Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Captación y procesado de datos para Industria 4.0 con un prototipo de sensorización

Autor: Juan Bautista Cita Muñoz

Tutor: José María Gutiérrez Martínez

2018

GRADO EN INGENIERIA INFORMATICA

ESCUELA POLITECNICA SUPERIOR TRABAJO

DE FIN DE GRADO

Captación y procesado de datos para Industria 4.0 con un prototipo de sensorización

Trabajo de Fin de Grado presentado por
Juan Bautista Cita Muñoz

Proyecto dirigido por

José María Gutiérrez Martínez

Julio de 2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado Ingeniería Informática

Trabajo Fin de grado

Captación y procesado de datos para Industria 4.0 con un prototipo de sensorización

Autor: Juan Bautista Cita Muñoz

Tutor: José María Gutiérrez Martínez

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACION: ..... FECHA: **.....de.....de….....**

ÍNDICE

ÍNDICE DE ILUSTRACIONES

Resumen:

En las puertas de la cuarta revolución industrial, la industria pretende transformar sus procesos productivos desde la automatización, a la aportación de inteligencia a los diferentes elementos del proceso productivo. Es en este contexto donde este proyecto aporta una solución de industria 4.0 al problema del mantenimiento y la optimización del proceso productivo.

En la actualidad, los procesos de mantenimiento preventivo pueden optimizarse con la incorporación de nuevos sensores gracias al abaratamiento de los componentes electrónicos, junto con las nuevas técnicas de adquisición y procesamiento de datos, transformando el proceso de mantenimiento preventivo, en mantenimiento predictivo.

Para llevar a cabo el proyecto se han estudiado las tecnologías actualmente disponibles a nivel de placas controladoras y de comunicación, protocolos de comunicación M2M (Machine to machine), arquitecturas de redes de sensorización, bases de datos y sistemas de procesamiento de datos que han permitido extraer información de los datos recopilados.

Palabras clave: Industria 4.0. IoT. Sensores. Mantenimiento predictivo. ESP8266.

Abstract:

At the beginning of de fourth industrial revolution, the industry pretends to transform its production processes from automatization to the contribution of intelligence to the different elements of the production process. It is in this context, where this project brings an industry 4.0 solution to the problem of maintenance and optimization of the production process.

Currently, preventive maintenance process can be optimized with the incorporation of new sensors thanks to cheaper electronic components. These sensors together with the new data acquisition and processing techniques are changing the preventive maintenance process into predictive maintenance process.

To carry out the project, we had studied current technologies available in terms of controller and communication boards, M2M (Machine to machine) protocols, sensorization networks architectures, databases, and data processing systems that have allowed extracting information from the collected data.

Key words: Industry 4.0. Internet of things. Sensor. Predictable manteinence. ESP8266.

Capítulo 1: Introducción

1 INTRODUCCIÓN

Desde el origen de los primeros individuos, el poder y la dominancia residían en el control, primeramente, del fuego, seguido del control sobre la naturaleza, y ya en las primeras civilizaciones, el control de las personas, estableciendo con ello el rumbo de las sociedades. Posteriormente, mientras el mundo se hacía más complejo, la capacidad para tomar ese poder se hacía también más compleja. El control se traduce en un poder tan significante, que muchas civilizaciones cuando desconocían el origen de ese poder, le atribuían orígenes mágicos o místicos.

Desde que comencé el grado y a medida que me adentraba en el mundo de la informática, fui descubriendo como está me iba a proporcionar el poder de controlar mejor mi entorno. Es por ello por lo que la domótica ha sido uno de mis grandes intereses y motivación de mis proyectos a lo largo del grado. Pero no solo ha sido la idea de controlar mejor el entorno lo que ha motivado este interés, sino además la posibilidad de hacer ecosistemas más cómodos, seguros, eficientes y ecológicos. Este interés me ha llevado a abarcar temas de seguridad informática, redes de computadores, Smart Cities, las Smart Homes y la recientemente descubierta industria 4.0. Así pues, este proyecto me permite ser partícipe de la cuarta revolución industrial, además de progresar en mis intereses sobre el sector de la domótica.

Para comprender la industria 4.0 o cuarta revolución industrial es necesario repasar los antecedentes de las revoluciones industriales. A lo largo de la historia, diversos inventores y descubridores han ido impulsando el avance de la sociedad, dando lugar a grandes descubrimientos tecnológicos que transformaban los diferentes ámbitos de la sociedad: como son el político, económico o tecnológico; definiendo así la forma de vivir de la misma. Estas grandes trasformaciones se han definido como revoluciones industriales.

En la segunda mitad del siglo XVIII en Reino Unido se produjo el primer proceso de trasformación social y tecnológica al que se denominó Primera Revolución Industrial. En esta primera revolución se desarrollaron inventos como la máquina de vapor que permitieron el mecanizado de tareas junto con la mejora de la productividad. Aunque algunos puestos de la mano de obra fueron sustituidos por esta nueva tecnología, estos avances dieron lugar a nuevos empleos como fueron, por ejemplo, los necesarios para desarrollar el ferrocarril.

Todos estos cambios aparecidos en la Primera Revolución Industrial aceleraron la llegada de la Segunda Revolución Industrial, datando su comienzo alrededor de 1880 con el aprovechamiento de la energía eléctrica que dio lugar a la fabricación en masa.

Después de la segunda guerra mundial la industria militar era la que lideraba la tecnología, pero no fue hasta la década de los sesenta cuando apareció internet. Primeramente, con fines gubernamentales y ya en la década de los noventa se comenzó a propagar por el mundo empresarial.

Internet fue la semilla que favoreció el desarrollo exponencial de las tecnologías de la información, dando lugar a la Tercera Revolución Industrial con la automatización de los procesos productivos. Pero los últimos avances en computación personal con la invención del Iphone en 2007, son los que han hecho evolucionar la Tercera Revolución Industrial hacia nuevos horizontes, pasando de una mera automatización de los procesos productivos, al objetivo de dotar de inteligencia a los diferentes elementos del proceso productivo mediante el uso de IoT o también denominado internet de las cosas. Esta evolución se puede observar en la ilustración 1.

Es este internet de las cosas, basado en sistemas embebidos, quien nos permite visionar un nuevo concepto de industria en la que se fusiona el mundo físico con el mundo digital, los procesos productivos son muy eficientes, de mucha calidad, con alta productividad y con productos personalizados. Esto es lo que se conoce como industria 4.0.

Para terminar con esta introducción, es necesario recordar el origen de la computación ubicua, ya que representa el paradigma sobre el que se apoya la Cuarta Revolución Industrial. En 1988 Mark Weiser introdujo este término, en él que hacía referencia a la integración de la tecnología informática en nuestro entorno de forma que no se detectaran como elementos diferenciados del resto de los objetos. Weiser definió cuatro principios de la tecnología en un grado de ubicuidad.

- El propósito de un ordenador es a ayudar.
- El mejor ordenador es un ordenador que actúa como sirviente invisible.
- Un ordenador debe de extender nuestro inconsciente y así ayudar en nuestra capacidad intuitiva.
- La tecnología debe de crear calma.

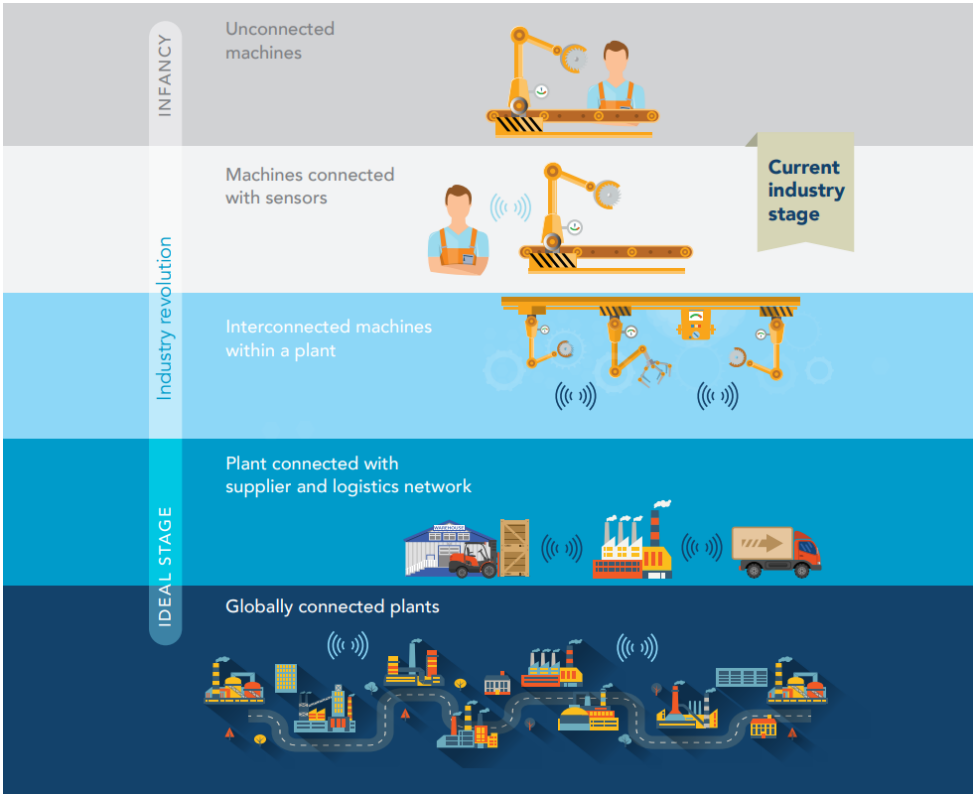


Ilustración 1 Evolución de la industria. Fuente: [7]

Capítulo 2: Objetivos

2 OBJETIVO

El objetivo de este proyecto es el desarrollo de un prototipo de sensorización aplicado a la industria 4.0:

Objetivos principales:

- Desarrollar un sensor con capacidades de comunicación mediante el uso de una placa controladora y un componente de sensorización que satisfaga las necesidades de comunicación, almacenamiento y procesado de datos que el proyecto establezca.
- Desarrollar un software que controle el sensor a modo de sistema operativo de forma que permita la adquisición de datos, su almacenamiento y él envió de los datos a una unidad de procesamiento. Este software se desarrollará de forma modular permitiendo su reutilización para otros propositos.

Objetivo secundario:

- Desarrollar una alternativa viable para la realización del mantenimiento predictivo y la optimización del proceso productivo.
- Estudiar las tecnologías actuales de placas controladoras, sensores, protocolos de comunicación M2M, arquitecturas de redes destinadas a las sensorización, bases de datos destinadas a la sensorización y sistemas de procesamiento de datos, de forma que este conocimiento permita adoptar las tecnologías que mejor se adapten al problema.

Capítulo 3: Estado del arte

3 ESTADO DEL ARTE

Para proseguir en este capítulo primeramente se analizará el contexto actual de la industria 4.0 y sus tecnologías, posteriormente se profundizará en el estado de las tecnologías del ámbito de la sensorización e IoT y por último se realizará un estudio del mantenimiento adentrándose en los tipos de mantenimiento y como la industria 4.0 pretende optimizar este proceso.

3.1 ESTADO DE LAS TECNOLOGÍAS PARA LA INDUSTRIA 4.0

Para comprender el estado de esta Cuarta Revolución Industrial, primeramente, vamos a exponer la situación de algunas de las tecnologías que están permitiendo su desarrollo. La primera vez que se habló del termino IoT fue en 1999, a pesar de ello no ha sido hasta la segunda década de siglo XXI donde se ha comenzado a formar una concepción real de la idea de Internet de las cosas. Así pues, el término IoT refleja el concepto de que cada objeto está permanente conectado a internet y por lo tanto toda su información y toda su capacidad de interactuar con el entorno es accesible desde cualquier parte de internet.

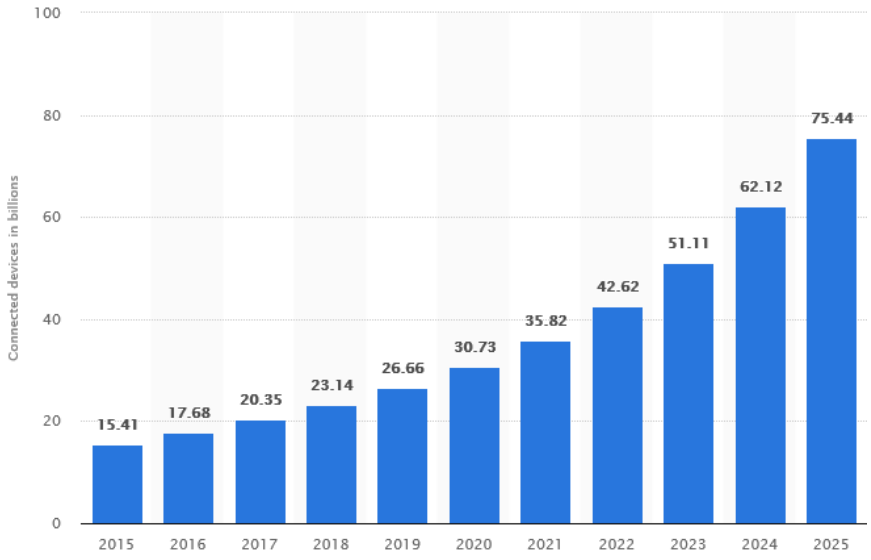


Ilustración 2 Previsión de dispositivos conectados a internet [29]

Como se puede visualizar en la ilustración 2, cada vez, más dispositivos están conectados a internet, obteniendo y almacenando datos. En la ilustración 3 se muestra la previsión de creación de datos hasta 2025. Este almacenamiento de datos a gran escala está generando nuevas técnicas de procesamiento de datos denominadas Big Data. Este nuevo modelo de adquisición de datos está cambiando los modelos de bases de datos tradicionales, de modelos relacionales a modelos no relacionales. Además, se están incorporando nuevos algoritmos de procesado de datos, basados en algoritmos genéticos y redes neuronales de nueva generación, que permiten el denominado Machine Learning.

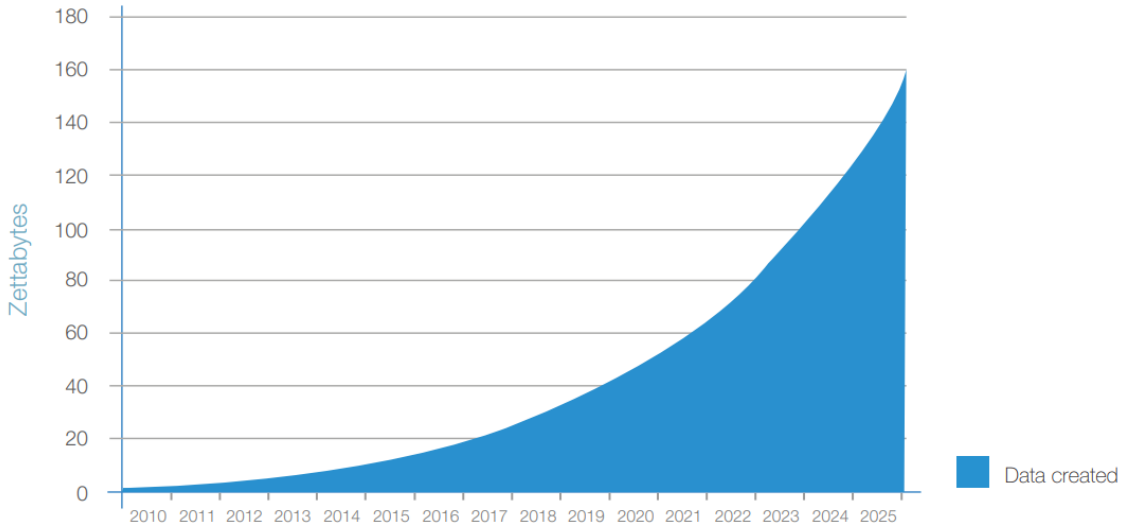


Ilustración 3 Previsión de creación de datos

Mientras, la electrónica es cada vez es más pequeña y más eficiente, posee una capacidad de computación mayor, las baterías son de mayor capacidad y con mejores prestaciones. Todo ello, en conjunto, está acelerando la aparición de más dispositivos IoT. Esta nueva arquitectura de múltiples dispositivos conectados a internet se está apoyando en los protocolos de comunicación denominados M2M (Machine to machine), además de nuevas redes de comunicación inalámbrica de larga distancia y bajo consumo como las que proponen Sigfox o LoRa. La descentralización de los servicios está modificando la arquitectura típica de cliente servidor a arquitecturas P2P, como la arquitectura con topología de red mallada bajo el estándar IEEE 802.11s.

Por otro lado, la capacidad divulgativa de internet y la aparición de placas controladoras programables asequibles, como Arduino en 2005, han generado un ecosistema de colaboración, creación y divulgación de proyectos caseros con diferentes niveles de tecnicidad. El comercio electrónico está acercando este tipo de tecnología a todo el mundo, aumentando cada día la cantidad de sensores y dispositivos conectados.

A partir del desarrollo de la placa controladora Arduino, declarada como hardware libre, han surgido numerosas placas controladoras, cada vez más económicas y con una amplia gama de características; desde placas enfocadas a mejorar las comunicaciones, a micro placas para uso en los denominados wherables (tecnología como prenda de complemento), o placas capaces de concebir las funcionalidades de un ordenador personal completo.

El domino de las comunicaciones inalámbricas, es un factor clave para dominar en el sector de la industria 4.0. El uso de redes LPWAN (Low-Power Wide-Area Network) es un requisito para el desarrollo de redes inalámbricas de larga distancia para sensores. Algunas empresas están desarrollando protocolos LPWAN

propietarios sobre los que funcionan sus placas controladoras, como los protocolos ZigBee, DigiMesh y LoRaWAN, los cuales se expondrán más adelante.

Mientras, en el campo industrial, los procesos productivos van incorporando, con prudencia, estas nuevas tecnologías. Se pueden destacar proyectos como el de la compañía Michelin, que consiste en la instalación de sensores en ruedas que permiten capturar información del uso de las ruedas, permitiendo a Michelin facturar a una empresa de logística o transporte por uso y no por producto como se podía hacer hasta ahora. Otro proyecto ambicioso a destacar en esta cuarta revolución, es el de la compañía Bosch con la monitorización de máquinas, detectando las maquinas lentas y puntos de mejora. Este proyecto permite que con los mismos recursos se pueda mejorar la producción logrando alcanzar un 275% de beneficio en un periodo de un año. También se debe destacar el proyecto de la compañía Siemens con MindSphere, que consiste en un sistema abierto en la nube para internet de las cosas que permite la conexión de máquinas físicas con el mundo digital utilizando técnicas de Big Data.

3.2 ESTADO DE LAS TECNOLOGÍAS PARA LA SENSORIZACIÓN

Una vez se ha expuesto el contexto actual de la industria 4.0 sobre el que se va a desarrollar el prototipo de sensorización, se procede a detallar algunas tecnologías que han sido necesarias para establecer la definición del prototipo. Primeramente, se comentarán las placas controladoras y sus posibles sistemas operativos. A continuación, se realizará un repaso sobre algunos módulos de comunicación junto con arquitecturas de red destinadas a IoT y algunos protocolos M2M; y por último se estudiarán las alternativas en bases de datos destinadas a IoT.

3.2.1 Placas controladoras

Para este proyecto, la placa controladora será el gestor de las operaciones de captación de datos y comunicación de los mismos.

3.2.1.1 ESP8266:

En 2014 Espressif System produjo las primeras versiones de chip ESP8266, desarrollando un chip destinado a IoT. Según el datasheet, las características del chip son:

- Procesador interno de 32 bits con una frecuencia de 80 MHz a 160MHz.
- 1 MB flash, 160KB RAM
- Stack TCPIP.
- F 802.11 b/g/n
- Modo wifi: Station/softAP/Station+SoftAP
- 16 pines digitales y 1 pin analógico
- Voltaje: 2.5 a 3.6 V 80mA

Ilustración 4 Placa controladora NodeMCU

3.2.1.2 BeagleBone Black:

Placa controladora incorporada al merado en 2014 con características de un pc personal de bajo coste. Está basada en su predecesora BeagleBone lanzada en 2011 a la que se le han incorporado mejoras. BeagleBone es un paso más allá de una simple palca controladora destinada a IoT. Permite la instalación de sistemas operativos como Ubuntu, cuenta con gestor de renderizado además de conexión micro HDMI. Las características son:

- AM335x 1GHz ARM® Cortex-A8
- 3D graphics accelerator
- 512MB DDR3
- Micro HDMI
- Ethernet
- 2x46 pines

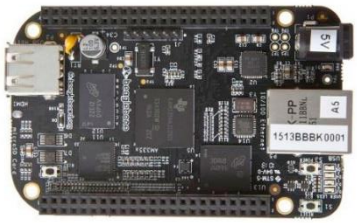


Ilustración 5 Placa controladora BeagleBone Black

3.2.1.3 Particle Photon:

Placa controladora con capacidades de comunicación desarrollada por Particle. Su principal factor destacado es la garantía de calidad ya que se basa en la utilización del chip BCM43362 probado a nivel industrial.

- STM32 ARM Cortex M3
- BCM43362 Wi-Fi
- 1MB flash, 128KB RAM
- 18 pines GPIO
- Sistema operativo (FreeRTOS)
- Voltaje: 3.3V

Ilustración 6 Placa controladora Particle Photon development kit

3.2.1.4 RaspBerryPi:

A pesar de que sus orígenes se remontan a 2006, la placa controladora Raspberry Pi comienza su venta en 2012. Esta placa fue desarrollada por la fundación Raspberry Pi la cual fue fundada en 2009 como una asociación sin ánimo de lucro. Desde esta primera versión en 2012 se han introducido diferentes mejoras. A continuación, se expone algunas de las características de la penúltima versión del modelo B:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- WiFI (BCM43438)
- Bluetooth Low Energy (BLE)
- 40 pines GPIO
- 4 puertos USB
- Puerto HDMI
- Micro SD

Ilustración 7 Placa controladora Raspberry Pi 3 Model B

3.2.1.5 LinkIt ONE:

Linklt ONE es un diseño de Seeed Studio y MediaTek. Se trata de una placa controladora con capacidades de comunicación 100% ideada para IoT. La placa contiene capacidades de comunicación GSM, GPRS, Bluetooth, Wifi incluso GPS. Sus características más destacadas son:

- Procesador ARM7 MT2502A a 260MHz
- 4MB de RAM
- 16 Pines GPIO
- Voltaje: 3.3V
- GPS (MT3332): GPS/GLONASS/BEIDOU
- GPRS: Class 12
- GSM: 850/900/1800/1900MHz
- Wi-Fi (MT5931): 802.11 b/g/n
- Lector de tarjetas SIM/SD hasta 32 GB clase 10

Ilustración 8 Placa controladora LinkINE ONE

3.2.2 Sistemas operativos para placas controladoras

Como se expondrá a lo largo del desarrollo, la complejidad de la incorporación de diferentes módulos requirió de la elección de un sistema operativo. Es por ello, que en este apartado se exponen un análisis los sistemas operativos para las diferentes placas controladoras.

Dependiendo del uso para el que este destinado la placa controladora, se pueden pensar en emplear un sistema operativo en tiempo real o en un sistema operativo de propósito general.

Existen diversas formas de implementar este tipo de sistemas operativos, la más simple es el procesamiento secuencial de las funciones mediante un bucle. Un ejemplo de este tipo de implementación es la que nos proporciona el IDE de Arduino con una función setup() de configuración, y una función loop() que representa a este bucle secuencial. El problema de este tipo de codificación es la latencia que se produce en la ejecución, ya que los eventos asíncronos no se pueden recoger hasta que no se pasa por la función que espera ese evento. Po otro lado la ejecución de tareas bloqueantes, bloquea, la ejecución de todo el sistema.

Una alternativa que permite solucionar este problema, a nivel software, es el uso de interrupciones a nivel software con timers, o a nivel hardware, con la incorporación de un gestor de interrupciones.

Otra alternativa, es el uso de los denominados RTOS (Real Time Operating Sistem). Este tipo de sistemas operativos realizan un interfaz entre el hardware y el software. El sistema gestiona cuando se ejecuta cada tarea, y la cantidad de memoria de cada tarea. Dentro de este tipo de sistemas operativos existen dos variantes, sistemas cooperativos donde el programador es quien define si dejar libre el uso del procesador y los sistemas apropiativos, donde el gestor de tareas decide cuanto tiempo se ejecuta cada tarea y su prioridad de ejecución. Analizando las ventajas e inconvenientes, se concluye que el uso de sistemas RTOS permite reducir el tiempo de latencia significativamente, aumenta la modularización del sistema, asegura la independencia de los procesos además de establecer vías de comunicación seguras. Por otro lado, el uso de RTOS requiere un mayor uso de memoria RAM en concreto el uso de un sistema apropiativo donde existe un gestor de tareas que aumenta el uso de estos recursos. Algunos de los RTOS que actualmente podemos encontrar son:

- FreeRTOS: Es un RTOS de código abierto
- Mongoose OS

3.2.3 Módulos de comunicación

Siguiendo con el estudio previo de las tecnologías, en este apartado se especifican algunos de los módulos que actualmente se ofrecen en el mercado:

XBee: Algunas de sus características son: • Usa el estándar IEEE 802.15.4 Zigbee • Funciona a 3.3 V	
Sigfox module: Algunas de sus características son: • Frecuencia: ISM 868 que requiere de licencia.	
LoRa RN2483, RFM98W, SX1276: Algunas de sus características son: • Frecuencia de 433 MHz, 868 MHz y 915 MHz • Funciona en capa 2	
Nordic semiconductor module (NRF2401): Algunas de sus características son: • Opera en la banda de 2.4 Ghz • Usa el bus SPI • Rango hasta 50Km	

3.2.4 Protocolos de M2M

Para llevar a cabo la comunicación entre los diferentes elementos del sistema de sensorización, se requiere establecer un protocolo de comunicación. Para ello en este apartado se exponen los diferentes protocolos de comunicación entre maquinas mayoritariamente utilizados en la actualidad.

3.2.4.1 XMPP (Extensible Messaging and Presence Protocol):

En 1999 Jeremie Miller anuncio el protocolo Jabber. Años después, este protocolo destinado para chats fue el predecesor del actual XMPP. A partir de la definición de sus siglas y de la información de su página web [1] del protocolo se puede extraer algunas características:

Se trata de un protocolo destinado a la comunicación en tiempo real que utiliza como formato de comunicación documentos XML. Además este formato define el concepto de presencia informado cada nodo al servidor de su estado (online, offline u ocupado).

3.2.4.2 MQTT (Message Queuing Telemetry Transport):

Protocolo basado en la suscripción y publicación de mensajes. Utiliza un servidor denominado broker encargado de gestionar y notificar a los nodos subscriptores cuando hay una nueva publicación. Utiliza un sistema en jerarquía basado en temas ("Edificio1/panta2/sala1/raspberry3/humedad"). Para asegurar las comunicaciones utiliza TLS. Actualmente se encuentra publicado bajo el estandar ISO/IEC 20922:2016 según se especifica en su web [2].

Un ejemplo del uso MQTT para un proyecto de sensorización se puede encontrar en la siguiente referencia [3].

3.2.4.3 CoAP The Constrained Application Protocol

Se trata de protocolo definido en el rfc7252 [4] y destinado a redes de baja energía LowPAN o redes con mucha perdida de paquetes, sus mensajes se encapsulan en paquetes UDP de poco tamaño. CoAP hace uso de dos tipos de mensajes, peticiones y respuestas. Dentro de las peticiones disponibles se puede realizar una suscripción con el método *observe* que informa del estado de un recurso. Además, CoAP soporta descubrimiento descentralizado multicast usando el método *discover*.

A nivel de seguridad permite el uso de DTLS Datagramas TLS/SSL, ECC (Elliptic Curve Cryptography), ECDH, ECDSA y AES.

3.2.5 Arquitecturas de red Redes

Una vez especificados algunos de los protocolos de comunicación que podrían ser utilizados para el sistema de sensorización, a continuación, se detallan algunas de las topologías de red típicamente utilizadas para sistemas de sensorización.

3.2.5.1 Redes malla

Las redes Mesh o redes malla es una topología de red que usa los criterios de las redes ad hoc como la descentralización de los puntos de acceso. La diferencia entre las redes malla y las redes ad hoc se basa en que un nodo localizado en una red ad hoc, que no tiene visibilidad con otro nodo, no puede comunicarse directamente mediante un intermediario, es decir, no permite rutas multi salto. Sin embargo, en una red malla los nodos pueden comunicarse con todos los nodos, bien de forma directa si se encuentra en el rango de alcance de la conexión Wireless, o a través de otros nodos.

Por otro lado, las redes malla realizan automáticamente un mantenimiento y organización de la red de forma que, si se añaden o desaparecen nodos, se modifican las rutas automáticamente.

Algunas ventajas que nos proporciona este tipo de red son la rápida incorporación de nodos gracias a la gestión automática de la red, la mejora de la cobertura inalámbrica haciendo más accesible la red a los nodos, o las comunicaciones P2P entre nodos cercanos, ya que la responsabilidad de la comunicación no está centralizada, sino que recae sobre todos los nodos de la red.

Por otro lado, al implementar este tipo de redes se encuentran algunos problemas. Según algunos estudios mencionados en el artículo [5], para lograr una comunicación ideal en la red malla, los nodos han de estar en contacto con otros 6 nodos cercanos. Por este motivo en redes grandes, pueden surgir problemas de escalabilidad ya que las tablas de ruta se pueden hacer infinitas. Otro problema asociado es el consumo energético y computacional, ya que todos los nodos han de tener habilidades de enrutamiento que deben

permanecer continuamente activas, lo que es contradictorio al diseño de sensores de baja capacidad computacional y bajo consumo energético puesto que elimina los periodos de reposo.

Protocolos:

- B.A.T.M.A.N (Better Approach To Mobile Adhoc Networking)
- Estándar IEEE 802.11s
- OLSR (Optimized Link Satate Routing)
- WDS (Wireless Distribution System)

3.2.5.2 Redes WPAN (Wireles Personal Area Network):

Dentro de este apartado podemos encontrar tecnologías como Bluetooth, WIFI en la banda de 2.4 o 5 GHz o tecnologías como ZigBee en la que se profundizará a continuación.

ZigBee:

Es un conjunto de protocolos basados en el estándar IEEE 802.15.4. ZigBee permite establecer redes con alcances de 10 a 100 metros con un consumo energético muy bajo además de una baja transferencia de datos. Este tipo de redes se componen de tres dispositivos:

- **Coordinador**: Dispositivo encargado de gestionar la red.
- **Router**: Dispositivo encargado de la interconexión de los dispositivos terminales.
- **Dispositivo final:** Dispositivo encargado de ofrecer la funcionalidad, no participa en gestionar las comunicaciones como ocurre en los nodos de las redes malladas. Este dispositivo únicamente se comunica con routers y coordinadores. Esto le permite activar el modo hibernación y ahorrar energía.

A nivel de seguridad permite la utilización de un cifrado simétrico de 128bits.

3.2.5.3 Redes LPWAN (Low-Power Wide-Area Network)

Las redes de área amplia y baja potencia, están diseñadas para permitir la comunicación con dispositivos a baja velocidad de bits, pero en rangos de distancia grandes.

SIGFOX:

Utiliza la modulación D-BPSK, con un bitrate 100bit/s. Además, utiliza la tecnología utra narrow band que le permite gran robusted frente a interferencias. Las estaciones son muy sensibles (-142 dbm.). Requiere de una licencia para su uso.

DIGIMESH:

DigiMesh proporciona un topologia de red mallada de tipo P2P. Todos los dispositivos son del mismo tipo y poseen las mismas capacidades de comunicación, ahorro energético, optimización. No requiere de otros elementos como hemos visto en las redes ZigBee.

LoRaWAN:

LoRaWAN es un protocolo destinado a redes de bajo consumo y largo alcance operando en la banda ISM de 868MHz. Este protocolo está destinado a gestionar la comunicación entre las pasarelas y los dispositivos finales. A diferencia de los otros protocolos de tipo red mallada, LoraWAN consta de una topología en estrella. En cuanto al nivel de seguridad ofrece cifrado AES 128.

3.2.6 Bases de datos

Para la realización de este proyecto analizaremos las ventajas e inconvenientes de las siguientes bases de datos:

Nombre	Descripción	Modelo Relacional	SQL	Escalabilidad
PostgreSQL	Base de datos open source con una trayectoria de mas de 30 años de desarrollo	SI	SI	En las nuevas versiones se ha mejorado este aspecto adaptando la tendencia a las bases de datos no relacionales
Mongo DB	Base de datos open source orientada a documentos	NO, orientada a documentos. Trabaja con documentos JSON	NO	Alta
Elasticsearch	Base de datos conceptuada para el almacenamiento y análisis en tiempo real de la información.	NO, orientada a documentos. Trabaja con documentos JSON	NO, sin embargo, proporciona una REST full API	Alta
Influx	Base de datos conceptuada para el almacenamiento y análisis en tiempo real de la información. Está programada en GO	NO	NO	Alta

3.3 EL MANTENIMIENTO Y LA INDUSTRIA 4.0

Como se ha expuesto a lo largo de este documento, uno de los objetivos de la industria 4.0 es la optimización de los procesos productivos. Este proyecto pretende optimizar el proceso de mantenimiento mediante el denominado mantenimiento predictivo. Para conocer este proceso y su estado actual, vamos a profundizar siguiendo como referencia documento [6].

3.3.1 Introducción al mantenimiento

El mantenimiento es una fase necesaria en el proceso productivo ya que la calidad y la propia cadena de producción depende que la maquinaria esté en condiciones óptimas para desarrollar su trabajo. Pero el mantenimiento no siempre ha sido entendido como se concibe en la actualidad. Hasta 1925, las

técnicas de manteniendo que se utilizaron eran técnicas correctoras o también denominado mantenimiento correctivo, en las que se esperaba hasta que el sistema presentaba un fallo para aplicar la corrección adecuada La aplicación de ésta técnica, suponía que el proceso productivo se podía parar de manera arbitraria en cualquier momento, añadiendo a la producción unos costes asociados, viéndose afectada la calidad de los productos o servicios y que se pusiera en riesgo la salud de los trabajadores. Tras estas circunstancias se pensó que el emplear técnicas de base científica podrían mejorar significativamente el proceso productivo.

3.3.2 El mantenimiento en la actualidad

En la actualidad se realiza el denominado mantenimiento periódico o mantenimiento preventivo, que consiste en la realización de inspecciones regulares, pruebas o reparaciones con el objetivo de reducir la frecuencia y el impacto de los fallos. Pero este mantenimiento provoca cambios innecesarios, problemas iniciales de operación, que se producen al poner piezas nuevas; el coste de mantenimiento de inventarios de piezas y el coste de la mano de obra.

Bajo estas circunstancias, puede darse la situación en que la maquinaria esté a punto de fallar y que no haya un mantenimiento planificado a corto plazo, o que la maquinaria esté en condiciones óptimas y se le realice un mantenimiento planificado. Todas estas casuísticas aumentan el coste de mantenimiento.

Para comprender mejor el efecto del mantenimiento preventivo y poder cuantificar el coste del proceso se muestra la siguiente ilustración:

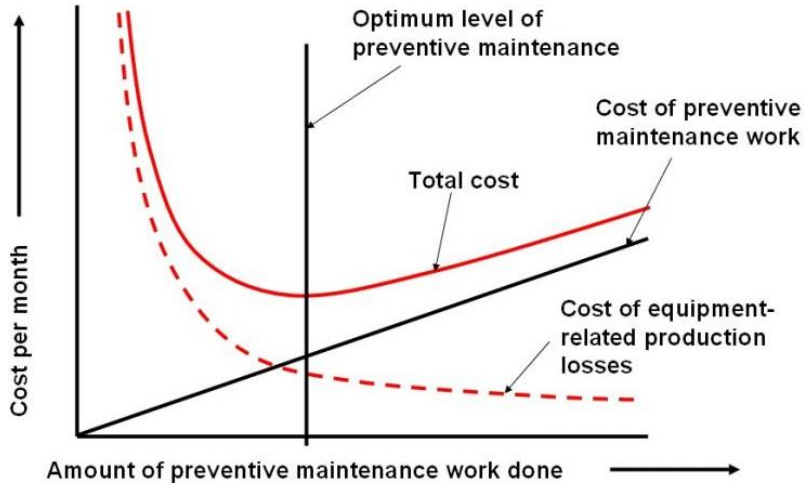


Ilustración 9 Costes de mantenimiento [28]

Como se puede apreciar en la ilustración, es complejo establecer el punto óptimo de mantenimiento, ya que si se hace poco mantenimiento los costes de fallo son muy grandes mientras que si se realizan demasiadas tareas de manteniendo los costes aumentan.

Para dar solución a este problema, la industria 4.0 proporciona el concepto de mantenimiento predictivo a fin de mejorar la eficiencia de este proceso. El mantenimiento predictivo se basa en la captación y procesado de los datos proporcionados por los sensores en la maquinaria, permitiendo una monitorización y seguimiento de la evolución de los fallos.

Estos fallos no tienen una evolución lineal respecto al tiempo, si no que la progresión define una forma denominada como "curva de bañera" en la que se pueden distinguir 3 etapas de fallo. En la siguiente figura se muestra su representación.

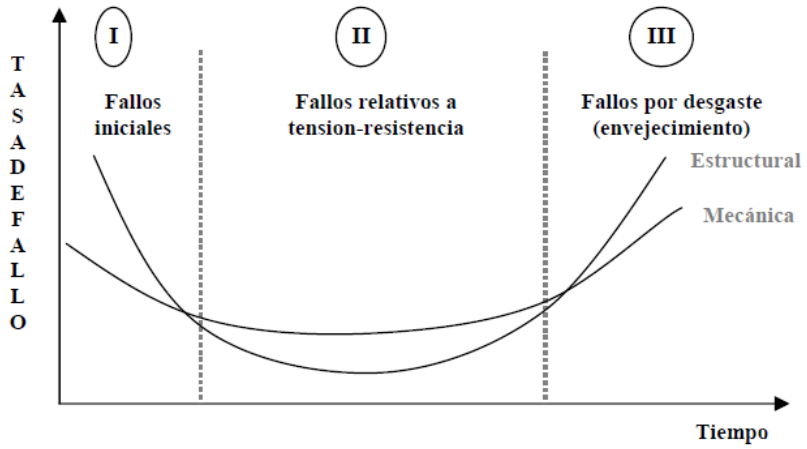


Ilustración 10 Curva de bañera [6]

- **Etapa de juventud (I)**: Esta etapa se caracteriza porque el fallo aparece poco después de su puesta en marcha. El tipo de fallos en esta fase son: errores de diseño, defectos de fabricación o malos reglajes en la puesta a punto.
- **Etapa de madurez (II):** Esta etapa se caracteriza por ser un periodo de estabilidad con pocos fallos donde la aparición de fallos es de carácter aleatorio.
- **Etapa de envejecimiento (III):** Esta etapa se caracteriza por tener al igual que en la etapa de juventud un alto índice de fallo, en la mayoría de los casos debido al deterioro y envejecimiento de los materiales y sistemas.

Profundizando en el mantenimiento predictivo o también denominado mantenimiento según condición o estado, es el mantenimiento más flexible y eficiente, pero no siempre se puede aplicar. Existen determinadas circunstancias de fallo que no pueden ser detectadas con antelación, por ejemplo, una sobre carga de tensión en el sistema o un error en la operación de un operario. Además, los sistemas de monitorización son muy costosos. Pero es aquí donde la industria 4.0 pretende aportar una solución, añadiendo más sensores gracias a su abarramiento, y de menor tamaño para así controlar y acotar la detección de fallos.

3.3.3 Tipos de mantenimiento predictivo

Para conocer mejor el mantenimiento predictivo, a continuación, se comenten los dos tipos dependiendo del tipo de control o técnica de monitorización:

- Control sin interrupción: El sistema se monitoriza sin interrupción del proceso productivo.
    - Existen diversas técnicas de control sin interrupción, entre ellas la monitorización de la temperatura, monitorización de lubricantes, monitorización de ruidos, monitorización de corrosión o monitorización de la vibración.
- Control con interrupción: Requiere poner el sistema en condiciones anormales de funcionamiento pudiendo requerir la parada del sistema.

- Detección de fugas mediante ultrasonidos, ensayo de vibraciones, ensayo con corrientes inducidas, ensayo de resistencia eléctrica, ensayos radiográficos…

3.3.4 Ventajas e inconvenientes del mantenimiento predictivo:

A modo de resumen, una vez se ha introducido el concepto de mantenimiento, se han comentado las diferencias entre el mantenimiento preventivo y predictivo, y se han definido los dos tipos de mantenimiento predictivo; se exponen a continuación las ventajas que proporciona el mantenimiento predictivo frente al mantenimiento preventivo.

Ventajas:

- Periodos mayores entre paradas.
- Las averías estarán más acotadas.
- No se requerirá de un stock de piezas, ya que se dispondrá de seguridad frente al funcionamiento dela maquinaria.
- Reducción de las paradas no planificadas.
- Eliminación de los daños secundarios.
- Reducción de las interrupciones de producción frente a SLA (Service-level agreement).
- Bonificaciones en seguros.

Para cuantificar las ventajas del mantenimiento predictivo se muestra una gráfica que representa la reducción de los costes con la aplicación del mantenimiento predictivo.

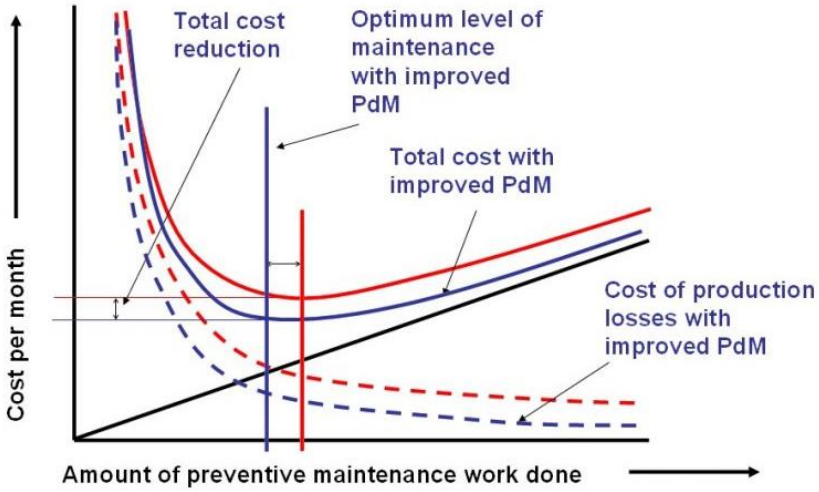


Ilustración 11 Mejora de los costes aplicando el mantenimiento predictivo [28]

Inconvenientes:

A pesar de las ventajas que supone el mantenimiento predictivo, se pueden considerar algunas tareas y costes que requiere su uso como inconvenientes:

- Selección y compra de instrumentos de medición.
- Selección de sistemas expertos o herramientas de Big Data para el análisis de la información.
- Coste de almacenamiento de los datos captados.
- Formación de los técnicos de mantenimiento sobre los nuevos sistemas de monitorización.

Capítulo 4: Desarrollo

4 DESARROLLO

Como se ha mencionado en el capítulo 2, este proyecto pretende desarrollar un prototipo de sensorización que permita realizar el mantenimiento predictivo de la maquinaria industrial y la optimización del proceso productivo. Para realizar este desarrollo, se necesitará primeramente un sensor inalámbrico que se colocará en la maquinaria. Este sensor estará formado por dos partes, una placa controladora junto un el módulo de comunicación, y un elemento de medición, pudiendo este cambiarse en función de la información que se desee medir. En segundo lugar, se necesitará de un sistema intermediario a modo de interfaz entre la red de sensorización y el servidor. Para esto se seleccionará una red de comunicación segura junto con un protocolo de comunicación M2M. Por último, se necesitará una base de datos donde almacenar los datos recopilados, y un sistema para procesar los datos recogidos por el sensor transformándolos en información.

4.1 DISEÑO

En esta parte del documento, se pretende establecer el diseño del prototipo. Para ello, primeramente se contextualizará un problema modelo para el cual diseño aportará una solución.

4.1.1 Contexto y problema concepto

Esta conceptualización de un problema particular no pretende definir al prototipo como solución única para este tipo de problemas, sino que el prototipo se diseñará de forma modular pensando en su reutilización para otro tipo de problemas del mismo ámbito.

El contexto del problema sobre el que se va a trabajar y que posteriormente se podrá llegar a modelizar, está basado en un entorno industrial con unos motores de ventilación eléctricos de gran tamaño y alto rendimiento que se encuentran en un subterráneo. Sin adentrarnos en los aspectos técnicos, este entorno conceptualizado será un espacio pequeño y con alta predisposición a la aparición de interferencias cuando la maquinaria esté en funcionamiento. Hay que destacar que la maquinaria no estará en constante funcionamiento si no que funcionará durante un determinado horario.

Bajo este contexto, el problema consistirá en la aportación de un método que permita mejorar proceso de mantenimiento de dicha maquinaria aportando una solución de industria 4.0.

Siguiendo los pasos enumerados en el artículo *From concept to production: a 5-step approach towards successful Industry 4.0 projects, Frost and Sullivan* [7], se exponen a continuación los dos primeros pasos que corresponden a nuestro objetivo de la elaboración de un prototipo.

4.1.2 Análisis de objetivos de negocio

Para Frost y Sullivan, el primer paso consiste en determinar los nuevos objetivos de negocio mejorando los procesos productivos y eliminando los procesos de la cadena de valor que no añadan valor al producto. Como se ha analizado en el capítulo 3, en la actualidad la industria usa el denominado mantenimiento preventivo. Para mejorar este proceso, este proyecto se apoyará en el uso del mantenimiento predictivo.

4.1.3 Definición del prototipo

Siguiendo la metodología de los cinco pasos que nos facilita Frost y Sullivan [7], el siguiente paso es la definición de un prototipo que posteriormente nos permita extrapolar la información a toda la producción. Para este aparado Frost Y Sullivan definen cuatro puntos que deberá de contener el proyecto:

- Los dispositivos han de estar conectados a internet de forma que permitan enviar información a una plataforma en la nube
- La nube analizara la información recibida por los usuarios, dispositivos y sensores.
- La seguridad de la información es vital en todos los ámbitos del proceso productivo para evitar espionaje industrial.
- Usar una nube publica para esta prueba de concepto permitirá reducir costes. Posteriormente ya se podrá pensar en la implementación de esta nube en servidores propios.

Combinando estos puntos con la teoría del mantenimiento. Este proyecto tomará como mantenimiento predictivo un control sin interrupción, seleccionado la técnica de la monitorización de la vibración. Esto se debe a la vibración nos permite detectar una amplia gama de fallos en el sistema desde desequilibrios, desalineamientos, fallos de desgaste, fallos de lubricación… Aunque la monitorización de la vibración no es la única variable que nos permite detectar este tipo de fallos. El ruido es también una variable que nos proporciona mucha información acerca del estado del sistema y es más sencilla de medir, ya que no requiere montar el sensor sobre la máquina. Sin embargo, existen los denominados ruidos parásitos que provocan interferencias de ruido y hacen que la medición de esta variable sea de menor precisión.

El aparato de medida que permite la monitorización es un transductor de vibración, el más común un acelerómetro piezoeléctrico, aunque existen otros tipos como traductores de vibración de bobina móvil. Es importante establecer la localización del sistema de medición ya que de ello dependerá la calidad de los datos recopilados. Siguiendo el documento *Vibraciones en Maquinas 2 del Máster de Ingeniería industrial del Dr. Higinio Rubio Alonso.* [8], es importante situarlo sobre un componente que tenga una frecuencia natural elevada como un componente rígido. Otra opción es situarlo en los soportes de los cojinetes.

Una vez profundizado en los detalles del sensor, definimos las tres partes diferenciadas del prototipo realizando un recorrido ascendente desde la fase de adquisición de datos hasta la fase de procesado.

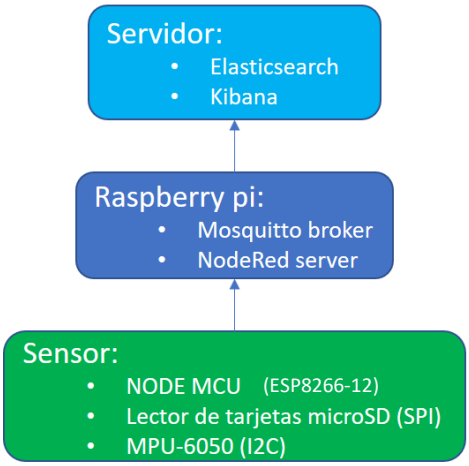


Ilustración 12 Diseño del prototipo

Comenzando por la sección de adquisición de datos, nos encontramos con el elemento sensor. Este está compuesto por tres componentes, en primer lugar, se encuentra el micro controlador NodeMCU que incorpora las habilidades control con capacidades de comunicación. La elección de este microcontrolador para el sensor está supeditado al uso de una red bajo el estándar IEEE 802.11. En segundo lugar, para captar la vibración el sensor se apoyará en el uso de un módulo de medida inercial MPU-6050 que se comunicará mediante el bus I2C. La elección de este sensor se debe a su alta precisión y capacidad de detección en seis ejes ya que incorpora un acelerómetro y un giroscopio. Además, incorpora un termómetro que permite la realización de posibles correcciones en la medición de los datos. Esta información se enviará mediante MQTT al bróker configurado en la Raspberry Pi. Para poder almacenar toda la información, ya que como comentamos en la descripción del problema modelo, se trata de un entorno industrial con alta predisposición a la aparición de interferencias. Para controlar este problema, se incorpora un lector de tarjetas micro SD que se comunicara con el microcontrolador mediante el bus SPI.

Continuando de manera ascendente, nos encontramos con la segunda parte del prototipo que sirve de interfaz para incorporar los datos adquiridos a la base de datos. Este interfaz se compone de una Raspberry Pi que como comentamos en el capítulo 3 en la sección de tipos de microcontroladores, este microcontrolador pose cualidades de un mini ordenador. Sobre él se ha instalado la versión oficial de Linux llamada Raspbian en la versión abril 2018. En su interior este mini ordenador contiene instalado el bróker necesario para las comunicaciones M2M mediante el protocolo MQTT. La versión de bróker elegida es Mosquitto en la compilación 1.4.10. Este bróker será el encargado de gestionar las comunicaciones entre el sensor y la herramienta de desarrollo, Node-Red.

Node-Red es el núcleo del interfaz entre el sensor y el servidor de almacenamiento y procesado de datos. En él se encuentran los nodos necesarios que permiten la suscripción al bróker y el envió de los datos recopilados al servidor mediante JavaScript y programación tipo scrach.

Finalmente, la última sección del prototipo es el servidor. Actualmente el servidor está instalado en una máquina virtual pero como comentaremos en el apartado de resumen, conclusiones y líneas futuras este servidor se podrá instalar en una maquina en la nube para aumentar sus prestaciones de computación y almacenamiento.

El servidor contiene una instalación de la base de datos en tiempo real Elasticsearch con los módulos: Kibana que sirve de "front end" y el módulo x-pack en versión de prueba que incorpora opciones de procesado de datos mediante técnicas de maching learning, con él se realiza la parte de procesado de datos para detectar fallos de mantenimiento predictivo y optimización del proceso productivo.

4.2 IMPLEMENTACIÓN

Debido a que la metodología utilizada para la gestión del proyecto es Scrum, esta fase de implementación se ha regido por la realización de Sprints y la propuesta de versiones, cada vez más evolucionadas.

4.2.1 Implementación de un sistema operativo para el sensor

Una vez analizados los sistemas operativos para microcontroladores en el capítulo 3, apartado sistemas operativos para la sensorización, se toma la elección de implementar un sistema operativo para el sensor de forma modular permitiendo así la reutilización de código.

4.2.1.1 Entornos de desarrollo

Antes de comenzar la exposición sobre la implementación, realizaremos una breve introducción sobre la evolución que se ha experimentado en el uso de diferentes entornos de desarrollo, a medida que la complejidad del problema evolucionaba.

Las primeras versiones del código se realizaron con el entorno de desarrollo que proporciona Arduino, Arduino IDE agregando las librerías que permitían compilar el código para el microcontrolador el ESP8266. Para usar esta librería se siguieron los pasos que se describen a continuación:

Agregar la dirección del repositorio:

Para ello se accede a la pestaña *"File/Preferences"* y en el apartado *"Aditional Boards Manager URL"* se agrega la siguiente dirección web según se muestra en la ilustración:

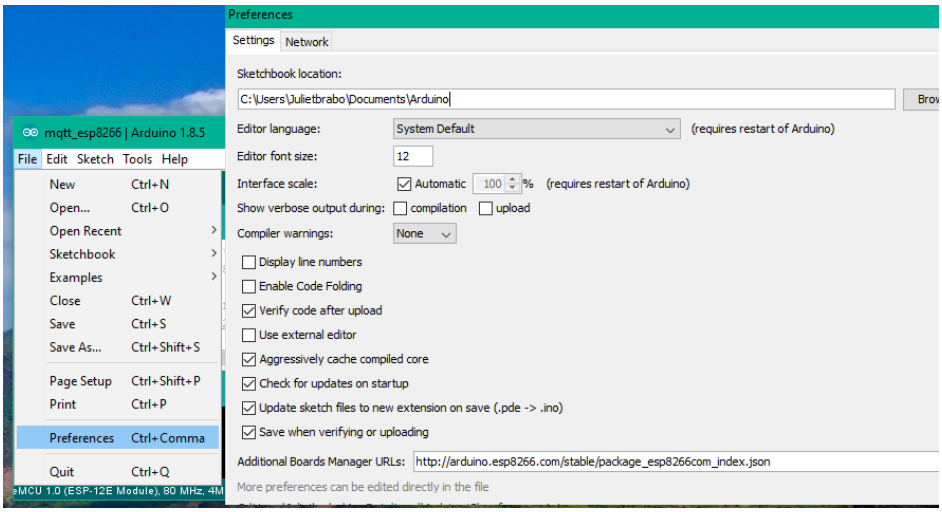


Ilustración 13 Configuración del repositorio del ESP8266 en Arduino IDE

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Una vez realizada esta operación hay que acceder a la pestaña *"Tools/Board/BoardManager"* donde deberemos buscar e instalar las librerías.

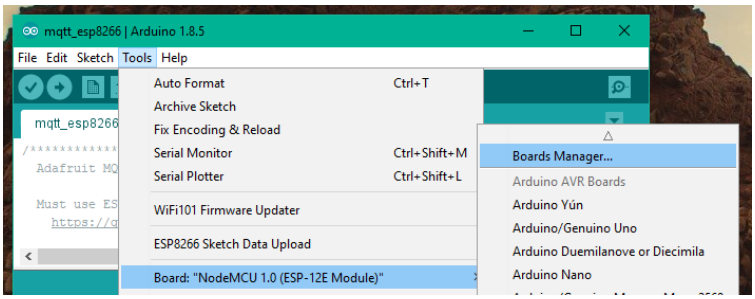


Ilustración 14 Ruta para la instalación de las librerías de ESP8266 en Arduino IDE

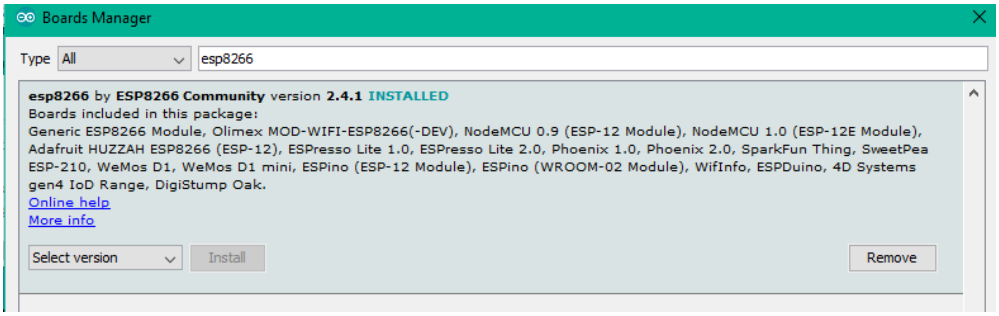


Ilustración 15 Instalación de las librerías ESP8266 en Arduino IDE

Una vez realizada la instalación, en el menú de *"Tools"* seleccionamos la placa controladora ESP8266, para este proyecto la versión NodeMCU 1.0.

Antes de continuar exponiendo las primeras versiones del sistema operativo, vamos a proceder a comentar el segundo IDE, Atom que se ha utilizado en las últimas versiones del proyecto. El cambio se debió como se comentará posteriormente, a la elección del lenguaje de programación de C++ frente a Arduino.

El IDE Atom es un entorno de desarrollo que soporta diferentes lenguajes. La elección de este entorno se debió principalmente al soporte que presenta para un ecosistema con las herramientas necesarias para el desarrollo de programas para el ESP8266 en C++ y otros lenguajes. Este ecosistema se llama Platformio. Su instalación y configuración se muestra a continuación:

Siguiendo las indicaciones para la instalación del plugin que se proporcionan en la página web de Platformio [9]:

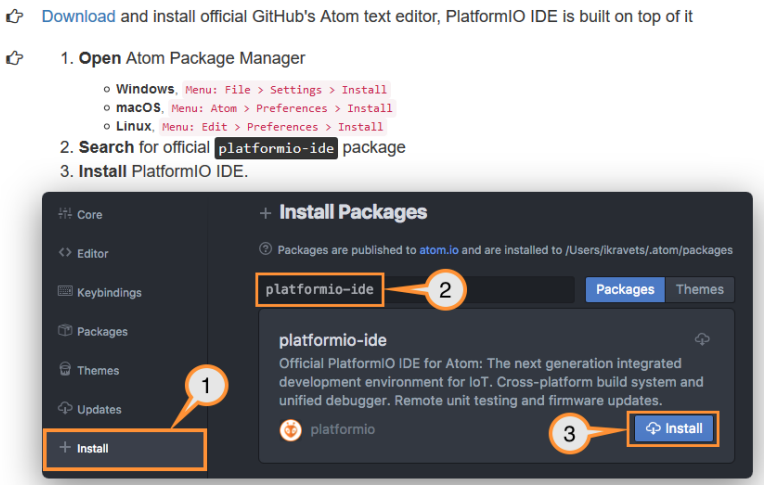


Ilustración 16 Captura de pantalla pasos intalacion Platformio en Atom

Una vez instalado el entorno posee todas las herramientas de depuración, compilado y carga necesaria para implementar el proyecto en C++.

4.2.1.2 Primeras versiones

Como se comentará a lo largo de las próximas secciones, la metodología utilizada para implementar las versiones ha consistido en la incorporación de módulos y funciones de forma progresiva. Estas primeras versiones se implementaban en el lenguaje de programación de Arduino sobre el entorno Arduino IDE. Al usar el lenguaje de programación de Arduino, el paradigma que nos proporciona inicialmente es un paradigma estructurado al estilo c con modificaciones al estilo C++, pero sin llegar a un paradigma orientado a objetos.

El propósito de estas primeras versiones era explorar los métodos, funcionalidades y atributos de las diferentes librerías del ESP8266 trabajando con elementos conocidos.

Así pues, la primera versión del proyecto consistió en un programa que permitía al NodeMCU conectarse a la red y hacer parpadear el led que lleva incorporado.

La siguiente versión tuvo grandes progresos, se implementó un programa que permitía él envió de información de un sensor DHT22 sobre el cual se poseía experiencia, mediante el protocolo MQTT. De esta forma esta segunda versión se exploraron las librerías de *"ESP8266WiFi.h", "Adafruit_MQTT.h", "DHT.h"* sobre el ESP8266. El hecho de utilizar un sensor que no era el destinado para este proyecto, pero sobre el cual se poseía experiencia, permitió fortalecer el concepto de alta cohesión y bajo acoplamiento. Debido esto, esta versión introdujo el concepto de modulo, pero no como concepto de clase, si no como sección del código que incorporaba las funciones destinadas a una tarea. Los módulos que se incorporaron a esta versión fueron: módulo MQTT, módulo wifi, módulo DHT y módulo Debug. A continuación, se muestra un ejemplo del concepto de módulo utilizado:

```
//-----------------------------------MODULO DEBUG--------------------
int contM = 0;
const bool DEBUG = true;

void iniciarModuloDebug() {
  Serial.begin(115200);
  delay(10);
}

void ejecutarModuloDeBug(String mensaje) {
  if (DEBUG) {
    Serial.println(mensaje);
  }
}

void setup() {
  iniciarModuloDebug();
  ejecutarModuloDeBug("\n\nSENSOR> Iniciando configuracion prototipo:...");
```

Ilustración 17 Ejemplo de implementación de modulo sobre el entorno de desarrollo Arduino IDE

Como se puede observar en esta ilustración, la definición era muy similar al utilizado para definir una clase, pero sin la encapsulación que esta define.

La siguiente versión que se implementó, no genero grandes avances, pero si permitió explorar el concepto de programación OTA (*Over the air*). Este concepto permite la carga remota de un firmware en el microcontrolador. De esta manera, cuando el sensor este posicionado en la zona de medición, si se quiere realizar una actualización del firmware, no se necesitará desacoplarlo para conectarlo al puerto serie del

ordenador. Además, como hemos observado, la carga del firmware a través de OTA es más rápida que mediante el interfaz USB.

Respecto a la seguridad, la librería nos proporcionaba funciones para establecer una capa de seguridad añadiendo una contraseña en texto plano o almacenando el hash en md5, siendo esta la opción elegida para esta versión.

Hay que destacar, que para que el módulo OTA funcionase, en la función loop() (Función de repetición implementada para sistemas operativos en tiempo real como comentamos en el capítulo 3 en la sección Sistemas operativos para placas controladoras) que nos proporciona Arduino para la programación de microcontroladores, había que añadir una función de escucha al puerto establecido para detectar si se realizaban solicitudes de actualización mediante OTA.

Para poder cargar el nuevo firmware en el microcontrolador previamente hay que realizar una configuración del entorno según se describe en el enlace [10]

En resumen, en esta versión se exploró la funcionalidad de las librerías: *"ESP8266mDNS.h", "WiFiUdp.h", "ArduinoOTA.h"*.

La siguiente versión profundizo en las funcionalidades de las librerías MQTT con la incorporación de no solo publicaciones, sino también una suscripción. Esta suscripción permitía desde un cliente publicar una petición de actualización de la información del sensor DHT. Además, en esta versión se implementaron sistemas de temporización que permitían ejecutar una tarea tras el paso de un número determinado de milisegundos. Para ello se calculaba con el método millis() los milisegundos en el instante de cada ejecución, y si los milisegundos de la nueva ejecución menos los milisegundos de la anterior ejecución, el intervalo era mayor de un umbral, la tarea se ejecutaba.

Esta versión fue la última en la que se utilizó el sensor de temperatura y humedad DHT. En la siguiente versión se tomó la estructura generada de módulos y se sustituyó el módulo DHT por el módulo encargado de medir la vibración, módulo IMU (Inertial measurement unit) que utilizada el sensor MPU6050. La librería que nos permitió trabajar con este módulo es la "*MPU6050.h"*.

Otro modulo que se incorporó en esta versión, fue el módulo SSDP (Simple Service Discovery Protocol) que agregaba funciones descubrimiento mediante las librerías *"ESP8266WebServer.h"* y *"ESP8266SSDP.h"*. Actualmente este módulo únicamente identifica al sensor en la red, pero en futuras versiones este módulo permitiría que, mediante esta identificación, el módulo se descargara configuraciones específicas para esa red.

Otro modulo que se incorporó en esta versión, fue un módulo de gestión de tareas. Este módulo permite abstraer las operaciones de la anterior versión de ejecución de tareas programadas. Para ello se incorporó la librería *"AsyncTaskLib.h"* que permite simplificar estas operaciones, proporcionando métodos que automáticamente calculan el tiempo que había transcurrido.

Por último, en esta versión se agregaron funciones que permitían obtener el tiempo a partir del protocolo NTP (Network Time Protocol) utilizando las librerías *"TimeLib.h", "WiFiUdp.h"*. La incorporación de este módulo vino dada por la necesidad de establecer una marca de tiempo para cada medición del sensor de medida inercial.

Esta versión incorporaba todas las funcionalidades sobre las cuales este proyecto estaba interesado. Pero tanta funcionalidad en un sistema operativo de tiempo real generaba problemas ejecución de tareas bloqueantes que se detallaran en la sección de problemas del capítulo 4.

Así pues, la última versión que se desarrolló utilizando la codificación y el entorno de Arduino debido a la complejidad y magnitud que estaba adquiriendo la codificación, fue la implementación de todos módulos anteriormente especificados con la incorporación de un sistema de interrupción. Este sistema permitía que

cuando se detectara una vibración, a pesar de que se estuviera procesando una tarea bloqueante, la ejecución saltara dando prioridad a la captación de la información proporcionada por el sensor.

Antes de finalizar este apartado es necesario comentar una versión alternativa que no se llegó a consolidar por problemas de temporalidad pero que, en el capítulo 6 se comentará. Esta versión utilizaba la memoria interna del microcontrolador NodeMCU para almacenar datos en la memoria flash. Para ello había que realizar una partición de la memoria flash del microcontrolador estableciendo una sección para almacenar el código del programa y otra para almacenar datos. En la siguiente página web se detallan más información de los pasos y las configuraciones a realizadas en el IDE de Arduino para cargar datos en esta partición: [11]

4.2.1.2.1 Resumen de versiones

- Versión 0:
  - Conectividad del ESP8266 mediante el uso de la librería *"ESP8266WiFi.h"*.
  - Parpadeo de led incorporado mediante el uso de la función delay().

- Versión 1:
  - Implementación de sistema de funciones modular aumentando la cohesión y reduciendo el acoplamiento.
  - Implementación del módulo DHT con la medición de la temperatura y humedad a partir del sensor DHT22 usando la librería *"DHT.h"*.
  - Implementación del módulo MQTT realizando operaciones de publicación de datos a partir de la librería *"Adafruit_MQTT.h"*.
  - Implementación del módulo debug de forma que permitía concentrar todas las operaciones de depuración en un módulo. De esta forma se abstraía la operación de depurar pudiendo realizar la operación sobre serial si estaba activado o mediante MQTT si el módulo estaba cargado.

- Versión 2:
  - Implementación del módulo OTA con el uso de las librerías: *"ESP8266mDNS.h", "WiFiUdp.h" y "ArduinoOTA.h"* que permitía la actualización del firmware de forma remota y segura.

- Versión 3:
  - Uso de las funcionalidades de suscripción MQTT permitiendo solicitar desde un cliente MQTT la actualización de la información.
  - Incorporación de la funcionalidad de ejecución de tareas tras un intervalo de tiempo eliminando el uso de delay(). Esto se debe a que la funcion delay() bloquea la ejecución del hilo principal causando problemas que se comentaran posteriormente en la sección de problemas con las operaciones de escucha del módulo OTA o las operaciones de suscripción del módulo MQTT.

- Versión 4:
    - Sustitución del módulo DHT por el módulo de medida inercial para captar la información del sensor MPU6050 mediante el uso de la librería "*MPU6050.h"*, además de la incorporación de métodos auxiliares para la calibración del módulo.
    - Implementación del módulo de descubrimiento SSDP a partir de las librerías *"ESP8266WebServer.h", "ESP8266SSDP.h"*.
    - Implementación del módulo gestor de tareas sustituyendo las operaciones de cálculo de tiempo transcurrido por funciones de la librería *"AsyncTaskLib.h"*.
    - Implementación del módulo NTP que permitía la obtención de una marca de tiempo para agregar a la información captada por el sensor IMU.

- Versión 4.2:
    - Incorporación de interrupción para la detección de movimiento en el módulo IMU.

4.2.1.3 Versiones en C++

Debido a la complejidad y magnitud que estaba adquiriendo el código se pensó en buscar una alternativa a la codificación proporcionada por Arduino. Arduino proporciona un lenguaje de programación sencillo basado en C++, pero no es el único lenguaje de programación con el que se puede programar el microcontrolador ESP8266.

El ESP8266 se puede programar con Arduino, Lua, MicroPython o C++. Entre estos lenguajes de programación aparte de Arduino el único con el que se poseía experiencia era con C++. Por este motivo y por su capacidad para implementar el paradigma orientado a objetos con su codificación, era el lenguaje idóneo para este proyecto.

Como se comentó en el apartado 4.2.2.1, para el desarrollo en C ++ se cambió el entorno de desarrollo a Atom con el plugin de Platformio que proporcionaba todas las herramientas para realizar esta codificación.

El propósito de este cambio era implementar una codificación que utilizara el paradigma orientado a objetos estableciendo unas clases modulo que agruparían todos los atributos y métodos necesarios para ejecutar las funcionalidades de ese modulo.

Las primeras versiones siguiendo esta implementación pretendían utilizar de forma similar los módulos a como se había utilizado en la codificación estructurada de Arduino, aunque tempranamente comenzaron a suceder problemas de implementación. El conjunto de estos problemas se detallará en una próxima sección, agrupándolos y comentando las soluciones aportadas.

4.2.1.4 Versión final

Tras la resolución de los problemas encontrados, en la versión final del sistema operativo para el sensor, se incorporó el uso de algunos patrones de diseño.

En primer lugar, debido a su diseño modular, se introdujo el patrón de creación "Abstrac Factory", patrón destinado a proporcionar un interfaz para crear familias de objetos con una relación sin necesidad de especificar la clase completa. De esta forma se estableció la clase "Factoría de Módulos" que permitía crear

objetos Modulo. Para poder crear los diferentes módulos se especificó la clase padre Modulo y las clases hijas con la definición de los diferentes módulos.

Debido a que la compilación de C++ para el ESP8266 conlleva algunas limitaciones, no se pudo definir en C++ la clase Modulo como clase abstracta o como se denomina en C++, clase con métodos puros virtuales.

El segundo patrón que se implementó también corresponde al ámbito de patrones de creación, se trata del patrón "Singleton". Este patrón garantiza que una clase solo tenga una instancia, proporcionando un único punto de acceso a ella. Gracias a la aplicación de este patrón sobre los módulos y la clase sensor, se solucionó el problema de las dependencias cíclicas típico de C++ que se especificara en la sección 4.2.5, ya que los módulos disponían de un acceso estático.

Una vez comentados los patrones utilizados en la implementación, se va a proceder a comentar la funcionalidad de cada una de las clases implementadas con sus atributos y métodos más destacados. El diagrama de clases UML completo se puede encontrar en Capitulo 8 Anexos.

4.2.1.4.1 Clase Factoría Módulos

La clase FactoiraModulos permite la creación de las instancias de los diferentes módulos del sistema. Su implementación al igual que el conjunto de los módulos, contiene el uso del patrón Singleton, por lo que contiene la referencia a una instancia estática de un objeto FactoriaModulos con acceso privado junto con el constructor y destructor con el mismo tipo de acceso. De esta forma su único punto de acceso es a través del método *getInstancia()* que devuelve un puntero a la única instancia estática del objeto FactoriaModulos.

Por otro lado esta clase contiene el método *crearModulo()* que pasándole el valor de un entero permite la creación de un tipo determinado de modulo devolviendo un puntero a la única instancia del módulo, ya que como comentamos, el conjunto de módulos implementan el patrón Singleton. A través del número entero que se le pase como valor, se puede implementar un módulo u otro. A continuación, se expone el número y la correspondencia con el tipo de modulo:

1. ModuloDebug
2. ModuloWifi
3. ModuloMQTT
4. ModuloNTP
5. ModuloSD
6. ModuloIMU
7. ModuloGestorTareas
8. ModuloOTA
9. ModuloSSDP

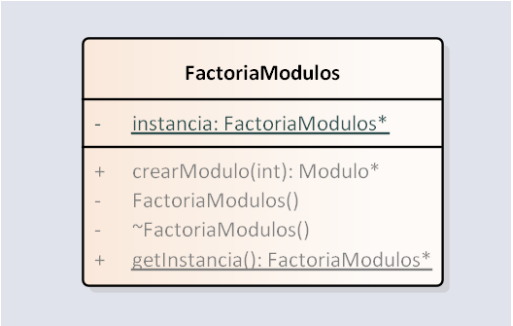


Ilustración 18 Diagrama UML Clase FactoriaModulos

4.2.1.4.2 Clase Modulo

La clase Modulo permite la abstracción del concepto de módulo, permitiendo relacionar los diferentes módulos para que desde la clase Factoría de Módulos puedan ser instanciados.

Como se ha comentado anteriormente, siguiendo el patrón de diseño *Abstrac Factory*, esta clase debería de ser una clase abstracta, pero debido a las limitaciones del compilador no se puede establecer en C++ como clase virtual pura ya que el microcontrolador pose un espacio reducido de memoria y no permite el uso de tablas virtuales.

La clase modulo únicamente contiene una variable de tipo String con seguridad de tipo protected para que pueda ser accedida desde las clases hijas. En ella se almacena el nombre del módulo para que a la hora de realizar operaciones de depuración se pueda identificar el módulo. Esta variable se inicializa desde el constructor de la clase

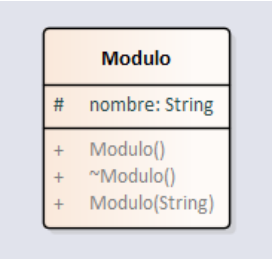


Ilustración 19 Diagrama UML Clase Modulo

4.2.1.4.3 Clase Modulo Debug

El propósito de esta clase es reunir las operaciones de depuración en un único punto de acceso. La clase ModuloDebug contiene un atributo privado estático donde almacena la única instancia del módulo y dos variables de tipo booleano que permiten determinar que método de depuración se desea utilizar, serial o MQTT.

ModuloDebug contiene un constructor privado al que se le pasa como parámetro los baudios a los que van a configurar el puerto serie, un booleano para determinar si se quiere habilitar la depuración serial o no, y el nombre del módulo destinado a identificarse en las operaciones de depuraciones.

Esta clase contiene como métodos públicos, esta clase contiene el método *getInstancia()* para obtener la única instancia del módulo, el método *ejecutarModuloDebug()* al que se le pasa como parámetro el origen del mensaje y el mensaje que se quiera depurar; y por ultimo contiene el método *activarDebugMQTT()* que permite una vez instanciado un ModuloMQTT habilitar su uso en operaciones depuración. De esta forma al llamar a *ejecutarModuloDebug()* si se ha activado la instancia del módulo MQTT también se puede depurar el sistema por este método.

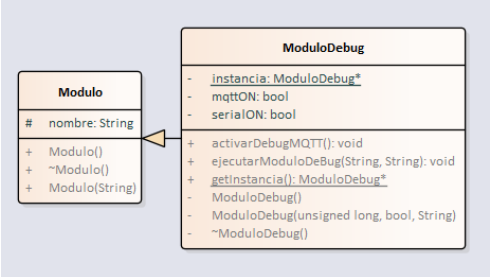


Ilustración 20 Diagrama UML Clase ModuloDebug

4.2.1.4.4 Clase Modulo Wifi

ModuloWifi es la clase encargada de realizar las operaciones necesarias para establecer una comunicación con el punto de acceso. Además de heredar de la clase Modulo y seguir el patrón mencionado en los anteriores módulos mediante el uso del patrón Singleton, esta clase contiene como atributos privados el SSID (Service Set Identifier) y la contraseña del punto de acceso. Además, se especifican en la variable *intentos* el número de reintentos que se desean establecer en caso de que no pueda asociarse con el punto de acceso.

Si analizamos los métodos, ModuloWifi contiene el método *isConected()* que permite conocer si el sistema está conectado con el punto de acceso o se ha perdido la conexión y el método *conectar()* que permite realizar las operaciones para conectarse con el punto de acceso.

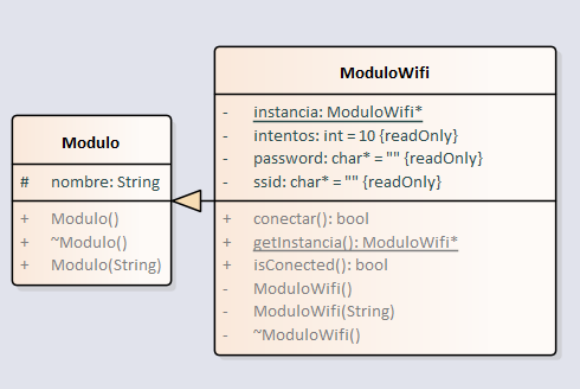


Ilustración 21 Diagrama UML Clase ModuloWifi

4.2.1.4.5 Clase Modulo MQTT

La clase ModuloMQTT es la encargada de gestionar las operaciones relacionadas con el protocolo MQTT, protocolo especificado en el capítulo 3 sección 3.2.4. En cuanto a su composición, en esta clase destacan los atributos: *server* donde se define la IP del bróker MQTT, los atributos *username* y *password* destinados a la identificación con el bróker y la variable *intentos*, que contiene el número de intentos para establecer la conexión en caso de fallo al igual que en la clase ModuloWifi.

Además incorpora punteros a objetos Adafruit_MQTT_Client , Adafruit_MQTT_Publish proporcionados por la librería *"Adafruit_MQTT.h"* encargados de almacenar el cliente y la ruta de publicación de MQTT.

Respecto a los métodos, ModuloMQTT contiene un método privado *enviarDato()* que permite abstraer la operación de enviar un mensaje por MQTT respecto del medio de publicación. Para ello se le pasa como parámetro este medio de publicación. De esta forma se pueden crear métodos auxiliares públicos para que sean utilizados desde objetos externos sin necesidad de entrar en el concepto de medio de publicación.

Por ejemplo, para enviar un mensaje de depuración mediante MQTT, únicamente hay que llamar al método *enviarLog()* pasando como parámetro el mensaje que se quiera enviar. Este método llama a *enviarDato()* pasándole como parámetro el mensaje recibido y el medio de publicación. Otro ejemplo es el caso de *enviarDatosAcelGyron()* que nos permite enviar los mensajes generados por el ModuloIMU únicamente pasándoselos como parámetro.

Además de estos métodos de envió de mensajes, la clase ModuloMQTT contiene los métodos *conectarMQTT()* que permite realizar la secuencia de conexión con el broker, *isConected()* que nos devuelve con un booleano si el sensor está conectado con el broker, y el método ping que nos permite enviar mensajes de ping para actualizar el estado de la conexión con el broker.

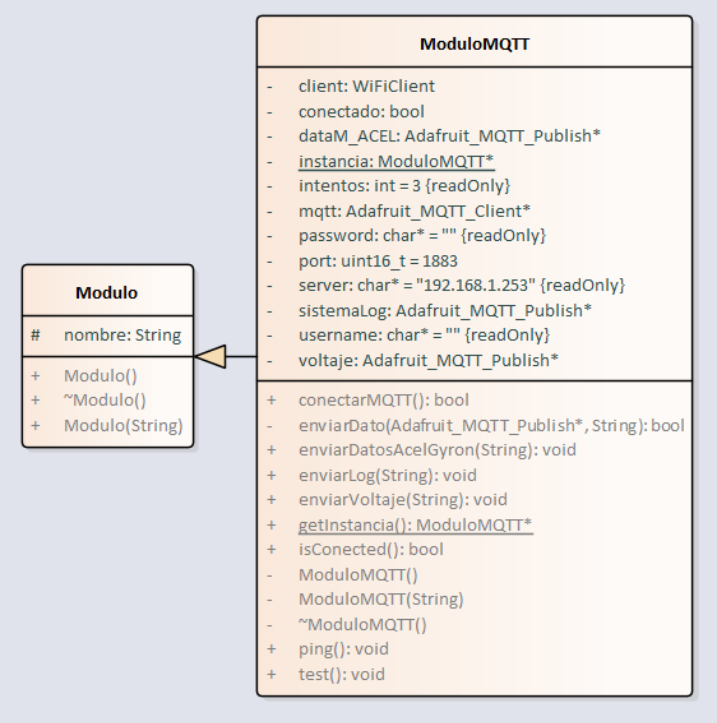


Ilustración 22 Diagrama UML ModuloMQTT

Resumen de vías de publicación:

- *habitacion/2/60:01:94:74:48:30/M_ACEL*
- *habitacion/2/60:01:94:74:48:30/log*

4.2.1.4.6 Clase Modulo SD

El propósito de esta clase es solucionar la problemática que surgía cuando se producía una desconexión del punto de acceso. Como se comentó en la sección 4.1.1, este proyecto trabajaría en condiciones de alta predisposición a la aparición de interferencias y problemas de conectividad. Cuando la maquinaria estuviera activa, está generaría interferencias, lo que provocaría que los datos capturados por el ModuloIMU, no se pudieran enviar al interfaz, generando la perdida de esa información.

Para solucionar este problema, se incorporó a nivel hardware un lector de tarjetas micro SD compatible con Arduino. De esta forma si se generaba un dato y no se podía enviar por falta de conectividad, este dato se almacenaba en la memoria SD hasta que pudiera ser enviado.

Destacando algunos atributos de la clase ModuloSD, podemos encontrar dos variables de tipo String, *nombreArchivo* y *nombreArchivo2* donde se almacena el nombre del archivo de datos principal y el archivo de datos auxiliar. Además, se puede encontrar un atributo donde se almacena la referencia al fichero denominada *archivo,* junto con el atributo *csPin* que indica el número de pin del microcontrolador NodeMCU al que está conectado el puerto CS del lector de tarjetas, y por último el atributo *caracterDeFrase*, que permite separar las cadenas de datos almacenadas en el archivo.

Respecto a los métodos, empezando por el constructor de tipo privado; contiene una secuencia de inicialización que primeramente establecerá la comunicación mediante el bus SPI con el lector de tarjetas micro SD, a continuación, ejecuta una llamada a *ejecutarModuloDebug()* mostrando el formato y el tamaño de la tarjeta micro SD detectada, a continuación realiza una nueva llamada a la función de depuración

mostrando los archivos en la tarjeta y por último abre en modo lectura el archivo definido con el nombre referente al archivo de datos principal. En caso de que no exista lo crea.

Continuando con los métodos, el siguiente método destacado, es el método *escribir()* que permite la escritura en el archivo de datos pero sin realizar la serialización del archivo para que sea un método que se ejecute con rapidez. A continuación encontramos el método *serializarArchivo()* que graba la información cargada en memoria sobre el archivo de la tarjeta micro SD. Esta operación de serialización es una operación lenta y como detallaremos posteriormente, esta tarea se ejecuta de forma periódica por el moduloGestorTareas cuando el archivo de datos no está vacío.

Por ultimo en el ámbito de métodos destacados, se encuentra el método *leerArchivo()*. Se trata de un método que requiere de tiempo para completar su ejecución. Por este motivo y por la falta de librerías de sincronización en esta versión, se tuvo que establecer un proceso para proteger la lectura de datos frente a escritura arbitraria generada por una interrupción.

Para ello el método se ayuda de un archivo auxiliar y atributos de control especificados en la clase como el atributo booleano *archivoControl,* que permite determinar si el archivo está preparado para su escritura o no.

De esta forma el primer paso que realiza el método *leerArchivo()* consiste en comprobar si el archivo contiene información. Si el archivo contiene información, se bloquea la posibilidad de realizar escrituras. A continuación, se cambia la referencia de archivo al nombre del archivo auxiliar, para que en caso de que se produzca una interrupción en este proceso de lectura de datos, se puedan almacenar los nuevos datos. Una vez realizada esta operación se reactiva el atributo *archivoControl.*

Una vez realizadas estas operaciones de protección, se lee y envía los datos mediante el moduloMQTT al bróker. Al terminar esta lectura y envió, se bloquea de nuevo *archivoControl*, se cambia la referencia al archivo principal, se reactiva *archivoControl* y por último se realiza la operación de lectura y envió del archivo auxiliar para almacenar los posibles datos generados durante el proceso de lectura del archivo principal.

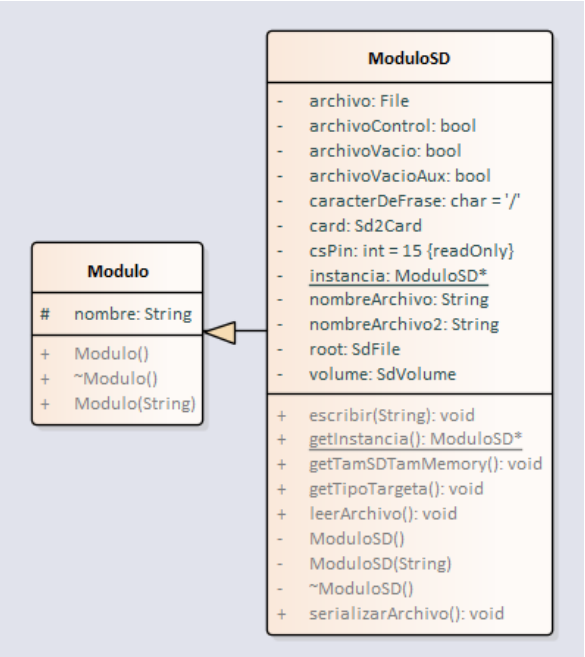


Ilustración 23 Diagrama UML ModuloSD

4.2.1.4.7 Clase Modulo IMU

La clase moduloIMU se define como la clase con mayor complejidad de implementación. En ella se encuentran los atributos y métodos necesarios para establecer la comunicación con el sensor de medida inercial MPU6050 y obtener los datos de aceleración y giro. La clase ModuloIMU se apoya en el desarrollo de la librería *"MPU6050.h"* implementada por Jeff Rowberg y la comunidad web a través de su repositorio [12]. Además, se ha podido implementar gracias a numerosos documentos informativos como el tutorial que se proporciona en esta referencia [13].

Profundizando en la implementación, la clase ModuloIMU contiene dos secciones diferenciadas; una sección encargada de proporcionar métodos y atributos para realizar una calibración, y otra sección encargada de proporcionar métodos y atributos para realizar la lectura de datos. A pesar de ello existen atributos comunes a las dos secciones como el atributo de tipo byte *interrpPin* donde se define el pin del microcontrolador donde el sensor envía la detección de interrupciones, o el atributo estático público de tipo MPU6050 donde se almacena una instancia de la librería que referencia al sensor. Esta variable es de tipo estático debido a que necesita poder ser accesible desde el método estático que ejecuta la interrupción.

Sección lectura de datos:

Si analizamos los atributos de la sección de lectura de datos, encontramos el atributo *currentMillis* donde se almacenan los milisegundos tras la última ejecución de una interrupción e *intervalInterupt* donde se almacena el intervalo de tiempo que se desea que transcurra entre interrupciones. La implementación de estos atributos surgió como respuesta a la problemática que aparecía cuando se detectaban muchas interrupciones en un periodo corto de tiempo, no dejando tiempo a que se ejecutaran. Este problema se detallará en la sección 4.2.5.

Otros atributos que podemos encontrar son los encargados de definir el perdido sobre el cual debe de existir movimiento para detectar la interrupción y la fuerza de la interrupción. Para los ambos atributos se han establecido como 1 siguiendo la referencia de la librería para obtener la máxima sensibilidad.

Una vez detallados los atributos dedicados a la captura, comentamos los dos métodos definidos para esta sección. El primero de ellos se encarga de establecer la configuración para la detección de la interrupción. Este se denomina *configInterrupIMU(),* el otro método que se puede encontrar es *interrupcionIMU()* que se encarga de establecer el método que ejecutará la interrupción. Este método tiene un carácter estático ya que cuando se produce la interrupción, el sistema detiene la ejecución del programa y ejecuta una sección diferente que debe de ser de tipo estático.

Sección calibración del módulo IMU:

Por otro lado, si comentamos la sección del código encargada de la calibración del módulo, podemos encontrar los métodos *calibrar_MPU6050()*, *calibrarAux()* y *meansensors()*. La ejecución de estos métodos retorna un offset que permite calibrar los ejes del sensor.

Profundizando en estos métodos, el primero que encontramos es *calibrar_MPU6050(),* este método establece a 0 el offset de cada uno de los ejes y procede a llamar al método *meansensors()* para inicializar los atributos con la media de los datos que se generan en ese periodo. El método *meansensors()* realiza un proceso de media de los datos recogidos por cada uno de los ejes, para ello los almacena en un array y establece la media de estos datos.

Tras la ejecución de este método a modo de inicialización, se procede a la llamada del método *calibrarAux()*; este método es el encargado de buscar un offset para los 6 ejes, que permita obtener valores establecidos dentro del umbral de error definido en el atributo *acel_deadzone* . Hay que destacar, que durante este periodo de búsqueda de offsets que satisfagan el umbral de error, se establece encendida la luz del LED del microcontrolador encendida para dar a conocer que se está llevando a cabo el proceso de calibración y que no se debe de mover el sensor. Una vez se consiguen los offsets se establecen para el módulo.

Esta operación de calibración únicamente requiere que sea ejecutada cada vez que se conecte un sensor nuevo, o se cambien las condiciones del entorno sobre el que el sensor va a operar.

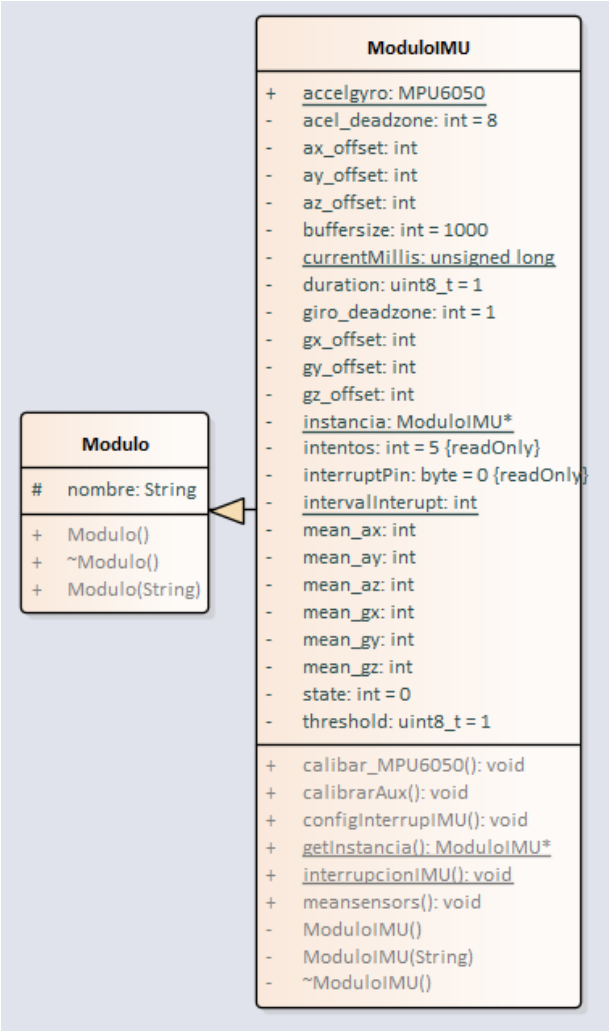


Ilustración 24 Diagrama UML Clase ModuloIMU

4.2.1.4.8 Clase Modulo Gestor de Tareas

La clase ModuloGestoTareas, es la clase encargada de definir los métodos y atributos necesarios para la ejecución de tareas del sistema en el sensor; para ello, esta clase se apoya en la librería *"AsyncTaskLib.h"* definiendo punteros a objetos de tipo AsyncTask. Cada uno de los atributos definidos como objeto AsyncTask, se define como una tarea del sistema. En el constructor de la clase, se define cada una de las tareas con su intervalo de ejecución.

Entre las tareas se encuentran:

- **Tarea de parpadeo LED**: Esta tarea es la encargada de hacer parpadear el LED interno de la placa NodeMCU, de esta forma permite conocer que el microprocesador está ejecutándose sin problemas. Ya que esta tarea necesita ejecutarse con determinada temporalidad, y puesto que se trata de un sistema en tiempo real sin gestión de procesos, cuando se provoque una tarea bloqueante, el LED se apagará hasta que deje de ejecutarse esa tarea. De esta forma se puede determinar, que si el LED se mantiene encendido por un periodo mayor que el establecido como intervalo, significa que se ha producido un error de ejecución.

- **Tarea de revisión de conectividad**: Esta tarea es la encargada de revisar la conectividad con el punto de acceso y con el bróker. Si se produce algún fallo tras la revisión se realiza una reconexión.

- **Tarea de envió de datos en SD:** Esta tarea comprueba si el archivo de la tarjeta SD contiene datos. En caso de que contenga datos y exista conexión con el bróker, estos son enviados.

- **Tarea de serialización de datos:** Esta tarea se encarga que los datos que están en memoria sean serializados en la tarjeta SD. Como comentamos en la definición de la clase ModuloSD, la tarea de serialización es una tarea lenta frente a una posible interrupción, donde el ModuloIMU genera un dato y se necesita que el ModuloSD lo guarde en la tarjeta micro SD. Por este motivo, el dato se almacena en memoria durante el tiempo que permanece la interrupción; pero es esta tarea, la encargada de serializar los datos, para que en caso de fallo o apagado, no se pierda información.

- **Tarea de ejecución de servidores:** Esta tarea, es la encargada de ejecutar lo métodos de espera de los servidores que contiene el ModuloSSDP y el ModuloOTA.

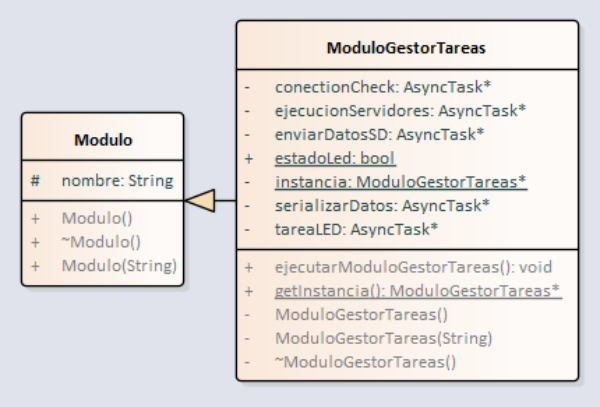


Ilustración 25 Diagrama UML Clase ModuloGestorTareas

4.2.1.4.9 Clase Modulo OTA

La clase ModuloOTA es la encargada de gestionar la funcionalidad de recibir actualizaciones a través de la red, sin necesidad de conectar el sensor mediante USB serial.

En la clase ModuloOTA destaca su atributo *MD5HASHPASS,* que permite almacenar el hash en md5 correspondiente a la contraseña. Además de este atributo, la clase contiene el atributo *OTAPORT* donde se especifica el puerto de escucha para la recepción de peticiones de actualización.

Como método destacado únicamente encontramos el método *ejecutarModuloOTA()* que contiene la función de espera de peticiones del servidor OTA.

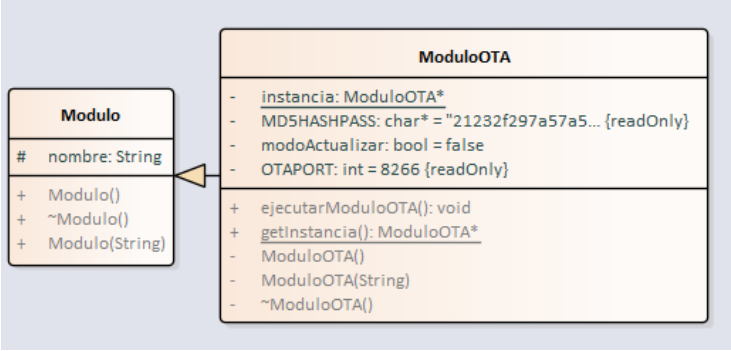


Ilustración 26 Diagrama UML Clase ModeuloOTA

4.2.1.4.10 Clase Modulo SSDP

La clase ModuloSSDP contiene el atributo a un objeto de tipo ESP8266WebServer donde se referenciará la instancia del servidor web de tipo UDP, para la escucha de peticiones de identificación.

Como método destacado encontramos el método *ejecutarModuloSSDP()* que contiene la espera del servidor hasta la llegada de posibles clientes requiriendo la información del dispositivo.

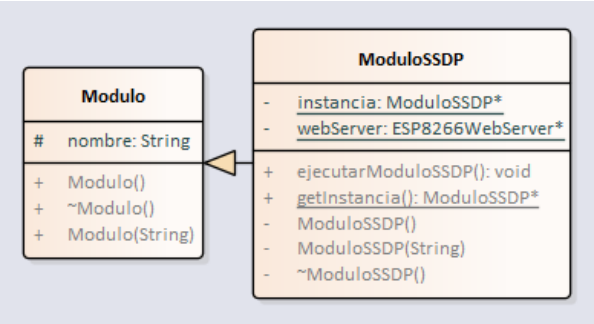


Ilustración 27 Diagrama UML Clase ModuloSSDP

4.2.1.4.11 Clase Sensor

La clase Sensor es el punto de acceso a los módulos del sistema. Como solución al problema de las dependencias cíclicas propio del lenguaje de C++, junto con el problema de la definición de módulos estáticos necesario para ejecutarse dentro de la interrupción, se tomó la decisión de implementar todos los módulos con una referencia estática accesible desde la clase Sensor, proporcionando de esta manera cierta protección.

La clase Sensor implementa dos métodos a destacar. El método *iniciarModulos(),* encargado de iniciar los módulos del sistema llamando a la clase FactoriaModulos y el método *procesoLecturaSDYEnviarDatos()*, encargado de detectar y enviar los datos en la tarjeta micro SD procedentes de la anterior ejecución.

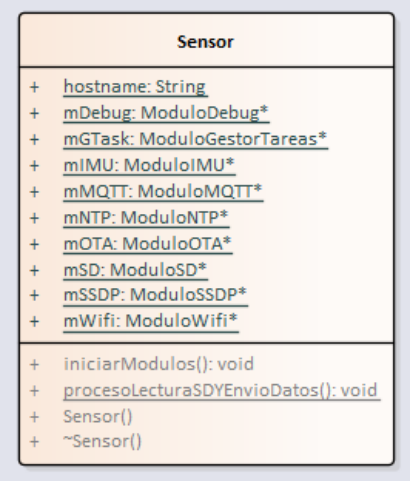


Ilustración 28 Diagrama UML Clase Sensor

4.2.1.4.12 Archivo Main

Para finalizar con este apartado de *Versión Final*, hay que destacar que aunque de poca complejidad en su implementación, se sigue manteniendo los métodos característicos de las implementaciones Arduino *setup()* y *loop()*, ya que Arduino nos proporciona los métodos principales para implementar un sistema en tiempo real con un bucle secuencial. A pesar de esa secuencialidad, como se ha ido especificando a lo largo de este desarrollo del sistema operativo, se ha podido simular cierto multiprocesamiento con la ejecución de tareas programadas y el uso de interrupciones.

De este modo, el archivo Main únicamente contiene cuatro características a destacar:

- Contiene un puntero a Sensor
- En la función *setup()*, la llamada al método *iniciarModulos()* a través de una instancia previamente definida de la clase Sensor.
- En la función *loop()*, el acceso al ModuloGestorTareas a través de la clase Sensor con su método *ejecutarModuloGestoTareas()*.

4.2.2 Implementación del interfaz entre el sensor y el servidor

Como se ha comentado anteriormente, para que el sensor se comunique con el servidor, se establece un intermediario a modo de interfaz. De esta forma se puede desarrollar de forma independiente las comunicaciones con el servidor, de las comunicaciones de la red de sensorización. Incluso se podría mover dicho servidor fuera de la red local a un sistema en la nube sin que la red de sensores se viera modificada.

Para implementar este interfaz se ha utilizado una Raspberry Pi, que como se comentó en el apartado 3.2.1, se trata de una placa controladora con características de mini ordenador, con un consumo eléctrico máximo de 12.5 W. Por este motivo, esta placa controladora es ideal para albergar el bróker y el interfaz de comunicación con el servidor, definido con la herramienta Node-Red.

El bróker instalado para este sistema es Mosquitto en su compilación 1.4.10. Este es el encargado de transmitir los mensajes entre la red de sensores y Node-Red mediante el protocolo MQTT.

Node-Red es una herramienta de programación basada en un sistema de módulos de tipo Scrach que combina la simplicidad de la programación con módulos, con la posibilidad de implementar codificaciones en JavaScript. Su instalación y configuración se expone en capítulo 8 de Anexos.

Node-Red proporciona un interfaz para acceder desde la red, por lo que a pesar de estar instalado en la Raspberry pi, se puede acceder a través de: http://192.168.1.253:1880/

Una vez se accede mediante los datos de usuario y contraseña previamente configurados en el proceso de instalación, Node-Red proporciona un espacio central donde insertar los módulos; una barra lateral a la izquierda donde seleccionar los módulos que se desean añadir y una barra lateral derecha donde se pueden ver las propiedades del módulo y opciones de depuración.

Sección suscripción a depuración:

Profundizando en la implementación, hay que destacar una primera sección destinada a la depuración mediante MQTT. Según se especificó en la sección 4.2.1, en lo relativo a la implementación de la clase ModuloDebug, ésta además permitía la depuración mediante MQTT; para ello se establece un canal de publicación en el sensor y suscripción en Node-Red. Para ello se agregó un nodo MQTT que se suscribía al tópico *"habitacion/2/60:01:94:74:48:30/log"* del bróker. Para visualizar esta información se agregó un nodo de depuración.

Ilustración 29 Diagrama Node-Red Suscripción depuración

Profundizando en la definición de tópico que establece la ruta suscripción, está esta compuesta por cuatro partes:

- Habitación o sala, que define el tipo de localización del sensor.
- Número que indica la localización dentro del tipo de localización.
- Dirección MAC del microcontrolador.
- Tipo de información.

Sección suscripción a recolección de datos:

Una vez comentada la primera sección, se procede a exponer la sección encargada de lectura de los datos del bróker y envió al servidor para su procesamiento.

Al igual que la sección anterior, esta sección comienza con la definición de un nodo subscriptor al bróker denominado LecturaDatosSensor que se suscribe al tópico que genera el sensor al capturar un dato *"habitacion/2/60:01:94:74:48:30/M_ACEL"*.

Este dato se recibe siguiendo el formato que envía el ModuloIMU:

(Marca de tiempo; Aceleración en eje X; Aceleración en eje Y; Aceleración en eje Z; Aceleración angular en eje X; Aceleración angular en eje Y; Aceleración angular en eje Z)

Al recibir el dato, éste se separa detectando el token ";" y se transforma en un vector. Sobre este vector se realiza un tratamiento para crear un documento en formato JSON que se enviara a la base de datos del servidor. Se estima que el coste de almacenamiento de cada documento JSON supone una media de 200 bytes.

Un ejemplo de este tratamiento consiste en la transformación del dato de milisegundos en formato UNIXTime al formato ISO 8601 UTC.

Una vez se genera el documento JSON, éste se envía a un nodo de http request que se encarga de enviar mediante una petición POST el documento al servidor de Elasticsearch.

La configuración de este nodo son:

- Metodo: POST
- URL: http://192.168.1.250:9200/tfg/mantenimiento
- Usuario: elastic
- Contraseña:123456
- Return: a parse json object

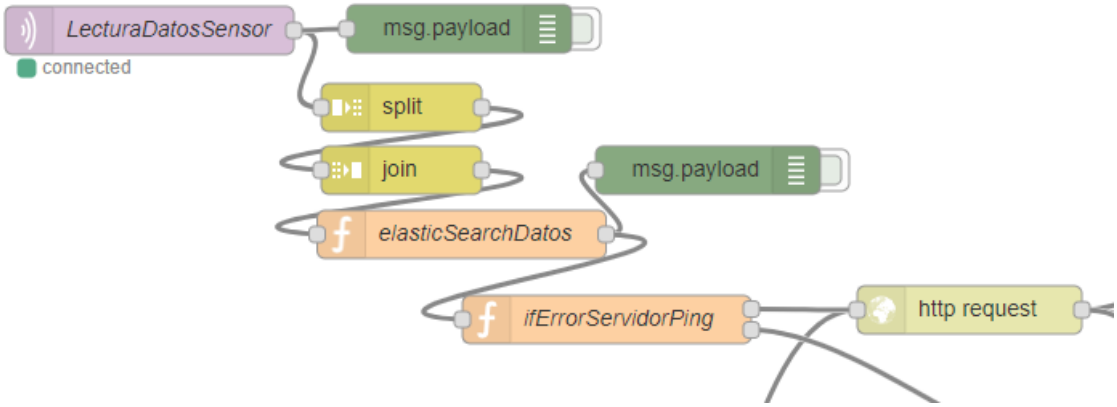


Ilustración 30 Diagrama Node-Red Flujo de suscripción datos ModuloIMU

De esta forma el dato será recibido por el servidor. Para controlar la posibilidad de fallo, en caso de no poder enviar el dato con éxito, en Node-Red se ha implementado una codificación de recuperación. Para ello previamente se realiza un sistema de máquina de estados mediante la variable *modoErrorServidor* de tipo booleano. Esta variable se inicializa como falso al arrancar el servidor en el flujo representado en la ilustración 31.

Por otro lado, para evitar la espera generada por el nodo "http request" en el intento de envió al servidor cuando se conoce la falta de conexión con el servidor, se implementa un flujo que realiza un ping cada 1s controlando de esta forma la accesibilidad del servidor. En caso de que el servidor no responda a la petición de ping, los datos se almacenan en un archivo.

Tanto si no se consigue hacer ping o se produce un error de conexión generado por el nodo "http request", se almacenan los datos en un archivo.

Ilustración 31 Node-Red Flujo de inicialización de variables

Ilustración 32 Node-Red Control "ping" estado conexión servidor

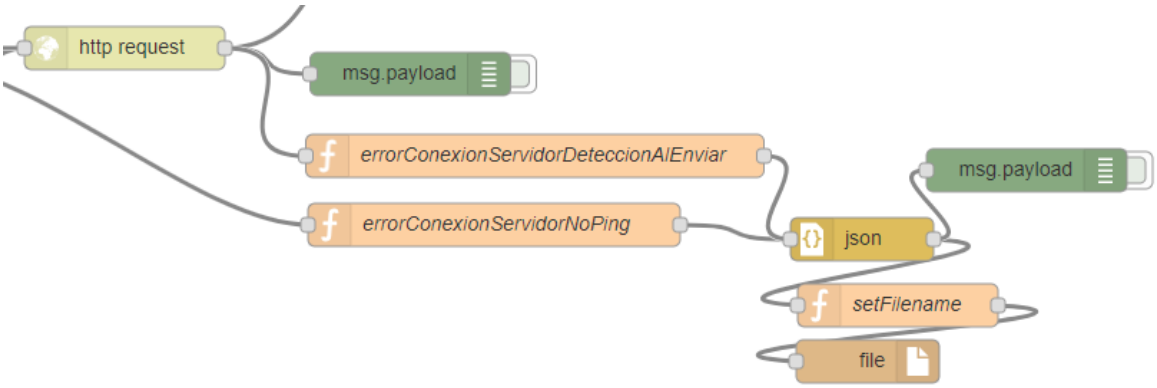


Ilustración 33 Diagrama Node-Red Flujo almacenamiento de datos tras el fallo en el envió al servidor

El nodo errorConexionServidorDeteccionAlEnviar contempla los siguientes fallos:

- *Error: socket hang up:* http://192.168.1.250:9200/tfg/mantenimiento
- *Error: connect EHOSTUNREACH 192.168.1.250:9200:* http://192.168.1.250:9200/tfg/mantenimiento

Si detecta alguno de estos errores establece la variable de flujo *modoErrorServidor* en verdadero. Induciendo al sistema que si consigue enviar algún mensaje ejecute la sección de recuperación definida en los nodos:

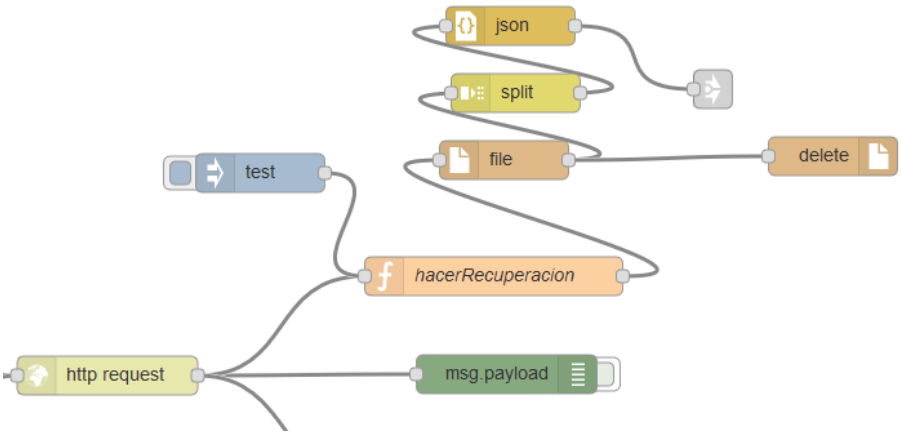


Ilustración 34 Diagrama Node-Red Flujo de recuperación de datos tras el fallo en el envió al servidor

En en el flujo representado en la ilustración 34, el nodo hacerRecuperacion detecta el mensaje creado por el nodo http request. Si se produce un envió exitoso y el sistema se encuentra en el estado de error determinado por la variable *modoErrorServidor*, éste comienza el modo de recuperación. Para ello primero lee el archivo que se ha creado en el proceso anterior, separa los objetos JSON actualmente serializados en

String por el token "\n". Una vez separados, convierte estas cadenas con formato, en objetos que se envían a un nodo de link (nodo gris) que permite enviar la información a otra sesión del flujo.

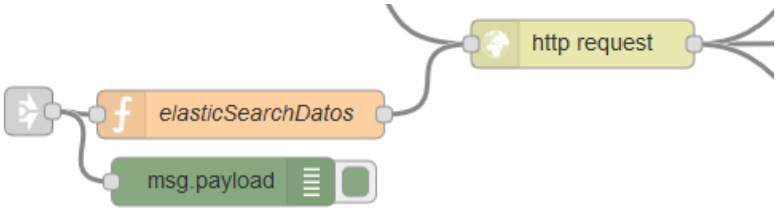


Ilustración 35 Diagrama Node-Red Flujo Envió de datos de recuperación

4.2.3 Implementación de la base de datos

Para implementar la parte encargada del almacenamiento y procesamiento, se ha configurado una máquina virtual con el sistema operativo Ubuntu, que contiene una instalación de la base de datos en tiempo real Elasticsearch, que a su vez incorpora una herramienta procedente del plugin X-pack de procesamiento de datos con técnicas de Machine Learning. Esta herramienta nos permitirá realizar el mantenimiento predictivo. Su instalación y configuración se puede ver en detalle en el capítulo 8.

Recordando el análisis de bases de datos expuesto en la sección 3.26, Elasticsearch trabaja con documentos en formato JSON y proporciona una API REST FULL para interactuar con ella. Actualmente el medio por el cual se puede acceder a la información que contiene, es a través del portal Kibana que se accede a través de la URL: http://192.168.1.250:5601/login?next=%2F#?_g=() mediante los datos de usuario y contraseña previamente configurados.

El portal Kibana proporciona una pestaña denominada Discover donde se pueden observar y filtrar los datos almacenados, pudiendo seleccionar los campos que se desean visualizar o el rango de fechas. Además, Kibana permite establecer un periodo de actualización de los datos mostrados, permitiendo que se muestre según se van generando.

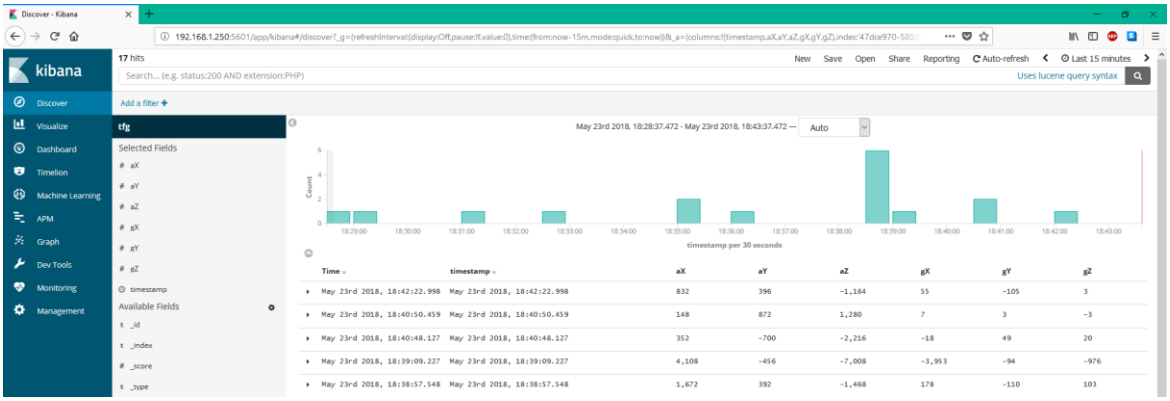


Ilustración 36 Pestaña Discover Kibana

Time	timestamp	aX	aY	aZ	gX	gY	gZ
May 23rd 2018, 18:42:22.998	May 23rd 2018, 18:42:22.998	832	396	-1,164	55	-105	3
May 23rd 2018, 18:40:50.459	May 23rd 2018, 18:40:50.459	148	872	1,280	7	3	-3
May 23rd 2018, 18:40:48.127	May 23rd 2018, 18:40:48.127	352	-700	-2,216	-18	49	20
May 23rd 2018, 18:39:09.227	May 23rd 2018, 18:39:09.227	4,108	-456	-7,008	-3,953	-94	-976
May 23rd 2018, 18:38:57.548	May 23rd 2018, 18:38:57.548	1,672	392	-1,468	178	-110	103
May 23rd 2018, 18:38:53.292	May 23rd 2018, 18:38:53.292	136	-716	-588	217	-246	28

Ilustración 37 Datos en la base de datos Elasticsearch

4.2.4 Ejemplo del proceso del mantenimiento predictivo

Para la realización de las pruebas de mantenimiento predictivo, se ha construido un entorno de pruebas a partir de un ventilador. El propósito del entorno creado es simular el contexto de problema expuesto en el apartado 4.1.1.

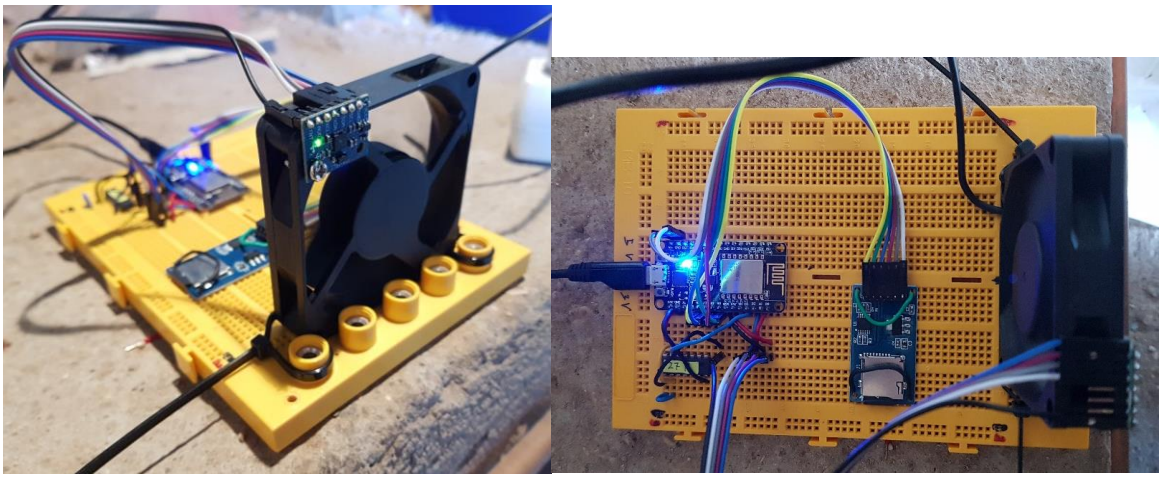


Ilustración 38 Entorno de medición junto con prototipo

Una vez montado el entorno de pruebas, se expuso al sistema a recolectar datos para realizar el mantenimiento predictivo.

A partir del panel de control de Elasticsearch, Kibana; se configuraron gráficas que mostraban información de los datos recopilados por el sistema en tiempo real.

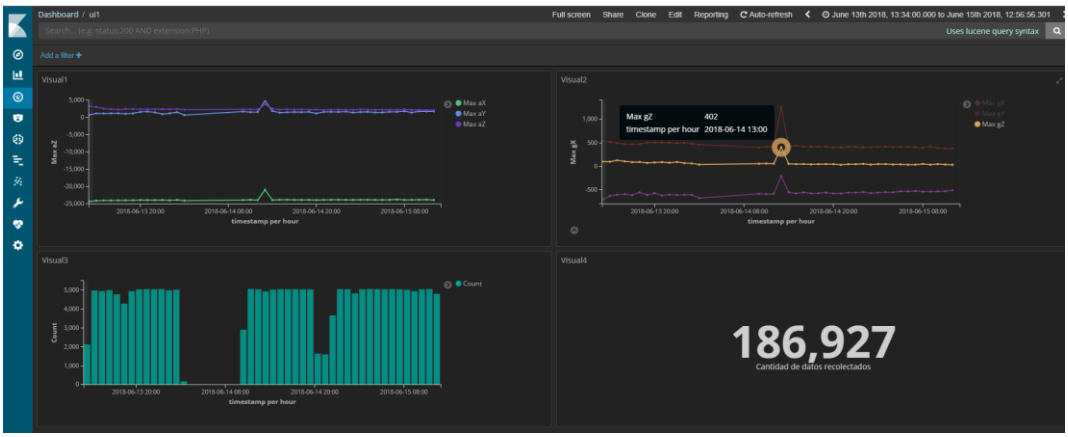


Ilustración 39 Entorno de monitorización de la maquinaria

Realizando un análisis de la información que muestran las gráficas, se pueden observar que se trata de un sistema constante con algunas alteraciones. Esta medición constante se debe a la escala que se muestran los datos. Come veremos posteriormente los datos se toman en un intervalo mínimo de 700 milisegundos aumentando la variedad de datos al aumentar la profundidad de detalle.

En el análisis realizado, se pueden observar un periodo de inactividad, y un periodo de fallo inducido, realizado a efectos de comprobar la detección de fallos por el sistema. El periodo de inactividad establecido en el rango desde las 2:00 hasta las 10:00 horas, es debido a que el ventilador se encontraba apagado Por otro lado, a las 13:43, se puede observar algunas mediciones alteradas fuera del rango de constancia provocadas por una alteración intencionada en el ventilador.

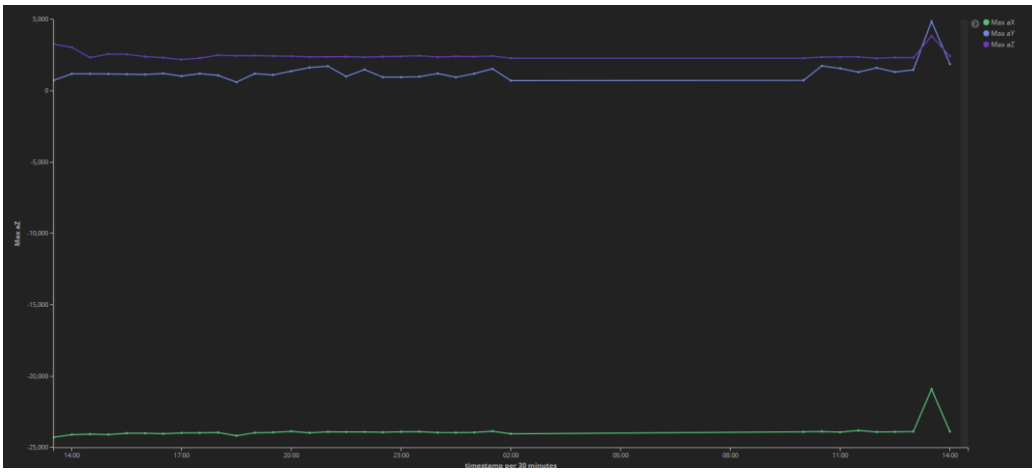


Ilustración 40 Datos recopilados por el acelerómetro

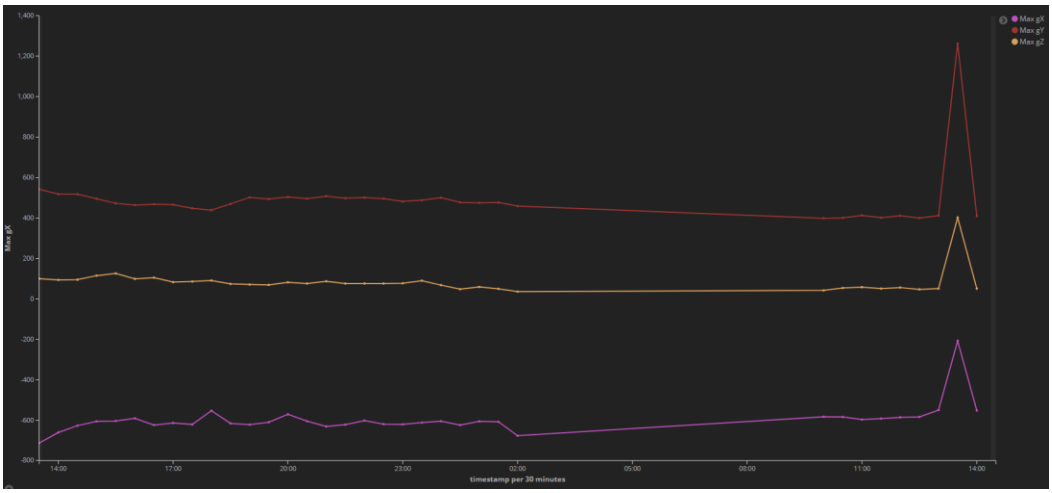


Ilustración 41 Datos recopilados por el giroscopio

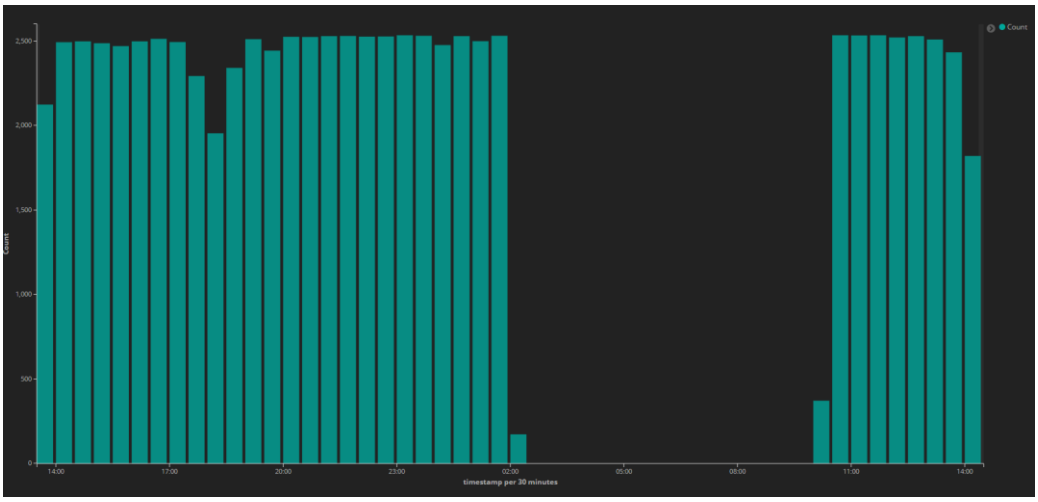


Ilustración 42 Cantidad de datos recogidos cada 30 minutos

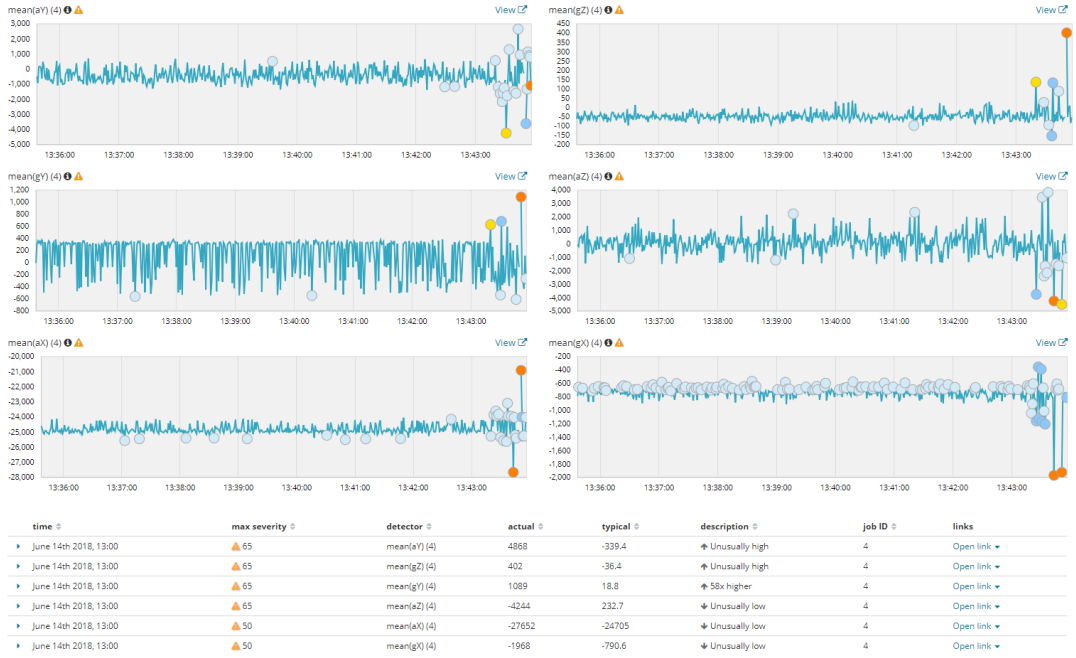
Después de la exposición realizada del sistema de monitorización en tiempo real, se procederá a estudiar los resultados proporcionados por la herramienta de Elasticsearch de Machine Learning.

El entorno de pruebas se inicia a las 13:34, proporcionando la herramienta de Machine Learning las siguientes estadísticas, en función de los datos generados desde el comienzo del experimento.

Ilustración 43 Datos estadísticos proporcionados por la herramienta de Machine Learning

En las gráficas, se pueden observar las variables de análisis tales como la dispersión de los datos, valores medios, valores máximos y mínimos, siendo esta información útil a la hora de supervisar la herramienta.

Respecto al Machine Learning, Kibana nos proporciona las siguientes visualizaciones:



time	max severity	detector	actual	typical	description	job ID	links
June 14th 2018, 13:00	65	mean(aY) (4)	4868	-339.4	Unusually high	4	Open link
June 14th 2018, 13:00	65	mean(gZ) (4)	402	-36.4	Unusually high	4	Open link
June 14th 2018, 13:00	65	mean(gY) (4)	1089	18.8	58x higher	4	Open link
June 14th 2018, 13:00	65	mean(aZ) (4)	-4244	232.7	Unusually low	4	Open link
June 14th 2018, 13:00	50	mean(aX) (4)	-27652	-24705	Unusually low	4	Open link
June 14th 2018, 13:00	50	mean(gX) (4)	-1968	-790.6	Unusually low	4	Open link

Ilustración 44 Pantalla de visualización de incidencias a partir de la media

Las gráficas detectan alertas de anomalía procedentes del fallo creado intencionadamente y registrado a las 13:43. Analizando en detalle el suceso, representado en la ilustración 45, se puede observar como las mediciones se encuentran alejadas de la curva desarrollada por el sistema de aprendizaje, detectando un fallo con posibilidad de avería del ventilador en estudio.

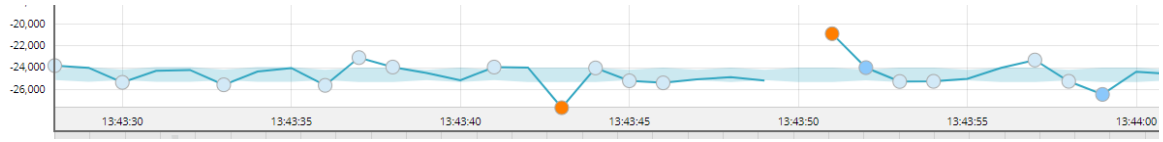


Ilustración 45 Detalle aprendizaje eje x del acelerómetro frente a datos reales y fallo inducido

La herramienta de Machine Learning de Elasticsearch, además permite realizar una previsión de comportamiento según se muestra en la ilustración 46.

Debido a la resolución de los datos capturados por el sensor, realizando mediciones cada intervalo de 700ms como mínimo. Existe dificultad para modelizar este sistema ya que requiere de mayor tiempo de aprendizaje que la licencia demo de Elasticsearch no permite. Por este motivo con los datos capturados hasta el momento, no se puede realizar una mayor definición del aprendizaje requiriendo mayor cantidad de datos para modelizar el movimiento del ventilador:

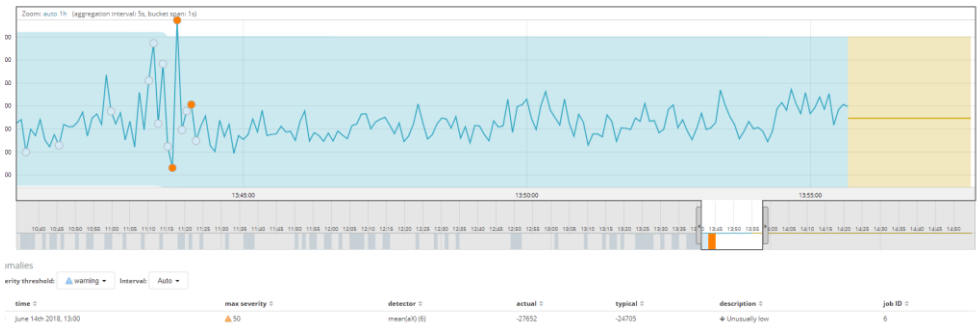


Ilustración 46 Visualización de predicción eje acelerómetro x

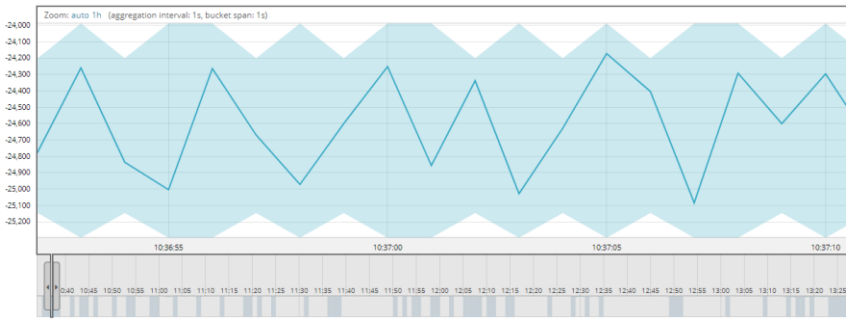


Ilustración 47 Modelización del aprendizaje herramienta Machine Learning

4.2.5 Resolución de problemas

En este apartado se exponen los problemas surgidos a lo largo del desarrollo del proyecto, las soluciones propuestas y algunas de las fuentes de información que han permitido la resolución de estos problemas.

4.2.5.1 Problemas de diseño

En esta sección se analizan los problemas referentes al diseño, las consecuencias que generaron y como se resolvieron.

4.2.5.1.1 Problema de las interferencias:

Desde la conceptualización de este proyecto, se ha tenido en cuenta el problema de su uso en entornos con interferencias ya que los entornos industriales tienen alta predisposición al surgimiento de interferencias provocadas por la maquinaria industrial. Para solucionar este problema, se introdujo la idea de almacenar la información hasta que el sensor pudiera establecer conexión con el interfaz, para posteriormente enviara la información.

El NodeMCU dispone de una memoria de 1 Mb, insuficiente para el almacenamiento de todos los datos que se generaran por el módulo MPU6050.

Como solución a este problema se introdujo a nivel hardware el uso de un lector de tarjetas micro SD para Arduino y a nivel software, un módulo que gestionará la tarea de almacenamiento de datos hasta que se pudiera realizar el envió.

4.2.5.1.2 Problema de la falta de pines:

La solución aportada para el problema anterior generó un problema de falta de pines, ya que este módulo requería del uso de cuatro pines de datos y el módulo MPU6050 ya utilizaba cinco de los ocho disponibles para propósito general.

Además de estos ocho, existían otros dos pines de propósito general, pero con limitaciones, ya que el sistema les atribuía por defecto funciones de detección de interrupción, aportando el pin una señal positiva mientras que el pin que nos faltaba por conectar, pin de interrupción del MPU6050, proporcionaba una señal negativa. Esto generaba la continuidad del circuito, induciendo que al arrancar el NodeMCU, se detectara como modo de programación y que provocaba que el sistema no arrancando el firmware previamente cargado.

Para solucionar esto, se pensó en agregar una puerta NOT al sistema, que no provocara esa activación del modo flash negando la salida del pin de interrupción del MPU6050. Al no disponer de una puerta NOT, se incorporó un chip 74LS27P que contenía tres puertas NOR de tres entradas, y tomando una de ellas se configuro para que cumpliera esta función.

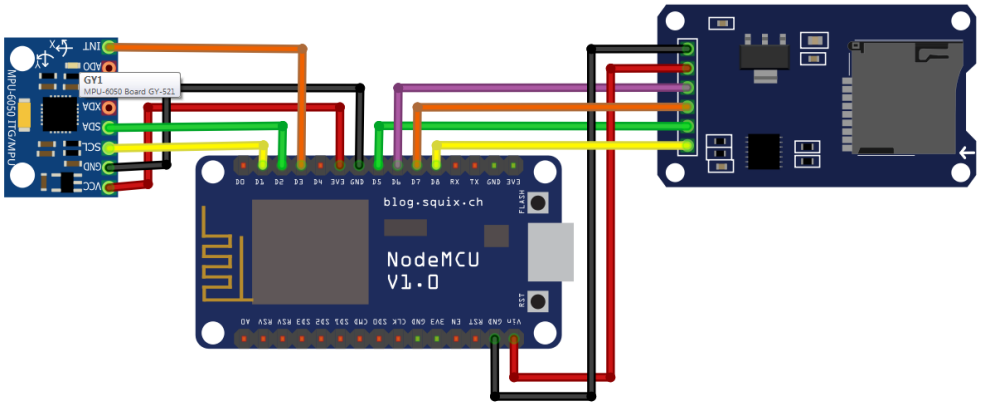


Ilustración 48 Diagrama de conexiones del sensor

**La señal proporcionada por el MPU6050 en el pin de interrupción se considera negada.*

4.2.5.1.3 Problema de las tareas bloqueantes:

Desde las primeras implementaciones, el uso del esquema de ejecución que proporcionaba Arduino con la función *setup()* y la función *loop()*, provocaba diversas limitaciones al incorporar tareas bloqueantes como la espera activa del servidor del moduloOTA o la espera activa del módulo MQTT para detectar publicaciones.

Para solucionar este problema se profundizó en el conocimiento sobre sistemas operativos en tiempo real y las alternativas que existían para implementarlos como se muestra en el apartado 3.2.2.

Tras el análisis de estas implementaciones que solucionaban este tipo de problemática como la de FreeRTOS, se decidió aplicar los conocimientos de programación aprendidos a lo largo del grado e implementar un sistema a partir del uso de interrupciones y tareas programadas.

4.2.5.2 Problemas de implementación

Una vez expuestos los problemas de diseño, se proceden a comentar los problemas de implementación que surgieron a lo largo del desarrollo.

4.2.5.2.1 Problema en la definición de los parámetros de los constructores:

Un problema que se repitió causando numerosos retrasos y problemas como la rotura de una tarjeta micro SD, fue la mala configuración de los módulos DHT y micro SD mediante el bus SPI. Esto se debió principalmente a la falta de documentación detallada para el uso de estos módulos con el microcontrolador ESP8266.

La solución a este problema residió en la correcta configuración de los constructores. Para llegar a la solución se realizaron tareas de depuración exhaustiva, búsqueda de documentación específica sobre el bus SPI y sobre la implementación de esas librerías.

Solución a los constructores:

- ModuloDHT: *DHT dht(DHTPIN, DHTTYPE, 15);*
- Modulo micro SD: *CS pin ESP8266 15 //D8*

4.2.5.2.2 Problema de referencias cíclicas:

En C++ se presenta un problema frente a otros lenguajes orientados a objetos como java que consiste en las limitaciones que provoca el uso de dependencias cíclicas debido a la estructura de archivos que define el proceso de compilación.

Primeramente, los problemas de compilación que se producen son provocados por una redundancia en la definición de las clases que se encuentra en los ".h". Esto se puede solucionar con el uso de clases no definidas, pero su uso implica algunas limitaciones.

Finalmente se consiguió solucionar este problema con el uso del parámetro *this* para acceder a los punteros, y el uso de un método auxiliar *set()*. En la siguiente ilustración se muestra el resumen de este tipo de implementación:

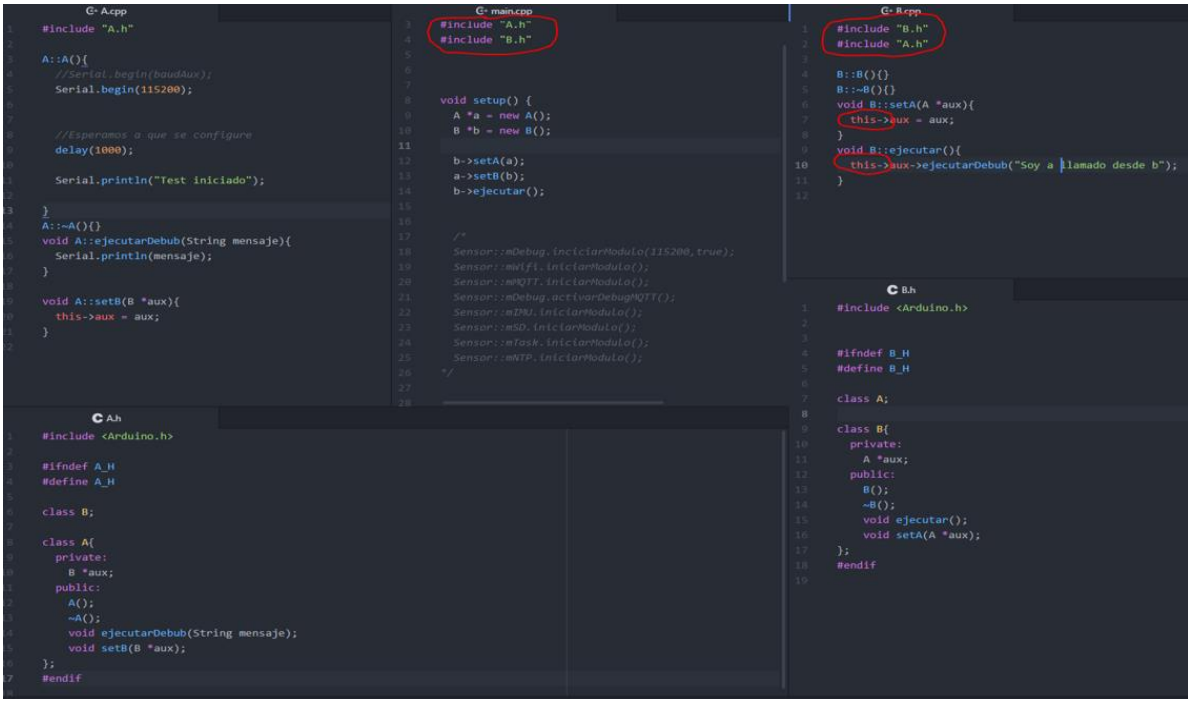


Ilustración 49 Solución aportada para el uso de dependencias cíclicas

Finalmente, no se optó por el uso de esta metodología ya que añadía mucha complejidad a la implementación; asimismo se pensó que los módulos al ser parte del sistema podían ser accesibles unos con otros si fuera necesario, además del problema de salto de contexto durante una interrupción que se comentara en el siguiente apartado.

El uso de una implementación alternativa apoyándose en el uso de variables y métodos estáticos, no reducía la seguridad, ya que los módulos se encapsularon dentro de la clase sensor clase Sensor. Para agregar mayor seguridad se incorporó el patrón de diseño Singleton, como se comentó en la implementación.

4.2.5.2.3 Problema de métodos estáticos:

El uso de interrupciones en el sistema generaba el problema de que el sistema apilaba el estado actual de la memoria saltando a un nuevo contexto de interrupción, donde las variables no estáticas no eran visibles. Esto obligó a que se definieran las clases como estáticas, por ejemplo para acceder al método depuración de la clase ModuloDebug durante la interrupción, o métodos estáticos como *ejecutarInterrupcion()*, encargado de ejecutar proceso al detectar una interrupción.

4.2.5.2.4 Problema falta de elementos de sincronización para la interrupción:

Como se comentó para la implementación del moduloSD, sobre el método de lectura del archivo, en el proceso de adquisición de conocimiento sobre el entorno de Arduino, no se encontró ninguna forma de establecer sincronización con el uso de locks o semáforos.

Para solucionar este problema se realizó la implementación particular para la función de lectura de la tarjeta micro SD en el moduloSD.

4.2.5.2.5 Problema de polimorfismo al aplicar el patrón Abstrac Factory:

Como se expuso en el apartado de implementación, no se pudo completar la definición exacta en el uso del patrón Abstrac Factory. Esto debía a que en C++ las clases abstractas se definen como clases con al menos un método puramente virtual.

El problema surgía a la hora de compilar ya que según se mostraba en el error, el compilador de Atom para el ESP8266 tenía activado la bandera (-fno-rtti) para que no definiera un espacio de memoria destinado al uso de tablas virtuales.

Tras investigar sobre este problema se concluyó que el uso de tablas virtuales en el ESP8266 estaba restringido por el espacio de memoria del microcontrolador.

Por este motivo la definición de la clase Modulo es una clase abstracta, aunque se considera ya que no aporta ninguna funcionalidad más que la relación de los diferentes módulos hijos. De esta forma no tendría sentido instanciar un objeto de la clase Modulo.

En la siguiente referencia e puede obtener más información sobre el uso de esta bandera [14].

4.2.5.2.6 Problema de la velocidad de las interrupciones:

La ejecución de interrupciones sin control generaba un desbordamiento de la pila; esto se debía a que la llegada de interrupciones al microcontrolador era mayor que la velocidad con la que este las ejecutaba.

Para solucionar este problema, se propuso el uso de un control de interrupciones mediante la anotación del momento en el que se comenzara a ejecutar la interrupción, y el establecimiento de un intervalo entre lanzamiento de ejecuciones. De esta forma si se generaran interrupciones en un tiempo menor del intervalo, estas no se ejecutarían.

Capítulo 5: Presupuesto y horas materiales

5 PRESUPUESTO Y HORAS MATERIALES

En este capítulo se analizan de los costes asociados al proyecto estableciendo una estimación presupuestal a nivel económico y a nivel de esfuerzo de desarrollo. Este presupuesto analizará los costes de desarrollo, los costes de producción, los gastos generales y el beneficio industrial.

5.1 COSTES DE DESARROLLO

Es este apartado se realizará una estimación del coste de desarrollo a través de los informes generados a partir de la planificación temporal del proyecto y siguiendo la metodología establecida en el anteproyecto de una adaptación de Scrum para una persona. Respecto a la planificación proporcionada en el anteproyecto se han realizado algunas modificaciones para representar mejor la experiencia de desarrollo del proyecto.

Como se ha ido mostrando a lo largo de esta memoria, el proyecto ha estado compuesto por una primera fase de adquisición de conocimiento denominada estado del arte, seguida de una fase de diseño donde a partir del conocimiento adquirido se ha presentado una solución a un problema de industria 4.0, se ha seguido con una fase de implementación donde se ha llevado a cabo el desarrollo del prototipo de sensorización, y por último se ha terminado con una fase de implantación y pruebas donde se han mostrado las diversas pruebas y resultados obtenidos a partir del prototipo.

Todas estas fases junto con su estimación de duración y atendiendo a una jornada de trabajo de 5 horas durante 6 días a la semana, junto con el uso de determinados recursos, se reflejan en las siguientes ilustraciones:

1		**Prototipo de sensorizacion para la industria 4.0**	**56.25 días**	**jue 2/1/18**	**vie 6/1/18**
1.1		**Fase de estado del arte**	**10 días**	**jue 2/1/18**	**lun 2/19/18**
1.1.1		Estudio y análisis del estado de la industria 4.0	10 horas	jue 2/1/18	vie 2/2/18
1.1.2		Estudio y análisis de placas controladoras	15 horas	dom 2/4/18	mar 2/6/18
1.1.3		Estudio y análisis de RTOS	5 horas	dom 2/18/18	dom 2/18/18
1.1.4		Estudio y análisis de sensores	10 horas	dom 2/11/18	lun 2/12/18
1.1.5		Estudio y análisis de protolocos de comunicación M2M	10 horas	mar 2/13/18	mié 2/14/18
1.1.6		Estudio y análisis de arquitecturas de red para sensorización	10 horas	jue 2/15/18	vie 2/16/18
1.1.7		Estudio y análisis de bases de datos y sistemas de procesamiento	15 horas	mié 2/7/18	vie 2/9/18
1.1.8		Estudio y análisis teoria del mantenimiento	5 horas	lun 2/19/18	lun 2/19/18
1.1.9		Fin de fase	0 días	lun 2/19/18	lun 2/19/18

Ilustración 50 Fase estado del arte. Diagrama Project

1.2	**Fase de diseño**	**5.63 días**	**lun 2/19/18**	**jue 3/1/18**
1.2.1	Eleción y montaje de los elementos hardware	15 horas	mar 2/20/18	jue 2/22/18
1.2.2	Eleción del protocolo M2M	5 horas	lun 2/26/18	lun 2/26/18
1.2.3	Eleción de la base de datos y el sistema de procesamiento	10 horas	vie 2/23/18	dom 2/25/18
1.2.4	Diseño del sistema jerarquico de red	5 horas	mar 2/27/18	mar 2/27/18
1.2.5	**Sprint planing**	**5.63 días**	**lun 2/19/18**	**jue 3/1/18**
1.2.5.1	Product backlog	5 horas	mié 2/28/18	mié 2/28/18
1.2.5.2	Sprint backlog	5 horas	jue 3/1/18	jue 3/1/18
1.2.5.3	Fin de fase	0 días	lun 2/19/18	lun 2/19/18
1.2.6	Fin de fase	0 días	jue 3/1/18	jue 3/1/18

Ilustración 51 Fase diseño. Diagrama Project

1.3	**Fase de Implementación**	**30 días**	**vie 3/2/18**	**dom 5/13/18**
1.3.1	**Sprint**	**30 días**	**vie 3/2/18**	**dom 5/13/18**
1.3.1.1	**Sprint 1**	**7.5 días**	**vie 3/2/18**	**jue 3/15/18**
1.3.1.1.1	Implementación del sistema operativo para el sensor	60 horas	vie 3/2/18	jue 3/15/18
1.3.1.1.2	Fin de fase	0 días	jue 3/15/18	jue 3/15/18
1.3.1.2	**Sprint 2**	**7.5 días**	**vie 3/16/18**	**vie 4/13/18**
1.3.1.2.1	Implementación del sistema operativo para el sensor	50 horas	vie 3/16/18	mié 4/11/18
1.3.1.2.2	Implementación del interfaz de comunicación	10 horas	jue 4/12/18	vie 4/13/18
1.3.1.2.3	Fin de fase	0 días	vie 4/13/18	vie 4/13/18
1.3.1.3	**Sprint 3**	**7.5 días**	**dom 4/15/18**	**vie 4/27/18**
1.3.1.3.1	Implementación del sistema operativo para el sensor	50 horas	dom 4/15/18	mié 4/25/18
1.3.1.3.2	Implementación del interfaz de comunicación	10 horas	jue 4/26/18	vie 4/27/18
1.3.1.3.3	Fin de fase	0 días	vie 4/27/18	vie 4/27/18
1.3.1.4	**Sprint 4**	**7.5 días**	**dom 4/29/18**	**dom 5/13/18**
1.3.1.4.1	Implementación del sistema operativo para el sensor	30 horas	dom 4/29/18	dom 5/6/18
1.3.1.4.2	Implementación del interfaz de comunicación	10 horas	vie 5/11/18	dom 5/13/18
1.3.1.4.3	Instalación y configuración de la base de datos y el sistema de procesamiento	20 horas	lun 5/7/18	jue 5/10/18
1.3.1.4.4	Fin de fase	0 días	dom 5/13/18	dom 5/13/18
1.3.1.5	Fin de fase	0 días	dom 5/13/18	dom 5/13/18

Ilustración 52 Fase implementación. Diagrama Project

1.4	**Fase de implantacion y pruebas**	**10.63 días**	**lun 5/14/18**	**vie 6/1/18**
1.4.1	Resolución de problemas	60 horas	lun 5/14/18	dom 5/27/18
1.4.2	Pruebas en un entorno industrial	25 horas	lun 5/28/18	vie 6/1/18
1.4.3	Fin de fase	0 días	vie 6/1/18	vie 6/1/18

Ilustración 53 Fase implantación y pruebas. Diagrama Project

Una vez mostrado el desarrollo de la planificación y las fases, se procede a especificar los detalles de los costos. En ellos primeramente se muestran los costos referentes los trabajadores teniendo en cuenta el 30% del sueldo destinado a la seguridad social, y en la siguiente ilustración donde se muestra el coste de los trabajadores frente a los medios materiales utilizados:

Nombre	Trabajo real	Costo real	Tasa estándar
Juan Bautista Cita Muñoz (Ingeniero)	210 horas	4,095.00 €	19.50 €/hora
Juan Bautista Cita Muñoz (Programador)	240 horas	3,120.00 €	13.00 €/hora

Ilustración 54 Detalles costos de todos los recursos de trabajo

En la tasa estándar se incluye el coste de los empleados junto con el 30% correspondiente de la seguridad social.

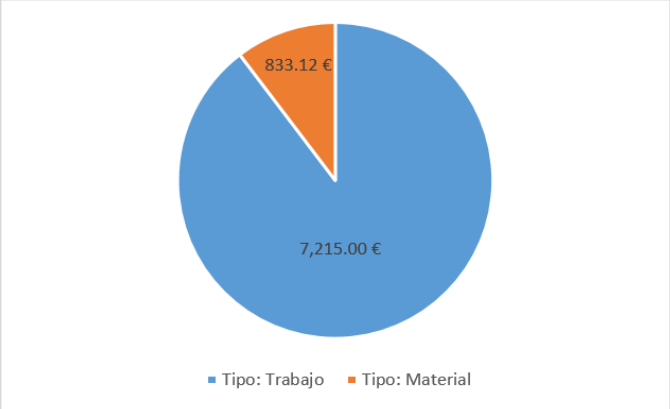


Ilustración 55 Distribución de costos

El coste total de desarrollo asciende a: **8.048,12 €** mientras que para desarrollarlo se ha necesitado de **450 horas aproximadamente**.

5.2 COSTE DE PRODUCCIÓN

Se entiende como costes de producción al coste que supone la fabricación del sistema de sensorización. Estos costes se pueden dividir en costes de producción del sensor, costes de producción del interfaz y coste de producción de servidor.

Hay que destacar que para el coste del servidor se proponen dos escenarios diferentes. El coste del sistema en un entorno local y en un entorno en la nube.

Por otro lado, el coste del sistema en un entorno local, no se ha podido completar ya que, tras una consulta sobre el coste de la licencia, Elasticsearch no ha respondido, entendiendo que no estaban interesados en proporcionar el precio de la licencia para el prototipo. Mientras, el coste del servicio en la nube si se ha podido consultar a partir de su página web, siendo este de 439 euros al mes, definido a partir de la siguiente configuración:

- 8 GB de RAM
- 192 GB de espacio en disco SSD
- Servidor en GCP (Google Cloud Platform) situado en Bélgica
- Replicado en dos centros de datos

Coste de produccion del Sensor	
NodeMCU	4.03
Adaptador de tarjetas micro SD	3.7
Tarjeta micro SD 4Gb	5.88
MPU6050	2.1
74LS27P	0.2
Cable para Arduino	0.5
Cable micro USB 3 metros	6
Cargador USB	1
Carcasa prototipo	5
Coste de montaje	2.17
Total:	**30.58**
Numero sensores:	1
Coste de produccion del Interfaz	
Raspberry Pi 3 B+	42.45
Tarjeta micro SD 32Gb	11.77
Transformador Raspberry Pi	8.99
Router banda dual	40
Carcasa Raspberry Pi	10
Total:	**113.21**
Coste de produccion del Servidor Local	
Coste de los equipos	3000
Coste de mantenimiento	300
Coste de la licencia de Elasticsearch	0
Total:	**3300**
Coste total producción:	3443.79
Coste de produccion del Servidor en la nube	
Coste del servicio	439
Total:	**439**
Coste total producción:	582.79

Ilustración 56 Coste de producción

En este coste de producción no se contempla el software desarrollado como un coste, esto se debe a que para ello se necesitaría realizar una estimación con diferentes casos de venta que permitieran amortizar el coste de su desarrollo, provocando que este proyecto se desviara de su objetivo. Por lo tanto, el coste de producción que se expone es una aproximación del coste real que tendría la producción.

Capítulo 6: Resumen, conclusiones y líneas futuras

6 RESUMEN, CONCLUSIONES Y LÍNEAS FUTURAS

6.1 RESUMEN

A lo largo de este proyecto se han mostrado las numerosas referencias y fuentes que han permitido crear una base de conocimiento sobre la que se ha desarrollado el prototipo de sensorización.

Esta base de conocimiento se ha formado a partir del estudio y análisis de placas microcontroladoras, elementos de sensorización, teoría de sistemas operativos en tiempo real, protocolos de comunicación entre maquinas, arquitecturas de redes de sensorización, bases de datos orientadas a documentos y teoría del mantenimiento industrial.

Todo este conocimiento junto con los conocimientos adquiridos a lo largo del grado, han permitido desarrollar un prototipo de sensorización compuesto por tres partes. El sensor, elemento encargado de capturar los datos generados por la maquinaria industrial; el interfaz, encargado de ser el intermediario entre la red de sensorización y el servidor; y por último el servidor, encargado del almacenamiento y procesado de datos.

Además de esta separación jerárquica que permite separar las funcionalidades, se ha profundizado en el desarrollo de un sistema operativo para el sensor, enfrentando el desarrollo a problemas típicos de sistemas operativos en tiempo real. Como solución se ha propuesto un programa modularizado, seguro, de alta cohesión y bajo acoplamiento, que incorpora el uso de patrones de diseño.

Por otro lado, en la parte del interfaz se ha descubierto la herramienta Node-Red con grandes funcionalidades y que puede ser útil para el desarrollo de otros tipos de proyectos de IoT.

Por último, el uso de una base de datos en tiempo real como se define Elasticsearch junto con su herramienta de Machine Learning, ha permitido completar el proyecto proporcionando información a partir de los datos recopilados sobre alertas de mantenimiento; logrando así el objetivo de obtener un sistema que permita la realización del mantenimiento predictivo además de la optimización del proceso productivo a partir de los datos estadísticos generados.

6.2 CONCLUSIONES

A pesar de las dificultades que ha comprendido la realización de este proyecto debido a la falta de previa experiencia con las tecnologías con las que se ha desarrollado el prototipo, además de la documentación no poca, pero difícil de encontrar debido a la tecnicidad y calidad que requería este trabajo. Se ha logrado desarrollar el prototipo de sensorización para el cual se había enfocado el presente trabajo final de grado.

En todo ello hay que destacar, que el esfuerzo se ha centrado sobre el desarrollo del sistema operativo del sensor y de la estructura del sistema, dejando en un nivel inferior la parte de procesamiento y almacenamiento de información. A pesar de todos los problemas e inconvenientes encontrados en el proceso, se ha conseguido encontrar una solución adecuada, que ha culminado con éxito, el objetivo propuesto para el conjunto del prototipo.

Por último, para finalizar, considerar que los altos costes del esfuerzo de desarrollo se han compensado con el bajo coste de los elementos electrónicos utilizados, proporcionando así un dispositivo viable para ser utilizado en la industria, previo a un esfuerzo de adaptación al entorno industrial especifico.

6.3 LÍNEAS FUTURAS

Este proyecto establece una base de desarrollo y unas fuentes de conocimiento a modo de guía a partir de las cuales proseguir con el desarrollo de proyectos enfocados a la industria 4.0 o a IoT, con elementos de sensorización.

Atendiendo a los aspectos a mejorar de este proyecto, la incorporación de nuevas funcionalidades sobre el software desarrollado para el sensor facilitará su implantación, configuración y puesta marcha. Alguna de estas mejoras podría considerar la incorporación de un módulo SSPIF que permitiese el almacenamiento de una configuración por defecto en la memoria del microcontrolador, o configuraciones como la automática creación de un punto de acceso con un "frot end" que permitiera su configuración en caso de que el sensor no haya sido configurado.

Otra posible mejora esta vez en la sección del interfaz, incorporaría la instalación y configuración de un servicio de VPN (Virtual Private Network) como OpenVPN o Hamachi. Esto permitiría crear un punto de acceso global y seguro a la red del sistema de sensorización, permitiendo realizar operaciones de mantenimiento y actualización desde una red externa apoyándose en el módulo OTA.

Por último, la globalidad del desarrollo del sistema se podría alcanzar desarrollando un sistema propio de procesamiento de datos eliminado así el coste de licencias que esta fase actualmente conlleva.

Capítulo 7: Bibliografía

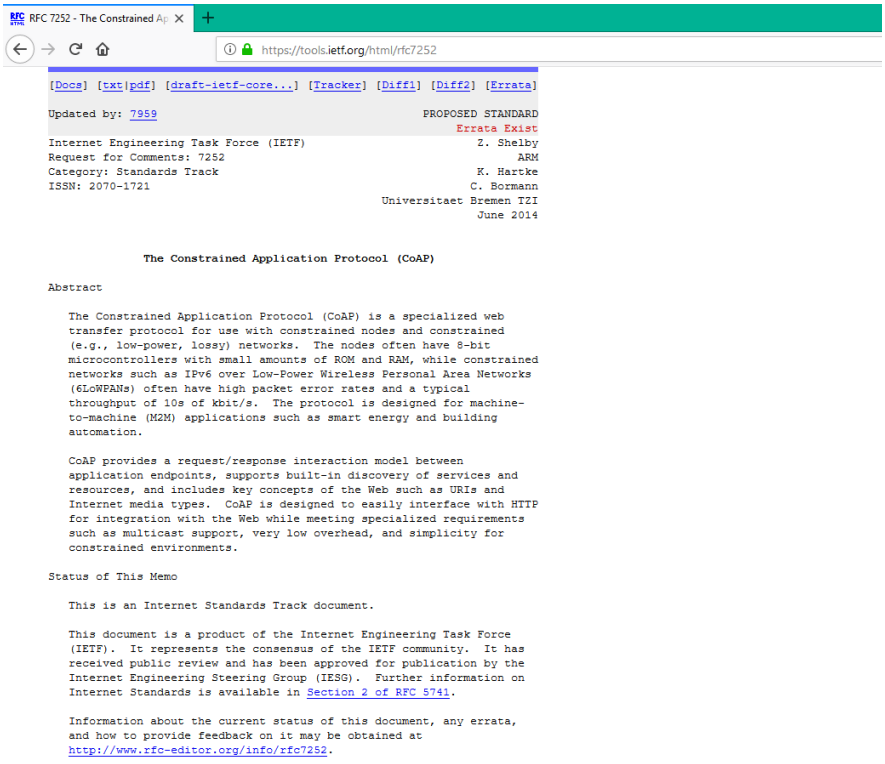
7 BIBLIOGRAFÍA

[1] «XMPP,» [En línea]. Available: https://xmpp.org. [Último acceso: 6 6 2018].

[2] «mqtt.org,» [En línea]. Available: http://mqtt.org/. [Último acceso: 4 30 2018].

[3] G. Grunwald, «Sensor Networks with Java on Embedded,» IoT Conference, 29 6 2015. [En línea]. Available: https://www.youtube.com/watch?v=6h8_bZ9D8G4. [Último acceso: 12 2 2018].

[4] «rfc7252,» 2018. [En línea]. Available: https://tools.ietf.org/html/rfc7252.


RFC 7252 - The Constrained Ap

https://tools.ietf.org/html/rfc7252

[Docs] [txt|pdf] [draft-ietf-core...] [Tracker] [Diff1] [Diff2] [Errata]

Updated by: 7959 — PROPOSED STANDARD — Errata Exist

Internet Engineering Task Force (IETF) — Z. Shelby, ARM
Request for Comments: 7252 — K. Hartke
Category: Standards Track — C. Bormann
ISSN: 2070-1721 — Universitaet Bremen TZI
June 2014

The Constrained Application Protocol (CoAP)

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7252.

[5] X. W. W. W. Ian F. Akyildiz, «Wireless mesh networks: a survey.,» *Elsevier,* p. 43, 2004.

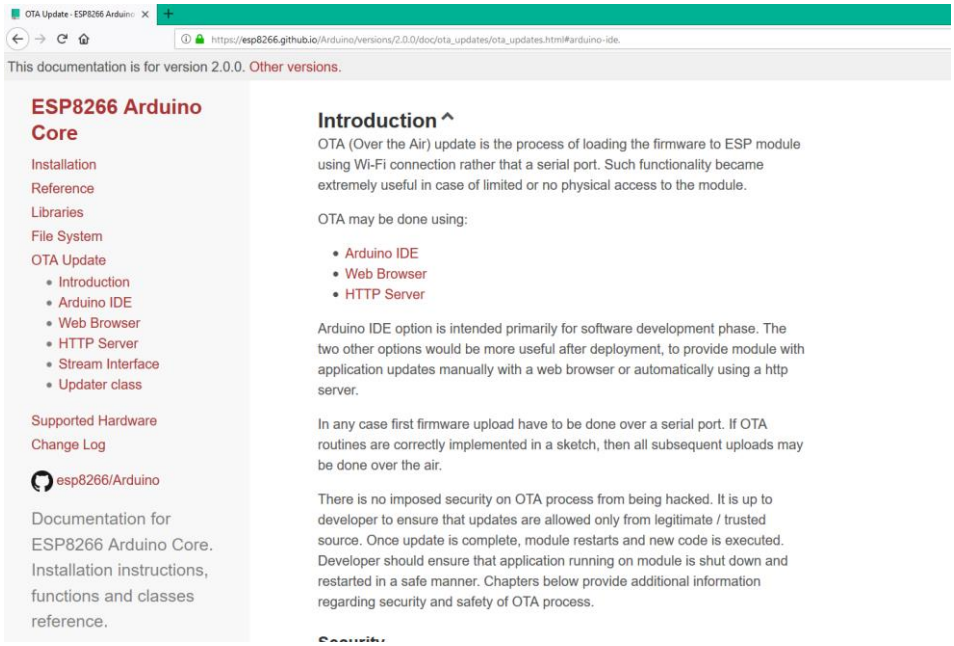
[6] M. B. M. Abella, «Mantenimiento Industrial,» Universidad Carlos III de Madrid.

[7] F. &. S. «From concept to production: a 5-step approach towards successful Industry 4.0 projects».

[8] D. H. R. Alonso, «Vibraciones en maquinas 2. Master en Ingenieria Industrial,» Universidad Carlos III.

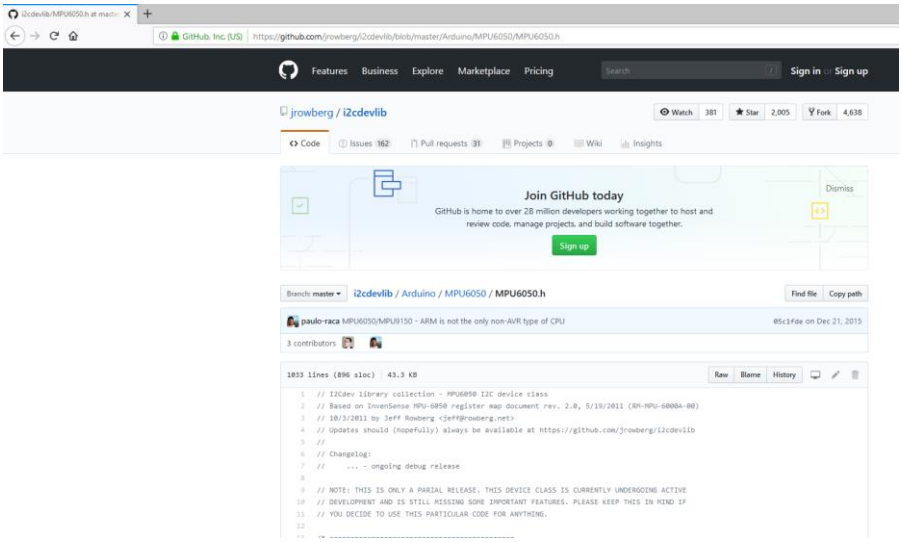
[9] «Platformio instalación Atom,» [En línea]. Available: https://platformio.org/get-started/ide?install=atom. [Último acceso: 6 6 2018].

[10] «OTA Updates ESP8266,» [En línea]. Available: https://esp8266.github.io/Arduino/versions/2.0.0/doc/ota_updates/ota_updates.html#arduino-ide. [Último acceso: 19 5 2018].

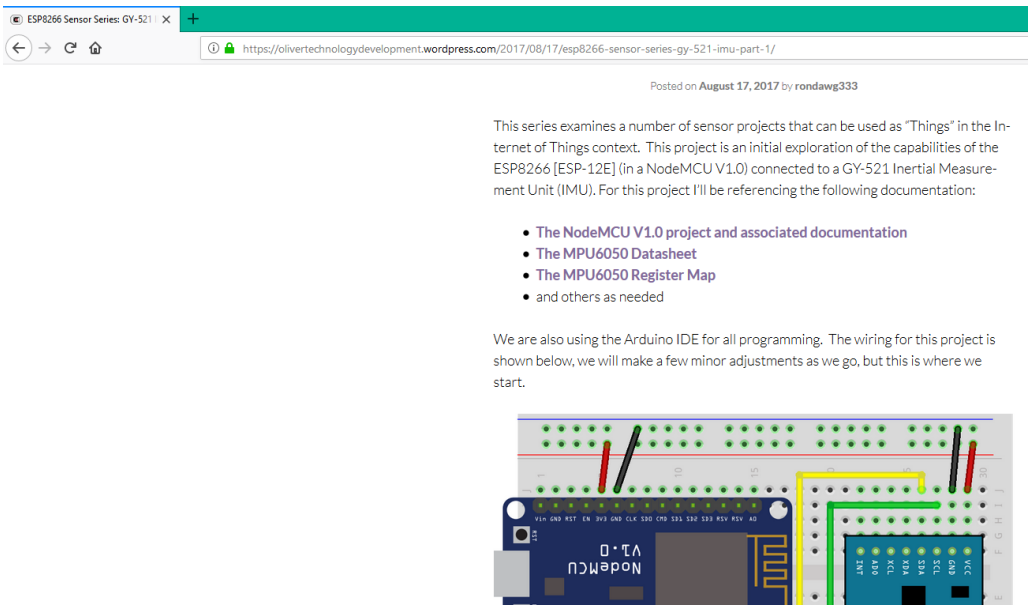


[11] SteveQuinn, «ESP8266-SPIFFS,» [En línea]. Available: http://www.instructables.com/id/Using-ESP8266-SPIFFS/. [Último acceso: 2018 5 19].

[12] «Repositorio MPU6050.h,» [En línea]. Available: https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050.h. [Último acceso: 21 5 18].



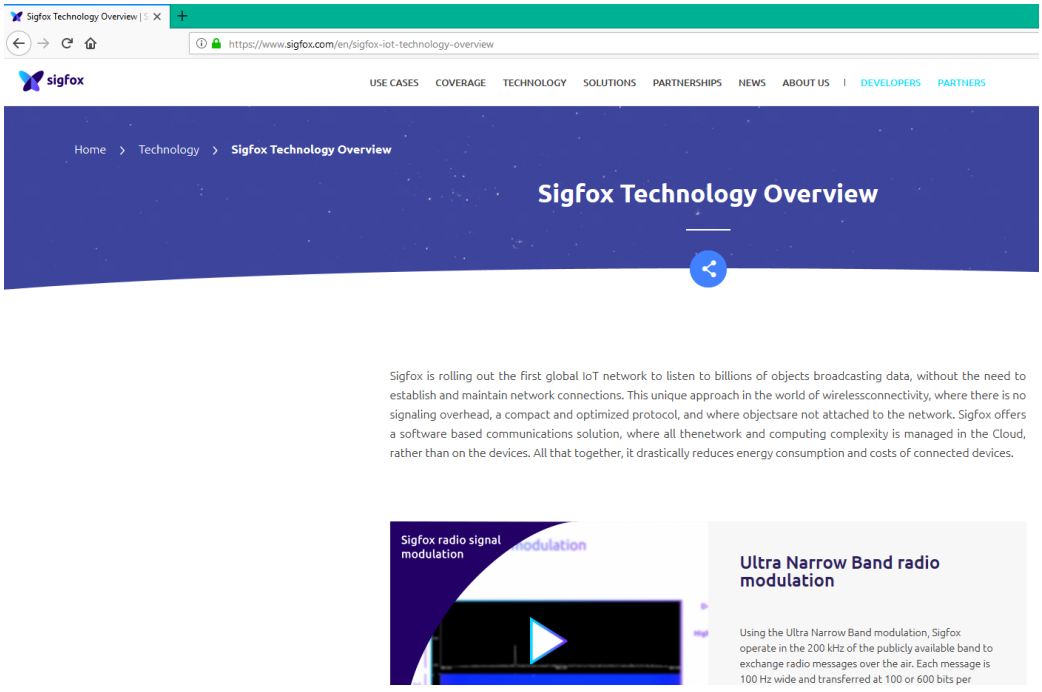
[13] «Oliver Technology Development,» [En línea]. Available: https://olivertechnologydevelopment.wordpress.com/2017/08/17/esp8266-sensor-series-gy-521-imu-part-1/. [Último acceso: 21 5 18].



[14] «Información flags compilacion c++,» [En línea]. Available: http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/C_002b_002b-Dialect-Options.html. [Último acceso: 7 6 2018].

[15] D. Reinsel, J. Gantz y J. Rydning, «Data Age 2025: The Evolution of Data to Life-Critical,» IDC Sponsored by Segagate, 2017.

[16] «www.sigfox.com,» [En línea]. Available: https://www.sigfox.com/en/sigfox-iot-technology-overview. [Último acceso: 1 4 2018].



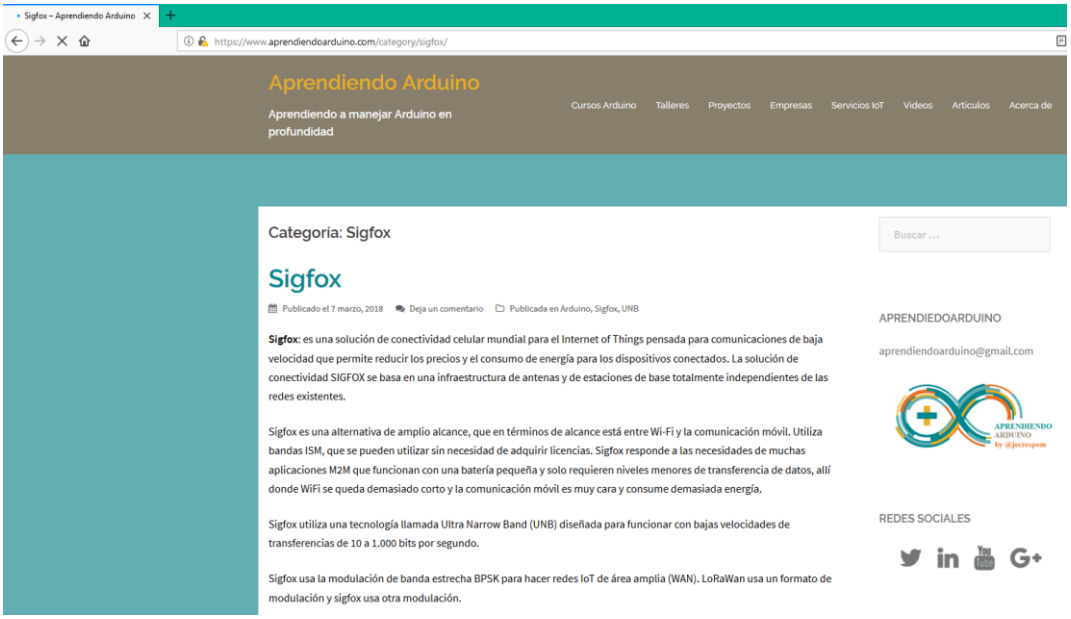
[17] «www.digi.com,» [En línea]. Available: https://www.digi.com/products/xbee-rf-solutions/digimesh. [Último acceso: 1 4 2018].

[18] «docs.oasis-open.org,» [En línea]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [Último acceso: 1 4 2018].

[19] «www.iso.org,» [En línea]. Available: https://www.iso.org/standard/69466.html. [Último acceso: 5 4 2018].

[20] D. M. Weiser, «Computer Science Challenges for the Next 10 Years,» 1996. [En línea]. Available: https://www.youtube.com/watch?v=7jwLWosmmjE. [Último acceso: 20 4 2018].

[21] E. Crespo, «Sigfox,» [En línea]. Available: https://www.aprendiendoarduino.com/category/sigfox/. [Último acceso: 4 30 2018].



[22] «Curso freeRTOS & ESP32,» 2 4 2017. [En línea]. Available: https://www.youtube.com/watch?v=Z7Z4_ENFjDM. [Último acceso: 2 4 2018].

[23] N. Instruments, «Do I Need a Real-Time System?,» 22 11 2013. [En línea]. Available: http://www.ni.com/white-paper/14238/en/. [Último acceso: 1 4 2018].

[24] E. Crespo, «Microcontroladores Arduino a Fondo,» [En línea]. Available: https://aprendiendoarduino.wordpress.com/tag/arquitectura-microcontrolador/. [Último acceso: 10 4 2018].

[25] «Arduino y las interrupciones,» [En línea]. Available: https://www.prometec.net/interrupciones/. [Último acceso: 10 4 2018].

[26] ZigBee, «ZigBee Specification FAQ,» 11 11 2014. [En línea]. Available: https://web.archive.org/web/20130627172453/http:/www.zigbee.org/Specifications/ZigBee/FAQ.aspx. [Último acceso: 15 4 2018].

[27] E. Crespo, «Bus: I2C,» 2017. [En línea]. Available: https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/. [Último acceso: 10 4 2018].

[28] «Veleda Service LTD,» [En línea]. Available: http://veleda.ca/optimum-maintenance/. [Último acceso: 1 5 2018].

[29] «Statista: Previsión dispositivos conectados,» [En línea]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Último acceso: 2018 05 10].

Capítulo 8: Anexos

8 ANEXOS

8.1 INSTALACIÓN Y CONFIGURACIÓN DE ELASCTICK SEARCH

CONCEPTOS

Conceptos:

- Índice: colección de documentos. Ejem películas
- Tipo: Permite dividir la categoría de índice. Ejem: acción
- Shards: La división de un índice a modo de réplica perfectamente operativo y funcional.
- Replicas: Copia de un índice
- Nodo: Parte de un clúster, almacena parte de los datos del clúster y participa en como parte de la información.
- Clúster: Colección de nodos. Contienen toda la información.

COMANDOS UTILES

```
sudo lsof -i -P -n | grep LISTEN
```

INSTALACION JAVA

```
sudo add-apt-repository -y ppa:webupd8team/java
sudo apt-get update
sudo apt-get update && sudo apt-get -y install oracle-java8-installer
```

INSTALACION ELASTIK SEARCH

```
//https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html
//Puerto 9200
//IP: 192.168.1.250
//Ruta logs: /var/log/elasticsearch/
//Ruta instalacion: /usr/share/elasticsearch/
//Ruta configuracion: /etc/elasticsearch/


//Instalamos apt-trasnport-https
sudo apt-get install apt-transport-https

//Guardamoe el repositorio:
echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo tee
-a /etc/apt/sources.list.d/elastic-6.x.list

//Instalamos desde el repositorio
sudo apt-get update && sudo apt-get -y install elasticsearch

//Configuramos elastic search para que se inicie automaticamente:
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service

//Iniciamos el servicio
sudo systemctl start elasticsearch.service

//Instalamos curlo para comprobar que elastic search se ha instalado
adecuadamente
sudo apt-get install curl -y

//Habilitamos el log:
sudo journalctl -f
sudo journalctl --unit elasticsearch
```

CONFIGURACION ELASTICK SEARCH

```
//Configuramos elastic search
sudo nano /etc/elasticsearch/elasticsearch.yml
network.host: 192.168.1.250


//Accedemos como su y escribimos:
echo "network.host: 192.168.1.250" >> /etc/elasticsearch/elasticsearch.yml

//Es necesario configurar ubuntu para que tenga una ip fija y el router para
que permita a la maquina poseer un a ip fija
```

INSTALACIÓN DE X-PACK ELASTICSEARCH

```
cd /usr/share/elasticsearch/bin/
sudo ./elasticsearch-plugin install x-pack
```

Configuracion de x-pack elasticsearch

```
//Configuramos elastic search
sudo nano /etc/elasticsearch/elasticsearch.yml
action.auto_create_index: true
#action.auto_create_index:
.security,.monitoring*,.watches,.triggered_watches,.watcher-history*,.ml*

//Configuramos  (TLS/SSL)
Como el cluster esta compuesto unicamente de un nodo, mientras que el interfaz
que utiliza es externo,
pero trabaja sobre un entorno local. Por lo tanto no activamos ssl.

//Establecemos la contraseña de acceso para los usuarios admin:
cd /usr/share/elasticsearch/bin/x-pack/
sudo ./setup-passwords interactive
usuarios:
       -elastic
       -kibana
       -logstash_system
(Establecemos 123456)
```

INSTALACION KIBANA

```
//Puerto 5601
//IP: 192.168.1.250
//Configuracion: /etc/kibana/kibana.yml
//Ruta instalacion: /usr/share/kibana/


sudo apt-get update && sudo apt-get install kibana

sudo /bin/systemctl daemon-reload && sudo /bin/systemctl enable kibana.service
&& sudo systemctl start kibana.service
```

CONFIGURACION KIBANA

```
sudo nano /etc/kibana/kibana.yml
server.host: "192.168.1.250"
elasticsearch.url: "http://192.168.1.250:9200"
elasticsearch.username: "kibana"
elasticsearch.password: "123456"
```

INSTALACIÓN DE X-PACK KIBANA

```
cd /usr/share/kibana/bin/
sudo -u kibana ./kibana-plugin install x-pack
```

```
CONFIGURACION DE X-PACK KIBANA


sudo nano /etc/kibana/kibana.yml

//Añadimos:
//elasticsearch.username: "kibana"
//elasticsearch.password: "123456"
```

8.2 INSTALACION DE MOSQUITTO

```
//Instalamos mosquitto
sudo apt-get update && sudo apt-get install mosquito mosquitto-clients


//Lo Podemos probar en dos terminales:
//Subscriptor
mosquitto_sub -h localhost -t test

//Publicador
mosquitto_pub -h localhost -t test -m "hello world"

//Instalamos certbot para securizar el broker. Cerbot se encargará de
//cifrar los certificados.

sudo apt-get install certbot

//Agregamos usuarios y contraseNas al brocker

sudo mosquitto_passwd -c /etc/mosquitto/passwd esp8266
sudo mosquitto_passwd -c /etc/mosquitto/passwd nodeRed

//Modificamos la configuración para que el broquer solicite
//identificacion

sudo echo >> /etc/mosquitto/conf.d/default.conf
echo allow_anonymous false >> /etc/mosquitto/conf.d/default.conf
echo password_file /etc/mosquitto/passwd >> /etc/mosquitto/conf.d/default.conf

//Reiniciamos el servicio
sudo systemctl restart mosquitto


//Creamos un MQTT Broker
sudo apt-get install mosquito
//Una vez instalado cada vez que se inicie el so el bróker se iniciara
//Apotamos seguridad al bróker
Etc/mosquitto  -------- sudo mosquitto_passwd -c passwordfile [nombreUsuario]

//Iniciar mosquito
sudo /etc/init.d/mosquitto start
```

8.3 INSTALACIÓN DE NODE-RED

```
update-nodejs-and-nodered
sudo npm install -g node-red-admin
cd ~/.node-red/
sudo npm install -g node-red-admin

//Generamos hash para la contraseña de los usuarios
node-red-admin hash-pw
sudo nano settings.js
```

Modificamos el usuario admin agregando nuestro hash, y podemos crear otros usuarios.

Además, habilitamos el http autentificación.

```
adminAuth: {
    type: "credentials",
    users: [{
        username: "admin",
        password: "$2a$08$CbeXhlcnGqgazHPltWy/ReDmgOBP/PNM4ghmqUNZ40rYdHkcE$
        permissions: "*"
    },

    {   username: "guest",
        password:"",
        permissions: "read"
    }]
},
```

```
    }]
},

// To password protect the node-defined HTTP endpoints (httpNodeRoot), or
// the static content (httpStatic), the following properties can be used.
// The pass field is a bcrypt hash of the password.
// See http://nodered.org/docs/security.html#generating-the-password-hash
httpNodeAuth: {user:"admin",pass:"$2a$08$CbeXhlcnGqgazHPltWy/ReDmgOBP/PNM4g$
//httpStaticAuth: {user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dN$
```

Configuramos para que node red se inicie automáticamente al iniciarse la rasberry

```
sudo systemctl enable nodered.service
```

reiniciamos las raspberry y todo debería de estar listo.

Para acceder a node red, accedemos desde un navegador a http://IPRASPBERRY:1880/

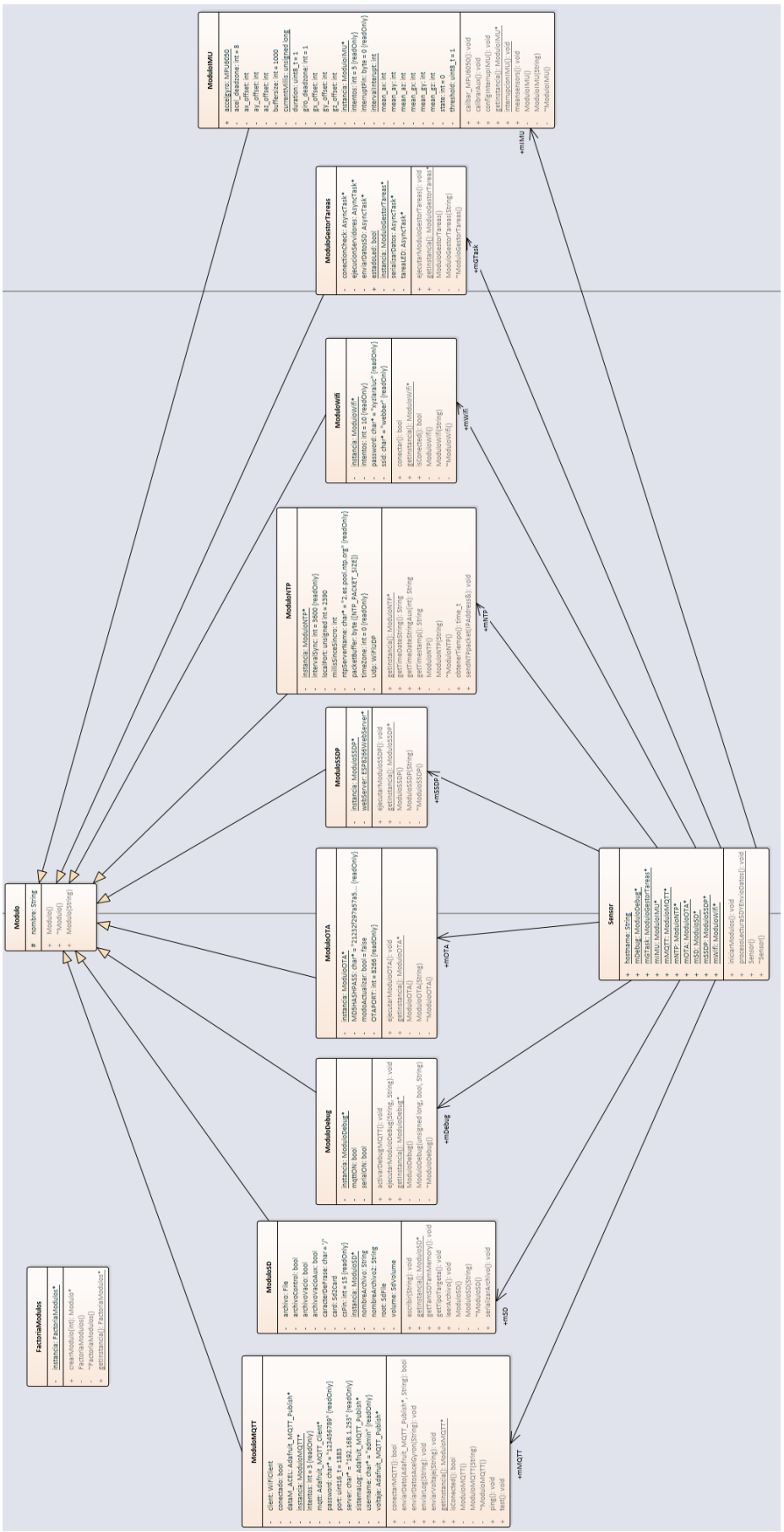
Otros:

- Para obtener el menú de / managepallete/ install/ (buscamos dashboard)/ instalamos node-red-dashboard.

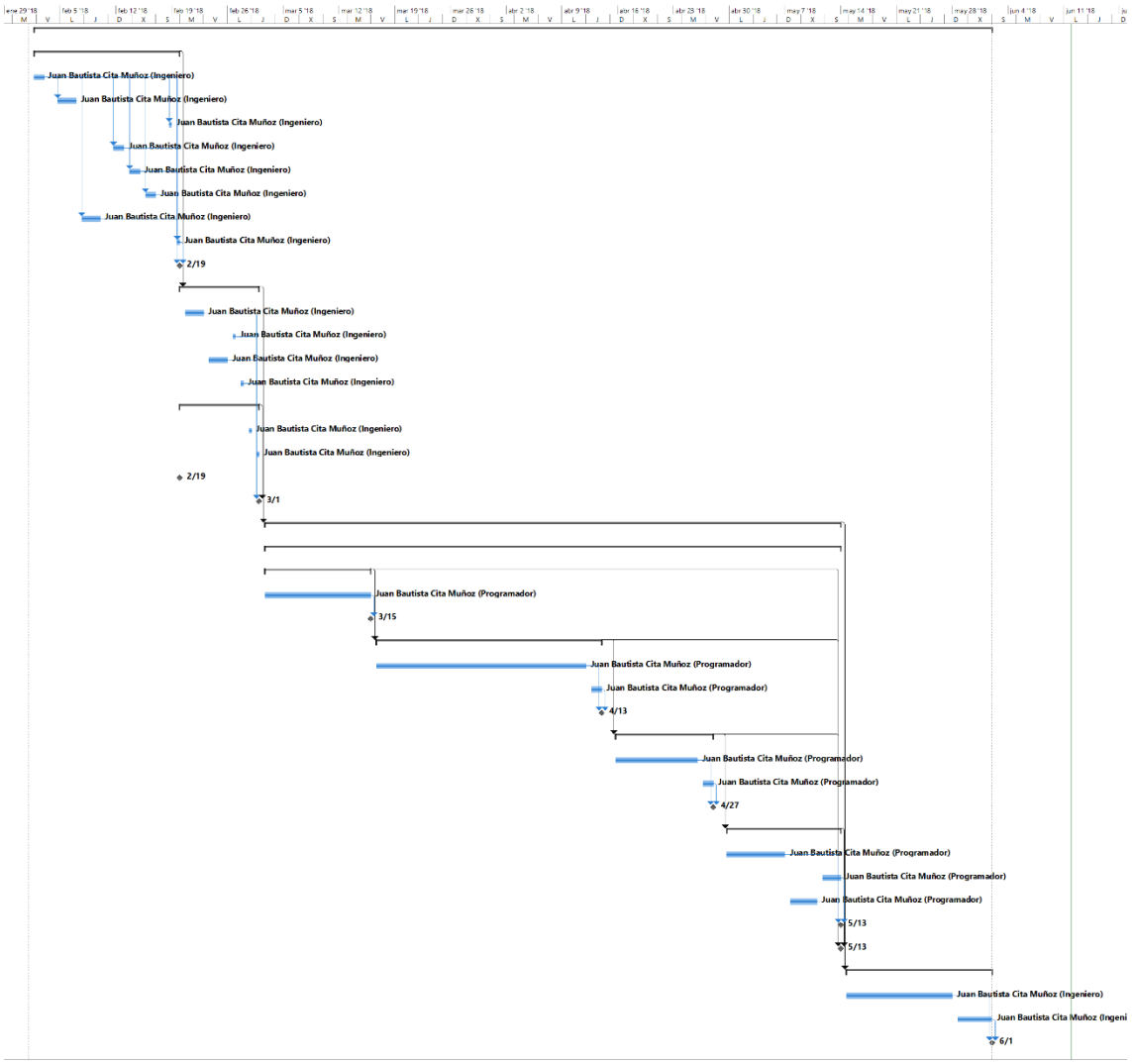
Fuentes:

- https://gist.github.com/gestadieu/25af5334a79d6c02d6413968a8bff572
- https://material.io/icons/
- https://www.youtube.com/watch?v=GeN7g4bdHiM

8.4 DIAGRAMA UML



8.5 DIAGRAMA GANT



Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá