UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA:

INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de:

INGENIERO ELECTRÓNICO

TEMA:

DESARROLLO DE UN RASTREADOR GPS PARA UBICACIÓN Y DESPLAZAMIENTO DE NIÑOS EN RUTAS O ZONAS CONFIGURADAS EN UNA APLICACIÓN

AUTOR:

REA ESPINOZA ALEX DAVID

DIRECTOR:

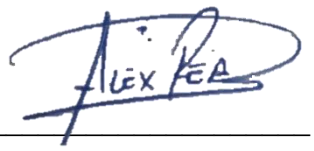
LENIN WLADIMIR AUCATOMA GUAMÁN

QUITO, AGOSTO DE 2021

CESIÓN DE DERECHOS DE AUTOR

Yo, Alex David Rea Espinoza, con documento de identificación N° 1720644275, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del trabajo de titulación titulado: DESARROLLO DE UN RASTREADOR GPS PARA UBICACIÓN Y DESPLAZAMIENTO DE NIÑOS EN RUTAS O ZONAS CONFIGURADAS EN UNA APLICACIÓN, mismo que ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.



Alex David Rea Espinoza

C.I.: 1720644275

Quito, agosto de 2021

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el Proyecto Técnico, DESARROLLO DE UN RASTREADOR GPS PARA UBICACIÓN Y DESPLAZAMIENTO DE NIÑOS EN RUTAS O ZONAS CONFIGURADAS EN UNA APLICACIÓN, realizado Alex David Rea Espinoza, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerado como trabajo final de titulación.

Quito, agosto de 2021.

Lenin Wladimir Aucatoma Guamán

Cédula: 1717985830

DEDICATORIA

Este trabajo es dedicado principalmente a mis padres Angel Cristóbal y María Eugenia quienes me dieron la oportunidad de desarrollarme en mi vida estudiantil formándome y acompañándome en todo momento pese a las dificultades.

A mi pareja Stephy por el amor y ser quien me ha motivado y dado fuerzas para no tirar la toalla, mi hijo Martín que se convirtió en la mayor motivación para lograr culminar mi carrera.

A mis hermanos Miguel y Andrés, mi cuñada Joan y mi sobrino Iker por acompañarme como familia a lo largo de este camino.

AGRADECIMIENTO

A Dios por permitirme culminar una etapa más de mi vida y poder compartirlo con toda mi familia, a mis padres Angel y María por todo el esfuerzo que han realizado para poder estudiar. Darme la confianza y apoyo durante este proceso. A Stephy por el amor, comprensión y sacrificio que me permitió tener mi propio hogar y culminar mis estudios a la vez.

Al Ingeniero Lenín Aucatoma MSc. por la confianza, apoyo y todo su conocimiento brindado para el desarrollo de este proyecto.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

La falta de seguridad que existe en el país, las constantes denuncias de la desaparición de niños es una realidad en la que vivimos. El uso de la tecnología puede llegar a servir como solución ante estos hechos.

Este proyecto tiene como objetivo el desarrollo e implementación de un dispositivo de rastreo de niños mediante una aplicación Android, el dispositivo modular capaz de ser ubicado en una mochila utilizará la tecnología de red Low Power Wide Area Network (LPWAN), Sigfox para la transmisión de datos. El dispositivo utiliza dos módulos electrónicos principales; el primero es un módulo LoPy Pycom 4 encargado de la transmisión de los datos por la red Sigfox además de integrar un microcontrolador necesario para el control del envío; y, el segundo un módulo GPS NEO6MV2 que se encargara obtener los datos de los satélites para dar con precisión la ubicación del dispositivo. Este dispositivo enviará la información a un servidor de la red y a la vez esta será enviada a un servidor propio, el cual manejará esta información y la representará en una aplicación Android desarrollada para el monitoreo del dispositivo.

El dispositivo fue sometido a pruebas de verificación de funcionamiento en gran parte de Quito. Obteniendo mediante un barrido un funcionamiento variable por sectores donde factores como: el clima, línea de vista y velocidad de desplazamiento influyen directamente en la calidad de la señal de la red y el envío de mensajes que son la base para mostrar la ubicación en tiempo real.

ABSTRACT

The lack of security in the country, the constant reports of disappearance of children, is the reality we face day by day. The use of technology can serve as a solution to these facts.

This project aims to develop and implement a child tracking device through an Android application, the modular device capable of being placed in a backpack will use the Low Power Wide Area Network (LPWAN) Sigfox network technology for data transmission. The device uses two main electronic modules; the first is a LoPy Pycom 4 module in charge of transmitting the data through the Sigfox network, as well as integrating a microcontroller necessary to control the shipment; and the second, a NEO6MV2 GPS module that will oversee obtaining the data from the satellites to accurately determine the location of the device. This device will send data to a network server and at the same time it will be sent to its own server, which will handle this information and will represent it in an Android application developed for monitoring the device.

The device was subjected to functional verification tests in much of Quito. Obtaining by means of a sweep of this a variable operation by sectors and in addition to other factors such as weather, line of sight and travel speed that directly influence the quality of the network signal and the sending of messages that are the basis for displaying the location in real time.

CAPÍTULO 1

ANTECEDENTES

1.1 PLANTEAMIENTO DEL PROBLEMA

En el Ecuador diariamente se pierden entre 30 y 35 niños, en su mayoría niños, niñas y adolescentes de entre 7 y 14 años, de cualquier estrato social. Sin embargo, los niños de bajos recursos son los más propensos al secuestro ya que por lo general son quienes deben regresar a sus casas por su cuenta, tomando transporte público o caminando solos. Debido a su inocencia es cuando los secuestradores aprovechan para mediante engaños o a la fuerza llevárselos para obtener un beneficio de aquello, ya sea secuestrándolos para obtener dinero o peor aún venderlos o utilizarlos para mendigar o prostituirse. (Hora, 2020).

Este es un problema que en la actualidad es muy común que tanto padre como madre deban trabajar, dejando así a sus hijos a cargo de las escuelas o colegios durante un periodo de tiempo prolongado y sin saber cómo sus hijos regresan a casa o al lugar destinado después de la jornada escolar, ya sea solos, a cargo de recorridos o de terceros y sin un monitoreo constante de la ruta normal de regreso.

A nivel mundial los sistemas de localización mediante Global Positioning System (GPS) se han usado en su mayoría en el rastreo de bienes (vehículos móviles, bicicletas, contenedores, etc.) muchos de estos están utilizan los sistemas de telefonía móvil, lo que implica dependencia de la cobertura de esta red y gastos de esta, lo que implica que cada dispositivo deberá tener por lo menos un módulo de la red Global System for Mobile Communications (GSM) con su correspondiente chip de pago, lo cual suele resultar costoso.

1.2 JUSTIFICACIÓN

Al tratarse de una problemática social tan grave como es la desaparición diaria de niños, niñas y adolescentes desde o hacia sus centros educativos surge la necesidad de utilizar diversas herramientas tecnológicas que ayuden a prevenir y alertar a las familias sobre posibles intentos de secuestro, violación etc.

Se desarrollará un dispositivo electrónico que podrá ser colocado en las mochilas de niños de forma oculta que permitirá a los padres o tutores del niño determinar una ruta específica que deberán seguir para llegar a sus casas o centros educativos alertando a padres o tutores en caso de un cambio de la misma. Además, este dispositivo les permitirá monitorear en tiempo real la ubicación del niño/a y enviar notificaciones de posición, salida de áreas seguras, desvío de ruta normal de movimiento esto en intervalos de tiempo preestablecidos y/o alertas automáticas. Debido a que las soluciones actuales basan su tecnología de transmisión por la red GSM se complica el rastreo en tiempo real ya que la búsqueda da la última posición de un terminal GSM depende de procesos de triangulación que tiene una espera tecnológica, pero sobre todo administrativa para el acceso a esa información, tiempo valioso que se desperdicia al no conocer el paradero del menor. Incluyendo el factor de que no todos los niños poseen teléfonos inteligentes.

El dispositivo que se pretende diseñar se basa en tecnología Sigfox, la cual ya se encuentra implementada en el país, pero cuyas aplicaciones aún no se encuentran desarrolladas. Al utilizar dispositivos inalámbricos de largo alcance y bajo consumo, las distancias que puede abarcar el dispositivo serán grandes.

Con el avance de la tecnología inalámbrica es posible usar módulos de la banda Industrial, Scientific and Medical (ISM) de largo alcance y bajo consumo para poder registrar el desplazamiento de un dispositivo a través de los diferentes puntos en que se encuentran las antenas receptoras, estas tecnologías están siendo desarrolladas para la nueva era de Internet of Things (IoT), en español internet de las cosas, por lo que se podría usar esta tecnología para monitorear personas y objetos.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Desarrollar un dispositivo de rastreo de niños para la monitorización de desplazamiento y ubicación mediante el uso de redes Sigfox.

1.3.2 OBJETIVOS ESPECÍFICOS

- Analizar las tecnologías de geolocalización y de las redes de largo alcance y bajo consumo para establecer el marco conceptual sobre el cual se definiría la estructura del prototipo.

• Diseñar un dispositivo que permita la lectura de posicionamiento basado en GPS para el envío de datos mediante una red Sigfox a una aplicación para visualización de ubicación y configuración de rutas y notificaciones.

• Implementar el dispositivo electrónico junto con la aplicación móvil para aprobar el funcionamiento del monitoreo de ubicación.

• Realizar las pruebas para la validación de las características técnicas de operación mínimas que tendrá el prototipo propuesto.

1.4 METODOLOGÍA

Para realizar el proyecto se usará el método analítico en donde se analizarán cada uno de los componentes que conformarán el dispositivo de forma individual, se harán experimentos de funcionamiento para detectar posibles comportamientos anómalos de los elementos o para detectar mejores formas de conexión o uso, (buscando la optimización de recursos) de los módulos en el dispositivo final, de modo que el proyecto pueda ser explicado desde el funcionamiento de sus conjuntos pequeños hasta explicar de mejor manera el funcionamiento del proyecto en general.

También se usará como metodología el método sintético, ya que al ser un proyecto técnico este que clasificará, cada una de sus partes en diferentes sistemas como es el conjunto de módulo de comunicación, módulo de control y lectura del GPS junto con el sistema de alimentación del dispositivo, además ayudará a reconstruir todos los sistemas individuales y formar en un todo, el sistema completo.

Al usar la metodología hipotética-deductiva se espera poder observar y reflexionar sobre el comportamiento de los diferentes sistemas que componen el conjunto del proyecto. Esto ayudará a examinar los diferentes problemas que se presenten en la creación del dispositivo, encontrar errores no evidentes en el funcionamiento del prototipo, además se espera que usando este método ayude a buscar problemas que no se tuvieron en cuenta durante el diseño y construcción y se comparará el funcionamiento de acuerdo con ciertas puestas en marcha para probar el dispositivo.

1.5 PROPUESTA DE SOLUCIÓN

Se fabricará un dispositivo electrónico el cual permitirá el rastreo de la ubicación de niños y adolescentes, basado en tecnología GPS para rastreo y posicionamiento de este,

utilizando tecnología ISM para la comunicación de redes Sigfox de largo alcance y bajo consumo de energía para el envío de datos.

Para la fabricación del dispositivo se utilizará un microcontrolador en el cual se establecerán los algoritmos necesarios para la determinación de posición GPS y comunicación en el marco de las propiedades específicas de una red de largo alcance y bajo consumo.

Los datos se enviarán a una aplicación en la cual se podrá monitorear en tiempo real la ubicación del dispositivo, además se programará esta de tal forma que se pueda configurar las rutas o zonas tantas como se desee: el colegio, domicilio, clases extras, casa de familiares. Así notificando al salir de estas zonas o desvío de las rutas establecidas.

CAPITULO 2

MARCO CONCEPTUAL

En este capítulo se detallará el marco teórico referente a redes Sigfox, su funcionamiento y los elementos que componen el desarrollo de este proyecto.

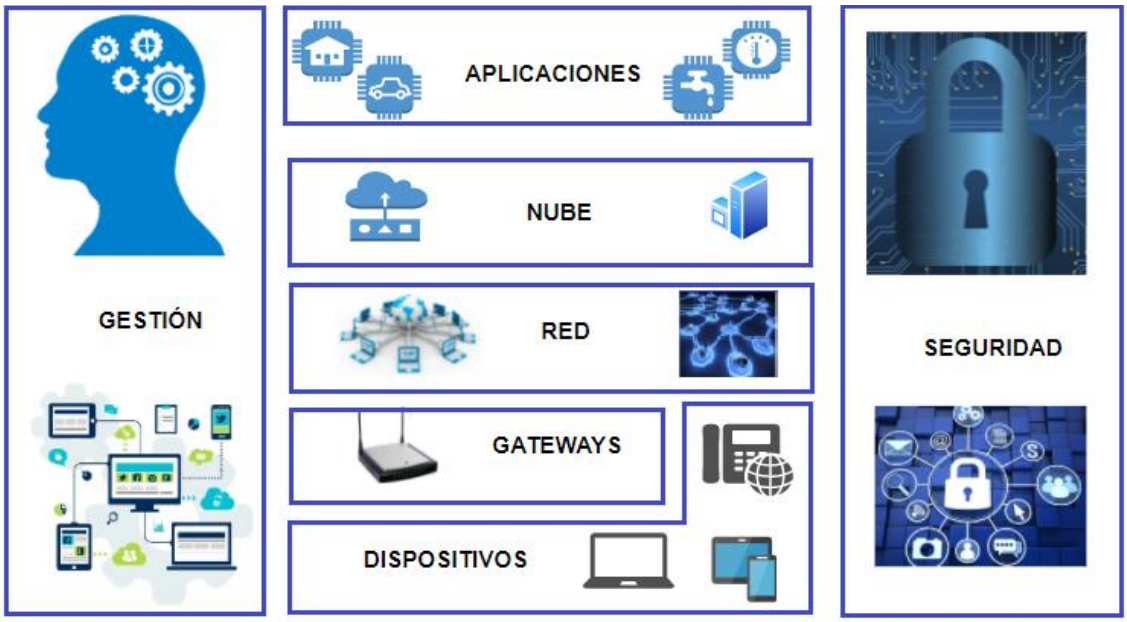
2.1 INTERNET DE LAS COSAS

El desarrollo tecnológico de la humanidad avanza rápido y simultáneamente las necesidades de los usuarios, esto ha provocado la necesidad de la creación de tecnologías que permitan la conexión de estos y sus objetos (cosas). Entendiendo de esta manera al "Internet de las cosas, IoT como un conjunto de objetos físicos conectados a una red para intercambiar datos a través de Internet (nube) que depende de una gran cantidad de tecnologías para su uso además de las interfaces de programación de aplicaciones (API) que se encargan de conectar los dispositivos a Internet" (SAP, 2016).

Al ser un concepto muy general las aplicaciones y usos de IoT en la actualidad incluyen varios sectores, desde lo más básico posible como un foco hasta incluso llegar a algo tan grande como una industria de producción. Es así, como es el sector privado en donde cada vez toma más fuerza y se hace más popular el uso de IoT. En la actualidad existe aplicaciones de uso de IoT en la industria automotriz, control de procesos de fabricación, control de sensores, permitiendo a las empresas centralizar el control de las infraestructuras. En el sector de la salud, sistemas que permiten a las clínicas y hospitales el monitoreo de pacientes de manera ambulatoria. Y un sin número de usos y aplicaciones en diferentes áreas (SAP, 2016).

Una de las soluciones de IoT incluye el rastreo y localización, esta es utilizada para diferentes servicios o casos de uso. Por ejemplo: aplicaciones que proporcionen información con acceso continuo a la localización de personas en tiempo real para el control y monitoreo de desplazamientos, tales como los trayectos que un niño hace desde su domicilio hacia su escuela y viceversa, trayecto a su curso de música o escuela de futbol y recibir alertas de emergencia o notificaciones en caso de que ocurriera algo extraño, permitiendo a los padres gozar de tranquilidad.

Figura 2.1 Arquitectura de Internet de las Cosas.



Elaborado por: Rea Alex

En la tabla 2.1 se muestra las características principales de tecnologías LPWAN de redes inalámbricas aplicadas a IoT, todas con características para una aplicación de localización geográfica. Observando de manera especial y teniendo en cuenta la relevancia de Sigfox ante las otras ya que es la tecnología de red propuesta para el proyecto. (Sanchez, J., 2020).

Tabla 2.1 Comparativo de tecnologías inalámbricas de largo alcance y aplicables a IoT (Sanchez, J., 2020).

Tecnología	LoRa		Sigfox		NB-IoT versión 13		GPRS	
	UL	**DL**	**UL**	**DL**	**UL**	**DL**	**UL**	**DL**
Espectro [MHz]	863-870	863-870	868.1-868.3	868.425-869.625	832-862	791-821	890-915	935-960
Potencia de transmisión[dBm]	14	14-27	14	27	23	37	33	37
Modulación	Chip extendido espectro		DBPSK	GFSK	GMSK	SC-FDMA	GMSK	GMSK
Ancho de banda [KHz]	125	125	0.1	0.6	180	180	200	200
Carga útil máxima [bytes]	51	51	12	8	128	85	22	22
Planificación	Enlace ascendente iniciado (Clase A)		Enlace ascendente iniciado		Red programada		Red programada	
MCL, Pérdida Máxima de Acoplamiento[dB	154	152	158	161	164	164	144	152

2.2 SIGFOX

Sigfox se trata de una red inalámbrica que conecta el mundo físico con el mundo virtual gracias al Internet de las cosas. Es una tecnología dedicada a aplicaciones de transmisión de datos de baja potencia y de largo alcance. "Ahora presente en más de 70 países, Sigfox es el primer jugador en construir el ecosistema de IoT más grande del mundo, desde grandes fabricantes hasta cientos de nuevas empresas y fabricantes de dispositivos en cuatro continentes" (Sigfox, 2020).

Figura 2.2 Presencia de Sigfox a nivel mundial hasta 2020.

Elaborado por: Rea Alex

Esta red utiliza tecnología Ultra Narrow Band (UNB) para el transporte la misma que es una tecnología de modulación ISM que opera sin licencia. Transmite desde 1KHz y alcanza distancias de 5Km en zona urbana y 25Km en campo abierto. (Sigfox, 2020)

La red Sigfox se caracteriza por:

- Ser una red de comunicación inalámbrica.
- Bajo consumo de potencia y largo alcance.

- Cobertura amplia.

El plus principal de Sigfox es la reducción de costos en comparación de otras tecnologías de red, pero a la vez ofrece únicamente el envío de información básica, no está pensada para la recolección de información en grandes servidores sino más bien para la conexión entre dispositivos e intercambio de información (Sanches J., 2016).

2.2.1 TOPOLOGIA DE LA RED SIGFOX

Sigfox simplifica la arquitectura de la red, es así como la topología estrella de la red Sigfox es simple. A diferencia de una red celular en Sigfox los dispositivos no se encuentra asociados a una sola estación base, debido a su alcance el mensaje enviado por un dispositivo es recibido por cualquier estación base, esto se puede explicar de mejor manera en tres pasos (Sigfox, 2020).

1. Los dispositivos (cosas) transmiten un mensaje a la red Sigfox a una estación base dentro del rango de cobertura.
2. La estación base que ha recibido el mensaje, cualquiera alrededor del mundo implementada por un operador Sigfox está conectada por medio de un enlace punto a puto con la Nube Sigfox. La cual recibe el mensaje de parte de la estación base.
3. La nube Sigfox se encargará de enviar el mensaje a los servidores o a las plataformas de aplicaciones IoT de los usuarios.

Figura 2.3 Arquitectura de la red Sigfox.

Elaborado por: Rea Alex

2.2.2 TECNOLOGÍA RADIO

Para la transmisión del mensaje la tecnología de Sigfox utiliza modulación Differential Binary Phase-Shift Keying (DBPSK) además de operar en la banda de frecuencias ISM es decir en la banda de frecuencias sin licencia esto conlleva a un consumo de ancho de banda muy reducido, ocupando menos espacio en el rango de frecuencia a diferencia de otras tecnologías LPWAN. Cada mensaje tiene 100Hz de ancho y se transmite a una velocidad de 100 o 600 bits según la región (Sigfox,2020).

Sigfox maneja la transmisión de mensajes pequeños, un mensaje en un enlace ascendente tendrá una extensión de 12 bytes y para un enlace descendente de 8bytes, una trama Sigfox utiliza 26 bytes en total (Sigfox,2020).

Un dispositivo transmite un mensaje en un promedio de 3 veces a 3 diferentes frecuencias (salto de frecuencias), es decir que la estación base no está sincronizada con el dispositivo y la red. Las estaciones bases siempre se encuentran buscando en el espectro señales UNB para poder demodularlas.

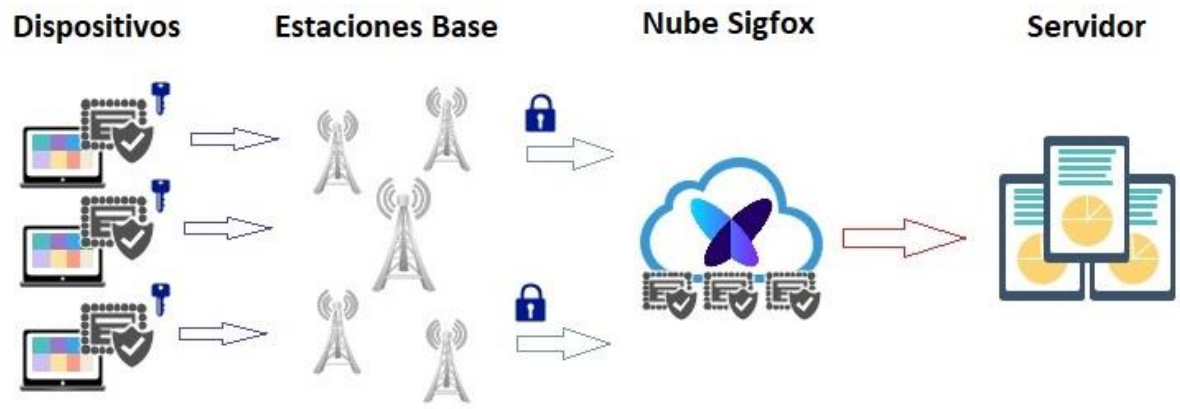
2.2.3 SEGURIDAD DE LA RED

Dentro de la red Sigfox intervienen los dispositivos, la tecnología de radio, estaciones base y la nube. Cada uno de los elementos que conforman esta red son seguros, robustos, escalables y confiables, en las aplicaciones o plataformas de los clientes se encuentra una conexión a la nube de Sigfox mediante interfaces cifradas mediante Protocolo de Transferencia de Hipertexto Seguro (HTTPS) (Sigfox, 2020).

El uso de conectividad mediante un enlace descendente proporciona seguridad adicional, cuando los objetos no están escuchando, no podrán recibir algo que envíe un hacker. Los objetos eligen cuando comunicarse esto los protege de un ataque de un pirata informático.

Sigfox junto con su equipo de profesionales dentro de la industria se encarga de todos los detalles tanto de la seguridad por el diseño y la operación de la red. Se encarga de la protección de los datos mientras se transmiten por intermedio de medidas dentro del protocolo (integridad, cifrado, autenticación, anti-jamming, anti-reproducción), protección de los datos en reposo utilizando credenciales y almacenamiento criptográfico todo esto en los dispositivos, estaciones base y la nube como se observa en la Figura 2.4 (Sigfox, 2020).

Figura 2.4 Representación de la seguridad de una red Sigfox.



Elaborado por: Rea Alex

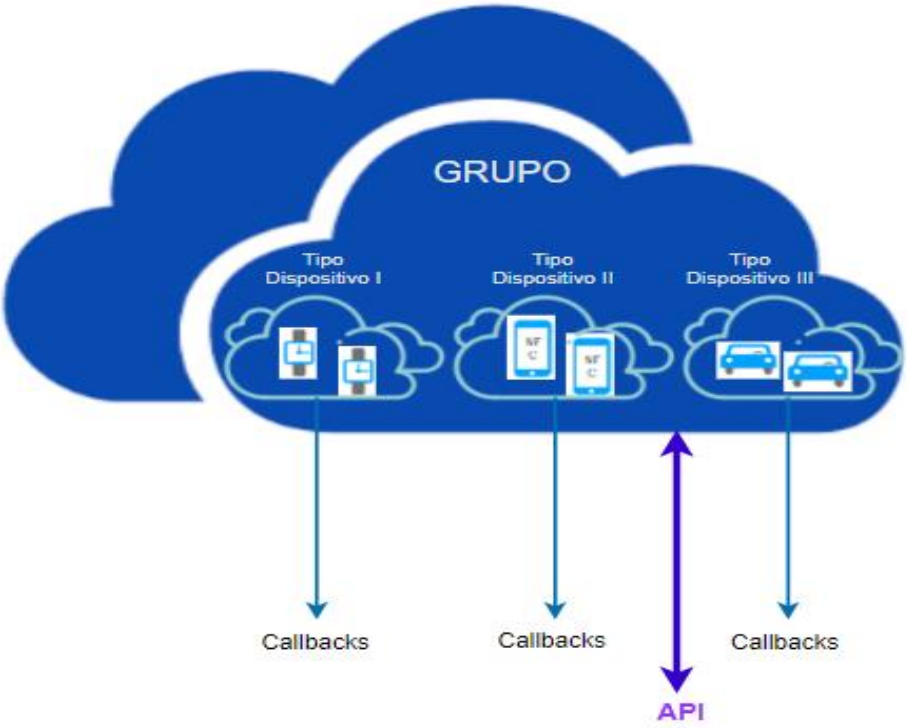
2.2.4 NUBE DE SIGFOX

La nube de Sigfox corresponde al centro en donde se guarda y almacena la información de un dispositivo conectado a la red. Estos datos se encuentran disponibles para su utilización y administración desde su plataforma (Sigfox Cloud Integration | Sigfox build, 2020).

La plataforma de Sigfox integra un enlace con la nube (Sigfox Cloud) con el propósito de que exista el intercambio de datos y la comunicación entre la nube y su servidor, para este propósito existen dos métodos: la API REST y la API de devolución de llamada. De tal forma que la devolución de llamada recupera los mensajes de un dispositivo mientras que la API REST administra varios dispositivos (Sigfox Cloud Integration | Sigfox build, 2020).

La gestión de nube Sigfox se la hace de una forma ordenada que se explica de la siguiente manera, una compañía se representa como un grupo y es la parte más general y grande de la nube, que contiene diferentes "tipos de dispositivos" (al menos uno) que se agrupa por similitud de características "familia" de dispositivos que se comportan de la misma manera al momento de intercambiar mensajes por la red Sigfox. De forma gráfica se presenta en la Figura 2.5 (Sigfox Cloud Integration | Sigfox build, 2020)

Figura 2.5 Representación de la nube de Sigfox



Elaborado por: Rea Alex

2.2.4.1 DEVOLUCIONES DE LLAMADA

La devolución de llamada corresponde a mensajes de notificación que están disponibles por los tipos de dispositivos y se activan cuando cualquiera de estos envíe un mensaje. El servidor únicamente obtiene el mensaje (los datos) y no administra los dispositivos en la nube. La nube al recibir un mensaje automáticamente responde con un mensaje de devolución de llamada directo al servidor (Sigfox Cloud Integration | Sigfox build, 2020).

Existen tres tipos de devoluciones de llamada. A continuación, en la Tabla 2.2 se detalla los mismos.

Tabla 2.2 Tipos de devoluciones de llamada (Sigfox Cloud Integration | Sigfox build, 2020).

DEVOLUCIÓN DE LLAMADA		DESCRIPCIÓN
DATOS	ENLACE ASCENDENTE	Los mensajes de devolución de llamada son enviados de forma ascendente a un servidor.
	BIDIR	Los mensajes de devolución de llamada son enviados de forma ascendente al servidor, pero además responde un mensaje de forma descendente desde el servidor.
SERVICIO	ESTADO	En el mensaje envía información sobre el dispositivo, son conocidos como mensajes de mantenimiento.
	RECONOCER	Envía un acuse de recibo de la red, cuando se transmite mensajes de enlace descendente.
	DATA_ADVANCED	Para el envío de datos como la geolocalización, se lo realiza con retraso ya que se lo calcula en función de las estaciones bases que utilizo la red.
ERROR		En caso de corte de comunicación, informa si se trata de un error en la red o el dispositivo.

Elaborado por: Rea Alex

2.2.4.2 API REST

La API REST permite el control de varios dispositivos en la nube, directamente de la plataforma. El servidor realiza requerimientos de los datos de la nube de Sigfox a través de la API (Sigfox Cloud Integration | Sigfox build, 2020).

Existen dos versiones API v1 y API v2 de las cuales API v1 se encuentra desactualizada y obsoleta ya que la versión 2 ofrece el uso de mensajes HTTPS simples (GET, POST,

PUT y DELETE), estándar de especificaciones y mayores funcionalidades de administración de dispositivos (Sigfox Cloud Integration | Sigfox build, 2020).

2.3 TECNOLOGÍAS DE GEOLOCALIZACIÓN.

Como indican sus siglas GPS o Sistema de Posicionamiento Global fue creado por el gobierno de los Estados Unidos y operado por el ejecito de este en el año 2000 fue liberado al mundo para el uso civil. Permitiendo así una gran cantidad de aplicaciones de este, pero en su mayoría concentrándose en encontrar soluciones de posicionamiento (rastreadores) inalámbrico y con funcionamiento en largas distancias para el empleo de la industria. Así mismo surgió la fabricación y venta de soluciones RTLS (Sistema de localización en tiempo real) para el rastreo de objetos y obtener su ubicación (Geolocation technologies | Sigfox build, 2020).

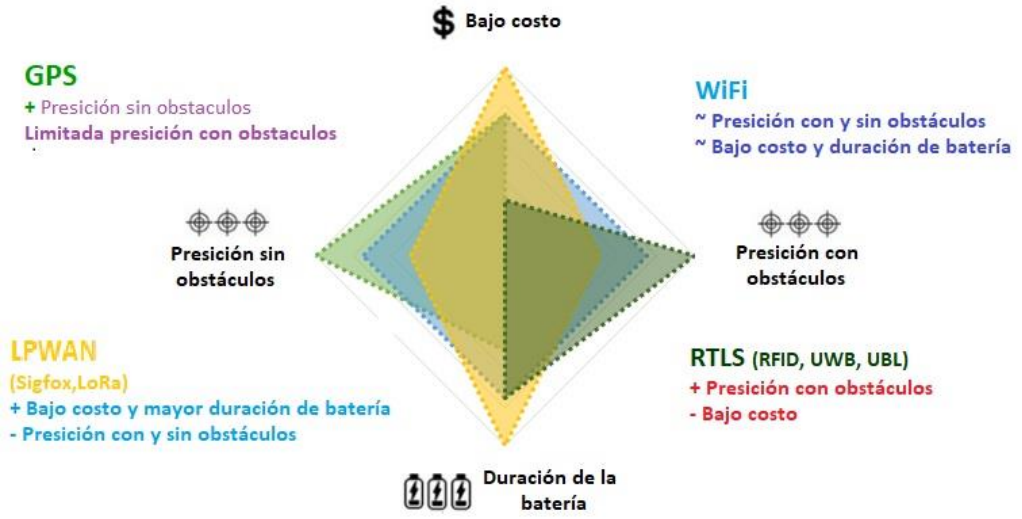
El aumento del mercado de este tipo de dispositivos para el rastreo de activos, personas, animales o cualquier tipo de objetos y de sus plataformas de administración y seguimiento para la geolocalización han permitido el desarrollo de tecnologías para este fin como por ejemplo tecnologías LPWAN.

Para soluciones IoT existen 5 principales tecnologías de uso:

1. GNSS (Sistema mundial de navegación por satélite), teniendo al GPS como pricnipal
2. WiFi
3. Por ID de celdas
4. RTLS (Sistemas de localización en tiempo real), RFID, BLE, UWB, etc.
5. Geolocalización LPWAN.

El uso de estas tecnologías depende de la aplicación que se desee dar a la misma, en la Figura 2.6 se observa un diagrama que muestra los pro y contras de cada una.

Figura 2.6 Comparativa de tecnologías de Geolocalización.



Elaborado por: Rea Alex

2.3.1 GPS.

Se trata de la tecnología de localización más popular para el rastreo de activos principalmente, siendo completamente sencillo la posición geográfica de un dispositivo al agregarle un módulo GPS y este se encuentre al aire libre (Geolocation technologies | Sigfox build, 2020).

Su funcionamiento se debe a que el sistema GPS tiene satélites ya desplegados, así este escucha a los satélites y luego envía las coordenadas a un servidor o plataforma a través de la red Sigfox, en teoría el mensaje de ubicación geográfica se almacena en 6 bytes por lo que se podría enviar 2 mensajes de posicionamiento GPS dentro de un mensaje de Sigfox (12 bytes) que una vez recibidos en el servidor ya pueden ser manejados para mostrar gráficamente en un mapa (Geolocation technologies | Sigfox build, 2020).

Ventajas:

- Precisión alta
- Utilizado en la gran mayoría de dispositivos.
- Funcionamiento global.

Desventajas:

- Pierde precisión en interiores.
- Alto consumo de batería.

2.3.2 POSICIONAMIENTO DE SIGFOX

Un dispositivo Sigfox cuenta con el indicador de la cantidad o intensidad de señal recibida (RSSI) proporcionada por los mensajes enviados por este y recibidos el backend de Sigfox. Es así como se puede obtener la posición del dispositivo en base a la red de Sigfox mediante un cálculo probabilístico de la red entregando la ubicación más probable del dispositivo que no depende del desplazamiento Doppler de la señal (Geolocation technologies | Sigfox build, 2020).

El cálculo se basa en los mensajes enviados por un dispositivo Sigfox mediante la infraestructura de la red y la ubicación de las diferentes estaciones bases que recibieron estos mensajes (Geolocation technologies | Sigfox build, 2020).

Funcionamiento

- Un dispositivo Sigfox envía un mensaje mediante la red a el servidor, en dicho mensaje se encuentra el indicador de RSSI.
- Las estaciones base desplegadas en un territorio reciben el mensaje. El cálculo de la ubicación será más preciso mientras más estaciones base reciban el mensaje
- Según los indicadores de RSSI de las estaciones base que recibieron el mensaje se puede realizar el cálculo y determinar la ubicación, se obtienen varias ubicaciones probables y se elige la de mayor probabilidad.

Ventajas:

- No necesita de otro hardware
- Reduce costos
- Manejo del uso de la batería.

Ventajas de la red Sigfox:

- Bajo consumo de energía para menor consumo de batería.
- Tecnología de largo alcance.
- Resistente a interferencias.

Limitaciones:

- El uso del indicador RSSI para el cálculo de la ubicación en función de las estaciones base no es tan preciso, el rango de precisión varia de 1 a 10 km dependiendo del número de estaciones base que recibieron mensajes del dispositivo.
- La geolocalización de Sigfox depende de la cobertura de la red en el territorio en que se encuentre el dispositivo.
- La geolocalización de Sigfox es muy útil para casos en proyectos donde la duración de la batería es clave, la precisión de la ubicación no debe ser precisa y se utilice una gran cantidad de dispositivos.

2.4 COMPONENTES ELECTRÓNICOS

2.4.1 MÓDULO SIGFOX.

En el mercado se presentan varias opciones de módulos, la mayoría de estos con poca o casi nula disponibilidad en Ecuador, considerando esta situación particular se han estudiado los siguientes módulos:

Tabla 2.3 Comparativa entre distintos módulos Sigfox (Sanchez, J., 2020).

	LoPy 4 (Pycom)	MKR 1200 (Arduino)	SiPy (Pycom)	UnaBiz's
Interfaz	UART/SPI/I2C	UART/SPI/I2C	UART/SPI/I2C	UART/SPI/I2C
Modos Uplink/Downlink	SI	SI	SI	SI
Consumo recepción	6-12 mA.	7mA.	6-12 mA.	12mA
Consumo de envío	6-12 mA.	7mA.	6-12 mA.	12mA
Salida de antena	SI	SI	SI	SI
Alimentación	3,5 - 5,5 V.	5V.	5V.	5V.
Sensibilidad	126dbm	----	126dbm	----
Precio	$35.00	$55.00	$45.00	$125.00

Tomando en cuenta el envío a Ecuador de los módulos se realizó el análisis de 2 módulos de comunicación Sigfox (LoPy 4.0 y Sipy) de la marca Pycom y 2 placas de desarrollo (Arduino MKR1200 y UnaBiz's) dentro del análisis, los 4 muestran características muy parecidas, además de trabajar en la frecuencia de funcionamiento de la red en el país.

En lo referente a costos se observa valores más altos en las placas de desarrollo sobre todo en la UnaBiz's, descartando esta por su alto costo, además por disponibilidad inmediata se descarta el Arduino MKR1200 ya que no se ha encontrado en stock, quedando con opciones principales los módulos de la marca Pycom que presentan prácticamente características idénticas con la diferencia de que el LoPy 4.0 presenta menor costo.

Debido a las características de consumo, alimentación y de precio se ha decidido como solución de módulo de comunicación el Pycom LopY 4.0.

Figura 2.7 Módulo Sigfox Pycom LoPy 4.0 ("PyCom LoPy4 ESP32 - LoRa module, WiFi, Bluetooth", 2021).

2.4.2 MÓDULO GPS NEO6MV2.

Módulo basado en el IC NEO6MV2 con capacidad para decodificar las señales de los satélites GPS (Global Position System). (U-blox, 2011)

Características:

- Se alimenta a 5V. Este módulo se conectará a través de UART con un puerto serie del microcontrolador.
- Tiene 4 pines, dos de ellos para la alimentación, Vcc y GND, otros dos para la comunicación serie, Rx y Tx.
- Este módulo tiene un tamaño de 25 x 35 mm y 25x 25 mm la antena.

Figura 2.8 Módulo GPS NEO6MV2 (Pic, 2021).

2.4.3 BATERÍA.

La batería que se va a usar en el dispositivo es una LIPO que proporciona un voltaje de 3.7V y con una corriente de descarga de 1000mAh.

El módulo Sigfox LoPy4 en funcionamiento tanto el chip ESP32 y la transmisión Sigfox tiene un consumo promedio de 120mAh. (Pycom, 2020)

El módulo GPS NEO-6M según su Datasheet indica un consumo de hasta 67mA en modo búsqueda y 11mA de consumo de tracking, sumados estos dos valores se tendría en total un consumo de 78mA por parte de modulo GPS. (U-blox, 2011)

Sumando el consumo del módulo Sigfox y el módulo GPS en modo activo tendríamos un consumo de 198mAh aproximadamente.

Para obtener el tiempo de autonomía de la batería se utiliza la siguiente formula (Ryan, 2018):

$$Tiempo\ de\ Autonomía\ = \frac{Carga\ de\ la\ bateria}{Consumo\ del\ dispositivo} \quad (1)$$

La capacidad de la batería se lo obtiene a partir de sus datos impresos en su envoltura para la batería seleccionada para este proyecto la corriente es 1000mAh para calcular su carga en Coulombs “C” se dispone la utilización de la siguiente formula donde “I” es la Corriente y “t” es el tiempo en segundos (Ryan, 2018):

$$C = I * t \quad (2)$$

Aplicando la Ecuación 2 se obtiene la capacidad de la batería propuesta para el dispositivo.

$$C = 1\ A * 3600\ s$$

$$C = 3600\ Coulombs$$

El consumo del dispositivo se obtiene de la suma del consumo del módulo Sigfox y el módulo GPS dando de 198 mA lo mismo que decir 0.198 Amperios, que nos servirá para el cálculo de la cantidad de carga (coulombs) necesarios para 1 hora (3600 segundos) (Ryan, 2018).

$$Carga\ por\ hora = \frac{Consumo\ M.Sigfox + Consumo\ M.\ GPS}{\#\ de\ horas}\ .3600\ s \quad (3)$$

Aplicando la Ecuación 3 se obtiene la cantidad de carga por hora necesaria para el consumo del dispositivo:

$$Carga\ por\ hora = \frac{0.198\ A}{h} * 3600\ s$$

$$Carga\ por\ hora = 712.8\ Coulombs/hora$$

Por lo que la autonomía del dispositivo se calcula con la Ecuación 4, en donde C es la capacidad de la batería y la carga por hora se obtuvo de la Ecuación 3 (Ryan, 2018).

$$tiempo\ autonomía\ = \frac{C}{Carga\ por\ hora} \quad (4)$$

$$tiempo\ autonomía = \frac{3600\ C}{712.8\ C/h} = 5.05\ horas$$

El cálculo del tiempo de autonomía se lo ha realizado considerando que el dispositivo estaría enviando y recibiendo datos todo el tiempo.

2.5 SOFTWARE.

2.5.1 ANDROID OS

Android es un sistema operativo diseñado para dispositivos inteligentes tales como teléfonos o tablets con pantalla táctil pero hoy en día es posible encontrar en dispositivos como relojes, televisores e incluso en automóviles de última generación. Desarrollado por Google y basado en el Kernel de Linux. (Aveda R., 2020)

Está caracterizado por tener un núcleo de Linux dentro de sus componentes principales lo cual proporciona seguridad, gestión de memoria, gestión de procesos entre otros beneficios de Linux. El entorno de ejecución de aplicaciones llamado Runtime de Android. Bibliotecas de C y C++. Aplicaciones que son todos los programas escritos en lenguaje de programación Java (Aveda R., 2020).

2.5.2 ANDROID STUDIO

Android Studio es el IDE o interfaz de desarrollo dedicado y diseñado para el desarrollo de aplicaciones de Android. Se encuentra basado en IntelliJ IDEA siendo el más potente editor de códigos para este tipo de desarrolladores, puede ofrecer funciones tales como un emulador rápido, entorno único para el desarrollo de diferente tipo de dispositivos Android, Integración con GitHub y plantillas de código para facilitar a compilar funciones de apps comunes, etc. ("Android developers", 2020).

El programa se encuentra desarrollado para que el usuario pueda realizar varios tipos de acciones a la vez, relacionadas al desarrollo de aplicaciones Android convirtiéndose en una herramienta indispensable para desarrolladores. Entre las acciones más comunes de Android Studio se tiene; crear una aplicación, usar como emulador de Android, Analizar aplicaciones de otros desarrolladores y ejecutar aplicaciones ("Android developers", 2020).

2.5.3 FIRE BASE DE GOOGLE

Se trata del servidor backend de almacenamiento o respaldo de aplicaciones móviles creado por Google y compatible para Android, iOS y web móvil. El Firebase de Google ofrece todas las herramientas necesarias para crear una aplicación exitosa.

Firebase dentro de sus características principales ofrece informe de fallas y laboratorio de pruebas para prevenir y diagnosticar errores. Solución de base de datos en tiempo real lo que resuelve los problemas de fallas de la infraestructura de backend, autenticación que permite a los usuarios acceder con facilidad y notificaciones como mensajes en la nube (Perez M., 2016).

CAPITULO 3

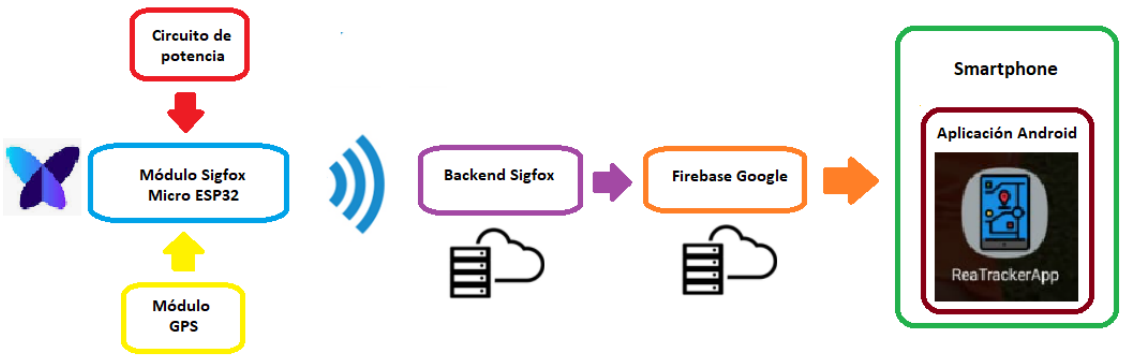
DESARROLLO E IMPLEMENTACIÓN.

En este capítulo se detalla el desarrollo del dispositivo como el aplicativo que conforman el proyecto. Se muestra el desarrollo para implementar una red Sigfox para la transmisión de datos, la interconexión de los elementos electrónicos para conformar el dispositivo y el desarrollo de una aplicación Android para el monitoreo del dispositivo.

3.1 PROPUESTA DE DISEÑO

Se llevará a cabo el desarrollo e implementación de un dispositivo con la suficiente autonomía de batería para el monitoreo de ubicación de un niño mientras sale de casa a desarrollar sus diarias (escuela, cursos, clubs, etc.). El mismo que a la vez permitirá ser cargado cada vez que sea necesario. El microcontrolador integrado en el módulo Sigfox será el encargado de recibir los datos proporcionados por un módulo GPS que luego pasarán a ser transmitidos mediante la tecnología de Sigfox al servidor de Sigfox y enseguida a un servidor Firebase de Google que permitirá la visualización en tiempo real del dispositivo en una aplicación Android instalada en un Smartphone que tendrá la capacidad de establecer rutas y recibir notificaciones.

Figura 3.1 Diagrama de bloques del sistema.



Elaborado por: Rea Alex

El bloque de microcontrolador lo conforma el procesador ESP32 el mismo que se encuentra integrado en el módulo Sigfox LoPy 4.0 que aprovechando esto reduce el tamaño del dispositivo en comparación con tener que colocar un chip externo, mediante el diseño de una PCB (Printed Circuit Board) se implementará el circuito de control de la

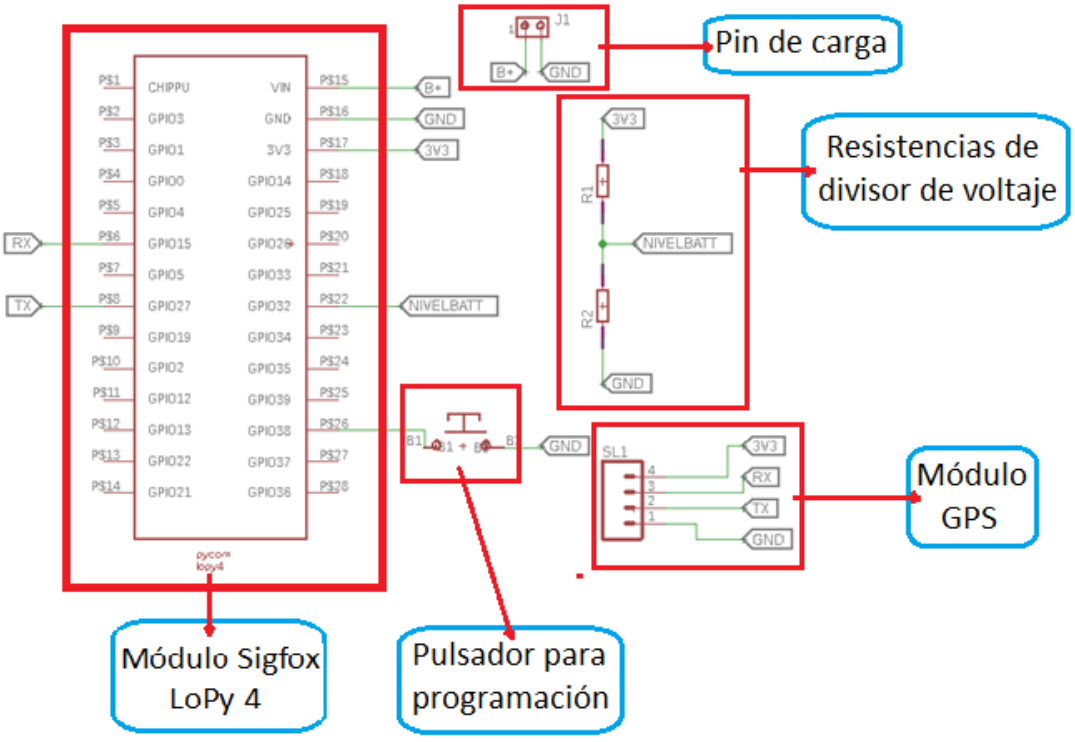
batería de 3.7 Voltios junto a un controlador para la carga de este tipo de baterías y el módulo de adquisición de datos GPS, la integración de estos dispositivos electrónicos más la batería conformarán el dispositivo.

Otra parte importante del proyecto es la aplicación Android instalada en un smartphone el cual recibirá la información de ubicación en tiempo real del dispositivo y lo mostrará en pantalla, permitirá el trazo rutas fijas y por último recibirá notificaciones.

3.2 DISEÑO DEL CIRCUITO ELECTRÓNICO EN AUTODESK EAGLE

Para el diseño del circuito se toma en cuenta que el mismo este de una forma ordenada para mejor análisis y lectura de este, se utilizó el software Autodesk Eagle para su desarrollo. La figura 3.2 muestra el diseño del circuito electrónico y la ubicación de cada uno de los elementos (resistencias, pulsador, espadines o borneras) y módulos como: módulo Sigfox Lopy 4.0, módulo GPS NEO 6M, módulo de carga de LyPo Tp4066.

Figura 3.2 Circuito Electrónico.



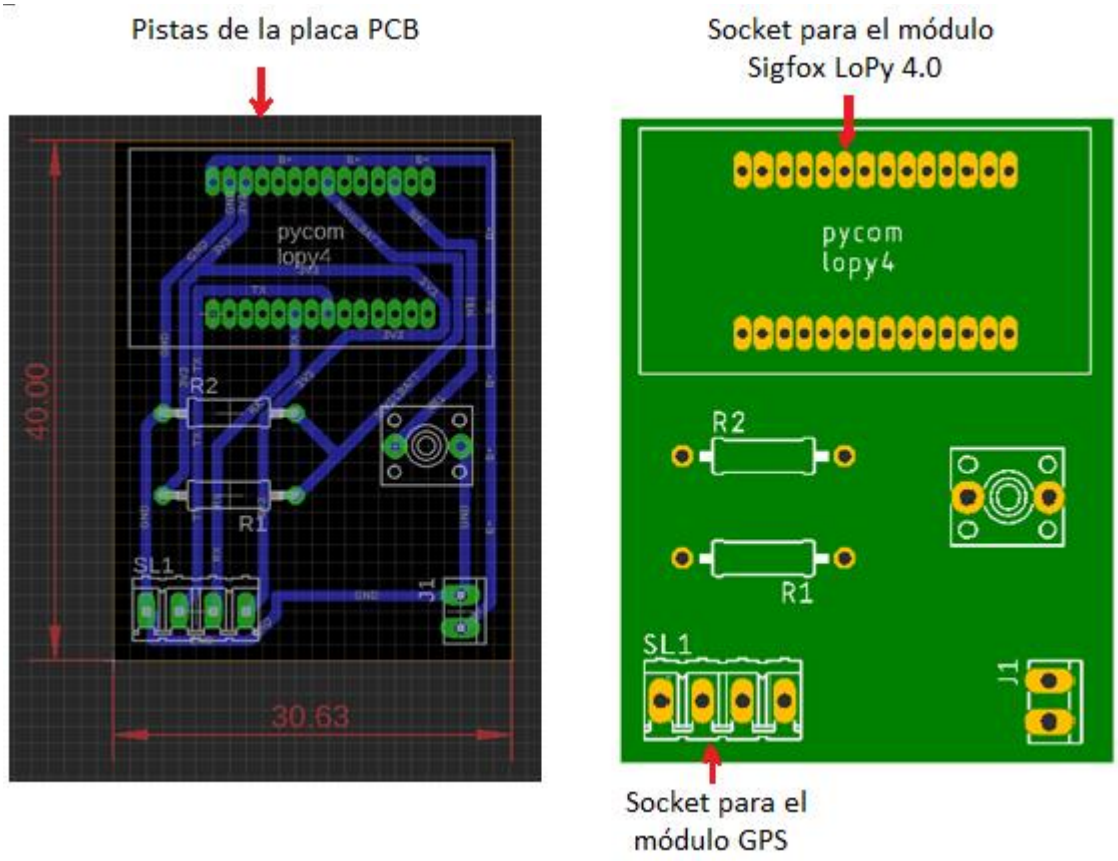
Elaborado por: Rea Alex

3.2 DISEÑO Y SIMULACIÓN DE LA PCB

Mediante el software Autodesk Eagle se realizó el diseño de la PCB, obteniendo con las dimensiones de 40x30mm. Tratando de tener un dispositivo lo más modular posible y

optimizando el espacio se colocó sockets para el módulo LoPy 4.0, módulo GPS NEO 6M, módulo de carga tp4056, además 2 conectores para la batería, un botón para la programación del módulo LoPy y 2 resistencias que conforman un divisor de voltaje encargado de enviar el nivel de la batería. Como se observa en la Figura 3.3

Figura 3.3 Diseño de PCB



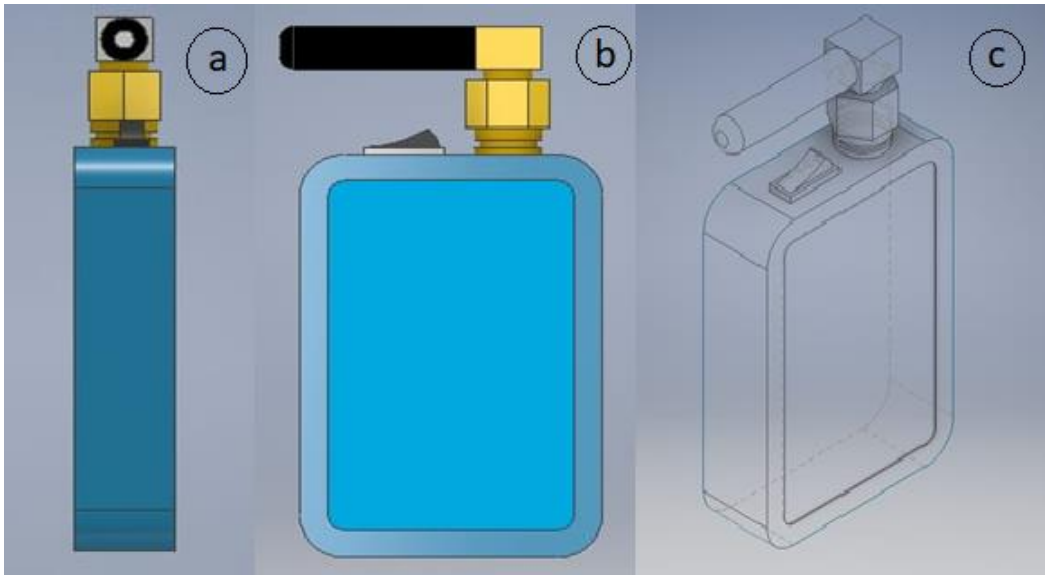
Elaborado por: Rea Alex

3.3 DISEÑO DE LA ESTRUCTURA

3.3.1 ARMAZÓN EXTERIOR DEL DISPOSITIVO.

El armazón exterior del dispositivo se realizó utilizando como herramienta el programa Autocad Inventor, en la figura 3.4 se observa el diseño de el armazón del dispositivo en vista lateral, superior y 3D donde se incorpora en el exterior una antena dipolo y el interruptor de encendido. Este diseño realizado en el software CAD.

Figura 3.4 Diseño de la carcasa del dispositivo. (a) Vista Lateral, (b) Vista superior, (c) Vista de la carcasa en 3D.

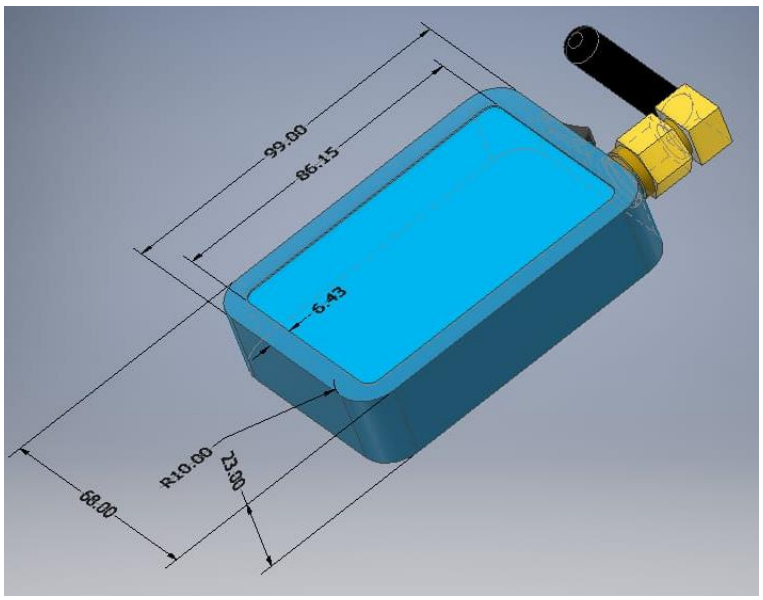


Elaborado por: Rea Alex

3.3.2 DIMENSIONES.

El armazón fue diseñado considerando la modularidad del dispositivo final y las dimensiones de cada uno de los módulos y elementos que conforman el mismo. En la figura 3.5 se observa las dimensiones finales del armazón exterior del dispositivo teniendo una estructura cubica de 99 cm de largo, 68 cm. de ancho y 23 cm. de alto.

Figura 3.5 Dimensiones finales del dispositivo.



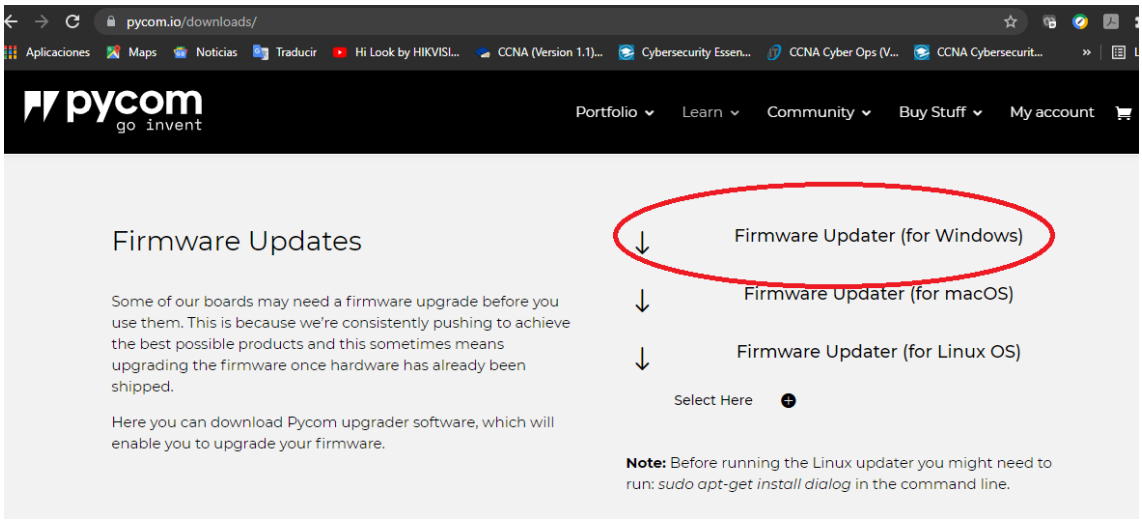
Elaborado por: Rea Alex

3.4 PROGRAMACIÓN DEL MÓDULO SIGFOX LoPy 4.0

El módulo de comunicación Sigfox Pycom LoPy 4.0 dentro de sus ventajas incluye un microcontrolador ESP32 dentro de su estructura el mismo que necesita de un IDLE compatible con micropython para su programación.

Para el proyecto se decidió utilizar e instalar el IDLE de micropython ATOM 1.51.0 para Windows por sobre otros más conocidos (Visual Studio Code) por su sencillez en la configuración y programación. Tras la instalación del IDLE se comprobó que el mismo no viene por defecto configurado para el uso de dispositivos Sigfox y se procedió a descargar el Firmware Updater para Windows disponible en la página oficial del fabricante del módulo Sigfox (Pycom).

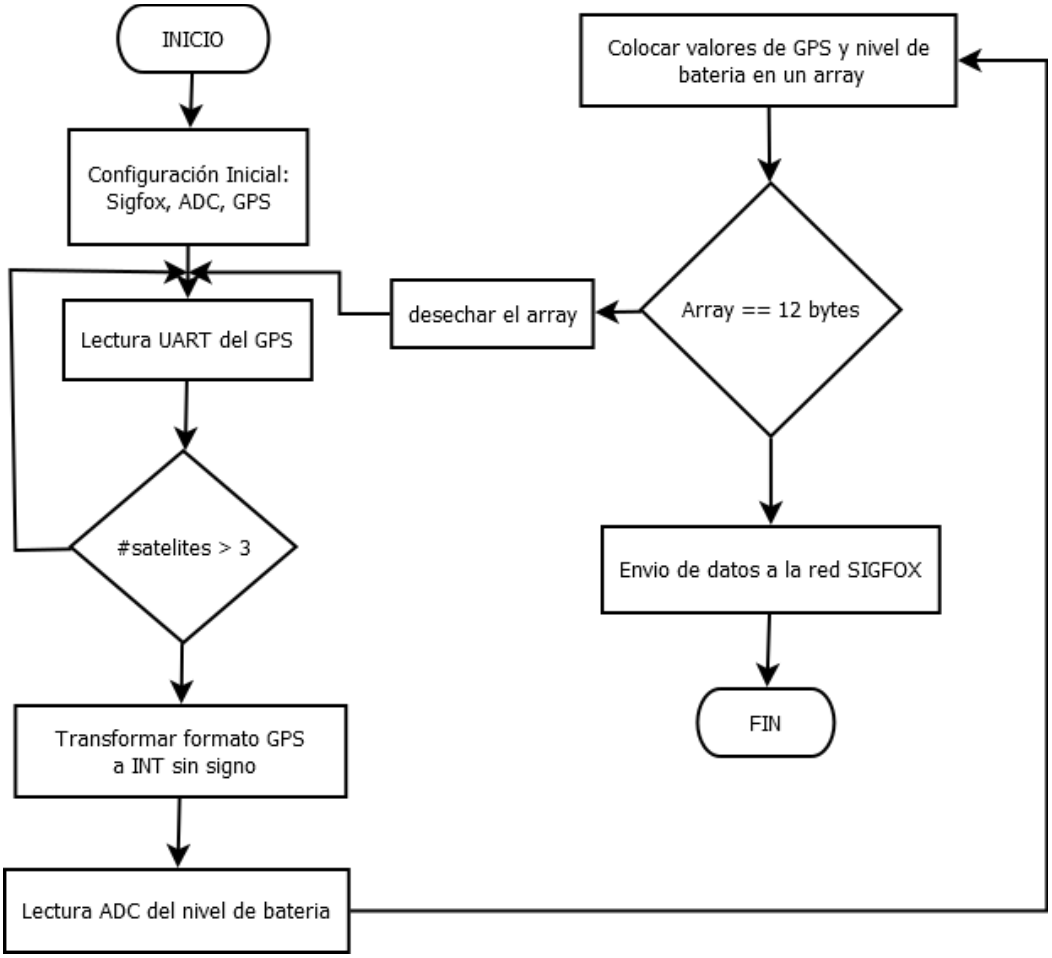
Figura 3.6 Página oficial de Pycom. Descarga de Firmware Updater (for Windows).



Tras la instalación del ATOM es necesario instalar la librería pymakr la cual permitirá comunicar el serial con Atom y escribir en la memoria flash del LoPy4, tras la instalación del firmware al final del proceso se obtendrá el Sigfox ID correspondiente al dispositivo el cual nos servirá para activar el producto en el backend de Sigfox.

En la figura 3.7 se observa el diagrama de flujo para el desarrollo de la programación del dispositivo Lopy 4.0 en el software Atom.

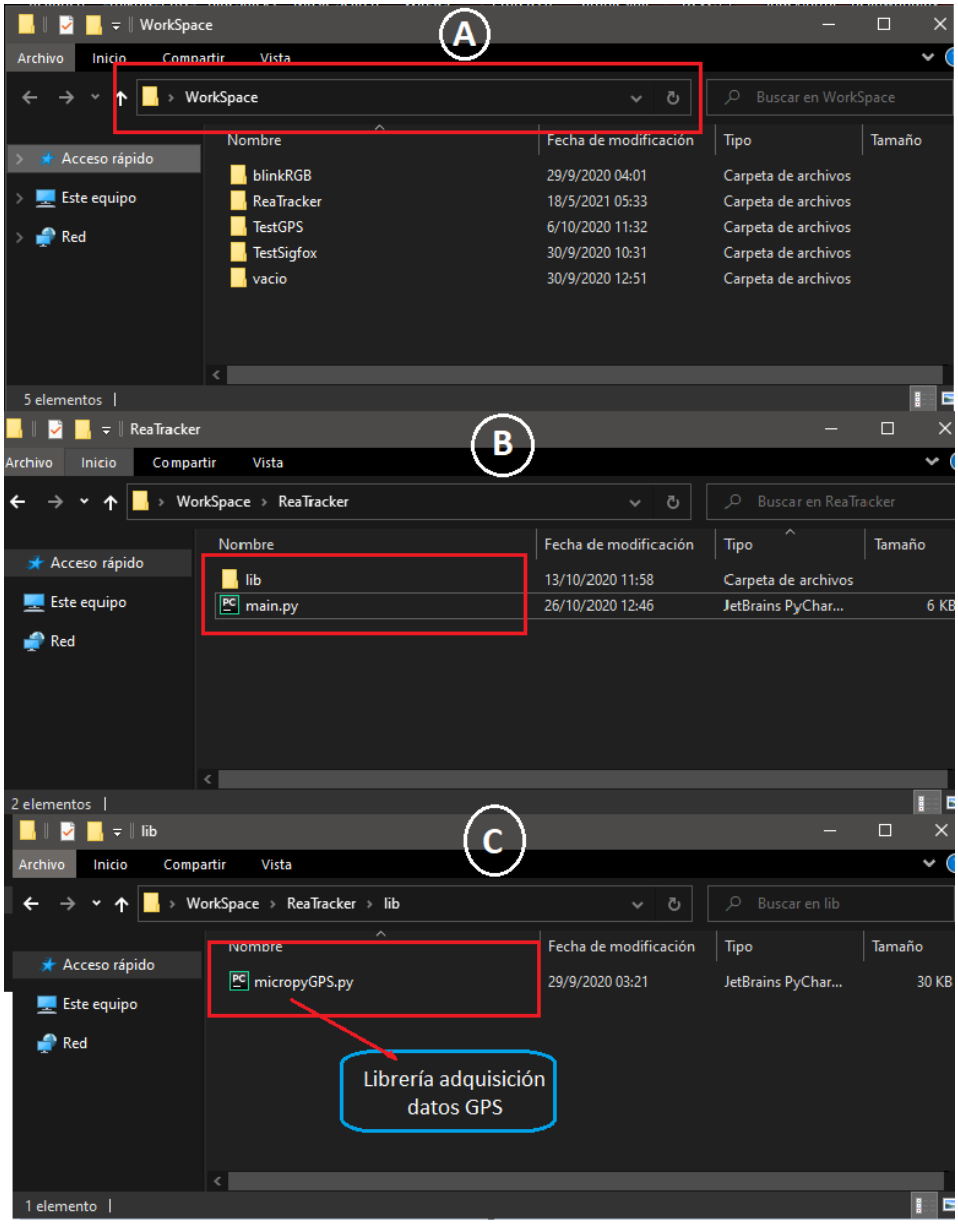
Figura 3.7 Flujograma programación LoPy 4.0



Elaborado por: Rea Alex

Para empezar la programación dentro de Atom es necesario crear una carpeta de nombre "WorkSpace" Figura 3.8 (a). En ella una carpeta de nombre "lib" y un archivo "main.py" que será la clase principal para la programación del módulo LoP7 Figura 3.8 (b). Dentro de la carpeta lib es necesario copiar la librería micropyGPS.py que permitirá la adquisición de datos del módulo GPS Figura 3.9 (c).

Figura 3.8 Carpetas necesarias para la programación en ATOM.



Para el correcto funcionamiento de la posición es necesario que el módulo GPS reciba al menos la cobertura de 4 satélites y por tal motivo se restringió al funcionamiento cuando el número mínimo de cobertura de satélites sea 4. La librería micropyGPS.py envía los datos de latitud y longitud del siguiente tipo:

Longitud: [78,32.5357, 'W']

Latitud: [0,15.4726, 'S']

Para el envío en la red Sigfox se debe tener un array de 12 bytes para lo cual se procedió a transformar estos datos mediante la función (gpsList), la cual entrega valores de este tipo:

Longitud: -0257877

Latitud: -78.542262

Estos valores aun no son manejables en un mensaje Sigfox y se necesita transformar en números enteros de 6 dígitos y sin signo mediante las Ecuaciones 5 y 6 descritas a continuación:

$$Longitud = (Valor\ + 90) * 1000 \quad (5)$$

$$Latitud = (Valor + 180) * 1000 \quad (6)$$

Al aplicar las Ecuaciones 5 y 6 se puede obtener los siguientes valores los mismos que ya pueden ser manejados para el envío por la red Sigfox.

Longitud: (-0257877 + 90) x 1000 = 101457

Latitud: (-78.542262 + 180) x 1000 = 897421

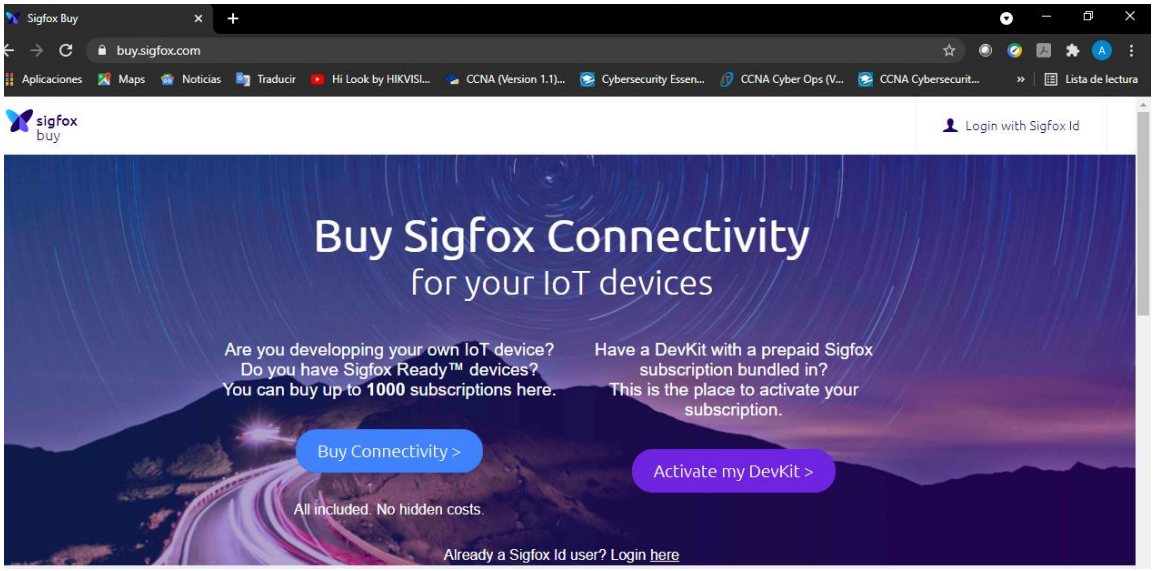
Luego de obtener estos valores los mismos son enviados en un array junto con el valor de ADC, estos datos llegan al Backend de Sigfox de forma hexadecimal.

En el ANEXO 1 se encuentra el código de mycrophyton que corresponde a la programación del dispositivo LoPy 4.0, tanto de la clase principal y de la configuración del GPS.

3.5 CONFIGURACIÓN DEL BACKEND DE SIGFOX

En la página del backend de Sigfox se realizará el registro del dispositivo y la creación de una cuenta para poder utilizar este. El registro consiste en el ingreso de información personal y lo más importante el ID del dispositivo mostrado tras la instalación del Firmware en Atom.

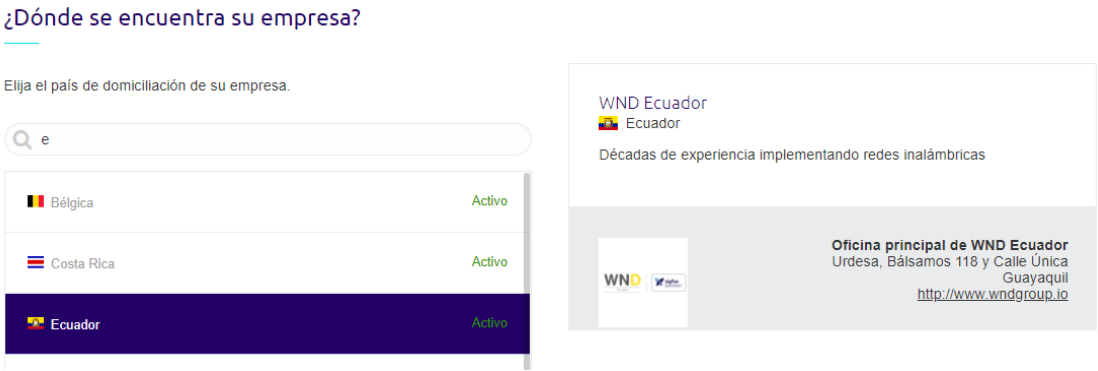
Figura 3.9 Pantalla para el registro del dispositivo Sigfox ("Sigfox Buy", 2021).



Se seleccionó la opción *Activate my DevKit* como se observa en la Figura 3.9 ya que este registro permitirá obtener un año de servicio de forma gratuita y el envío de hasta 140 mensajes por día, cada mensaje con un máximo de 12 bytes.

Se escogió el país en donde se va a realizar la comunicación Sigfox. Como muestra la Figura 3.10 Ecuador cuenta con cobertura de la red y además indica el operador que maneja la misma en el país, en este caso se trata de WND Ecuador.

Figura 3.10 Selección del país a implementar la red ("Sigfox Buy", 2021).

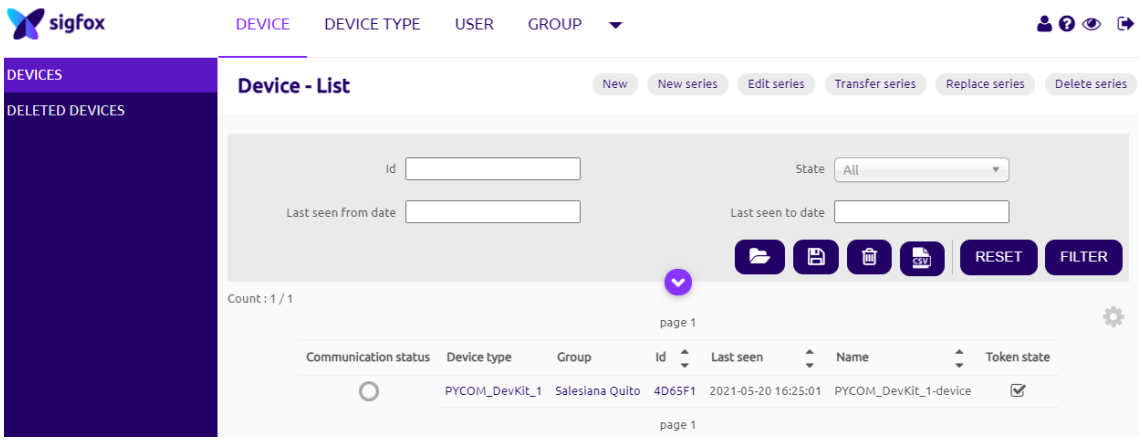


El dispositivo seleccionado para el desarrollo del proyecto, el LoPy 4.0 incluye un ID único y un PAC los mismos que deberán ser ingresados para su registro en la web oficial como se observa en la Figura 3.11, estos están disponibles en el empaque de fábrica del módulo LoPy. Además, esta ventana permitirá ingresar una descripción y el propósito del proyecto que en este caso en particular es del desarrollo de un dispositivo de rastreo.

Figura 3.11 Ingreso del ID y PAC del dispositivo Sigfox ("Sigfox Buy", 2021).

Adicional solicita el registro de los datos personales del usuario que registra el dispositivo Sigfox y un correo electrónico, una vez registrado correctamente toda la información aparece un mensaje de registro satisfactorio y la web nos dirige al Backend de Sigfox que se muestra en la Figura 3.12.

Figura 3.12 Página principal del backend de Sigfox ("Backend Sigfox", 2021).

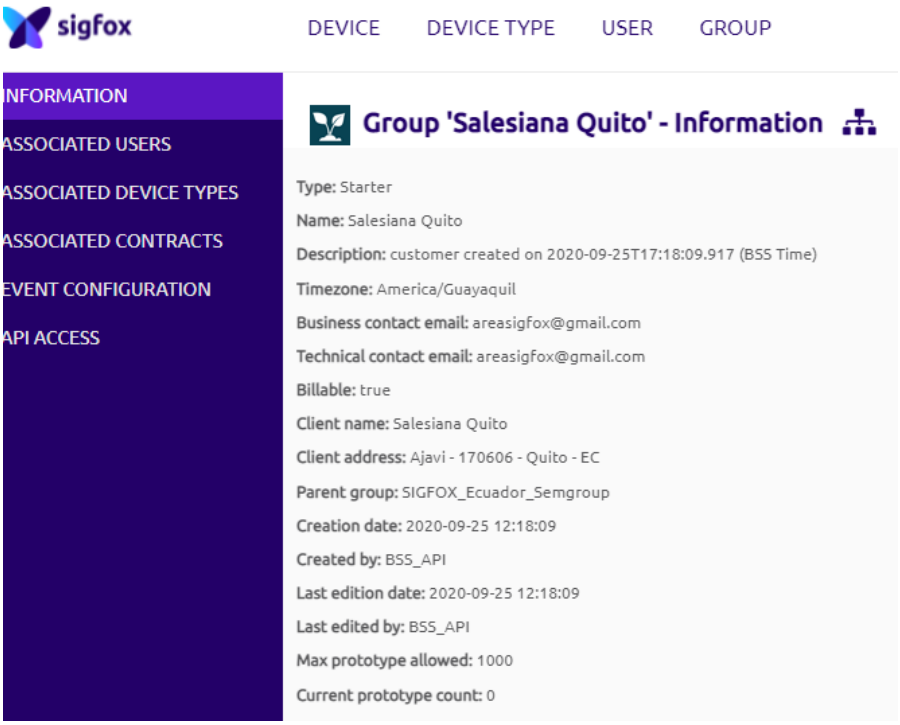


En el Backend de Sigfox en la pestaña *Device* se puede observar el listado de dispositivos registrados en este caso en particular PYCOM_DevKit_1 que corresponde al dispositivo usado en el proyecto además del grupo que fue configurado *Salesiana Quito* y el ID del mismo, tal como muestra la figura 3.12 cada uno de estos apartados además dirigen a diferentes configuraciones.

Al dar Click sobre el Grupo *Salesiana Quito* dirige hacia una nueva ventana mostrada en la Figura 3.13 la cual muestra varios parámetros de información del grupo que se los detalla a continuación:

- Información: ofrece la información de configuración del grupo, se muestran datos como el nombre, fecha de creación, zona horaria, dirección, correo asignado, entre otros.
- Usuarios Asociados: despliega la lista de usuarios asociados en el grupo en este caso en la lista se tiene un único usuario, el cual detalla el nombre, correo, fecha de creación y si es o no el administrador.
- Tipo de dispositivos asociados: despliega una lista de los dispositivos asociados con su tipo, en este caso muestra el dispositivo DevKit1 (PYCOM) que es el que se utilizará en el proyecto.
- Contratos asociados: para los dispositivos asociados al grupo muestra el tipo de suscripción contratada.
- Configuración de eventos: permite configurar eventos al momento de llegar un mensaje.
- Acceso API: permite la configuración de API's externas, compatibles con Sigfox y la versión API v2.

Figura 3.13 Ventana de Grupo ("Backend Sigfox", 2021).



Al dar click en el ID dirige a una pantalla similar a la de grupo tal como se observa en la Figura 3.14 que presenta información general del dispositivo en varios parámetros:

- Información: ofrece la información general del dispositivo como el nombre, protocolo, PAC, tipo de dispositivo, estado, indicador de calidad de señal, estado de comunicación, fecha de creación, entre otros.
- Localización: indica en un mapa la ubicación del registro del dispositivo, esto se observa en la Figura 3.16.
- Mensajes: despliega el listado de mensajes "calbacks" enviados por el dispositivo y de cada uno parámetros como: "Time" que indica la hora y fecha que fue enviado el callback, LQI (Link Quality Indicator) es la intensidad de la señal de la red cuando se envió el callback, "SEQ NUMBER" este es un número que se va aumentando desde el dispositivo cada vez que se envía un dato a la red SIGFOX este conteo de datos es independiente de la intensidad de la señal de la red, es decir cuenta los intentos para enviar de forma exitosa un callback. Todos estos parámetros se observan claramente en la Figura 3.15, tomado de un segmento aleatorio de datos recibidos en el Backend.
- Eventos: presenta un listado de cada vez que se cometió un error al momento de enviar un mensaje, muestra cada SEQ NUMBER que no se registra en el apartado Mensajes.
- Estadísticas: muestra en forma de gráficas respecto a los mensajes los siguientes parámetros: número de callbacks, cantidad de bytes, SNR (Signal to noise radio) promedio y RSSI (Received Signal Strength Indicator) todos estos por unidad de tiempo.
- Configuración de eventos: permite configurar otras opciones como envió de un correo al recibir un determinado mensaje.

Figura 3.14 Ventana de ID del dispositivo ("Backend Sigfox", 2021).

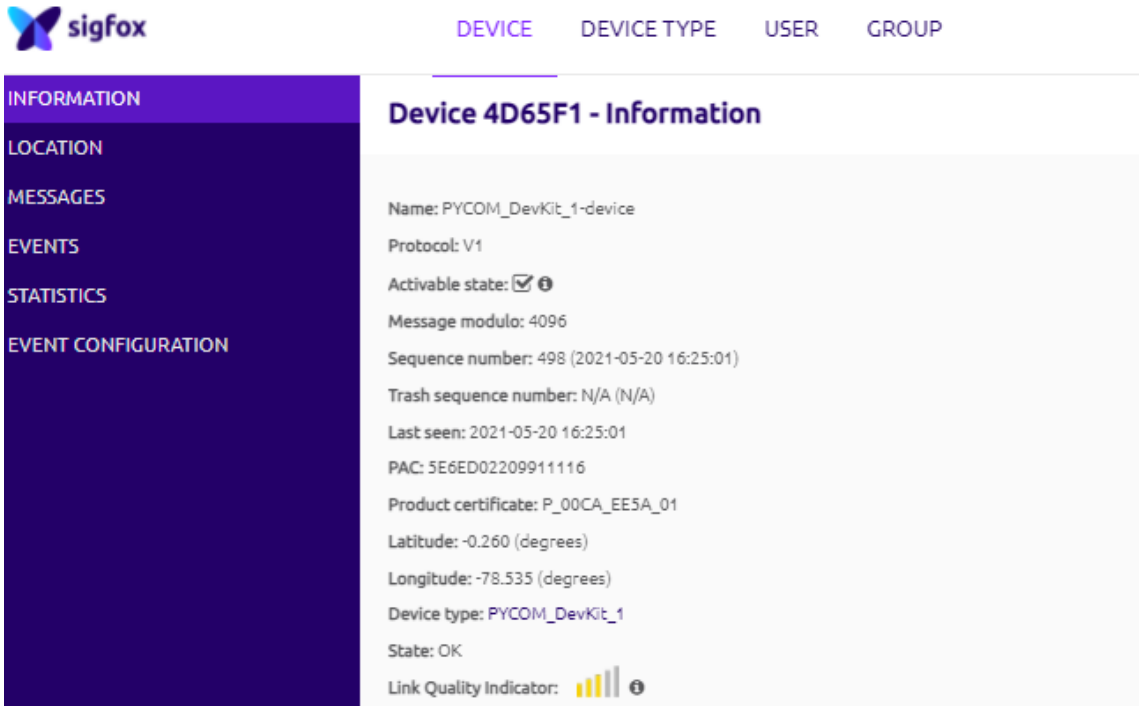


Figura 3.15 Listado de mensajes recibidos del dispositivo ("Backend Sigfox", 2021).

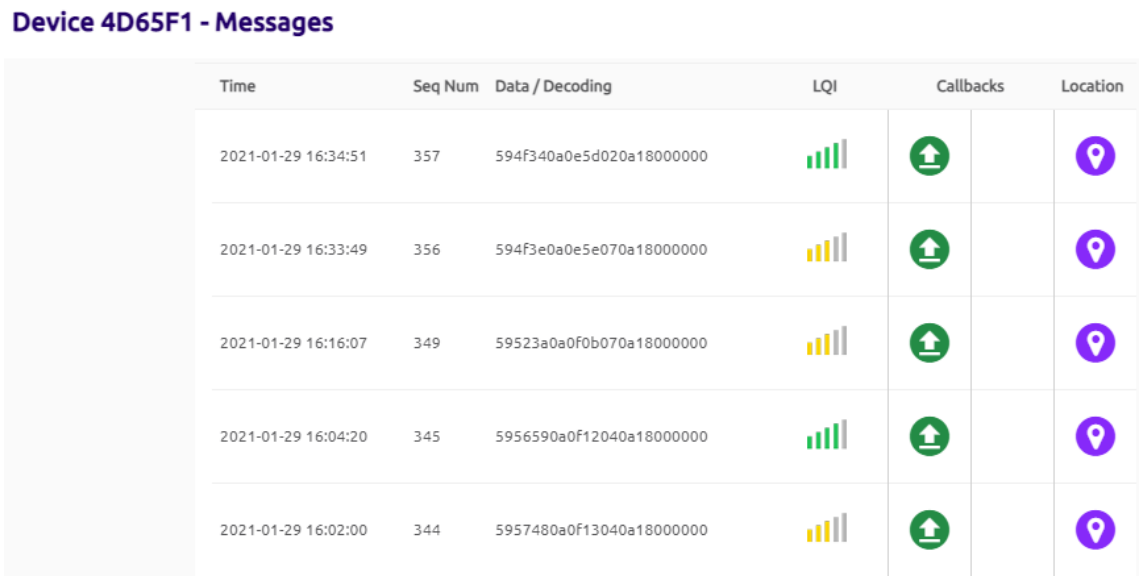
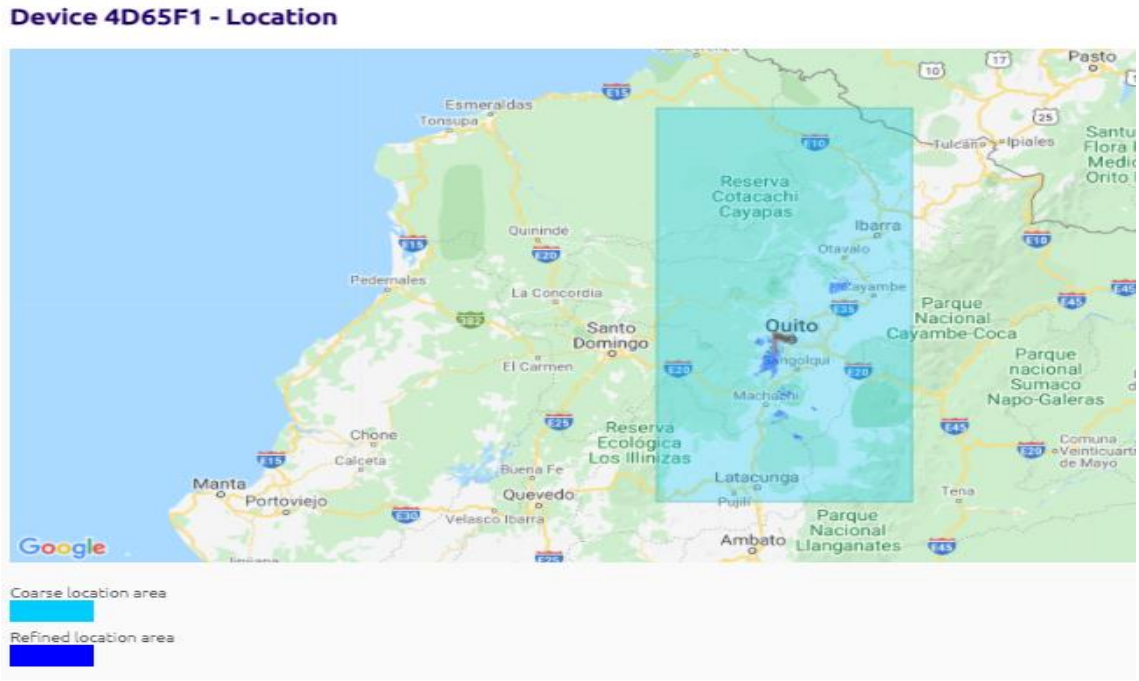


Figura 3.16 Localización del dispositivo ("Backend Sigfox", 2021).

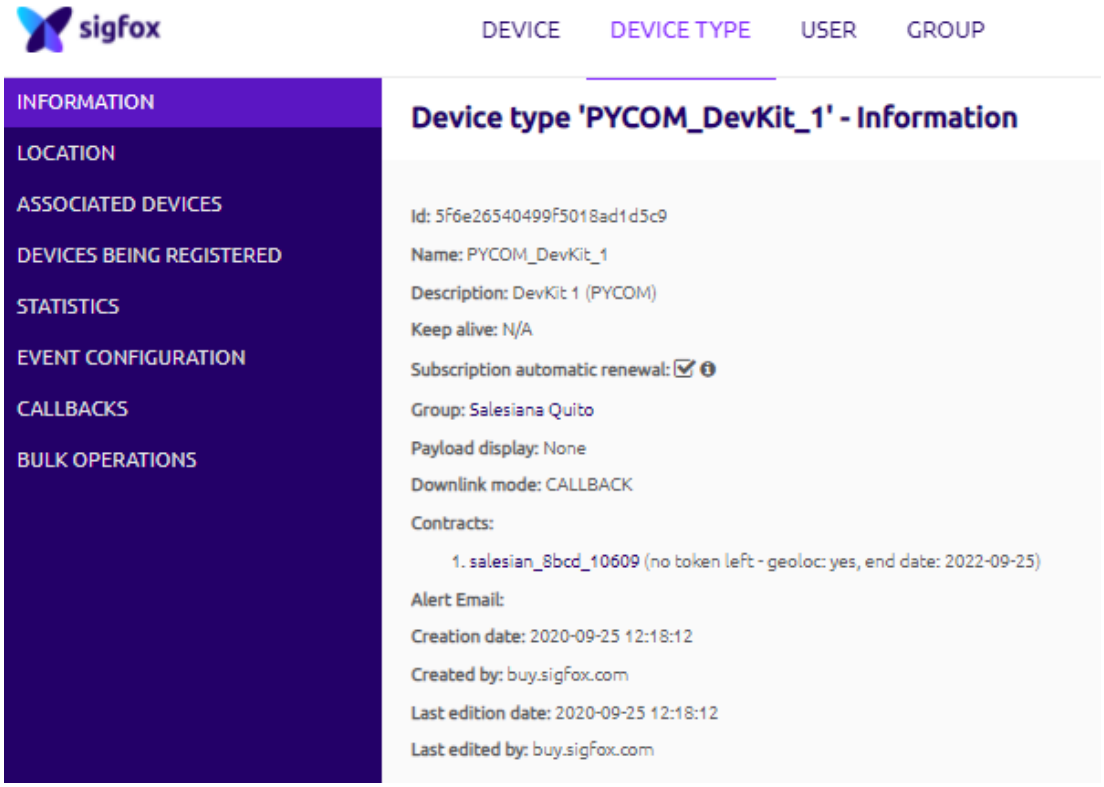


Nota: Sobre el mapa se muestra la localización de registro del dispositivo, en color celeste se muestra localización general y en azul una ubicación más precisa.

En el tipo de dispositivo al dar click sobre *PYCOM_DevKit_1* que corresponde al dispositivo del proyecto se desplegará una nueva ventana que se observa en la Figura 3.17 donde aparecerá la información y varios parámetros del dispositivo como:

- Información: se obtiene los datos generales del dispositivo como el nombre, grupo al que pertenece, fecha de registro, tipo de contrato y varios datos propios del dispositivo.
- Localización: sobre un mapa presenta la ubicación (coordenadas) del dispositivo.
- Dispositivos asociados: despliega una lista con los dispositivos Sigfox asociados a nuestro dispositivo.
- Estadísticas:muestra gráficas de parámetros como: número de callbacks, cantidad de bytes, SNR (Signal to noise radio) promedio y RSSI (Received Signal Strength Indicator) todos estos por unidad de tiempo.
- Configuración de eventos: permite configurar otras opciones como envío de un correo al recibir un determinado mensaje.
- Callbacks: se configura que hacer con los mensajes una vez que han llegado al backend de Sigfox es decir cómo manejarlos, para el proyecto se configurará el envío a un servidor.

Figura 3.17 Ventana de información del tipo de dispositivo ("Backend Sigfox", 2021).



Callbacks:

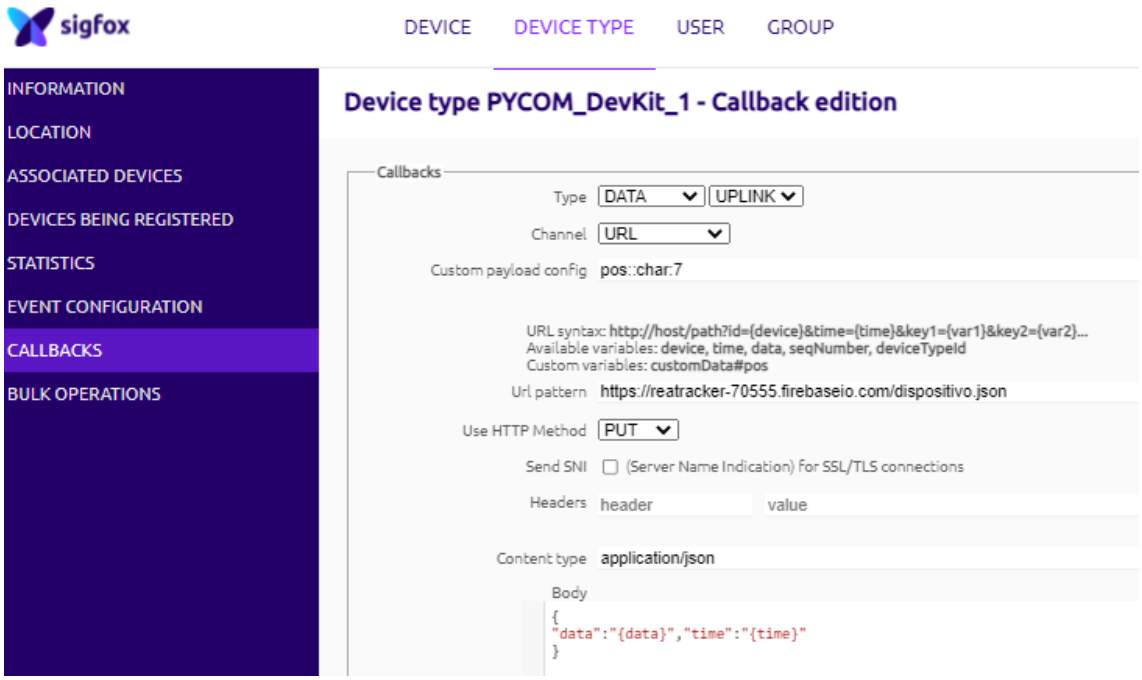
Los callbacks van a ser transportados a un servidor, para el proyecto se ha decidido utilizar Firebase de los servicios de Google. Por tal motivo es necesario realizar la configuración para la exportación de estos datos para luego ser manejados en el servidor para el desarrollo de la app Android.

Es necesario que el Callback o mensaje de la base de datos del backend de Sigfox sea enviado en un objeto de tipo JSON a la base de datos de Google ya que esta es la forma en la que Google puede leer los archivos de otras páginas. Es importante mencionar que en caso de ser necesario Sigfox si permite enviar otros tipos de formatos.

En lo que corresponde a la configuración del servidor externo es necesario en el Backend de Sigfox en la sección de Callbacks se configurare e ingrese el URL que corresponde a la base de datos de Google. En la Figura 3.18 se observa la configuración que además del ingreso de la URL es importante la configuración del tipo de petición HTTP elegida de tipo PUT que permite que el dato sea ubicado en la dirección especificada en la URL,

además que dentro del objeto JSON únicamente se va a enviar los datos correspondientes a “data” que es la ubicación obtenida por el GPS y “time” que es la hora y fecha.

Figura 3.18 Configuración de Callbacks para el envío al servidor Firebase de Google ("Backend Sigfox", 2021).

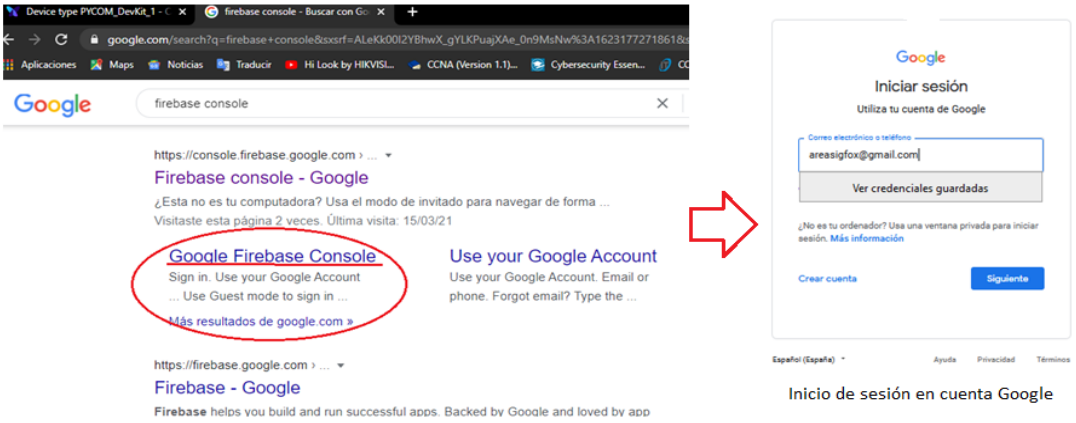


3.6 FIREBASE DE GOOGLE

Para el proyecto se utiliza el servicio de *FIREBASE de GOOGLE* como servidor para recibir los datos del backend de Sigfox, se escogió ya que presenta dos ventajas importantes: es gratuito y al ser un servicio de Google es de fácil manejo para la presentación de los datos en una aplicación Android.

Es necesario tener una cuenta de correo electrónico de Gmail para el ingreso por lo que únicamente es necesario ingresar a Firebase Console y luego a Firebase Google.

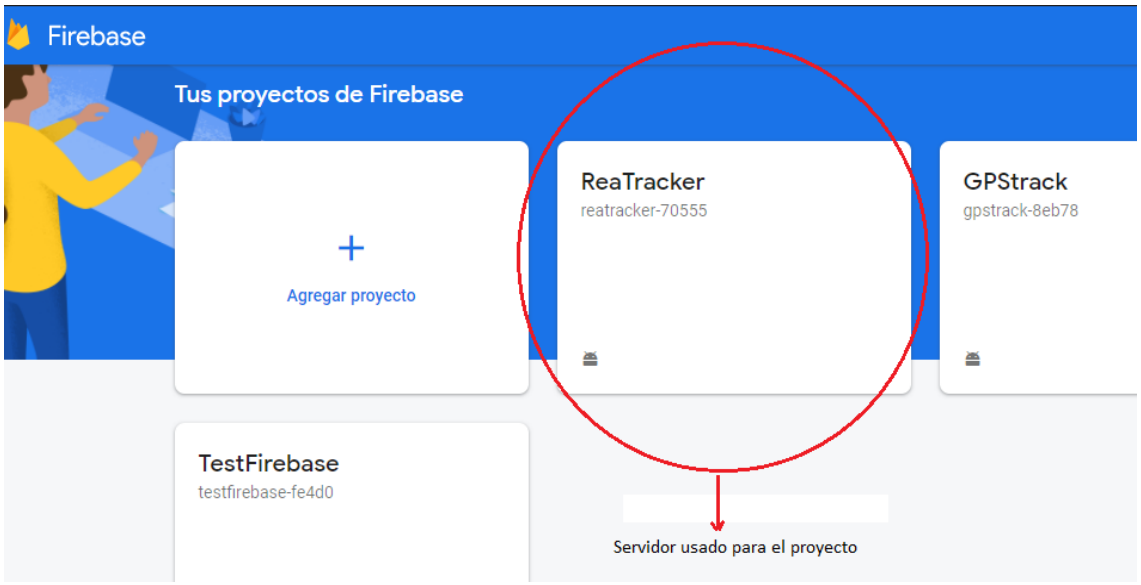
Figura 3.19 Ingreso a cuenta Google y Firebase.



Inicio de sesión en cuenta Google

Una vez en Firebase se crea un nuevo proyecto, para lo cual solicita ingresar el nombre del proyecto, la ubicación y aceptar los términos y condiciones del servicio, es permitido la creación de varios proyectos como se puede observar en la Figura 3.20.

Figura 3.20 Servidor Firebase Google ("Console Firebase", 2021).



Al ingresar al proyecto muestra una ventana nueva con varias pestañas de información, configuraciones y parámetros. En la pestaña *Realtime Database* se puede observar los mensajes que ha recibido de parte del backend de Sigfox donde anteriormente ya se configuro que los envíe a este servidor, en esta sección se observa todos los mensajes con dos parámetros que corresponden a la ubicación y el tiempo que se observan en la Figura 3.22 que luego serán interpretados gráficamente en la aplicación Android, además se tiene los usuarios y contraseñas para la autenticación en la Aplicación mostrados en la Figura 3.21.

Figura 3.21 Base de datos, registro de usuarios y contraseñas ("Console Firebase", 2021).

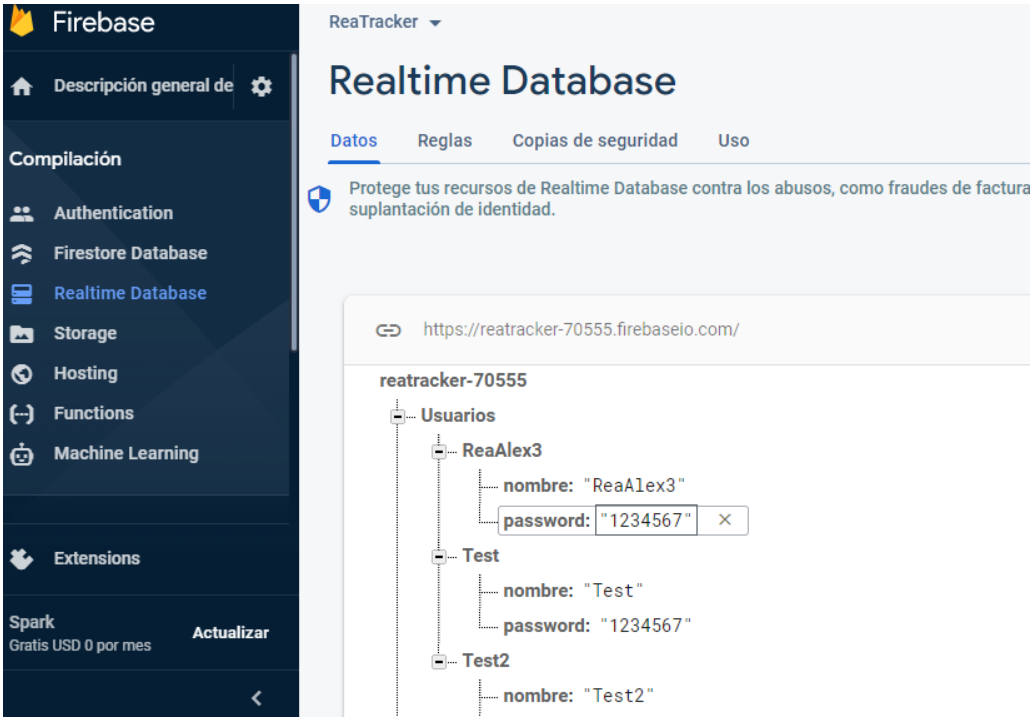
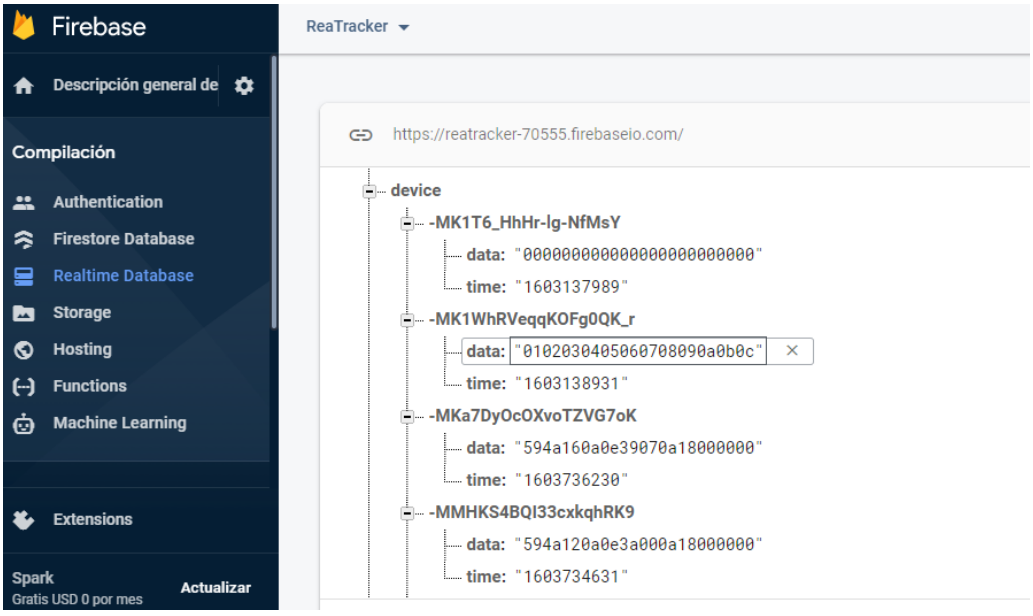


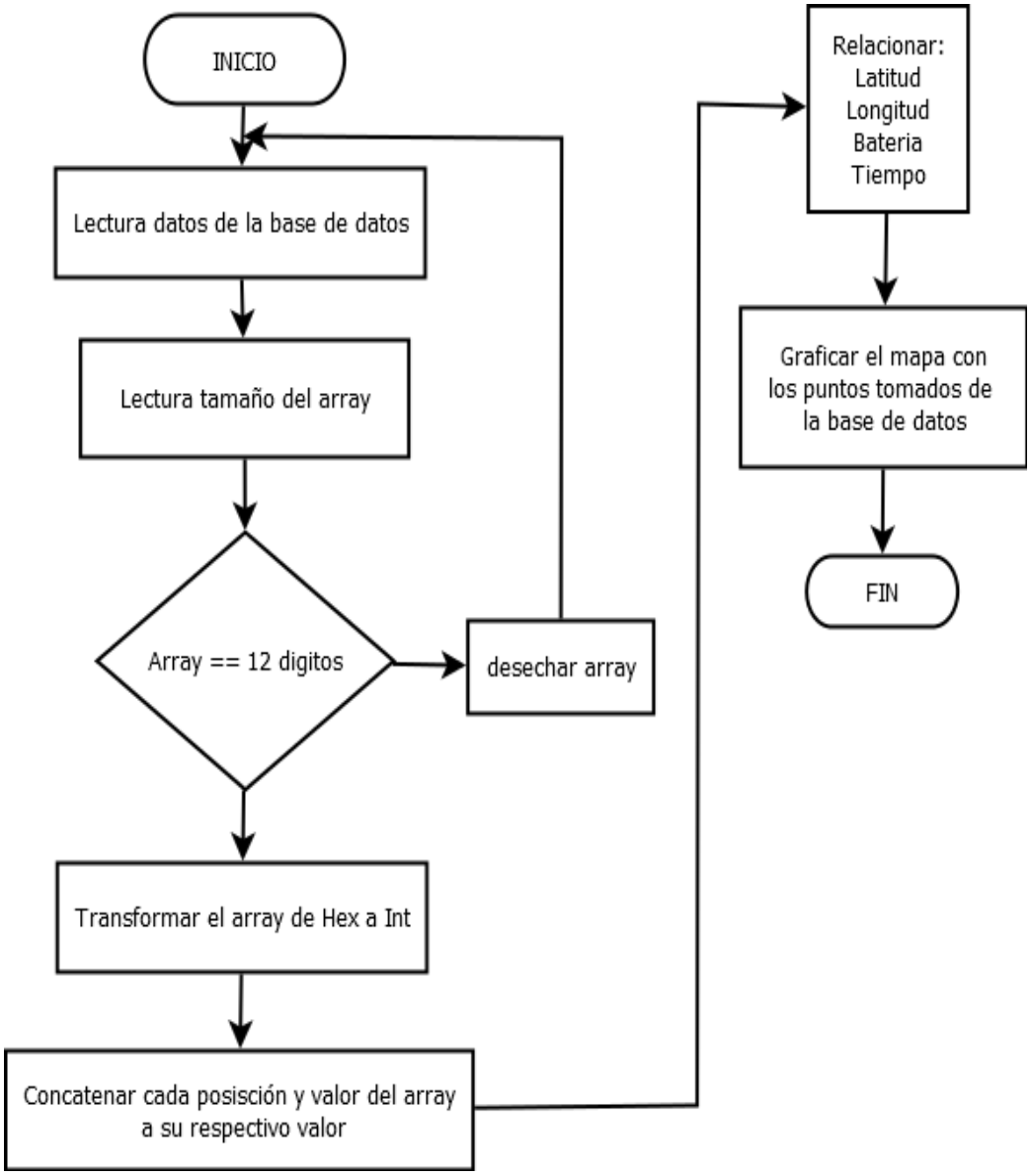
Figura 3.22 Base de datos, mensajes recibidos del Backend de Sigfox ("Console Firebase", 2021).



3.7 INTERFAZ DEL USUARIO – APLICACIÓN.

Se ha implementado una aplicación para Android con el programa Android Studio, en la Figura 3.23 se observa el diagrama de flujo propuesto para el desarrollo de la Aplicación.

Figura 3.23 Flujograma programación Android Studio.

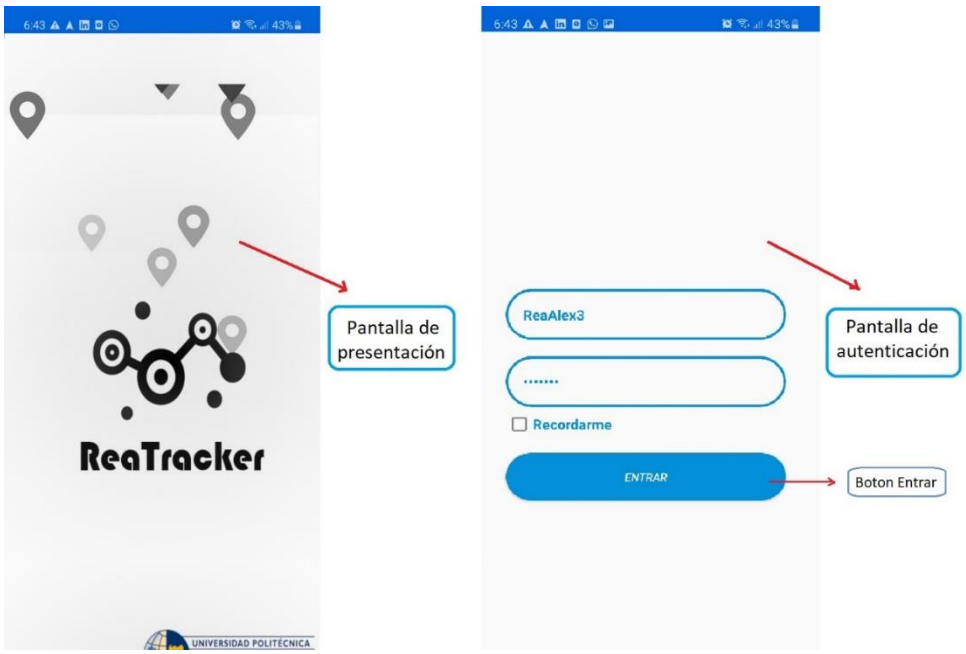


En la Figura 3.24 se observa la aplicación "Rea TrackerApp" instalada en un dispositivo Android y que se encuentra disponible el archivo .apk en: https://drive.google.com/drive/folders/1pnz7tRdnw8TP9sgLjxx3SvoODwrjfiYc?usp=sharing

Figura 3.24 Aplicación instalada en un smartphone Android.

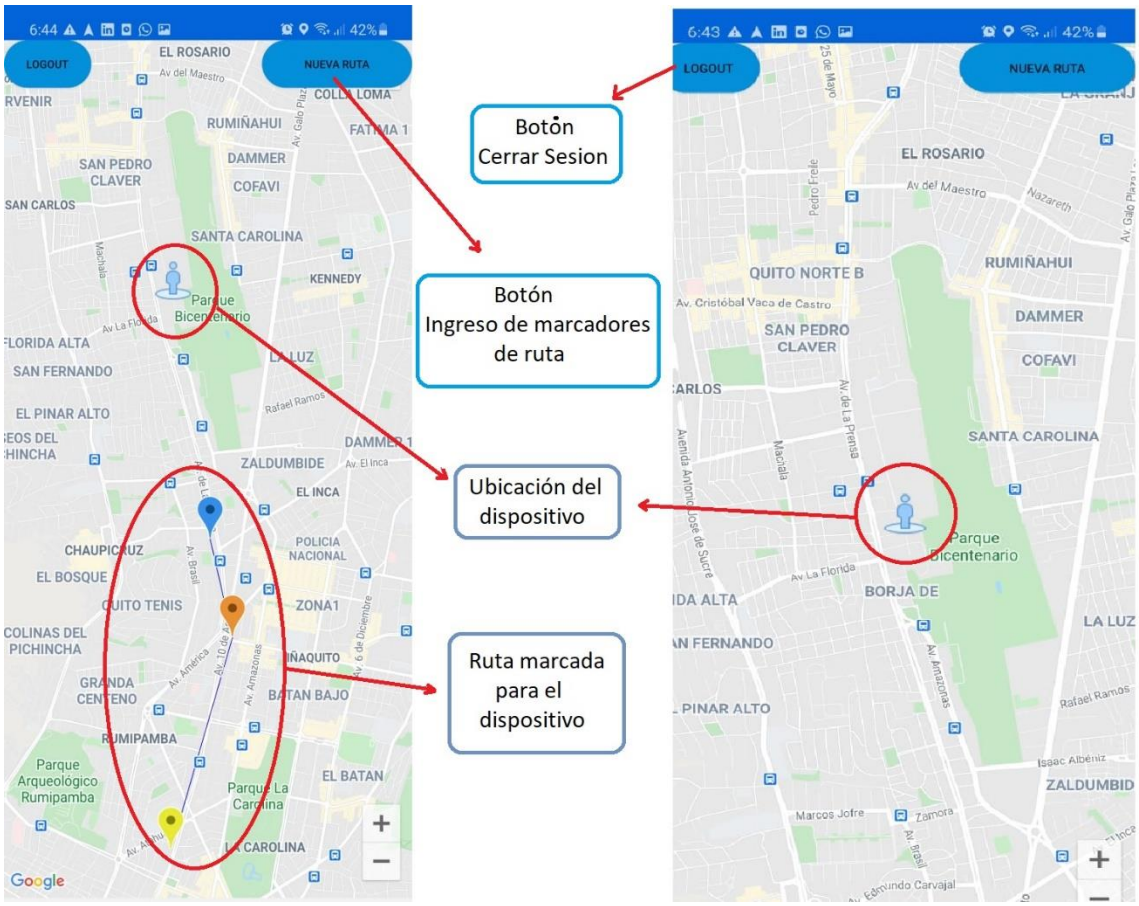
Al ingresar a la aplicación muestra una presentación animada de la misma y a continuación muestra la primera pantalla correspondiente a la autenticación de usuario que consta de 2 *TextBox* para el ingreso del usuario y contraseña y un *Botón* para el ingreso tal como se muestra en la Figura 3.25. La contraseña y usuario es la asociada al dispositivo de rastreo.

Figura 3.25 Pantalla de inicio y autenticación de usuario.



Una vez verificadas las credenciales la aplicación muestra la pantalla de rastreo en tiempo real en la cual se puede observar el servicio de Google maps sobre el cual se muestra ubicación geográfica del dispositivo representado por la imagen de una persona tal como muestra la Figura 3.26, adicional se muestran dos botones *LOGOUT* el cual permite abandonar la aplicación y el botón *NUEVA RUTA* el cual permitirá colocar marcadores en el mapa para trazar una ruta específica y en caso de abandonar la misma recibir una notificación de la aplicación en el smartphone.

Figura 3.26 Pantalla principal de rastreo y ubicación de marcadores de ruta.



Para el trazado de rutas específicas el usuario tiene la opción del botón "NUEVA RUTA" el cual permitirá colocar 3 marcadores para el ingreso de una ruta para el desplazamiento del dispositivo que en caso de desviarse de la misma la aplicación enviará una notificación al smartphone indicado el particular.

Para el cálculo del desvío de la ruta se considera que no se realiza sobre una superficie plana ya que la Tierra es esférica y ligeramente elipsoidal pero únicamente se considerará como una esfera así teniendo un error de hasta el 0.3% además que el GPS también cuenta con errores en su precisión se estimó un máximo de 200 metros de distancia entre la ruta y el dispositivo.

Para el cálculo es necesario las coordenadas geográficas tanto de los puntos de la ruta y del dispositivo en las Ecuación 1, Ecuación 2 y Ecuación 3 se tiene la fórmula de Haversine que sirve para calcular el circulo máximo entre dos puntos, y entrega la distancia entre 2 puntos sobre la Tierra en línea recta sin considerar las elevaciones (Chris Veness, 2021).

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \,.\, \cos\varphi_2 \,.\, sin^2\left(\frac{\Delta\lambda}{2}\right) \quad (7)$$

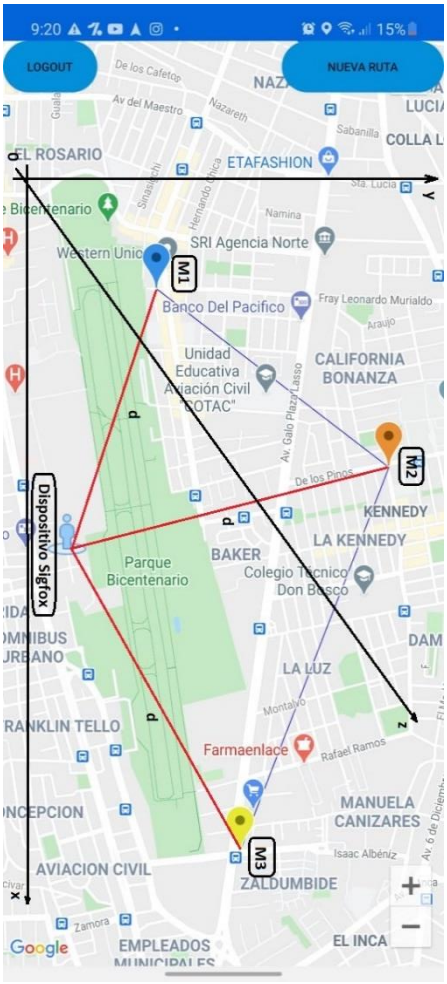
$$c = 2\,.\,tg^{-1}\,2(\sqrt{a}\,,\sqrt{1-a}) \quad (8)$$

$$d = R\,.\,c \quad (9)$$

Para la utilización de las fórmulas es necesario transformar las coordenadas en latitud y longitud a ángulos expresados en radianes donde φ representa la latitud, λ la longitud y R es el radio de la Tierra (R = 6.371 km) (Chris Veness, 2021).

En la Figura 3.27 se representa el diagrama para el cálculo de la distancia respecto a los marcadores, considerando de forma plana para este propósito y en donde: M1, M2 y M3 (Latitud y Longitud) representan los marcadores colocados para indicar la ruta del dispositivo Sigfox, las líneas en rojo representan la distancia entre el dispositivo y los marcadores, que en caso de ser superior a 220m. se envía una notificación al smartphone indicando el desvío de la ruta.

Figura 3.27 Representación de la distancia, dispositivo y ruta.



CAPITULO 4

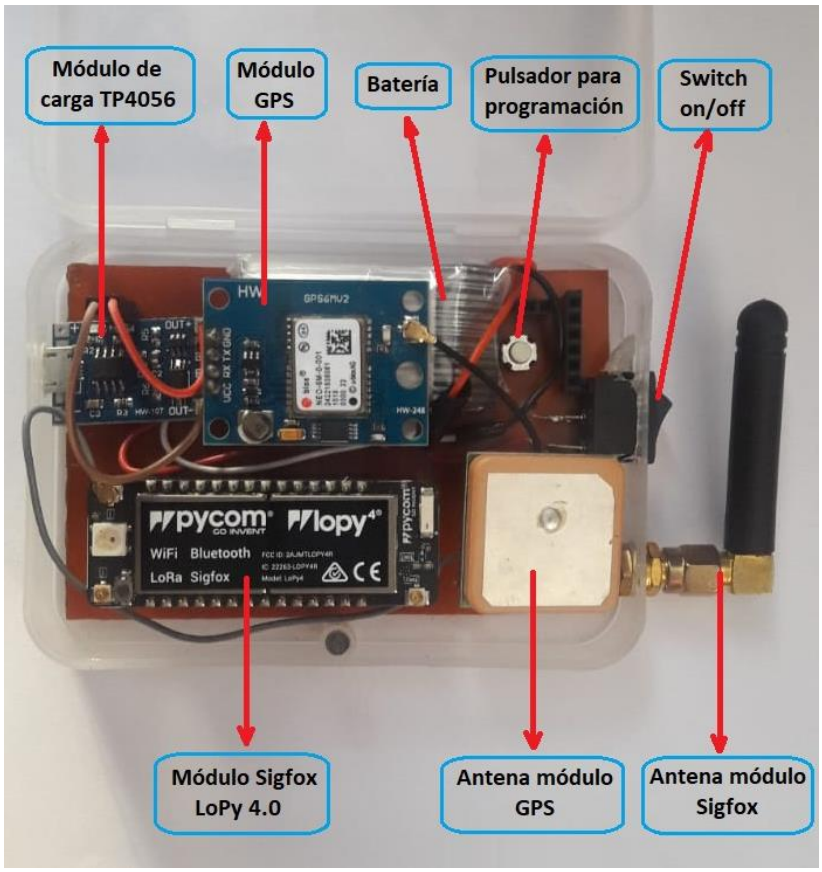
PRUEBAS Y RESULTADOS

Se construyó el prototipo propuesto con relación a los objetivos planteados para el proyecto, el mismo que consta de dos partes. El dispositivo de rastreo el cual será el encargado de enviar las ubicaciones a la aplicación móvil Android desarrollada para el monitoreo de dichas ubicaciones. Para las pruebas del dispositivo y aplicación es importante mencionar que la red se encuentra desplegada para Ecuador por un operador (WGN Group).

4.1 MODELO IMPLEMENTADO

El dispositivo se organizó de tal forma que resultó lo más modular posible, así permitiendo el poco uso de espacio en una mochila. En la figura 4.1 se observa el dispositivo con cada una de sus partes y módulos que lo conforman dentro del armazón exterior.

Figura 4.1 Modelo implementado con todas sus partes.



4.2 PRUEBA DE FUNCIONAMIENTO

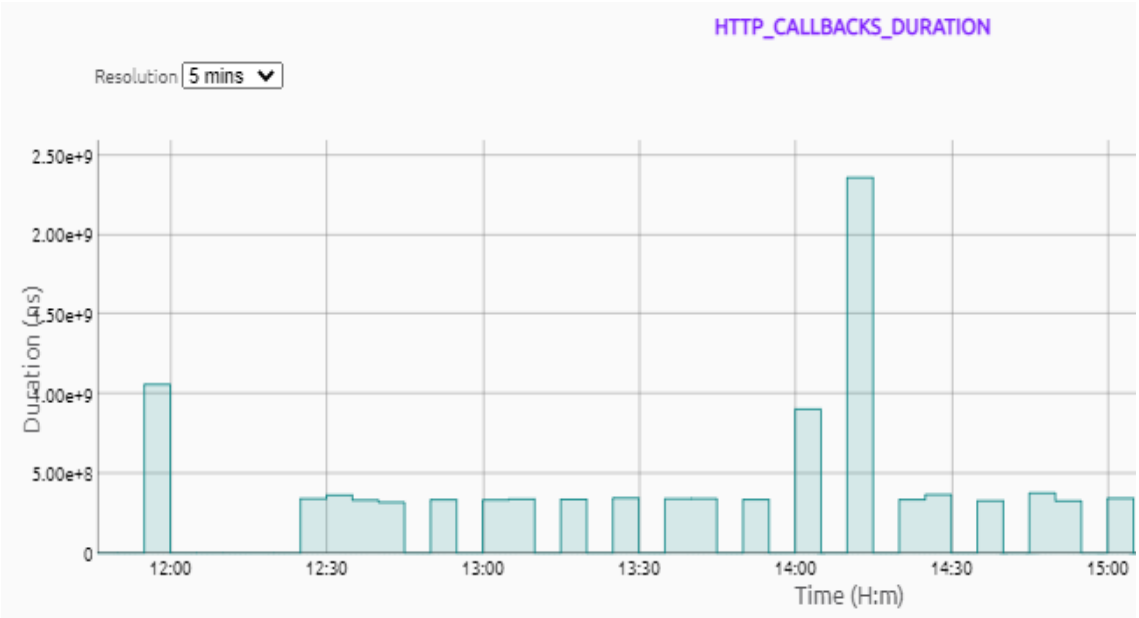
Tras desarrollar el dispositivo de rastreo y la aplicación Android se procedió a realizar pruebas de funcionamiento para validar los beneficios que brinda el mismo además de las desventajas y el análisis de la red.

En relación con la red Sigfox se puede analizar el tiempo de respuesta de esta (retardos), el dispositivo al trabajar con esta red envía los datos de forma inalámbrica desde el dispositivo (modelo implementado) hasta la Aplicación Android instalada en un smartphone. Para lograr esto el mensaje o datos deben cumplir un proceso de envío manejo y recepción por diferentes sistemas de comunicación lo cual provoca retardos, esto porque al transmitirse de forma inalámbrica se lo realiza por medio de ondas que se propagan por el aire, provocando un retardo de propagación.

Sigfox al tratarse de una red licenciada no se puede determinar individualmente los retardos de propagación, de transmisión y de procesamiento, pero se puede mencionar 3 que son parte del proceso para el desarrollo del proyecto:

- Retardo de la red Sigfox: El retardo que se da en el envío de los datos desde el dispositivo hacia el backend de Sigfox, el cálculo de este retardo se complica debido a que la infraestructura de la red ya se encuentra desplegada.
- Retardo de Callback: El retardo del envío de datos desde el backend de Sigfox hacia el servidor Firebase. Este retardo se lo puede obtener de los servicios de estadísticas del backend como se muestra en la Figura 4.2.
- Retardo Aplicación móvil: el retardo que se da entre el servidor Firebase hasta que los datos lleguen a la aplicación móvil, este dependerá de la conexión del smartphone a Internet, esto puede ser por WiFi o conexión con datos móviles de la red celular.

Figura 4.2 Duración de Callback ("Backend Sigfox", 2021).



Tras las pruebas y con la red con buena intensidad de señal, los diferentes tipos de retardos antes mencionados y sin poder calcularlos debido a la infraestructura de la red, se comprueba que son mínimos, siempre considerando sectores con la red óptima.

En lo que concierne a distancias máxima o mínima de recorrido no existe un límite, ya que el dispositivo es capaz de funcionar independientemente del recorrido siempre y cuando disponga de batería lo cual se explicará más adelante en detalle el consumo energético del dispositivo.

Para el correcto funcionamiento del dispositivo es necesario de varios parámetros, uno de ellos es la vista del módulo GPS con los satélites para la precisión de la ubicación, ya que se configuró para que envíe la información de ubicación geográfica con mínimo 4 satélites de vista ya que con menos satélites los valores que enviaba de ubicación eran muy imprecisos, para esto se procedió a probar el dispositivo en diferentes escenarios físicos que se detallan en la Tabla 4.1

Tabla 4.1 Prueba del dispositivo en distintos escenarios.

ESCENARIO	LUGAR	VALIDACIÓN
Espacios Abiertos	Parque “La Carolina”	SI
	Parque Miraflores	SI
Espacios Cerrados	Casa sector Mañosca	SI
	Túnel San Juan	NO
	Edificio Antisana Piso 6	SI
	Hospital Metropolitano	SI
	Subsuelo C.C. Quicentro Norte	NO
	Subsuelo C.C. Iñaquito	NO

Es importante mencionar que la validación en escenarios físicos se realizó en localidades ubicadas en un sector donde ya se comprobó que la red Sigfox funcionaba de forma eficiente y correcta.

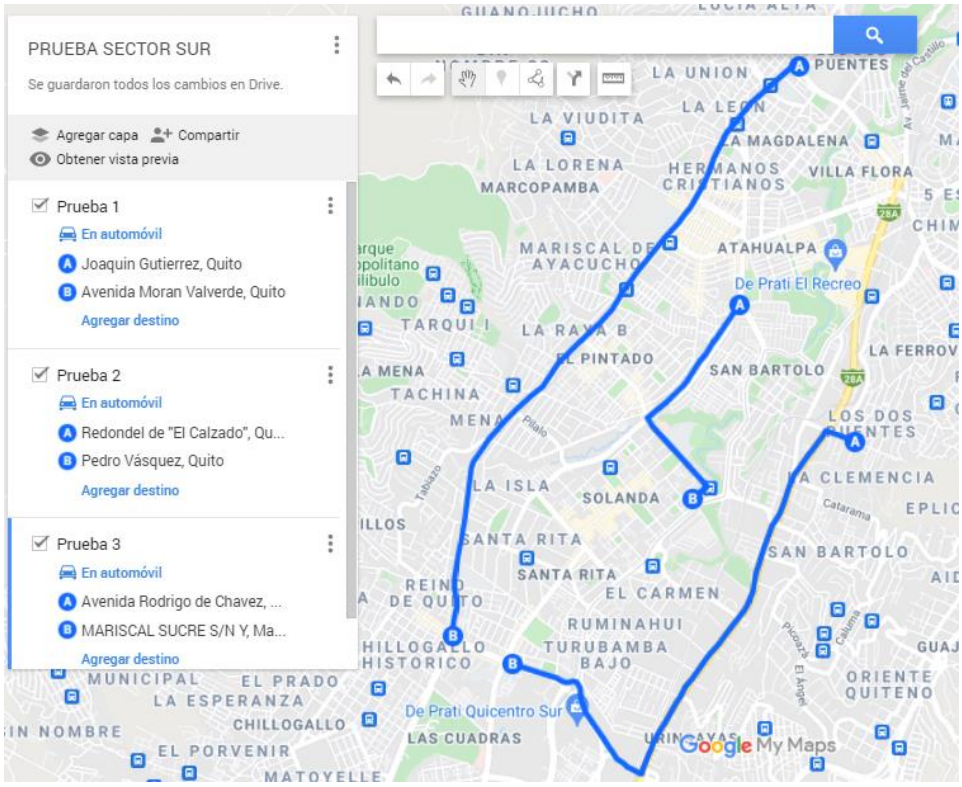
Tras un barrido en diferentes sectores de la ciudad se tomó los recorridos en donde el dispositivo al menos envió 1 mensaje sin tomar en cuenta de la intensidad de la señal recibida tal como se observa en la Tabla 4.2, para este propósito el dispositivo fue movilizado en un vehículo y con velocidad variable dependiendo del sector.

Tabla 4.2 Resumen de funcionamiento en varios recorridos del dispositivo en la ciudad de Quito.

SECTOR	RECORRIDO	# MENSAJES ENVIADOS	TIEMPO RECORRIDO	DISTANCIA (km)
SUR	Av. Maldonado Desde: Joaquín Gutiérrez Hasta: Av. Morán Valverde y Rumichaca	4	25 min.	5.1
	Cardenal de la Torre Desde: Redondel del Calzado Hasta: Av. Ajaví y Pedro Vazquez	4	14 min.	2.1
	Av. Mariscal Sucre Desde: Av. Rodrigo de Chávez Hasta: Av. Morán Valverde	5	32 min.	6
CENTRO	Av. Pichincha Desde: Playón de la Marín Hasta: Calle Yaguachi	1	18 min.	2.2
	Av. Mariscal Sucre Desde: Miraflores Hasta: San Roque	1	23 min.	1.97
	Calle Guayaquil – Av 10 de Agosto Desde: Av. 5 de Junio Hasta: AV Patria	2	21 min.	2.98
NORTE	Av. 6 de diciembre Desde: Av. El Inca Hasta: Av. Naciones Unidas	26	16 min.	2.56
	Av. Naciones Unidas Desde: Av. 6 de Diciembre Hasta: Av. América	19	13 min.	1.6
	Av. América Desde: Av. Naciones Unidas Hasta: Granda Centeno	19	6 min	0.63
	Granda Centeno Desde: Brasil Hasta: Francisco Cruz	14	4 min.	1.08

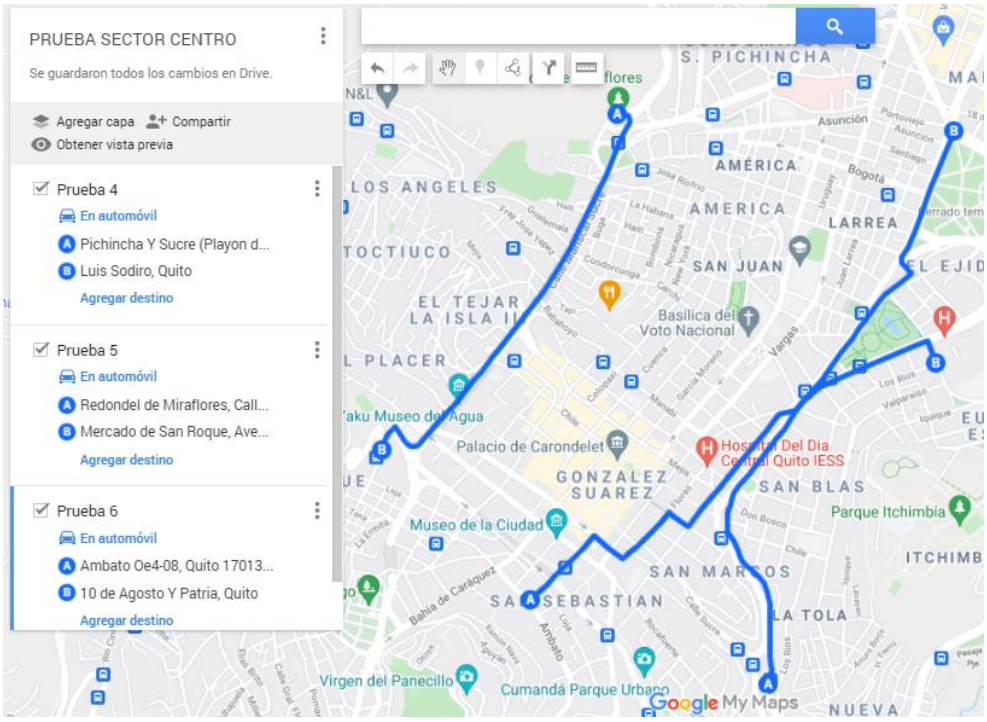
Elaborado por: Rea Alex

Figura 4.3 Pruebas de recorridos en el sector SUR de Quito.



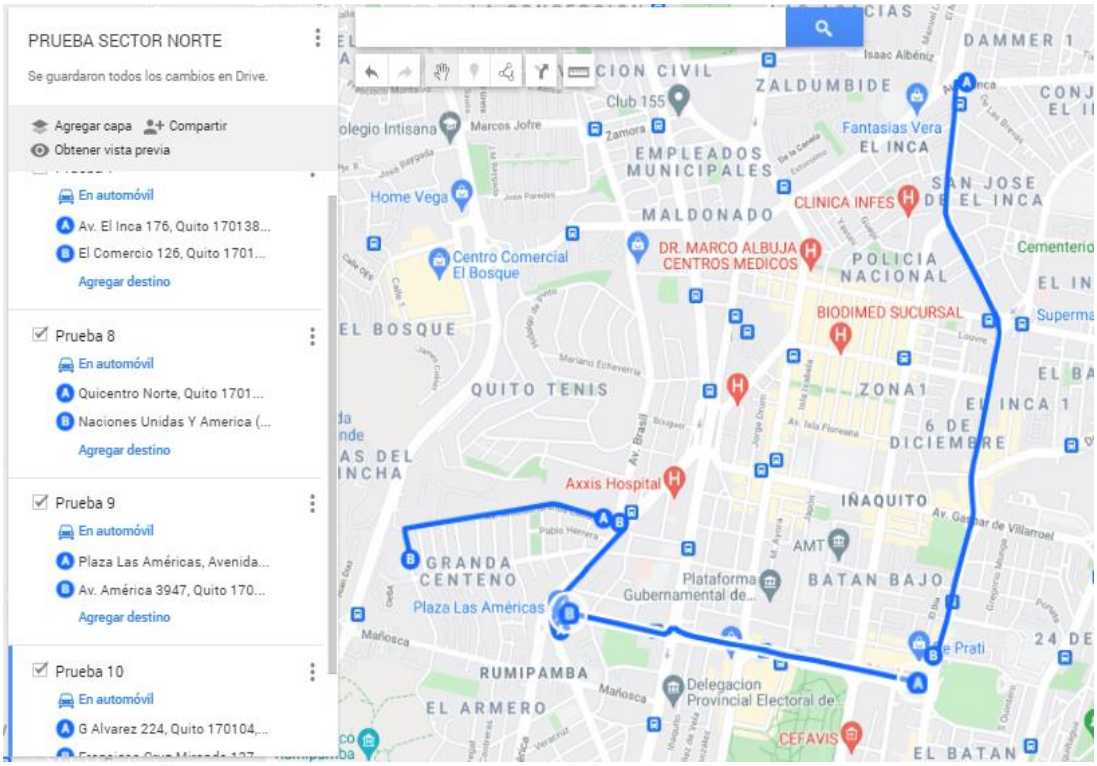
Nota: Los recorridos fueron realizados en automóvil, el dispositivo dentro de una mochila y con clima despejado.

Figura 4.4 Pruebas de recorridos en el sector CENTRO de Quito.



Nota: Los recorridos fueron realizados en automóvil, el dispositivo dentro de una mochila y con clima despejado.

Figura 4.5 Pruebas de recorridos en el sector NORTE de Quito.



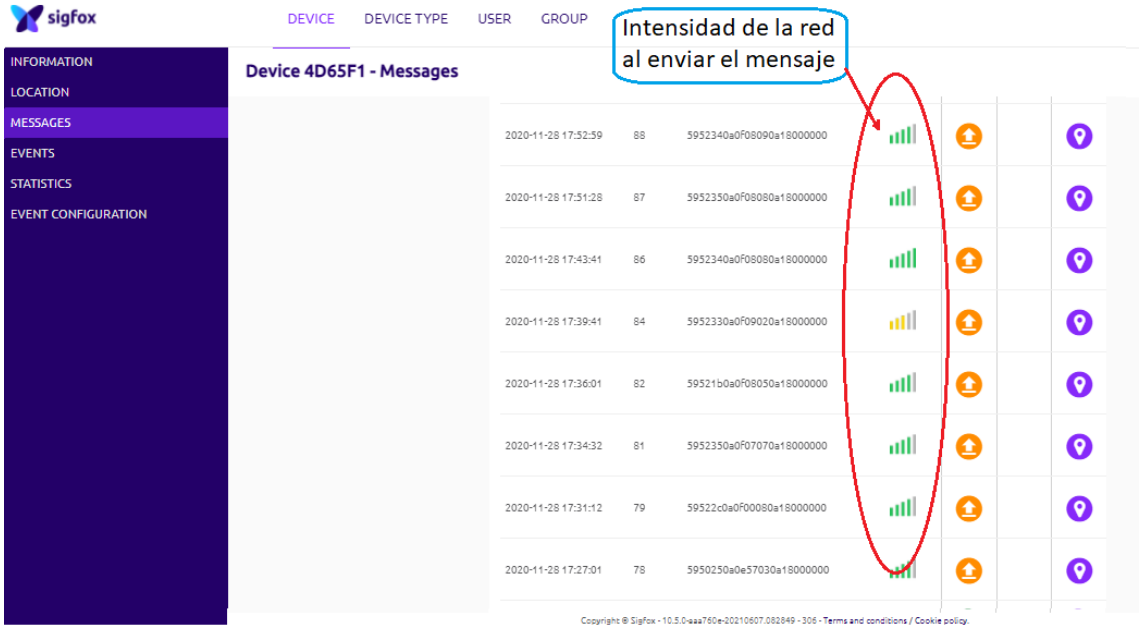
Nota: Los recorridos fueron realizados en automóvil, el dispositivo dentro de una mochila y con clima despejado.

Si bien en todos los recorridos mencionados en la Tabla 4.2 el dispositivo logró enviar mensajes, en los sectores SUR y CENTRO el envío de los mensajes a través de la red es complicado ya que las pruebas se realizaron en un vehículo en movimiento. En estas zonas la cantidad de mensajes es reducida y casi nula, pero observando algunas particularidades:

- El backend recibe mensajes dispersos y no los suficientes para una buena interpretación en tiempo real de la ubicación sobre la aplicación.
- La intensidad de la señal con la que reciben los mensajes es media o baja, es posible observar esto en el backend de forma gráfica. En color naranja señal media y en color rojo señal baja tal como muestra la Figura 4.7.
- Con la intensidad de señal media o baja el envío de mensajes se complica aún más con la velocidad a la que se traslade el dispositivo. Este en algunos lugares con esta señal logró enviar mensajes, pero sin movimiento.

En el sector NORTE y con vista directa hacia el Cerro Pichincha se tiene una mejor calidad de señal representada de color verde como se observa en la Figura 4.6 pero así mismo esta varía entre buena y regular pero con un incremento considerable de mensajes enviados desde el dispositivo al Backend de esta forma logrando un buen seguimiento del dispositivo sobre la aplicación Android, cabe mencionar que en el sector NORTE el funcionamiento no es correcto completamente ya que mientras el dispositivo más se aleje hacia el norte y occidente de la ciudad la cantidad de mensajes y la intensidad de la señal disminuye teniendo resultados similares a los del sector SUR y CENTRO y también es influyente la velocidad de desplazamiento para el correcto envío de los mensajes.

Figura 4.6 Mensajes recibidos en el Backend. Recorrido sector NORTE ("Backend Sigfox", 2021).



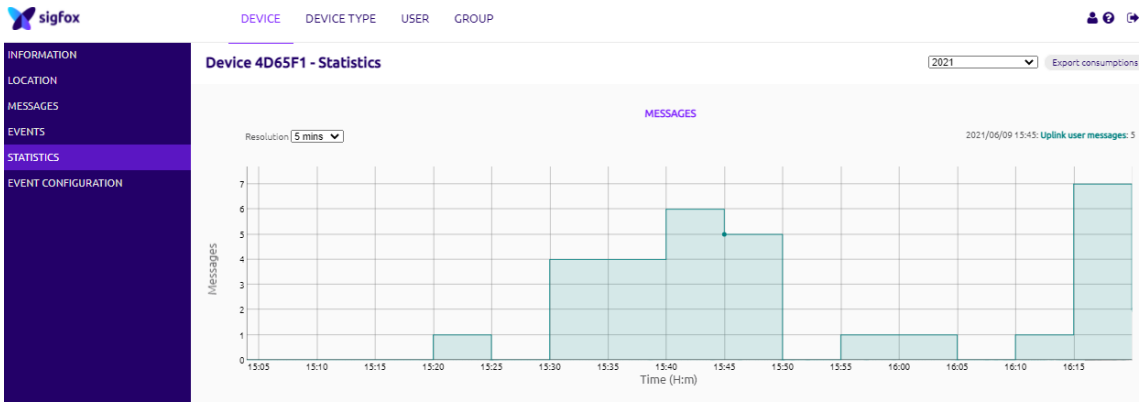
Nota: La intensidad de la señal representada de forma gráfica indica niveles según el color, se observa color verde que corresponde a buena señal y naranja a señal intermedia.

Figura 4.7 Mensajes recibidos en el Backend. Recorrido sector SUR ("Backend Sigfox", 2021).

Nota: La intensidad de la señal representada de forma gráfica indica niveles según el color, se observa color naranja que corresponde a señal intermedia y rojo a mala señal.

En la figura 4.2.3 se observar cómo en intervalos de tiempo de 5 minutos el dispositivo dependiendo de su ubicación varía considerablemente el en número de mensajes enviados en dicho intervalo de tiempo, tomado de las estadísticas del dispositivo en el Backend.

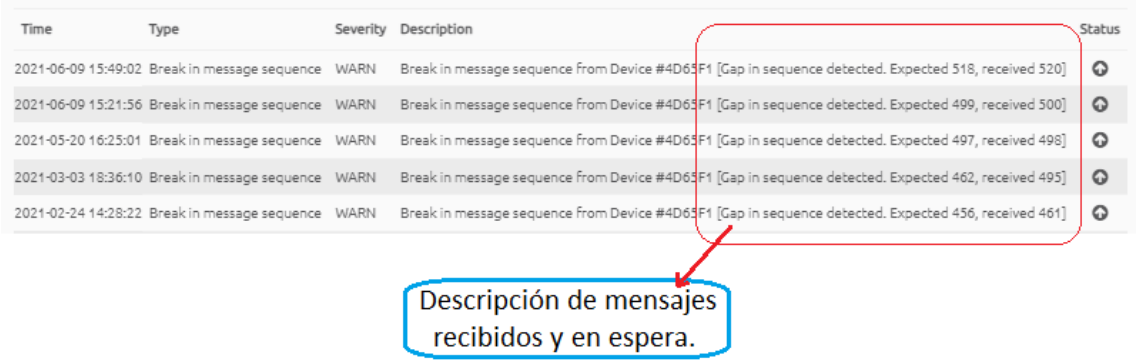
Figura 4.8 Número de mensajes enviados en intervalos de tiempo de 5 minutos ("Backend Sigfox", 2021).



El dispositivo en caso de no estar en zona de cobertura todo el tiempo está intentando enviar los mensajes al backend y se lo comprobó en este tal como muestra la figura 4.9. La secuencia de mensajes no siempre es continua, esto quiere decir que envía un mensaje,

pero por alguna razón el siguiente o los siguientes el dispositivo envió, pero el backend no recibió.

Figura 4.9 Backend, ventana de eventos ("Backend Sigfox", 2021).



En la Figura 4.9 en la primera línea se observa que el backend recibió el mensaje número 520 enviado por el dispositivo, pero sigue a la espera del número 18, esto quiere decir que el número 18 y número 19 por algún motivo no han llegado al backend. En este caso en particular los mensajes no recibidos son 2 pero esto puede variar y pueden ser más mensajes. Esto se ve reflejado directamente en los datos que son enviados a la aplicación Android y no se obtiene de forma correcta una ubicación precisa y en tiempo real.

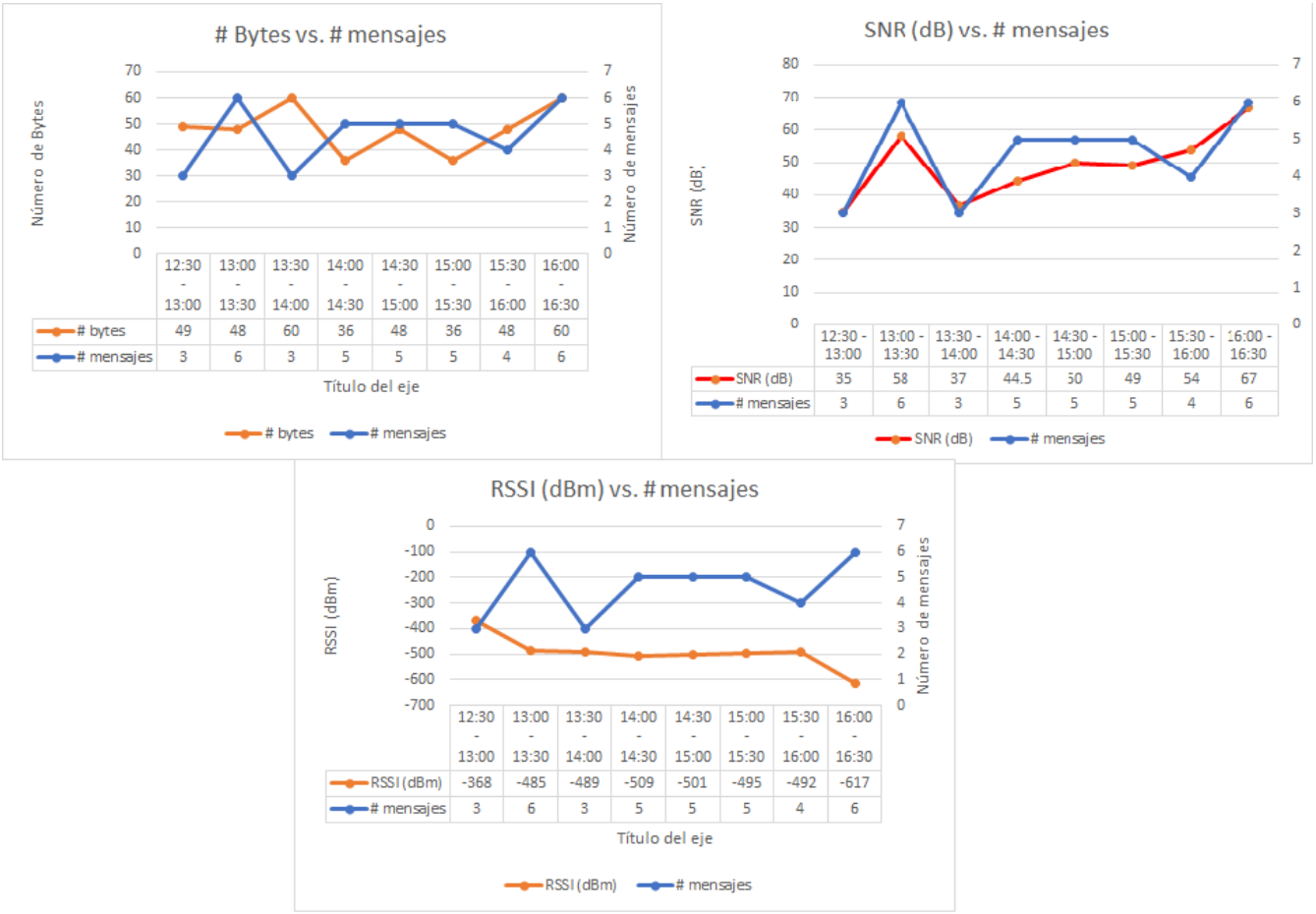
En la Tabla 4.2 se tiene una muestra del backend de Sigfox en la cual se muestra en un intervalo de tiempo de 4 horas: el número de callbacks, número de bytes, SNR en dB y RSSI en dBm en intervalos de 30 minutos, es importante mencionar que son datos aislados tomados al azar.

Tabla 4.3 Estadísticas de parámetros del dispositivo Sigfox ("Backend Sigfox", 2021).

Tiempo	# mensajes	# bytes	SNR (dB)	RSSI (dBm)
12:30 - 13:00	3	49	35	-368
13:00 - 13:30	6	48	58	-485
13:30 - 14:00	3	60	37	-489
14:00 - 14:30	5	36	44.5	-509
14:30 - 15:00	5	48	50	-501
15:00 - 15:30	5	36	49	-495
15:30 - 16:00	4	48	54	-492
16:00 - 16:30	6	60	67	-617

En la Figura 4.10 se representa de forma gráfica cada uno de los parámetros del dispositivo presentados en las estadísticas del backend versus el número de callbacks en un periodo de tiempo dividido en intervalos de 30 minutos.

Figura 4.10 Gráficas de # de mensajes versus: # de bytes, SNR (dB) y RSSI (dBm)



4.3 PRUEBA DE CONSUMO DE ENERGÍA

Con el dispositivo terminado se probó la batería, para este propósito se realizó las pruebas con la batería LIPO de 3.7 V. completamente cargada, realizando recorridos constantes en dos días y lugares distintos tanto en donde se obtiene buena cobertura de señal y donde suele no funcionar correctamente en movimiento. Además, se comprobó el tiempo de carga de la batería en su estado completo una vez agotada.

Tabla 4.4 Pruebas de tiempo de duración de batería.

Fecha	Hora de inicio	Hora de final	Duración
09/02/2021	14:00	18:27	6 horas 27 minutos
11/02/2021	09:00	15:49	6 horas 49 minutos

Elaborado por: Rea Alex

Tabla 4.5 Pruebas de tiempo de carga de batería.

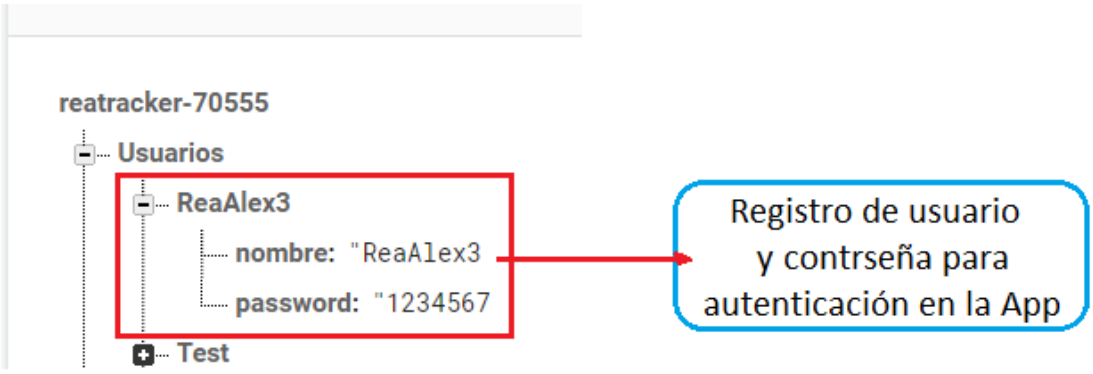
Fecha	Hora de inicio	Hora de final	Duración
09/02/2021	08:00	08:47	47 minutos
10/02/2021	21:00	21:43	43 minutos

Elaborado por: Rea Alex

4.4 FUNCIONAMIENTO DE LA APLICACIÓN.

Conformada por dos pantallas principales se instaló la aplicación en un teléfono inteligente Android comprobando en primera instancia el funcionamiento correcto de la autenticación, la cual se realizó con un usuario y una contraseña el cual fue configurado en FireBase tal como se puede observar en la figura 4.10

Figura 4.11 Registro de usuario y contraseña en Firebase

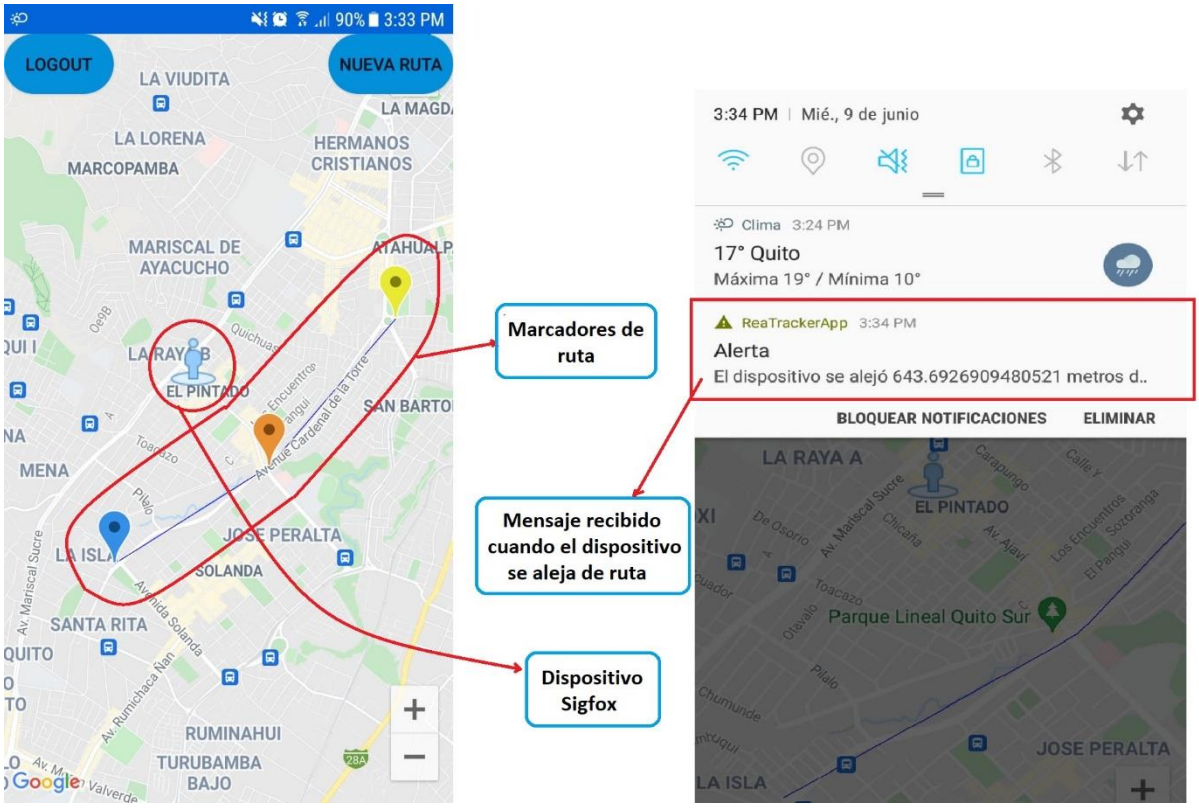


La segunda pantalla muestra la parte principal de la aplicación. Se observa el mapa del servicio Google Maps. Sobre este se observa la ubicación del dispositivo representado por el dibujo de una persona, la precisión y actualización en la aplicación Android depende del estado de la red, si el dispositivo envió mensajes y estos llegaron a la base de datos. Esta pantalla también muestra dos botones, el “LOGOUT” permitiendo de

forma correcta abandonar la autenticación en la aplicación y el de "NUEVA RUTA" el cual permite de forma correcta colocar 3 marcadores para marcar una ruta de control, tal como se indica esta permite correctamente el control de la ruta ya que en caso de alejarse de la ruta la aplicación envía una notificación al teléfono inteligente indicando este particular.

La aplicación envía una notificación de alerta de ruta en el caso de alejarse una distancia mayor a 220 m. esta distancia se ha tomado en cuenta debido a la falta de precisión del GPS y de la geografía de las calles, tal como se observa en la Figura 4.12 la aplicación muestra la alerta en la pantalla principal y además envía un mensaje al panel de notificaciones del teléfono inteligente. A la izquierda se observa la aplicación y sobre ella la ubicación y los 3 marcadores colocados para indicar una ruta. Al alejarse el dispositivo de los marcadores más de 220m. A la derecha se observa la notificación de la aplicación mostrando los metros que se ha alejado.

Figura 4.12 Alerta de desvío de ruta tanto en la aplicación como en el panel de notificaciones.



CONCLUSIONES

- Se desarrolló un dispositivo de rastreo de niños para la monitorización del desplazamiento y la ubicación, de forma parcial ya que al trabajar depende de factores externos y descubiertos a lo largo del desarrollo del proyecto

- Este proyecto permite establecer una nueva idea para la implementación de comunicaciones inalámbricas con tecnologías poco convencionales que permiten la cobertura de grandes extensiones y con un bajo consumo de potencia.

- Gracias al módulo GPS incorporado en el prototipo se pudo obtener las coordenadas geográficas del dispositivo. Mediante la red Sigfox son almacenados en un servidor y mostrados en un smartphone mediante una aplicación Android, teniendo así la ubicación en tiempo real.

- Se observó que la precisión de la ubicación en tiempo real del dispositivo depende de diversos factores que influyen para la exactitud de esta. Por parte del GPS la precisión de las coordenadas depende de la cantidad de satélites que estén a la vista del módulo mientras que por parte del módulo Sigfox dependerá de la intensidad de la señal con la que se envíen los mensajes esto a su vez depende de la red desplegada por el operador que en este caso es WND Group para el Ecuador.

- Se pudo constatar el funcionamiento de la red Sigfox considerando como posible estación base el cerro Pichincha ya que con línea de vista directa la intensidad de señal de la red es buena, esta señal se ve afectada por otros factores como son: el clima, mostrando un correcto funcionamiento con el cielo despejado y empeorando con niebla o lluvia, la velocidad de desplazamiento del dispositivo, mientras más lento se mueve mejor será el envío de mensajes.

- Se comprobó que un parámetro importante para el envío de mensajes por la red Sigfox es la velocidad de desplazamiento del dispositivo influyendo negativamente en el monitoreo en tiempo real, sobre todo en zonas donde la intensidad de la red es de media y baja calidad, al trasladarse a mayor velocidad el dispositivo no logra enviar los mensajes de ubicación.

- Al contar el dispositivo con un módulo GPS, el mismo que cuenta con su propia antena se comprobó que el correcto funcionamiento del dispositivo también depende del lugar donde se encuentre este ya que en lugares cerrados al no contar con disponibilidad de satélites y sumado a la mala señal de la red Sigfox no se cumple con el monitoreo del dispositivo de forma correcta

- Fue posible comprobar el sistema de notificaciones en caso de desvío de ruta, considerando un correcto funcionamiento y en caso de el desvío el usuario recibe en aproximadamente 3 segundos la notificación de alerta y la distancia desviada respecto a la ruta (marcadores) ingresada en el mapa de la aplicación.

- Se implementó el dispositivo electrónico considerando la discreción de este al almacenarlo en una mochila de niño y por tal razón se realizó de la manera más modular posible teniendo como resultado un dispositivo en tamaño normal y que no molesta al ser llevado por un niño. Además de contar con una carcasa de plástico que protege al circuito electrónico.

- Tras implementar el dispositivo y la aplicación móvil Android se comprobó el funcionamiento de estos cuando la red se encuentra disponible y validando los posibles sectores donde funciona correctamente y otros donde no es posible el funcionamiento del prototipo. Esto tras probarlo en varios sectores de la ciudad de Quito.

- Al tratarse de una red de bajo consumo se comprobó la durabilidad de la batería en una jornada estudiantil de un niño y la carga de esta, permitiendo una autonomía del dispositivo.

RECOMENDACIONES

- El dispositivo al contar con una batería LIPO deberá mantenerse en un lugar fresco y seco para su correcto funcionamiento, evitar accidentes y alargar su vida útil.

- Implementar una solución al problema de la cobertura de la red Sigfox ya que la desplegada en la ciudad presenta sectores sin cubrirlos.

- Implementar una opción adicional al envío de las alertas en el seguimiento de ruta, como puede ser envío de un correo electrónico, sms o mensaje de Wathsapp.

- Utilizar un módulo GPS con mejor calidad de recepción de señal, el cual permitirá obtener información en lugares cerrados como subsuelos.

- Utilizar una red de comunicación inalámbrica alterna como GSM o LORA Wan la cual permita activarse en caso de perder la de Sigfox.

REFERENCIAS

Hora (2020). Cientos De Niños Desaparecidos. Recuperado el 12 de marzo 2020, de https://www.lahora.com.ec/noticia/1000101114/cientos-de-nios-desaparecidos

SAP (2016). What is the Internet of Things (IoT)?. Recuperado el 13 de marzo 2020, de https://www.sap.com/trends/internet-of-things.html

ResearchGate. (2020). Propuesta de arquitectura para el Internet de las Cosas. Recuperado el 4 de abril 2020, de https://www.researchgate.net/figure/Arquitectura-para-Internet-de-las-Cosas-En-la-cima-de-la-arquitectura-esta-la-Capa-de_fig1_320353907.

Technology | Sigfox. (2020). Recuperado el 9 de junio 2020, de https://www.sigfox.com/en/what-sigfox/technology#id_security

Technology | Sigfox. (2020). Recuperado el 12 de junio 2020, de https://www.sigfox.com/en/what-sigfox/technology#id_technology

Technology | Sigfox. (2020). Recuperado el 12 de junio 2020, de

https://www.sigfox.com/en/what-sigfox/technology#id_radio

Technology | Sigfox. (2020). Recuperado el 12 de junio 2020, de

https://www.sigfox.com/en/what-sigfox/technology#id_network

Sanchez J., (2020). Diseño de un registrador compacto basado en tecnología de ultra bajo consumo y comunicación sigfox para internet de las cosas (iot) con aplicación en proyectos de ciudades inteligentes [Tésis de máster, Universidad Politécnica de Valencia] Repositorio Institucional UPV. http://hdl.handle.net/10251/147504

Sanchez, J. (2016). Aplicación y análisis de un sistema SIGFOX en una empresa dedicada a la producción de materiales cerámicos [Tésis de grado, Universitat Politécnica de Catalunya Barcelonatech] Repositorio Institucional UPCommons. http://hdl.handle.net/2117/105196

Android developers (2020). Recuperado el 11 de noviembre de 2020, de https://developer.android.com/studio/intro?hl=es-419

Aveda R., (2020). Qué es Android: todo sobre el sistema operativo de Google. Recuperado el 11 de noviembre 2020, de https://www.adslzone.net/reportajes/software/que-es-android/

Perez M. (2016). Firebase, qué es y para qué sirve la plataforma de Google. Recuperado el 11 de noviembre de 2020, de https://www.iebschool.com/blog/firebase-que-es-para-que-sirve-la-plataforma-desarroladores-google-seo-sem/

U-blox. (2011) NEO-6 GPS Modules DataSheet. Recuperado el 23 de febrero de 2021, de https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf

Chris Veness, w. (2021). Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. Recuperado el 2 de abril de 2021, de https://www.movable-type.co.uk/scripts/latlong.html

PyCom LoPy4 ESP32 - LoRa module, WiFi, Bluetooth. (2021). PyCom LoPy4 ESP32 - LoRa module, WiFi, Bluetooth BLE, SigFox + Python API Recuperado el 26 de julio de 2021, de https://botland.store/pycom-iot-esp32/12390-pycom-lopy4-esp32-lora-module-wifi-bluetooth-ble-sigfox-python-api-700461908555.html

Pic, M. (2021). Módulo Gps Gy-Neo6mv2 Neo-6m Arduino Pic. Recuperado el 26 de julio de 2021, de https://www.arcaelectronica.com/products/modulo-gps-gy-neo6mv2-neo-6m-arduino-pic

Ryan. (2018). Best C Rating for RC LiPo Battery -. Recuperado el 26 de julio de 2021, de https://www.radiocontrolinfo.com/best-c-rating-for-rc-lipo-battery/

Sigfox Buy. (2021). Buy Sigfox Connectivity Recurepado el 27 de julio de 2021, de https://buy.sigfox.com/

Backend Sigfox. (2021). Sigfox Device List. Recuperado el 27 de julio de 2021, de https://backend.sigfox.com/device/list

Console Firebase. (2021). Firebase Recuperado el 27 de julio de 2021, de https://console.firebase.google.com/u/0/project/reatracker-70555/overview

ANEXO 1 Programacion del dispositivo Sigfox LoPy 4.0

Clase Principal

```
from network import Sigfox
import socket
from machine import UART
#from pyb import UART
import time
import pycom
import array as arr
from micropyGPS import MicropyGPS
from machine import Pin

print("inicio TEST GPS")
#Funcion para leer el ADC con el valor del voltaje de bateria
def lecturaADC():
    #Habilitar la lectura del divisor de voltaje
    MosfetPin.value(1)
    time.sleep(5)

    #desabilitar la lectura del ADC para ahorrar bateria
    MosfetPin.value(0)
    time.sleep(5)
    #transformar valor a un array de 2 BYTES
    nivel_de_batt = arr.array('B',([10,24]))
    return nivel_de_batt

#FUNCION PARA CONVERTIR LIST DEL GPS A DECIMAL
def convertirDec

    direccion = {'N':1, 'S':-1, 'E': 1, 'W':-1}
    #new = old.replace(u'°',' ').replace('\'',' ').replace('"',' ')
    #new = new.split()
    #new_dir = new.pop()
    new_dir = gpsList.pop()
    #new.extend([0,0,0])
    #return (int(gpsList[0])+int(gpsList[1])/60.0+int(gpsList[2])/3600.0) *
direccion[new_dir]
    print("-->",gpsList[1])
    return round((int(gpsList[0])+(gpsList[1])/60.0) * direccion[new_dir],6)
#FUNCION PARA CONVERTIR EN BYTES DE ARRAY
def convertir_LatLon(latitud, longitud):
    lat = int((latitud+90)*10000)
    lon = int((longitud+180)*10000)
    print("lat: ",lat)
    print("lon: ",lon)
    latStr= str(lat)
    lonStr= str(lon)
    #la B es porque necesitamos los valores en bytes en el array q vamos a
enviar
    coords = arr.array('B',[0,0,0,0,0,0])
    print("tipo",type(coords))
    """
```

```
    coords[0] = lat
    coords[1] = (lat >> 8)
    coords[2] = (lat >> 16)

    coords[3] = lon
    coords[4] = (lon >> 8)
    coords[5] = (lon >> 16)
    """
    coord2 = arr.array('B')
    #coord2.extend([])
    coord2.append(int(latStr[0:2]))
    coord2.append(int(latStr[2:4]))
    coord2.append(int(latStr[4:6]))
    #coord2.append(int(latStr[6:8])) #necesario para latitudes mas orientales

    coord2.append(int(lonStr[0:2]))
    coord2.append(int(lonStr[2:4]))
    coord2.append(int(lonStr[4:6]))
    if(lonStr[6:8]!=""):
        coord2.append(int(lonStr[6:8]))

    return coord2
```

Clase Librerias del GPS

```
"""
# MicropyGPS - a GPS NMEA sentence parser for Micropython/Python 3.X
# Copyright (c) 2017 Michael Calvin McCoy (calvin.mccoy@protonmail.com)
# The MIT License (MIT) - see LICENSE file
"""

# TODO:
# Time Since First Fix
# Distance/Time to Target
# More Helper Functions
# Dynamically limit sentences types to parse
```

```
from math import floor, modf

# Import utime or time for fix time handling
try:
    # Assume running on MicroPython
    import utime
except ImportError:
    # Otherwise default to time module for non-embedded implementations
    # Should still support millisecond resolution.
    import time


class MicropyGPS(object):
    """GPS NMEA Sentence Parser. Creates object that stores all relevant GPS
data and statistics.
    Parses sentences one character at a time using update(). """

    # Max Number of Characters a valid sentence can be (based on GGA
sentence)
    SENTENCE_LIMIT = 90
    __HEMISPHERES = ('N', 'S', 'E', 'W')
    __NO_FIX = 1
    __FIX_2D = 2
    __FIX_3D = 3
    __DIRECTIONS = ('N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S',
'SSW', 'SW', 'WSW', 'W',
                    'WNW', 'NW', 'NNW')
    __MONTHS = ('January', 'February', 'March', 'April', 'May',
                'June', 'July', 'August', 'September', 'October',
                'November', 'December')

    def __init__(self, local_offset=0, location_formatting='ddm'):
        """
        Setup GPS Object Status Flags, Internal Data Registers, etc
            local_offset (int): Timzone Difference to UTC
            location_formatting (str): Style For Presenting
Longitude/Latitude:
                                        Decimal Degree Minute (ddm) - 40°
26.767' N
                                        Degrees Minutes Seconds (dms) - 40°
26' 46" N
                                        Decimal Degrees (dd) - 40.446° N
        """

        ######################
        # Object Status Flags
        self.sentence_active = False
        self.active_segment = 0
        self.process_crc = False
        self.gps_segments = []
        self.crc_xor = 0
        self.char_count = 0
        self.fix_time = 0
```

```
        ######################
        # Sentence Statistics
        self.crc_fails = 0
        self.clean_sentences = 0
        self.parsed_sentences = 0

        ######################
        # Logging Related
        self.log_handle = None
        self.log_en = False

        ######################
        # Data From Sentences
        # Time
        self.timestamp = [0, 0, 0]
        self.date = [0, 0, 0]
        self.local_offset = local_offset

        # Position/Motion
        self._latitude = [0, 0.0, 'N']
        self._longitude = [0, 0.0, 'W']
        self.coord_format = location_formatting
        self.speed = [0.0, 0.0, 0.0]
        self.course = 0.0
        self.altitude = 0.0
        self.geoid_height = 0.0

        # GPS Info
        self.satellites_in_view = 0
        self.satellites_in_use = 0
        self.satellites_used = []
        self.last_sv_sentence = 0
        self.total_sv_sentences = 0
        self.satellite_data = dict()
        self.hdop = 0.0
        self.pdop = 0.0
        self.vdop = 0.0
        self.valid = False
        self.fix_stat = 0
        self.fix_type = 1

    ########################################
    # Coordinates Translation Functions
    ########################################
    @property
    def latitude(self):
        """Format Latitude Data Correctly"""
        if self.coord_format == 'dd':
            decimal_degrees = self._latitude[0] + (self._latitude[1] / 60)
            return [decimal_degrees, self._latitude[2]]
        elif self.coord_format == 'dms':
            minute_parts = modf(self._latitude[1])
            seconds = round(minute_parts[0] * 60)
            return [self._latitude[0], int(minute_parts[1]), seconds,
```

```
self._latitude[2]]
        else:
            return self._latitude

    @property
    def longitude(self):
        """Format Longitude Data Correctly"""
        if self.coord_format == 'dd':
            decimal_degrees = self._longitude[0] + (self._longitude[1] / 60)
            return [decimal_degrees, self._longitude[2]]
        elif self.coord_format == 'dms':
            minute_parts = modf(self._longitude[1])
            seconds = round(minute_parts[0] * 60)
            return [self._longitude[0], int(minute_parts[1]), seconds,
self._longitude[2]]
        else:
            return self._longitude

    ########################################
    # Logging Related Functions
    ########################################
    def start_logging(self, target_file, mode="append"):
        """
        Create GPS data log object
        """
        # Set Write Mode Overwrite or Append
        mode_code = 'w' if mode == 'new' else 'a'

        try:
            self.log_handle = open(target_file, mode_code)
        except AttributeError:
            print("Invalid FileName")
            return False

        self.log_en = True
        return True

    def stop_logging(self):
        """
        Closes the log file handler and disables further logging
        """
        try:
            self.log_handle.close()
        except AttributeError:
            print("Invalid Handle")
            return False

        self.log_en = False
        return True

    def write_log(self, log_string):
        """Attempts to write the last valid NMEA sentence character to the
active file handler
        """
```

```
        try:
            self.log_handle.write(log_string)
        except TypeError:
            return False
        return True

    ########################################
    # Sentence Parsers
    ########################################
    def gprmc(self):
        """Parse Recommended Minimum Specific GPS/Transit data (RMC)Sentence.
        Updates UTC timestamp, latitude, longitude, Course, Speed, Date, and
fix status
        """

        # UTC Timestamp
        try:
            utc_string = self.gps_segments[1]

            if utc_string:  # Possible timestamp found
                hours = (int(utc_string[0:2]) + self.local_offset) % 24
                minutes = int(utc_string[2:4])
                seconds = float(utc_string[4:])
                self.timestamp = (hours, minutes, seconds)
            else:  # No Time stamp yet
                self.timestamp = (0, 0, 0)

        except ValueError:  # Bad Timestamp value present
            return False

        # Date stamp
        try:
            date_string = self.gps_segments[9]

            # Date string printer function assumes to be year >=2000,
            # date_string() must be supplied with the correct century
argument to display correctly
            if date_string:  # Possible date stamp found
                day = int(date_string[0:2])
                month = int(date_string[2:4])
                year = int(date_string[4:6])
                self.date = (day, month, year)
            else:  # No Date stamp yet
                self.date = (0, 0, 0)

        except ValueError:  # Bad Date stamp value present
            return False

        # Check Receiver Data Valid Flag
        if self.gps_segments[2] == 'A':  # Data from Receiver is Valid/Has
Fix

            # Longitude / Latitude
            try:
```

```
            # Latitude
            l_string = self.gps_segments[3]
            lat_degs = int(l_string[0:2])
            lat_mins = float(l_string[2:])
            lat_hemi = self.gps_segments[4]

            # Longitude
            l_string = self.gps_segments[5]
            lon_degs = int(l_string[0:3])
            lon_mins = float(l_string[3:])
            lon_hemi = self.gps_segments[6]
        except ValueError:
            return False

        if lat_hemi not in self.__HEMISPHERES:
            return False

        if lon_hemi not in self.__HEMISPHERES:
            return False

        # Speed
        try:
            spd_knt = float(self.gps_segments[7])
        except ValueError:
            return False

        # Course
        try:
            if self.gps_segments[8]:
                course = float(self.gps_segments[8])
            else:
                course = 0.0
        except ValueError:
            return False

        # TODO - Add Magnetic Variation

        # Update Object Data
        self._latitude = [lat_degs, lat_mins, lat_hemi]
        self._longitude = [lon_degs, lon_mins, lon_hemi]
        # Include mph and hm/h
        self.speed = [spd_knt, spd_knt * 1.151, spd_knt * 1.852]
        self.course = course
        self.valid = True

        # Update Last Fix Time
        self.new_fix_time()

    else:  # Clear Position Data if Sentence is 'Invalid'
        self._latitude = [0, 0.0, 'N']
        self._longitude = [0, 0.0, 'W']
        self.speed = [0.0, 0.0, 0.0]
        self.course = 0.0
        self.valid = False
```

```
        return True

    def gpgll(self):
        """Parse Geographic Latitude and Longitude (GLL)Sentence. Updates UTC
timestamp, latitude,
        longitude, and fix status"""

        # UTC Timestamp
        try:
            utc_string = self.gps_segments[5]

            if utc_string:  # Possible timestamp found
                hours = (int(utc_string[0:2]) + self.local_offset) % 24
                minutes = int(utc_string[2:4])
                seconds = float(utc_string[4:])
                self.timestamp = (hours, minutes, seconds)
            else:  # No Time stamp yet
                self.timestamp = (0, 0, 0)

        except ValueError:  # Bad Timestamp value present
            return False

        # Check Receiver Data Valid Flag
        if self.gps_segments[6] == 'A':  # Data from Receiver is Valid/Has
Fix

            # Longitude / Latitude
            try:
                # Latitude
                l_string = self.gps_segments[1]
                lat_degs = int(l_string[0:2])
                lat_mins = float(l_string[2:])
                lat_hemi = self.gps_segments[2]

                # Longitude
                l_string = self.gps_segments[3]
                lon_degs = int(l_string[0:3])
                lon_mins = float(l_string[3:])
                lon_hemi = self.gps_segments[4]
            except ValueError:
                return False

            if lat_hemi not in self.__HEMISPHERES:
                return False

            if lon_hemi not in self.__HEMISPHERES:
                return False

            # Update Object Data
            self._latitude = [lat_degs, lat_mins, lat_hemi]
            self._longitude = [lon_degs, lon_mins, lon_hemi]
            self.valid = True
```

```
            # Update Last Fix Time
            self.new_fix_time()

        else:  # Clear Position Data if Sentence is 'Invalid'
            self._latitude = [0, 0.0, 'N']
            self._longitude = [0, 0.0, 'W']
            self.valid = False

        return True

    def gpvtg(self):
        """Parse Track Made Good and Ground Speed (VTG) Sentence. Updates
speed and course"""
        try:
            course = float(self.gps_segments[1])
            spd_knt = float(self.gps_segments[5])
        except ValueError:
            return False

        # Include mph and km/h
        self.speed = (spd_knt, spd_knt * 1.151, spd_knt * 1.852)
        self.course = course
        return True

    def gpgga(self):
        """Parse Global Positioning System Fix Data (GGA) Sentence. Updates
UTC timestamp, latitude, longitude,
        fix status, satellites in use, Horizontal Dilution of Precision
(HDOP), altitude, geoid height and fix status"""

        try:
            # UTC Timestamp
            utc_string = self.gps_segments[1]

            # Skip timestamp if receiver doesn't have on yet
            if utc_string:
                hours = (int(utc_string[0:2]) + self.local_offset) % 24
                minutes = int(utc_string[2:4])
                seconds = float(utc_string[4:])
            else:
                hours = 0
                minutes = 0
                seconds = 0.0

            # Number of Satellites in Use
            satellites_in_use = int(self.gps_segments[7])

            # Get Fix Status
            fix_stat = int(self.gps_segments[6])

        except (ValueError, IndexError):
            return False

        try:
```

```
        # Horizontal Dilution of Precision
        hdop = float(self.gps_segments[8])
    except (ValueError, IndexError):
        hdop = 0.0

    # Process Location and Speed Data if Fix is GOOD
    if fix_stat:

        # Longitude / Latitude
        try:
            # Latitude
            l_string = self.gps_segments[2]
            lat_degs = int(l_string[0:2])
            lat_mins = float(l_string[2:])
            lat_hemi = self.gps_segments[3]

            # Longitude
            l_string = self.gps_segments[4]
            lon_degs = int(l_string[0:3])
            lon_mins = float(l_string[3:])
            lon_hemi = self.gps_segments[5]
        except ValueError:
            return False

        if lat_hemi not in self.__HEMISPHERES:
            return False

        if lon_hemi not in self.__HEMISPHERES:
            return False

        # Altitude / Height Above Geoid
        try:
            altitude = float(self.gps_segments[9])
            geoid_height = float(self.gps_segments[11])
        except ValueError:
            altitude = 0
            geoid_height = 0

        # Update Object Data
        self._latitude = [lat_degs, lat_mins, lat_hemi]
        self._longitude = [lon_degs, lon_mins, lon_hemi]
        self.altitude = altitude
        self.geoid_height = geoid_height

    # Update Object Data
    self.timestamp = [hours, minutes, seconds]
    self.satellites_in_use = satellites_in_use
    self.hdop = hdop
    self.fix_stat = fix_stat

    # If Fix is GOOD, update fix timestamp
    if fix_stat:
        self.new_fix_time()
```

```
        return True

    def gpgsa(self):
        """Parse GNSS DOP and Active Satellites (GSA) sentence. Updates GPS
fix type, list of satellites used in
        fix calculation, Position Dilution of Precision (PDOP), Horizontal
Dilution of Precision (HDOP), Vertical
        Dilution of Precision, and fix status"""

        # Fix Type (None,2D or 3D)
        try:
            fix_type = int(self.gps_segments[2])
        except ValueError:
            return False

        # Read All (up to 12) Available PRN Satellite Numbers
        sats_used = []
        for sats in range(12):
            sat_number_str = self.gps_segments[3 + sats]
            if sat_number_str:
                try:
                    sat_number = int(sat_number_str)
                    sats_used.append(sat_number)
                except ValueError:
                    return False
            else:
                break

        # PDOP,HDOP,VDOP
        try:
            pdop = float(self.gps_segments[15])
            hdop = float(self.gps_segments[16])
            vdop = float(self.gps_segments[17])
        except ValueError:
            return False

        # Update Object Data
        self.fix_type = fix_type

        # If Fix is GOOD, update fix timestamp
        if fix_type > self.__NO_FIX:
            self.new_fix_time()

        self.satellites_used = sats_used
        self.hdop = hdop
        self.vdop = vdop
        self.pdop = pdop

        return True

    def gpgsv(self):
        """Parse Satellites in View (GSV) sentence. Updates number of SV
Sentences,the number of the last SV sentence
        parsed, and data on each satellite present in the sentence"""
```

```
        try:
            num_sv_sentences = int(self.gps_segments[1])
            current_sv_sentence = int(self.gps_segments[2])
            sats_in_view = int(self.gps_segments[3])
        except ValueError:
            return False

        # Create a blank dict to store all the satellite data from this
sentence in:
        # satellite PRN is key, tuple containing telemetry is value
        satellite_dict = dict()

        # Calculate  Number of Satelites to pull data for and thus how many
segment positions to read
        if num_sv_sentences == current_sv_sentence:
            # Last sentence may have 1-4 satellites; 5 - 20 positions
            sat_segment_limit = (sats_in_view - ((num_sv_sentences - 1) * 4))
* 5
        else:
            sat_segment_limit = 20  # Non-last sentences have 4 satellites
and thus read up to position 20

        # Try to recover data for up to 4 satellites in sentence
        for sats in range(4, sat_segment_limit, 4):

            # If a PRN is present, grab satellite data
            if self.gps_segments[sats]:
                try:
                    sat_id = int(self.gps_segments[sats])
                except (ValueError,IndexError):
                    return False

                try:  # elevation can be null (no value) when not tracking
                    elevation = int(self.gps_segments[sats+1])
                except (ValueError,IndexError):
                    elevation = None

                try:  # azimuth can be null (no value) when not tracking
                    azimuth = int(self.gps_segments[sats+2])
                except (ValueError,IndexError):
                    azimuth = None

                try:  # SNR can be null (no value) when not tracking
                    snr = int(self.gps_segments[sats+3])
                except (ValueError,IndexError):
                    snr = None
            # If no PRN is found, then the sentence has no more satellites to
read
            else:
                break

            # Add Satellite Data to Sentence Dict
            satellite_dict[sat_id] = (elevation, azimuth, snr)
```

```
        # Update Object Data
        self.total_sv_sentences = num_sv_sentences
        self.last_sv_sentence = current_sv_sentence
        self.satellites_in_view = sats_in_view

        # For a new set of sentences, we either clear out the existing sat
data or
        # update it as additional SV sentences are parsed
        if current_sv_sentence == 1:
            self.satellite_data = satellite_dict
        else:
            self.satellite_data.update(satellite_dict)

        return True

    ##########################################
    # Data Stream Handler Functions
    ##########################################

    def new_sentence(self):
        """Adjust Object Flags in Preparation for a New Sentence"""
        self.gps_segments = ['']
        self.active_segment = 0
        self.crc_xor = 0
        self.sentence_active = True
        self.process_crc = True
        self.char_count = 0

    def update(self, new_char):
        """Process a new input char and updates GPS object if necessary based
on special characters ('$', ',', '*')
        Function builds a list of received string that are validate by CRC
prior to parsing by the  appropriate
        sentence function. Returns sentence type on successful parse, None
otherwise"""

        valid_sentence = False

        # Validate new_char is a printable char
        ascii_char = ord(new_char)

        if 10 <= ascii_char <= 126:
            self.char_count += 1

            # Write Character to log file if enabled
            if self.log_en:
                self.write_log(new_char)

            # Check if a new string is starting ($)
            if new_char == '$':
                self.new_sentence()
                return None

            elif self.sentence_active:
```

```
                # Check if sentence is ending (*)
                if new_char == '*':
                    self.process_crc = False
                    self.active_segment += 1
                    self.gps_segments.append('')
                    return None

                # Check if a section is ended (,), Create a new substring to
feed
                # characters to
                elif new_char == ',':
                    self.active_segment += 1
                    self.gps_segments.append('')

                # Store All Other printable character and check CRC when
ready
                else:
                    self.gps_segments[self.active_segment] += new_char

                    # When CRC input is disabled, sentence is nearly complete
                    if not self.process_crc:

                        if len(self.gps_segments[self.active_segment]) == 2:
                            try:
                                final_crc =
int(self.gps_segments[self.active_segment], 16)
                                if self.crc_xor == final_crc:
                                    valid_sentence = True
                                else:
                                    self.crc_fails += 1
                            except ValueError:
                                pass  # CRC Value was deformed and could not
have been correct

                # Update CRC
                if self.process_crc:
                    self.crc_xor ^= ascii_char

                # If a Valid Sentence Was received and it's a supported
sentence, then parse it!!
                if valid_sentence:
                    self.clean_sentences += 1  # Increment clean sentences
received
                    self.sentence_active = False  # Clear Active Processing
Flag

                    if self.gps_segments[0] in self.supported_sentences:

                        # parse the Sentence Based on the message type,
return True if parse is clean
                        if
self.supported_sentences[self.gps_segments[0]](self):
```

```
                            # Let host know that the GPS object was updated
by returning parsed sentence type
                            self.parsed_sentences += 1
                            return self.gps_segments[0]

                # Check that the sentence buffer isn't filling up with Garage
waiting for the sentence to complete
                if self.char_count > self.SENTENCE_LIMIT:
                    self.sentence_active = False

        # Tell Host no new sentence was parsed
        return None

    def new_fix_time(self):
        """Updates a high resolution counter with current time when fix is
updated. Currently only triggered from
        GGA, GSA and RMC sentences"""
        try:
            self.fix_time = utime.ticks_ms()
        except NameError:
            self.fix_time = time.time()

    ########################################
    # User Helper Functions
    # These functions make working with the GPS object data easier
    ########################################

    def satellite_data_updated(self):
        """
        Checks if the all the GSV sentences in a group have been read, making
satellite data complete
        :return: boolean
        """
        if self.total_sv_sentences > 0 and self.total_sv_sentences ==
self.last_sv_sentence:
            return True
        else:
            return False

    def unset_satellite_data_updated(self):
        """
        Mark GSV sentences as read indicating the data has been used and
future updates are fresh
        """
        self.last_sv_sentence = 0

    def satellites_visible(self):
        """
        Returns a list of of the satellite PRNs currently visible to the
receiver
        :return: list
        """
        return list(self.satellite_data.keys())
```

```
    def time_since_fix(self):
        """Returns number of millisecond since the last sentence with a valid
fix was parsed. Returns 0 if
        no fix has been found"""

        # Test if a Fix has been found
        if self.fix_time == 0:
            return -1

        # Try calculating fix time using utime; if not running MicroPython
        # time.time() returns a floating point value in secs
        try:
            current = utime.ticks_diff(utime.ticks_ms(), self.fix_time)
        except NameError:
            current = (time.time() - self.fix_time) * 1000  # ms

        return current

    def compass_direction(self):
        """
        Determine a cardinal or inter-cardinal direction based on current
course.
        :return: string
        """
        # Calculate the offset for a rotated compass
        if self.course >= 348.75:
            offset_course = 360 - self.course
        else:
            offset_course = self.course + 11.25

        # Each compass point is separated by 22.5 degrees, divide to find
lookup value
        dir_index = floor(offset_course / 22.5)

        final_dir = self.__DIRECTIONS[dir_index]

        return final_dir

    def latitude_string(self):
        """
        Create a readable string of the current latitude data
        :return: string
        """
        if self.coord_format == 'dd':
            formatted_latitude = self.latitude
            lat_string = str(formatted_latitude[0]) + '° ' +
str(self._latitude[2])
        elif self.coord_format == 'dms':
            formatted_latitude = self.latitude
            lat_string = str(formatted_latitude[0]) + '° ' +
str(formatted_latitude[1]) + "' " + str(formatted_latitude[2]) + '" ' +
str(formatted_latitude[3])
        else:
            lat_string = str(self._latitude[0]) + '° ' +
```

```
str(self._latitude[1]) + "' " + str(self._latitude[2])
        return lat_string

    def longitude_string(self):
        """
        Create a readable string of the current longitude data
        :return: string
        """
        if self.coord_format == 'dd':
            formatted_longitude = self.longitude
            lon_string = str(formatted_longitude[0]) + '° ' +
str(self._longitude[2])
        elif self.coord_format == 'dms':
            formatted_longitude = self.longitude
            lon_string = str(formatted_longitude[0]) + '° ' +
str(formatted_longitude[1]) + "' " + str(formatted_longitude[2]) + '" ' +
str(formatted_longitude[3])
        else:
            lon_string = str(self._longitude[0]) + '° ' +
str(self._longitude[1]) + "' " + str(self._longitude[2])
        return lon_string

    def speed_string(self, unit='kph'):
        """
        Creates a readable string of the current speed data in one of three
units
        :param unit: string of 'kph','mph, or 'knot'
        :return:
        """
        if unit == 'mph':
            speed_string = str(self.speed[1]) + ' mph'

        elif unit == 'knot':
            if self.speed[0] == 1:
                unit_str = ' knot'
            else:
                unit_str = ' knots'
            speed_string = str(self.speed[0]) + unit_str

        else:
            speed_string = str(self.speed[2]) + ' km/h'

        return speed_string

    def date_string(self, formatting='s_mdy', century='20'):
        """
        Creates a readable string of the current date.
        Can select between long format: Januray 1st, 2014
        or two short formats:
        11/01/2014 (MM/DD/YYYY)
        01/11/2014 (DD/MM/YYYY)
        :param formatting: string 's_mdy', 's_dmy', or 'long'
        :param century: int delineating the century the GPS data is from (19
for 19XX, 20 for 20XX)
```

```
        :return: date_string  string with long or short format date
        """

        # Long Format Januray 1st, 2014
        if formatting == 'long':
            # Retrieve Month string from private set
            month = self.__MONTHS[self.date[1] - 1]

            # Determine Date Suffix
            if self.date[0] in (1, 21, 31):
                suffix = 'st'
            elif self.date[0] in (2, 22):
                suffix = 'nd'
            elif self.date[0] == (3, 23):
                suffix = 'rd'
            else:
                suffix = 'th'

            day = str(self.date[0]) + suffix  # Create Day String

            year = century + str(self.date[2])  # Create Year String

            date_string = month + ' ' + day + ', ' + year  # Put it all
together

        else:
            # Add leading zeros to day string if necessary
            if self.date[0] < 10:
                day = '0' + str(self.date[0])
            else:
                day = str(self.date[0])

            # Add leading zeros to month string if necessary
            if self.date[1] < 10:
                month = '0' + str(self.date[1])
            else:
                month = str(self.date[1])

            # Add leading zeros to year string if necessary
            if self.date[2] < 10:
                year = '0' + str(self.date[2])
            else:
                year = str(self.date[2])

            # Build final string based on desired formatting
            if formatting == 's_dmy':
                date_string = day + '/' + month + '/' + year

            else:  # Default date format
                date_string = month + '/' + day + '/' + year

        return date_string

    # All the currently supported NMEA sentences
```

```
    supported_sentences = {'GPRMC': gprmc, 'GLRMC': gprmc,
                           'GPGGA': gpgga, 'GLGGA': gpgga,
                           'GPVTG': gpvtg, 'GLVTG': gpvtg,
                           'GPGSA': gpgsa, 'GLGSA': gpgsa,
                           'GPGSV': gpgsv, 'GLGSV': gpgsv,
                           'GPGLL': gpgll, 'GLGLL': gpgll,
                           'GNGGA': gpgga, 'GNRMC': gprmc,
                           'GNVTG': gpvtg, 'GNGLL': gpgll,
                           'GNGSA': gpgsa,
                          }

if __name__ == "__main__":
    pass
```