Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

NICOLÁS MARTÍNEZ JIMÉNEZ

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Sistema IIoT para monitorización de motores eléctricos y detección de anomalías mediante el análisis de vibraciones en rodamientos.

IIoT system for electric motors monitoring and bearing anomaly detection through vibration analysis.

Tutores: Víctor Manuel González Suárez, Miguel Alonso González

Junio, 2022

Índice

Índice de Tablas

Índice de Ilustraciones

1 Introducción

1.1 Planteamiento del problema

La empresa Intermark, en concreto su división Intermark IT, dedicada al sector de la industria 4.0 ha comenzado a tener interés por el campo del mantenimiento de motores eléctricos. Gracias a este interés, se han sentado las bases para el desarrollo de este proyecto.

Tradicionalmente, se han utilizado las siguientes estrategias de mantenimiento [1] [2]:

- *Mantenimiento Correctivo o funcionamiento hasta avería:* no se realizan tareas de mantenimiento hasta que se produce una avería. Los equipos siguen trabajando hasta la rotura provocando que las averías tienen una mayor gravedad y el coste de reparación es más elevado.

- *Mantenimiento Preventivo*: esta estrategia consiste en programar tareas de mantenimiento periódicas del estado de la maquinaria. De este modo, se consigue reducir las paradas en la producción por averías, pero aumenta considerablemente el coste del proceso de mantenimiento ya que pueden darse situaciones en las que se realice el mantenimiento de maquinaria que no lo necesita.

- *Condition-Based Maintenance: es* un mantenimiento preventivo que incluye una combinación de magnitudes y características monitorizadas, inspección, análisis y la aplicación de tareas de mantenimiento. Su objetivo es aplicar tareas de mantenimiento ante la evidencia mostrada por las condiciones monitorizadas.

- *Mantenimiento Predictivo (Predictive Maintenance):* es un tipo de *Condition-base Maintenance* que realiza una predicción en base a análisis o conocimientos adquiridos previamente para determinar si es necesario o no realizar tareas de mantenimiento.

Existen diversos ejemplos en la literatura donde se ha demostrado que invertir en Mantenimiento Predictivo puede suponer un mayor ahorro a largo plazo [3] [4].

Dentro de la diversidad de maquinaria que puede existir en la industria, los motores eléctricos de inducción de jaula de ardilla son los más utilizados. Esto se debe principalmente a su bajo coste de adquisición y mantenimiento, su estructura compacta y su capacidad para trabajar en ambientes industriales. [5]

Las averías en este tipo de motores pueden tener diferentes orígenes [6]. Estas averías pueden estar provocadas por errores en la fabricación o deterioro de los materiales. Estos fallos se les denomina *Fallos de origen interno*.

Fallos de origen interno	
Mecánicos	**Eléctricos**
Desplazamiento de las láminas o las bobinas	Defectos de aislamiento
Fallos en los rodamientos	Rotura rotor
Excentricidades estáticas o dinámicas	Fallo magnético del circuito

Tabla 1: Fallos origen interno

Las averías pueden también estar producidas por la intervención de otros elementos externos del ambiente, la carga o la fuente de energía. A estos últimos se les denomina *Fallos de origen externo*.

Fallos de origen externo		
Mecánico	**Eléctrico**	**Entorno**
Variación de la carga	Transitorios y fluctuaciones eléctricas	Humedad
Sobrecarga	Fallos en la instalación o la conexión	Temperatura
Mal ensambla miento	Desbalanceo de voltaje	Polución

Tabla 2: Fallos origen externo

Según la experiencia reportada por fabricantes, centros de reparación y estudios académicos, aproximadamente el 45% de las averías tienen como origen los rodamientos [7]. El porcentaje restante se distribuye entre el estátor, en el rotor y otros elementos.

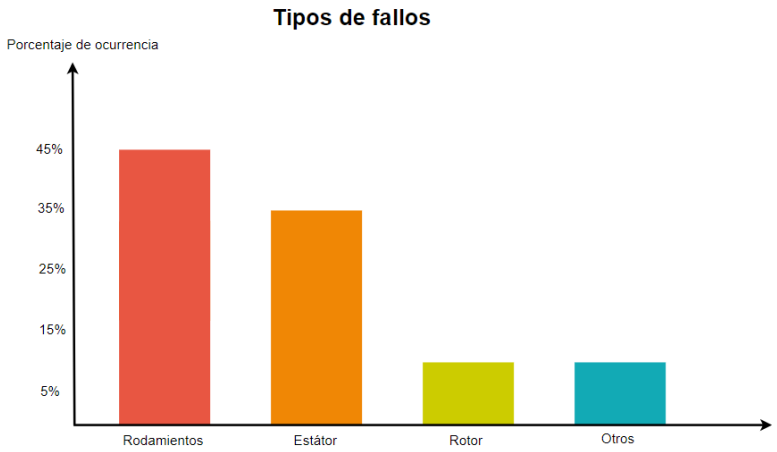

Ilustración 1: Tipos de fallos en motores eléctricos

El mantenimiento predictivo puede llevarse a cabo a través del análisis de diferentes magnitudes físicas. En función de la magnitud elegida, pueden aplicarse diferentes técnicas basadas en la monitorización [8]:

- Análisis de vibraciones
- Análisis de emisión acústica
- Fusión de análisis de vibraciones y emisión acústica
- Análisis de Corriente eléctrica
- Análisis de aceite y lubricación
- Análisis termográfico
- Inspección visual
- Monitorización de rendimiento
- Monitorización de tendencias

En este proyecto los esfuerzos se han centrado en la detección de fallos o defectos en rodamientos debido a que son componentes muy sensibles a cualquier variación y pueden ser utilizados como un elemento representativo del estado de la máquina. De las técnicas descritas anteriormente, la más eficaz para la detección de fallos en rodamientos es el Análisis de vibraciones [9].

Los rodamientos pueden presentar diferentes tipos de fallos o defectos [10]:

- *Defectos localizados (localised defects):* estos defectos se producen en las primeras etapas y pueden ser grietas o hendiduras en la superficie de alguno de los componentes del rodamiento: pista interior, pista exterior, jaula o los elementos rodantes. En rodamientos que se encuentren correctamente instalados y lubricados estos defectos aparecen debido al desgaste y la fatiga de los propios componentes.

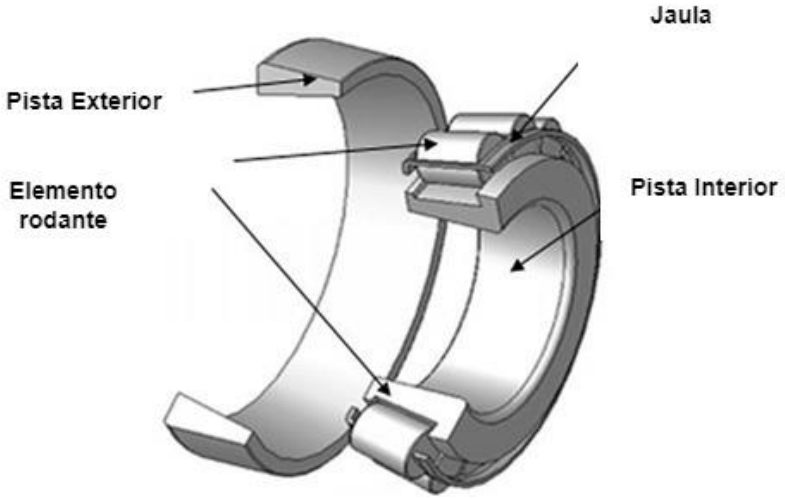

Ilustración 2: Partes de un rodamiento

Este tipo de defectos provocan la aparición de ciertas frecuencias características, denominadas frecuencias de fallo que pueden ser calculadas con las fórmulas de la Tabla 3.

Fault Frequency Formulas	
Bearing Fault	**Formula**
Ballpass Frequency Outer Race	$BPFO = \frac{n\text{fr}}{2}\left(1 - \frac{d}{D}cos\Phi\right)$
Ballpass Frequency Inner Race	$BPFI = \frac{n\text{fr}}{2}\left(1 + \frac{d}{D}cos\Phi\right)$
Fundamental Train Frequency (cage speed)	$FTF = \frac{\text{fr}}{2}\left(1 - \frac{d}{D}cos\Phi\right)$
Ball Spin Frequency	$BSF = \frac{D}{2d}[1 - \left(\frac{d}{D}cos\Phi\right)^2]$

Tabla 3: Frecuencias de fallo en rodamientos

D representa el diámetro de la pista, d el diámetro de las bolas, n el número de bolas del rodamiento, f_r la frecuencia de giro del eje en Hz y Φ el ángulo de incidencia. En la Ilustración 3 [11] se muestra la localización de los valores necesarios para calcular las frecuencias de fallo.

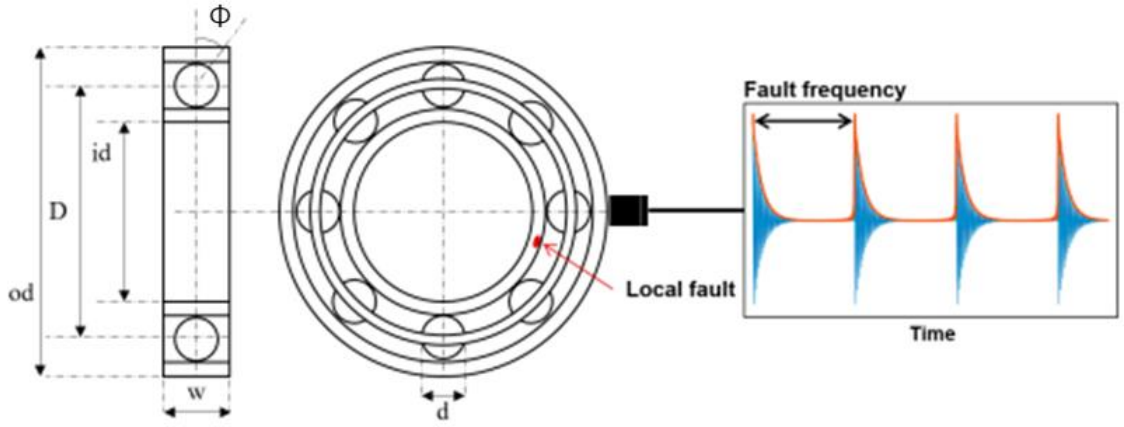

Ilustración 3: Geometría del rodamiento

- *Defectos extendidos (extended defects):* estos defectos son más severos que los defectos localizados. Normalmente, cuando tiene lugar un defecto localizado, la continua acción de las bolas al pasar por el defecto hace que este se agrave y se extienda transformándolo en un defecto extendido.

- *Defectos distribuidos (distributed defects):* son los defectos de mayor severidad. Están relacionados con imperfecciones de fabricación como como aspereza, ondulación, componentes fuera de tamaño y pistas desalineadas.

Lo más interesante y beneficiosos es detectar defectos localizados ya que al tener una alta impulsividad facilitan la detección y además se sustituir el rodamiento antes de que el defecto pueda influir en el funcionamiento del motor.

La empresa se ha marcado el objetivo de aumentar su conocimiento en el campo del análisis de vibraciones, de modo que tenga la capacidad de implementar una herramienta para monitorizar motores eléctricos y detectar anomalías en su funcionamiento mediante el análisis de vibraciones.

Con la elaboración de este proyecto se pretende implementar un sistema IIoT que permita monitorizar las vibraciones producidas en motores eléctricos y detectar la existencia de comportamientos anómalos.

Además, hay interés también por tener la capacidad de simular situaciones de fallo sin modificar los elementos físicos del propio motor. Por tanto, además de la construcción del sistema de detección de anomalías, será necesario analizar las posibilidades disponibles para llevar a cabo esta tarea.

En la Ilustración 4 se muestra un resumen gráfico de los 3 grandes bloques planteados al inicio del proyecto.

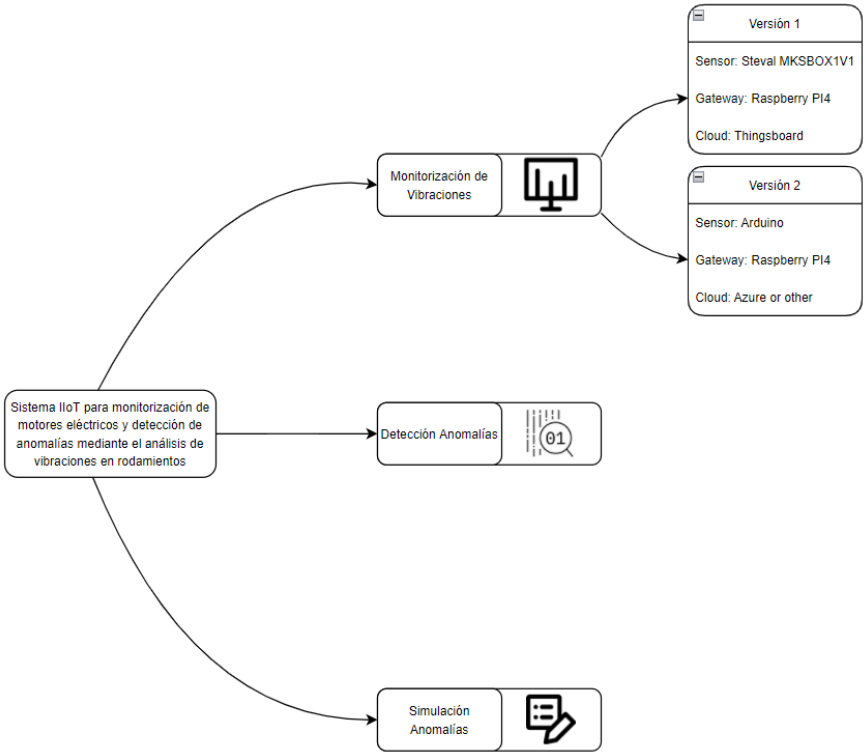

Ilustración 4: Planteamiento inicial

2 Alcance y Objetivos

En este proyecto se busca ganar conocimiento en el campo de análisis de vibraciones aplicado a la detección de fallos en rodamientos en motores eléctricos y aplicar estos conocimientos al desarrollo de un sistema IIoT que permita detectar cuándo se han producido defectos en rodamientos de modo que puedan anticiparse posibles situaciones de avería.

Se han establecido los siguientes objetivos:

- Búsqueda de las tecnologías existentes para la detección de fallos en rodamientos mediante la técnica del análisis de vibraciones.

- Desarrollo de una aplicación IIoT con capacidad para recogida de datos, procesado, análisis, visualización y persistencia. Dentro de la lógica de esta aplicación se implementarán las técnicas de interés encontradas en la literatura con el objetivo de detectar o prever fallos producidos por rodamientos en mal estado.

- Uso de componentes hardware y software de interés para Intermark (Steval-MKSBOX1V1, Raspberry Pi y Thingsboard)

- Mostrar de forma detallada el despliegue de las diferentes herramientas utilizadas y de los elementos software desarrollados para la comunicación de los componentes de la arquitectura y la implementación de las funcionalidades demandas de modo que pueda ser replicado fácilmente por personal de la empresa.

- Debido a la imposibilidad de contar con un banco de pruebas donde poder forzar las diferentes situaciones de fallo de los rodamientos, se estudiarán las alternativas para la simulación de fallos, generando situaciones ficticias de fallo en rodamientos lo más parecidas a las que se darán en un futuro.

- Generar una segunda versión de la aplicación utilizando componentes hardware alternativos a los utilizados en la primera versión y realizando un despliegue de nube pública en lugar de nube privada.

- Automatización del proceso de detección de anomalías para la generación de alertas utilizando técnicas de Machine Learning.

3 Planificación y presupuesto

3.1 Planificación de tareas

El reparto de tareas a lo largo del proyecto ha sido como se muestra en el diagrama de Gantt que puede verse en la Ilustración 5. Dado que el horario acordado para asistir a la oficina de Intermark ha sido de 15:30 a 18:30, cada día de trabajo son 3 horas.

	Nombre de tarea	Duración	Comienzo	Fin
1	**Desarrollo firmware**	**25 días**	**lun 11/10/21**	**lun 15/11/21**
2	Configuración enteorno desarrollo	2 días	lun 11/10/21	mar 12/10/21
3	Desarrollo firmware	20 días	mié 13/10/21	mar 09/11/21
4	**Configuración gateway**	**14 días**	**mié 10/11/21**	**lun 29/11/21**
5	Configuración enteorno desarrollo	2 días	mié 10/11/21	jue 11/11/21
6	Desarrollo scripts	12 días	vie 12/11/21	lun 29/11/21
7	**Configuración cloud**	**9 días**	**vie 12/11/21**	**mié 24/11/21**
8	Instalación	2 días	vie 12/11/21	lun 15/11/21
9	Configuración	2 días	mar 16/11/21	mié 17/11/21
10	Pruebas	5 días	jue 18/11/21	mié 24/11/21
11	**Detección anomalías**	**40 días**	**jue 25/11/21**	**mié 19/01/22**
12	Estudio de la literatura	25 días	jue 25/11/21	mié 29/12/21
13	Implementación	15 días	jue 30/12/21	mié 19/01/22
14	Simulación de anomalías	25 días	jue 20/01/22	mié 23/02/22
15	Desarrollo 2 versión	30 días	jue 24/02/22	mié 06/04/22
16	Machine Learning	20 días	jue 07/04/22	mié 04/05/22
17	Documentación	150 días	lun 11/10/21	vie 06/05/22

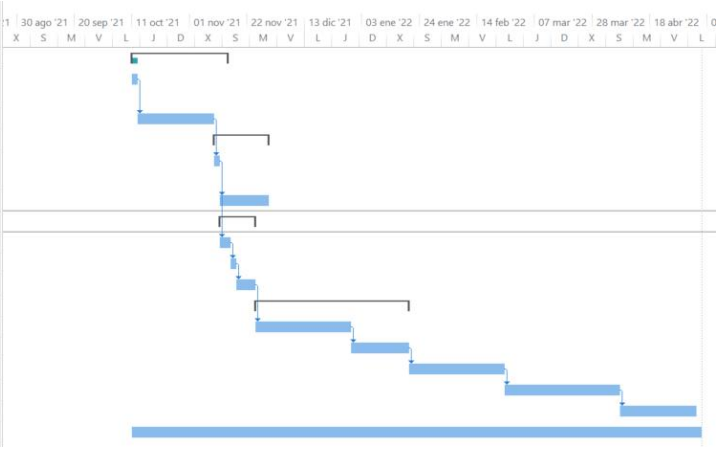

Ilustración 5: Planificación

3.2 Presupuesto

En esta sección del documento se recoge el presupuesto del proyecto.

3.2.1 Recursos Humanos

En la Tabla 4 se muestra el presupuesto destinado a recursos humanos.

Categoría	Número de personas	N.º Horas	Precio Unitario	Importe Total
Programador	1	450	15€	6.750€
			Total	6.750€

Tabla 4: Recursos humanos

3.2.2 Materiales

En la Tabla 5 se muestra el presupuesto destinado a materiales.

Descripción	Cantidad	Precio Unitario	Importe Total
Motor	1	184,89€	184,89€
Steval MKSBOX1V1	1	52,31€	52,31€
ST LinkV2	1	18,72€	18,72€
Raspberry Pi 4	1	51,84€	51,84€
Ordenador Lenovo Thinkpad	1	629,10	629,10€
		Total	936,86€

Tabla 5: Materiales

3.2.3 Resumen del presupuesto

En la Tabla 6 se muestra el resumen de los importes de cada una de las categorías y el importe total del presupuesto del proyecto.

Nombre de la categoría	Importe
Recursos humanos	6.750€
Materiales	936,86€
Total	7.686,86€

Tabla 6: Resumen presupuesto

4 Estado del arte

Existen múltiples técnicas para analizar las vibraciones producidas en los motores eléctricos [12]. Las más simples son las *Técnicas en el Dominio del Tiempo.* Este tipo de técnicas consisten en analizar la forma de la señal de vibración respecto al tiempo. Para analizar la señal obtenida, se utilizan indicadores como Amplitud pico a pico, RMS (Root Mean Square), factor de cresta (Crest Factor) o la Kurtosis.

Por otro lado, existen las *Técnicas en el Dominio de la Frecuencia*. Estas técnicas consisten en transformar la señal en una serie de componentes de frecuencia discretos de modo que puedan analizarse fácilmente los componentes de frecuencias de interés. Para realizar esta transformación, típicamente se utiliza la Transformada Rápida de Fourier (FFT) [13]. Sin embargo, la FFT no es la mejor técnica. Existen otras alternativas como el Espectro de Potencia (Power Spectrum) o el Envelope Analysis.

Otra posibilidad es hacer uso de las *Técnicas en el dominio tiempo-frecuencia.* Este tipo de técnicas proporciona un enfoque simultaneo del dominio del tiempo y el de la frecuencia. Los algoritmos disponibles para realizar este tipo de transformaciones son la Transformada de Fourier de Tiempo Reducido (STFT), la distribución de Wagner-Ville (WVD) y el Wavelet Analysis (WA).

Por último, existe la posibilidad de utilizar técnicas más novedosas basadas en Machine Learning. Algunos algoritmos utilizados en la literatura han sido K-Vecinos, Máquinas de Soporte Vectorial (SVM) o Redes Neuronales [14] [15].

Cuando existe la presencia de un defecto localizado en un rodamiento, la superficie de los elementos del rodamiento interacciona con el defecto produciendo un impacto que se repite en cada giro del eje. Estos impactos excitan resonancias de alta frecuencia.

Sin embargo, la información de diagnóstico que revela el tipo de defecto está en la frecuencia de repetición no en la frecuencia de resonancia como se muestra en la Ilustración 6 [16].

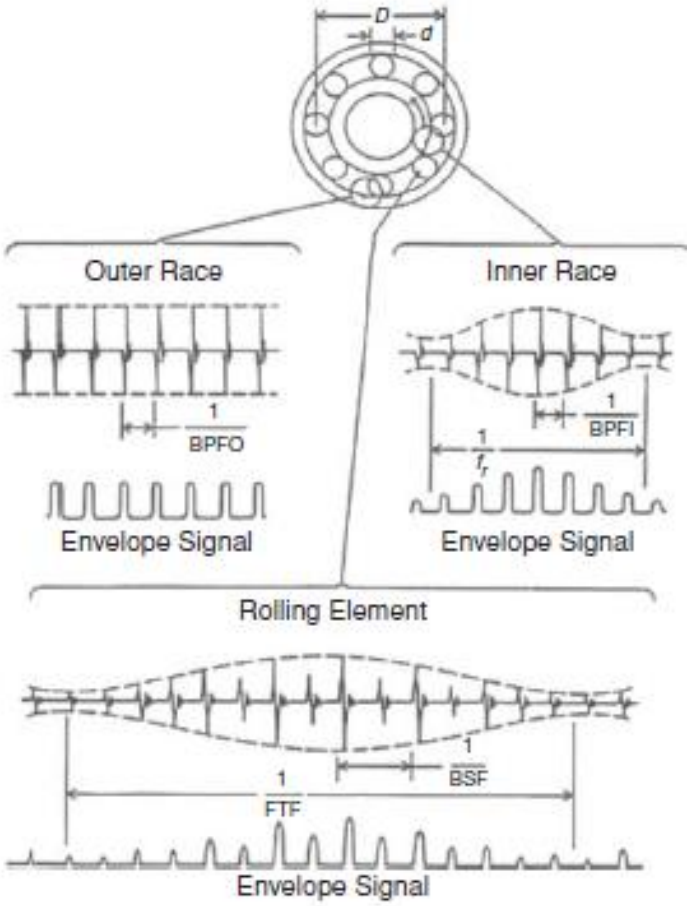

Ilustración 6: Señales típicas y señal envolvente de defectos locales

A esto hay que añadirle que, debido a las variaciones de la carga producidas por el paso del defecto por la zona de carga, se produce una modulación en amplitud de la frecuencia de repetición en la señal [17]. Por lo tanto, es necesario realizar una demodulación que permita extraer la señal del defecto para poder analizarla en el dominio de la frecuencia. Este efecto se muestra en la Ilustración 7 [18].

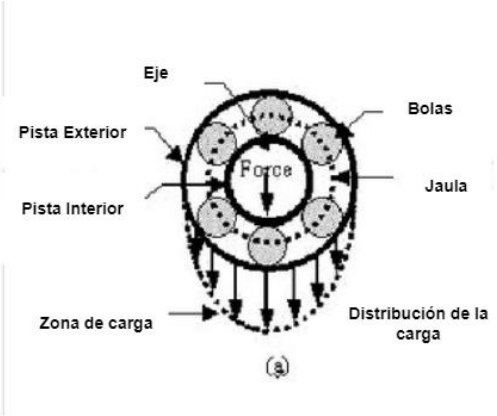

Ilustración 7: Variaciones en el paso por la zona de carga

El Envelope Analysis permite llevar a cabo esta demodulación, aunque se produzcan fluctuaciones aleatorias y extraer la señal enmascarada en la señal portadora que contiene, además, ruido y señales de otros componentes.

Para realizar una extracción óptima, debe analizarse la señal en la banda adecuada, donde la ratio señal/ruido (SNR) sea mayor. Para esto se han desarrollado técnicas como Order Tracking (OR), Adaptative Noise Cancelation (ANC), self-adaptative Noise Cancelation (SANC) o Discrete/Random Separation (DRS) que tienen el objetivo de eliminar el ruido de fondo que dificulta el diagnóstico.

Otra alternativa es el uso del Kurtograma para extraer la banda de frecuencia óptima, en especial, cuando no se tiene conocimiento previo del comportamiento de la máquina monitorizada [19].

Esta técnica está basada en la Spectral Kurtosis (SK) [20]. SK tiene valores elevados en aquellas bandas donde la señal provocada por el defecto es dominante y valores cercanos a cero cuando la banda está dominada por componentes estacionarios [16]. Es decir, tiene alta sensibilidad para detectar en que banda de frecuencia la impulsividad de la señal generada por los defectos es mayor.

El Kurtograma consiste en realizar la SK de todas las frecuencias y anchos de banda para extraer en qué banda el valor de la Kurtosis es mayor.

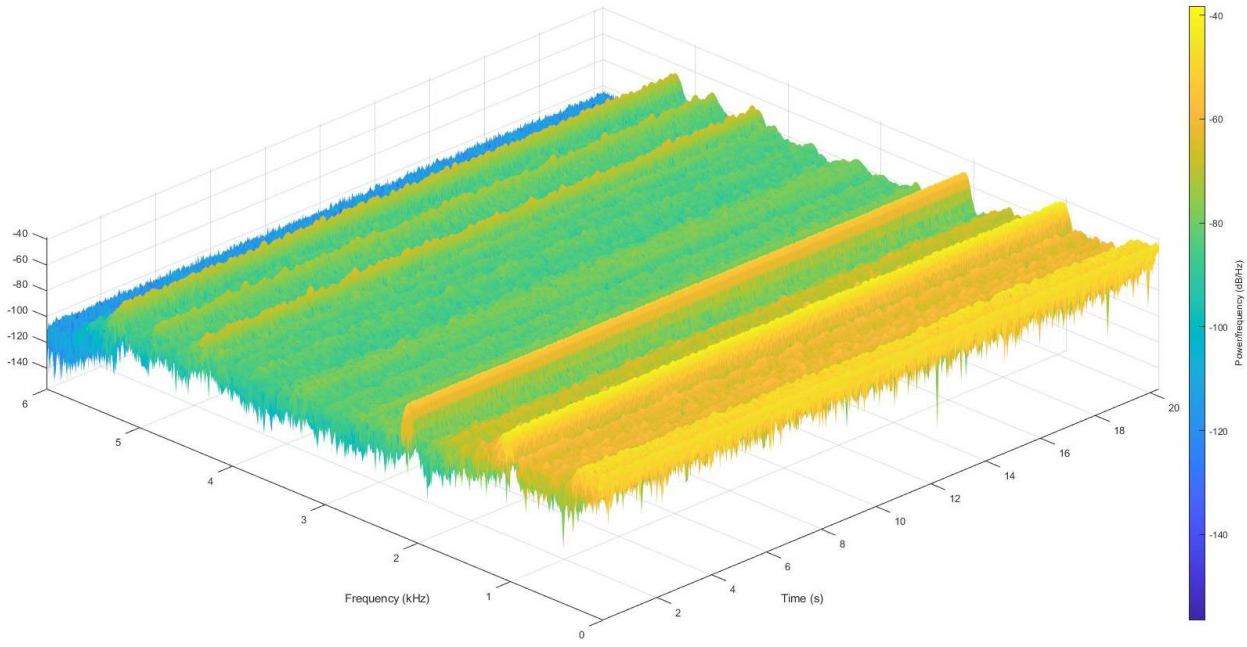

Ilustración 8: Kurtosis Espectral

Sin embargo, este proceso es muy costoso computacionalmente hablando. Existe una versión menos costosa, The Fast Kurtogram [21]. En la Ilustración 9 se muestra el resultado del Fast Kurtogram aplicado sobre el mismo conjunto de datos. Se obtiene la misma banda de frecuencia. El sacrificio de un menor coste computacional puede suponer en algunos casos resultados no tan eficientes como los del Kurtograma.

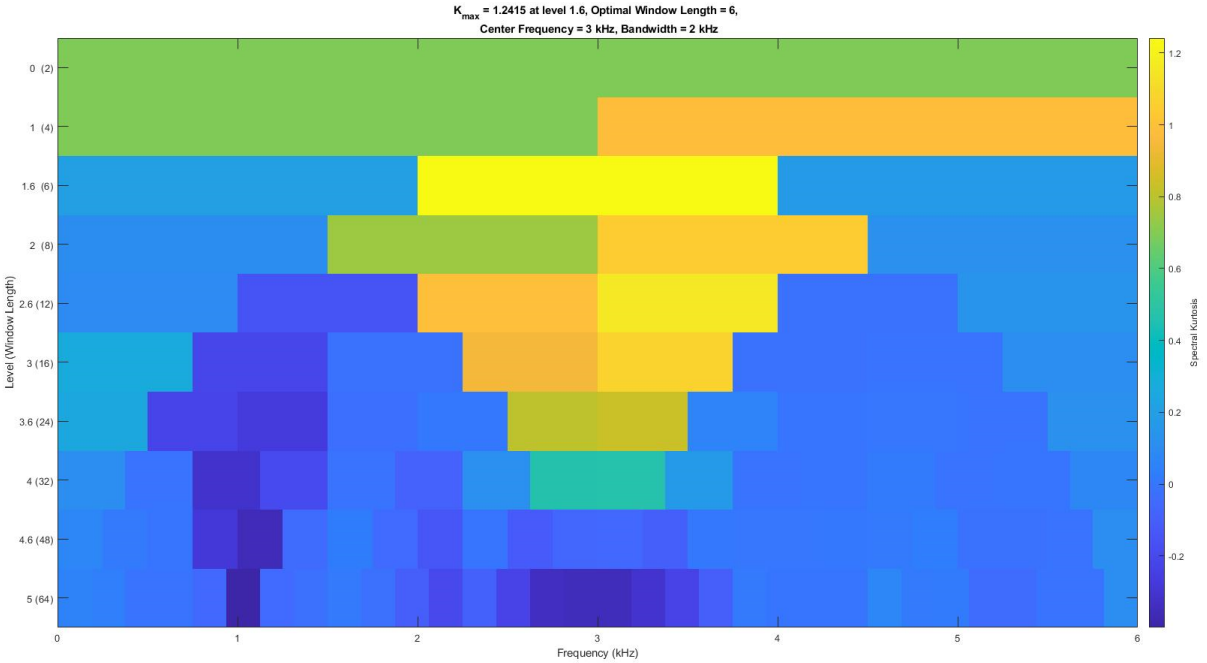

Ilustración 9: Kurtograma

Con las técnicas descritas anteriormente es posible detectar anomalías en las diferentes partes de un rodamiento según las frecuencias características presentes en el espectro. Sin embargo, la dificultad para detectar fallos en los rodamientos es diferente en función del elemento del rodamiento donde tenga lugar.

Cuando existe un defecto en la pista interior o en la pista exterior, la posibilidad de que el Envelope Analysis detecte un armónico en las frecuencias de fallo característica es mucho mayor que si el fallo está presente en las bolas del rodamiento.

En [22] se aplica PCA, una técnica de reducción de la dimensionalidad, al Envelope spectrum obtenido a partir de un conjunto de datos de vibraciones de un motor eléctrico para un funcionamiento sin fallo y para un funcionamiento con fallos en la pista interior, pista exterior y en las bolas.

En la Ilustración 10 obtenida de [22] se muestra una comparativa de la función de densidad de probabilidad bidimensional.

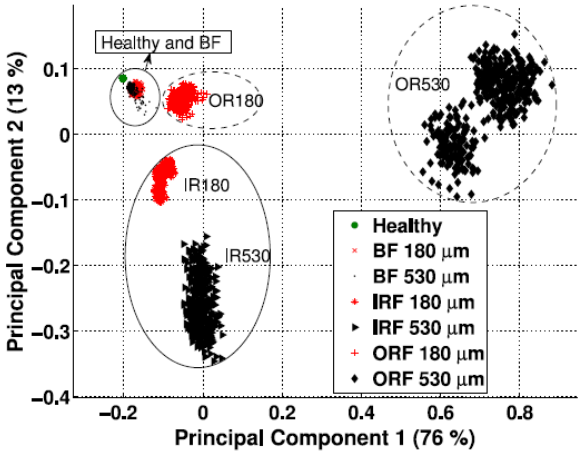

Ilustración 10: Comparativa PDF de fallos en rodamientos

Esta ilustración explica por qué la detección de defectos en las bolas es más compleja que en las pistas interiores y exteriores, puesto que las vibraciones generadas por un rodamiento sin ningún tipo de defecto y un rodamiento con un defecto en las bolas son muy similares.

En la literatura también se ha estudiado cómo simular las vibraciones producidas por motores que presentan defectos. Esto es espacialmente de interés para aquellas técnicas que necesitan conocer el comportamiento del sistema en los diferentes estados.

Para la simulación de los defectos localizados de distinguen 4 tipos de modelos a lo largo del tiempo [10]:

1. *Periodic impulse-train models*: Estos modelos simulan los defectos con la generación de impulsos con un periodo constante. El primer modelo para simular un defecto localizado en la pista interior de un rodamiento fue surgió en 1984 [23]. Las desventajas de este tipo de modelos son principalmente que no tiene en cuenta parámetros físicos de los rodamientos ni las variaciones producidas en la amplitud, forma, ancho o modulación al pasar por la zona de carga. En definitiva, son demasiado sintéticos.

2. *Quasi-periodic impulse-train models*: estos modelos son una evolución de los anteriores en los que se introducen fluctuaciones aleatorias debido al deslizamiento entre las pistas y las bolas del rodamiento. El estudio de este tipo de modelos generó el interés de los autores para introducir características físicas de los rodamientos dando paso a la creación del siguiente tipo de modelos.

3. *Nonlinear multi-body dynamic models*: estos modelos tratan de simular el comportamiento de rodamientos y elementos asociados como un conjunto de parámetros. En el contexto de los modelos mecánicos, los sistemas se simulan como un conjunto de masas rígidas unidas por resortes y amortiguadores. En el caso de los rodamientos, se consideran las pistas interior y exterior como masas

rígidas y el contacto entre las bolas y las pistas como resortes no lineales. Estos modelos no han sido capaces de solucionar el problema de la simulación del valor de la amplitud de la señal.

4. *Finite Element models*: este tipo de modelos se generan utilizando software comercial. Con este tipo de modelos se obtienen los mejores resultados minimizando el número de suposiciones necesarias en modelos anteriores. Estos modelos, pese a conseguir resultados más precisos, siguen presentando resultados dispares en cuanto a la amplitud.

Muchos autores han optado por utilizar conjuntos de datos públicos existentes para probar la validez y eficacia de las técnicas y algoritmos desarrollados. El conjunto de datos más conocido y utilizado es el CWRU dataset [24]. Este conjunto de datos fue generado por la Universidad Case de la Reserva Occidental de Cleveland (CWRU). Se realizaron experimentos en los cuales se recogieron datos sobre las aceleraciones producidas por un motor eléctrico en diferentes situaciones, variando el tipo de fallo, su severidad o la velocidad de giro del eje.

En concreto, se introdujeron rodamientos con fallos en la corona interior, en las bolas y en la pista exterior con diferentes diámetros desde las 0.007 pulgadas hasta las 0.04 pulgadas. Además, estos ensayos se realizaron variando la velocidad de giro y la carga.

Además del CWRU dataset, existen otros conjuntos de datos [25] los cuales se recogen en la Tabla 7:

Nombre	Método de fallo	Frecuencia de muestreo	Tipo de fallo
FEMTO-ST	Prueba de vida celerada	25.6 kHz	• 3 condiciones de operación • 6 learning datasets • 11 test datasets
IMS	Prueba de vida celerada	20 kHz	• Pista interior, pista exterior, bolas y jaula • 3 experimentos de funcionamiento hasta fallo
XJTU-SY	Prueba de vida celerada	25.6 kHz	• Pista interior, pista exterior, bolas y jaula • 3experimentos variando la carga
CWRU	Fallos artificiales	12kHz /48 kHz	• Pista interior, pista exterior, bolas y jaula • Variación del diámetro del fallo • Variación de la carga
MFPT	Prueba de vida celerada/ Fallos artificiales	97656Hz/48828Hz	• Pista interior y pista exterior • Variación de la carga • 3 experimentos de funcionamiento hasta fallo

Nombre	Método de fallo	Frecuencia de muestreo	Tipo de fallo
Padeborn	Fallos artificiales	64kHz	• 6 rodamientos sin daño • 12 rodamientos con fallos artificiales en la pista interior y exterior • 2 niveles de daño • 14 experimentos de funcionamiento hasta fallo

Tabla 7: Datasets públicos

5 Componentes

En este apartado se describen los componentes hardware y software utilizados en las diferentes capas de la arquitectura. El uso de estos componentes viene impuesto por Intermark.

5.1 Motor eléctrico

Con el objetivo de disponer de datos reales en cualquier momento de la implementación del sistema, se ha instalado un motor eléctrico sobre el cual sea posible colocar el sensor y monitorizar vibraciones.

En concreto, se ha utilizado un motor eléctrico de inducción trifásico Cemer modelo MYTE 711-4 con las siguientes características técnicas:

- Potencia: 025kW (0.33 CV)
- Velocidad: 1400 rpm
- Voltaje: 230V

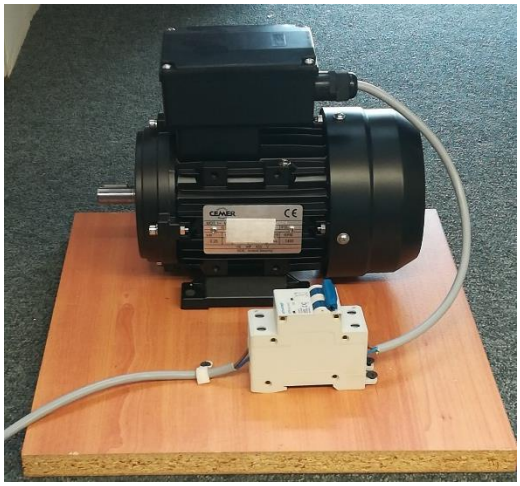

Ilustración 11: Instalación motor eléctrico

Para facilitar su uso y la recogida de datos se ha modificado para que se comporte como un motor monofásico girando a una velocidad constante y siempre en la misma dirección. Para ello, se le ha añadido un condensador como se muestra en la Ilustración 12.

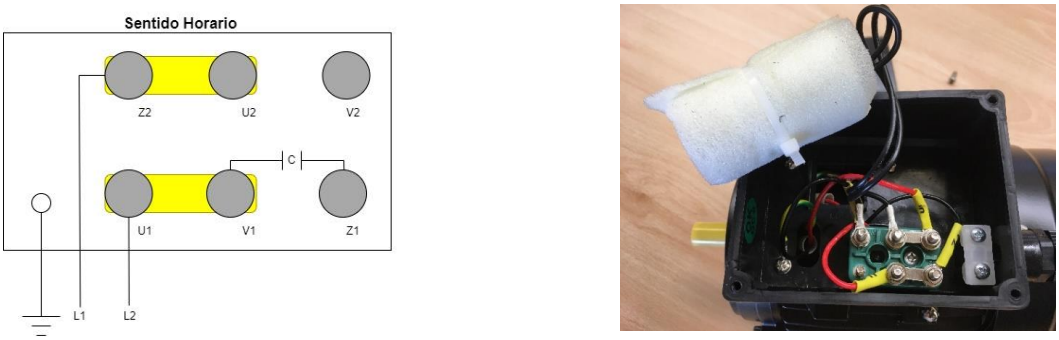

Ilustración 12: Conexionado motor eléctrico

Este modelo incorpora rodamientos 6202 2RS C3. Este tipo de rodamientos son rodamientos rígidos de bolas con obstrucciones de caucho a ambos lados y están lubricados de por vida de modo que no necesitan ningún tipo de mantenimiento.

Los detalles técnicos sobre estos rodamientos necesarios para calcular las frecuencias de fallo se recogen en la Tabla 8.

Elemento	Valor
Diámetro de la pista (D)	29.2 mm
Diámetro de las bolas (d)	11.6 mm
Número de bolas (n)	6 bolas
Ángulo de impacto (Φ)	0º

Tabla 8: Especificación técnica del rodamiento

5.2 Steval-MKSBOX1V1

El Steval-MKSBox1V1 [26] es un dispositivo multi sensor diseñado para el desarrollo aplicaciones IoT y aplicaciones para wearable devices. Además, proporciona un entorno de programación y desarrollo que permite la configuración y desarrollo de un firmware personalizado y capaz de adaptarse a las necesidades específicas de distintos proyectos.

Ilustración 13: Steval-MKSBOX1V1

Sus principales ventajas son la comunicación wireless a través del protocolo BLE y que incorpora múltiples sensores que permiten la captación de datos de diferentes magnitudes:

- STTS751: sensor de temperatura digital
- LSM6DSOX: sensor inercial 6-ejes
- LIS2DW12 y LIS3DHH: acelerómetros 3-ejes
- LIS2MDL: magnetómetro 3 ejes
- LPS22HH: sensor de presión
- MP23ABS1: sensor de sonido
- HTS221: sensor de humedad y temperatura

En el desarrollo del firmware de este proyecto se han utilizado los sensores que por defecto utilizan los proyectos base proporcionados por STMicroeletronics. En concreto, el LSM6DSOX como acelerómetro y el HTS221 como sensor de temperatura.

5.3 ST LINK V2

El ST LINK V2 es un “debugger” y “programmer” (depurador y programador) para las familias de microcontroladores de STM32 y STM8 que permite la descarga de ficheros binarios en los componentes desarrollados por STMicroelectronics. Su uso es necesario debido a la necesidad de desarrollar un firmware con funcionalidades específicas para el proyecto. De entre todos los elementos y cables de conexión, se han utilizado los indicados a continuación:

Ilustración 14: cables ST LINK

A. ST-LINK V2 depurador y programador
B. Cable estándar USB a mini-usb
C. Adaptador JTAG 20-10 pines
D. Cable de cinta con conector de 20 pines
E. Cable de cinta de 14-10 pines

El debugger tiene las siguientes interfaces y elementos:

Ilustración 15: Partes ST LINK

A. Interfaz para conector STM32 JTAG y SWD
B. Interfaz para conector STM8 SWIM
C. LED de actividad
D. Interfaz para cable mini USB

5.4 Raspberry Pi 4

Se utiliza un ordenador de placa reducida Raspberry Pi 4 Model B. Sus características principales se enumeran a continuación [27]:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 4GB LPDDR4-3200 SDRAM
- 2.4 GHz y 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 × puerto USB 3.0 y 2 puertos USB 2.0.
- 2 × puertos micro-HDMI
- Micro-SD slot para almacenamiento

En la Ilustración 16 [27] se muestra una RPi 4 y se señalan los principales componentes y características.

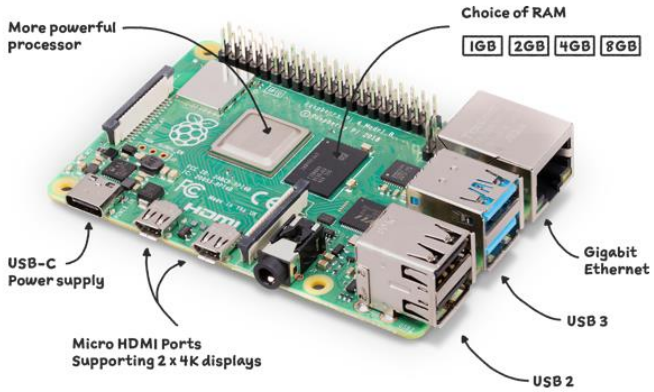

Ilustración 16: Raspberry Pi Model B

5.5 Thingsboard

Thingsboard [28] es una plataforma IoT de código abierto que permite la recolección de datos, el procesamiento, la visualización y la gestión de dispositivos y activos. Esta herramienta permite gestionar fácilmente las diferentes redes IoT desplegadas, los elementos que las forman y los datos recopilados. En la Ilustración 17 [28] se muestra la arquitectura principal de Thingsboard y el flujo de información desde el Edge.

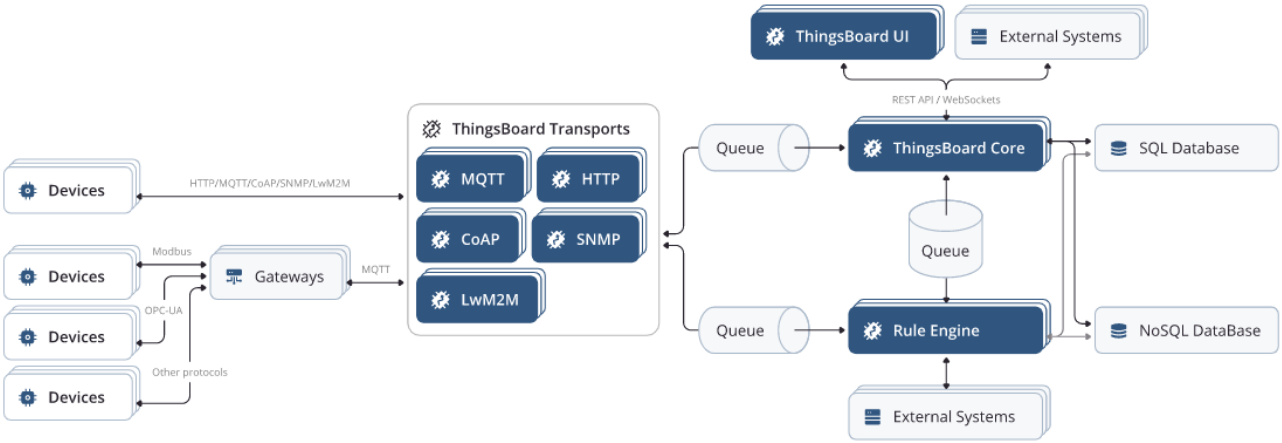

Ilustración 17: Flujo de informaciónThingsboard

Soporta “multitenancy” o multitenecia, es decir, que una única instancia de Thingsboard puede ejecutar múltiples instancias de aplicación. Las entidades que permite gestionar Thingsboard son las siguientes [28]:

- **Tenants o Inquilinos:** son una entidad comercial separada. Los inquilinos pueden ser individuos u organizaciones que poseen o producen dispositivos y activos. Cada inquilino puede tener varios usuarios administradores de inquilinos y diversos clientes, dispositivos y activos.

- **Clientes:** también son una entidad comercial separada, un individuo u organización que compra o usa dispositivos y/o activos de inquilinos. El Cliente puede tener múltiples usuarios y diversos dispositivos y/o activos.

- **Usuarios**: son aquellos individuos que tienen la capacidad de visualizar los paneles generados para mostrar las métricas recogidas.

- **Dispositivos:** todos aquellos dispositivos que estén situados en el borde y que recopilen información o realicen alguna acción. Por ejemplo, sensores. Pueden comunicarse con Thingsboard a través de los diferentes protocolos que soporta Thingsboard: MQTT, CoAP, HTTP, etc.

- **Activos o assets:** son entidades abstractas que pueden estar relacionadas con otros dispositivos o activos. Thingsboard permite jerarquizar entidades a través de relaciones entre activos. En la Ilustración 18 [28],se muestra la jerarquía de una organización que gestione la domótica de las casas de un barrio tendrá un activo por cada casa, y un activo barrio que contendrá todas las casas.

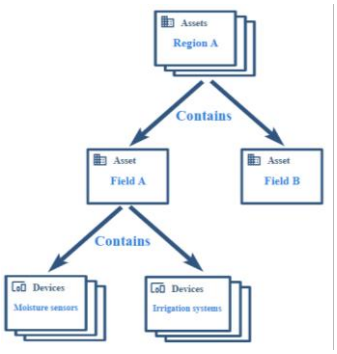

Ilustración 18: Ejemplo de relaciones entre activos

- **Alarmas:** eventos que identifican problemas con sus activos, dispositivos u otras entidades. Thingsboard permite establecer alarmas que se activan cuando ocurre un suceso que se quiere registrar. Por ejemplo, cuando un sensor falla al intentar actualizar la telemetría.

- **Tableros o dashboards**: elementos que permiten la visualización de los datos de los dispositivos y permiten para controlar dispositivos particulares a través de la interfaz de usuario.

- **Rule engine:** Thingsboard incluye un motor de reglas que permite establecer cadenas de reglas donde se describen acciones a realizar cuando ocurren eventos. Por ejemplo, se puede definir una regla que establezca que cada vez que un dispositivo publique una lectura, esta se almacene en la base de datos. Por defecto, existe una cadena de reglas que define las acciones más básicas. En la Ilustración 19 se puede ver la cadena de reglas que por defecto viene creada. En ella se define que hacer cuando un cliente actualiza sus atributos, cuando actualiza los datos (telemetría), cuando envía una llamada RPC (Remote Procedure Call) u otras acciones.

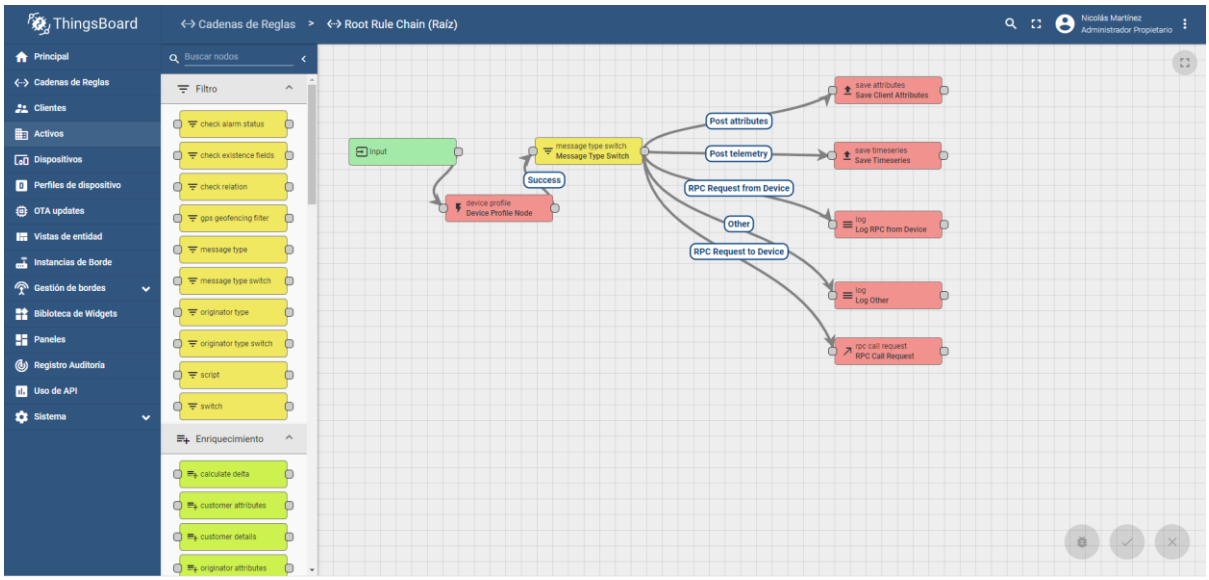

Ilustración 19: Ejemplo cadena de reglas Thingsboard

5.6 InfluxDB

Desde Intermark se requirió que el gateway tuviera la capacidad de almacenar las vibraciones captadas por el sensor, independientemente de que esta funcionalidad en un principio es de la capa del cloud. Además, debería utilizarse una base de datos de tipo NoSQL.

Por tanto, fue necesario realizar un estudio de las diferentes alternativas disponibles. La base de datos seleccionada ha sido InfluxDB por ser la única base de datos orientada al almacenamiento de series temporales capaz de ser instalada en Raspberry Pi.

InfluxDB es una base de datos no relacional para el almacenamiento de series temporales desarrollada por InfluxData. Las aplicaciones IoT tienden a utilizar este paradigma por su escalabilidad, consistencia, flexibilidad y su mejor rendimiento a la hora de trabajar con series temporales.

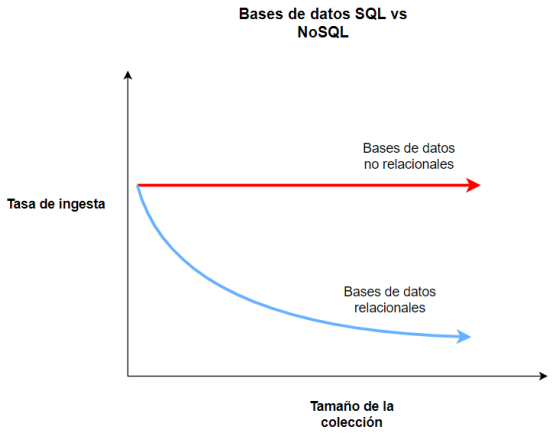

Ilustración 20: Gráfico de bases de datos SQL vs NoSQL

En InfluxDB, el contenedor principal de información es el "bucket". El "bucket" tiene una estructura de tabla y cada uno de los registros tiene una serie de columnas específicas. En la Ilustración 21 se muestra un ejemplo de bucket con posibles valores obtenido de la documentación de InfluxDB [29].

_time	_measurement	location	scientist	_field	_value
2019-08-18T00:00:00Z	census	klamath	anderson	bees	23
2019-08-18T00:00:00Z	census	portland	mullen	ants	30
2019-08-18T00:06:00Z	census	klamath	anderson	bees	28
2019-08-18T00:06:00Z	census	portland	mullen	ants	32

Ilustración 21: Ejemplo bucket InfluxDB

La primera columna "_time" almacena la marca de tiempo del registro. InfluxDB almacena este dato con formato Epoch con precisión de nanosegundos, aunque es formateado para su visualización. Es posible cambiar el formato en el que se muestra este valor con el comando "precision". El formato más común es "rfc3339", aunque pue mostrarse la marca de tiempo en horas, minutos, segundos, milisegundos, microsegundos o nanosegundos. Por defecto se muestra en nanosegundos.

En segundo lugar, la columna "_measurement" almacena el nombre del contenedor al que irán asociados los diferentes valores de los tags, "_fields" o campos y marcas de tiempo. Los campos están formados por una clave y un valor. En la tabla anterior, los campos "_field" y "_value" respectivamente.

Opcionalmente pueden definirse tags. Están formados por claves y valores al igual que los campos. En el ejemplo anterior las claves serían "location" y "scientist" y los valores las cadenas en esas columnas. La finalidad de definir tags es mejorar el rendimiento de las búsquedas. Los campos no están indexados mientras que los tags sí. Por lo tanto, cuando se introduce una query donde el filtro se establece con valores de los campos, la búsqueda debe recorrer la tabla al completo y mostrar solo aquellos valores que satisfagan

el filtro. Sin embargo, los tags están indexados de modo que las búsquedas son más eficientes.

InfluxDB soporta múltiples sistemas operativos. En este caso, se utilizará una Raspberry. Los requisitos para poder instalar InfluxDB en este dispositivo son los siguientes:

- Modelo Pi 4 o 400
- Sistema operativo con arquitectura de 64 bits

Sin embargo, al obtener la información de la arquitectura de la Raspberry se ha comprobado que la arquitectura de esta no es compatible con la versión 2.0 de InfluxDB por ser una arquitectura de 32 bits. Por tanto, se ha optado por utilizar una versión anterior. En concreto InfluxDB 1.8, la cual si es compatible con arquitecturas de 32 bits.

Una vez instalado ya es posible comenzar a operar con la base de datos. InfluxDB ofrece varias posibilidades para operar con la base de datos. En concreto, usar el plugin de Telegraf, “scrape data” de endpoints HTTP o utilizar API, CLI o librerías para crear clientes. Existen librerías en diversos lenguajes: Python, C#, Java, JavaScript …

6 Arquitectura del sistema

En este apartado se describe la arquitectura del sistema, la cual se rige según el paradigma del Fog Computing [30].

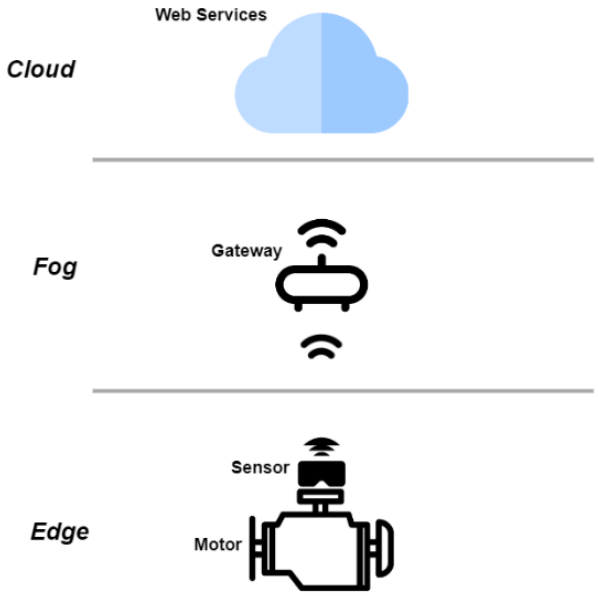

Ilustración 22: Arquitectura del sistema

6.1 Edge

Esta capa está formada por los dispositivos encargados de realizar la recolección de información sobre el proceso monitorizado, en este caso concreto, las vibraciones y temperatura de un motor eléctrico. Para este proyecto se ha utilizado un módulo multisensorial Steval-MKSBOX1V1 que realiza lecturas de vibraciones y temperatura.

El Steval- MKSBOX1V1 incorpora como método de comunicación wireless el protocolo Bluetooth Low Energy (BLE). Por tanto, este ha sido el protocolo de comunicación utilizado entre el Edge y la capa superior.

Las señales capturadas del proceso no son enviadas en crudo. El nodo multisensorial realiza un mínimo procesamiento de modo que los datos puedan ser enviados en paquetes BLE que tienen un tamaño máximo de 200 bytes.

6.2 Fog

Esta es la capa inmediatamente superior al Edge. Su funcionalidad principal es la de recibir información de los sensores, realizar el procesamiento necesario y encaminarla hacía la nube. El dispositivo utilizado como gateway ha sido la Raspberry Pi 4.

La comunicación con el Edge es a través del protocolo BLE. En el caso de este proyecto, solo se cuenta con un único nodo en el Edge. Si el número de nodos aumentara, el gateway tiene la capacidad de comunicarse con todos los nodos activos de forma simultánea.

Una vez se ha recibido la información del Edge, se realiza un procesado de esta. En el caso de las vibraciones, el gateway es el encargado realizar la transformación de la señal al dominio de la frecuencia para facilitar su análisis. Una vez realizado, estos datos son enviados al Cloud para su almacenamiento y visualización.

Para el envío de información hacia la nube, se ha utilizado el protocolo MQTT, que es un protocolo de mensajería estándar orientado al envío de información de dispositivos IoT basado en el protocolo de mensajería publicación-suscripción. MQTT funciona sobre el protocolo de red TCP, por lo tanto, es necesario que el gateway tenga conexión a Internet para que se complete el flujo de información.

Desde Intermark se solicitó que el gateway tuviera la capacidad de almacenamiento de la señal previa al procesamiento. Para cumplir este requisito, también se ha implementado la funcionalidad de almacenamiento en el gateway. Antes de transformar la señal al dominio de la frecuencia, se almacenan las vibraciones en una base de datos de tipo NoSQL, en concreto se utiliza InfluxDB.

6.3 Cloud

Esta capa es la encargada de proporcionar los servicios de almacenamiento y visualización de la información. En este nivel, existen diferentes alternativas para la implementación del cloud:

- **Nube pública:** propiedad de un proveedor externo de servicios, que proporciona servicios informáticos a través de internet. Accesibles para cualquier usuario.

- **Nube privada:** propiedad de una empresa u organización exclusivamente. Solo accesible para algunos usuarios.

- **Nube híbrida:** combina nube pública y nube privada.

En este proyecto se ha optado por una nube privada por ser la opción más económica. Se ha desplegado una instancia de Thingsboard alojada en un servidor proporcionado por la Universidad de Oviedo.

Thingsboard cada vez que recibe un dato, lo almacena en su base de datos de forma persistente de modo que todos los usuarios que accedan a la plataforma pueden visualizarlos a través de los diferentes paneles implementados.

En la Ilustración 23 se muestra un resumen de los componentes que forman cada una de las capas de la arquitectura y los protocolos de comunicación utilizados para la comunicación entre cada una de las capas.

Sistema
IIoT para monitorización de motores eléctricos y detección de anomalías mediante el análisis de vibraciones en rodamientos

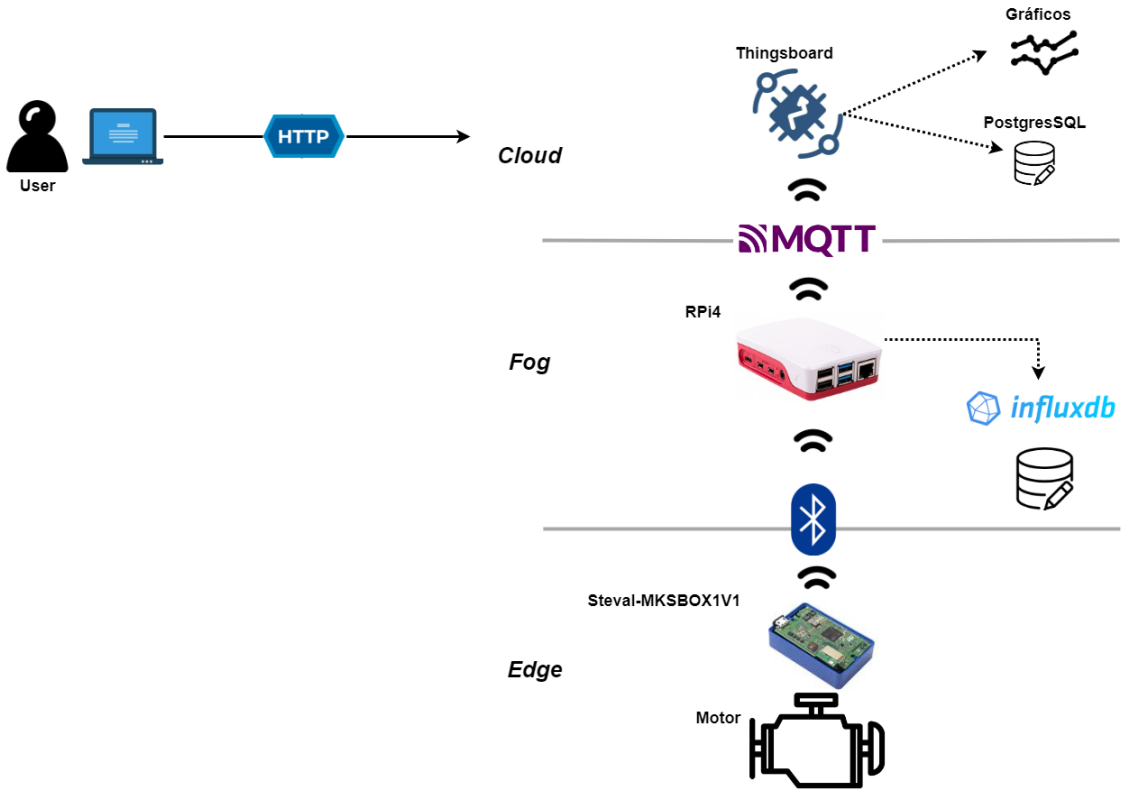

Ilustración 23: Arquitectura Final

7 Configuraciones para versión del firmware FP-SNS-ALLMEMS2

En este apartado se describen las diferentes tareas que han sido necesarias para poder implementar la primera versión operativa del *Sistema IIoT para monitorización de motores eléctricos y detección de anomalías mediante el análisis de vibraciones en rodamientos*.

El primer planteamiento ha sido construir un sistema capaz de monitorizar vibraciones para posteriormente realizar ampliaciones y conseguir la funcionalidad deseada. En los siguientes apartados se hace un recorrido de la configuración de herramientas, dispositivos u otros elementos necesarios para transportar los datos leídos en el Edge hacia el Cloud.

En este apartado se recogen las tareas más relevantes si se desea realizar desde cero la implementación o tener el contexto necesario para poder realizar mejoras o cambios.

7.1 Configuración Edge device

La tarea de captación de datos del motor eléctrico monitorizado es realizada por un dispositivo fabricados por STMicroelectronics. En concreto, se ha utilizado el STEVAL-MKSBOX1V1.

Ha sido necesario realizar modificaciones en el firmware de este dispositivo según las necesidades. Para ello, se ha hecho uso de las herramientas del entorno de desarrollo proporcionadas por el fabricante para el desarrollo de un firmware adaptado. El proceso realizado para la instalación y configuración de las herramientas utilizadas se describe a continuación.

7.1.1 Configuración entorno desarrollo

7.1.1.1 STM32 Cube IDE

Es una herramienta de desarrollo que permite la generación y compilación de proyectos con la configuración del sensor. Con el uso de esta herramienta es posible obtener un archivo binario que pueda descargarse en la placa del nodo, adaptando su funcionamiento a las necesidades del proyecto.

Esta herramienta se encuentra accesible en la web de STMicroelectronics [31], en el apartado “Tools and software” es posible descargar un archivo ejecutable con el que instalar la herramienta.

Al lanzar el programa, este solicita la ruta la ruta donde del espacio de trabajo. En este caso se ha escogido la ruta por defecto. Es posible tener varios espacios de trabajo es un mismo equipo con diferentes versiones de los proyectos.

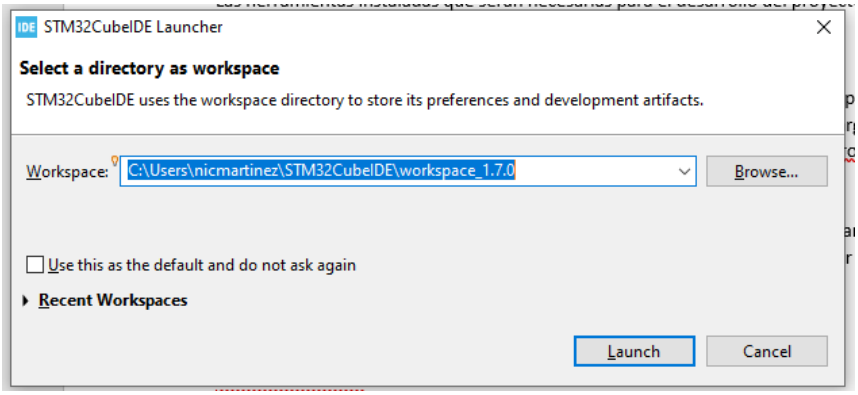

Ilustración 24: IDE Workspace path

A continuación, es necesario crear un proyecto o importar uno ya existente. En este caso, se parte de un proyecto existente.

Para importar un proyecto se debe seleccionar la opción “import” del menú File.

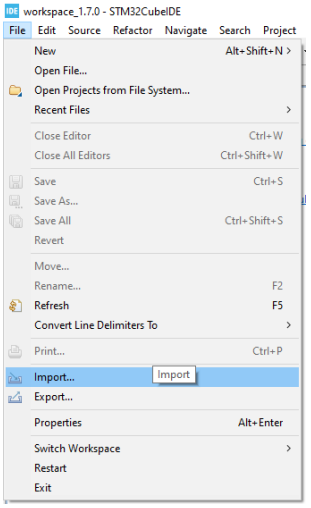

Ilustración 25: IDE Import project

Aparece una ventana emergente donde se elige la opción “Exisiting projects into workspace”.

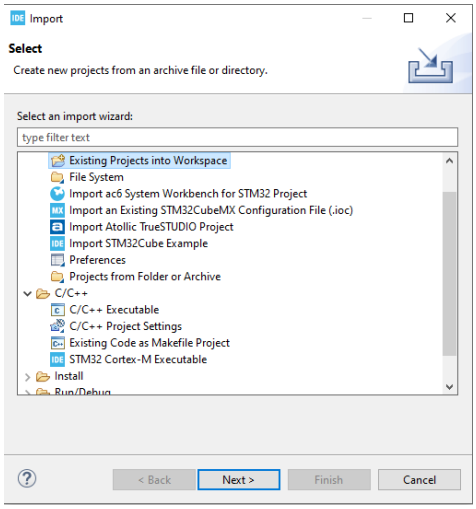

Ilustración 26: IDE Existing projects

En último lugar, se debe indicar la ruta del proyecto que se desea importar:

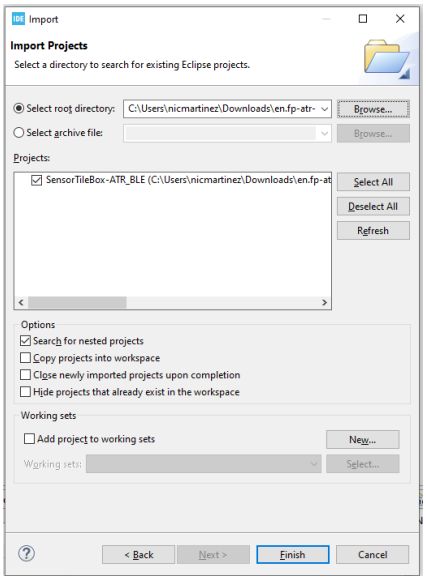

Ilustración 27: IDE ruta del proyecto

Al finalizar el proceso descrito en este apartado, el proyecto se habrá importado correctamente y es posible comenzar a editar el código de los diferentes ficheros que lo componen. La estructura del proyecto se muestra en la vista “Project Explorer”.

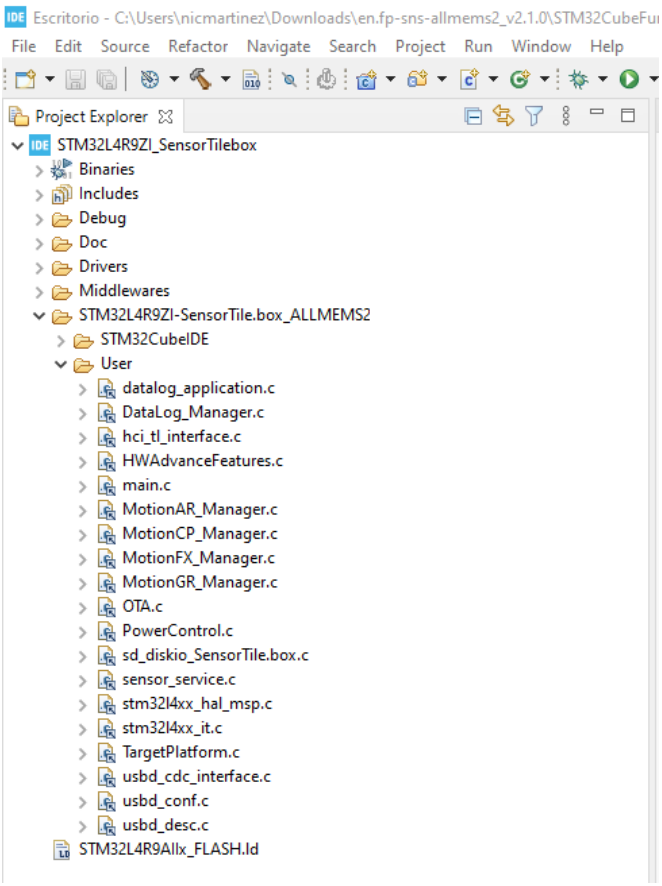

Ilustración 28: IDE Proyecto importado

Si esta vista no está configurada por defecto, puede activarse desde la barra de herramientas, en el menú Window.

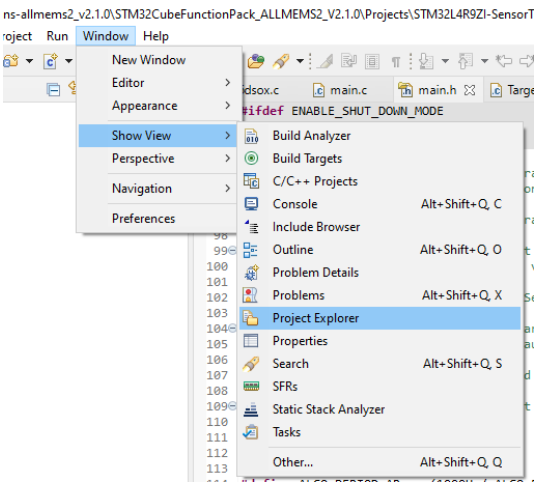

Ilustración 29: IDE Activación vista Project Explorer

7.1.1.2 STM32 CubeProgrammer

Esta herramienta proporciona un entorno para leer, escribir y verificar la memoria de dispositivos como el STEVAL-MKSBOX1V1. Se ha utilizado para descargar en el nodo el archivo generado al compilar un proyecto.

Para su instalación, únicamente es necesario ejecutar el archivo ejecutable disponible en la web de STMicroelectronics [32]:

Para descargar un archivo es necesario conectar el sensor al equipo donde se ejecuta la herramienta y establecer una conexión a través del puerto SWD. Para realizar la conexión es necesario un elemento intermediario que gestione la comunicación. Este elemento es el ST-Link V2.

7.1.1.3 Configuración sensor STEVAL-MKSVOX1V1

En primer lugar, se ha verificado el funcionamiento del STEVAL-MKSVOX1V1. Para ello, se establece una conexión entre el dispositivo y un teléfono móvil que permita obtener valores de los diferentes sensores que se utilizarán más adelante. El sensor ya cuenta con un firmware de fábrica preparado para conectarse mediante BLE con otros dispositivos. Por otro lado, en el teléfono móvil es necesario instalar la aplicación STBLESesnsor APP disponible en Google Play.

Como se muestra en la Ilustración 30, el sensor aparece como disponible para conexión.

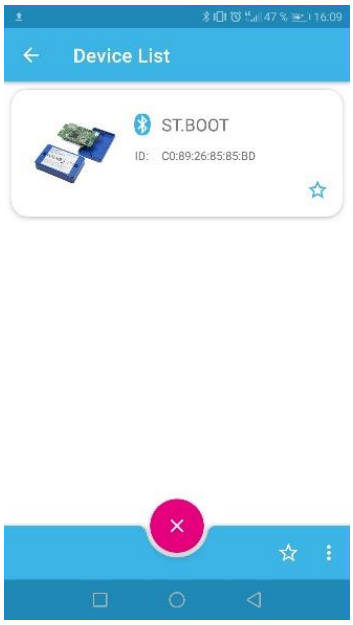

Ilustración 30: ST BLESensor App Lista dispositivos

Al establecer la conexión, la aplicación nos indica que es necesario realizar una actualización del firmware del sensor necesaria para su correcto funcionamiento. Este

firmware será sustituido en un futuro por un firmware adaptado a las necesidades del proyecto.

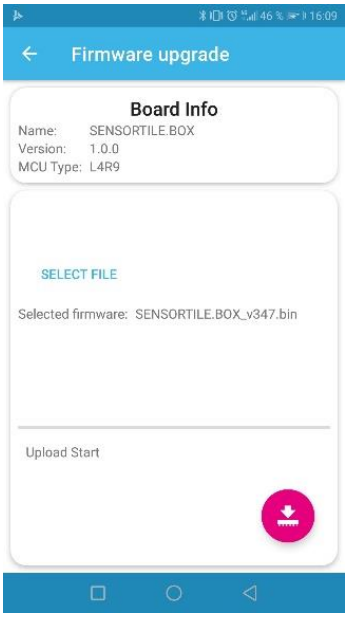

Ilustración 31: ST BLESensor App Firmware upgrade

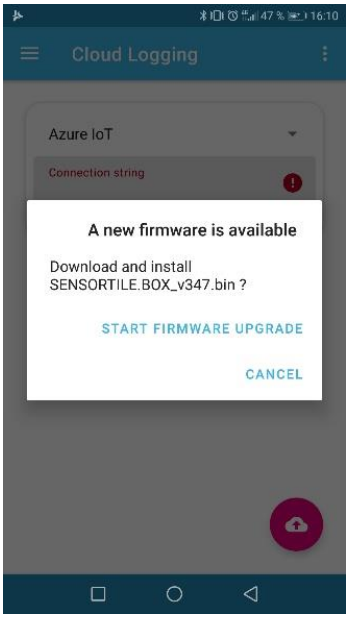

Ilustración 32: ST BLESensor App Firmware available

Una vez finalizada la instalación del firmware en el sensor, es posible comenzar a leer datos. El firmware instalado por defecto en el sensor únicamente permite acceder a los datos de los sensores ambientales. Por tanto, ha sido necesario desarrollar un firmware que permita obtener datos de los sensores inerciales además de los sensores ambientales.

7.1.2 Desarrollo del firmware

Para el estudio de las vibraciones que se realiza en este proyecto, es necesario medir aceleraciones a una alta frecuencia, más elevada que la frecuencia de 2Hz que por defecto utiliza el sensor. Por este motivo, ha sido necesario desarrollar un firmware donde se aumente la frecuencia de muestreo de los sensores inerciales y sea posible modificar las frecuencias de muestreo de los sensores ambientales para adecuarlas a necesidades futuras.

Desde STMicroelectronics se proporcionan diversos proyectos de ejemplo que sirven como base para el desarrollo de un firmware donde sea posible modificar los parámetros que sea necesario. En concreto, el paquete de funciones FP-SNS-ALLMEMS2 [33].

Se trata de un conjunto de proyectos para varios dispositivos de STMicroelectronics que permiten conectar estos dispositivos mediante el protocolo BLE y acceder a todos los datos de sus sensores. Además, este proyecto proporciona funciones de ultra-bajo coste que permiten disminuir el consumo de energía. En la Ilustración 33 [34] se muestra un resumen de la estructura de este paquete de funciones.

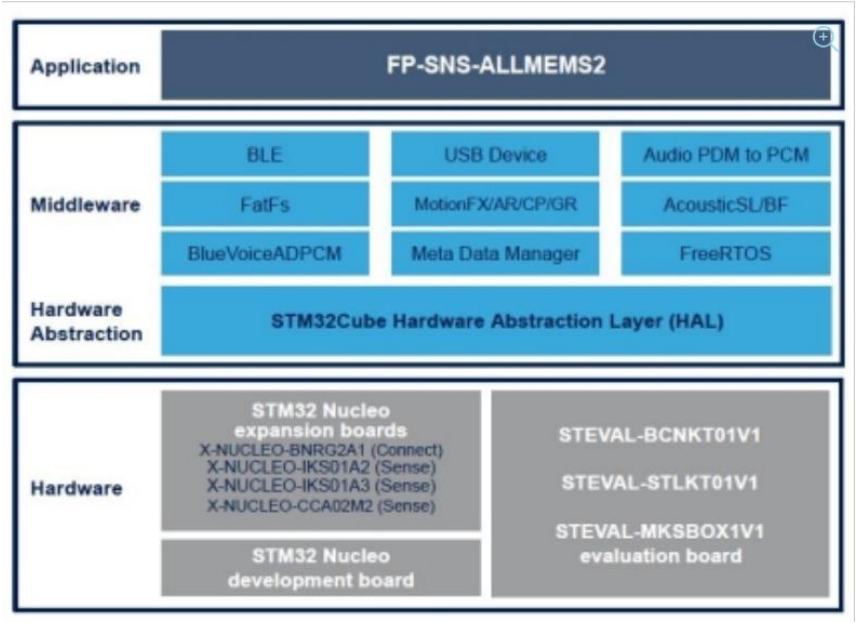

Ilustración 33: FP-SNS-ALLMEMS

En primer lugar, debe importarse el proyecto como se describió en el apartado **STM32 Cube IDE.** Una vez importado, se puede observar cómo los desarrolladores de este proyecto han agrupado los diferentes ficheros que componen el proyecto en la vista "Project Explorer".

Los directorios que componen la estructura del proyecto son los siguientes:

- **Doc**: contiene información relativa al proyecto.
- **Drivers**: contine los diferentes drivers necesarios para acceder a los componentes hardware de la placa.
- **Middlewares**: contine librerías y protocolos para el manejo de comunicaciones, conexiones USB, almacenamiento en tarjeta SD …
- **STM32L4R9ZI-SensorTile.box_ALLMEMS2**: contiene el código fuente de la aplicación desarrollada. En este directorio se encuentran los ficheros que ha sido necesario modificar para la creación del firmware adaptado al proyecto. Este directorio tendrá siempre el nombre de la aplicación elegida.

Los ficheros que se han modificado se encuentran en los directorios "Drivers" y "STM32L4R9ZI-SensorTile.box_ALLMEMS2\User".

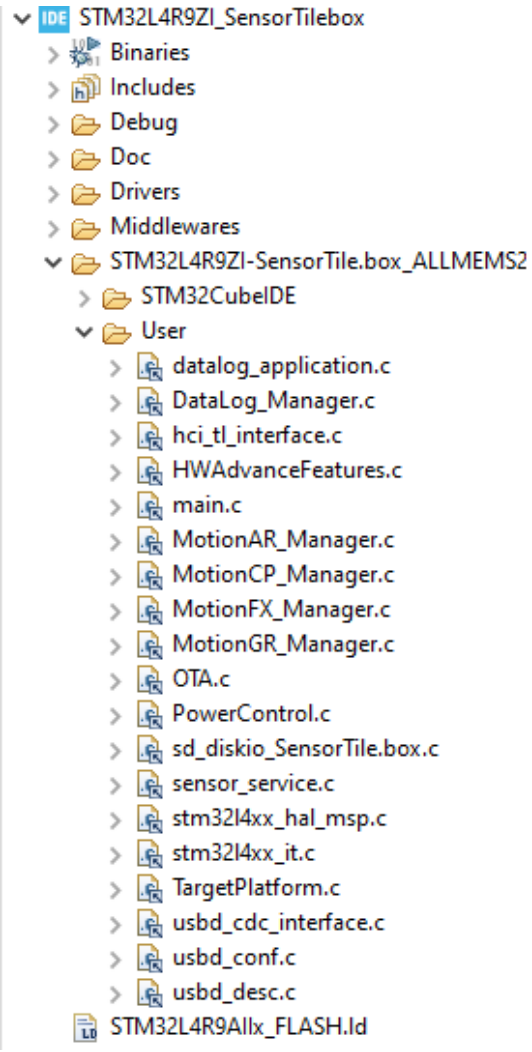

Ilustración 34: Estructura del proyecto

En el primero se encuentran los drivers de los diferentes sensores que incorpora la placa y donde es posible modificar algunos parámetros de funcionamiento como la frecuencia de muestreo de estos.

En el segundo directorio se encuentra el fichero main.c, donde se desarrolla el código principal del programa.

Como se ha comentado anteriormente, por defecto la frecuencia de muestreo de los sensores inerciales es por defecto 2Hz. El acelerómetro de este dispositivo puede recoger datos con una frecuencia de hasta 5 kHz.

Para aumentar esta frecuencia es necesario realizar 2 modificaciones. En primer lugar, modificar el “Output Data Rate” (ODR) del acelerómetro. El ODR define la frecuencia con la que el sensor proporciona un nuevo dato. Este parámetro se encuentra definido en el fichero “./Drivers/BSP/Components/lsm6dsox.c”, que contiene los drivers necesarios para leer los datos recogidos por este sensor.

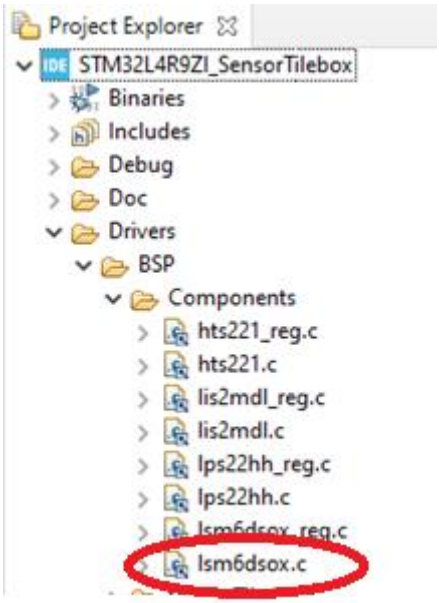

Ilustración 35: Fichero lsm6dsox.c

La configuración inicial establece que el ODR del sensor es de 104Hz.

```
/* Select default output data rate. */
pObj->acc_odr = LSM6DSOX_XL_ODR_104Hz;
```

Ilustración 36: Default ODR

Modificando el valor de este objeto es posible modifica el ODR del acelerómetro. Por defecto, en el proyecto se define una estructura de datos donde se recogen las frecuencias en las que el sensor puede trabajar. Esta estructura se define en el fichero ..\Drivers\BSP\Components\lsm6dsox\lsm6dsox_reg.h.

```
typedef enum {
  LSM6DSOX_XL_ODR_OFF    = 0,
  LSM6DSOX_XL_ODR_12Hz5  = 1,
  LSM6DSOX_XL_ODR_26Hz   = 2,
  LSM6DSOX_XL_ODR_52Hz   = 3,
  LSM6DSOX_XL_ODR_104Hz  = 4,
  LSM6DSOX_XL_ODR_208Hz  = 5,
  LSM6DSOX_XL_ODR_417Hz  = 6,
  LSM6DSOX_XL_ODR_833Hz  = 7,
  LSM6DSOX_XL_ODR_1667Hz = 8,
  LSM6DSOX_XL_ODR_3333Hz = 9,
  LSM6DSOX_XL_ODR_6667Hz = 10,
  LSM6DSOX_XL_ODR_6Hz5   = 11, /* (low power only) */
} lsm6dsox_odr_xl_t;
```

Ilustración 37: Posibles valores ODR

```
/* Select default output data rate. */
pObj->acc_odr = LSM6DSOX_XL_ODR_6667Hz;
```

Ilustración 38: Nuevo valor ODR

En segundo lugar, es necesario modificar la frecuencia con la que el sensor comunica los datos leídos por el sensor mediante BLE. Este parámetro se encuentra definido en el fichero ".\Projects\STM32L4R9ZI-SensorTile.box\Applications\ALLMEMS2\Inc\main.h". Por defecto, el sensor transmite un nuevo dato de los sensores inerciales al dispositivo con el que se encuentren conectado por BLE cada 5000 ms.

```
//10kHz/20  For Acc/Gyro/Mag@20Hz
#define DEFAULT_uhCCR4_Val  500
```

Ilustración 39: Frecuencia muestreo sensores inerciales

El valor de este parámetro debe reducirse al máximo para permitir al dispositivo que se comunique con el sensor datos con una frecuencia de muestreo similar a la del acelerómetro. Para conseguir que el sensor comunique los datos con una frecuencia de 5 kHz, debe colocarse en este parámetro el valor 2.

```
//10kHz/20  For Acc/Gyro/Mag@20Hz
//#define DEFAULT_uhCCR4_Val  500
#define DEFAULT_uhCCR4_Val  2
```

Ilustración 40: Frecuencia de muestro sensores inerciales modificada

Este proceso ha de realizarse también en todos aquellos sensores que se utilicen, teniendo en cuenta que los valores de ODR son diferentes para cada sensor. Además, el parámetro que establece la frecuencia con la que se envía los datos de los sensores ambientales se encuentra definido en otro lugar diferente al de los sensores inerciales. En este caso, se ha definido en el fichero ".\Projects\STM32L4R9ZI-SensorTile.box\Applications\ALLMEMS2\Inc_config.h".

```
/* Define the default transmission interval for enviromental sensors */
#define DEFAULT_ENV_PERIOD 500
```

Ilustración 41: Frecuencia muestreo sensores ambientales

En la primera versión del firmware únicamente se han modificado los parámetros relacionados con los sensores inerciales para aumentar la frecuencia de muestro al máximo posible y observar el comportamiento del sensor y del gateway trabajando en esta frecuencia de muestreo.

Por último, es necesario compilar el proyecto y generar un fichero binario. Esto puede realizarse desde el IDE. Este fichero permite descargar en la placa del sensor la aplicación desarrollada con todas las configuraciones y funcionalidades implementadas. En la Ilustración 42 se muestra el botón "Build All" de la barra de herramientas que permite generar el fichero binario.

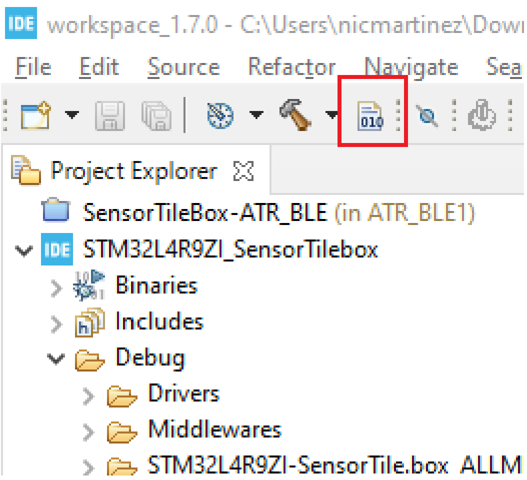

Ilustración 42: IDE botón: Build All

Si el proceso se ha realizado con éxito, en la consola del IDE se muestra el siguiente mensaje:

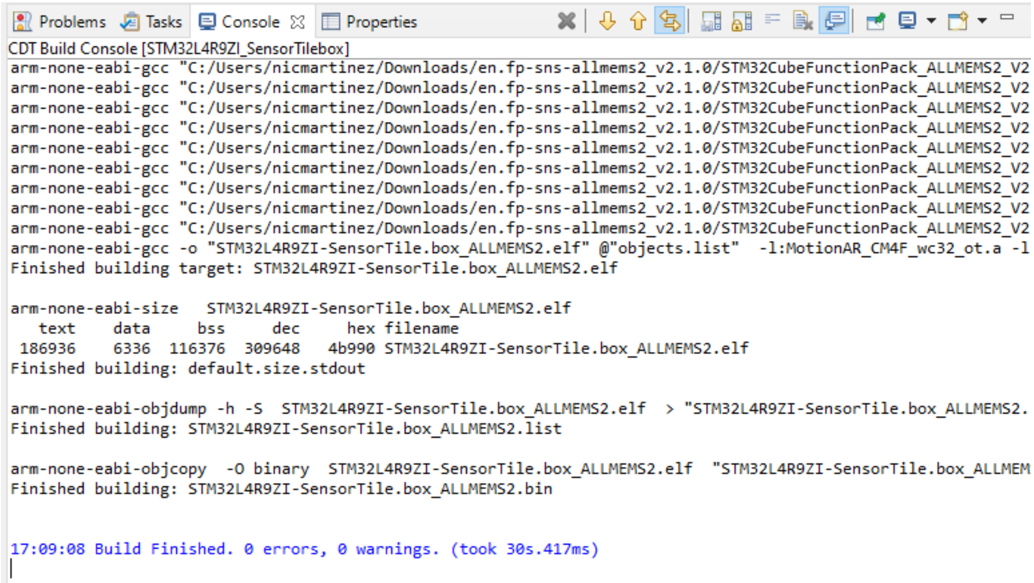

Ilustración 43: IDE Mensaje consola para Build All

7.1.3 Actualización del firmware

El último paso antes de completar la fase de configuración de sensor es descargar el firmware desarrollado en la placa del sensor para comenzar la captura de datos con una frecuencia de muestreo específica. Para ello se ha hecho uso de la herramienta STM32CubeProgrammer y el ST-Link V2.

En primer lugar, se debe conectar el sensor al ordenador desde el que se ha compilado el proyecto y que contiene el fichero binario que será descargado en el sensor. Esta conexión debe ser a través del ST-Link V2 tal y como se muestra en la Ilustración 44.

Ilustración 44: Conexionado placa y debugger

A continuación, desde el menú “Errasing and programming”, se indica la ruta del archivo binario generado, la dirección de memoria desde la que se va a escribir y se inicia el proceso con el botón “Start Programming”.

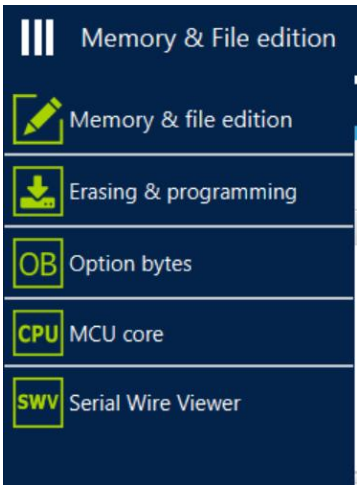

Ilustración 45: Menú Cube Programmer

Los parámetros configurados en la herramienta para establecer la conexión entre el equipo y el sensor han de ser los mostrados en la Ilustración 46.

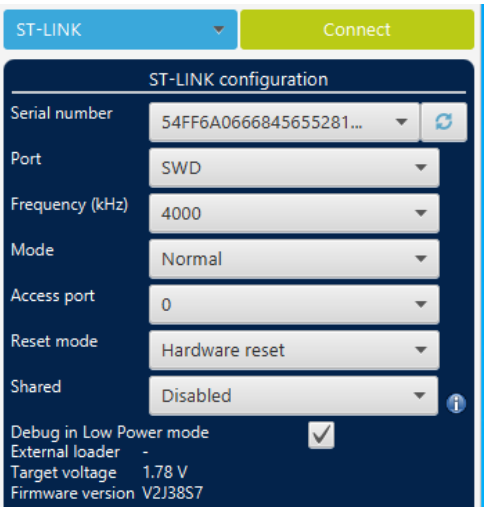

Ilustración 46: Configuración Cube Programmer

Si la conexión se establece éxito, se debe seleccionar la ruta del archivo binario generado. En el caso de los proyectos proporcionados por STMicroelectronics y compilados utilizando el STMCube IDE el fichero binario se encuentra en:
"en.fp-sns-allmems2_v2.1.0\STM32CubeFunctionPack_ALLMEMS2_V2.1.0\Projects\STM32L4R9ZI-SensorTile.box\Applications\ALLMEMS2\STM32CubeIDE\STM32L4R9ZI_SensorTilebox\Debug"

La dirección física de memoria en la que se comience a descargar el fichero ha de ser la dirección "**0x08004000**". En los casos en los que el propio proyecto contenga también un bootloader, la dirección física ha de ser la dirección "0x08000000".

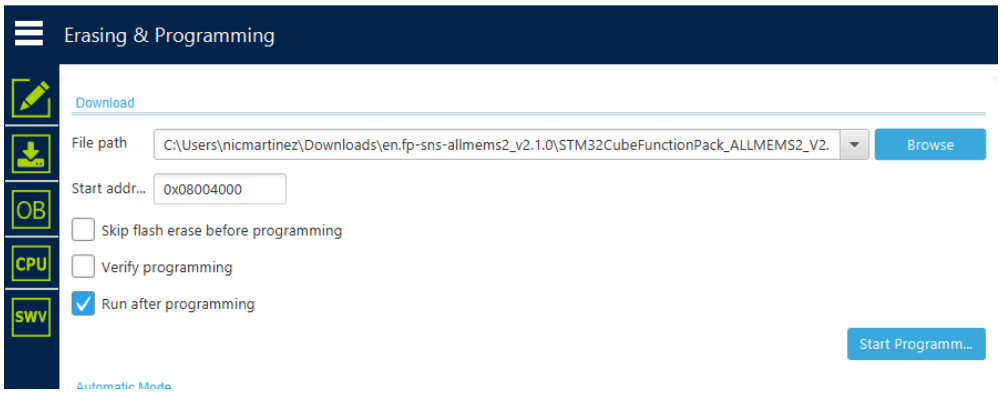

Ilustración 47: Cube Programmer- Erasing and Programming

En último lugar, para iniciar el proceso de descarga del firmware en la placa se debe pulsar el botón "Start Programming". Si la operación se realiza con éxito, se mostrará el siguiente cuadro de diálogo:

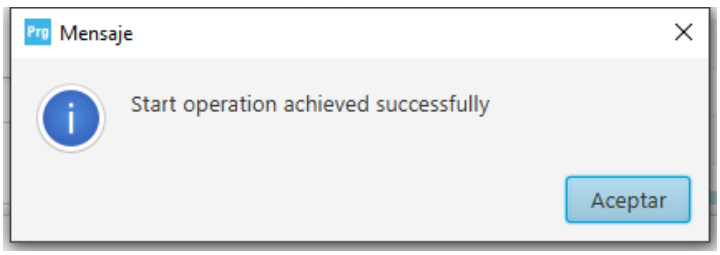

Ilustración 48: Mensaje de operación exitosa Cube Programmer

7.2 Gateway

Una vez que se ha conseguido captar las señales de campo, es necesario configurar un Gateway que sea capaz de leer la información captada por el sensor y la encamine hacia la nube. Para desempeñar esta funcionalidad se ha utilizado una Raspberry Pi 4. El protocolo de comunicación entre ambos dispositivos es el protocolo BLE.

En la primera etapa del proyecto se ha desarrollado un script que permita la conexión con el sensor y la obtención de los datos del acelerómetro, el magnetómetro y el sensor de temperatura. Este script se apoya en la librería Bluepy que permite realizar conexiones BLE mediante lenguaje Python.

Para poder realizar la conexión es necesario conocer la dirección MAC que identifica al sensor del resto de dispositivos bluetooth. Es posible realizar un escaneado de dispositivos donde se muestran sus direcciones MAC entre otra información. Este escaneado se realiza con el comando "blescan" importado al descargar la librería Bluepy.

En la Ilustración 49 se muestra un ejemplo de un escaneo realizado donde se aprecia la dirección MAC del sensor.

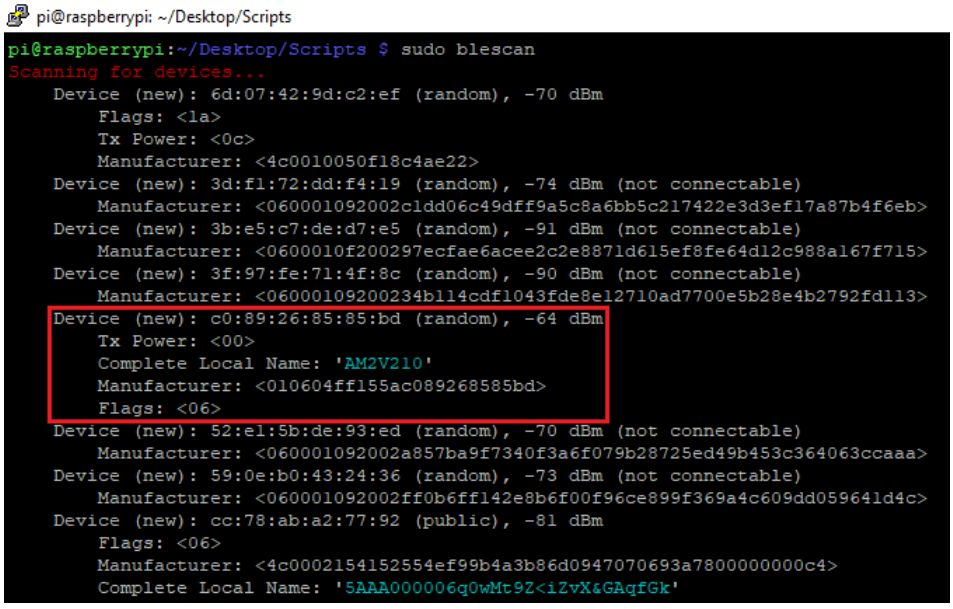

Ilustración 49: Salida comando blescan

Una vez conocida la dirección MAC del dispositivo, es posible establecer una conexión y acceder a los datos inerciales y ambientales que el mismo exporta. Tanto la conexión con el sensor como el filtrado de la información recibida se ha realizado en el fichero "lecturaAccTempV1.py". Este fichero obtiene del sensor las lecturas del acelerómetro, el magnetómetro y el sensor de temperatura, y muestra por pantalla estos valores. En la Ilustración 50 se muestra la salida obtenida al ejecutar el fichero.

```
pi@raspberrypi:~/Desktop/Scripts $ sudo python3 lecturaAccTempV1.py
Conectando con dispositivo c0:89:26:85:85:bd

Conectado
Notificación recibida de los sensores ambientales

Temperatura sensor 1: 31.2 °C
Notificación recibida de los sensores inerciales


datosAcc X: -509


datosAcc Y: 823


datosAcc Z: 926


datosMag X: -508


datosMag Y: 820


datosMag Z: 921
```

Ilustración 50: Salida fichero "lecturaAccTempV1.py"

7.2.1 Descripción Script

En este apartado se pretende describir el funcionamiento del fichero "lecturaAccTempV1.py" desarrollado para la conexión mediante BLE debido a su complejidad, de modo que sea posible su uso en futuros proyectos con dispositivos de la compañía STMicroelectronics. El desarrollo de este script se apoya en el SDK Bluepy.

El intercambio de información mediante BLE se define en GATT (Generic Attribute Profile), donde se establece como se organiza la información trasmitida mediante este protocolo. Los dispositivos Bluetooth exponen al exterior la información ordenada en objetos anidados de alto nivel llamados perfiles GATT. Los perfiles pueden estar definidos por Bluetooth SIG o por el fabricante del dispositivo. A su vez, los perfiles se dividen en servicios, que se identifican con un UUID (universally unique identifier) de 128 bits. Cada servicio puede estar formado por múltiples características. Las características son las que realmente contienen el dato que se transmite. En la Ilustración 51 [35] se muestra un esquema de cómo se organizan los perfiles BLE.

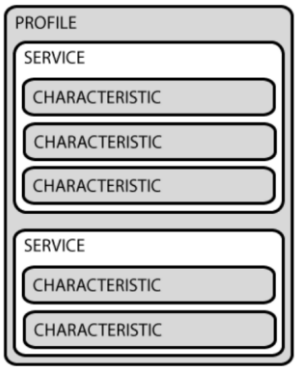

Ilustración 51: Perfil BLE

En el caso del Steval-MKSBOX1V1, las lecturas de los sensores incorporados se encuentran en el servicio 00000000-0001-11e1-ac36-0002a5d5c51b. Dentro de este servicio, existen 11 características que contiene información relativa a lecturas de sensores, nivel de batería u otros [36].

Para acceder desde el gateway a esta información, en primer lugar, se establece conexión entre el sensor y el gateway. Para ello se crea un objeto de tipo Peripheral tal como se muestra a continuación:

```
p = btle.Peripheral(deviceID,btle.ADDR_TYPE_RANDOM)
```

Ilustración 52: Conexión con dispositivo mediante BLE

El primer parámetro, deviceID, contiene una cadena de texto con la dirección MAC del sensor. El segundo parámetro es una constante que define el tipo de dispositivo. Los dispositivos pueden ser Random o public.

En segundo lugar, se indica al objeto de tipo Peripheral que función debe de ejecutar cuando se recibe una notificación por parte del sensor. Esta acción se realiza con la siguiente línea de código:

```
p.setDelegate(MyDelegate())
```

Ilustración 53: Asignación de un manejador

MyDelegate es un objeto de la clase DefaultDelegate donde se debe definir un constructor y la función handleNotification() que se ejecuta cada vez que llega una nueva notificación. El parámetro cHandle es un número que permite identificar la característica de la que se ha recibido una notificación (sensores ambientales, sensores inerciales …). Por otro lado, data contiene la información recibida en bytes.

En la Ilustración 54 se muestra el ejemplo de cómo se obtienen los datos de las aceleraciones en el eje X.

```python
#Clase MyDelegate
class MyDelegate(btle.DefaultDelegate):
    def __init__(self):

        btle.DefaultDelegate.__init__(self)

    #Escuchador
    def handleNotification(self, cHandle, data):

        #Si notificacion sensores inerciales
        if cHandle == 17:
            #Variable global

            print("Notificación recibida de los sensores inerciales\n")
            ############Acelerómetro##############
            accX = struct.unpack('<h',data[2:4])[0]
            print("\ndatosAcc X: " + str(accX) +"\n")
```

Ilustración 54: Código del manejador

Para conocer el número que identifica el manejador de una característica concreta hay que comprobar que el tamaño de la información exportada en una característica concreta concuerda con el tamaño de la información enviada. Esto se puede comprobar en la función AccGyroMag_Update(), en la que se realiza el envío de los datos de los sensores inerciales. La función AccGyroMag_Update() se implementa en el fichero sensor_service.c.

```c
/**
 * @brief  Update acceleration/Gryoscope and Magneto characteristics value
 * @param  MOTION_SENSOR_Axes_t Acc Structure containing acceleration value in mg
 * @param  MOTION_SENSOR_Axes_t Gyro Structure containing Gyroscope value
 * @param  MOTION_SENSOR_Axes_t Mag Structure containing magneto value
 * @retval tBleStatus      Status
 */
tBleStatus AccGyroMag_Update(MOTION_SENSOR_Axes_t *Acc,MOTION_SENSOR_Axes_t *Gyro,MOTION_SENSOR_A
{
  tBleStatus ret;
  int32_t AXIS_X;
  int32_t AXIS_Y;
  int32_t AXIS_Z;

  uint8_t buff[2+3*3*2];

  STORE_LE_16(buff   ,(HAL_GetTick()>>3));

  STORE_LE_16(buff+2 ,Acc->x);
  STORE_LE_16(buff+4 ,Acc->y);
  STORE_LE_16(buff+6 ,Acc->z);
```

Ilustración 55: Función AccGyroMag_Update

Se ha comprobado que el buffer enviado tiene un tamaño de 20 bytes. Solo existe una característica que envíe un buffer con este tamaño, por lo tanto, es sencillo conocer que manejador corresponde a esta característica.

Para descodificar la información es necesario conocer en qué posición del buffer se encuentra cada dato. Esta información es proporcionada por el fabricante.

Una vez asignado un manejador, es necesario activar las notificaciones de aquellas características que se desea recibir información. Para ello, es necesario escribir en el registro "valHandle + 1" el valor 0x00100.

Por último, se debe llamar a la función de la clase peripheral waitForNotification(), para que el gateway espere a recibir una notificación del sensor con un nuevo valor. Para activar el envío de los datos de los sensores ambientales se debe seguir el mismo procedimiento.

```
#Funcion para establecer que caracterisitcas se leen y asociar escuhadores
def Read(p):

    #Acceso al servicio correspondiente (sensor ambiental e inercial)
    services=p.getServiceByUUID("00000000-0001-11e1-9ab4-0002a5d5c51b")

    #ObObtencion de todas las caracteristicas de ese servicio
    charact = services.getCharacteristics()

    #Caracteristica de sennsores ambientales
    characteristicTemp = charact[0]
    #Caracteristica de sensores inerciales
    characteristicAcc = charact[1]

    #Activacion las notificaciones
    p.writeCharacteristic(characteristicAcc.valHandle+1, _NOTIFICATION_ON)
    p.writeCharacteristic(characteristicTemp.valHandle+1, _NOTIFICATION_ON)

    #Espera a recibir notificaciones
    while True:
        p.waitForNotifications(10.0)
```

Ilustración 56: Activación de notificaciones y espera

En resumen, cada vez que el sensor envíe un nuevo valor, se ejecuta la función handleNotification() explicada anteriormente. En este primer ejemplo, únicamente se muestra la información por pantalla, aunque en versiones futuras se realiza un tratamiento más exhaustivo de la información y se encamina hacia el Cloud.

7.3 Cloud

En la primera versión del proyecto se ha optado por utilizar un Cloud de tipo privado. En concreto, se despliega una instancia de Thingsboard en una máquina virtual. Esta máquina virtual se encuentra dentro de la red de equipos de la Universidad de Oviedo y es necesario estar dentro de esta red para establecer una comunicación.

Tanto el gateway como el equipo utilizado para acceder a la interfaz web de Thingsboard se encuentran fuera de esta red. Para poder establecer la comunicación con éxito es necesario instalar un cliente de VPN para simular que estos equipos se encuentran dentro de la red de la Universidad.

7.3.1 Instalación y configuración

Para su instalación se ha seguido la guía de instalación proporcionada por Thingsboard [37].

El matiz más importante de la instalación es el uso de una base de datos híbrida utilizando PostgresSQL para el almacenamiento de toda la información relativa a usuarios, dispositivos, relaciones… mientras que para los datos de temperatura, vibraciones y magnetismo se utiliza TimescaleDB, más indicada para el almacenamiento de datos de tipo serie temporal.

Una vez finalizada la instalación, es posible acceder al panel de configuración de Thingsboard, disponible en el puerto 8080 de la máquina virtual, a través de un navegador web. La instalación precarga en la base de datos unos usuarios que permiten comenzar a configurar esta herramienta rápidamente.

En primer lugar, utilizando el usuario con rol de administrador, se debe crear un nuevo propietario en el panel de “Propietarios”. En este caso, se ha definido el propietario IntermarkIT.

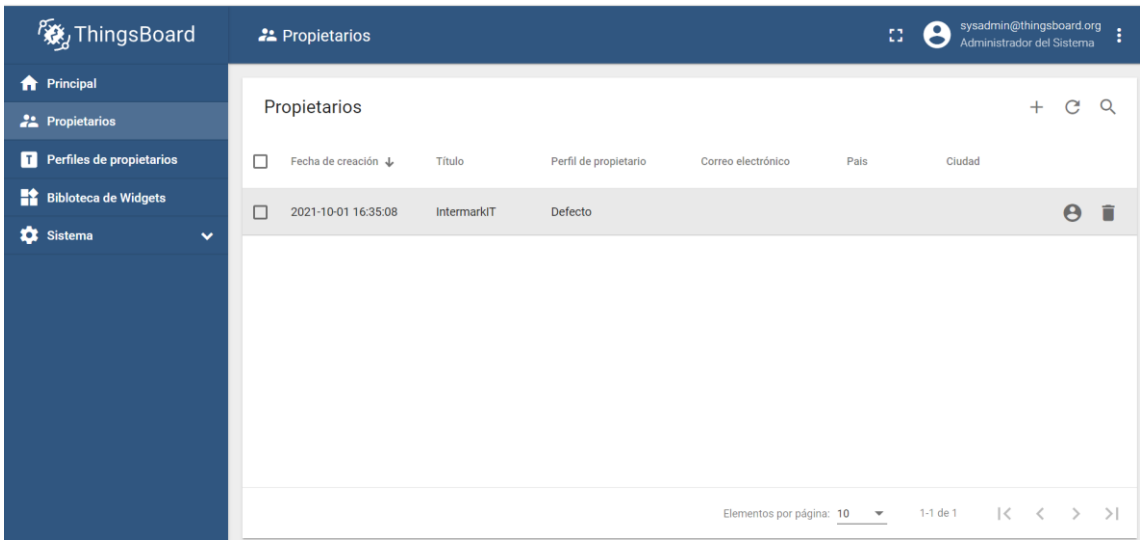

Ilustración 57: Propietarios Thingsboard

A continuación, se debe agregar un usuario propietario a este propietario. Desde este usuario podrán gestionarse los diferentes clientes, dispositivos o entidades. Esta acción

se realiza desde el panel “Propietarios”, en el icono mostrado en la Ilustración 58 del propietario al que pertenecerá el usuario.

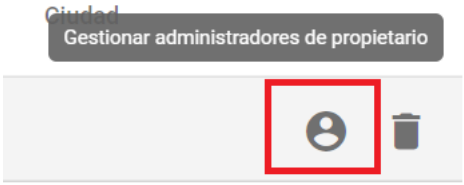

Ilustración 58: Botón crear usuario Thingsboard

Con este usuario será posible crear nuevas entidades como activos, clientes, paneles de visualización, etc. En nuestro caso, es suficiente con crear un dispositivo (el sensor) de modo que sea posible comenzar a enviar datos hacía la nube.

Desde la cuenta de propietario creada anteriormente, en el panel “Dispositivos” se crea un nuevo dispositivo. En este momento ya es posible realizar el envío de información y Thingsboard, de forma automática se encarga de almacenarla de forma que sea accesible y pueda visualizarse desde los paneles que se han creado más adelante.

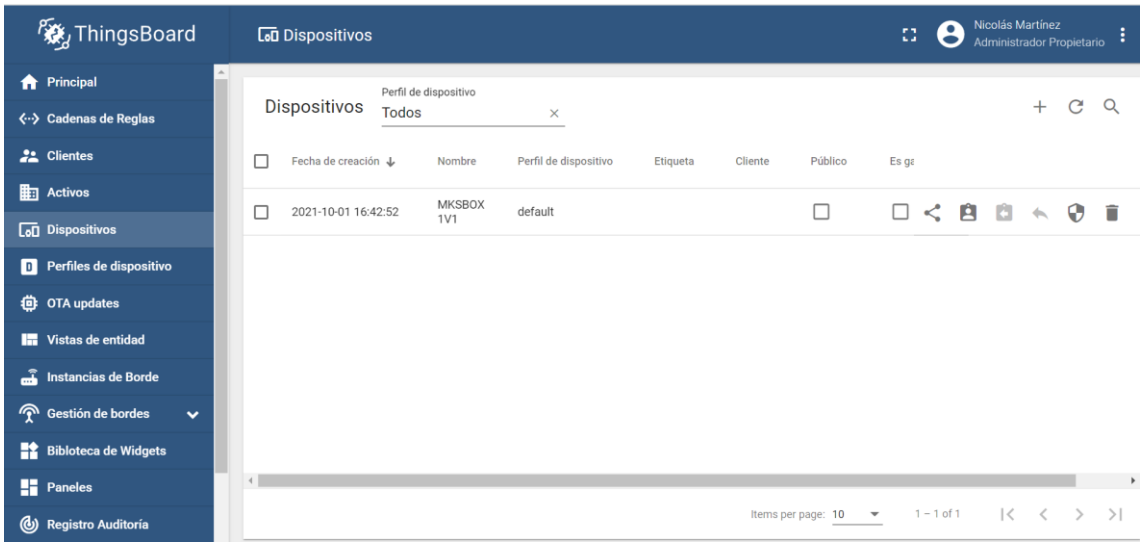

Ilustración 59: Lista de dispositivos creados

Desde el menú de Dispositivos, mostrado en la Ilustración 59, es posible gestionar cada uno de los dispositivos con los botones del menú lateral mostrados en la Ilustración 60.

Ilustración 60: Botones gestión dispositivo

Las acciones permitidas son las siguientes:

- **Hacer dispositivo público**: el dispositivo y su información podrá ser accesible a otros posibles propietarios dados de alta en Thingsboard.

- **Asignar a cliente**: permite asignar dispositivos a clientes de modo que estos tengan acceso a los datos del dispositivo desde su interfaz.
- **Desasignar a cliente**: permite desasignar los dispositivos asignados con la opción anterior.
- **Gestionar credenciales**: permite seleccionar el tipo de credencial del dispositivo para su acceso mediante API. Permite 3 métodos de autenticación, token de acceso, mediante certificado o MQTT Basic. En este proyecto se ha utilizado el método token de acceso.
- **Borrar dispositivo**: permite eliminar el dispositivo.

7.4 Primer envío de información

Thingsboard soporta múltiples protocolos de comunicación para el envío de información desde los diferentes dispositivos. Los protocolos soportados son MQTT, COAP, HTTP y LWM2M. De entre estos protocolos se ha escogido MQTT para el envío de información entre el Gateway y la nube. Se ha elegido este método por ser el más extendido para las comunicaciones IoT de dispositivos y haber trabajado anteriormente con él en otros proyectos.

MQTT es un protocolo de mensajería estándar orientado al envío de información de dispositivos IoT basado en el protocolo de mensajería publicación-suscripción. Su pequeño tamaño, la facilidad para su implantación y el pequeño ancho de banda de sus comunicaciones hace que sea un protocolo muy utilizado en las comunicaciones IoT.

Es necesario un cliente MQTT que se encargue de publicar la telemetría para que el bróker MQTT de Thingsboard pueda procesarla y almacenarla. Los suscriptores son los dispositivos que previamente se han creado en Thingsboard.

Para enviar los mensajes MQTT se ha utilizado un cliente de línea de comandos desarrollado por la fundación Eclipse. Esta fundación ha desarrollado Mosquitto, un bróker MQTT de código abierto y diversas librerías en lenguaje C que permiten crear clientes MQTT [38], además del cliente de línea de comandos ya comentado.

Existen 2 versiones diferentes. Una de ellas, mosquitto_sub permite suscribirse a un tópico y acceder a todos los mensajes que se publiquen en este. La segunda versión, mosquitto_pub, permite publicar mensajes en un tópico. Será esta última la utilizada cada vez que es necesario enviar un dato hacia la nube.

En el script donde se realizan las lecturas del sensor y el procesamiento de la información, también debe incluir el código necesario para la comunicación con la nube. Para ello, se ha utilizado la siguiente línea de código que lanza un script donde se llama al publicador de mensajes de Mosquitto.

```
subprocess.run(['bash','./mqttClientV2.sh', payload, ACCESS_TOKEN])
```

Ilustración 61: Comando para ejecutar cliente MQTT

El script “mqttClientV2.sh” contiene la llamada a mosquitto_pub. Es necesario indicar la dirección IP en la que se encuentra el bróker MQTT, el tópico en el que se publica, el

token de acceso del dispositivo que se recibe en el segundo parámetro y el payload (telemetría en formato JSON). Además, se redirige la salida al fichero a /dev/null para evitar que se muestren por pantalla mensajes informativos.

```
#!/bin/sh
mosquitto_pub -d -h "156.35.163.171" -t "v1/devices/me/telemetry" -u "$2" -m "$1" > /dev/null
```

Ilustración 62: Contenido del fichero mqttClientV2.ch

Thingsboard aporta documentación para el uso de sus APIs. En concreto, la documentación relativa a la API del protocolo MQTT se encuentra disponible en [39]

Por último, con el objetivo de conocer si se está recibiendo correctamente la información, se debe crear un dashboard que permita visualizar las lecturas de temperatura y de los sensores inerciales.

Esa tarea se realiza también desde la cuenta de Administrador propietario, en el apartado "Paneles".

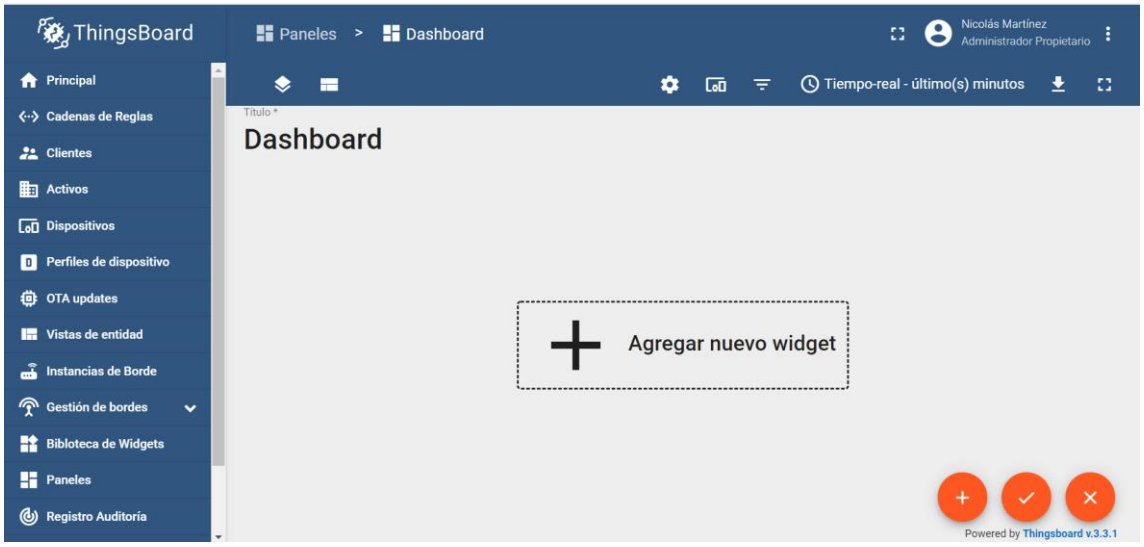

Ilustración 63: Menú Paneles Thingsboard

En el panel se deben añadir widgets donde se visualiza la información deseada. En este caso se muestra la evolución de la temperatura y de las aceleraciones y los campos magnéticos. Para cada magnitud se ha generado un widget distinto. De entre los diferentes tipos de widgets disponibles, se opta por el tipo "tabla de series temporales", ya que es el que mejor se ajusta a la visualización deseada.

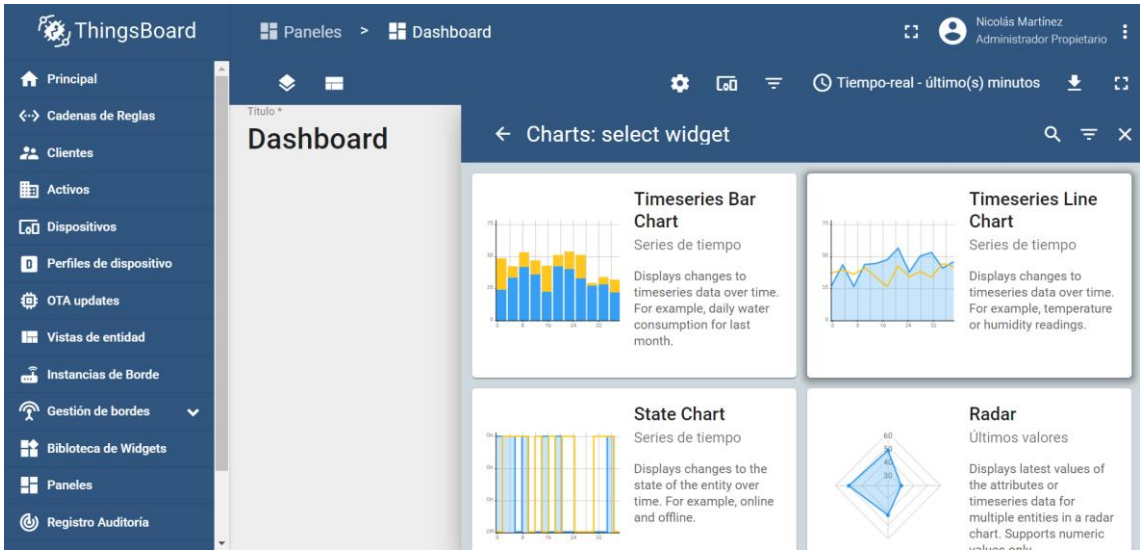

Ilustración 64: Tipos de widget

Una vez seleccionado el tipo de widget, se debe indicar el set de datos del que se desea mostrar la información. Desde esta ventana de configuración también es posible configurar otros elementos del widget, tanto visuales como de comportamiento.

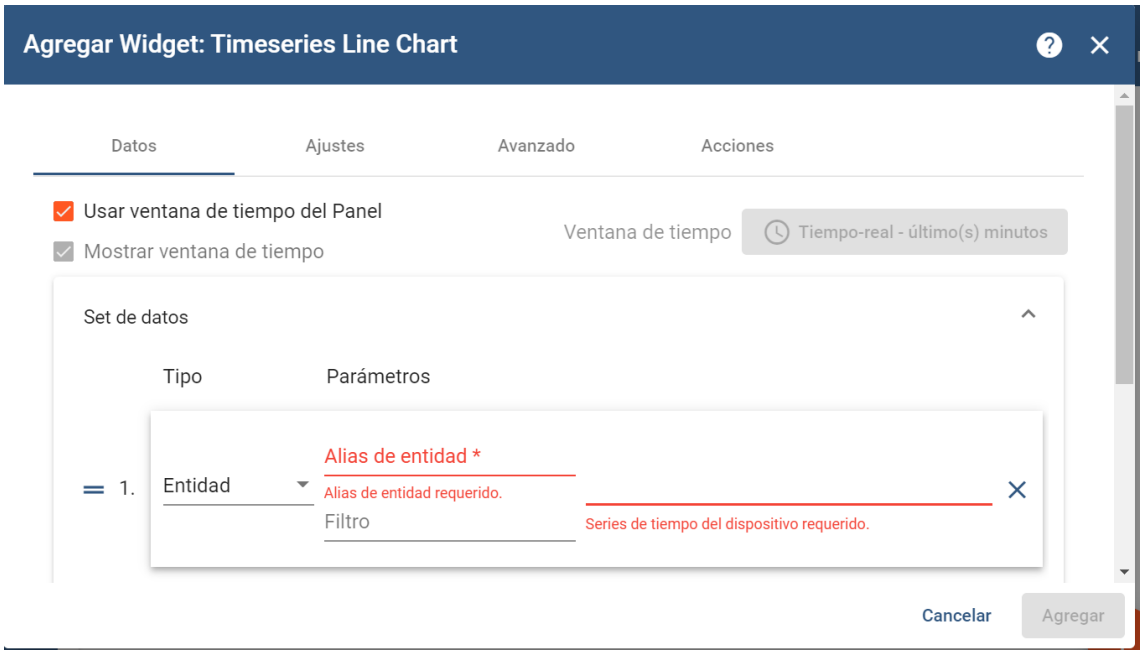

Ilustración 65: Configuración widget

Para referenciar el dispositivo del que se deben mostrar los datos es necesario generar un alias donde se indica el elemento o dispositivo deseado. Al indicar el tipo, se muestra una lista con todos los dispositivos disponibles. Como solo existe uno, esta lista contiene únicamente el dispositivo creado con anterioridad.

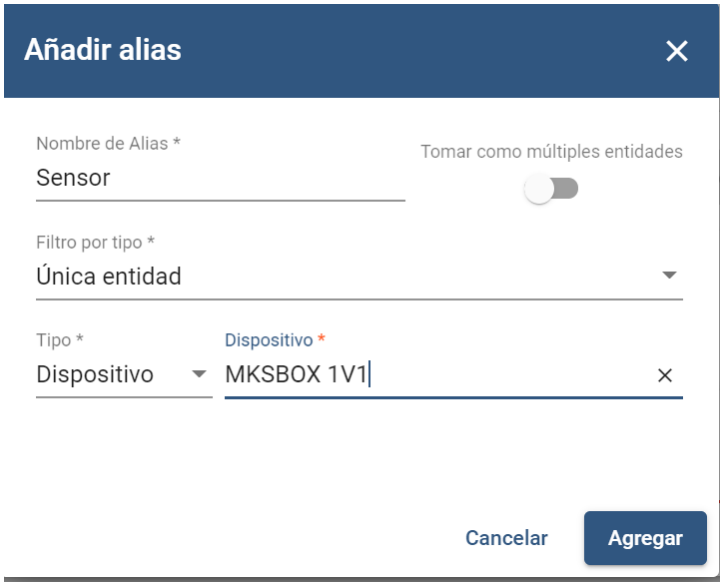

Ilustración 66: Configuración nuevo alias

En último lugar, se deben seleccionar las claves, que son las magnitudes que se han almacenado en Thingsboard. Es posible escoger entre temperatura, acelerómetro eje X, acelerómetro eje Y, acelerómetro eje Z, magnetómetro eje X, magnetómetro eje Y o magnetómetro eje Z.

Ilustración 67: Claves disponibles del dispositivo

El resultado final se muestra en la Ilustración 68. Se puede comprobar que los gráficos se actualizan en tiempo real a medida que el Gateway envía información.

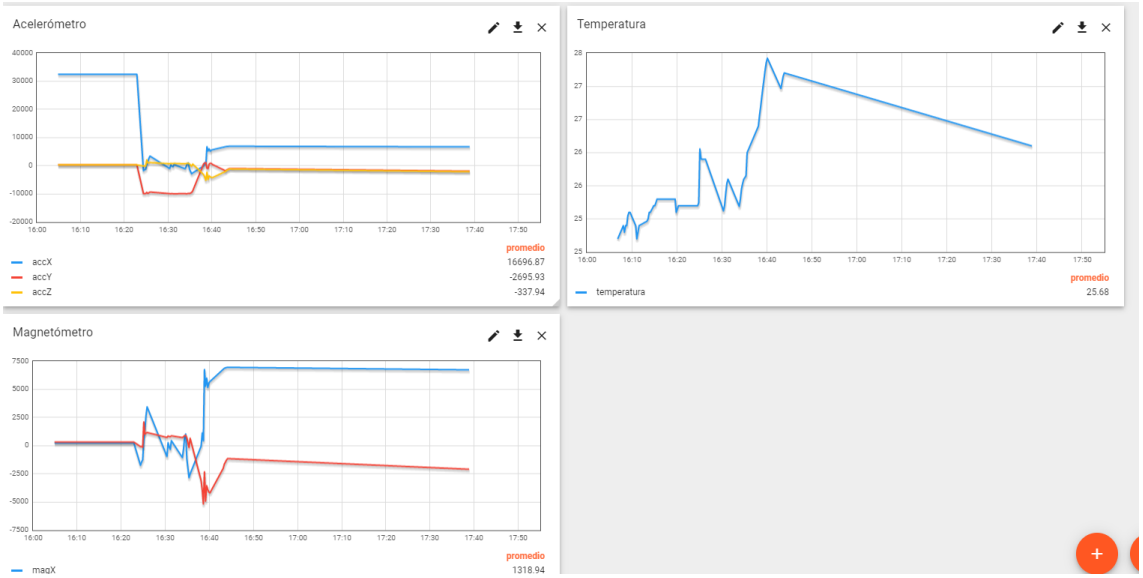

Ilustración 68: Resultados primera configuración

7.5 Evaluación resultados con firmware FP-SNS-ALLMEMS2

Al observar los gráficos generados en Thingsboard, se ha observado que el número de muestras recibido es menor que el número de muestras que el sensor está capturando. Aunque el objetivo final no es almacenar las vibraciones en Thingsboard si no la señal transformada al dominio de la frecuencia, sí que es de interés analizar el porqué de esta situación para averiguar en qué punto del sistema se está produciendo una pérdida de datos.

La primera hipótesis es que el gateway esté actuando como cuello de botella y no sea capaz de procesar toda la información que el sensor envía. Para comprobar si se está produciendo esta situación, se ha llevado a cabo un experimento donde los datos recibidos se almacenan en un buffer durante un tiempo determinado. En concreto, cada una de las magnitudes se almacena en un buffer durante un periodo de 60 segundos, con el objetivo de determinar cuántos paquetes de información se han perdido durante este intervalo de tiempo.

La frecuencia de muestreo de los sensores inerciales se configuró para recoger medidas con una frecuencia 5 kHz. De modo que, en un minuto deberían capturarse 300000 muestras. Por otro lado, la frecuencia de muestreo de los sensores ambientales se configuró en 1 Hz, por lo tanto, deberían recibirse 60 muestras de temperatura.

Este experimento ha rebelado que el número de muestras de los sensores inerciales recibidas por el gateway era de 2000 muestras, lo que equivale a una frecuencia de muestreo de 33 Hz. Respecto a las muestras de temperatura, se reciben 53 muestras en lugar de 60 que se envían desde el sensor.

En ambos casos el número de muestras recibidas es menor al número de muestras enviadas por el sensor. Si además de mostrar el dato por pantalla se realiza un envío a la nube de estos datos, el número de muestras recibidas desciende hasta 60 muestras. Se

confirma la hipótesis planteada, el gateway no es capaz de procesar todas las muestras que el sensor le comunica.

Por tanto, se han detectado dos situaciones donde la RPi actúa como cuello de botella en el sistema provocando una pérdida de información en el transcurso de esta desde el Edge hasta la nube:

1. Pérdida de datos en la comunicación sensor-gateway.
2. Pérdida de datos en la comunicación gateway-cloud.

Respecto al primer punto, no es posible optimizar el código del gateway de forma que se solvente esta situación. El sistema se encuentra limitado por el módulo de comunicaciones BLE de la RPi. Para solventar este problema será necesario recurrir al Edge computing para gestionar el envío de información desde el sensor sin que haya pérdida de información. En **Solución pérdida de datos en la comunicación Edge-Gateway** se describe en detalle la solución propuesta.

Respecto al segundo punto, se ha tratado de realizar modificaciones en el código que permitiesen disminuir el número de muestras perdidas en la comunicación entre estos elementos de la arquitectura.

Para evitar pérdidas en la comunicación entre el gateway y el cloud, se ha utilizado la programación basada en hilos. En lugar de realizar la comunicación con Thingsboard en el hilo principal, se despliega un hilo que se encarga de realizar esta tarea. Mientras tanto, en el hilo principal, se siguen recibiendo datos del sensor.

```
t1 = threading.Thread(target=enviaDatosInerciales,args=(dfInercial_aux,count,))

t1.start()
```

Ilustración 69: Ejemplo creación hilo

```
#Funcion que envía el buffer de datos inerciales a la nube
def enviaDatosInerciales(dfInercial, count):
    for i in range(len(dfInercial)):
        payload=json.dumps({"ts":dfInercial.iloc[i,0],"values":{"accX":dfInercial.iloc[i,1],"accY":dfInercial.iloc[i,2],
        "accZ":dfInercial.iloc[i,3],"magX":dfInercial.iloc[i,4],"magY":dfInercial.iloc[i,5],"magZ":dfInercial.iloc[i,6]}})
        print("ts:" + str(dfInercial.iloc[i,0]))
        #Ejecutamos el script cliente_MQTT_TB_MTDI.sh para que envie el payload generado
        with open(os.devnull, 'w') as devnull:
            subprocess.run(['mosquitto_pub' ,'-d' ,'-h','156.35.163.171' ,'-t',  'v1/devices/me/telemetry' ,'-u',  ACCESS_TOKEN, '-m', str(payload)],stdout=devnull)
        time.sleep(1)
```

Ilustración 70: Función para enviar datos al Cloud

Estos cambios se implementan en el script “lecturaAccTempV7.py”

Con esta solución se ha conseguido solucionar uno de los problemas identificados anteriormente, la pérdida de información debido al envío de datos a la nube ya que el gateway es capaz de procesar la misma cantidad de información independientemente de las operaciones que se realicen.

8 Solución pérdida de datos en la comunicación Edge-Gateway

Anteriormente se ha comentado que existe una pérdida de información en la comunicación entre los dispositivos del Edge y del Fog. El gateway no es capaz de procesar todas las lecturas que se envían desde el sensor y gran parte de las lecturas no son procesadas. Esto provoca que la frecuencia de muestreo se vea limitada a la frecuencia con la que el gateway es capaz de procesar los paquetes BLE enviados desde el sensor.

La solución planteada para resolver este problema ha sido modificar el firmware del sensor para aprovechar al completo el tamaño de cada paquete BLE enviado. En la primera versión del firmware, en cada lectura de los sensores inerciales se envía al gateway en un paquete BLE de 20 bytes.

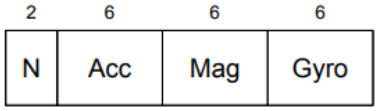

Ilustración 71: Paquete BLE datos inerciales

En la nueva versión, se han creado 3 buffers donde se almacenan los datos correspondientes a las aceleraciones en los ejes X, Y y Z respectivamente. En el momento en el que se han recogido tantos datos como los buffers son capaces de almacenar, se realiza un envío de estos de forma particionada.

En concreto, se han declarado 3 buffers con un tamaño de 52000 elementos. Como se recogen 5000 muestras por segundo, en total se están midiendo de forma continua 10,4 segundos.

Para realizar el envío, se ha aumentado el tamaño de los paquetes BLE con el objetivo de reducir el tiempo en el que el sensor se encuentra transmitiendo datos, ya que, durante este periodo, no se están realizando lecturas del acelerómetro.

En lugar de enviarse paquetes de 20 bytes, se aumenta su tamaño hasta los 152 bytes. El tamaño máximo permitido para un paquete BLE es de 155 bytes. Teniendo en cuenta que es necesario destinar 2 bytes para identificar cada paquete con un entero, el máximo tamaño de paquete que permite enviar el mismo número de lecturas para las 3 coordenadas es 152 bytes. De este modo, en cada paquete se envían 25 lecturas de aceleración por eje.

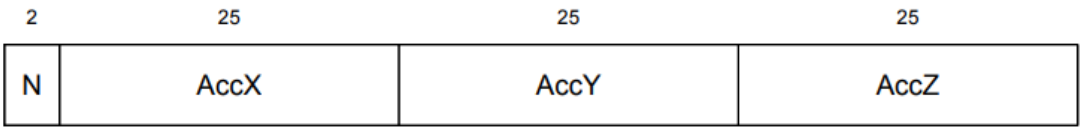

Ilustración 72: Paquete BLE sensores inerciales

8.1 Modificación firmware sensor

Con el objetivo de facilitar las labores de desarrollo de las funcionalidades descritas en el apartado anterior se ha optado por utilizar un proyecto diferente. En concreto, la aplicación BLESensors disponible dentro del pack de funciones FP-SNS-STBOX1 [40]. Esta aplicación permite, de forma sencilla, obtener datos de los diferentes sensores y transmitirlos a través de BLE.

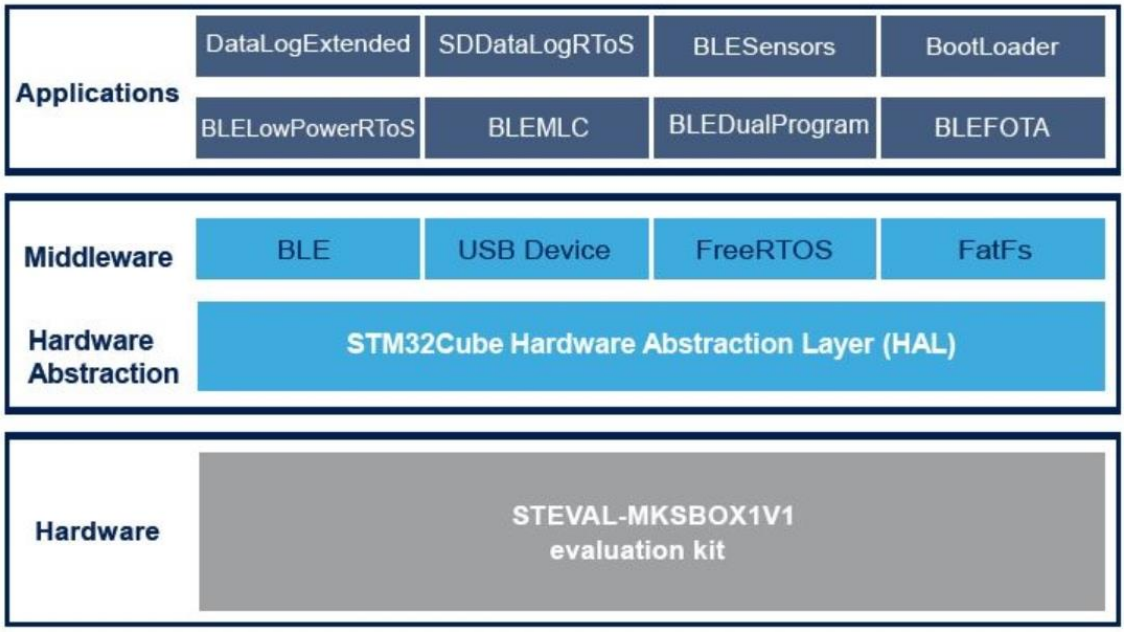

Ilustración 73: Arquitectura aplicación BLE Sensors

La principal diferencia con el paquete de funciones FP-SNS-ALLMEMS2 es el modelo de programación utilizado. En el paquete de funciones FP-SNS-ALLMEMS2 utiliza el modelo foreground/background. Existen dos hilos, ProcessThread y HostThread. El primero de ellos es el encargado de procesar las interrupciones generadas por los timers cíclicos asociados a cada una de las diferentes características que proporciona el sensor (sensores ambientales, sensores inerciales …) y añadir en una cola las lecturas que se obtuvieron de cada uno de los sensores.

```
static void ProcessThread(void const *argument)
{
  msgData_t msg;

  while (1){
#ifdef ALLMEMS2_ENABLE_SD_CARD_LOGGING
    RTC_GetCurrentDateTime();
#endif /* ALLMEMS2_ENABLE_SD_CARD_LOGGING */

   if (semRun != NULL){
       if(osSemaphoreWait(semRun, osWaitForever) == osOK){
         if(set_connectable){
         InitLibraries();
         if(NecessityToSaveMetaDataManager) {
           uint32_t Success = EraseMetaDataManager();
           if(Success) {
             SaveMetaDataManager();
           }
         }
         msg.type  = SET_CONNECTABLE ;
         SendMsgToHost(&msg);
         set_connectable =0;
       }
       /* Handle Interrupt from MEMS */
       if(MEMSInterrupt) {
         MEMSCallback();
         MEMSInterrupt=0;
#ifdef ENABLE_SHUT_DOWN_MODE
         ShutdownTimeOutReset();
#endif /* ENABLE_SHUT_DOWN_MODE */
       }
```

Ilustración 74: Función ProcessThread proyecto FP-SNS-ALLMEMS2

Si se produce una interrupción de los sensores de movimiento (MEMSInterrupt), se ejecuta la rutina MEMSCallback() donde se extraen los valores de los sensores correspondientes y se envían a la cola para que sean procesados.

El otro hilo, HostThread, se encarga de extraer de esta cola las lecturas insertadas por el otro hilo y comunicarlas mediante BLE.

```
static void HostThread(void const *argument)
{
  msgData_t *msgPtr;
  osEvent  evt;

  for (;;) {
    evt = osMailGet(mail, osWaitForever);        // wait for mail
    if (evt.status == osEventMail) {
      msgPtr = evt.value.p;
      switch(msgPtr->type)
      {
        case SET_CONNECTABLE:
          hciProcessEnable = 1 ;
          setConnectable();
          hostConnection = NOT_CONNECTED;
        case  ACC :
          AccEvent_Notify(msgPtr->acc, 2);
          break;

        case  ACC_STEP :
          if(W2ST_CHECK_HW_FEATURE(W2ST_HWF_PEDOMETER))
            AccEvent_Notify(msgPtr->stepCnt, 2);
          if(W2ST_CHECK_HW_FEATURE(W2ST_HWF_MULTIPLE_EVENTS))
            AccEvent_Notify(msgPtr->stepCnt, 3);
          break;

        case  AUDIO_LEV :
          AudioLevel_Update(msgPtr->DBNOISE_Value_Ch);
          break;

        case  ENV :
          Environmental_Update(msgPtr->env.press,msgPtr->env.hum,msgPtr->env.temp2, msgPtr->env.temp1);
          break;
```

Ilustración 75: Función HostThread FP-SNS-ALLMEMS2

Estas funciones se encuentran definidas en el fichero main.c

Sin embargo, en el paquete de funciones FP-SNS-STBOX1 se utiliza un modelo superloop. En este modelo, se revisa en un bucle infinito si los timers asociados a las diferentes características han modificado los flags que indican si hay un nuevo valor disponible. La programación de este modelo es más sencilla y por lo tanto facilita las labores de desarrollo.

En la Ilustración 77 se muestra el código del bucle infinito mencionado anteriormente. En este bucle comprueba si se modifican los valores de los flags de cada una de las características. Por ejemplo, el flag SendBaterryInfo tendrá valor 1 cada vez que el timer asociado al envío de información sobre la batería del dispositivo. Esto sucede cada segundo, tal y como se especifica en el parámetro STBOX1_UPDATE_LED_BATTERY del fichero Projects\STM32L4R9ZI-SensorTile.box\Applications\BLESensors\Inc TargetFeatures.h.

```
/* Every second for Led Blinking and for Updating the Battery status */
#define STBOX1_UPDATE_LED_BATTERY 10000
```

Ilustración 76: Frecuencia de refresco del nivel de batería

```
while (1){


  /* Battery Info Data */
  if(SendBatteryInfo) {
    SendBatteryInfo=0;
    SendBatteryInfoData();
  }

  /* Environmental Data */
  if(SendEnv) {
    int32_t PressToSend;
    uint16_t HumToSend;
    int16_t TempToSend;

    SendEnv=0;

    /* Read all the Environmental Sensors */
    ReadEnvironmentalData(&PressToSend,&HumToSend, &TempToSend);

    /* Send the Data with BLE */
    Environmental_Update(PressToSend,HumToSend,TempToSend);
  }
```

Ilustración 77: Bucle main.c FP-SNS-STBOX1

Como se ha comentado anteriormente, es preciso modificar este código para que, en lugar de enviar una lectura de los sensores inerciales cada vez que el timer asociado finalice, se almacenen estos datos en un buffer de modo que no se pierdan datos debido a la imposibilidad del gateway de procesar tantos paquetes BLE por segundo.

En primer lugar, se han definido 3 estructuras de datos donde se almacenan los datos leídos por el acelerómetro, uno para cada eje de coordenadas. Adicionalmente, se ha declarado un número de tipo entero que se utiliza para iterar sobre los buffers donde se almacenan los valores del acelerómetro.

```
//Buffer iterator
int iterAcc=0;

//Acc buffers
int32_t BufferAx[27000];
int32_t BufferAy[27000];
int32_t BufferAz[27000];
```

Ilustración 78: Declaración de variables auxiliares

En segundo lugar, se ha modificado el código en el que se tratan las interrupciones de los sensores inerciales para que, cada vez que se produzca esta interrupción, se realice únicamente una lectura del acelerómetro y se almacene en el buffer hasta completar 27.000 lecturas. Una vez completadas, se transmiten en paquetes de 152 bytes.

En total, es necesario realizar 1080 envíos para transmitir la totalidad de los datos recogidos. Este valor se obtiene al dividir el número total de muestras, 27.000 muestras, entre el número de muestras que se transmiten en cada envío, 25 muestras.

```
/* Motion Data */
if(SendAccGyroMag) {

    SendAccGyroMag=0;

    BSP_MOTION_SENSOR_Axes_t ACC_Value;
    //BSP_MOTION_SENSOR_Axes_t GYR_Value;
    //BSP_MOTION_SENSOR_Axes_t MAG_Value;

    //ReadInertialData(&ACC_Value,&GYR_Value,&MAG_Value);
    ReadAccData(&ACC_Value);

    //Send the Data with BLE
    //AccGyroMag_Update(&ACC_Value,&GYR_Value,&MAG_Value);

    //BSP_MOTION_SENSOR_Axes_t ACC_Value;
    //ReadAccData(&ACC_Value);

    //Sending Acc values

    //Buffer Ingest
    if(iterAcc<27000){
        BufferAx[iterAcc]=ACC_Value.x;
        BufferAy[iterAcc]=ACC_Value.y;
        BufferAz[iterAcc]=ACC_Value.z;
        iterAcc++;

    }

    //If buffer is full send via BLE
    if((iterAcc==27000)){
        for(int pos=0;pos<1080;pos++){
            Acc_Update(&BufferAx[25*pos],&BufferAy[25*pos],&BufferAz[25*pos],pos);
            HAL_Delay(90);
        }

        //Reset iterator value
        iterAcc=0;
        //Wait for gateway to process buffer
        HAL_Delay(15000);
    }
```

Ilustración 79: Procesamiento de aceleraciones

Cada uno de los envíos se realiza con la misma frecuencia con la que el timer de los sensores inerciales finaliza. El timer de los sensores inerciales se ha configurado para que finalice con una frecuencia de 5 kHz. Por tanto, la RPi no será capaz de procesar todos los paquetes salvo que se introduzca un retardo entre cada envío.

Durante el desarrollo del firmware se han realizado diferentes configuraciones variando el tiempo de espera del sensor entre cada envío para conocer cuál es el tiempo mínimo para realizar envíos sin pérdida de paquetes. En los siguientes ejemplos se muestran 2 casos donde se ha configurado el sensor para almacenar 2048 lecturas en los buffers para su posterior envío en paquetes BLE de 152 bytes.

En el primer ejemplo, se ha introducido una espera entre cada envío de 10 milisegundos. Por tanto, los envíos se realizan con una frecuencia de 100 Hz. Como ya se ha comentado anteriormente, la RPi no es capaz de procesar información con tanta rapidez y se pierden paquetes en la comunicación.

Con esta configuración, el sensor debe enviar 82 paquetes de 152 bytes. Por consiguiente, el tamaño del buffer leído debería ser de 2050 elementos. Sin embargo, el número de elementos recibidos no alcanza ni siquiera los 2000 elementos.

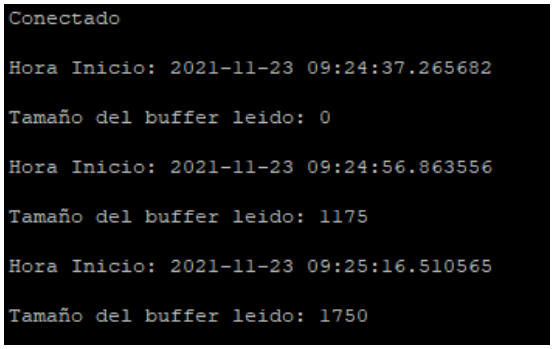

Ilustración 80: Prueba de envío con espera de 10ms

Previamente se ha observado que la máxima frecuencia con la que la RPi era capaz de leer paquetes BLE era de aproximadamente 33Hz. Por tanto, la espera del sensor entre cada envío ha de ser de al menos de 30 milisegundos. No obstante, se ha observado que hasta que la espera no se aumenta hasta los 60 milisegundos, no es posible realizar envíos sin pérdidas de paquetes. Si con la misma configuración se aumenta el tiempo entre cada envío a 60 milisegundos, en el gateway se reciben los 2050 elementos esperados.

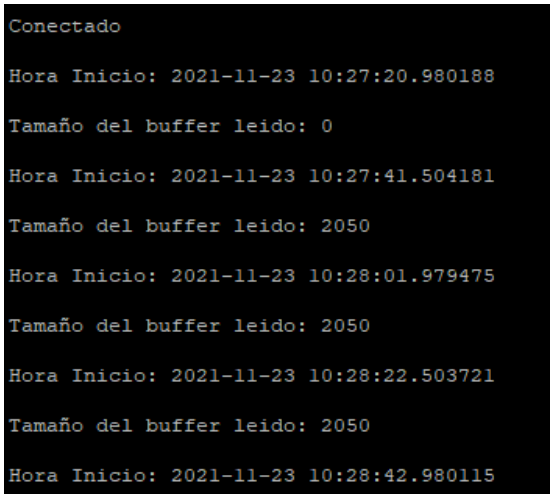

Ilustración 81: Prueba de envío con espera de 60ms

En la versión del firmware desarrollada para la aplicación, se envían 27.000 muestras. El tiempo entre cada envío se ha fijado en 90 milisegundos, y el tiempo de espera tras realizar el último paquete del buffer en 15 segundos. De este modo, se hacen lecturas cada 2 minutos y se asegura el envío completo de los buffers.

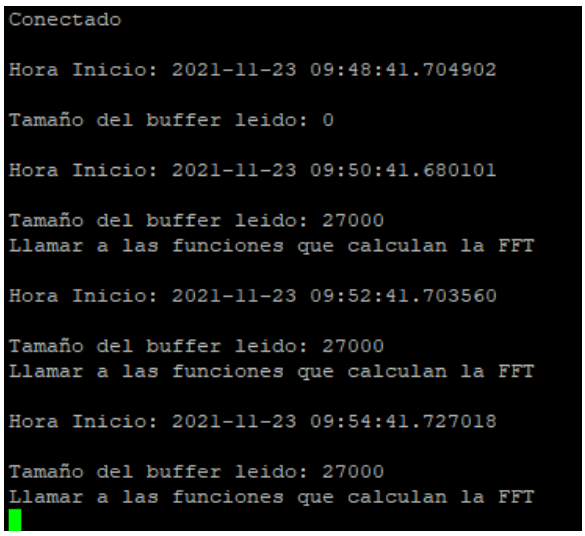

Ilustración 82: Prueba de envío con espera de 90 ms

Para leer únicamente datos del acelerómetro se ha creado una nueva función, ReadAccData(), que sustituye a la función ReadInertialData() en la que además del acelerómetro también se leen los valores del sensor de magnetismos y del giroscopio.

```
void ReadInertialData(BSP_MOTION_SENSOR_Axes_t *ACC_Value,BSP_MOTION_S
{
  /* Read the Acc values */
  BSP_MOTION_SENSOR_GetAxes(LSM6DSOX_0,MOTION_ACCELERO,ACC_Value);

  /* Read the Gyro values */
  BSP_MOTION_SENSOR_GetAxes(LSM6DSOX_0,MOTION_GYRO,GYR_Value);

  /* Read the Magneto values */
  BSP_MOTION_SENSOR_GetAxes(LIS2MDL_0,MOTION_MAGNETO,MAG_Value);
}
```

Ilustración 83: Función ReadInertialData

```
void ReadInertialAcc(BSP_MOTION_SENSOR_Axes_t *ACC_Value)
{
  /* Read the Acc values */
  BSP_MOTION_SENSOR_GetAxes(LSM6DSOX_0,MOTION_ACCELERO,ACC_Value);

}
```

Ilustración 84: Función ReadAccData

Los cambios descritos anteriormente se han realizado en el fichero main.c.

Por último, es necesario modificar la función AccGyroMag_Update() donde se realiza el envío del paquete BLE. Esta función está implementada para recibir por parámetro 3 estructuras con los valores del acelerómetro, giroscopio y magnetómetro y enviar un paquete BLE de 25 bytes con estos datos.

Se ha creado una nueva función, Acc_Update() que sustituye a la función AccGyroMag_Update() y donde se realizan las modificaciones necesarias. Esta función recibe como parámetros los 3 vectores correspondientes a 150 lecturas del acelerómetro (25 de cada eje) y un entero que permite conocer cuando comienza el envío de un conjunto de datos nuevo.

```
tBleStatus Acc_Update(int32_t *AccX,int32_t *AccY,int32_t *AccZ,int pos)
{
  tBleStatus ret;

  uint8_t buff[2+3*25*2];

  STORE_LE_16(buff   ,pos);

  for(int i=0;i<25;i++){
      STORE_LE_16(buff+2*(i+1) ,AccX[i]);
  }

  for(int j=0;j<25;j++){
      STORE_LE_16(buff+50+2*(j+1) ,AccY[j]);
  }

  for(int k=0;k<25;k++){
      STORE_LE_16(buff+100+2*(k+1) ,AccZ[k]);
  }
  ret = ACI_GATT_UPDATE_CHAR_VALUE(HWServW2STHandle, AccGyroMagCharHandle, 0, 2+3*25*2, buff);

  if (ret != BLE_STATUS_SUCCESS){
    if(ret != BLE_STATUS_INSUFFICIENT_RESOURCES) {
      if(W2ST_CHECK_CONNECTION(W2ST_CONNECT_STD_ERR)){
        BytesToWrite =sprintf((char *)BufferToWrite, "Error Updating Acc/Gyro/Mag Char\n");
        Stderr_Update(BufferToWrite,BytesToWrite);
      } else {
        STBOX1_PRINTF("Error Updating Acc/Gyro/Mag Char ret=%x\r\n",ret);
      }
    } else {
      STBOX1_PRINTF("Error Updating Acc/Gyro/Mag Char ret=%x\r\n",ret);
    }
  }

  return ret;
}
```

Ilustración 85: Función Acc_Update

Adicionalmente, es necesario modificar la función Add_HW_SW_ServW2ST_Service() donde se establecen las características hardware del sensor para indicar que las notificaciones BLE de los sensores inerciales tienen un tamaño de 152 bytes.

```
tBleStatus Add_HW_SW_ServW2ST_Service(void)
{
  tBleStatus ret;

  uint8_t uuid[16];

  COPY_HW_SENS_W2ST_SERVICE_UUID(uuid);
  BLUENRG_memcpy(&service_uuid.Service_UUID_128, uuid, 16);
  ret = aci_gatt_add_service(UUID_TYPE_128,  &service_uuid, PRIMARY_SERVICE, 1+3*3 /* Batt/Env/Inertial */,&HWServW2STHandle);

  if (ret != BLE_STATUS_SUCCESS) {
    goto EndLabel;
  }

  COPY_ACC_GYRO_MAG_W2ST_CHAR_UUID(uuid);
  BLUENRG_memcpy(&char_uuid.Char_UUID_128, uuid, 16);
  ret =  aci_gatt_add_char(HWServW2STHandle, UUID_TYPE_128, &char_uuid, 2+25*3*2,
                           CHAR_PROP_NOTIFY,
                           ATTR_PERMISSION_NONE,
                           GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP,
                           16, 0, &AccGyroMagCharHandle);

  if (ret != BLE_STATUS_SUCCESS) {
    goto EndLabel;
  }
```

Ilustración 86: Modificación función Add_HW_SW_ServW2ST_Service

Tanto la declaración de la función Acc_Update como la modificación de la función Add_HW_SW_ServW2ST_Service() se han realizado en el fichero sensor_service.c.

8.2 Modificaciones script gateway

Con las modificaciones introducidas en **Modificación firmware sensor**, se ha modificado la estructura de los paquetes BLE recibidos por el gateway. Por lo tanto, la RPi debe también modificar la forma de interpretarlos y del procesamiento que realiza.

Las modificaciones se realizan en la función handleNotification(), que es la función encargada de procesar los paquetes BLE enviados por el sensor.

```
def handleNotification(self, cHandle, data):
    #Si datos de tipo inercial
    if cHandle == 14:

        #Decodificacion de los bytes recibidos
        datosAcc = struct.unpack('<76h',data)

        #Si primer envio
        if (datosAcc[0]==0):
            #Obtener marca de tiempo
            self.date=time.time_ns()


            #Si el buffer está completo
            if (len(self.datosFFTEjeX) >= _DATA_BUFFER):
                print("Llamar a las funciones que calculan la FFT")
                #Exportar datos a fichero
                guardaFichero(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,self.count)
                #Almacenamiento de vibraciones en gateway
                guardarBBDD(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,self.date)
                #Calculo de FFT y subida a la nube
                calculaFFT(self.datosFFTEjeX, 'X', self.count,self.date)
                calculaFFT(self.datosFFTEjeY, 'Y', self.count,self.date)
                calculaFFT(self.datosFFTEjeZ, 'Z', self.count,self.date)
                self.count=self.count+1


        #Reset valor de los buffers
        self.datosFFTEjeX=[]
        self.datosFFTEjeY=[]
        self.datosFFTEjeZ=[]
#Se recorren los datos recibidos y se almacenan en el buffer corresppondiente
    for acc in datosAcc[1:26]:
        self.datosFFTEjeX.append(acc)
    for acc in datosAcc[26:51]:
        self.datosFFTEjeY.append(acc)
    for acc in datosAcc[51:76]:
        self.datosFFTEjeZ.append(acc)
```

Ilustración 87: Modificaciones en el manejador de notificaciones

En esta nueva versión de la función, se comprueba que el id del paquete recibido es 0 y por tanto el paquete corresponde a los 25 primeros elementos del buffer. En este momento, se habrá completado ya el envío del buffer anterior, y es posible realizar el tratamiento de los datos.

Además de las variaciones derivadas del cambio en el envío de los paquetes BLE, es esta versión del script también se han añadido otras funcionalidades. Cuando se completa el envío, se comprueba que el tamaño del buffer es el esperado y se realizan las labores de:

- ***Almacenamiento:*** el script permite el almacenamiento de las aceleraciones recibidas tanto en un fichero con formato csv como en una base de datos de tipo NoSQL. Para realizar estas tareas se pueden utilizar las funciones guardaFichero() y guardarBBDD() respectivamente. En **Gateway** se describe en detalle cómo se realiza el almacenamiento en el gateway, en una base de datos NoSQL .

- ***Procesado:*** se realiza el cálculo de la FFT de los datos recibidos, de modo que se obtenga el espectro de frecuencias. Se realiza con la función calculaFFT(). Pese a que esta no ha sido la técnica de análisis de vibraciones utilizada finalmente, ha servido como primera toma de contacto para probar el almacenamiento de datos en el dominio de la frecuencia en Thingsboard.

- ***Envío al cloud:*** una vez calculadas las transformadas, se envían al cloud para su almacenamiento y representación. Esta tarea se realiza con la función envioCloud().

Por último, se vacían los buffers donde se almacenan los datos recibidos para comenzar a almacenar los datos que se están enviando en ese momento.

Tanto la conexión con el sensor, como las funciones descritas en este apartado se encuentran implementadas en el script "Acc_bbdd_cloud_V1.py"

8.3 Almacenamiento en el Gateway

Debido a la alta frecuencia de muestreo, se cuenta con una gran cantidad de datos relativos a las vibraciones. Sin embargo, estos datos sin un procesado previo no son de gran interés. Por este motivo, se ha optado por realizar únicamente el envío a la nube de la información una vez ha sido procesada.

Sí que resulta de interés tener un histórico de las vibraciones por si en algún momento fuese necesario analizar el comportamiento de las vibraciones del motor o extraer datos de periodos de tiempo específicos.

El gateway, además de realizar el tratamiento y procesado de las vibraciones, tendrá la función de almacenar en una base de datos de tipo NoSQL cada una de las muestras capturadas por el sensor. La base de datos utilizada ha sido InfluxDB 1.8.

Las bases de datos NoSQL están orientadas al almacenamiento de datos de series temporales. Los datos se almacenan con un formato clave-valor. Como clave se utiliza la marca de tiempo en la que fue recogido o recibido el dato. Debido a la frecuencia de muestreo elevada, es requerida una alta precisión a la hora de registrar la marca de tiempo de cada valor.

El sensor no tiene la capacidad de obtener la hora en la que se ha realizado una lectura. Por tanto, esta tarea se ha delegado a la RPi. Con la primera recepción de datos se fija una marca de tiempo que se irá aumentando en función de la frecuencia de muestreo de los datos.

Para generar la marca de tiempo se utiliza el módulo de funciones de Python time. Este módulo proporciona una función, time_ns() que permite generar una marca de tiempo utilizando el sistema de tiempo de Unix.

Para realizar el envío se ha utilizado la HTTP API de Influx que permite insertar datos mediante una petición HTTP de tipo POST [41].

8.3.1 Instalación de InfluxDB

El primer paso es crear la base de datos en el gateway sobre la que serán volcados los datos relativos a las aceleraciones medidas por el sensor. Se ha seguido el manual de instalación para dispositivos Raspberry Pi [42]:

Una vez finalizado el proceso de instalación, es posible interactuar con la base de datos insertando nuevos datos y extrayendo información para su análisis con querys. Para ello, accedemos al servicio con el comando "influx".

```
pi@raspberrypi:~ $ influx
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
```

Ilustración 88: Salida comando influx

En este momento, a través de la línea de comandos es posible una nueva base de datos, insertar datos en ella o ejecutar querys que permitan obtener información.

Haciendo uso de los comandos más básicos se procede a crear una primera base de datos de prueba que permita verificar el funcionamiento del servicio. En este proceso se procede a crear una nueva base de datos llamada "mydb", en la que se insertará de forma manual un dato relacionado con una posible aceleración en el eje X. Para ello, se crea la base de datos con el comando "CREATE DATABASE mydb".

```
> CREATE DATABASE mydb
```

Ilustración 89: Comando para crear base de datos

Es posible verificar la correcta creación de esta utilizando el comando SHOW DATABASES que muestra todas las bases de datos creadas.

```
> SHOW DATABASES
name: databases
name
----
_internal
mydb
```

Ilustración 90: salida comando SHOW DATABASES

A continuación, accedemos a la base de datos creada e insertamos un valor. Al no especificar un timestamp influx utiliza por defecto el timestamp en el que se ejecuta la acción de inserción.

```
> use mydb
Using database mydb
> INSERT Vibracion, accX=234
```

Ilustración 91: comando INSERT

Por último, ejecutamos una query donde se extrae el valor insertado. Se comprueba que todos los pasos se han ejecutado con éxito, ya que el resultado de la query devuelve el valor que se había insertado previamente.

```
> INSERT Vibracion accX=234
> select * from Vibracion
name: Vibracion
time                accX
----                ----
1634920202366294367 234
```

Ilustración 92: Salida comando Select

En futuras implementaciones, en lugar de insertar manualmente cada uno de los datos, será un script el encargado de realizar esta tarea haciendo uso de la API que permite interactuar con Influx por medio de peticiones a través del protocolo HTTP.

8.3.2 InfluxDB API

Para ver el funcionamiento de la API se ha intentado insertar el mismo dato que se ha insertado previamente para comprobar que es posible insertar información a través de la API. El comando utilizado ha sido el siguiente:

```
pi@raspberrypi:~/Desktop/Scripts/Script_BD $ curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'Vibracion accX=234'
HTTP/1.1 204 No Content
Content-Type: application/json
Request-Id: 86efe658-3353-11ec-8034-dca6321e1372
X-Influxdb-Build: OSS
X-Influxdb-Version: 1.8.10
X-Request-Id: 86efe658-3353-11ec-8034-dca6321e1372
Date: Fri, 22 Oct 2021 16:17:18 GMT
```

Ilustración 93: Ejemplo comando INSERT desde API

A continuación, desde el servicio de influx, ejecutamos la query que obtenga todos los datos almacenados en el bucket Vibración. Como se aprecia en la Ilustración 94, el bucket contiene 2 datos, el insertado a través de la línea de comandos y el insertado a través de la API.

```
> select * from Vibracion
name: Vibracion
time                accX
----                ----
1634920202366294367 234
1634920235397659906 234
```

Ilustración 94: Salida comando SELECT II

8.3.3 Comunicación con sensor y almacenamiento en el gateway

Una vez que se ha creada la base de datos y se ha verificado la correcta implementación de funciones que permiten ejecutar querys para el almacenamiento de datos en la misma, es posible trasladar estas nuevas funcionalidades al script donde se realiza la comunicación con los dispositivos del Edge.

En la Ilustración 95 se muestra el script desarrollado para insertar las componentes X, Y y Z de cada muestra en la base de datos. En el primer parámetro se indica el hostname, el tipo de operación (escritura) y el nombre de la base de datos.

Seguidamente, con la opción -- data-binary se indican el measurement donde se almacena, los campos y sus valores y la marca de tiempo en nanosegundos.

```
#Script para insertar en influxDB datos inserciales
#USO-> insertBD.sh measurement ts accX accY accZ
#!/bin/sh

curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary "$1 accX=$3,accY=$4,accZ=$5 $2" >/dev/null
```

Ilustración 95: Script para comunicación mediante InfluxDB API

El script anterior es llamado desde una función auxiliar dentro del script que contiene toda la lógica de negocio del gateway. Esta función, guardarBBDD() recibe como parámetro los 3 buffers y la marca de tiempo en la que fue recibido el primer dato, y se encarga de recorrer los buffers e insertar en la base de datos cada una de las lecturas. Además, en cada iteración se aumenta el valor de la marca de tiempo en 1/frecuenciaMuestreo nanosegundos para que cada muestra tenga la marca de tiempo que le corresponde en función de la marca de tiempo de la primera muestra. Esta función se muestra en la Ilustración 96.

```
#Funcion para guardar en influx los datos de las aceleraciones
def guardarBBDD(Ax,Ay,Az,date):
    print("\nTS primer dato: " + str(date))
    for i in range(len(Ax)):
        retcode=subprocess.run(['bash','./insertBD_Acc.sh',"Vibracion",str(date), str(Ax[i]),str(Az[i]),str(Az[i])],
                stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
        date=date+_PERIOD_NS
```

Ilustración 96: Función guardarBBDD

Debido a que la operación de insertado consume más tiempo que el disponible entre cada comunicación con el sensor, ha sido necesario crear un hilo que, en segundo plano, realice el proceso de almacenamiento de los buffers en la base de datos.

```
if (len(self.datosFFTEjeX) >= _DATA_BUFFER):
                    print("Llamar a las funciones que calculan la FFT")

                    #Almacenamiento de vibraciones en gateway
                    #guardaFichero(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,self.count)
                    t1 = threading.Thread(target=guardarBBDD,args=(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,self.date,))
                    t1.start()
```

Ilustración 97: Creación de hilo para almacenamiento en InfluxDB

El almacenamiento de las vibraciones es demasiado lento. Almacenar los datos recibidos durante una lectura necesitaba aproximadamente 15 minutos para su finalización. Para reducirlo, se ha utilizado la librería para lenguaje Python Influxdb-python, que permite crear un cliente para conexiones con bases de datos de InfluxDB 1.8 o versiones posteriores.

Se ha reescrito la función guardarBBDD() para que, en lugar de realizar peticiones HTTP, se hagan llamadas a la función write de esta libraría. En la Ilustración 98 se muestran los

cambios. Además, no es necesario crear un hilo para la ejecución de esta función, ya que el hilo principal es capaz de ejecutar esta tarea en pocos segundos.

```
#Funcion para guardar en influx los datos de las aceleraciones
def guardarBBDD(Ax,Ay,Az,date):
    #Inicializacion del cliente para conexion con influxdb
    with InfluxDBClient(url='http://'+hostname+':8086', token=f'{username}:{password}', org='-') as client:
        #Conexion con la base de datos
        with client.write_api() as write_api:
            #print("\nInicio guardar BBDD: " + str(datetime.now()))
            #Insercion de cada elemento con el formato {accX:valor, accY:valor, accZ:valor}
            for i in range(len(Ax)):
                point = Point("VibracionPrueba").field("accX", Ax[i]).field("accY",
                                Ay[i]).field("accZ", Az[i]).time(date)
                write_api.write(bucket=bucket, record=point)
                date=date+_PERIOD_NS
```

Ilustración 98: Función guardarBBDD II

Los parámetros de configuración definidos se muestran en la Ilustración 99.

```
#Variables para influxdb client
username = ''
password = ''
hostname = 'localhost'
database = 'mydb'
retention_policy = 'autogen'
bucket = f'{database}/{retention_policy}'
```

Ilustración 99: Parámetros InfluxDB API

En este ejemplo, la base de datos y el cliente que lanza querys se encuentran en la misma máquina. En caso de que la base de datos se encuentre en otra máquina, habrá que especificar su dirección IP en el parámetro localhost y configurar la base de datos para que todas las operaciones requieran identificación mediante usuario y contraseña para evitar brechas de seguridad.

8.4 Resultados

El objetivo fundamental de esta nueva versión era conseguir datos con una frecuencia de muestreo lo más elevada posible. Este objetivo se ha cumplido, ya que el gateway es capaz de procesar todos los paquetes que recibe desde el Edge.

Si analizamos los gráficos del espectro de frecuencias obtenidos, se observa un comportamiento diferente al esperado. Teóricamente, deben aparecer armónicos en la banda en la que se encuentra la frecuencia fundamental de giro y en sus múltiplos X2, X3, etc. Sin embargo, estos aparecen en la frecuencia de los 100 Hz y sus múltiplos.

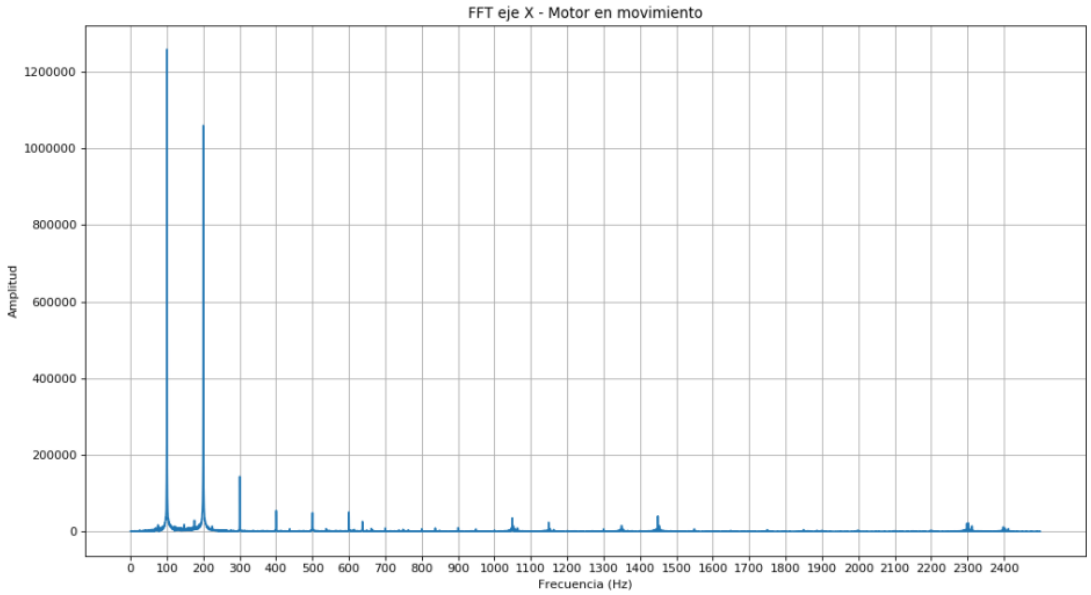

Ilustración 100: FFT Aceleraciones Eje X

Estos armónicos proceden de perturbaciones de carácter electromagnéticas provocadas por la fuente de energía, las cuales provocan vibraciones en los múltiplos pares (x2, x4…) de la frecuencia de la red eléctrica, 50 Hz [43].

Las vibraciones provocadas por estas fuerzas son mayorees que las provocadas por los rodamientos. Esto provoca que los harmónicos provocados por la red eléctrica eclipsen al resto de componentes del espectro.

Si realizamos un zoom de la FFT, se aprecian los harmónicos provocados por las vibraciones de los rodamientos.

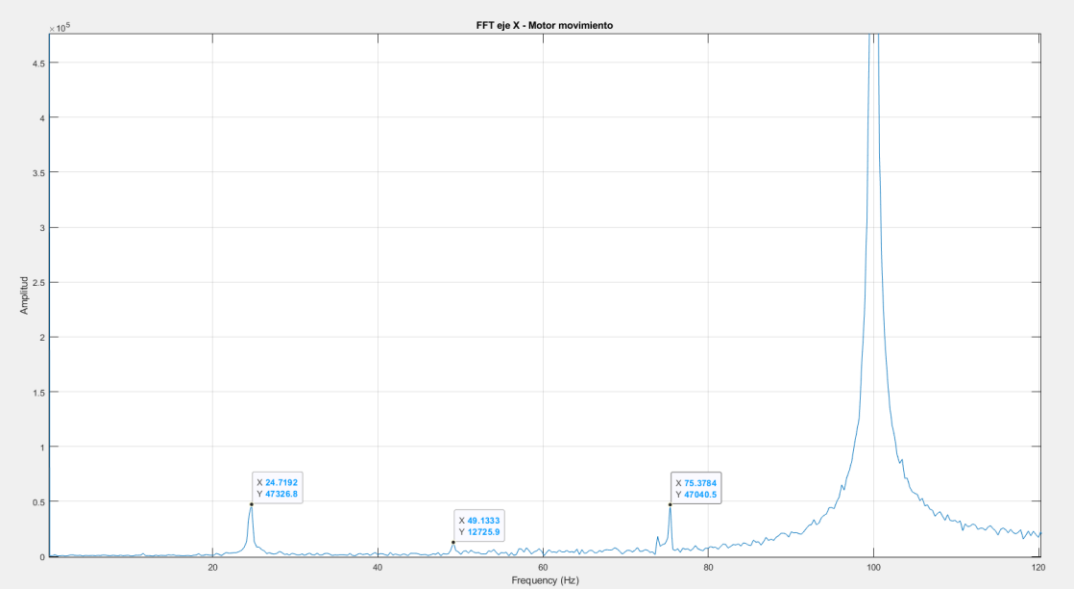

Ilustración 101: Zoom FFT Aceleraciones eje X

9 Detección Anomalías

Para dotar al sistema de una mayor capacidad de detección de fallos en rodamientos es necesario utilizar una técnica más sofisticada que simplemente analizar el espectro de frecuencias obtenido a través de la FFT. De entre las técnicas disponibles se ha escogido el Envelope Analysis.

Esta técnica es muy adecuada para detectar las vibraciones resonantes provocadas por los fallos o defectos en los rodamientos. La frecuencia con la que se repiten estos impulsos es la que permite diagnosticar en que componente del rodamiento se ha producido el defecto. Sin embargo, en el espectro obtenido a través de la FFT, la señal predominante es la de las frecuencias resonantes excitadas.

El Envelope Analysis demodula la señal y extrae la señal envolvente (Envelope Signal) que contendrá armónicos en las frecuencias de fallo (BPFI, BPFO, BSF y FTF). Este proceso se realiza a través de la transformada de Hilbert [44].

Sin embargo, el Envelope Analysis necesita que se aplique un filtro paso-banda a la señal original para extraer la banda con la mayor información de diagnóstico. Para extraer esta banda, se ha utilizado otra técnica, el Fast Kurtogram.

Gracias al Fast Kurtogram se puede obtener la banda de frecuencias en la que la ratio señal/ruido es mayor y por tanto contiene la mayor información de diagnóstico. Cuanta más precisión se consiga detectando la banda de frecuencias que el Envelope Analysis va a demodular más fácil será detectar un fallo en un rodamiento.

En resumen, al aplicar esta técnica se extrae una parte de la señal que contiene la información de diagnóstico (analitical signal). Para aumentar el ratio señal/ruido y hacer que la detección de las frecuencias de fallo sea más eficaz, se aplica un filtro paso banda donde existe una mayor impulsividad provocada por el defecto. Esta banda se detecta mediante el Fast Kurtogram. Por último, se obtiene el espectro de la señal analítica para observar si existen harmónicos en las frecuencias de fallo de los rodamientos.

9.1 Ejemplo demostrativo

Para verificar que la técnica era capaz de localizar fallos en los datos generados por el motor eléctrico utilizado, se ha añadido a una de las mediciones una señal que simule un defecto en un rodamiento con un modelo del tipo *Periodic impulse-train.* Con este modelo se ha simulado la presencia de un defecto en la pista interior (BPFI), el cuál debe provocar la aparición de un harmónico entorno a la banda de los 96 Hz. Se ha escogido este tipo de modelo para las pruebas por ser el más sencillo de reproducir.

La señal medida por el sensor de vibración está compuesta por 3 columnas de 16384 elementos. Cada columna corresponde a las vibraciones en un eje de coordenadas. En la Ilustración 102 se muestran las señales en los 3 ejes de coordenadas antes y después de haber introducido el fallo. Se puede apreciar que no existe gran diferencia.

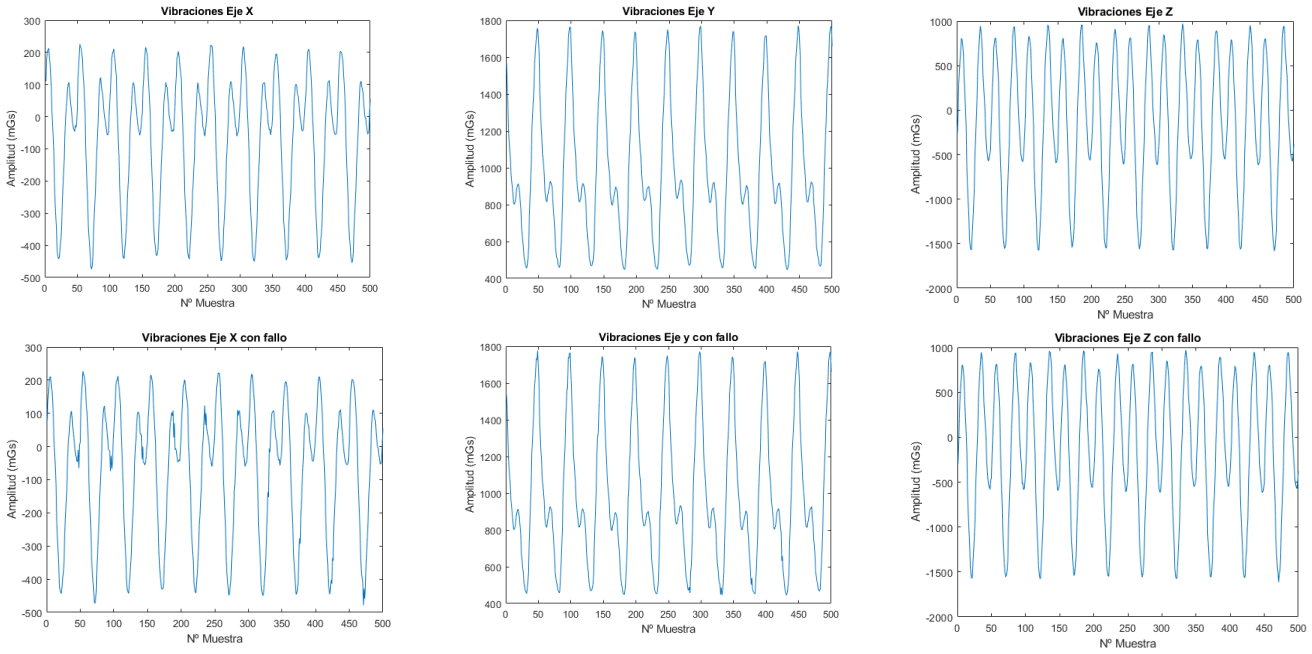

Ilustración 102: Señales en el dominio del tiempo

Si se transforman las señales al dominio de la frecuencia a través de la FFT, se aprecia que sigue sin ser haber diferencia entre las señales sin fallo y la señales con fallo. Los armónicos que predominan en todas las gráficas son los producidos por las perturbaciones introducidas por la red eléctrica en la banda de los 100Hz y sus múltiplos.

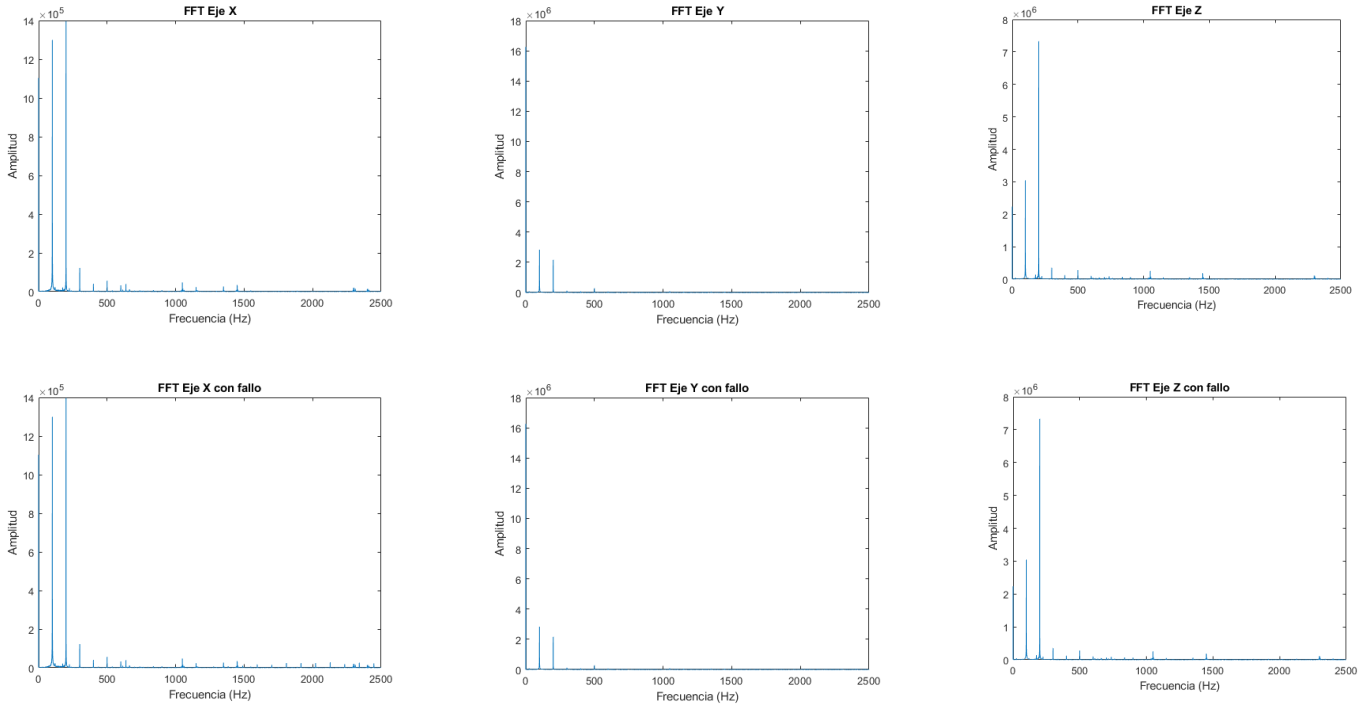

Ilustración 103: Señales en el dominio de la frecuencia

Para aplicar el Envelope Analysis sobre las señales capturadas es necesario establecer una banda donde la señal de diagnóstico tenga más presencia. Aplicando la técnica del Fast Kurtogram, se obtiene que el filtro debe eliminar todas aquellas frecuencias que no se encuentren dentro de la banda 2.0833 kHz ± 0.83333 kHz

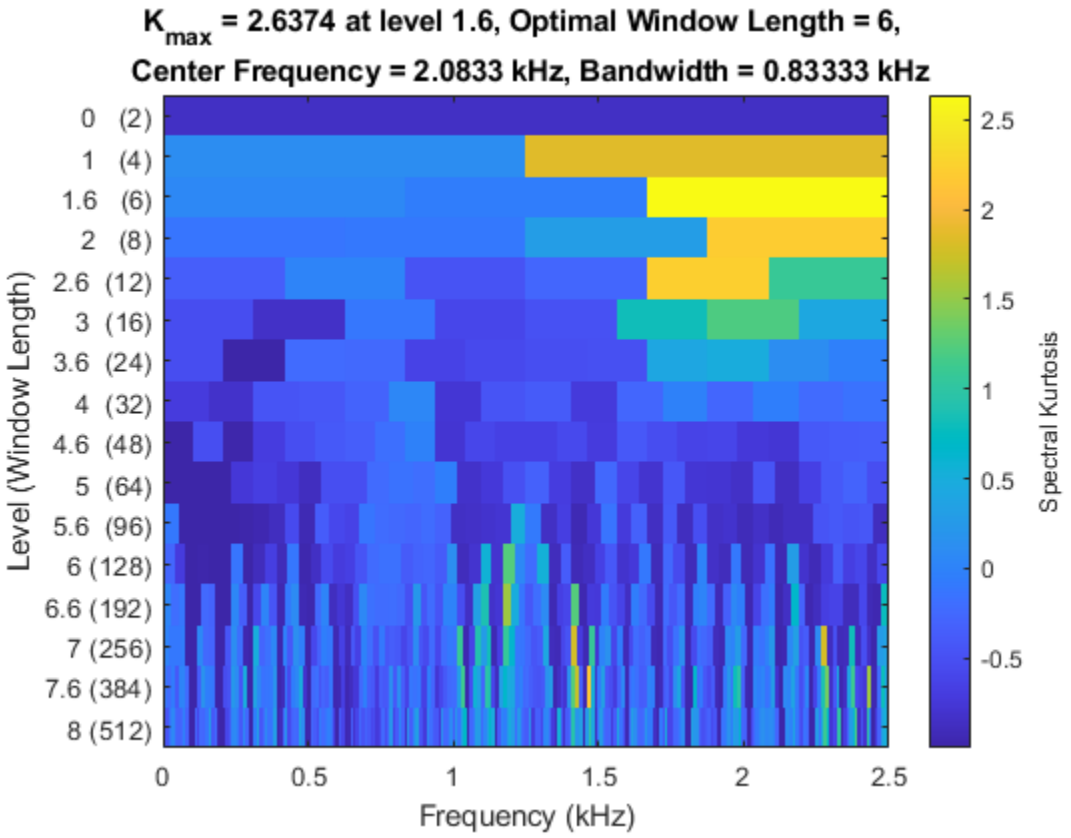

Ilustración 104: Kurtograma

El resultado de aplicar el Envelope Analysis a la señal filtrada se muestra en la Ilustración 105. En ella se han enfrentado el espectro de la señal sin fallo (Healthy) y el espectro de la señal con fallo (Damaged). Además, se han añadido líneas de puntos verticales en la frecuencia de fallo BPFI y sus múltiplos para facilitar el diagnóstico.

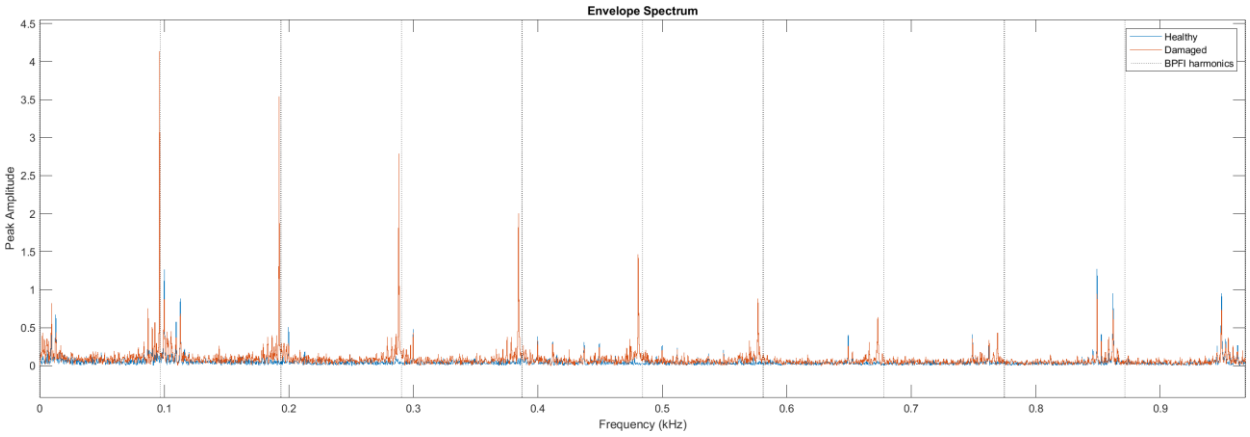

Ilustración 105: Envelope Spectrum

Se aprecia que han aparecido harmónicos en la frecuencia de fallo de la pista interior BPFI. Estos resultados pueden observarse con más claridad en la Ilustración 106.

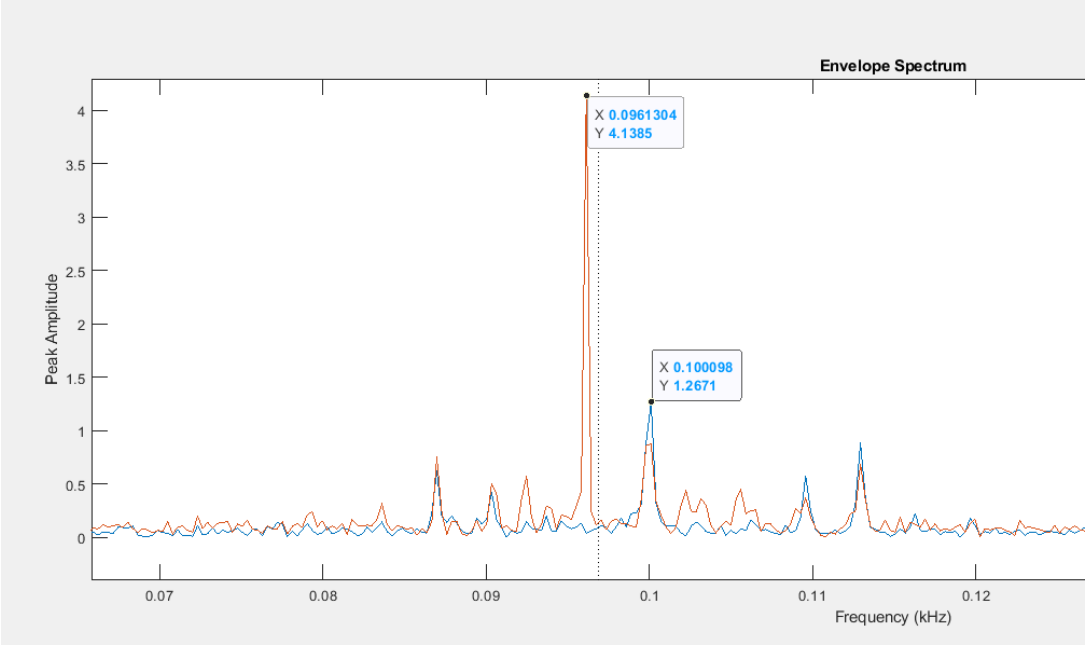

Ilustración 106: Zoom Envelope Spectrum

10 Simulación Anomalías

En este proyecto se cuenta con un motor eléctrico donde colocar el nodo multisensorial y capturar las vibraciones que produce el motor cuando se encuentra en movimiento. Sin embargo, no existe la posibilidad de simular defectos en los rodamientos de forma mecánica.

Por este motivo, desde Intermark se ha solicitado que se explore la posibilidad de simular las situaciones de fallo más características de los rodamientos de modo que, cuando sea requerido en un futuro, sea posible generar a partir de un conjunto de datos que simule una situación de fallo en los rodamientos. En concreto, el requisito consiste en simular defectos de tipo localizados.

Como se ha descrito anteriormente, existen múltiples tipos de modelos para la simulación. En **Detección Anomalías** se optó por utilizar el tipo de modelo más simple, *periodic impulse train models*. A la señal recogida por el sensor, se le añade la señal que introduciría un defecto en un rodamiento. Esta señal de fallo se genera como un una senoide que se repite con una determinada frecuencia. En la Ilustración 107 se muestra un ejemplo de la señal generada.

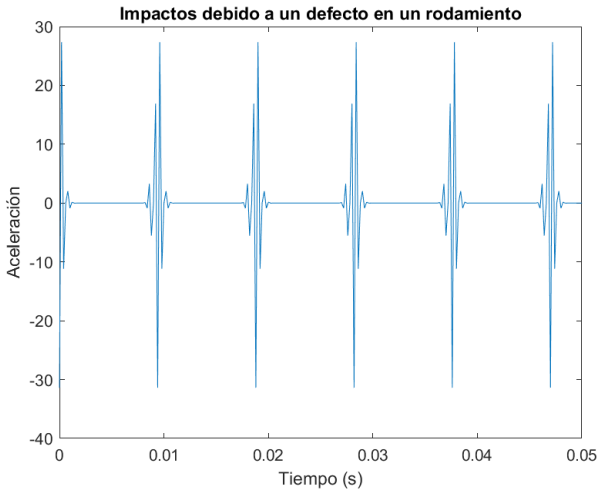

Ilustración 107: Señal de fallo de rodamiento simulada

Los detalles de la implementación se recogen en el fichero “simulacionLocalFault_V1.m” que sigue los pasos descritos en la documentación de Matlab [45].

Sin embargo, como se ha comentado anteriormente, este método no es el que más se ajusta a la realidad. Los impactos sufren variaciones aleatorias que provocan ciertas variaciones entre cada impacto.

Se ha optado por utilizar un modelo de tipo *Quasi-periodic impulse train models*. Pese a haber modelos más complejos que los *Quasi-periodic impulse train models* que pueden llegar a generar una señal más exacta, estos otros tipos de modelos se han descartado por los motivos que se describen a continuación.

En primer lugar, los *FE models* necesitan adquirir programas específicos con coste económico el cual no es posible asumir.

En segundo lugar, tanto lo *FE models* como los *Nonlinear multi-body dynamic models* son más indicados para la simulación de sistemas más complejos que cuentan con trasmisiones por engranajes.

Por último, los *Quasi-periodic impulse train models* tienen la capacidad de predecir con éxito las frecuencias significativas relacionadas con los defectos, por lo tanto, son suficiente para realizar la simulación del comportamiento del motor monitorizado [46].

En la literatura existe un gran número de modelos diferentes. En la gran mayoría de los artículos no se incorpora la implementación del modelo desarrollado. Sin embargo, en [47] además de realizarse una descripción detallada del modelo presentado, también se adjunta una implementación del modelo en Octave, haciéndolo accesible para todo el mundo sin necesidad de depender de herramientas de pago o realizar la implementación desde cero.

Con el modelo planteado es posible generar señales donde se simulen defectos localizados en la pista interior y exterior de un rodamiento. Además, permite la generación de una señal con defectos extendidos, aunque en este proyecto no se han utilizado.

Se ha realizado una comparativa entre una señal real de un rodamiento con un defecto en la pista interior y la señal simulada para observar la precisión de la simulación. En la Ilustración 108 se enfrentan la señal simulada frente a una señal real obtenida de un conjunto de datos creado por la división MFPT del Vibration Institute [48].

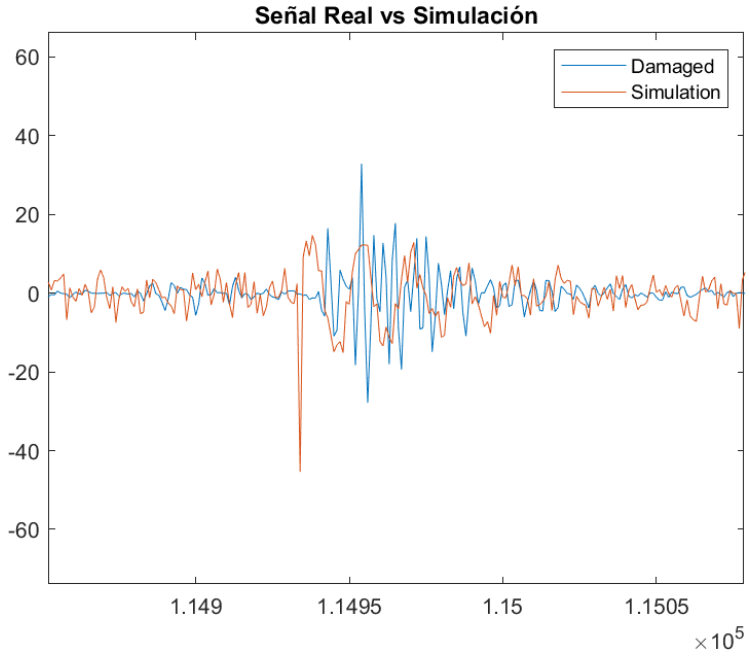

Ilustración 108: Señal real frente a simulación

Además de visualizar la señal en el dominio del tiempo, lo cual no aporta información respecto a la presencia de defectos en los rodamientos, también se muestra la comparativa de la gráfica obtenida después de aplicar el Envelope Analysis a ambas señales.

Como se puede apreciar en la Ilustración 109, en ambos espectros aparece el armónico característico de un defecto en la pista interior del rodamiento.

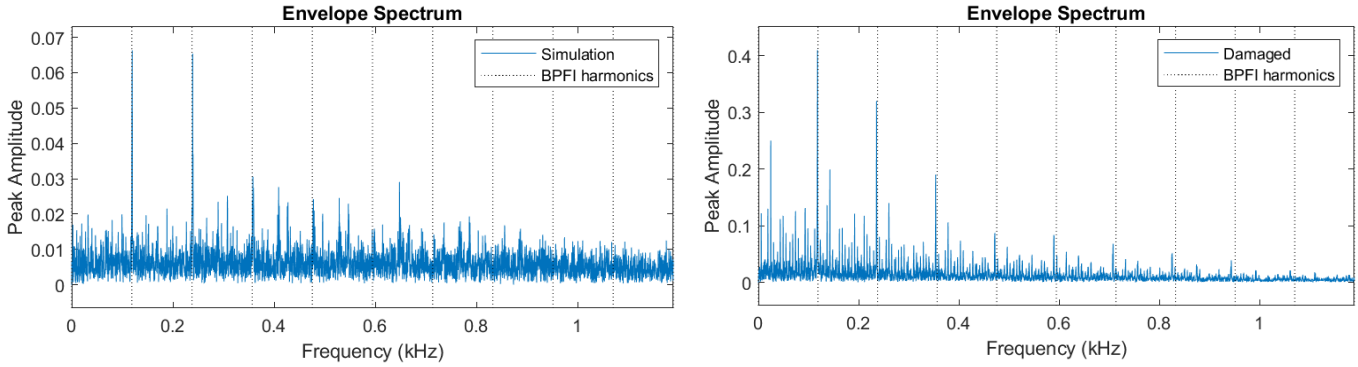

Ilustración 109: Envelope Spectrum de señal real frente a simulación

11 Versión final

En el siguiente apartado se describe la última configuración del sistema, en la que se han introducido cambios respecto de la versión presentada en el apartado **Solución pérdida de datos en la comunicación Edge-Gateway**.

11.1 Edge

Con la configuración actual del nodo multisensorial, se realizan 27.000 lecturas del acelerómetro con una frecuencia de muestreo de 5kHz.

Sin embargo, en su posterior procesado en el Fog, solo 16.384 muestras son utilizadas. Se estableció este valor porque algunas técnicas como la FFT necesitan una señal formada por un número de muestras potencia entera de base 2. Por este motivo, el tamaño de los buffers donde se almacenan las aceleraciones se ha reducido.

Debe tenerse en cuenta, aunque no es habitual, que pueden producirse pérdidas de algún paquete durante la comunicación. Si el número de muestras enviadas por el nodo es exactamente 16.384 o es inferior a este valor, no será posible realizar el cálculo correctamente cuando algún paquete de datos no llegue al destinatario. Con el objetivo de lidiar con esta posible situación, se ha establecido que el tamaño de los buffers sea de 17.000 muestras. De este modo, aunque se produzca la pérdida de algún paquete durante la comunicación, se asegura que el total de paquetes sea mayor que 16.384.

Además de las vibraciones, esta versión del firmware también se ha configurado para que el nodo realice lecturas de temperatura. En concreto, cada vez que comience el proceso de comunicación entre nodo y gateway, se realiza un envío de temperatura.

```
//Buffer Ingest
if(iterAcc<17000){
    BufferAx[iterAcc]=ACC_Value.x;
    BufferAy[iterAcc]=ACC_Value.y;
    BufferAz[iterAcc]=ACC_Value.z;
    iterAcc++;

}

//If buffer is full send via BLE
if((iterAcc==17000)){

    int16_t TempToSend;
    ReadTemperatureData(&TempToSend);
    Temperature_Update(TempToSend);
    HAL_Delay(90);
    for(int pos=0;pos<680;pos++){

        Acc_Update(&BufferAx[25*pos],&BufferAy[25*pos],&BufferAz[25*pos],pos);
        HAL_Delay(90);
    }

    //Reset iterator value
    iterAcc=0;
    //Wait for gateway to process buffer
    HAL_Delay(12000);
}
```

Ilustración 110: Procesado de vibraciones y temperatura en el nodo

Ha sido necesario implementar 2 funciones auxiliares, ReadTemperatureData() para la lectura del valor y Temperature_Update() para exponer el dato vía BLE. Estas funciones, cuyo código fuente se muestra en la Ilustración 111 y la Ilustración 112, se han definido en los ficheros “main.c” y “sensor_service.c” respectivamente.

```
/**
 * @brief  Read The Environmental Data (Temperature)
 * @param  int16_t *TempToSend pointer to Temperature Value
 * @retval None
 */
void ReadTemperatureData(int16_t *TempToSend)
{
    float SensorValue;
    int32_t decPart, intPart;

    *TempToSend=0;

    /* Read Temperature */
    BSP_ENV_SENSOR_GetValue(STTS751_0,ENV_TEMPERATURE,&SensorValue);
    MCR_BLUEMS_F2I_1D(SensorValue, intPart, decPart);
    *TempToSend = intPart*10+decPart;
}
```

Ilustración 111: Función ReadTemperaureData

```
/**
 * @brief  Update Temperature characteristic value
 * @param  int16_t Temp Temperature in tenths of degree
 * @retval tBleStatus   Status
 */
tBleStatus Temperature_Update(int16_t Temp)
{
  tBleStatus ret;

  uint8_t buff[2+2/*Temp*/];

  STORE_LE_16(buff  ,(HAL_GetTick()>>3));
  STORE_LE_16(buff+2,Temp);

  ret = ACI_GATT_UPDATE_CHAR_VALUE(HWServW2STHandle, EnvironmentalCharHandle, 0, 4,buff);

  if (ret != BLE_STATUS_SUCCESS){
    if(W2ST_CHECK_CONNECTION(W2ST_CONNECT_STD_ERR)){
      BytesToWrite =sprintf((char *)BufferToWrite, "Error Updating Environmental Char\n");
      Stderr_Update(BufferToWrite,BytesToWrite);
    } else {
      STBOX1_PRINTF("Error Updating Environmental Char\r\n");
    }
  }
  return ret;
}
```

Ilustración 112: Función Temperature_Update

11.2 Gateway

Los cambios introducidos en el firmware del sensor provocan variaciones en los datos enviados y es necesario adecuar el script que contiene la lógica para su adaptación al nuevo formato.

Los principales cambios introducidos y la implementación de las funciones utilizadas para el almacenamiento de los datos en el gateway y la nube se describen en este apartado.

Estas variaciones se encuentran disponibles en el script “Acc_bbdd_cloud_V5.py”

11.2.1.1 Cambios principales

El primer cambio introducido respecto a lo descrito en apartados anteriores es el tamaño del buffer que almacena las aceleraciones. Este pasa a ser de 17.000 elementos, por tanto, el gateway debe esperar a recibir un buffer de este tamaño y no del tamaño establecido previamente, 27.000 elementos.

En segundo lugar, debe ser capaz de procesar las notificaciones BLE procedentes de los sensores ambientales y así poder recibir y procesar las lecturas de temperatura.

Las notificaciones de los sensores ambientales utilizan el manejador con valor 17. Por lo tanto, cuando la variable cHandle tenga este valor el paquete BLE contiene 2 bytes con un identificador y 2 bytes con el valor de temperatura.

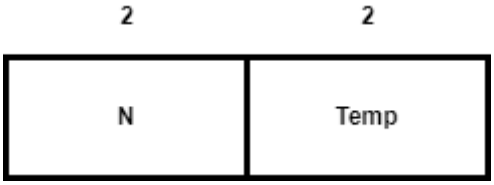

Ilustración 113: Estructura paquete BLE sensor ambiental

Una vez se ha obtenido el valor de temperatura, se envía hacía el cloud. En la Ilustración 114 se muestra el proceso descrito.

```
#Manejador de temperatura y humedad
if cHandle == 17:
    #Procesamiento de la temperatura
    datostemp = struct.unpack('<h',data[2:4])
    temp=datostemp[0]/10.0
    #Obtencion del timestamp
    ts_temp=str(time.time_ns())
    #Imprimimos por pantalla
    #print(str(datetime.now()) + " --> "+  "Temperatura: " + str(datostemp[0]/10.0) + " ºC")
    #Almacenamiento en gateway
    guardarBBDDTemp(temp,ts)
    #Envio a Thingsboard
    envioCloudTemp(temp,_ACCESS_TOKEN_TEMP,ts)
```

Ilustración 114: Manejador de interrupciones ambientales

11.2.2 Implementación de funciones

Una vez obtenidos los datos relativos a las aceleraciones se realizan 2 tareas. Por un lado, se almacenan en una base de datos no relacional en el gateway como se ha explicado en **Almacenamiento en el Gateway** . Por otro lado, se realiza el Envelope Analysis de la señal para detectar la existencia de algún tipo de anomalía y se envía al cloud tanto el espectro obtenido y si se ha detectado algún tipo de anomalía en un rodamiento.

Anteriormente se había definido la función calculaFFT() donde se transforma al dominio de la frecuencia la señal obtenida del sensor de aceleraciones. Además de esto, previamente se debe realizar el Envelope Analysis para aislar el fallo, en caso de que existiera, para su posterior detección.

Estas funcionalidades se han añadido a la función calculaFFT(). En lugar de realizar directamente la transformación, se realizan las operaciones descritas en el apartado **Detección Anomalías**, filtrando la señal en la banda obtenida al aplicar el Fast Kurtogram

[49] y realizando la transformada de Hilbert. Esta implementación se muestra en la Ilustración 115.

```
#Kurtograma
Kwav, Level_w, freq_w, c, max_Kurt, bandwidth, level_max = fast_kurtogram(data, _FREQUENCY_HZ, nlevel=6)
#Frecuencia centro y bandwidth
minw = np.where(Level_w == level_max)[0][0]
kurtw = np.where(Kwav[minw, :] == max_Kurt)[0][0]
bandw = freq_w[kurtw]
Fc=bandw + bandwidth/2

#Banda de filtrado
lowCut=Fc-bandwidth
highCut=Fc+bandwidth

#Filtrado señal
signal = butter_bandpass_filter(data, lowCut, highCut, 5000, order=6)

#Calculo del la envolvente de la señal mediante la transofrmada de Hilbert
analytic_signal = hilbert(signal)
amplitude_envelope = np.abs(analytic_signal)
```

Ilustración 115: implementación Envelope Analysis

En último lugar, se analiza la señal para realizar el diagnóstico. Con las especificaciones técnicas de la geometría del rodamiento y las fórmulas detalladas en la Tabla 3 es posible conocer las frecuencias en las que pueden aparecer armónicos que revelen la presencia de fallos en los rodamientos.

En la Ilustración 116 se muestra la implementación del cálculo de las frecuencias de fallo para el rodamiento equipado en el motor Cemer utilizado. Estos valores pueden variar en función del tipo de rodamiento monitorizado.

```
#Parametros del rodamiento
n = 6;          # Numero de bolas
d = 0.11;       # Diametro de las bolas
p = 0.20;       # Diametro de la pista
thetaDeg = 15*np.pi/180; # Angulo de contacto
fPin=23.33;     # Frecencia del eje

#Formulas de calculo de frecuencia de fallo
bpfi = n*fPin/2*(1 + d/p*np.cos(thetaDeg))
bpfo = n*fPin/2*(1-d/p*np.cos(thetaDeg))
bsf  = fPin*p/d*(1-(d/p*np.cos(thetaDeg)**2))
ftf  = fPin/2*(1-(d/p*np.cos(thetaDeg)))
```

Ilustración 116: Cálculo de las frecuencias de fallo

Para realizar el diagnóstico se buscan en el espectro armónicos que revelen la presencia de fallos. Se ha optado por analizar la energía de la señal en cada una de las frecuencias de fallo, de modo que si se supera un umbral de energía se puede concluir que existe un fallo. En función de la frecuencia en la que se sobrepase este umbral, es posible conocer el tipo de fallo detectado.

Para el cálculo de la energía de los armónicos se utiliza el RMS (Root Mean Square). Pero las frecuencias proporcionadas por las fórmulas del cálculo de las frecuencias de

fallo no son exactas, por tanto, no se busca el defecto en una única frecuencia si no que se analiza una banda con frecuencia centro la frecuencia de fallo y un ancho de banda de ± 10 Hz.

Para implementar la detección de fallos se ha creado la función buscaFallo(). Esta función analiza las bandas que contienen la información de diagnóstico. Si el RMS obtenido supera el umbral definido, se activa el flag de fallo detectado en la posición que corresponde a ese tipo de fallo. Este vector está formado por 4 flags, que corresponden a los cuatro tipos de fallos detectables en los rodamientos (bpfi, bpfo, bsf y ftf).

```
def buscaFallo(frec,tr,faults):
    #Flags
    fFlags=[0,0,0,0]
    for i,f in enumerate(faults):
        idx=np.where((frec>f-10) & (frec<f+10))
        rms=np.sqrt(np.sum((2*(tr[idx])**2)))/len(idx)
        if(rms>_FAULT_THRESHOLD):
            fFlags[i]=1
    return fFlags
```

Ilustración 117: función buscaFallo

La función buscaFallo() retorna el vector de flags. Si este vector contiene algún flag activo, se realiza un envío al cloud detallando que existe un fallo y de que tipo es. De este modo el cloud puede generar una alarma como se describe en el apartado **Alarmas acelerómetro.**

```
#Buscamos Frecuencias de fallo en el espectro
bF=buscaFallo(frec,tr,faults)

#Buscamos flags activos y enviamos en caso de encontrar un 1
for i,f in enumerate(bF):
    #Descripcion de tipo de fallo
    faultType=['bpfi','bpfo','bsf','ftf']
    #Si flag igual a 1 enviamos al cloud
    if(f==1):
        envioFalloCloud(faultType[i],at)
```

Ilustración 118: envió fallo al Cloud

La función envioFalloCloud() se encarga de realizar el envío de la detección del fallo al cloud a través del protocolo MQTT. Además de enviar el atributo fault, de tipo booleano, donde se indica que se ha detectado un fallo, se envía el atributo faultType donde se especifica el tipo de fallo. En este atributo podría añadirse una descripción más detallada para ser mostrada en el panel de alarmas de Thingsboard.

```
#Funcion para enviar datos de temperatura a thingsboard
def envioCloudTemp(temp,at,ts):
    #Rellenado del payload con el formato {clave1:valor, clave2:valor...}
    payload={"ts":str(ts)[0:13],"values":{"temp":temp}}
    #Transformación a formato JSON
    payload=json.dumps(payload)
    #Envio a thingsboard de la temperatura
    subprocess.run(['bash','./mqttClientV3.sh',payload, at])
```

Ilustración 119: Función EnvioFalloCloud

En última instancia, se realiza el envío del espectro al cloud. Esta tarea se realiza con la función envioCloud(). En las primeras versiones de esta función, se realizaba un envío secuencial de cada uno de los valores de la FFT con el formato frecuencia:amplitud. Sin embargo, este proceso se prolongaba en el tiempo excesivamente.

Con el objetivo de reducir el tiempo de comunicación entre el gateway y el cloud, se optó por enviar todos los datos en un fichero JSON. Thingsboard únicamente permite realizar envíos de ficheros JSON través del protocolo MQTT si se utiliza el formato {“ts”:timstamp,”values”:{“key1”:valor, “key2”:valor, “keyN”:valorN }} y el tamaño del fichero no excede los 52 bytes.

Si la totalidad de elementos que componen la FFT se envían en fichero JSON, el tamaño máximo permitido se sobrepasa y la conexión es rechazada. Para solventar este inconveniente, se ha realizado un particionado de los datos en ventanas de 512 elementos. De este modo, en lugar de enviar un fichero con toda la información, se envían 32 ficheros con un tamaño menor inferior al tamaño máximo permitido.

Para la creación del fichero, se declara una variable de tipo string a la que se le añade cada uno de los valores de la obtenidos tras haber aplicado el Envelope Analysis dentro de esa ventana. Una vez se han insertado los 512 valores, se almacena el contenido del JSON en un fichero y se realiza el envío. A continuación, se procede a enviar los siguientes 512 elementos hasta realizar el envío al completo del espectro.

Dado que el contenido del campo values es un string y no un conjunto de pares clave-valor, es necesario eliminar las comillas entre las que se encuentran cada uno de los campos y así conseguir el formato deseado. La forma más sencilla de realizarlo es mediante el comando de Linux “awk” antes de realizar el envío mediante el cliente MQTT.

```python
#Funcion para enviar datos a thingsboard en ventanas de 512 elementos
def envioCloud(frec,fft,at,eje,ts):
    #Vector con los indices de las ventanas
    ventana=np.arange(0,len(frec)+1,_WINDOW_STEP)
    #Envio de la fft en ventanas de 512 elementos
    for i in range(len(ventana)-1):
        #Inicializacion payload
        payload="{ts:"+str(ts)[0:13]+",values:{"
        #Rellenado del payload con el formato {clave1:valor, clave2:valor...}
        for j in range(ventana[i],ventana[i+1]):
            payload=payload + str(frec[j]) + ":" + str(fft[j]) + ","
        #Eliminamos la utlima coma
        payload=payload[0:-1]
        #Cierre llaves objeto
        payload+='}}'
        #Transformación a formato JSON
        payload=json.dumps(payload)
        #Almacenamiento en fichero
        jsonFile = open("dataAcc.json", "w")
        jsonFile.write(payload)
        jsonFile.close()
        #Envio a thingsboard del payload
        subprocess.run(['bash','./mqttClientV3.sh',"dataAcc.json", at])
```

Ilustración 120: Función envioCloud con envío particionado

Tanto el procesamiento del fichero JSON como el envío de los valores de la FFT o de temperatura se realizan en el fichero "mqttClientV3.sh".

```
#!/bin/sh
if [ $1 = "dataAcc.json" ]; then
        #echo "primera rama"
    cat "$1" | awk '{print substr($0, 2, length($0) - 2)}' > "dataAccProc.json"
    mosquitto_pub -d -q 1 -h "156.35.163.171" -t "v1/devices/me/telemetry" -u "$2" -f "dataAccProc.json"
else
    mosquitto_pub -d -q 1 -h "156.35.163.171" -t "v1/devices/me/telemetry" -u "$2" -m "$1"
fi
```

Ilustración 121: Contenido del fichero mqttClientV3.sh

Se ha diferenciado entre 2 situaciones. La primera, que se trate del fichero dataAcc.json, en cuyo caso, es necesario eliminar las comillas. En caso contrario, no es necesario realizar ningún tratamiento especial del fichero antes del envío y se puede realizar.

La función utilizada para realizar el envío de temperatura a Thingsboard, envioCloudTemp() es más sencilla que su homónima, ya que al componerse de un único elemento clave-valor es posible componer el fichero JSON de forma directa.

```python
#Funcion para enviar datos de temperatura a thingsboard
def envioCloudTemp(temp,at,ts):
    #Rellenado del payload con el formato {clave1:valor, clave2:valor...}
    payload={"ts":str(ts)[0:13],"values":{"temp":temp}}
    #Transformación a formato JSON
    payload=json.dumps(payload)
    #Envio a thingsboard de la temperatura
    subprocess.run(['bash','./mqttClientV3.sh',payload, at])
```

Ilustración 122: Función envioCloudTemp

11.2.2.1 Detección estado del motor

Se ha observado que el Envelope Analysis proporciona resultados confusos cuando el motor se encuentra parado. Para evitar un diagnóstico erróneo y evitar la realización de cálculos innecesarios, ya que cuando no existe vibración no es posible detectar defectos en los rodamientos, se ha optado por no aplicar el Envelope Analysis sobre las señales en las que se detecte que no se ha producido vibración.

Cuando no hay movimiento del eje del motor, teóricamente, el sensor no debería registrar variaciones en las aceleraciones medidas. Sin embargo, como se observa en la Ilustración 123, existe una pequeña vibración que puede provocar diagnósticos equivocados.

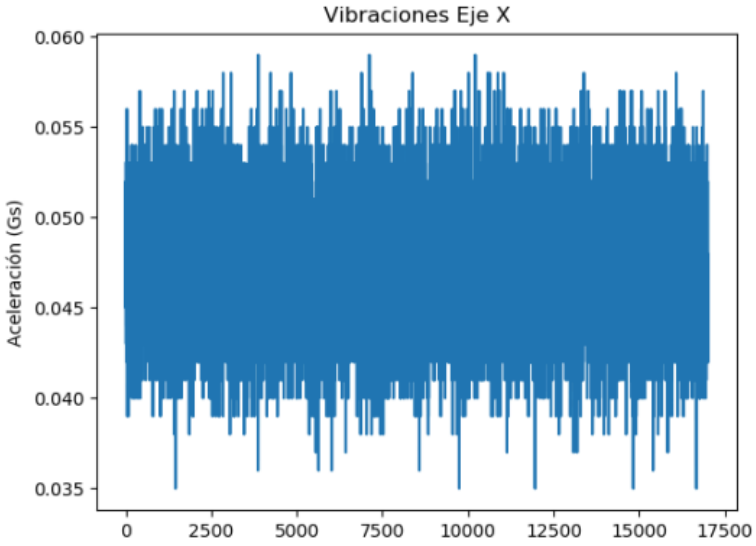

Ilustración 123: Vibraciones motor parado

La variación de un datos respecto a la media es pequeña, por tanto, con un indicador como la desviación típica se puede conocer el estado en el que se encuentra el motor. En la Tabla 9 se muestran los valores de la desviación cuando el motor se encuentra parado o en movimiento.

	Motor Parado	**Motor Movimiento**
Valor de la desviación típica	±3	±150

Tabla 9: Desviación típica de las señales

La implementación de esta lógica se ha introducido dentro de la función handleNotification(). Cuando se ha completado el envío de un buffer, se comprueba que la desviación típica de la señal supere el umbral establecido. En caso de no superarlo, se envía una notificación al cloud para cambiar el estado en el panel informativo. En caso de detectar que la señal contiene información útil para el diagnóstico, se procede a realizar el Envelope Analysis. En la Ilustración 124 se muestra la implementación de esta lógica.

```
            if (datosAcc[0]==0):
                #Obtener marca de tiempo
                ts_acc=time.time_ns()
                #print("\nFecha dato recibido: " + str(datetime.now()))
                #print("\nMarca de tiempo del primer envío: " + str(ts_acc))
                print("\nTamaño del buffer: " + str(len(self.datosFFTEjeX)))

                #Si el buffer está completo
                if (len(self.datosFFTEjeX) >= _DATA_BUFFER):
                    if(np.std(self.datosFFTEjeX)<_STD_THRESHOLD):
                        print('\nMotor parado')
                        self.datosFFTEjeX=[]
                        self.datosFFTEjeY=[]
                        self.datosFFTEjeZ=[]
                        motorParadoCloud(True)
                    else:
                        print("Llamar a las funciones que calculan la FFT")
                        motorParadoCloud(False)
                        #Exportar datos a fichero
                        #guardaFichero(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,self.co
                        #Almacenamiento de vibraciones en gateway
                        guardarBBDD(self.datosFFTEjeX,self.datosFFTEjeY,self.datosFFTEjeZ,ts_acc)
                        #Calculo de FFT y subida a la nube
                        #print("Calculo FFT y subida a la nube")
                        calculaFFT(self.datosFFTEjeX, 'X', self.count,ts_acc,_ACCESS_TOKEN_X)
                        #calculaFFT(self.datosFFTEjeY, 'Y', self.count,ts_acc,_ACCESS_TOKEN_Y)
                        #calculaFFT(self.datosFFTEjeZ, 'Z', self.count,ts_acc,_ACCESS_TOKEN_Z)
                        self.count+=1
                        print("Fin procesamiento ACC")
```

Ilustración 124: Implementación cálculo de la desviación típica

Para el envío del estado, Parado o Movimiento, al cloud se ha creado una función donde se realiza un envío a través del protocolo MQTT con un fichero JSON donde se indica con un valor booleano si el motor se encuentra o no parado.

```
#Funcion para enviar al cloud estado del motor
def motorParadoCloud(estado):
    if(estado):
        #Rellenado del payload
        payload={"estado":"Parado"}
    else:
        #Rellenado del payload
        payload={"estado":"Movimiento"}

    #Transformación a formato JSON
    payload=json.dumps(payload)
    subprocess.run(['bash','./mqttClientV3.sh',payload,_ACCESS_TOKEN_MOTOR],
                                    stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
```

Ilustración 125: Implementación función motorParado

11.3 Cloud

Thingsboard está diseñado para el manejo de series temporales, y, por tanto, los gráficos que se proporcionan no permiten la representación de los datos en el dominio de la frecuencia ya que son gráficos de evolución en el tiempo o último valor.

Para poder crear una visualización de los datos en el dominio de la frecuencia se ha optado por utilizar una herramienta adicional, Grafana [50].

Grafana es una herramienta de código abierto que permite la consulta, visualización, alerta y estudio de datos a través de paneles de control independientemente de la ubicación de estos. Esta herramienta, pese a estar orientada a datos de series temporales permite el uso de librerías adicionales para crear diferentes representaciones y gráficos.

11.3.1 Instalación Grafana

Para la instalación de Grafana se han seguido los pasos descritos en la documentación disponible en la web [51].

Una vez lanzado el servicio, es posible acceder a la interfaz gráfica a través un navegador web en la dirección http://156.35.163.171:3000/. Desde este interfaz es posible realizar las diferentes tareas de administración: agregar plugins, crear y editar paneles, agregar datasources…

Grafana, al igual que Thingsboard, está orientado a la visualización de series temporales y no proporciona por defecto gráficos que permitan la representación de la señal en el dominio de la frecuencia. Sin embargo, con el uso de un plugin adicional, es posible crear un panel que se adapte a datos de una naturaleza diferente.

En concreto, el plugin instalado ha sido Plotly. Este plugin permite renderizar métricas utilizando la librería para JavaScript Plotly [52].

Para su instalación, se debe acceder al buscador de plugins de Grafana en el menú de configuración y buscar el plugin mencionado.

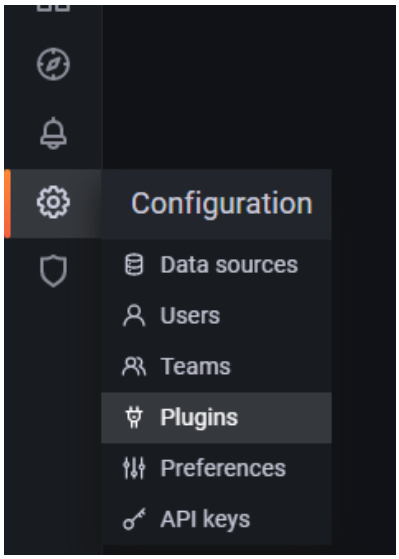

Ilustración 126: Menú configuración Grafana - Plugins

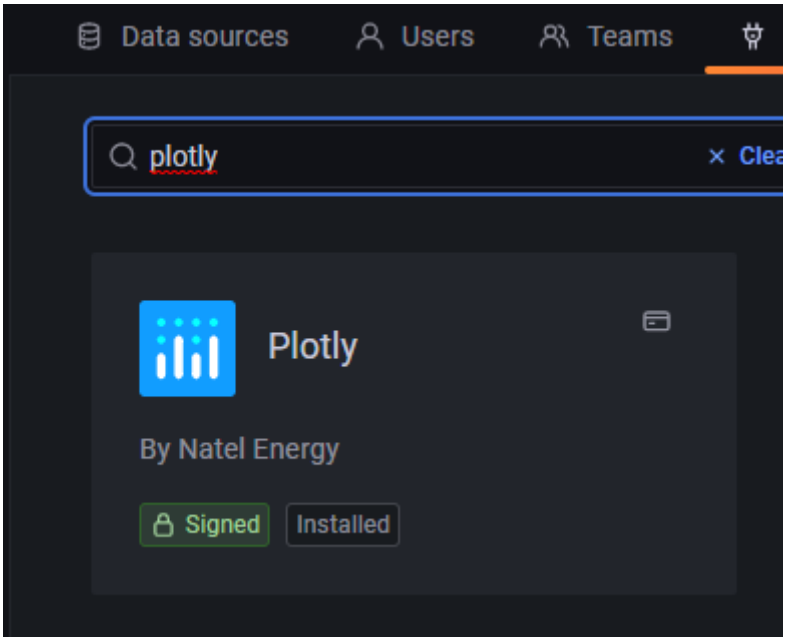

Ilustración 127: Plugin Plotly

El siguiente paso es configurar la fuente de datos de modo que los paneles que se creen en el futuro recojan la información de la base de datos de Thingsboard. Esta acción se realiza en el apartado Data sources del menú de configuración.

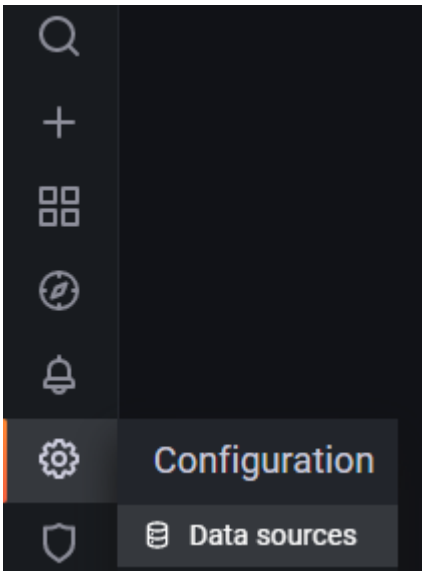

Ilustración 128: Menú configuración Grafana - Data sources

En primer lugar, hay que indicar el tipo de fuente de datos. Como Thingsboard utiliza una base de datos de tipo PostgreSQL, debe indicarse el mismo tipo en Grafana.

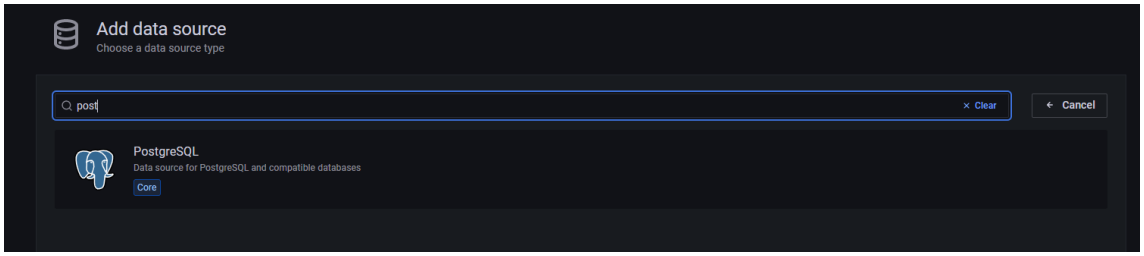

Ilustración 129: Agregar data source Grafana

Para que Grafana pueda conectarse a la base de datos de Thingsboard y acceder a la información que esta contiene, es necesario configurar el hostname, el puerto y el usuario con el que se accederá. La configuración realizada se muestra en la Ilustración 130.

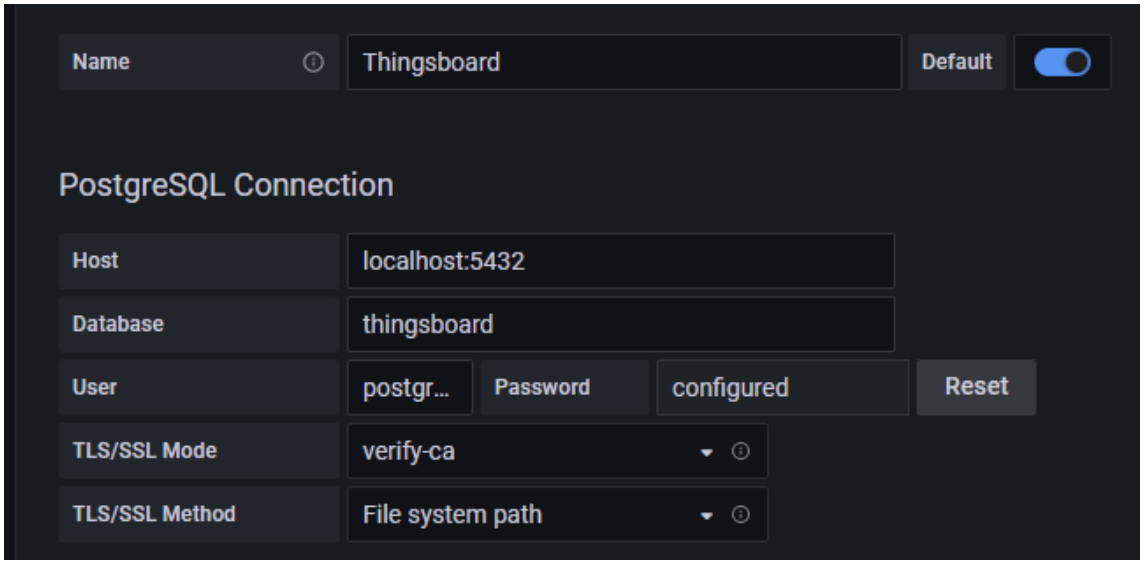

Ilustración 130: Configuración data source Grafana

En este momento ya es posible crear un dashboard donde se muestren la FFTs calculadas. Esta acción se realiza en el apartado Create Dashboard del menú lateral.

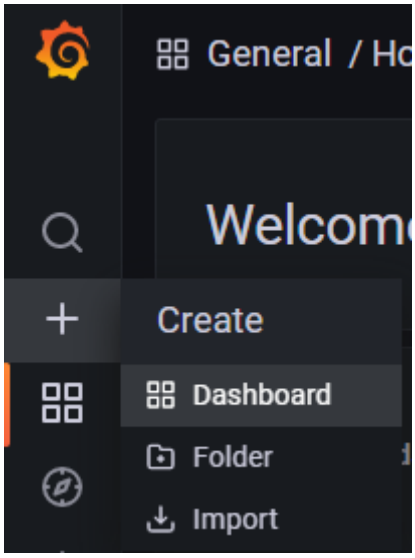

Ilustración 131: Menú configuración Grafana - Dashboard

Dentro de este dashboard se crearán 3 paneles donde se muestren las gráficas de los espectros calculados para las aceleraciones medidas en los ejes X, Y y Z.

En la creación de nuevos paneles deben indicarse principalmente 3 elementos:

- ***Data source:*** Fuente de datos donde se almacena la información que se quiere representar. Se debe indicar la fuente de datos configurada previamente.

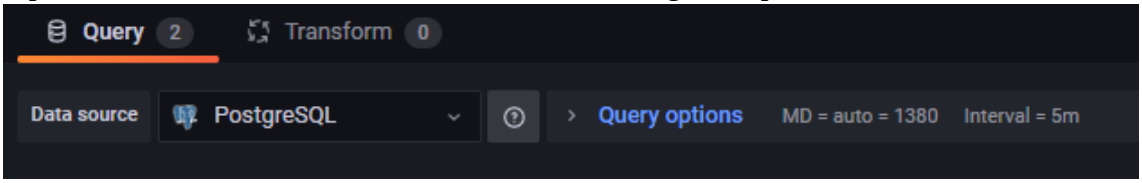

Ilustración 132: Data source Grafana

- **Query:** por medio del generador de querys, debe indicarse que datos en concreto se quieren visualizar. En el caso de la FFT será necesario crear 2 querys. La primera de ellas extrae cada una de las frecuencias que forman el vector de frecuencias. La segunda obtiene el valor de la amplitud de cada una de las frecuencias.

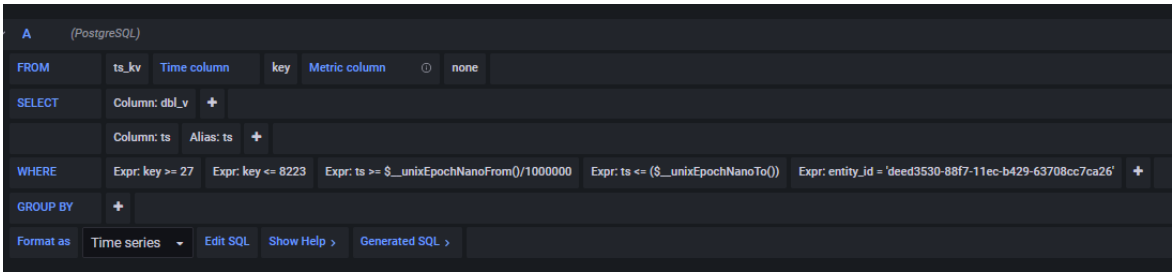

Ilustración 133: Query A Grafana

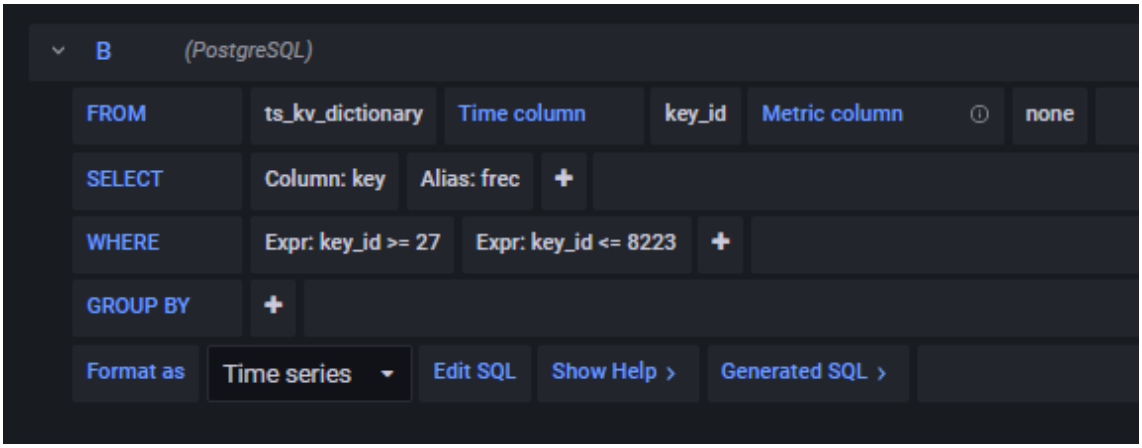

Ilustración 134: Query B Grafana

- **Tipo de visualización:** tipo de gráfico que se utiliza. Por defecto, Grafana utiliza el tipo serie temporal. Para la representación de la FFT el tipo de visualización ha de ser "Plotly."

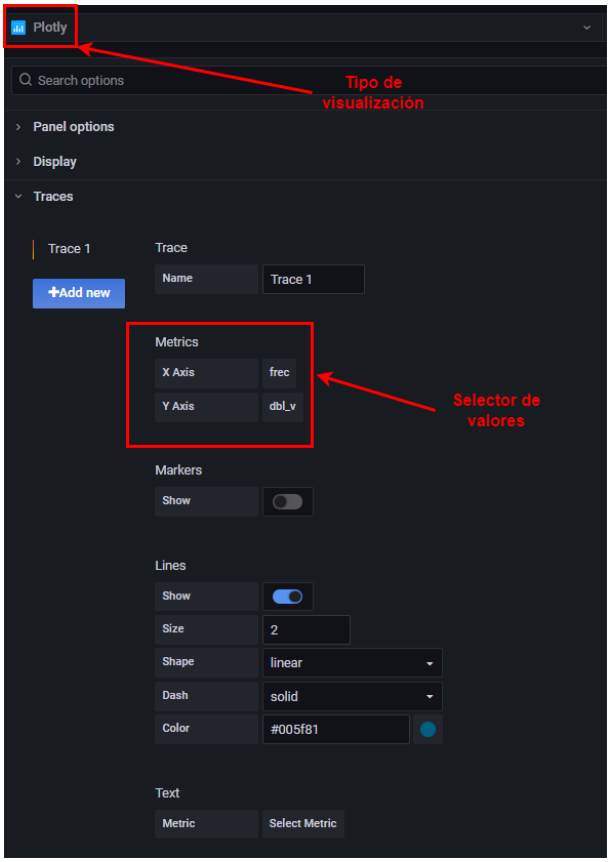

Ilustración 135: Tipo visualización Grafana

Adicionalmente, en la configuración del panel deben indicarse los valores que se representan seleccionados con el generador de querys. En la Ilustración 136 se muestra la vista resumen del configurador de querys de Grafana.

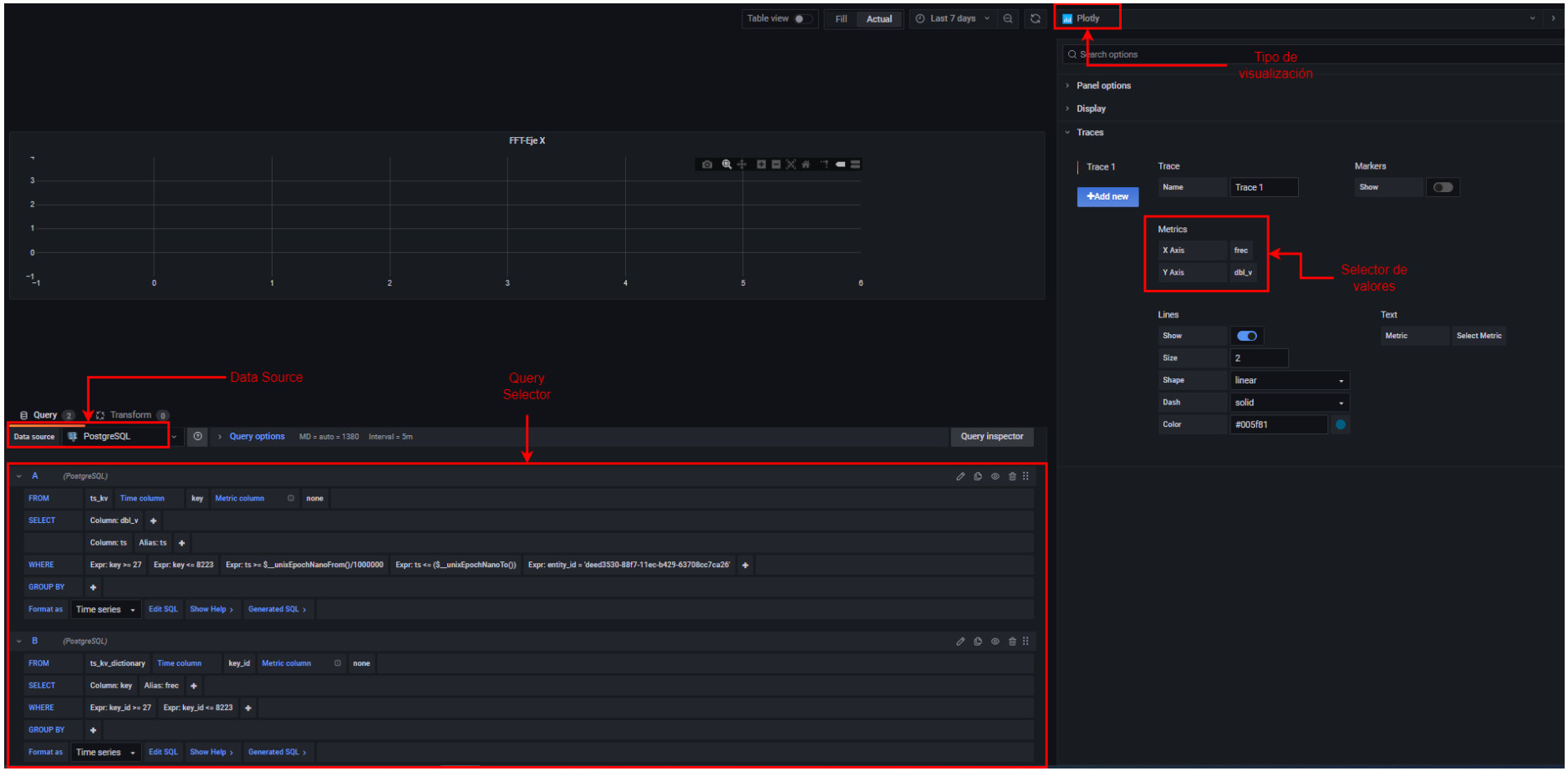

Ilustración 136: Vista general Grafana

11.3.2 Generación Query

La base de datos de Thingsboard está formada por múltiples tablas con una determinada nomenclatura donde se almacena información sobre dispositivos, alarmas, cadena de reglas, configuraciones, telemetría …

Existen múltiples tablas, pero las más reseñables sobre las cuales se realizan las consultas para la visualización de los datos son las siguientes:

- ***device:*** contiene la información de cada uno de los dispositivos creados.
- ***ts_kv:*** esta tabla almacena todas las telemetrías de todos los dispositivos creados recibidas.
- ***ts_kv_latest:*** esta tabla almacena el último valor de telemetría de todos los dispositivos creados.
- ***ts_kv_dictionary:*** esta tabla almacena los ids asociados a cada una de las claves de los pares clave-valor de telemetría.

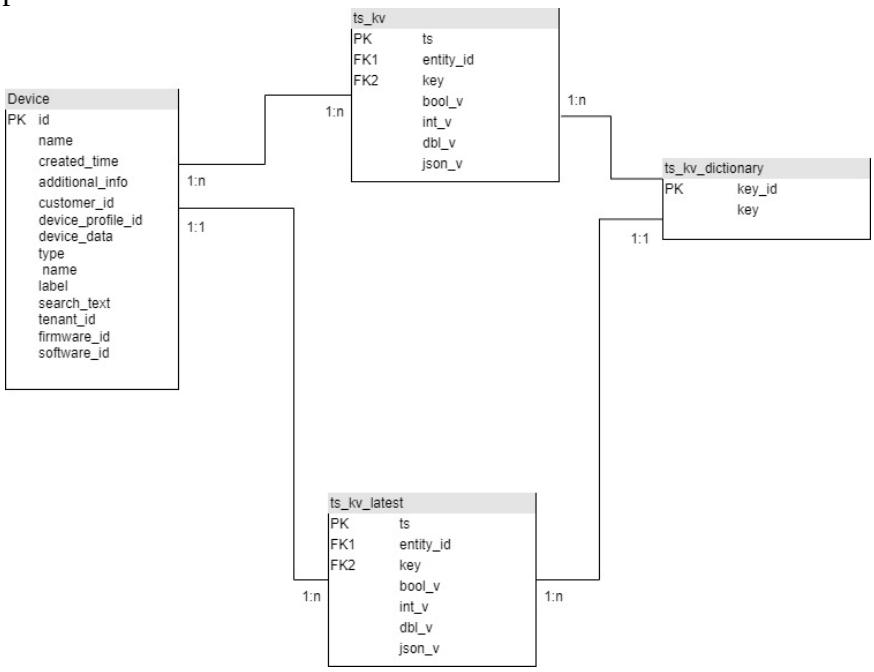

Ilustración 137: Entidades base de datos de Thingsboard

Para representar en el panel los datos almacenados en Thingsboard se deben generar 2 querys. En la primera, se extrae el valor de la amplitud de cada una de las frecuencias que componen el espectro. Estos valores se almacenan en la columna “dbl_v” de la tabla “ts_kv_latest”. Además, se debe especificar los id de las claves pertenecientes a las frecuencias para evitar que se obtengan otros valores no deseados. La relación id-clave se encuentra en la tabla “ts_kv_dictionary”. En último lugar, se debe especificar el id del dispositivo correspondiente a Acelerómetro-Eje X.

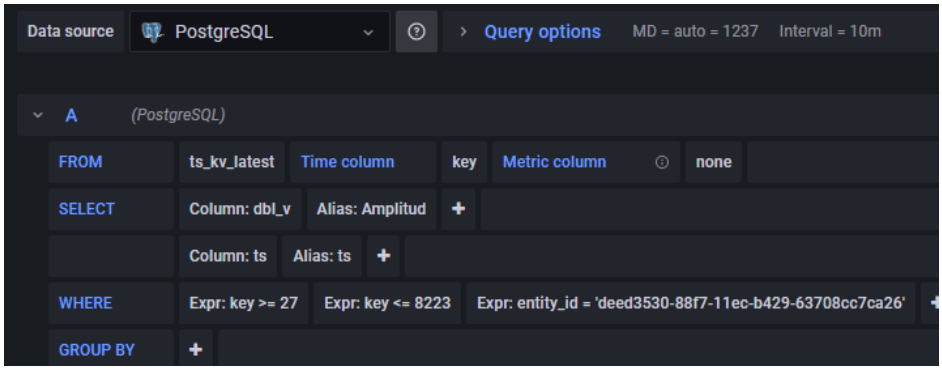

Ilustración 138: Ejemplo de configuración query

En la segunda, se extrae el valor de cada una de estas frecuencias. Esta consulta debe extraer de la "tabla ts_kv_dictionary" las claves de los pares clave-valor almacenados en la tabla "ts_kv_latest". Por este motivo, el filtro de del atributo "key_id" ha de ser idéntico al de la consulta anterior.

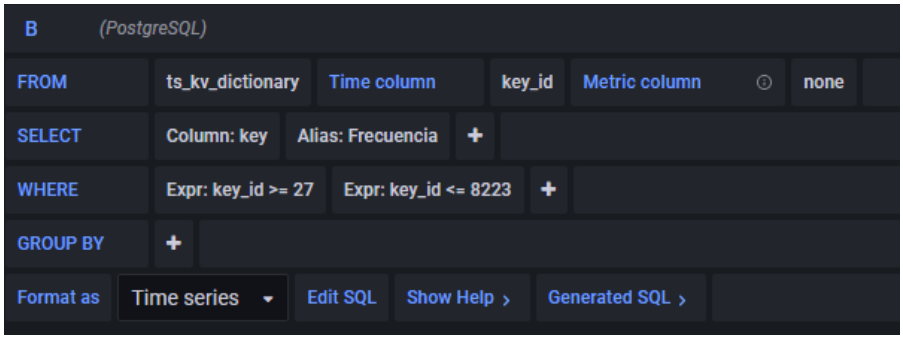

Ilustración 139: Ejemplo de configuración query II

En último lugar, se debe indicar que datos representar en el panel. Esta acción se realiza en la configuración del Tipo de visualización, en el apartado Traces. En el eje Y del panel se deben seleccionar los valores obtenidos en la primera consulta (Amplitud). En el eje de X, los valores obtenidos con la segunda consulta (Frecuencias).

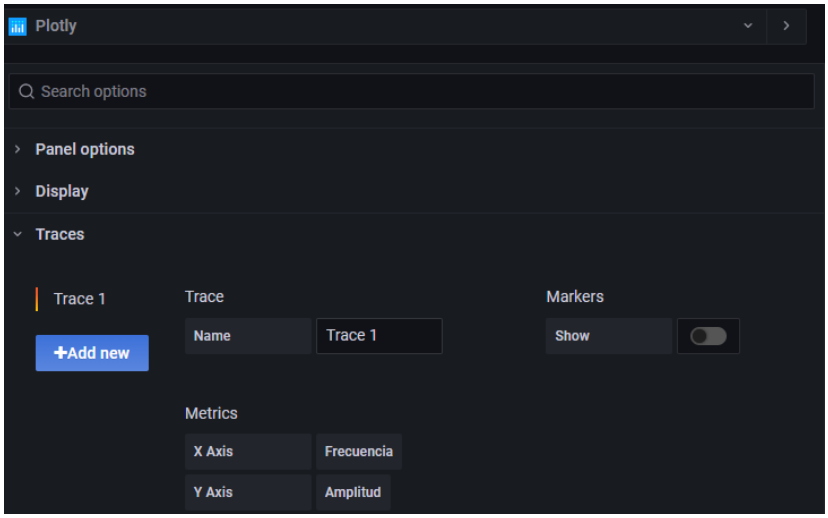

Ilustración 140: Selección de datos del panel

En este momento el panel se actualizará automáticamente y mostrará los últimos valores recibidos.

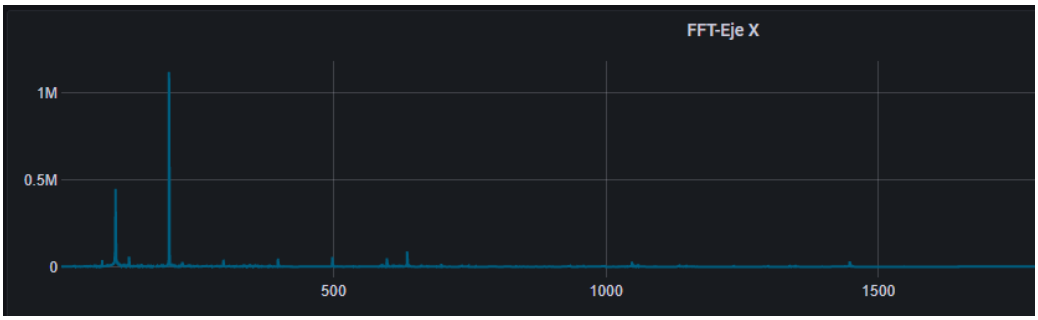

Ilustración 141: Panel ejemplo

Además de los paneles para el último valor recibido, también se han creado 3 paneles que permiten visualizar la FFT de los datos de un periodo de tiempo específico. Este periodo de tiempo se selecciona con el selector de rangos de tiempo de Grafana.

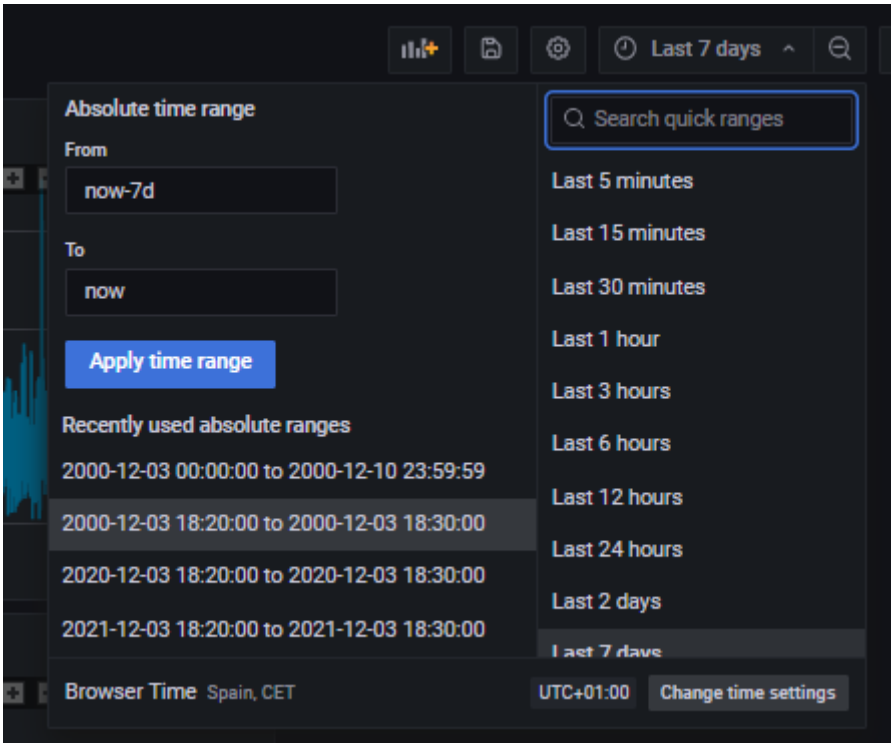

Ilustración 142: Selector de timestamp Grafana

Las querys configuradas en esto paneles son similares a las querys utilizadas en los paneles de último valor, con la diferencia de que también se utiliza como condición que el timestamp de las lecturas almacenadas se encuentre entre los valores del rango establecido.

Grafana proporciona macros que simplifican la sintaxis de las querys y facilitan la aplicación de filtros. Gracias a estas macros, es posible obtener los valores de los timestamps que marcan el comienzo y el final del periodo de tiempo marcado por el usuario. Se encuentran disponibles en "$_unixEpochNanoFrom" y "$_unixEpochNanoTo" respectivamente. Este valor está expresado en nano segundos, sin

embargo, los datos subidos en Thingsboard se han subido en milisegundos, ya que es la mínima precisión con la que trabajan los gráficos de la herramienta. Por lo tanto, es necesario convertir este valor a milisegundos antes de realizar la consulta. En otro caso, la query no retornará ningún valor.

La query generada con para la obtención de la FFT de los valores del acelerómetro para el eje X en un periodo específico se muestra en la Ilustración 143.

Ilustración 143: Ejemplo de configuración de query con timestamp

Llegados a este punto, se han conseguido generar diferentes dashboards que permiten el análisis del espectro de frecuencias de las vibraciones. Sin embargo, estos gráficos solamente son accesibles desde la interfaz web de Grafana.

Es de mayor interés que los dashboards de las magnitudes que requieran herramientas adicionales para su visualización se encuentren accesibles en Thingsboard con el fin de integrar todas las funcionalidades en un único interfaz.

Por tanto, es necesario generar un nuevo widget donde se integre la visualización creada por Grafana. La creación de nuevos widgets se realiza en el apartado “Biblioteca de Widgets”.

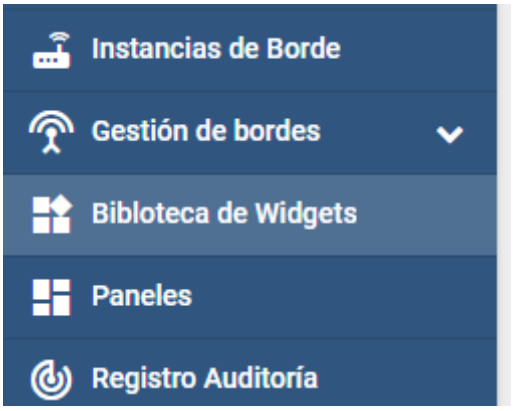

Ilustración 144: Menú de Thingsboard - Biblioteca de widgets

Desde este lugar es posible generar nuevos widgets, importarlos o modificar los ya existentes. En este caso, se crea un nuevo widget de tipo Widget estático llamado “Grafana”. Haciendo uso del editor de widgets proporcionado por Thingsboard se añade el código HTML necesario para incluir en el widget una vista de la interfaz web de Grafana que permite visualizar los diferentes paneles e interactuar con ellos. En la Ilustración 145 se muestra la implementación descrita.

Ilustración 145: Configuración widget personalizado

Si se crea un nuevo panel en Thingsboard que contenta el widget creado anteriormente, se concederá acceso a los paneles de Grafana, ya sea desde el usuario administrador, o desde otros usuarios con rol de cliente que tengan acceso a ese panel.

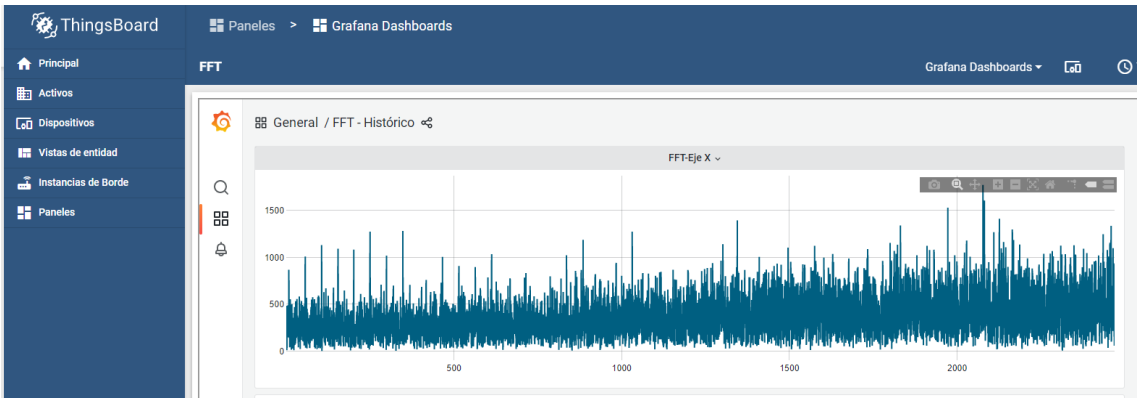

Ilustración 146: Dashboard widget personalizado

11.3.3 Gestión de alarmas

Una de las funcionalidades que aporta Thingsboard es la gestión de alarmas. En versiones anteriores de Thingsboard, para realizar la gestión de alarmas era necesario un alto conocimiento del uso de la cadena de reglas para crear alarmas. En la versión de Thingsboard instalada, se incorpora una nueva funcionalidad, las reglas de alarmas, que permiten gestionar fácilmente las alarmas.

Para poder gestionar las alarmas con la funcionalidad de regla de alarmas es necesario configurar correctamente los perfiles de cada uno de los dispositivos, ya que cada perfil tendrá su conjunto de alarmas propias.

Hasta el momento no se ha comentado nada acerca de los perfiles de dispositivos. Cuando se crea un dispositivo, es necesario asignarle un perfil. La creación de perfiles es dependiente del criterio del propietario. En este caso, se ha optado por crear dos perfiles diferentes, uno para los sensores de temperaturas y otro para los acelerómetros, entendiendo por acelerómetro el resultado de aplicar el Envelope Analysis a la señal captada por el acelerómetro.

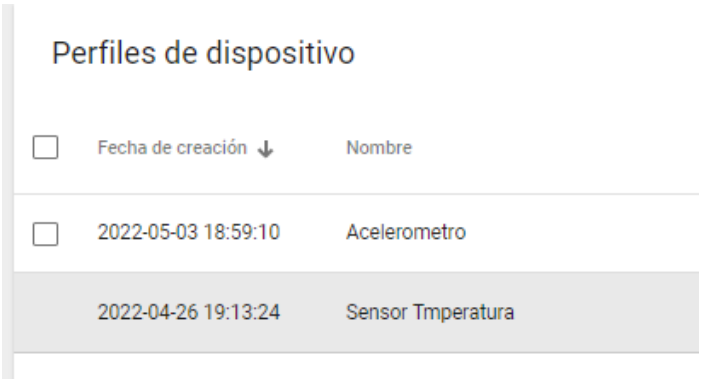

Ilustración 147: Perfiles de dispositivo creados

La creación de perfiles de dispositivo puede realizarse desde el menú de "Perfiles de dispositivo"

Ilustración 148: Menú perfiles de dispositivo

Dentro de cada uno de los perfiles, se pueden crear Reglas de alarma que disparen las diferentes alarmas deseadas. En este proyecto se han generado alarmas para ambos perfiles.

11.3.3.1 Alarmas sensor temperatura

Durante el funcionamiento normal del motor, el valor de la temperatura no sufrirá grandes cambios. Sin embargo, cuando la temperatura se eleve por encima de valores no esperados, es de interés generar una alarma que avisé de esta situación. Pese a que el operario dispone de un elemento informativo donde se informa de la temperatura actual, este puede no ser consciente de que se ha superado el umbral establecido.

Para configurar esta alarma se han de seguir los pasos explicados a continuación.

Al acceder al perfil del dispositivo Sensor de temperatura, se debe seleccionar la opción "Reglas de alarma" del menú superior y pulsar el botón "Añadir regla de alarma".

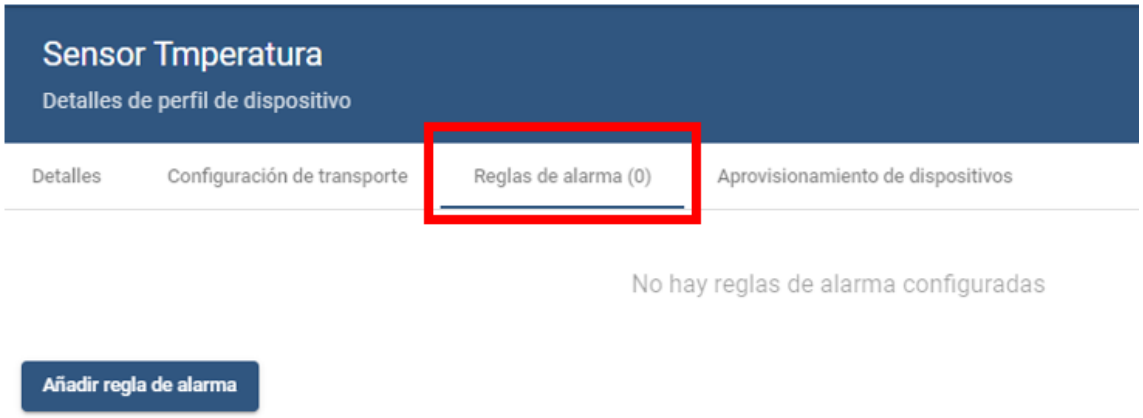

Ilustración 149:Menú reglas de alarma

A continuación, aparecerá un nuevo panel donde es posible configurar las reglas de activación de la alarma. Es posible configurar el tipo de la alarma, la gravedad, el horario en el que se activa, la condición de activación, añadir más detalles y elegir el panel donde se muestra en dispositivos móviles.

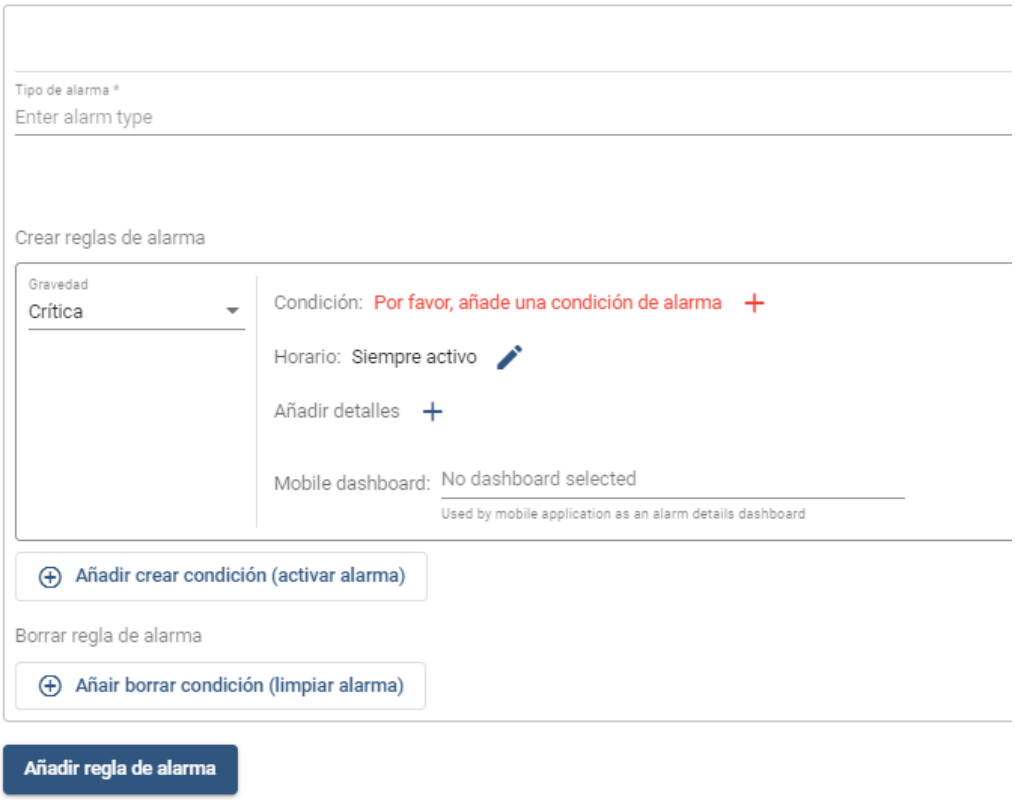

Ilustración 150: Configuración reglas de alarma

Cuando el usuario pulsa sobre el botón de configuración de la condición de activación de la alarma, se muestra un nuevo panel, donde se debe configurar la clave usada como threshold y el umbral a partir de cuál debe lanzarse la alarma. La configuración realizada para este ejemplo se muestra en la Ilustración 151.

Ilustración 151: Configuración condición de alarma temperatura elevada

También es posible añadir una regla de desactivación, de modo que la alarma se desactive si el valor de la clave vuelve a tener un valor por debajo del umbral establecido.

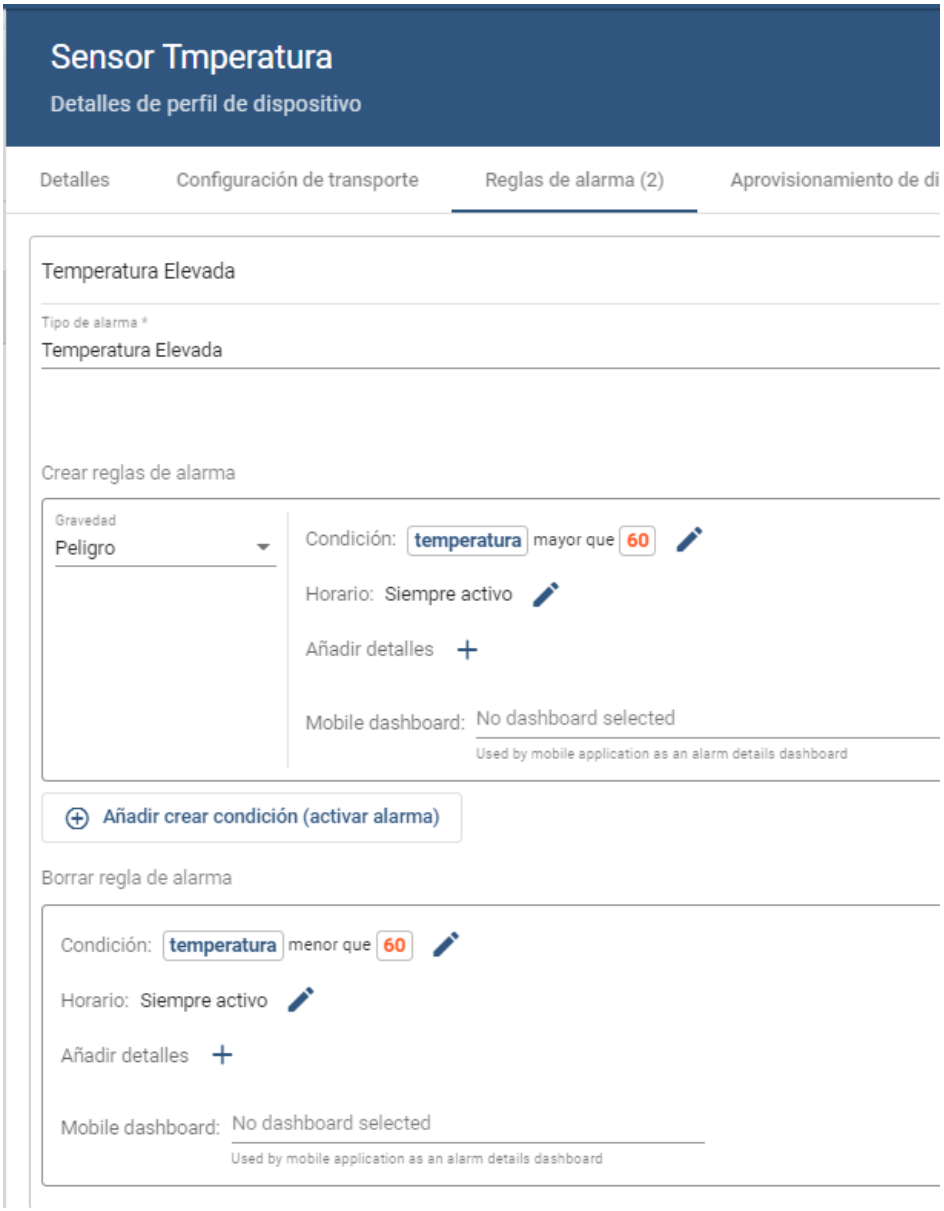

Ilustración 152: Resumen alarma sensor temperatura

Una vez configuradas las reglas de alarma, se han simulado lecturas del sensor de temperatura que dispararan la alarma de temperatura elevada de modo que se pueda verificar su correcta configuración.

La simulación es tan simple como publicar desde el gateway un mensaje MQTT donde se especifique un valor de temperatura por encima de 60.

En la Ilustración 153 se muestra el mensaje publicado para activar la regla de alarma. Como la configuración es correcta, es posible ver la alarma en todos los dispositivos de tipo sensor de temperatura, en este caso solo existe uno.

```
$ mosquitto_pub -d -h "156.35.163.171" -t "v1/devices/me/telemetry" -u "a" -m "{"temperatura": 99}"
```

Ilustración 153: Mensaje MQTT para activar alarma temperatura elevada

Desde el menú de dispositivos, accediendo al dispositivo y en el apartado de alarmas, se observa que existe una nueva alarma. Desde este lugar se tiene acceso a todas las alarmas generadas durante la vida del dispositivo.

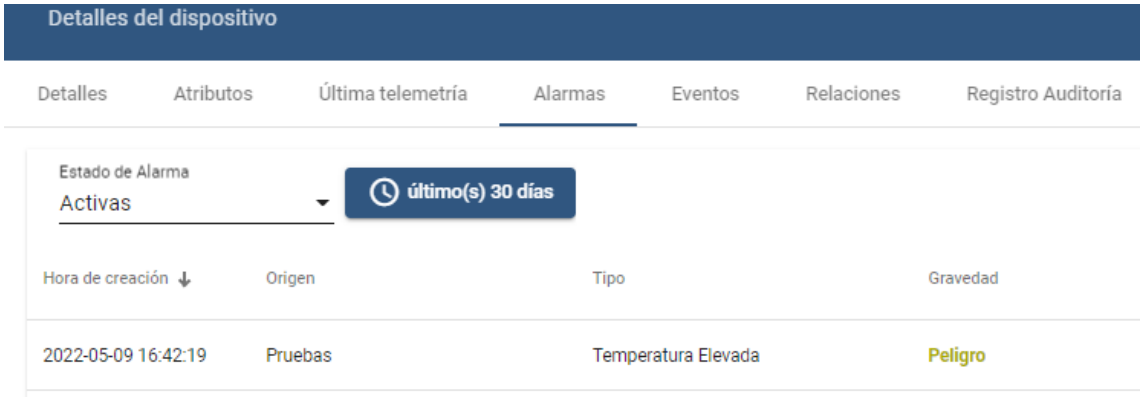

Ilustración 154: Alarma Temperatura Elevada activa

Adicionalmente, se ha simulado el envío de una temperatura inferior a la de activación de la alarma para observar el comportamiento de la plataforma frente a este escenario.

```
mosquitto_pub -d -h "156.35.163.171" -t "v1/devices/me/telemetry" -u "a" -m "{"temperatura": 10}"
```

Ilustración 155: Mensaje MQTT para desactivar alarma temperatura elevada

Una vez publicado el mensaje donde se ha modificado el valor del campo temperatura para que tenga el valor 10, se observa que la alarma ha cambiado su estado de activa a ignorada.

Ilustración 156: Alarma Temperatura Elevada ignorada

Desde el apartado Detalles del dispositivo es posible visualizar las alarmas, pero no interactuar con ellas para modificar su estado. Para realizar esta tarea, es necesario crear un panel de visualización de alarmas que permite visualizar todas las alarmas, tanto activas como en otros estados e interactuar con ellas para modificar su estado.

Al igual que se ha explicado anteriormente para otro tipo de paneles, para los paneles de gestión de alarmas es necesario definir Entidades donde se especifica de que dispositivo se deben visualizar los datos.

Para el caso explicado en este apartado, se ha configurado una entidad para el dispositivo llamada Prueba, al igual que el dispositivo.

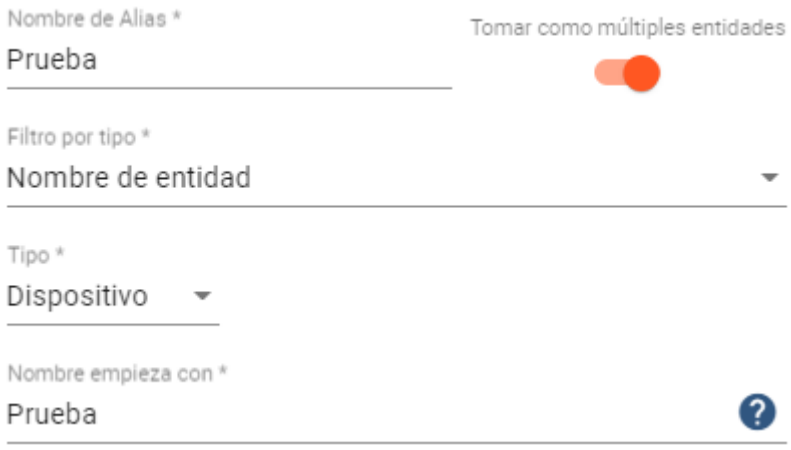

Ilustración 157: Entidad Prueba

Por otro lado, en la configuración del panel se debe indicar que el origen de los datos es la entidad creada anteriormente. Por defecto, se muestran todos los atributos de cada una de las alarmas, aunque es posible eliminar aquellos que se consideren imprescindibles en la representación.

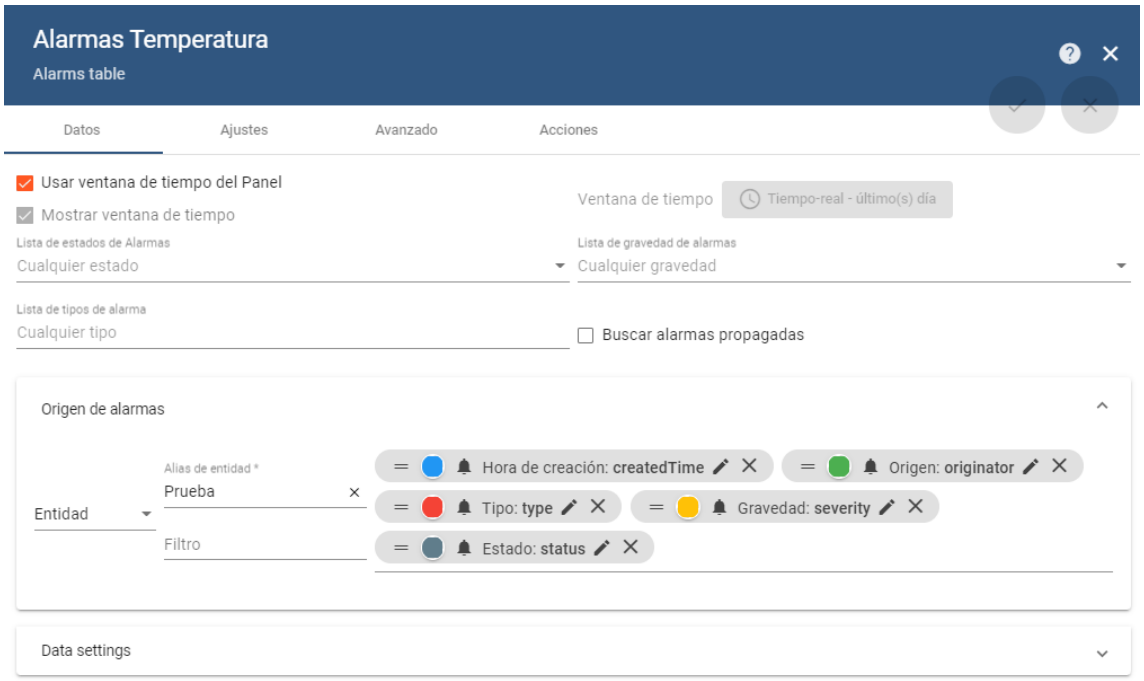

Ilustración 158: Atributos alarma temperatura elevada

Una vez creado, el panel muestra todas las alarmas disponibles para la entidad definida. Además de mostrar sus atributos, es posible reconocer y/o normalizar cada una de las alarmas como se muestra en la Ilustración 159.

Ilustración 159: Panel de alarmas de temperatura

Las alarmas pueden normalizarse y/o reconocerse. En función de estos dos valores la alarma adquiere un estado. Los posibles estados de alarma se recogen en la Tabla 10: Estados de alarma Thingsboard

	Normalizada	***Reconocida***
Activa		
Ignorada	**X**	
Reconocida		**X**
Borrada	**X**	**X**

Tabla 10: Estados de alarma Thingsboard

11.3.3.2 Alarmas acelerómetro

En el apartado anterior se ha mostrado la configuración de las alarmas de temperatura. Es sencillo realizar esta configuración, ya que solo es necesario comprobar que el dato recibido esté por debajo de un umbral.

La configuración de alarmas se complica cuando es necesario configurar alarmas que informen de fallos en los rodamientos. Esta dificultad viene dada en gran medida debido a que Thingsboard está orientado a datos de tipo serie temporal.

La posibilidad ofrecida por la plataforma es crear una regla similar a la de temperatura elevada donde se realiza la comprobación de que el valor de una frecuencia superase un valor.

Las frecuencias en las que pueden aparecer armónicos indicativos de fallos en los rodamientos son conocidas, por lo que esta opción puede ser viable. Sin embargo, las frecuencias de fallo pueden sufrir pequeñas variaciones. Por tanto, es necesario analizar, no solo la frecuencia exacta en la que puede aparecer el defecto, si no una banda de frecuencias.

Esto implica tener que crear tantas reglas de alarma como frecuencias quieren inspeccionar. Además de ser una tarea muy tediosa, no es escalable. Si por ejemplo se sustituyen los rodamientos por otros con características geométricas diferentes, sería necesario reconfigurar todas las reglas ya que se ha producido un cambio en la banda de frecuencia que se debe observar.

La alternativa planteada ha sido delegar en el gateway la tarea de detección de la anomalía en el rodamiento para generar una señal de aviso que permita al cloud hacer saltar la alarma correspondiente.

Actualmente, el gateway envía hacía la nube el espectro de frecuencias obtenido tras aplicar el Envelope Analysis. Además de realizar este envío, se ha configurado el gateway para que analice las bandas de frecuencia donde pueden aparecer las frecuencias de fallo del rodamiento y en caso de detectar un defecto, se lo comunique a Thingsboard.

El dispositivo creado en Thingsboard tiene por tanto cada una de las frecuencias que componen el espectro y 2 atributos adicionales. Por un lado, el atributo fault, de tipo booleano, que actúa como flag de alarma. Si el gateway detecta un fallo en un rodamiento, cambia el valor del atributo fault a verdadero. Por otro lado, el atributo faultType, de tipo cadena de texto, donde se detalla el tipo de fallo.

En la Ilustración 160 se muestra la configuración realizada para la activación de la alarma de fallo en rodamiento. Esta regla comprueba que el flag está activo y que el tipo de fallo corresponda a unos de los posibles fallos en los rodamientos. Si en un futuro se añaden funcionalidades adicionales que permitan la detección de fallos en otros componentes será posible realizar la gestión de alarmas sin interferir en las alarmas de fallo en rodamientos.

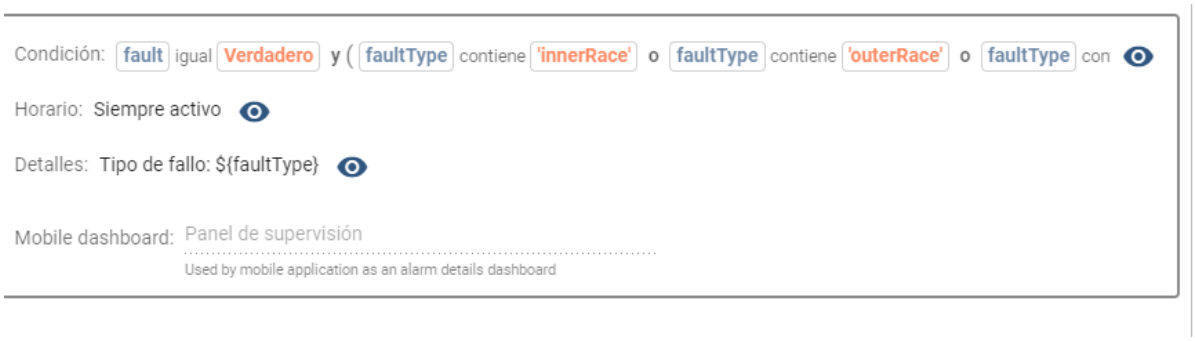

Ilustración 160: Configuración condición de alarma fallo en rodamiento

También se ha configurado una regla de eliminación de alarma para desactivarla cuando el flag retome el valor falso. Esta configuración se muestra en la Ilustración 161.

Ilustración 161: Borrar regla de alarma de fallo de rodamiento

11.3.4 Detección del estado del motor

En la capa del Fog se ha implementado una funcionalidad que detecta si las vibraciones recogidas por el sensor permiten un diagnóstico de rodamiento y que informan de si el motor se encuentra parado o en movimiento.

Para reflejar esta información en Thingsboard, ha sido necesario dar de alta un nuevo dispositivo "Motor" tendrá un único atributo que refleja el estado del motor.

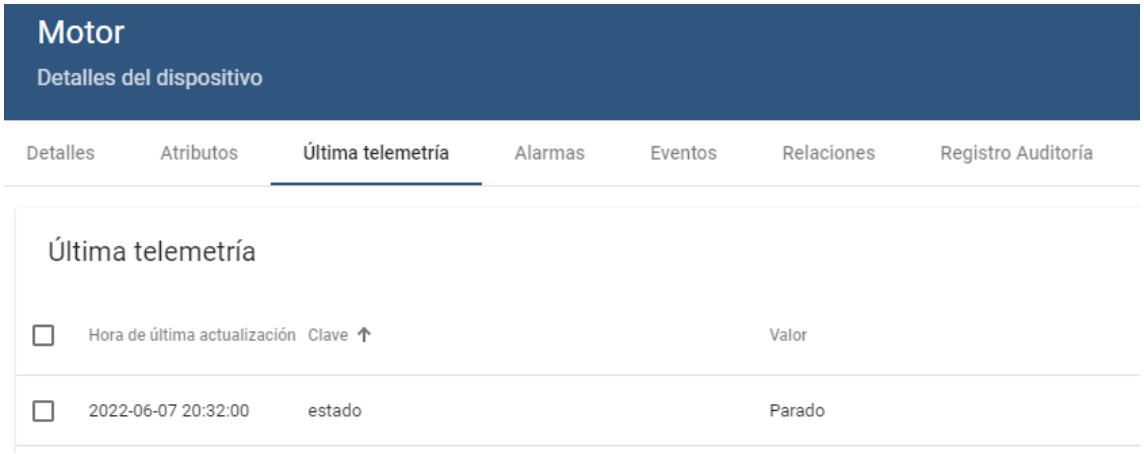

Ilustración 162: Dispositivo motor

Para visualizar la información se ha creado un widget de tipo "Label widget" que muestra el texto "Estado del motor según vibración: <Parado| Movimiento>".

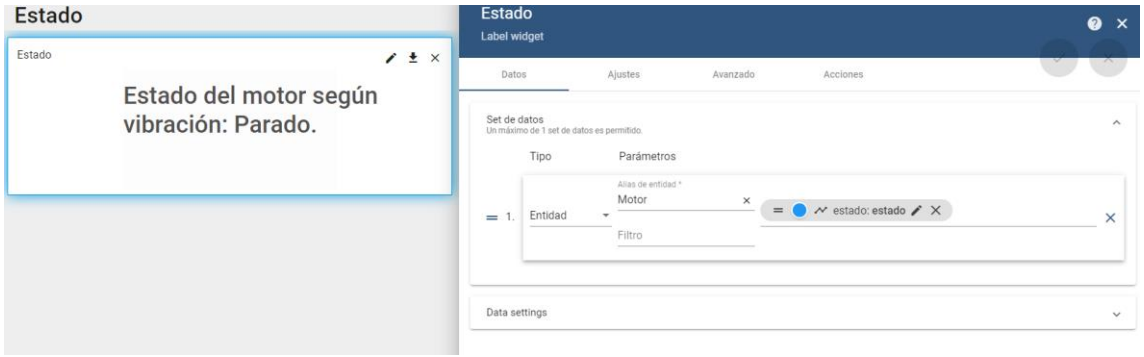

Ilustración 163: Label widget

11.3.5 Panel general de visualización

Para facilitar la visualización de las métricas monitorizadas y los paneles de gestión de alarmas, se ha creado un panel donde se muestran los diferentes widgets descritos anteriormente:

- **Panel Izquierdo**
  - Panel de alarmas de temperatura
  - Gráfica de temperatura
- **Panel Derecho**
  - Panel de alarmas de fallo en rodamiento
  - Estado del motor según vibraciones
  - Envelope Spectrum

En la Ilustración 164 se muestra la disposición los elementos enumerados.

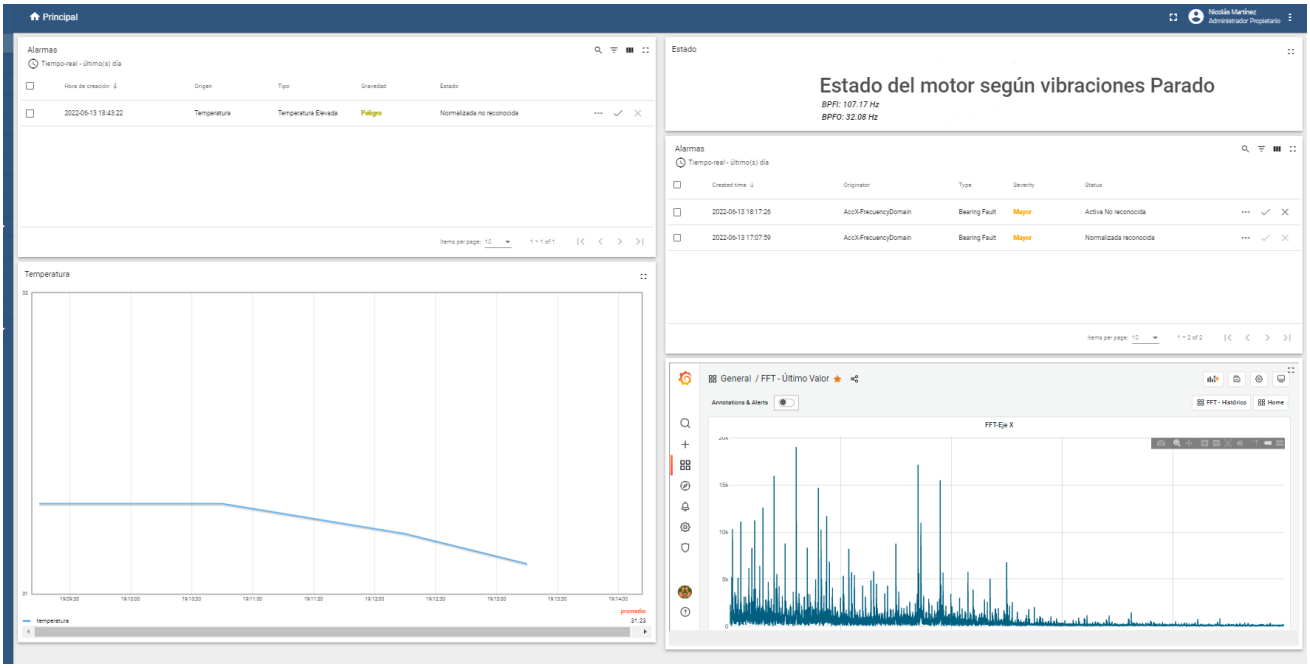

Ilustración 164: Panel general Thingsboard

12 Discusión General

12.1 Aportaciones

Las aportaciones que han supuesto la realización de este proyecto se enumeran a continuación:

1. Estudio de la literatura en materia de detección de fallos en motores mediante la técnica del análisis de vibraciones y simulación de defectos mediante herramientas software.

2. Elaboración de documentación donde se detalla en profundidad las tareas a realizar para configurar las diferentes herramientas y elementos hardware para su uso en proyectos IoT.

3. Desarrollo y despliegue de un sistema de detección de anomalías en rodamientos mediante análisis de vibraciones.

4. Evaluación de los componentes de la arquitectura para proyectos IIoT de características similares.

12.2 Conclusiones

El objetivo primordial que Intermark había marcado era el desarrollo de un sistema capaz de detectar, mediante análisis de vibraciones, defectos en rodamientos. Adicionalmente, se requirió que el sistema desarrollado tuviera una arquitectura específica y estableció que dispositivos hardware/herramientas software deberían emplearse para componer el sistema, realizando 2 versiones diferentes en la que se deben emplear diferentes componentes en cada capa para poder evaluar qué componentes son más adecuados.

Debido a limitaciones temporales, solo ha sido posible realizar una de las dos versiones. Sin embargo, con este despliegue ha sido posible extraer las ventajas e inconvenientes de cada componente utilizado.

Respecto a al dispositivo Steval-MKSBOX1V1, utilizado como Edge device, se concluye que es un dispositivo adecuado para el desarrollo de proyectos IIoT debido a su capacidad para medir simultáneamente múltiples magnitudes (temperatura, humedad, presión, aceleración …). También incluye múltiples ejemplos para desarrollar firmwares ajustados a las necesidades de cada proyecto.

Como desventajas cabe destacar que la frecuencia de muestreo máxima es insuficiente para proyectos que requieran captura de datos a altas frecuencias, como el análisis de vibraciones, donde las frecuencias de muestreo suelen ser elevadas (entre 10 y 100 kHz). Además, la documentación disponible para el uso de las herramientas del entorno de desarrollo de STMicroelectronics no es muy detallada.

Respecto a la Raspberry Pi utilizada como gateway, se concluye que es un dispositivo adecuado para desempeñar esta funcionalidad, ya que permite la comunicación con el resto de las capas y es capaz de realizar las operaciones de procesamiento sobre la señal.

La desventaja encontrada respecto al uso de este elemento como gateway es la limitación establecida para la comunicación mediante BLE con el Edge device.

Respecto a Thingsboard como cloud, se ha concluido que es una herramienta útil para la gestión de activos y datos en aplicaciones IIoT, ya que proporciona las necesidades básicas para almacenamiento de información, visualización y gestión de dispositivos. Su despliegue es rápido, sencillo y la documentación existente es completa. Además, no requiere equipos con altas capacidades para su despliegue.

Sus principales desventajas son que, está totalmente orientado a datos de tipo serie temporal y que existe una limitación respecto a la computación en la nube. No se integran gráficos para realizar representaciones de datos en el dominio de la frecuencia y la capacidad para ejecutar funciones sobre los datos almacenados es muy limitada. Es necesario recurrir a herramientas externas que accedan a la base de datos o usar alguno de los nodos existentes en su motor de reglas para almacenar los datos en otro proveedor de servicios web. En proyectos en los que sea necesaria una mayor computación en la nube, no se recomienda el uso de Thingsboard.

El despliegue y configuración de los elementos/ herramientas de la arquitectura ha sido detallado en este documento, como se demandaba desde Intermark. Para verificar que todos los procedimientos se encontraban lo suficientemente detallados, desde Intermark se ha reproducido el despliegue verificando que en proyectos futuros pueda realizarse siguiendo los pasos descritos en este proyecto. Por tanto, se puede concluir que este objetivo se ha cumplido.

Otro de los objetivos marcados era el de realizar implementar algún método de simulación de anomalías mediante software, de modo que pudieran generarse un conjunto de datos sin necesidad de forzar físicamente los defectos en los rodamientos.

Este punto, al igual que las técnicas de detección de fallos ha sido muy estudiado ya por diferentes autores. La técnica seleccionada para la simulación permite llevar a cabo la labor demandada. Además, incorpora una implementación del modelo, lo cual ha acelerado el cumplimiento de este objetivo.

Respecto al uso de técnicas de machine learning para la detección de anomalías, no ha sido posible implementar un modelo capaz de detectar cuando los rodamientos presentan algún tipo de anomalías y/o en que componente de este. Sin embargo, se ha observado que el uso de estas técnicas es aplicable en este campo, ya que algunos autores ya las han utilizado previamente.

12.3 Trabajos futuros

El sistema desarrollado puede ser ampliado desde múltiples puntos de vista. El primero de ellos, añadiendo técnicas de machine learning que mejoraren la detección de los diferentes tipos de anomalías en los rodamientos u otros componentes. El sistema de análisis de las frecuencias de fallo es demasiado básico y puede ser mejorado, por ejemplo, utilizando un mayor número de indicadores para aumentar su fiabilidad. Sin embargo, sería de interés recabar datos del motor en diferentes situaciones de fallo para, aplicando técnicas de machine learning como han hecho algunos autores de la literatura, optimizar el proceso de diagnóstico.

Otra posible ampliación está relacionada con el uso de otros componentes o herramientas en las diferentes capas de la arquitectura, de modo que pudiera establecerse una comparativa entre las diferentes opciones disponibles en el mercado. De este modo, sería posible extraer conclusiones aún más definitorias de cuando es más indicado utilizar cada uno de los componentes.

Respecto a las comunicaciones entre gateway y las diferentes capas del sistema, sería de gran utilidad desarrollar un interfaz gráfico que permitiera gestionar con mayor facilidad las comunicaciones. Actualmente, solo es posible añadir nuevos dispositivos realizando nuevas versiones de los scripts de código desarrollados, siendo necesario cierto conocimiento de programación.

13 Bibliografía

[1] S. Lacey, «The Role of Vibration Monitoring in Predictive Maintenance,» *Asset Management & Maintenance Journal*, pp. 42-51, 2011.

[2] O. Merkt, «n the Use of Predictive Models for Improving the Quality of Industrial Maintenance: an Analytical Literature Review of Maintenance Strategies,» *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 693-704, 2019.

[3] P. J. C. J. L. B. D. J. A. K. &. M. V. Vlok, «Optimal component replacement decisions using vibration,» *Journal of the operational research society*, pp. 193-202, 2002.

[4] B. Al-Najjar, «maintenance model for identification, quantification and elimination of losses in companies,» 2009.

[5] B. S. a. V. Thomas, «iagnoses of internal faults of three phase squirrel cage induction motor — A review,» *International Conference on Advances in Energy Conversion Technologies (ICAECT)*, pp. 48-54, 2014.

[6] S. P. a. M. P. P. S. Bhowmik, «Fault Diagnostic and Monitoring Methods of Induction Motor: A Review,» *Int. J. Appl. Control. Electr. Electron. Eng*, pp. 1-18, 2013.

[7] S. P. a. M. P. P. S. Bhowmik, «Fault Diagnostic and Monitoring Methods of Induction Motor: A Review,» *Int. J. Appl. Control. Electr. Electron*, pp. 1-18, 2013.

[8] A. K. N. Ahmed Hosameldin, Condition Monitoring with Vibration Signals, London, 2020.

[9] C. D. a. D. D. J. Harmouche, «Improved Fault Diagnosis of Ball Bearings Based on the Global Spectrum of Vibration Signals,» *IEEE TRANSACTIONS ON ENERGY CONVERSION*, pp. 376-383, 2015.

[10] C. Q. H. C. H. H. Sarabjeet Singh, «An extensive review of vibration modelling of rolling element bearings with localised and extended defects,» *Journal of Sound and Vibration*, pp. 300-330, 2015.

[11] S. a. A. D. a. C. J.-H. Kim, «Diagnostics 101: A Tutorial for Fault Diagnostics of Rolling Element Bearing Using Envelope Analysis in MATLAB,» *Applied Sciences*, pp. 2076-3417, 2020.

[12] S. &. S. P. K. Patidar, «An overview on vibration analysis techniques for the diagnosis of rolling element bearing faults,» *International Journal of Engineering Trends and Technology (IJETT)*, pp. 1804-1809, 2013.

[13] S. Goldman, Vibration spectrum analysis: a practical approach, Industrial Press Inc, 1999.

[14] D. C. &. M. J. Baillie, « A comparison of autoregressive modeling techniques for fault diagnosis of rolling element bearings.,» *Mechanical Systems and Signal Processing*, pp. 1-17, 1996.

[15] B. Z. A. Y. L. XIAO ZHANG, «Machine Learning Based Bearing Fault Diagnosis,» *IEEE Access*, 2021.

[16] R. B. Randall, Vibration based condition monitoring : industrial, aeroespace and automative applications, Australia: Wiley, 2011.

[17] R. R. D. HO, «OPTIMISATION OF BEARING DIAGNOSTIC TECHNIQUES USING SIMULATED AND ACTUAL BEARING FAULT SIGNALS,» *Mechanical Systems and Signal Processing*, pp. 763-788, 2000.

[18] H. &. S. N. Endo, «Gearbox Simulation Models with Gear and Bearing Faults,» 2012.

[19] R. R. Jérôme Antoni, «The spectral kurtosis: application to the vibratory surveillance and diagnostics of rotating machines,» *Mechanical Systems and Signal Processing*, pp. 308-331, 2006.

[20] R. R. J. Antoni, «The spectral kurtosis: a useful tool for characterising nonstationary signals,» *Mechanical Systems and Signal Processing*, 2004.

[21] A. J., «Fast computation of the kurtogram for the detection of transient faults,» *Mechanical Systems and Signal Processing*, pp. 108-204, 2006.

[22] C. D. a. D. D. J. Harmouche, «Improved Fault Diagnosis of Ball Bearings Based on the Global Spectrum of Vibration Signals,» *IEEE Transactions on Energy Conversion*, pp. 376-383, 2015.

[23] J. D. S. P. D. McFadden, «Model for the vibration produced by a single point defect in a rolling element bearing,» *Journal of Sound and Vibration 96*, pp. 69-82, 1984.

[24] CRWU, «Case School of Engineering,» Marzo 2022. [En línea]. Available: https://engineering.case.edu/bearingdatacenter.

[25] X. Z. B. &. L. Y. Zhang, «Machine Learning Based Bearing Fault Diagnosis Using the Case Western Reserve University Data: A Review,» *IEEE Access.*, 2021.

[26] STMicroelectronics, «Steval - MKSBOX1V1,» 2022. [En línea].

[27] «Raspberry Pi 4 model B,» 2021. [En línea]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[28] «Thingsboard,» 2021. [En línea]. Available: https://thingsboard.io/.

[29] «InfluxDB,» 2022. [En línea]. Available: https://docs.influxdata.com/influxdb/v2.0/reference/key-concepts/data-elements/#bucket-schema.

[30] S. D. H. N. T. Q. Pengfei Hu, «Survey on fog computing: architecture, key technologies, applications and open issues,» *Journal of Network and Computer Applications*, pp. 27-42, 2017.

[31] «STM Cube IDE,» Octubre 2021. [En línea]. Available: https://www.st.com/en/development-tools/stm32cubeide.html.

[32] «STM CubreProgrammer,» Octubre 2021. [En línea]. Available: https://www.st.com/en/development-tools/stm32cubeprog.html.

[33] «FP-SNS-ALLMEMS,» Octubre 2021. [En línea]. Available: https://www.st.com/en/embedded-software/fp-sns-allmems2.html.

[34] STM, «FP-SNS-ALLMEMS2 Data brief,» 2022. [En línea]. Available: https://www.st.com/resource/en/data_brief/fp-sns-allmems2.pdf.

[35] «Bluetooth BLE el conocido desconocido,» Noviembre 2021. [En línea]. Available: https://ahorasomos.izertis.com/solidgear/bluetooth-ble-el-conocido-desconocido/.

[36] STMicroelectronics, «ST Protocol,» 2021. [En línea]. Available: https://www.st.com/resource/en/user_manual/dm00550659-getting-started-with-the-bluest-protocol-and-sdk-stmicroelectronics.pdf. [Último acceso: 2022].

[37] «Manual instalación Thingsboard,» Octubre 2021. [En línea]. Available: https://thingsboard.io/docs/user-guide/install/ubuntu/.

[38] «Mosquito MQTT,» Octubre 2021. [En línea]. Available: https://mosquitto.org/.

[39] Thingsboard, «Communication Protocol Documentation,» 2021. [En línea]. Available: https://thingsboard.io/docs/reference/mqtt-api/. [Último acceso: 2022].

[40] STMicroelectronics, «FP-SNS-STBOX1 Firmware,» [En línea]. Available: https://www.st.com/en/embedded-software/fp-sns-stbox1.html.

[41] InfluxDB, «InfluxDB API,» 2022. [En línea]. Available: https://docs.influxdata.com/influxdb/v1.8/tools/api/#write-http-endpoint.

[42] InfluxDB, «Manual instalación InfluxDB,» 2022. [En línea]. Available: https://docs.influxdata.com/influxdb/v1.8/introduction/install/.

[43] T. H. T. K. a. K. K. S. Sakamoto, «Vibration analysis considering higher harmonics of electromagnetic forces for rotating electric machines,» *IEEE Transactions on Magnetics*, pp. 1662-1665, 1999.

[44] M. Feldman, «Hilbert transform in vibration analysis,» *Mechanical Systems and Signal Processing*, pp. 735-802, 2011.

[45] «Vibration Analysis of Rotating Machinery,» 2022. [En línea]. Available: https://es.mathworks.com/help/signal/ug/vibration-analysis-of-rotating-machinery.html.

[46] U. G. K. C. Q. H. D. P. Sarabjeet Singh, «Analyses of contact forces and vibration response for a defective rolling element bearing using an explicit dynamics finite element model,» *Journal of Sound and Vibration*, pp. 5356-5377, 2014.

[47] M. C. E. M. Gianluca D'Elia, «An algorithm for the simulation of faulted bearings in non-stationary conditions,» *Meccanica*, 2017.

[48] «Vibration Institute,» 2022. [En línea]. Available: https://www.mfpt.org/about-the-vibration-institute/.

[49] K. A. U. 7. School of Aerospace & Mechanical Engineering, «Fast Kurtogram Implementation,» [En línea]. Available: https://www.kau-sdol.com/matlab-code.

[50] «Grafana,» Enero 2022. [En línea]. Available: https://grafana.com/oss/grafana.

[51] «Tutorial Instalación Grafana,» 2022. [En línea].

[52] «Plotly,» 2022. [En línea]. Available: https://plotly.com/javascript/.

[53] «BlueST Protocol,» Noviembre 2021. [En línea]. Available: https://www.st.com/resource/en/user_manual/dm00550659-getting-started-with-the-bluest-protocol-and-sdk-stmicroelectronics.pdf .