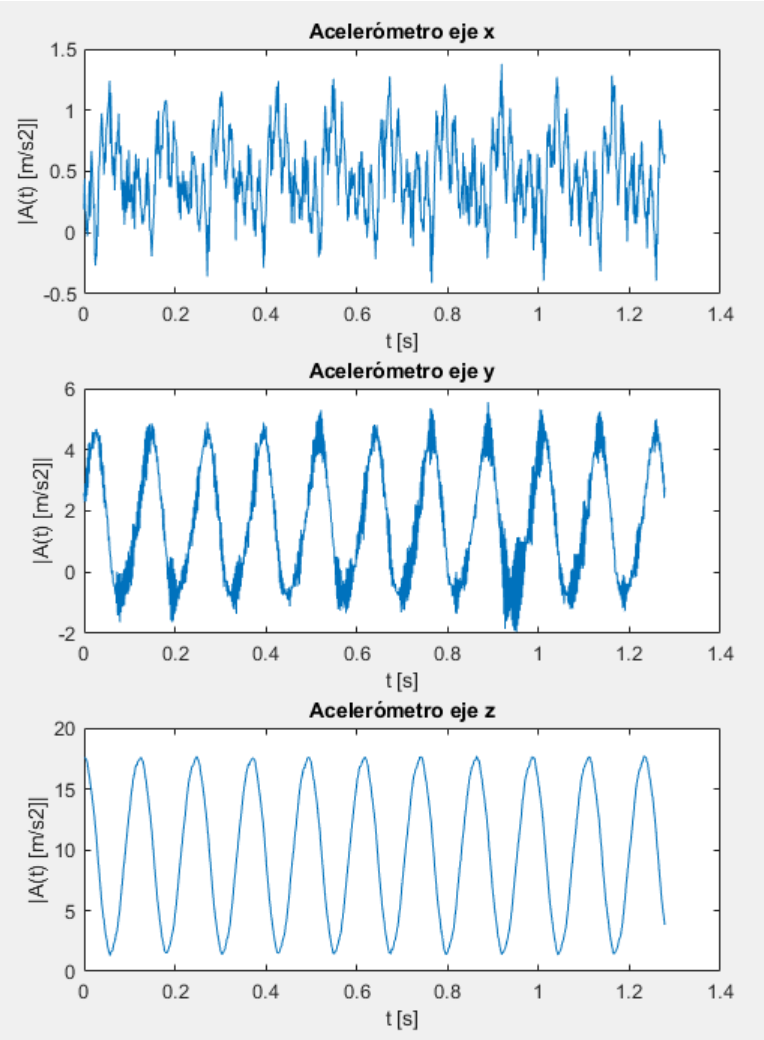


Fig. 6.15: Ventana de aceleraciones del sistema harnero.

A continuación, en la Fig. 6.16 podemos ver el cálculo del espectro de frecuencias de la muestra de la Fig. 6.15 calculado por el sistema embebido comparado con los cálculos realizados en el software Matlab. En la Fig. 6.17 podemos ver una imagen ampliada de ambos cálculos FFT y en donde podemos ver claramente la cercanía de los resultados. En la Fig. 6.18 podemos ver el resultado del detector de harmónicos del algoritmo del sistema embebido, podemos ver también que el sistema cumple con la identificación de los puntos de interés de los cálculos, de donde podemos ver que el sistema tiene una vibración fundamental de 7.8 Hz. Comentar que en el eje y se identifican harmónicos en frecuencias superiores a 350 Hz, esta información puede ser relevante o no dependiendo de la aplicación.

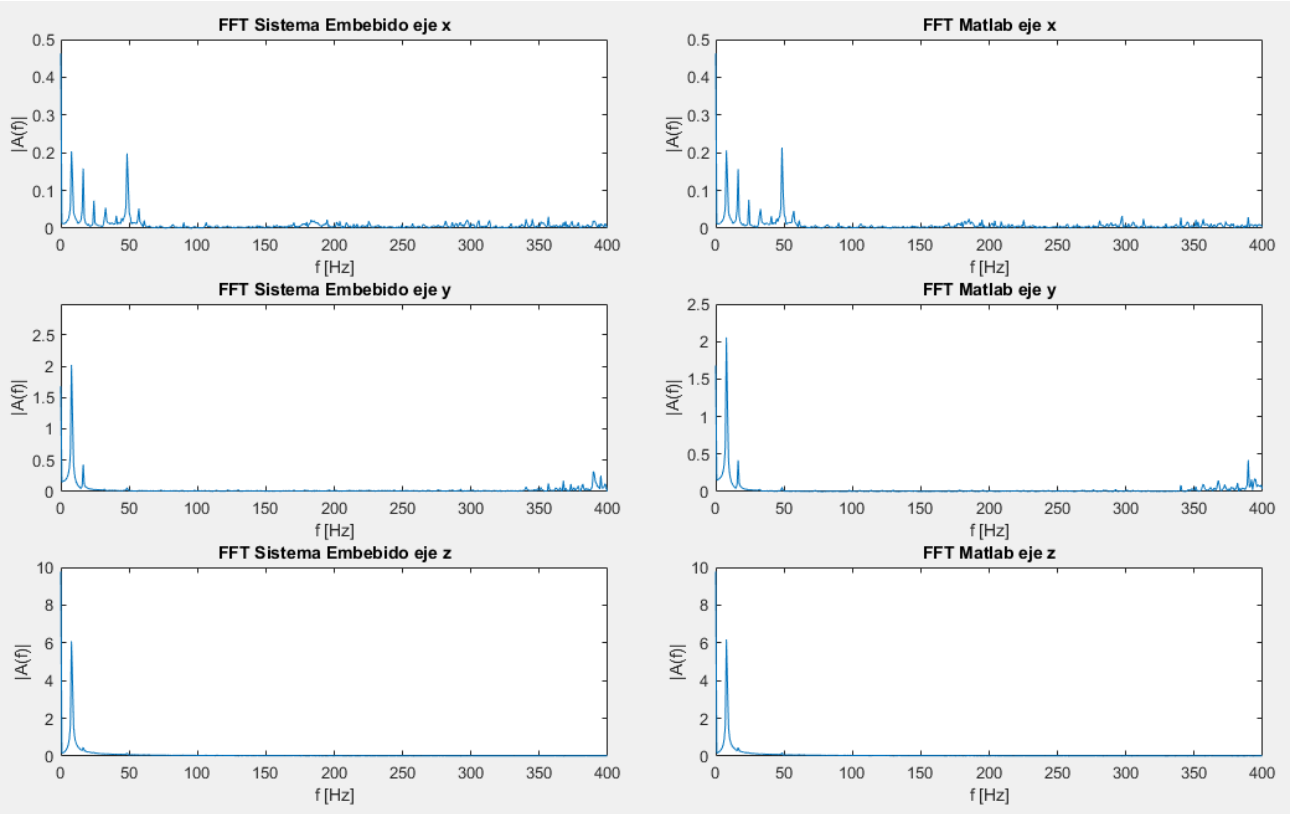


Fig. 6.16: Cálculos FFT por el Sistema embebido v/s Matlab.

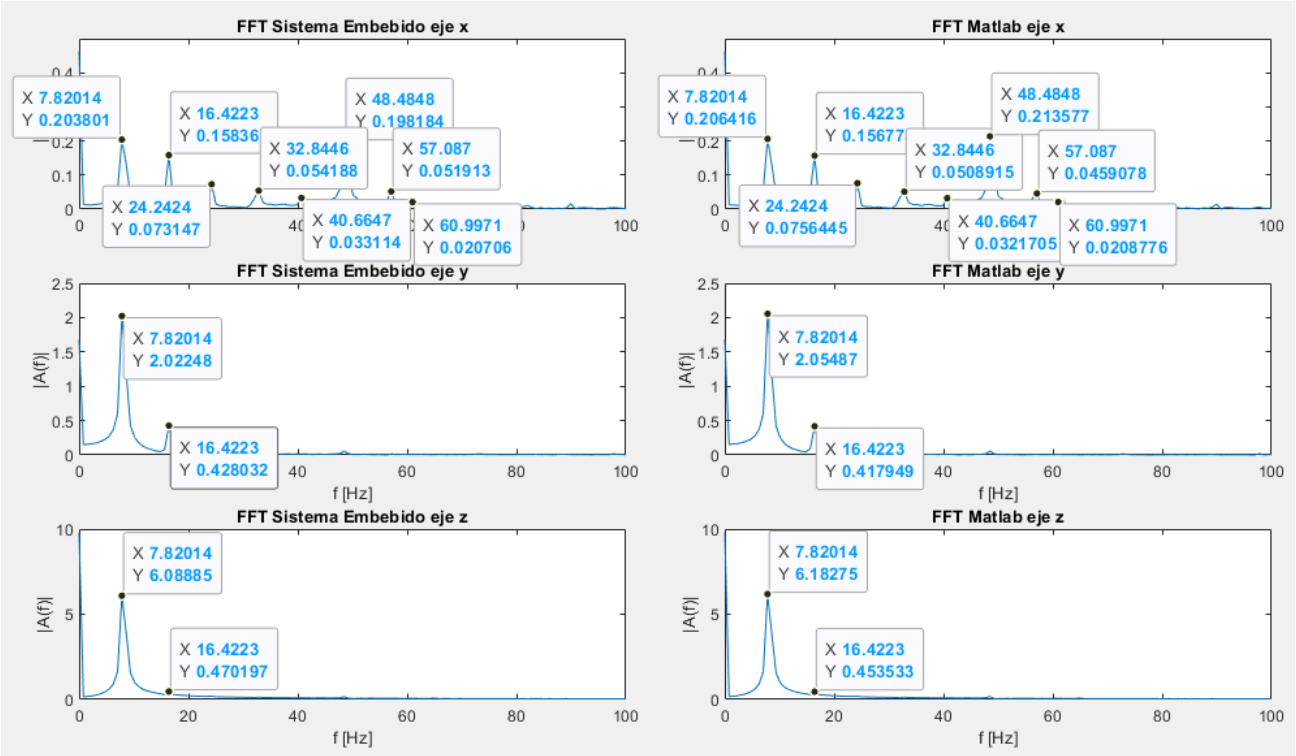


Fig. 6.17: Ubicación de armónicos Sistema embebido v/s Matlab.

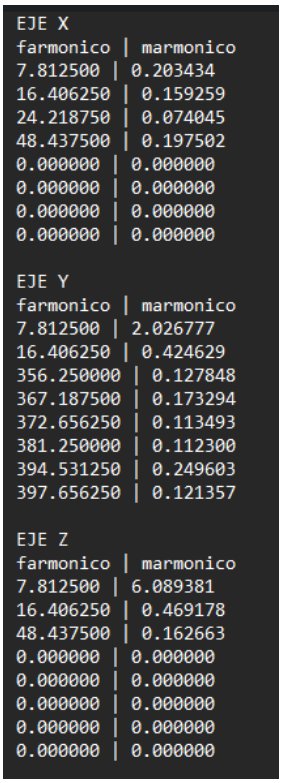


Fig. 6.18: Armónicos identificados por el algoritmo.

Para realizar una mejor conclusión de ambos resultados lo que haremos será comparar la amplitud de los picos detectados de los espectros de frecuencias, que representan los armónicos. Se consideró estudiar solo la amplitud de los picos ya que en cuanto a la ubicación de las frecuencias no hubieron diferencias entre ambos resultados y ambos sistemas detectaron los picos en las mismas frecuencias.

Para esto se tomaron 13 ventanas de muestras con el sistema funcionando sin intervenciones externas. Luego se calculó el error entre las amplitudes de estos puntos en las 13 muestras usando los resultados de Matlab como el resultado más confiable y al que se aspira llegar, posteriormente se realizó un trabajo de estadística calculando la distribución normal del error en cada eje y en cada armónico. La prueba se realizó para la frecuencia fundamental y sus siguientes 5 armónicos. En la Fig. 6.19 podemos ver la identificación de los picos de los espectros para cada espectro. En la Fig. 6.20 podemos ver la distribución normal de los errores y en donde podemos notar una clara tendencia a un error cercano a 0 para cada resultado. El promedio de error es 0.0023 unidades de amplitud.

Las simulaciones fueron realizadas en Matlab usando los Toolbox Signal Processing Toolbox y Statistics and Machine Learning Toolbox

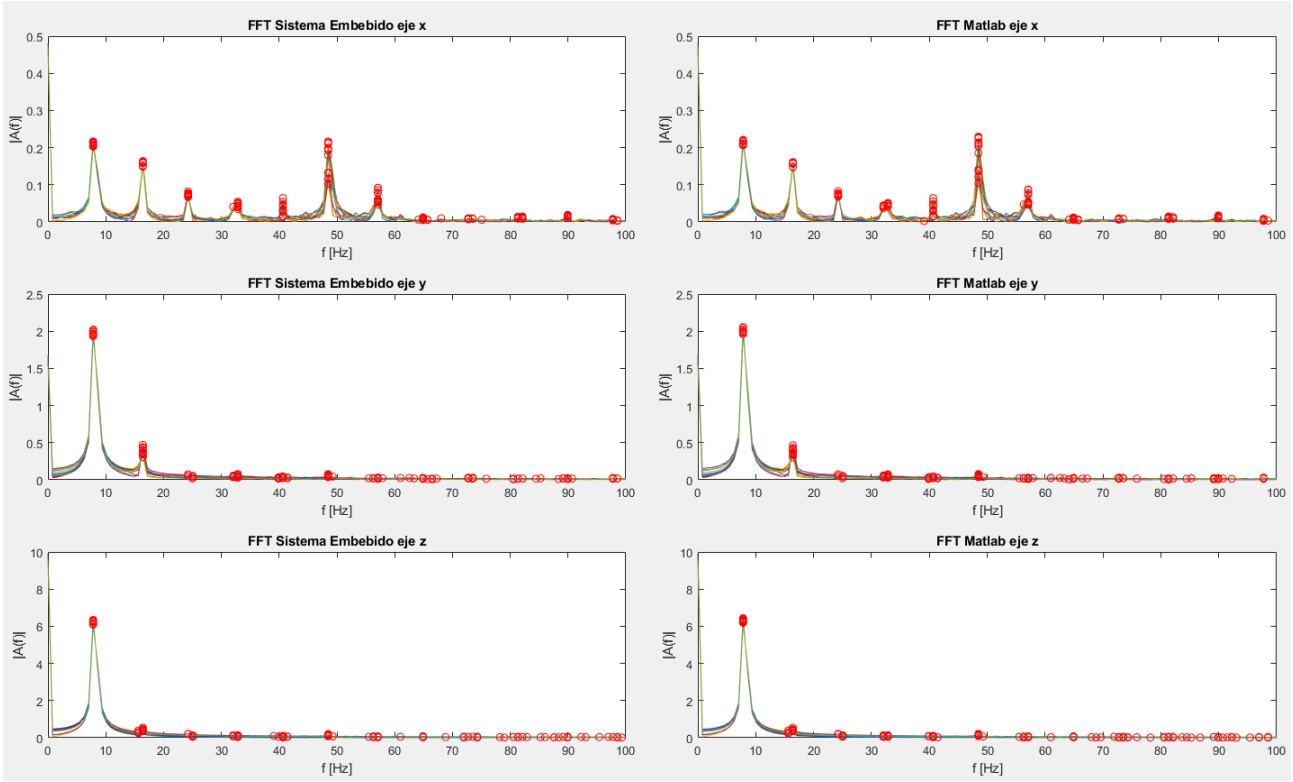


Fig. 6.19: Armónicos identificados en los resultados del Sistema Embebido y en los de Matlab.

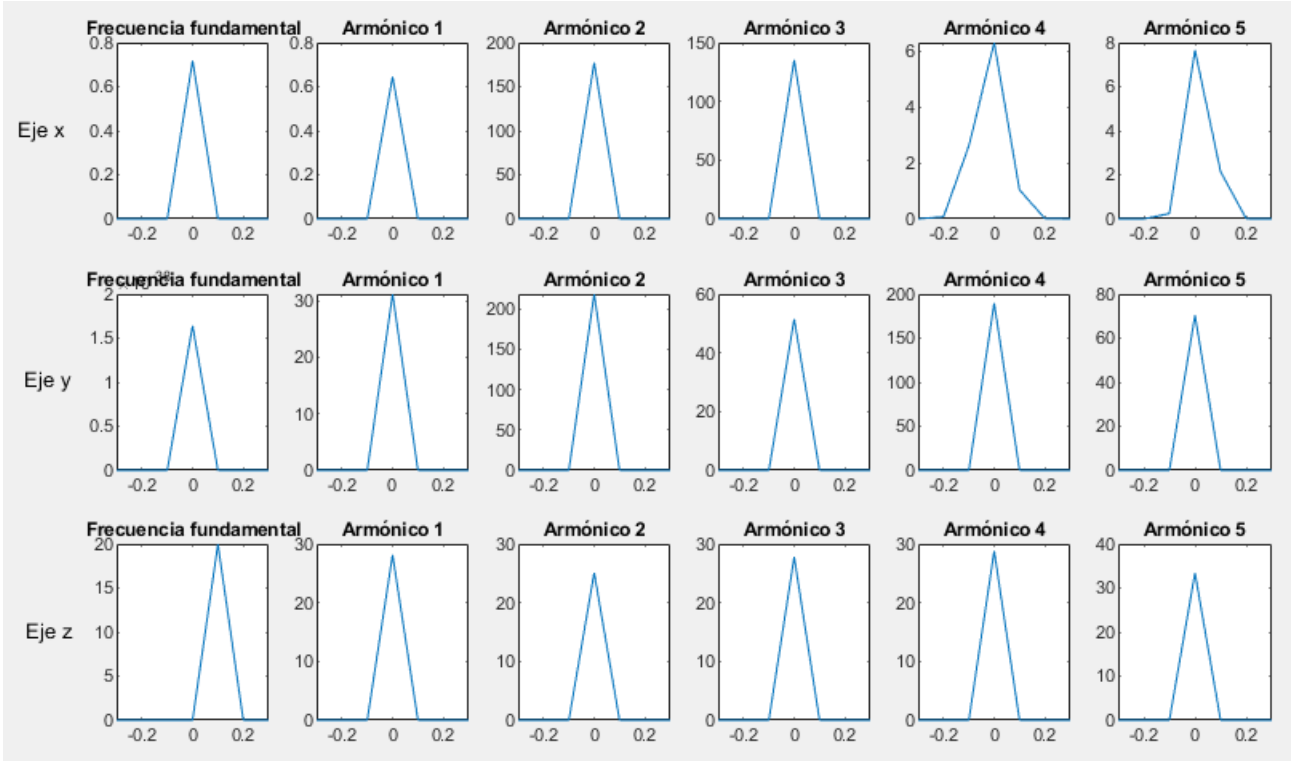


Fig. 6.20: Distribución normal del error.

Con fines estadísticos se realizó una segunda prueba con el sistema vibrando a otra frecuencia y se tomaron nuevamente 13 muestras en esta condición. En la Fig. 6.17 podemos ver que el sistema se encuentra vibrando con una frecuencia fundamental en los 7.8Hz, en cambio en la prueba de la Fig. 6.21 la frecuencia fundamental se encuentra a los 13.3 Hz.

En las Fig. 6.20 y 6.22 podemos comprobar que la tendencia del error de los cálculos del sistema embebido en comparación al software Matlab tiende a un valor cercano a 0. El promedio del error en las dos pruebas es de 1.8 %, por lo que podemos concluir que el algoritmo implementado cumple correctamente con su objetivo, realizando un análisis espectral muy cercano a uno realizado en un software complejo ejecutado en un procesador de arquitectura y capacidades de cómputo muy superiores al usado en el sistema embebido.

No se realizaron pruebas de error para el algoritmo RMS ya que este procedimiento realiza operaciones matemáticas simples a partir de una ventana de datos, por lo que no hay diferencia entre estos resultados y uno realizado en un software de capacidades superiores.

El sistema embebido consume mA en su funcionamiento, por lo que por esto y por sus características de red podemos decir que cumple las características IIoT que se discutieron a lo largo del trabajo.

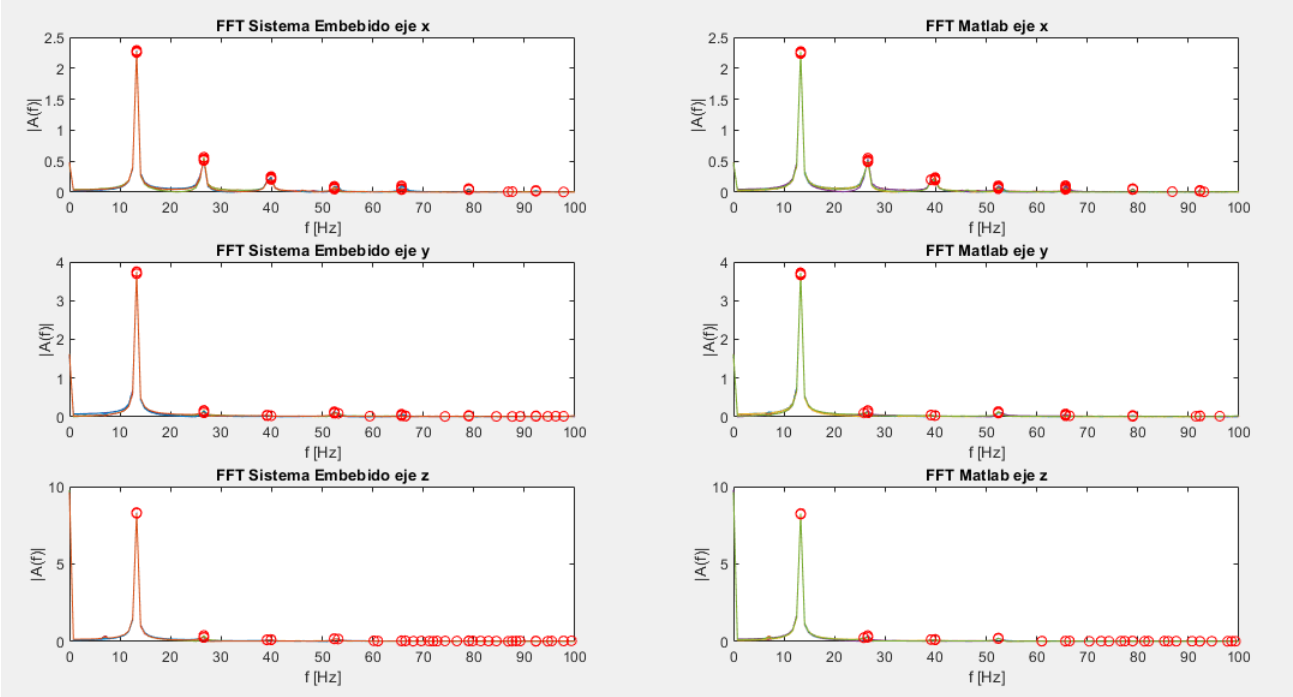


Fig. 6.21: Armónicos identificados en los resultados del Sistema Embebido y en los de Matlab. Segunda prueba.

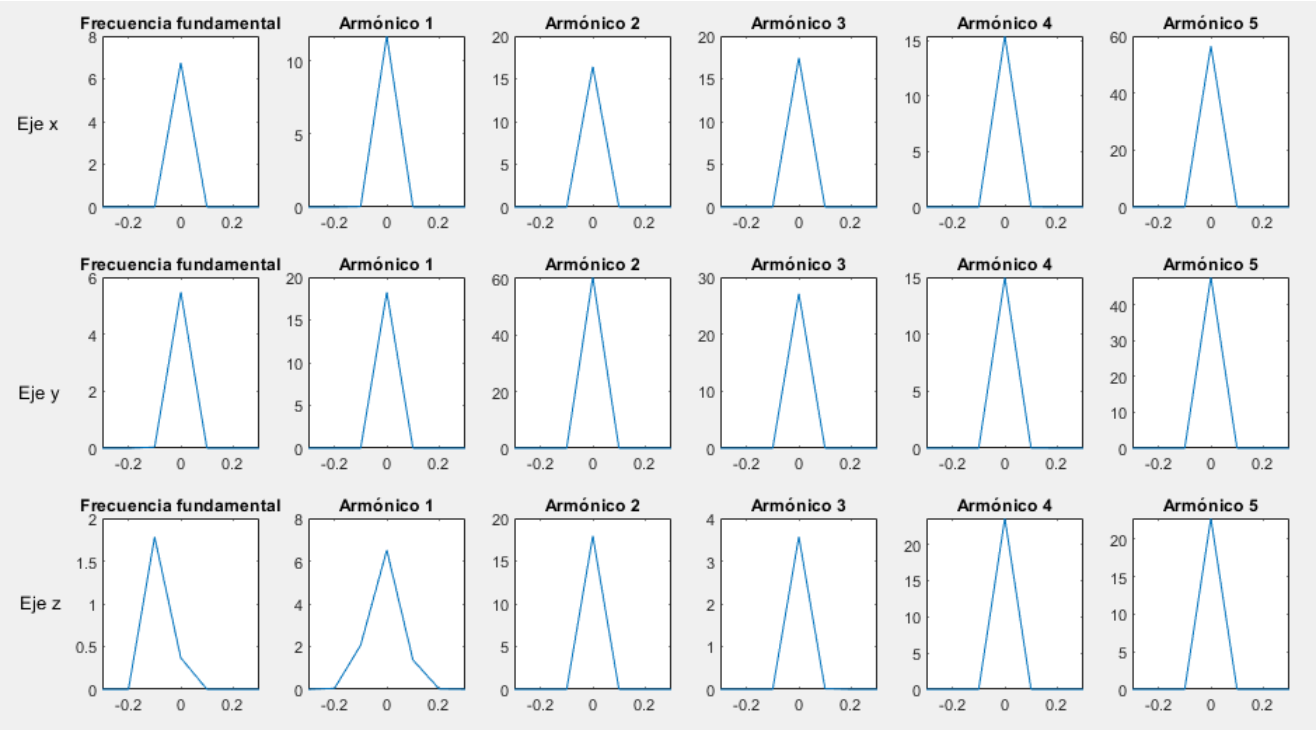


Fig. 6.22: Distribución normal del error. Segunda prueba

CAPÍTULO 7

Discusión y conclusiones

En el capítulo anterior pudimos concluir que el sistema propuesto pudo ser implementado de forma correcta cumpliendo con todas las características requeridas. El código final superó las 600 líneas, sin contar las librerías en las que se apoyaron los algoritmos. En la Fig. 7.1 podemos comprobar que el código utiliza un 31.8 % de memoria DRAM, del total disponible en el ESP32, para la definición de variables y un 65.5 % de IRAM en donde se ejecutan los algoritmos. Además podemos ver que el tamaño total del código es de 1.1 megabytes aproximadamente. El sistema embebido tiene un consumo promedio de 3.6 mA mientras se realizan lecturas de acelerómetro y se ejecutan los algoritmos de detección, y asciende a 18.4 mA al momento de hacer uso de la red WiFi para enviar un mensaje.

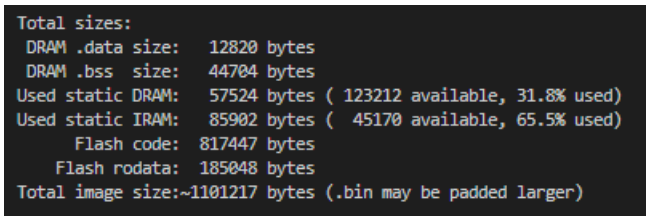


Fig. 7.1: Tamaño y uso de memoria del código implementado.

En cuanto a la implementación en escenario real de este sistema debemos destacar que depende completamente del estudio y preparación previa a la implementación, es decir, de la realización de pruebas de un sistema motor eléctrico sano y en las condiciones de operación deseadas. Luego de definir los parámetros y holguras de detección el sistema será capaz de trabajar y tomar conclusiones tal como lo vimos en el capítulo anterior.

Comentar además que este sistema no es exclusivo del hardware electrónico usado en esta ocasión, y sí se podría implementar usando otro microcontrolador y otro IMU de características similares. En un futuro cercano podríamos contar con un microprocesador más económico y con mejores capacidades de cómputo, y con un sensor inercial con mayor frecuencia de muestreo, permitiendo ejecutar los algoritmos con un mejor rendimiento. Es por esto que también debemos comentar que los algoritmos son mejorables vía código respecto a tiempos de ejecución y disminución de error con respecto a referencias más sólidas como los es el Matlab.

Bibliografía

[1] V. Govindraj, M. Sathiyanarayanan, and B. Abubakar, "Customary homes to smart homes using internet of things (iot) and mobile application", in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, 2017, pp. 1059–1063. DOI: 10.1109/SmartTechCon.2017.8358532.

[2] L. Liu, Y. Liu, L. Wang, A. Zomaya, and S. Hu, "Economical and balanced energy usage in the smart home infrastructure: A tutorial and new results", *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 4, pp. 556–570, 2015. DOI: 10.1109/TETC.2015.2484839.

[3] R. Sotomayor, C. Bottner, C. Hojas, and A. Svriz, "Informe de la situacion actual de los requerimientos de transmision de datos y la estimacion de la demanda prospectiva de consumo de datos para zonas agrícolas.", 2017, p. 69, [Online]. Available: https://www.subtel.gob.cl/wp-content/uploads/2017/10/02_InformeII_Demanda_Agro_Industrias.pdf.

[4] D. Ghosh, A. Agrawal, N. Prakash, and P. Goyal, "Smart saline level monitoring system using esp32 and mqtt-s", in *IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, 2018, pp. 1–5.

[5] A. Škraba, A. Koložvari, D. Kofjač, R. Stojanović, E. Semenkin, and V. Stanovov, "Prototype of group heart rate monitoring with esp32", in *8th Mediterranean Conference on Embedded Computing (MECO)*, 2019, pp. 1–4. DOI: 10.1109/MECO.2019.8760150..

[6] I. Allafi and T. Iqbal, "Design and implementation of a low cost web server using esp32 for real-time photovoltaic system monitoring", in *IEEE Electrical Power and Energy Conference (EPEC)*, 2017, pp. 1–5. DOI: 10.1109/EPEC.2017.8286184.

[7] S. B. Biswas and M. T. Iqbal, "Solar water pumping system control using a low cost esp32 microcontroller", in *IEEE Canadian Conference on Electrical Computer Engineering (CCECE)*, 2018, pp. 1–5. DOI: 10.1109/CCECE.2018.8447749.

[8] J. Åkerberg, M. Gidlund, and M. Björkman, "Future research challenges in wireless sensor and actuator networks targeting industrial automation", in *2011 9th IEEE International Conference on Industrial Informatics*, 2011, pp. 410–415. DOI: 10.1109/INDIN.2011.6034912.

[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018. DOI: 10.1109/TII.2018.2852491.

[10] P. O'Donovan, K. Leahy, and K. Bruton, "An industrial big data pipeline for data-driven analytics maintenance applications in large-scale smart manufacturing facilities", in *Journal of Big Data 2*, vol. 25, 2015. DOI: https://doi.org/10.1186/s40537-015-0034-z.

[11] D. Ghosh, A. Agrawal, N. Prakash, and P. Goyal, "Smart saline level monitoring system using esp32 and mqtt-s", in *IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, 2018, pp. 1–5. DOI: 10.1109/HealthCom.2018.8531172.

[12] A. Zare and M. T. Iqbal, "Low-cost esp32, raspberry pi, node-red, and mqtt protocol based scada system", in *IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2000, pp. 1–5. DOI: 10.1109/IEMTRONICS51293.2020.9216412.

[13] S. R. Misal, S. R. Prajwal, H. M. Niveditha, H. M. Vinayaka, and S. Veena, "Indoor positioning system (ips) using esp32, mqtt and bluetooth", in *Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 79-82. DOI: 10.1109/ICCMC48092.2020.ICCMC-00015.

[14] M. Iorgulescu and R. Beloiu, "Study of dc motor diagnosis based on the vibration spectrum and current analysis", in *2012 International Conference on Applied and Theoretical Electricity (ICATE)*, 2012, pp. 1–4. DOI: 10.1109/ICATE.2012.6403430.

[15] D. Meng, "Vibration motor fault diagnosis system based on labview", in *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, 2020, pp. 303–306. DOI: 10.1109/ICMCCE51767.2020.00074.

[16] M. Tsypkin, "Vibration of induction motors operating with variable frequency drives — a practical experience", in *2014 IEEE 28th Convention of Electrical Electronics Engineers in Israel (IEEEI)*, 2014, pp. 1–5. DOI: 10.1109/EEEI.2014.7005894.

[17] M. Tsypkin, "Induction motor condition monitoring: Vibration analysis technique — diagnosis of electromagnetic anomalies", in *2017 IEEE AUTOTESTCON*, 2017, pp. 1–7. DOI: 10.1109/AUTEST.2017.8080483.

[18] M. Tsypkin, "Induction motor condition monitoring: Vibration analysis technique - a practical implementation", in *2011 IEEE International Electric Machines Drives Conference (IEMDC)*, 2011, pp. 406–411. DOI: 10.1109/IEMDC.2011.5994629.

[19] H.-C. Chen and H.-Y. Pu, "Fault analysis of induction motor based on discrete fractional fourier transform", in *2016 International Symposium on Computer, Consumer and Control (IS3C)*, 2016, pp. 69–72. DOI: 10.1109/IS3C.2016.28.

[20] H. N. Phyu, N. L. H. Aung, Y. Q. Yu, and Q. Jiang, "Analysis of magnet field degradation on pm motor vibration", in *Magnetics Symposium 2014 - Celebrating 50th Anniversary of IEEE Magnetics Society (MSSC50)*, 2014, pp. 1–2. DOI: 10.1109/MSSC.2014.6947947.

[21] D. P. N. Saavedra, "TUTORIAL SEVERIDAD VIBRATORIA. PARTE I" Laboratorio de vibraciones mecánicas, Departamento de ingeniería mecánica, Facultad de Ingeniería, Universidad de Concepción. [Online]. Available: http://vu2004.admin.hosting8.ing.udec.cl/upload/paginas/archivos/20-10-2017-15-22-35_26-04-2017-15-43-21_tutorial-severidad-vibratoria-parte-i.pdf.

[22] B. L. O. Vera, Tesis para optar al título de Ingeniero Naval: "Medición y análisis de vibraciones en el sistema propulsivo naval.", Universidad Austral de Chile. [Online]. Available: http://cybertesis.uach.cl/tesis/uach/2010/bmfcio.39m/doc/bmfcio.39m.pdf.

[23] H.-C. Chen and H.-Y. Pu, "Fault analysis of induction motor based on discrete fractional fourier transform", in *2016 International Symposium on Computer, Consumer and Control (IS3C)*, 2016, pp. 69–72. DOI: 10.1109/IS3C.2016.28.

[24] M. Iorgulescu and R. Beloiu, "Study of dc motor diagnosis based on the vibration spectrum and current analysis", in *2012 International Conference on Applied and Theoretical Electricity (ICATE)*, 2012, pp. 1–4. DOI: 10.1109/ICATE.2012.6403430.

[25] M. A. Ugwiri, M. Carratù, A. Pietrosanto, V. Paciello, and A. Lay-Ekuakille, "Vibrations measurement and current signatures for fault detection in asynchronous motor", in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6. DOI: 10.1109/I2MTC43012.2020.9128433.

[26] A. Musthofa, D. A. Asfani, I. M. Y. Negara, D. Fahmi, and N. Priatama, "Vibration analysis for the classification of damage motor pt petrokimia gresik using fast fourier transform and neural network", in *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 2016, pp. 381–386. DOI: 10.1109/ISITIA.2016.7828690.

[27] G. Betta, C. Liguori, A. Paolillo, and A. Pietrosanto, "A dsp-based fft-analyzer for the fault diagnosis of rotating machine based on vibration analysis", in *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, vol. 1, 2001, 572–577 vol.1. DOI: 10.1109/IMTC.2001.928883.

[28] W. G. González, "Análisis causa raíz mediante vibraciones a compresor aerzen de tornillo", in *Tesis para obtener el título de: Ingeniero mecánico eléctrico.*, 2021.

[29] S. Seker and A. H. Kayran, "Neural network application for fault detection in electric motors", in *2009 Australasian Universities Power Engineering Conference*, 2009, pp. 1–4.

[30] N. Ahmad, R. A. Raja Ghazilla, N. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (imu) sensor applications", in *International Journal of Signal Processing Systems*, vol. 1, 2013, pp. 256–262. DOI: 10.12720/ijsps.1.2.256-262.

[31] I. Narrett and B. Strabel, "Magnetic field measurement suite for cubesat applications", in *Earth and Space Science Open Archive*, 2020. DOI: 10.1002/essoar.10505522.1.

[32] E. Systems, ESP32-D0WD Datasheet [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[33] C. G. C. Carducci, A. Monti, M. H. Schraven, M. Schumacher, and D. Mueller, "Enabling esp32-based iot applications in building automation systems," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0IoT)*, 2019, pp. 306–311. DOI: 10.1109/METROI4.2019.8792852.

[34] V. M. Ionescu and F. M. Enescu, "Investigating the performance of micropython and c on esp32 and stm32 microcontrollers", in *IEEE 26th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2020, pp. 234–237. DOI: 10.1109/SIITME50350.2020.9292199.

[35] O. Barybin, E. Zaitseva, and V. Brazhnyi, "Testing the security esp32 internet of things devices", in *IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC ST)*, 2019, pp. 143–146. DOI: 10.1109/PICST47496.2019.9061269.

[36] E. Systems, ESP32-WROOM-32D Datasheet [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf.

[37] Bosch, BMI270 Datasheet [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi270-ds000.pdf.

[38] A. Foster, "Messaging technologies for the industrial internet and the internet of things whitepaper", in *PrismTech*, 2015.

[39] N. Q. Uy and V. H. Nam, "A comparison of amqp and mqtt protocols for internet of things", in *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*, 2019, pp. 292–297. DOI: 10.1109/NICS48868.2019.9023812.

[40] E. Condes, "Arduinofft: Fast fourier transform for arduino", [Online]. Available: https://github.com/kosme/arduinoFFT.

[41] H. Shi, L. Niu, and J. Sun, "Construction of industrial internet of things based on mqtt and opc ua protocols", in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2020, pp. 1263–1267. DOI: 10.1109/ICAICA50127.2020.9182598.

[42] T. Mizuya, M. Okuda, and T. Nagao, "A case study of data acquisition from field devices using opc ua and mqtt", in *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 2017, pp. 611–614. DOI: 10.23919/SICE.2017.8105594.

[43] M. Silveira Rocha, G. Serpa Sestito, A. Luis Dias, A. Celso Turcato, and D. Brandão, "Performance comparison between opc ua and mqtt for data exchange", in *2018 Workshop on Metrology for Industry 4.0 and IoT*, 2018, pp. 175–179. DOI: 10.1109/METROI4.2018.8428342.

[44] T. Yokotani and Y. Sasaki, "Comparison with http and mqtt on required network resources for iot", in *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, 2016, pp. 1–6. DOI: 10.1109/ICCEREC.2016.7814989.

[45] T. Stefanec and M. Kusek, "Comparing energy consumption of application layer protocols on iot devices", in *2021 16th International Conference on Telecommunications (ConTEL)*, 2021, pp. 23–28. DOI: 10.23919/ConTEL52528.2021.9495993.

[46] C. B. Gemirter, Ç. Şenturca, and Ş. Baydere, "A comparative evaluation of amqp, mqtt and http protocols using real-time public smart city data", in *2021 6th International Conference on Computer Science and Engineering (UBMK)*, 2021, pp. 542–547. DOI: 10.1109/UBMK52708.2021.9559032.

[47] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http", in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7. DOI: 10.1109/SysEng.2017.8088251.

[48] Espressif, "Esp-mdf framework", [Online]. Available: https://github.com/espressif/esp-idf.

[49] Espressif, "Esp-mdf framework", [Online]. Available: https://github.com/espressif/esp-mdf.

[50] Microchip, "Mcp73831 datasheet", [Online]. Available: https://www.mouser.cl/datasheet/2/268/20001984g-846362.pdf.

[51] T. Instruments, "Tlv75533 datasheet", [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv755p.pdf.

[52] Espressif, "Esp32 hardware design guidelines", [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_hardware_design_guidelines_en.pdf.

[53] Bosch, "Bmi260 family inertial measurement units handling, soldering mounting instructions", [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/handling_soldering_mounting_instructions/bst-mis-hs001.pdf.

[54] E. Bogatin, in *Bogatin's practical guide to prototype breadboard and PCB design*, Norwood, MA, USA: Artech House, 2021, ch. 6.

[55] R. Scheibler, "Esp32-fft: Vanilla single-precision radix-2 fft for the esp32.", [Online]. Available: https://github.com/fakufaku/esp32-fft.

[56] M. Borgerding, "Kissfft: A fast fourier transform (fft) library that tries to keep it simple, stupid.", [Online]. Available: https://github.com/mborgerding/kissfft.

[57] A. Mukhin, "Mifft: A small and fast discrete fourier transform library.", [Online]. Available: https://github.com/aimukhin/minfft.

Anexos

A. Sistema Embebido

A.1 Circuito completo

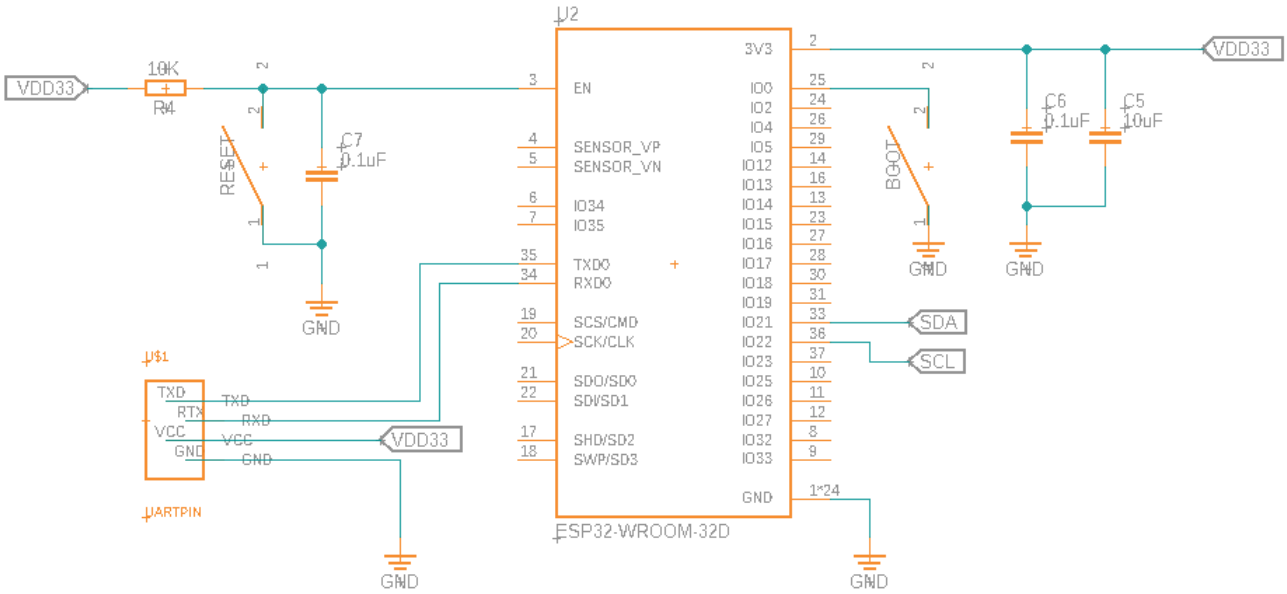


Figura 7.2: Circuito esquemático ESP32.

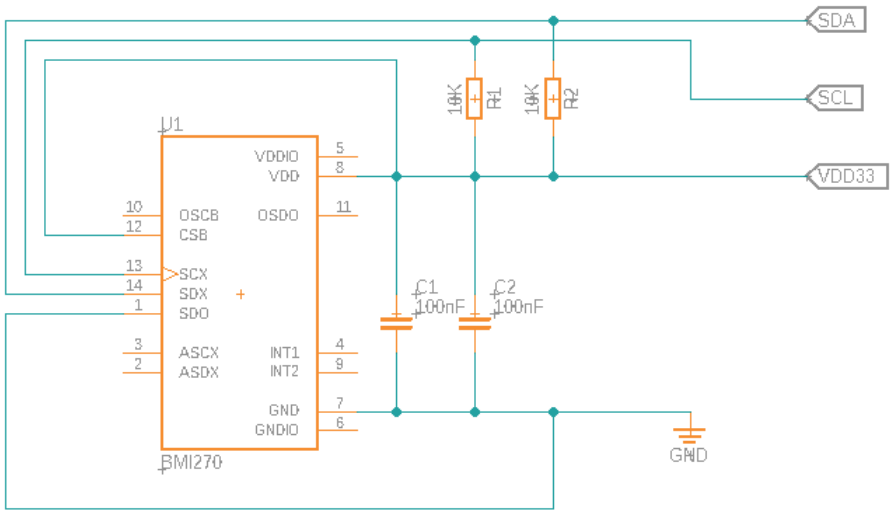


Figura 7.3: Circuito esquemático BMI270.

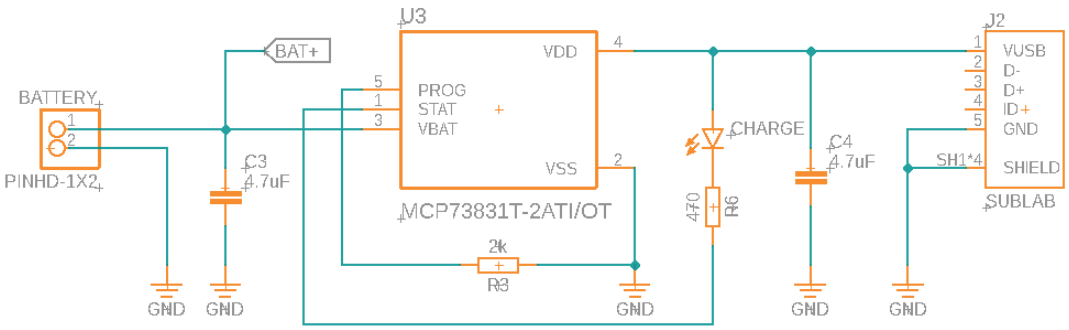


Figura 7.4: Circuito esquemático MCP73831.

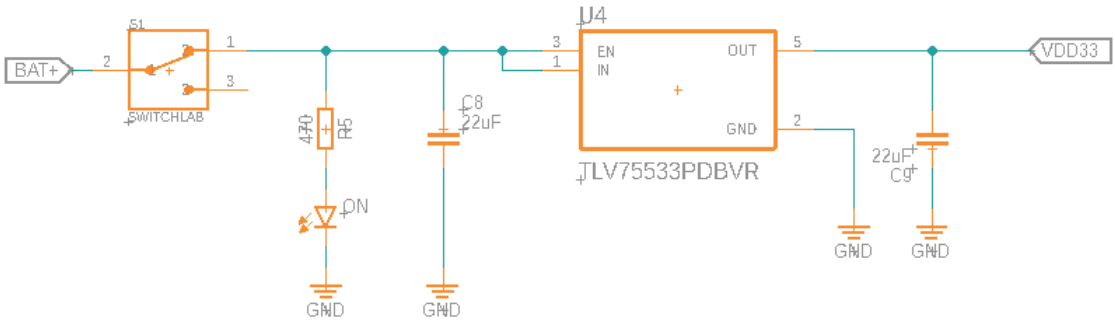


Figura 7.5: Circuito esquemático TLV75533.

A.2 Placas de desarrollo

En un principio se hicieron uso de placas de desarrollo con el fin de realizar las primeras pruebas y de asegurarnos que el sistema propuesto en un principio se puede construir. Para el ESP32 se hizo uso de una placa de desarrollo producida por Espressif llamada ESP32-DevKitC V4, en el cual la mayoría de los pines de entrada y salida del chip se encuentran disponibles para la conexión de periféricos de forma sencilla. La placa incluye el chip ESP32-WROOM-32D, además de un puente USB-UART, Micro USB y un módulo de alimentación.

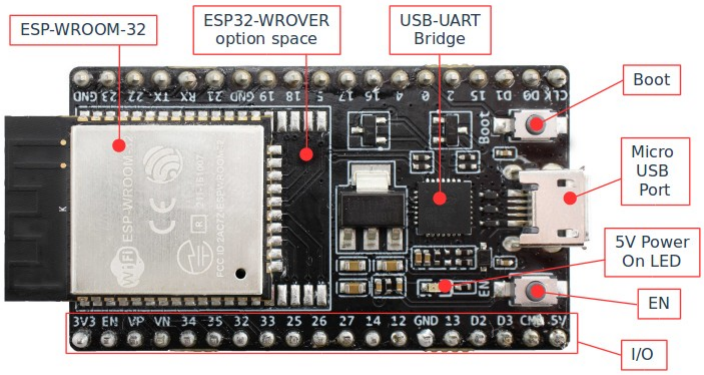


Figura 7.6: ESP32-DevKitC V4 con el módulo ESP-WROOM-32D incluido.

Para el sensor BMI270 se contó con una placa de desarrollo que se corresponde con el circuito facilitado por el fabricante para la conexión I^2C. Esta placa permitió realizar una conexión de forma sencilla con el microcontrolador.

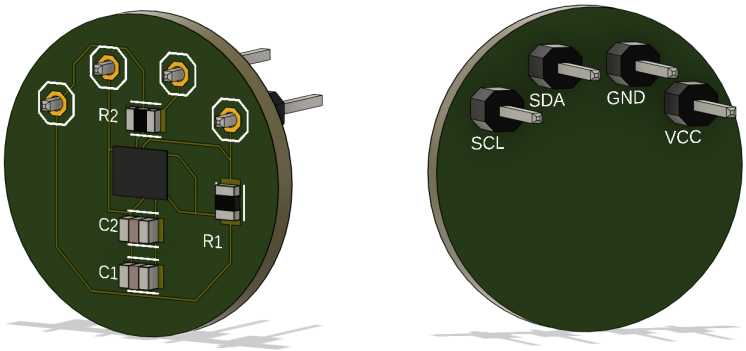


Figura 7.7: Placa de desarrollo BMI270 preparada para comunicación I^2C.

```c
#define I2C_MASTER_SCL_IO               GPIO_NUM_22
#define I2C_MASTER_SDA_IO               GPIO_NUM_21
#define I2C_MASTER_FREQ_HZ                  400000
#define ESP_SLAVE_ADDR                  0x68
#define WRITE_BIT               0x0
#define READ_BIT                0x1
#define ACK_CHECK_EN            0x0
#define ACK_VAL                 0x0
#define NACK_VAL                0x1

esp_err_t bmi_init(void)
{
    int i2c_master_port = I2C_NUM_0;
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = I2C_MASTER_SDA_IO;
    conf.sda_pullup_en = GPIO_PULLUP_DISABLE;
    conf.scl_io_num = I2C_MASTER_SCL_IO;
    conf.scl_pullup_en = GPIO_PULLUP_DISABLE;
    conf.master.clk_speed = I2C_MASTER_FREQ_HZ;
    conf.clk_flags = I2C_SCLK_SRC_FLAG_FOR_NOMAL; // 0
    i2c_param_config(i2c_master_port, &conf);
    return i2c_driver_install(i2c_master_port, conf.mode, 0, 0, 0);
}
```

Figura 7.8: Código configuración comunicación I^2C.

A.3 ESP-IDF

Para compilar los programas escritos en C hacemos uso de ESP-IDF con los siguientes comandos:

```
idf.py set-target esp32

idf.py menuconfig

idf.py build

idf.py -p (PORT) flash monitor
```

Al ejecutar estos comandos lo que estamos haciendo es primero configurar el compilador para trabajar con un ESP32, luego se debe crear o editar la configuración relacionada al ESP del programa ingresando el comando `menuconfig`, luego se construye y compila el programa con `build` y finalmente se escribe en el microcontrolador. El microcontrolador se conecta al computador mediante puerto USB y este le asigna un puerto COM, en este caso el sistema otorga el puerto COM5 al microcontrolador por lo tanto el último comando queda como: `idf.py -p COM5 flash monitor`. Este último comando escribe nuestro programa en el microcontrolador y además nos ejecuta un monitor serial en el puerto designado, por lo que al momento de cargar el programa podremos recibir mensajes desde el ESP32 a través de este monitor, muy útil para desarrollar técnicas de debugging de programas.

A.4 Algoritmos

Para el funcionamiento del sensor debemos seguir una serie de algoritmos que son facilitados por el fabricante, esto nos ofrece libertades con respecto a la configuración del dispositivo como elegir un modo de ahorro de energía o de alto rendimiento, o si se aplican filtros a la entrada o no, entre otros. Esto se realiza leyendo y escribiendo registros del sensor.

El sistema se implementó en un comienzo haciendo uso de las placas de desarrollo de acuerdo a la Figura 7.9.

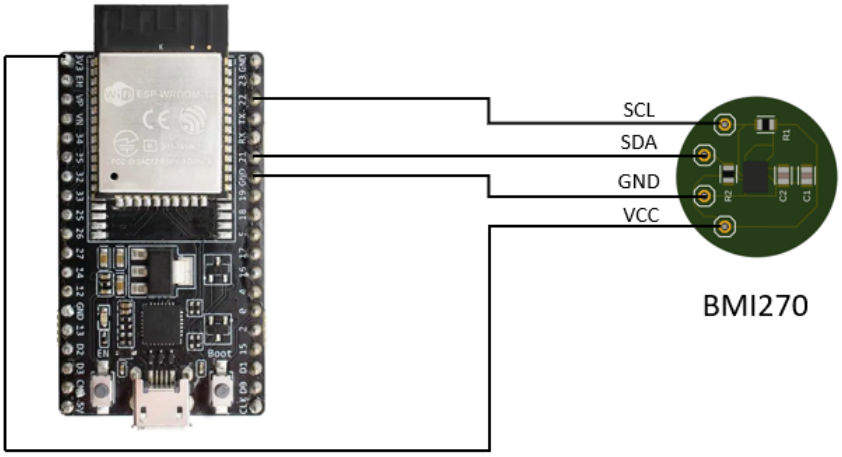


Figura 7.9: Diagrama de la conexión del ESP32 y el sensor BMI270.

A.4.1 Acceso a escritura BMI270

El algoritmo para escribir en un registro queda descrito en la Figura 7.10 y comienza con una señal START generado por el Maestro, seguido de 7 bits que corresponden a la dirección del esclavo a escribir y un bit de escritura (RW=0). El esclavo envía un bit ACKS = 0 y libera el bus de datos. Luego el Maestro envía un byte de la dirección del registro a escribir. El Esclavo nuevamente envía un ACKS=0 y espera los 8 bits de data a escribir. Luego el Esclavo envía un ACKS=0 al recibir la data, el Maestro envía un STOP y finaliza la comunicación.

Figura 7.10: Escritura de datos en IBM270.

A.4.2 Acceso a lectura BMI270

El algoritmo para leer los registros está compuesto por una primera parte de escrituta y luego una de lectura. Primero se escribe (RW=0) la dirección del registro a leer, y luego comienza la parte de lectura con otro START, posterior a esto se pide leer (RW=1) los registros requeridos. Para finalizar la lectura después del último registro se debe enviar un NACK=1 desde el maestro y luego se envía un STOP para finalizar la comunicación.

Figura 7.11: Lectura de registros del BMI270.

```c
esp_err_t bmi_read(i2c_port_t i2c_num, uint8_t *data_addres, uint8_t *data_rd , size_t size){
    if (size == 0) {
        return ESP_OK;
    }
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | WRITE_BIT, ACK_CHECK_EN);
    i2c_master_write(cmd, data_addres, size, ACK_CHECK_EN);
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | READ_BIT, ACK_CHECK_EN);
    if (size > 1) {
        i2c_master_read(cmd, data_rd, size - 1, ACK_VAL);
    }
    i2c_master_read_byte(cmd, data_rd + size - 1, NACK_VAL);
    i2c_master_stop(cmd);
    ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);
    return ret;
}

esp_err_t bmi_write(i2c_port_t i2c_num,uint8_t *data_addres, uint8_t *data_wr ,size_t size)
{
    uint8_t size1 = 1;
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | WRITE_BIT, ACK_CHECK_EN);
    i2c_master_write(cmd, data_addres, size1, ACK_CHECK_EN);
    i2c_master_write(cmd, data_wr, size, ACK_CHECK_EN);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);
    return ret;
}
```

Figura 7.12: Código para lectura y escritura de registros BMI270.

A.4.3 Chip ID

Este algoritmo es la confirmación en que los algoritmos de escrituras y escrituras y la conexión física con el chip es correcta y se logra establecer una comunicación I^2C entre el sensor y el microprocesador. Esto sucede ya que el Registro 0x00 del sensor contiene un dato que representa la identidad del mismo, en nuestro caso el datasheet nos muestra que este dato corresponde a un 0x24. Se escribe el algoritmo y efectivamente al leer el Registro 0x00 el sensor nos entrega un 0x24, con lo que podemos afirmar que el sensor se encuentra en buen estado y nuestra comunicación y nuestro programa funciona.

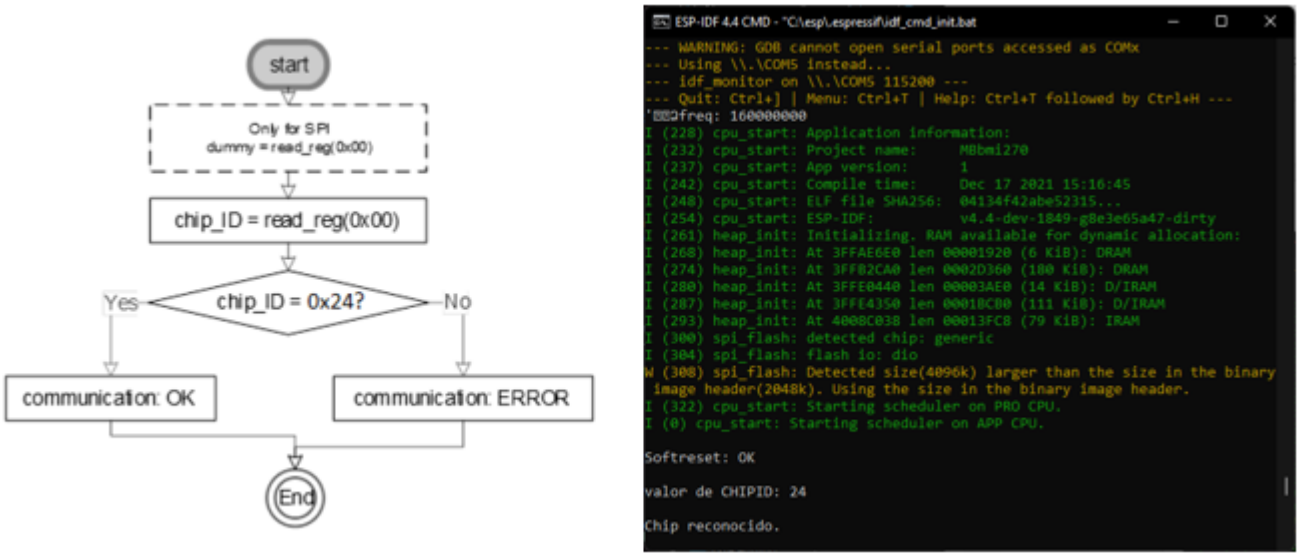


Figura 7.13: Algoritmo Chip ID

A.4.4 Inicialización

Luego de que estamos seguros que podemos escribir y leer registros del sensor procedemos a cargar un vector de datos de configuración dispuesto por el fabricante llamado *bmi270_config_file*, que inicializa el sensor y lo deja habilitado para su uso. Sin cargar este archivo el chip no permite hacer lecturas del sensor y se debe cargar cada vez que se encienda. Destacar que el *config_file* tiene un tamaño de 8 kB por lo que este paso termina justificando casi todo el tiempo de encendido del sistema.

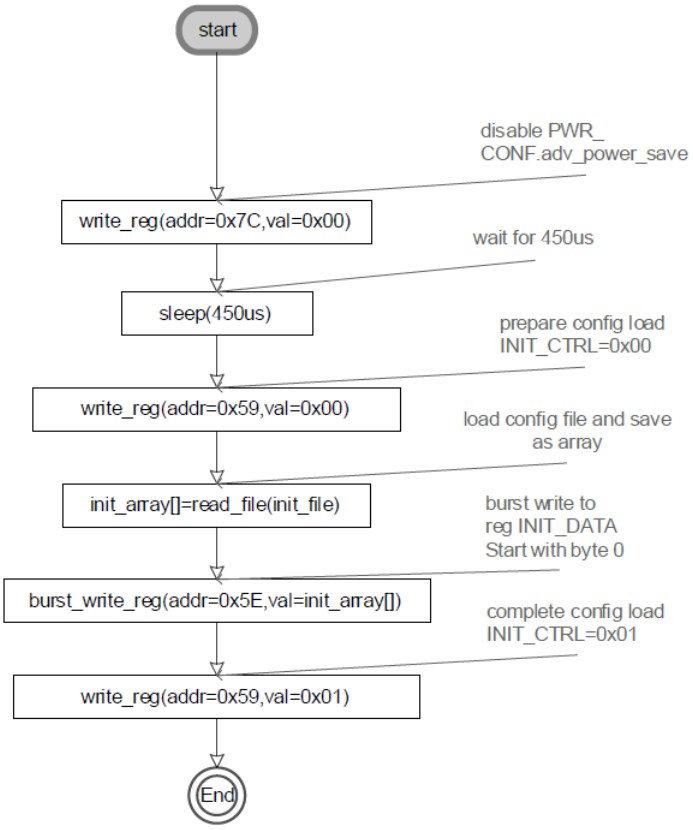


Figura 7.14: Algoritmo de inicialización BMI270.

Para comprobar que el dispositivo se encuentra inicializado y listo para operar se hace una lectura al registro 0x21 que nos indica con un bit si la inicialización se realizó con éxito o si ocurrió un error. En caso de que el sensor no se inicialice no se puede realizar la lectura de los datos.

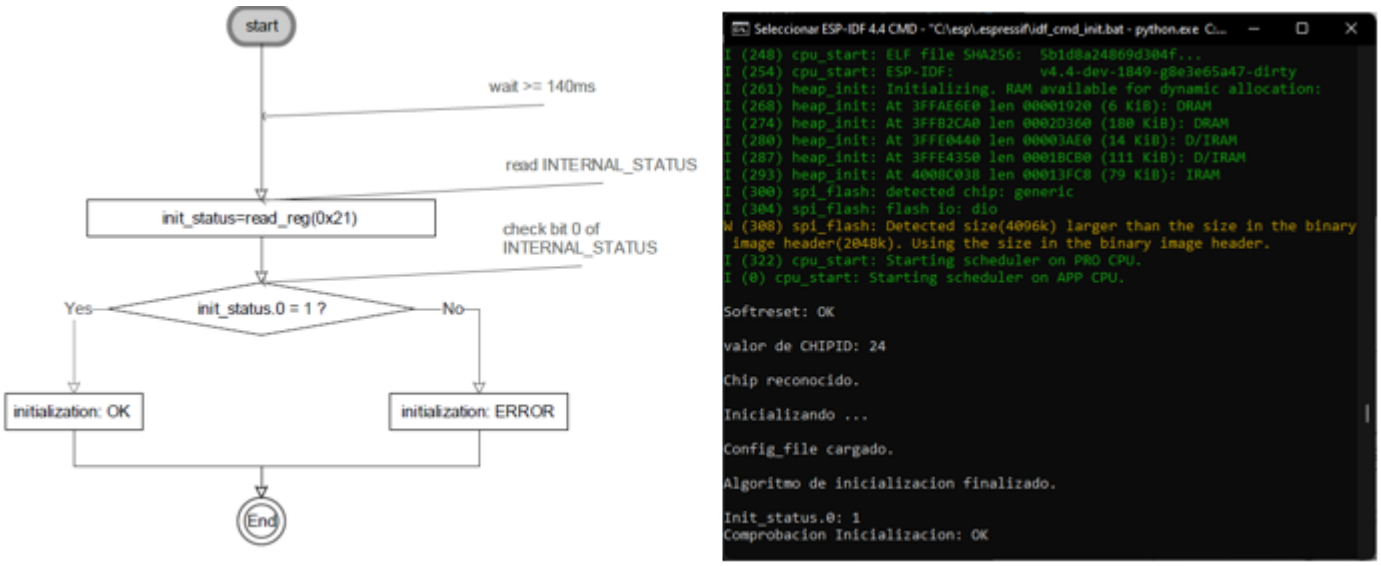


Figura 7.15: Algoritmo de comprobación de inicialización.

B. Overview Datasheets

B.1 ESP32-WROOM-32D

1 Overview

ESP32-WROOM-32D and ESP32-WROOM-32U are powerful, generic Wi-Fi+BT+BLE MCU modules that target a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding.

ESP32-WROOM-32U is different from ESP32-WROOM-32D in that ESP32-WROOM-32U integrates a U.FL connector. For detailed information of the U.FL connector please see Chapter 10. Note that the information in this data sheet is applicable to both modules. Any differences between them will be clearly specified in the course of this document. Table 1 lists the difference between ESP32-WROOM-32D and ESP32-WROOM-32U.

Table 1: ESP32-WROOM-32D vs. ESP32-WROOM-32U

Module	ESP32-WROOM-32D	ESP32-WROOM-32U
Core	ESP32-D0WD	ESP32-D0WD
SPI flash	32 Mbits, 3.3 V	32 Mbits, 3.3 V
Crystal	40 MHz	40 MHz
Antenna	onboard antenna	U.FL connector (which needs to be connected to an external IPEX antenna)
Dimensions (Unit: mm)	(18.00±0.10) × (25.50±0.10) × (3.10±0.10) (See Figure 6 for details)	(18.00±0.10) × (19.20±0.10) × (3.20±0.10) (See Figure 7 for details)
Schematics	See Figure 3 for details.	See Figure 4 for details.

At the core of the two modules is the ESP32-D0WD chip that belongs to the ESP32 series* of chips. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz. The chip also has a low-power co-processor that can be used instead of the CPU to save power while performing tasks that do not require much computing power, such as monitoring of peripherals. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I²S and I²C.

Note:
* For details on the part numbers of the ESP32 family of chips, please refer to the document *ESP32 Datasheet*.

The integration of Bluetooth®, Bluetooth LE and Wi-Fi ensures that a wide range of applications can be targeted, and that the module is all-around: using Wi-Fi allows a large physical range and direct connection to the Internet through a Wi-Fi router, while using Bluetooth allows the user to conveniently connect to the phone or broadcast low energy beacons for its detection. The sleep current of the ESP32 chip is less than 5 μA, making it suitable for battery powered and wearable electronics applications. The module supports a data rate of up to 150 Mbps, and 20 dBm output power at the antenna to ensure the widest physical range. As such the module does offer industry-leading specifications and the best performance for electronic integration, range, power consumption, and connectivity.

The operating system chosen for ESP32 is freeRTOS with LwIP; TLS 1.2 with hardware acceleration is built in as well. Secure (encrypted) over the air (OTA) upgrade is also supported, so that users can upgrade their products even after their release, at minimum cost and effort.

Table 2 provides the specifications of ESP32-WROOM-32D and ESP32-WROOM-32U.

Table 2: ESP32-WROOM-32D and ESP32-WROOM-32U Specifications

Categories	Items	Specifications
Certification	RF Certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi Certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green Certification	REACH/RoHS
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®, compatible with ISO11898-1)
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash [1]	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range [2]	–40 °C ~ +85 °C
	Moisture sensitivity level (MSL)	Level 3

Notice:

1. ESP32-WROOM-32D and ESP32-WROOM-32U with 8 MB flash or 16 MB flash are available for custom order.
2. ESP32-WROOM-32D and ESP32-WROOM-32U with high temperature range (–40 °C ~ +105 °C) option are available for custom order. 4 MB SPI flash is supported on the high temperature range version.
3. For detailed ordering information, please see *Espressif Product Ordering Information*.

5 Electrical Characteristics

5.1 Absolute Maximum Ratings

Stresses beyond the absolute maximum ratings listed in Table 5 below may cause permanent damage to the device. These are stress ratings only, and do not refer to the functional operation of the device that should follow the recommended operating conditions.

Table 5: Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Unit
VDD33	Power supply voltage	–0.3	3.6	V
I_{output} [1]	Cumulative IO output current	-	1,100	mA
T_{store}	Storage temperature	–40	150	°C

1. The module worked properly after a 24-hour test in ambient temperature at 25 °C, and the IOs in three domains (VDD3P3_RTC, VDD3P3_CPU, VDD_SDIO) output high logic level to ground. Please note that pins occupied by flash and/or PSRAM in the VDD_SDIO power domain were excluded from the test.
2. Please see Appendix IO_MUX of *ESP32 Datasheet* for IO's power domain.

5.2 Recommended Operating Conditions

Table 6: Recommended Operating Conditions

Symbol	Parameter	Min	Typical	Max	Unit
VDD33	Power supply voltage	3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T	Operating temperature	–40	-	85	°C

5.3 DC Characteristics (3.3 V, 25 °C)

Table 7: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter		Min	Typ	Max	Unit
C_{IN}	Pin capacitance		-	2	-	pF
V_{IH}	High-level input voltage		0.75×VDD[1]	-	VDD[1]+0.3	V
V_{IL}	Low-level input voltage		–0.3	-	0.25×VDD[1]	V
I_{IH}	High-level input current		-	-	50	nA
I_{IL}	Low-level input current		-	-	50	nA
V_{OH}	High-level output voltage		0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage		-	-	0.1×VDD[1]	V
I_{OH}	High-level source current (VDD[1] = 3.3 V, V_{OH} >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain [1, 2]	-	40	-	mA
		VDD3P3_RTC power domain [1, 2]	-	40	-	mA
		VDD_SDIO power domain [1, 3]	-	20	-	mA

Symbol	Parameter	Min	Typ	Max	Unit
I_{OL}	Low-level sink current (VDD[1] = 3.3 V, V_{OL} = 0.495 V, output drive strength set to the maximum)	-	28	-	mA
R_{PU}	Resistance of internal pull-up resistor	-	45	-	kΩ
R_{PD}	Resistance of internal pull-down resistor	-	45	-	kΩ
V_{IL_nRST}	Low-level input voltage of CHIP_PU to power off the chip	-	-	0.6	V

Notes:

1. Please see Appendix IO_MUX of *ESP32 Datasheet* for IO's power domain. VDD is the I/O voltage for a particular power domain of pins.
2. For VDD3P3_CPU and VDD3P3_RTC power domain, per-pin current sourced in the same domain is gradually reduced from around 40 mA to around 29 mA, V_{OH}>=2.64 V, as the number of current-source pins increases.
3. Pins occupied by flash and/or PSRAM in the VDD_SDIO power domain were excluded from the test.

B.2 BMI270

Basic Description

BMI270

The device is a highly integrated, low power inertial measurement unit (IMU) that combines precise acceleration and angular rate (gyroscopic) measurement with intelligent on-chip motion-triggered interrupt features.

BMI270 integrates:

- 16-bit digital, triaxial accelerometer with ±2g/±4g/±8g/±16g range
- 16-bit digital, triaxial gyroscope with ±125dps/±250dps/±500dps/±1000dps/±2000dps range

Key features

- Compact standard size LGA mold package, 14 pins, footprint 2.5x3.0mm² height 0.83mm
- Output data rates (ODR): 25 Hz ... 6.4 kHz (gyroscope) and 0.78 Hz ... 1.6 kHz (accelerometer)
- Programmable low-pass filter (accelerometer | gyroscope): bandwidth 5.5 | 11 ... 740 | 751 Hz
- Wide power supply range: Analog VDD 1.71V ... 3.6V and independent VDDIO 1.2V...3.6V
- Ultra-low current consumption: typ. 685 µA (in full ODR and aliasing free operation)
- Performance mode for gyroscope to minimize noise level: typ. < 7 mdps /√Hz.
- Built-in power management unit (PMU) for advanced power management and low power modes
- Rapid startup time: 2 ms for gyroscope (in fast start mode) and 2 ms for accelerometer

- Freely configurable secondary digital interface
  - 400 kHz I²C (Fm) master interface hub for 1 I2C AUX sensor (e.g. ext. magnetometer, pressure)
    - data synchronized to IMU
  - 10 MHz slave SPI (4-wire, 3-wire) for high speed, calibration free OIS / Dual OIS (SPI) applications
    - Up to 6.4 kHz ODR, control register access and down to 680 µs group delay
    - Connectible latency optimized low pass-filters with programmable cut-off frequencies

- 2 KB on-chip FIFO buffer for accelerometer, gyroscope, timestamps, and AUX sensor data

- Fast offset error compensation for accelerometer and gyroscope
- Fast sensitivity error compensation for gyroscope (CRT, reducing the error down to typ. 0.4%)

- HW synchronization of accelerometer, gyroscope, and AUX sensor (< 1 µs)
- Sensortime stamps for accurate system (host) and sensor (IMU) time synchronization (<40 µs)
- 2 independent programmable I/O pins for interrupt and synchronization events
- RoHS compliant, halogen and lead free

- BMI270 Features
  - Significant motion/Any motion/Motion detect/No motion/Stationary detect/Wrist wear wakeup/Wrist worn step counter and detector/Activity change recognition/Push arm down/Pivot up/Wrist jiggle/Flick in /out

B.3 MCP73831

MICROCHIP

MCP73831/2

Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

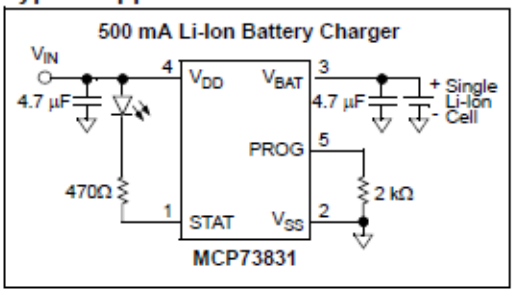
Features:

- Linear Charge Management Controller:
  - Integrated Pass Transistor
  - Integrated Current Sense
  - Reverse Discharge Protection
- High Accuracy Preset Voltage Regulation: ± 0.75%
- Four Voltage Regulation Options:
  - 4.20V, 4.35V, 4.40V, 4.50V
- Programmable Charge Current: 15 mA to 500 mA
- Selectable Preconditioning:
  - 10%, 20%, 40%, or Disable
- Selectable End-of-Charge Control:
  - 5%, 7.5%, 10%, or 20%
- Charge Status Output
  - Tri-State Output - MCP73831
  - Open-Drain Output - MCP73832
- Automatic Power-Down
- Thermal Regulation
- Temperature Range: -40°C to +85°C
- Packaging:
  - 8-Lead, 2 mm x 3 mm DFN
  - 5-Lead, SOT-23

Applications:

- Lithium-Ion/Lithium-Polymer Battery Chargers
- Personal Data Assistants
- Cellular Telephones
- Digital Cameras
- MP3 Players
- Bluetooth Headsets
- USB Chargers

Typical Application



Description:

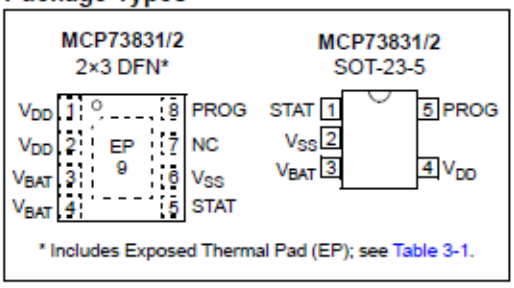
The MCP73831/2 devices are highly advanced linear charge management controllers for use in space-limited, cost-sensitive applications. The MCP73831/2 are available in an 8-Lead, 2 mm x 3 mm DFN package or a 5-Lead, SOT-23 package. Along with their small physical size, the low number of external components required make the MCP73831/2 ideally suited for portable applications. For applications charging from a USB port, the MCP73831/2 adhere to all the specifications governing the USB power bus.

The MCP73831/2 employ a constant-current/constant-voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831/2 devices limit the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability.

Several options are available for the preconditioning threshold, preconditioning current value, charge termination value and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio or percentage of the programmed constant current value. Preconditioning can be disabled. Refer to Section 1.0 "Electrical Characteristics" for available options and the Product Identification System for standard options.

The MCP73831/2 devices are fully specified over the ambient temperature range of -40°C to +85°C.

Package Types



* Includes Exposed Thermal Pad (EP); see Table 3-1.

B.4 TLV75533

TLV755P 500-mA, Low I_Q, Small Size, Low Dropout Regulator

1 Features

- Input Voltage Range: 1.45 V to 5.5 V
- Low I_Q: 25 µA (Typical)
- Low Dropout:
  - 238 mV (Maximum) at 500 mA (3.3 V_{OUT})
- Output Accuracy: 1% (Maximum at 85°C)
- Built-In Soft-Start With Monotonic V_{OUT} Rise
- Foldback Current Limit
- Active Output Discharge
- High PSRR: 46 dB at 100 kHz
- Stable With a 1-µF Ceramic Output Capacitor
- Packages:
  - 2.9-mm × 1.6-mm SOT-23-5
  - 1-mm x 1-mm X2SON-4
  - 2 mm × 2 mm WSON-6

2 Applications

- Set-Top Boxes, TV, and Gaming Consoles
- Portable and Battery-Powered Equipment
- Desktop, Notebooks, and Ultrabooks
- Tablets and Remote Controls
- White Goods and Appliances
- Grid Infrastructure and Protection Relays
- Camera Modules and Image Sensors

3 Description

The TLV755P is an ultra-small, low quiescent current, low-dropout regulator (LDO) that sources 500 mA with good line and load transient performance. The TLV755P is optimized for a wide variety of applications by supporting an input voltage range from 1.45 V to 5.5 V. To minimize cost and solution size, the device is offered in fixed output voltages ranging from 0.6 V to 5 V to support the lower core voltages of modern microcontrollers (MCUs). Additionally, the TLV755P has a low I_Q with enable functionality to minimize standby power. This device features an internal soft-start to lower inrush current, thus providing a controlled voltage to the load and minimizing the input voltage drop during start up. When shutdown, the device actively pulls down the output to quickly discharge the outputs and ensure a known start-up state.

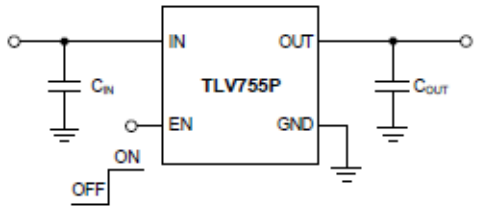
The TLV755P is stable with small ceramic output capacitors allowing for a small overall solution size. A precision band-gap and error amplifier provides a typical accuracy of 1%. All device versions have integrated thermal shutdown, current limit, and undervoltage lockout (UVLO). The TLV755P has an internal foldback current limit that helps reduce the thermal dissipation during short-circuit events.

Device Information[1]

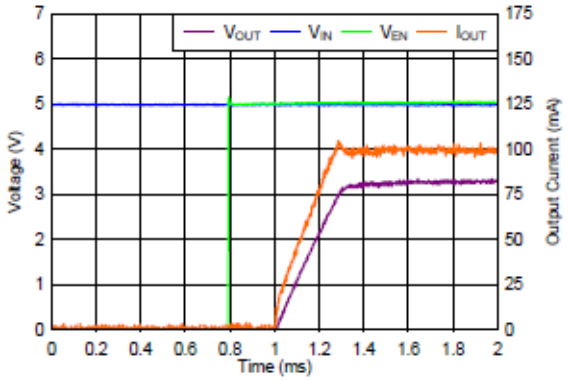
PART NUMBER	PACKAGE	BODY SIZE (NOM)
TLV755P	X2SON (4)	1.00 mm × 1.00 mm
	SOT-23 (5)	2.90 mm × 1.60 mm
	SON (6)	2.00 mm × 2.00 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Typical Application



Startup Waveform



C. Códigos

C.1 referencias.c

```
1
2  const int identificadormesh = 1;
3
4  //PARAMETROS REFERENCIA ALGORITMO RMS
5  const float rmspromref = 15.6;
6  const float holgurarms = 1;
7
8  //PARAMETROS REFERENCIA ALGORITMO FFT
9
10 const float farm_x_ref [8] = {20.312500, 39.843750, 60.156250, 100.000000, 120.312500,
   0.0, 0.0, 0.0};
11 const float farm_y_ref [8] = {20.312500, 39.843750, 60.156250, 80.468750, 100.000000,
   120.312500, 160.156250, 260.156250};
12 const float farm_z_ref [8] = {20.312500, 39.843750, 60.156250, 80.468750, 100.000000,
   160.156250, 200.000000, 220.312500};
13
14
15 const float magarm_x_ref [8] = {0.205560, 0.130936, 0.347650, 0.886892, 0.093426, 0.0,
   0.0, 0.0};
16 const float magarm_y_ref [8] = {0.465119, 0.180383, 1.080558, 0.918666, 0.291487,
   0.084512, 0.095190, 0.099675};
17 const float magarm_z_ref [8] = {19.179335, 0.577938, 0.826395, 0.217319, 0.204536,
   0.095647, 0.110846, 0.184067};
18
19 const float holguraFFT_mag = 1; // Holgura de comparación de amplitudes de la FFT
   actuales vs ref
20 const float holguraFFT_f = 1;
21
22
```

C.2 Código Principal

```
// 2022, Moisés Alejandro Barraza Riquelme

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "esp_log.h"
#include "driver/i2c.h"
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"

#include "FFT.h"
#include "MQTT.h"

#include "bmi270_config_file.c"
#include "referencias.c"

#define I2C_MASTER_SCL_IO       GPIO_NUM_22       //GPIO pin
#define I2C_MASTER_SDA_IO       GPIO_NUM_21       //GPIO pin
#define I2C_MASTER_FREQ_HZ          400000
#define ESP_SLAVE_ADDR          0x68
#define WRITE_BIT       0x0
#define READ_BIT        0x1
#define ACK_CHECK_EN      0x0
#define ACK_VAL         0x0
#define NACK_VAL        0x1

esp_err_t ret = ESP_OK;
esp_err_t ret2 = ESP_OK;

// PARAMETROS FUNCIONAMIENTO BMI270

#define Fodr 800 // Output Data Rate: 200, 400, 800, 1600 Hz

// PARAMETROS ALGORITMO FALLAS OPERACIONALES

#define gateFFT 5 // NUMERO DE VECES DE ALGORITMO 1 vs ALGORITMO 2
#define TamanoVentana 1024 // Número de muestras para FFT (potencia de 2)

float valx[1024], valx2[1024], valy[1024], valy2[1024], valz[1024], valz2[1024];
int j=0, calib=0;
float rmscalib[] = {0,0,0};


esp_err_t bmi_read(i2c_port_t i2c_num, uint8_t *data_addres, uint8_t *data_rd , size_t
size){
    if (size == 0) {
        return ESP_OK;
    }
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | WRITE_BIT, ACK_CHECK_EN);
    i2c_master_write(cmd, data_addres, size, ACK_CHECK_EN);
    i2c_master_start(cmd);
```

```
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | READ_BIT, ACK_CHECK_EN);
    if (size > 1) {
        i2c_master_read(cmd, data_rd, size - 1, ACK_VAL);
    }
    i2c_master_read_byte(cmd, data_rd + size - 1, NACK_VAL);
    i2c_master_stop(cmd);
    ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);
    return ret;
}

esp_err_t bmi_write(i2c_port_t i2c_num,uint8_t *data_addres, uint8_t *data_wr ,size_t
size)
{
    uint8_t size1 = 1;
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | WRITE_BIT, ACK_CHECK_EN);
    i2c_master_write(cmd, data_addres, size1, ACK_CHECK_EN);
    i2c_master_write(cmd, data_wr, size, ACK_CHECK_EN);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);
    return ret;
}

esp_err_t bmi_init(void)
{
    int i2c_master_port = I2C_NUM_0;
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = I2C_MASTER_SDA_IO;
    conf.sda_pullup_en = GPIO_PULLUP_DISABLE;
    conf.scl_io_num = I2C_MASTER_SCL_IO;
    conf.scl_pullup_en = GPIO_PULLUP_DISABLE;
    conf.master.clk_speed = I2C_MASTER_FREQ_HZ;
    conf.clk_flags = I2C_SCLK_SRC_FLAG_FOR_NOMAL; // 0
    i2c_param_config(i2c_master_port, &conf);
    return i2c_driver_install(i2c_master_port, conf.mode, 0, 0, 0);
}

void chipid(void)
{
    uint8_t reg_id=0x00;
    uint8_t tmp;

    bmi_read(I2C_NUM_0, &reg_id, &tmp,1);
    printf("valor de CHIPID: %2X \n\n",tmp);
    if(tmp == 0x24){
        printf("Chip reconocido.\n\n");
    }
    if(tmp != 0x24) {
        printf("Chip no reconocido. \nCHIP ID: %2x\n\n", tmp); // %2X
        exit(EXIT_SUCCESS);
    }
}
```

```
110
111 void softreset(void)
112 {
113     uint8_t reg_softreset=0x7E, val_softreset = 0xB6;
114
115     ret = bmi_write(I2C_NUM_0, &reg_softreset, &val_softreset,1);
116     vTaskDelay(1000 /portTICK_RATE_MS);
117     if(ret != ESP_OK){
118         printf("\nError en softreset: %s \n",esp_err_to_name(ret));
119     }
120     else {
121          printf("\nSoftreset: OK\n\n");
122     }
123 }
124
125 void powermode(void)
126 {
127     uint8_t reg_pwr_conf=0x7C, reg_pwr_ctrl=0x7D;
128     uint8_t tmp,tmp2;
129
130     ret= bmi_read(I2C_NUM_0, &reg_pwr_conf, &tmp,1);
131     printf("valor de PWR_CONF: %2X \n",tmp);
132     if(ret != ESP_OK){
133         printf("Error en PWR_CONF: %s \n",esp_err_to_name(ret));
134     }
135
136     ret2= bmi_read(I2C_NUM_0, &reg_pwr_ctrl, &tmp2,1);
137     printf("valor de PWR_CTRL: %2X \n",tmp2);
138     if(ret2 != ESP_OK){
139         printf("Error en PWR_CTRL: %s \n",esp_err_to_name(ret2));
140     }
141 }
142
143 void inicializacion(void){
144
145     uint8_t reg_pwr_conf_advpowersave=0x7C, val_pwr_conf_advpowersave=0x00;
146     uint8_t reg_init_ctrl=0x59, val_init_ctrl=0x00, val_init_ctrl2=0x01;
147     uint8_t reg_init_data=0x5E;
148
149     printf("Inicializando ...\n");
150
151     bmi_write(I2C_NUM_0, &reg_pwr_conf_advpowersave, &val_pwr_conf_advpowersave,1);
152
153     vTaskDelay(1000 /portTICK_RATE_MS);
154
155     ret=bmi_write(I2C_NUM_0, &reg_init_ctrl, &val_init_ctrl,1);
156
157     int config_size = sizeof(bmi270_config_file);
158
159     ret=bmi_write(I2C_NUM_0, &reg_init_data, (uint8_t*)bmi270_config_file,
config_size);
160     if(ret != ESP_OK){
161         printf("\nError cargando config_file\n");
162     }
163     else {
164         printf("\nConfig_file cargado.\n");
```

```
    }

    vTaskDelay(1000 /portTICK_RATE_MS);

    ret=bmi_write(I2C_NUM_0, &reg_init_ctrl, &val_init_ctrl2,1);

    printf("\nAlgoritmo de inicializacion finalizado.\n\n");

}

void check_inicializacion(void){
    uint8_t reg_internalstatus=0x21;
    uint8_t tmp;

    vTaskDelay(1000 /portTICK_RATE_MS);

    bmi_read(I2C_NUM_0, &reg_internalstatus, &tmp,1);
    printf("Init_status.0: %x \n",(tmp & 0b00001111));
    if((tmp & 0b00001111)==1){
        printf("Comprobacion Inicializacion: OK\n\n");
    }
    else {
        printf("Inicializacion fallida\n\n");
        exit(EXIT_SUCCESS);
    }
}

void bmipowermode(void)
{
    //PWR_CTRL: disable auxiliary sensor, gryo and temp; acc on
    //400Hz en datos acc, filter: performance optimized, acc_range +/-8g (1g = 9.80665 m/s2, alcance max: 78.4532 m/s2, 16 bit= 65536 => 1bit = 78.4532/32768 m/s2)
    uint8_t reg_pwr_ctrl=0x7D, val_pwr_ctrl=0x04;
    uint8_t reg_acc_conf=0x40, val_acc_conf;
    uint8_t reg_pwr_conf=0x7C, val_pwr_conf=0x00;

    // 0xA8 100hz, 0xA9 para 200Hz, 0xAA 400hz, 0xAB 800hz, 0xAC 1600hz
    if(Fodr==200 ) val_acc_conf=0xA9;
    else if(Fodr==400) val_acc_conf=0xAA;
    else if(Fodr==800) val_acc_conf=0xAB;
    else if(Fodr==1600) val_acc_conf=0xAC;
    else {
        printf("FRECUENCIA DE MUESTREO BMI270 INCORRECTO.\n");
        exit(EXIT_SUCCESS);
    };

    bmi_write(I2C_NUM_0, &reg_pwr_ctrl, &val_pwr_ctrl,1);
    bmi_write(I2C_NUM_0, &reg_acc_conf, &val_acc_conf,1);
    bmi_write(I2C_NUM_0, &reg_pwr_conf, &val_pwr_conf,1);

    vTaskDelay(1000 /portTICK_RATE_MS);
}

void internal_status(void)
{
    uint8_t reg_internalstatus=0x21;
```

```
    uint8_t tmp;

    bmi_read(I2C_NUM_0, &reg_internalstatus, &tmp,1);
    //printf("Initial status: %x \n",(tmp & 0b00001111));
    printf("Internal Status: %2X\n\n", tmp);
}

void RMS(void *arg)
{
    //ESP_LOGI("algoritmo1","Algoritmo RMS iniciado. core = %d",xPortGetCoreID());

    float rmsprom = 0;

    if (calib == 0){
        for (int i = 0; i < TamanoVentana-1; i++) {
        rmscalib[0] += valx[i];
        rmscalib[1] += valy[i];
        rmscalib[2] += valz[i];
        }

        rmscalib[0] = rmscalib[0] / TamanoVentana;
        rmscalib[1] = rmscalib[1] / TamanoVentana;
        rmscalib[2] = rmscalib[2] / TamanoVentana;
        calib = 1;
    }

    if(j==1){
        for (int i = 0; i < TamanoVentana-1; i++) {
        rmsprom += sqrt(pow(valx[i]-rmscalib[0],2)+pow(valy[i]-
rmscalib[1],2)+pow(valz[i]-rmscalib[2],2));
        }
    }

    if(j==0){
        for (int i = 0; i < TamanoVentana-1; i++) {
        rmsprom += sqrt(pow(valx2[i]-rmscalib[0],2)+pow(valy2[i]-
rmscalib[1],2)+pow(valz2[i]-rmscalib[2],2));
        }
    }

    rmsprom = rmsprom / TamanoVentana;

    // printf("Valores RMS:\n");
    // for (int i = 0; i < TamanoVentana-1; i++){
    //     printf(" %f,", rms[i]);

    // }
    // printf("\n\n");

    // printf("Promedio RMS: %f\n", rmsprom);

    if(rmsprom > rmspromref + holgurarms || rmsprom < rmspromref - holgurarms)
    {
        size_t size = 0;
        char *data = NULL;
        mwifi_data_type_t data_type = { 0x0 };
```

```
        uint8_t sta_mac[MWIFI_ADDR_LEN] = { 0 };
        mesh_addr_t parent_mac = { 0 };

        MDF_LOGI("Nodo escribiendo");

        esp_wifi_get_mac(ESP_IF_WIFI_STA, sta_mac);

        esp_mesh_get_parent_bssid(&parent_mac);

        size = asprintf(&data, "{\"Alerta\": \"Promedio RMS\", \"Dispositivo\": \"%d\", \"RMS actual\": \"%f\"}", identificadormesh, rmsprom);

        MDF_LOGD("Node send, size: %d, data: %s", size, data);
        ret = mwifi_write(NULL, &data_type, data, size, true);
        MDF_FREE(data);

        MDF_LOGW("Nodo, escritura finalizada");
    }

    //ESP_LOGI("algoritmo1","Algoritmo RMS finalizado. core = %d\n",xPortGetCoreID());
    vTaskDelete(NULL);

}

void FFT(void *arg)
{
    ESP_LOGI("algoritmo2","Algoritmo FFT iniciado. core = %d",xPortGetCoreID());

    int alarma = 0;
    float freq[TamanoVentana/2+1];
    float fft_input[TamanoVentana], fft_output[TamanoVentana];
    float fft_x[TamanoVentana/2+1], fft_y[TamanoVentana/2+1], fft_z[TamanoVentana/2+1];

    // EJE X

    fft_config_t *real_fft_plan = fft_init(TamanoVentana, FFT_REAL, FFT_FORWARD, fft_input, fft_output);

    if(j==1){
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_plan->input[i] = valx[i];
    }
    else {
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_plan->input[i] = valx2[i];
    }

    fft_execute(real_fft_plan);

    fft_x[0]=real_fft_plan->output[0]/TamanoVentana;
    fft_x[TamanoVentana/2]=real_fft_plan->output[1]/TamanoVentana;

        for (size_t i = 1; i < TamanoVentana/2; i++)
    {
```

```
        fft_x[i]= 2*(sqrt(pow(real_fft_plan->output[2*i],2)+pow(real_fft_plan->output[2*i+1],2))/TamanoVentana);
    }

    fft_destroy(real_fft_plan);

    //EJE Y

    fft_config_t *real_fft_plany = fft_init(TamanoVentana, FFT_REAL, FFT_FORWARD, fft_input, fft_output);

    if(j==1){
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_plany->input[i] = valy[i];
    }
    else {
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_plany->input[i] = valy2[i];
    }

    fft_execute(real_fft_plan);

    fft_y[0]=real_fft_plany->output[0]/TamanoVentana;
    fft_y[TamanoVentana/2]=real_fft_plany->output[1]/TamanoVentana;

        for (size_t i = 1; i < TamanoVentana/2; i++)
    {
        fft_y[i]= 2*(sqrt(pow(real_fft_plany->output[2*i],2)+pow(real_fft_plany->output[2*i+1],2))/TamanoVentana);
    }

    fft_destroy(real_fft_plany);

    // EJE Z

    fft_config_t *real_fft_planz = fft_init(TamanoVentana, FFT_REAL, FFT_FORWARD, fft_input, fft_output);

    if(j==1){
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_planz->input[i] = valz[i];
    }
    else {
        for (int i = 0 ; i < TamanoVentana; i++)
        real_fft_planz->input[i] = valz2[i];
    }

    fft_execute(real_fft_planz);

    fft_z[0]=real_fft_planz->output[0]/TamanoVentana;
    fft_z[TamanoVentana/2]=real_fft_planz->output[1]/TamanoVentana;

        for (size_t i = 1; i < TamanoVentana/2; i++)
    {
        fft_z[i]= 2*(sqrt(pow(real_fft_planz->output[2*i],2)+pow(real_fft_planz->output[2*i+1],2))/TamanoVentana);
```

```
    }

    fft_destroy(real_fft_planz);


    // ENCONTRAR F FUNDAMENTAL Y ARMONICOS

    float farm_x[8] = {0}, magarm_x[8] = {0}, farm_y[8] = {0}, magarm_y[8] = {0},
farm_z[8] = {0}, magarm_z[8] = {0};
    int n=0;

    for (size_t i = 0; i < TamanoVentana/2+1; i++)
    {
        freq[i] = i*(float)Fodr/(float)TamanoVentana;
    }

    for (int i = 1; i < TamanoVentana/2+1; i++)
    {
        if (fft_x[i]-fft_x[i-1]>0.05 && fft_x[i]-fft_x[i+1]>0.05)
        {
            magarm_x[n] = fft_x[i];
            farm_x[n] = freq[i];
            n++;
        }
    }

    n = 0;
    for (int i = 1; i < TamanoVentana/2+1; i++)
    {
        if (fft_y[i]-fft_y[i-1]>0.05 && fft_y[i]-fft_y[i+1]>0.05)
        {
            magarm_y[n] = fft_y[i];
            farm_y[n] = freq[i];
            n++;
        }
    }

    n = 0;
    for (int i = 1; i < TamanoVentana/2+1; i++)
    {
        if (fft_z[i]-fft_z[i-1]>0.05 && fft_z[i]-fft_z[i+1]>0.05)
        {
            magarm_z[n] = fft_z[i];
            farm_z[n] = freq[i];
            n++;
        }
    }

    // printf("\nEJE X\nfarmonico | marmonico                    j=%d\n",j);
    // for (size_t i = 0; i < 8; i++)
    // {
    //     printf("%f | %f\n", farm_x[i], magarm_x[i]);
    // }
    // printf("\nEJE Y\nfarmonico | marmonico                    j=%d\n",j);
    // for (size_t i = 0; i < 8; i++)
    // {
```

```
    //     printf("%f | %f\n", farm_y[i], magarm_y[i]);
    // }
    // printf("\nEJE Z\nfarmonico | marmonico                    j=%d\n",j);
    // for (size_t i = 0; i < 8; i++)
    // {
    //     printf("%f | %f\n", farm_z[i], magarm_z[i]);
    // }


    for (size_t i = 0; i < 8; i++)
    {
        if(magarm_x[i] > magarm_x_ref[i] + holguraFFT_mag || magarm_x[i] < magarm_x_ref[i] - holguraFFT_mag || magarm_y[i] > magarm_y_ref[i] + holguraFFT_mag || magarm_y[i] < magarm_y_ref[i] - holguraFFT_mag || magarm_z[i] > magarm_z_ref[i] + holguraFFT_mag || magarm_z[i] < magarm_z_ref[i] - holguraFFT_mag)
        {
            alarma=1;
            i=8;
        }

        if (farm_x[i] > farm_x_ref[n] + holguraFFT_f || farm_x[i] < farm_x_ref[n] - holguraFFT_f || farm_y[i] > farm_y_ref[n] + holguraFFT_f || farm_y[i] < farm_y_ref[n] - holguraFFT_f || farm_z[i] > farm_z_ref[n] + holguraFFT_f || farm_z[i] < farm_z_ref[n] - holguraFFT_f)
        {
            alarma=1;
            i=8;
        }

    }

    if (alarma==1)
    {
        size_t size = 0;
        char *data = NULL;
        mwifi_data_type_t data_type = { 0x0 };
        uint8_t sta_mac[MWIFI_ADDR_LEN] = { 0 };
        mesh_addr_t parent_mac = { 0 };

        MDF_LOGI("Nodo escribiendo");

        esp_wifi_get_mac(ESP_IF_WIFI_STA, sta_mac);

        esp_mesh_get_parent_bssid(&parent_mac);

        size = asprintf(&data, "{\"Alerta\": \"Analisis FFT\", \"Dispositivo\": \"%d\", \"mag_x\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\", \"f_x\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\", \"mag_y\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\", \"f_y\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\", \"mag_z\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\", \"f_z\": \"{%f, %f, %f, %f, %f, %f, %f, %f}\"}",
        identificadormesh, magarm_x[0], magarm_x[1], magarm_x[2], magarm_x[3], magarm_x[4], magarm_x[5], magarm_x[6], magarm_x[7], farm_x[0], farm_x[1], farm_x[2], farm_x[3], farm_x[4], farm_x[5], farm_x[6], farm_x[7], magarm_y[0], magarm_y[1], magarm_y[2], magarm_y[3], magarm_y[4], magarm_y[5], magarm_y[6], magarm_y[7], farm_y[0], farm_y[1], farm_y[2], farm_y[3], farm_y[4], farm_y[5], farm_y[6], farm_y[7], magarm_z[0], magarm_z[1], magarm_z[2], magarm_z[3], magarm_z[4],
```

```
magarm_z[5], magarm_z[6], magarm_z[7], farm_z[0], farm_z[1], farm_z[2], farm_z[3],
farm_z[4], farm_z[5], farm_z[6], farm_z[7]);

        MDF_LOGD("Node send, size: %d, data: %s", size, data);
        ret = mwifi_write(NULL, &data_type, data, size, true);
        MDF_FREE(data);

        MDF_LOGW("Nodo, escritura finalizada");
    }

    //ESP_LOGI("algoritmo2","Algoritmo FFT finalizado. core = %d\n",xPortGetCoreID());
    vTaskDelete(NULL);
}

void lectura(void* arg)
{
    uint8_t reg_intstatus=0x03, tmp;
    int bytes_data8 = 6;
    uint8_t reg_data = 0x0C, data_data8[bytes_data8];
    uint16_t acc_x, acc_y, acc_z;
    int n = 0, m = 0; //contador de lecturas

    ESP_LOGI("lectura","Comienza Lectura. core = %d\n",xPortGetCoreID());

    while (1)
    {
        bmi_read(I2C_NUM_0, &reg_intstatus, &tmp,1);
        // printf("Init_status.0: %x - mask: %x \n", tmp, (tmp & 0b10000000));
        //ESP_LOGI("leturabmi", "acc_data_ready: %x - mask(80): %x \n", tmp, (tmp &
0b10000000));

        if ((tmp & 0b10000000) == 0x80)
        {
            ret= bmi_read(I2C_NUM_0, &reg_data, (uint8_t*) data_data8, bytes_data8);

            if(ret != ESP_OK){
                printf("Error lectura: %s \n",esp_err_to_name(ret));
            }

            acc_x = ((uint16_t) data_data8[1] << 8) | (uint16_t) data_data8[0];
            acc_y = ((uint16_t) data_data8[3] << 8) | (uint16_t) data_data8[2];
            acc_z = ((uint16_t) data_data8[5] << 8) | (uint16_t) data_data8[4];

            //printf("acc_x: %f m/s2     acc_y: %f m/s2     acc_z: %f m/s2\n",
(int16_t)acc_x*(78.4532/32768), (int16_t)acc_y*(78.4532/32768), (int16_t)acc_z*
(78.4532/32768));
            //ESP_LOGI("leturabmi", "n:%d - acc_x: %f g     acc_y: %f g     acc_z: %f
g  \n", n,(int16_t)acc_x*(8.000/32768), (int16_t)acc_y*(8.000/32768), (int16_t)acc_z*
(8.000/32768) );

            if (j==0) {
                valx[n] = (int16_t)acc_x*(78.4532/32768);
                valy[n] = (int16_t)acc_y*(78.4532/32768);
                valz[n] = (int16_t)acc_z*(78.4532/32768);
            }
            if (j==1) {
```

```
522                 valx2[n] = (int16_t)acc_x*(78.4532/32768);
523                 valy2[n] = (int16_t)acc_y*(78.4532/32768);
524                 valz2[n] = (int16_t)acc_z*(78.4532/32768);
525             }
526 
527             if (n == TamanoVentana-1){
528 
529                 if (j==0) {
530                     j=1;
531                     // ESP_LOGI("lectura","Buffer 1 completado. core = %d",
    xPortGetCoreID());
532                     // ESP_LOGI("lectura","Buffer 2 sobreescribiendo. core = %d",
    xPortGetCoreID());
533                 }
534                 else {
535                     j=0;
536                     // ESP_LOGI("lectura","Buffer 2 completado. core = %d",
    xPortGetCoreID());
537                     // ESP_LOGI("lectura","Buffer 1 sobreescribiendo. core = %d",
    xPortGetCoreID());
538                 }
539 
540                 // printf("\n\nVentana datos:");
541                 // if(j==1){
542                 //     printf(" j=%d\n\n",j);
543                 //     printf("eje x:\n");
544                 //     for (int i = 0; i < TamanoVentana-1;i++){
545                 //     printf(" %f,", valx[i]);
546                 //     }
547                 //     printf("\n\neje y:\n");
548                 //     for (int i = 0; i < TamanoVentana-1;i++){
549                 //     printf(" %f,", valy[i]);
550                 //     }
551                 //     printf("\n\neje z:\n");
552                 //     for (int i = 0; i < TamanoVentana-1;i++){
553                 //     printf(" %f,", valz[i]);
554                 //     }
555                 // }
556                 // else {
557                 //     printf(" j=%d\n",j);
558                 //     printf("eje x:\n");
559                 //     for (int i = 0; i < TamanoVentana-1;i++){
560                 //     printf(" %f,", valx2[i]);
561                 //     }
562                 //     printf("\n\neje y:\n");
563                 //     for (int i = 0; i < TamanoVentana-1;i++){
564                 //     printf(" %f,", valy2[i]);
565                 //     }
566                 //     printf("\n\neje z:\n");
567                 //     for (int i = 0; i < TamanoVentana-1;i++){
568                 //     printf(" %f,", valz2[i]);
569                 //     }
570                 // }
571 
572                 xTaskCreatePinnedToCore(RMS, "promedio", 4*1024, NULL, 1, NULL,1);
573 
```

```
                if (m == gateFFT-1 ){

                    xTaskCreatePinnedToCore(FFT, "fft", 20*1024, NULL, 1, NULL,1);
                    m = -1;
                }
                m++;
                n = -1;
            }
            n++;
        }
            //ESP_LOGI("ddd","n=%d, Sigue leyendo... core = %d\n", n,
xPortGetCoreID());
    }
}

void app_main()
{

    mwifi_init_config_t cfg = MWIFI_INIT_CONFIG_DEFAULT();
    mwifi_config_t config = {
        .router_ssid = CONFIG_ROUTER_SSID,
        .router_password = CONFIG_ROUTER_PASSWORD,
        .mesh_id = CONFIG_MESH_ID,
        .mesh_password = CONFIG_MESH_PASSWORD,
    };

    //Tags para impresiones en monitor serial

    esp_log_level_set("*", ESP_LOG_INFO);
    esp_log_level_set(TAG, ESP_LOG_DEBUG);
    esp_log_level_set("mesh_mqtt", ESP_LOG_DEBUG);


    //Inicializar red mesh

    MDF_ERROR_ASSERT(mdf_event_loop_init(event_loop_cb));
    MDF_ERROR_ASSERT(wifi_init());
    MDF_ERROR_ASSERT(mwifi_init(&cfg));
    MDF_ERROR_ASSERT(mwifi_set_config(&config));
    MDF_ERROR_ASSERT(mwifi_start());

    //Creación de tareas

    xTaskCreate(node_read_task, "node_read_task", 4 * 1024,
                NULL, CONFIG_MDF_TASK_DEFAULT_PRIOTY, NULL);

    TimerHandle_t timer = xTimerCreate("print_system_info", 10000 / portTICK_RATE_MS,
                                       true, NULL, print_system_info_timercb);
    xTimerStart(timer, 0);

    //BMI

    ESP_ERROR_CHECK(bmi_init());

    softreset();
    chipid();
```

```
629
630     inicializacion();
631     check_inicializacion();
632     bmipowermode();
633
634     internal_status();
635
636     xTaskCreatePinnedToCore(lectura, "lectura_bmi", 4*1024, NULL, 1, NULL, 0);
637 }
```

UNIVERSIDAD DE CONCEPCION – FACULTAD DE INGENIERIA
RESUMEN DE MEMORIA DE TITULO

Departamento **:** Departamento de Ingeniería Eléctrica
Carrera **:** Ingeniería Civil Electrónica
Nombre del memorista **:** Moisés Alejandro Barraza Riquelme
Título de la memoria **:** Desarrollo de un sistema de detección de anomalías operacionales en procesos industriales basados en sensores IoT.
Fecha de la presentación oral: 26 de Agosto de 2022

Profesor(es) Guía **:** Pablo Esteban Aqueveque Navarro
Profesor(es) Revisor(es) **:** Sergio Sobarzo G., Mario Medina C.
Concepto **:**
Calificación **:**

Resumen

El trabajo desarrollado en esta memoria presenta un estudio del arte actual de las tecnologías IoT con fines industriales, se discuten sus capacidades, beneficios y desafíos que presenta.
Se propone la identificación del estado de salud de motores eléctricos a partir del análisis de sus vibraciones de operación. Para esto se desarrolla un sistema electrónico embebido, capaz de conectarse a una red inalámbrica por la cual poder enviar un mensaje de alerta en el caso de que se compruebe, por medio de su vibración, que un motor eléctrico está operando fuera de las condiciones esperadas.

El trabajo involucra el diseño de una red inalámbrica de características industriales, el diseño y desarrollo de la placa electrónica del sistema embebido, y el software necesario para el control del hardware, el análisis de la vibración y la determinación de fallas operacionales.

Se implementa el sistema en laboratorio y se realizan pruebas que permiten concluir con el cumplimiento de todas las características establecidas en su diseño. Se comprueba el funcionamiento esperado del dispositivo embebido, sus algoritmos y la red propuesta. Se presentan además las diferencias entre los resultados del dispositivo embebido y los que se obtienen en un software matemático avanzado.