Universidad Zaragoza
1542

Trabajo Fin de Máster

Mantenimiento predictivo; estado del arte y desarrollo de herramienta para monitorización.

Autor/es

Pablo Martínez Galindo

Director/es

Roberto Casas Nebra

Escuela de Ingeniería y Arquitectura

2019

MANTENIMIENTO PREDICTIVO; ESTADO DEL ARTE Y DESARROLLO DE HERRAMIENTA PARA MONITORIZACIÓN.

RESUMEN

El presente trabajo se enfoca en el mantenimiento predictivo. Este concepto adquiere gran relevancia con el auge de la denominada Industria 4.0, caracterizada por las fábricas conectadas e inteligentes. *(López Ramón y Cajal, 2016)*

El trabajo se divide en dos partes. En primer lugar, se ha realizado un estudio del estado del arte actual del mantenimiento predictivo. Se han analizado y explicado las diferentes etapas que se deben llevar a cabo para establecer un programa de mantenimiento predictivo, así como las técnicas y herramientas utilizadas en cada fase. El objetivo de este estudio es determinar cuáles son las técnicas de monitorización que permiten predecir la mayoría de las averías comunes en máquinas.

En segundo lugar, se ha desarrollado una herramienta de monitorización de máquinas. Esta herramienta, basándose en el estudio del estado del arte realizado, pretende poder predecir aquellos fallos que ocurren con mayor frecuencia en la mayoría de las máquinas. Para ello se compone de un acelerómetro, un medidor de consumo eléctrico, un micrófono y un medidor de gradientes de temperatura.

La adquisición, almacenamiento y procesado de datos se realiza con una Raspberry Pi, la cual puede servir de interfaz al usuario o enviar de forma remota los resultados a otros equipos.

ABSTRACT

This text is focused on predictive maintenance, concept which has earned significant attention due to Industry 4.0 growth.

This paper is divided into two parts. First one, consists on an essay based on the state of art of predictive maintenance. Several stages must be carried out in order to develop a predictive maintenance program. For this study all these stages have been analyzed, as well as tools and techniques used in them. The goal of the study is established which monitoring techniques can predict most common machines failures.

In the second part, has been developed a monitoring tool, able to predict these most common machines failures. This tool has an accelerometer, a microphone, a temperature gradient meter and an electric current meter.

Data acquisition, storage and processing is developed on a Raspberry Pi, which also works like a human-machine interface or send results to another computer remotely.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE ANEXOS

I. INTRODUCCIÓN

En la industria actual el mantenimiento es un punto clave durante el desarrollo y la fabricación de un producto. Se deben minimizar tiempos y costes de fabricación, con los mejores estándares de calidad posibles. De este modo, paradas en las líneas de producción o gastos innecesarios en reparar maquinaria son inadmisibles.

En la nueva era de las fábricas inteligentes, digitalizadas y conectadas, el papel del mantenimiento predictivo adquiere gran importancia. La posibilidad de poder disponer y procesar grandes cantidades de datos a tiempo real, gracias a tecnologías como el Internet de las cosas, IoT (Internet of Things), o el BigData, hace que el mantenimiento predictivo pueda reducir significativamente despilfarros de tiempo y material, uno de los principales objetivos de la metodología Lean. *(Seebo, 2018).*

Debido a esta dependencia de las nuevas tecnologías, el mantenimiento predictivo, actualmente, va fuertemente asociado al concepto de Industria 4.0.

Implantando un programa de mantenimiento predictivo se pretende asegurar la disponibilidad y rendimiento de los elementos que componen un sistema. Para ello, se utilizan diferentes tecnologías de instrumentación que permiten medir y analizar ciertos parámetros y variables con los que se puede pronosticar la avería de un componente o sistema.

Se reducen así los costos derivados del mantenimiento correctivo como pueden ser las paradas no planificadas y la rotura de otros elementos del sistema, y del preventivo, como la reducción del stock de piezas de recambio y el mejor aprovechamiento de estas, respectivamente. *(Seebo, 2018).*

El objetivo final es conseguir un mantenimiento “Just In Time”, de forma que si se sabe cuándo va a fallar una máquina, puede reorganizarse la línea de fabricación para que esa parada afecte de la menor manera posible, optimizando el funcionamiento de la planta.

El mantenimiento predictivo es una técnica reciente. Hasta ahora, se intentaba llevar a cabo este planteamiento, pero las mediciones se realizaban periódicamente por un operario, con equipos portátiles de medición. Rara vez se colocaban sensores en el propio sistema para monitorizarlo continuamente. La información obtenida se analizaba junto con la información suministrada por el fabricante para obtener una aproximación del estado de la máquina y finalmente era una persona la que tomaba la decisión sobre en qué momento parar la máquina.

Con la llegada de la industria 4.0 se pueden equipar estos sensores en cualquier sistema y disponer de los datos en cualquier lugar de forma remota. Los sensores se encargan de las mediciones, y de forma inalámbrica envían estos datos a un centro de control, donde se realiza el análisis de estos. Las nuevas tecnologías inalámbricas permiten enviar los datos sin latencia, lo que permite conocer el estado instantáneo de cada máquina sin necesidad de estar delante de la misma

Esta accesibilidad a la información, unido al machine learning, permite que la máquina sepa cuando va a averiarse, avisando con tiempo para poder planificar la reparación, así como deteniéndose antes de llegar a producirse una avería. *(Seebo, 2018).*

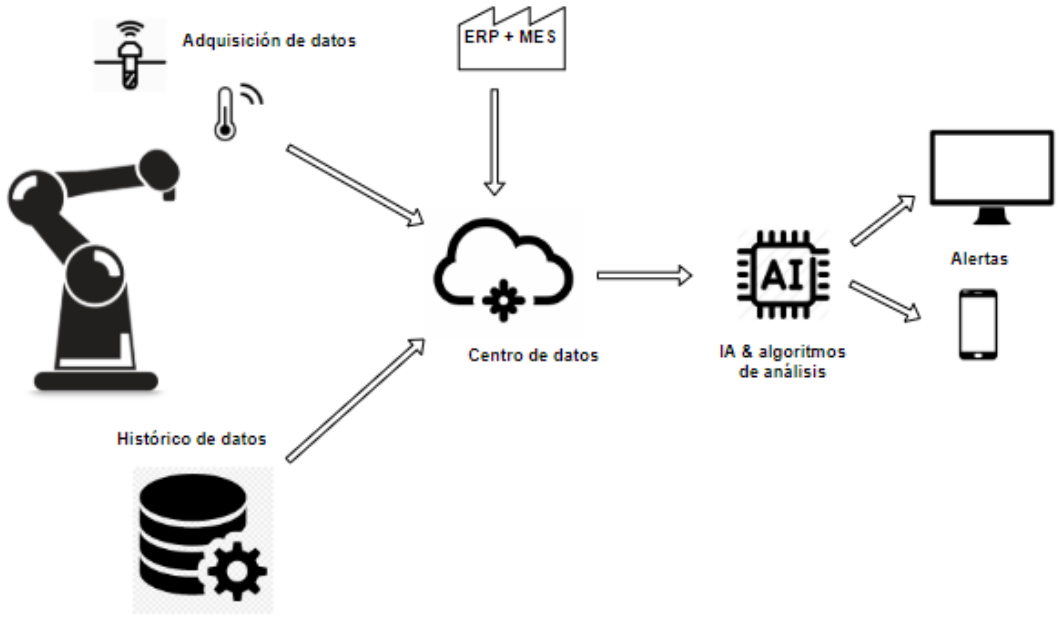

Ilustración 1- Arquitectura mantenimiento predictivo

La ilustración 1 representa la arquitectura de un sistema de mantenimiento predictivo. Los sensores equipados en la máquina transmiten los datos al centro de datos. Estos datos, junto con los datos históricos y las ordenes de los sistemas ERP y MES son procesados por algoritmos que predicen los fallos y optimizan las paradas para realizar labores de mantenimiento.

Esta información no solamente es útil para el cliente, si no que el propio fabricante de la maquinaria puede utilizar estos datos para ofrecer planes de servicio técnico.

El mantenimiento predictivo se debe entender como una técnica que ayuda a mejorar el funcionamiento global de un proceso de producción. Deben identificarse las máquinas y procesos más críticos para el proceso de producción y aplicar en ellos estas técnicas, para así optimizar el funcionamiento de la planta.

Conocida la importancia que está adquiriendo el mantenimiento predictivo en la industria actual y la estructura de este tipo de sistemas, se plantean dos objetivos clave a abordar en este trabajo.

En primer lugar, se va a realizar un análisis del estado del arte actual del mantenimiento predictivo. Se pretende definir cuáles son las técnicas utilizadas con mayor frecuencia, así como las causas más comunes de fallo y los elementos de las máquinas cuyas averías se pueden predecir. Del mismo modo se pretende analizar las técnicas de análisis y la

instrumentación utilizada, con el fin de conseguir una visión global completa del mantenimiento predictivo actual.

En segundo lugar, se va a desarrollar una herramienta, basada en una Raspberry Pi, la cual dispondrá de diferentes sensores, los cuales podrán monitorizar aquellas variables de operación que permitan predecir averías con mayor frecuencia.

La estructura de esta herramienta se basa en el estudio del estado del arte realizado previamente.

II. ESTADO DEL ARTE

En función del objetivo de la monitorización, se diferencian tres tipos de técnicas para la detección de funcionamientos inadecuados:

- Técnicas que monitorean las variables de operación de la máquina.
  Consiste en analizar el funcionamiento global de la máquina, y cuando el rendimiento o sus variables de operación son inferiores a un valor determinado se detiene la máquina para su revisión y reparación. Un ejemplo de sistemas en los que se aplica esta técnica son las turbomáquinas de grandes tamaños, midiendo el rendimiento global del sistema, o en máquinas herramienta, midiendo la rugosidad o tolerancia de las piezas producidas. *(Saavedra, 2010).*

- Técnicas que monitorean los fluidos y residuos de la máquina.
  Consiste en analizar periódicamente los fluidos de trabajo de la máquina, generalmente aceite, de forma que si alguno de los elementos de la máquina se deteriora se manifestará en estos fluidos. *(Saavedra, 2010)*.

- Técnicas que monitorean el comportamiento dinámico de la máquina.
  Analizando diferentes variables de funcionamiento de la máquina, cómo pueden ser las vibraciones, la temperatura de trabajo o el consumo eléctrico, se puede predecir el fallo de elementos concretos de la máquina. Esta técnica es la que mayor dedicación requiere, debiendo identificar previamente el problema a solucionar y las causas de este, para posteriormente poder definir el sistema de monitorización.

La posibilidad de disponer de los datos de forma instantánea y analizarlos en tiempo real, convierte la última técnica en la más fiable a la hora de predecir averías, ya que no solo indica cuando va a fallar la máquina, sino que también se conoce el elemento que debe ser reparado.

A continuación, se explican las diferentes etapas a seguir a la hora de implantar un programa de mantenimiento predictivo, recogiendo para cada una de ellas los principales fallos y técnicas utilizadas actualmente.

i. Fases del mantenimiento predictivo

A la hora de planificar un programa de mantenimiento predictivo hay que tener en cuenta que monitorizar y controlar una máquina tiene un coste de implantación y de explotación, y que el objetivo es optimizar el funcionamiento de una planta o proceso, por lo que no es rentable monitorizar cada una de las máquinas.
Llevar a cabo correctamente la implantación de un programa de mantenimiento predictivo comienza por detectar qué máquinas de la planta sufren averías críticas para el proceso de producción, o aquellas averías muy costosas de reparar.

Una vez identificadas las máquinas, se debe localizar el elemento que produce la avería de cada máquina. Cuando se ha localizado el elemento crítico, se deben analizar las causas del fallo, es decir, las razones físicas por las que este elemento se rompe. (*Royo*).

Por ejemplo, en una línea de mecanizado, se localiza un cuello de botella en un taladro que sufre averías frecuentemente, ralentizando todo el proceso. Se descubre que el elemento que produce las averías es un rodamiento, el cual se rompe debido a un desalineamiento.

Para poder predecir la rotura es necesario definir aquellas variables o parámetros a monitorizar, que conociendo sus valores permitirán determinar el estado de la condición física que produce el fallo. Definidas estas variables, se seleccionan los sensores e instrumentos de medida adecuados para obtener estos valores.
Por último, solo queda enviar y almacenar la información obtenida, para poder procesarla junto a otros datos y analizar los resultados obtenidos.

Etapas básicas en la implantación de un plan de mantenimiento predictivo

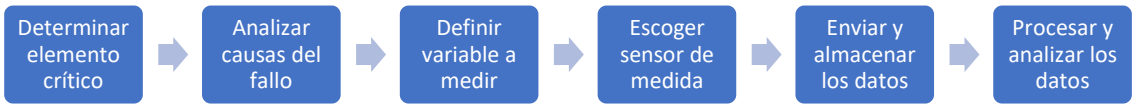

Ilustración 2 - Etapas básicas

ii. Elementos críticos

Son multitud los elementos que pueden fallar en un sistema o máquina, pero existen ciertos elementos cuyas averías se reconocen utilizando estas técnicas con mayor frecuencia.

La gran mayoría de máquinas incorporan elementos mecánicos rotativos o mecanismos de transmisión. Estos componentes, debido al gran desgaste que sufren provocan frecuentes averías. Además, muchas veces su rotura puede desencadenar graves consecuencias afectando a otros componentes de la máquina. Conociendo el momento exacto en el que va a fallar el componente, se evita su rotura y además se aprovecha al máximo su vida útil. Algunos ejemplos de estos elementos son rodamientos, cojinetes, poleas, engranajes o uniones entre ejes. *(International Atomic Energy Agency, 2007).*

Dos de las máquinas que con mayor frecuencia se pueden encontrar en fábricas y máquinas son los motores eléctricos y los transformadores. Debido al constante uso y a sus partes rotativas, es frecuente que se produzcan averías en estos equipos. Estas máquinas eléctricas son las responsables del correcto funcionamiento de muchas plantas y procesos, por lo que también es habitual que se monitoricen con el fin de predecir sus averías. Cortocircuitos en el estator o averías en el rotor, como la aparición de excentricidad y balanceo o la rotura de barras y anillos son algunos de los fallos más comunes que pueden ser detectados mediante técnicas de mantenimiento predictivo. (Castelli, 2007).

Las instalaciones hidráulicas y neumáticas son también muy comunes en la industria, y muchas de estas instalaciones tienen una gran responsabilidad sobre el funcionamiento de la planta. De esta forma, se trata de monitorizar el sistema para detectar fugas y fallos en válvulas, evitando fugas de fluidos y pérdidas de vacío. *(Labaien, 2009).*

No obstante, cada máquina y cada sistema es diferente, y en cada uno de estos el componente crítico puede ser uno determinado, pero los anteriores recogen una gran cantidad de las roturas que se producen en máquinas y cuya rápida sustitución puede ahorrar mucho tiempo y dinero en un proceso de producción.

Identificados los componentes que se quieren monitorizar, la siguiente fase consiste en averiguar las causas del fallo de dichos componentes.

iii. Analizar las causas del fallo

Las roturas de los elementos mencionados anteriormente, en su mayoría, no son roturas repentinas, si no que sufren desgaste con el tiempo hasta que se rompen. Este desgaste se manifiesta en comportamientos anómalos que, a su vez, incrementan el desgaste del componente. Estos comportamientos anómalos son los que deben ser identificados y posteriormente monitorizados, para poder predecir el momento de la rotura.

Para detectar averías próximas en los elementos mecánicos mencionados, algunos de los efectos más comunes son los desalineamientos entre ejes, las excentricidades y balanceos o las holguras.
En las máquinas eléctricas, uno de los síntomas más significativos es la variación del consumo eléctrico respecto del consumo durante el funcionamiento normal.
En las instalaciones hidráulicas o neumáticas, los indicios de fallo se traducen en fugas de fluidos o aire, pérdidas de presión o fugas en tanques. Los aceites sucios o la falta de lubricación son síntomas de un mal funcionamiento en aquellas máquinas que requieren lubricación.

Las causas o síntomas de fallo anteriormente nombrados pueden agruparse en cinco grupos, en función de la manera en que se manifiestan:

- Vibraciones
- Variaciones de temperatura
- Generación de ultrasonidos
- Diversos efectos eléctricos
- Mal estado de los lubricantes

Conociendo de qué manera se manifiesta el fallo en el elemento crítico, la siguiente fase consiste en definir cuál es la variable física que se va a monitorizar y la técnica a utilizar para medirla.

iv. Definir la variable física a medir. Técnicas de análisis.

Para la detección de fallos se pueden aplicar diferentes técnicas. De hecho, un mismo fallo se puede detectar con diferentes técnicas. Esto depende principalmente de la forma en la que se manifiesta el fallo.

Siguiendo con la clasificación anterior de las causas de fallo, las cinco técnicas más utilizadas son:

- Análisis de vibraciones

  Actualmente es la técnica más empleada a la hora de implantar un plan de mantenimiento predictivo.

  Se aplica fundamentalmente en el estudio de máquinas rotativas. Consiste en analizar continuamente las vibraciones de la máquina durante su funcionamiento. Las vibraciones se pueden analizar tanto midiendo su amplitud como analizando su frecuencia, de forma que si la amplitud aumenta o la frecuencia varía se identifica un fallo. *(William Olarte C., 2010)*

  Las causas más comunes de fallo son los fallos en los acoplamientos. Estos se convierten en desequilibrios, desalineamientos o falta de apriete en las uniones. Todos ellos se manifiestan en vibraciones anómalas. En función del tipo de fallo, estás vibraciones serán radiales (perpendiculares al eje de rotación) o axiales (paralelas ala eje). *(Solar)*

- Análisis termográficos

  Consiste en una técnica que analiza la temperatura de las máquinas para determinar si están funcionando de manera correcta. Durante el funcionamiento normal de la máquina, la distribución de temperatura en esta sigue un patrón. Al producirse un fallo, y comenzar a funcionar de forma anómala, hay zonas de la máquina que varían su temperatura. (*William Olarte C., 2010*)

  Por lo general, esta técnica se utiliza para detectar y predecir fallos de tipo electromecánico (componentes mecánicos que se calientan debido a un rozamiento excesivo, intensidades eléctricas elevadas, etc.).

- Análisis por ultrasonidos

  Esta técnica estudia las ondas de sonido de alta frecuencia, producidas por las máquinas cuando presentan algún tipo de problema. Estas ondas tienen la capacidad de atenuarse muy rápido debido a su corta longitud, facilitando así la detección de la fuente que las produce. Este análisis permite detectar múltiples fallos de elementos, como el aumento de fricción en máquinas rotativas, fugas en válvulas y pérdidas de vacío, verificar la integridad de juntas de recintos estancos o detectar arco eléctrico. *(William Olarte C., 2010)*

- Análisis eléctricos o de corriente eléctrica

  La mayoría de los análisis eléctricos se realizan sobre motores. Son cuatro las principales averías que pueden sufrir los motores eléctricos; cortocircuito en el estator, rotura de las barras, excentricidades o rotura de rodamientos.

  Estos fallos pueden deberse a multitud de causas. Aunque algunos de ellos pueden ser detectados por análisis de vibraciones o termografía, analizando el espectro de la señal de la corriente del motor pueden predecirse todas estas averías. (Castelli, 2007)

  Esta técnica conocida como Análisis de Firma Eléctrica o MCSA (del inglés: Motor Current Signature Analisys), consiste en analizar la señal de las corrientes del estator mediante un sensor de efecto Hall. Se analizan las frecuencias y los componentes de la señal, y mediante técnicas de inteligencia artificial se detectan los fallos asociados a esas características de la señal. *(García Santamaría, 2017)*

- Análisis de lubricantes

  De todas las técnicas mencionadas, el análisis de lubricantes es la que mayor dificultad presenta a la hora de automatizar, ya que la mayoría de las pruebas deben realizarse en laboratorio. No obstante, la lubricación es muy importante en las máquinas ya que las protege del desgaste y controla su temperatura.
  Mediante esta técnica se cuantifica el grado de contaminación y desgaste del aceite. En función de estos dos parámetros se puede conocer el estado de la máquina y predecir así sus averías. *(William Olarte C., 2010).*

v. Instrumentos y sensores de medida

Una vez se ha realizado el estudio del sistema, y se ha determinado con que técnica de análisis van a identificarse los síntomas que indican que el componente va a fallar, se debe escoger el instrumento de medida con el que se realizarán las mediciones.

Hay dos tipos básicos de transductores para monitoreo del estado de vibración; dispositivos sísmicos que están normalmente montados en la estructura de la máquina y cuya salida es una medición de la vibración absoluta de la estructura, y los transductores de desplazamiento relativo, que miden el desplazamiento vibratorio y la posición media entre los elementos rotatorios y no rotatorios de la máquina. *(Zamora Guitérrez, 2016)*
Para el análisis de vibraciones se utilizan transductores de vibración. En función del rango de frecuencias de la vibración que son capaces de medir, se diferencian tres tipos:

- De desplazamiento relativo (0 – 1000 Hz). Los transductores de desplazamiento relativo miden el desplazamiento entre los elementos rotatorios y no rotatorios de la máquina. *(Zamora Guitérrez, 2016)*
- Sísmicos de velocidad (20 – 2000Hz). Los transductores sísmicos normalmente se montan en la estructura de la máquina y miden la vibración absoluta de la estructura.
- Piezoeléctricos de aceleración o velocidad (1 – 20000Hz). Estos acelerómetros o velocímetros miden la aceleración o velocidad a lo largo de los 3 ejes.

En los análisis termográficos, el instrumento utilizado es la cámara termográfica. Su funcionamiento consiste en captar la energía térmica que desprende un cuerpo. Esta energía es emitida desde la superficie del cuerpo, y se denomina radiación infrarroja. La cámara termográfica convierte la energía infrarroja proveniente del espectro infrarrojo, al espectro visual. Para poder realizar una medición continua, se utilizan programas de visión por computador y redes neuronales para detectar los cambios anómalos de temperatura que suponen síntomas de fallo en la máquina. *(Grijalba Davalillo, 2013)*

Además de la cámara termográfica, se utilizan diversos tipos de transductores para medir temperaturas concretas. Estos pueden ser eléctricos, como los termopares, mecánicos, que consisten en mecanismos de dilatación fundamentalmente o de radiación térmica.
El fundamento de esta técnica consiste en tomar medidas de temperatura en varios puntos, y detectar si en algún punto se está incrementando más la temperatura.

En el análisis por ultrasonidos, se utilizan dos tipos de transductores; de contacto o de inmersión.
Los transductores de contacto pueden ser de contacto directo o con línea de retardo. Los de contacto directo se emplean en contacto directo con la pieza y son los más sencillos de implementar. Existen de un solo elemento o de elemento doble, que son los que se utilizan con mayor frecuencia para superficies rugosas, como puede ser una tubería.
Los transductores con línea de retardo incorporan un cilindro (conocido como línea de retardo) entre el sensor y la pieza. De esta forma se consigue aislar el sensor de altas temperaturas o adaptarlo a la forma de la pieza.
Los transductores de inmersión están sumergidos en un fluido y no tienen contacto con la pieza. Son los más utilizados en sistemas de escaneo automático. *(De máquinas y herramientas, 2015)*

El análisis eléctrico, en su mayoría empleado para monitorizar motores eléctricos, se basa en el análisis de la corriente que consume el motor. En función del tipo de motor, existen diferentes equipos para medir corriente, voltaje u otros parámetros de funcionamiento. Dichos equipos están habilitados para almacenar o enviar los datos leídos al centro de datos donde serán analizados.

El análisis de lubricantes, a pesar de ser uno de los más utilizados, presenta el inconveniente de que en la mayoría de los casos las muestras de lubricantes deben ser analizadas en laboratorio, con instrumentación especializada. Algunos ejemplos pueden ser equipos para contaje de partículas, espectrómetros, máquinas de rayos X o equipos de análisis ferrográficos. *(Escuela Técnica Superior de Ingenieros Industriales, 2002)*

vi. Gestión de la información. Transmisión y almacenamiento de datos.

Todas las fases anteriores no tendrían ningún sentido si no se tratasen adecuadamente los datos obtenidos.
Los datos se pueden gestionar de dos maneras, en la propia máquina o transmitirlos a una oficina central, que puede estar en la misma planta o no.

Si los datos no son transmitidos, una vez son convertidos a una señal eléctrica por medio del instrumento de medida, se pueden manipular y transformar para mostrarlos en una pantalla, mediante un panel Scada, por ejemplo, de forma que el operario sea capaz de identificar los fallos. Este es el sistema tradicional y más rudimentario.

Cuando los datos son transmitidos, puede hacerse de dos maneras, a través de un bus de comunicaciones (Modbus, Canopen, Profibus, etc.) o de manera inalámbrica (WiFi, ZigBee, SigFox, Lora, Narrowband, etc.). *(Cendón, s.f.) (Hurtado)*

vii. Análisis e interpretación de los datos

Cuando los datos son recolectados, es necesario analizarlos e interpretarlos para obtener buenos resultados.

Es necesario recopilar y almacenar todos los datos de la máquina a lo largo del tiempo, así como los que suministra el fabricante, para poder compararlos con los datos medidos. Además, si se desea obtener datos con una gran precisión, pueden implementarse diversos tipos de algoritmos, basados en estos datos históricos y otros parámetros para poder interpretar de mejor manera los datos medidos.

III. DESARROLLO DE LA HERRAMIENTA DE MONITORIZACIÓN

i. Especificaciones del sistema

Se plantea el diseño de una herramienta de monitorización, que permita predecir la mayor cantidad de averías posibles, en la mayor variedad de máquinas posibles. De esta forma, no se enfoca en la predicción de un elemento en concreto para una máquina determinada, sino que se pretende que sea una herramienta versátil.

Se define la herramienta de forma que todos los sensores que incorpore puedan ser instalados en la máquina, para poder realizar la adquisición de datos continuamente y en tiempo real. Todos ellos deben poder ir conectados a una Raspberry Pi, donde se realizará el almacenamiento, procesado y análisis de los datos obtenidos.

El uso de la Raspberry Pi está justificado en la distribución de la monitorización de diferentes máquinas, es decir, cada máquina es monitorizada por una sola Raspberry Pi y sus correspondientes sensores. De esta forma, cada Raspberry Pi se encarga de la adquisición y análisis de los datos de cada máquina. Estos datos pueden ser visualizados en el mismo puesto de trabajo y/o enviados de forma inalámbrica a un centro único donde pueden controlarse todas las Raspberry de la planta. Se evita así saturar la red de comunicaciones y el almacenamiento de datos centralizado.

Analizando estos requisitos y las conclusiones obtenidas en el estudio del arte, se decide que la herramienta debe incluir las siguientes técnicas de monitorización:

- Análisis de vibraciones
- Análisis de consumo eléctrico
- Análisis de temperatura
- Análisis de ultrasonidos

El análisis de vibraciones y el análisis de ultrasonidos permiten detectar las anomalías de funcionamiento de aquellas partes mecánicas de máquinas rotativas. La predicción de la rotura de este tipo de elementos es fundamental, ya que la mayoría de las máquinas o sistemas incorporan este tipo de elementos.

El análisis de temperaturas, entre otras anomalías, permite detectar los fallos en mecanismos de transmisión de potencia. Para ello se analiza el gradiente de temperatura entre la máquina y el elemento en cuestión.

Con el análisis de consumo eléctrico se pueden detectar fallos de funcionamiento en máquinas eléctricas, tales como motores y transformadores.

Un ejemplo de aplicación podría ser una punzonadora de chapa metálica. Incorporando el acelerómetro en su cabezal, podrían verse las aceleraciones que experimenta este a lo largo de los tres ejes. Cuando las vibraciones leídas son anómalas, indica que alguno de los mecanismos encargados del movimiento del cabezal está comenzando a fallar. De esta forma se localiza la avería sin necesidad de que el fallo se traduzca en piezas defectuosas, y puede planificarse su reparación sin necesidad de afectar a la producción.

El análisis de ultrasonidos puede utilizarse para identificar fallos en los rodillos que hacen avanzar las chapas. Al avanzar los rodillos emiten un sonido con una frecuencia específica, que varía cuando alguno de los rodillos o de las partes rotativas se desalinea o comienza a fallar.

Se puede medir la temperatura en diferentes puntos de la máquina encargados de la transmisión de potencia, como puede ser en la salida del motor que acciona los rodillos anteriormente nombrados. Cuando aquí se produce un aumento de temperatura elevado respecto al resto de la máquina, significa que algún elemento está comenzando a fallar, por ejemplo, un rodamiento, lo que produce mayor rodamiento aumentando la temperatura.

Siguiendo el ejemplo, analizando la corriente de entrada a los motores de la máquina, se puede ver el consumo de electricidad requerido, de forma que si este varía respecto al nominal nos indica que algún elemento está fallando. Del mismo modo, puede analizarse la frecuencia de la corriente, detectando los cambios respecto de la frecuencia normal de funcionamiento.

ii. Arquitectura del sistema

En base a las especificaciones anteriores, se diseña la arquitectura del sistema.

Para realizar los análisis especificados, la adquisición de datos se realizará con los siguientes sensores:

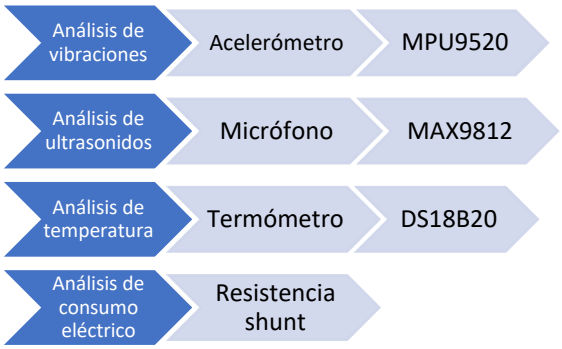

Ilustración 3 - Esquema sensores utilizados

El análisis de vibraciones se realiza con un acelerómetro digital; un transductor de vibración piezoeléctrico. El sensor escogido es un MPU9250, una unidad inercial de medición, compuesta por un acelerómetro, un magnetómetro y un giroscopio. Para la medición de la vibración solo se utilizará el acelerómetro, que se conectará con la Raspberry Pi mediante bus i2c.

Se trata de un acelerómetro de 3 ejes de bajo consumo, con un rango de medida seleccionable de ±2g, ±4g, ±8g y ±16g. El sensor incorpora un conversor analógico digital de 16 bit que permite tener una gran precisión en la medida (16384 LSB/g, 8192 LSB/g, 4096 LSB/g y 2048 LSB/g en función del rango de escala escogido.

Existen multitud de acelerómetros digitales en el mercado. La comunicación por bus i2c que permite conectar directamente con las Raspberrry Pi y el rango de escala configurable

son las principales razones por las que se ha escogido este modelo. La cuantificación con 16 bits es otro punto a favor, ya que proporciona una gran resolución en la medida.

El MAX9812 es un amplificador de sonido de bajo consumo, con una ganancia fija de 20dB. Posee un ancho de banda de 500kHz, y una baja distorsión de señal (0.015%THD), ideal para ambientes ruidosos.

El inconveniente de este sensor es la carencia de conversor analógico-digital, pero la baja distorsión de señal y el ancho de banda de 500kHz son ideales para ambientes industriales. El amplificador de 20dB de ganancia fija es también una de las razones por las que se ha escogido, ya que la amplitud de las bajas frecuencias es menor y por lo tanto más difícil de detectar.

El termómetro seleccionado es un DS18B20. Se trata de un sensor de temperatura que adquirir en formato de sonda impermeable, ideal para introducirlo en ambientes industriales. Funciona con protocolo 1-wire, lo que facilita el cableado y permite conectarlo directamente con la Raspberry Pi, una de las razones por las que ha sido seleccionado. El rango de medida va desde los -55ºC hasta los 125ºC, un rango suficientemente amplio.

El consumo eléctrico se mide directamente utilizando un conversor analógico-digital, y utilizando una resistencia de shunt. En función de la tensión de alimentación de la máquina este montaje variará, pudiendo ser necesario un divisor resistivo para adecuar los valores de tensión de entrada al conversor.

Tanto el acelerómetro escogido como el termómetro pueden ser conectados directamente con la plataforma escogida (Raspberry Pi), pero el micrófono y la resistencia de shunt proporcionan medidas analógicas, de forma que deben ser tratadas previamente con un conversor analógico-digital. Se utiliza como conversor un Arduino, que será el encargado de procesar y transmitir los datos obtenidos a la Raspberry Pi en formato digital.

Una vez se realiza la adquisición de datos, se almacenan en una base de datos. Se ha utilizado para ello PostgreSQL. Cada sensor tiene su propia base de datos, alojadas en el mismo servidor, de forma que pueden trabajar de forma independiente, ya que no en todos los casos se utilizarán todos los sensores.

Estos datos se procesan para obtener los datos de interés, que serán analizados junto a los datos históricos y los valores de consigna establecidos, con el fin de predecir las averías en los elementos. Los resultados obtenidos se representarán de forma gráfica, bien en la propia máquina o en otros dispositivos de forma remota.

De esta forma, en la propia Raspberry Pi, ubicada en la máquina, se toman las medidas de las variables a monitorizar, se procesan y se analizan. Esto ofrece una gran ventaja en caso de ser elevado el número de máquinas a monitorizar, ya que cada Raspberry se encarga de monitorizar una máquina, evitando así saturar una red de comunicaciones de datos.

La siguiente imagen recoge la arquitectura del sistema:

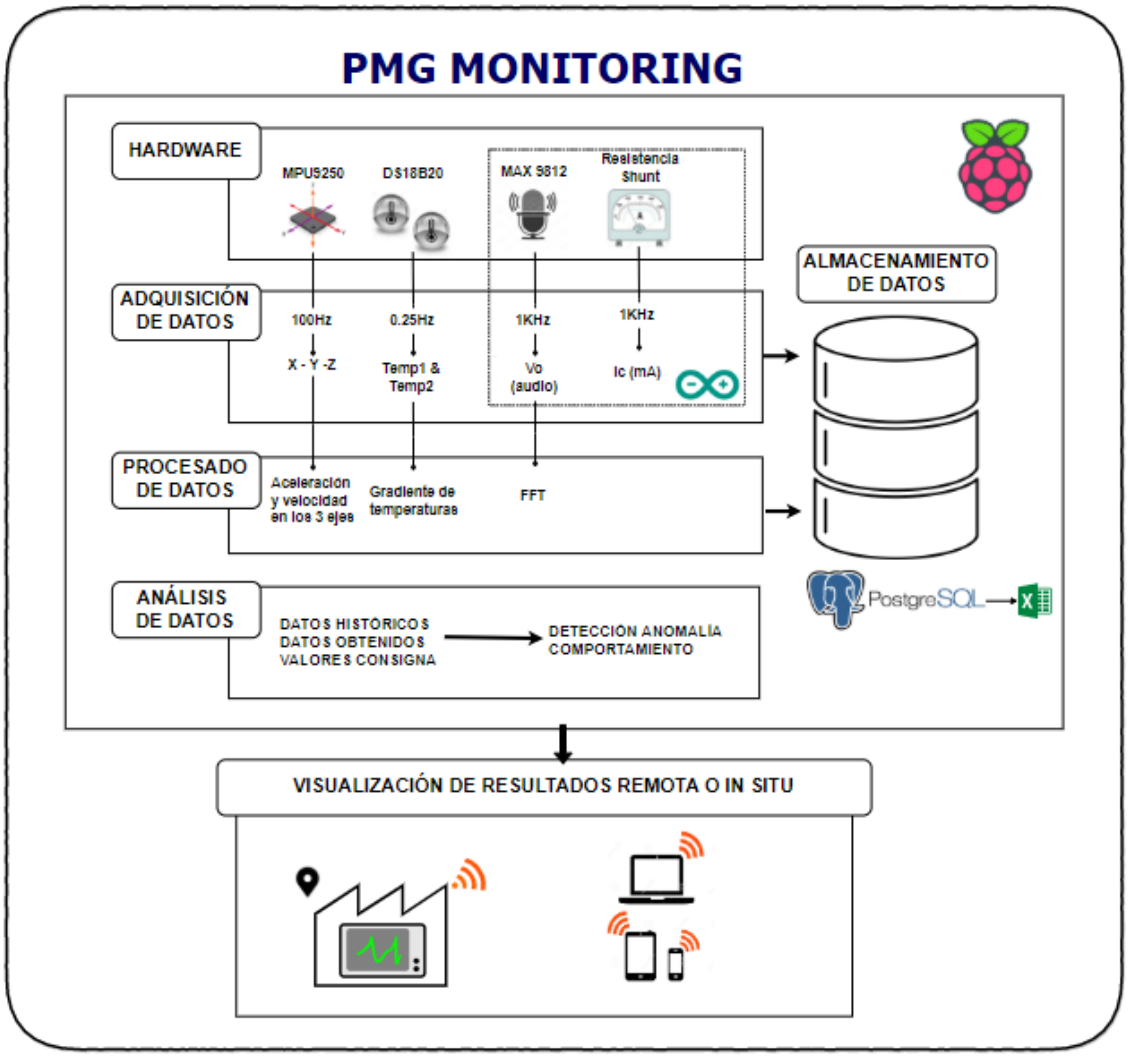

Ilustración 4 - Arquitectura del sistema

iii. Firmware

El firmware se ha desarrollado en lenguaje Python, y se ejecuta en una Raspberry Pi 3 Model B+ con sistema operativo Raspbian.

La estructura del programa se divide en varios subprogramas. Cada uno de estos tiene una tarea que realizar, y son lanzados en diferentes hilos controlados por el programa principal.

Para la adquisición y procesado de los valores provenientes de sensores analógicos se ha utilizado un Arduino.

El Arduino tiene doble función, en primer lugar, lee el valor de la variable a monitorizar, lo convierte a señal digital, y la procesa y acondiciona. En segundo lugar, se encarga de enviar el valor de esa señal a la Raspberry Pi, la cual se encargará de almacenar los valores en la base de datos para poder analizarlos.

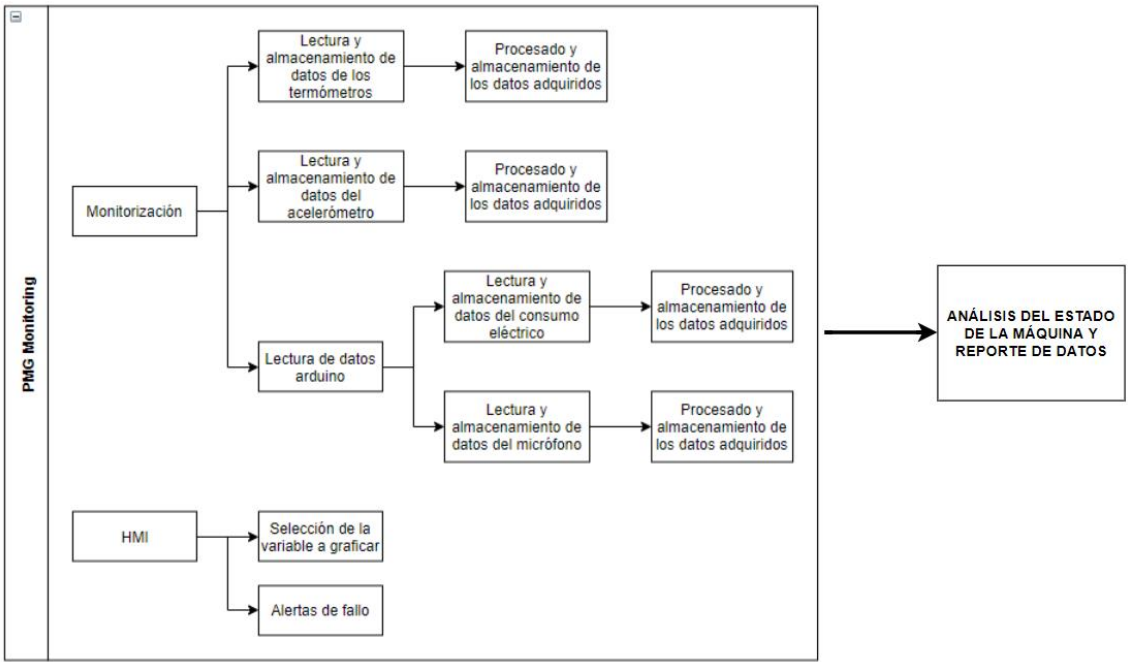

Ilustración 5 - Esquema firmware

iv. Sensorización

III.iv.1 Acelerómetro

La configuración inicial del acelerómetro depende del entorno dónde se vaya a instalar. En el Anexo I se explica la configuración escogida y los parámetros que pueden variar y cómo según el entorno de trabajo.

Para el análisis de la vibración, se debe obtener el desplazamiento en mm/s a lo largo de los tres ejes.

El valor de la velocidad en cada eje, en mm/s, es el que se utiliza para conocer el estado de la máquina. En la siguiente imagen, se observa la gráfica de la velocidad (mm/s) a lo largo del eje Z en función del tiempo (s).

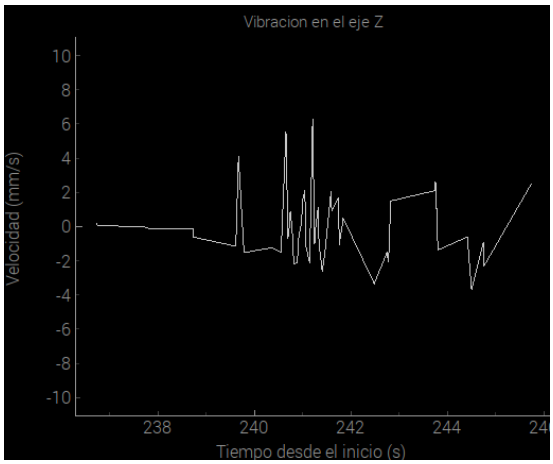

Ilustración 6 - Vibración a lo largo del eje Z

Todos los valores obtenidos son almacenados en una base de datos, alojada en la Raspberry Pi, dónde se forma un histórico de los datos adquiridos, que ayuda a la identificación de problemas.

La siguiente imagen muestra el diagrama de flujo de la función de adquisición de datos:

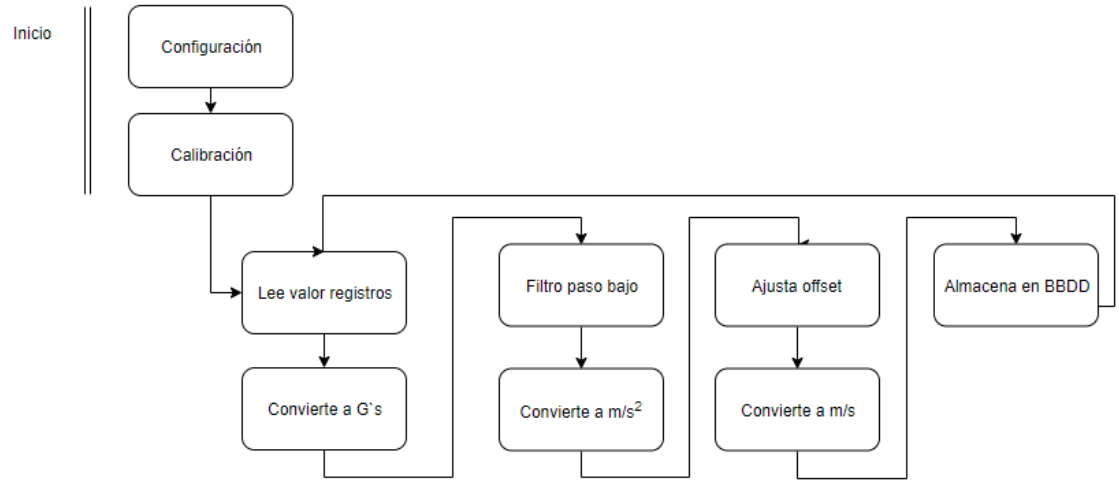

Ilustración 7 - Diagrama de flujo acelerómetro

Al arrancar la aplicación, lo primero que se hace es configurar los parámetros del acelerómetro, y este se auto calibra.

El bucle de adquisición de datos comienza leyendo el valor de los registros. Por cada eje se tienen dos registros de 8 bits, los cuales se unen en un solo registro. Este registro de 16 bits va desde 0 hasta 65535, proporcionando datos de aceleración en unidades LSB/g. En función del rango de medida, se tiene una relación diferente LSB/g. Con esta relación, se obtiene la lectura del acelerómetro en unidades de fuerzas G.

Los acelerómetros se caracterizan por tener una repetibilidad baja, de forma que se aplica un filtro paso bajo mediante software para estabilizar la lectura leída y garantizar una mayor repetibilidad. Consiste en que la nueva medida tenga un porcentaje en peso del valor leído y el resto del porcentaje sea del valor anterior.

El siguiente paso es convertir a unidades del sistema internacional. Para ello se multiplica la medida obtenida por el valor de la fuerza de la gravedad terrestre[1], y se le resta el offset obtenido durante la calibración (obtenido en m/s^2).

De esta forma se tiene a lo largo de los tres ejes el valor de la aceleración. Integrando la aceleración respecto del tiempo, se obtiene la velocidad de desplazamiento en cada eje. La frecuencia de medición es de 100hz.

[1] Este valor se toma como constante (9.80665), pero hay que tener en cuenta que varía en función de la altura y la latitud.

Por último, se guarda en la base de datos el valor leído en bits, en fuerzas G, en mm/s y el instante de tiempo de la medida.

Se han utilizado diferentes módulos de Python para la adquisición de datos; *smbus* para las comunicaciones, *time* para el tiempo, *numpy* para trabajar con vectores, *RPi.GPIO* para poder utilizar las GPIO de la Raspberry Pi, *math* para poder realizar operaciones matemáticas y *psicopg2* para poder trabajar con PostgreSQL.

En el Anexo I se encuentra el código Python utilizado para la adquisición de datos.

III.iv.2 Micrófono

Se utiliza un micrófono MAX9812, el cuál es gestionado por el Arduino. Mediante comunicación por el puerto serie, el Arduino envía los datos a la Raspberry Pi a una frecuencia de un hertzio. Los datos a enviar son el valor de la amplitud de la medida, en decibelios y la frecuencia del audio en hertzios.

El diagrama de flujo de la adquisición de datos con el micrófono es el siguiente:

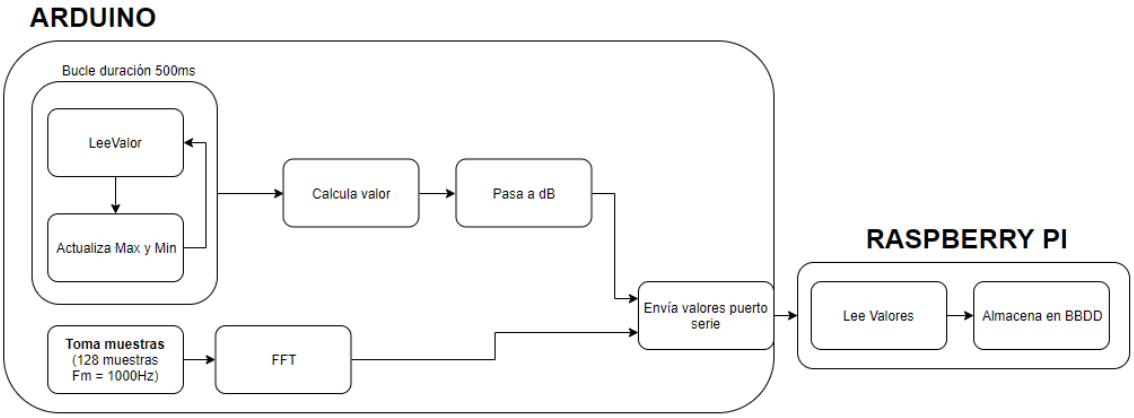

Ilustración 8 - Diagrama de flujo MAX9812

Para obtener la amplitud del sonido, se toman muestras del sonido durante una ventana de tiempo de 500ms. Se guardan los valores máximo y mínimo obtenidos durante esta ventana, y a partir de la diferencia entre estos se obtiene el valor de la amplitud de la muestra. El valor obtenido se procesa para obtener el valor en decibelios.

Para calcular la frecuencia del sonido se utiliza la librería arduinoFFT.h. Esta librería realiza la transformada rápida de Fourier de las muestras tomadas, y devuelve la frecuencia. Se toman 128 muestras con una frecuencia de muestreo de 1000Hz, se procesan las muestras y se obtiene la transformada de Fourier de la señal.

Los datos procesados se envían a la Raspberry Pi por el puerto serie. La Raspberry Pi se encarga de leer los datos y almacenarlos en la base de datos (amplitud actual, frecuencia y tiempo). Se han utilizado los módulos *time, psicopg2* y *serial* (para la comunicación por puerto serie) de Python.

En la siguiente imagen, se observa la frecuencia representada en función del tiempo. Se tiene una frecuencia de 500Hz, interrumpida durante un instante de tiempo:

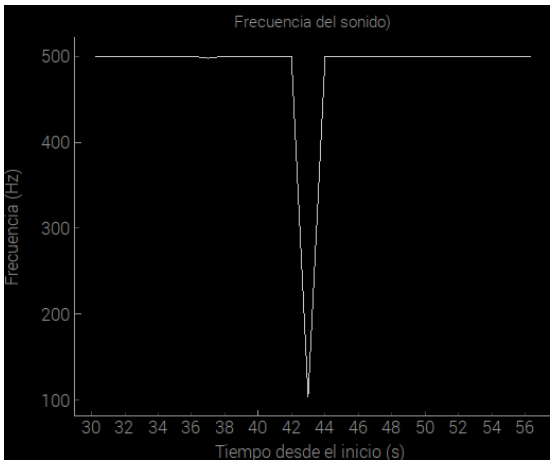

Ilustración 9 - Frecuencia del sonido en función del tiempo

En el anexo III encuentra el código Python y el código de Arduino utilizado.

III.iv.3 Termómetro

La temperatura es una de las variables que más información proporciona sobre el funcionamiento de una máquina. Para su análisis, se ha optado por utilizar dos termómetros digitales, uno colocado en la carcasa o envolvente de la máquina y otro en el elemento a monitorizar. De esta forma, si el gradiente entre ambas temperaturas se sale del rango habitual, el elemento en cuestión empieza a funcionar en condiciones anómalas.

La frecuencia de medición es de 0,25Hz para cada termómetro, ya que las variaciones de temperatura no suceden de forma rápida.

El termómetro utilizado es un ds18b20. Necesita alimentarse a 5V y solo utiliza un pin digital de la Raspberry Pi con una resistencia de pull-up, tal y como se muestra en la figura:

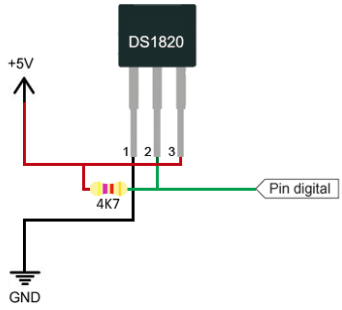

Ilustración 10 - Conexión DS1820

El propio sensor, devuelve el valor de temperatura en ºmC. En el Anexo II se explica el código utilizado para la adquisición de temperaturas.

III.iv.4 Resistencia shunt

Uno de los parámetros que más información proporciona sobre el funcionamiento de una máquina eléctrica es su consumo de corriente. Existen técnicas, como la MCSA (del

inglés Measurement Current Signal Analysis) capaces de detectar mediante el consumo eléctrico la gran mayoría de averías en un motor eléctrico. (Castelli, 2007)

Para otras máquinas que no sean motores eléctricos, también se puede extraer información de su consumo. Utilizando una resistencia shunt, que básicamente es una resistencia colocada en serie con la carga, podemos conocer la intensidad de corriente que está consumiendo la carga. *(Nieto Vilardell)*

La siguiente imagen muestra el esquema eléctrico de la resistencia shunt empleada:

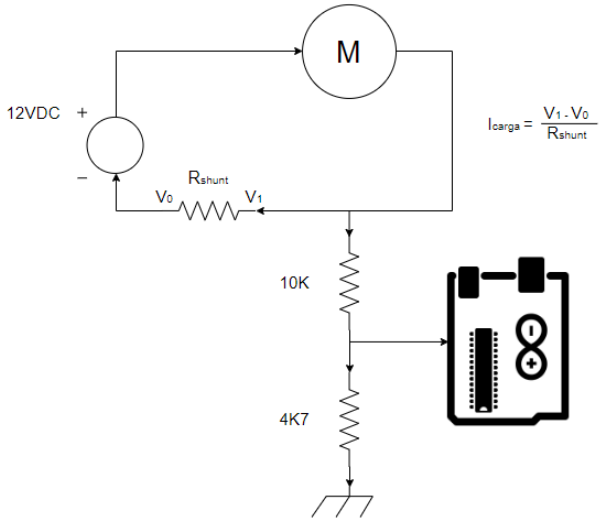

Ilustración 11 - Esquema eléctrico Rshunt

De esta forma, conociendo el valor de la resistencia de shunt, y midiendo el voltaje en V_1 con el Arduino, se puede conocer el valor de la intensidad de corriente.

v. Análisis de los datos

El objetivo de toda la monitorización llevada a cabo es la detección de fallos de manera predictiva. Para ello, los datos obtenidos deben ser analizados, mediante técnicas de machine learning. Estas técnicas utilizan los datos históricos de la máquina, así como los datos proporcionados por el fabricante, y los datos obtenidos durante la monitorización para poder predecir la avería. Los datos, introducidos en redes neuronales, son procesados para generar las alertas. Estás técnicas, se escapan al alcance del proyecto.

No obstante, existen otras técnicas más clásicas, para poder conocer el estado de la máquina y ayudar a la detección de fallos.

- **Vibraciones**: en primer lugar, hay que tener en cuenta que todas las máquinas vibran (máquinas similares, en condiciones normales, vibran del mismo modo). Un cambio en la vibración básica indica que se está produciendo una anomalía, de forma que diferentes fallos dan diferentes vibraciones. *(Solar)*
  Para la detección de la vibración anómala, se utiliza el método de la vibración severa. La norma ISO 10816-1 recoge los niveles de vibración admitidos según el tipo de máquina:

- Grupo K: motores eléctricos hasta 15kW
- Grupo M: motores eléctricos de 15kW hasta 75kW
- Grupo G: grandes motores
- Grupo T: turbomáquinas

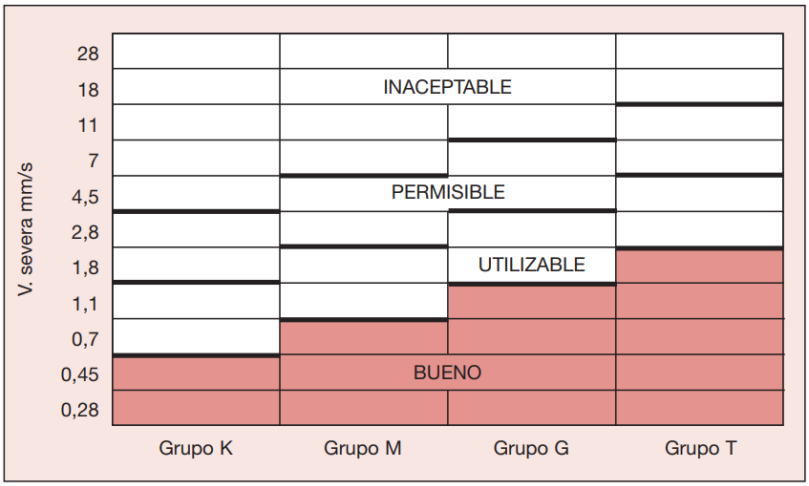

Ilustración 12- Rango de vibraciones admisibles

De esta forma, en función del tipo de máquina a monitorizar, los valores consigna a establecer serán mayores o menores.

Valores de vibración en rango *bueno* o *utilizable* son admisibles. Por el contrario, valores continuados en el tiempo, dentro del rango *permisible*, o medidas en rango *inaceptable* harán que el sistema alerte de vibraciones anómalas. Cuando se producen medidas continuadas en rango *inaceptable* el sistema deberá detener la máquina para evitar daños mayores.

Existen máquinas que no se ajustan a este criterio, de forma que se ha implementado un módulo *(MPU9250.analysisData)* que permite obtener los valores normales de funcionamiento. Se toman 25000 medidas de velocidad y aceleración a lo largo de los tres ejes, con la máquina funcionando correctamente. Se obtiene el valor medio de estas medidas, y el valor máximo y mínimo. Estos valores se utilizan como valores consigna.

Durante la monitorización, se cogen las 100 últimas medidas tomadas por el acelerómetro, se calcula la media y se compara con los valores medios establecidos anteriormente. Se utilizan dos criterios para indicar una alerta; el primero, si el valor medio de las 100 medidas se aleja de la media nominal más de un 50% del valor de esta. El segundo, si el valor máximo de la amplitud es superado en algún momento.

- **Temperatura:** en la herramienta desarrollada, el análisis de la temperatura se va a realizar mediante una de las técnicas más clásicas. Se utilizan dos termómetros, uno para medir la temperatura exterior de la máquina a

monitorizar, y otro para medir la temperatura de los elementos críticos (elementos de transmisión de potencia, ejes de motores, etc).

La técnica consiste en analizar la diferencia de temperatura entre ambas partes de la máquina. Una sola medida no tendría sentido, ya que podría variar debido a la temperatura ambiente. Se utiliza la diferencia entre ambas, de manera que, si el elemento crítico incrementa su temperatura respecto a la de la carcasa de la máquina, quiere decir que está trabajando de forma incorrecta. En máquinas rotativas o de transmisión de potencia, la mayoría de los fallos se traducen en rozamiento, que implica la pérdida de energía en forma de calor, que podrá ser detectado por este sistema.
La siguiente imagen muestra la gráfica del gradiente de temperaturas, en función del tiempo, mientras uno de los dos termómetros está siendo calentado:

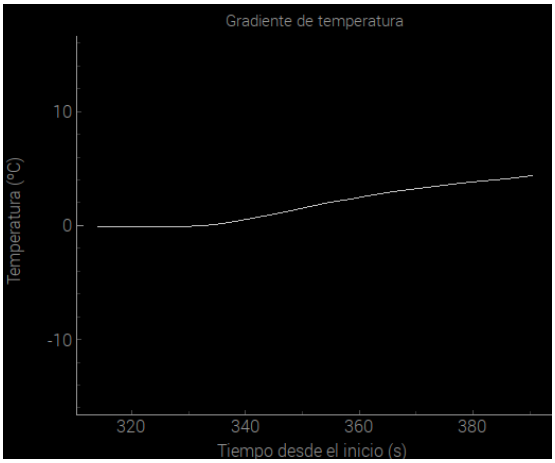

Ilustración 13 - Gradiente de temperatura en función del tiempo

Se pueden dar dos opciones. Una diferencia elevada de temperaturas, que supone un calentamiento extremo en el elemento en cuestión, lo cual implica detener la máquina, ya que es probable que el elemento esté a punto de romper y pueda provocar la rotura de otros elementos. Otra opción es un calentamiento progresivo, superior a un cierto valor consigna. En este caso se señalizará al encargado de la máquina el incremento de temperatura, para que pueda planificar la parada de esta y realizar las labores oportunas.

- **Sonido:** las máquinas rotativas, en condiciones normales de funcionamiento, tiene un sonido característico, de manera que, máquinas similares en condiciones similares, suenan igual.
De esta forma, la amplitud y la frecuencia del sonido de una máquina son característicos, y mientras la máquina se encuentre trabajando de manera correcta estos no variarán.
Cuando la máquina presenta algún problema, tanto la frecuencia como la amplitud del sonido se modifican, y son indicadores de que algo no funciona como debería.

La herramienta analiza tanto la amplitud como la frecuencia. Se debe conocer la frecuencia que emite la máquina durante su comportamiento normal, y cuando presenta una frecuencia diferente, es un indicativo de que las partes en rotación de la máquina están funcionando incorrectamente. Del mismo modo, un incremento en la amplitud puede indicar un aumento en el rozamiento, pero debe ser prolongado y periódico, ya que la contaminación acústica del entorno puede falsear la medición de la amplitud.

Del mismo modo que para la vibración, se ha desarrollado un módulo para obtener los valores de funcionamiento normal. Este módulo devuelve los valores máximo, mínimo y medio de la amplitud y de la frecuencia del sonido emitido por la máquina.

- **Consumo eléctrico:**

Una gran mayoría de los fallos en máquinas se deben al funcionamiento anormal del motor principal. Son muchos los parámetros que permiten extraer información del funcionamiento de este (par motor, vibración, temperatura, …), pero recientemente la mayoría de los estudios se han centrado en el monitoreo de la parte eléctrica del motor, ya que esta es una técnica no invasiva.

Todas las máquinas que funcionan con electricidad tienen un consumo nominal de corriente. Cuando la máquina comienza a dañarse, la corriente se altera. Esta alteración en la corriente eléctrica es la que se va a utilizar como indicador de que algo no va como debería.

Los mayores indicadores son la amplitud de la corriente, es decir, el consumo de la máquina, que se verá modificado respecto del nominal, y la corriente analizada en el espectro de la frecuencia. En funcionamiento normal, la corriente eléctrica de los motores tiene una frecuencia principal. Al deteriorarse alguno de los elementos del motor, además de la frecuencia principal aparecen otras frecuencias, que indican el fallo. (Castelli, 2007)

Se ha implementado un módulo para la obtención de los valores máximo, mínimo y medio de la amplitud y de la frecuencia. Estos valores son los que se utilizarán para verificar que el consumo de la corriente eléctrica se encuentra en condiciones normales.

vi. Visualización de datos

Para el desarrollo de la interfaz gráfica se ha utilizado el software PyQt, junto a su aplicación QtDesigner.

Desde la aplicación se diseña el interfaz, se definen los botones, textos, etc. Cuando la interfaz está diseñada, se pueden modificar sus elementos desde un script de Python. Para ello es necesario utilizar varios módulos (*pyqtgraph*, *PyQt4* y *PyQt4.Gui)* con los cuales se pueden crear funciones asociadas a botones, graficar valores leídos desde la base de datos, etc. En el anexo V se encuentra el código Python utilizado.

La aplicación se compone de una sola ventana, dónde se encuentra un panel de información, un espacio dedicado a graficar las variables monitorizadas y un selector de la variable que se desea visualizar. Las variables disponibles son:

- Velocidad de la vibración en mm/s
- Amplitud del sonido en dB
- Frecuencia del sonido en Hz
- Temperatura
- Consumo eléctrico en mA
- Frecuencia de la corriente eléctrica

Tras abrir la aplicación, si se pulsa el botón *Iniciar*, comienza la monitorización. En primer lugar, se realiza la calibración del acelerómetro, y una vez ha sido calibrado comienzan a medirse valores con todos sensores.

En el lateral izquierdo se encuentra un selector de la variable a visualizar. Si se escoge una de las diferentes variables a visualizar y se pulsar sobre *visualizar* comienza a graficarse la variable seleccionada en función del tiempo. En la siguiente imagen, se visualiza la frecuencia del sonido en función del tiempo:

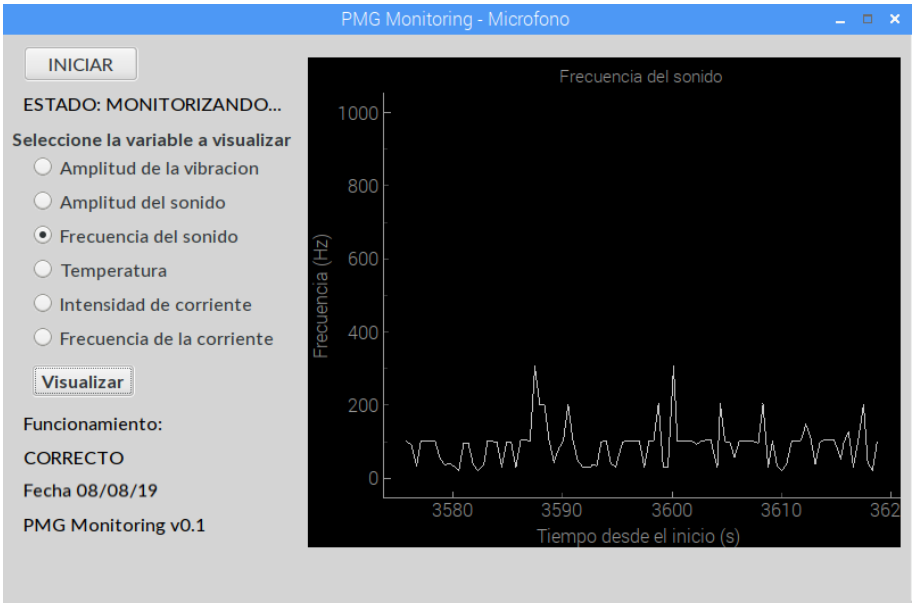

Ilustración 14 - Gráfica frecuencia sonido

Si durante la visualización de una determinada variable, se detecta un valor anómalo en las medidas realizadas por alguno de los sensores, aparece un mensaje de alerta en pantalla, comenzándose a visualizar la variable en cuestión.

En la siguiente imagen se observa el mensaje de alerta provocado por un exceso de temperatura, mientras se visualizaba la intensidad de la corriente.

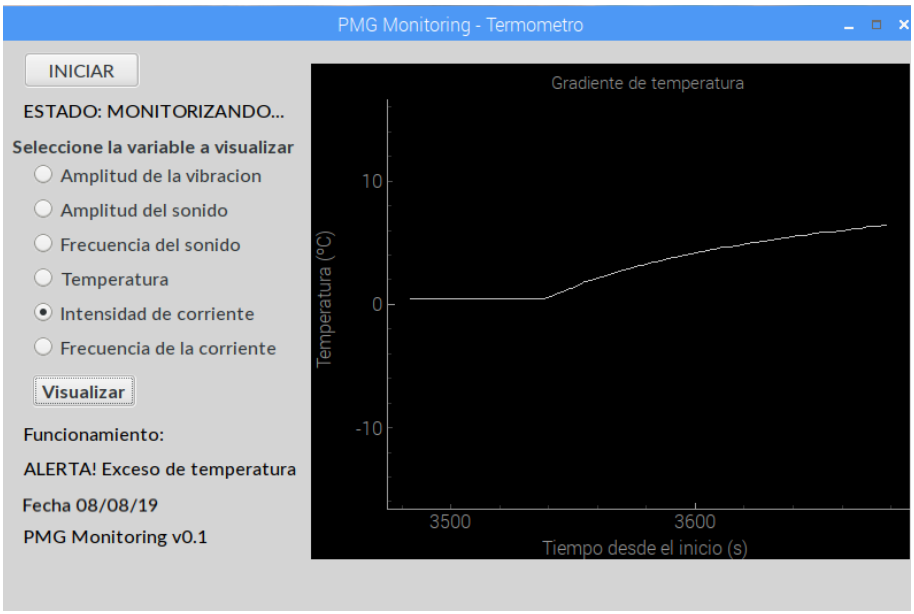

Ilustración 15 - Mensaje de alerta

vii. Resultados

Para poder probar el funcionamiento de la herramienta, se ha utilizado un compresor de aire portátil. Se ha elegido un compresor porque es útil para poder probar todos los sensores de la herramienta; en funcionamiento normal vibra con amplitud determinada, emite un sonido característico y tiene un consumo nominal de corriente. Además, la temperatura interior se ve incrementada con un tiempo de funcionamiento excesivo.

El micrófono y el acelerómetro se han colocado sobre una pieza de madera, acoplada a la parte superior del compresor, de forma que las mediciones siempre se realicen en las mismas condiciones.

El siguiente esquema muestra de forma simplificada el montaje que se ha realizado.

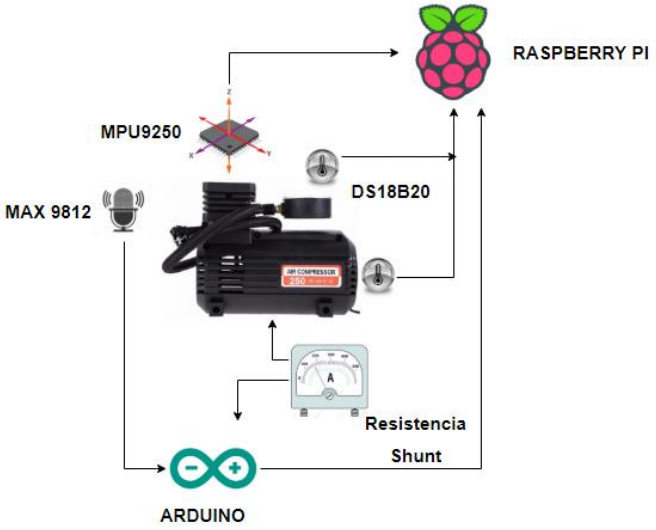

Ilustración 16 - Esquema montaje

Dada la falta de datos previos, se han tomado medidas durante un tiempo determinado con cada sensor, con el compresor funcionando en vacío, con el fin de obtener sus valores medios de funcionamiento en situación normal.

A continuación, se explican con mayor detalle los montajes para la monitorización, así como los resultados obtenidos para cada sensor.

- **Acelerómetro**

El acelerómetro se ha acoplado sobre la parte superior del compresor. Para reducir el coste computacional, únicamente se monitoriza la vibración a lo largo de un eje, en este caso el eje Z, aunque la herramienta está preparada para monitorizar cualquiera de los 3 ejes.
Los datos leídos medidos por el acelerómetro son procesados para obtener la aceleración y la velocidad de la vibración, en m/s^2 y mm/s respectivamente.

Como se indica en el apartado de análisis, se han obtenido los valores normales de vibración, los cuales se utilizarán para comprobar el estado del funcionamiento. Se realizan 25000 mediciones con el compresor funcionando en vacío para obtener el valor medio de aceleración y velocidad, y los valores máximo y mínimo respectivamente. Los valores obtenidos son los siguientes:

	Aceleración (m/s^2)	**Velocidad (mm/s)**
Valor mínimo	-4.0709	-2.7088×10^5
Valor medio	9.4384	1.1839×10^3
Valor máximo	25.3727	2.9731×10^7

Con estos valores se comparan los obtenidos durante la monitorización, y en base a ellos se determina el estado del funcionamiento de la máquina.

Durante la monitorización, se observa que los valores se suavizan con el tiempo gracias al filtro paso bajo, de forma que se evita el problema de la baja repetibilidad del sensor.
Cuando el compresor es apagado, cae la amplitud de la vibración, de forma que se detecta la variación, alertando del problema. Del mismo modo, cuando el compresor es forzado a suministrar aire a presiones elevadas, la amplitud de la vibración se ve afectada, la herramienta detecta este evento y alerta de ello.

- **Termómetro**

La temperatura es un parámetro de funcionamiento del compresor que varía de forma lenta con el tiempo. Se ha monitorizado la variación de la temperatura interior del compresor durante 10 minutos, haciendo trabajar al compresor con carga (1.5 bar aproximadamente).
La variación de la temperatura interior respecto de la exterior ha sido de 7.625 grados.
Se ha establecido este gradiente de temperaturas como valor límite de funcionamiento en el módulo de análisis, de forma que un gradiente superior

a este valor consigna enviaría un mensaje de alerta indicando del exceso de temperatura interior.
Para poder probar la efectividad de los sensores en la herramienta, se ha realizado un montaje paralelo, forzando a variar la temperatura de manera más rápida.

Con una resolución de 0,125 ºC y una frecuencia de muestreo de 0,25Hz, la monitorización de la temperatura es apta para el correcto funcionamiento de la herramienta.

- **Micrófono**

Del mismo modo que el acelerómetro, el micrófono ha sido colocado en la parte superior del compresor, de forma que ni la amplitud ni la frecuencia del sonido puedan variar por la posición del micrófono respecto de la fuente del sonido.
Durante la fase de caracterización del compresor, se obtuvieron los siguientes valores para la amplitud y frecuencia del sonido:

	Frecuencia (Hz)	**Amplitud (dB)**
Valor máximo	394.42	59.25
Valor medio	192.428	58.687
Valor mínimo	25.24	58.33

Si se analizan los valores de la frecuencia, se observa una gran diferencia entre el valor máximo y mínimo.
En el análisis de los datos obtenidos durante la monitorización, se debe obtener cada cierto tiempo el valor medio de la frecuencia, para compararlo con el valor medio calculado. En el caso del compresor, tanto frecuencia como amplitud variarán en función de la presión de carga que se le solicite al mismo, por lo tanto, el valor medio no tiene gran relevancia. Por el contrario, el valor máximo de frecuencia sí que es un buen indicador, ya que si este se supera indica que el compresor no está funcionando como debería.
Respecto a los valores de amplitud, se puede apreciar que no hay gran diferencia entre el valor máximo y mínimo. Esto indica que, independientemente de la carga del compresor, grandes diferencias entre el valor máximo y mínimo no son admisibles. Además, los decibelios se miden en escala logarítmica, de forma que valores muy alejados de este valor medio obtenido, tanto superiores como inferiores, indicarán fallo.
El módulo de análisis desarrollado se ha configurado para que detecte varios tipos de alerta. Tanto la frecuencia como la amplitud media de las 100 medidas se comparan con los valores medios establecidos como consigna, de forma que si la media se aleja del valor consigna más de un 50% del valor de este, salta la alerta. Del mismo modo, también se analizan los valores máximos medidos, de forma que si superan el valor máximo permitido salta la alerta.

- **Resistencia**

Como se ha explicado en el punto vii, se ha utilizado una resistencia de shunt para medir el consumo eléctrico. Se ha comprobado que en vacío el consumo medio es de 1.6A, y la frecuencia de la corriente es 51.2 Hz. El consumo máximo del compresor, según especificaciones, es de 3A. Se ha forzado al compresor a suministrar una mayor presión de salida, incrementando su consumo, pero sin exceder nunca el valor límite.

	Frecuencia (Hz)	**Amplitud (A)**
Valor máximo	52.0	1.71
Valor medio	51.2	1.591
Valor mínimo	49.0	1.43

Durante la monitorización, el análisis realizado compara el valor máximo y mínimo de la amplitud del consumo, de forma que se la amplitud de la corriente sale de ese rango, se notificará bien del exceso de consumo o bien del consumo menor de lo normal.
En la siguiente imagen se observa la frecuencia de la corriente representada frente al tiempo:

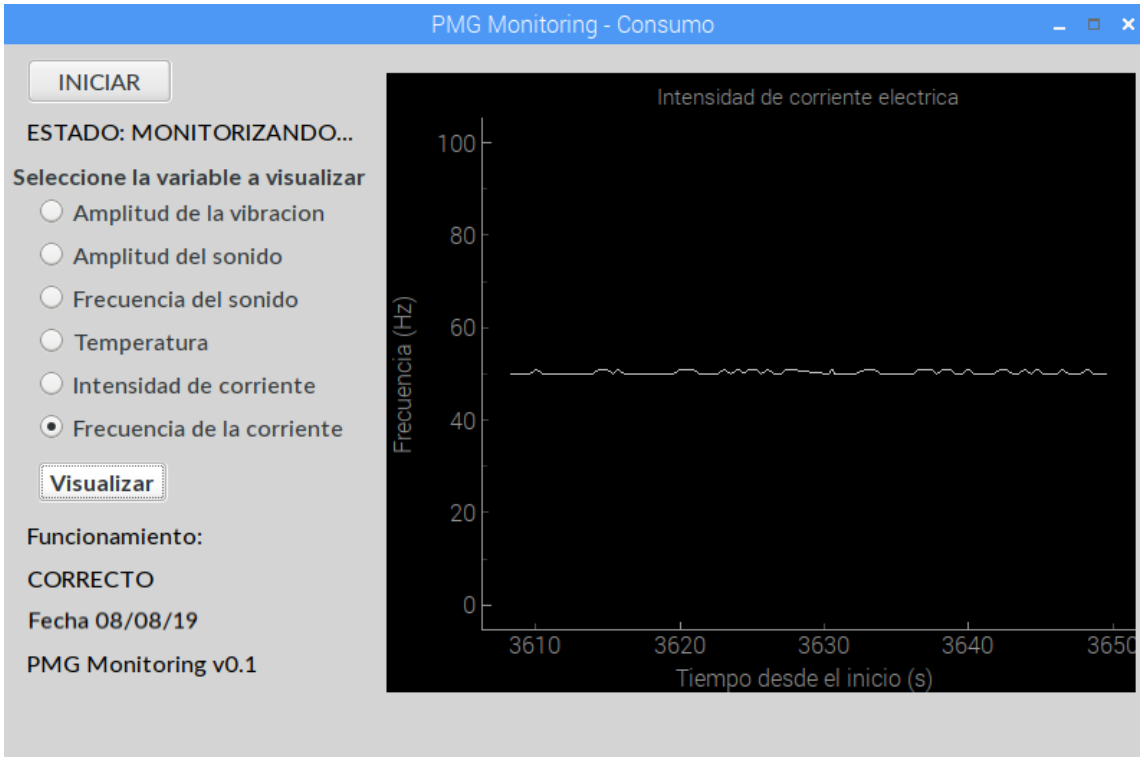

Ilustración 17- Frecuencia de la corriente

IV. CONCLUSIONES

Para la realización de la primera parte del trabajo, se ha realizado una investigación sobre el estado del arte del mantenimiento predictivo. De esta parte se puede extraer una gran conclusión a modo de resumen; las nuevas tecnologías, especialmente las relacionadas con las comunicaciones inalámbricas, van a permitir un desarrollo enorme en el mundo del mantenimiento industrial.

De esta forma, todas las fábricas que deseen ser competitivas en el futuro tendrán que adaptar este tipo de tecnologías. La monitorización permite conocer el estado de las máquinas, y poder planificar la producción y las paradas necesarias de la manera óptima, abaratando costes de fabricación. Con la llegada del 5G, en los próximos años se va a experimentar un auge de este tipo de tecnologías, de forma que investigar sobre este tipo de técnicas es una gran inversión.

En relación con los objetivos planteados, se ha obtenido una visión completa de todas las fases necesarias para implementar un sistema de mantenimiento predictivo, recogiendo para cada una de ellas las características necesarias para predecir las averías más comunes en la industria. El resultado de esta parte es satisfactorio, se ha conseguido el objetivo propuesto, pero siempre hay que tener en cuenta que este campo avanza rápidamente, y las nuevas tecnologías harán que se desarrollen nuevas técnicas.

Durante el estudio del arte, se ha observado que en un entorno real donde se aplican este tipo de técnicas, se utilizan sensores muy específicos para cada sistema o máquina, de forma que cada proyecto es diseñado a medida. Los resultados obtenidos son muy buenos, pero la inversión es elevada.

La herramienta diseñada, con sensores genéricos, está bien planteada desde el punto de vista de conseguir una herramienta de monitorización económica, apta para la gran mayoría de máquinas, aunque la precisión obtenida sea menor.

De esta forma se han conseguido cumplir los objetivos planteados inicialmente. Hay que tener en cuenta que la implementación de un sistema completo, con capacidad para analizar diferentes variables de fallo, engloba multitud de disciplinas, algunas de las cuales se escapan al alcance del trabajo.

Se deja abierto para un futuro el poder realizar una herramienta de monitorización genérica. Implementada sobre un miniPC de mayor capacidad, junto a una DAQ con un gran abanico de sensores, y que únicamente necesite ser calibrada y configurada para la máquina a monitorizar, pero sin necesidad de modificaciones hardware.

La parte que se ha quedado fuera del alcance de este proyecto es la destinada al análisis de los datos obtenidos mediante técnicas de machine learning, para poder procesarlos junto a los demás datos del sistema y poder predecir las averías. Se deja a disposición de otros alumnos realizar esta fase del proyecto para finalizar la herramienta de monitorización.

IX. BIBLIOGRAFÍA

Castelli, M. y. (2007). *Universidad de Mendoza*. Obtenido de Metodología de monitoreo, detección y diagnóstico de fallos en motores asíncronos de inducción: http://www.um.edu.uy/_upload/_investigacion/web_investigacion_66_metodologiademonitoreo.pdf

Cendón, B. (s.f.). *Blog personal de Bruno Cendón*. Obtenido de http://www.bcendon.com/las-redes-mas-usadas-en-el-iot/

De máquinas y herramientas. (12 de 2 de 2015). Obtenido de http://www.demaquinasyherramientas.com/herramientas-de-medicion/tipos-de-medidores-de-espesor-por-ultrasonido

Escuela Técnica Superior de Ingenieros Industriales. (2002). MANTENIMIENTO PREDICTIVO-PROACTIVO A TRAVÉS DEL ANÁLISIS DEL ACEITE. Las Palmas de Gran Canaria.

García Santamaría, C. (2017). Análisis espectral de señales para la detección de fallos de motores de inducción. *Universidad de Valladolid*.

Grijalba Davalillo, I. (2013). Sistema autónomo de análisis y detección de puntos calientes en cuadros eléctricos. *Universidad de la Rioja*.

Hurtado, J. M. (s.f.). Introducción a las redes de comunicación industrial. *Ciclo superior de automatización y robótica industrial*.

International Atomic Energy Agency. (2007). *IAEA*. Obtenido de Implementation Strategies and Tools for Condition Based Maintenance at Nuclear Power Plants: https://www-pub.iaea.org/MTCD/Publications/PDF/te_1551_web.pdf

Labaien, E. y. (2009). *Ingenierak*. Obtenido de Mantenimiento Predictivo y sus distintas técnicas de aplicación: https://docplayer.es/3755977-Mantenimiento-predictivo.html

López Ramón y Cajal, J. y. (2016). Industria 4.0, la gran oportunidad. *Economía Aragonesa*, 109-122.

Nieto Vilardell, E. (s.f.). *fidestec*. Obtenido de https://fidestec.com/blog/resistencias-shunt-que-son-como-funcionan/

Royo, J. G. (s.f.). *Researchgate*. Obtenido de https://www.researchgate.net/publication/268430716_Analisis_de_vibraciones_e_interpretacion_de_datos

Saavedra, P. (2010). *Docplayer*. Obtenido de https://docplayer.es/17674734-Mantenimiento-predictivo-y-monitoreo-segun-condicion.html

Seebo. (2018). *Smart Industry*. Obtenido de Why Predictive Maintenance is Driving Industry 4.0: https://www.smartindustry.com/whitepapers/2018/why-predictive-maintenance-is-driving-industry-4-0/

Solar, G. L. (s.f.). Análisis de vibraciones para el mantenimiento predictivo.

William Olarte C., M. B. (2010). Técnicas de mantenimiento predictivo utilizadas en la industria. *Universidad Tecnológica de Pereira*.

Zamora Guitérrez, F. (2016). Análisis y aplicación de técnicas de mantenimiento predictivo aplicado a malfuncionamientos por desalineamiento y desbalance en máquinas rotatorias. *Instituto politécnico nacional de México*.

V. ANEXOS

i. ANEXO I.- acelDataStorage.py

```
# -*- coding: utf-8 -*-
"""   Lectura y procesado MPU9250

            Acentos obviados intencionadamente.
            v1.0
            Pablo Martinez Galindo
"""
import smbus
import time
import numpy as np
import RPi.GPIO as GPIO
import math
from datetime import datetime
import psycopg2

try:
    connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                                  host = "192.168.1.46",
                                  port = "5432",
                                  database = "acel")
    cursor = connection.cursor()
    print ("Me conecto a la BBDD para guardar valores")

except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)

""" INICIO RUTINA DE SETUP """
def MPU9250_setup():
      #Variables globales
      global acc
      global SELF_TEST_X_ACCEL
      global SELF_TEST_Y_ACCEL
      global SELF_TEST_Z_ACCEL
      global SMPLRT_DIV
      global CONFIG
      global ACCEL_CONFIG             #configuracion del rango de escala
      global ACCEL_CONFIG2            #filtro interno
      global LP_ACCEL_ODR             #low power acc Output Data Rate control
      global WOM_Threshold            #valor para despertar el acc
      global FIFO_EN                  #activar sensores en fifo. todos cero
      menos el acc

      global INT_ENABLE               #habilitar interrupciones
      global INT_STATUS               #leer interrupts

      global ACCEL_XOUT_H             #lectura de datos
      global ACCEL_XOUT_L
      global ACCEL_YOUT_H
      global ACCEL_YOUT_L
      global ACCEL_ZOUT_H
      global ACCEL_ZOUT_L
```

```
        global MOT_DETECT_CTRL
        global PWR_MGMT_1               #ambas config de power
        global PWR_MGMT_2
        global WHO_AM_I
        global MPU_ready

        #Direccion del sensor en el bus i2c
        acc=0b1101000                           #0b110100X; X es 0 si AD0 esta en
bajo y 1 si AD0 esta en alto

        #Direcciones de los registros
        SMPLRT_DIV=0x19
        #SAMPLE_RATE=internal_sample_rate/(1+SMPLRT_DIV)
        CONFIG=0x1A

        ACCEL_CONFIG=0x1C               #configuracion del rango de escala
        ACCEL_CONFIG2=0x1d              #filtro interno
        LP_ACCEL_ODR=0x1E               #low power acc Output Data Rate control
        WOM_Threshold=0x1F              #valor para despertar el acc
        FIFO_EN=0x23                    #activar sensores en fifo. todos cero
menos el acc

        INT_PIN_CFG=0x37                #configuracion de interrupcion
        INT_ENABLE=0x38                 #habilitar interrupciones
        INT_STATUS=0x3A                 #leer interrupts

        ACCEL_XOUT_H=0x3B               #lectura de datos
        ACCEL_XOUT_L=0x3C
        ACCEL_YOUT_H=0x3D
        ACCEL_YOUT_L=0x3E
        ACCEL_ZOUT_H=0x3F
        ACCEL_ZOUT_L=0x40

        MOT_DETECT_CTRL=0x69
        PWR_MGMT_1=0x6B                 #ambas config de power
        PWR_MGMT_2=0x6C
        WHO_AM_I=0x75

        #Configuracion GPIO
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BOARD)

        print  '\n    - - - - - - - - - - - -  \n'
        device=hex(i2c.read_byte_data(acc, WHO_AM_I))
        print("      Direccion del dispositivo:")
        print(device)

        #Configuracion de los parametros del acelerometro
        i2c.write_byte_data(acc, PWR_MGMT_1, 0x00)
        #Make sure Accel is running
        i2c.write_byte_data(acc, PWR_MGMT_2, 0b00000111)
        #Acelerometro activo

        i2c.write_byte_data(acc, CONFIG, 0x00)
        #FSYNC desactivado. Se sobreescriben datos del FIFO
```

```
	i2c.write_byte_data(acc, ACCEL_CONFIG, 0x00)		#2g,
XYZ SELF TEST desactivado
	i2c.write_byte_data(acc, ACCEL_CONFIG2, 0b00001000)		#Data
output rate = 1kHz. Bandwith = 460Hz

	i2c.write_byte_data(acc, FIFO_EN, 0b00001000)		#FIFO
activo para acelerometro
	i2c.write_byte_data(acc, INT_ENABLE, 0x00)
	#Todas interrupciones desactivadas



	print("\n	Rutina de setup finalizada")
	print  '\n    - - - - - - - - - - - -  \n'

""" FIN RUTINA SETUP ACELEROMETRO """
#Puerto 1 del bus i2c
i2c=smbus.SMBus(1)

# Variables globales
global sleeping
global acel_data
global offset

gravity = 9.80665

def MPU9250_calibrating():
#Funcion para obtener valor inicial de offset
	num_iteraciones=0
	alpha=0.15
	z_gs0=1
	z_ms0=0
	offset=0
	offset_total=0
	print "	Calibrando..."

	while num_iteraciones<=10000:
		z_dataL=i2c.read_byte_data(acc, ACCEL_ZOUT_L)
		z_dataH=i2c.read_byte_data(acc, ACCEL_ZOUT_H)
		z_data=z_dataH<<8|z_dataL


		if z_data<32767:
			z_gs=z_data*(1/16384.0)
		else:
			z_gs=(z_data-65535)*(1/16384.0)

		z_gs=alpha*z_gs+(1-alpha)*z_gs0
		z_gs0=z_gs

		num_iteraciones=num_iteraciones+1
		offset_aux=z_gs-1
		offset_total=(offset_total+offset_aux)

	offset=offset_total/num_iteraciones
	print("\n	Rutina de calibracion finalizada")
	print  '\n    - - - - - - - - - - - -  \n'
```

```
	return offset

def MPU9250():
#Funcion principal toma de datos acelerometro
	MPU9250_setup()                         #Inicializa registros
	offset=MPU9250_calibrating()            #Calcula offset

	'''Funcion de lectura del acelerometro MPU9250'''
	#Inicializacion de variables
	z_gs0=1      #Eje z incial 1g
	z_ms0=9.8    #Eje z inicial 9,8m/s2

	gravMed=9.80665

	#Parámetro filtro paso bajo
	alpha=0.15

	tiempo=0
	inicio=time.time()

	while True:
		now=time.time()
		tiempo=now-inicio

		''' Lectura del valor de los registros. Dos registros de 8 bits
por cada eje.
		Se guardan en una misma variables los 16 bits.'''
		z_dataL=i2c.read_byte_data(acc, ACCEL_ZOUT_L)
		z_dataH=i2c.read_byte_data(acc, ACCEL_ZOUT_H)
		z_data=z_dataH<<8|z_dataL

		''' Con la configuración de 16 bits de resolución y +-2g de rango
se obtienen un total de 65535 pasos.
		Esto implica 16384LSB/g. El formato de los bits es en complemento
A2. Valores positivos de 0 a 32767.
		Valores negativos de 32768 a 65535.
		Se obtiene el valor de la medida en cada eje, en medidas de fuerza
g.'''
		if z_data<32767:
			z_gs=z_data*(1/16384.0)
		else:
			z_gs=(z_data-65535)*(1/16384.0)

		''' Se aplica filtro paso bajo software para mejorar la
repetibilidad de la medida.
		Cuanto mayor es el parametro alpha, mayor es el peso de la medida
anterior, respecto de la leida,
		para obtener la nueva medida.'''
		z_gs=z_gs-offset
		z_gs=alpha*z_gs+(1-alpha)*z_gs0
		#print ('g', z_gs)
		z_gs0=z_gs

		#CALIBRACION
		z_ms=z_gs*gravity
```

```
            ''' Obtencion de la velocidad en mm/s. Integrar la aceleracion
respecto del tiempo.'''
            z_vel= 1000 * (z_ms - z_ms0) / tiempo
            z_ms0=z_ms

            ''' Almacenar todos los datos adquiridos a lo largo de los 3 ejes.
'''
            #postgres_insert_query = "INSERT INTO measures (xdata, ydata,
zdata, xgs, ygs, zgs, xvel, yvel, zvel) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s)"
            #record_to_insert = (x_data, y_data, z_data, x_gs, y_gs, z_gs,
x_vel, y_vel, z_vel)

            #postgres_insert_query = "INSERT INTO measures (zdata, zgs, zvel,
tiempo, timestam) VALUES (%s, %s, %s, %s, %s)"
            ''' Almacenar todos los datos adquiridos a lo largo del eje Z.
'''
            record_to_insert = (z_data, z_gs, z_vel, tiempo, now)
            cursor.execute(postgres_insert_query, record_to_insert)
            connection.commit()

            while (time.time() < (now + 0.01)):
                  pass

def MPU9250_analysisData():
      offset=MPU9250_calibrating()
      '''Funcion de lectura del acelerometro MPU925. Obtencion de valores
medios de funcionamiento'''
      #Inicializacion de variables
      z_gs0=1
      z_ms0=9.8

      z_ms_min=1000000
      z_ms_max=0
      z_vel_min=1000000
      z_vel_max=0

      z_ms_tot = 0
      z_vel_tot = 0

      gravMed=9.80665

      #Parámetro filtro paso bajo
      alpha=0.15

      tiempo=0
      inicio=time.time()

      print'Comienzo mediciones'

      i=0
      numeroMediciones = 25000

      while i < numeroMediciones:
            now=time.time()
            tiempo=now-inicio
```

```
        z_dataL=i2c.read_byte_data(acc, ACCEL_ZOUT_L)
        z_dataH=i2c.read_byte_data(acc, ACCEL_ZOUT_H)
        z_data=z_dataH<<8|z_dataL

        if z_data<32767:
              z_gs=z_data*(1/16384.0)
        else:
              z_gs=(z_data-65535)*(1/16384.0)

        z_gs=z_gs-offset
        z_gs=alpha*z_gs+(1-alpha)*z_gs0
        z_gs0=z_gs

        z_ms=z_gs*gravity

        z_vel= 1000 * (z_ms - z_ms0) / tiempo
        z_ms0=z_ms

        ''' Rutina de obtencion de valores normales '''
        #Valores limite de aceleracion
        if  z_ms <= z_ms_min:
              z_ms_min = z_ms
        elif z_ms >= z_ms_max:
              z_ms_max = z_ms

        #Valores limite de velocidad
        if  z_vel < z_vel_min:
              z_vel_min = z_vel
        elif z_vel > z_vel_max:
              z_vel_max = z_vel

        z_ms_tot = z_ms_tot + z_ms
        z_vel_tot = z_vel_tot + z_vel

        i=i+1;

        while (time.time() < (now + 0.01)):
              pass

z_ms_med = z_ms_tot / numeroMediciones
z_vel_med = z_vel_tot / numeroMediciones

print ('Valor medio aceleracion')
print z_ms_med
print ('Valor maximo aceleracion')
print z_ms_max
print ('Valor minimo aceleracion')
print z_ms_min
print ('Valor medio velocidad')
print z_vel_med
print ('Valor maximo velocidad')
print z_vel_max
print ('Valor minimo velocidad')
print z_vel_min

print'Finalizo mediciones'
```

ii. ANEXO II.- tempDataStorage.py

```
# -*- coding: utf-8 -*-
"""    Programa para la medicion de temperatura
       Termometro DS18B20.
       centos obviados intencionadamente.
       v1.0
       Pablo Martinez Galindo
"""

#Librerias
import RPi.GPIO as GPIO
import time
import psycopg2

global consigna_temp
consigna_temp = 5

try:
    connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                                  host = "192.168.1.46",
                                  port = "5432",
                                  database = "temp")
    cursor = connection.cursor()

    #print ("Connected to temp database")

except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)

def get_temp_sens1():
    tfile = open("/sys/bus/w1/devices/28-031722a97bff/w1_slave")
    text = tfile.read()
    tfile.close()
    secondline = text.split("\n")[1]
    temperaturedata = secondline.split(" ")[9]
    temperature = float(temperaturedata[2:])
    temperature = temperature / 1000
    return float(temperature)

def get_temp_sens2():
    tfile = open("/sys/bus/w1/devices/28-0317229b64ff/w1_slave")
    text = tfile.read()
    tfile.close()
    secondline = text.split("\n")[1]
    temperaturedata = secondline.split(" ")[9]
    temperature = float(temperaturedata[2:])
    temperature = temperature / 1000
    return float(temperature)

def tempDataAcquisition():
    tiempo=0
    inicio=time.time()

    while True:
        now=time.time()
```

```
            tiempo=now-inicio

            t2=float(get_temp_sens1())
            t1=float(get_temp_sens2())
            deltaTemp=t2-t1
             #Almaceno valores en base de datos
            postgres_insert_query = "INSERT INTO measures (sensor_a, sensor_b,
    gradiente, tiempo, timestam) VALUES (%s, %s, %s, %s, %s)"
            record_to_insert = (t1, t2, deltaTemp, tiempo, now)
            cursor.execute(postgres_insert_query, record_to_insert)
            connection.commit()
            while (time.time() < (now + 2)):
                pass
```

iii. ANEXO III.- audDataStorage.py

```
# -*- coding: utf-8 -*-
"""   Programa para la monitorizacion de la amplitud y frecuencia del audio.
      Microfono + amplificador MAX9812
      Programa para la monitorizacion de la amplitud y frecuencia de la
corriente electrica. Resistencia de shunt + arduino
      Acentos obviados intencionadamente.
      v1.0
      Pablo Martinez Galindo
"""
#Librerias
import psycopg2
import serial
import time

#Puerto serie del arduino
ser = serial.Serial('/dev/ttyUSB0',9600)
now=0

try:
    connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                                  host = "192.168.1.46",
                                  port = "5432",
                                  database = "audi")
    cursor = connection.cursor()

    #print ("Connected to audi database")

except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)


def audDataAcquisition():
      #Lee valores del fichero PMG_Monitoring-v_0_1.ino
      tiempo=0
      inicio=time.time()

      while True:

            now=time.time()
            tiempo=now-inicio

            #Leo valores por el puerto serie
            read_serial_decibeles=(ser.readline())
            read_serial_freqA=(ser.readline())
            read_serial_int=(ser.readline())
            read_serial_freqI=(ser.readline())

            #Almaceno los valores en la base de datos
            postgres_insert_query = "INSERT INTO measures (aud_ampl,
aud_freq, tiempo, timestam, int_ampl, int_freq) VALUES (%s, %s, %s, %s, %s,
%s)"
            record_to_insert = (read_serial_decibeles, read_serial_freqA,
tiempo, now, read_serial_int, read_serial_freqI)
            cursor.execute(postgres_insert_query, record_to_insert)
```

```
                connection.commit()

                while (time.time() < (now + 0.1)):
                        pass

def MAX9812_analysisData():
#Rutina para obtencion de valores medios de funcionamiento normal
        iAmMax=0
        iAmMin=10000
        iAmTot=0

        iFrMax=0
        iFrMin=10000
        iFrTot=0

        aAmMax=0
        aAmMin=10000
        aAmTot=0

        aFrMax=0
        aFrMin=10000
        aFrTot=0

        tiempo=0
        inicio=time.time()

        print'Comienzo mediciones'

        i=0
        numeroMediciones = 25000

        while i < numeroMediciones:
                now=time.time()
                tiempo=now-inicio

                #Leo valores por el puerto serie
                read_serial_decibeles=float(ser.readline())
                read_serial_freqA=float(ser.readline())
                read_serial_int=float(ser.readline())
                read_serial_freqI=float(ser.readline())

                print read_serial_freqI

                if read_serial_decibeles <= aAmMin:
                        aAmMin = read_serial_decibeles
                elif read_serial_decibeles >= aAmMax:
                        aAmMax = read_serial_decibeles

                aAmTot=aAmTot+read_serial_decibeles

                if read_serial_freqA <= aFrMin:
                        aFrMin = read_serial_freqA
                elif read_serial_freqA >= aFrMax:
                        aFrMax = read_serial_freqA

                aFrTot=aFrTot+read_serial_freqA
```

```
        if read_serial_int <= iAmMin:
                iAmMin = read_serial_int
        elif read_serial_int >= iAmMax:
                iAmMax = read_serial_int

        iAmTot=iAmTot+read_serial_int

        if read_serial_freqI <= iFrMin:
                iFrMin = read_serial_freqI
        elif read_serial_freqI >= iFrMax:
                iFrMax = read_serial_freqI

        iFrTot=iFrTot+read_serial_freqI

        i=i+1

        while (time.time() < (now + 0.01)):
                pass

aAmMed=aAmTot/numeroMediciones
aFrMed=aFrTot/numeroMediciones
iAmMed=iAmTot/numeroMediciones
iFrMed=iFrTot/numeroMediciones

print ('Valor medio amplitud sonido')
print aAmMed
print ('Valor maximo amplitud sonido')
print aAmMax
print ('Valor minimo amplitud sonido')
print aAmMin

print ('Valor medio frecuencia sonido')
print aFrMed
print ('Valor maximo frecuencia sonido')
print aFrMax
print ('Valor minimo frecuencia sonido')
print aFrMin

print ('Valor medio amplitud intensidad')
print iAmMed
print ('Valor maximo amplitud intensidad')
print iAmMax
print ('Valor minimo amplitud intensidad')
print iAmMin

print ('Valor medio frecuencia intensidad')
print iFrMed
print ('Valor maximo frecuencia intensidad')
print iFrMax
print ('Valor minimo frecuencia intensidad')
print iFrMin


print'Finalizo mediciones'
```

PMG_Monitoring-v_0_1.ino

```
/*****************************************
 *                                       *
   Devuelve la frecuencia del sonido y su
   amplitud.
   Devuelve la frecuencia de la corriente
   electrica y su amplitud.
   Acentos obviados intencionadamente
 *                                       *
 *****************************************/


//Librerias utilizadas
#include "arduinoFFT.h"
arduinoFFT FFT = arduinoFFT();


//Variables y constantes utilizadas en el procesado de audio (MAX9812)
#define SAMPLES 128                     //Numero de muestras para realizar la
FFT
#define SAMPLING_FREQUENCY 1000         //Frecuencia de muestreo en Hz para la
FFT. Limitacion ADC -> 10000Hz


unsigned int gananciaMAX9812 = 10;      //Ganancia del amplificador
unsigned int sampling_period_us;
unsigned long microseconds;


double vReal[SAMPLES];
double vImag[SAMPLES];


const int sampleWindow = 50;            //Ventana de muestreo en ms


double media = 0;
unsigned int numSamples = 0;             //Numero de muestras para conversion
de la amplitud a dB
```

```
//Variavles y constantes utilizadas en el procesado de la corriente (shunt)
const float Rshunt = 0.6;              // 600 miliohmnios
const int Rcarga = 4;                  // 4 ohminios

float lectura;                         // Valor leido
float escala = 3.96 / 1024;            // Valor de cuantificacion

float conversion = 12.4 / 3.96;        // Conversion de valores de voltaje
divisor resistivo

float voltaje_divisor;                 // Voltaje en el divisor de tension
float voltaje_real;                    // Voltaje real
float voltaje_ref = 0;                 // Voltaje a la salida del shunt
float voltaje_total;
float voltaje_medio;

float intensidad;                      // Consumo eléctrico instantaneo
float intensidad_media = 3;            // Consumo eléctrico medio por ciclo

// FFT
#define SAMPLES_i 64                   // Numero de muestras
#define SAMPLING_FREQUENCY_i 500       // Frecuencia de muestreo

unsigned int sampling_period_us_i;
unsigned long microseconds_i;

double vReal_i[SAMPLES_i];
double vImag_i[SAMPLES_i];

//Pinout
const int shuntPin = A0;               // Pin lectura datos corriente
const int sensorPIN = A1;              //Pin lectura datos microfono

void setup()
```

```
{
  Serial.begin(9600);
  //pinMode(shuntPin, INPUT);
  sampling_period_us = round(1000000 * (1.0 / SAMPLING_FREQUENCY));
  sampling_period_us_i = round(1000000 * (1.0 / SAMPLING_FREQUENCY_i));
}

void loop()
{
  /* ------ COMIENZO PROCESADO AUDIO ------ */
  /* --- COMIENZO CALCULO AMPLITUD --- */

  unsigned long startMillis = millis();

  unsigned int signalMax = 0;
  unsigned int signalMin = 1024;

  // Muestro para dB
  unsigned int sample;
  while (millis() - startMillis < sampleWindow)
  {
    sample = analogRead(sensorPIN);
    if (sample < 1024)
    {
      if (sample > signalMax)
      {
        signalMax = sample;  // Actualizar máximo
      }
      else if (sample < signalMin)
      {
        signalMin = sample;  // Actualizar mínimo
      }
    }
```

```
}

unsigned int peakToPeak = signalMax - signalMin;  // Amplitud del sonido

//Calculo dB
numSamples++;
long x = 10 * peakToPeak;
long y = 100 * x;
double z = log10(y);
double decibeles = 10 * z;

/* --- FIN CALCULO AMPLITUD --- */

/* -- COMIENZO CALCULO FFT --- */

for (int i = 0; i < SAMPLES; i++)
{
  microseconds = micros();    //Overflows after around 70 minutes!

  vReal[i] = analogRead(sensorPIN);
  vImag[i] = 0;

  while (micros() < (microseconds + sampling_period_us)) {
  }
}

//FFT
FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);
double peak = FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);

/* --- FIN CALCULO FFT --- */
```

```
/* ------ FIN PROCESADO AUDIO ------ */

/* ------ COMIENZO PROCESADO CORRIENTE ------ */
//Calculo FFT
for (int i = 0; i < SAMPLES_i; i++)
{
  microseconds_i = micros();    //Overflows after around 70 minutes!
  /* --- COMIENZO CALCULO INTENSIDAD --- */
  //Shunt

  lectura = analogRead(shuntPin);
  voltaje_divisor = lectura * escala;             // Convierto a milivolts
  voltaje_real = voltaje_divisor * conversion;    // Conversion a 12v
  intensidad = (voltaje_real - voltaje_ref) / (Rcarga + Rshunt);

  /* --- FIN CALCULO INTENSIDAD --- */
  vReal_i[i] = intensidad;
  vImag_i[i] = 0;

  voltaje_total = voltaje_total + voltaje_real;

  while (micros() < (microseconds_i + sampling_period_us_i)) {
  }
}

//FFT
FFT.Windowing(vReal_i, SAMPLES_i, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
FFT.Compute(vReal_i, vImag_i, SAMPLES_i, FFT_FORWARD);
FFT.ComplexToMagnitude(vReal_i, vImag_i, SAMPLES_i);
double peak_i = FFT.MajorPeak(vReal_i, SAMPLES_i, SAMPLING_FREQUENCY_i);

voltaje_medio = voltaje_total / SAMPLES_i;
intensidad_media = voltaje_medio / (Rcarga + Rshunt);
```

```
  voltaje_total = voltaje_medio;
  /* ------ FIN PROCESADO CORRIENTE ------ */

  //Envio valores por puerto serie
  Serial.println(decibeles);
  delay(10);
  Serial.println(peak);
  delay(10);
  Serial.println(intensidad_media);
  delay(10);
  Serial.println(round(peak_i));

}
```

iv. ANEXO IV.- app.py

```
# -*- coding: utf-8 -*-
"""    Interfaz grafica de la aplicacion.
       Acentos obviados intencionadamente.
       v1.0
       Pablo Martinez Galindo
"""

import sys
import time
import pyqtgraph as pg
import numpy as np
import psycopg2
import analysis
from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtGui import QApplication, QMainWindow, QMessageBox, QPushButton,
QLabel, QDialog
from scipy.fftpack import fft, rfft
from multiprocessing import Process

global y_val
global x_val

y_val = np.empty([0])
x_val = np.empty([0])

n_acel=5
n_temp=5
n_micro=5
n_shunt=5

def amplVib():
#funcion para leer valores de la BBDD del acelerometro
      global n_acel

      if (n_acel >= 100):
            n_acel=100
      else:
            n_acel=n_acel+5

      global y_val
      global x_val

      y_val = np.empty([0])
      x_val = np.empty([0])

      try:
            connection = psycopg2.connect(user = "pmg",
                                          password = "pmg.psql",
                                    host = "192.168.1.46",
                                       port = "5432",
                                          database = "acel")
            cursor = connection.cursor()

            #print ("Me conecto a la BBDD para leer valores")
```

```
	except (Exception, psycopg2.Error) as error :
		print ("Error while connecting to PostgreSQL", error)

	postgreSQL_select_Query = "select * from measures order by timestam
desc limit 100"
	cursor.execute(postgreSQL_select_Query)

	#Guardar ultimas 100 medidas
	last_100_measures = cursor.fetchmany(n_acel)
	connection.close()
	for row in reversed(last_100_measures):
		y_val = np.append(y_val, row[3])
	for row in reversed(last_100_measures):
		x_val = np.append(x_val, row[2])

def amplSonido():
#funcion para leer valores de la BBDD del arduino
	global n_micro

	if (n_micro >= 100):
		n_micro=100
	else:
		n_micro=n_micro+5

	global y_val
	global x_val

	y_val = np.empty([0])
	x_val = np.empty([0])

	try:
		connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                               host = "192.168.1.46",
                                  port = "5432",
                                    database = "audi")
		cursor = connection.cursor()

		#print ("Me conecto a la BBDD para leer valores")

	except (Exception, psycopg2.Error) as error :
		print ("Error while connecting to PostgreSQL", error)

	postgreSQL_select_Query = "select * from measures order by timestam
desc limit 100"
	cursor.execute(postgreSQL_select_Query)

	#Guardar ultimas 100 medidas
	last_100_measures = cursor.fetchmany(n_micro)
	connection.close()
	for row in reversed(last_100_measures):
		y_val = np.append(y_val, row[2])
	for row in reversed(last_100_measures):
		x_val = np.append(x_val, row[0])

def freqSonido():
#funcion para leer valores de la BBDD del arduino
```

```
	global n_micro

	if (n_micro >= 100):
		n_micro=100
	else:
		n_micro=n_micro+5

	global y_val
	global x_val

	y_val = np.empty([0])
	x_val = np.empty([0])

	try:
		connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                               host = "192.168.1.46",
                                  port = "5432",
                                    database = "audi")
		cursor = connection.cursor()

		#print ("Me conecto a la BBDD para leer valores")

	except (Exception, psycopg2.Error) as error :
		print ("Error while connecting to PostgreSQL", error)

	postgreSQL_select_Query = "select * from measures order by timestam
desc limit 100"
	cursor.execute(postgreSQL_select_Query)

	#Guardar ultimas 100 medidas
	last_100_measures = cursor.fetchmany(n_micro)
	connection.close()
	for row in reversed(last_100_measures):
		y_val = np.append(y_val, row[2])
	for row in reversed(last_100_measures):
		x_val = np.append(x_val, row[1])

def difTemp():
#funcion para leer valores de la BBDD de los termometros

	global y_val
	global x_val

	y_val = np.empty([0])
	x_val = np.empty([0])

	global n_temp

	if (n_temp >= 100):
		n_temp=100
	else:
		n_temp=n_temp+1

	try:
		connection = psycopg2.connect(user = "pmg",
```

```
                                        password = "pmg.psql",
                                    host = "192.168.1.46",
                                       port = "5432",
                                          database = "temp")
                cursor = connection.cursor()

                #print ("Connected to temp database")

        except (Exception, psycopg2.Error) as error :
                print ("Error while connecting to PostgreSQL", error)

        postgreSQL_select_Query = "select * from measures order by timestam
desc limit 100"
        cursor.execute(postgreSQL_select_Query)

        #Guardar ultimas 100 medidas
        last_100_measures = cursor.fetchmany(n_temp)
        connection.close()
        for row in reversed(last_100_measures):
                x_val = np.append(x_val, row[2])
        for row in reversed(last_100_measures):
                y_val = np.append(y_val, row[4])

def amplInt():
        global y_val
        global x_val

        y_val = np.empty([0])
        x_val = np.empty([0])

        global n_shunt

        if (n_shunt >= 100):
                n_shunt=100
        else:
                n_shunt=n_shunt+1

        try:
                connection = psycopg2.connect(user = "pmg",
                                        password = "pmg.psql",
                                    host = "192.168.1.46",
                                       port = "5432",
                                          database = "audi")
                cursor = connection.cursor()

                #print ("Me conecto a la BBDD audi para leer valores")

        except (Exception, psycopg2.Error) as error :
                print ("Error while connecting to PostgreSQL", error)

        postgreSQL_select_Query = "select * from measures order by timestam
desc limit 100"
        cursor.execute(postgreSQL_select_Query)

        #Guardar ultimas 100 medidas
        last_100_measures = cursor.fetchmany(n_shunt)
        connection.close()
```

```
        for row in reversed(last_100_measures):
                x_val = np.append(x_val, row[5])
        for row in reversed(last_100_measures):
                y_val = np.append(y_val, row[2])

def freqInt():
#funcion para leer valores de la BBDD del arduino

        global y_val
        global x_val

        y_val = np.empty([0])
        x_val = np.empty([0])

        global n_shunt

        if (n_shunt >= 100):
                n_shunt=100
        else:
                n_shunt=n_shunt+1

        try:
                connection = psycopg2.connect(user = “pmg”,
                                        password = “pmg.psql”,
                                host = “192.168.1.46”,
                                   port = “5432”,
                                      database = “audi”)
                cursor = connection.cursor()

                #print (“Me conecto a la BBDD audi para leer valores”)

        except (Exception, psycopg2.Error) as error :
                print (“Error while connecting to PostgreSQL”, error)

        22ostgreSQL_select_Query = “select * from measures order by timestam
desc limit 100”
        cursor.execute(22ostgreSQL_select_Query)

        #Guardar ultimas 100 medidas
        last_100_measures = cursor.fetchmany(n_shunt)
        connection.close()
        for row in reversed(last_100_measures):
                x_val = np.append(x_val, row[4])
        for row in reversed(last_100_measures):
                y_val = np.append(y_val, row[2])

#Clase heredada de QmainWindow
#Clase que define la ventana de la aplicación (botones, textos, etc)
class Ventana(QMainWindow):
        def __init__(self, app):
                QMainWindow.__init__(self)
                uic.loadUi("app.ui", self)      #Fichero de la interfaz
                #Definir tamaño y nombre ventana
                self.setWindowTitle("PMG Monitoring")
                self.setMinimumSize(800, 500)
                self.setMaximumSize(800, 500)
```

```
                self.app = app

                self.graphicsView = pg.PlotWidget(self.centralwidget)
                self.graphicsView.setGeometry(270,20, 521, 431)

                #mostrar la fecha
                self.labelFecha.setText("Fecha "  + time.strftime("%x"))

                self.visBot.clicked.connect(self.radio_value)
                self.initBot.clicked.connect(self.iniciarEvent)

                #~ self.p = Process(target=self.plot)

        def iniciarEvent(self):
        #boton INICAR
                self.labelNombre.setText("ESTADO: MONITORIZANDO...")

        def closeEvent(self, event):
        #cerrar la aplicacion
                resultado = QMessageBox.question(self, "Salir", "Detener la
aplicacion", QMessageBox.Yes | QMessageBox.No)
                if resultado == QMessageBox.Yes:
                        event.accept()
                        exit()
                else: event.ignore()

        def radio_value(self):
        #selector de variable a visualizar
                global y_val
                global x_val

                global func

                func = None
                temp_ = 0

                if self.acelBot.isChecked():
                #variable amplitud vibracion. Modificacion de la interfaz
                        self.graphicsView.clear()
                        self.setWindowTitle("PMG Monitoring - MPU9250")
                        self.graphicsView.setTitle("Vibracion en el eje Z")
                        self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                        self.graphicsView.setLabel("left", "Velocidad (mm/s)")
                        self.graphicsView.enableAutoRange()
                        self.graphicsView.setYRange(-20, 20, padding=None,
update=False)

                        func = amplVib
                        temp_ = 1

                elif self.amplSonBot.isChecked():
                #variable amplitud del sonido. Modificacion de la interfaz
                        self.graphicsView.clear()
                        self.setWindowTitle("PMG Monitoring - MAX9812")
                        self.graphicsView.setTitle("Amplitud del sonido")
```

```
                    self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                    self.graphicsView.setLabel("left", "Amplitud (dB)")
                    self.graphicsView.enableAutoRange()
                    self.graphicsView.setYRange(0, 80, padding=None,
update=False)

                    func = amplSonido
                    temp_ = 1

             elif self.freqSonBot.isChecked():
             #variable frecuencia sonido. Modificacion de la interfaz
                    self.graphicsView.clear()
                    self.setWindowTitle("PMG Monitoring - Microfono")
                    self.graphicsView.setTitle("Frecuencia del sonido")
                    self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                    self.graphicsView.setLabel("left", "Frecuencia (Hz)")
                    self.graphicsView.enableAutoRange()
                    self.graphicsView.setYRange(0, 1000, padding=None,
update=False)

                    func = freqSonido
                    temp_ = 1

             elif self.tempBot.isChecked():
             #variable temperatura. Modificacion de la interfaz
                    self.graphicsView.clear()
                    self.setWindowTitle("PMG Monitoring - DS18B20")
                    self.graphicsView.setTitle("Gradiente de temperatura")
                    self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                    self.graphicsView.setLabel("left", "Temperatura (ºC)")
                    self.graphicsView.enableAutoRange()
                    self.graphicsView.setYRange(-15, 15, padding=None,
update=False)

                    func = difTemp
                    temp_ = 2

             elif self.amplIntBot.isChecked():
             #variable amplitud intensidad de corriente. Modificacion de la
interfaz
                    self.graphicsView.clear()
                    self.setWindowTitle("PMG Monitoring - Consumo")
                    self.graphicsView.setTitle("Intensidad de corriente
electrica")
                    self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                    self.graphicsView.setLabel("left", "Intensidad (A)")
                    self.graphicsView.enableAutoRange()
                    self.graphicsView.setYRange(0, 12, padding=None,
update=False)

                    func = amplInt
                    temp_ = 1
```

```
            elif self.freqIntBot.isChecked():
            #variable frecuencia corriente. Modificacion de la interfaz
                  self.graphicsView.clear()
                  self.setWindowTitle("PMG Monitoring - Consumo")
                  self.graphicsView.setTitle("Intensidad de corriente
electrica")
                  self.graphicsView.setLabel("bottom", "Tiempo desde el
inicio (s)")
                  self.graphicsView.setLabel("left", "Frecuencia (Hz)")
                  self.graphicsView.enableAutoRange()
                  self.graphicsView.setYRange(0, 100, padding=None,
update=False)

                  func = freqInt
                  temp_ = 1
            else:
                  self.setWindowTitle("PMG Monitoring")

            while True:

                  #Comentar a partir de aqui para quitar alertas
                  #llama a modulo de análisis, que devuelve estado de la
monitorización. Si hay una alerta, se representa la variable en cuestión y se
indica el mal funcionamiento.
                  analysis.analysis()
                  alerta = analysis.alerta
                  if alerta[0] != 0:
                        func = amplVib
                        if alerta[0] == 1:
                              self.labelFunc.setText("ALERTA! Vibraciones
anomalas")
                        else:
                              self.labelFunc.setText("ALERTA! Vibraciones
elevadas")
                        self.graphicsView.clear()
                        self.setWindowTitle("PMG Monitoring -
Acelerometro")
                        self.graphicsView.setTitle("Vibracion en el eje Z")
                        self.graphicsView.setLabel("bottom", "Tiempo desde
el inicio (s)")
                        self.graphicsView.setLabel("left", "Velocidad
(mm/s)")
                        self.graphicsView.enableAutoRange()
                        self.graphicsView.setYRange(-20, 20, padding=None,
update=False)
                  elif alerta[1] != 0:
                        func = difTemp
                        self.labelFunc.setText("ALERTA! Exceso de
temperatura")
                        self.graphicsView.clear()
                        self.setWindowTitle("PMG Monitoring - Termometro")
                        self.graphicsView.setTitle("Gradiente de
temperatura")
                        self.graphicsView.setLabel("bottom", "Tiempo desde
el inicio (s)")
                        self.graphicsView.setLabel("left", "Temperatura
(ºC)")
```

```
                        self.graphicsView.enableAutoRange()
                        self.graphicsView.setYRange(-15, 15, padding=None,
update=False)
                elif alerta [2] != 0:
                        if alerta [2] == 1:
                                func = freqSonido
                                self.labelFunc.setText("ALERTA!
Funcionamiento incorrecto")
                                self.graphicsView.clear()
                                self.setWindowTitle("PMG Monitoring -
Microfono")
                                self.graphicsView.setTitle("Frecuencia del
sonido")
                                self.graphicsView.setLabel("bottom", "Tiempo
desde el inicio (s)")
                                self.graphicsView.setLabel("left",
"Frecuencia (Hz)")
                                self.graphicsView.enableAutoRange()
                        else:
                                func = amplSonido
                                self.labelFunc.setText("ALERTA!
Funcionamiento incorrecto")
                                self.graphicsView.clear()
                                self.setWindowTitle("PMG Monitoring -
Microfono")
                                self.graphicsView.setTitle("Amplitud del
sonido")
                                self.graphicsView.setLabel("bottom", "Tiempo
desde el inicio (s)")
                                self.graphicsView.setLabel("left", "Amplitud
(dB)")
                                self.graphicsView.enableAutoRange()
                elif alerta [3] != 0:
                        func = amplInt
                        if alerta [3] == 1:
                                self.labelFunc.setText("ALERTA! Consumo
electrico elevado")
                        else:
                                self.labelFunc.setText("ALERTA! Consumo
electrico bajo")
                        self.graphicsView.clear()
                        self.setWindowTitle("PMG Monitoring - Consumo")
                        self.graphicsView.setTitle("Intensidad de corriente
electrica")
                        self.graphicsView.setLabel("bottom", "Tiempo desde
el inicio (s)")
                        self.graphicsView.setLabel("left", "Intensidad
(A)")
                        self.graphicsView.enableAutoRange()
                        self.graphicsView.setYRange(0, 15, padding=None,
update=False)
                else:
                        func = func
                        self.labelFunc.setText("CORRECTO")
                #Comentar hasta aqui para quitar alertas

                func()
```

```
                    self.graphicsView.clear()
                    self.graphicsView.plot(y_val, x_val)
                    self.app.processEvents()
                    self.graphicsView.repaint()
                    #self.graphicsView.enableAutoRange()
                    time.sleep(temp_)

def appExec():
      #Instancia para crear una aplicacion
      app = QApplication(sys.argv)
      #Crear un objeto de la clase
      _ventana = Ventana(app)
      #Mostrar la ventana
      _ventana.show()
      #Ejecutar la aplicacion
      app.exec_()
```

v. ANEXO V.- análisis.py

```
# -*- coding: utf-8 -*-
"""    Analisis de las variables almacenadas durante la monitorización.

            Acentos obviados intencionadamente.
            v1.0
            Pablo Martinez Galindo
"""

import psycopg2
import numpy as np

n=100

#Acelerometro
global consigna_acelMax
global consigna_acelMed
global x_val

consigna_acelMax = 2.9731*10e7
consigna_acelMed = 1.1839*10e3


#Termometro
global consigna_temp
global consigna_temp2
global t_val

consigna_temp = 8

#Arduino
global consigna_amplMaxAud
global consigna_freqMaxAud
global consigna_amplMedAud
global consigna_freqMedAud
global consigna_intMax
global consigna_intMin

global f_val
global a_val

consigna_amplMaxAud = 60
consigna_freqMaxAud = 400
consigna_amplMedAud = 58.687
consigna_freqMedAud = 192.428
consigna_intMax = 1.8
consigna_intMin = 1.4


#Funcion analisis
def analysis():
#funcion que analiza los valores leidos y genera el vector de alertas
  global func
  global alerta
  alerta = np.array([0,0,0,0])
```

```
  #Valores leidos del acelerometro
  x_val = np.empty([0])
  j=0
  try:
      connection = psycopg2.connect(user = "pmg",
                                    password = "pmg.psql",
                                    host = "192.168.1.46",
                                    port = "5432",
                                    database = "acel")
      cursor = connection.cursor()

  except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)

  postgreSQL_select_Query = "select * from measures order by timestam desc
limit 100"
  cursor.execute(postgreSQL_select_Query)

  #Guardar ultimas 100 medidas
  last_100_measures = cursor.fetchmany(n)
  connection.close()
  for row in reversed(last_100_measures):
    x_val = np.append(x_val, row[2])

  if (abs(np.median(x_val) - consigna_acelMed) >= consigna_acelMed/2):
      alerta_acel = 1

  for i in range (0, n):
    if abs(x_val[i]) >= consigna_acelMax:
      alerta_acel=2
    else:
      alerta_acel=0

  alerta[0]=alerta_acel


  #Valores leidos de los termometros
  t_val = np.empty([0])
  j=0
  try:
    connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                                  host = "192.168.1.46",
                                  port = "5432",
                                  database = "temp")
    cursor = connection.cursor()

  except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)

  postgreSQL_select_Query = "select * from measures order by timestam desc
limit 100"
  cursor.execute(postgreSQL_select_Query)

  #Guardar ultimas 100 medidas
  last_100_measures = cursor.fetchmany(n)
  connection.close()
```

```
  for row in reversed(last_100_measures):
    t_val = np.append(t_val, row[2])

  for i in range (0, n):
    if abs(t_val[i]) >= consigna_temp:
      alerta_temp=1
    else:
      alerta_temp=0

  alerta[1]=alerta_temp


  #Valores leidos del microfono
  f_val = np.empty([0])
  a_val = np.empty([0])
  j=0
  j2=0

  try:
    connection = psycopg2.connect(user = "pmg",
                                  password = "pmg.psql",
                                  host = "192.168.1.46",
                                  port = "5432",
                                  database = "audi")
    cursor = connection.cursor()
  except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)

  postgreSQL_select_Query = "select * from measures order by timestam desc
limit 100"
  cursor.execute(postgreSQL_select_Query)

  #Guardar ultimas 100 medidas
  last_100_measures = cursor.fetchmany(n)
  connection.close()
  for row in reversed(last_100_measures):
    f_val = np.append(f_val, row[1])
  for row in reversed(last_100_measures):
    a_val = np.append(a_val, row[0])

  if (abs(np.median(f_val)) - consigna_freqMedAud) >= consigna_freqMedAud/2:
      alerta_audi = 1

  if (abs(np.median(a_val)) - consigna_amplMedAud) >= consigna_amplMedAud/2:
      alerta_audi = 2

  for i in range (0, n):
    if f_val[i] >= consigna_freqMaxAud:
      alerta_audi=1
    elif a_val[i] >= consigna_amplMaxAud:
        alerta_audi=2
    else:
      alerta_audi=0

  alerta[2]=alerta_audi

  i_val = np.empty([0])
```

```
j=0

for row in reversed(last_100_measures):
  i_val = np.append(i_val, row[5])

for i in range (0, n):
  if i_val[i] >= consigna_intMax:
      alerta_int=1
  elif i_val[i] <= consigna_intMin:
     alerta_int=0
  else:
    alerta_int=0

alerta[3] = alerta_int
```

vi. ANEXO VI.- main.py

```
"""   Bucle principal del pograma

            Adquisicion, analisis y representacion
            de datos orientados a mantenimiento predictivo

            Acentos obviados intencionadamente.
            v1.0
            Pablo Martinez Galindo
"""

import smbus
import time
import RPi.GPIO as GPIO
import threading
import psycopg2
import serial
import analysis

#Threads routines
import tempDataStorage
import acelDataStorage
import audDataStorage
import app

#Puerto 1 del bus i2c
i2c=smbus.SMBus(1)
#Puerto serie (Arduino)
ser = serial.Serial('/dev/ttyUSB0',9600)

#Constants
acc=0b1101000                          #X es 0 si AD0 esta en bajo y 1 si AD0
esta en alto

#Threads definition
routineTemp=threading.Thread(target=tempDataStorage.tempDataAcquisition)
routineAcel=threading.Thread(target=acelDataStorage.MPU9250)
routineAudi=threading.Thread(target=audDataStorage.audDataAcquisition)
routineApp=threading.Thread(target=app.appExec)

try:
      routineAcel.daemon=True
      routineAcel.start()

      routineTemp.daemon=True
      routineTemp.start()

      routineAudi.daemon=True
      routineAudi.start()

      routineApp.daemon=True
      routineApp.start()

      while True:
            time.sleep(28800)
except (KeyboardInterrupt, SystemExit):
```

```
            print '\n Cierro hilo debido a interrupcion de teclado. \n'
      else:
            True
```