

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL  
Proyecto Fin de Grado



Sistema de geolocalización de autobuses mediante tecnología LoRaWAN

Índice

[Memoria 1](#_Toc1)

[1.1 Objeto 1](#_Toc2)

[1.2 Alcance 1](#_Toc3)

[1.3 Antecedentes 1](#_Toc4)

[1.3.1 Motivación 1](#_Toc5)

[1.3.2 Historia e introducción 2](#_Toc6)

[1.4 Normas y referencias (HACER) 4](#_Toc7)

[1.4.1 Programas de cálculo 4](#_Toc8)

[1.5 Definiciones y abreviaturas 4](#_Toc9)

[1.6 Requisitos de diseño (NO ME CONVENCE) 7](#_Toc10)

[1.7 Análisis de soluciones 7](#_Toc11)

[1.7.1 El microcontrolador y la tecnología IoT 7](#_Toc12)

[1.7.2 Programación 9](#_Toc13)

[1.7.3 La red intermedia 9](#_Toc14)

[1.7.4 La base de datos 9](#_Toc15)

[1.7.5 El agente de servidor 10](#_Toc16)

[1.7.6 Representación de la posición 10](#_Toc17)

[1.8 Resultados finales 11](#_Toc18)

[1.8.1 Resumen de la comunicación 11](#_Toc19)

[1.8.2 Programa de adquisición y envío de datos SEGUIR EXPLICANDO LO DEL FAIR USE, DR, DUTY CYCLE, AIR TIME, ETC. AÑADIR CAMBIOS EN EL CODIGO 12](#_Toc20)

[1.8.2.1 Setup inicial. Frecuencia y modo de envío 12](#_Toc21)

[1.8.2.2 El programa 17](#_Toc22)

[1.8.3 Envío a la TTN 22](#_Toc23)

[1.8.4 Implementación del protocolo MQTT 25](#_Toc24)

[1.8.5 Primeras pruebas 26](#_Toc25)

[1.8.6 Conexión a través de Telegraf 28](#_Toc26)

[1.8.7 Configuración y puesta en marcha de InfluxDB 28](#_Toc27)

[1.8.8 Representación y configuración de Grafana 30](#_Toc28)

[1.8.9 Programa de cálculo del tiempo de llegada 35](#_Toc29)

[1.8.10 Implementación física 38](#_Toc30)

[Bibliografía METER LO DEL SEMIVERS4ENO, LO DE GTSF 39](#_Toc31)

[Anexos 40](#_Toc32)

[1. Datasheet Seeeduino LoRaWAN 40](#_Toc33)

Índice de imágenes

[Ilustración . Microcontrolador TMS 1000 de Texas Instruments 1](#_Toc1)

[Ilustración . Logotipo de la marca Arduino 2](#_Toc2)

[Ilustración . Representación del "ecosistema IoT", donde todos los dispositivos comparten información entre ellos en una red 3](#_Toc3)

[Ilustración . Representación de cómo se calcula la posición del receptor GPS con 3 satélites 4](#_Toc4)

[Ilustración . Estructura de la red LoRaWAN 5](#_Toc5)

[Ilustración . Seeeduino LoRaWAN, elección final para realizar el proyecto 7](#_Toc6)

[Ilustración . Logo de Arduino, fabricante de placas del mismo nombre 8](#_Toc7)

[Ilustración . Logo de InfluxDB, elección final para la base de datos 9](#_Toc8)

[Ilustración . Logo de Telegraf, el intermediario entre InfluxDB y la TTN 9](#_Toc9)

[Ilustración . Logo de Grafana, el visualizador de datos usado 10](#_Toc10)

[Ilustración . Diagrama de bloques que representa el funcionamiento del sistema de comunicación 10](#_Toc11)

[Ilustración . Entorno de desarrollo Arduino 11](#_Toc12)

[Ilustración . Frecuencias disponibles para usar con el Seeeduino LoRaWAN 15](#_Toc13)

[Ilustración . Medidas incorrectas al inicio del programa 20](#_Toc14)

[Ilustración . Página de inicio de la web The Things Network 20](#_Toc15)

[Ilustración . Configuraciones del End device. 21](#_Toc16)

[Ilustración . Parámetros de configuración finales del dispositivo en la TTN 22](#_Toc17)

[Ilustración 18. Interfaz de conexión del programa MQTT Explorer 23](#_Toc18)

[Ilustración 19. Topic donde se publican los mensajes que lleguen a la TTN 23](#_Toc19)

[Ilustración 20. Interfaz de MQTT Explorer mostrando el último mensaje recibido 24](#_Toc20)

[Ilustración . Monitor serie del IDE de Arduino mostrando las configuraciones 24](#_Toc21)

[Ilustración . Mensajes del Seeeduino sobre su conexión con la red 25](#_Toc22)

[Ilustración . Mensajes de conexión del Seeeduino con la TTN, visto desde la web 25](#_Toc23)

[Ilustración 24. Interfaz de bienvenida de InfluxDB 26](#_Toc24)

[Ilustración 25. Ventana de visualización de datos de InfluxDB 27](#_Toc25)

Índice de tablas

# Memoria

## Objeto

El objetivo de este proyecto es crear un sistema de geolocalización de autobuses mediante tecnología IoT, de forma que se pueda hacer un seguimiento casi en tiempo real de la posición del vehículo. Mediante esta información y conociendo las paradas habituales del autobús, se busca obtener una estimación del tiempo de llegada a cada parada.

Se tendrá en cuenta la normativa vigente sobre telecomunicaciones para la transmisión de la información.

La idea principal de este proyecto, es hacer que la información de la localización del vehículo sea accesible por el usuario promedio, de forma que cualquiera pueda conocerla.

## Alcance

El alcance consistirá en la programación de un microcontrolador con GPS y tecnología IoT, como LoRa o LoRaWAN, junto con la gestión tanto de una base de datos como del envío de estos. Adicionalmente, se creará una interfaz con la que visualizar los datos obtenidos.

Se busca principalmente la precisión en la medida de la localización del vehículo, y no se profundizará en la eficiencia energética. En cuanto a la instalación física del aparato, se discutirán diferentes posibilidades, pero sin profundizar en ello.

## Antecedentes

* + 1. Motivación

Como estudiante, he tenido que utilizar el transporte público casi a diario, de forma que he podido comprobar de primera mano tanto sus beneficios como sus inconvenientes y problemas, siendo la principal desventaja de la utilización de transporte tipo autobuses, el desconocimiento de si el vehículo se ha retrasado, está estropeado, ha pasado ya…

Si bien es cierto que existen aplicaciones que buscan solucionar este problema, como puede ser “BusKBus Algeciras”, disponible tanto para dispositivos Android como para IOS, esta solo proporciona información sobre los itinerarios y horarios de cada línea urbana. Por lo tanto, un residente de Los Barrios no puede valerse de esta para obtener información sobre la línea entre Los Barrios y Algeciras.

Además de esa, existe una famosa aplicación también para dispositivos móviles, “Moovit”. Esta app incluye tanto planificación de viajes como horarios del transporte público, así como su propio sistema de geolocalización. Sin embargo, esta no permite visualizar la localización de los vehículos de forma gratuita, además de que sus servicios no están disponibles en varias zonas, como es la del Campo de Gibraltar.

Con este proyecto, se tratará de realizar una mejora a las redes de transporte públicos, permitiendo a los usuarios tener un mayor conocimiento de la situación del vehículo, de forma que puedan tomar una decisión acorde.

* + 1. Historia e introducción

Este proyecto involucra el uso y programación de microcontroladores, por lo que es conveniente hacer un breve repaso a la historia de estos y a su evolución, así como una introducción a los mismos.

¿Qué son los microcontroladores? Esencialmente, son un ordenador en miniatura reducido a un pequeño chip integrado. Poseen las piezas clave para considerarse como tal: una unidad de procesamiento, memoria y periféricos de entrada/salida. Es importante diferenciarlos de los microprocesadores, que son los chips utilizados en los ordenadores convencionales. Estos solo son capaces de procesar la información, y necesitan de componentes externos para funcionar, tales como la memoria. Los dispositivos objetivo de este proyecto tienen todos esos componentes integrados, lo que permite utilizarlos sin necesidad de componentes extra. Un ejemplo de uno de los primeros fabricados es el de la Ilustración 1.



Ilustración . Microcontrolador TMS 1000 de Texas Instruments

Estos se diseñan para reducir los costos económicos y energéticos de un sistema, ya que suelen tener prestaciones reducidas. Se elige el chip indicado en función de la tarea que se quiera realizar. El control de un sistema sencillo como un sensor de temperatura simple, puede realizarse con un microcontrolador poco potente pero muy económico, de forma que se aprovechen al máximo los recursos de este.

Debido a esto, se fabrican sin datos en la memoria. El usuario que lo adquiera debe encargarse de realizar su programación para la aplicación especifica en la que lo quiera usar.

Antes de ver el estado actual de estos componentes, repasemos un poco su historia. En 1971 se lanzó el primer microprocesador de la historia, el Intel 4004. Estos, como se mencionó anteriormente, al ser microprocesadores requerían de componentes externos para funcionar, elevando el costo. Y fue en ese mismo año cuando los ingenieros de Texas Instruments, empresa de electrónica mundialmente conocida hoy en día, crearon el primer microcontrolador, el TMS 1000 (Ilustración 1). Este combinaba memoria RAM, ROM, microprocesador y reloj, y estaba orientado a sistemas embebidos (circuitos dedicados a una función concreta).

Años más tarde, en el 77, Intel desarrolló su primer microcontrolador, el Intel 8048. En su día era usual encontrarlos en teclados de IBM PC compatible (un tipo de ordenadores clones del IBM PC, lanzado por la empresa homónima IBM en los 80 como respuesta al Apple Computer). En esa época, el Intel 8048 fue uno de los productos más exitosos de la compañía.

Hasta este punto, los microcontroladores en el mercado eran de dos tipos: los que tenían memoria EPROM y los que tenían PROM. Los del segundo tipo solo permitían la programación de la memoria una vez, volviendo el programa insertado en la memoria inalterable. Los primeros, sin embargo, bastante más costosos que los segundos, permitían el borrado mediante luz ultravioleta.

No fue hasta 1993 que se comercializaron los primeros con memoria EEPROM, que permitían el borrado de la memoria de forma eléctrica, facilitando mucho su programación además de reducir el coste. Ese mismo año, Atmel lanza al mercado el primer microcontrolador con memoria flash, un tipo de memoria como la EEPROM, pero superior en velocidad. [1]

Con el tiempo, los precios se fueron reduciendo más y más, a le vez que se facilitaba el trabajo con estos dispositivos. Y es gracias al proyecto Arduino que hoy en día cualquiera puede adquirir un microcontrolador, desde aficionados buscando realizar proyectos personales hasta profesionales implementando una de las placas de Arduino en una empresa.

Iniciado en 2005, el proyecto Arduino se enfoca en la creación de placas de desarrollo para microcontroladores libres, es decir, cualquiera puede acceder a los diseños de las placas e incluso modificarlas.



Ilustración . Logotipo de la marca Arduino

Estas placas (también conocidas como shields) no son más que un microcontrolador central con una serie de componentes externos que facilitan su programación, así como su conexión con otros periféricos externos.

Gracias a este tipo de iniciativas, la programación de microcontroladores se ha facilitado mucho, no solo por las propias mejoras en el hardware, sino también por la creación de software dedicado a facilitar la implementación del código en las placas, así como para la conexión con periféricos externos tales como pantallas, sensores, actuadores, etc.

Hoy en día, los microcontroladores están muy extendidos, y con el auge del IoT y la domótica, cada vez existen más aplicaciones personales y profesionales que usan este tipo de placas.

## Normas y referencias (HACER)

<https://support.moovitapp.com/hc/es>

<https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_regional_parameters_v1.0.3reva_0.pdf>

<https://avance.digital.gob.es/espectro/informacion/interfaces/Documents/Resolucion-271-282.pdf>

ERC/REC 70-03 y ETSI EN 304 220-1

UNE 157001

* + 1. Programas de cálculo

El software utilizado para la realización del proyecto ha sido principalmente Arduino IDE, con el que se realizó la programación del microcontrolador. Por otro lado, Se usó InfluxDB para la base de datos y Grafana para la representación de estos

## Definiciones y abreviaturas

* **IoT**: [Según IBM](https://www.ibm.com/es-es/topics/internet-of-things): “El Internet de las cosas (IoT) se refiere a una red de dispositivos físicos, vehículos, electrodomésticos y otros objetos físicos que están integrados con sensores, software y conectividad de red que les permite recopilar y compartir datos.”. Esto, sumado al uso de actuadores y al análisis del *big data*, permite tanto a particulares como a empresas modificar el comportamiento de estos, mejorando la experiencia de usuario y aumentando la eficiencia

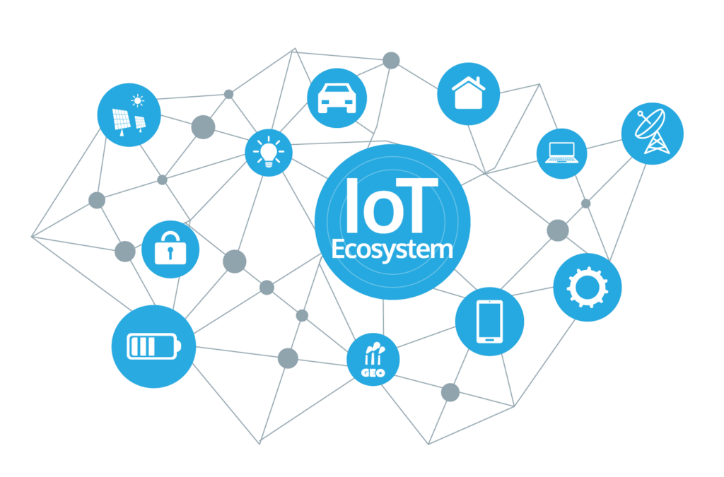


Ilustración . Representación del "ecosistema IoT", donde todos los dispositivos comparten información entre ellos en una red

* **GPS**<https://www.geotab.com/es/blog/que-es-gps/>: El Sistema de Posicionamiento Global (GPS) es un sistema de navegación que se basa en el uso de satélites y algoritmos para calcular datos tanto de posición como de velocidad de los dispositivos receptores. Para ello se utilizan 4 satélites, 3 de ellos aportan la posición del receptor, mientras el cuarto valida la información y especifica la altitud. [Fuente](https://www.geotab.com/es/blog/que-es-gps/)

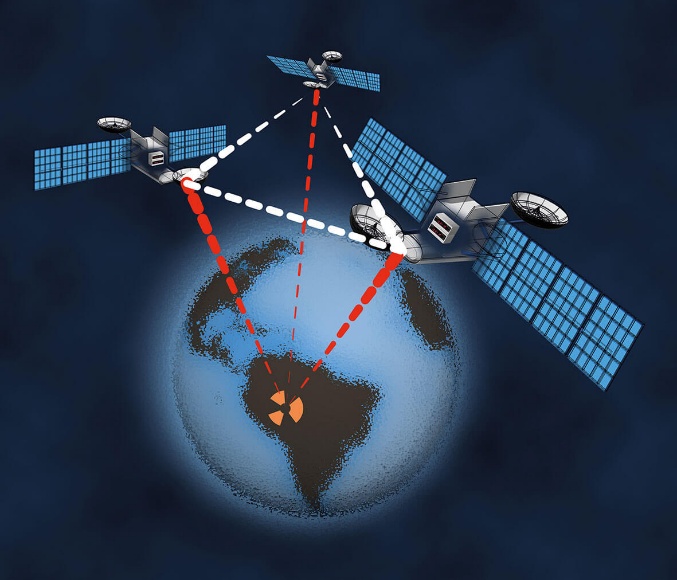


Ilustración . Representación de cómo se calcula la posición del receptor GPS con 3 satélites

* **LoRa**: Del inglés *Long Range* (Largo Alcance), es una tecnología de comunicación inalámbrica de largo alcance, bajo consumo energético y resistencia a interferencias diseñada principalmente para aplicaciones de IoT. Es capaz de transmitir información a mayor distancia que otras tecnologías como Wifi o Bluetooth, y es la base del uso del protocolo LoRaWAN. [Fuente](https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/)
* **LoRaWAN**: [Fuente](https://www.unir.net/revista/ingenieria/lorawan/) (Long Range Wide Area Network) Es un protocolo de comunicación sin cables y de bajo consumo basado en la tecnología LoRa. Para su uso, los nodos (dispositivos que recopilan datos que se quieren transmitir) envían información mediante la tecnología LoRa. Estos datos son captados por las gateways (estaciones que reciben la información), que los transmiten a través de internet al servidor correspondiente. Este gestiona la infraestructura de la red y organiza la transmisión de datos, enviando los recibidos del nodo a una aplicación, que los recibirá y tratará.

Este protocolo es ideal para una comunicación a larga distancia en la que no es necesaria una gran velocidad, así como su uso en zonas donde no hay la infraestructura para otras comunicaciones más convencionales escasean. Esto, sumado a su bajo consumo, ha permitido que el protocolo se use en una amplia variedad de aplicaciones en la industria, tales como monitorización en tiempo real en fábricas, agricultura inteligente, seguridad y control de accesos, etc.

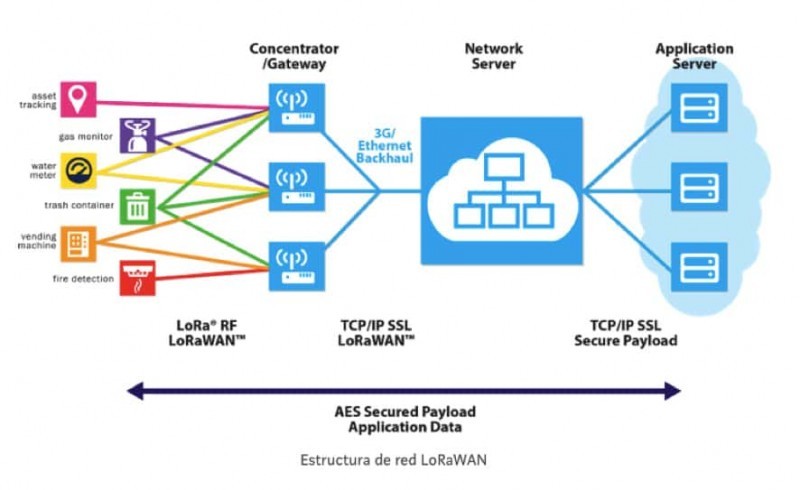
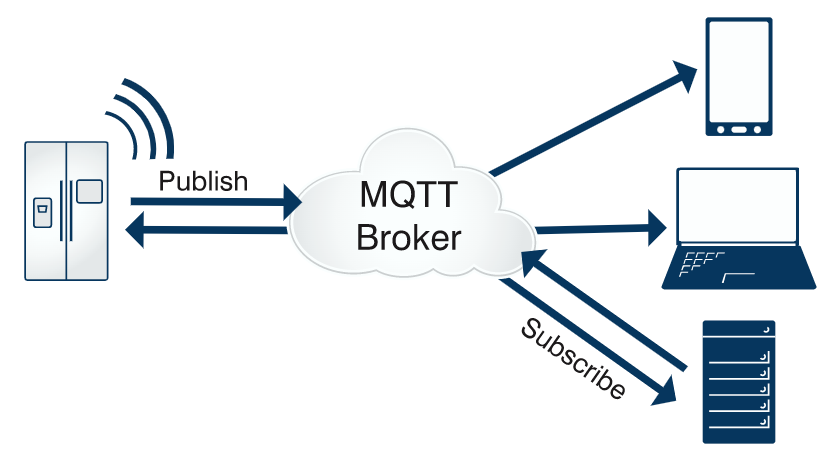


Ilustración . Estructura de la red LoRaWAN

* **TTN**: (The Things Network, Red de las Cosas) Es una plataforma global, abierta y gratuita de redes LoRaWAN que permite la implementación de redes IoT mediante este protocolo. Permite gestionar la comunicación entre nodos y aplicaciones.
* **Payload:** Carga útil en español, es el conjunto de datos transmitidos en informática y telecomunicaciones.
* **MQTT: (**Message Queuing Telemetry Transport) Es un protocolo de comunicación creado por IBM muy extendido en el área de IoT, debido a eficiencia en dispositivos de bajos recursos. Este se basa en la arquitectura cliente-servidor, de forma que los dispositivos cliente (emisores) envían la informacón a los receptores a través de un broker. El cliente se suscribe a un tópico (asunto) determinado, en el que publican (envían) la información. El receptor, que también se suscribe al tópico, recibe la información y la procesa. Este diseño es muy escalable, ya que cada dispositivo suscrito es independiente del resto [REFERENCIA].



## Requisitos de diseño (NO ME CONVENCE)

El sistema proyectado debe cumplir las siguientes condiciones:

* Ser capaz de proporcionar una medida lo más precisa posible mediante el GPS
* Enviar esa medida mediante tecnología IoT a una red como la TTN
* Almacenar esa información en una base de datos
* Mostrar la posición aproximada del vehículo de forma visual, así como una estimación del tiempo de llegada a la próxima parada
* Ser accesible por el usuario con relativa facilidad, de forma que cualquiera pueda ver esa información.
* Para cumplir la normativa europea (ERC/REC 70-03 y ETSI EN 300 220) la banda en la que el dispositivo debe operar es la de 868 MHz
* Permitir mejoras a su funcionamiento

## Análisis de soluciones

Como se vio en el capítulo 1.1, el objetivo de este proyecto es el diseño de un sistema de comunicación que permita geolocalizar vehículos de transporte público tales como autobuses. Para ello, se necesita conocer la localización del vehículo, así como una forma de enviar esa información. Para el primer paso se utilizará un microcontrolador que irá dentro del autobús, de forma que sea capaz de obtener sus coordenadas en tiempo real. Una vez obtenida esa información, hay que graficarla, realizando tratamiento previo. Será necesario almacenarla en una base de datos de forma que se pueda conocer y visualizar el recorrido del vehículo a lo largo del tiempo, y para poder enviarla habrá que usar otros sistemas intermedios. A continuación, se verá la selección de dispositivos y sistemas realizada para llevar a cabo el proyecto:

* + 1. El microcontrolador y la tecnología IoT

Primero se mencionarán las alternativas a la hora de seleccionar el microcontrolador. Inicialmente se barajó el uso de un ESP32, como el de la Ilustración 5

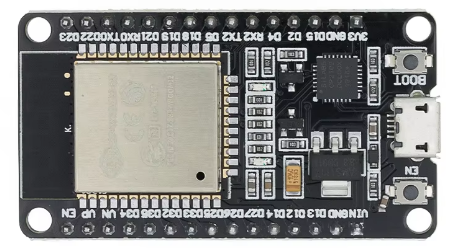


Ilustración 5. ESP32, posible opción para desarrollar el proyecto

Un dispositivo económico a la vez que potente, el ESP32 de *Espressif Systems* es conocido por su conectividad bluetooth y wifi, lo cual lo ha vuelto en los últimos tiempos uno de los microcontroladores más populares. Asimismo, su compatibilidad con el programa Arduino IDE, con la amplia documentación que existe online para su programación, lo vuelven una opción más que viable para este proyecto.

No obstante, su falta de conectividad GPS implica la conexión con un módulo externo que le aporte esa funcionalidad. Y, por otro lado, su conectividad bluetooth y wifi integradas no son de mucha utilidad en este proyecto, ya que ambas son de muy corto alcance. A esto se le suma que, en caso de utilizar wifi, debería tener un suministro externo continuo para poder utilizarlo. Debido a estos inconvenientes, habría que instalar otro módulo externo con una tecnología más adecuada para cumplir correctamente su función, lo que se traduce en tener 2 módulos externos que pueden dar problemas a la hora de la conexión.

Y es en este punto donde hay que tantear algunas posibilidades sobre qué tecnología IoT utilizar, ya que en caso de tener que añadir un módulo externo, ¿cuál sería la mejor opción?

Ya se ha hablado un poco de las conexiones Wifi y Bluetooth. Sin embargo, existen muchas alternativas en cuanto a la implementación de tecnologías IoT, como la Sigfox, muy eficiente energéticamente y con gran alcance, pero con una transmisión de datos muy limitada (además de que no está disponible en todas las regiones) o la Zigbee, también muy popular y que funciona de forma parecida al Bluetooth, pero con mayor alcance.

Finalmente, tras revisar varias posibilidades, se acabo decidiendo por el uso del microcontrolador de Ilustración 6. Este es el Seeeduino LoRaWAN, una placa de desarrollo con unos añadidos muy útiles para el proyecto: incorpora un módulo GPS, así como un módulo LoRa. Como se explica en el capítulo 1.5, LoRa es una tecnología que permite la comunicación mediante el protocolo LoRaWAN, una red de largo alcance que permitirá el envío de los datos que se requieren para el proyecto. La baja velocidad de este protocolo no será un problema, ya que no es necesario que la información transmitida sea especialmente veloz. El único inconveniente es que requiere de una pasarela/Gateway relativamente cerca. ¿ESTUDIAR POSICION DEL GATEWAY?

Esta incorporación de los módulos necesitados genera una placa ideal para el proyecto, pues se evitan problemas de compatibilidad con módulos externos, conexionado, etc.

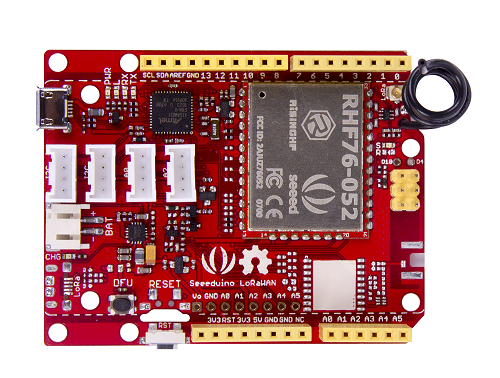


Ilustración . Seeeduino LoRaWAN, elección final para realizar el proyecto

El Seeeduino LoRaWAN, de *Seeed Studio*, es una placa de desarrollo basada en el chip ATSAMD21 sacado al mercado en 2016. Este cuenta con la posibilidad de ser alimentado con una batería de polímero de litio de 3.7 V, ya que su tensión de alimentación es de 3.3 V. Además de esto, posee una memoria flash de 256 KB, una SRAM de 32 KB y una frecuencia de reloj de 48 MHz.

PLANO AL ANEXO

* + 1. Programación

Para programar la placa, se usará el IDE de Arduino, de su fabricante de placas homónimo del que se habló en la historia de los microcontroladores. En este caso, se sacará provecho del procesador ATSAMD21, que es el mismo que el que utiliza el Arduino Zero. Esto permite que el Seeeduino sea programable con el software ya mencionado, donde se escriben los programas en C++.



Ilustración . Logo de Arduino, fabricante de placas del mismo nombre

Aunque tenga menos características de calidad de vida en comparación con otros muy extendidos como Visual Studio Code o CLion, facilita mucho la configuración y conexión con el microcontrolador, además de contar con una igualmente extensa comunidad de usuarios muy activos.

Este software funciona en los principales sistemas operativos de ordenador, y tiene una curva de aprendizaje relativamente baja, con pocas configuraciones necesarias para poder empezar a programar y una biblioteca de placas y librerías fácil de acceder y muy intuitiva, así como una amplia comunidad online. Además, tiene una implementación sencilla con los códigos de ejemplo que vienen incluidos en las librerías, lo cual ayuda mucho al trabajar con librerías específicas por primera vez.

* + 1. La red intermedia

Como bien se dijo anteriormente, LoRaWAN es una solución más que suficiente para el proyecto. Sin embargo, por su propia cuenta no es un protocolo perfecto. Como se ha comentado ya, este requiere de una pasarela que envíe la información a través de internet, permitiendo un tratamiento de los datos con tecnologías más convencionales. Es aquí donde entra en juego la TTN (*The Things Network*), una red global y abierta que facilita gestión e implementación de tecnologías que usen este protocolo. Esta red recibirá los datos del GPS cifrados y los almacenará temporalmente. De aquí podremos dirigir los datos al resto de software para completar la aplicación, enviándolos mediante el protocolo MQTT.

* + 1. La base de datos

Para poder tratar la información adquirida por el microcontrolador, es necesario almacenarla primero. De esta forma, posteriormente se podrá mostrar con un software aparte que se tratará en el siguiente punto.

Para ello, existen múltiples opciones gratuitas con las que realizar esta tarea. La opción elegida para este proyecto ha sido InfluxDB, una base de datos temporales muy útil para series temporales, como es el caso de este proyecto, en el que se realizarán varias mediciones en un intervalo constante.



Ilustración . Logo de InfluxDB, elección final para la base de datos

Esta plataforma es muy popular en el contexto de IoT, ofreciendo incluso un visualizador para mostrar los datos que se reciben, pudiéndose personalizar con varios ajustes.

* + 1. El agente de servidor

No obstante, Influx no tiene integración nativa con MQTT, por lo que por sí solo no puede suscribirse al bróker (TTN) para almacenar los datos. Para solucionar esto, es necesario utilizar algo llamado agente de servidor. Esto es un programa que actúa de intermediario, de forma que se suscribirá al tópico y transmitirá el mensaje a Influx mediante un protocolo compatible con este. Se puede entender este software como una pasarela digital, ya que realiza funciones muy similares a las de un Gateway.



Ilustración . Logo de Telegraf, el intermediario entre InfluxDB y la TTN

* + 1. Representación de la posición

Como se mencionó en el apartado 1.1, uno de los objetivos de este proyecto es que el usuario pueda acceder a la información de la localización del vehículo, de forma que pueda entenderla fácilmente. Para ello se usará un software que permita visualizar la información almacenada en la base de datos.

En primera instancia, una gran opción sería *Chronograf,* un visualizador con integración directa con InfluxDB, el software ya mencionado. Si bien sería muy sencillo realizar la conexión de los datos entre softwares, se prefirió optar por uno más potente, con más opciones de personalización, aunque se pierda facilidad en la configuración.

Debido a esto, se acabó seleccionando el software *Grafana*. Este es muy popular cuando se quiere mostrar información de varias fuentes simultáneamente, entre ellas InfluxDB. Además de eso, cuenta con mayores opciones de personalización, y el hecho de haber sido utilizado durante mis estudios, reduce mucho la curva de aprendizaje necesaria para configurarlo.



Ilustración . Logo de Grafana, el visualizador de datos usado

## Resultados finales

En este capítulo se desarrollará el proyecto en sí mismo, avanzando en orden cronológico. Primero se detallará la programación, seguido del envío de la información. Después se mostrará el almacenamiento de la información en la base de datos y su posterior representación

* + 1. Resumen de la comunicación

Hasta ahora se ha hablado de muchos conceptos, microcontroladores, protocolos de comunicación, software, etc. Para no perder de vista el objetivo del proyecto, es conveniente hacer un breve comentario sobre cuál será el papel de cada componente de este.

Como se habló en el capítulo 1.7, el Seeeduino LoRaWAN será el encargado de enviar la información. Este, con su GPS integrado, obtendrá las coordenadas de su localización y las procesará, enviándolas posteriormente mediante el protocolo LoRaWAN a un Gateway o pasarela. Allí, las coordenadas se transmitirán a través de internet a los servidores de la TTN, donde se decodifican y se envían por MQTT a InfluxDB, donde la TTN actuará de bróker. InfluxDB se suscribirá al tópico en el que se publiquen las coordenadas y almacenará las medidas. Para esta comunicación TTN-InfluxDB se utilizará Telegraf como intermediario. Por último, Grafana accederá a la base de datos, leerá los valores almacenados de latitud y longitud y los representará.

Todo este proceso se ve de forma más visual en el siguiente diagrama de bloques:

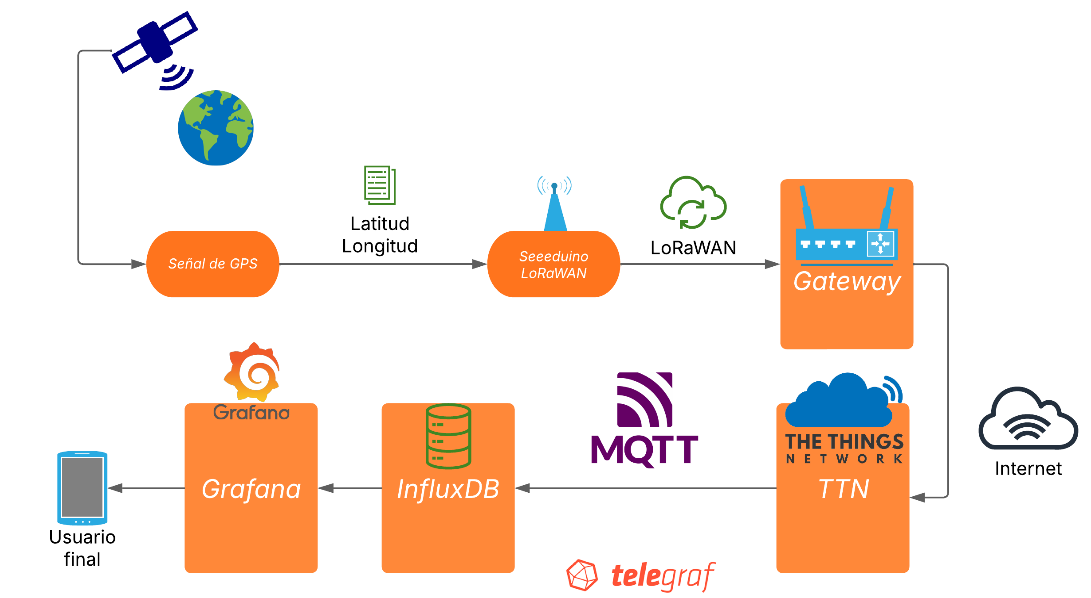


Ilustración . Diagrama de bloques que representa el funcionamiento del sistema de comunicación

Ahora se mostrará la programación del Seeeduino y las configuraciones de los componentes

* + 1. Programa de adquisición y envío de datos SEGUIR EXPLICANDO LO DEL FAIR USE, DR, DUTY CYCLE, AIR TIME, ETC. AÑADIR CAMBIOS EN EL CODIGO

Como se mencionó anteriormente, se realizará la programación del Seeeduino en el IDE de Arduino.

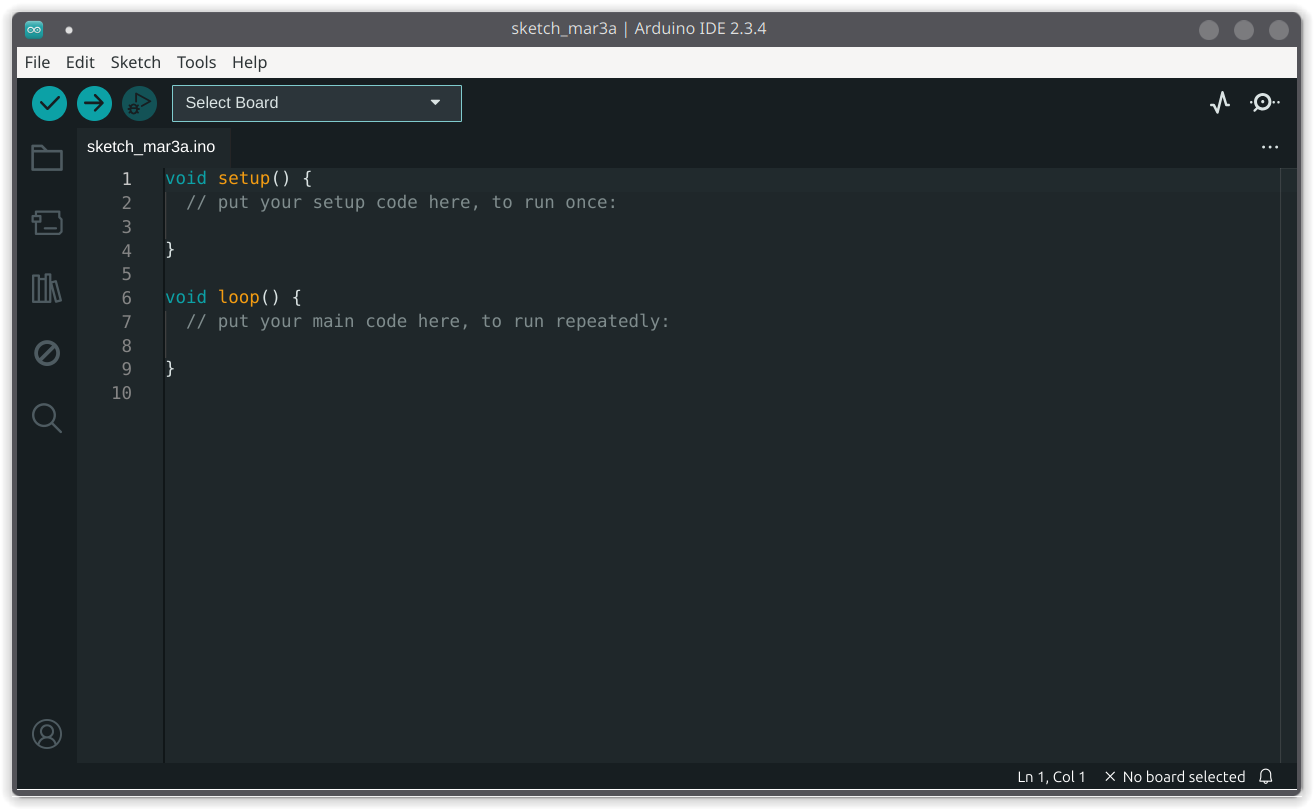


Ilustración . Entorno de desarrollo Arduino

En la Ilustración 9 se muestra el entorno de desarrollo utilizado. A la izquierda ha un gestor de archivos para los proyectos, así como un acceso rápido a las bibliotecas de placas y librerías. Justo encima están los botones de compilación (izquierda), de despliegue (centro) y de depuración (derecha). A la derecha de estos se encuentra el selector de puerto, donde se elige el puerto del ordenador en el que está conectado el microcontrolador, así como el tipo de placa de la que se trata.

A la derecha de la pantalla tenemos primero el Serial plotter, que grafica los datos recibidos por el puerto serie.

El siguiente botón es el Serial monitor, que abre una ventana en la que podemos leer los mensajes programados en el puerto serie. Finalmente, ocupando la mayor parte de la pantalla está el espacio de desarrollo, donde se escribe el código.

* + - 1. Setup inicial. Frecuencia y modo de envío

En “void setup()” aparecerán las configuraciones iniciales, mientras que en “void loop()” aparecerá el código del programa como tal.

Antes de comenzar hay que realizar ciertas acciones para configurar el programa y que este sea capaz de comunicarse con la placa. Para ello, además de los drivers disponibles en la wiki oficial de Seeeduino, es necesario incluir el enlace con la información de varias placas de Seeeduino, entre las que se encuentra el LoRaWAN GPS. Este enlace está también disponible en la web oficial.

Después de incluir lo anterior y conectar por USB la placa, se puede comenzar la programación.

En la primera parte se incluyen ciertas acciones iniciales como incluir bibliotecas, inicialización de variables y demás.

|  |
| --- |
| #include <TinyGPSPlus.h>  #include <LoRaWan.h>  #include <tinyFrame.h>  // Structs y objects  struct tinyFrame frame;  TinyGPSPlus gps;  // Las coordenadas se promediarán con tantos valores como se haya establecido en "precision"  #define precision 10  // Variables globales  float lat[precision];  float lng[precision];  int i = 0;  //char buffer[256];  // Credenciales para la TTN  char AppKey[] = "4FD4027BF712477561B43A6ABA38B129";  char devEUI[] = "70B3D57ED006BB05";  char AppEUI[] = "0000000000000000"; |

Las credenciales son únicas para cada dispositivo que se conecta a la TTN, de forma que esta sea capaz de reconocerlo

En el setup se inicia la comunicación con el gps, junto con el módulo LoRa.

|  |
| --- |
| void setup()  {    // Comienza la comunicación serie con el ordenador para debug y con el GPS (Serial2)    SerialUSB.begin(115200);    Serial2.begin(9600);    // Inicia el módulo LoRa    setup\_lora();    while(!lora.setOTAAJoin(JOIN));    Serial.println("JOINED");    frame.printDecoder = true; // Decoder de tinyframe. Útil para decodificar el payload en la TTN    delay(1000);  } |

Con la función “setup\_lora()” se realizan todas las acciones necesarias para el envío de los datos:

|  |
| --- |
| void setup\_lora (){  // LoRa module      lora.init();      lora.setKey(NULL, NULL, AppKey);      lora.setId(NULL, devEUI, AppEUI);      lora.setDeciveMode(LWOTAA);      lora.setDataRate(DR0, EU868);      lora.setDutyCycle(false); // Poner a true cuando este terminando??      lora.setJoinDutyCycle(false);      lora.setPower(14);  } |

Esta utiliza las funciones del objeto “lora” para establecer los valores de “Key” y “Id”, es decir, asigna las credenciales que se introduzcan en la parte inicial del código a la placa en la que este se ejecute. Posteriormente se hablará de como configurarlos.

Con “DeciveMode” (el nombre de la función tiene una errata, realmente debería llamarse “DeviceMode”) se indica el modo de conexión de la placa. Este paso se conoce como la “activación” del dispositivo. Existen 2 modos de activación diferentes:

* **ABP (Activation By Personalization)**: Requiere la codificación manual de la dirección en el propio dispositivo, así como las claves de seguridad. Es el menos seguro, además de que no permite un cambio en el proveedor de red sin modificar las claves en el dispositivo.

Cuando se codifican las credenciales en el dispositivo, estas están pensadas para una red en concreto, generando ese problema en el cambio de proveedor.

Precisamente por la necesidad de codificar manualmente las credenciales este procedimiento es menos seguro. El dispositivo tendrá la misma “contraseña” durante toda su vida, haciéndolo vulnerable.

Realmente su mayor utilidad en ciertos casos es que puede enviar la información independientemente de si se ha conectado a la red o no.

* **OTAA (On-The-Air Activation)**: El método más seguro y recomendado. Cuando el dispositivo se conecta a la red, se le asigna una dirección dinámica y unas claves de seguridad.

Esto presenta una gran ventaja: el propio dispositivo recibe unas credenciales evitando que el usuario tenga que introducirlas. Y, lo que es más, unas credenciales dinámicas proveen al sistema de una mayor seguridad.

Es un modo de activación superior al ABP en prácticamente todos los aspectos, con el único inconveniente de que el dispositivo no enviará información hasta que no se haya conectado a la red.

Se utilizó el método OTAA porque realmente no tiene inconvenientes en la mayoría de los casos, y presenta mejor seguridad que el ABP.

La siguiente función establece la frecuencia de envío de paquetes de datos. El primer argumento, DR0, indica que la velocidad de transmisión es la mínima, por lo que el alcance es el mayor. Teniendo en cuenta que la información a transmitir no es muy extensa, con eso es suficiente y se asegura que el dispositivo alcanza al gateway.

El segundo argumento es la frecuencia a la que se envían los datos. La librería utilizada, “LoRaWAN.h” incluye información sobre las frecuencias disponibles para el dispositivo según la región. En el caso de Europa, están disponibles las siguientes:

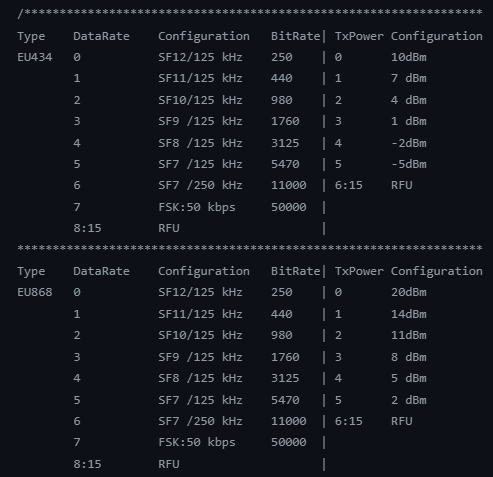


Ilustración . Frecuencias disponibles para usar con el Seeeduino LoRaWAN

“EU868” y “EU434” se refieren a las bandas de frecuencias normalizadas. Aunque ambas sirven para este caso, la elegida, como se ve en el código de la Ilustración 13 es la EU868. Su frecuencia superior permite mayor penetración de obstáculos, a pesar de que la EU434 permite mayor alcance.

Posteriormente aparece la configuración del ciclo de trabajo. Para cumplir con la política de uso justo de la TTN, se utilizarán estas dos funciones con el argumento “true”. Para el envío de datos existen varias bandas posibles de transmisión, y esta configuración permite que el mensaje se envíe por la banda que esté libre en ese momento automáticamente.

Se ha mencionado que la TTN tiene una política de uso justo (Fair Use Policy, (The Things Network, s.f.)). Esto son unas pautas para mantener el uso de sus servicios de forma pública. Se especifica que cada dispositivo tiene permitido al día un máximo de 30 segundos para enviar sus mensajes. ¿A qué se refiere esto? Para transmitir un mensaje, es necesario disponer de un canal de transmisión, un medio a través que el mensaje pueda desplazarse hasta llegar a su objetivo. Este canal puede ser tanto físico, como un hilo de cobre, o radioeléctrico, como puede ser el aire. Para este proyecto, como los mensajes se envían inalámbricamente, el canal será el segundo.

Otra condición que se establece, es que el límite de trabajo se limita al 1%. Esto es, el tiempo que pasa el mensaje siendo enviado debe ser como mucho el 1% de la suma entre el tiempo siendo enviado y el tiempo hasta el siguiente mensaje.

Por último, la TTN restringe el tiempo que esos mensajes pueden pasar siendo transmitidos a través del aire, tiempo que depende del tamaño de estos, así como de su velocidad de transmisión. Como se describe arriba, la velocidad de envío del mensaje será la mínima, lo que permite un mayor alcance, aunque nos restrinja el número de mensajes que podremos enviar.

Para terminar con esta función, aparece la configuración del consumo. Este valor indica la potencia transmitida en dBm (relativo a mW). En este caso se indica un consumo de 14 dBm, equivalente aproximadamente a 25 mW, valor normalizado para esta banda.

Finalizado el setup del módulo LoRa, el dispositivo se conecta en modo OTAA y se activa el decoder de tinyframe, que servirá más adelante para decodificar el payload que se enviará a la TTN.

* + - 1. El programa

Con las configuraciones listas es momento de comenzar con el programa de envío de datos propiamente dicho.

Este primeramente comprobará si la comunicación con el GPS está disponible. En caso de estarlo, lee el valor de las coordenadas. Lo más importante que se hace en este momento es la comprobación de valores coordenadas correctas (gps.location.isValid()). Si no se hace esto, se enviarán mensajes con las coordenadas (0,0), que no aportan la información que se busca y solo utilizaría recursos de la red para nada. Por ello, el código esperará hasta que las coordenadas sean válidas antes de seguir.

Una vez los valores son válidos, toma el número de medidas indicadas por la variable “precision” , para luego hacer una media entre todos los valores. Este resultado será el enviado. Esto es importante dada la naturaleza inexacta del GPS. Aunque este es capaz de aproximarse bastante bien a la localización del aparato, es muy posible que haya ligeras variaciones en las medidas que desplacen las coordenadas ligeramente. Para tratar de disminuir esto, se optó por hacer una media con 10 valores, que en la práctica resultaron efectivos.

Con el valor listo para enviarse se llama tanto a la función “envio\_lora” como a la función “dist\_semiverseno”. De estas se hablará en páginas posteriores. Tras ello se guardan los valores de latitud y longitud calculados en este ciclo (necesario para “dist\_semiverseno”, como se ve en sus argumentos).

Hecho esto, aparece un condicional que compara el valor de la variable “distancia”, el cual se ha calculado con “dist\_semiverseno”. Si este valor es superior a 50 km, se considerará un cambio brusco, siendo cualquier otro valor considerado como cambio no brusco. Aunque se explicará más específicamente más adelante, conviene hacer un pequeño adelanto. Ya se ha mencionado que el GPS no es muy exacto, y aunque no es muy común, es posible que aparezcan algunas coordenadas extremas debido a pérdida de señal, desconexión y posterior reconexión rápidas del Seeeduino, etc. Esta variable permitirá conocer si ha ocurrido este evento, de forma que la temporalización entre mensajes se ajuste.

La temporalización mencionada aparece justo después del uso de “memset”, con el que se reinician los vectores a 0, así como con el reinicio de la variable “i”. La lógica es la siguiente: si la distancia de una medida a otra ha sido de menos de 50 km, se considerará que no ha habido un cambio brusco y se hará un envío cada 5 minutos. Es suficiente tiempo como para conocer la trayectoria del vehículo, así como para cumplir con el fair use de la TTN. Sin embargo, se ha considerado que ha habido un cambio brusco, se enviará la siguiente medida a los 2 minutos. De esta forma, si ha habido algún problema con el GPS, se tratará de corregir enviando las siguientes coordenadas (que se esperan correctas) con mayor frecuencia.

Viendo el código de la página siguiente, es sencillo darse cuenta que aparecen variables relacionadas con la velocidad de las que no se ha hablado. Durante el desarrollo del proyecto se llegó a un problema a la hora de representar el tiempo restante para llegar a la siguiente parada, tema que se aborda en profundidad en el punto 1.8.9. Como breve resumen, para calcular el tiempo se utiliza la medida de velocidad obtenida por el GPS. Este valor se obtiene promediando las mediciones obtenidas cada minuto durante los 5 minutos que el Seeeduino no está enviando datos. Una vez llegue el momento de realizar el envío de las coordenadas, se añadirá al payload un byte extra con la medida de la velocidad promediada.

CAMBIAR EL CODIGO DE ABAJO

|  |
| --- |
| void loop()  {    while (Serial2.available() > 0){      if (gps.encode(Serial2.read())){        if (gps.location.isValid()){            lat[i] = gps.location.lat();  // latitud          lng[i] = gps.location.lng();  // longitud          i++;          if (i >= precision){  // Cuando los vectores de tamaño "precision" se llenan:            // Inicia valores medios:            float lat\_m = 0;            float lng\_m = 0;            // Promedia las medidas            lat\_m = average(lat, precision);            lng\_m = average(lng, precision);            // Descomentar para ver los valores medios en pantalla            //Serial.println("Latitude:");            //Serial.println(lat\_m, 6);            //Serial.println("Longitude:");            //Serial.println(lng\_m, 6);            // Envia datos a la TTN            //memset(buffer, 0, 256);            envio\_lora(lat\_m, lng\_m);              // Reinicia los vectores            memset(lat, 0, sizeof(lat));            memset(lng, 0, sizeof(lng));            i = 0;          }        }      }    }  } |

Ahora se explicará el papel de las funciones que aparecen en el loop, “envio\_lora” y “dist\_semiverseno” PONER CODIGOS BUENOS

|  |
| --- |
| void envio\_lora(float latitude, float longitude){  // Envia la información    frame.clear();    frame.append\_int32\_t((int32\_t) (latitude\*1000000.0));    frame.append\_int32\_t((int32\_t) (longitude\*1000000.0));    // Envía el mensaje a TTN    if (lora.transferPacket(frame.buffer, 10)){      SerialUSB.println("Data sent");    } else{      SerialUSB.println("Error");    }  } |

Al enviar los datos a la TTN, se codifican. Para poder leerlos es necesario deshacer esto, y las 3 primeras líneas de esta función (Ilustración 12) se encargan de ello. Muestran en pantalla el código necesario para descodificar la información que se vaya a enviar en Javascript. Esto solo es necesario verlo en el proceso de desarrollo, en el despliegue final no se necesita.

Cabe destacar que, aunque las coordenadas reales se representen como valores con varios decimales, antes de enviarlo se convertirán a entero, multiplicándolo por 1000000 para mantener los decimales. Esto permitirá reducir el tamaño del paquete, ya que un entero de 32 bits ocupa menos memoria que un flotante. Esto se traduce en que se enviará un paquete con 2 números de 4 bytes cada uno. Para la velocidad, debido a la menor necesidad de precisión del valor y a los valores que este podrá tomar, se optó por codificarlo en un byte, optimizando el tamaño del paquete.

Más abajo se hace el envío del mensaje como tal, donde surge un problema. Las políticas de uso libre de la TTN limitan el número de mensajes a enviar hacia la red, así como el número de mensajes a recibir de ella. Esta primera condición no es realmente un problema, puesto que se pueden ajustar los parámetros para optimizar el envío y asi tener más cantidad disponible, como ya se ha hecho. Sin embargo, la segunda condición si que es un problema: solo se permite que la red envíe hasta 10 mensajes a un mismo dispositivo al día. Esto limita la comunicación en tanto que no se podrán recibir mensajes de confirmación (ACK), es decir, el Seeeduino tratará de enviar el mensaje, pero no sabrá si este ha llegado a su destino o no.

Y no solo eso, la función que hace el envío devuelve “true” si ha enviado el mensaje, y “false” si no lo ha hecho, pero en la práctica es irrelevante. Hubo casos en los que devolvió “true” y no hizo el envío, o al menos no llegó a la TTN, y casos en los que devolvió “false” y el envío llegó correctamente.

Es por esto que se decidió solucionarlo de la siguiente manera: se realizará el envío del mensaje sin confirmación de forma repetida. Es decir, en vez de enviar el mensaje hasta que la función devuelva “true”, se enviará varias veces seguidas, ignorando lo que devuelva la función. De esta forma, es más probable que llegue un mensaje a la red, aunque no se pueda saber realmente. Unas 3-4 repeticiones son más que suficientes para esto. Más adelante se retomará este tema, cuando se hable de las primeras pruebas.

Por último, la función “dist\_semiverseno” es la siguiente:

float dist\_semiverseno(float lat\_1, float lat\_2, float lng\_1, float lng\_2){

  // Funcion para calcular la distancia entre 2 coordenadas mediante la formula del semiverseno

  float R = 6378\*1000;

  float lat\_1\_rad = lat\_1\*PI/180.0;

  float lat\_2\_rad = lat\_2\*PI/180.0;

  float lng\_1\_rad = lng\_1\*PI/180.0;

  float lng\_2\_rad = lng\_2\*PI/180.0;

  float h = (1 - cos(lat\_1\_rad - lat\_2\_rad))/2 + cos(lat\_1\_rad) \* cos(lat\_2\_rad)\*(1 - cos(lng\_1\_rad - lng\_2\_rad))/2;

  float d = 2 \* R \* asin(sqrt(h));

  return d;

}

Esta función realiza el calculo de la función semiverseno. Esta es una expresión trigonométrica muy importnate en la navegación astronómica, habiendo registros de du uso en el siglo XVIII. Esta permite el cálculo de la distancia entre dos puntos de una esfera conocidas sus coordenadas, es decir, su latitud y longitud. Su expresión es la siguiente:

A partir de esa expresión, se puede explicar la siguiente para calcular la distancia:

λ)

Donde d es la distancia entre los dos puntos, R es el radio de la Tierra, φ1 y φ2 son las latitudes de los puntos y λ la diferencia de longitudes.

La función hace ese cálculo; primero pasa las coordenadas a radianes y después aplican las expresiones para calcular “h”. Tras ello se calcula el valor de la distancia, y se devuelve.

Terminado el código, pulsar el botón “Upload” hará que el software envíe el programa a la placa tras su compilación. Utilizando la función “displayInfo”, obtenida de la wiki de Seeduino (Seeeduino LoRaWAN, s.f.) y que se incluye en el programa, se puede ver la señal de GPS que va adquiriendo la placa. Estos valores inicialmente son erróneos, es necesario que pase un tiempo hasta que se obtenga una medida correcta

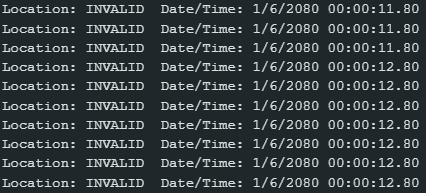


Ilustración . Medidas incorrectas al inicio del programa

No obstante, esto solo ocurre al encender el dispositivo, mientras esté alimentado mantendrá la señal de GPS. Si en algún momento la pierde, tardará relativamente poco tiempo en recuperarla. Esto siempre y cuando siga habiendo señal.

SEGUIR EXPLICANDO

* + 1. Envío a la TTN

Terminada la explicación del código, se tratará la conexión de la placa con los servicios de la TTN

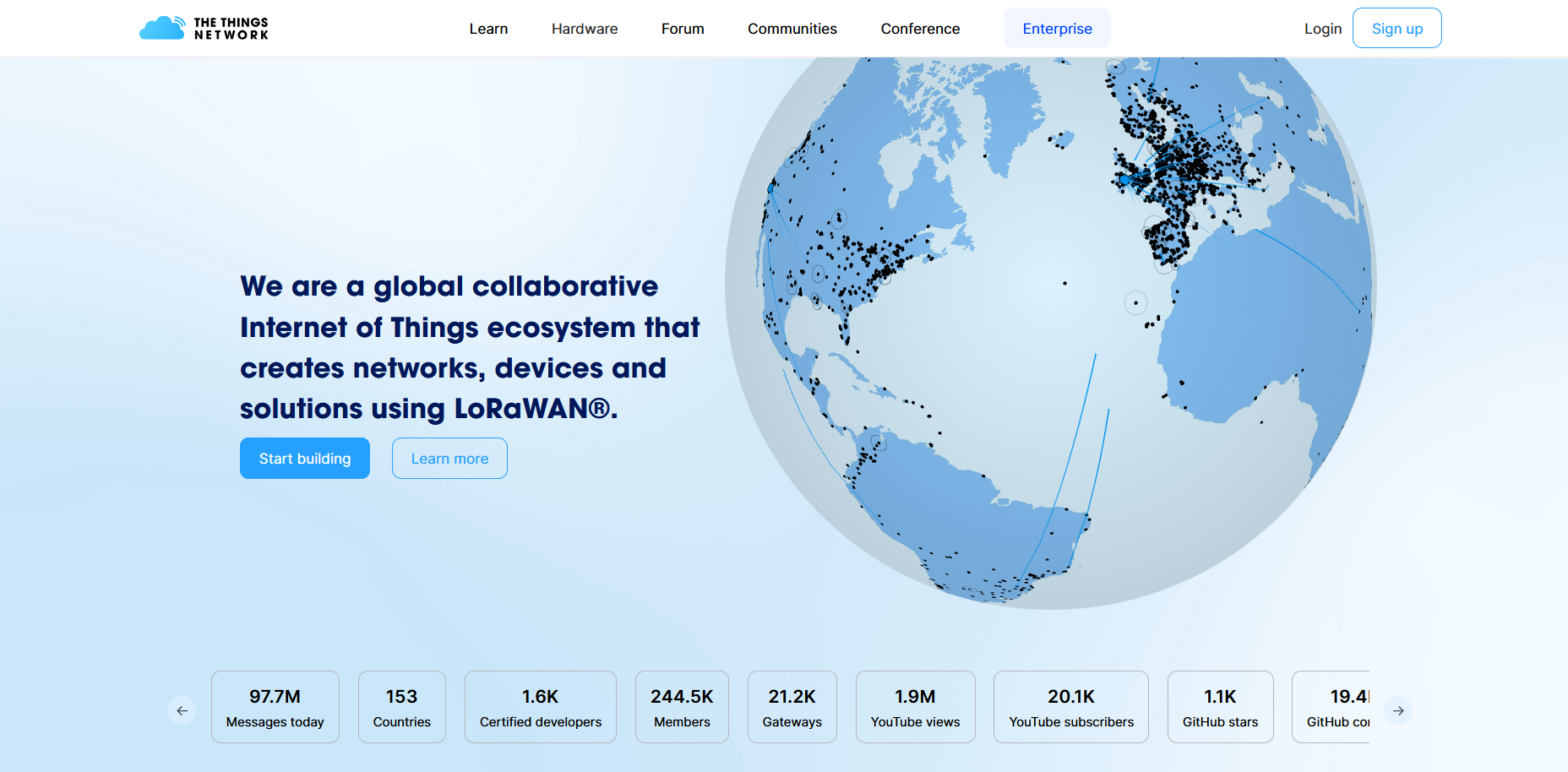


Ilustración . Página de inicio de la web The Things Network

La web ofrece varios planes de suscripción, pero para este proyecto la versión gratuita es suficiente, ahorrando costes.

Completo el registro, se crea una aplicación y se añade el dispositivo, “end device”. Un end device no es más que un dispositivo que se encuentra en un “extremo” del proceso de envío de información. Este puede ser tanto emisor como receptor. En el caso de este proyecto, se comportará siempre como emisor.

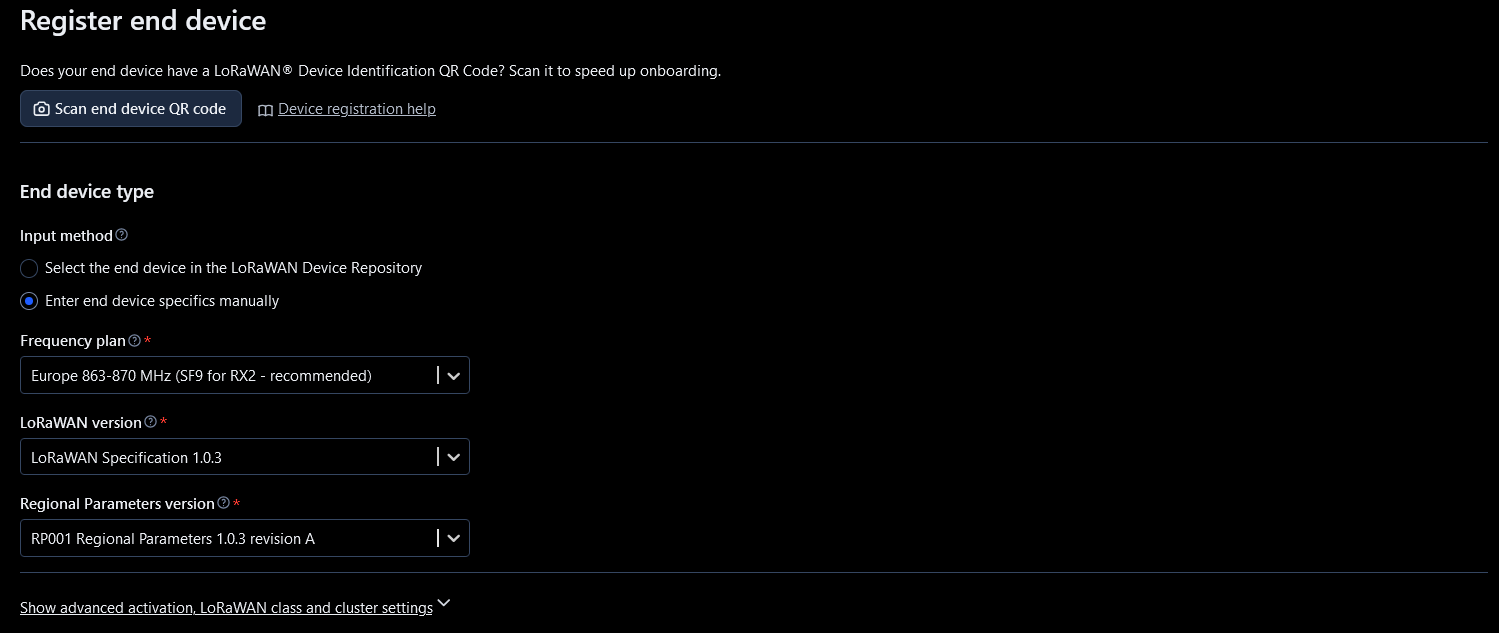


Ilustración . Configuraciones del End device.

Para añadirlo, es necesario especificar información sobre el dispositivo, como puede ser el plan de frecuencia (868 MHz, como ya se mencionó), la versión de LoRaWAN (en este caso es la 1.0.3) y el código JoinEUI. Este último es un identificador de 64 bits que diferencia cada dispositivo a la hora de conectarse a los servidores. Puede proveerlo el fabricante, pero también es posible implementarlo a mano, siempre y cuando el dispositivo pueda programarse para incluir ese código. En el código mostrado ya, este valor coincide con el AppEUI. Aunque el nombre aparezca como uno diferente, realmente son el mismo código, solo que en versiones más recientes de LoRaWAN el AppEUI pasó a llamarse JoinEUI

Hecho esto, aparecen más configuraciones. Aparece el DevEUI, un código de 64 bits también que sirve como identificador único de cada end device. Este también debería ser provisto por el fabricante, pero la web ofrece la generación de una clave por si no se dispone de la del dispositivo. En el programa se codifica en la variable “devEUI”.

Tras esto aparece el campo “AppKey”. Como antes, el fabricante la da, pero el usuario puede establecer la suya propia. Es una clave de encriptación utilizada durante la conexión OTAA, que es la que se utiliza en este proyecto, como se mencionó durante la explicación de la programación. En el código aparece en la variable homónima, y la web permite su generación automática.

El siguiente campo por rellenar es el “End device ID”. Este no es más que el nombre que se le quiera dar al dispositivo. Este debe ser único dentro de la aplicación, y no se puede editar después.

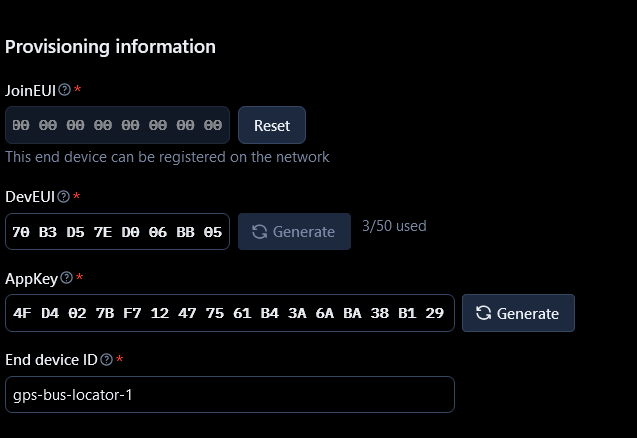


Ilustración . Parámetros de configuración finales del dispositivo en la TTN

Registrado el microcontrolador, este aparece en la web y ya se permite la comunicación entre este y el servidor.

No obstante, no todo el trabajo está terminado, pues el Seeeduino LoRaWAN codifica el payload al enviarlo. Para recuperar el mensaje original, se utiliza la función de la aplicación creada “Payload formatters”, en concreto la de “uplink”, es decir, mensajes entrantes al servidor de TTN. Ahí se implementa el siguiente código en Javascript:

AÑADIR EL DE LA VELOCIDAD

function decodeUplink(input) {

decoded = {}

decoded.latitud = (input.bytes[0] << 0 | input.bytes[1] << 8 | input.bytes[2] << 16 | input.bytes[3] << 24)/1000000.0;

decoded.longitud = (input.bytes[4] << 0 | input.bytes[5] << 8 | input.bytes[6] << 16 | input.bytes[7] << 24)/1000000.0;

return {

data: {

decoded,

//bytes: input.bytes,

},

warnings: [],

errors: []

};

El paquete enviado, como se expuso en el código, son dos números de 32 bits que enviamos juntos en uno solo. Para separarlos, dividimos el payload en bytes, utilizando los 4 primeros para la latitud y los siguientes para la longitud. Después, se divide entre 1000000 para recuperar los decimales.

* + 1. Implementación del protocolo MQTT

Como ya se dijo, necesitaremos enviar las coordenadas desde la TTN hasta la base de datos de alguna forma. Para ello, se eligió el protocolo MQTT, uno muy sencillo a la vez que escalable. La web de la TTN ofrece la posibilidad de generar diversas API keys. Esto no es más que un código privado que permite la conexión de un software con otro. En este caso, crearemos una para integrarla con Telegraf, del cual se hablará más adelante, así como otra para supervisor que el protocolo esté funcionando bien. Esto se hizo con el software “MQTT Explorer”, que permite de forma sencilla ver el flujo de mensajes.

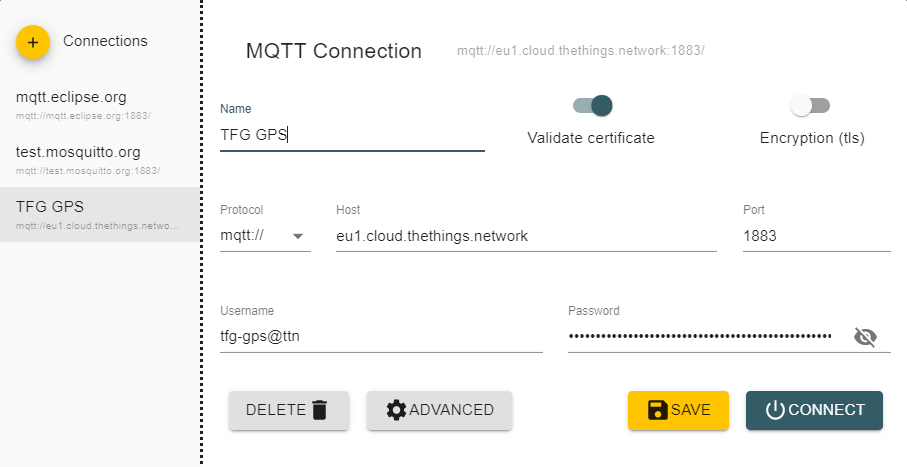


Ilustración 18. Interfaz de conexión del programa MQTT Explorer

Desde el programa mencionado, se rellenan los campos con los parámetros generados en la key de la TTN. La información sobre el host y el puerto también se encuentra allí. Tras esto se pulsa en “connect”, y se mostrará a la izquierda el tópico en el que se irán publicando los mensajes:



Ilustración 19. Topic donde se publican los mensajes que lleguen a la TTN

Por otro lado, a la derecha se tiene el mensaje en sí:

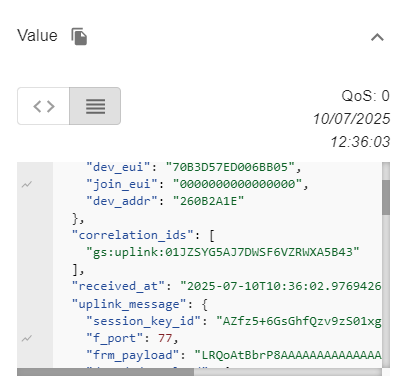


Ilustración 20. Interfaz de MQTT Explorer mostrando el último mensaje recibido

Gracias a este software, se tiene una forma sencilla de comprobar que la transmisión por MQTT está funcionando correctamente, muy útil a la hora de desarrollar el proyecto.

* + 1. Primeras pruebas

Con las configuraciones requeridas listas, se puede realizar una prueba para ver si el servidor recibe los mensajes. Conectando el Seeeduino al ordenador permite leer los mensajes que este envíe por USB, por lo que las pruebas ser harán así. En la práctica, cualquier fuente de alimentación que suministre suficiente potencia al aparato permitirá que este funcione.

Al alimentar el Seeeduino, comenzará a ejecutar el programa empezando por la configuración. Pulsando en el botón de monitor serie del IDE de Arduino, se pueden ver los mensajes que envía el dispositivo a través del puerto serie. Las configuraciones muestran una salida como sigue:

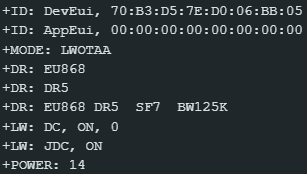


Ilustración . Monitor serie del IDE de Arduino mostrando las configuraciones

Se ve como coinciden con las configuraciones descritas en la explicación del programa. Tras esto, el microcontrolador intenta conectarse a la TTN, mostrando lo siguiente:



Ilustración . Mensajes del Seeeduino sobre su conexión con la red

El último mensaje, “JOINED”, aparece una vez la conexión ha concluido satisfactoriamente. Volviendo a la web de la TTN, se puede ver como trata de conectarse el dispositivo a través de la pantalla “Live data”. Esta información también se refleja en el MQTT Explorer, descrito anteriormente:



Ilustración . Mensajes de conexión del Seeeduino con la TTN, visto desde la web

El primer mensaje recibido, el inferior, indica que se ha aceptado la petición de conexión. Tras esto, se muestra que la conexión ha sido realizada con éxito, recibiendo un último mensaje para confirmarlo. A la derecha se pueden ver las credenciales del dispositivo conectado, mientras que a la izquierda aparece el nombre asignado al mismo, así como la hora de llegada del mensaje.

Una vez ha finalizado todo este proceso, el Seeeduino comenzará a tomar medidas del GPS, enviándolas como se vio en la descripción del código.

Durante las pruebas se encontró que, aunque el Seeeduino indique que el mensaje se no se ha podido enviar correctamente, en la TTN este aparece como recibido. Esto se explicó anteriormente, con la función “envio\_lora”. Como se dijo entonces, que la red envie un mensaje consume muchos recursos, limitandolo a 10 al día. Por ello, los mensajes con confirmación se suelen reservar para datos críticos que se necesita conocer si han llegado a su destino o no. En este proyecto no es necesario por no haber involucrados ese tipo de mensajes, por lo que no se utilizó confirmación.

Esta falta de entendimiento entre nodo y servidor hace que los mensajes puedan ser enviados repetidamente, ya que el código lo intenta enviar continuamente hasta conseguirlo sin comprobar que este haya llegado a su destino, o que aunque el Seeeduino haya podido enviar el mensaje, este no llegue a su destino.

Para solucionarlo, como no es posible usar el ACK, se optó por realizar repeticiones del envío, como ya se comentó. Si se envía solo una vez y falla, no llegará un nuevo mensaje hasta que haya pasado el tiempo de espera. Si por el contrario se envía el mismo mensaje varias veces, es más probable que este llegue al menos una vez, aunque en algunas ocasiones el mismo mensaje se envíe correctamente varias veces y llegue a la red. El valor de repeticiones se fijó en 3 pues las pruebas se comprobó que pocas veces se llega a este valor, lo más usual es que llegue uno o dos mensajes a lo sumo. Aumentar este valor causaría que pudiera darse el caso en el que lleguen muchos mensajes repetidos, ya que si bien es cierto que no se dieron muchos casos donde llegaron los 3 mensajes, es una posibilidad que existe y que es preferible evitar para no saturar la red de mensajes redundantes.

En etapas posteriores se tratarán los datos para no utilizar las medidas repetidas y solo utilizar la última en llegar.

Durante estas pruebas se comprobó que en un día de cielo despejado, el gps puede tardar en dar señal hasta unos 20 minutos, aunque este tiempo es muy inconsistente y no se puede dar un valor exacto, pues depende de muchos factores fuera del control del proyecto. Por otro lado, se vio que en días nublados puede llegar a ta

* + 1. Conexión a través de Telegraf

Como ya se comentó, para poder comunicar el servidor de la TTN con la base de datos será necesario utilizar un intermediario. Esta función será la que realice Telegraf.

Una vez descargado el software, es necesario configurarlo. En la descarga se encuentra un fichero llamado “telegraf.config” que incluye toda la configuración. Para ello, primero es necesario crear una API key en la web de TTN e incluirla en este fichero, como se hizo para MQTT Explorer. Aquí se debe generar un usuario y contraseña para Telegraf.

Con esto hecho, aún queda un ultimo paso para vincular Telegraf con la base de datos, pero para ello primero hay que preparar esta última.

* + 1. Configuración y puesta en marcha de InfluxDB

Tras descargar el software desde la web oficial, se obtiene un archivo ejecutable a través del terminal, que inicia el servicio utilizando el ordenador en el que se ejecute como servidor. Para trabajar con el programa, hay que utilizar un navegador cualquiera y buscar “localhost:8086” si se esta accediendo desde el propio ordenador donde se ejecuta el programa. Al entrar en la web, hay que crear un nuevo usuario en caso de no tenerlo, creando también una organización, que no es más que una forma de separar las configuraciones y usos del software dependiendo de la aplicación en la que se esté trabajando.

Una vez iniciada la sesión, se muestra la interfaz de bienvenida:

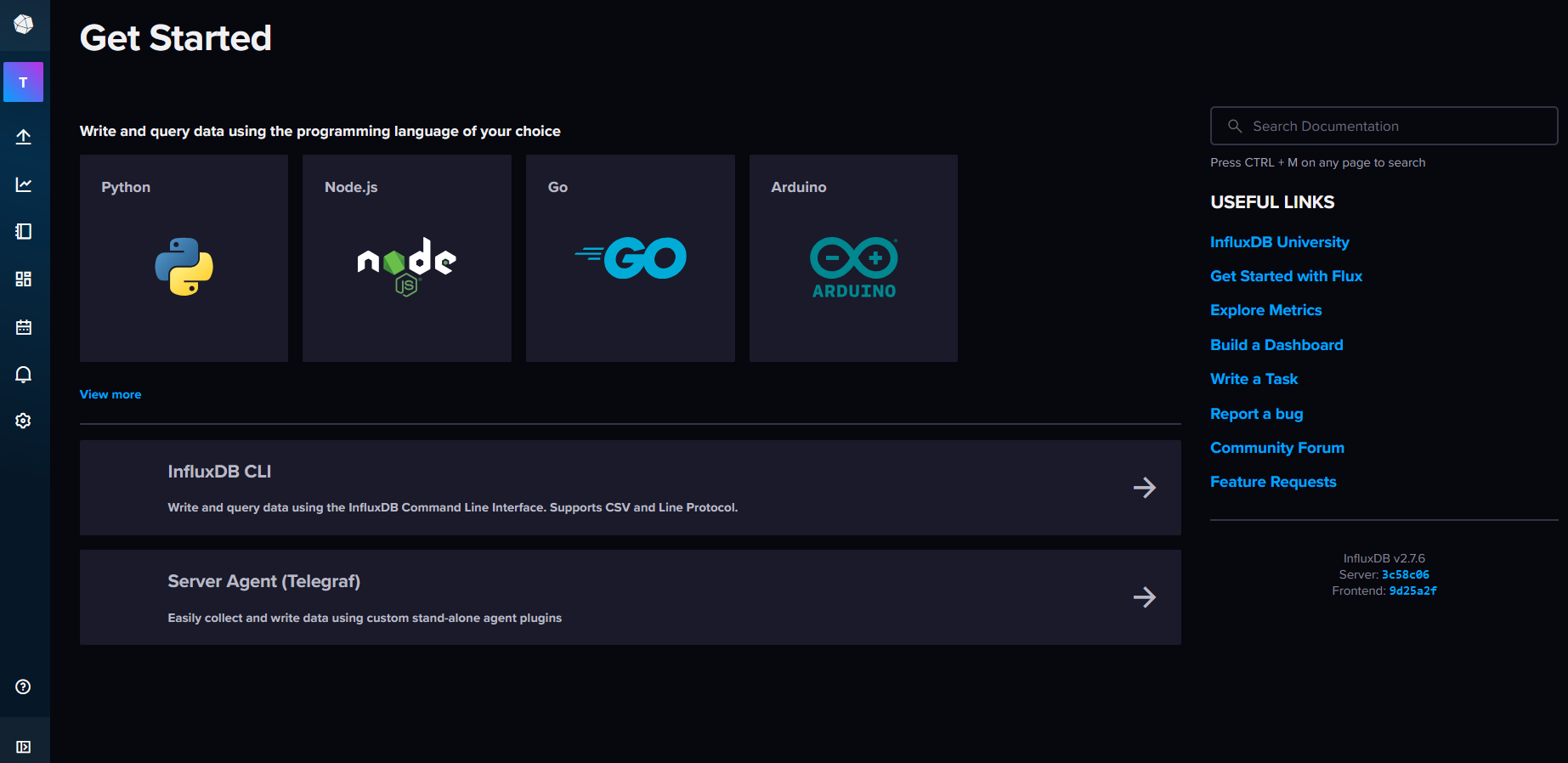


Ilustración 24. Interfaz de bienvenida de InfluxDB

Aunque antes de nada se terminará la configuración de Telegraf, generando un token en Influx. En la barra lateral de la izquierda, seleccionando el segundo icono, aparece la pestaña “API Tokens”, donde se puede generar una clave como antes en la TTN, permitiendo elegir entre un token de acceso completo a todas las funciones o uno con acceso personalizado.

Esta clave se implementa en el fichero de configuración de Telegraf, así como la URL utilizada en el navegador, el nombre de la organización y el “bucket”. Esto último hace las veces de “contenedor” de datos. InfluxDB va almacenando los datos que le llegan en estos buckets, de forma que se van organizando en subcategorías. Desde el mismo desplegable de antes, se puede seleccionar “Buckets” para crear y gestionar los mismos. Cuando se crea uno y se le indica a Telegraf que envíe los datos a este, en el “Data Explorer” de Influx (en la barra lateral, tercer botón) aparece una ventana donde se mostrará de forma visual los datos recogidos, que se puede ver en la Ilustración 25:

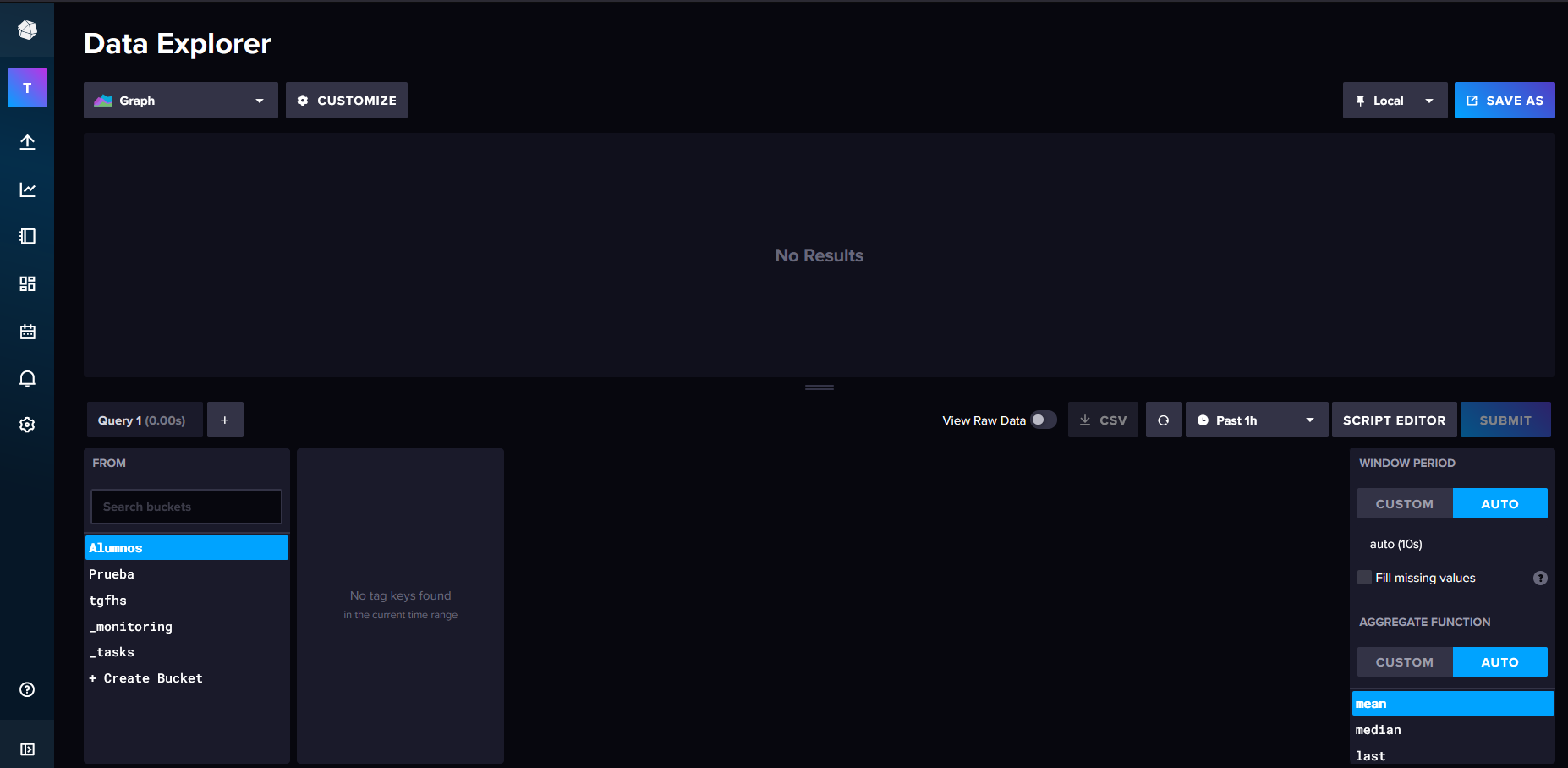


Ilustración 25. Ventana de visualización de datos de InfluxDB

En la zona inferior aparece el “Query Builder”, el constructor de peticiones. En él, se pueden seleccionar los datos que se mostrarán en la zona superior, así como personalizaciones simples como el tipo de gráfico (arriba), la opción de realizar la petición mediante un script, el cálculo de la media, mediana o el último valor, etc.

Este script se escribe en el lenguaje “Flux”, implementado en la versión 2.0 del programa. Es uno diseñado para hacer peticiones de datos, analizarlos y tratarlos. La propia web de influx ofrece documentación para utilizarlo REFERENCIA

Realmente no es necesario visualizar los datos en este paso, pero como ocurre con el MQTT Explorer, es útil para comprobar que en cada parte de la cadena de transmisión todo funciona correctamente. Y no solo eso, antes se mencionó que es posible realizar la petición mediante un script. Este código también se genera automáticamente si generamos la petición con el builder, y se puede utilizar en el siguiente paso, Grafana, para seleccionar los datos que queremos mostrar. En ese paso posterior será importante conocer más el lenguaje Flux, ya que habrá que hacer cierto tratamiento de los datos que en influx no es necesario.

* + 1. Representación y configuración de Grafana

Después de comprobar que la base de datos recibe correctamente las coordenadas, solo queda hacer la representación gráfica de estos valores. Como se expuso anteriormente, se utilizará Grafana.

Los primeros pasos son muy similares a Influx, decargando y creando una cuenta en la web. Igual que en el software anterior, la interfaz gráfica se accede desde el navegador, a través de la url “https://localhost:3000”.

Creada la cuenta Grafana mostrará su interfaz de bienvenida:

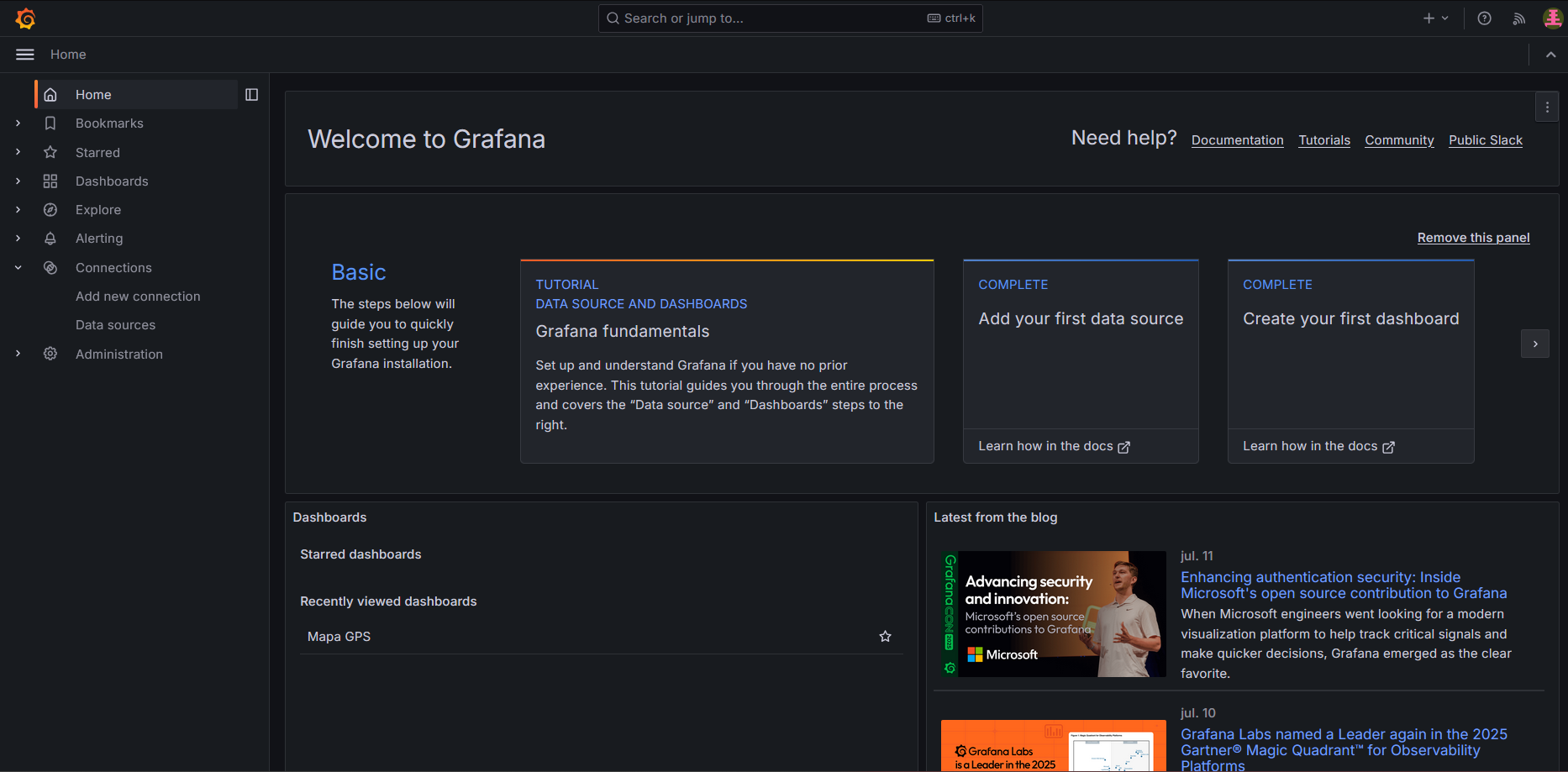


Ilustración 26. Ventana de bienvenida de Grafana, aparece al acceder a la web

Antes de configurar nada, como antes, hay que crear un token en InfluxDB para comunicarlo con este programa. Para ello, se accede a la interfaz de creación de tokens de Influx, haciendo lo mismo que con Telegraf. Con esa clave y volviendo a Grafana, se accede al desplegable “Connections”, a la izquierda, lo que abre una interfaz de selección de conexión con otros programas. Buscando “InfluxDB” y creando una nueva fuente de datos, se abre una interfaz como la siguiente:

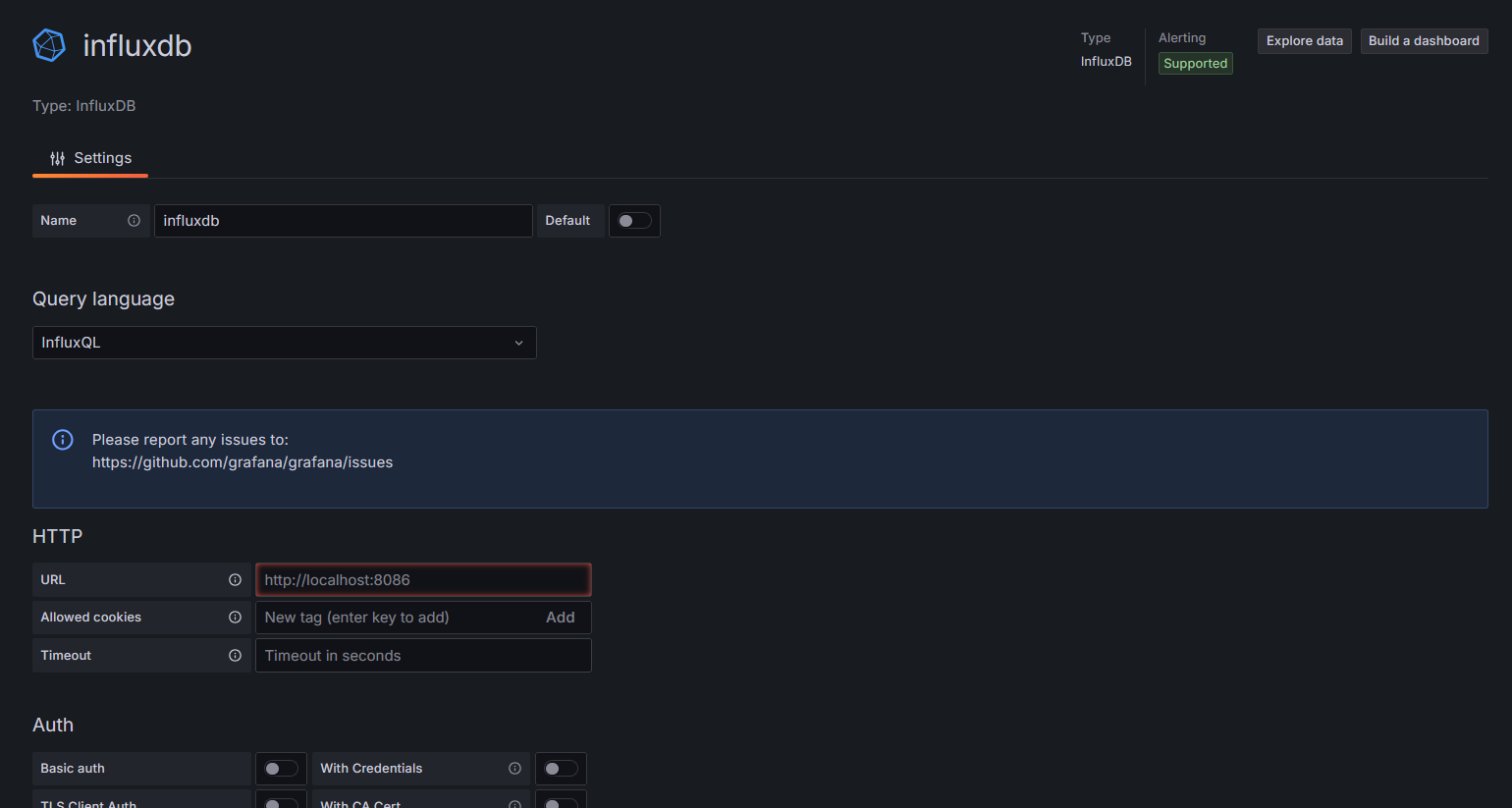


Ilustración 27. Interfaz de creacion de fuente de datos en Grafana

Aquí se puede configurar la conexión con InfluxDB, comenzando por el nombre que se le quiera dar a la conexión. Justo debajo se tiene la selección del lenguaje de peticiones, donde se selecciona “Flux”. En el apartado HTTP, se indica la url con la que se accede a la base de datos, que ya se mostró antes. Bajando un poco más, aparecen unos ajustes de autorización, así como los detalles de la base de datos. Estos son los datos con los que se configuró InfluxDB, (organización, bucket).

Con la conexión lista, se selecciona “dashboards” a la izquierda, donde se creará el panel de visualización de las coordenadas. Aquí se copia el Query generado en Influx, con ciertos extras para la visualización:

from(bucket: "Alumnos")

  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)

  |> filter(fn: (r) => r["\_measurement"] == "mqtt\_consumer")

  |> filter(fn: (r) => r["\_field"] == "uplink\_message\_decoded\_payload\_decoded\_latitud" or r["\_field"] == "uplink\_message\_decoded\_payload\_decoded\_longitud")

  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)

  |> pivot(rowKey:["\_time"], columnKey: ["\_field"], valueColumn: "\_value")

  |> map(fn: (r) => ({

      r with

      latitude: r["uplink\_message\_decoded\_payload\_decoded\_latitud"],

      longitude: r["uplink\_message\_decoded\_payload\_decoded\_longitud"]

    }))

  |> keep(columns: ["\_time", "latitude", "longitude"])

Desde el inicio hasta la línea 10 aparecen las funciones necesarias para hacer la petición a la base de datos, siendo las restantes necesarias para separar latitud y longitud, de forma que sea sencillo tratar con ellos.

Grafana dispone de diferentes estilos de gráficos, pero el que interesa en este proyecto es el “Geomap”, un mapa donde se mostrarán las coordenadas como puntos.

A la derecha aparece la personalización del visor. Se mostraran los puntos con la opción “route”, que unirá los puntos que el software vaya recibiendo, formando una ruta aproximada. En la zona superior se muestra un desplegable con el que se puede modificar la cantidad de datos representados, en función del tiempo seleccionado.

FOTO DE UNA RUTA

No obstante, que solo muestre las coordenadas del autobús no es suficiente. También es importante que se muestre la localización de las paradas, de forma que el usuario tenga una estimación de cuanto tardará en llegar el vehículo a la parada que desea. Para esta tarea, se usarán los archivos GTSF.

El estándar GTSF (General Transit Feed Specification) es un formato de datos estandarizado y abierto que se utiliza para almacenar y compartir información sobre el transporte público. Las agencias de transporte público pueden publicar información sobre tránsito en este formato, de modo que terceros pueden acceder a ellos para utilizarlos en aplicaciones de software. A efectos prácticos, no son más que documentos CSV estructurados y organizados de forma estandarizada y sencilla, facilitando su uso.

Desde la web oficial de la [Red de Consorcios de Transporte de Andalucía](https://api.ctan.es/), se obtuvieron los archivos GTFS de la comunidad. Aunque este proyecto podría implementarse en cualquier zona, se seleccionó solo Andalucía para mostrar su funcionamiento, pues para el resto de comunidades se haría de igual forma.

Desde la web se descargó un archivo comprimido con toda la información en su interior con la forma estandarizada de GTFS:

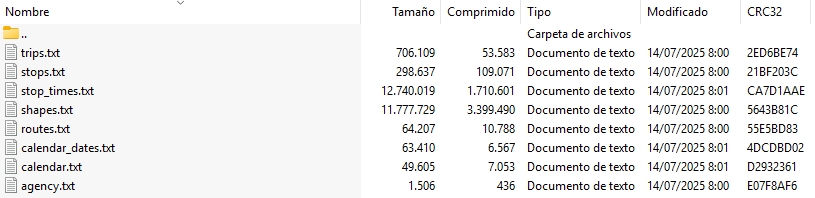


Ilustración 28. Carpeta comprimida con los archivos GTFS de Andalucía.

Para este proyecto interesa realmente las paradas, aunque el resto de información podría ser útil más adelante. Viendo el archivo “stops.txt”:

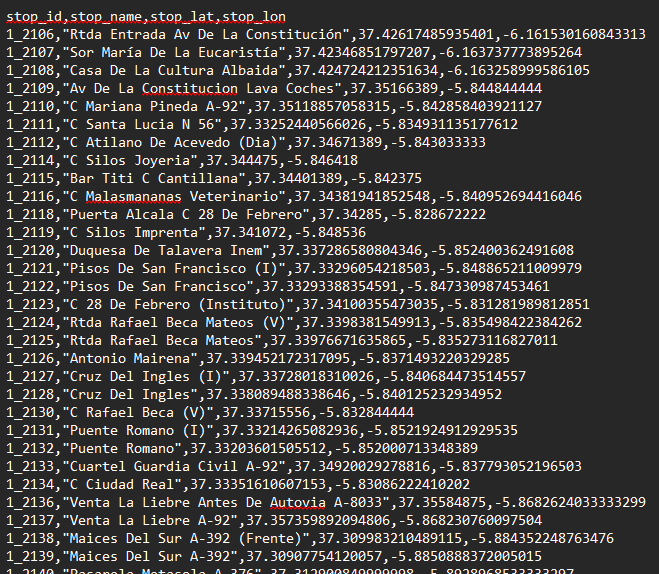


Ilustración 29. Archivo "stops.txt", en formato CSV

Se ve como este es un archivo CSV con la identificación de las paradas, su nombre y coordenadas. Esto será muy útil, pues Grafana permite implementar este tipo de archivos con formato GEOjson. Para pasarlo de txt a GEOjson, se desarrolló un pequeño código en python: NO SE SI DEJARE EL CODIGO

import csv

import json

geojson = {

"type": "FeatureCollection",

"features": []

}

with open('stops.txt', newline='', encoding='utf-8') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

feature = {

"type": "Feature",

"geometry": {

"type": "Point",

"coordinates":[float(row["stop\_lon"]), float(row["stop\_lat"])]

},

"properties": {

"stop\_id": row["stop\_id"],

"stop\_name": row["stop\_name"]

}

}

geojson["features"].append(feature)

with open("stops.geojson", "w", encoding='utf-8') as outfile:

json.dump(geojson, outfile, indent=2)

Este simplemente hace la conversión a GEOjson, para poder usarlo en Grafana. Una vez convertido, se sube a un servicio en la nube como github para que Grafana pueda a acceder a él. Se crea una nueva capa del tipo geojson y se pega en enlace de github, apareciendo las paradas marcadas:

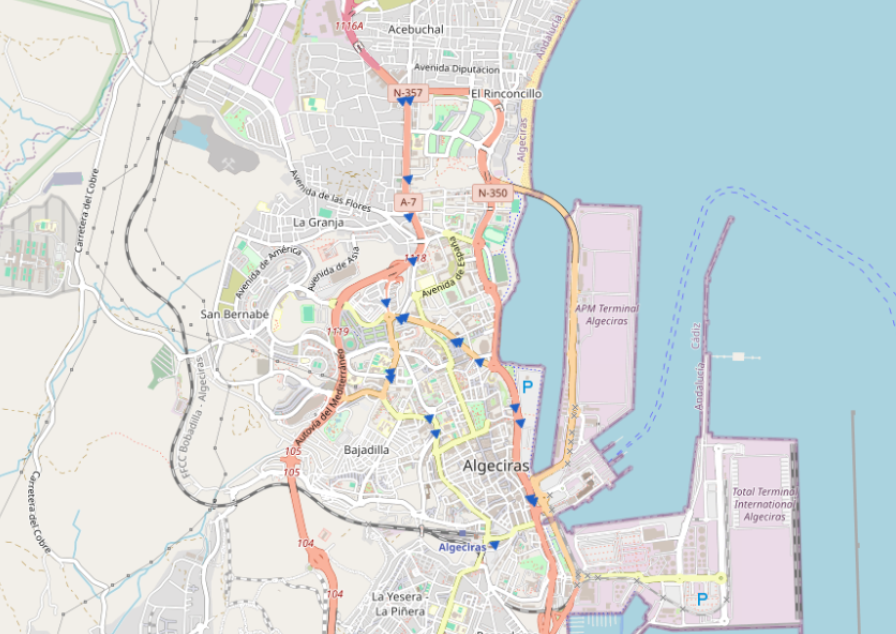


Ilustración 30. Paradas de Algeciras mostradas en el mapa

Siendo que ya se visualiza en el mapa tanto la posición actual del vehículo como la de las paradas, queda estudiar como realizar la temporalización para obtener una medida del tiempo restante para llegar a cada una de estas.

Los programas que se utilizan no proveen de herramientas demasiado potentes para realizar cálculos, aunque se puede realizar una aproximación en Grafana. Para hacer algo más concreto, habría que realizar un programa externo que se encargue unicamente de este papel. Esto genera un problema, pues si se hacen los cálculos con el Seeeduino, el mensaje a enviar sería mayor y limitaría la cantidad de mensajes que se podrían enviar. Por lo tanto, se tratará de buscar una solución evitando utilizar el microcontrolador.

* + 1. Programa de cálculo del tiempo de llegada

La solución adoptada para el problema del cálculo del tiempo de llegada a la siguiente parada es la creación de un programa externo que se encargue de ello, pues no se podría implementar con las etapas ya establecidas en el proyecto sin aumentar la carga del mensaje en gran medida.

En etapas anteriores del desarrollo del proyecto, previo al diseño de este programa, se enviaba por LoRaWAN únicamente las coordenadas del Seeeduino. Sin embargo, al no poseer un histórico de medidas del tiempo que transcurre de una parada a otra, se decidió utilizar la velocidad del vehículo que también puede obtenerse con el GPS. Por tanto, se hizo inevitable el aumento del tamaño del payload a enviar, aunque debido a la menor necesidad de precisión del valor, se puede utilizar solo un byte para enviarlo, volviendo este aumento prácticamente despreciable para la red.

Ya se vio como se toman las medidas de velocidad en el punto 1.8.2. Se recordará con el fin de facilitar la lectura de este apartado. Teniendo en cuenta que el Seeeduino enviará su posición cada 5 minutos aproximadamente, para la velocidad se toma una medida cada minuto durante ese tiempo, obteniendo así 5 medidas. Estas se promedian y se envían cuando sea el momento de enviar las coordenadas.

Se utilizará la base de datos para tratar las medidas de velocidad, utilizando el código en Python descrito a continuación, que se explicará por partes para facilitar su comprensión:

# Main #

# Configuración inicial

[writer, reader, org, bucket] = config\_influx()

stops = read\_file()

last\_data = None # Variable para almacenar los últimos datos leídos

while True:

data\_influx = read\_influx(reader, org)

if data\_influx == last\_data:

print("No hay nuevos datos en Influx. Esperando...")

time.sleep(1\*60)

continue

lat\_bus = data\_influx[0].records[0].get\_value()

lon\_bus = data\_influx[1].records[0].get\_value()

v\_bus = data\_influx[2].records[0].get\_value()

[\_, dist\_parada] = parada\_mas\_cercana(lat\_bus, lon\_bus, stops)

tiempo\_restante = calc\_tiempo(dist\_parada, v\_bus)

print(f"Tiempo restante para llegar a la parada más cercana: {tiempo\_restante:.2f} segundos")

write\_influx(writer, bucket, org, tiempo\_restante)

time.sleep(1)

print("Datos escritos en Influx")

last\_data = data\_influx

time.sleep(5\*60)

El en el main se llaman a las funciones que se explicarán más adelante. Primero se llama a “config\_influx”, donde se prepara el cliente de InfluxDB de la librería “influxdb-client”. Este permite la comunicación con diversas bases de datos de forma sencilla, pues a partir de cada cliente se puede crear un objeto para hacer querys, así como para escribir datos. En caso de que no se pueda conectar a influx correctamente, volverá a intentarlo

Tras esto, se llama a “read\_file”, función que abre el archivo “stops.csv”, el mismo que se utiliza en Grafana para mostrar la localización de las paradas. En este caso se utiliza como csv debido a que es más sencillo trabajar con él de esta forma. La función devuelve un array de diccionarios de python con toda la información contenida en el archivo.

Lo siguiente es el bucle principal. En este se inicia con la función “read\_influx”, la cual utiliza un query escrito en Flux para leer la última medida recibida en la base de datos, es decir, la de los últimos 5 minutos. En la variable de salida de la función se almacena el resultado de la petición. Habiendo leído la ultima media, se comprueba que sea diferente de la de los últimos 5 minutos mediante la marca de tiempo. En caso de que sea la misma o no haya recibido nada, espera 30 segundos antes de volver a intentarlo. Esto puede pasar por lo que ya se habló, que en algún momento se pierda la señal, o se debilite, y se pierda el mensaje.

Si se ha leído un mensaje nuevo, se accede a las coordenadas y a la velocidad guardadas, así como a la hora de llegada de dichos datos. Los primeros se utilizan en la siguiente función, “parada\_mas\_cercana”. Junto con la función semiverseno, concepto que ya se comentó, tiene el papel de calcular cuál es la parada de autobús más cercana a las coordenadas del vehículo. La función semiverseno en este caso se calcula con una librería llamada “haversine”, que ya hace el cálculo.

Conociendo cuál es la parada más cercana y la distancia a esta, se llama a la función “calc\_tiempo”, que calcula el tiempo restante hasta esa parada usando la velocidad.

Lo siguiente es llamar a “write\_influx”, que envía este tiempo a Influx, de forma que se pueda acceder a el desde Grafana fácilmente para mostrarlo en el visualizador.

Por último, se ajusta la espera hasta la siguiente ejecución del programa. Para ello, se utiliza la hora de llegada a influx de las últimas coordenadas tratadas. Se le suman 2 o 5 minutos en función de si hubo cambio brusco o no, obteniendo la hora a la que el Seeeduino debería realizar el siguiente envío. Por tanto, se compara la hora actual con la calculada, esperando el tiempo necesario. En caso de que el Seeeduino no haya realizado el envío justo a los 5 minutos, o se hayan perdido los primeros paquetes, el programa esperará los 30 segundos que se comentaron anteriormente.

Con las medidas de tiempo en Influx, se crea otro panel en Grafana para mostrar el tiempo, quedando algo así al final:



Ilustración 31. Panel de visualización de Grafana.

A la izquierda se tiene el mapa con la ubicación del Seeeduino, mientras que a la derecha se tiene el tiempo hasta la siguiente parada, así como una breve descripción

* + 1. Implementación física

Habiendo hablado ya de la parte técnica del proyecto, se procederá a comentar la instalación física del Seeeduino en el vehículo, comentando las distintas posibilidades estudiadas.

El dispositivo deberá estar protegido por una caja o algún tipo de recipiente, con aberturas para acceder al puerto de alimentación así como para la antena LoRaWAN. Este deberá disponerse en la zona del asiento del conductor, de forma que se pueda conectar de forma sencilla. A poder ser, deberá colocarse junto a la ventana, para facilitar que el Seeeduino pueda obtener la señal GPS de la mejor forma posible.

# Bibliografía METER LO DEL SEMIVERS4ENO, LO DE GTSF

*Arduino*. (s.f.). Obtenido de Arduino.cc: https://www.arduino.cc/

*Chronograf*. (s.f.). Obtenido de Influxdata: https://www.influxdata.com/time-series-platform/chronograf/

*Compatible IBM PC*. (s.f.). Obtenido de wikipedia.org: https://es.wikipedia.org/wiki/Compatible\_IBM\_PC

*Duty Cycle*. (s.f.). Obtenido de thethingsnetwork.org: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/

*End Device Activation*. (s.f.). Obtenido de thethingsnetwork.org: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/

Gunther Gridling, B. W. (2006). *Introduction To Microcontrollers.* Viena: Vienna University of Technology.

*LoRa + MQTT, una potente combinación de dos tecnologías para IoT*. (s.f.). Obtenido de internetdelascosas.xyz: https://internetdelascosas.xyz/articulo.php?id=2873&secc=TUTORIAL

*LoRaWAN EU868 vs. LoRaWAN US915 : Key Differences and Considerations*. (s.f.). Obtenido de Enless Wireless: https://enless-wireless.com/lorawan-eu868-us915-differences/

*Memoria Flash*. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/w/index.php?title=Memoria\_flash&oldid=165411678.

*Microcontroladores y sus aplicaciones*. (s.f.). Obtenido de microcontroladoressesv.wordpress.com: https://microcontroladoressesv.wordpress.com/

*Seeeduino LoRaWAN*. (s.f.). Obtenido de wiki.seeedstudio.com: https://wiki.seeedstudio.com/Seeeduino\_LoRAWAN/

The Things Network. (s.f.). *Learn: The Things Network*. Obtenido de thethingsnetwork.org: https://www.thethingsnetwork.org/docs/

# Anexos

## Datasheet Seeeduino LoRaWAN

