Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE

ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

MONITORIZACIÓN EN TIEMPO REAL DE MOTORES ELÉCTRICOS DE VELOCIDAD VARIABLE ORIENTADA A LA DETECCIÓN DE ANOMALÍAS DE FUNCIONAMIENTO

D. ALVES PEREIRA, Jesús Gabriel

TUTORES:D. Francisco José Suarez Alonso

D. Luis Magadán Cobo

FECHA: Julio, 2021

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

El Internet de las Cosas o IoT no posee una definición unánime, es considerada como la mayor evolución del Internet tal como se conoce ya que todavía se define como una tecnología nueva y emergente.

Este hecho se debe a los más de 29.3 miles de millones de dispositivos que existirán para el año 2023 conectados a redes IP (ver Ilustración 1), lo que representa 18.4 miles de millones de dispositivos más conectados a la red que en el año 2018 [1]. Dentro de este grupo destacan los 31% de dispositivos machine to machine (M2M), término que se refiere a la tecnología que permite el intercambio de información entre maquinas sin la necesidad de intervención por parte de personas.

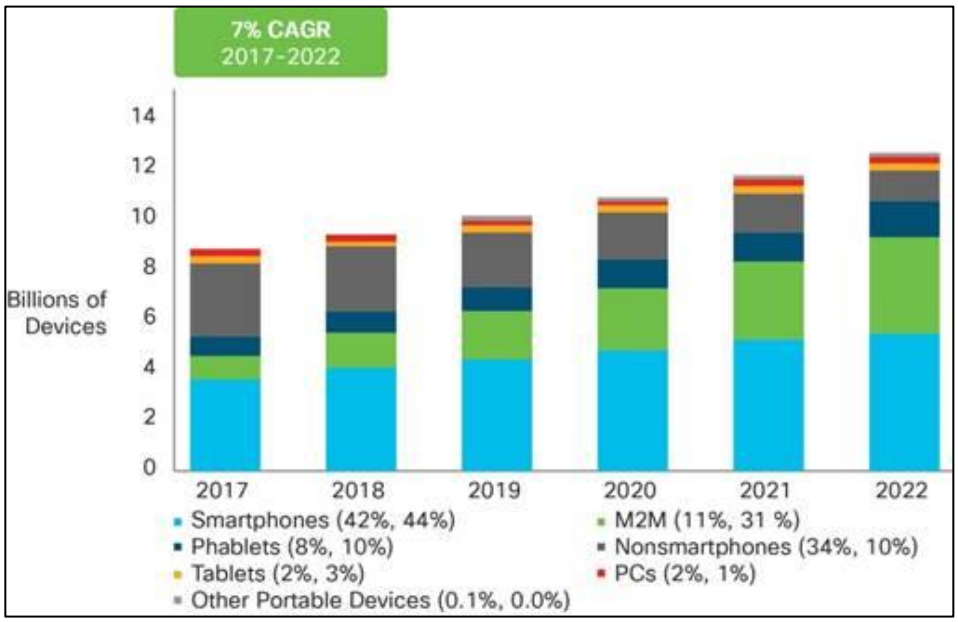

Ilustración 1: Gráfico de dispositivos conectados [1]

Esto hace pensar que el Internet de las Cosas tenga la posibilidad de modificar el mundo como lo conocemos, haciendo énfasis principalmente en conveniencia, accesibilidad y rendimiento. Estás dos primeras características son fácilmente reconocibles ya que es posible integrar los objetos de uso cotidiano con otros sistemas ya existentes, por lo que existen infinidad de servicios que se pueden ofrecer para facilitar la vida de los usuarios de dichos objetos. Por otro lado, el rendimiento está más sujeto a la capacidad extraordinaria de recolección de datos, haciendo del IoT "un enorme salto en su capacidad para reunir, analizar y distribuir datos que podemos convertir en información, conocimiento y en última instancia, sabiduría" [2]

En relación con este último enfoque nos podemos encontrar conceptos bastante interesantes en los que podemos ver su aprovechamiento, como puede ser el caso del mantenimiento predictivo.

El concepto de mantenimiento es considerado como el "Conjunto de operaciones y cuidados necesarios para que instalaciones, edificios, industrias, etc., puedan seguir funcionando adecuadamente" [3]. Existen diversos tipos de mantenimiento que se tienen en cuenta hoy en día, especialmente por la irrupción de nuevas tecnologías. Entre estos podemos destacar tres tipos:

- Mantenimiento reactivo: Tipo de mantenimiento que suele responder a una avería en el dispositivo, edificio, sistema, etc. Se realiza en el momento de un fallo en el equipo e involucra la detención de este.

- Mantenimiento preventivo: Es aquel que se realiza según las recomendaciones generales del fabricante según el producto. Se realiza en paradas programadas sin responder a la existencia o ausencia de un fallo en el equipo.

- Mantenimiento predictivo: Se basa en determinar la condición actual del objeto, de forma que se puedan realizar correcciones antes de que se presente un fallo.

Entre estos tres tipos de mantenimiento que se mencionan, es el predictivo el que representa la mayor ventaja. Si bien es el conjunto de estos los que forman usualmente las estrategias de mantenimiento especialmente en la industria, esta última representa un gran beneficio con respecto a las demás, ya que evita el hecho de detener la producción de forma inesperada, como ocurriría en el caso del mantenimiento correctivo.

Por otro lado, también es una evolución del mantenimiento preventivo por su adaptación al caso particular de cada equipo, situación que no se presenta en este último, ya que por parte del fabricante se establecen unas reglas generales a cumplir para el mantenimiento de los equipos, pero evidentemente no responden a la situación particular de cada uno, donde podemos encontrar desgaste o averías particulares según múltiples variables en cuanto a utilización.

Este tipo de mantenimiento, hoy en día base fundamental de las mejoras de rendimiento en máquinas y sistemas tanto de producción como cotidianos, vienen dadas gracias a la integración de estos sistemas anteriormente aislados o dependientes de una persona, permitiendo su constante monitorización sin necesidad de detener la producción suponiendo un mayor beneficio con respecto a otros tipos de mantenimiento.

Este proyecto se desarrolla en colaboración con la Empresa Municipal de Medio Ambiente Urbano de Gijón (EMULSA), encargada de las labores de higiene urbana y mantenimiento de zonas verdes. Dicha empresa es poseedora de una gran flota de vehículos eléctricos utilizados para diferentes fines dentro del consejo de Gijón. Entre los diferentes vehículos eléctricos el prototipo se desarrollará con el objetivo de monitorizar los motores de los vehículos ALKE 340 E 20kW LITIO. Estos poseen dos motores encargados el primero de tipo eléctrico trifásico asíncrono de inducción de la tracción del vehículo, y el segundo de imán permanente para el sistema de bombas de agua. Ambos motores representan un candidato ejemplar para el

mantenimiento predictivo, ya que son los componentes claves para el correcto funcionamiento del vehículo junto con la batería, permitiendo así un mayor beneficio a EMULSA al determinar la condición de estos equipos sin necesidad de realizar revisiones.

Para llevar a cabo el mantenimiento predictivo de estos motores, se dispone de un nodo multi sensor por cada uno de estos, encargado de capturar la temperatura, humedad y vibraciones. Por otro lado, se utiliza un Gateway encargado de recibir los datos capturados, realizar las transformaciones necesarias sobre estos y enviar los datos relevantes a una instancia Cloud, donde se almacenarán finalmente.

2. Objetivos y Alcance

Los objetivos de este proyecto serán los siguientes:

- Estudio del modelo de computación más apropiado para la monitorización del estado de los motores de los vehículos eléctricos de EMULSA.
- Estudio de las diferentes tecnologías existentes para la comunicación entre los dispositivos multi sensores, el Gateway y la instancia Cloud que almacena los datos enviados.
- Estudio del funcionamiento de los dispositivos multi sensores y Gateway.
- Implementación de un prototipo que se encargue de recolectar la información pertinente a la humedad y la temperatura del motor principal y del motor de bombas de aguas, además de capturar las vibraciones de ambos. Está información debe ser tratada de forma que luego pueda ser enviada a la nube mediante Gateways, para la monitorización de los vehículos eléctricos.

3. Estado del arte

El vehículo de EMULSA para el cual se desarrolla el prototipo es un ALKE 340 E 20kW LITIO. Este dispone de dos motores eléctricos, siendo uno de ellos un motor de tracción asíncrono de inducción de 48 V AC de 14kW y el otro, un motor auxiliar de 3kW de potencia de imán permanente. Son estos dos tipos de motores en los que el prototipo será desplegado, por lo que a continuación se procede a explicar los fallos más comunes en ellos y las técnicas que se emplearán para determinar su condición.

3.1. FALLOS MÁS COMUNES EN MOTORES ELÉCTRICOS Y SU MANIFESTACIÓN FÍSICA

Otro dato muy importante para la realización del prototipo y su correcto funcionamiento en cuanto a determinar la condición actual de los motores, son las manifestaciones físicas de las diferentes fallas que se pueden presentar. Entre las más frecuentes se encuentran las siguientes [4]:

- Degradación del aislamiento: Siendo una de las averías más comunes en motores eléctricos, el aislamiento de estos suele encontrase expuesto a vapores corrosivos, cambios de temperatura frecuentes, humedad, entre otros, que generan su continuo envejecimiento. La degradación del aislamiento los estos es una de las causas más comunes de la disminución de la vida útil de los motores eléctricos y sus fallos.

- Instalación inadecuada: Problemas o deficiencias en el montaje del motor puede generar rozaduras y desalineamientos en los componentes de este, generando como consecuencia vibraciones indeseadas y desgastes de las piezas.

- Sobrecarga eléctrica: Generalmente ocurre cuando el flujo de corriente en los devanados o bobinas del motor es excesivo, es decir, que supera la corriente que el motor puede transportar de manera segura. Esto puede ser causado por un bajo voltaje de suministro, lo que se traduce en mayor consumo de corriente por parte del motor en busca de mantener su par, o un cortocircuito. Este fallo suele producir un aumento de temperatura como consecuencia física.

- Fallos en los rodamientos: Este fallo generalmente ocurre cuando existe una mala instalación de estos, una lubricación inadecuada, una mala alineación con los componentes, contaminación del mecanismo, etc. Suele traducirse en el aumento de rozamientos y fricción entre los elementos del motor, produciendo así altas temperaturas y vibraciones indeseadas.

- Contaminación o cantidad indeseada de lubricante: El exceso, inexistencia o contaminación del lubricante necesario en un motor eléctrico produce fricciones o esfuerzos innecesarios en los acoplamientos de los componentes del motor, generando altas temperaturas, vibraciones y perdidas de rendimiento.

3.2. ESTUDIO DE LAS TÉCNICAS PARA DETERMINAR LA CONDICIÓN DE UN MOTOR ELÉCTRICO

Existen diferentes técnicas aplicadas hoy en día para determinar los fallos anteriormente mencionados, algunas de estas son aprovechables para el mantenimiento predictivo, ya que proporcionan la capacidad de monitorización sin la necesidad de intervención. Algunas de las técnicas más frecuentes son [4]:

- Análisis de vibraciones: Es uno de los métodos más importantes en cuanto al mantenimiento de motores eléctricos se refiere ya que, como se puede observar en el apartado anterior, muchos de los diferentes fallos en estos se manifiestan físicamente mediante estas. El principal interés en análisis de vibraciones es la identificación de las amplitudes predominantes de las vibraciones emitidas por las maquinas, de forma que se puedan detectar anomalías cuando estas se generen, mediante la utilización de la Transformada de Fourier o Fast Fourier Transformation (FFT).

- Análisis de lubricantes: Consiste en el análisis en laboratorio del lubricante del motor en busca de contaminantes que puedan existir en el mismo. Dicho proceso busca descartar posibles desgastes en elementos que friccionan entre sí como pueden ser los rodamientos.

- Análisis por ultrasonidos: Este método utiliza el estudio de sonidos de baja frecuencia que no son perceptibles por el oído, en busca de rozamientos o fricciones entre componentes mecánicos o fugas.

- Termografía: Como bien se mencionó anteriormente, muchos de los fallos posibles poseen como manifestación física el aumento de temperatura, por lo que el análisis de la temperatura del sistema es un método clave para el mantenimiento predictivo de las maquinas. En este caso existen diferentes maneras de implementar dicha técnica ya sea a través de imágenes termográficas o capturando la temperatura del motor.

Dentro de todos los métodos anteriormente mencionados, se pueden destacar principalmente el análisis de vibraciones y la termografía como los más importantes para

determinar la condición actual de un motor eléctrico ya que, permiten capturan la mayoría de los síntomas físicos relacionados con los fallos más comunes de estos. Además, poseen la ventaja de ser métodos aplicables con los motores en funcionamiento, lo que provee un gran abanico de posibilidades. Es por ello por lo que estas serán las técnicas para utilizar en el desarrollo del prototipo de este proyecto, sin embargo, cabe mencionar que, para el caso de la termografía será implementada la captura de la temperatura ambiental del motor y no de imágenes térmicas.

3.3. ESTUDIO DE LOS DATOS RELEVANTES ASOCIADOS A LAS VIBRACIONES

La captura de vibraciones en elementos como pueden ser los motores eléctricos proveen mucha información sobre el estado de los mismos como se comentó anteriormente, pero para ello se deben conocer las diferentes variables que resultan importantes para determinar la condición de estos, Entre estas variables se deben diferenciar primeramente aquellas extraídas en el dominio del tiempo, es decir, las señales capturadas directamente del motor, y aquellas en el dominio de la frecuencia mediante la FFT.

3.3.1. VARIABLES SEGÚN EL DOMINIO DEL TIEMPO

Las vibraciones en el dominio del tiempo son aquellas señales capturadas directamente de los motores. Los tres componentes que pueden ser capturados de un motor eléctrico en el dominio del tiempo son:

- Desplazamiento: Distancia de un objeto con respecto a un punto de referencia anteriormente establecido.
- Velocidad: Variación que existe en el desplazamiento, generalmente medida en milímetros/segundo.
- Aceleración: Proporción del cambio de velocidad, generalmente medida en milímetros/segundo^2.

En cada una de las componentes anteriormente mencionadas, existen medidas estadísticas que pueden proveer información sobre el comportamiento y condición del motor. Entre las más importantes se encuentran:

- Desviación típica: Esta medida describe el nivel de vibración total que existe en un motor, por lo que las variaciones de dicho valor en diferentes muestras indican posibles inconvenientes en la condición de este. Sin embargo, existen fallos que no provocan alteraciones significantes en el nivel global de las vibraciones, lo que hace que pasen desapercibidos hasta que generen problemas secundarios o fallos catastróficos para el equipo.
- Kurtosis: Esta medida estadística se utiliza como medida de la forma de onda de una señal, haciendo de la comparación en diferentes muestras, una medida interesante a analizar. Esto se debe a que si bien existen fallos que no provocan alteraciones significativas en el nivel global de las vibraciones, si pueden presentar diferencias significativas en la forma de la onda que genera la señal de la vibración, por lo que puede determinarse que existe un fallo en dichos equipos.

3.3.2. VARIABLES SEGÚN EL DOMINIO DE LA FRECUENCIA

Cada una de las componentes presentadas en el apartado anterior poseen sus ventajas y desventajas a la hora de ser capturadas y utilizadas para establecer la condición de un motor eléctrico, especialmente en relación con la amplitud de vibración que se emplea para expresar cada medida. En el caso del desplazamiento se pueden apreciar sus mayores amplitudes en bajas frecuencias (< 10 Hz). Por otro lado, en la velocidad se observan en frecuencias intermedias (10 Hz > x < 1 KHz) y en la aceleración en frecuencias altas (>1 KHz), como se puede ver en la Ilustración 2 como se presenta el comportamiento de estas unidades de amplitud en todo el rango de la frecuencia.

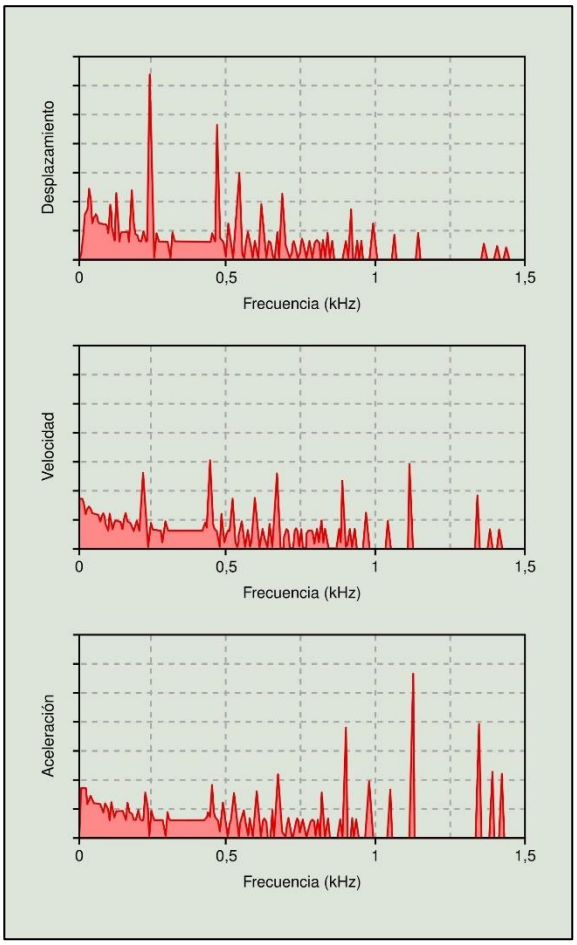

Ilustración 2: Comparativa diferentes unidades amplitud [5]

Estas señales extraídas a partir de los datos capturados en el dominio del tiempo proveen mucha información sobre las condiciones del sistema, sin embargo, cada una de estas representa mucha información de manera compleja por lo que resulta complicado distinguir entre los comportamientos de los componentes que se pueden extraer de esta. Es por ello por lo que esta información suele analizarse en el dominio de la frecuencia. Para ello se grafica la amplitud frente a la frecuencia de la señal capturada. En este caso, se utiliza la Trasformada Rápida de Fourier o Fast Fourier Transformation anteriormente comentada, para calcular la serie de señales

sinusoidales que posee una señal compleja capturada, mostrando posteriormente cada una de estas en una gráfica de espectro como se puede ver en la Ilustración 3.

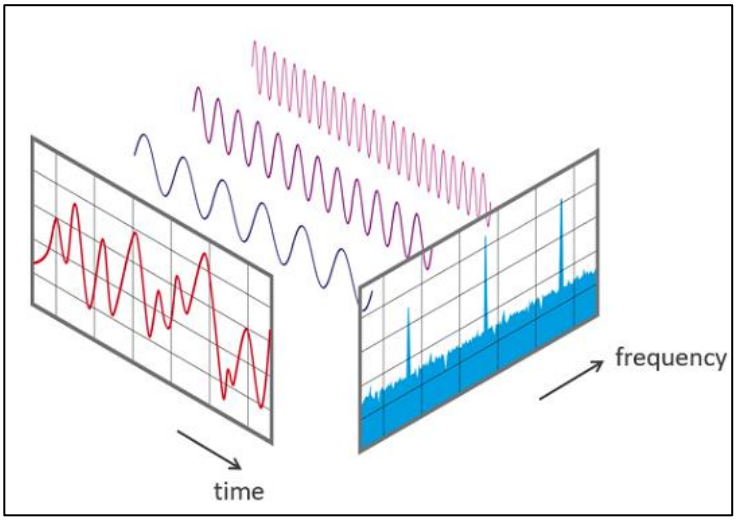

Ilustración 3: Funcionamiento de la Transformada Rápida de Fourier [6]

En el caso de las vibraciones en el dominio de la frecuencia, las variables que proveen información de la condición del motor serían primordialmente la frecuencia y la amplitud. La primera de estas nos permite determinar la razón de la vibración o el dónde, mientras que la amplitud se utiliza para determinar la gravedad de la avería.

En términos de frecuencia existen ciertos valores que poseen un interés especial, entre los cuales se encuentran:

- Pico de primer orden: También llamado primer armónico, que usualmente corresponde con la velocidad de rotación del eje del motor. Generalmente se observarán picos o armónicos múltiplos enteros de este que pueden ser de interés.
- Frecuencias de fallos en cojinetes: Existen ciertas bandas espectrales que denotan fallos en los cojinetes del motor eléctrico, para los cuales se debe aplicar para cada una de estas una formula dependiente de las revoluciones por minuto alcanzadas por el motor en cuestión. En estos podemos destacar la posibilidad de destacar desequilibrios y desalineaciones en holguras, defectos de lubricación, etc.
- Frecuencias de fallos de un rodamiento: En estas se pueden encontrar la frecuencia de deterioro de la pista exterior, la pista interior, los elementos rodantes o incluso de la jaula del rodamiento.
- Frecuencia de la red eléctrica: Permite comprobar problemas en el suministro de la energía al motor eléctrico, aunque no necesariamente es el método más fiable para la detección de estos.
- Doble frecuencia de la red eléctrica: Al alimentarse el motor de tracción del vehículo por corriente alterna, la señal recibida corresponde con una senoide por lo que en cada ciclo se recibe dos veces la energía, haciendo así que la frecuencia más comúnmente encontrada sea el doble de la frecuencia del suministro eléctrico del motor.
- Frecuencia del paso de bobina: Permite comprobar problemas de alineamiento y holgura en el rotor.

Para determinar la condición de cada uno de estos componentes se debe tener en cuenta la amplitud que poseen cada una de las frecuencias relacionadas con estos. Este valor es el responsable de indicar la importancia de una avería en el caso de que se encuentre, sin embargo, siempre es necesario el análisis de todos los picos encontrados al realizar la FFT, ya que la relación existente entre estos es de suma importancia. Cabe destacar que, en el caso del prototipo a desarrollar, el análisis de estos elementos no se encuentra dentro de los objetivos, sin embargo, su captura sí.

3.4. ESTUDIO DE LOS TIPOS DE COMPUTACIÓN EN APLICACIONES IOT

Uno de los factores más importantes del mantenimiento predictivo es la recolección de datos para determinar el comportamiento, en este caso, de los motores eléctricos. Sin embargo, existe otra característica igual de importante que es el procesamiento de datos.

En una aplicación IoT como la del prototipo a desarrollar, existen múltiples lugares donde se puede realizar el procesamiento de datos. Dentro de las diferentes arquitecturas de computación que se pueden asociar a las tecnologías Cloud, se tienen las siguientes:

- Cloud computing: Es aquella en la que el procesamiento de los datos se realiza directamente en la nube.
- Fog computing: Es la arquitectura de computación en la que los datos son tratados en algún nodo intermedio entre el dispositivo encargado de recolectar los datos y la nube.
- Edge computing: Arquitectura de computación en la que los datos son tratados directamente en el dispositivo encargado de capturar los datos.

Las principales ventajas y desventajas entre cada una de estar arquitecturas de computación relacionadas con la nube se pueden observar en la Tabla 1.

	Ventajas	Desventajas

Cloud computing	-Mayor capacidad de procesamiento	-Requiere un ancho de banda importante para el envío de datos desde donde son recolectados -Tiempo de respuesta largo
Fog computing	-Tiempo de respuesta menor que el Cloud computing y mayor que el Edge computing -Requiere menor ancho de banda de conexión con la nube	-Capacidad de procesamiento limitada a las características del equipo donde se produce
Edge computing	-Tiempo de respuesta inexistente	-Capacidad de procesamiento limitada a las características del dispositivo

Tabla 1: Ventajas y desventajas de las arquitecturas de computación Cloud

Si bien en la mayoría de los casos la computación se realiza en la nube, existen muchos casos donde la trasferencia de datos, la cantidad o la finalidad para la cual se capturan dichos datos hace innecesario o redundante la necesidad de depender de la nube para el procesamiento. Es por ello y ante el aumento de la cantidad de dispositivos que capturan datos de forma simultánea, que existen conceptos como el Fog computing o Computación en la niebla [7]. Esta tecnología busca mejorar el rendimiento de los esquemas de Cloud Computing, haciendo a los Gateways elementos mucho más activos, de forma que se puedan procesar datos más cerca de donde son recolectados permitiendo el análisis de datos en tiempo real o lo más cerca de su origen, además de reducir el ancho de banda necesario para el envío de los datos al enviar solo aquellos realmente necesarios. La Computación en la niebla o Fog computing permite

establecer un esquema jerárquico distribuido que se utiliza en el prototipo desarrollado, donde los equipos multi sensores capturan los datos que envían a los Gateway, encargados a su vez de realizar un análisis básico de los datos mediante la aplicación de la FFT a los datos extraídos de las vibraciones, para posteriormente enviar todos los datos más relevantes de dichas transformaciones y operaciones a la nube donde se almacenarán y podrán ser extraídos en un futuro para su análisis correspondiente.

4. Tecnologías empleadas

4.1. GATEWAY

Para este componente del prototipo realizado se utiliza la Raspberry Pi 4 Este es categorizado como un miniordenador u ordenador de placa reducida, ideal para el desarrollo y prototipado de proyectos. Las principales características de dicho modelo son [8]:

- Procesador y GPU: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Ram: 4GB LPDDR4-3200 SDRAM.
- Conexión inalámbrica: 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless LAN.
- Bluetooth 5.0 y BLE.
- Alimentación: 5V DC.
- Micro SD.

Este elemento del proyecto será el encargado de recibir los datos capturados de los equipos multi sensores para posteriormente, ante la presencia de una conexión inalámbrica previamente establecida, tratar los datos almacenados aplicando las transformaciones necesarias y eliminando datos redundantes. Por último, enviara los resultados a la plataforma IoT encargada de su almacenamiento final.

Por último, se dispone de una carcasa Strompi3 de aluminio que permite la correcta disipación de calor mediante enfriamiento pasivo por su material de fabricación y activo mediante un disipador. Por otro lado, también ofrece espacio para la colocación de HAT que se mencionará posteriormente y funcionalidades añadidas como un botón de encendido para el dispositivo. Se puede apreciar el montaje final con carcasa de la Raspberry Pi en la Ilustración 4.

Ilustración 4:Raspberry PI 4 con carcasa y sin carcasa

4.2. MULTI SENSOR

El equipo multi sensor seleccionado es el SmartBond DA14585 IoT Multi Sensor, fabricado por la compañía Dialog Semiconductor y que se muestra en la Ilustración 5. Es un nodo multi sensor diseñado para el desarrollo de múltiples aplicaciones según las necesidades de cada proyecto, proporcionando diversos sensores en un mismo dispositivo, además de un bajo

consumo de energía para su funcionamiento y la transmisión de datos mediante el protocolo Bluethoot Low Energy 5.

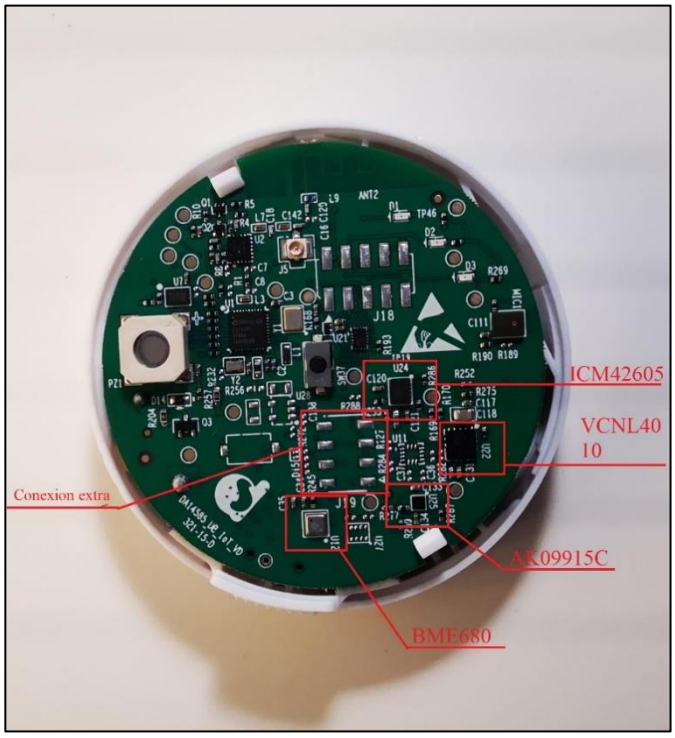

Ilustración 5: Sensores del nodo Multi Sensor

Como se observa en la Ilustración 5 el dispositivo está formado de 5 sensores de los cuales se utilizarán para el desarrollo del prototipo dos de ellos. Entre estos dos sensores se encuentran el sensor BME680 de Bosch y el ICM42605 motion sensor de TDK Invensense . El primero es un sensor medioambiental capaz de medir cambios de temperatura, humedad, presión atmosférica y calidad del aire. Por otro lado, el sensor de TDK combina un giroscopio de 3 ejes con un acelerómetro de 3 ejes conectado al microcontrolador mediante la interfaz SPI con velocidades de hasta 24Mhz.

Ambos sensores serán utilizados como se comentó anteriormente para capturar datos relevantes de los motores. Estos datos serán la temperatura y la humedad para el caso del sensor BME 680, y la aceleración para el caso del ICM42605.

4.3. HAT NB-IOT SIM7000E

Para el envío de los datos se dispone de un módulo HAT NB-IoT SIM7000E. Este proporciona a la Raspberry Pi conexión 2G mediante tarjeta SIM, permitiendo así el envío de datos desde el dispositivo a la nube una vez que estos sean procesados. El montaje del HAT con la Raspberry Pi, puede observarse en la Ilustración 6.

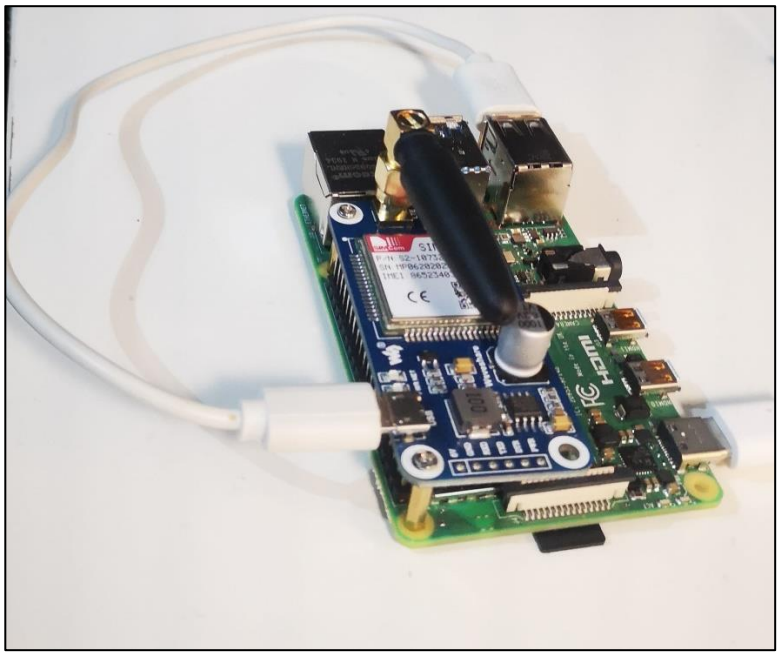

Ilustración 6: Montaje Raspberry Pi 4 + SIM7000E

Entre las características más destables de dicho módulo de comunicaciones están:

- Conectividad con Raspberry Pi, compatible con todos sus modelos.
- Soporte para protocolos TCP, UDP, PPP, HTTP, FTP, MQTT, SMS, Mail, etc.
- Soporte para posicionamiento mediante GNSS.
- Interfaz USB integrada para la ejecución de comandos AT, recolección de datos GPS, entre otros.
- Pins de control UART para conexión con Arduinos y/o STM32.
- Compatible con tarjetas SIM normales y específicas para para NB-IoT.

4.4. HERRAMIENTAS DE DESARROLLO

Para el prototipo desarrollador se utilizaron los entornos recomendados por la empresa fabricante del módulo multi sensor, Dialog-Semiconductor. Este paquete de herramientas se llama SmartSnippets Studio en el que podremos encontrar los siguientes programas o herramientas:

- SmartSnippets IDE: Este es un entorno de programación basado en el IDE de Eclipse con todos los plugins necesarios para realizar el depuración y compilación de la solución.
- SmartSnippets Toolbox: Posee todas las herramientas necesarias para realizar la programación y carga del firmware desarrollado en la memoria Flash del equipo multi sensor, además de permitir realizar un perfil de consumo del dispositivo según el firmware desarrollado.

El IDE, como se comentó anteriormente, es muy similar al entorno de desarrollo de Eclipse IDE en el que se basa. En él se importa el proyecto facilitado por Dialog-Semiconductor a partir del cual se realizarán todos los cambios necesarios para cumplir con las necesidades de nuestro proyecto particular. Una vez realizado este proceso se compilan dos archivos con el nombre del proyecto, uno con extensión .elf y el otro .hex. El primero de ello es utilizado para depurar el proyecto, mientras que el segundo es utilizado para "quemar" directamente en la unidad Flash del multi sensor. Este último se debe mover al directorio …\utilities\mkimage_utils_scripts de la carpeta raíz del proyecto desarrollado. Allí encontraremos la herramienta make_image_iot.bat, que se encarga de convertir el fichero hexadecimal en conjunto con la carpeta config, que podremos encontrar a su vez en la raíz del proyecto, en un fichero .bin que será "quemado" posteriormente en la tarjeta Flash.

Para copiar la imagen del código en el dispositivo, se debe conectar este de la forma indicada en la Ilustración 7.

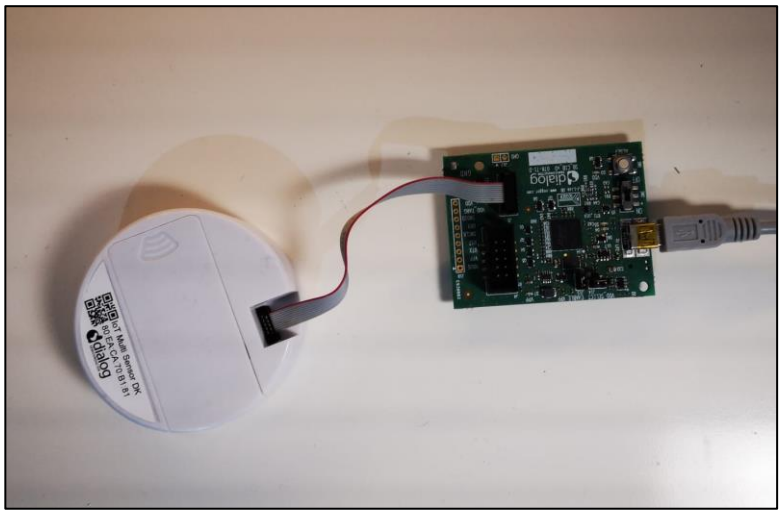

Ilustración 7: Multi sensor conectado a JTAG

Una vez conectado el dispositivo y depurado, se inicia SmartSnippets Toolbox donde se seleccionará el interfaz JTAG, el dispositivo en cuestión que reconoce el ordenador y el modelo del multi sensor. Al realizar estos pasos se obtiene en pantalla la herramienta mostrada en la Ilustración 8.

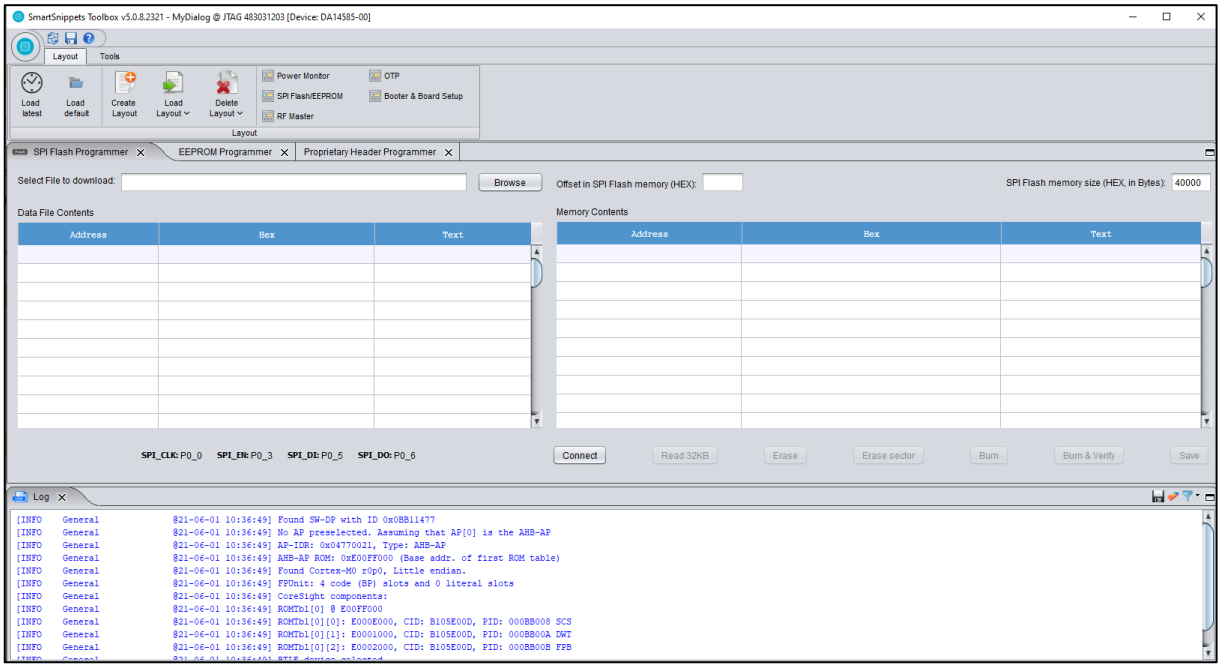

Ilustración 8: Smart toolbox

Una vez que se accede a esta interfaz, se procede a seleccionar el archivo binario anteriormente generado y enviar la orden de conexión con el dispositivo mediante los botones “Browse” y “Connect” respectivamente. En cuanto el dispositivo confirme su conexión en la pantalla lateral, se utiliza el botón “Erase” para eliminar el contenido actual de la memoria SPI y se procede a seleccionar “Burn & Verify” que se encarga de “quemar” la imagen en la memoria SPI y verificar que su contenido es correcto. En caso de que el proceso sea correcto se obtienen los mensajes mostrados en la Ilustración 9.

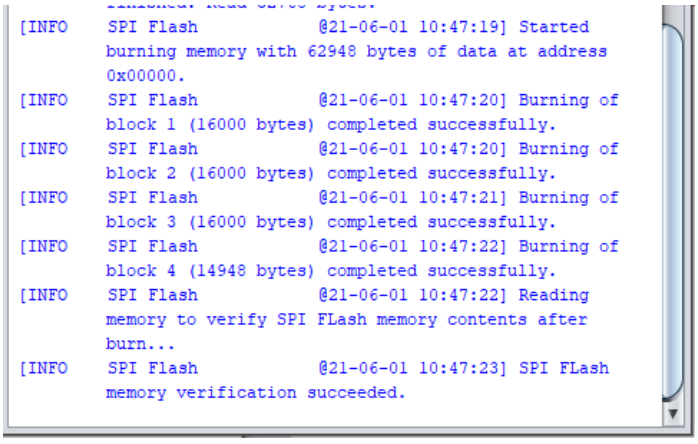

Ilustración 9: "Burn & Verify" se completó correctamente

4.5. PLATAFORMA IOT

La plataforma IoT es uno de los componentes fundamentales del sistema o prototipo que se va a desarrollar. Esta es la encargada de recibir los datos del Gateway y su almacenamiento para su posterior análisis. Existen una gran cantidad de plataformas que se pueden utilizar para este fin, sin embargo, se describe a seguir la plataforma elegida para el desarrollo del prototipo.

4.5.1. THINGSBOARD

Thingsboard [9] es una plataforma Open-Source para la recolección de datos, procesamiento, visualización, etc. Al igual que otras plataformas, está posee diversas versiones con diferentes limitaciones. En este caso las versiones difieren no solo en funcionalidad si no también en la localización de la plataforma en cuestión. Para el prototipo se analiza la versión gratuita o “Community Edition”, que posee las siguientes características:

- Visualización de datos históricos mediante gráficos.
- Motor de reglas que permite el procesamiento y acciones sobre datos entrantes.
- Filtrado de datos entrantes según entidades, atributos y ultimas telemetrías.
- Permite la configuración y el control de los dispositivos en remoto.
- Open-Source.
- Soporta los protocolos de comunicación MQTT, CoAP y HTTP.

Por otro lado, también se poseen ciertas desventajas a la hora de utilizar esta plataforma, destacando principalmente la necesidad de desplegar la plataforma en un equipo que se encuentre en las instalaciones. Este factor no representa un problema ya que al tratarse del desarrollo de un prototipo no se contempla una carga muy importante para sistemas no especializados. Por otro lado, la utilización por parte de Thingsboard de bases de datos como SQL, NoSQL o tecnologías hibridas, permite la extracción de la información para su análisis a posteriori. Por estas razones es que se desarrolla el prototipo con la plataforma IoT de Thingsboard.

4.6. ALKE 340E 24KW LITIO

El Alke 340E, es un vehículo eléctrico de trabajo versátil, diseñado para resistir duras condiciones de trabajo. En el caso de la modificación o versión realizada para EMULSA, esté vehículo dispone de una batería de 24 kW de Litio y dos motores en los que se centrará el desarrollo del prototipo. Los motores son los siguientes:

4.6.1. MOTOR DE TRACCIÓN

El vehículo está equipado con un motor asíncrono por inducción de 48V de corriente alterna CFR AME 200, que a su vez cuenta con 14kW de potencia nominal, un par máximo de 113 Nm, velocidad máxima de 44 km/h, 2900 revoluciones por minuto y frecuencia de rotación de 100 Hz. Este será el encargado de la tracción del vehículo y de convertirse en generador de energía durante las frenadas del vehículo para recargar baterías.

4.6.2. MOTOR DE BOMBAS

Por otro lado, también cuenta dicho vehículo con un motor de alta eficiencia de 48V y 3kW Moternergy inc, que a su vez cuenta con una velocidad de giro de 3000 revoluciones por minuto. Este motor de tipo corriente continua de imán permanente sin escobillas es el encargado de accionar el sistema de pistones de bombas de agua.

5. Metodología del trabajo

Durante el desarrollo del prototipo, se abordaron diferentes fases para intentar conseguir de manera satisfactoria todos los objetivos anteriormente expuestos. Estas fases pueden agruparse en tres grupos diferentes:

- Configuración del módulo multi sensor: En este apartado del proyecto se realizan modificaciones al firmware del nodo multi sensor, de forma que se pueda reducir el consumo de batería de este.
- Configuración del Gateway: Se configura y desarrolla el software necesario para la recepción, tratamiento y posterior envió de la información de la Raspberry Pi (Gateway), además de realizar las conexiones hardware necesarias para el correcto funcionamiento del prototipo.
- Configuración de la plataforma IoT: Se despliega y configura Thingsboard de forma que se puedan utilizar los servicios ofrecidos por la plataforma.

Por otro lado, las fases por las cuales se ha desarrollado el proyecto serían:

1. Configuración inicial del multi sensor.
2. Configuración inicial del Gateway.
3. Configuración inicial del HAT de comunicaciones.

4. Desarrollo firmware del multi sensor
5. Desarrollo del módulo de recepción y tratamiento de datos en el Gateway (Versión 1)
6. Pruebas de diferentes configuraciones de frecuencia de muestreo para el acelerómetro
7. Configuración de Thingsboard
8. Pruebas envío de datos mediante el HAT de comunicaciones por mqtt
9. Generación final del módulo de envió de datos a Thingsboard en el Gateway
10. Desarrollo script de conexión a WiFi en el Gateway
11. Desarrollo del módulo de recepción y tratamiento de datos en Gateway (Versión 2)
12. Integración de módulos en el Gateway
13. Scripts alternativos para el HAT de comunicaciones

Todas las fases antes mencionadas serán explicadas detalladamente en el apartado de **6. Desarrollo de la investigación**.

6. Desarrollo de la investigación

En el este apartado, se procede a detallar el trabajo realizado durante el desarrollo del prototipo anteriormente comentadas en el apartado de **5. Metodología del trabajo**. A continuación, se detallan dichos apartados.

6.1. CONFIGURACIÓN INICIAL DEL MULTI SENSOR

Una vez obtenidos los dispositivos multi sensores se procede a realizar pruebas para comprobar su correcto funcionamiento. El kit IoT Multi sensor ya dispone de una demo precargada que se ejecuta al encender el dispositivo, esta demo permite conocer el funcionamiento del dispositivo, los parámetros de configuración disponibles en la aplicación y diferentes características de los datos que produce el dispositivo después de cada lectura. Para la visualización de los datos capturados se necesita de un dispositivo Android con la aplicación IoT Sensors de Dialog-Semiconductor.

Después de instalar dicha aplicación y el multi sensor encendido con baterías, este último se encuentra anunciándose a otros dispositivos por bluetooth, por lo que será visible desde la aplicación como se muestra en la Ilustración 10.

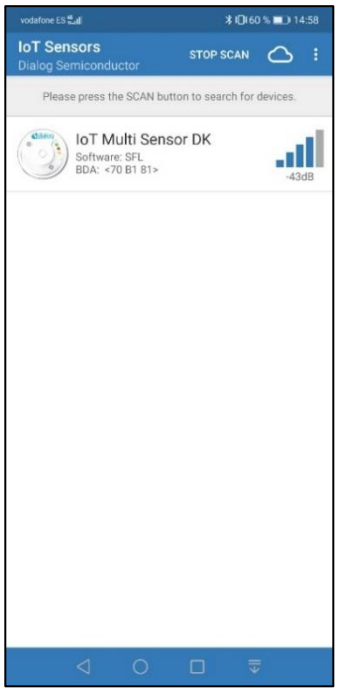

Ilustración 10: Selección dispositivo

A continuación, seleccionamos el dispositivo IoT Multi Sensor DK, lo que envía una señal al dispositivo multi sensor que activa la captura de datos y su correspondiente envió a la aplicación. Dicha acción nos permite ver los datos que están siendo capturados en tiempo real como podemos ver en la Ilustración 11 e Ilustración 12.

Ilustración 11: Captura de datos ambientales vía app

Ilustración 12: Captura de datos IMU sensors vía app

Se puede observar que en el caso de la demo preinstalada se encuentra activado el sensor ambiental, recolectando datos de temperatura, humedad y presión, así como el sensor de luz ambiental. Por otro lado, se encuentra desactivado el acelerómetro y giroscopio del nodo, razón por la cual no se aprecia ningún dato capturado en la Ilustración 12.

6.2. CONFIGURACIÓN INICIAL DEL GATEWAY

Una vez comprobada la configuración inicial del dispositivo DA14585 IoT multi sensor, se procede a realizar las configuraciones pertinentes de forma que el dispositivo Gateway consiga comunicarse correctamente con este y recibir los datos capturados.

El primer paso en la configuración de la Raspberry Pi utilizada como Gateway, será la instalación de un sistema operativo. Este será el sistema operativo Raspberry Pi OS, basado en Debian. Cabe destacar que, en busca de la menor carga posible por parte del sistema operativo sobre el Gateway se utiliza una imagen ligera sin entorno de escritorio. Luego se procede a activar el servicio de SSH de forma que no se necesiten conectar dispositivos periféricos al Gateway, permitiendo su instalación, configuración y programación desde un ordenador conectado a la misma red mediante Putty como muestra la Ilustración 13.

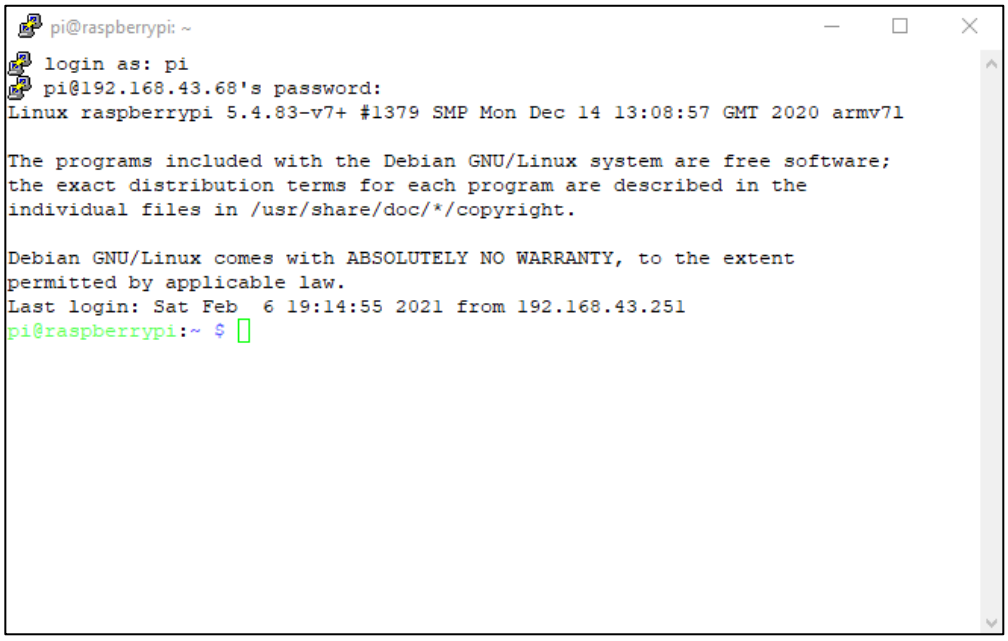

Ilustración 13: Conexión con Gateway mediante Putty

Para la instalación de dicho servicio basta con encender el Gateway conectado a una pantalla y en el caso de no disponer de interfaz gráfica, un teclado. Una vez encendido el dispositivo, se inicia sesión con las credenciales de fabrica del mismo y se introduce el comando ***sudo raspi-***

config, esté presenta un menú por pantalla con diferentes configuraciones posibles para el equipo. En él se accede a *Interface Options* y luego a *SSH*. Dicha opción nos permite iniciar el servidor SSH de la Raspberry, de forma que se puedan realizar cambios en el dispositivo desde el ordenador como se comentó anteriormente.

Posteriormente se deben instalar ciertas librerías de forma que se pueda ejecutar correctamente el cliente a desarrollar en Python. Entre estos componentes necesarios encontramos:

- Paho-mqtt (v1.3.1): Librería utilizada para la creación del cliente MQTT mediante el cual se envían los resultados de las lecturas del Gateway a la plataforma IoT Thingsboard.
- PyBluez (v0.22): Librería que proporciona las utilidades Bluetooth al prototipo, necesaria para la comunicación con los nodos multi sensores.
- Enum34 (v1.1.6), Requests (v2.18.74) y Urllib3 (v1.22): Librerías necesarias por dependencias con una o más de las anteriormente mencionadas.
- Python 3: Lenguaje de programación utilizado para el desarrollo del programa principal del Gateway.

Una vez instaladas se realizan ciertas modificaciones sobre el proyecto de demostración proporcionado por Dialog-semiconductor ya que, en el caso de dicho proyecto base, existen muchas funcionalidades que no serán de utilidad ninguna para el prototipo que se desarrolla. De todo el código proporcionado en el proyecto base, se ha decidido eliminar las clases pertinentes al envío de datos a la nube ya que no utiliza el protocolo MQTT para el envío de estos, por lo que no es compatible con la plataforma IoT de Thingsboard utilizada. Por otro lado, se eliminan clases auxiliares que corresponden con la configuración del dispositivo debido a que, en el caso del prototipo, dichas configuraciones se realizarán en el firmware del multi sensor y no en el Gateway.

La anteriormente mencionada reducción de código dejo dos clases del proyecto llamadas DEKGateway y DEKMain, ya que se consideró que el resto de código no sería reutilizado en el proyecto. Estas clases también reciben cambios en su composición, reduciendo así el código a lo estrictamente necesario para el desarrollo del prototipo permitiendo unir ambos archivos y clases en un archivo único llamado **main.py**, para un mejor manejo de variables y claridad al momento de desarrollar el resto de la solución.

La funcionalidad conseguida por este nuevo archivo **main.py** sería la siguiente:

1. Conectar al kit multi sensor mediante la utilización de Peripheral, componente de la librería bluepy y la dirección MAC del dispositivo definida en la variable MAC_ADDRS.
2. Asignar un manejador que reciba la información dada mediante llamadas asíncronas como son las notificaciones Bluetooth.
3. Entra en un bucle del que solo se sale en caso de que exista un error en la conexión Bluetooth con el dispositivo. Dentro de dicho bucle se espera mediante p.waitForNotifications(1.0), que la clase delegado asignada a p, reciba una notificación. En caso de que sea recibida dicha notificación, se ejecuta el método post("0").
4. El método post(), utiliza el parámetro enviado para seleccionar solo los sensores que interesan y extraer o descartar la información recibida según estos. El "0" en este caso representa todos los sensores que se encuentren activados en el dispositivo multi sensor. Posteriormente llama con este string a get_data_msg, que se encarga de procesar los datos en crudo mediante process_data() y otros métodos auxiliares, para luego devolverlo a post(), donde se tratarán los datos convertidos.

A grandes rasgos esta es la funcionalidad dada por la clase **main.py** en el proyecto base. Ante su ejecución se pueden ver en la Ilustración 14 los mensajes obtenidos de la depuración, mostrando para este caso los valores recuperados por el sensor de temperatura, humedad y presión de la demo disponible en el dispositivo multi sensor.

```
chosen sensor has data
data[0:1]
2
data[1:]
60.75
Mensaje
['2', '60.75']
chosen sensor has data
data[0:1]
1
data[1:]
20.28
Mensaje
['1', '20.28']
chosen sensor has data
data[0:1]
3
data[1:]
100405
Mensaje
['3', '100405']
```

Ilustración 14: Resultados datos ambientales proyecto base

6.3. CONFIGURACIÓN INICIAL DEL HAT DE COMUNICACIONES

Otro de los aspectos claves del prototipo es la comunicación mediante el HAT SIM7000E que permite el envío de datos mediante LTE como tecnología de comunicación. Dicho modulo o HAT se debe conectar con el Gateway para posteriormente comprobar su funcionamiento. Esto se consigue a través de la interfaz USB que posee el dispositivo, permitiendo así mediante un cable USB conectar el HAT de comunicaciones con el Gateway. Luego se monta uno encima del otro como muestra la Ilustración 6 de forma que el módulo ocupe el menor espacio posible.

Por otro lado, el módulo de comunicación necesita una tarjeta SIM para el envío de la información por lo que se debe dar de alta en la plataforma ThingsMobile, que es la que se utilizará para dicho propósito. En este portal se debe ingresar las credenciales de acceso dadas al adquirir las tarjetas SIM para poder acceder al área restringida que permite la monitorización de las SIM contratadas (ver Ilustración 15).

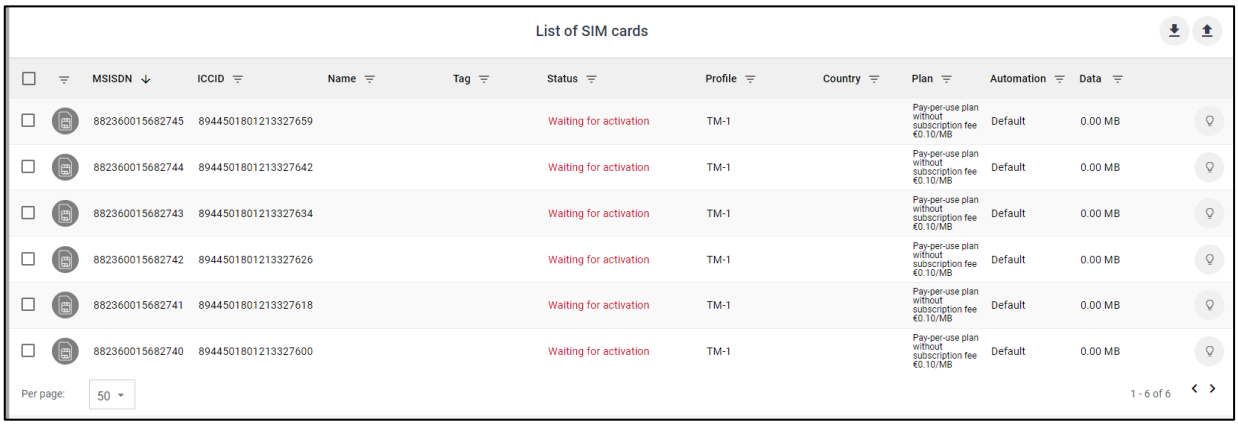

Ilustración 15: Dashboard ThingsMobile

Una vez dentro se selecciona la tarjeta SIM a activar y se escoge el plan más conveniente para la utilización de cada una de estas, como se observa en la Ilustración 16. En este caso se decide por un plan de tarifa según datos consumidos sin gasto fijo mensual, es decir, solo se pagan los datos que se utilizan al mes sin un plus mensual por subscripción. Además, se destaca la posibilidad de automatizar el pago o recarga de fondos, aunque, al tratarse de un prototipo se escoge la recarga manual difiriendo así del escenario más probable en un despliegue real de una herramienta similar (ver Ilustración 17).

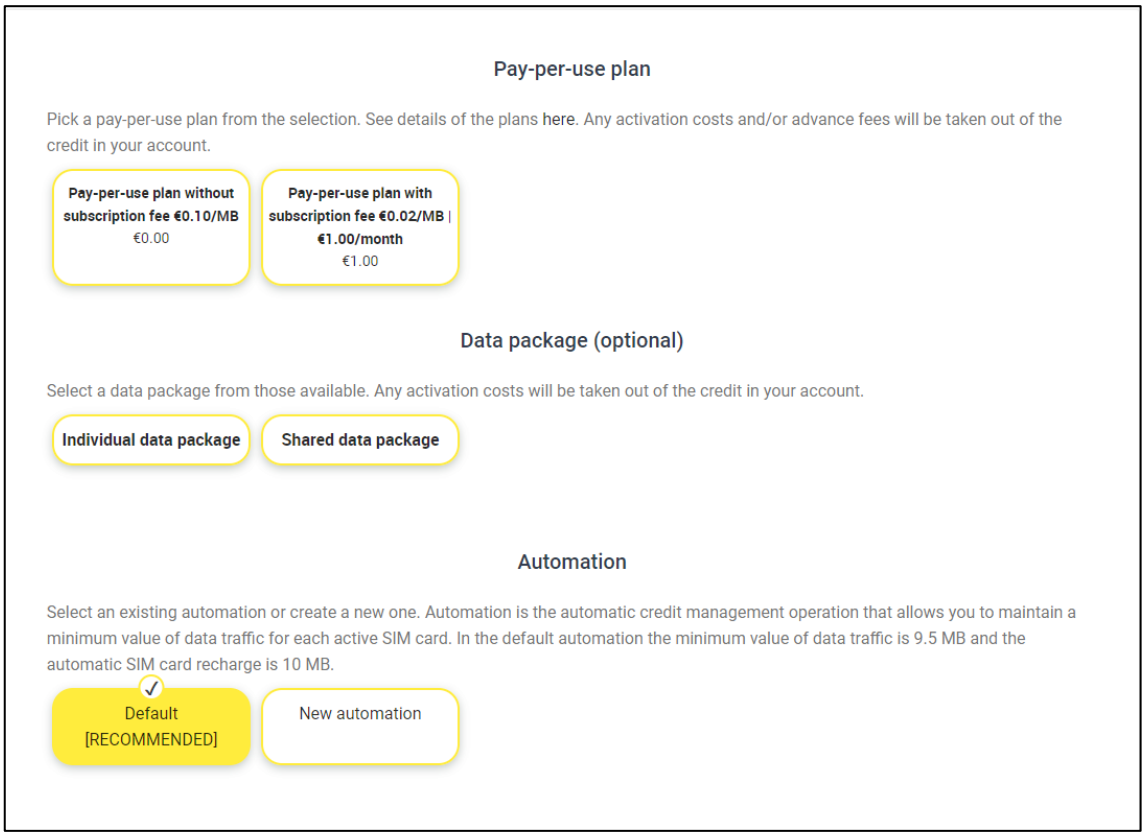

Ilustración 16: Opciones de activación tarjeta SIM

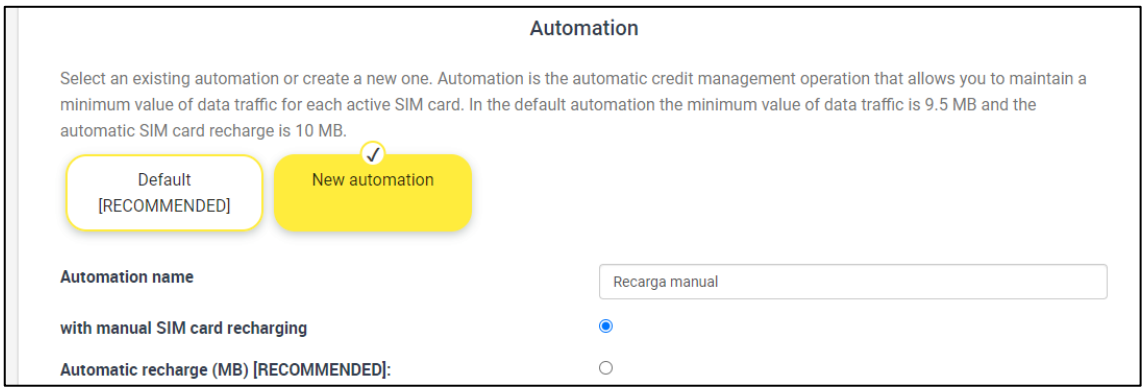

Ilustración 17: Opciones de recarga tarjetas SIM

Una vez aceptadas las condiciones del contrato, se activa automáticamente la tarjeta SIM permitiendo observar detalles de esta como datos consumidos, número, código de operador, etc. (ver Ilustración 18), atributos interesantes de monitorizar en entornos de despliegue reales.

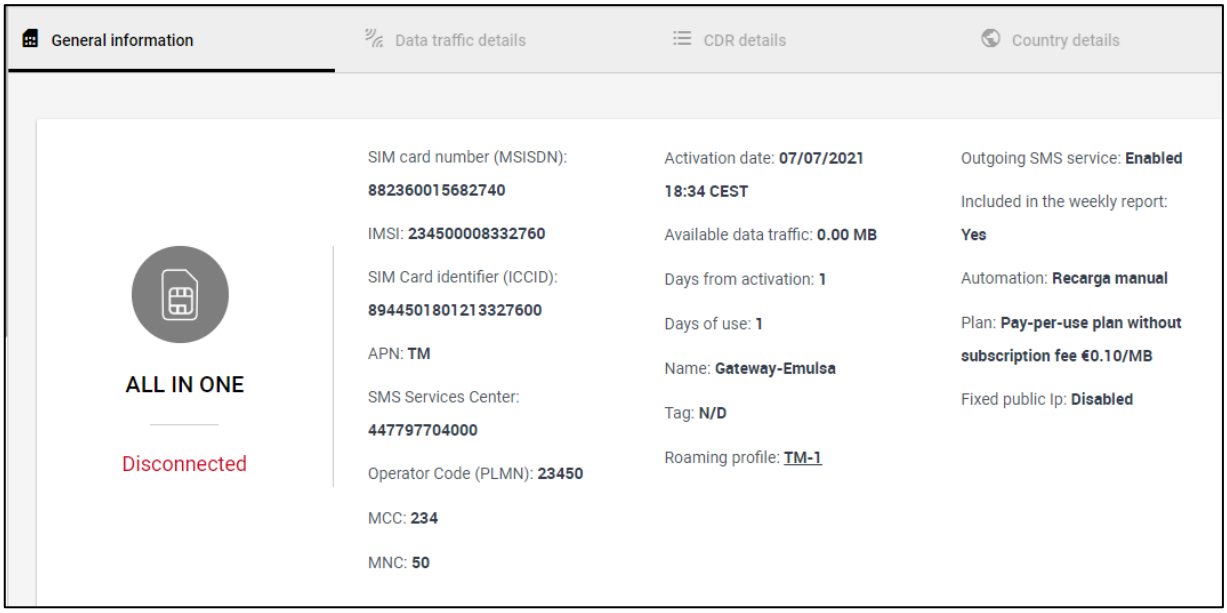

Ilustración 18: Datos de la tarjeta SIM

En cuanto se dispone de una tarjeta SIM funcional se procede a comprobar el correcto funcionamiento del módulo de comunicación. Para ello en una primera instancia se conecta el cable USB del HAT al PC de forma que las primeras pruebas puedan ser realizadas cómodamente, dicho proceso se replica posteriormente en el Gateway pero no será comentado para evitar redundancias.

En primer lugar, mediante el programa AT Command Tester SM se debe acceder al puerto COM correspondiente con el módulo de comunicación. Para encontrar el puerto COM que corresponde con el módulo, se debe acceder al administrador de dispositivos donde encontraremos algo similar a lo mostrado en la Ilustración 19. En esta se aprecia el puerto COM3 como el puerto designado para el envío de comandos AT.

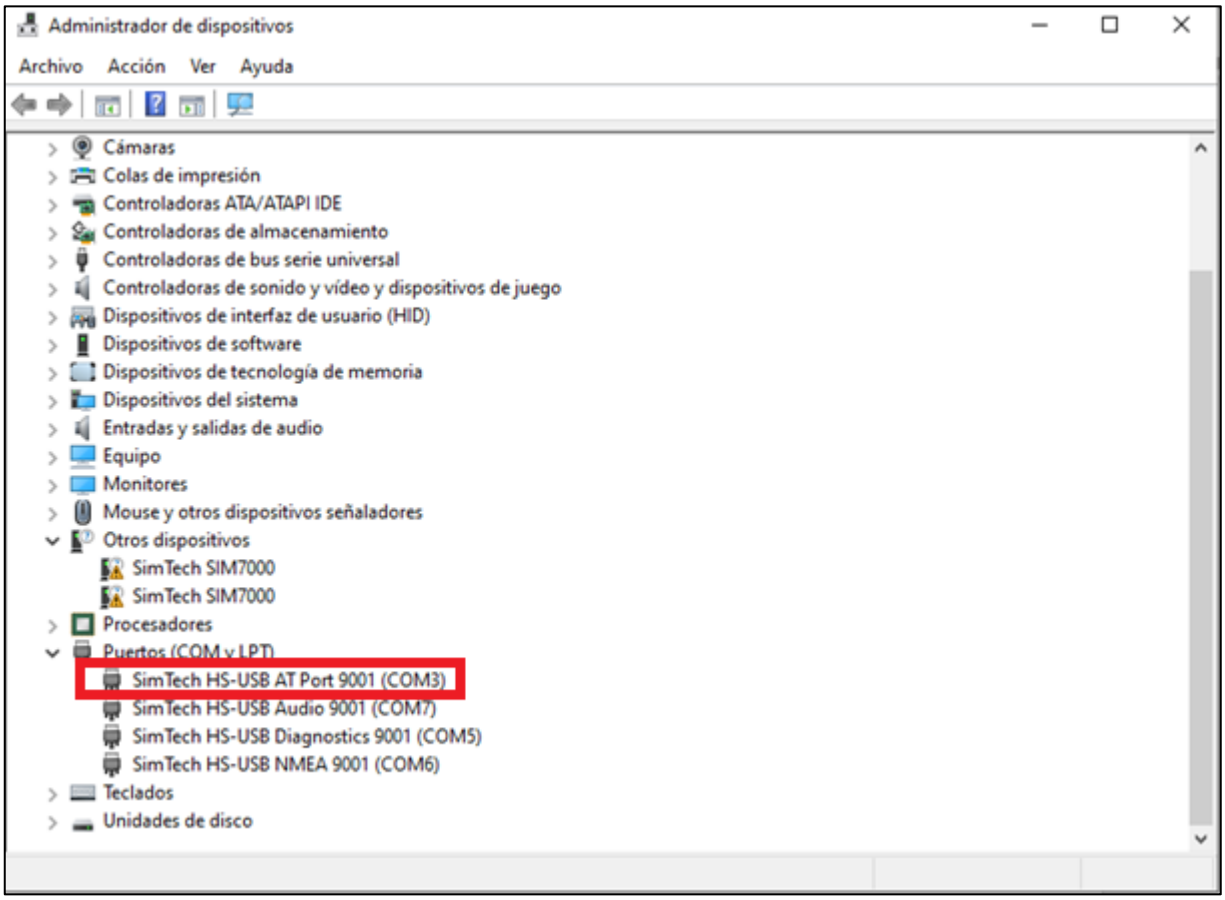

Ilustración 19: Puertos COM del módulo de comunicaciones

Una vez identificado el puerto de comunicación del dispositivo para el envío de comandos AT, se utiliza la aplicación comentada anteriormente para conectar con dicho puerto. Para ello, al iniciar AT Command Tester SM se selecciona “Find ports” y se elige en el desplegable de puertos en COM3 anteriormente mencionado para conseguir conectar con el dispositivo, donde se obtienen mensajes de confirmación de conexión con el dispositivo como se aprecia en la Ilustración 20.

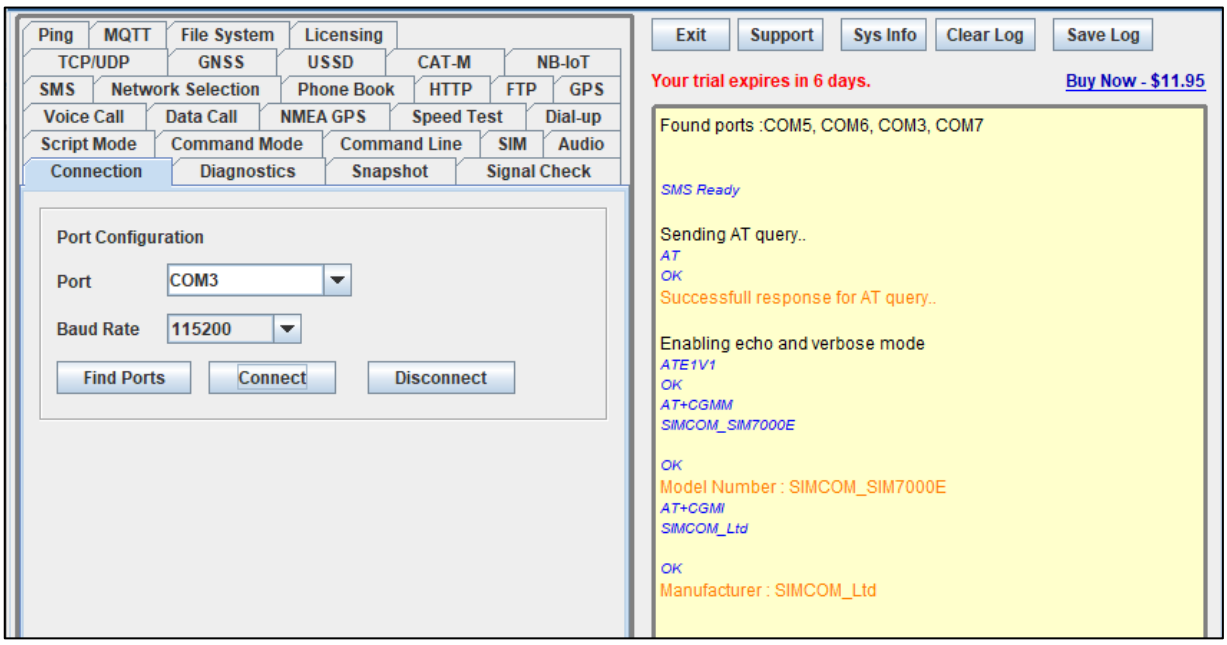

Ilustración 20: Conexión con módulo SIM7000E

Una vez obtenidos los mensajes de confirmación, se pueden enviar comandos AT para confirmar la comunicación con el proveedor de red. Para ello primero se debe acceder al apartado SIM e introducir la contraseña de está con el comando AT+CPIN=XXXX, reemplazando las X con el código PIN de la tarjeta SIM. En caso de ser satisfactorio el PIN

introducido se obtiene la respuesta "+CPIN: READY" al comando "AT+CPIN?". Posteriormente se solicitan datos propios del módulo, así como datos propios del proveedor de la red mediante el apartado "Snapshot". Dicho apartado envía los comandos necesarios de forma que se obtiene el resultado que se muestra en la Ilustración 21.

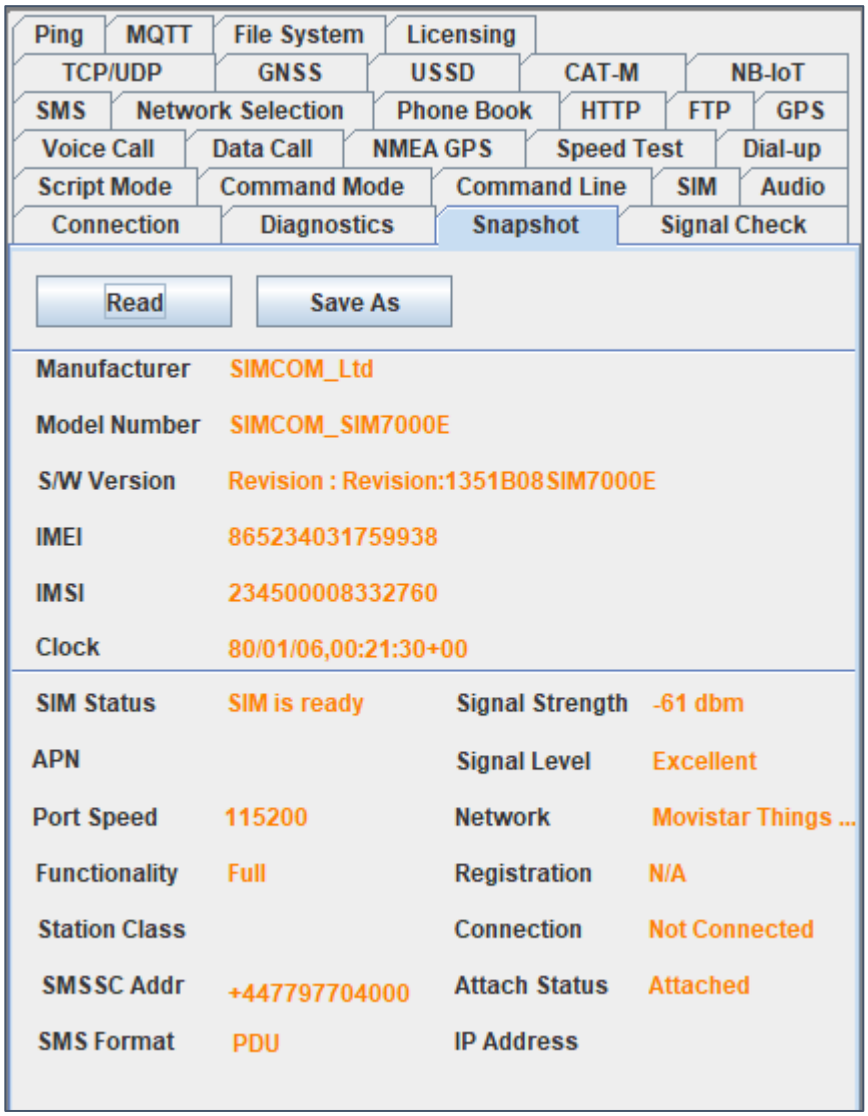

Ilustración 21: Datos del módulo y del proveedor de red

Con lo anteriormente comentado se comprueba el correcto funcionamiento del módulo de comunicación, así como la conexión con el proveedor de red. Si bien existen algunos parámetros que se deben definir para el envío de datos, estos se deben realizar durante el arranque del sistema del Gateway, por lo que será abordado en apartados posteriores.

6.4. DESARROLLO DEL FIRMWARE DEL MULTI SENSOR

Con el fin de cumplir con los objetivos propuestos para el prototipo, se debe desarrollar un firmware específico para los nodos multi sensores de forma que cómo se comentó en apartados anteriores, el dispositivo sea capaz de registrar temperatura y humedad ambiental, además de las vibraciones de los motores.

En vez de comenzar a desarrollar un firmware desde 0 para los dispositivos multi sensores, se utilizará el código fuente facilitado por Dialog-Semiconductor. Dicho código fuente posee ciertas características que no son necesarias para los objetivos que se buscan alcanzar, entre ellas podemos destacar la utilización de ciertos sensores como pueden ser el de presión atmosférica, el sensor de luz ambiental, el giroscopio y el magnetómetro, sensores que si se encuentran activos en el código base del proyecto. Por otro lado, el archivo descargado también posee diferentes proyectos según la aplicación que s ele desee dar al nodo multi sensor. Entre todos los proyectos posibles se decide emplear el de IoT de sensores que permite lo descrito anteriormente.

Una vez descargado e importado el proyecto se procede en primer lugar a activar los sensores necesarios y más importante desactivar aquellos que no serán utilizados en la solución

final, buscando de esta forma que el dispositivo funcione de la forma más eficiente para aprovechar la batería del dispositivo, de la mejor manera posible y promover así un funcionamiento más duradero.

Para la activación/desactivación de los diferentes sensores disponibles en el dispositivo multi sensor se debe acceder a la clase "da1458x_config_basic.h". Este fichero de cabecera es el encargado de las configuraciones básicas del dispositivo DA14585 para las funciones de seguridad, impresión de depuración y control de sensores incluidos en el proyecto. En este podemos encontrar en las líneas 75 y 94, los siguientes parámetros que se muestran en la Ilustración 22.

```
75 #define VCNL4010_OPTO_SENSOR_AVAILABLE
76 #define AK099XX_MAGNETO_SENSOR_AVAILABLE
77 #define ACCEL_SENSOR_AVAILABLE                  // Must select only ONE type of accel sensor
78 #ifdef ACCEL_SENSOR_AVAILABLE
79     #define ICM4XX_ACCEL_SENSOR_AVAILABLE
80     #undef BMI160_ACCEL_SENSOR_AVAILABLE
81     #ifdef  BMI160_ACCEL_SENSOR_AVAILABLE
82         #undef BMI_ON_EXTERNAL_HEADER           // Select if bmi160 sensor is placed in external header
83     #endif
84 #endif
85
86 #define BME680_ENVIRONM_SENSOR_AVAILABLE
87 #ifdef BME680_ENVIRONM_SENSOR_AVAILABLE
88     #undef IAQ_ENABLED                          // Include the iAQ library, process gas data on board
89     #ifdef IAQ_ENABLED
90         #define IAQ_RESTORE_STATUS_ENABLE        // Save progress on iAQ calculation
91     #else
92         #define RAW_GAS_ENABLED                 // Raw mode, iAQ not included, application will calculate the iAQ
93     #endif
94 #endif//BME680_ENVIRONM_SENSOR_AVAILABLE
95 // if both IAQ_ENABLED & RAW_GAS_ENABLED are undefined the GAS sensor will be disabled
```

Ilustración 22: da1458x_config_basic.h

Estos parámetros poseen el valor por defecto por la palabra que les precede. En este caso se debe cambiar el #define de VCNL4010_OPTO_SENSOR_AVAILABLE y AK099XX_MAGNETO_SENSOR_AVAILABLE por #undef, para desactivar el sensor de luz

y magnetómetro existente. También es necesario cambiar el valor de la variable CTF_PRINTF ubicado en la línea 70 por #define cuando se realice la depuración del proyecto como se muestra en la Ilustración 23, ya que permite activar los mensajes en modo debug del firmware.

```
65 /*****************************************************************************************************************/
66 /*UART Console Print. Enables serial interface logging mechanism. If CFG_PRINTF is defined CFG_PRINTF_UART2     */
67 /* controls the uart module used. If it is defined UART2 is used. If not, UART is used. uart or uart2 driver    */
68 /* must be included in project respectively.                                                                     */
69 /*****************************************************************************************************************/
70 #define CFG_PRINTF
71 #ifdef CFG_PRINTF
72     #define CFG_PRINTF_UART2
73 #endif
```

Ilustración 23: Activar mensajes de depuración

Por otro lado, también es necesario acceder a la clase “user_config.h” y cambiar la definición de la constante “app_default_sleep_mode”, que se encuentra en la línea 68, por “ARCH_SLEEP_OFF” para poder realizar la depuración del programa correctamente, como se puede apreciar en la Ilustración 24. Dicha variable deshabilita el sleep mode del procesador del módulo multi sensor. Cabe destacar que tanto este parámetro como el anterior deben ser cambiados al terminar el desarrollo del código, ya que son necesarios para el desarrollo del firmware, pero no son útiles para la puesta en producción de este.

```
59 /****************************************
60  * Default sleep mode. Possible values are:
61  *
62  * - ARCH_SLEEP_OFF
63  * - ARCH_EXT_SLEEP_ON
64  * - ARCH_EXT_SLEEP_OTP_COPY_ON
65  *
66  ****************************************
67  */
68 const static sleep_state_t app_default_sleep_mode = ARCH_SLEEP_OFF;
69
```

Ilustración 24: Definición del sleep mode

Una vez realizados estos pasos, se procede a conectar el dispositivo multi sensor con la aplicación móvil en un dispositivo Android. Como se puede observar en la Ilustración 25, el sensor de luz ambiental se encuentra deshabilitado y solo se disponen de los sensores icm42605 y bme680 activados, por lo que se puede comprobar que las modificaciones realizadas para ello son correctas.

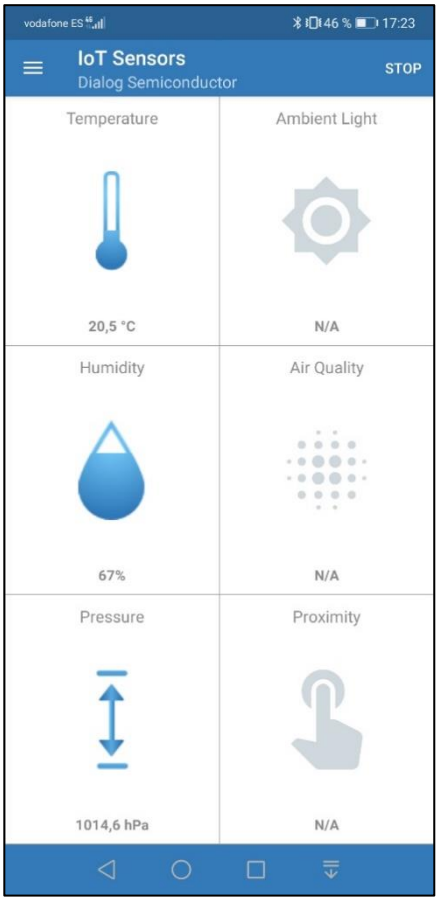

Ilustración 25: Sensor de luz ambiental desactivado

Existe un pequeño matiz con respecto a la desactivación de los sensores no esenciales para el funcionamiento del prototipo. El sensor de humedad y temperatura es el sensor ambiental bme680 como se comentó anteriormente. El problema de este sensor es que además de registrar esos dos parámetros necesarios para nuestro prototipo, también recolecta la presión atmosférica, dato que no utilizamos en el prototipo a realizar. Algo similar sucede con el sensor icm42605, ya que además de proveer la información del acelerómetro, también provee información del giroscopio.

En primera instancia, se procede a eliminar algunas de las funciones que se ejecutarían al comenzar el funcionamiento del dispositivo en ciertos sensores que no se van a utilizar, pero que como antes se comentó, se encuentran asociados con una métrica que si es de interés. Para ello en la clase "user_app_iot_config.h", encargada de las definiciones específicas de la aplicación como los parámetros de calibración de los sensores, se alteraran los valores de las variables "GYRO_SDC_ENABLED", MAGNETO_CAL_ENABLES y MAGNETO_CAL_AUTO_STORE, que se encuentran entre las líneas 40 y 46 del código como se muestra en la Ilustración 26 a #undef, ya que estás proveen la definición de unas funciones de calibración que se ejecutan al principio del ciclo de vida del sensor, particularmente del giroscopio y magnetómetro.

```
40 /// Compilation Switches
41 #define USE_FAST_ACC_CAL
42 #undef USE_FAST_GYRO_CAL
43 #define GYRO_SDC_ENABLED        // Gyro static drift compensation
44 #define MAGNETO_CAL_ENABLED
45 #define MAGNETO_CAL_AUTO_STORE
46 #define USE_SPI_FLASH_CONFIG    // If client does not want ot use external flash, use default configuration
47
```

Ilustración 26: user_app_iot_config.h

Por otro lado, existen otros cálculos innecesarios que se pueden impedir de forma que sea posible reservar la mayor cantidad de energía posible, ya que si bien no se pueden desactivar las secciones de un sensor encargadas de capturar ciertos parámetros, como puede ser en el caso del sensor ambiental la presión atmosférica, si es posible deshabilitar su funcionamiento. En este caso se habla sobre impedir la búsqueda de los valores por parte del controlador en la memoria del dispositivo.

Para el sensor bme680, accedemos a la clase que posee su mismo nombre con extensión c, está clase en conjunto con otras ubicadas en el mismo archivo, poseen los drivers de fábrica de

dicho sensor. Una vez en esta clase se busca la línea 951 del código donde se encontrará el código mostrado en la Ilustración 27. Esta función es la encargada de leer los valores recolectados por este sensor en la memoria del dispositivo y de realizar ciertas operaciones de forma que los datos no sean entregados en crudo al usuario.

```
951 static int8_t read_field_data(struct bme680_field_data *data, struct bme680_dev *dev)
952 {
953         int8_t rslt;
954         uint8_t buff[BME680_FIELD_LENGTH] = { 0 };
955         uint8_t gas_range;
956         uint32_t adc_temp;
957         uint32_t adc_pres;
958         uint16_t adc_hum;
959         uint16_t adc_gas_res;
960         uint8_t tries = 10;
961
962         /* Check for null pointer in the device structure*/
963         rslt = null_ptr_check(dev);
964         do {
965                 if (rslt == BME680_OK) {
966                         rslt = bme680_get_regs(((uint8_t) (BME680_FIELD0_ADDR)), buff, (uint16_t) BME680_FIELD_LENGTH, dev);
967
968                         data->status = buff[0] & BME680_NEW_DATA_MSK;
969                         data->gas_index = buff[0] & BME680_GAS_INDEX_MSK;
970                         data->meas_index = buff[1];
971
972                         /* read the raw data from the sensor */
973                         adc_pres = (uint32_t) (((uint32_t) buff[2] * 4096) | ((uint32_t) buff[3] * 16) | ((uint32_t) buff[4] / 16));
974                         adc_temp = (uint32_t) (((uint32_t) buff[5] * 4096) | ((uint32_t) buff[6] * 16) | ((uint32_t) buff[7] / 16));
975                         adc_hum = (uint16_t) (((uint32_t) buff[8] * 256) | (uint32_t) buff[9]);
976                         adc_gas_res = (uint16_t) ((uint32_t) buff[13] * 4 | (((uint32_t) buff[14]) / 64));
977                         gas_range = buff[14] & BME680_GAS_RANGE_MSK;
978
979                         data->status |= buff[14] & BME680_GASM_VALID_MSK;
980                         data->status |= buff[14] & BME680_HEAT_STAB_MSK;
981
982                         if (data->status & BME680_NEW_DATA_MSK) {
983                                 data->temperature = calc_temperature(adc_temp, dev);
984                                 data->pressure = calc_pressure(adc_pres, dev);
985                                 data->humidity = calc_humidity(adc_hum, dev);
986                                 data->gas_resistance = calc_gas_resistance(adc_gas_res, gas_range, dev);
987                                 break;
988                         } else {
989                                 dev->delay_ms(BME680_POLL_PERIOD_MS);
990                         }
991                 }
992                 tries--;
993         } while (tries);
994
995         if (!tries)
996                 rslt = BME680_W_NO_NEW_DATA;
997
998         return rslt;
999 }
```

Ilustración 27: función de lectura en memoria bme680

En dicha función se elimina la línea 973, para evitar la búsqueda de los valores de presión y se modifica la línea 984, ambas mostradas en la Ilustración 28, asignándole un valor de 0 ya que espera que se le retorne un valor de tipo uint32_t, evitándonos así modificar diferentes llamadas que se realicen a la estructura "data" que contiene los datos.

```
972                     /* read the raw data from the sensor */
973                     adc_pres = (uint32_t) (((uint32_t) buff[2] * 4096) | ((uint32_t) buff[3] * 16) | ((uint32_t) buff[4] / 16));
974                     adc_temp = (uint32_t) (((uint32_t) buff[5] * 4096) | ((uint32_t) buff[6] * 16) | ((uint32_t) buff[7] / 16));
975                     adc_hum = (uint16_t) (((uint32_t) buff[8] * 256) | (uint32_t) buff[9]);
976                     adc_gas_res = (uint16_t) ((uint32_t) buff[13] * 4 | (((uint32_t) buff[14]) / 64));
977                     gas_range = buff[14] & BME680_GAS_RANGE_MSK;
978
979                     data->status |= buff[14] & BME680_GASM_VALID_MSK;
980                     data->status |= buff[14] & BME680_HEAT_STAB_MSK;
981
982                     if (data->status & BME680_NEW_DATA_MSK) {
983                             data->temperature = calc_temperature(adc_temp, dev);
984                             data->pressure = calc_pressure(adc_pres, dev);
985                             data->humidity = calc_humidity(adc_hum, dev);
986                             data->gas_resistance = calc_gas_resistance(adc_gas_res, gas_range, dev);
987                             break;
```

Ilustración 28: Líneas a cambiar en bme680.c

Con las acciones se consigue que el dispositivo no acceda a la memoria en busca de datos capturados por el sensor de presión, así como evitar las operaciones de interpretación de estos datos al mostrar siempre el valor cero como se puede apreciar en la Ilustración 29, resultado de conectar el multi sensor a la aplicación móvil para comprobar que los cambios son efectivos.

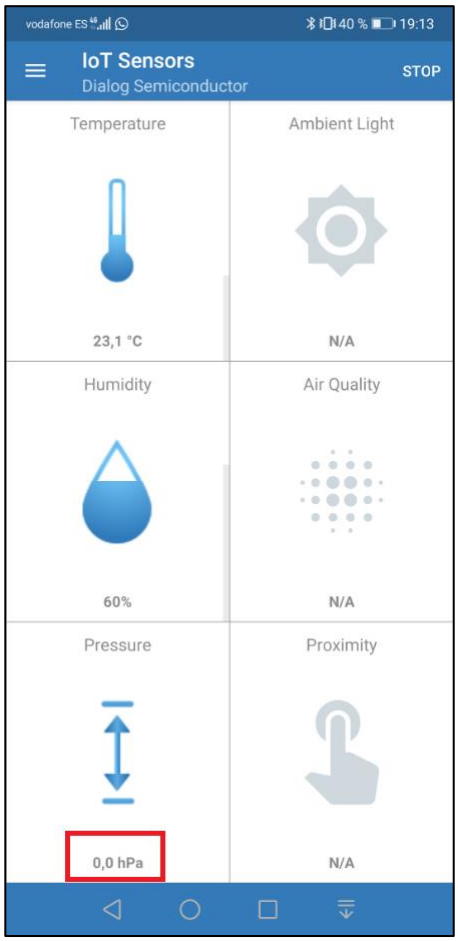

Ilustración 29: Resultado de las modificaciones del sensor bme680

Posteriormente se procede a desactivar el giroscopio. Este caso difiere de lo anteriormente visto. Si bien este sensor posee también información de interés para el producto final, este sí incluye dentro del firmware del fabricante la opción para desactivar los módulos del sensor (acelerómetro y giroscopio). Para ello existen diversas formas que pueden ser utilizadas, realizándolo en este caso mediante la clase "motion_icm4x6.c", encargada de definir una interfaz común para el acceso a los sensores. En esta se busca la línea 36 del código, donde se encontrará la función mostrada en la Ilustración 30.

```
uint8_t motion_setup(void)
{
    uint8_t res = 0;
    if(GetBits16(SYS_STAT_REG, PER_IS_DOWN))
        periph_init();

    ICM426XX_SERIAL_IF_TYPE_t serif_type = ICM426XX_UI_SPI4;
    ICM426XX_INTR_SEL_t SelIntr = ICM426XX_INTR_2;
    uint8_t fifoSize = user_accel_setup.wm_fifo_size;

    res = SetupInvDevice426(accel_spi_bus_read, accel_spi_bus_write, serif_type, SelIntr,fifoSize, user_accel_setup.accel_en, user_accel_setup.gyro_en);

    // Configure Icm426xx, this will also start the module
    res = ConfigureInvDevice(
    user_accel_setup.power_mode,
    user_accel_setup.acc_sampl_freq_g,
    user_accel_setup.gyr_sampl_freq_dps,
    user_accel_setup.acc_output_data_rate,
    user_accel_setup.gyr_output_data_rate);

    return res;
}
```

Ilustración 30: Función motion_setup de motion_icm4x6.c

Dicha función es llamada cuando se configuran los parámetros del sensor icm42605, por lo que añadiremos antes de la devolución de res la llamada a la función DisableGyroModule(). Esta función definida en la implementación del firmware de fabrica del sensor (Icm426xx_impl.c), se encarga de desactivar el giroscopio. La efectividad de dicho cambio se puede apreciar en la Ilustración 31, donde se observa la captura de datos por parte del acelerómetro, mostrado por el sombreado que se presenta en la casilla de este y su inexistencia para el caso del giroscopio.

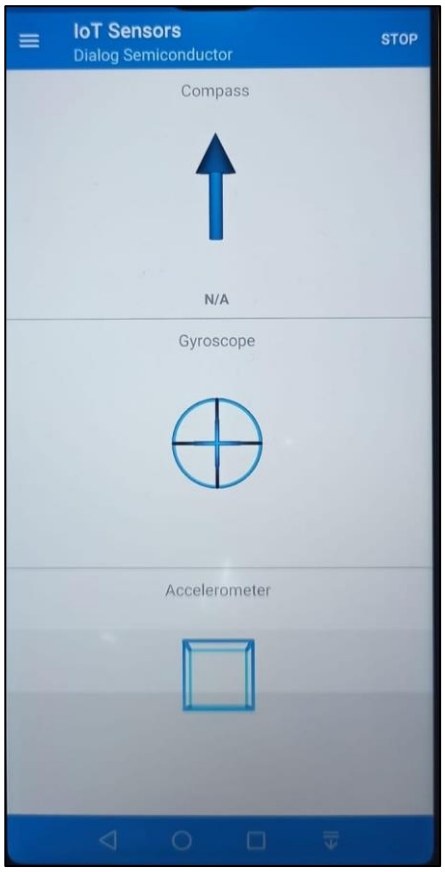

Ilustración 31: Resultado de desactivar el giroscopio

Una vez activados todos los sensores necesarios y desactivados aquellos que no proveen utilidad para el prototipo, el siguiente paso será encontrar la mejor configuración del acelerómetro para conseguir la mayor precisión a la hora de capturar los datos. Para ello primero se accede al archivo "ICM423XXDefs.h", archivo de cabecera que define parámetros que serán utilizados durante la configuración del sensor. En dicha clase se debe cambiar la frecuencia a la que se capturan los datos del acelerómetro, para lo que se debe ir a la línea 895 donde se encontrará lo mostrado en la Ilustración 32.

```
893 /* ACCEL_ODR */
894 #define BIT_ACCEL_CONFIG0_ODR_POS       0
895 #define BIT_ACCEL_CONFIG0_ODR_MASK   0x0F
896
897 /** @brief Accelerometer ODR selection
898  */
899 typedef enum
900 {
901         ICM426XX_ACCEL_CONFIG0_ODR_500_HZ    = 0xF,  /*!< 500 Hz (2 ms)*/
902         ICM426XX_ACCEL_CONFIG0_ODR_1_5625_HZ = 0xE,  /*!< 1.5625 Hz (640 ms)*/
903         ICM426XX_ACCEL_CONFIG0_ODR_3_125_HZ  = 0xD,  /*!< 3.125 Hz (320 ms)*/
904         ICM426XX_ACCEL_CONFIG0_ODR_6_25_HZ   = 0xC,  /*!< 6.25 Hz (160 ms)*/
905         ICM426XX_ACCEL_CONFIG0_ODR_12_5_HZ   = 0xB,  /*!< 12.5 Hz (80 ms)*/
906         ICM426XX_ACCEL_CONFIG0_ODR_25_HZ     = 0xA,  /*!< 25 Hz (40 ms)*/
907         ICM426XX_ACCEL_CONFIG0_ODR_50_HZ     = 0x9,  /*!< 50 Hz (20 ms)*/
908         ICM426XX_ACCEL_CONFIG0_ODR_100_HZ    = 0x8,  /*!< 100 Hz (10 ms)*/
909         ICM426XX_ACCEL_CONFIG0_ODR_200_HZ    = 0x7,  /*!< 200 Hz (5 ms)*/
910         ICM426XX_ACCEL_CONFIG0_ODR_1_KHZ     = 0x6,  /*!< 1 KHz (1 ms)*/
911         ICM426XX_ACCEL_CONFIG0_ODR_2_KHZ     = 0x5,  /*!< 2 KHz (500 us)*/
912         ICM426XX_ACCEL_CONFIG0_ODR_4_KHZ     = 0x4,  /*!< 4 KHz (250 us)*/
913         ICM426XX_ACCEL_CONFIG0_ODR_8_KHZ     = 0x3,  /*!< 8 KHz (125 us)*/
914         ICM426XX_ACCEL_CONFIG0_ODR_16_KHZ    = 0x2,  /*!< 16 KHz (62.5 us)*/
915         ICM426XX_ACCEL_CONFIG0_ODR_32_KHZ    = 0x1,  /*!< 32 KHz (31.25 us)*/
916         ICM426XX_ACCEL_CONFIG0_ODR_64_KHZ    = 0x0,  /*!< 64 KHz (15.625 us)*/
917 } ICM426XX_ACCEL_CONFIG0_ODR_t;
```

Ilustración 32: Frecuencias del acelerómetro

Como bien se puede observar en la Ilustración 32, en la línea 895 se define el ODR del acelerómetro, que corresponde a la velocidad de salida (Output Data Rate) del sensor. Este valor se encuentra inicialmente definido a 500 Hz lo que representaría una frecuencia de muestreo pequeña de forma que la precisión no es la ideal. Sin embargo, se puede apreciar que existen muchos otros valores posibles para el ODR del acelerómetro, llegando en su punto máximo a 64 KHz según la estructura definida a partir de la línea 899. A pesar de lo indicado por lo anteriormente mencionado, este dispositivo no soporta una frecuencia de muestreo de 64 KHz ya que según el fabricante del sensor TDK InvenSense solo soporta como frecuencia máxima de muestreo los 8KHz.

Por otro lado, este valor también debe estar presente en la clase "sensors_periph_interface.h", ya que esta se encarga de definir los valores esperados de la aplicación, por lo que se utilizará el identificador de ACC_RATE_1KHZ para introducir el valor de frecuencia de muestreo que se desea para el sensor como se puede ver en la Ilustración 33.

```
68 // Sensors RATES ================= Expected values from application
69 #define ACC_RATE_25HZ      0x0A
70 #define ACC_RATE_50HZ      0x09
71 #define ACC_RATE_100HZ     0x08
72 #define ACC_RATE_200HZ     0x07
73 #define ACC_RATE_1KHZ      0x03
74
75 #define GYRO_RATE_25HZ     0x0A
76 #define GYRO_RATE_50HZ     0x09
77 #define GYRO_RATE_100HZ    0x08
78 #define GYRO_RATE_200HZ    0x07
79 #define GYRO_RATE_1KHZ     0x03
```

Ilustración 33: Frecuencias esperadas por la aplicación

En un principio estás variables se definen a 0x03, que corresponde con la frecuencia de muestreo máxima según las especificaciones del sensor, sin embargo, en el apartado **6.5.- Pruebas de diferentes configuraciones de frecuencia de muestreo para el acelerómetro**, se realiza un análisis a las diferentes configuraciones y se detalla el proceso de pruebas realizado una vez desarrollado el módulo de recepción de datos del Gateway.

6.5. DESARROLLO DEL MÓDULO DE RECEPCIÓN Y TRATAMIENTO DE DATOS EN EL GATEWAY (VERSIÓN 1)

Una vez desarrollado el módulo básico de captura de datos como se describe en el apartado **6.2.-Configuración inicial del Gateway**, se procede a mejorar la implementación realizada para tratar los datos recibidos. En una primera instancia el tratamiento será la realización de la FFT sobre los datos capturados por el acelerómetro y su posterior almacenamiento en archivos csv para su análisis. Ya realizados estos cambios, se hacen las pruebas detalladas en el apartado **6.5. Pruebas de diferentes configuraciones de frecuencia de muestreo para el acelerómetro**. Posteriormente se realizarán análisis más complejos sobre los datos recabados y ciertos filtros de la información recabada de dichos análisis. A continuación, se describe el proceso del desarrollo de ambas versiones.

En primer lugar, se realizan modificaciones al código al método post() de la clase main.py, de forma que se pueda diferenciar de una forma más clara la información enviada por cada sensor. Para este cambio y con vistas a ahorrar al máximo posible el código que no tendrá una utilización en la solución final, solo se tratarán los mensajes provenientes de las lecturas de humedad, temperatura y aceleración. Para ello primero debemos observar la composición del método one_sensor_data() que se muestra en la Ilustración 34. En ella se puede observar que cada uno de los sensores tratados posee un identificador que se corresponde con el rango [0,9] excepto aquellos con el id 4 y 7, por lo que basta con comprobar cuales son aquellos correspondientes a los sensores antes comentados. En este caso, el sensor de temperatura posee el identificador 1 y el de humedad el 2. Cabe destacar la inexistencia en esta versión base del tratamiento de los datos del acelerómetro, por lo que será añadida posteriormente.

```
29  def one_sensor_data(input_data,rep_length,sensor_data_length,msg_type):
30      #rep_length to mikos toy report_data
31      sensor_data=input_data[rep_length:rep_length+sensor_data_length]
32      if msg_type==1:
33          #temp
34          data=sensor_data[6:]+sensor_data[4:6]+sensor_data[2:4]+sensor_data[:2]
35          data=int(data,16)
36          data/=100.0
37          return "1"+str(data)
38      if msg_type==2:
39          #himidity
40          data=sensor_data[6:]+sensor_data[4:6]+sensor_data[2:4]+sensor_data[:2]
41          data=int(data,16)
42          data/=1024.0
43          return "2"+str(data)
44      if msg_type==3:
45          #pressure
46          data=sensor_data[6:]+sensor_data[4:6]+sensor_data[2:4]+sensor_data[:2]
47          data=int(data,16)
48          return "3"+str(data)
49      if msg_type==5:
50          #light
51          data=sensor_data[6:]+sensor_data[4:6]+sensor_data[2:4]+sensor_data[:2]
52          data=int(data,16)
53          data/=4.0
54          return "5"+str(data)
55      if msg_type==6:
56          #proximity
57          data=sensor_data[6:]+sensor_data[4:6]+sensor_data[2:4]+sensor_data[:2]
58          data=int(data,16)
59          if data==0:
60              data="false"
61          else:
62              data="true"
63          return "6"+data
64      if msg_type==8:
65          #fusion
66          qw=-fusion_process(sensor_data[:4])
67          qx=-fusion_process(sensor_data[4:8])
68          qy=-fusion_process(sensor_data[8:12])
69          qz=fusion_process(sensor_data[12:])
```

Ilustración 34: Composición método one_sensor_data

Una vez identificados esos se añaden las líneas de código que discriminan según estos en el método post() como se puede apreciar en la Ilustración 35, y se obtiene al ejecutar el código lo mostrado en la Ilustración 36.

```
112  def post(sensors_choosed):
113      message= get_data_msg(sensors_choosed)
114      if message !=None:
115          try:
116              #AQUI TRATAMOS LO DATOS LEIDOS
117              if(message[0]=="1"):
118                  print("Temperatura = "+message[1])
119              if (message[0] == "2"):
120                  print("Humedad = " + message[1])
121              if (message[0] == "8"):
122                  print("Fusion = " + message[1])
123
124          except:
125              pass
126      else:
127          print("No data")
128          pass
```

Ilustración 35: Discriminación por sensor en la función post

```
Fusion = 0.005 -0.007 0.014 -1.0
No data
Fusion = 0.005 -0.007 0.015 -1.0
Humedad = 68.0166015625
Temperatura = 21.0
Fusion = 0.005 -0.007 0.016 -1.0
```

Ilustración 36: Resultados discriminación datos en post

Estos casos como se puede observar funcionan para los sensores de temperatura y humedad, sin embargo, existen algunos sensores o funcionalidades que sobran y alguno que falta. En cuanto a las funcionalidades sobrantes, se encuentra el sensor fusión. Este es una funcionalidad del firmware del dispositivo que permite extraer información más compleja de los datos obtenidos por el acelerómetro, giroscopio y magnetómetro, como puede ser la orientación absoluta del dispositivo con respecto al marco de referencia de la Tierra [10]. Dicha componente es desactivada en el firmware desarrollado del dispositivo en el apartado **6.3.- Desarrollo firmware del multi sensor** al quitar ciertos sensores excedentes como se comenta en dicho apartado. Posteriormente se elimina de la función process_data(), todos los casos de procesamiento de datos que no correspondan con el sensor de humedad y temperatura, quedando el código como muestra la Ilustración 37.

```
80  def process_data():
81      raw_dat=raw_data_queue.get()
82      raw_data_queue.task_done()
83      raw_dat=raw_dat[4:] # preamble and timestamp OUT
84      print("raw_dat =" + str(raw_dat))
85      if raw_dat.find("040203")==0:
86          data_queue.put(one_sensor_data(raw_dat[14:],6,8,2)) #humidity
87      if raw_dat.find("060203")==0:
88          data_queue.put(one_sensor_data(raw_dat,6,8,1))#temp
```

Ilustración 37: Process_data() del sensor de temperatura y humedad

Este proceso anteriormente mencionado se debe realizar de la misma forma en la función one_sensor_data. Para ello, solo se quedan como validos los tipos 1 y 2 en el switch de control, resultando tras la ejecución los mensajes que se muestran en la Ilustración 38.

```
Humedad = 59.232421875
Temperatura = 18.98
Humedad = 59.205078125
Temperatura = 18.98
Humedad = 59.1416015625
Temperatura = 18.99
Humedad = 59.09375
Temperatura = 18.99
Humedad = 59.0390625
Temperatura = 19.0
Humedad = 59.0009765625
Temperatura = 19.0
Humedad = 58.9736328125
Temperatura = 19.01
```

Ilustración 38: Depurado después de modificaciones en one_sensor_data

Una vez reducido el código a solo los elementos necesarios, se debe encontrar la forma de recibir los datos del sensor que falta para el prototipo final. Para ello primero se identifica el mensaje en crudo que se recibe por el dispositivo multi sensor, de forma que se pueda discriminar según cada caso.

En una primera capa los datos recibidos siguen las componentes detalladas en la Tabla 2, donde el primer componente de dicho string es el valor 0xA5 que sirve de preámbulo. El siguiente valor que se recibe es un timestamp, que se incrementa con cada reporte enviado por el multi sensor con tamaño de 1 byte, y, por último, se recibe el reporte de cada uno de los sensores. Estos pueden variar en tamaño hasta unos 107 bytes.

Preámbulo (1 Byte)	Timestamp (1 Byte)	Reporte de sensores (Hasta 107 Bytes)
Siempre 0xA5	Int que incrementa después de cada reporte	Reporte de los sensores concatenado

Tabla 2: Reporte multi sensor

En cuanto al reporte de sensores, suele estar estructurado como se muestra en la Tabla 3. El primero de los valores corresponde como bien indica su nombre al Id que posee cada uno de los sensores, estos son utilizados en el caso de nuestra clase main.py, en la función process_data(). Los siguientes valores del string recibido son estados del sensor, de calibración y datos de este, todos ellos dependen del tipo de sensor que envía los datos, por lo que en la Tabla 4 se describen los posibles valores de interés para los sensores de temperatura, humedad y acelerómetro.

ID reporte (1Byte)	Estado sensor (1 Byte)	Estado calibración (1 Byte)	Datos sensor (N Bytes)
1 al 24	Valor dependiente del tipo de sensor	Valor dependiente del tipo de sensor	Valor dependiente del tipo de sensor

Tabla 3: Reporte de sensor

Sensor	ID	Estado del sensor	Estado de calibración	Datos del sensor
acelerómetro	1	-0: In_data_valid (pre-calibración) -1: Out_data_valid (post-calibración)	0: Calibración desactivada. 3: Calibración OK	Val_x (2 Bytes) Val_y (2 Bytes) Val_z (2 Bytes)

		-2: Cal_enabled -3: Cal_settled -4: Cal_converged -5:7: Calibrando sensor		
Humedad	5	Siempre 2 (Sensor listo)	Siempre 3	Val32 (4 Bytes)
Temperatura	6	Siempre 2 (Sensor listo)	Siempre 3	Val32 (4 Bytes)

Tabla 4: Valores de interés del reporte de sensores

Una vez identificados los diferentes valores que interesan para la recepción de los datos por parte del Gateway, se continúa desarrollando el código para tratar los diferentes casos.

En primer lugar, se añaden en la función process_data() los condicionales que permitan identificar la cadena enviada por el acelerómetro cuando está en fase de calibración y cuando envía datos validos de output. Para ello se añaden las líneas de código que se muestra en la Ilustración 39.

```
69    if raw_dat.find("01E700")==0:
70      data_queue.put(one_sensor_data(raw_dat,6,12,3))#acel calibrando
71    if raw_dat.find("010300")==0 or raw_dat.find("010700")==0:
72      data_queue.put(one_sensor_data(raw_dat,6,12,4))#acel data
```

Ilustración 39: Discriminando datos recibidos desde el acelerómetro

Estos valores cambian solamente en el tercer y cuarto byte como se explicó anteriormente. En el caso del tercer byte solo interesa que los últimos tres bits del byte sean uno, por lo que

resultaría en el número 0xE7, que indica que el acelerómetro está enviando datos de calibración. Por otro lado, cuando no se encuentre el acelerómetro en calibración, puede enviar datos con el 0x03 indicando que no se encuentra activado o 0x07 en caso contrario.

La segunda función a cumplimentar para el correcto funcionamiento es one_sensor_data(). En dicha función se debe agregar nuevamente el caso de estos dos datos recibidos. Para ello como se puede apreciar en la Ilustración 39, se envía como ultimo parámetro de la función un identificador que simboliza el sensor del cual proviene el mensaje. Al ser ambos provenientes del acelerómetro no serán notables las diferencias más allá del preámbulo que se utilizó previamente para distinguir el mensaje, por lo que solo se destaca en la Ilustración 40 la definición de uno de los casos.

```
41  if msg_type==3:
42      #acel calibrando
43      qx = fusion_process(sensor_data[:4])
44      qy = fusion_process(sensor_data[4:8])
45      qz = fusion_process(sensor_data[8:12])
46
47
48      return "3"+str(qx)+" "+str(qy)+" " +str(qz)+" "
```

Ilustración 40: one_sensor_data recepción de datos acelerómetro

Cabe destacar que para tratar los datos correctamente se deben adecuar primero, para lo cual se dividen estos en 2 bytes por cada componente del eje de coordenadas, para luego enviarlos a la función fusion_process() ya encontrada en el proyecto base. Esta se encarga de cambiar la posición de los datos de Big endian a Little endian y mediante una función auxiliar, calcula el complemento a 2 de estos números con signo. Estos valores calculados son posteriormente

enviados nuevamente hasta la función post() encargada en esta etapa de mostrar los datos por pantalla.

Una vez conseguidos los mensajes de depuración y el acceso a los datos correctamente, se busca aumentar la funcionalidad del proyecto, almacenando los datos que se capturan y calculando la FFT de los datos recibidos desde el acelerómetro. Para ello se comienza descargando en el Gateway la librería scipy, la cual posee el módulo FFT que se utilizará en el prototipo. Luego se importan en el proyecto de forma que puedan ser utilizables en el código de la clase main.py

En principio se considera la opción de incluir la librería de numpy para el manejo y almacenamiento de datos recopilados. Sin embargo, a pesar de la posibilidad de albergar más datos que en un array en Python con numpy, cuando se añade un elemento a un array de estos se realiza una copia del array ya existente con todos sus datos y se almacena en con el dato agregado. Este proceso puede producir una carga de trabajo innecesaria al dispositivo Gateway, ya que en la versión final del prototipo se procurará realizar una lectura cada 2 minutos de unos 20 segundos de duración, por lo que el array disponible en Python será suficiente sin añadir la carga de trabajo antes mencionada. Por lo tanto, primero se crean cinco arrays, tres de ellos orientados a los datos capturados por cada eje del acelerómetro, uno para la temperatura y el ultimo de humedad. Estos arrays vacíos al inicio de la ejecución del código, son alimentados con datos en la función one_sensor_data() dentro de su correspondiente caso. A modo de ejemplo se muestra en la Ilustración 41 el código resultado de la modificación del acelerómetro.

```
58  if msg_type==4:
59      #acel
60      qx = fusion_process(sensor_data[:4])
61      qy = fusion_process(sensor_data[4:8])
62      qz = fusion_process(sensor_data[8:12])
63
64
65      data_x.append(qx)
66      data_y.append(qy)
67      data_z.append(qz)
68
69      return "4"+str(qx)+" "+str(qy)+" " +str(qz)+" "
```

Ilustración 41: Guardado de datos del acelerómetro en arrays

Posteriormente se añade la función doFFT(), que será llamada para esta primera aproximación una vez se desconecte el equipo multi sensor. Dicho método se encuentra definido como se muestra en la Ilustración 42. Esté se encarga de realizar la FFT en cada uno de los ejes del acelerómetro y posteriormente guarda tanto el resultado de cada una de estas componentes como sus valores originales.

```
210     def doFFT():
211         fft_x = fft(data_x)
212         np.savetxt("data_x.csv",data_x, delimiter=" ")
213         np.savetxt("fft_x.csv",fft_x,delimiter=" ")
214         fft_y = fft(data_y)
215         np.savetxt("data_y.csv", data_y, delimiter=" ")
216         np.savetxt("fft_y.csv", fft_y, delimiter=" ")
217         fft_z = fft(data_z)
218         np.savetxt("fft_z.csv", fft_z, delimiter=" ")
219         np.savetxt("data_z.csv", data_z, delimiter=" ")
```

Ilustración 42: doFFT()

6.6. PRUEBAS DE DIFERENTES CONFIGURACIONES DE FRECUENCIA DE MUESTREO PARA EL ACELEROMETRO

Una vez desarrollado el módulo de recepción y tratamiento de datos en el dispositivo Gateway como se describe en el apartado **6.4.- Desarrollo módulo de recepción y tratamiento de datos en Gateway, primera aproximación**, se proceden a realizar pruebas con las diferentes configuraciones posibles. Para dichas pruebas se utilizan las configuraciones apropiadas para 100, 200, 500, 1000, 2000, 4000 y 8000 Hz.

Cada una de estas pruebas se realiza sobre un disco duro externo Western Digital Elements SE con una frecuencia de giro de 90 Hz que emitirá vibraciones capturadas posteriormente por el multi sensor. Serán tomadas 1024 muestras continuas en el dominio del tiempo en cada una de las pruebas y se guardan las aceleraciones en cada uno de los ejes obtenidas en un .csv, al

igual que los valores extraídos de la FFT de cada uno de estos. Los datos respecto a las aceleraciones capturadas de cada eje permiten asegurar la captura correcta de las aceleraciones al ser representados gráficamente, siendo un caso de captura correcta el mostrado en la Ilustración 43, donde no se observan datos anómalos producto de algún fallo de lectura o movimiento involuntario de la superficie donde se encuentra el disco duro externo y equipo multi sensor.

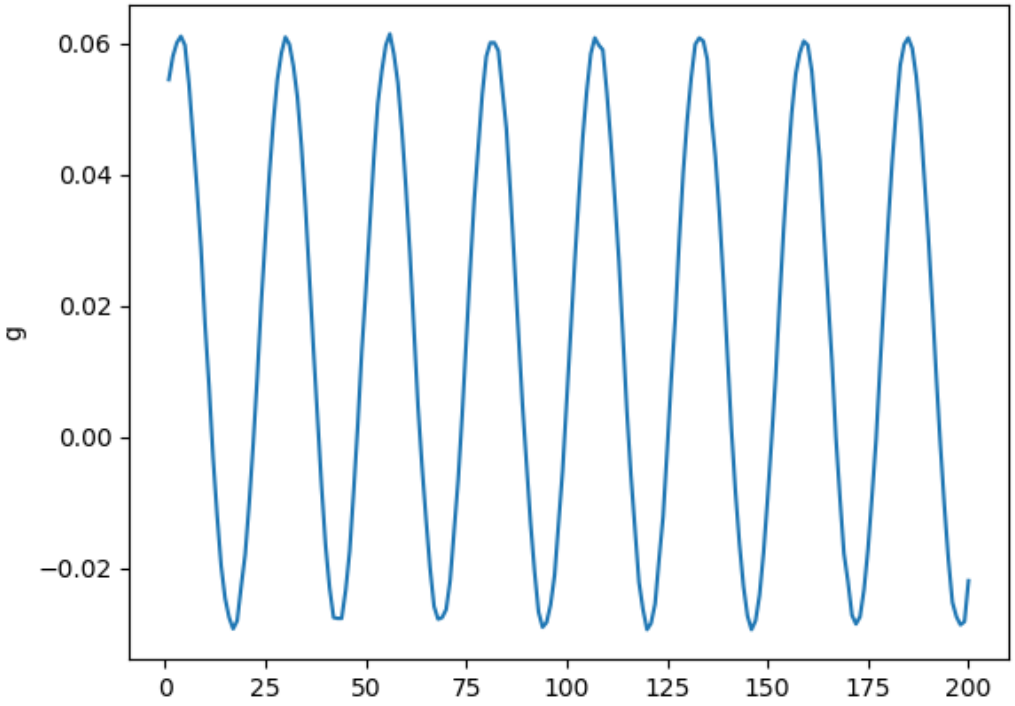

Ilustración 43: Ejemplo de captura de datos

Estas pruebas de configuración son necesarias por dos razones. En primer lugar, existen ciertas discrepancias entre las frecuencias de muestreo máximas permitidas en la documentación técnica del sensor ICM42605 y del dispositivo DA14585. Por otro lado, durante la realización de estas se encontraron diferencias entre las frecuencias de muestreo configuradas según el fabricante y las frecuencias de muestro resultantes del análisis de los datos, es decir, discrepancias entre la frecuencia de muestreo teórica y la frecuencia de muestreo real.

Para entender las pruebas realizadas, se deben tener en cuenta lo siguiente:

- Frecuencia teórica: Se le llama frecuencia teórica a la frecuencia de muestreo determinada por el fabricante del módulo multi sensor, es decir, es la frecuencia de muestreo a la que captura el dispositivo según el valor que se introduce en el firmware de este.
- Frecuencia real: Se le llama frecuencia real a la frecuencia de muestreo obtenida a partir del análisis de los datos resultantes de la FFT.
- Frecuencia de Nyquist: Se le llama frecuencia de Nyquist a la máxima frecuencia que se puede reconstruir a partir del muestreo que se realiza. Está responde al Teorema de Nyquist y corresponde con la mitad de la frecuencia de muestreo. El Teorema de Nyquist demuestra que la frecuencia de muestreo debe ser 2 veces más alta que la frecuencia más alta medida. Este hecho se debe a la perdida de información que se produce al intentar reconstruir una señal analógica a partir de la cantidad de muestras tomadas, es decir, la frecuencia de muestreo que se posee.

Una vez entendidos los conceptos anteriormente descritos, se realizan las pruebas a las diferentes configuraciones posibles de forma que se pueda determinar la mayor frecuencia obtenible al margen de lo expuesto por los pliegues técnicos de los dispositivos y, por otro lado, la frecuencia real obtenida con cada una de las configuraciones. Para lo cual se realizan los siguientes pasos:

1. Se configura la frecuencia teórica en el nodo multi sensor. La forma en la que se realizan las diferentes configuraciones se trata en el apartado **6.3.-Desarrollo firmware del multi sensor**. Cada una de estas corresponde con una frecuencia de muestreo determinada por el fabricante.
2. Se calcula la frecuencia de Nyquist. Está como se comentó anteriormente, corresponde con la mitad de la frecuencia de muestreo que se configura en el paso anterior.
3. Se calcula la FFT de los valores recibidos del acelerómetro.
4. Se calcula el valor absoluto de cada valor resultado de la FFT de forma que se obtenga un valor real que corresponde con la magnitud de cada frecuencia. Del array resultante de dicha operación, solo se tomarán la mitad de los valores, ya que son aquellos que corresponden con la parte positiva de la señal capturada.
5. Se gráfica las magnitudes calculadas anteriormente con respecto al rango de frecuencias determinado por la frecuencia de Nyquist, de forma que se pueda observar los armónicos que encontramos en el dispositivo. En dicha grafica se debe observar el primer armónico en la frecuencia de 90 Hz que corresponde con la frecuencia del disco duro. En este punto es donde se determinan las inconsistencias entre la frecuencia de muestreo determinado por el fabricante y la frecuencia de muestreo realmente obtenida. A modo de ejemplo se observa en la Ilustración 44, como para el caso de 1 Khz el armónico no se encuentra en los 90 Hz antes mencionados, lo que quiere decir que la frecuencia de muestreo no se encuentra realmente en 1 Khz.

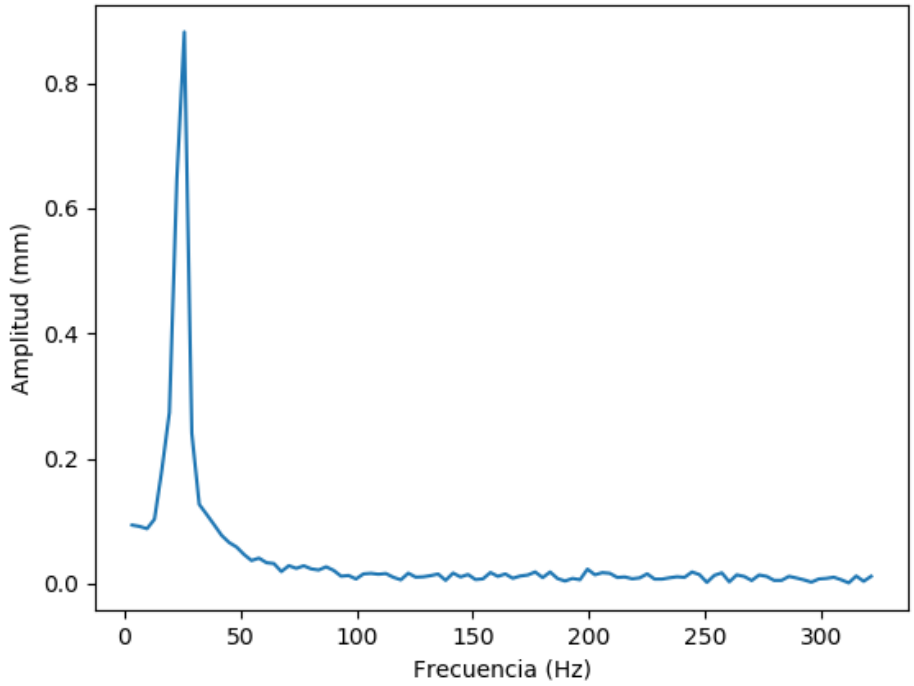

Ilustración 44: Ejemplo de gráfica FFT con frecuencia de muestreo incorrecta

6. Se determina la frecuencia del armónico según la gráfica, siendo este el pico que observamos en la Ilustración 44.
7. Se utiliza una regla de tres de forma que se pueda determinar la frecuencia de Nyquist o que se reconstruye real de forma que posteriormente se pueda conseguir la frecuencia de muestreo real del dispositivo. Para ello se utiliza la siguiente formula:

$$\frac{Frecuencia\ disco\ *\ Frecuencia\ Nyquist}{Frecuencia\ armónico}$$

8. El resultado de la anterior ecuación resulta en la frecuencia de Nyquist real a la que se obtienen los datos, por lo que al ser multiplicada por 2, obtendremos la frecuencia de muestreo del dispositivo.
9. Con dicho resultado se vuelve a realizar el proceso anterior, modificando la frecuencia de Nyquist por la calculada en el paso anterior y se obtiene la Ilustración 45, en la cual se puede observar que si se presenta el armónico en los 90 Hz del disco duro.

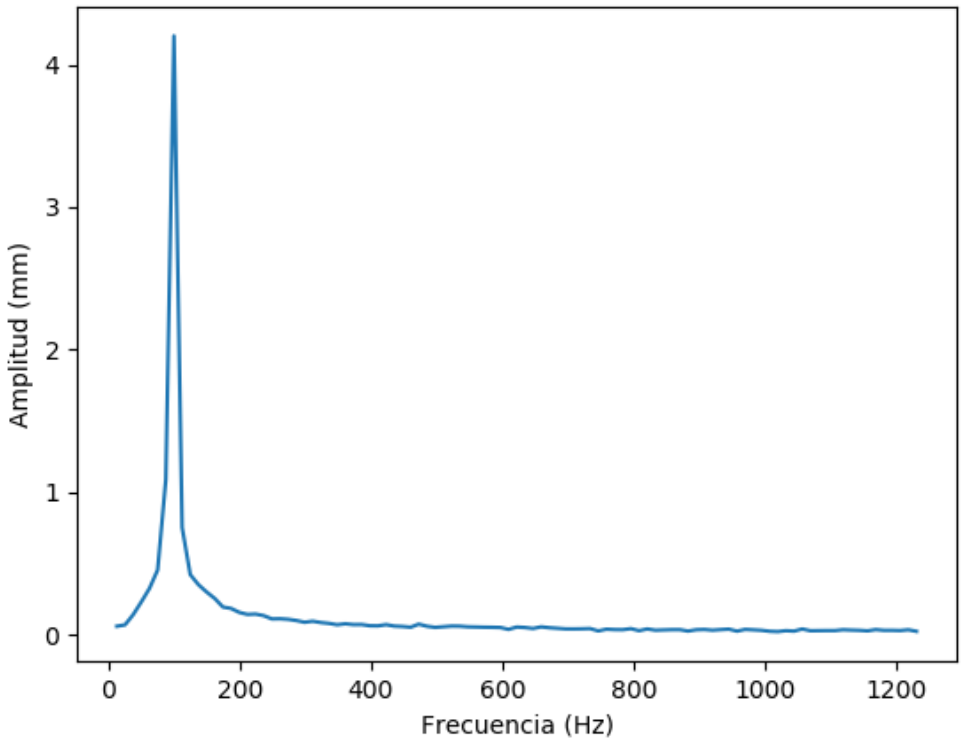

Ilustración 45: Ejemplo de gráfica FFT con frecuencia de muestreo correcta

El proceso anteriormente mencionado se realizó para las cinco muestras tomadas de cada una de las configuraciones del dispositivo multi sensor, donde se obtuvieron los resultados que se observan en la Tabla 5 entre cada una de las configuraciones. Cabe destacar que los resultados entre muestras de una misma configuración no presentaron variaciones importantes por lo que se toma la media de los valores arrojados.

Frecuencia teórica (Hz)	Frecuencia capturada (Hz)
100	300
200	1800
500	2250
1000	3000
2000	803
4000	242
8000	120

Tabla 5: Resultados pruebas de frecuencias del acelerómetro

Según los resultados vistos en la Tabla 5, la configuración que se corresponde con la captura de datos a 1000 Hz es aquella que captura a mayor frecuencia con 3000 Hz.

Cabe destacar que la incongruencia entre la frecuencia máxima permitida entre el sensor ICM42605 y el dispositivo DA14585 no provienen de la velocidad de la interfaz SPI que conecta al sensor con el controlador, ya que esta soporta velocidades de hasta 24MHz.

6.7. CONFIGURACIÓN DE THINGSBOARD

En el caso de la plataforma Thingsboard como ya se comentó anteriormente existe una opción de Comunidad que será utilizada en el desarrollo del prototipo. En esta se podrá tener acceso ilimitado a las actualizaciones de software, dispositivos registrados sin límite, con la única salvedad de que la plataforma donde se guardarán todos los datos debe ser un equipo propio.

En primer lugar, se debe realizar la instalación de Java 8 de OpenJDK. Dicha instalación se realizará en una máquina virtual ubicada en la Universidad de Oviedo. También cabe destacar que durante la instalación es necesario agregar el Java al Path y JAVA_HOME como variable de entorno que se presenta en uno de los apartados de la instalación, como se muestra en la Ilustración 46.

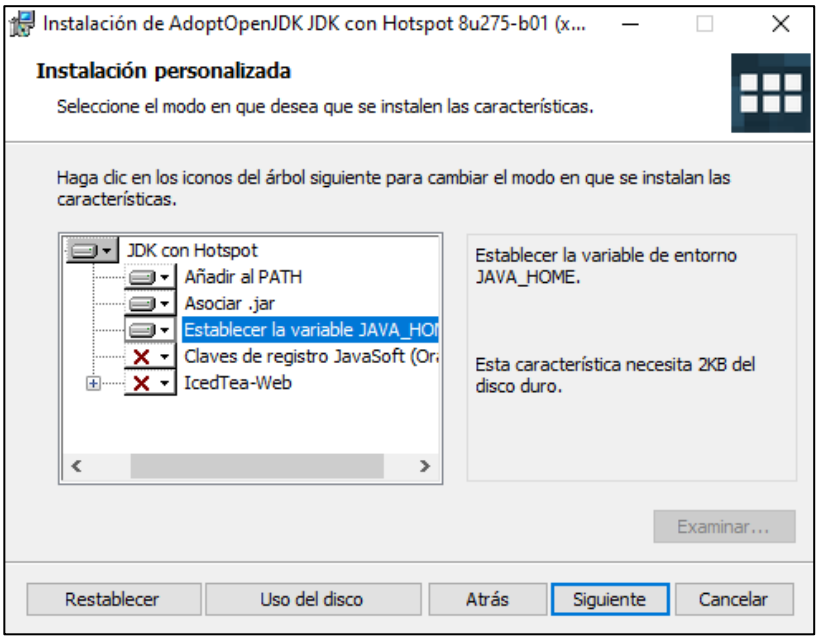

Ilustración 46: Instalación Java8 OpenJDK

Una vez instalado se comprueba que este instalado correctamente con el comando java -version, con lo que se debería obtener el resultado que se presenta en la Ilustración 47.

```
C:\Users\jesus>java -version
openjdk version "1.8.0_275"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_275-b01)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.275-b01, mixed mode)
```

Ilustración 47: Comprobar versión de java instalada

Por otro lado, la plataforma permite la integración con PostgreSQL, PostgreSQL+Cassandra o PostgreSQL+TimescaleDB para el almacenamiento de datos. Para el desarrollo del prototipo se utilizará PostgreSQL como base de datos por ser ya conocida, además de recomendada por los creadores de la plataforma en el caso de enviar menos de 5000 mensajes por segundo.

Primero se realiza la instalación de PostgreSQL, donde se recomienda la versión 11.7. Una vez descargada e instalada la base de datos se ejecuta el archivo pgAdmin para continuar crear la base de datos que será utilizada por ThingsBoard. En cuanto se ejecute el programa anteriormente descrito y autenticado el super usuario, se selecciona en el menú lateral sobre "Databases", y posteriormente las opciones "create" y "database…" como se aprecia en la Ilustración 48.

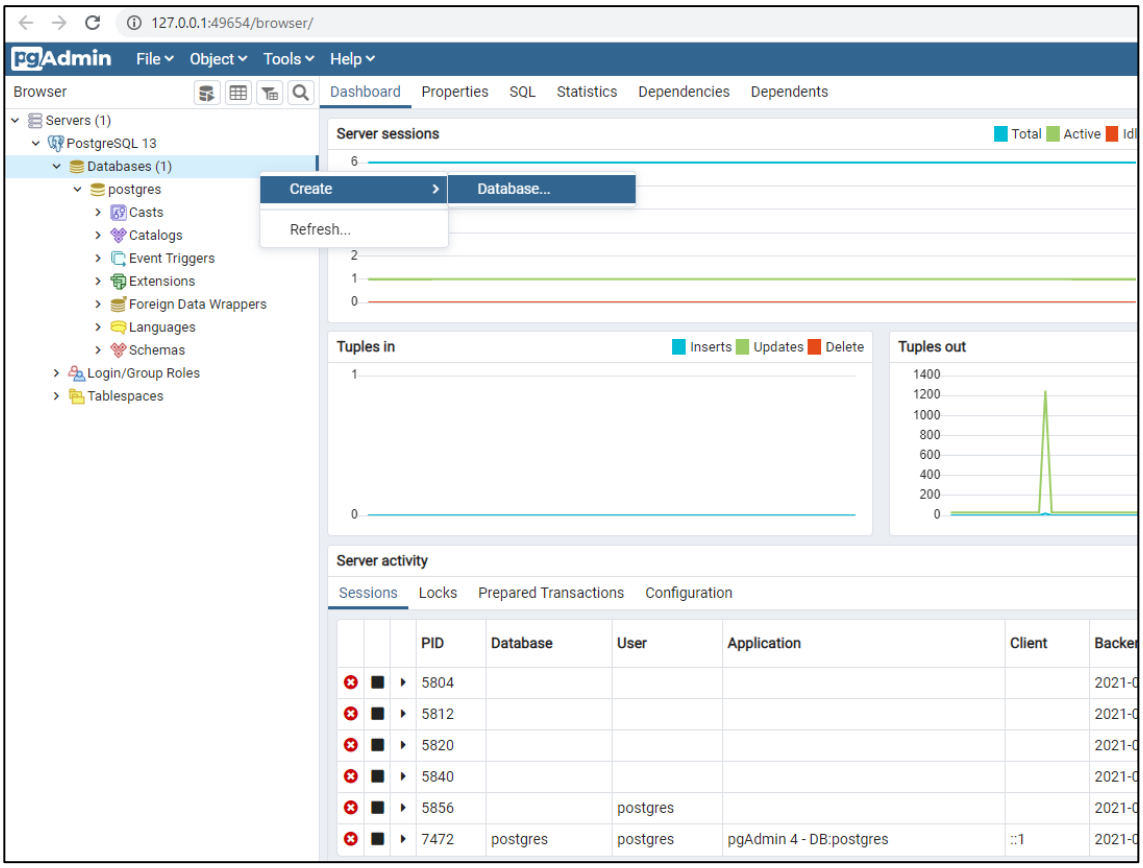

Ilustración 48: Creando base de datos Thingsboard

Estas acciones mostrarán como resultado un dialogo donde se solicita información sobre la base de datos a crear. Para la mayor claridad posible se llama a dicha base de datos “thingsboard” con dueño “postgres” que es el super usuario en este caso.

Una vez realizados estos pasos previos a la instalación ya detallados, se procede a descargar e instalar la plataforma Thingsboard. En la ruta de descarga después descomprimir el archivo, en la carpeta conf se encontrará un archivo llamado thingsboard.yml donde será necesario realizar ciertos cambios antes de proceder a la instalación de la plataforma con su archivo .bat correspondiente. En la línea 455 del archivo thingsboard.yml anteriormente mencionado, se

debe cambiar el valor “postgres” mostrado en la Ilustración 49, por la contraseña del super usuario de la base de datos de PostgreSQL.

```
# SQL DAO Configuration
spring:
  data:
    jpa:
      repositories:
        enabled: "true"
  jpa:
    open-in-view: "false"
    hibernate:
      ddl-auto: "none"
    database-platform: "${SPRING_JPA_DATABASE_PLATFORM:org.hibernate.dialect.PostgreSQLDialect}"
  datasource:
    driverClassName: "${SPRING_DRIVER_CLASS_NAME:org.postgresql.Driver}"
    url: "${SPRING_DATASOURCE_URL:jdbc:postgresql://localhost:5432/thingsboard}"
    username: "${SPRING_DATASOURCE_USERNAME:postgres}"
    password: "${SPRING_DATASOURCE_PASSWORD:postgres}"
    hikari:
      maximumPoolSize: "${SPRING_DATASOURCE_MAXIMUM_POOL_SIZE:16}"

# Audit log parameters
```

Ilustración 49: Contraseña del super usuario PostgreSQL

Con lo anteriormente descrito finaliza la configuración de la base de datos en lo que a Thingboard se refiere, sin embargo, existe otro aspecto a configurar antes de realizar la instalación propiamente dicha de la plataforma. Este elemento es el bróker MQTT de mensajes que se utilizará para conectar dispositivos externos, en este caso el Gateway, con Thingsboard.

Los brókers de mensajes que se consideraron en una primera aproximación fueron Kafka y RabbitMQ, sin embargo, ambas opciones fueron descartadas después de su instalación y configuración, ya que la versión comunidad de Thingsboard no permite la utilización de un plugin necesario para realizar la conexión entre un bróker de mensajes externo y la plataforma, aunque permite su configuración en el fichero thingsboard.yml. Ante esta imposición por parte de la versión de la plataforma IoT, se utiliza el bróker de mensaje simple que posee está donde los elementos enviados por parte del Gateway serán almacenados temporalmente en la memoria

del dispositivo. Para realizar la configuración necesaria, basta con modificar la línea 603, donde se define el tipo de cola que se utiliza para el bróker de mensajes, por la cadena “in-memory” como se muestra en la Ilustración 50.

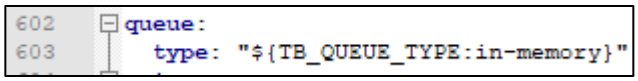

Ilustración 50: Configuración bróker de mensajes Thingsboard

Es importante destacar que por defecto el bróker MQTT que se instalará con la plataforma recibirá los parámetros de configuración definidos a partir de la línea 551. En lo que al proyecto concierne, no es necesario modificar ninguno de estos parámetros, sin embargo, se debe tener en cuenta que la plataforma escuchará en el puerto 1883 de la ip del host que la ejecuta.

Una vez terminadas las configuraciones de la plataforma, se procede a su instalación. Para ello se ejecuta el cmd y se accede al directorio donde se encuentran los archivos anteriormente modificados. Luego, se ejecuta el archivo install.bat y se debe observar lo mostrado por la Ilustración 51 al finalizar dicho instalador.

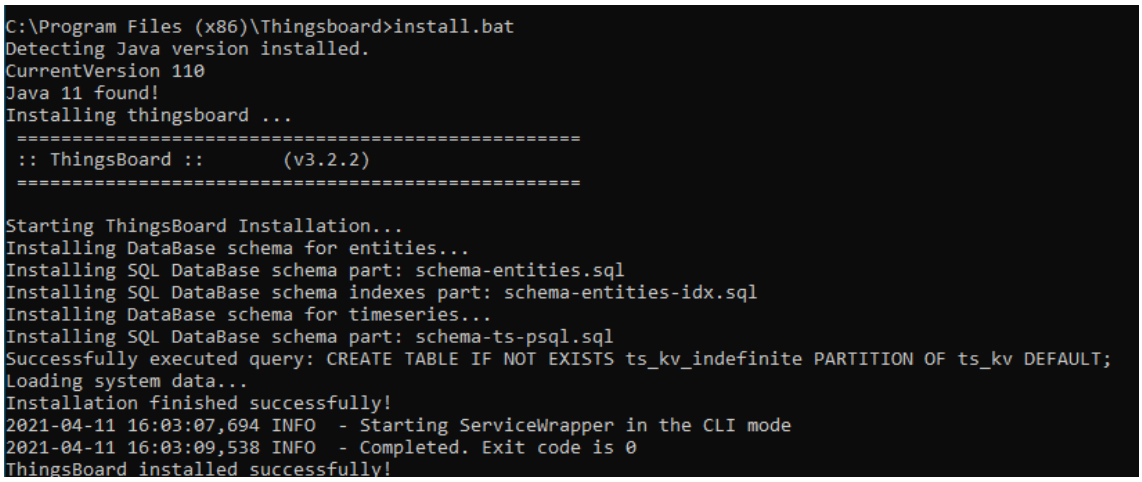

Ilustración 51: Resultado de la instalación de Thingsboard

Thingsboard ya se encuentra instalado, por lo que solo queda ejecutar ***net start thingsboard*** para iniciar el servicio y poder conectar con él a través del puerto 8080 a su interfaz web.

El siguiente paso será realizar las configuraciones pertinentes en Thingboard mediante la interfaz web de forma que los datos puedan ser almacenados en sus correspondientes entidades.

Como se estaba comentado, para realizar las configuraciones en la interfaz web primero, se crea un usuario “tenent” o inquilino que posee los permisos sobre paneles informativos, dispositivos que se conectan a la plataforma entre otros. Este usuario se crea accediendo al apartador “Propietarios”. Una vez creado, se accede a la plataforma con las credenciales definidas para dicho usuario donde se observa lo mostrado en la Ilustración 52.

Ilustración 52: Pagina de bienvenida Thingsboard

Antes de la creación de los activos a los cuales se les asociarán los datos, se debe crear el perfil de los diferentes dispositivos que se creen. En principio estos estarán categorizados en "termómetros" y "acelerómetros". A su vez estos perfiles nos permiten definir MQTT como protocolo de transporte.

El siguiente paso será crear un activo. Este correspondería al elemento que contiene a los sensores, es decir, el vehículo que posee los sensores. Para conseguir esto basta con acceder al apartado de "Activos" y crearlo. Lo mismo se realiza para la creación de dispositivos en su correspondiente apartado, donde se creará uno para el Gateway del vehículo y otro para cada uno de los sensores de cada motor. Es importante a la hora de crear los dispositivos, definir las relaciones que existen entre ellos, de forma que se actualice automáticamente la información que sea común o recibida por uno de ellos pero que pertenece a otro, como puede ser la relación entre cualquier sensor y el Gateway.

Por último, asignamos diferentes widgets en el apartado “Paneles” donde se crea un panel de pruebas. Una vez creado, se accede al panel y se crean un widget para cada uno de los sensores de forma que se puedan visualizar los datos que se reciben como se puede observar en la Ilustración 53.

Ilustración 53: Configuración widgets acelerómetro, temperatura y humedad

6.8. PRUEBAS ENVÍO DE DATOS MEDIANTE EL HAT DE COMUNICACIONES POR MQTT

Antes de desarrollar el módulo final de envió de datos a la plataforma Thingsboard, se realizan pruebas al HAT de comunicaciones de forma que se pueda determinar la forma más eficiente de enviar los datos a esta. Dichas pruebas se realizan en el PC mediante AT command Tester SM.

En primer lugar, se realiza el registro a la red mediante comandos AT para el envío de la información, para lo cual una vez establecida la conexión con el módulo como se comenta en apartados anteriores, se ejecuta el siguiente proceso para obtener una IP por parte del proveedor de la red:

1. Se verifica el correcto registro del dispositivo con el proveedor de la red, de forma que sea posible asignar una IP al Gateway. Esto se consigue ejecutando el comando "AT+CREG?" a lo cual se espera el resultado "0,5" que denota que el dispositivo se encuentra registrado y en modo roaming. En caso de que no se reciba este código, se debe esperar a que el dispositivo se registre.
2. Se adjunta el dispositivo a la red GPRS mediante el comando "AT+CGATT=1" a lo que se espera el resultado "OK", en caso contrario se debe verificar el paso numero 1 antes detallado.
3. Se activa el contexto del Packet Data Protocol (PDP), que viene siendo la dirección de capa de red del dispositivo de forma que se puedan enviar datos mediante el comando "AT+CGACT=1,13", a lo que se espera el resultado "OK". Al igual que en el caso anterior, de no recibir este resultado, se debe comprobar desde el punto 1.
4. Se activa la red de aplicaciones con el comando "AT+CNACT=1,13", de forma que se le asigne la misma dirección IP conseguida en el apartado anterior.

Al finalizar todos los pasos anteriormente mencionados, el módulo tendrá una dirección IP establecida mediante la cual se pueden realizar las comunicaciones pertinentes.

Estas comunicaciones se pueden realizar de dos formas diferentes. La primera de ellas es mediante comandos AT, debido a que el módulo dispone de funcionalidades integradas para la comunicación a través del protocolo MQTT. Por otro lado, al establecer una IP se pueden realizar comunicaciones de la misma forma como si la conexión existiese mediante una red WiFi. No obstante, es necesario realizar unas configuraciones previas para confirmar la correcta ejecución y dotar del protocolo DHCP a la conexión, ya que esta se realiza mediante una red WAN. Está segunda opción solo es válida para el Gateway, por lo que las pruebas de está serán realizadas en él.

Para conseguir el primero de los modelos de envío de datos anteriormente descritos, se deben realizar los siguientes pasos:

1. Se añaden los datos de configuración del cliente MQTT mediante una consecución de comandos “AT+SMCONF=valores”, donde valores será en la primera ejecución igual a ‘“URL”,”test.mosquitto.org”,”1883”’ y en su segunda ejecución igual a “‘KEEPTIME’,60”, estableciendo para el caso de prueba el servidor público de pruebas de Mosquitto MQTT, y un tiempo de vida del cliente de 60 segundos sin comunicación.
2. Se realiza la comunicación con el servidor anteriormente definido mediante “AT+SMCONN”.
3. Se envía el valor al bróker MQTT mediante el comando “AT+SMPUB=’TOPIC’,LEN,QOS,RETAIN”, donde “TOPIC” corresponde con el tópico al cual se desean enviar los mensajes, “LEN” es la longitud del mensaje a enviar, “QOS” el nivel de servicio del bróker que puede ser 0 o 1 y “RETAIN” si se desea que el servidor quede con el mensaje después de despachado a los suscriptores del tópico (0 o 1). Una vez se envía dicho comando se debe introducir el mensaje a enviar, seguido de un salto de línea y la combinación “CTRL+Z”.
4. En cuanto se envíen todos los datos necesarios se cierra la conexión mediante “AT+SMDISC”.

Por otro lado, utilizando un PC se realiza una conexión mediante el cliente ligero de Mosquitto, al tópico al cual se enviarán los datos como se observa en la Ilustración 54, de forma que se pueda comprobar él envió correcto de los mismos.

```
C:\Program Files\Mosquitto>mosquitto_sub -d -h test.mosquitto.org -p 1883 -t trabajo/tfg
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: trabajo/tfg, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) sending PINGREQ
Client (null) received PINGRESP
```

Ilustración 54: Conexión de subscriptor al tópico trabajo/tfg

Luego se ejecutan los pasos comentados anteriormente obteniendo como resultado en el suscriptor al tópico al cual se envían los mensajes, lo mostrado en la Ilustración 55.

```
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=209.997
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=214.887
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=209.997
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=216.897
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=201.182
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=198.987
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=197.951
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=202.231
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=196.155
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=190.987
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=192.098
Client (null) sending PINGREQ
Client (null) received PINGRESP
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=200.444
Client (null) received PUBLISH (d0, q0, r0, m0, 'trabajo/tfg', ... (19 bytes))
sensor_data=212.555
Client (null) sending PINGREQ
Client (null) received PINGRESP
```

Ilustración 55: Resultado de envío de datos mediante comandos AT

En cuanto a la segunda opción de conexión se deben realizar un conjunto de operaciones diferentes a las comentadas en la primera opción. Primero se debe realizar la instalación de las librerías qmicli y udhcpc a través del comando "sudo apt-get libqmi-utils udhcpc". La primera es la encargada de comunicarse con módems WWAN y dispositivos que poseen el protocolo QMI, mientras que la segunda es un pequeño cliente DHCP para sistemas embebidos.

Para conseguir comunicar el dispositivo mediante IP, se deben realizar las siguientes acciones:

1. Se desactiva la interfaz wwan0 para realizar la configuración necesaria a dicha interfaz mediante "ifconfig wwan0 down".
2. Se establece el valor "Y" en el archivo raw_ip de la interfaz wwan0 mediante el comando echo Y > /sys/class/net/wan0/qmi/raw_ip. Este establece el modo raw_ip de direccionamiento a la interfaz wwan0, siendo este necesario en el caso del HAT SIM7000E y no persistente si este es apagado, por lo que debe realizarse dicho proceso en cada encendido.
3. Se activa la interfaz mediante "ifconfig wwan0 up".
4. Se deben realizar los pasos detallados al comienzo de este apartado, dando de alta al módulo de forma que se le asigne una IP.
5. Se establece un dispositivo cdc-wm0 a través de qmicli con el comando "qmicli -d /dev/cdc-wdm0 --device-open-net="net-raw-ip|net-no-qos-header" --wds-start-network="apn='XXXX',ip-type=4" --client-no-release-cid", donde XXXX corresponde con el punto de acceso del proveedor de red contratado.

6. Se activa el protocolo DHCP para la interfaz wwan0 con el comando “udhcpc -i wwan0”, de forma que está pueda resolver nombres y direcciones IP para conseguir comunicarse con otros dispositivos.

De forma que se puedan realizar pruebas de forma más eficiente, se aglutinan los primeros tres pasos en un ejecutable sh llamado qmicli.sh y los últimos dos pasos en un segundo fichero llamado qmicli2p.sh.

Antes de ejecutar los pasos anteriormente señalados se comienza por verificar el estado inicial de la interfaz de red wwan0 (ver Ilustración 56). En esta se observa la presencia de una IP privada por lo que la comunicación no será posible.

```
wwan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 169.254.128.57  netmask 255.255.0.0  broadcast 169.254.255.255
        inet6 fe80::3277:6476:951d:6026  prefixlen 64  scopeid 0x20<link>
        ether f6:f9:21:fc:0b:5e  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 34  bytes 5823 (5.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ilustración 56: Estado inicial de la interfaz wwan0

Luego se ejecuta el primero de los ficheros qmicli.sh y se vuelve a observar la configuración de la interfaz wwan0 como se observa en la Ilustración 57. Posteriormente se ejecutan los comandos AT para la obtención de la dirección IP por parte del módulo como se puede ver en la Ilustración 58.

```
pi@raspberrypi:~ $ sudo ./qmicli.sh
pi@raspberrypi:~ $ ifconfig

wwan0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 37  bytes 6384 (6.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ilustración 57: Estado wwan0 post ejecución qmicli.sh

```
Welcome to minicom 2.7.1

OPCIONES: I18n
Compilado en Aug 13 2017, 15:25:34.
Port /dev/ttyUSB2, 03:06:00

Presione CTRL-A Z para obtener ayuda sobre teclas especiales

AT
OK
AT+CPIN=1234
OK

+CPIN: READY

SMS Ready

AT+CREG?
+CREG: 0,5

OK

AT+CGATT=1
OK
AT+CGACT=1,13
OK

OK
AT+CNACT=1,13
OK
```

Ilustración 58: Comandos AT para la obtención de la IP

Por último, se ejecuta el fichero qmicli2p.sh, obteniendo como resultado lo mostrado por la Ilustración 59. En esta se observa la obtención de la dirección IP 10.115.211.168.

```
pi@raspberrypi:~ $ sudo ./qmicli2p.sh
[/dev/cdc-wdm0] Network started
        Packet data handle: '2201645488'
[/dev/cdc-wdm0] Client ID not released:
        Service: 'wds'
            CID: '4'
udhcpc: started, v1.30.1
No resolv.conf for interface wwan0.udhcpc
udhcpc: sending discover
udhcpc: sending select for 10.115.211.168
udhcpc: lease of 10.115.211.168 obtained, lease time 7200
Too few arguments.
Too few arguments.
```

Ilustración 59: Resultado qmicli2p.sh

Una vez realizadas las pruebas mencionadas anteriormente para ambas opciones de comunicación, se observa un problema en el módulo de comunicación que no permite que ninguna de estas pueda ser viable para el prototipo. Se observa que, en ambos casos al realizar operaciones de mayor consumo como puede ser la conexión al servidor MQTT o un ping a un servidor en el caso de la comunicación por IP, el módulo se reinicia, perdiendo así conexión además de eliminar todas las configuraciones realizadas y forzar al usuario a realizarlas nuevamente. En el caso de la comunicación por IP se puede observar esta pérdida de comunicación en la Ilustración 60. La única configuración que permite el envío de datos constante mediante alguno de los métodos es la conexión al ordenador mediante la interfaz USB

del módulo a un extensor de USB con alimentación propia. Está opción es la utilizada para verificar el correcto funcionamiento del método uno de conexión anteriormente comentado.

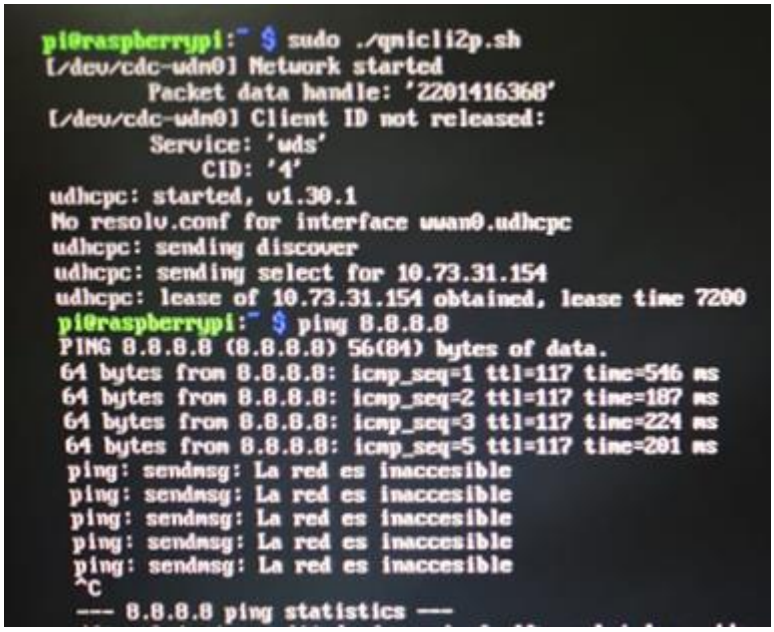

Ilustración 60: Perdida de conexión con SIM7000E

Las pruebas fueron realizadas en dos modelos de Raspberry Pi diferentes siendo una de ellas la detallada en el presente documento y otra la versión anterior Model 3B+. También se utilizaron tres tarjetas SIM de diferentes proveedores de red para descartar limitaciones en su uso por parte de estos, siendo una de ellas la detallada en el proyecto.

Ante esta situación se realiza una consulta al fabricante del módulo sobre el consumo del módulo y la posible necesidad de una alimentación particular para este, ya que este hecho no es mencionado en las especificaciones de este ni en sus manuales de utilización. Al no existir respuesta por parte de Waveshare se busca una respuesta a esta consulta en la comunidad de

desarrolladores, donde se encuentran múltiples reportes de problemas similares y la necesidad de soldar una batería de 5v al módulo de comunicación. Ante la falta de información por parte del fabricante y la falta de recursos para realizar las modificaciones necesarias se descarta la utilización del HAT SIM7000E para las comunicaciones, aunque se realizarán implementaciones de cara al futuro.

6.9. GENERACIÓN FINAL DEL MÓDULO DE ENVIÓ DE DATOS A THINGSBOARD EN EL GATEWAY

Una vez determinado el funcionamiento del HAT de comunicaciones y configurada e instalada correctamente la plataforma IoT de Thingsboard, se desarrolla el módulo en el Gateway encargado de realizar él envió de los datos recolectados a la plataforma. Para ello se debe desarrollar un cliente MQTT que realice dicho trabajo.

El primer paso es definir ciertas constantes en el proyecto que se necesitan para el correcto envió de los datos. Comenzamos con la constante TOPIC, que será el nombre del tópico de la cola MQTT definida en Thingsboard. En este caso será *"v1/devices/me/telemetry"*. Por otro lado, definimos la constante IP_BROKER que tendrá la ip del host donde se encuentra la plataforma ejecutándose. Los siguientes parámetros serán los id o Access tokens que provee Thingsboard al crear los dispositivos de forma que la plataforma consiga discriminar entre los diferentes orígenes y destinos de la información. Luego, se define un array llamado IDS que posee los diferentes ids definidos previamente de forma que se pueda iterar sobre estos, para reutilizar la mayor cantidad de código posible más adelante en el desarrollo. Por último, se define un diccionario que relaciona el id de cada uno de los dispositivos con el nombre del

fichero de datos de cada uno de estos. Todas las constantes anteriormente mencionadas y su definición pueden apreciarse en la Ilustración 61.

```
6   TOPIC = "v1/devices/me/telemetry"
7   IP_BROKER = 'localhost'
8   #Ids de los diferentes dispositivos registrados en thingsboard
9   ID_TEMP_MOTOR = 'Ln5fWCUieGUaWpBQC7v0'
10  ID_TEMP_BOMBAS = '4BO5WFMBqZu6UvLMaHsc'
11  ID_FFT_MOTOR = 'gohe9lt0lXNC9T3s9hB5'
12  ID_FFT_BOMBAS = 'wtmgDtLsQgzqGPMCyiU4'
13  #Array que posee los diferentes ids
14  IDS = [ID_TEMP_MOTOR, ID_FFT_MOTOR, ID_TEMP_BOMBAS,ID_FFT_BOMBAS]
15  #Diccionario que posee los nombres de los archivos según el id que tiene
16  JSONFILES = {ID_TEMP_MOTOR : 'tempMotor.txt',ID_FFT_MOTOR : 'acelMotor.txt', ID_TEMP_BOMBAS : 'tempBombas.txt', ID_FFT_BOMBAS : 'acelBombas.txt'}
17
```

Ilustración 61: Constantes del cliente MQTT

En cuanto al método main del programa, tenemos un bucle encargado de recorrer el array de ids. Cada uno de estos valores serán utilizados para llamar a la función connect_thingsboard(id). Está función se encarga de crear el cliente de paho.mqtt con parámetros por defecto. A continuación, se definen las funciones callback del cliente, que en este caso serán usadas solamente para depuración de cara a la correcta conexión al servidor de MQTT, su desconexión y notificaciones de mensajes publicados.

Una vez realizado lo anterior, se añaden las credenciales de autenticación del bróker al cliente de paho.mqtt, que será el id pasado por parámetro a la función. Posteriormente se utiliza el id del dispositivo nuevamente para sacar el nombre del fichero donde están almacenados los datos del dispositivo, y se pasa por parámetro a la función open_json(), la cual abre el archivo y devuelve los datos para ser utilizados en la función do_work(), como se muestra en la Ilustración 62.

```python
def connect_thingsboard(id):
    # el id_cliente puede ser la mac del sensor pasada como argumento al prog.
    # clean_sesion = false de forma que no se elimine toda la información del cliente cuando se desconecte. Así los mensajes persisten
    cliente = cl.Client()

    # Definimos la función callback a la que se debe llamar cuando se conecte el cliente con el broker
    cliente.on_connect = on_connect
    cliente.on_disconnect = on_disconnect
    cliente.on_publish = on_publish
    # cliente.on_subscribe = on_subscribe

    # Autenticación con el broker MQTT
    cliente.username_pw_set(id)

    #Buscamos el nombre de archivo según el diccionario definido al principio
    datos = open_json(JSONFILES[id])
    print("nombre del json  --------> " + str(JSONFILES[id]))
    #Terminamos de conectar con Thingsboard y enviamos los datos
    do_work(cliente,datos)


if __name__ == '__main__':

    for id in IDS:

        connect_thingsboard(id)
    print("Fin del script")
    exit(0)
```

Ilustración 62: Definición de main y connect_thingsboard del client MQTT

Por su parte la función do_work(), se encarga de terminar la conexión con el servidor MQTT al puerto 1883 para posteriormente comenzar el bucle del cliente con el método loop_start de la librería paho-mqtt, el cual implementa un bucle permanente en un hilo en el background, de forma que el principal pueda seguir ejecutando código.

Este hilo principal continúa accediendo al array de valores que se encuentran en los datos bajo “values” donde mediante un bucle se pasa por cada uno de los valores registrados. Con cada uno de estos valores se utiliza la función json.dumps() de la librería json que devuelve un objeto json a partir del array recibido, de forma que a continuación se puedan enviar mediante publish el valor del registro al servidor de MQTT.

Una vez finalizado el bucle se utiliza time.sleep(5) de manera que todos los mensajes enviados a la cola puedan terminar de ser recibidos por esta, para luego detener el hilo que se

está ejecutando en el background y el cual escucha las respuestas del servidor y, por último, se desconecta el cliente para continuar con la siguiente iteración del bucle main, donde se repetirán los pasos anteriormente mencionados con el siguiente dispositivo.

El código que responde a la definición de la función do_work(), se puede apreciar en la Ilustración 63.

```
#Metodo que se debe encargar de buscar los datos de un json y enviarlos a RabbitMQ
def do_work(cliente, datos):
    # Conexión con el broker MQTT
    rc = cliente.connect(host=IP_BROKER, port=1883, keepalive=60)
    print("res connect cliente = " + str(rc))

    cliente.loop_start()
    try:
        #Bucle encargado de recorrer todos los valores del json y enviarlos a Thingsboard
        for reg in datos["values"]:
            print("Valor del msg a enviar = "+str(reg))
            val = json.dumps(reg)
            cliente.publish(topic=TOPIC, payload=val, qos=1, retain=False)
    except KeyboardInterrupt:
        pass
    #Hacemos una cadena de sleep, para estar seguros de que todos los mensajes fueron enviados correctamente
    print("Esperando que se envien todos los datos")
    time.sleep(5)
    cliente.loop_stop()
    print("Parando el bucle de envios")
    #time.sleep(5)
    cliente.disconnect()
```

Ilustración 63: Función do_work() del cliente MQTT

Por último, se realizan pruebas para comprobar el correcto funcionamiento de los datos a la plataforma. Estos pueden ser observados en la Ilustración 53 del apartado **6.7.-Configuración de thingsboard**.

6.10. DESARROLLO DEL SCRIPT DE CONEXIÓN A WIFI EN EL GATEWAY

Para el script de conexión wifi del Gateway se deben tener varias circunstancias en cuenta. El dispositivo se debe conectar cada vez que consiga estar al alcance de una cierta wifi de forma que pueda enviar los datos a la plataforma Thingsboard. En la Raspberry Pi está situación puede no ser tan trivial como cuando encuentre la conexión, que se realice automáticamente, ya que en situaciones donde se pierde la comunicación, el Gateway puede no volver a reintentar la conexión si no se recupera inmediatamente, situación a la que se estará expuesto constantemente.

Por esto comentado anteriormente se desarrolla en un script que permita forzar esta conexión con la red cuando se encuentre disponible.

En este script, se comienza inicializando un contador a 0 encargado del reintento de las operaciones que se comentarán a continuación. Luego se utiliza el valor resultante del escaneo de las redes disponibles que contenga coincidencias con el nombre de la wifi mediante el comando sudo iwnlist wlan0 scan | grep $nombre_red y se ejecuta un sleep 2 esperando el resultado. Este proceso se realiza en un bucle en busca de un valor no nulo durante tres intentos. En caso de no conseguirlo se asume que la red no se encuentra disponible en el momento del escaneo y termina la ejecución del script, además de enviar un mensaje de error por el pipeline stderr. En el caso de que si se encuentre utilizamos el comando wpa_cli -i wlan 0 reconfigure de forma que la interfaz de red del Gateway se reconfigure forzando un nuevo intento de conexión con la red configurada con antelación en la configuración inicial del Gateway, también se realizan las comprobaciones necesarias en caso de error, de forma que se pueda terminar el programa correctamente.

Por último, se ejecuta un sleep 20 que esperará veinte segundos a que el comando anterior termine de ejecutarse y reiniciarse la interfaz de red para a continuación, determinar mediante una cadena de comandos si el dispositivo ya dispone de dirección ip en la interfaz wlan0, en caso de que no sea así se asume que existe un problema de conexión y se termina el script con código erróneo o en caso contrario se termina el script con código cero, de forma que pueda ser tratado por el programa principal de forma diferente según este. Todo lo anteriormente comentado, se puede observar en la Ilustración 64.

```
#!/bin/bash

nombre_red="Jesus"


i=0
res=sudo iwlist wlan0 scan | grep $nombre_red
sleep 2

while [ $? -ne 0 ]
do
 if [ $i -eq 3 ]
 then
  echo "error" >&2
  exit 1
 fi
 i=$((i+1))
 res=sudo iwlist wlan0 scan | grep $nombre_red
 sleep 2
done

#Reconfiguramos la interfaz de red para forzar la conexión a la wifi
wpa_cli -i wlan0 reconfigure
if [ $? -ne 0 ]
then
 echo "error" >&2
 exit 1
fi

#Esperamos 20 segundos para que reconfigure la red
sleep 20
ifconfig | grep -1 "wlan0" | grep "inet"
if [ $? -ne 0 ]
then
 echo "error" >&2
 exit 1
fi

exit 0
```

Ilustración 64: Código script conexión a WiFi

6.11. DESARROLLO DEL MÓDULO DE RECEPCIÓN Y TRATAMIENTO DE DATOS EN EL GATEWAY (VERSIÓN 2)

En este aparatado se describe la segunda versión del código desarrollado en el apartado **6.5.- Desarrollo del módulo de recepción y tratamiento de datos en el Gateway (Versión 1)**. Si bien en dicho apartado se desarrolló software capaz de recolectar y realizar tratamientos básicos a los datos capturados, esté no cumple con los objetivos pautados para el prototipo final.

La primera diferencia entre ambos es la necesidad de implementa algunos de los métodos y estructuras para dos módulos multi sensores en vez de uno. Para ello se comienzan definiendo las mismas estructuras que existen en la versión 1 y se modifican los nombres de forma que puedan ser diferenciadas entre ellas como muestra la Ilustración 65.

```
16  MAC_ADDRS_MOTOR="80:ea:ca:70:b1:81"
17  MAC_ADDRS_BOMBAS="80:ea:ca:70:b0:98"
18  CCCD_UUID = 0x2902
19
20  data_x_motor = []
21  data_y_motor = []
22  data_z_motor = []
23  temp_motor = []
24  hum_motor = []
25
26  data_x_bombas = []
27  data_y_bombas = []
28  data_z_bombas = []
29  temp_bombas = []
30  hum_bombas = []
```

Ilustración 65: Estructuras de datos sensores

Por otro lado, en la función doFFT(), se agrega el código necesario para realizar las mismas operaciones que ya se realizaban para los datos recolectados del motor al motor de bombas. También se redefine el método process_data(), asignándose un método similar para el caso de los datos del motor de tracción y otro para los datos del motor de bombas de agua, ya que estos se controlan mediante dos queues diferentes.

Otro cambio con respecto al código anterior es la inclusión de un nuevo delegado como muestra la Ilustración 66. En esta versión es necesario la existencia de dos delegados diferentes de forma que existan dos hilos distintos encargados de procesar la información que llega de forma asíncrona de los multi sensores. Cada uno de estos se encargan de recibir dichos datos, transformar los datos recibidos a hex e introducirlos en la queue correspondiente de forma que puedan ser tratados posteriormente.

```
189 class MyDelegate_bombas(DefaultDelegate):
190
191     #Constructor (run once on startup)
192     def __init__(self):
193         DefaultDelegate.__init__(self)
194     #func is caled on notifications
195     def handleNotification(self, cHandle, data):
196         raw_data=binascii.hexlify(data)
197         raw_data=str(raw_data)[2:-1]
198         raw_data_queue_bombas.put(raw_data)
```

Ilustración 66: Mydelegate del motor de bombas

En cuanto al método main existen cambios necesarios realizados para incorporar la utilización de dos equipos multi sensores en el mismo Gateway. Primero, se inicializan las queues correspondientes con los datos en crudo y los datos procesados para luego crear los objetos Peripheral encargados de las conexiones a ambos nodos. Luego se realiza el mismo procedimiento que en la versión anterior listando las características de los dispositivos, identificando los de control y lectura, de forma que se pueda asignar el delegado y se envié la orden de iniciar la lectura de los sensores con las sentencias mostradas en la Ilustración 67 entre las líneas 376 y 393 del código.

```
376             chList = pb.getCharacteristics()
377
378             for ch in chList:
379                 print(ch.uuid)
380                 if str(ch.uuid) == '2ea78970-7d44-44bb-b097-26183f402409':
381                     control_service_bombas = ch
382                 elif str(ch.uuid) == '2ea78970-7d44-44bb-b097-26183f402410':
383                     read_service_bombas = ch
384
385             control_service_motor.write(struct.pack('<b', 0x01), True)
386             pm.setDelegate(MyDelegate_motor())
387             ch_sensors_cccd = read_service_motor.getDescriptors(forUUID=CCCD_UUID)[0]
388             ch_sensors_cccd.write(b"\x01\x00", True)  # sensors on
389
390             control_service_bombas.write(struct.pack('<b', 0x01), True)
391             pb.setDelegate(MyDelegate_bombas())
392             ch_sensors_cccd = read_service_bombas.getDescriptors(forUUID=CCCD_UUID)[0]
393             ch_sensors_cccd.write(b"\x01\x00", True)  # sensors on
394
395             thread_m = threading.Thread(target=work_threads,args=(pm,'m'))
396             thread_m.start()
397             print("Hilo motor comenzó")
398
399             thread_b = threading.Thread(target=work_threads,args=(pb,'b'))
400             thread_b.start()
401             print("Hilo bombas comenzó")
```

Ilustración 67: Encender sensores en main e inicio de threads

Una vez enviada la instrucción que se encarga de encender los sensores y comenzar la captura de datos, se lanzan dos hilos mediante la clase threading como se puede observar en la Ilustración 67 a partir de la línea 395. Estos poseen como objetivo la ejecución del anterior bucle de escucha que poseía la versión anterior del código mediante la función work_threads(), donde recibe además como parámetro un identificador ‘m’ o ‘b’ en caso de que se trate del motor de tracción o del motor de bombas respectivamente. La función work_threads() como bien se comentaba, posee una definición parecida al bucle interno que existe en la primera versión del

código, sin embargo, como se puede apreciar en la Ilustración 68, existe un condicional nuevo que sirve para el control de la ejecución de los hilos. Esta variable exit_event de tipo threading.Event permite en el hilo principal cambiar el estado de ella para forzar a los hilos secundarios a detener la captura de los datos de los sensores. Otro elemento diferente que destacar es la necesidad de enviar el atributo pasado por parámetro al método post(). Dicho método se redefine al igual que el método get_data_msg() utilizado en este, de forma que se pueda discriminar entre los dos dispositivos multi sensor como se puede ver en la Ilustración 69.

```
411 #Función de trabajo de los hilos, está se encarga de esperar por las notificaciones del periferico
412 # y mediante post realizar el tratamiento pertinente
413 def work_threads(p,disc):
414     while (True):
415         if p.waitForNotifications(20.0):
416             if exit_event.is_set():
417                 exit(0)
418             else:
419                 post("0",disc)  # Pendiente valor k
```

Ilustración 68: función work_threads

```
152  def post(sensors_choosed,disc):
153      message = get_data_msg(sensors_choosed,disc)
154
155      if message != None:
156          values = str(message[1]).split("/")
157          if disc == 'm':
158              if message[0] == '1':
159                  temp_motor.append(float(values[0]))
160              if message[0] == '2':
161                  hum_motor.append(float(values[0]))
162              if message[0] == '4':
163                  data_x_motor.append(float(values[0]))
164                  data_y_motor.append(float(values[1]))
165                  data_z_motor.append(float(values[2]))
166          elif disc == 'b':
167              if message[0] == '1':
168                  temp_bombas.append(float(values[0]))
169              if message[0] == '2':
170                  hum_bombas.append(float(values[0]))
171              if message[0] == '4':
172                  data_x_bombas.append(float(values[0]))
173                  data_y_bombas.append(float(values[1]))
174                  data_z_bombas.append(float(values[2]))
```

Ilustración 69: Redefinición del método post

Estos cambios engloban la modificación o agregación de código para conseguir la misma funcionalidad que se obtiene en el apartado **6.5.- Desarrollo del módulo de recepción y tratamiento de datos en el Gateway (Versión 1)** para dos dispositivos concurrentemente. Sin embargo, existen funcionalidades que se requieren en la versión final que no han sido abordadas hasta el momento.

En primer lugar, se añade un método llamado save_txt (ver Ilustración 70) que recibe el número de elementos a guardar, el nombre de hasta cuatro variables, las cuatro variables y el nombre del archivo a guardar. Dicho método como indica su nombre se encarga de transformar los datos guardados en array a JSON y guardar dicho JSON en un archivo txt de forma que pueda ser utilizado por el módulo de envío de datos más adelante en la ejecución del programa. Esta función será llamada al finalizar de tratar los datos calculados por la FFT y en la función save_hum_temp.

```
391  #Función que se carga de guardar en un json el array pasado como argumento y posteriormente lo guarda en un txt
392  def save_txt(n_elementos,var_uno,array_uno,var_dos,array_dos,var_tres,array_tres, var_cuatro, array_cuatro,nombre):
393      json_values = []
394      if var_tres == None:
395          # Recorremos la cantidad de valores que hay en cualquiera de los arrays de datos
396          for i in range(0, n_elementos):
397              json_values.append({var_uno: array_uno[i], var_dos: array_dos[i]})
398      else:
399          for i in range(0, n_elementos):
400              json_values.append({var_uno: array_uno[i], var_dos: array_dos[i], var_tres: array_tres[i], var_cuatro: array_cuatro[i]})
401
402      json_var = {"values": json_values}
403      with open(nombre, 'w') as file:
404          json.dump(json_var, file)
```

Ilustración 70: Método save_txt

Por otro lado, también es creada la función save_hum_temp() mencionada en el párrafo anterior. Está se encarga de llamar a la función presentada en la Ilustración 70 con los datos del humedad y temperatura de cada sensor, de forma que estos puedan ser almacenados para su posterior envío.

Por último en el método main del script, se llama al método disconnect() de ambos Peripherals de forma que los dispositivos multi sensores cesen la lectura de datos, y se utiliza

un time.sleep(120) para comenzar el tiempo de espera o inactividad. Dichos cambios se realizan después de detenidos los hilos encargados de la obtención de los datos y procesados estos.

6.12. INTEGRACIÓN DE MÓDULOS EN EL GATEWAY

En el caso de la integración de los diferentes módulos desarrollados durante el proyecto, basta con añadir el código del módulo MQTT encargado del envío de los datos, al script principal o main.py. Para ello se disponen de las constantes y variables globales necesarios para su funcionamiento al comienzo del script, así como los métodos desarrollados en dicho modulo. A continuación, se disponen de los métodos originales del script principal y se añaden las líneas de código mostradas en la Ilustración 71.

```
542         if check_wifi():
543             #Por cada una de las iteraciones del código realizadas se deben buscar los archivos correspondientes
544             for i in range(0,iteracion+1):
545                 # Funcionalidad del script MQTT
546                 for id in IDS:
547                     connect_thingsboard(id,i)
548             #Volvemos a reestablecer las iteraciones a registrar
549             iteracion = 0
550         else:
551             # Nos permite saber la iteración en la que se encuentra el código para
552             # concatenarlo al nombre del archivo a guardar
553             iteracion = iteracion + 1;
```

Ilustración 71: Funcionalidad script MQTT

Dichas líneas como se comentan en el apartado **6.9 Generación final del módulo de envió de datos a thingsboard en el gateway**, se encargan de iniciar el envío de los datos a la

plataforma Thingsboard. Estás se ubican en el método main del script entre la desconexión de los Peripherals y el tiempo de espera del dispositivo. Cabe destacar que existe alguna diferencia con respecto al código del módulo de envío de datos, ya que en este se puede observar un bucle "for" que no existe en la versión presentada en el apartado antes mencionado, así como un condicional que llama a check_wifi().

Estás líneas de código son necesarias para que la versión final pueda guardar los archivos en local para posteriormente se puedan enviar todos los datos cuando se posea conexión a la WiFi configurada. En primer lugar, se establece una variable "iteración" que actúa de contador, está permite al programa conocer la iteración en la que se encuentra del bucle principal para concatenar dicho valor al nombre de los archivos .txt que conservan los datos.

Por otro lado, el condicional como bien se comentaba, llama a la función check_wifi() cuya definición se aprecia en la Ilustración 72. Dicha función se encarga de llamar al script de conexión a WiFi y discriminar según su salida de forma que se pueda determinar si la conexión es correcta o no. En caso de que lo sea, se ejecuta el bucle "for" antes mencionado, encargado de recorrer los valores de 0 a iteración, de forma que se recorran todos los archivos guardados anteriormente por el programa, y se envíen los valores contenidos en estos a Thingsboard. En caso contrario, se aumenta en uno la variable "iteración" y se procede al sleep del tiempo inactivo.

```
422  #Función encargada de ejecutar el script wifi_check y comprobar su salida
423  def check_wifi():
424      #Ejecutamos el script encargado de realizar las operaciones sobre el WiFi y su conexión
425      process = subprocess.run(["./wifi_check"],stdout=subprocess.PIPE,stderr=subprocess.PIPE)
426      #Si el programa termina con el código 0, se devuelve True, es deci, hay conexión
427      if process.returncode == 0:
428          return True
429      else:
430          #En caso contrario False (sin conexión)
431          return False
```

Ilustración 72: Definición de check_wifi()

6.13. SCRIPTS ALTERNATIVOS PARA EL HAT DE COMUNICACIONES

Como se comenta en el apartado **6.8. Pruebas envío de datos mediante el hat de comunicaciones por mqtt** , los problemas que presenta el módulo SIM7000E hacen que este sea inviable para la solución final de este proyecto. Sin embargo, se han desarrollado dos scripts alternativos que utilizan las opciones de comunicación mediante MQTT que permite el HAT de comunicaciones, de forma que puedan ser utilizados de cara a un futuro. A continuación, se describen las principales diferencias y modificaciones con respecto al código de la solución final para cada una de las alternativas.

6.13.1. SCRIPT CON CONEXIÓN MEDIANTE COMANDOS AT

En primer lugar, el envío de la información a la plataforma de Thingsboard se desarrolla una vez que el Gateway finaliza el procesamiento de los datos, por lo que se elimina el código encargado de contar las iteraciones del bucle principal, así como el bucle "for" anidado que existe en la versión final (ver Ilustración 71). Por otro lado, se elimina la necesidad de revisar la conexión WiFi en cualquier momento dado por lo que se elimina el método encargado de hacer este proceso, así como el condicional que lo ejecuta.

Luego, se debe crear una función llamada inicializar_sim() que es utilizada en la primera ejecución del script, de forma que se pueda establecer la comunicación como se comenta en el apartado **6.8 Pruebas envío de datos mediante el hat de comunicaciones por mqtt**. La definición de dicha función se puede observar en la Ilustración 73. Todas las comunicaciones con el módulo de comunicaciones se realizan mediante un objeto serial por el puerto "/dev/ttyUSB2".

```python
def inicializar_sim():
    #Conexión con el puerto serial encargado de llevar a cabo los
    ser = serial.Serial("/dev/ttyUSB2",baudrate=115200)
    ser.flushInput()
    #Se introduce el PIN de la SIM y se lee el resultado
    ser.write('AT+CPIN=1234')
    ser.read(100)
    #Se esperan 10 segundos para que el dispositivo se registre en red
    #Y se comprueba que así sea
    time.sleep(10)
    ser.write('AT+CREG?')
    ser.read(100)
    #Se adjunta el dispositivo a la red
    ser.write('AT+CGATT=1')
    ser.read(100)
    time.sleep(5)
    #Se activa el perfil 13 de IP
    ser.write('AT+CGACT=1,13')
    ser.read(100)
    time.sleep(2)
    #Se seleccióna el perfil 13 como activo
    ser.write('AT+CNACT=1,13')
    ser.read(100)
    time.sleep(2)

    #Se establece la configuración global del MQTT
    ser.write('AT+SMCONF="URL","' + str(IP_BROKER) + '","1883"')
    res = ser.read(100)
    if res.stdout != 'OK':
        return -1
    #Se cierran las comunicaciones
    ser.close()
    pass
```

Ilustración 73: inicializar_sim() script con comandos AT

Por otro lado, también es necesario modificar las funciones connect_thingsboard() y do_work(). Su funcionamiento es exactamente el mismo, pero mediante comandos AT como se observa en la Ilustración 74 e Ilustración 75.

```
94  def connect_thingsboard(id):
95      # Conexión con el puerto serial encargado de llevar a cabo los
96      ser = serial.Serial("/dev/ttyUSB2", baudrate=115200)
97      ser.flushInput()
98
99      # Autenticación con el broker MQTT
100     ser.write('AT+SMCONF="USERNAME","' + str(id) + '"')
101     res = ser.read(100)
102     if res.stdout != 'OK':
103         return -1
104
105     # Buscamos el nombre de archivo según el diccionario definido al principio
106     datos = open_json(JSONFILES[id])
107     print("nombre del json  --------> " + str(JSONFILES[id]))
108     # Terminamos de conectar con Thingsboard y enviamos los datos
109     do_work(datos,ser)
```

Ilustración 74: connect_thingsboard() con comandos AT

```python
def do_work(datos,ser):
    # Conexión con el broker MQTT
    ser.write('AT+SMCONN')
    res = ser.read(100)
    try:
        # Bucle encargado de recorrer todos los valores del json y enviarlos a Thingsboard
        for reg in datos["values"]:
            val = json.dumps(reg)
            tam = len(val)
            # Se envia la información de los datos a enviar
            ser.write('AT+SMPUB="' + str(TOPIC) + '",1,"' + tam + '",')
            time.sleep(0.2)
            # Se introduce por consola el valor a enviar
            ser.write([val])
            # Se espera a que el proceso se termine
            time.sleep(1)

    except KeyboardInterrupt:
        pass
    time.sleep(1)
    # Se desconecta el cliente MQTT
    ser.write('AT+SMDISC')
    ser.read(100)
    #Se cierra la conexión
    ser.close()
```

Ilustración 75: do_work() con comandos AT

6.13.2. SCRIPT CON CONEXIÓN MEDIANTE IP

En el caso de la conexión por IP con el módulo SIM7000E, es necesario realizar un paso previo antes de la inicialización de la tarjeta SIM y un paso posterior. Estos son los ya comentados en el apartado **6.8 Pruebas envío de datos mediante el hat de comunicaciones por mqtt**, por lo que mediante subprocess se ejecuta el archivo “qmicli.sh” al comienzo de la inicialización y al archivo “qmicli2p.sh” al finalizar la comunicación.

Una vez realizado esto el dispositivo posee una dirección IP que puede ser utilizada para enviar los datos, por lo que basta con enviar los datos recolectados directamente sin necesidad de realizar el check_wifi(), ni guardar diferentes archivos donde retener los valores recopilados hasta poseer conexión a una red WiFi como en el caso anterior.

En cuanto a las funciones connect_thingsboard() y do_work(), estás se encuentran definidas como la versión original, enviando los datos mediante paho-mqtt como muestran las Ilustraciones 62 y 63.

7. Resultados obtenidos

Una vez desarrollado y probada la solución final del script del Gateway, así como todos aquellos componentes auxiliares en su ejecución y el firmware del dispositivo multi sensor, se realizan pruebas definitivas en dos entornos diferentes de forma que se pueda observar su correcto funcionamiento. El primero de estos será llevado a cabo en el laboratorio como otras pruebas ya realizadas durante el desarrollo del prototipo, mientras que la segunda tanda de pruebas se busca realizar en el vehículo eléctrico ALKE 340E de la empresa EMULSA, haciendo énfasis en sus motores de tracción y de las bombas de agua.

7.1. LABORATORIO

Se realizan pruebas en el laboratorio similares a aquellas realizadas y comentadas en apartados anteriores del documento. Estas pruebas son llevadas a cabo sobre el disco duro externo de 90 Hz y el dispositivo multi sensor en su configuración de 1 KHz que corresponde con los 3KHz reales como se comenta en el apartado **6.6. Pruebas de diferentes configuraciones de frecuencia de muestreo para el acelerómetro**. Cabe destacar que el envío de los datos recolectados durante dichas pruebas es realizado mediante WiFi, ya que durante la realización de estas no se dispone del módulo HAT de comunicación.

La disposición de los elementos es idéntica a la mencionada anteriormente, donde el nodo multi sensor se ubica encima del disco duro externo y este a su vez se encuentra conectado al ordenador.

Una vez identificado en arranque del disco duro externo, se comienza la ejecución del script en el Gateway para la conexión con los dispositivos y la recolección de datos. Se puede observar el correcto funcionamiento en tiempo real de la plataforma Thingsboard y el envío de datos a esta en la Ilustración 76 y la Ilustración 77. En la primera se observan los valores que corresponden con los valores enviados de la FFT, y en la segunda se observan los datos correspondientes con la humedad y temperatura recolectados.

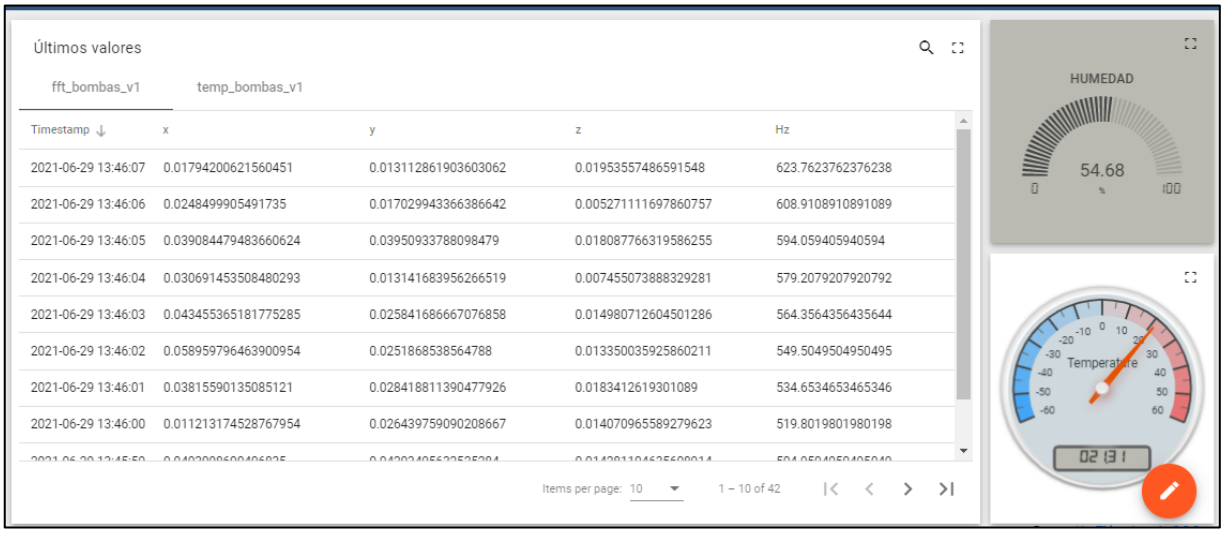

Ilustración 76: Recepción de datos FFT pruebas finales laboratorio

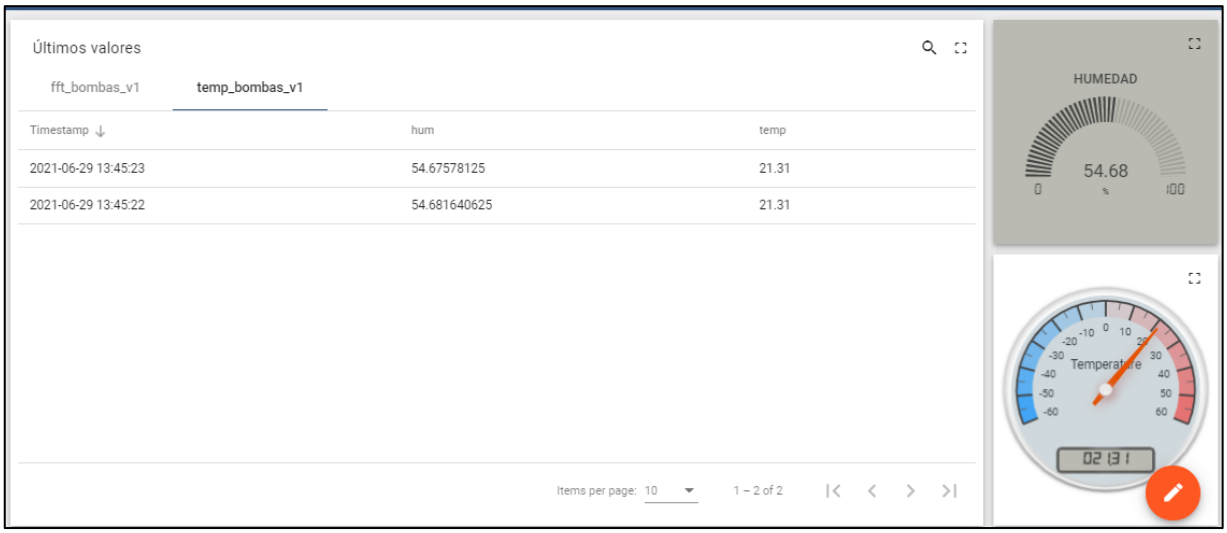

Ilustración 77: Recepción de datos temperatura y humedad pruebas finales laboratorio

Por otro lado, se observa en la Ilustración 78 la gráfica de la FFT resultante de la primera tanda de datos capturados donde destaca la presencia del primer armónico alrededor de los 90Hz que corresponde con la frecuencia del disco duro externo, por lo que se confirma la correcta calibración del dispositivo.

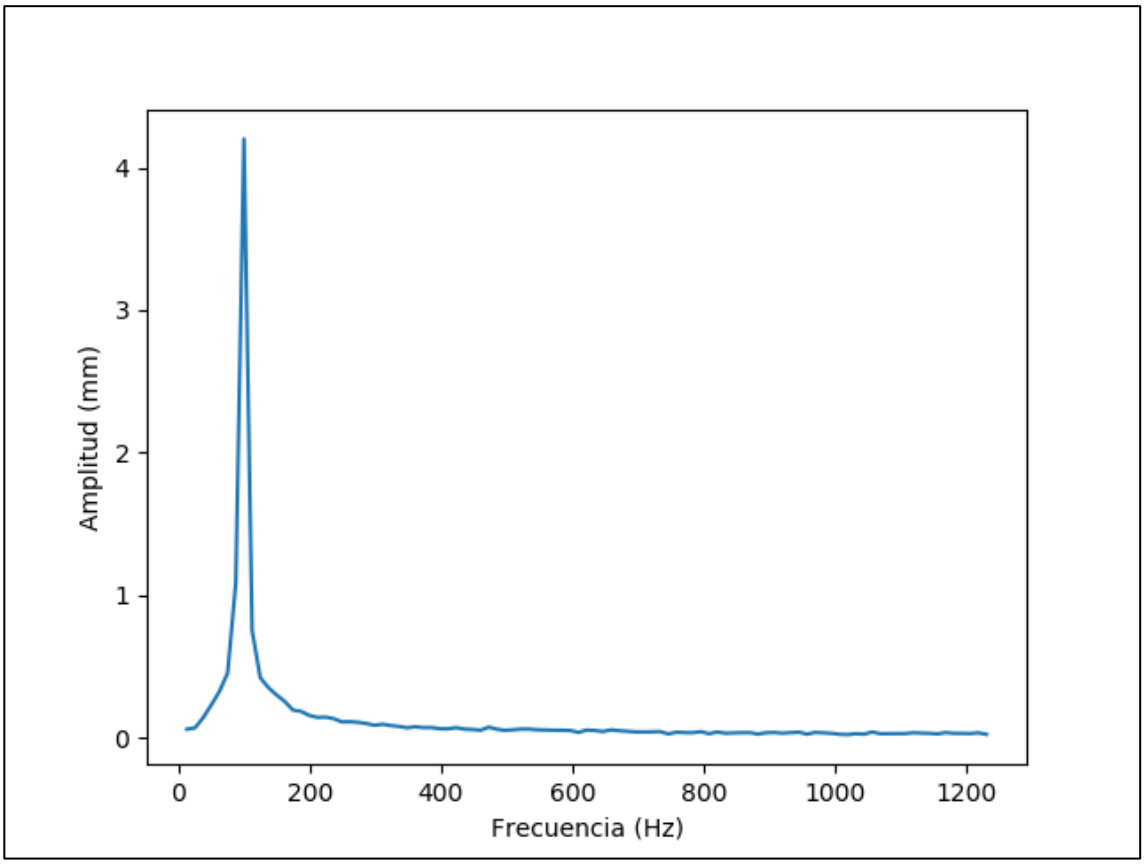

Ilustración 78: FFT resultante de las pruebas en el laboratorio

7.2. EMULSA

En la primera visita realizada a la nave donde se ubican los vehículos eléctricos de EMULSA, se inspeccionan los vehículos para determinar la ubicación ideal de los diferentes dispositivos que conforman el prototipo. Como se ha comentado durante el desarrollo del presente documento, uno de los dispositivos multi sensores será ubicado en el motor de tracción del vehículo que se encuentra en la zona del eje trasero, entre este y el chasis. Ante una detallada

observación, se determina que la ubicación ideal del primer dispositivo será en la superficie plana que se señala en la Ilustración 79, de forma que el multi sensor puede fijarse mediante bridas metálicas alrededor del motor y un compuesto adhesivo para máxima seguridad del dispositivo.

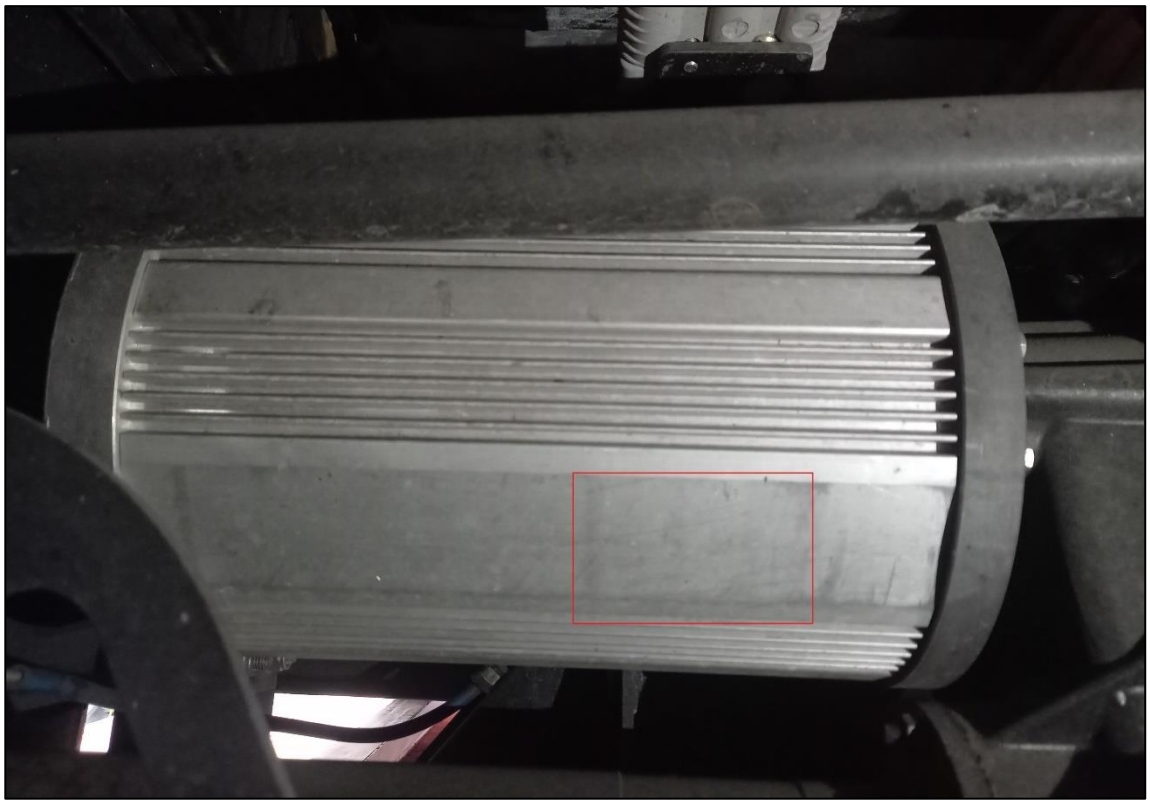

Ilustración 79: Ubicación del dispositivo multi sensor (motor de tracción)

Por otro lado, el segundo nodo multi sensor estará ubicado en el motor de bombas de agua. Este se encuentra mucho más resguardado que el anterior como se muestra en la Ilustración 80.

La sujeción de este nodo será parecida a la anterior, ubicándose está en la superficie mostrada en la Ilustración 80.

Ilustración 80:Ubicación del dispositivo multi sensor (motor de bombas de agua)

Por último, el dispositivo Gateway se ubicará en la cabina del conductor. Está se encontrará debajo del asiento del conductor, ya que dispone de espacio suficiente, ventilación y seguridad (compartimiento cerrado bajo llave).

En cuanto a las pruebas a realizar se dispone de una cita para ello, aunque está excede el plazo para la entrega del presente documento.

8. Conclusiones

En función de los objetivos planteados al inicio del documento y los resultados obtenidos, se puede concluir que estos han sido alcanzados, logrando desarrollar un prototipo encargado de capturar, tratar y enviar los datos a la plataforma Thingsboard. Los resultados obtenidos del desarrollo de los diferentes módulos y las versiones alternativas servirán como base para futuras iteraciones sobre la monitorización en tiempo real de motores eléctricos, ya que dicho tema es un trabajo a largo plazo que se busca desarrollar mediante una beca del IUTA (Instituto Universitario de Tecnología Industrial de Asturias), además de poseer código extrapolable, permitiendo que sea utilizado con otros dispositivos multi sensores y Gateways.

También cabe destacar que la recolección del análisis frecuencial fruto de los datos recolectados a partir del tratamiento de las aceleraciones de los motores mediante la FFT, permitirá elaborar estudios en cuanto al comportamiento normal y anormal de estas, abriendo la posibilidad al desarrollo de aplicaciones que permitan informar de fallos en los motores sin necesidad de interacción humana, de forma precisa.

En términos personales, el estudio del modelo de computación asociado a las plataformas Cloud, la configuración de una plataforma de esta índole como Thingsboard, los diferentes protocolos de comunicación posibles y su capacidad de integración con todo tipo de dispositivos, me ha permitido entender la potencia y versatilidad que existe en esta tecnología emergente que marca y marcara tendencia.

Por otro lado, el desarrollo de software para los dispositivos Edge como son los DA14585 y los dispositivos que se encuentran en el Fog como la Raspberry Pi, me ha dado la oportunidad de aprender tecnologías nuevas que no son estudiadas durante la realización del grado, tan

necesarias para el desarrollo de aplicaciones reales que atienden a problemas relevantes en todo tipo de Industrias.

El presente proyecto sirve de base para un desarrollo más profundo mediante la beca del IUTA como ya se había comentado, por lo que se mencionan las posibles tareas o mejoras que se pueden realizar en un futuro.

- Realizar pruebas en el entorno final de funcionamiento del prototipo, es decir, en el vehículo ALKE 340E de EMULSA.
- Realizar las modificaciones pertinentes según la información que provea el fabricante del módulo SIM7000E, de forma que se puedan enviar los resultados cuando no se disponga de conexión WiFi.
- Realizar un análisis de frecuencia sobre los motores del vehículo ALKE 340E para determinar las frecuencias más relevantes para determinar la condición del vehículo. Una vez determinadas estas, se podrá realizar solamente el tratamiento y envío de dichos datos para aligerar la carga de trabajo del dispositivo Gateway.
- Implementar un modelo que permita relacionar la frecuencia y la amplitud con las averías más comunes en motores eléctricos, de manera que se pueda realizar un correcto mantenimiento predictivo a los motores del vehículo. Para ello será necesario el estudio de los motores en condiciones nominales y observar su evolución con respecto al tiempo, relacionando esto con las tareas de mantenimiento habituales y los fallos que se presenten.

9. Referencias

[1] Cisco. (Marzo, 2020). Cisco Annual Internet Report (2018-2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Dave Evans. (Abril, 2011). Internet de las cosas, Como la próxima evolución de Internet lo cambia todo. https://www.cisco.com/c/dam/global/es_mx/solutions/executive/assets/pdf/internet-of-things-iot-ibsg.pdf

[3] Real academia española. Recuperado el 11 de Mayo de 2021 de https://dle.rae.es/mantenimiento

[4] Reinaldo E. Puga C. (2014). Diseño de un Programa de Mantenimiento Predictivo Basado en el Análisis de Criticidad de los Motores Eléctricos de Inducción Trifásica de Inyección de Agua y Transferencia de Crudo de la Planta UM-2 de la Superintendencia de Mantenimiento de PDVSA-PETRODELTA. (Trabajo de Grado). Universidad Nacional Experimental Politécnica, Ciudad de Guayana.

[5] Alfonso Fernández (s.f.). Fundamentos del análisis de las vibraciones. https://power-mi.com/es/content/estudio-de-las-vibraciones

[6] FFT-Time-Frecuency-View (2017). Wikimedia Commons. https://commons.wikimedia.org/wiki/File:FFT-Time-Frequency-View.png

[7] National Institute of Standarts and Technology, Michaela Iorga, Larry Feldman, Robert Barton, Michael J. Martin, Nedim Goren, Charif Mahmoudi (s.f.). Fog Computing Conceptual Model. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf

[8] Las características de la Raspberry Pi 4 han sido extraídas de https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

[9] Las características de la plataforma Thingsboard han sido extraídas de https://thingsboard.io/

[10] Dialog semiconductor, Christopher Piggin, Wissel Lubberhuizen (Mayo,2018). Low Power Sensor Fusion for Wearables and Wireless Devices. https://www.dialog-semiconductor.com/sites/default/files/whitepaper_low_power_sensor_fusion_2018.pdf

10. Bibliografía

(Mayo, 2021). Obtenido de Alke:

https://www.alke.eu/es/

(Marzo, 2021). Características del multi sensor DA148585. Obtenido de Dialog semiconductor:

https://www.dialog-semiconductor.com/products/da14585-iot-multi-sensor-development-kit#tab-field_tab_content_overview

(Marzo, 2021). Características del sensor BME680. Obtenido de Bosch:

https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme680/

(Marzo, 2021). Características del sensor IMC42605.Obtenido de Invensense TDK:

https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42605/

(Mayo, 2021). Características del vehículo ATX340E. Obtenido de Alke:

https://www.alke.eu/es/vehiculos-electricos-atx340e

(Enero, 2021). Aplicación IoT Sensors. Obtenido de Google play store:

https://play.google.com/store/apps/details?id=com.dialog.wearables&gl=ES

(Enero, 2021). Sistema operativo Raspian OS. Obtenido de Raspberry Pi:

https://www.raspberrypi.org/software/

(Febrero,2021). Descarga de Open JDK. Obtenido de Adopt Open JDK:

https://adoptopenjdk.net/index.html?variant=openjdk8&jvmVariant=hotspot

(Febrero,2021). Descarga de PostgreSQL. Obtenido de Enterprise DB:

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows

(Febrero,2021). Descarga de la plataforma Thingsboard. Obtenido de Thingsboard:

https://github.com/thingsboard/thingsboard/releases/download/v3.2/thingsboard-windows-3.2.zip

(Febrero,2021). Características de la plataforma Kafka. Obtenido de Kafka:

https://kafka.apache.org/

(Febrero,2021). Características de la plataforma RabbitMQ. Obtenido de RabbitMQ:

https://www.rabbitmq.com/

(Enero,2021). Características de debugger Segger. Obtenido de Segger:

https://www.segger.com/products/debug-probes/j-link/models/j-link-ob/

(Enero, 2021). Manual de usuario DA14585. Obtenido de Dialog semiconductor:

https://www.dialog-semiconductor.com/sites/default/files/user_manual_um-b-057.pdf#page=23&zoom=100,94,528

(Enero, 2021). Manual de programador DA145858. Obtenido de Dialog semiconductor:

http://lpccs-docs.dialog-semiconductor.com/UM-102-DA14585_IoT_MSK_Getting+Started+Guide/05_Hardware%20description/Hardware%20description.html

(Marzo, 2021). MAGADÁN, L., et al. Low-cost real-time monitoring of electric motors for the Industry 4.0. Procedia Manufacturing, 2020, vol. 42, p. 393-398.

(Marzo, 2021). MAGADÁN, Luis, et al. Real-Time Monitoring of Electric Motors for Detection of Operating Anomalies and Predictive Maintenance. En International Summit Smart City 360°. Springer, Cham, 2019. p. 301-311.

(Enero, 2021). Información sobre plataformas IoT. Obtenido de GeekFlare:

https://geekflare.com/es/iot-platform-tools/

(Febrero, 2021). Documentación de la clase Peripheral. Obtenido de Github:

https://ianharvey.github.io/bluepy-doc/peripheral.html

(Febrero, 2021). Información del manejo de notificaciones. Obtenido de Github:

https://ianharvey.github.io/bluepy-doc/notifications.html#notifications

(Marzo, 2021). Información sobre Matplotlib. Obtenido de Matplotlib: https://matplotlib.org/

(Marzo, 2021). Descarga y utilización de paho-mqtt. Obtenido de Pypi: https://pypi.org/project/paho-mqtt/

11. Anexo 1: Estructura del código fuente

En conjunto con la entrega del presente documento, se entrega una carpeta comprimida con el código fuente desarrollado durante el proyecto. En este apartado se procede a mencionar la estructura de dicha carpeta.

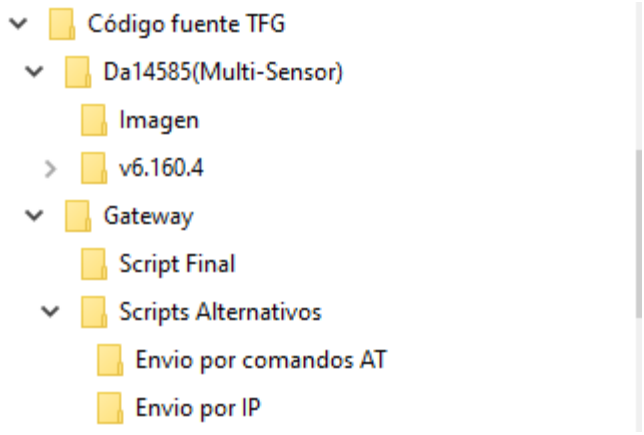

Ilustración 81: Estructura entregable código fuente

En primer lugar, en la carpeta raíz se encuentran por un lado el código del módulo Multi Sensor bajo el directorio “Da14585(Multi.Sensor)”, mientras que por el otro se posee el código del Gateway bajo el directorio “Gateway”.

Dentro del directorio “Da14585(Multi-Sensor)”, se observan dos carpetas como se muestra en la Ilustración 81.

En la primera de ellas llamada “Imagen”, se guardan los archivos de extensión .hex, .elf y .bin definitivos de la solución, es decir, en ella se guardan los archivos utilizados en primera instancia para la depuración del código (.elf) y el archivo .hex resultante utilizado para generar él .bin que es “quemado” en la memoria flash del nodo Multi Sensor como se comenta en el desarrollo del documento.

En la segunda carpeta llamada “v6.160.4”, se dispone del código fuente utilizado para el desarrollo del firmware del Multi Sensor. Este puede ser importando en SmartSnippets de forma que se pueda observar de manera organizada su estructura.

Por otro lado, en el directorio “Gateway” se disponen dos subdirectorios llamados “Script Final” y “Scripts Alternativos” como se observa en la Ilustración 81.

En el primer subdirectorio llamado “Script Final” se guardan los siguientes archivos:

- Script_local.py: Código fuente de la solución final del prototipo desarrollado en el apartado **6.12. Integración de módulos en el gateway**.
- Wifi_check: Script .sh desarrollado para comprobar la conexión mediante WiFi en el apartado **6.10 Desarrollo del script de conexión a wifi en el gateway** y utilizado por script_local.py.

Por otro lado, el subdirectorio llamado “Scripts Alternativos” posee a su vez dos carpetas como se muestra en la Ilustración 81. Dicho subdirectorio posee el código necesario para la ejecución de los scripts comentados en el apartado **6.13 Scripts alternativos para el hat de comunicaciones**.

Como sucede en el caso anterior, cada una de estas carpetas posee el código fuente necesario para la ejecución correcta de cada una de las soluciones. En el caso de "Envío por comandos AT" se dispone de:

- Script_AT.py: Código fuente del script de conexión por comandos AT comentado en el apartado **6.13.1 Script con conexión mediante comandos at**.

Por último, en el directorio "Envío por IP" se dispone de los archivos:

- Script_IP.py: Código fuente del script de conexión por IP comentado en el apartado **6.13.2 Script con conexión mediante IP**.
- Qmicli.sh: Script .sh de apoyo para la desconexión de la interfaz wwan0 y modificación del tipo de IP del interfaz comentado en el apartado **6.8 Pruebas envío de datos mediante el hat de comunicaciones por mqtt.**
- Qmicli2p.sh: Script .sh de apoyo para la iniciar la red mediante la herramienta qmicli y dotar del protocolo DHCP a la interfaz wwa0 como se comenta en el apartado **6.8 Pruebas envío de datos mediante el hat de comunicaciones por mqtt.**