

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de
Telecomunicación

Aplicación web con Freeboard para la recolección de
datos de sensores de acelerómetro y gps mediante
Orion Context Broker de Fiware

Autor: Luis Martínez Ruiz

Tutor: María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018



Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware

Autor:

Luis Martínez Ruiz

Tutor:

María Teresa Ariza Gómez

Profesor titular

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Aplicación web con Freeboard para la recolección de datos de sensores de acelerómetro y gps mediante Orion Context Broker de Fiware

Autor: Luis Martínez Ruiz

Tutor: María Tereza Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Antes de comenzar con mi Proyecto de Fin de Grado, el cual me va a permitir obtener mi título en el Grado de Ingeniería de las Tecnologías de Telecomunicación, he de agradecer a todas las personas que me han apoyado para conseguir este título.

En primer lugar, todo esto se lo debo a mi familia, priorizando en mis padres y mi hermano que son los que me han apoyado en todo momento y en todos los sentidos. Además, son las personas que principalmente han hecho que sea como soy hoy en día. También, debo agradecer a todos mis amigos que me han ayudado para que esto sea posible, tanto los que he conocido en la Universidad como a mis amigos de toda la vida.

En segundo lugar, quiero agradecer a todos los profesores todo lo que me han hecho aprender, ya que de todos y cada uno de ellos he aprendido algo, y seguro que me van a aportar cosas en mi futuro. También, merecen mi agradecimiento todos mis compañeros que me han ayudado en cada una de las asignaturas y con los que he pasado unos años fantásticos.

Por último, quiero nombrar algo que desde pequeño hizo que mi vida cambiara y que me llevara a donde estoy en la actualidad. Hace ya 15 años, comencé a practicar un deporte, Remo, el cual me ha aportado unos valores de los que debo estar muy agradecido y, aunque haya tenido que dedicar mucho tiempo al mismo y haya tenido que renunciar a muchas cosas, nada es comparable con lo que me ha dado hasta la actualidad. Entre una de esas cosas, ha permitido que haya venido a vivir a Sevilla y haya realizado esta carrera en la US.

Luis Martínez Ruiz

Sevilla, 2018

Resumen

En la actualidad el Internet de las Cosas está llevando a cabo una transformación digital, que está suponiendo un cambio total de la sociedad. Esto es debido a que se estamos instaurando esta tecnología en todos los objetos cotidianos del ser humano.

En este proyecto, hemos llevado a cabo la monitorización de datos de diversos sensores, que hoy en día están contenidos en la mayoría de dispositivos *wearables*¹. Esta monitorización la hemos llevado a cabo mediante una Raspberry Pi a la que hemos conectado los sensores. Una vez obtenidos los datos deseados, hemos trabajado con una nueva plataforma, desarrollada para el Internet del Futuro y que tiene una funcionalidad muy potente. Además, hemos realizado un cuadro de mandos con una plataforma web denominada Freeboard, que permite al usuario ver los datos obtenidos para usarlos como crea más preciso.

¹ Conjunto de aparatos y dispositivos electrónicos que se incorporan en alguna parte de nuestro cuerpo interactuando de forma continua con el usuario y con otros dispositivos con la finalidad de realizar alguna función concreta

Abstract

Nowadays the Internet of things is conducting a digital transformation, which is assuming a total change of the society. This is because you are instituting this technology in all the everyday objects of the human being.

In this project we have carried out the monitoring of data from various sensors, which nowadays are contained in most wearables devices. This monitoring has been carried out by a Raspberry Pi to which we have connected the sensors. Once obtained the desired data, we have worked with a new platform, developed for Internet of the Future and that has a very powerful experience. In addition, we have made a dashboard with an open web platform of Freeboard, which allows the user to work with the data obtained to use them as he believes more accurate.

Índice

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xiv
Índice de Tablas	xvii
Índice de Ilustraciones	xix
1 Introducción	1
1.1 <i>Motivación</i>	1
1.2 <i>Objetivos</i>	1
1.3 <i>Descripción de la solución</i>	2
1.3.1 <i>Obtención de datos del sensor MPU-6050</i>	2
1.3.2 <i>Obtención de datos del sensor NEO-6M-0-001</i>	3
1.3.3 <i>Uso de FIWARE Orion Context Broker</i>	3
1.3.4 <i>Generación de Dashboard mediante Freeboard</i>	3
1.4 <i>Estructura de la memoria</i>	3
2 Recursos Utilizados	7
2.1 <i>Recursos Hardware</i>	7
2.1.1 Ordenador Portátil SONY VAIO SVE151G17M	7
2.1.2 RaspBerry Pi 3 Model B	8
2.1.3 Sensor MPU-650	9
2.1.4 Sensor U-blox NEO-6M-0-001	9
2.2 Recursos Software	9
2.2.1 Docker	9
2.2.2 MongoDB	9
2.2.3 Restlet Client - REST API Testing	10

2.2.4	Freeboard	10
3	Arquitectura y análisis del sistema	13
3.1	<i>Arquitectura del sistema</i>	13
3.1.1	Lectura y envío de datos de los sensores	14
3.1.2	Cloud	15
3.1.3	Presentación de datos: Freeboard	16
4	Sensores	17
4.1	<i>Sensor MPU-6050</i>	17
4.1.1	Transmisión de datos I2C	18
4.2	<i>Sensor NEO-6M-0-001</i>	20
4.2.1	Transmisión de datos UART	22
5	Fiware	23
6.1	<i>Proyecto FIWARE</i>	23
6.2	<i>Data/Context Management</i>	25
6.3	<i>Publish/Subscribe Context Broker GE</i>	26
6.3.1	Actores básicos	26
6.3.2	Estándares empleados por el Context Broker	27
6.4	<i>Uso del Context Broker</i>	32
6.4.1	Creación de entidades	32
6.4.2	Actualización de entidades	34
6	Freeboard	37
6.1	<i>Configuración de Datasources</i>	38
7.2	<i>Configuración de Widgets</i>	40
7.3	<i>Presentación de los datos</i>	41
7	Ejecución y Resultados	43
7.1	<i>MPU-6050</i>	44
7.2	<i>NEO-6M-0-001</i>	45
8	Conclusiones y Líneas futuras	47
9.1	<i>Conclusiones</i>	47
9.2	<i>Líneas futuras</i>	47
Anexo A:		49
Manual de instalación de Orion Context Broker utilizando Docker		49
A.1	<i>Instalación de Docker</i>	49
A.2	<i>Instalación de Docker</i>	50
A.3	<i>Creación de aplicación Orion en Docker</i>	50
Anexo B:		57
Configuración y conexión de sensores con Raspberry Pi		57
B.1	<i>Sensor MPU-6050</i>	57
B.2	<i>Sensor U-blox NEO-6M-0-001</i>	60
Referencias		65

ÍNDICE DE TABLAS

Tabla 1: MPU6050 - Conexión	57
Tabla 2: NEO-6M-0-001 - Conexión	60

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Arquitectura del sistema	2
Ilustración 2: Arquitectura del sistema realizado	13
Ilustración 3: MPU-6050	17
Ilustración 4: MPU-6050 - Esquema I2C	19
Ilustración 5: MPU-6050 - Transferencias I2C	20
Ilustración 6: Ublox NEO-6M-0-001	21
Ilustración 7: NEO-6M-0-001 - Conexión UART	22
Ilustración 8: FIWARE - Arquitectura	25
Ilustración 9: FIWARE - Context Broker GE	27
Ilustración 10: FIWARE - Interacciones Básicas y Entidades relacionadas	28
Ilustración 11: FIWARE - Interacciones relacionadas con los proveedores de contexto	28
Ilustración 12: FIWARE - Interacciones relacionadas con las actualizaciones reenviadas a los proveedores de contexto	29
Ilustración 13: Interacciones para forzar a los consumidores de contexto a suscribirse a notificaciones específicas	29
Ilustración 14: FIWARE - Registro de disponibilidad de entidades y atributos	30
Ilustración 15: FIWARE - Suscripción de aplicaciones al registro de Entidades/Atributos	30
Ilustración 16: Entidad GyroscopeAccelerometer	32
Ilustración 17: Entidad GPS	33
Ilustración 18: Creación de entidad GyroscopeAccelerometer	34
Ilustración 19: Creación de entidad GPS	34
Ilustración 20: Actualización de entidad GyroscopeAccelerometer	35
Ilustración 21: Actualización de entidad GPS	35
Ilustración 22: Freeboard – Datasource Giroscopio/Acelerómetro	39
Ilustración 23: Freeboard – Datasource GPS	39
Ilustración 24: Freeboard – Widget Gauge	40
Ilustración 25: Freeboard – Widget Google Maps	41
Ilustración 26: Freeboard – Dashboard 1	42
Ilustración 27: Freeboard – Dashboard 2	42
Ilustración 28: Ejecución - Docker	43
Ilustración 29: Ejecución – Docker Arrancado	44
Ilustración 30: Ejecución – Resultado de medición con MPU-6050	44
Ilustración 31: Ejecución – Resultado Freeboard MPU-6050	45

Ilustración 32: Ejecución – Resultado de medición con NEO-6M-0-001	45
Ilustración 33: Ejecución – Resultado Freeboard NEO-6M-0-001	46
Ilustración 34: Docker - Instalación	50
Ilustración 35: Configuración OCB y MongoDB con Docker	51
Ilustración 36: Docker - Ejecución	52
Ilustración 37: Orion CB -Ejecución	52
Ilustración 38: Orion CB – Test de ejecución: Petición HTTP	53
Ilustración 39: Orion CB – Test de ejecución: Resultado de realizar petición HTTP	53
Ilustración 40: MPU6050 - Conexión Sensor MPU-6050 con Raspberry Pi	58
Ilustración 41: MPU6050 – Conexión Sensor MPU-6050 con Raspberry Pi (2)	58
Ilustración 42: MPU6050 - Configuración I2C-1	59
Ilustración 43: MPU6050 - Configuración I2C-2	59
Ilustración 44: MPU6050 - Test de funcionamiento	60
Ilustración 45: NEO-6M-0-001 - Conexión del sensor NEO-6M-0-001 con Raspberry Pi	61
Ilustración 46: NEO-6M-0-001 - Conexión del sensor NEO-6M-0-001 con Raspberry Pi (2)	61
Ilustración 47: NEO-6M-0-001 – Test de Funcionamiento	63

1 INTRODUCCIÓN

La única manera de hacer un gran trabajo, es amar lo que haces. Si no lo has encontrado, sigue buscando. No te conformes.

- Steve Jobs -

En este primer capítulo se van a comentar tanto los objetivos como la motivación por las que he realizado este proyecto y se comentará la solución que he llevado a cabo para el mismo. Por último, se va a indicar la estructura de esta memoria.

1.1 Motivación

Este proyecto se ha enfocado a trabajar con varias tecnologías que están teniendo un impacto muy importante en la sociedad y que están enfocadas en el Internet del Futuro. Para ello se han utilizado una gran cantidad de conceptos aprendidos en el grado.

Se ha indagado en una tecnología que está en auge en la actualidad, el llamado Internet de las Cosas (IoT). Además, cada día es más común el uso del Cloud Computing tanto por parte de empresas como por parte de usuarios, por ello hemos abordado también esta tecnología. Para afrontar un problema utilizando las dos tecnologías comentadas, hemos decidido trabajar con dos de los sensores más comunes en los dispositivos cotidianos del ser humano, sobretodo en los Smartphones.

Por tanto, el motivo principal de la realización de este proyecto ha sido poder trabajar con estas tecnologías que si no es en la actualidad, en un futuro muy cercano van a tener un impacto muy importante, a esto le debemos de sumar que son tecnologías las cuales me resultan tremendamente interesantes y que en un futuro me gustaría seguir trabajando con ellas.

1.2 Objetivos

El objetivo de este proyecto es en primer lugar la obtención de los datos correspondientes para los dos sensores con los que hemos trabajado, en el caso del Giroscopio/Acelerómetro (MPU-6050), obtendremos las coordenadas “X” / “Y”, y en el del GPS (NEO-6M-0-001) obtendremos la Longitud y la Latitud. Una

vez obtenidos estos datos, deberemos almacenarlos en la nube para poder consultarlos en tiempo real a través de una aplicación web (Freeboard).

1.3 Descripción de la solución

Arquitectura de la solución escogida:

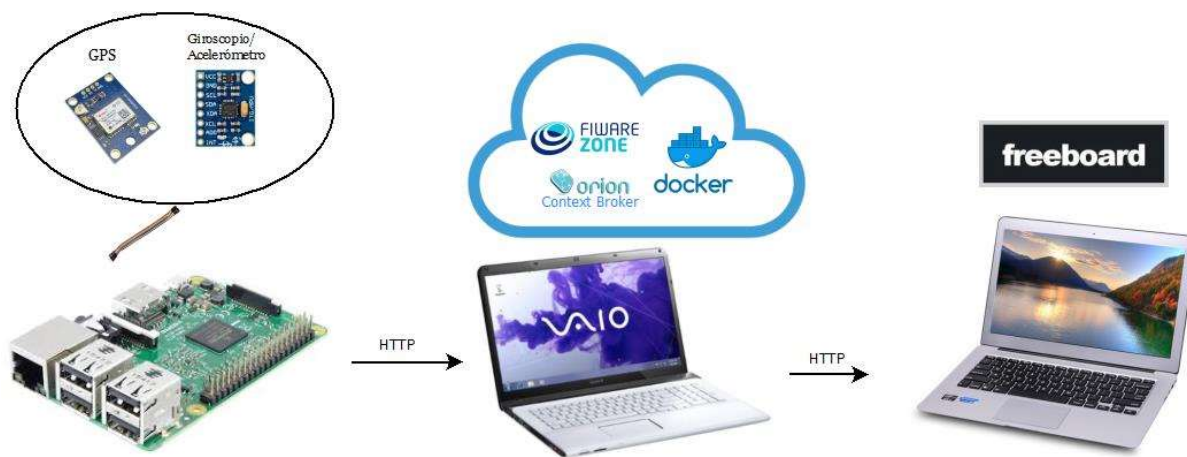


Ilustración 1: Arquitectura del sistema

Como podemos ver en la ilustración 1, nuestro sistema está formado por tres partes, las cuales se describen a continuación:

1. Mediante el uso de la Raspberry Pi, en nuestro caso el modelo 3B, la cual dispone tanto de pines I2C como UART que son los necesarios para trabajar con los dos sensores utilizados, hemos realizado la lectura de los datos mediante librerías Python.
2. La información obtenida será guardada en la nube. En nuestro caso para llevar a cabo este paso, hemos utilizado la plataforma FIWARE la cual nos proporciona una gran variedad de capacidades Clouds. En particular hemos trabajado con Orion Context Broker.

Para llevar a cabo la conexión con dicho componente, hemos utilizado el protocolo de comunicación HTTP, utilizando en este caso las peticiones PUT, para actualizar la información.

3. Por último, hemos trabajado con Freeboard, que es una plataforma Web, la cual actúa como Servicio Web, y permite generar dashboards con la información obtenida.

1.3.1 Obtención de datos del sensor MPU-6050

Para realizar las mediciones de movimiento, hemos trabajado con el sensor MPU-6050, el cual mediante la combinación de un MEMS, Sistemas Microelectromecánicos, giroscopio de 3 ejes y un MEMS acelerómetro de 3 ejes en la misma pastilla de silicio junto con un DMP™, Procesador de Movimiento Digital, es capaz de procesar los algoritmos de movimientos complejos de 9 ejes (MotionFusion™) en una placa. Posteriormente, hablaremos con más detalle de este sensor.

La obtención de dichas mediciones la hemos llevado a cabo mediante un programa desarrollado en Python, utilizando la librería “smbus”, la cual nos permite obtener datos a través del bus I2C. Mediante este programa, obtenemos los canales “X”, “Y” y “Z” (el cual es fijo) de forma periódica.

1.3.2 Obtención de datos del sensor NEO-6M-0-001

Otro de los sensores que hemos utilizado es “U-blox NEO-6M-0001”. Este sensor es un receptor GPS independiente, el cual posee un motor de procesamiento de alto rendimiento denominado “U-blox 6”. Como en el caso anterior, más adelante se detallarán las características del mismo.

Para llevar a cabo la recolección de las mediciones en la Raspberry, hemos tenido que instalar “GPSD”, que es una aplicación que corre en segundo plano y que recoge los datos GPS del módulo, y hemos realizado un programa desarrollado en Python, utilizando la librería “gps”. Los datos que obtenemos mediante dicho programa son la “Latitud” y “Longitud”, que son con los que trabajaremos posteriormente.

1.3.3 Uso de FIWARE Orion Context Broker

Hemos trabajado con Orion Context Broker, tanto para permitir la comunicación con los usuarios finales, los cuales desean trabajar con los datos monitorizados, como para almacenar dichos datos y hacerlos persistentes. Este componente lo hemos desplegado en docker, debido a que nos facilita mucho su uso, y trabaja con “MongoDB” para la persistencia de los datos.

El motivo por el cual hemos trabajado con este componente, es porque nos permite registrarnos a elementos del contexto y administrarlos mediante instrucciones de consulta y actualización.

1.3.4 Generación de Dashboard mediante Freeboard

Para la presentación de los datos, hemos trabajado con una plataforma libre denominada Freeboard, la cual nos permite la creación de dashboards, formados por widgets, los cuales muestran los datos en tiempo real. Esta plataforma además trabaja como Web Services, y permite la obtención de datos mediante peticiones a nuestro sistema de persistencia. Las peticiones que hemos definido para que trabaje el mismo son de tipo GET.

1.4 Estructura de la memoria

La memoria de este trabajo se va a organizar de la siguiente manera:

- **Capítulo 1: Introducción**
En este capítulo se encuentran tanto la motivación por la cual se ha elaborado este proyecto como el problema y la solución adoptada.
- **Capítulo 2: Recursos utilizados**
En este capítulo nos centramos en los recursos que hemos utilizado tanto a nivel de *software* como de *hardware*.
- **Capítulo 3: Arquitectura y análisis del sistema**
En este capítulo vamos a profundizar en el funcionamiento del sistema, así como en la iteración conjunta de las diversas partes que lo forman.

- **Capítulo 4: Sensores**
En este capítulo vamos a hablar con más detalle de las características y funcionalidad de los dos sensores utilizados.
- **Capítulos 5: FIWARE**
En este capítulo vamos a ahondar en esta plataforma y específicamente en el *Generic Enabler* que hemos utilizado en este proyecto.
- **Capítulo 6: Freeboard**
En este capítulo nos vamos a centrar en esta plataforma, de la cual vamos a hablar de su funcionamiento y en específico vamos a explicar el uso que le hemos dado para la realización de este proyecto.
- **Capítulo 7: Ejecución**
En este capítulo se va a mostrar un ejemplo de ejecución del sistema completo y se mostrarán los resultados.
- **Capítulo 8: Conclusiones**
En este capítulo se van a exponer las líneas para la continuación de este proyecto y las conclusiones a las que hemos llegado tras haber realizado este trabajo.

2 RECURSOS UTILIZADOS

Para lograr tener éxito, tu deseo de éxito debe ser mayor que tu miedo al fracaso.

- Bill Cosby -

Este capítulo se va a centrar en comentar los recursos que se han utilizado en el proyecto ya que es imprescindible trabajar con los recursos más adecuados para obtener un resultado correcto y que sea lo más eficiente posible.

2.1 Recursos Hardware

En este apartado vamos a hablar de los recursos hardware que se han utilizado en el proyecto.

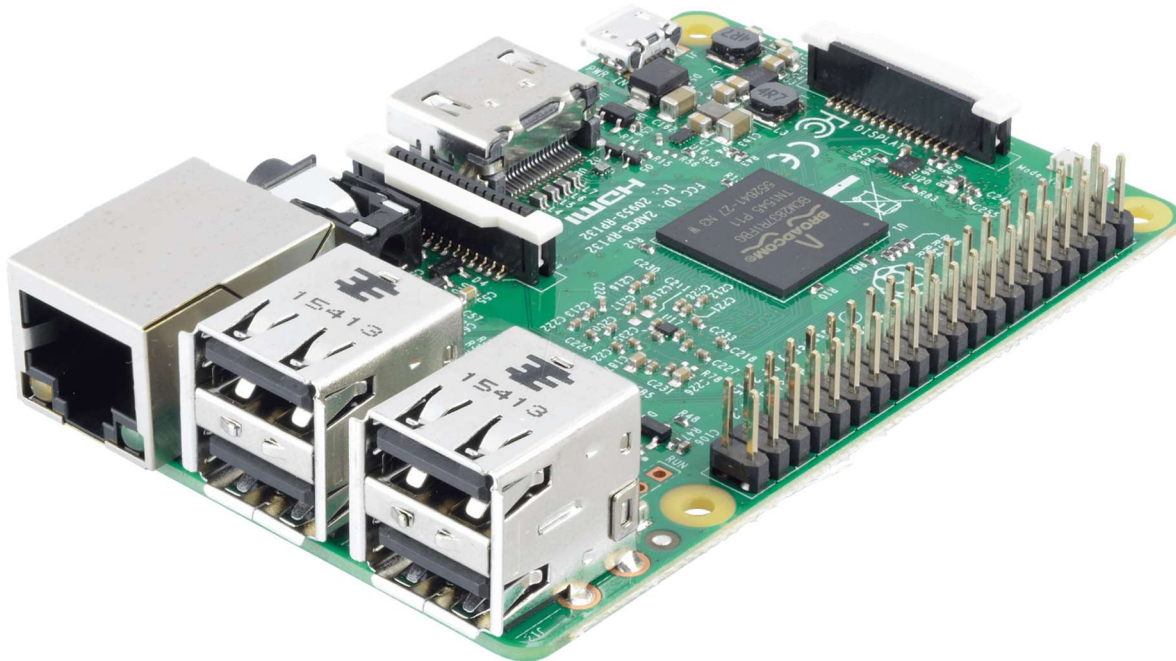
2.1.1 Ordenador Portátil SONY VAIO SVE151G17M



- Procesador: Intel® Core™ i7-3632QM CPU @ 2.20GHz × 8
- Memoria RAM: 5,8 GiB
- Tarjeta gráfica: AMD TURKS (DRM 2.50.0 / 4.15.0-32-generic, LLVM 6.0.0)

- Disco duro: 732,0 GB

2.1.2 RaspBerry Pi 3 Model B



- Procesador:
 - Chipset Broadcom BCM2387.
 - 1,2 GHz de cuatro núcleos ARM Cortex-A53
- GPU
 - Dual Core VideoCore IV ® Multimedia Co-procesador. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware, y 1080p30 H.264 de alto perfil de decodificación.
 - Capaz de 1 Gpixel / s, 1.5Gtexel / s o 24 GFLOPs con el filtrado de texturas y la infraestructura DMA.
- RAM: 1GB LPDDR2.
- Conectividad
 - Ethernet socket Ethernet 10/100 BaseT
 - 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE)
 - Salida de vídeo
 - HDMI rev 1.3 y 1.4
 - RCA compuesto (PAL y NTSC)
 - Salida de audio
 - jack de 3,5 mm de salida de audio, HDMI
 - USB 4 x Conector USB 2.0
 - Conector GPIO
 - 40-clavijas de 2,54 mm (100 milésimas de pulgada) de expansión: 2x20 tira
 - Proporcionar 27 pines GPIO, así como 3,3 V, +5 V y GND líneas de suministro
 - Conector de la cámara de 15 pines cámara MIPI interfaz en serie (CSI-2)
 - Pantalla de visualización Conector de la interfaz de serie (DSI) Conector de 15 vías plana flex cable con dos carriles de datos y un carril de reloj
 - Ranura de tarjeta de memoria Empuje / tire Micro SDIO

2.1.3 Sensor MPU-650

Este sensor es un Giroscopio y Acelerómetro, cuyas características se detallarán en el capítulo 4.

2.1.4 Sensor U-blox NEO-6M-0-001

Este sensor es un GPS, cuyas características se detallarán en el capítulo 4.

2.2 Recursos Software

En este apartado vamos a hablar de los recursos software que se han utilizado en el proyecto.

2.2.1 Docker



Docker [1] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombre para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

El soporte del kernel Linux para los espacios de nombres aísla la vista que tiene una aplicación de su entorno operativo, incluyendo árboles de proceso, red, ID de usuario y sistemas de archivos montados, mientras que los cgroups del kernel proporcionan aislamiento de recursos, incluyendo la CPU, la memoria, el bloque de E/S y de la red. Desde la versión 0.9, Docker incluye la biblioteca *libcontainer* como su propia manera de utilizar directamente las facilidades de virtualización que ofrece el kernel Linux, además de utilizar las interfaces abstraídas de virtualización mediante *libvirt*, *LXC (Linux Containers)* y *systemd-nspawn*.

2.2.2 MongoDB



MongoDB [15] es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En lugar de guardar los datos en tablas como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (MongoDB utiliza una especificación llamada BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

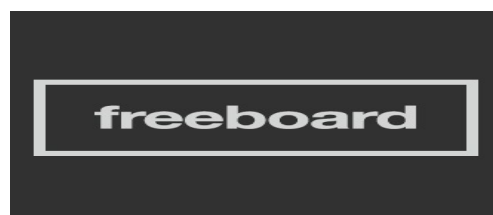
2.2.3 Restlet Client - REST API Testing



Restlet Client [16] es una extensión de Google Chrome la cual nos posibilita realizar peticiones a APIs o servicios RestFul. Puedes hacer cualquier tipo de petición además de las habituales GET, POST, PUT y DELETE. Permite pasar parámetros a las peticiones y muestra el resultado devuelto por el servicio que queremos probar.

Hemos utilizado esta extensión en nuestro proyecto para comprobar los datos que se encontraban en el Context Broker.

2.2.4 Freeboard



Freeboard [17] es una plataforma OpenSource que nos permite crear dashboards interactivos y visualizarlos en tiempo real mediante una sencilla interfaz. Esta plataforma será detallada en el capítulo 6.

3 ARQUITECTURA Y ANÁLISIS DEL SISTEMA

*Estoy agradecido por todos los que me dijeron NO.
Es gracias a ellos estoy siendo yo mismo.*

- Albert Einstein -

En este capítulo se va a comentar la arquitectura de nuestro sistema. Explicando el conjunto de sistemas que forman el mismo, y el diseño que hemos realizado para conseguir el mejor diseño posible.

3.1 Arquitectura del sistema

En la ilustración 2 podemos ver la arquitectura diseñada para el sistema:

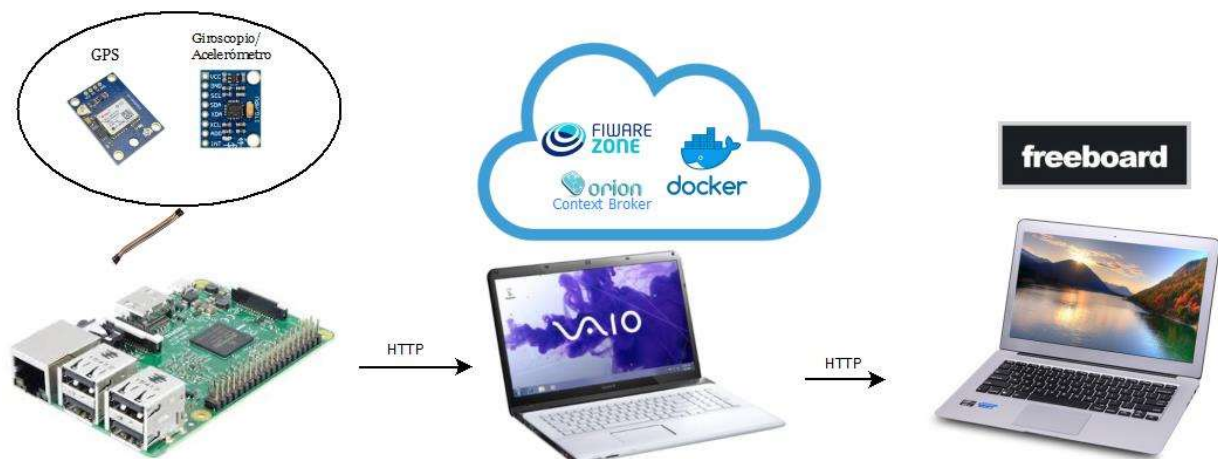


Ilustración 2: Arquitectura del sistema realizado

Como podemos observar en la ilustración 2, nuestro sistema está compuesto por 3 entidades fundamentales:

1. La primera de las entidades se trata de una Raspberry Pi, en donde se conectan los dos sensores con los que trabajamos y se ejecutan los respectivos programas que hemos desarrollado para enviar la información al Context Broker.

2. En segundo lugar, se encuentra la nube. En este sistema hemos utilizado la plataforma FiWare, en concreto trabajando con Orion Context Broker y una base de datos MongoDB. Este componente lo hemos desplegado en Docker.
3. La tercera entidad, es Freeboard. Esta plataforma nos permite visualizar los datos obtenidos a través de un navegador web.

3.1.1 Lectura y envío de datos de los sensores

Para llevar a cabo la lectura de los datos proporcionados por ambos sensores, hemos trabajado con la tecnología Python, porque gracias a sus librerías, es la que más nos ha facilitado el trabajo.

Para explicar mejor el funcionamiento, vamos a dividir este apartado en dos subapartados, cada uno de los cuales se centrará en un sensor distinto. Dichos apartados son los siguientes:

3.1.1.1 Programa: Sensor MPU-6050

En primer lugar, vamos a explicar cómo se ha llevado a cabo la conexión de dicho sensor a la Raspberry Pi. Dicha conexión se ha realizado mediante el bus de datos I2C, cuyos pines son “I2C1 SDA” e “I2C1 SCL” que corresponden con los pines 3 y 4 respectivamente de nuestra Raspberry Pi. Para trabajar con dicho bus de datos, lo hemos tenido que habilitar de forma correcta en nuestra Raspberry Pi usando *raspi-config* y además hemos instalado dos herramientas necesarias para trabajar con dicho bus en python, estas herramientas son “i2c-tools” y “python-smbus”.

Una vez hemos configurado correctamente nuestra Raspberry Pi, se ha llevado a cabo la conexión mediante un panel de conexión y 4 cables tipo hembra-macho. Dos de estos pines se han utilizado para conectarnos al bus de datos I2C y los otros dos se han destinado para alimentar el circuito y para llevar a cabo la conexión a tierra.

La configuración de la Raspberry Pi y la conexión del sensor con la misma, se detalla en el *Anexo B*.

Tras haber finalizado la conexión, vamos a profundizar en el desarrollo del programa para la realización de las mediciones [11]. Este programa trabaja principalmente con la librería “smbus” para la obtención de los datos de los buses y con la librería “httplib” que se utiliza para llevar a cabo las peticiones al Context Broker. Una vez se ejecuta, se llevan a cabo los siguientes pasos:

1. Configuración de las variables para realizar la conexión con Orion Context Broker.
2. Conexión con los buses de datos para la obtención de las mediciones buscadas.
3. Realiza la conexión con Orion Context Broker y crea la entidad indicada, teniendo en cuenta que si dicha entidad ya existe, no realiza ninguna acción.
4. Lectura de los datos que nos proporciona el sensor y envío a Orion Context Broker. Para la actualización de la entidad realizamos una petición tipo PUT mediante las funciones que nos proporciona la librería “httplib”, citada anteriormente.
5. Tras realizar una medición, se queda dormido durante un determinado tiempo hasta realizar la siguiente.

En las mediciones realizadas con este programa, los datos que vamos a obtener y con los que

posteriormente vamos a trabajar, son los correspondientes a las coordenadas “X”, “Y” y “Z”, este último se trata de un valor fijo.

3.1.1.2 Programa: Sensor U-blox NEO-6M-0-001

Como en el caso anterior, lo primero va a ser explicar cómo se ha llevado a cabo la conexión de dicho sensor a la Raspberry Pi. En este caso la conexión se va a llevar a cabo a través del puerto serie de la Raspberry Pi, cuyos pines son los denominados “TXD0” y “RXD0”, que corresponden con el pin 8 y 10 respectivamente. Para trabajar con el puerto citado, debemos de configurar debidamente nuestra Raspberry Pi e instalar la aplicación GPSD, la cual corre en segundo plano y sirve para recoger los datos del módulo.

Tras configurar nuestra Raspberry Pi, hemos conectado la misma con el sensor mediante un panel de conexión y 4 cables tipo hembra-macho. Dos de estos pines se han utilizado para conectarnos al puerto serie y los otros dos se han destinado para alimentar el circuito y para llevar a cabo la conexión a tierra.

La configuración y conexión se detalla en el *Anexo B*.

Una vez conectado y configurado correctamente, nos centramos en el programa desarrollado para realizar las mediciones [9]. Este programa trabaja principalmente con la librería “gps” para la obtención de los datos y con la librería “httplib” que se utiliza para llevar a cabo las peticiones al Context Broker. Una vez se ejecuta, se llevan a cabo los siguientes pasos:

1. Configuración de las variables para realizar la conexión con Orion Context Broker.
2. Conexión con el puerto serie para la obtención de las mediciones buscadas.
3. Realiza la conexión con Orion Context Broker y crea la entidad indicada, teniendo en cuenta que, si dicha entidad ya existe, no realiza ninguna acción.
4. Lectura de los datos que nos proporciona el sensor y envío a Orion Context Broker. Para la actualización de la entidad realizamos una petición tipo PUT mediante las funciones que nos proporciona la librería “httplib”, citada anteriormente.
5. Tras realizar una medición, se queda dormido durante un determinado tiempo hasta realizar la siguiente.

En las mediciones realizadas con este programa, los datos que vamos a obtener y con los que posteriormente vamos a trabajar, son los correspondientes a la “longitud” y “latitud”.

3.1.2 Cloud

Debido a que en nuestro sistema vamos a trabajar con dos sensores distintos, debemos almacenar sus respectivos datos de forma idónea. Por ello, vamos a crear en la nube dos tipos de entidades distintas que puedan manejarse de forma segura, sin posibilidad de confusión. Dichos tipos se denominan *GPS* y *GyroscopeAccelerometer*, cuyo nombre identifica claramente el sensor que lo va a utilizar.

Todo el tema de configuración y creación de entidades, así como la forma con la que se ha llevado a cabo el despliegue del Context Broker, se va a explicar posteriormente en el capítulo 5. Este capítulo se va a centrar en FIWARE, de tal forma que va a introducirnos en dicha plataforma y va a detallar todo lo que hemos realizado para trabajar con el mismo.

3.1.3 Presentación de datos: Freeboard

La presentación de los datos obtenidos en las mediciones realizadas con los programas explicados en los apartados anteriores, se ha realizado mediante un *Dashboard*. Nuestro *Dashboard* está formado por una serie de Widgets, en los cuales se presentan los valores obtenidos de la forma más gráfica y entendible posible para el usuario.

En función de los valores obtenidos de cada sensor, hemos trabajado con distintos tipos de Widgets, los cuales son los siguientes:

1. Sensor MPU-6050:
 - Widget de tipo: *Gauge*.

2. Sensor U-blox NEO-6M-0-001:
 - Widget de tipo: *Google Maps*.

Como se ha comentado anteriormente, este dashboard lo hemos generado a través de la plataforma *OpenSource Freeboard* [17]. En el capítulo 7 vamos a hacer una introducción a esta plataforma y vamos a describir los pasos realizados para configurar el Dashboard para conseguir que se presenten los datos obtenidos.

4 SENSORES

El logro de un objetivo debe ser el punto de partida de otro.

- Alexander Graham Bell -

En este capítulo nos vamos a centrar en los dos sensores utilizados para el proyecto. Vamos a hacer un estudio de dichos sensores, detallando tanto sus características, funcionalidades, así como la forma en que se conectan y comunican en este caso con la Raspberry Pi.

4.1 Sensor MPU-6050

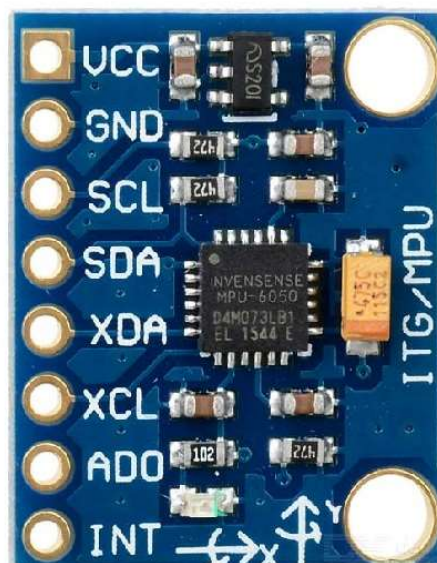


Ilustración 3: MPU-6050

En este apartado nos vamos a centrar en el *Sensor MPU-6050*.

En primer lugar, vamos a citar la introducción que proporciona el fabricante [18] [19]. Esta introducción es la siguiente:

La unidad de procesamiento de movimiento MPU-60X0 es la primera solución del mundo en procesamiento de movimientos, con la fusión integrada de sensores con 9-ejes, la fusión de sensores utiliza su motor propiedad MotionFusion™ probado en el campo de teléfonos móviles, tabletas, aplicaciones, controladores de juegos, mandos a distancia, puntero de movimiento y otros dispositivos de consumo.

El MPU-60X0 tiene integrado un giroscopio MEMS de 3 ejes, un acelerómetro de 3 ejes MEMS, y un procesador digital de movimiento (DMP™) motor acelerador de hardware con un puerto I2C auxiliar que se conecta a las interfaces de sensores digitales de terceras partes tales como magnetómetros. Cuando se conecta a un magnetómetro de 3 ejes, el MPU-60X0 entrega una salida completa de 9 ejes MotionFusion para su primario I2C o puerto SPI (SPI está disponible sólo en MPU-6000)™.

El MPU-60X0 combina la aceleración y el movimiento de rotación más la información de rumbo en un único flujo de datos para la aplicación. Esta integración de la tecnología MotionProcessing™ presenta un diseño más compacto y tiene ventajas de costos inherentes en comparación con soluciones discretas de giroscopio más acelerómetro. El MPU-60X0 también está diseñado para interactuar con múltiples sensores digitales no inerciales, tales como sensores de presión, en su bus I2C auxiliar maestro. El MPU-60X0 es un procesador de movimiento de segunda generación y es la huella compatible con la familia MPU-30X0.

Por lo tanto, este sensor tiene como finalidad el procesamiento de movimiento. Para conseguir dicho propósito, el sensor esta compuesto por un acelerómetro de 3 ejes y un giroscopio de 3 ejes.

El MPU-6050 elimina los problemas de alineación del eje transversal que puede arrastrarse hacia arriba en porciones discretas. Las piezas integran el algoritmo *MotionFusion*², Fusión de Datos, para 9 ejes que pueden incluso acceder a magnetómetros externos u otros sensores a través de un bus I2C auxiliar maestro, permitiendo reunir un conjunto completo de dispositivos sensores de datos, sin la intervención del procesador del sistema. El MPU-6050 es un 6 DOF, Grados de Libertad, o un sensor IMU, Unidad de Medición Internacional, de seis ejes, lo que significa que da seis valores de salida.

El procesador digital de movimiento (DMP) incorporado se encuentra dentro de la MPU-6050 y descarga el cálculo de los algoritmos de procesamiento de movimiento desde el procesador host. Los datos resultantes pueden ser leídos de los registros de la DMP, o pueden estar tamponados en un *FIFO*³. El DMP tiene acceso a uno de los pines externos de la MPU, Unidad de Procesamiento Múltiple, que pueden ser utilizados para la generación de interrupciones. El propósito del DMP es descargar los requisitos de temporización y la potencia de procesamiento del procesador anfitrión.

El sensor MPU-6050 es muy preciso, ya que contiene una conversión hardware de 16 bits de A/D por cada canal, para la digitalización de las salidas del acelerómetro. Para ello capta los canales x, y y z al mismo tiempo. Como se ha comentado, el sensor utiliza el I2C-bus para realizar la interconexión con Raspberry Pi.

Para completar esta información habría que ver el datasheet del fabricante [19].

4.1.1 Transmisión de datos I2C

Como se ha comentado anteriormente, la transmisión de los datos que se va a realizar con la Raspberry

² Uso sinérgico de la información proveniente de diferentes sensores para lograr una tarea requerida por el sistema.

³ Primero en entrar, primero en salir» es un concepto utilizado en estructuras de datos, contabilidad de costes y teoría de colas.

Pi, va a ser a través del puerto I2C. Por ello, en este apartado vamos a comentar las características principales de este bus de datos.

I2C [13] viene del inglés *Inter-Integrated Circuit*, o lo que es lo mismo Circuito Inter-Integrado, es un bus serie de datos que fue desarrollado por Philips Semiconductors (hoy en día NXP Semiconductors). Su principal utilidad es para la comunicación entre diferentes partes de un circuito.

Este bus presenta una topología maestro-esclavo. En nuestro proyecto el maestro es la Raspberry Pi y el esclavo es el sensor MPU-6050. A continuación, se muestra un esquema que representa la topología citada:

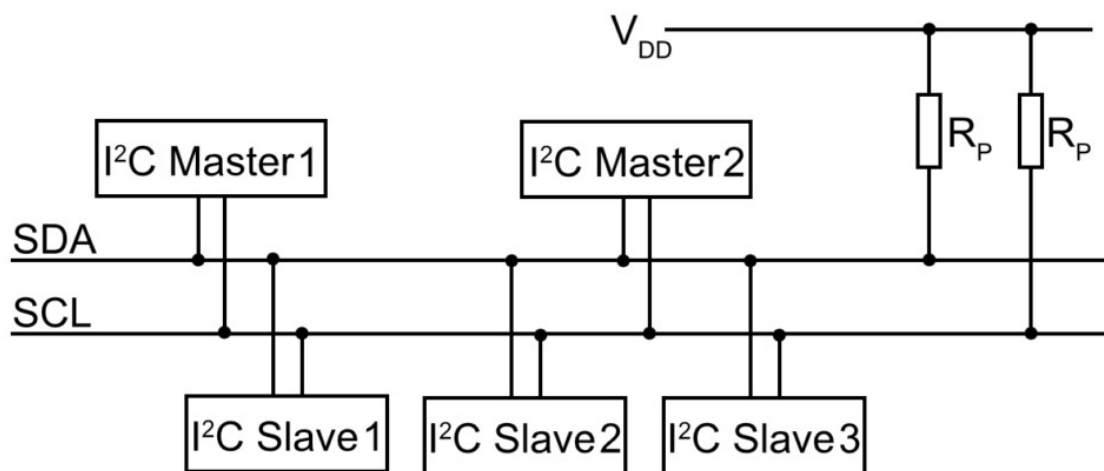


Ilustración 4: MPU-6050 - Esquema I2C

Apoyándonos en el esquema de la ilustración 4, vamos a explicar cómo funciona dicho bus. La transferencia se realiza mediante *SDA* y, por otro lado, *SCL* sincroniza los datos de *SDA* mediante una señal de reloj. Por lo tanto, especificando el funcionamiento en nuestro proyecto, la Raspberry Pi (maestro) lleva la iniciativa de la comunicación, por lo que genera dicha señal de reloj y controla cuándo se están enviando o recibiendo datos. Esto no significa que solo pueda transferir datos el maestro, ambos lo pueden hacer, pero los esclavos nunca pueden iniciar la transferencia.

Además, se puede observar en la ilustración 4 que pueden conectarse más de un esclavo a la vez, aunque debemos de aclarar que el maestro sólo se puede comunicar con uno en el mismo instante de tiempo. Para ello, cada dispositivo tiene que estar identificado con una dirección única dentro del bus (7 o 10 bits) y se comunica con el maestro mediante el uso de un protocolo de “pregunta/respuesta”.

La organización de los datos de I2C se denomina Transferencias y se pueden observar en la ilustración 5.

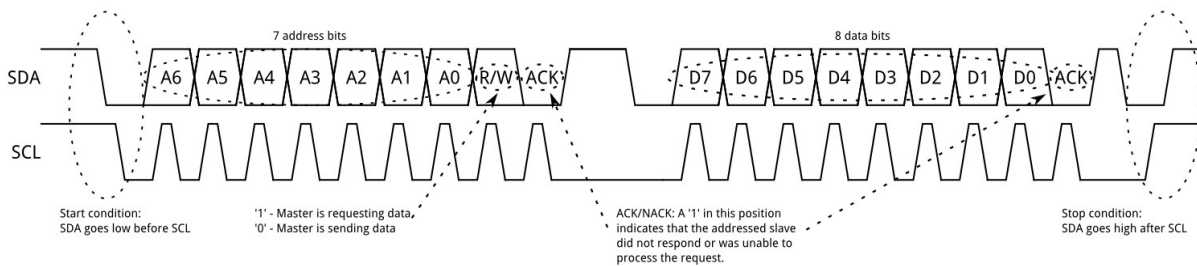


Ilustración 5: MPU-6050 - Transferencias I2C

Como podemos observar, cada transferencia comienza con una señal llamada “START” (el bus pasa a considerarse ocupado) y termina con otra llamada “STOP” (el bus pasa a considerarse libre). La longitud de los datos es de **1 byte** y terminan con el bit **ACK/NACK**.

En cuanto a los modos de operación son 4 y varían en función de la velocidad de transmisión de los datos que atraviesan SDA, la cual está marcada por el reloj. Como hemos citado, el estándar I2C soporta 4 modos de operación y son los siguientes:

- Standard Mode, con una velocidad de hasta 100 kbit/s.
- Fast mode, con una velocidad de hasta 400 kbit/s.
- Fast mode plus, con una velocidad de hasta 1 Mbit/s.
- High-speed mode, con una velocidad de hasta 3.4 Mbit/s.

Además, vamos a citar otro Bus con el cual hemos trabajado. Este bus se denomina SMBus y proviene del inglés *System Management Bus* y es un bus de transmisión en dos direcciones de forma simultánea (two-wire), el cual deriva de I2C.

Por último, quedaría explicar cómo se ha llevado a cabo la conexión y configuración de nuestros dispositivos para permitir que se comuniquen y con ello la obtención de los datos. Esto se detalla en un el *Anexo B* más adelante.

4.2 Sensor NEO-6M-0-001



Ilustración 6: Ublox NEO-6M-001

En este apartado nos vamos a centrar en el *Sensor U-blox NEO-6M-0-001*.

En primer lugar, vamos a citar la introducción que proporciona el fabricante [20]. Esta introducción es la siguiente:

La serie de módulos NEO-6 es una familia de receptores GPS independientes que poseen un motor de posicionamiento de alto rendimiento denominado u-blox 6. Estos receptores flexibles y rentables ofrecen numerosas opciones de conectividad en un paquete en miniatura de 16 x 12.2 x 2.4mm. Su arquitectura compacta y las opciones de potencia y memoria, hacen que los módulos NEO-6 sean ideales para dispositivos móviles que funcionen con un coste muy estricto de batería y limitaciones de espacio.

El motor de posicionamiento u-blox 6 de 50 canales cuenta con una corrección Time-To-First-Fix (TFFF) de menos de 1 segundo. El motor de adquisición dedicado, con 2 millones de correladores, es capaz de realizar búsquedas masivas de espacio/tiempo en paralelo, permitiéndole encontrar satélites al instante. El diseño y la tecnología innovadora suprimen las fuentes de interferencia y mitigan los efectos de multitrayectoria, lo que otorga a los receptores GPS NEO-6M un excelente rendimiento de navegación incluso en los entornos más desafiantes.

Por lo tanto, la finalidad de este sensor es obtener el posicionamiento, para ello trabaja con un potente motor, el cual se ha fabricado con un bajo consumo de energía y permite localizar los satélites de forma inmediata.

La transmisión de datos mediante la Raspberry Pi, se ha realizado a través del puerto serial, en el modelo 3B hemos utilizado el ttyS0. Y de él vamos a hablar a continuación.

Para completar esta información habría que ver el datasheet del fabricante [20].

4.2.1 Transmisión de datos UART

UART [10], viene del inglés *Universal Asynchronous Receiver-Transmitter* o lo que es lo mismo Transmisor-Receptor Asíncrono Universal. Su finalidad es controlar los puertos y los dispositivos serie. Este dispositivo lo podemos encontrar integrado en la placa base o en la tarjeta adaptadora del dispositivo.

Los puertos UART están formados principalmente por tres pistas que son las denominadas TX, RX y GND. Para realizar la conexión, el pin TX se conecta con el pin RX del otro dispositivo, el pin RX con el TX del otro dispositivo y el pin GND con el recíproco. En el caso de que solo se conectara un pin TX a un RX, la comunicación existiría de forma unidireccional.

Como advertencia, debemos comentar que el bus UART permite la comunicación de dos o más elementos, pero no es el más potente, por lo que es recomendable abrir distintos puertos para cada uno de los dispositivos con los que se quiera comunicar.

UART trabaja con niveles lógicos TTL (transistor-transistor logic) cuya finalidad es permitir que dos micros se comuniquen sin tener que convertir los voltajes a través de transeptores/drivers.

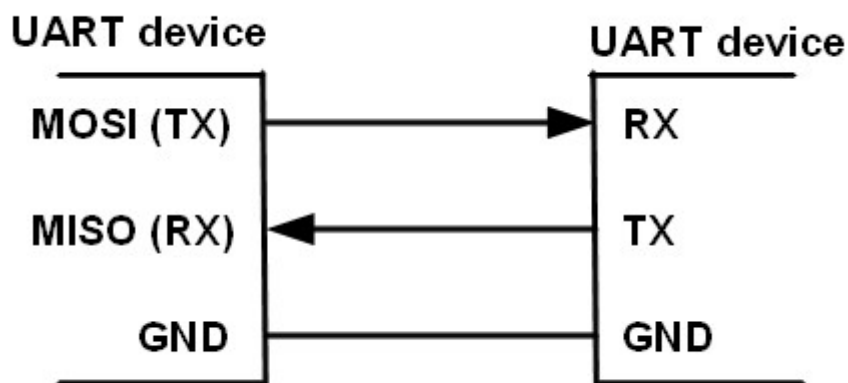


Ilustración 7: NEO-6M-0-001 - Conexión UART

Si hablamos de transmisión de datos, UART trabaja tomando datos y transmitiendo estos de forma secuencial. En el destino, un segundo UART reensambla los bits en bytes completos. Además, UART soporta velocidades de transmisión de hasta los 921.6Kbps.

Por último, quedaría explicar cómo se ha llevado a cabo la conexión y configuración de nuestros dispositivos para permitir que se comuniquen y con ello la obtención de los datos. Esto se detalla en el *Anexo B* más adelante.

5 FIWARE

La vida es una aventura atrevida o no es nada.

- Helen Keller -

En el siguiente capítulo nos vamos a centrar en FIWARE [4] [5] [14]. Vamos a hacer una extensa introducción sobre esta tecnología y también vamos a comentar cómo hemos conseguido la comunicación a través de ella de las distintas partes del proyecto.

6.1 Proyecto FIWARE

FIWARE es un proyecto europeo, el cual fue financiado por el VII Programa Marco de la Unión Europea dentro de su proyecto de colaboración público privada para el Internet del Futuro (FI-PPP - Future Internet Public-Private Partnership). El principal objetivo es mejorar la competitividad de Europa y sus principales metas se enfocan en el 2020.

Para entender el propósito de FIWARE, hay que tener en cuenta dos aspectos fundamentales. El primero de los aspectos es el denominado desarrollo tecnológico, que está contribuyendo a la creación de un nuevo contexto en donde las aplicaciones a través de la nube y el Internet de las cosas (IoT), son capaces de darnos los datos en tiempo real. Debido a esto, las empresas necesitan actualizarse y adaptarse a este nuevo entorno para poder ser competitivas, problema que se pretende solventar con FIWARE.

En cuanto al segundo aspecto, hoy en día sabemos que el futuro de Internet va a tener una relación muy estrecha con el compartir datos. Un entorno colaborativo donde las aplicaciones son desarrollados por terceros e integradas en plataformas donde todo va enfocado a establecer un futuro donde las empresas son más eficientes, más competitivas y se les presta un mejor servicio tanto a ciudadanos en general como a los consumidores.

Por tanto, FIWARE se desarrolla en torno a estos dos aspectos. Su diseño está enfocado a ayudar a las organizaciones y empresas para llevar a cabo esta transición, lo que les va a permitir adaptarse al entorno actual.

Dentro del contexto del Open Data, IoT y el cloud computing, FIWARE es una plataforma de *opensource* que permite la integración de apps y soluciones desarrolladas por terceros a través de un proceso muy

sencillo, totalmente seguro y de bajo coste. Todo ello es posible gracias al desarrollo de API's con tecnologías *opensource*.

Para lograr el objetivo comentado anteriormente, esta plataforma ofrece un conjunto de especificaciones y una arquitectura abierta que permiten el desarrollo y la distribución de las aplicaciones y los servicios digitales. Los recursos más relevantes son los siguientes:

- **FIWARE Lab:** se trata de un espacio de experimentación para los desarrolladores y empresarios del sector de las tecnologías de la información para crear aplicaciones y proyectos los cuales pueden ser testados con datos reales.
- **FIWARE Academy:** ofrece cursos, tutoriales y lecciones para ponerse en contacto con las tecnologías desarrolladas por FIWARE.
- **FIWARE Operations:** permite el desarrollo de nuevas plataformas FIWARE y datacenters, que se pueden expandir por el mundo. Esto hace que FIWARE sea una tecnología integrable.

En cuanto a la arquitectura de FIWARE, está constituida por el siguiente conjunto de categorías:

- **Applications and Services Ecosystem and Delivery Framework:** Este capítulo permite crear un ecosistema de aplicaciones y servicios sostenible, que les permiten ser accesibles por los usuarios finales desde cualquier dispositivo. Apps Generic Enablers admite la administración de servicios en un marco empresarial en todo el ciclo de vida del servicio, desde la creación y composición de servicios hasta la monetización y el reparto de ingresos.
- **Cloud Hosting:** Este capítulo ofrece GE que conforman la base para el diseño de una infraestructura de alojamiento moderna en la nube, la cual se puede utilizar para desarrollar, implementar y administrar aplicaciones y servicios para el Internet del Futuro.
- **Data/Context Management:** Este capítulo tiene como objetivo proporcionar GEs superiores y similares a plataformas que faciliten el desarrollo y el aprovisionamiento de aplicaciones innovadoras que requieren gestión, procesamiento y explotación de información de contexto, así como flujos de datos en tiempo real y escala masiva.
- **Advanced middleware and interfaces to Network and Devices (I2ND):** Este capítulo tiene como objetivo proporcionar GEs para ejecutar una infraestructura de red abierta y estandarizada. La infraestructura tendrá que lidiar con terminales altamente sofisticados, así como con proxies altamente sofisticados, por un lado, y con la infraestructura del operador de red en el otro lado.
- **Internet of Things (IoT) Services Enablement:** Este capítulo tiene como objetivo proporcionar GEs para que las cosas se conviertan en ciudadanos de Internet, los cuales estarán disponibles, buscables, accesibles y utilizables dentro de Internet, lo cual permitirá a las aplicaciones de FIWARE la interacción con objetos de la vida real.
- **Security:** Este capítulo tiene como objetivo proporcionar GEs que permitan la identificación, autenticación y autorización. La seguridad abarca desde la infraestructura hasta la capa de aplicación.
- **Advanced Web-based User Interface:** El objetivo de la interfaz de usuario avanzada basada en web (también conocida como WebUI, anteriormente MiWi) es mejorar significativamente la experiencia del usuario para el Internet del Futuro al agregar nuevas capacidades de interacción y entradas de usuario, como gráficos 3D interactivos, interacción inmersiva con el mundo real y virtual (Realidad Aumentada), virtualizando y, por lo tanto, separando la pantalla del dispositivo informático (móvil) para operaciones ubicuas, y muchas más.

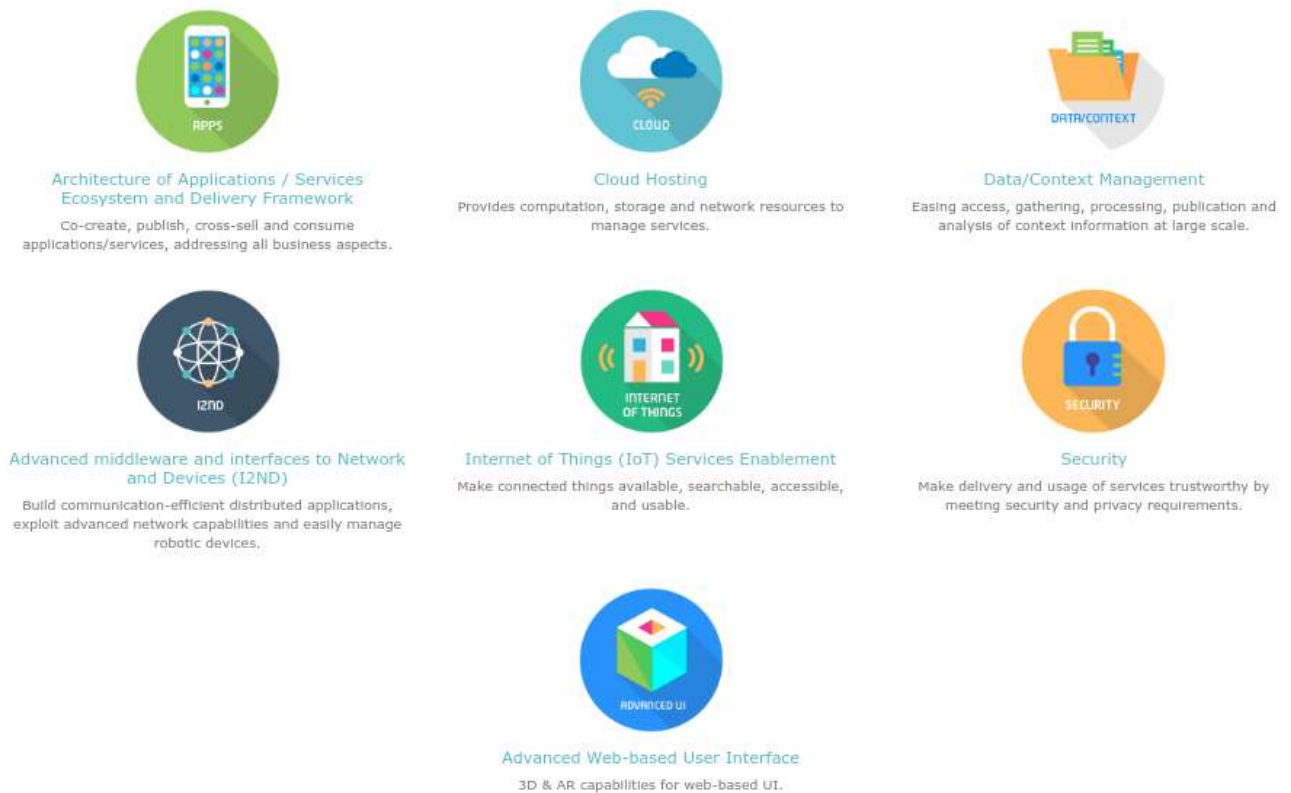


Ilustración 8: FIWARE - Arquitectura

6.2 Data/Context Management

Una de las principales funcionalidades de FIWARE es que permite realizar aplicaciones y servicios inteligentes gracias a los GEs que son capaces de recopilar, publicar, intercambiar, procesar y analizar datos masivos de forma rápida y eficiente.

Los GEs definidos pueden crear instancias de forma flexible y coherente, permitiendo que distintas instancias de FI-WARE seleccionen y configuren diferentes GEs de acuerdo con las demandas y requisitos de las aplicaciones.

Los GEs que componen el módulo se describen a continuación:

- **Publish/Subscribe Context Broker GE**, que permite a las aplicaciones intercambiar eventos heterogéneos siguiendo el patrón estándar publicador/suscriptor.
- **Complex Event Processing GE**, que tiene que ver con el procesamiento de flujos de eventos en tiempo real que generará una visión inmediata, permitiendo a las aplicaciones responder instantáneamente a condiciones cambiantes en ciertos clientes u objetos (entidades tales como dispositivos, aplicaciones, sistemas, etc.)
- **BigData Analysis GE**, que permite realizar un análisis *Map-Reduce* de una gran cantidad de datos.

- **Multimedia Analysis Generation GE**, que realiza la extracción automática o semiautomática de metainformación basada en el análisis de contenido multimedia.
- **Unstructured data analysis GE**, que permite la extracción de metadatos basados en el análisis de información no estructurada obtenida de recursos web.
- **Meta-data pre-processing GE**, que facilita la generación de objetos de programación a partir de varios formatos de metadatos.
- **Location GE**, que proporciona información de geolocalización como información de contexto obtenida de los dispositivos.
- **Query Broker GE**, que se ocupa del problema de proporcionar un mecanismo de consulta uniforme para la recuperación de datos almacenados en formatos heterogéneos.
- **Semantic Annotation GE**, que permite enriquecer la información multimedia u otros datos con etiquetas de metadatos semánticos para ser explotados por aplicaciones web semánticas.
- **Semantic Application Support GE**, que proporciona soporte para el conjunto básico de funcionalidades web semánticas que facilitan la programación de aplicaciones web semánticas.

6.3 Publish/Subscribe Context Broker GE

La función de este context broker es permitir la publicación de información de contexto por parte de unas entidades, denominadas Productores de contexto, de tal forma que dicha información esté disponible para otras entidades, las cuales se denominan Consumidores de contexto, cuyo objetivo es procesar dicha información.

Debemos de tener en cuenta que en la plataforma FIWARE, tanto aplicaciones como otros GEs pueden desempeñar el rol de Productores de contexto, Consumidores de contexto o ambos roles. Además, otro factor a tener en cuenta es que cuando se produce un cambio en la información de contexto, este se denomina evento.

El objetivo principal de FIWARE Context Broker GE es lograr un desacoplamiento total entre los productores y los consumidores de contexto. Entendemos esto como que los productores de Context Broker publican datos sin saber quién, dónde y cuándo los consumidores consumen dicha información, lo que implica que no estén conectados entre ellos. Por otra parte, los consumidores consumen aquella información que le interese, sin tener en cuenta quién la generó, únicamente se interesan en el evento.

6.3.1 Actores básicos

- **Context Broker(CB)**: Componente principal, el cual funciona como controlador y agregador de datos de contexto y como interfaz entre los actores de la arquitectura.
- **Productor de contexto(CP)**: Es un actor capaz de generar y actualizar contexto. En nuestro proyecto, este actor sería la Raspberry Pi.
- **Proveedor de contexto(CPr)**: Es un tipo de productor de contexto, el cual proporciona información de contexto bajo demanda. Eso significa que el Context Broker o incluso el

consumidor de contexto puede invocar al proveedor de contexto con el fin de adquirir información.

- **Consumidor de contexto(CC):** Es la entidad encargada de explotar la información de contexto. En nuestro proyecto, dicha entidad será el dispositivo en el que se encuentra freeboard.

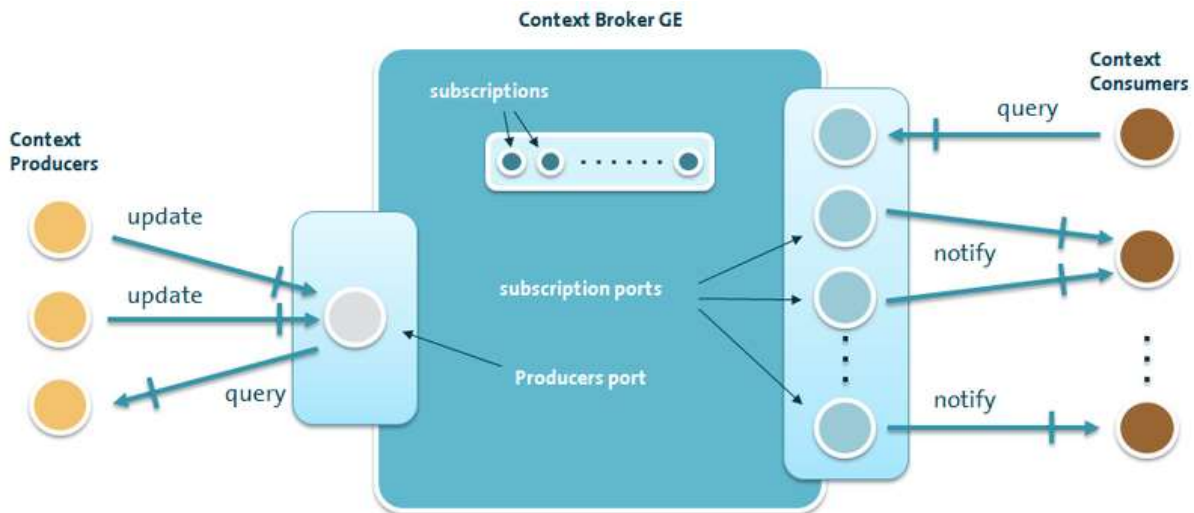


Ilustración 9: FIWARE - Context Broker GE

6.3.2 Estándares empleados por el Context Broker

Las comunicaciones que se generan entre los distintos componentes que conforman la arquitectura de Context Broker se realizan a través de la interfaz RESTful de FIWARE NGSI. Esta interfaz, es una interfaz estándar que permite manejar cualquier tipo de datos, incluidos los metadatos.

- **NGSI10:**
Las especificaciones de administración de contexto se basan en las especificaciones de gestión de contexto NGSI, las cuales han sido definidas por OMA (Open Mobile Alliance). Toman la forma de una especificación de enlace RESTful de las dos interfaces definidas en las especificaciones de gestión de contexto NGSI de OMA, concretamente NGSI-9 y NGSI-10 (ver FI-WARE NGSI Open RESTful API Specification). Sin embargo, las especificaciones de administración de contexto de FI-WARE NGSI no implementarán aquellas partes de las especificaciones de OMA que resulten ser inútiles o innecesariamente complejas. Además, resuelven algunas ambigüedades en las especificaciones de OMA y las amplían cuando es necesario para implementar la Visión FI-WARE.

A continuación vamos a nombrar las interacciones principales:

- **Interacciones Básicas y Entidades relacionadas:**



Ilustración 10: FIWARE - Interacciones Básicas y Entidades relacionadas

- Los productores de contexto publican elementos de datos/contexto invocando la operación *updateContext* en un Context Broker.
- Los consumidores de contexto pueden recuperar elementos de datos/contexto invocando la operación *queryContext* en un Context Broker.
- Los Context Brokers prestan habitualmente los datos de contexto impulsados por un productor de contexto y están listos para ser consultados. Esta es una característica distintiva del modelo de gestión de contexto de OMA en comparación con algunos estándares de Event Brokering o Event Notification.

○ **Interacciones relacionadas con los proveedores de contexto:**

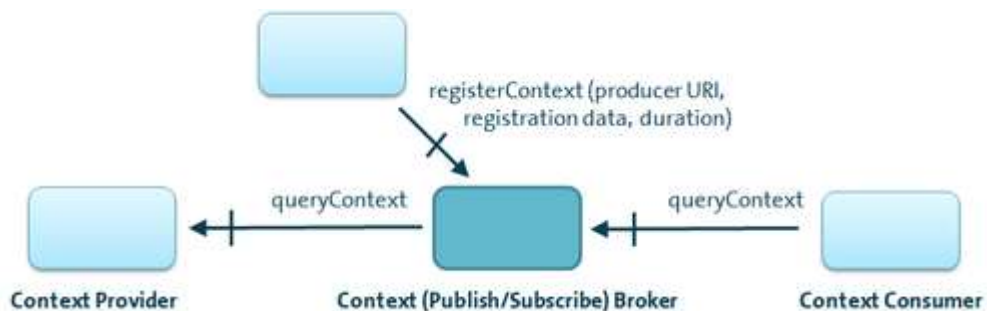


Ilustración 11: FIWARE - Interacciones relacionadas con los proveedores de contexto

Los productores de contexto publican elementos de datos/contexto invocando la operación *updateContext* en un Context Broker. Algunos productores de contexto (proveedores de contexto denominados) también pueden exportar una operación *queryContext*. Los intermediarios de contexto pueden invocar como resultado el *queryContext* reenviado de un consumidor de contexto en el caso de que haya algún registro que cubra las entidades del proveedor de contexto.

- Los consumidores de contexto pueden recuperar elementos de datos/contexto invocando la operación *queryContext* en un Context Broker.
- El intermediario de contexto reenvía la consulta al proveedor de contexto apropiado y devuelve el resultado al consumidor de contexto solicitante.

○ **Interacciones relacionadas con las actualizaciones reenviadas a los proveedores de contexto:**

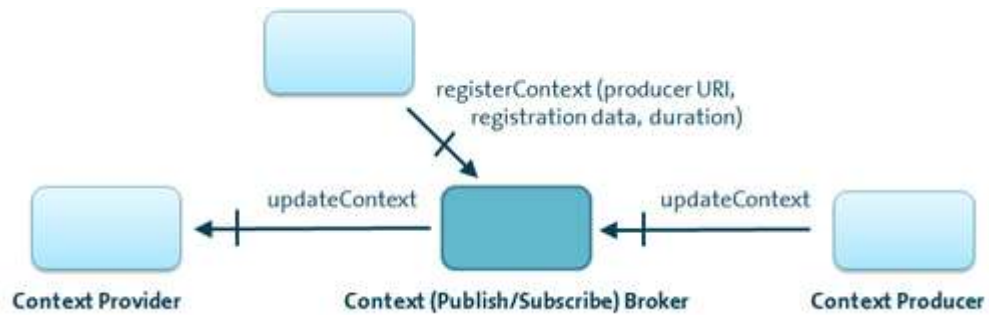


Ilustración 12: FIWARE - Interacciones relacionadas con las actualizaciones reenviadas a los proveedores de contexto

Un Context Broker puede reenviar actualizaciones de contexto a proveedores de contexto capaces de procesar dichos eventos.

- Un productor de contexto puede enviar una operación `updateContext` a un Context Broker.
 - El Context Broker reenvía el `updateContext` al proveedor de contexto adecuado
- **Interacciones para forzar a los consumidores de contexto a suscribirse a notificaciones específicas:**

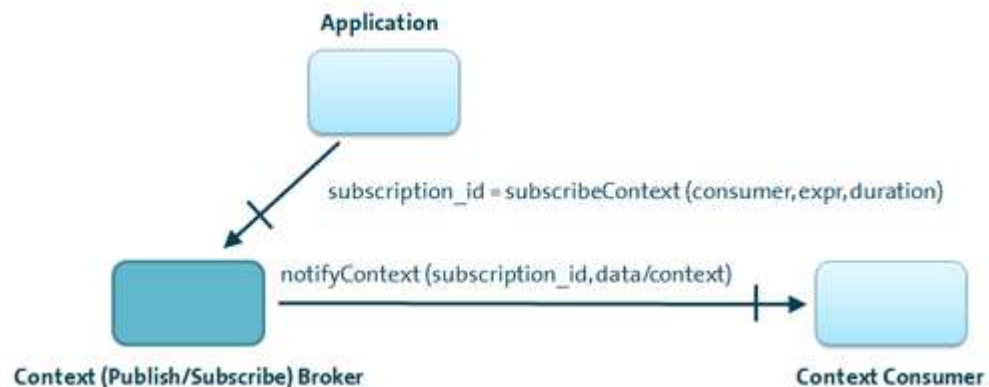


Ilustración 13: Interacciones para forzar a los consumidores de contexto a suscribirse a notificaciones específicas

Algunos consumidores de contexto pueden suscribirse a la recepción de elementos de datos/contexto que cumplan con ciertas condiciones, utilizando la operación `subscribeContext` que exporta el Context Broker. Se puede asignar una duración a las suscripciones.

- Los consumidores suscritos reciben espontáneamente elementos de datos/contexto que cumplen con esa suscripción a través de la operación `notifyContext` que exportan.
 - Tenga en cuenta que la aplicación que suscribe un consumidor de contexto particular puede ser o no el consumidor de contexto.
- **Registro de disponibilidad de entidades y atributos:**

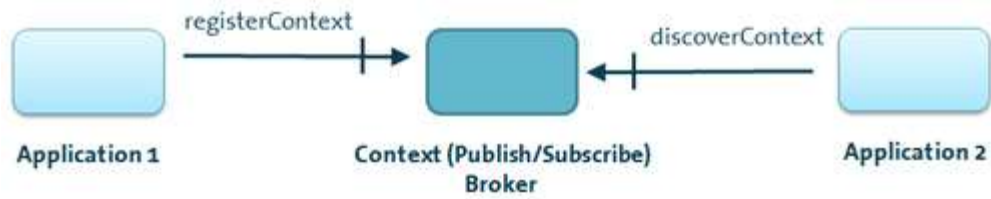


Ilustración 14: FIWARE - Registro de disponibilidad de entidades y atributos

La operación `registerContext` en Context Brokers se puede utilizar no solo para registrar Context Providers en qué operaciones `queryContext` se pueden invocar sino también para registrar la existencia de entidades en el sistema y la disponibilidad de atributos.

- Los intermediarios de contexto publicador/suscriptor pueden exportar operaciones para descubrir entidades o incluso atributos que se han registrado en el sistema.

○ **Suscripción de aplicaciones al registro de Entidades/Atributos:**



Ilustración 15: FIWARE - Suscripción de aplicaciones al registro de Entidades/Atributos

Algunas aplicaciones pueden suscribirse al registro de entidades o a la disponibilidad de atributos que cumplen con ciertas condiciones. Lo hacen mediante el uso de la operación `subscribeContextAvailability` que un intermediario de contexto puede exportar. Se puede asignar una duración a las suscripciones.

- Las aplicaciones suscritas reciben espontáneamente actualizaciones sobre nuevas entidades o atributos que cumplen con esa suscripción a través de la operación `notifyContextAvailability` que exportan.
- Tenga en cuenta que el suscriptor y las aplicaciones suscritas pueden no ser las mismas.

• **HTTP:**

El Protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol o HTTP) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web para comunicarse. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita

frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de sesión, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje, con cierto formato al servidor. El servidor le envía un mensaje de respuesta.

Los mensajes HTTP, son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

Los mensajes tienen la siguiente estructura:

- Línea inicial (termina con retorno de carro y un salto de línea):
 - Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.
 - Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.
- Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.
- Cuerpo del mensaje. Es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor. Por ejemplo, para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

HTTP define una serie predefinida de métodos de petición [12] (algunas veces referido como "verbos") que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se han ido aumentando según se iba avanzando en las versiones. Cada método indica la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa depende de la aplicación del servidor.

Los métodos más comunes son los siguientes:

- **GET:** recoger un recurso existente. La URL contiene toda la información necesaria que el servidor necesita para localizar y devolver el recurso.
 - **POST:** Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.
 - **PUT:** Sube, carga o realiza un upload de un recurso especificado (archivo), es el camino más eficiente para subir archivos a un servidor, esto es porque en POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor. En contraste, el método PUT te permite escribir un archivo en una conexión socket establecida con el servidor. La desventaja del método PUT es que los servidores de hosting compartido no lo tienen habilitado.
 - **DELETE:** Borra el recurso especificado.
- **Serialización de datos:**

La serialización de datos consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible.

En concreto, el Context Broker GE de FIWARE soporta tanto la serialización en formato "eXtensible Markup Language" (XML) como en formato "JavaScript Object Notation" (JSON).

Por un lado, XML define un conjunto de reglas para la codificación de documentos en un formato que sea legible y de lectura mecánica.

6.4 Uso del Context Broker

En este punto vamos a comentar cómo hemos hecho uso del Orion Context Broker [2] [6] en nuestra aplicación haciendo uso de toda la información detallada en los apartados anteriores.

Vamos a dividir este apartado en dos secciones. Por un lado, vamos a explicar cómo se lleva a cabo la creación de las entidades y, por otro lado, cómo se actualizan los valores de las entidades.

6.4.1 Creación de entidades

En función de los datos obtenidos mediante los dos sensores utilizados en este proyecto, hemos creado dos entidades diferenciadas para cada uno de ellos. Estas entidades se denominan *GPS* y *GyroscopeAccelerometer*, cuyo nombre identifica claramente para qué sensor los vamos a utilizar.

La creación de ambos tipos de entidades se realizará mediante una petición POST dirigida a la IP donde tengamos alojado el Context Broker con los siguientes cuerpos:

- Entidad *GyroscopeAccelerometer*:

```
"contextElements": [
  {
    "type": "GyroscopeAccelerometer",
    "isPattern": "false",
    "id": "GyroscopeAccelerometer",
    "attributes": [
      {
        "name": "X",
        "type": "float",
        "value": "0"
      },
      {
        "name": "Y",
        "type": "float",
        "value": "0"
      }
    ]
  }
],
"updateAction": "APPEND"
```

Ilustración 16: Entidad GyroscopeAccelerometer

- Entidad **GPS**:

```
"contextElements": [
  {
    "type": "GPS",
    "isPattern": "false",
    "id": "GPS",
    "attributes": [
      {
        "name": "latitude",
        "type": "float",
        "value": "0"
      },
      {
        "name": "longititude",
        "type": "float",
        "value": "0"
      }
    ]
  },
  {
    "updateAction": "APPEND"
  }
]
```

Ilustración 17: Entidad GPS

A parte de los campos tipo e id que definen el tipo de entidad y el identificador, la carga de la petición, también llamada payload, contiene un conjunto de atributos. Cada atributo contiene un valor y un tipo. En el caso de la entidad *GyroscopeAccelerometer* sólo tendremos los atributos “X” y “Y”, mientras en la entidad *GPS* tendremos los atributos “longititude” y “latitude”.

Estas entidades serán creadas en el momento en el que se obtengan datos de los sensores por primera vez. Para ello, se debe ejecutar la siguiente parte del código.

- URL de la petición:
 - http://IP_ContextBroker:1026/v1/queryContext

- Envío de la petición POST:

El siguiente fragmento de código, muestra cómo realizamos el envío de esta petición desde la Raspberry Pi.

```

orion_con = http.client.HTTPConnection(TCP_IP_ORION,
TCP_Puerto_ORION)
headers = {"Content-type": "application/json", "Accept":
"application/json"}
data = json.loads('{ "id" : "GyroscopeAcelerometer", "type" :
"GYRO", "X" : {"value" : 0, "type" : "Float"}, "Y" : {"value" : 0,
"type" : "Float"} }')
# Creacion de la entidad
try:
orion_con.request('POST', '/v2/entities', json.dumps(data),
headers)
resp = orion_con.getresponse()
print ("\t", resp.read())
except Exception as e:
print ('\t', e)

```

Ilustración 18: Creación de entidad GyroscopeAcelerometer

```

orion_con = httplib.HTTPConnection(TCP_IP_ORION, TCP_Puerto_ORION)
headers = {"Content-type": "application/json", "Accept":
"application/json"}
data = json.loads('{ "id" : "GPS", "type" : "GPS", "latitude" :
{"value" : 0, "type" : "float"}, "longitude" : {"value" : 0,
"type" : "Float"} }')
# Creacion de la entidad
try:
orion_con.request('POST', '/v2/entities', json.dumps(data),
headers)
resp = orion_con.getresponse()
print ("\t", resp.read())
except Exception as e:
print ('\t', e)

```

Ilustración 19: Creación de entidad GPS

6.4.2 Actualización de entidades

En cuanto a la actualización de las entidades, utilizaremos el método PUT. Para ello, en el cuerpo de la petición deberíamos modificar el valor del campo `updateAction` a `UPDATE`. El payload de la petición será la misma que en la creación de entidades.

A continuación, se incluye el fragmento que muestra cómo realizamos las actualizaciones:

```
# Envío de datos a Orion Context Broker
data = json.loads('{"X" : {"value" : ' + str(valueX) + ', "type" :
"Float"}, "Y" : {"value" : ' + str(valueY) + ', "type" : "Float"}}')
headers = {"Content-type": "application/json", "Accept":
"application/json"}
orion_con.request("PUT", "/v2/entities/GyroscopeAcelerometer/attrs", json.
dumps(data), headers)
```

Ilustración 20: Actualización de entidad GyroscopeAcelerometer

```
# Envío de datos a Orion Context Broker
data = json.loads('{"latitude" : {"value" : ' + str(gpsd.fix.
latitude) + ', "type" : "Float"}, "longitudo" : {"value" : ' +
str(gpsd.fix.longitudo) + ', "type" : "Float"}}')
headers = {"Content-type": "application/json", "Accept":
"application/json"}
orion_con.request("PUT", "/v2/entities/GPS/attrs", json.dumps(
data), headers)
```

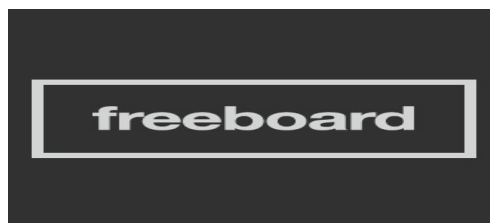
Ilustración 21: Actualización de entidad GPS

6 FREEBOARD

*No juzgues cada día por la cosecha que has obtenido,
sino por las semillas que has plantado.*

- Robert Louis Stevenson -

En este capítulo nos vamos a centrar en hablar de Freeboard. Vamos a dar una introducción de esta plataforma para conocer sus objetivos y posteriormente vamos a comentar cómo se trabaja con ella, cómo se configura y vamos a mostrar algunos resultados.



Freeboard fue desarrollado por Bug Labs, los creadores de un poderoso servicio llamado dweet, un poderoso servicio de mensajería diseñado para dispositivos de IoT.

Freeboard es un proyecto de panel de código abierto gratuito con suscripciones opcionales alojadas, el cual es fácil de integrar con diversas fuentes de datos y está diseñado de una forma muy elegante.

Freeboard está organizado con un panel de configuración arriba. El menú de *Datasources* el cual se encuentra en la parte superior izquierda del panel de configuración es donde se agregan las fuentes de datos de donde se obtendrán los datos que se van a mostrar a través de los widgets.

En la parte superior izquierda de la interfaz de usuario, se encuentra un menú cuya finalidad es permitir cargar y agregar los paneles de visualización, los llamados *widgets*. Los widgets incluidos son de tipo texto, indicador, sparkline, puntero, imagen, luz indicadora (una pantalla activa o inactiva, excelente para informes de estado del servicio y del servidor), Google Maps, Dweet y HTML.

Uno de los motivos principales por los que hemos escogido Freeboard para este proyecto, es debido a que presenta soporte para recuperar datos de FIWARE Orion Context Broker.

A continuación, vamos a comentar cómo hemos configurado nuestro lienzo para conseguir el resultado deseado en el proyecto.

6.1 Configuración de Datasources

Como en la mayoría del proyecto, se ha establecido una configuración para cada tipo de sensor, por lo que hemos creado dos *Datasources*.

Para llevar a cabo la configuración de las mismas, simplemente hay que rellenar un formulario en el que aparecen los siguientes campos:

- **TYPE:** En este campo debemos indicar el tipo de Datasource que queremos crear. En nuestro caso ambos se han creado con el tipo JSON, ya que los datos se obtienen con este formato.
- **NAME:** En este campo se establece el nombre.
- **URL:** En este campo se debe introducir la URL mediante la cual vamos a realizar la petición para la obtención de los valores deseados.
- **TRY THINGPROXY:** Este campo permite activar o desactivar *Thingproxy*(Un servidor proxy directo simple para procesar llamadas API a servidores que no envían encabezados CORS o admiten HTTPS).
- **REFRESH EVERY:** Este campo permite establecer el tiempo en segundos que se desee para realizar una nueva petición. Por defecto se establecen 5 segundos.
- **METHOD:** En este campo se introduce el tipo de método utilizado para realizar la petición. En nuestro caso vamos a utilizar el método GET.
- **BODY:** En este campo se introduce el cuerpo de la petición, únicamente si se realizan peticiones de tipo POST. En nuestro caso se quedará en blanco ya que no realizamos peticiones tipo POST.
- **HEADERS:** En este campo se introducen las cabeceras que se quieran utilizar para realizar las peticiones. En nuestro caso no utilizamos ninguna cabecera.

A continuación, podemos ver las ilustraciones 22 y 23, que muestran la configuración establecida para cada uno de los *Datasources* creados.

DATASOURCE

A datasource to load JSON data from a url.

TYPE: JSON

NAME: TFG-GYRO

URL: `http://217.217.50.98:1026/v2/entities/GyroscopeAcelerometer?o`

TRY THINGPROXY: YES

A direct JSON connection will be tried first, if that fails, a JSONP connection will be tried. If that fails, you can use thingproxy, which can solve many connection problems to APIs. [More information.](#)

REFRESH EVERY: 5 SECONDS

METHOD: GET

BODY:
The body of the request. Normally only used if method is POST

HEADERS:

Name	Value
Accept	application/json

ADD

SAVE CANCEL

Ilustración 22: Freeboard – Datasource Giroscopio/Acelerómetro

DATASOURCE

A datasource to load JSON data from a url.

TYPE: JSON

NAME: TFG-GPS

URL: `http://217.217.50.98:1026/v2/entities/GPS?options=keyValues&a`

TRY THINGPROXY: YES

A direct JSON connection will be tried first, if that fails, a JSONP connection will be tried. If that fails, you can use thingproxy, which can solve many connection problems to APIs. [More information.](#)

REFRESH EVERY: 5 SECONDS

METHOD: GET

BODY:
The body of the request. Normally only used if method is POST

HEADERS: ADD

SAVE CANCEL

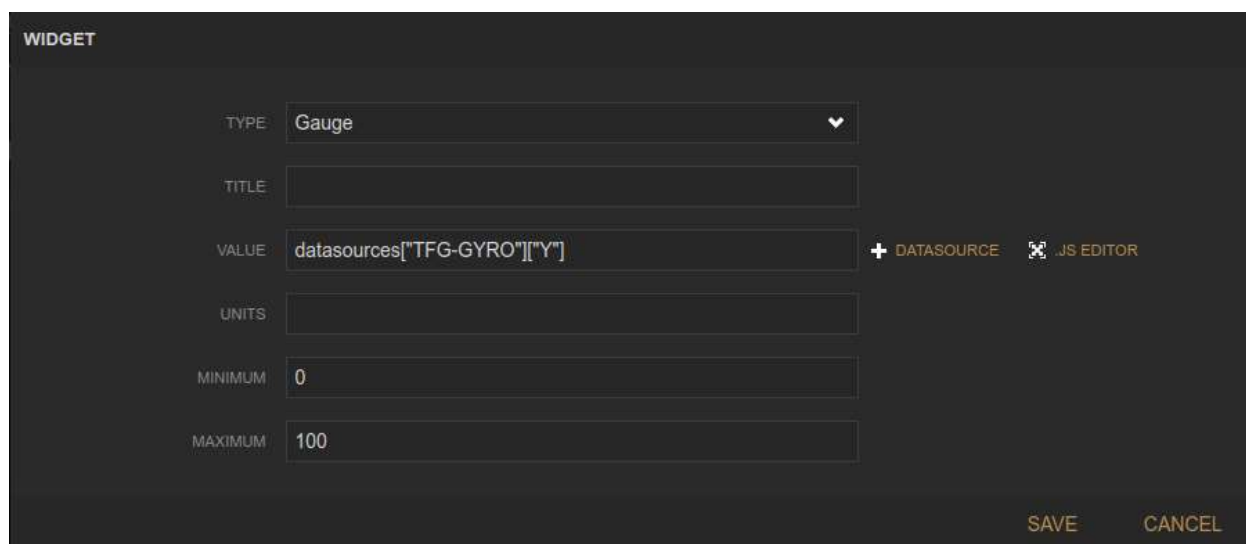
Ilustración 23: Freeboard – Datasource GPS

7.2 Configuración de Widgets

Para la creación y configuración de los widgets se debe rellenar un formulario que variará en función del tipo de widget que se desee. En este proyecto, hemos utilizado dos tipos distintos, por un lado, para el Giroscopio/Acelerómetro se han utilizado 3 widgets de tipo *Gauge*, uno para cada coordenada, y por otro lado para el GPS se ha utilizado un único widget de tipo *Google Maps*.

En el caso de los widgets de tipo *Gauge*, el formulario a rellenar contiene los siguientes campos:

- TYPE: Campo en el cual se indica el tipo de widget deseado, en este caso es tipo *Gauge*.
- TITLE: Campo que permite establecer un título para el widget.
- VALUE: Campo en el cual se configura el atributo de la entidad del cual se quieren mostrar los datos.
- UNITS: Campo que permite indicar el tipo de unidades de los datos que se van a mostrar.
- MINIMUM: Campo en el cual se establece el valor mínimo que se quiere establecer en el manómetro creado.
- MAXIMUM: Campo en el cual se establece el valor máximo que se quiere establecer en el manómetro creado.



The screenshot shows a dark-themed configuration window titled "WIDGET". It contains several input fields and buttons:

- TYPE:** A dropdown menu with "Gauge" selected.
- TITLE:** An empty text input field.
- VALUE:** A text input field containing the expression `datasources["TFG-GYRO"]["Y"]`. To the right of the field are two buttons: a plus sign followed by "DATASOURCE" and a JS icon followed by ".JS EDITOR".
- UNITS:** An empty text input field.
- MINIMUM:** A text input field containing the value "0".
- MAXIMUM:** A text input field containing the value "100".

At the bottom right of the window, there are two buttons: "SAVE" and "CANCEL".

Ilustración 24: Freeboard – Widget Gauge

En el caso de los widgets de tipo *Google Maps*, el formulario a rellenar contiene los siguientes campos:

- TYPE: Campo en el cual se indica el tipo de widget deseado, en este caso es tipo *Google Maps*.

- LATITUDE: Campo donde se introduce el atributo que corresponde con la latitud.
- LONGITUDE: Campo donde se introduce el atributo que corresponde con la longitud.
- DRAW PATH: Campo que permite que se active una propiedad de *Google Maps*.
- SIZE: Desplegable que permite indicar el tamaño que deseamos para el widget.
- CUSTOM GMAP STYLE: Campo que permite configurar el diseño del widget mediante código JavaScript.

WIDGET

TYPE: Google Map

LATITUDE: `datasources["TFG-GPS"]["latitude"]` + DATASOURCE .JS EDITOR

LONGITUDE: `datasources["TFG-GPS"]["longitude"]` + DATASOURCE .JS EDITOR

DRAW PATH: YES

SIZE: Medium

Small: 300x180 (1 column), Medium: 600x420 (2 columns), Large: 900x600 (3 columns).
NOTE: You must also set the containing Pane's COLUMNS value to 1 (Small), 2 (Medium) or 3 (Large)

CUSTOM GMAP STYLE

Paste your custom google map javascript style array here. For examples and inspiration, check out <https://snazzymaps.com>

SAVE CANCEL

Ilustración 25: Freeboard – Widget Google Maps

7.3 Presentación de los datos

A continuación, se van a incluir varias ilustraciones en las que se va a mostrar nuestro dashboard con datos reales obtenidos.

En la ilustración 26, podemos ver el panel de configuración, en el cual se puede ver que hemos creado dos Datasources distintas, una para cada sensor.

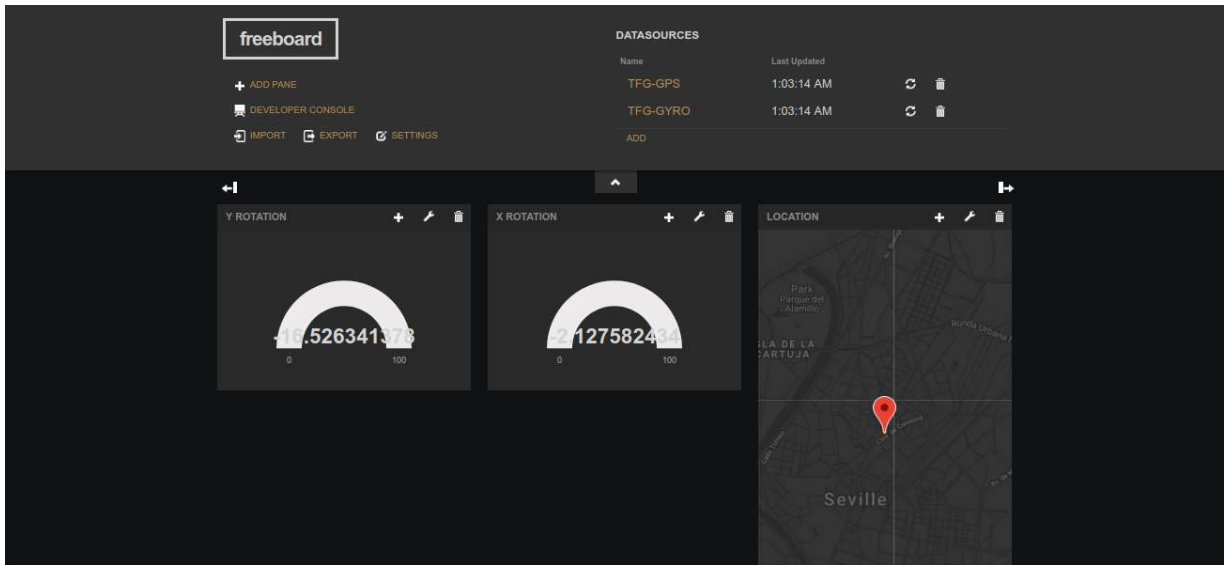


Ilustración 26: Freeboard – Dashboard 1

En la ilustración 27, se muestran los tres *widets* creados para mostrar los datos deseados. En el lado izquierdo podemos observar los datos del Acelerómetro/Giroscopio y en el lado derecho los de el GPS.

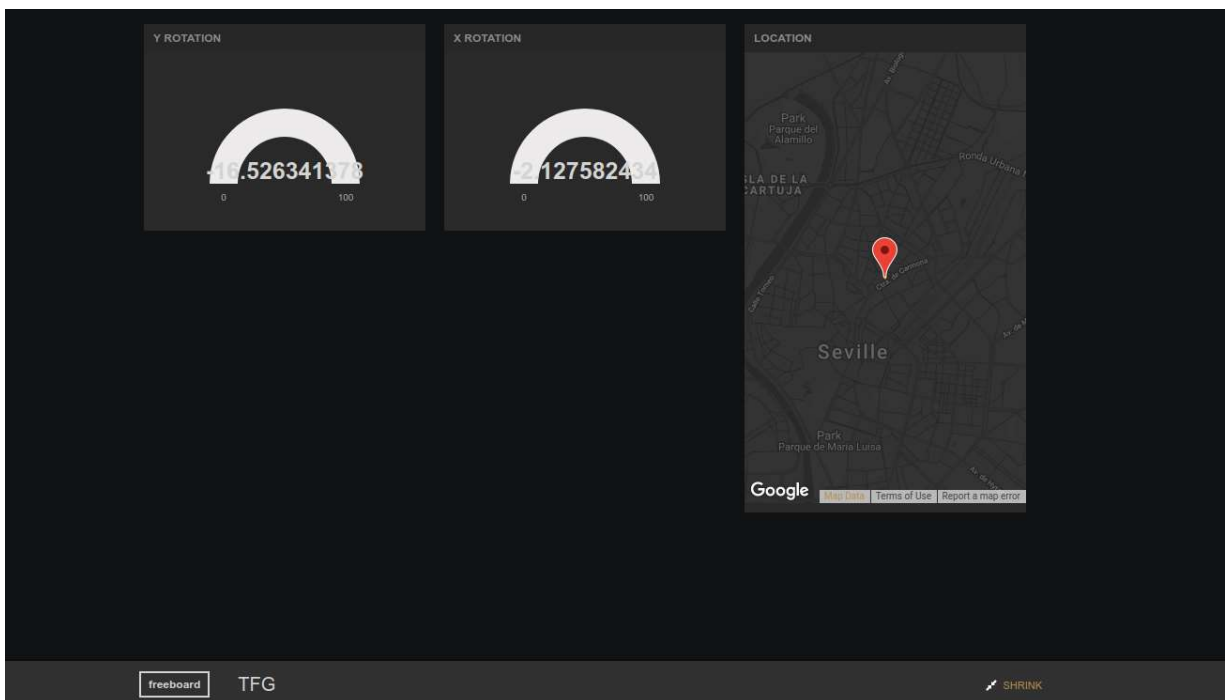


Ilustración 27: Freeboard – Dashboard 2

7 EJECUCIÓN Y RESULTADOS

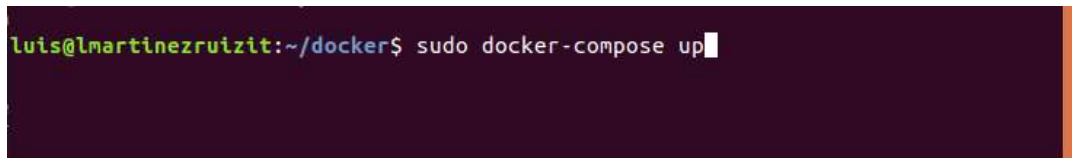
Si la oportunidad no llama, construye una puerta.

- Milton Berle -

En este capítulo vamos a realizar una ejecución del sistema completo y vamos a mostrar mediante capturas tanto el flujo de acciones que debemos de realizar como los resultados obtenidos en la ejecución.

En primer lugar, debemos de ejecutar el Docker en el que hemos desplegado el Orion Context Broker, aunque se detallará todo el proceso de despliegue en Docker en el *Anexo A*. A continuación, se muestra como se ejecuta y el resultado de su ejecución:

1. En primer lugar, se muestra la ejecución:



```
luis@lmartinezruizit:~/docker$ sudo docker-compose up
```

Ilustración 28: Ejecución - Docker

2. Ahora mostramos los datos que muestra cuando está arrancado:

```

Starting docker_mongo_1 ...
Starting docker_mongo_1 ... done
Starting docker_orion_1 ...
Starting docker_orion_1 ... done
Attaching to docker_mongo_1, docker_orion_1
mongo_1 | 2018-09-10T23:31:00.151+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=10ea80a0fd6e
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] db version v3.4.15
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] git version: 52e5b5fbaa3a2a5b1a217f5e647b5061817475f9
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1t 3 May 2016
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] allocator: tcmalloc
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] modules: none
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] build environment:
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten]   distmod: debian81
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten]   distarch: x86_64
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten]   target_arch: x86_64
mongo_1 | 2018-09-10T23:31:00.185+0000 I CONTROL [initandlisten] options: { storage: { journal: { enabled: false } } }
mongo_1 | 2018-09-10T23:31:00.201+0000 I - [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine,
so setting the active storage engine to 'wiredTiger'.
mongo_1 | 2018-09-10T23:31:00.201+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredT
iger storage engine
mongo_1 | 2018-09-10T23:31:00.201+0000 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
mongo_1 | 2018-09-10T23:31:00.201+0000 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=2441M,session_max=20000,eviction=(
threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(c
lose_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),verbose=(recovery_progress),,log=(enabled=false),
mongo_1 | 2018-09-10T23:31:02.701+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
mongo_1 | 2018-09-10T23:31:02.701+0000 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
mongo_1 | 2018-09-10T23:31:02.701+0000 I CONTROL [initandlisten]
mongo_1 | 2018-09-10T23:31:03.110+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diag
nostic_data'
mongo_1 | 2018-09-10T23:31:03.110+0000 I NETWORK [thread1] waiting for connections on port 27017
mongo_1 | 2018-09-10T23:31:04.187+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59264 #1 (1 connection now open)
mongo_1 | 2018-09-10T23:31:04.267+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59266 #2 (2 connections now open)
mongo_1 | 2018-09-10T23:31:04.268+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59268 #3 (3 connections now open)
mongo_1 | 2018-09-10T23:31:04.477+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59270 #4 (4 connections now open)
mongo_1 | 2018-09-10T23:31:04.479+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59272 #5 (5 connections now open)
mongo_1 | 2018-09-10T23:31:04.480+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59274 #6 (6 connections now open)
mongo_1 | 2018-09-10T23:31:04.481+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59276 #7 (7 connections now open)
mongo_1 | 2018-09-10T23:31:04.482+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59278 #8 (8 connections now open)
mongo_1 | 2018-09-10T23:31:04.483+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59280 #9 (9 connections now open)
mongo_1 | 2018-09-10T23:31:04.484+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:59282 #10 (10 connections now open)

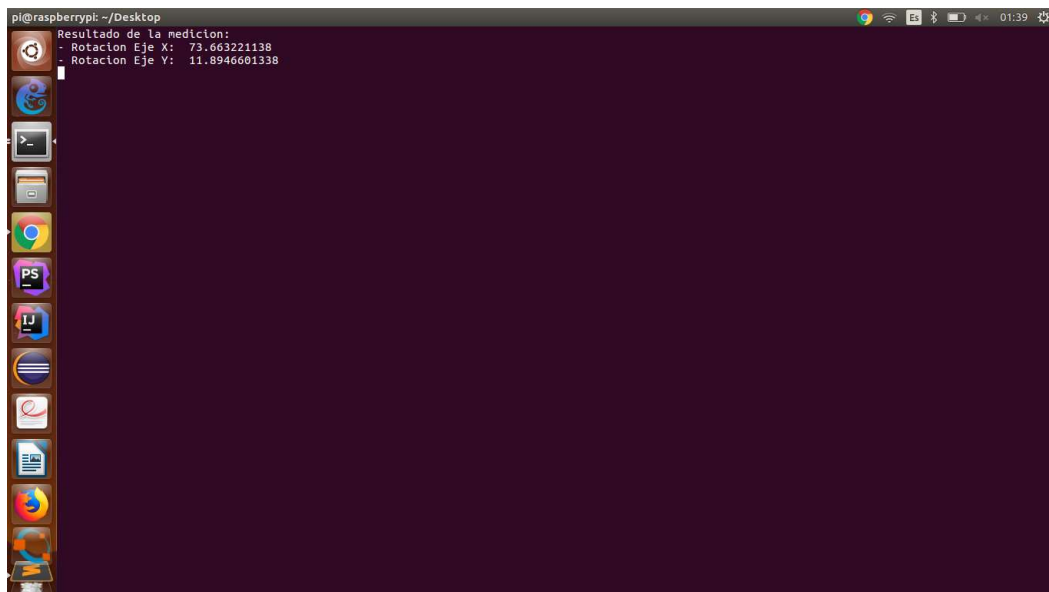
```

Ilustración 29: Ejecución – Docker Arrancado

7.1 MPU-6050

En este apartado vamos a mostrar una ejecución utilizando el sensor MPU-6050. En las ilustraciones 30 y 31 se pueden ver los resultados obtenidos al realizar una medición con este sensor:

1. En primer lugar, debemos de ejecutar el programa realizado en Python:



```

pi@raspberrypi: ~/Desktop
Resultado de la medición:
- Rotacion Eje X: 73.663221138
- Rotacion Eje Y: 11.8946601338

```

Ilustración 30: Ejecución – Resultado de medición con MPU-6050

2. A continuación, se van a mostrar los resultados obtenidos por Freeboard:

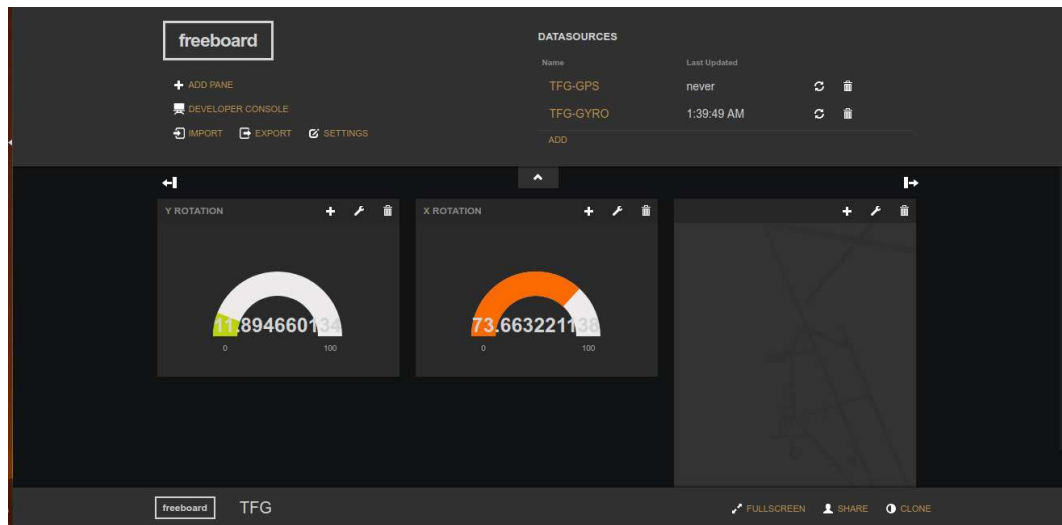


Ilustración 31: Ejecución – Resultado Freeboard MPU-6050

7.2 NEO-6M-0-001

En este apartado vamos a mostrar una ejecución utilizando el sensor NEO-6M-0-001. En las ilustraciones 32 y 33 se pueden ver los resultados obtenidos al realizar una medición con este sensor:

1. En primer lugar, debemos de ejecutar el programa realizado en Python:

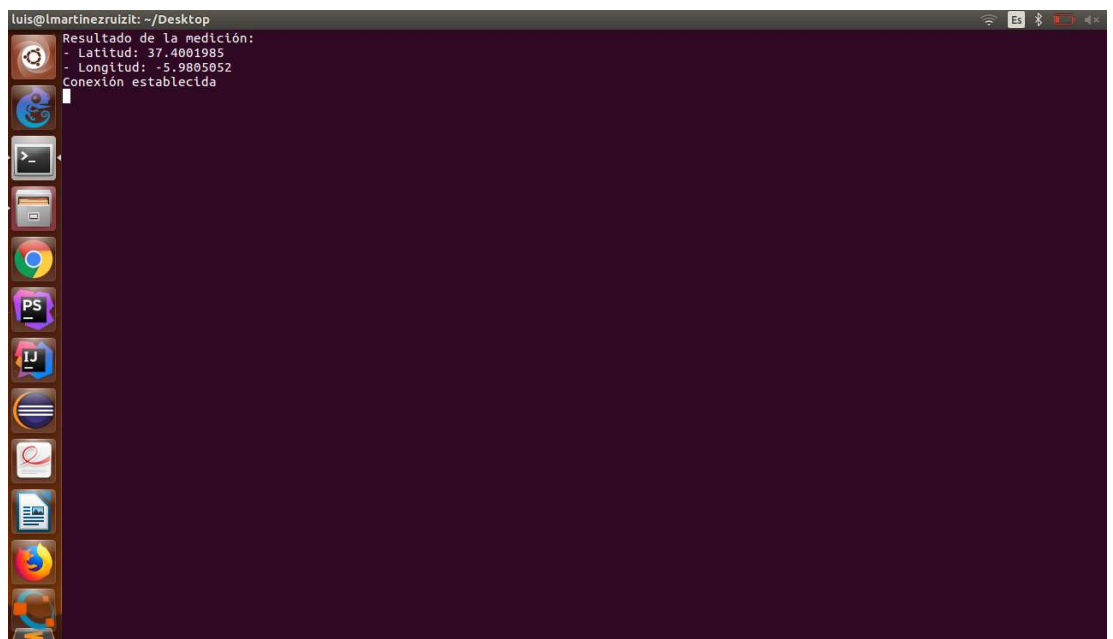


Ilustración 32: Ejecución – Resultado de medición con NEO-6M-0-001

2. A continuación, se van a mostrar los resultados obtenidos por Freeboard:

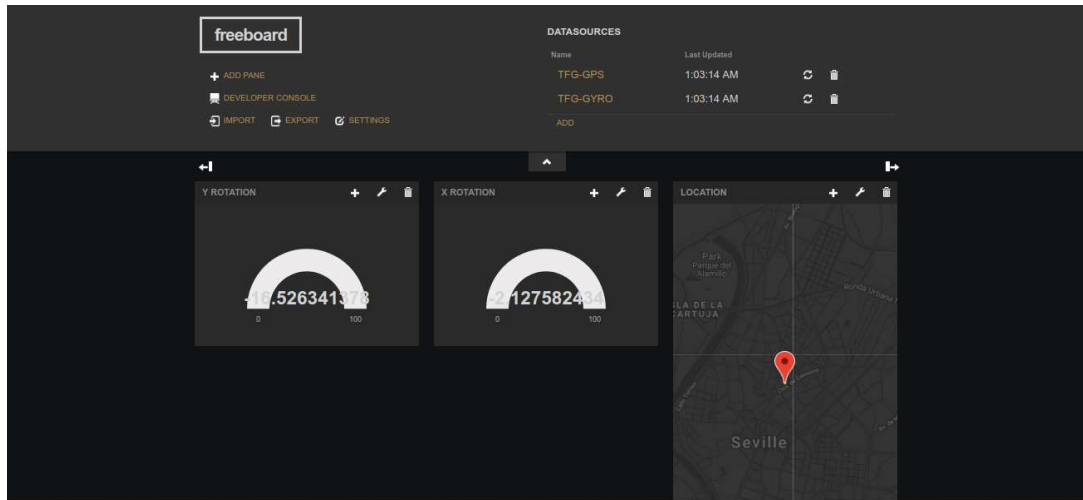


Ilustración 33: Ejecución – Resultado Freeboard NEO-6M-0-001

8 CONCLUSIONES Y LÍNEAS FUTURAS

He fallado una y otra vez en mi vida. Esa es la razón principal de mi éxito.

- Michael Jordan -

En este capítulo vamos a comentar un poco todo lo que nos ha aportado la realización del proyecto, indicando los aspectos más relevantes para nosotros del mismo y posteriormente vamos a indicar posibles mejoras.

9.1 Conclusiones

Como visión general, la realización de este proyecto ha sido una muy buena experiencia, ya que hemos trabajado con diversas tecnologías que tienen un presente y un futuro muy prometedor en el mundo del Internet de las Cosas (IoT).

En cuanto a lo personal, trabajar con Cloud Computing e IoT me ha aportado muchísimos conocimientos que desconocía por completo y con todos los conocimientos adquiridos durante estos años en la carrera, creo que me van a ayudar mucho en mi vida profesional.

Para llevar a cabo la realización del proyecto he tenido que indagar en diversos temas como el lenguaje de programación Python, con el cual no había trabajado nunca y que me ha sorprendido por su sencillez, el proyecto FIWARE, que desconocía por completo y que se trata de una plataforma muy sencilla para trabajar a la vez que potente, y por último Docker, el cual sí conocía pero no había trabajado con él y que creo que es una de las aplicaciones más potentes hoy en día.

Además, he podido conocer y comprender el funcionamiento de diversos sensores que hoy en día los contienen la mayoría de dispositivos *wearables*.

9.2 Líneas futuras

En cuanto a la mejora del proyecto, creo que se puede trabajar con los datos obtenidos en el mismo en diversos sectores, debido a que en la actualidad, ambos sensores tienen distintas funcionalidades.

A continuación, voy a comentar las funcionalidades que pueden ser más relevantes desde mi punto de vista:

- **Trabajar con sensores en el mundo Automovilístico:**

El motivo fundamental por el que se ha lanzado este proyecto es para poder proporcionar ayuda al conductor de un vehículo. La utilización de estos sensores, proporciona un gran abanico de recursos entre los que destacamos los siguientes:

- Se pueden calcular los recorridos que queremos realizar.
- Se pueden ver los recorridos que hemos realizado, permitiendo almacenarlos para futuras visualizaciones.
- En caso de que haya algún contratiempo, mediante el sensor GPS, podemos estar localizados.

- **Trabajar con los sensores en el mundo del Fitness:**

Hoy en día, en el mundo del Fitness es muy común usar dispositivos *wearables* los cuales poseen diversidad de sensores entre ellos, GPS y Acelerómetro. Estos sensores permiten obtener calcular recorridos, permiten estar localizado en caso de deportes de alto riesgo o una vez realizado un recorrido, obtener el trayecto realizado. Por lo que con los datos obtenidos, podríamos realizar aplicaciones cuyos objetivos obtener las funcionalidades comentadas. Hay que tener en cuenta que para conseguir esto, es necesario proporcionar internet a la Raspberry Pi.

- **Trabajar con los sensores en el mundo de los videojuegos:**

Cómo bien sabemos, en la actualidad, los videojuegos es uno de los sectores más potente y que evoluciona de forma diaria y a pasos agigantados. Los Smartphones principalmente, hacen uso del Acelerómetro/Giroscopio en la mayoría de videojuegos para detectar la rotación de la pantalla y obtener la orientación de la pantalla. Por ello, si integramos dicha funcionalidad con la de un GPS, podemos crear videojuegos tales como el famoso *Pokémon GO*, que dió la vuelta al mundo hace unos años.

ANEXO A:

MANUAL DE INSTALACIÓN DE ORION CONTEXT BROKER UTILIZANDO DOCKER

En este anexo nos vamos a centrar en explicar cómo hemos realizado la instalación de Orion Context Broker mediante la utilización de Docker [3].

Para trabajar con Docker, necesitamos una distribución Linux de 64 bits y las más extendidas y compatibles con Docker son Ubuntu y Debian. En nuestro caso, hemos utilizado una distribución Ubuntu 16.04 LTS.

A.1 Instalación de Docker

1. CREACIÓN DE DIRECTORIO DE TRABAJO

```
$ mkdir Docker
```

2. ACTUALIZAR

```
$ sudo apt-get update  
$ sudo apt-get install apt-transport-https ca-certificates -y
```

3. AÑADIR LA NUEVA CLAVE GPG

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys  
58118E89F3A912897C070ADBF76221572C52609D
```

4. INSTALACIÓN

```
$ sudo apt-get update  
$ sudo apt-get purge lxc-docker  
$ sudo apt-get install linux-image-extra-$(uname -r) -y
```

```
$ sudo apt-get install Docker-engine cgroup-lite apparmor -y
$ sudo usermod -a -G Docker $USER
```

Una vez ejecutados esta secuencia de comandos, ya sería posible ejecutar docker en nuestro entorno de trabajo. Ahora nos queda preparar los contenedores con Orion Context Broker.

Para ello seguiremos la documentación elaborada por Docker y FiWARE, donde el primer paso consiste en disponer de Docker Compose, una herramienta que nos permitirá definir y ejecutar contenedores y aplicaciones que empleen varios contenedores. Esto será necesario debido a Orion no estará solo, sino que necesitaremos también MongoDB para el almacenamiento de los datos. Para ello, seguiremos el apartado de instalación correspondiente a Linux en la documentación de instalación de Docker Compose.

A.2 Instalación de Docker

1. DESCARGA DE LA ÚLTIMA VERSIÓN DE DOCKER-COMPOSE

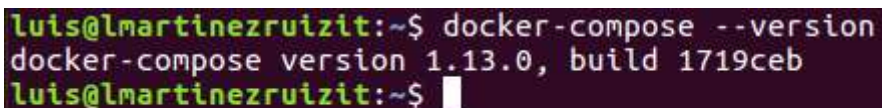
```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-
compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
```

2. ASIGNAR PERMISOS DE INSTALACIÓN AL FICHERO BINARIO

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

3. TEST DE INSTALACIÓN

```
$ docker-compose --version
```



```
luis@lmartinezruizit:~$ docker-compose --version
docker-compose version 1.13.0, build 1719ceb
luis@lmartinezruizit:~$
```

Ilustración 34: Docker - Instalación

A.3 Creación de aplicación Orion en Docker

Por último, es necesario crear nuestra aplicación multicontenedor. Esta aplicación se creará en un fichero denominado *docker-compose.yml*.

El contenido del fichero se muestra en la ilustración 35.

mediante Orion Context Broker de Fiware

```
# Note that mongo container is started before Orion, as Orion needs it
# as dependency and reserve the order has been found problematic in some
# low resource hosts
mongo:
  image: mongo:3.4
  command: --nojournal

orion:
  image: fiware/orion
  links:
    - mongo
  ports:
    - "1026:1026"
  command: -dbhost mongo
```

Ilustración 35: Configuración OCB y MongoDB con Docker

El contenido del fichero está definiendo tanto la aplicación *mongoDB* como *ORION*. En el caso de *ORION*, se están indicando una serie de parámetros que hay que pasarle para realizar la conexión con *mongoDB* y los puertos que deben exponerse fuera de los contenedores, que además son mapeados dentro y fuera del contenedor. En este caso como el contenedor no va a tener ninguna otra aplicación funcionando y nosotros tampoco la tendremos en el host, el mapeo puede ser al mismo puerto tanto dentro como fuera del contenedor.

Una vez creado el fichero, podemos pasar a la ejecución de nuestra aplicación a través de Docker. Esta ejecución se lleva a cabo mediante el siguiente comando:

```
$ sudo docker-compose up
```

```

luis@lmarinezruizit:~/docker$ sudo docker-compose up
[sudo] password for luis:
Starting docker_mongo_1 ...
Starting docker_mongo_1 ... done
Starting docker_orion_1 ...
Starting docker_orion_1 ... done
Attaching to docker_mongo_1, docker_orion_1
mongo_1 | 2018-09-02T11:45:39.500+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=10ea80afd6e
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] db version v3.4.15
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] git version: 52e5b5fbaa3a2a5b1a217f5e647b5061817475f9
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1t 3 May 2016
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] allocator: tcmalloc
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] modules: none
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] build environment:
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] distmod: debian81
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] distarch: x86_64
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] target_arch: x86_64
mongo_1 | 2018-09-02T11:45:39.534+0000 I CONTROL [initandlisten] options: { storage: { journal: { enabled: false } } }
mongo_1 | 2018-09-02T11:45:39.550+0000 I - [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine,
so setting the active storage engine to 'wiredTiger'
mongo_1 | 2018-09-02T11:45:39.550+0000 I STORAGE [initandlisten]
mongo_1 | 2018-09-02T11:45:39.550+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredT
iger storage engine
mongo_1 | 2018-09-02T11:45:39.550+0000 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-filesystem
mongo_1 | 2018-09-02T11:45:39.550+0000 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=2441M,session_max=20000,eviction=(
threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(c
lose_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),verbose=(recovery_progress),,log=(enabled=false),
mongo_1 | 2018-09-02T11:45:43.152+0000 I CONTROL [initandlisten]
mongo_1 | 2018-09-02T11:45:43.152+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
mongo_1 | 2018-09-02T11:45:43.153+0000 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
mongo_1 | 2018-09-02T11:45:43.153+0000 I CONTROL [initandlisten]
mongo_1 | 2018-09-02T11:45:44.023+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diag
nostic.data'
mongo_1 | 2018-09-02T11:45:44.024+0000 I NETWORK [thread1] waiting for connections on port 27017
mongo_1 | 2018-09-02T11:45:45.395+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43220 #1 (1 connection now open)
mongo_1 | 2018-09-02T11:45:45.566+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43222 #2 (2 connections now open)
mongo_1 | 2018-09-02T11:45:45.569+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43224 #3 (3 connections now open)
mongo_1 | 2018-09-02T11:45:45.570+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43226 #4 (4 connections now open)
mongo_1 | 2018-09-02T11:45:45.571+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43228 #5 (5 connections now open)
mongo_1 | 2018-09-02T11:45:45.572+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43230 #6 (6 connections now open)
mongo_1 | 2018-09-02T11:45:45.573+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43232 #7 (7 connections now open)
mongo_1 | 2018-09-02T11:45:45.574+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43234 #8 (8 connections now open)
mongo_1 | 2018-09-02T11:45:45.575+0000 I NETWORK [thread1] connection accepted from 172.17.0.3:43236 #9 (9 connections now open)

```

Ilustración 36: Docker - Ejecución

Esta descarga de los diferentes programas, se trata en realidad de la descarga de los diferentes contenedores (imágenes) ya preparadas y disponibles en la plataforma Docker, de manera que su despliegue se facilita al poder proporcionar una configuración mínima mediante el método empleado, y al encontrarse ya la dependencias necesarias dentro de los contenedores.

Una vez ejecutado, podemos comprobar que Orion se está ejecutando en nuestro host, de las dos siguientes maneras:

1. Ejecutando el comando:

```
$ curl localhost:1026/version
```

```

luis@lmarinezruizit:~$ curl localhost:1026/version
{
  "orion" : {
    "version" : "1.14.0-next",
    "uptime" : "0 d, 0 h, 5 m, 24 s",
    "git_hash" : "f3f6f62c7caaed624db03a236699e59afc04ea59",
    "compile_time" : "Wed Jul 4 16:26:04 UTC 2018",
    "compiled_by" : "root",
    "compiled_in" : "387d4b194446",
    "release_date" : "Wed Jul 4 16:26:04 UTC 2018",
    "doc" : "https://fiware-orion.readthedocs.org/en/master/"
  }
}

```

Ilustración 37: Orion CB -Ejecución

- Realizando la siguiente petición HTTP:

METHOD SCHEME // HOST [":" PORT] [PATH ["?" QUERY]]

GET http://localhost:1026/version

length: 29 bytes

Ilustración 38: Orion CB – Test de ejecución: Petición HTTP

```
GET /version HTTP/1.1
Accept: application/json
Host: localhost:1026

HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 362
Content-Type: application/json
Fiware-Correlator: f817e30a-aea5-11e8-ba53-0242ac110003
Date: Sun, 02 Sep 2018 11:47:29 GMT

{
  "orion" : {
    "version" : "1.14.0-next",
    "uptime" : "0 d, 0 h, 1 m, 45 s",
    "git_hash" : "f3f6f62c7caaed624db03a236699e59afc04ea59",
    "compile_time" : "Wed Jul 4 16:26:04 UTC 2018",
    "compiled_by" : "root",
    "compiled_in" : "387d4b194446",
    "release_date" : "Wed Jul 4 16:26:04 UTC 2018",
    "doc" : "https://fiware-orion.readthedocs.org/en/master/"
  }
}
```

Ilustración 39: Orion CB – Test de ejecución: Resultado de realizar petición HTTP

ANEXO B:

CONFIGURACIÓN Y CONEXIÓN DE SENSORES CON RASPBERRY PI

En este anexo vamos a detallar cómo hemos realizado tanto la conexión para la obtención de los datos de ambos sensores, como la configuración de los mismos a la Raspberry Pi.

B.1 Sensor MPU-6050

En este apartado nos centramos en el Giroscopio/Acelerómetro. [11]

1. CONEXIÓN

Para poder trabajar con este sensor, es necesario disponer de cuatro cables macho/hembra. Los cuales vamos a conectar a los pines de la Raspberry Pi mostrados en la Tabla 1.

Raspberry Pi	MPU-6050
Pin 1 (3.3V)	VCC
Pin 3 (SDA)	SDA
Pin 5 (SCL)	SCL
Pin 6 (GND)	GND

Tabla 1: MPU6050 - Conexión

Una vez identificados dichos pines, la conexión quedaría de la siguiente manera:

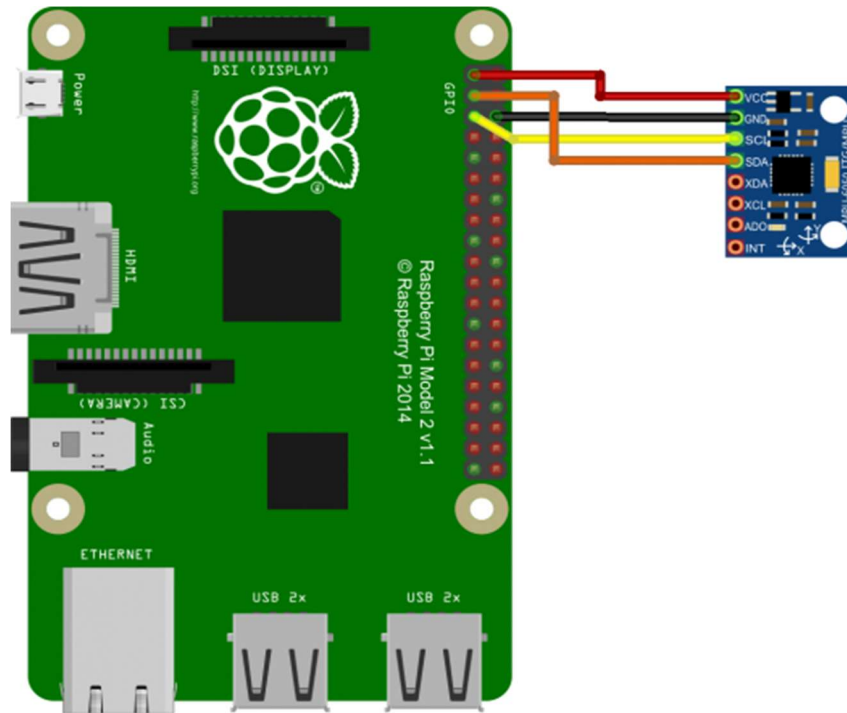


Ilustración 40: MPU6050 - Conexión Sensor MPU-6050 con Raspberry Pi

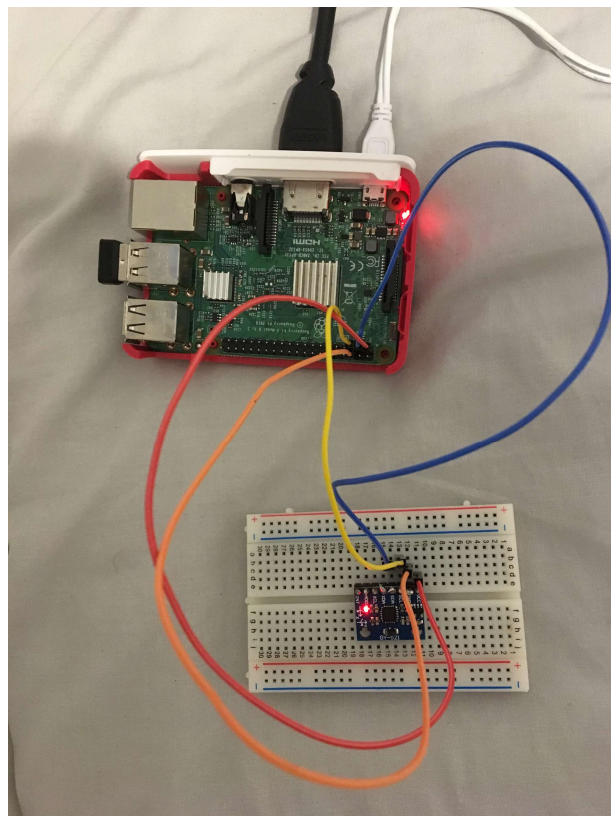


Ilustración 41: MPU6050 – Conexión Sensor MPU-6050 con Raspberry Pi (2)

2. CONFIGURACIÓN

Una vez que se ha conectado el sensor a la Raspberry Pi de forma correcta, vamos a configurar la misma para poder escuchar a través de los pines en los que se ha conectado.

1. En primer lugar se debe habilitar SPI e I2C. Para ello debemos utilizar raspi-config, que se trata de un menú para establecer la configuración inicial de nuestra Raspberry Pi. Para poder trabajar con este menú debemos de ejecutar el siguiente comando:

```
$ sudo raspi-config
```

Una vez hemos accedido a este menú, activamos I2C que se encuentra en “8.Opciones avanzadas”->”A7 I2C”.

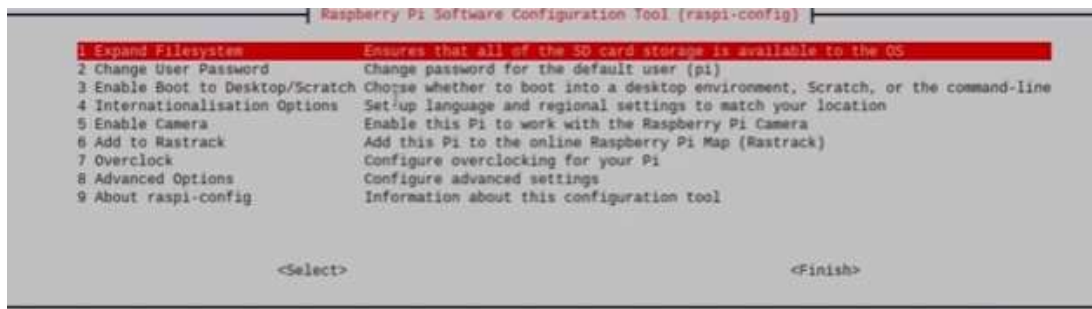


Ilustración 42: MPU6050 - Configuración I2C-1

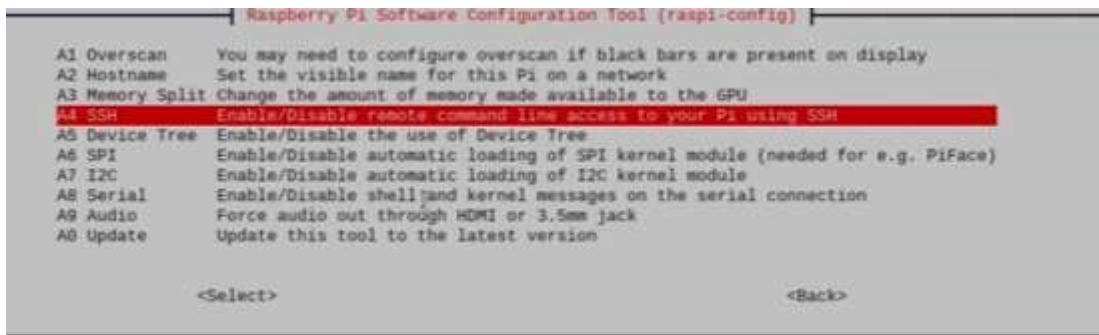


Ilustración 43: MPU6050 - Configuración I2C-2

2. Posteriormente editamos el fichero `/etc/modules` y añadimos las dos siguientes líneas:

```
i2c-bcm2708  
i2c-dev
```

3. Una vez modificado el fichero, instalamos las herramientas que van a ser necesarias para el correcto funcionamiento. Para ello, ejecutamos el siguiente comando:

```
$ sudo apt-get install i2c-tools python-smbus
```

4. Para finalizar, realizamos una pequeña prueba para asegurarnos su correcto funcionamiento, mediante el siguiente comando:

```
$ sudo i2cdetect -y 1
```

El resultado mostrado debería ser el siguiente:

```
pi @ raspberrypi ~ $ sudo i2cdetect -y 1
 0 1 2 3 4 5 6 7 8 9 abcdef
00: - - - - -
10: - - - - -
20: - - - - -
30: - - - - -
40: - - - - -
50: - - - - -
60: - - - - - 68 - - - - -
70: - - - - -
```

Ilustración 44: MPU6050 - Test de funcionamiento

Una vez realizados todos estos pasos, nuestra Raspberry Pi estaría lista para trabajar de forma adecuada.

B.2 Sensor U-blox NEO-6M-0-001

En este apartado nos centramos en el GPS. [7] [8] [9]

1. CONEXIÓN

Para poder trabajar con este sensor, es necesario disponer de cuatro cables macho/hembra. Los cuales vamos a conectar a los pines de la Raspberry Pi mostrados en la Tabla 1.

Raspberry Pi	MPU-6050
Pin 1 (3.3V)	VCC
Pin 6 (GND)	GND
Pin 8 (TXD0)	TXD0
Pin 10 (RXD0)	RXD0

Tabla 2: NEO-6M-0-001 - Conexión

Una vez identificados dichos pines, la conexión quedaría de la siguiente manera:

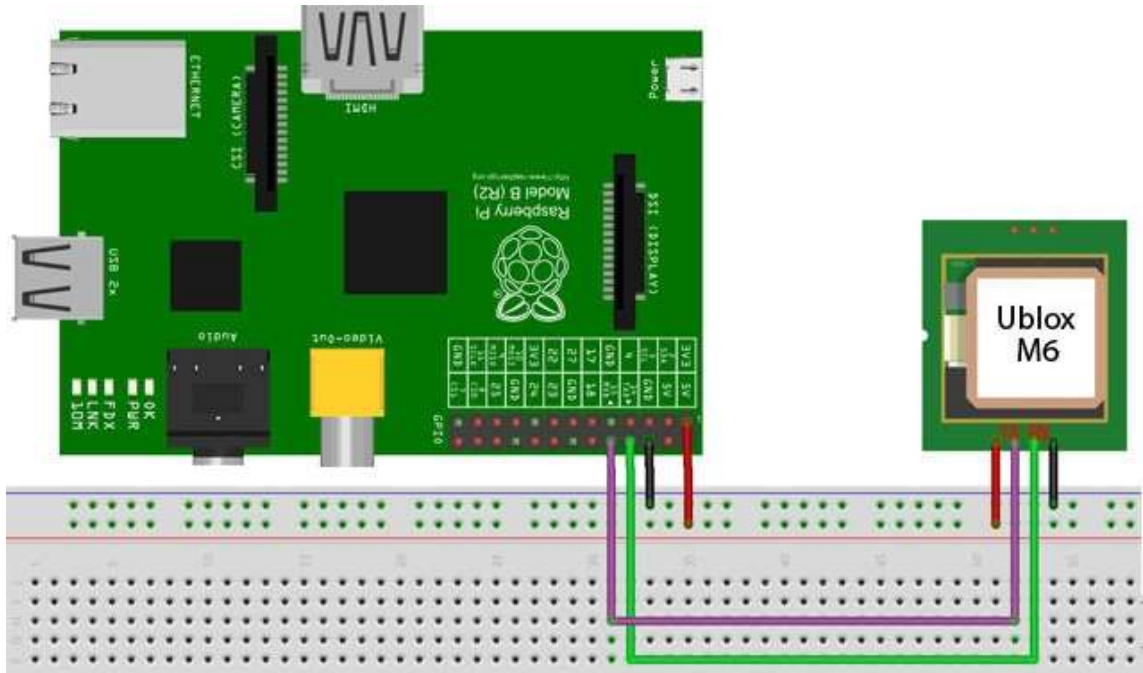


Ilustración 45: NEO-6M-0-001 - Conexión del sensor NEO-6M-0-001 con Raspberry Pi

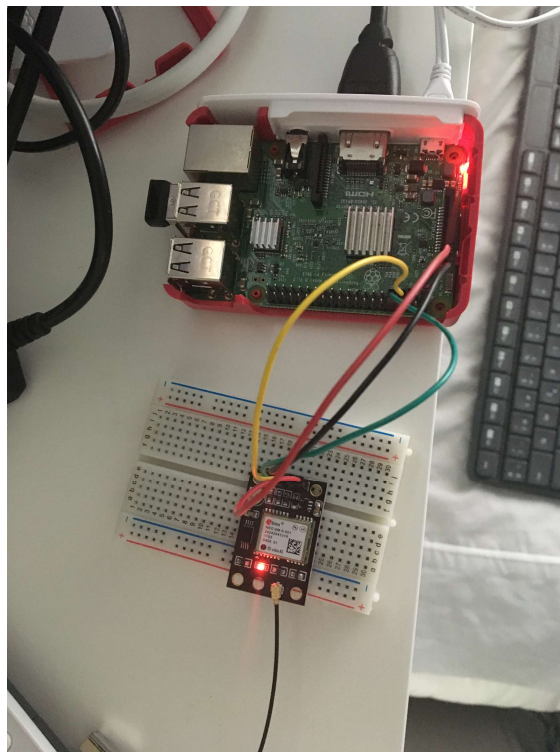


Ilustración 46: NEO-6M-0-001 - Conexión del sensor NEO-6M-0-001 con Raspberry Pi (2)

2. CONFIGURACIÓN

Una vez que se ha conectado el sensor a la Raspberry Pi de forma correcta, vamos a configurar la misma para poder escuchar a través de los pines en los que se ha conectado.

1. Modificamos el fichero `/boot/cmdline.txt`:

```
$ nano /boot/cmdline.txt
```

Modificamos:

```
dwc_otg.lpm_enable=0      console=tyl      console=tyAMA0,115200
root=/dev/mmcblk0p2      rootfstype=ext4      elevator=deadline      rootwait
fbcon=map:10 fbcon=font:ProFont6x11 logo.nologo systemd.unit=rescue.target
```

Por:

```
dwc_otg.lpm_enable=0 console=tyl root=/dev/mmcblk0p2 rootfstype=ext4
elevator=deadline rootwait fbcon=map:10 fbcon=font:ProFont6x11
logo.nologo systemd.unit=rescue.target
```

2. Modificamos el fichero `/boot/config.txt`, añadiendo:

```
enable_uart=1
```

3. Reiniciamos la Raspberry Pi para que se apliquen los cambios realizados.
4. Una vez se ha reiniciados, se van a instalar una serie de dependencias que serán necesarias, para ello ejecutamos el siguiente comando:

```
$ sudo apt-get install gpsd gpsd-clients python-gps
```

5. Posteriormente arrancamos GPSD (que se trata de una aplicación que corre en segundo plano y que recoge los datos del módulo GPS y los envía a través de un socket que debe ser activado cada vez que iniciamos la Raspberry Pi), para ello debemos de ejecutar el siguiente comando:

```
$ sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock
```

6. Una vez realizados los pasos anteriores podemos comprobar que está funcionando de forma correcta, ejecutando:

```
$ cgps -s
```

Al ejecutar este comando debería de aparecer algo parecido a lo que aparece en la ilustración 47.

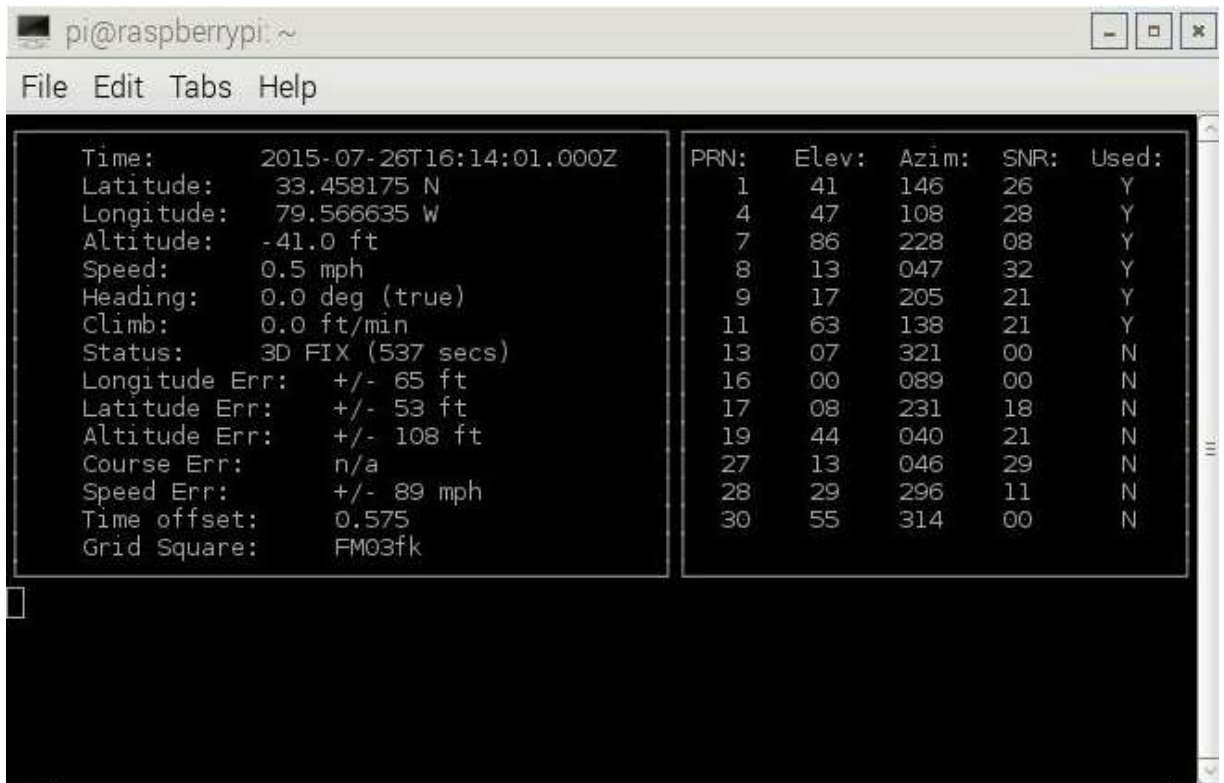


Ilustración 47: NEO-6M-0-001 – Test de Funcionamiento

Hay que tener en cuenta que el módulo GPS puede tardar un poco en comenzar a enviar datos. Cuando esté en funcionamiento, el led que contiene comenzará a parpadear y en el daemon de la ilustración anterior, comenzarán a aparecer datos reales.

REFERENCIAS

- [1] «Docker». [En línea]. Disponible: <https://docker.com/>
- [2] «fiware-orion.readthedocs». [En línea]. Disponible: <https://fiware-orion.readthedocs.io/en/0.24.0/user/docker/>
- [3] Docker, «How to use Orion Context Broker with Docker». [En línea]. Disponible: <https://hub.docker.com/r/fiware/orion/>
- [4] «FIWARE». [En línea]. Disponible: <https://www.fiware.org/>
- [5] «Forge FIWARE». [En línea]. Disponible: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Welcome_to_the_FIWARE_Wiki
- [6] Fiware-orion, «Welcome to Orion Context Broker». [En línea]. Disponible: <https://fiware-orion.readthedocs.io/en/master/>
- [7] Spence, Instructables, «Raspberry-Pi-the-Neo-6M-GPS». [En línea]. Disponible: <https://www.instructables.com/id/Raspberry-Pi-the-Neo-6M-GPS/>
- [8] Big Dan, Bigdanzblog, «Connecting u-blox neo-6m GPS to Raspberry Pi». [En línea]. Disponible: <https://bigdanzblog.wordpress.com/2015/01/18/connecting-u-blox-neo-6m-gps-to-raspberry-pi/>
- [9] Minitronica, «Localizador GPS con Raspberry Pi y Ublox NEO 6M». [En línea]. Disponible: <https://www.minitronica.com/tracker-gps-raspberry-pi-ublox-neo/>
- [10] Wikipedia, «Universal Asynchronous Receiver-Transmitter». [En línea]. Disponible: https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter
- [11] Tutorials-Raspberrypi, «Measuring Rotation and acceleration with the Raspberry Pi». [En línea]. Disponible: <https://tutorials-raspberrypi.com/measuring-rotation-and-acceleration-raspberry-pi/>
- [12] Luis Alberto Balam Guzmán, Youtube, «Llamadas Rest al Orion Context Broker». [En línea]. Disponible: <https://www.youtube.com/watch?v=PHOmmBOUc3M>
- [13] Vicente García, «INTRODUCCIÓN AL I2C BUS». [En línea]. Disponible: <https://www.diarioelectronicohoy.com/blog/introduccion-al-i2c-bus>

[14] Azahara Benito Carrillo, Viavansi, «FIWARE: Te explicamos que és». [En línea]. Disponible: <https://www.viavansi.com/blog-xnoccio/es/fiware-e/>

[15] «MongoDB». [En línea]. Disponible: <https://www.mongodb.com/>

[16] «RestletClient». [En línea]. Disponible: <https://restlet.com/modules/client/>

[17] «Freeboard». [En línea]. Disponible: <https://freeboard.io/>

[18] Vicente García, «SENSOR MPU6050». [En línea]. Disponible: <https://www.diarioelectronicohoy.com/blog/sensor-mpu6050>

[19] Invensense, «Datashet MPU-6050». [En línea]. Disponible: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>

[20] U-blox, «Datashet NEO-6». [En línea]. Disponible: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf