

Recolección de datos de sensores en la IoT con Arduino



Se pretende crear un sistema compuesto por un Arduino UNO al que se conectarán varios sensores, y configurarlos adecuadamente para poder obtener los datos tras la lectura de ambos. Una vez recibidos dichos datos, se pretende realizar publicaciones constantes hacia la nube, con el objetivo de que un cliente suscrito a esta pueda obtener el valor de los sensores en tiempo real. Estos datos se pretenden disponer en un panel de control a modo visual, desde una aplicación móvil que sea accesible en cualquier momento, mientras que el dispositivo presente conexión a la red de internet.

- El Arduino es conectado a su componente esencial en este proyecto, el *Ethernet Shield*, y a dos sensores, el MPU-6050 y NEO-6M.
- El segundo elemento es un servidor en la nube al cual se conectará el Arduino para compartir sus datos. Este servidor admite mensajes de publicación y suscripción.
- El último elemento es una aplicación usada para disponer los datos de los sensores leídos visualmente. Esta aplicación está suscrita al servidor al partir del cual le llega la información a mostrar.

Para llevar esto a cabo, Arduino dispone de una placa con puerto ethernet acoplable a la placa principal, y de otra placa, también acoplable, con tecnología Wi-Fi. Ambas placas soportan la lectura o escritura de una tarjeta SD, gracias a su puerto correspondiente. Si comparamos ambos dispositivos, llegamos a la conclusión de que la opción que más se adapta a nuestras necesidades es la placa con puerto Ethernet. Este dispositivo ofrece comunicación fiable por TCP presentando, a su vez, un rendimiento decente sin la aparición del llamado “cuello de botella”.

La obtención de datos de los sensores se realiza por comunicación I²C y UART. Su lectura se desarrolla mediante un código escrito en lenguaje C e implementado en el Arduino gracias a su software de código abierto, Arduino IDE.

El sensor MPU-6050 presenta un acelerómetro y un giroscopio de tres ejes cada uno, el cual permite gracias a su avanzada tecnología, añadir como entrada un compás de otros tres ejes para proporcionar una salida MotionFusion™ completa de nueve ejes.

El otro sensor es un módulo GPS capaz de proporcionar la longitud y latitud del dispositivo. Esto es posible gracias a una antena conectada al módulo mediante un conector U.FL.

Uso de CloudMQTT

Una vez obtenidos los datos de ambos sensores, el Arduino, anteriormente conectado a internet gracias a su puerto ethernet, comenzará una conexión con el servidor MQTT. Este elemento, llamado CloudMQTT, será el encargado de recibir la lectura de los sensores. Tras ello, enviará la información recibida a aquellos clientes del servidor que estén suscritos a ella. En este caso, el cliente que reclama información es la aplicación IoT MQTT Panel.

Disposición de los datos en *dashboard*

Los datos leídos a través del Arduino se representarán en un panel de control para una mejor gestión de los mismos.

Para la visualización de la temperatura obtenida con el MPU-6050, se utilizará un calibrador. De este modo, cuando la temperatura sobrepase los 33 grados Celsius, su color pasará a ser rojo, mientras que si la temperatura es aceptable estará en verde y finalmente en azul cuando baje de los 10 grados.

Para los ejes x, y, z del acelerómetro y del giroscopio, se utilizarán dos gráficos de puntos, cuyas medidas aparecerán en el eje y de la gráfica, y al eje x le corresponderá la hora de llegada de cada dato.

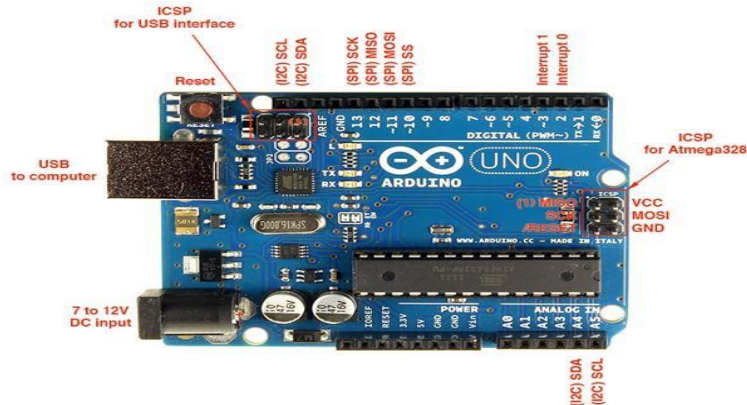
Finalmente, la ubicación ofrecida por el módulo NEO-6M vendrá dada a modo de dirección URI. Una vez se lance esta dirección, la aplicación móvil redirigirá al usuario a la aplicación de mapas instalada en el dispositivo, la cual mostrará la localización del módulo mediante una chincheta.

Recursos hardware

Arduino UNO

El Arduino Uno es una placa que incluye un microcontrolador de código abierto y presenta las siguientes características:

- Microcontrolador: ATmega328
- Voltaje Operativo: 5v
- Voltaje de Entrada (Recomendado): 7 – 12 v
- Pines de Entradas/Salidas Digital: 14 (De las cuales 6 son salidas PWM)
- Pines de Entradas Análogas: 6
- Memoria Flash: 32 KB (ATmega328) de los cuales 0,5 KB es usado por Bootloader.
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Velocidad del Reloj: 16 MHz



Arduino Ethernet Shield

Este dispositivo se conecta como un suplemento de Arduino UNO, añadiéndole así la capacidad de conexión a internet vía Ethernet y la posibilidad de manejo de una tarjeta de memoria SD, permitiendo su lectura y escritura. Para su comunicación con la placa principal a través de SPI (Interfaz Periférica en Serie), es necesario que los pines digitales del Arduino 10, 11, 12 y 13 no estén ocupados por otros dispositivos [10, 11].



Sensor MPU-6050

El sensor MPU-6050 permite combinar la aceleración y el movimiento de rotación más la información de rumbo en un único flujo de datos para la aplicación. El MPU-6050 es un dispositivo de seguimiento de movimiento de 6 ejes que combina un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un Procesador Digital de Movimiento (DMP). Gracias a ello se obtienen las medidas de los 3 ejes cartesianos, tanto para la aceleración como para el movimiento de rotación. Este dispositivo permite también obtener la temperatura en el instante de lectura.

La comunicación maestro/esclavo se realiza a partir de la interfaz I²C. Para ello, el sensor debe conectarse a los pines analógicos A4 (SDA) y A5 (SCL) del Arduino.



Sensor NEO-6M

Gracias a la antena GPS presente en este sensor, se puede obtener la latitud y longitud para así, posteriormente, hallar la ubicación del dispositivo.

La comunicación de este periférico con el Arduino se realiza mediante UART, siendo simplemente necesario para la obtención de datos la configuración de dos pines cualesquiera, como puerto de recepción y puerto de transmisión.



Recursos Software

A continuación, se comentan los recursos software envueltos en este proyecto.

MQTT Cloud

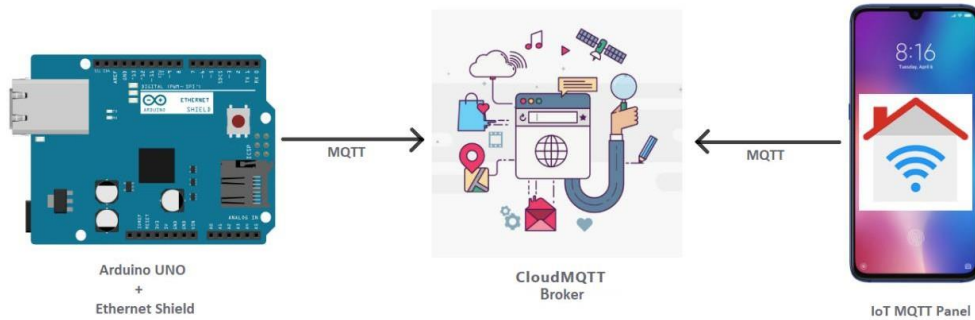
MQTT Cloud es un *broker* en la nube encargado de manejar las peticiones de clientes MQTT y publicar datos a través de servidores basados en Mosquitto. Los clientes se subscriben a un tema, o publican en el mismo, a través del *broker* y este les hace llegar los mensajes de los temas a los que están suscritos, una vez ha llegado nueva información.

IoT MQTT Panel

IoT MQTT Panel es una aplicación móvil que permite observar los datos leídos de los sensores gráficamente. Mediante la creación de paneles podemos tanto enviar datos como recibirlos fácilmente, a través del servidor en la nube conectado a nuestros dispositivos a través de internet.

Arduino IDE

El Entorno de Desarrollo Integrado (IDE) de Arduino es una aplicación para ordenador escrita en Java. Este software es usado para el desarrollo de código y su posterior carga como programa a los dispositivos Arduino compatibles [15].

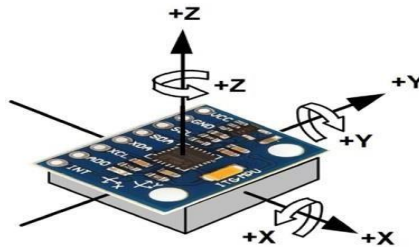


Sensores

La mayor parte de aplicaciones creadas para IoT se encargan de controlar sensores y recibir información crucial de los mismos. En este apartado se presentan los sensores usados en nuestro proyecto y se indaga en sus características más importantes.

MPU-6050

El MPU-6050 es un módulo dirigido a dispositivos con procesamiento de movimiento. Este dispositivo incluye una unidad de medición inercial o IMU (Inertial Measurement Units) de 6 grados de libertad (DoF) debido a la combinación de un acelerómetro y giroscopio de 3 ejes cada uno. Es capaz de procesar algoritmos de 9 ejes, y capturar movimiento en los ejes X, Y y Z a la misma vez.



Características del MPU-6050

MPU-6050 consiste en un Procesador Digital de Movimiento (DMP), el cual tiene la propiedad de resolver cálculos complejos. Este dispositivo también cuenta con un convertidor analógico-digital de 16 bits, gracias al cual es posible capturar movimiento de tres dimensiones al mismo tiempo.

A parte de tener integrado un acelerómetro y un giroscopio, el MPU-6050 también presenta un sensor de temperatura como característica distintiva.

La comunicación del módulo se realiza a través de I²C, esto le permite trabajar con la mayoría de los microcontroladores. I²C es un bus direccional de dos cables usado para enviar datos entre circuitos integrados. Los pines SCL y SDA tienen una resistencia pull-up en la placa para obtener una conexión directa al microcontrolador o Arduino. El pin ADDR presenta internamente una resistencia a tierra por lo que, si no se conecta, la dirección por defecto será 0x68 (si se conectase, la dirección correspondiente al módulo sería 0x69) [19].

Módulo GPS NEO-6M

El módulo GPS permite obtener la localización del dispositivo mediante dos valores, la latitud y la longitud. A parte de estos dos parámetros, se pueden conocer el número de satélites con los que tiene visión, así como la hora a la que recibió la información, entre otros.

Para entender cuál es el funcionamiento de este dispositivo, es necesario conocer cómo funciona un receptor GPS, ya que este módulo funciona principalmente como uno, gracias a su antena extraíble.

Receptores GPS

El funcionamiento de los receptores GPS se basa en averiguar cómo de lejos se encuentran de un número de satélites, ya que están preprogramados para saber dónde se encuentran los satélites de GPS en cualquier momento.

Los satélites transmiten información sobre su posición en el momento dado a través de radio señales hacia la Tierra. Esas señales identifican a los satélites y le dicen al receptor dónde se encuentran situados.

El receptor entonces calcula cómo de lejos se encuentra cada satélite dependiendo de cuánto tardaron en llegar las señales. Una vez que obtiene información de al menos tres satélites, puede encontrar la ubicación en la Tierra. Este proceso se conoce como Trilateración Satelital.

Características NEO-6M

El módulo NEO-6M GPS puede seguir hasta 22 satélites en 50 canales y conseguir el mayor nivel de sensibilidad del mercado. Es capaz de identificar ubicaciones en cualquier parte del mundo a la vez que presenta un bajo consumo de potencia.

A diferencia de otros módulos GPS, este es capaz de obtener 5 actualizaciones de ubicación por segundo, con una precisión de 2.5 metros con respecto a la posición horizontal. Además, sus 6 funciones de posicionamiento cuentan con un TTFF definido como el tiempo requerido para que un dispositivo de navegación GPS adquiera señales de satélite y datos de navegación, y calcule una solución de posición, de menos de 1 segundo.

Una de las mejores características que el chip posee es el modo de ahorro de energía (PSM). Este modo permite una reducción en el consumo de energía del sistema al encender y apagar selectivamente partes del receptor.

El módulo se comunica con el microcontrolador a través de la UART, admitiendo una velocidad de transmisión de 4800bps a 230400bps, con una velocidad de transmisión predeterminada de 9600.

Hay un LED presente en el módulo que se encarga de indicar si se ha obtenido comunicación con algún satélite. Parpadeará a diferente velocidad dependiendo del estado en el que se encuentre:

- Si el LED no parpadea entonces indica que está buscando satélites.
- Al parpadear cada segundo nos indica que el módulo puede ver suficientes satélites.

A pesar de todas sus funcionalidades integradas, NEO-6M necesita una antena para realizar cualquier tipo de comunicación, por lo que presenta un conector U.FL hembra al que conectarla.



Una vez conexionada la antena al módulo, será necesario que la antena tenga visión directa con los satélites, ya que no es capaz de obtener información de los satélites necesarios en espacios cerrados

Protocolo MQTT

La documentación oficial de este protocolo lo define como:

MQTT es un protocolo de transporte de mensajes *publish/subscribe* entre cliente y servidor. Es ligero, abierto, simple y está diseñado para ser fácil de implementar. Estas características lo convierten en una opción ideal en muchas situaciones, incluyendo entornos restringidos como la comunicación en contextos Máquina a Máquina (M2M) e Internet de las cosas (IoT), donde se requiere una pequeña huella de código y/o el ancho de banda de la red es muy importante [1].

Como hemos podido saber gracias a la definición anterior, MQTT es un protocolo estándar de mensajería e intercambio de datos para Internet de las cosas. Es un protocolo abierto, estandarizado por OASIS e ISO, el cual provee una forma escalable y rentable de conectar los dispositivos a través de internet. Este protocolo, binario y muy ligero gracias a su mínima sobrecarga de paquetes, sobresale frente a otros protocolos como HTTP cuando se trata del envío de datos a través de cables.

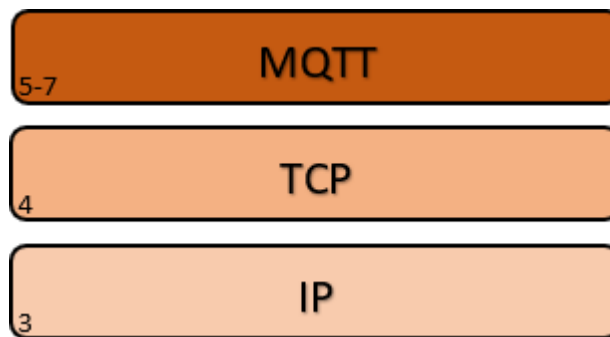
Un cliente MQTT es cualquier dispositivo que ejecute una librería MQTT y sea capaz de conectarse a un *broker* MQTT a través de internet. Por lo tanto, cualquier dispositivo que se comunique por MQTT a través de la pila de protocolos TCP/IP, puede ser llamado cliente MQTT. Tanto los editores como los suscriptores son clientes MQTT. A su vez, un mismo cliente es capaz de publicar y recibir mensajes.

El anterior llamado *broker*, es el elemento complementario al cliente MQTT, constatado como el elemento principal de cualquier protocolo de publicación/subscripción. Es el responsable de recibir todos los mensajes, filtrarlos, determinar quién está suscrito a cada mensaje y enviar los mensajes a aquellos clientes suscritos.

Para poder manejar los mensajes correctamente, el *broker* dispone de varias opciones de filtrado: basado en el tema, basado en el contenido o basado en el tipo. El filtro de mensajes usado por MQTT es el basado en un tema. Este tipo de filtrado se basa en un *topic* (tema) que es parte de cada mensaje. El cliente se suscribe al *broker* para estar al tanto de los *topics* de interés. A partir de ese momento, el *broker* se asegura de que el cliente obtiene todos los mensajes que han sido publicados en esos temas.

Conexión

El protocolo MQTT está basado en TCP/IP por lo que, tanto el cliente como el *broker* necesitan tener una pila con estas características.



Pila de protocolos de los dispositivos MQTT

Las conexiones de los clientes siempre son manejadas por el *broker*. Para iniciar una conexión, el cliente le envía un mensaje CONNECT, el cual responde con CONNACK y un código de estado, devolviendo 0 en caso de que la conexión haya sido aceptada.

CONTENIDO	CAMPO OPCIONAL	DESCRIPCIÓN
clientId	Obligatorio	Identificador para cada cliente MQTT que se conecta a un <i>broker</i> , el cual lo utiliza para identificar al cliente y su estado actual. Debe ser único por cliente y <i>broker</i> .

cleanSession	Obligatorio	Bandera para indicar al <i>broker</i> si el cliente quiere establecer o no una sesión persistente. Si la sesión es persistente, el <i>broker</i> almacenar todas las subscripciones del cliente y los mensajes perdidos para los que está suscrito con un QoS de nivel 1 o 2.
username	Sí	Para la autenticación del cliente se puede enviar una combinación de usuario y contraseña. Si esta información no se encuentra encriptada, la contraseña será enviada como texto plano.
username	Sí	
lastWillTopic	Sí	Este mensaje notifica a otros clientes cuando un cliente se desconecta sin delicadeza (con un mensaje DISCONNECT incorrecto). En él se indica un <i>topic</i> , una bandera de mensaje retenido, QoS y un <i>payload</i> . El <i>broker</i> almacena este mensaje hasta que detecta que el cliente se ha desconectado. Cuando esto ocurre, el <i>broker</i> envía el mensaje indicado a los clientes suscritos al lastWillTopic.
lastWillQoS	Sí	
lastWillMessage	Sí	
lastWillRetain	Sí	
keepAlive	Obligatorio	Intervalo en segundos indicado al <i>broker</i> para definir el máximo periodo de tiempo que el <i>broker</i> y el cliente pueden mantener una conexión sin enviar un mensaje.

Contenido mensaje CONNECT

CONTENIDO	CAMPO OPCIONAL	DESCRIPCIÓN
sessionPresent	Obligatorio	Bandera que indica al cliente si el <i>broker</i> ya tiene una sesión persistente disponible de anteriores interacciones entre ambos.

returnCode	Obligatorio	Código de retorno que indica si la sesión ha sido establecida o, en caso contrario, por qué no se ha podido establecer.
------------	-------------	---

Tabla 3: Contenido mensaje CONNACK

CÓDIGO	SIGNIFICADO
0	Conexión aceptada.
1	Conexión rechazada, la versión de protocolo no es aceptable.
2	Conexión rechazada, identificador rechazado.
3	Conexión rechazada, el servidor no se encuentra disponible.
4	Conexión rechazada, nombre de usuario o contraseña incorrectos.
5	Conexión rechazada, sin autorización.

Códigos de retorno del mensaje CONNACK

Una vez que la conexión ha sido establecida, el *broker* la mantiene abierta hasta que el cliente se desconecta o la conexión se pierde.

Si el mensaje CONNECT está mal formado o ha pasado demasiado tiempo entre la apertura del socket y el envío del mensaje, el *broker* cierra la conexión para evitar que haya clientes que ralenticen al servidor [6].

Modelo Publish/Subscribe

El modelo publicación/suscripción proporciona una alternativa a la arquitectura tradicional de cliente/servidor. En el modelo cliente/servidor, un cliente se comunica directamente con el extremo. El modelo pub/sub separa al cliente que envía el mensaje (publicador) del cliente o clientes que reciben el mensaje (suscriptores). Los publicadores y los suscriptores nunca se comunican entre ellos directamente. De hecho, puede que no estén al tanto de la existencia de los otros. La conexión entre ellos es manejada por un tercer componente, el *broker*. El trabajo del *broker* es filtrar todos los mensajes recibidos y distribuirlos correctamente a los suscriptores.

El aspecto más importante de esta arquitectura es la capacidad de dejar al margen al publicador en un mensaje enviado por el suscriptor.

En resumen, el modelo pub/sub elimina la comunicación directa entre el publicador del

mensaje y el receptor. La capacidad de filtrado del *broker* hace posible controlar qué cliente recibirá qué mensaje. Para publicar o recibir mensajes, los publicadores y subscriptores solo necesitan saber la dirección IP, o el nombre del servidor, y el puerto del *broker*.

Pub/Sub escala mejor que el enfoque tradicional de cliente/servidor. Esto se debe a que las operaciones en el *broker* pueden ser paralelizadas y los mensajes pueden procesarse de una manera controlada por eventos. El almacenamiento en caché de mensajes y el enrutamiento inteligente de mensajes son, a menudo, factores decisivos para mejorar la escalabilidad [4].

Existen cinco tipos de mensajes para la comunicación entre cliente y servidor: publish, subscribe, suback, unsubscribe y unsuback [5].

PUBLISH

Cada mensaje presenta un *payload* (relleno) que contiene los datos a transmitir en el formato que el cliente indique. El contenido de este mensaje se puede observar en la tabla 5.

CONTENIDO	DESCRIPCIÓN
packetId	Identifica de forma exclusiva un mensaje a medida que fluye entre el cliente y el <i>broker</i> .
topicName	Cadena simple que está estructurada jerárquicamente con barras diagonales como delimitadores.
qos	Indica el nivel de la calidad de servicio (QoS) del mensaje. Hay tres niveles: 0, 1 y 2. El nivel de servicio determina qué tipo de garantía tiene un mensaje de alcanzar el destino previsto.
retainFlag	Esta bandera define si el mensaje ha sido guardado por el <i>broker</i> como el último valor conocido para un <i>topic</i> específico. Cuando un cliente nuevo se suscribe a un <i>topic</i> , recibe el último mensaje almacenado para ese tema.
payload	Este campo es el contenido real del mensaje. Es posible enviar imágenes, texto en cualquier codificación, datos encriptados y cualquier dato binario.
dupFlag	Esta bandera indica que el mensaje es un duplicado y fue reenviado porque el destino previsto no asintió el mensaje original.

Cuando un cliente envía un mensaje al *broker* para ser publicado, este lee el mensaje, lo asiente, y lo procesa, de manera que determina qué clientes están suscritos al *topic* y les envía el mensaje.

SUBSCRIBE

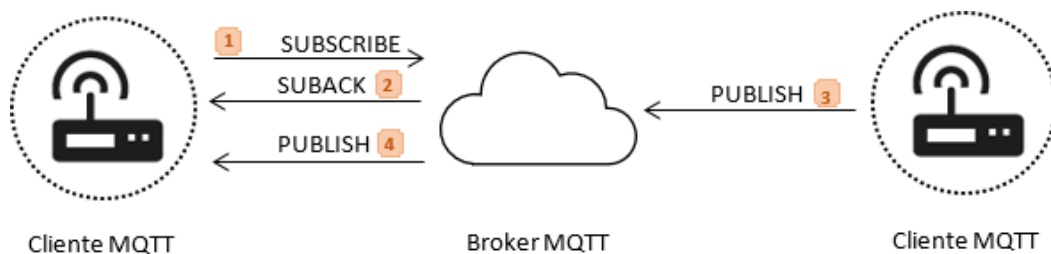
Para recibir mensajes de un tema, el cliente envía un mensaje de suscripción al *broker*. Este mensaje es muy simple, conteniendo un identificador de paquete y una lista de suscripciones.

CONTENIDO	DESCRIPCIÓN
packetId	Identifica de forma exclusiva un mensaje a medida que fluye entre el cliente y el <i>broker</i> . Este identificador es fijado por la librería del cliente o por el <i>broker</i> .
qos1 topic1 qos2 topic2 ...	Un mensaje SUBSCRIBE puede contener múltiples suscripciones para un cliente. Cada suscripción se realiza a partir de un <i>topic</i> y un nivel de QoS.

Tabla 6: Contenido mensaje SUBSCRIBE

SUBACK

Para confirmar cada suscripción, el *broker* envía un mensaje de asentimiento al cliente. Este mensaje contiene el identificador de paquete del mensaje original de suscripción y una lista de códigos de retorno, devolviendo 128 en caso de fallo.



Después de que un cliente envíe correctamente el mensaje de suscripción y reciba el asentimiento, obtendrá todos los mensajes publicados que coinciden con un tema en las suscripciones que contenía el mensaje SUBSCRIBE.

CONTENIDO	DESCRIPCIÓN
packetId	Identificador único usado para identificar a un paquete. Es el mismo que en el mensaje SUBSCRIBE.
returnCode1 returnCode2 ...	El <i>broker</i> envía un código de retorno por cada par de <i>topic</i> /QoS que recibe en el mensaje SUBSCRIBE.

Contenido mensaje SUBACK

CÓDIGO	SIGNIFICADO
0	Éxito, con un máximo QoS de 0.
1	Éxito, con un máximo QoS de 1.
2	Éxito, con un máximo QoS de 2.
128	Ha habido fallo para un <i>topic</i> específico.

Códigos de retorno del mensaje SUBACK

UNSUBSCRIBE

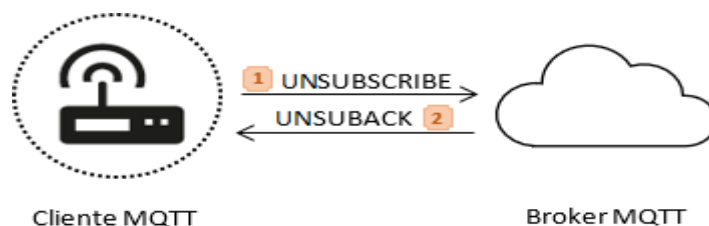
El mensaje complementario a SUBSCRIBE es el mensaje de UNSUBSCRIBE. Este mensaje elimina las suscripciones de un cliente existentes en el *broker*. Al igual que el mensaje de suscripción, este mensaje contiene un identificador de paquete y una lista de *topics*.

CONTENIDO	DESCRIPCIÓN
packetId	Identifica de forma exclusiva un mensaje a medida que fluye entre el cliente y el <i>broker</i> . Este identificador es fijado por la librería del cliente o por el <i>broker</i> .
topic1 topic2 ...	La lista de <i>topics</i> puede contener múltiples temas de los cuales el cliente quiere cancelar la suscripción. No es necesario enviar el QoS como en el mensaje SUBSCRIBE. El <i>broker</i> cancela la suscripción sin importar el nivel de servicio al que se encontraba originalmente suscrito el cliente.

Contenido mensaje UNSUBSCRIBE

UNSUBACK

Para confirmar la cancelación de la subscripción, el *broker* envía un mensaje de asentimiento al cliente. Este mensaje contiene el identificador original del mensaje UNSUBSCRIBE.



Paso de mensajes cancelación de subscripción

Plataforma CloudMQTT

CloudMQTT es un *broker* globalmente distribuido, constituido por servidores Mosquitto en la nube. Mosquitto implementa el protocolo MQTT que, como hemos visto anteriormente, provee métodos ligeros para el envío de mensajes usando el modelo de colas de publicación/subscripción.

Las colas de mensajes proveen un protocolo de comunicaciones asíncronas, gracias al cual el emisor y el receptor del mensaje no necesitan interactuar con la cola de mensajes al mismo tiempo. Los mensajes que son introducidos en la cola son guardados hasta que el receptor lo reclama o hasta que el mensaje expira.

El *broker* CloudMQTT usa números de puertos aleatorios que son generados por el *broker* en sí. Para conexiones sin seguridad, el número de puerto comienza por 1, para TLS/SSL en 2, y para soporte Websocket en 3.

El protocolo MQTT posee un campo de usuario y contraseña en el mensaje CONNECT para la autenticación. De esta forma, poder obtener comunicación con el *broker*, simplemente es necesario el nombre del servidor o dirección IP, un usuario y su contraseña.

Para comenzar a usar CloudMQTT es necesario registrarse y, tras ello, crear una instancia. Esta aplicación ofrece un plan de instancias gratuito, llamado Cute Cat, que es el que usaremos para este proyecto. El proveedor de los servidores es Amazon Web Services, con región en Virginia del Norte.

Una vez creada la instancia, el *broker* nos proporcionará todos los datos necesarios para que podamos conectarnos a él.

IoT MQTT Panel

Para una mejor comprensión de los datos recogidos, existen aplicaciones en las que exponer las lecturas de forma gráfica. De este modo, este apartado se centra en la presentación de la aplicación elegida para mostrarnos nuestros datos.

Dashboard

Cuando realizamos la lectura de los sensores, los datos son recogidos en la API de CloudMQTT, gracias a su función de Websocket. Aun así, su visualización no es fácil ya que obtenemos una lista de datos constantemente actualizándose.

Los paneles de control o *dashboards* permiten monitorizar el nivel de los sensores en tiempo real, ofreciendo la administración rápida y fácil de los mismos.

En un panel de control se pueden añadir *widgets* para la gestión de los sensores, tales como gráficas o botones, y crear alarmas para ser notificados una vez ha habido un cambio en alguno de ellos [20].

Aplicación

IoT MQTT Panel es una aplicación móvil a modo de panel de control que permite gestionar y visualizar proyectos de IoT basados en el protocolo MQTT. Esta aplicación permite la conexión al servidor tanto a través del protocolo TCP como de Websocket, usado especialmente en redes restringidas por firewall. Además, permite una comunicación segura gracias al soporte de certificados SSL.

Proporciona soporte JSON para los mensajes de publicación y suscripción. Aun así, para mantener la aplicación simple, todos los *payloads* están implementados como cadenas.

Los paneles están configurados para publicar o recibir datos automáticamente, por lo tanto, son actualizados en tiempo real. IoT MQTT Panel está diseñada no solo para visualizar los estados de los distintos sensores, sino para organizar las conexiones, mensajes y dispositivos, entre otros. Esta aplicación puede gestionar múltiples conexiones, cada una conectada a un *broker* distinto. De este modo, los múltiples dispositivos decada conexión son fácilmente separables, permitiendo una mejor gestión del sistema.

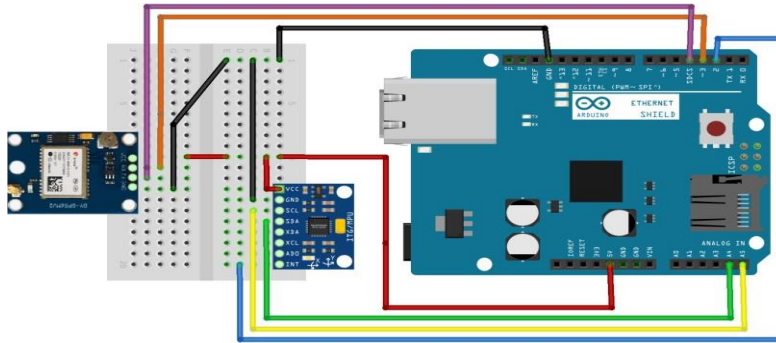
Para comenzar la comunicación con el *broker*, es necesario introducir los datos de la instancia creada en CloudMQTT. Tras ello, podremos crear los paneles necesarios para nuestro proyecto.

Desarrollo

Una vez indagados todos los componentes del sistema, se va a llevar a cabo el desarrollo de la aplicación realizada. Primero comenzaremos por la lectura de los sensores, para su posterior envío de datos al servidor, y finalmente, la recepción de los datos en la aplicación móvil.

Lectura y envío de datos de los sensores

Para proceder a la lectura de los datos proporcionados por los sensores, se han realizado dos programas codificados en c, a través del software Arduino IDE y las librerías necesarias. Además de estos programas, se dispone de un código conjunto en el que mediante una única conexión al *broker*, ambos sensores publicansus datos.



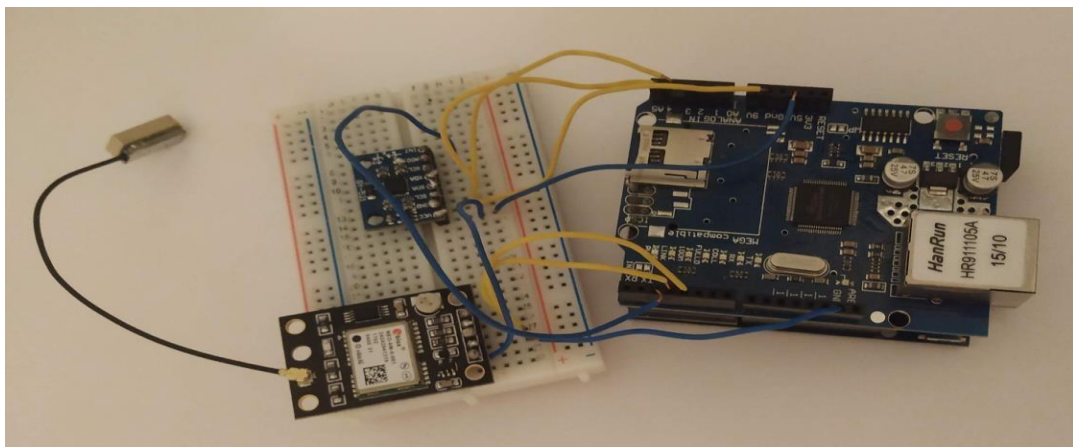
El *Ethernet Shield* provee a Arduino la capacidad de internet, pero esto no es suficiente para que el dispositivo disponga de conexión a internet. Para ello, se necesitan las librerías 'SPI' y 'Ethernet'. La primera de ellas es necesaria debido a que el módulo Ethernet se comunica con el Arduino por medio del bus SPI. La MAC del dispositivo es establecida aleatoriamente. En este caso, como solo presentamos un dispositivo Arduino, no es posible que haya conflicto entre direcciones MAC iguales. Para configurar la conexión, es necesario añadir una dirección IP estática por si la capacidad DHCP del dispositivo fallara. Además, es necesario proporcionar la IP del servidor DNS, así como el *gateway* y la dirección de la subred.

```
// Internet Configuration
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 100);
IPAddress dnServer(192, 168, 1, 1);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
```

Configuración acceso a internet

Una vez definidos los parámetros necesarios, podemos comenzar la conexión a internet mediante la siguiente instrucción:

```
Ethernet.begin(mac, ip, dnServer, gateway, subnet);
```



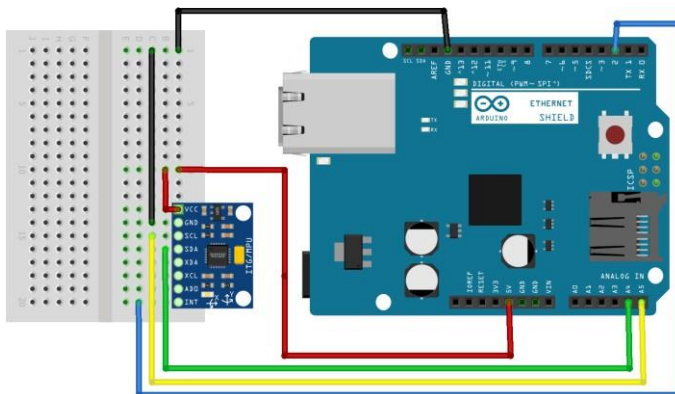
Conexión completa entre Arduino y sensores

En los siguientes subapartados se procede a la explicación de ambos programas, para una mayor profundización en cada sensor.

Sensor MPU-6050

El sensor MPU-6050 presenta funcionalidad de acelerómetro y giroscopio, a la vez que ofrece medidas de temperatura. Dispone de comunicación vía I²C, por lo que sus pines SDA y SCL van conectados a los puertos analógicos A4 y A5 respectivamente. Para que el Arduino realice la lectura de este sensor una vez haya habido cambios, se conectará el pin INT de interrupción al puerto 2 para que, de esta forma, se reduzca el consumo de potencia, evitando lecturas innecesarias del dispositivo [13].

Para que el Arduino pueda mantener comunicación por I²C, simplemente es necesario añadir la librería 'Wire'.



Esquema de conexión MP6050 con Arduino

La configuración del MPU se realiza comenzando la transmisión con la dirección del esclavo, 0x68. Tras esto, se envía un 0 al registro de gestión de la energía (0x6B) para despertar al dispositivo del modo *sleep*.

```
// MPU configuration
Wire.begin();
Wire.beginTransmission(MPU_ADDR); // Begins a transmission to the I2C slave (GY-521 board)
Wire.write(0x6B); // PWR_MGMT_1 register
Wire.write(0); // set to zero (wakes up the MPU-6050)
Wire.endTransmission(true);
```

Configuración MPU

Para empezar a leer los datos, se comienza con el registro 0x3B y a partir de este se obtienen 14 registros en total. A esta función se le denomina 'Burst Read'. De este modo, con una única petición obtenemos en orden la lectura de los tres ejes del acelerómetro, la temperatura, y los ejes del giroscopio.

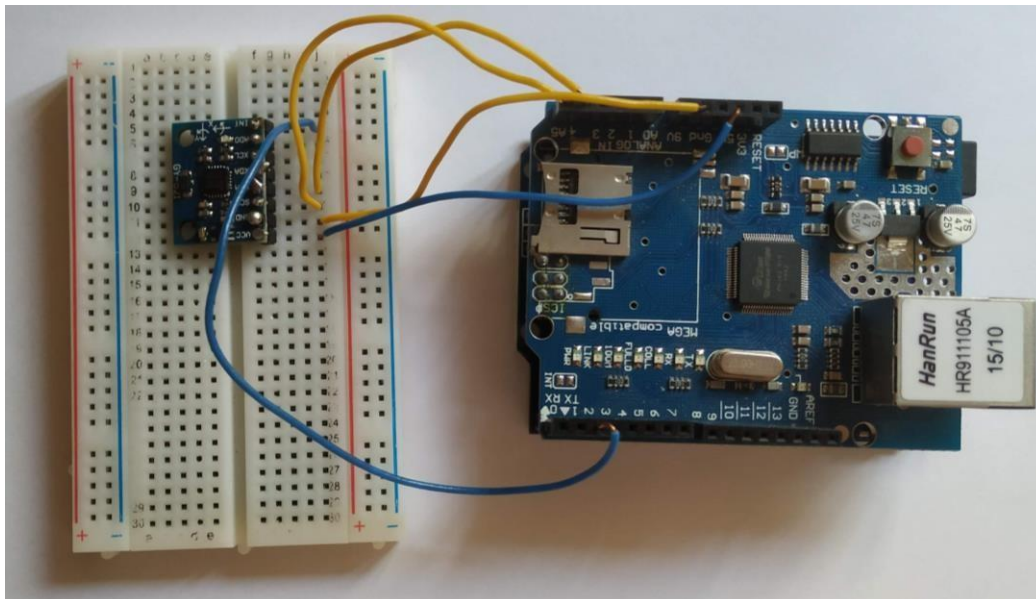

```
Wire.beginTransmission(MPU_ADDR);
Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false); // the parameter indicates that the Arduino will send a restart.
Wire.requestFrom(MPU_ADDR, 7*2, true); // request a total of 7*2=14 registers

// "Wire.read()<<8 | Wire.read();" means two registers are read and stored in the same variable
accelerometer_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x3B (ACCEL_XOUT_H) and 0x3C (ACCEL_XOUT_L)
accelerometer_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x3D (ACCEL_YOUT_H) and 0x3E (ACCEL_YOUT_L)
accelerometer_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x3F (ACCEL_ZOUT_H) and 0x40 (ACCEL_ZOUT_L)
temperature = Wire.read()<<8 | Wire.read(); // reading registers: 0x41 (TEMP_OUT_H) and 0x42 (TEMP_OUT_L)
gyro_x = Wire.read()<<8 | Wire.read(); // reading registers: 0x43 (GYRO_XOUT_H) and 0x44 (GYRO_XOUT_L)
gyro_y = Wire.read()<<8 | Wire.read(); // reading registers: 0x45 (GYRO_YOUT_H) and 0x46 (GYRO_YOUT_L)
gyro_z = Wire.read()<<8 | Wire.read(); // reading registers: 0x47 (GYRO_ZOUT_H) and 0x48 (GYRO_ZOUT_L)

temperature = temperature/340.00+36.53;
```

Lectura de datos MPU-6050

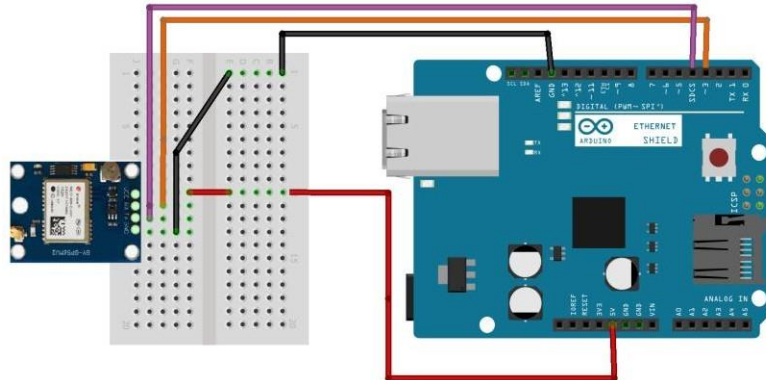
Los valores de los ejes de ambos instrumentos, así como de la temperatura, vienen dados por 16 bits en modo complemento a dos. Es por esto, por lo que se necesitan leer dos registros para obtener la lectura correcta de un dato. Estos registros son llamados alto o bajo, dependiendo de la posición final de sus bits [23].



Conexión MPU con Arduino

Módulo NEO-6M

El sensor NEO-6M es un módulo GPS que ofrece una localización a modo de longitud y latitud. Estos datos los recibe gracias a una pequeña antena conectada al dispositivo a través de un diminuto conector coaxial. La comunicación del módulo con el Arduino la realiza a través de UART.



Esquema de conexión NEO-6M con Arduino

Los puertos 3 y 4 del Arduino son configurados como transmisor y receptor, en ese orden [14]. De este modo, para poder obtener comunicación con el esclavo es necesario incluir la librería 'SoftwareSerial' e iniciar los puertos mencionados anteriormente, añadiendo la velocidad de transmisión del GPS, 9600 baudios.

```
static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);
```

Configuración NEO-6M

Una vez iniciados los puertos para la comunicación UART, comenzamos a recibir información a través del monitor serie presente en el IDE. La información enviada por el dispositivo sigue el protocolo NMEA (National Marine Electronics Association), de modo que los datos recibidos son sentencias estándares para la recepción de datos GPS.

```
while (ss.available() > 0)
    if (gps.encode(ss.read()))
        displayInfo();
```

Lectura de datos GPS

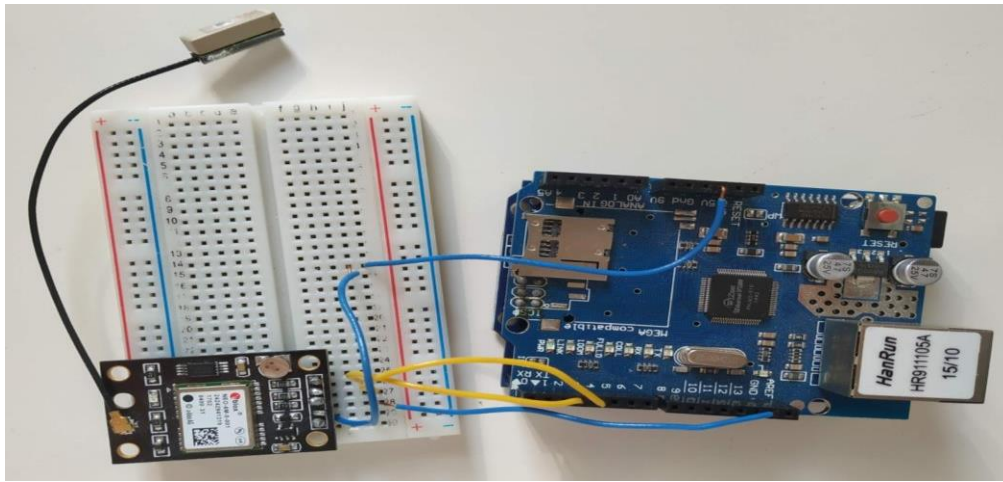
Estas sentencias tienen una longitud media de 70 caracteres, en la que la información se presenta separadas simplemente por comas. Debido a esto, obtener los datos se convierte en una tarea tediosa. Para ello, incluiremos la librería 'TinyGPS++', la cual se encarga de facilitarnos la substracción de los distintos campos a tener en cuenta.

```
LAT_deg = gps.location.rawLat().deg;
LAT_min = gps.location.rawLat().billionths;
LAT_coord = gps.location.rawLat().negative ? "-" : "+";

LONG_deg = gps.location.rawLng().deg;
LONG_min = gps.location.rawLng().billionths;
LONG_coord = gps.location.rawLng().negative ? "-" : "+";
```

Tratamiento de datos GPS para la localización

Los campos que hemos necesitado tanto para la latitud como para la longitud han sido los grados de ambos, con 6 decimales de precisión, y el signo de ellos, para poder hallar si se trata de norte o sur. Gracias a estos datos se consigue obtener la localización completa del dispositivo.

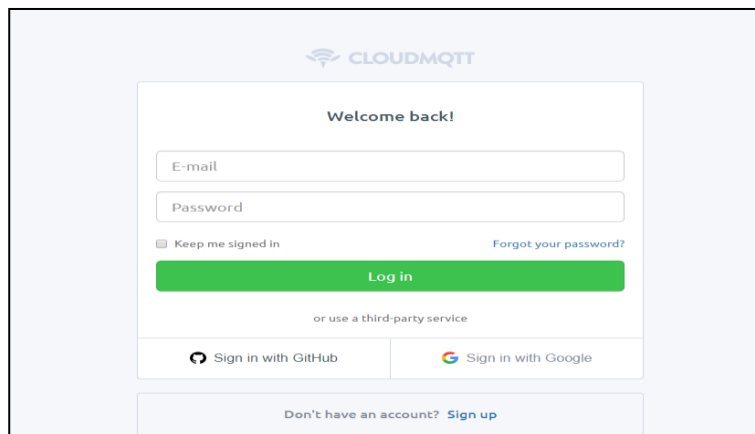


Conexión Arduino con GPS

Broker

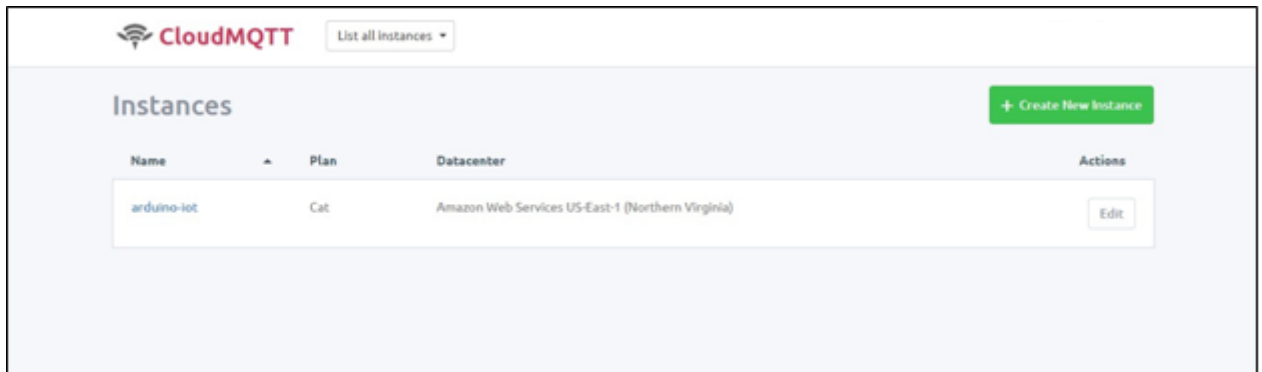
Para que los clientes puedan comunicarse con el servidor, es necesaria la creación de una instancia, la cual ofrece el nombre del servidor, los puertos de conexión y distintos datos para la seguridad de la comunicación.

Para obtener una instancia del *broker* MQTT debemos acceder a la página oficial de CloudMQTT y registrarnos, si no lo hemos hecho anteriormente. También es posible acceder con una cuenta de GitHub o Google, por lo que en nuestro caso utilizaremos el acceso a través de la plataforma GitHub.

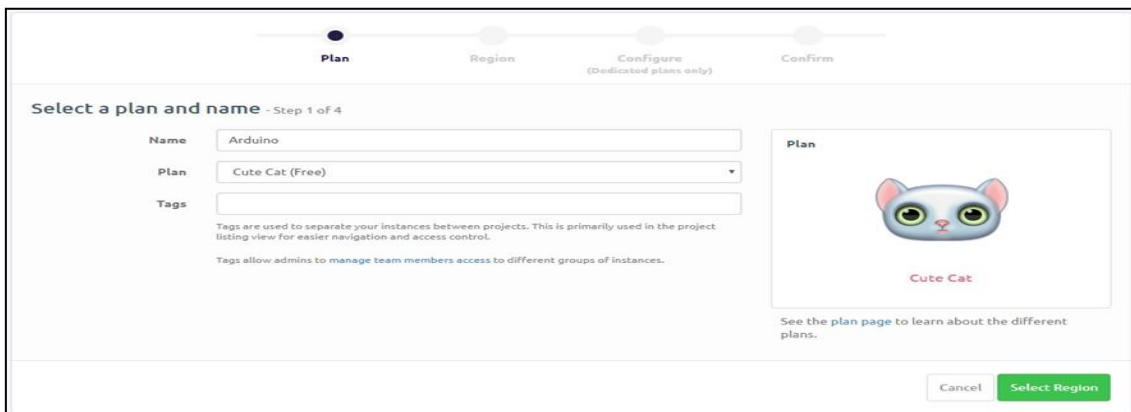


Página de acceso a CloudMQTT.

Tras acceder, nos encontraremos en la página principal de nuestra cuenta, en la que se muestra una lista de nuestras instancias creadas.



Para proceder a crear una nueva instancia, pulsaremos sobre el botón verde de la esquina superior derecha, que indica 'Create New Instance'.



Esta plataforma dispone de distintos planes de subscripción dependiendo de las características del servidor que se vayan buscando. En nuestro caso, el plan Cute Cat es adecuado para nuestras necesidades ya que permite la conexión de hasta 5 clientes y es gratuito.

Una vez seleccionada una región para los servidores de AWS, podremos confirmar la instancia y obtener toda la información de esta.

Información de la instancia creada en CloudMQTT

La comunicación de ambos sensores con el *broker* se lleva a cabo gracias a la librería 'PubSubClient'. Esta librería permite la conexión de un cliente MQTT con el servidor para el posterior envío/recepción de mensajes.

```
// Ethernet and MQTT related objects
EthernetClient ethClient;
PubSubClient mqttClient(ethClient);
```

Creación cliente MQTT

Para la configuración, se crea un cliente MQTT a partir de un cliente Ethernet. Con este cliente, se establece el servidor mediante el *hostname* y el puerto indicado en la instancia. Tras esto, ya es posible conectarse al *broker*.

```
const char* server = "postman.cloudmqtt.com";
mqttClient.setServer(server, 11318);

// Attempt to connect to the server with the ID "dolraairom", and instance's user and password.
while(!mqttClient.connect("dolraairom", "pbpsqoco", "lfsizmmJ2K4R")){
  Serial.println("Attempting to connect");
  delay(1000);
}
Serial.println("Connection has been established, well done");
mqttClient.setCallback(subscribeReceive);
```

Conexión al *broker* MQTT desde el Arduino

La conexión al servidor se llevará a cabo mediante la autenticación usuario-contraseña indicados, de nuevo, en la instancia. El campo principal de la conexión es un identificador de cliente con el que se identifican los paquetes de publicación/suscripción. El resto de los campos obligatorios indicados en la tabla 2, se encuentran definidos en la función 'connect', tomando los valores por defecto de 15 segundos para 'keepAlive' y true para 'cleanSession'.

Si la conexión ha sido exitosa, es necesario establecer un *callback* al que acudiré el programa tras recibir algún evento. En nuestro caso, esta función simplemente se encarga de imprimir al monitor.

Para proceder a la publicación de los datos, es necesario estar suscrito con anterioridad al tema al que se quiere enviar la información.

```
mqttClient.subscribe("map");
mqttClient.subscribe("temperature");
mqttClient.subscribe("acc_x");
mqttClient.subscribe("acc_y");
mqttClient.subscribe("acc_z");
mqttClient.subscribe("gyro_x");
mqttClient.subscribe("gyro_y");
mqttClient.subscribe("gyro_z");
```

Suscripción a los *topics* desde el Arduino

De esta forma, se van a necesitar 8 *topics* a los que suscribirse:

- MPU-6050
 - acc_x : eje x acelerómetro.

- acc_y : eje y acelerómetro.
- acc_z : eje z acelerómetro.
- temperature : temperatura dada por el acelerómetro.
- gyro_x : eje x giroscopio.
- gyro_y : eje y giroscopio.
- gyro_z : eje z giroscopio.
- NEO-6M
 - map : localización obtenida por el GPS.

Los datos enviados al *broker* deben ser cadenas, por lo que las lecturas del MPU son convertidas a tipo *String* antes de ser publicadas al *topic*.

```
// Attempt to publish a value to the topic "temperature"
if (mqttClient.publish("temperature", convert_int16_to_str(temperature))){
    Serial.println("Publish message success");
}
else{
    Serial.println("Could not send message :(");
    while(!mqttClient.connect("dolrairrom", "pbpsqoco", "lfzizmmJ2K4R")){
        Serial.println("Attempting to reconnect");
    }
}
```

Publicación de la temperatura obtenida en el *topic* 'temperature'

Presentación de datos

Una vez que los datos son recogidos por el *broker*, todos los clientes suscritos a dichos *topics* obtendrán la información conforme vaya llegando al servidor.

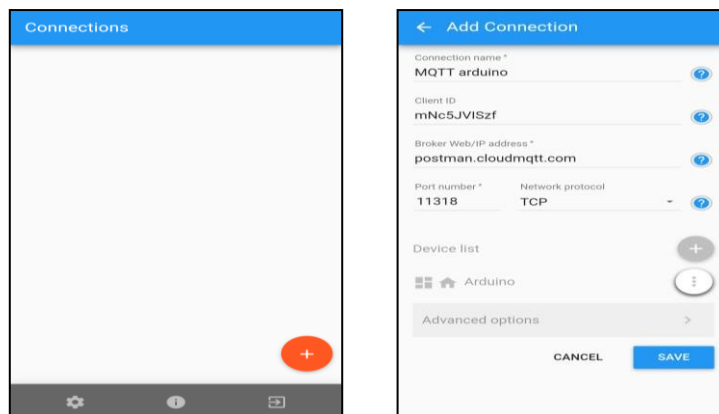
La aplicación IoT MQTT Panel es un cliente MQTT que se encuentra suscrito a los *topics* definidos en el apartado anterior, y los presenta de forma visual. Para ello, la aplicación dispone de una larga lista de *widgets* de la cual se han elegido los siguientes:

- Gauge: la temperatura variará desde un mínimo de 0 a 50 grados con diferencia de color según el tramo en el que se encuentre.
- Gráfico: los ejes del acelerómetro, como los del giroscopio se disponen en dos gráficos, dentro de los cuales se diferencian tres

gráficas de puntos, una por cada eje.

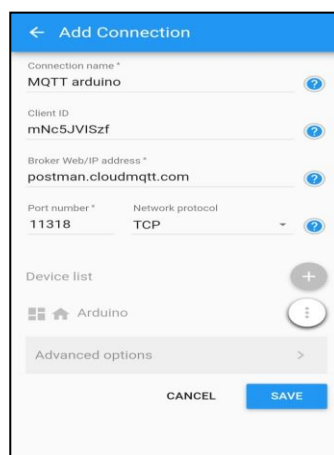
- URI: la URL enviada al *topic* 'map' podrá ser lanzada a través de este *widget* una vez recibida.

Esta aplicación se puede obtener gratuitamente en Google Play. Una vez instalada, hay que crear una nueva conexión al *broker* en cuestión. Para ello se pulsa sobre el círculo naranja en la esquina inferior derecha y nos lleva a la configuración del servidor al que conectarnos.



Página principal IoT MQTT Panel y nueva conexión

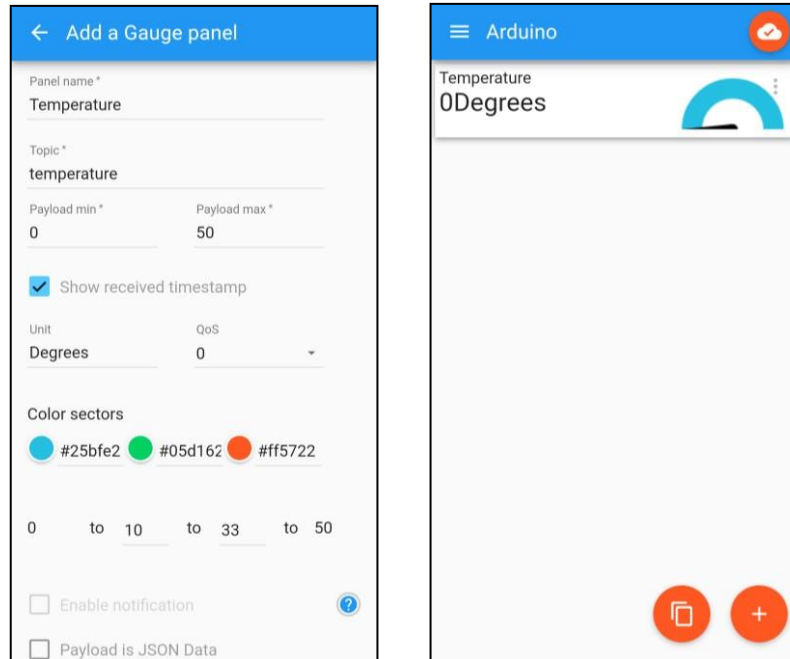
Los campos se rellenan con los datos recogidos en la instancia de CloudMQTT. En opciones avanzadas es necesario proporcionar el nombre del usuario y contraseña con el que acceder a la instancia.



Configuración avanzada de la conexión

Una vez conectados al *broker* podemos comenzar a añadir los *widgets* necesarios para la visualización de los datos. Para ello, pulsaremos sobre el círculo naranja que aparece tal y como el de la ilustración.

Comenzaremos por el *widget* 'gauge', para el *topic* 'temperature'.



Creación Panel Gauge

El acelerómetro y giroscopio se disponen en dos gráficos de líneas, con la misma configuración, exceptuando el nombre de los *topics*.

← Add a Line Graph panel

Panel name*
Accelerometer

X axis divisor
5

No of persistence
10

Delete this graph
Topic for graph 1*
acc_X

Label for graph 1
X axis

Chart color
#cad800

☐ Show area ☒ Show points

☐ Enable notification

☐ Payload is JSON Data

Delete this graph

← Add a Line Graph panel

Topic for graph 2*
acc_y

Label for graph 2
Y axis

Chart color
#d24d7b

☐ Show area ☒ Show points

☐ Enable notification

☐ Payload is JSON Data

Delete this graph
Topic for graph 3*
acc_z

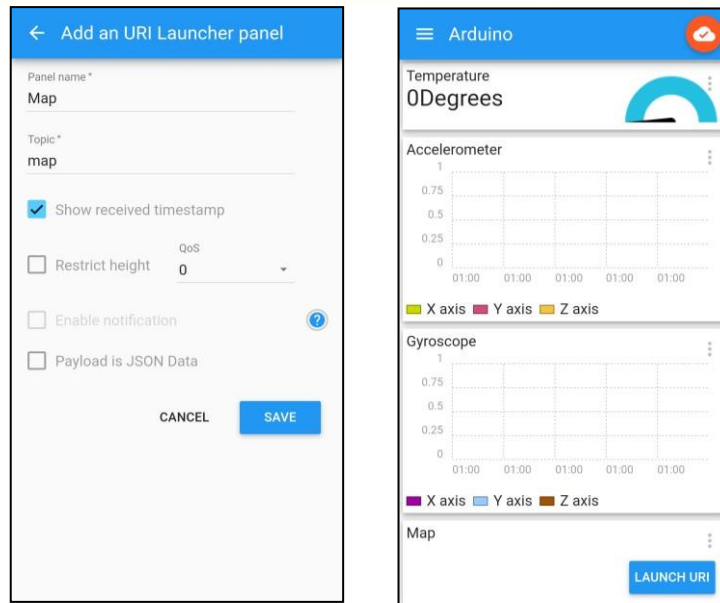
Label for graph 3
Z axis

Chart color
#f4c63d



Configuración panel
gráficos

El último panel está dedicado al GPS, con una configuración muy sencilla. Este *widget* se encarga de lanzarla URL recibida al *topic* 'map'.



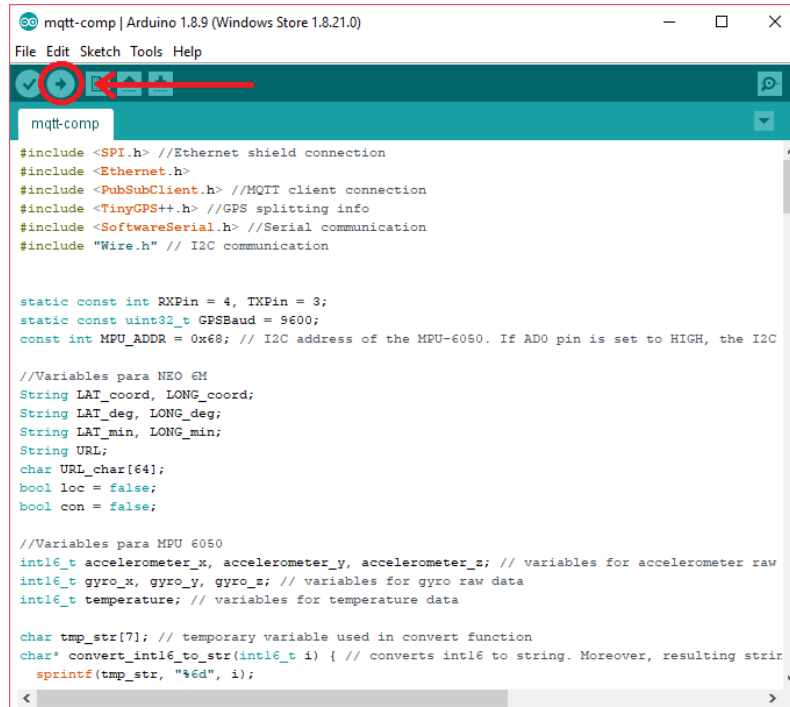
Configuración panel URI y *dashboard* final.

El siguiente y penúltimo capítulo, muestra un ejemplo de ejecución del proyecto realizado y los resultados obtenidos tras él.

Ejecución

El Arduino presenta una entrada para la carga de tensión, a la cual conectaremos un cable USB con puerto final el ordenador indicado en el subapartado 3.1.1. Ambos módulos se deben conectar a Arduino según la conexión especificada en el capítulo 4, como podemos comprobar en las ilustraciones 10 y 12. Además de esta conexión, es necesario que exista acceso a internet, por lo que debemos conectarle al módulo *EthernetShield* un cable Ethernet desde un *router* o desde algún puerto con conexión directa a la red.

Tras ello, abriremos el software Arduino IDE, gracias al cual podremos cargar el programa en el dispositivo.



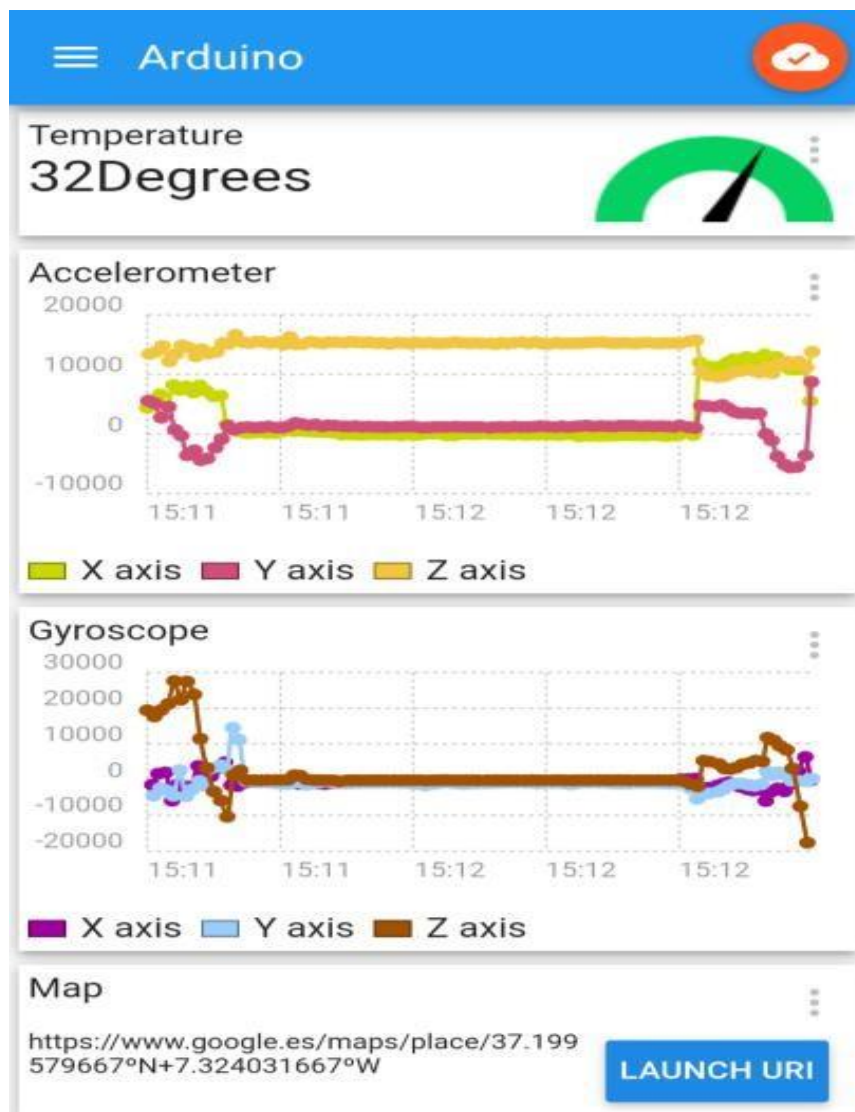
Cargar programa en el Arduino

Para que el módulo GPS sea capaz de comunicarse con los satélites, debe tener visión directa con ellos, por lo que es necesario que la antena se encuentre en el exterior.

La instancia se encuentra activa en todo momento, por lo que no es necesario iniciarla. Lo único necesario para una correcta comunicación, es que el cliente se haya conectado al servidor antes de enviar/recibir los datos.

Resultados

Una vez cargado el programa en el Arduino, simplemente es necesario esperar a que haya una conexión estable a internet, y que el cliente se haya conectado al *broker* con éxito. Tras ello, comenzaremos a recibir datos por parte del MPU-6050, y del NEO-6M una vez que haya encontrado los satélites suficientes para una localización correcta. Tras ello, podremos visualizar en nuestra aplicación móvil los datos de los *topics* a los que estamos suscritos.



Datos recogidos en la aplicación móvil

Como podemos comprobar en la ilustración 40, en un instante de las 15:11 y de las 15:12 horas, tanto el acelerómetro como el giroscopio han recogido movimiento inusual. Esto se debe a que en ese instante el dispositivo fue desplazado con velocidad y girado en el mismo momento. La temperatura como vemos se mantiene en 32 grados. Finalmente, el módulo GPS ha enviado una localización que se puede lanzar mediante el botón 'LAUNCH URI'.



Localización obtenida gracias al módulo NEO-6M

En la ilustración es posible visualizar la ubicación del dispositivo gracias a la aplicación de mapas lanzada. En el borde superior de la misma observamos la latitud y la longitud recogidas por el módulo, y en el borde inferior esos mismos valores convertidos a grados y minutos. Gracias a esta información, obtenemos en el mapa una marca color roja llamada 'chincheta' que nos indica el punto exacto al que se refieren los datos recogidos.