Universidad de la República
Facultad de Ingeniería

PARRiOT: Implementación de RTK Sobre Red LoRa/LoRaWAN

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Florencia Montaldo, Federico Sierra, Irene Tolosa

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Dr. Germán Capdehourat ..... Universidad de la República
Dr. Federico La Rocca ..... Universidad de la República

Tribunal

Prof. Alicia Fernández ..... Universidad de la República
Dra. Claudina Rattaro ..... Universidad de la República
Dr. Leonardo Steinfeld ..... Universidad de la República

Montevideo
martes 19 noviembre, 2019

PARRiOT: Implementación de RTK Sobre Red LoRa/LoRaWAN, Florencia Montaldo, Federico Sierra, Irene Tolosa.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 148 páginas.
Compilada el jueves 16 enero, 2020.
http://iie.fing.edu.uy/

Kalman, todo está en Kalman.

Jorge Drexler ft. PARRIOT

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Agradecemos a nuestros tutores Germán y Federico por el acompañamiento continuo y el asesoramiento a lo largo de este proyecto. A nuestras familias y amigos por el apoyo en todo momento y a nuestros compañeros de El Nido por recibirnos en el techo con tanto cariño. A la empresa Teliot por confiar en nosotros y por los materiales brindados para la realización del proyecto. A Gabriel Gómez por sus prontas respuestas.

Esta página ha sido intencionalmente dejada en blanco.

A la familia y amigos que nos bancaron en este tiempo.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

La creciente automatización de los procesos productivos y tareas cotidianas hace que se vuelva una necesidad encontrar soluciones para el problema de la localización de objetos (como máquinas o incluso seres vivos). Es en este marco que surgen soluciones basadas en el mapeo del espacio mediante distintos tipos de sensores, así como también aquellas basadas en la navegación por satélite. En este proyecto se hará foco en este último tipo de solución. Dadas las características y restricciones del sistema planteado, la principal de ellas la necesidad de tener línea vista con los satélites, es que se piensa en aplicaciones agrícolas como ser el riego de precisión.

Cualquier tipo de posicionamiento vía satelital se basa en inferir la ubicación de un objeto en el globo terrestre a partir de las distancias calculadas a varios satélites. La distancia se calcula midiendo el tiempo de viaje de la señal desde un satélite a un receptor. El método básico para la ubicación vía GNSS (Global Navigation Satellite System) tiene una precisión de entre 5 y 10 metros, que resulta insuficiente para aplicaciones de precisión como ser el movimiento autónomo de robots. Existen variadas técnicas, con distintos rendimientos, para mejorar la precisión de este tipo de solución, una de ellas RTK (Real-Time Kinematics).

La técnica RTK logra eliminar las mayores fuentes de error, siendo éstas las debidas al defasaje de los relojes de satélite y receptor y los retrasos introducidos por la refracción de las señales en la atmósfera, logrando una precisión de unos pocos centímetros.

Para implementar una solución utilizando la técnica RTK se requiere, además del receptor propio del objeto que se quiere ubicar, una estación base de referencia que le transmitirá información adicional. Se crearon entonces dos prototipos, uno de nodo y otro de estación base, que se comunican entre sí mediante un enlace de radiofrecuencia con modulación LoRa. Así mismo se integró este sistema a una red LoRa/LoRaWAN con una aplicación final capaz de desplegar en un mapa las coordenadas calculadas.

Se logró desarrollar los prototipos para la solución RTK con un costo menor a 20 % de las opciones que ofrece el mercado y se los integró a un sistema cuyo desempeño se encuentra en el entorno de los 2 cm de precisión^{o}.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

1.1. Contexto y Motivación

El constante crecimiento en la industria, agricultura y ganadería, ha exigido la creación de nuevas tecnologías y técnicas de comunicación y localización, que permitan llevar adelante procesos de forma autónoma o remota. Una de las áreas con mayor potencial para el crecimiento es la del desarrollo de nuevas tecnologías capaces de comunicar objetos a Internet. Surge así el concepto de Internet de las cosas (IoT por sus siglas en inglés) [1] y junto a él algunas tecnologías como las redes LPWAN (Low Power Wide Area Network) [2], cuya principal característica de bajo consumo las hace ideales para conectar dispositivos remotos.

Para el problema de la localización, o incluso el trazado de rutas para el movimiento autónomo de robots, surgen distintas soluciones más o menos convenientes para distintos tipos de aplicaciones. Desde soluciones basadas en el mapeo del espacio mediante distintos tipos de sensores, hasta aquellas basadas en la navegación por satélite. En este proyecto se busca desarrollar un prototipo de dispositivo capaz de calcular su ubicación haciendo únicamente uso de señales satelitales con una precisión mejor a la alcanzada con los métodos estándar de posicionamiento satelital. A su vez se busca integrarlo a una red LoRa/LoRaWAN [3] [4] donde pueda reportar su posición.

El método básico para la ubicación vía GNSS (Global Navigation Satellite System) [5] tiene una precisión de entre 5 y 10 metros, lo que resulta para algunas aplicaciones, como la localización de objetos, robots o incluso plantines, insuficiente. En estos casos se requiere de técnicas de aumento de la presición, entre ellas se encuentran DGPS (Differencial Global Positioning System) [6], SBAS (Satellite-Based Augmentation System) [7], PPP (Precise Point Positioning) [8] y RTK (Real-Time Kinematics) [9]. Se trabajará con esta última en el desarrollo de este proyecto.

El posicionamiento vía satelital se basa en deducir la ubicación de un objeto en función de la distancia del mismo a al menos cuatro satélites, de esta forma queda determinada inequívocamente la ubicación en el globo terrestre. La distancia se calcula midiendo el tiempo de viaje de la señal desde un satélite a un receptor y

luego mediante trilateración se calcula la posición.

En particular, la técnica RTK se basa en la utilización de la fase de la portadora de las señales transmitidas por los satélites que forman parte de las constelaciones globales, como por ejemplo GPS (Global Positioning System) [10] y GLONASS (Global'naya Navigatsionnaya Sputnikovaya Sistema) [11]. Para la implementación de dicha técnica se requiere de dos elementos: el dispositivo móvil (nodo) al que se quiere ubicar y un dispositivo de referencia (estación base). El último le envía información adicional de forma constante al elemento móvil, el cual utilizando RTK logra eliminar las mayores fuentes de error, siendo éstas las debidas al defasaje de los relojes de satélite y receptor y los retrasos introducidos por la refracción de las señales en la atmósfera, obteniendo de esta forma una precisión de centímetros.

Las estaciones base transmiten en modo broadcast, por lo que una misma estación puede ser utilizada por varios nodos. La única restricción que se impone es que la distancia ente base y nodo no sea mayor a 10 km, límite para considerar que la influencia de la atmósfera en las señales recibidas es igual para ambos receptores. En Uruguay, el Instituto Geográfico Militar tiene 24 estaciones repartidas en el territorio, una de ellas en la antártida, además de otras 3 en territorio argentino próximos a la frontera uruguaya, todas ellas disponibles para uso civil [12]. A pesar de contar con estas bases disponibles, se optó por crear dos prototipos, uno de nodo y otro de estación base.

Dado que el objetivo es poder ubicar un nodo, es de esperar que el mismo tenga libertad de movimiento con respecto a la base, lo que lleva a pensar en una comunicación inalámbrica con un alcance del orden de kilómetros. Es por estos motivos que se optó por LoRa para implementar esta comunicación. A su vez estos dispositivos son parte de una red LoRaWAN que tiene como aplicación final una plataforma donde el usuario pueda ver la posición calculada.

Las redes LoRa/LoRaWAN pertenecen a las tecnologías LPWAN, de bajo consumo y largo alcance. La modulación utilizada denominada LoRa, basada en CSS (Chirp Spread Spectrum) [13], modula la señal con un pulso que realiza un barrido en frecuencia, generando mayor robustez a las interferencias.

Una red LoRa/LoRaWAN consta de tres elementos básicos, un nodo, un gateway y un servidor. El nodo es el dispositivo final que quiere reportar información al servidor. El gateway hace de pasarela enviando al servidor la información recibida desde el nodo. Finalmente en el servidor se obtienen y procesan los datos del nodo. El enlace gateway-nodo es a través de la modulación LoRa y el gateway envía los datos a través de UDP/IP al servidor.

En el diagrama de la Figura 1.1 se puede observar el sistema creado que incluye los elementos necesarios para realizar la técnica RTK (nodo y estación base) y la red LoRaWAN (servidor, gateway y nodo) necesaria para reportar a un servidor las coordenadas obtenidas.

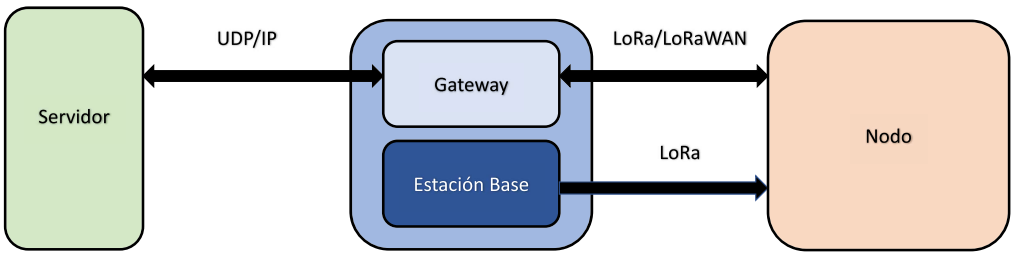


Figura 1.1: Diagrama del sistema a implementar

1.2. Objetivo del Proyecto

El objetivo de este proyecto es desarrollar una solución de localización de alta precisión de bajo costo basada en RTK. Se busca lograr la comunicación entre el nodo y la base de referencia utilizando LoRa. Finalmente se pretende implementar una red LoRa/LoRaWAN, para obtener un sistema que sea capaz de devolver las coordenadas calculadas a un servidor.

1.3. Estructura del Documento

El resto de esta documentación consta de los siguientes capítulos:

- **Capítulo 2**: Contiene una introducción a los sistemas globales de navegación por satélite y se presenta la técnica RTK, técnica que será fundamental para el desarrollo del proyecto.

- **Capítulo 3**: Se desarrolla sobre las redes LPWAN, luego se profundiza en particular en LoRa/LoRaWAN y el tipo de modulación LoRa. Una de estas redes es implementada para el reporte de datos del nodo a un servidor. La modulación LoRa es utilizada para el envío de datos entre estación base y nodo.

- **Capítulo 4**: Se exponen las plataformas de trabajo utilizadas tanto de hardware como software y se explica el por qué de la elección de los mismos.

- **Capítulo 5**: Se describe la implementación de una red LoRaWAN y de los elementos que forman parte de la misma. Se explican las modificaciones realizadas en las distintas bibliotecas utilizadas y en los archivos de configuración que intervienen.
- **Capítulo 6**: Contiene el proceso realizado para obtener una solución al problema de la ubicación. Se presentan las configuraciones utilizadas y el desarrollo de la comunicación entre estación base y nodo.
- **Capítulo 7**: Se expone la integración total del sistema, es decir el reporte de las coordenadas, calculadas con RTK, en un servidor a través de la red LoRa/LoRaWAN creada.
- **Capítulo 8**: Se realiza un estudio de los resultados obtenidos. Se evalúa el desempeño de la biblioteca utilizada para el desarrollo de la técnica RTK y del sistema creado.
- **Capítulo 9**: En él se desarrollan las conclusiones finales del proyecto y posibles trabajos a futuro.

Capítulo 2

GNSS y RTK

Con el fin de darle un marco de índole más teórico al proyecto que se presenta en este documento, es que se introducen a continuación algunos conceptos básicos referidos a la navegación por satélite. Se profundiza en aspectos específicos pertinentes al proyecto, como la técnica RTK (Real-Time Kinematics) utilizada para mejorar la precisión.

2.1. Sistema Global de Navegación por Satélite

Un sistema global de navegación por satélite (GNSS) es una constelación de satélites orbitando el planeta, cuyo fin es la localización, que asegura una cobertura global en todo momento. Las técnicas de posicionamiento satelital se basan en calcular la distancia entre el observador y al menos cuatro satélites, para luego mediante trilateración determinar la posición en el globo del receptor.

Ejemplos de sistemas GNSS son GPS de Estados Unidos, GLONASS de Rusia y Galileo [14] de la Unión Europea. En este proyecto se trabajará con señales procedentes de satélites pertenecientes a las constelaciones GPS y GLONASS. Al ser las más antiguas, son robustas en su funcionamiento y se encuentra mucha información disponible.

Mencionaremos a continuación algunos conceptos necesarios para comprender mejor el funcionamiento de los distintos sistemas.

- **Sistema Horario**
  Debido a que para las aplicaciones basadas en señales de GNSS es de particular importancia la sincronización entre los satélites, cada sistema tiene su propio reloj bajo el que está coordinada toda la constelación.

- **Efemérides**
  Es información de la órbita esperada para cada satélite, es decir, la posición del satélite para cada momento de tiempo dado.

- **Almanaque**
  Es información general del sistema satelital completo. Una aproximación

de las órbitas de los satélites, el estado de cada uno de ellos, un modelo ionosférico e información del sistema horario.

En las secciones 2.1.1 y 2.1.2 se detallan algunas características de las constelaciones de GPS y GLONASS, en la 2.2 se detallan algunos métodos para el cálculo de la posición.

2.1.1. GPS

GPS es un sistema GNSS desarrollado por Estados Unidos cuyos satélites transmiten a las frecuencias 1575,42 MHz (señal L1), 1227,60 MHz (señal L2), 1381,05 MHz (señal L3) y 1176,45 MHz (señal L5). En este proyecto se trabajará únicamente con la señal L1, debido a que de otro modo debería preverse el uso de antenas y receptores especiales capaces de trabajar con varias señales simultáneamente.

En la L1 se transmite un mensaje de navegación continuo a 50 bps. Cada transmisión dura 30 segundos y se requieren 12,5 minutos para recibir un mensaje completo. Cada transmisión comienza precisamente al minuto o medio minuto según el reloj atómico de cada satélite. Cada transmisión es un marco y cada marco se divide en cinco submarcos. En el submarco 1 se transmite el número y hora de la semana e información respecto a la salud del satélite. En los submarcos 2 y 3 se tramite la efemérides y en los 4 y 5 el almanaque e información para la corrección de errores. El almanaque se transmite por partes y se requiere de 25 transmisiones (marcos) para completarlo.

Todos los satélites de la constelación transmiten a la misma frecuencia y se utiliza modulación por multiplexación de código para distinguir las señales de los distintos satélites. La técnica conocida como CDMA (acceso múltiple por división de código, por sus siglas en inglés) se basa en codificar los bits 1 con una secuencia de chip y los 0 con la secuencia opuesta (Figura 2.1). Una secuencia de chip es una secuencia binaria de mucho mayor frecuencia que la tasa de transferencia de bits del mensaje.

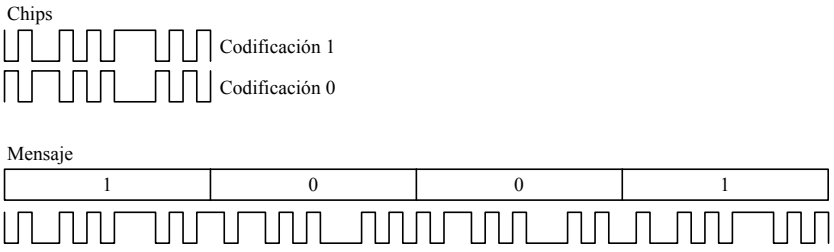


Figura 2.1: Representación de bits con chips.

Cada satélite es asignado una secuencia de chip, a la que se le llama código C/A (Coarse Acquisition) y con la que se codificará el mensaje a transmitir. Estos códigos son todos ortogonales entre sí, lo que implica que al momento de hacer la decodificación el mensaje se anulará para todos los códigos a excepción del correcto.

El receptor será entonces capaz de decodificar los mensajes satelitales con los respectivos códigos C/A, que son accesibles al público general. Conociendo el almanaque, el receptor conoce qué satélites tiene al alcance y es capaz de decodificar los correspondientes mensajes. En caso de no contar aún con la información del almanaque, se ingresa en un modo de búsqueda hasta que se logra la conexión con algún satélite.

2.1.2. GLONASS

Los satélites pertenecientes a esta constelación transmiten originalmente dos señales, L1 (1598,0625 - 1609,3125 MHz) y L2 (1242.9375 - 1251.6875 MHz) multiplexadas en frecuencia. Nuevos satélites que han sido incorporados a la constelación transmiten además señales que utilizan la misma técnica que GPS (multiplexación por código).

Para los satélites de GLONASS también se trabajará con la señal L1, por la misma razón ya expuesta para GPS. En esta señal se transmite un mensaje de navegación continuo a 50 bps y el cual tiene una estructura similar al de GPS, en tanto está dividido en marcos y submarcos. Un mensaje completo consta de 5 marcos y demora 2,5 minutos en transmitirse completamente. Cada marco dura 30 segundos y se compone de 12 submarcos, en los primeros 4 se envía la efemérides, reloj, salud del satélite e información de la frecuencia portadora. En el resto una porción del almanaque, que se completa al recibir los 5 marcos.

Para la transmisión de la señal L1 se utiliza la técnica FDMA, lo que implica que cada satélite transmite a distinta frecuencia. Para disminuir el ancho de banda necesario para transmitir señales de todos los satélites de la constelación, dos satélites antipodales que se encuentren al mismo nivel de elevación comparten la frecuencia de transmisión. En la Figura 2.2 se ilustra cómo dos satélites en estas condiciones nunca podrán ser percibidos simultáneamente por un receptor en la superficie terrestre, evitando de esta forma interferencia entre las señales.

La multiplexación en frecuencia permite que todos los satélites de la constelación GLONASS transmitan codificadas con el mismo código C/A.

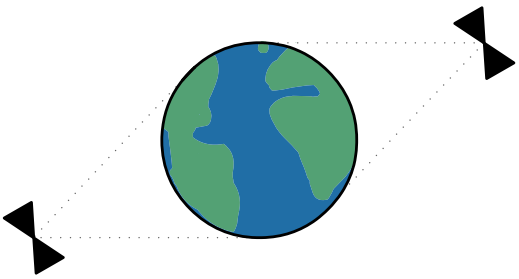


Figura 2.2: Satélites antipodales.

2.2. Posicionamiento Vía Satelital

2.2.1. Tipos de Posicionamiento

La idea general sobre la que se basa cualquier técnica de posicionamiento vía señales satelitales, es la de calcular el tiempo de propagación de la onda para deducir así la distancia entre satélite y receptor. Una vez que se conoce la distancia a al menos cuatro satélites, queda determinado inequívocamente el punto en que se encuentra el receptor. Existen dos categorías de posicionamiento: por código o por fase.

Por código:

En este tipo de posicionamiento, el receptor intenta decodificar las señales de los satélites que tiene a la vista, generando los correspondientes códigos C/A. En base al tiempo que le lleva sincronizarse con la señal es que se calcula la posición. La sincronización sucede cuando la correlación entre las señales (generada y recibida) es máxima.

Satélite y receptor generan simultáneamente la secuencia de pulsos, cada uno según su propia base temporal, por lo tanto el tiempo que demora la señal en salir del satélite y llegar al receptor es el defasaje entre las señales y es el que se mide sincronizando los códigos. Con el tiempo de viaje de la señal se calcula fácilmente la distancia del receptor al satélite como $R_{r,s} = c \times \Delta t$, con c velocidad de la luz. En la Figura 2.3 se ilustra la situación anterior.

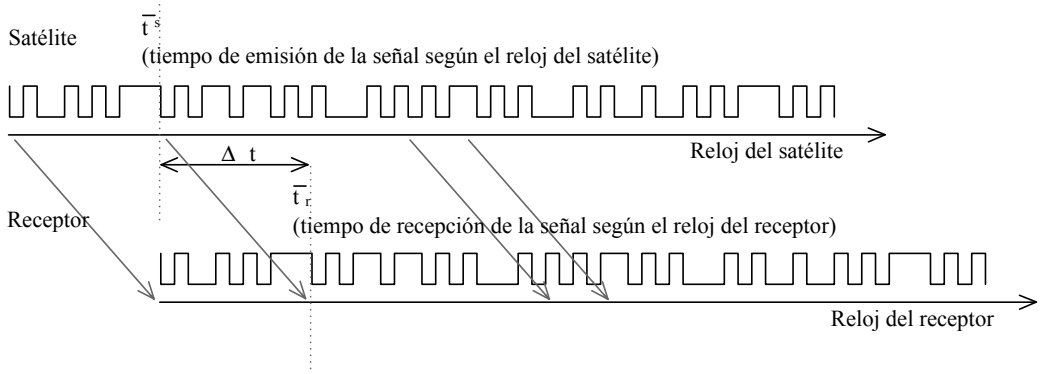


Figura 2.3: Defasaje de la señal satelital entre satélite y receptor.

Este tipo de posicionamiento tiene una precisión de entre 10 y 15 metros [15] y no es ambiguo. Esto quiere decir que si bien las soluciones pueden ser dispersas dentro del margen mencionado, se garantiza que la posición real está en ese entorno.

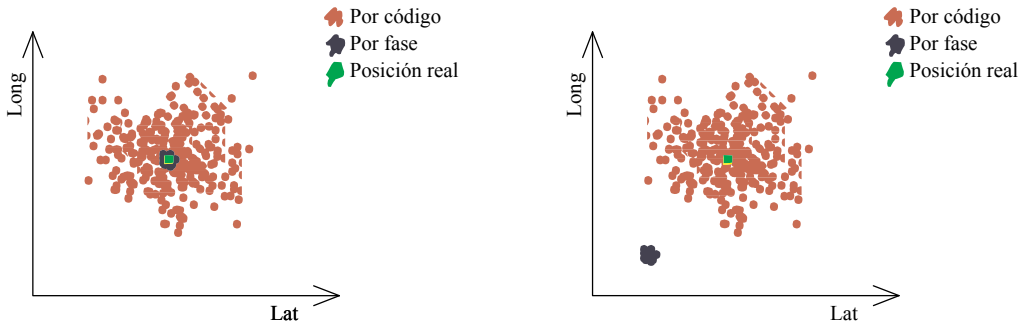
Por fase:

El posicionamiento por fase se basa en calcular la distancia al satélite en términos de longitudes de onda. La distancia resulta en un múltiplo entero de longitudes de onda más un defasaje. A este número entero se le llama ambigüedad y resolverla

es la clave para una buena estimación de la posición. La distancia del receptor al satélite puede calcularse como $n \times \lambda + \phi$, con n entero, λ longitud de onda y ϕ defasaje.

A diferencia del tipo anterior, el posicionamiento por fase se caracteriza por tener una precisión centimétrica, pero es ambiguo. Significa que las medidas no van a ser dispersas pero pueden tener errores grandes.

En la Figura 2.4 se ilustran las características de ambos tipos de posicionamiento. Se hace notar la dispersión de las medidas utilizando el método por código, mientras que en el caso del método por fase no hay dispersión pero el resultado puede ser incorrecto.



(a) Resultados correctos.

(b) Resultado por fase incorrecto.

Figura 2.4: Características de la solución para los distintos métodos.

2.2.2. Fuentes de Error en las Medidas

La principal fuente de error en las medidas es la refracción de las ondas en la atmósfera, especialmente en la ionósfera. La desviación de la señal hace que el tiempo de viaje desde el satélite al receptor se vea alterado, por lo que la distancia calculada deja de ser la geométrica y pasa a ser la del trayecto real de la señal. Otro error que se presenta más directamente es un posible error en los relojes del satélite, lo que hace que el tiempo medido de viaje parta de una base errónea. Recordando que la luz viaja 30 centímetros en un nanosegundo, pequeños defasajes pueden generar grandes errores. También son fuentes de error desviaciones en las órbitas de los satélites y en menor grado el ruido en el receptor y multicaminos de las señales.

Debido a los errores, la medida que finalmente se obtiene es una pseudo-distancia entre receptor y satélite. Cuanto menos errores se tenga, más cercana será la distancia medida a la distancia geométrica, siendo esta última la que idealmente se quiere obtener. En la Tabla 2.1 se pueden ver algunas de las fuentes de error más importantes y el orden en que afectan la medida.

Fuente	Error
Reloj del satélite	± 2 m
Errores en las órbitas	± 2.5 m
Ionósfera	± 5 m
Tropósfera	± 0.5 m
Ruido en el receptor	± 0.3 m
Multicaminos	± 1 m

Tabla 2.1: Fuentes de errores en las medidas.

2.3. La Técnica RTK

RTK es una técnica de posicionamiento por fase de la portadora. Un nodo es capaz de calcular su posición con ayuda de una estación fija de referencia (de ahora en adelante, base) que le envía información de sus propias medidas de las señales satelitales. Receptor y base no deben encontrarse a más de 10 km de distancia, que es el límite para considerar que la influencia de la ionósfera es igual sobre ambos receptores. Se describe a continuación el argumento matemático que respalda esta técnica.

Se referirá por "número de ciclos" entre dos puntos a la cantidad de longitudes de onda entre ellos, midiéndolo en términos de fase. Un ciclo son 2π rad y corresponden a una longitud de onda. Sea ϕ_r^s el número se ciclos entre el receptor r y el satélite s, $\phi_r(\bar{t_r})$ la fase en el receptor en el momento de recepción (medido en el reloj del receptor), $\phi^s(\bar{t^s})$ la fase en el satélite en el momento de transmisión (medido en el reloj del satélite), N_r^s un número entero de ciclos y ϵ_ϕ un error, éstas se relacionan de la siguiente manera:

$$\phi_r^s = \phi_r(\bar{t_r}) - \phi^s(\bar{t^s}) + N_r^s + \epsilon_\phi \ . \tag{2.1}$$

Pasando a un sistema de tiempo absoluto de referencia t, y considerando que $f \times \Delta t = num.\, ciclos$ con f la frecuencia portadora, se pueden reescribir los términos $\phi_r(\bar{t_r})$ y $\phi^s(\bar{t^s})$ como:

$$\phi_r(\bar{t_r}) = f \times \bar{t_r} = f \times (t_r + dt_r(t_r)) \ , \tag{2.2}$$
$$\phi^s(\bar{t^s}) = f \times \bar{t^s} = f \times (t^s + dT^s(t^s)) \ , \tag{2.3}$$

donde se elige $t_0 = 0$, $\phi_0 = 0$ por simplicidad y se denota $dt_r(t_r)$ al defasaje del reloj del receptor respecto del sistema de referencia en el tiempo t_r de recepción de la señal, y $dT^s(t^s)$ al defasaje del reloj del satélite respecto al sistema de referencia en el tiempo t^s de emisión de la señal.

Reescribiendo (2.1) en función de (2.2) y (2.3), y tomando $\lambda = f \times c$ con λ longitud de onda de la señal:

$$\phi_r^s = \frac{c}{\lambda} \times (t_r - t^s) + \frac{c}{\lambda} \times (dt_r(t_r) - dT^s(t^s)) + N_r^s + \epsilon_\phi \ . \tag{2.4}$$

Tomando en cuenta que la pseudo-distancia (se recuerda que no es la distancia geométrica) entre satélite y receptor se puede calcular como $\Phi_r^s = \phi_r^s \times \lambda$:

$$\Phi_r^s = c \times (t_r - t^s) + c \times (dt_r(t_r) - dT^s(t^s)) + \lambda \times N_r^s + \lambda \times \epsilon_\phi \tag{2.5}$$

$$\text{con } c \times (t_r - t^s) = \rho_r^s - I_r^s + T_r^s \ . \tag{2.6}$$

En la ecuación (2.6) se muestra que el tiempo de viaje de la señal multiplicado por la velocidad de la luz, representa la distancia geométrica ρ_r^s entre receptor y satélite más ciertas desviaciones causadas por la ionósfera I_r^s y tropósfera T_r^s.

Juntando lo anterior:

$$\Phi_r^s = \rho_r^s + c.(dt_r(t_r) - dT^s(t^s)) - I_r^s + T_r^s + \lambda N_r^s + \epsilon_\Phi \ . \tag{2.7}$$

Para eliminar los errores introducidos por la atmósfera y por la desviación de los relojes (fuentes predominantes de error), se toman las medidas de dos receptores r y b para dos satélites s_1 y s_2 y se realiza la diferencia en diferencias:

$$\Phi_{r,b}^{s_1,s_2} = (\Phi_b^{s_2} - \Phi_r^{s_2}) - (\Phi_b^{s_1} - \Phi_r^{s_1}) \ . \tag{2.8}$$

Sustituyendo (2.7) en (2.8) se obtiene (2.9), donde se supuso que la distancia entre receptores es menor a 10 km, para poder asumir los errores introducidos por la atmósfera iguales:

$$\Phi_{r,b}^{s_1,s_2} = \rho_n^{s_1} - \rho_b^{s_1} - \rho_n^{s_2} + \rho_b^{s_2} + \lambda.N + \epsilon \ . \tag{2.9}$$

RTK parte de la base que la posición del segundo receptor (b, base) es conocida, por lo que en la ecuación anterior los términos ρ_b^{s1} y ρ_b^{s2} son conocidos. RTK logra de esta manera eliminar las mayores fuentes de error en las medidas.

Para estimar la posición suele utilizarse el filtro extendido de Kalman para sistemas no lineales (EKF, por sus siglas en inglés). Usando el EKF, un vector de estados x y su matriz de covarianza M, pueden estimarse con un vector de entradas y_k en el tiempo t_k, de acuerdo a las siguientes ecuaciones:

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k[y_k - h(\hat{x}_k(-))] \ , \tag{2.10}$$

$$M_k(+) = [I - K_k H(\hat{x}_k(-))]M_k(-) \ , \tag{2.11}$$

$$K_k = M_k(-)H(\hat{x}_k(-))[H(\hat{x}_k(-))M_k(-)H(\hat{x}_k(-))^T + R_k]^{-1} \ , \tag{2.12}$$

donde $\hat{x}_k$ y M_k son los estimados en el tiempo t_k y $(+)$ y $(-)$ indican momentos posteriores o anteriores a la actualización del filtro de Kalman. $h(x)$ representa un vector modelo de medidas, $H(x)$ su matriz de derivadas parciales y R_k la matriz de covarianza de los errores en las medidas, por más detalles respecto a estos parámetros referirse a [16].

El vector de estados a estimar corresponde a $x = (r_r^T, v_r^T, B^T)$ con r_r la posición del receptor, v_r la velocidad y $B = (N_{r,b}^{s_1}, N_{r,b}^{s_2}, ...N_{r,b}^{s_m})$, donde se mantuvo la suposición de $t_0 = 0$, $\phi_0 = 0$ y $N_{r,b}^s = N_b^s - N_r^s$. El vector de entradas corresponde a $y_k = (\Phi^T, P^T)$, con Φ y P según:

$$\Phi = (\Phi_{r,b}^{s_1,s_2}, \Phi_{r,b}^{s_1,s_3}, ...\Phi_{r,b}^{s_1,s_m})^T \ , \tag{2.13}$$

$$P = (P_{r,b}^{s_1,s_2}, P_{r,b}^{s_1,s_3}, ...P_{r,b}^{s_1,s_m})^T \ , \tag{2.14}$$

donde los elementos de Φ son como en (2.9) y los de P corresponden a la diferencia en diferencias para las medidas por código. Para las medidas por código el desarrollo de ecuaciones es similar al desarrollado anteriormente y se muestra sin entrar en detalles a continuación.

Sea P_r^s la pseudo-distancia medida utilizando la técnica por código, se cumple que:

$$P_r^s = c(\bar{t_r} - \bar{t^s}) \ . \tag{2.15}$$

Cambiando a un sistema temporal fijo de referencia:

$$P_r^s = c((t_r + dt_r(t_r)) - (t^s + dT^s(t^s))) + \epsilon_p \ . \tag{2.16}$$

Desarrollando:

$$P_r^s = \rho_r^s + c(dt_r(t_r) - dT^s(t^s)) + I_r^s + T_r^s + \epsilon_p \ . \tag{2.17}$$

Realizando la diferencia en diferencias:

$$P_{r,b}^{s_1,s_2} = \rho_n^{s_1} - \rho_b^{s_1} - \rho_n^{s_2} + \rho_b^{s_2} + \epsilon_p \ . \tag{2.18}$$

Mientras la solución del filtro no ha convergido, se la denomina *float*, luego se la denomina *fixed*.

A modo de resumen, la base debe enviarle al nodo (receptor r) sus medidas de fase de las señales satelitales y sus medidas de pseudo-distancia por código. El nodo, contando además con sus propias medidas, puede entonces calcular $\Phi_{r,b}^{s_1,s_2}$ y $P_{r,b}^{s_1,s_2}$, entradas para el EKF.

Capítulo 3

Redes LoRa/LoRaWAN

Este capítulo pretende ser una introducción a las redes LoRa/LoRaWAN. Se comienza presentando las redes LPWAN y luego se profundizará particularmente en las redes LoRa/LoRaWAN.

3.1. Redes LPWAN

Los avances tecnológicos llevaron a que el concepto de IoT tomara más fuerza y permitieron la conexión entre objetos a través de Internet, logrando controlar o recibir información de forma remota de todo tipo de dispositivos, maquinaria, seres vivos, etc. Se crearon nuevos protocolos de red que lograron conectar dispositivos o sensores entre ellos y a Internet de manera sencilla. Entre estos protocolos se encuentran las redes LPWAN.

Las redes LPWAN son típicamente para nodos de bajo consumo, implementadas para comunicaciones de largo alcance con una baja velocidad de envío de datos (pocos kbps). Este tipo de redes fueron diseñadas especialmente para aplicaciones en las que los dispositivos o sensores envían poca información y a largas distancias.

En la Figura 3.1 se presentan algunas de las tecnologías LPWAN y sus características principales. Entre ellas se encuentran LoRaWAN, SigFox [17], INGENU [18] y TELENSA [19]. Se profundizará particularmente en las redes LoRa/LoRaWAN ya que una de estas redes es implementada en este proyecto para el envío de datos de un nodo a un servidor.

3.2. Redes LoRa/LoRaWAN

En este apartado se presentan las redes LoRa/LoRaWAN que se encuentran dentro de las tecnologías LPWAN y son de las opciones elegidas a la hora de crear redes de sensores inalámbricos conectados a Internet. Permiten conectar una variedad de dispositivos, requieren de un bajo consumo y debido a la tecnología utilizada para la comunicación (LoRa) permiten un largo alcance. La velocidad de envío de datos en estas redes es como máximo de 50 kbps.

	SigFox	LoRaWAN	Ingenu	Telensa
Modulation	UNB DBPSK(UL), GFSK(DL)	CSS	RPMA-DSSS(UL), CDMA(DL)	UNB 2-FSK
Band	Sub-GHz ISM:EU (868MHz), US(902MHz)	Sub-GHz ISM:EU (433MHz 868MHz), US (915MHz), Asia (430MHz)	ISM 2.4GHz	Sub-GHz bands including ISM:EU (868MHz), US (915MHz), Asia (430MHz)
Data rate	100 bps(UL), 600 bps(DL)	0.3-37.5 kbps (LORa), 50 kbps (FSK)	78kbps (UL), 19.5 kbps(DL) [39]	62.5 bps(UL), 500 bps(DL)
Range	10 km (URBAN), 50 km (RURAL)	5 km(URBAN), 15 km (RURAL)	15 km (URBAN)	1 km (URBAN)
Num. of channels / orthogonal signals	360 channels	10 in EU, 64+8(UL) and 8(DL) in US plus multiple SFs	40 1MHz channels, up to 1200 signals per channel	multiple channels
Link symmetry	×	✓	×	×
Forward error correction	×	✓	✓	✓
MAC	unslotted ALOHA	unslotted ALOHA	CDMA-like	?
Topology	star	star of stars	star, tree	star
Adaptive Data Rate	×	✓	✓	×
Payload length	12B(UL), 8B(DL)	up to 250B (depends on SF & region)	10KB	?
Handover	end devices do not join a single base station	end devices do not join a single base station	✓	?
Authentication & encryption	encryption not supported	AES 128b	16B hash, AES 256b	?
Over the air updates	×	✓	✓	✓
SLA support	×	×	×	×
Localization	×	✓	×	×

Figura 3.1: Tecnologías de comunicación LPWAN para IoT [20].

En el diagrama de la Figura 3.2 se presenta la forma básica de una red LoRa/LoRaWAN, conformada por un nodo, un gateway y un servidor. Además se pueden observar las capas utilizadas en cada enlace. La comunicación entre el nodo y gateway es a través de la tecnología LoRa, tipo de modulación que se describirá más adelante. LoRaWAN es una especificiación creada y mantenida por la LoRa Alliance. Especifica el uso de los canales, el ancho de banda, el cifrado de los datos y la capa MAC [21] en el enlace gateway-nodo.

El nodo envía información a través de radiofrecuencia utilizando LoRa hacia el gateway. En el gateway, la especificación LoRaWAN arma el paquete con el payload y lo envía encapsulado sobre una red UDP/IP pudiendo utilizar diferentes medios de comunicación como Ethernet [22], 3G [23] y WiFi [24]. GWMP (Gateway Message Protocol) es el tipo de mensaje enviado por el gateway [25].

En síntesis, el gateway funciona como puente entre los dispositivos finales (nodos) y el servidor, encargándose de transformar paquetes de radiofrecuencia en paquetes UDP/IP y viceversa. El servidor recibe la información desde el gateway y tiene como responsabilidad el manejo de las tramas que recibe, la capa MAC LoRaWAN (del enlace nodo-gateway) y las transmisiones hacia el gateway. Además se encarga de administrar (definir, configurar) los nodos y gateways.

LoRaWAN utiliza la tecnología LoRa para el envío de datos entre gateway y nodo ya que la misma permite comunicaciones de largo alcance. A continuación se presenta dicha tecnología.

3.2.1. Modulación LoRa

LoRa es un tipo de modulación en radiofrecuencia patentado por Semtech [26] (quien fabrica los chips de radio o licencia a terceros), forma parte de la capa física del modelo OSI [27]. Las frecuencias utilizadas para transmitir se encuentran dentro de la banda ISM (Industrial, Scientific and Medical) [28], la misma es una banda de radiofrecuencia no comercial y está reservada para el uso de la industria

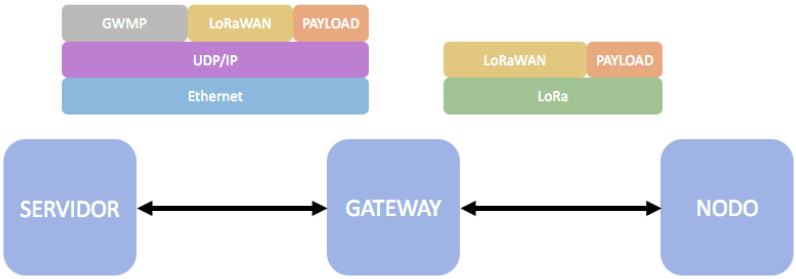


Figura 3.2: Diagrama de una red LoRa/LoRaWAN.

y medicina.

LoRa toma como base la técnica de espectro ensanchado CSS. Esta técnica de modulación emplea un pulso que barre todas las frecuencias para codificar la información (chirp). De esta forma se expande la señal espectral en todo el ancho de banda. Cada chirp representa un símbolo y su representación matemática en el tiempo es la siguiente:

$$x(t) = e^{(\varphi_0 + 2\pi(\frac{k}{2}t^2 + f_0 t))} \ , \tag{3.1}$$

siendo φ_0 la fase inicial, k la tasa de cambio de frecuencia y f_0 la frecuencia inicial.

En la Figura 3.3 se representan dos gráficos, el primero presenta el pulso base utilizado (chirp) y el segundo muestra la variación lineal de la frecuencia. Los parámetros utilizados son el tiempo de chirp (T_{chirp}), el ancho de banda (BW) que son las frecuencias que barre el chirp en el tiempo T_{chirp} y Spreading Factor (SF) que define el número de bits utilizados para codificar un símbolo (del 7 al 12). Existen 2^{SF} símbolos posibles y cada uno se envía en un pulso chirp, la tasa de transferencia de símbolo es entonces $r_s = \frac{1}{T_{chirp}}$ y por tanto la tasa de transferencia de bits es $r_b = \frac{SF}{T_{chirp}}$.

Observando la representación matemática del pulso chirp (3.1) se puede concluir que dado un ancho de banda BW la frecuencia varía entre $f_0 - \frac{BW}{2}$ y $f_0 + \frac{BW}{2}$ dentro de un tiempo T_{chirp}. El tiempo de chirp se define en función de los parámetros antes definidos como $T_{chirp} = \frac{2^{SF}}{BW}$. Este tiempo es dividido en 2^{SF} sub-intervalos de manera de poder representar cada símbolo, como se explica a continuación.

En la Figura 3.4 se puede observar un espectro con símbolos enviados, se verifica la variación lineal de la frecuencia realizada para el envío y se pueden visualizar cambios abruptos en la frecuencia. El momento en que éstos ocurren indica a qué sub-intervalo se pertenece (dentro de T_{chirp}), es decir se identifica qué símbolo se está mandando (de los 2^{SF} posibles).

LoRa cuenta con corrección de errores en la recepción de los datos, FEC (Co-

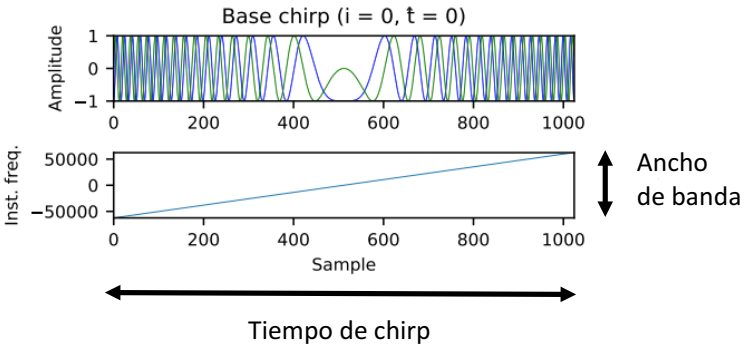


Figura 3.3: Base chirp [29].

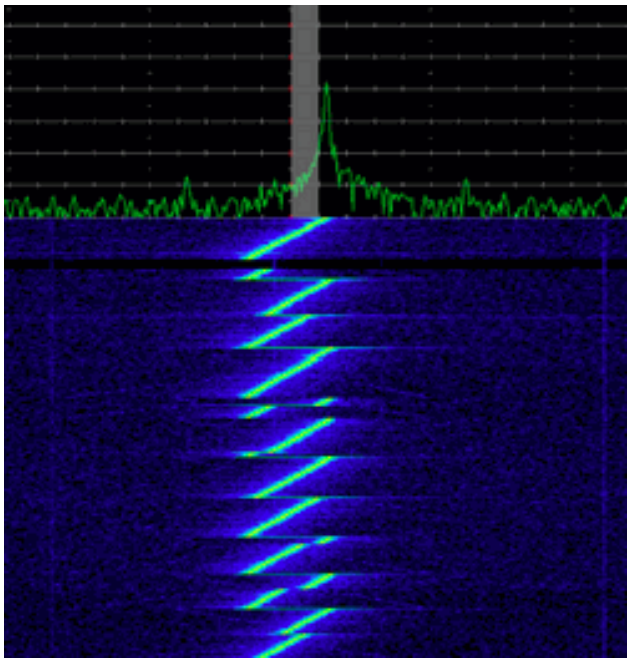


Figura 3.4: Modulación LoRa representada en espectro [30].

rrección de datos hacia adelante o Forward Error Correction) [31]. Este tipo de corrección de errores se encuentra en el receptor y no implica retransmisión de los datos. Se agregan bits en los datos enviados, los cuales ayudan a reconstruir el mensaje recibido si hubo alguna interferencia. Se define el coding rate (CR) que representa la porción de bits enviados que realmente tienen información (4/5, 4/6, 4/7 y 4/8). Por ejemplo si se elige 4/5, de 5 bits enviados, hay 4 bits que transportan información y hay uno agregado para la redundancia. Es decir que entonces la tasa de bits que contienen información es:

DataRate	SF	BW	bits/segundo
0	SF12	125 kHz	250
1	SF11	125 kHz	440
2	SF10	125 kHz	980
3	SF9	125 kHz	1760
4	SF8	125 kHz	3125
5	SF7	125 kHz	5470
6	SF8	500 kHz	12500
7	RFU	RFU	
8	SF12	500 kHz	980
9	SF11	500 kHz	1760
10	SF10	500 kHz	3900
11	SF9	500 kHz	7000
12	SF8	500 kHz	12500
13	SF7	500 kHz	21900
14..15	RFU	RFU	

Tabla 3.1: Datarate de LoRa [32].

$$r_b = SF \times CR \times (\frac{BW}{2^{SF}}) \tag{3.2}$$

En la tabla 3.1 se presenta la tasa de bps según el ancho de banda BW y el spreading factor SF elegidos. Vale aclarar que RFU significa reservado para futuros usos.

Finalmente, el paquete LoRa tiene la forma que se presenta en la Figura 3.5. Consta de un preámbulo que es una serie de bits necesarios para el sincronismo en tiempo y frecuencia entre el receptor y transmisor, un encabezado y payload, ambos seguidos por su correspondiente CRC (Cyclic Redundancy Check) [33] para asegurar la transmisión.

Figura 3.5: Paquete LoRa.

3.2.2. LoRaWAN

La especificación LoRaWAN se encarga de definir la capa MAC del modelo OSI en el enlace entre el gateway y el nodo (Figura 3.2). Como se expresó anteriormente, LoRaWAN especifica cómo gestionar el uso de los canales, el ancho de banda y el cifrado de los datos.

Respecto a las frecuencias LoRaWAN las organiza en sub-bandas y canales. Las sub-bandas están definidas según la región y se presentan a continuación:

- AS 923
- AU 915-928
- CN 470-510
- CN 779-787
- EU 433
- EU 863-870
- IN 865-867
- KR 920-923
- US 902-928
- RU 864-870

Según la especificación de parámetros regionales [32] la banda utilizada en Uruguay es AU 915-928. Por lo tanto las frecuencias con las que se trabaja se encuentran dentro de la banda 915 MHz y 928 MHz (esta banda se encuentra libre en Uruguay [34]). Dentro de la banda AU915-928 los canales están divididos de la siguiente forma:

- Upstream: 64 canales, del 0 al 63, comenzando en la frecuencia 915,2 MHz y terminando en 927,8 MHz, aumentando cada 200 kHz y con BW de 125 kHz. Utiliza el coding rate 4/5 y data rate 0 (Ver 3.1).
- Upstream: 8 canales, del 64 al 71, comenzando en la frecuencia 915,9 MHz y terminando en 927,1 MHz, aumentando cada 1,6 MHz y con BW de 500 kHz. Utiliza data rate 6 (Ver 3.1).
- Downstream: 8 canales, del 0 al 7, comenzando en la frecuencia 923,3 MHz y terminando en 927,5 MHz, aumentando cada 600 kHz y con BW de 500 kHz.

En LoRaWAN el nodo debe cambiar de canal de forma pseudoaleatoria para cada transmisión, haciendo que el sistema sea más robusto ante interferencias.

En la Figura 3.7 es posible ver las capas definidas por LoRaWAN en el enlace entre el gateway y nodo. Las clases de los nodos definidas en la capa MAC difieren en el consumo y en la latencia en el downlink [35] y se exponen a continuación.

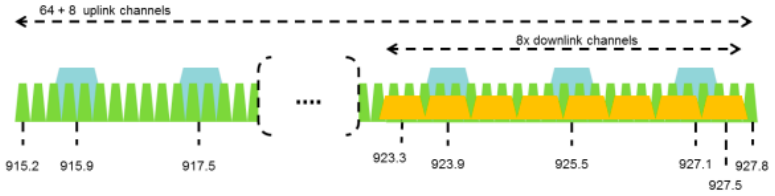


Figura 3.6: Canales de frecuencia en AU915-928 [32].

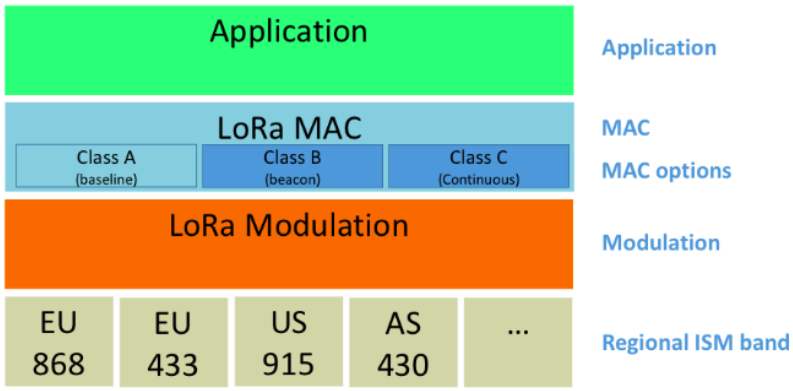


Figura 3.7: Capas definidas por LoRaWAN del enlace nodo-gateway [35].

- **Clase A**: Un nodo clase A permite comunicaciones bidireccionales donde luego de que el mismo envía información. Se pone en modo escucha en dos breves períodos de tiempo, $1s \pm 20\mu s$ después de la transmisión y luego de $1s \pm 20\mu s$ de la primera vez que se pone en modo escucha (Ver Figura 3.8). Esta clase de nodos es la que menos genera consumo y es el tipo más utilizado.
- **Clase B**: Los nodos se ponen en modo escucha luego de transmitir, al igual que los clase A, pero también lo hacen en horarios programados con el gateway. El consumo depende de los períodos de tiempo programados.
- **Clase C**: En este caso los nodos se encuentran en modo escucha en todo momento menos cuando están transmitiendo, por lo tanto es el modo en el que el consumo es mayor.

En LoRaWAN existen dos formas de asociarse a la red, ABP (Activation By Personalization) y OTAA (Over The Air Activation). En ambos modos es necesario ingresar información en los nodos.

- **ABP:** Esta modalidad tiene como ventaja que requiere menos tiempo para asociarse a la red debido a que el nodo ya cuenta con claves fijas asignadas. Es el modo más inseguro ya que las claves están en cada dispositivo y podrían ser robadas. Si el nodo quiere cambiarse de red necesita reconfigurarse para cambiar las claves.

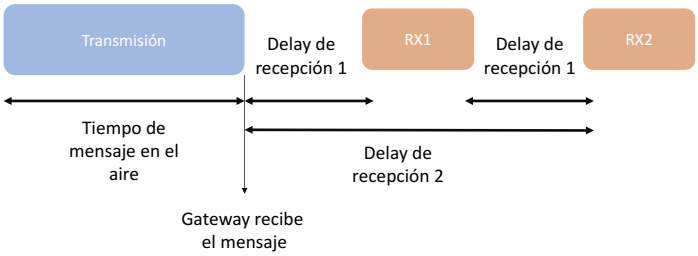


Figura 3.8: Nodos clase A.

En este tipo de conexión se requieren los siguientes parámetros:

DevAddress: Es una dirección lógica que identifica al nodo en la red, es análoga a una dirección IP en una red TCP/IP. Tiene un largo de 4 bytes.

NetworkSessionKey (NwkSKey): Utilizada por el nodo y el servidor para las distintas transmisiones y comprobación de la integridad de los mensajes. Tiene un largo de 16 bytes.

ApplicationSessionKey (AppSKey): Utilizada por el nodo y la aplicación para las distintas transmisiones y comprobación de la integridad de los mensajes. Tiene un largo de 16 bytes.

El DevAddress y las dos claves de sesión NwkSKey y AppSKey se guardan directamente en el nodo (y son únicas para cada nodo). Éste, al comenzar a asociarse, tiene toda la información que se requiere para participar de una red LoRaWAN específica. Para establecer la conexión el nodo le envía estos parámetros al gateway, si el mismo los valida la conexión queda establecida, en caso contrario el gateway rechaza la conexión.

- **OTAA:** Es la forma más segura de conectarse a una red LoRaWAN ya que existe un proceso para asociarse a la red durante el cual se negocian las claves de seguridad con el dispositivo. Los parámetros involucrados en esta conexión son los siguientes:

  DevEUI: Es un identificador de fábrica de cada dispositivo. Es análogo a una dirección MAC en una red TCP/IP. Tiene 8 bytes.

  AppEUI: Identifica al propietario de la aplicación a la que el dispositivo pertenece. Tiene 8 bytes.

  AppKey: Clave secreta de 16 bytes entre el nodo y la red. Utilizada para establecer las distintas claves de sesión (ApplicationSessionKey y Network-

SessionKey). El nodo puede reutilizar estas claves siempre y cuando no hayan sido expiradas por el servidor, si esto sucede debe realizar un nuevo inicio de sesión.

Previo al envío de datos el nodo debe asociarse a la red. Para esto transmite las claves de configuración al gateway y se pone en modo escucha. El gateway las reenvía al servidor, el cual las verifica. Si los parámetros son correctos el servidor le asigna una sesión temporal al nodo, en caso contrario, el inicio de sesión se rechaza.

En las Figuras 3.9 y 3.10 se pueden observar ambos modos. En el caso de ABP el nodo ya cuenta con las claves DevAddress, ApplicationSessionKey y NetworkSessionKey. En el caso OTAA, se le asignan dichas claves a partir de un "Join Accept" luego de un proceso de asociación en el cual el nodo debe enviar un mensaje "Join Request".

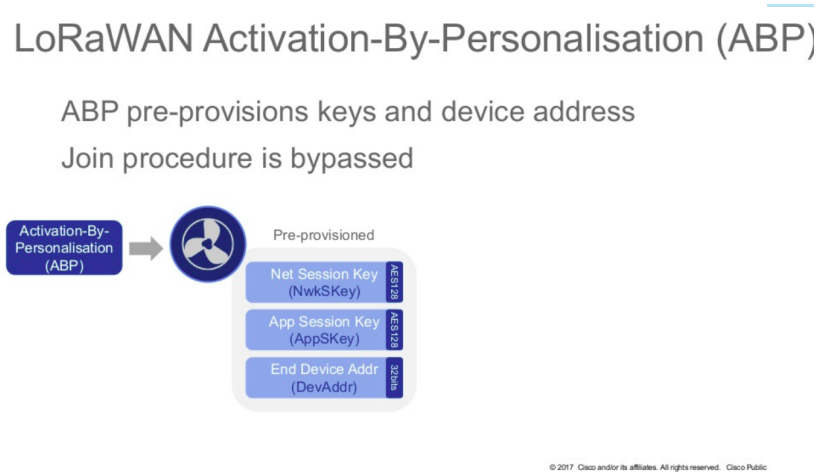


Figura 3.9: Modo ABP [35].

En síntesis, las redes LoRa/LoRaWAN son parte de las tecnologías LPWAN creadas para comunicaciones inalámbricas de largo alcance y bajo consumo. Para establecer una red LoRa/LoRaWAN es necesario contar con tres elementos fundamentales: nodo, gateway y servidor. Nodo y gateway se comunican mediante la modulación LoRa. El gateway funciona como puente entre nodo y servidor. El servidor se encarga de procesar los datos recibidos y puede contar con una aplicación capaz de gestionar los mismos. Una de estas redes será creada en este proyecto para la recepción de coordenadas del nodo en un servidor. Además se utilizará la modulación LoRa para el envío de datos de la estación base al nodo para permitir una solución RTK.

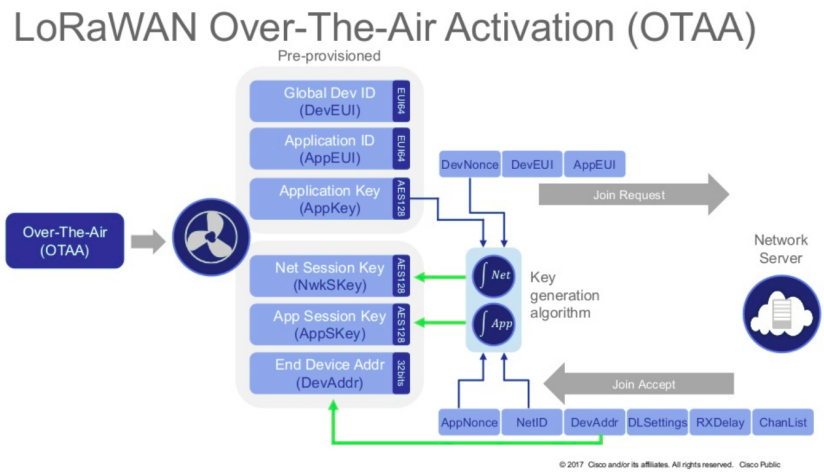


Figura 3.10: Modo OTAA [35].

Capítulo 4

Plataformas de Trabajo

En este capítulo se mencionarán las diferentes plataformas con las que se trabajó para lograr la totalidad del proyecto. Se podrá ver una breve descripción, en el caso de ser pertinente, tanto del hardware como del software empleado. Un diagrama del sistema completo implementado puede verse en la Figura 4.1.

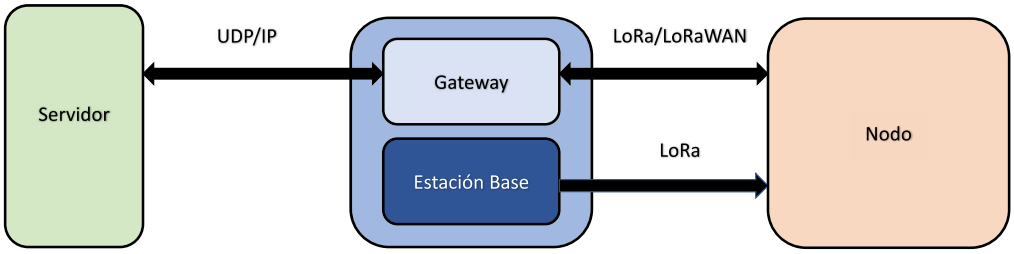


Figura 4.1: Diagrama del sistema a implementar

A modo de organizar este capítulo se describirá cada uno de los elementos de la solución por separado, tanto en hardware como en software. Se comenzará describiendo el servidor, luego el gateway y para finalizar el nodo.

4.1. Servidor

Un servidor, en general, es una aplicación capaz de atender peticiones de un cliente y enviarle una respuesta. Esta aplicación puede correrse en cualquier tipo de computadora. Hoy en día existe la posibilidad de tener un servidor físico propio o contratar espacio de memoria en un servidor a un tercero.

En este caso en particular el servidor LoRaWAN, representado a la izquierda en la Figura 4.1, se utilizó como receptor de los datos generados por el nodo. Estos datos son luego publicados a través de una aplicación, pudiendo visualizar las coordenadas en un mapa.

4.1.1. Hardware

Para la implementación del servidor se tuvo en cuenta tanto la opción de tener un servidor físico, instalado en el Instituto de Ingeniería Eléctrica, como la de utilizar un servidor virtual. Dado que en las primeras etapas del proyecto se encontraba disponible un servidor HP ProLiant DL380 G5, cuyas características pueden verse en su hoja de datos [36], se inclinó la decisión hacia la instalación de un servidor físico. Se consideró una buena instancia de aprendizaje y que podría ser utilizado en futuros proyectos.

4.1.2. Software

Para la ejecución del servidor se optó por utilizar el sistema operativo Ubuntu 18.04 con el cual se tenía experiencia previa. Para la instalación del servidor LoRaWAN, se optó por el servidor LoRaServer. Puede encontrarse mayor información en su página web [37]. Se eligió el mismo dado que es un software open source, con una instalación relativamente sencilla y cumple con los requisitos necesarios para la aplicación en este proyecto, por ejemplo, manejar activación de los nodos, recepcionar mensajes LoRaWAN y publicar estos mensajes a alguna aplicación.

A continuación se describen los distintos elementos que forman parte del servidor LoRaWAN implementado.

- **LoRa Server:** Se trata de un servidor de red de código abierto LoRaWAN. La función del servidor es la del manejo de las tramas recibidas por el gateway y además se encarga del envío de información hacia el nodo a través del gateway.

- **LoRa App Server:** Provee una interfaz web para el manejo de usuarios, gateways y nodos. Para poder recibir los paquetes del nodo, utiliza MQTT [38], HTTP o directamente escribe en una base de datos.

- **LoRa Gateway Bridge:** Elemento encargado de "traducir" la información recibida desde gateway. Tomando un paquete UDP y transformándolo en un archivo en formato JSON.

- **LoRa Geo Server:** Es un componente opcional que proporciona servicios de geolocalización y proporciona la ubicación de cada nodo. En este proyecto no será utilizado.

El protocolo de aplicación MQTT es elegido para la publicación de los mensajes en una aplicación. Se trata de un protocolo de mensajería basado en publicación-suscripción. Este protocolo funciona sobre UDP/IP. El mismo se puede utilizar para conexiones entre dispositivos con ubicaciones remotas y para minimizar el ancho de banda de la red. Además este protocolo garantiza la confiabilidad y cierto grado de seguridad de la entrega.

El MQTT broker es un servidor de traducción y almacenamiento de datos. El mismo se encarga de gestionar la publicación y suscripción de los clientes y administra la información publicada o recibida. Esta información se organiza en "topics", siendo un topic es una cadena de etiquetas que contiene una estructura jerárquica. Los emisores y receptores deben estar suscritos al mismo topic para poder comunicarse. En la imagen 4.2 se encuentra el esquemático del funcionamiento de este protocolo. Cada cliente puede publicar o suscribirse en un topic.

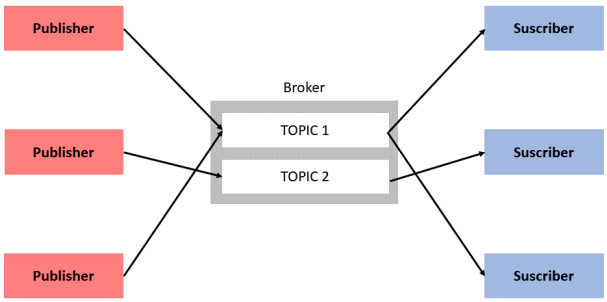


Figura 4.2: Esquema de protocolo MQTT.

De forma de poder entender mejor la instalación del network server, en la Figura 4.3 se observan los elementos del servidor. El elemento LoRa Gateway Bridge recibe los paquetes del gateway a través de UDP y luego LoRa App Server los publica en formato JSON en MQTT. El MQTT broker se encarga de almacenar dichos datos, cualquier cliente MQTT que se conecte al mismo topic podría acceder a esta información.

4.2. Gateway y Estación Base

Es importante aclarar que aunque se trate de un único dispositivo, éste cumplirá la función tanto de gateway para la red LoRa/LoRaWAN como de estación base para la estimación RTK, como se puede ver en la Figura 4.1.

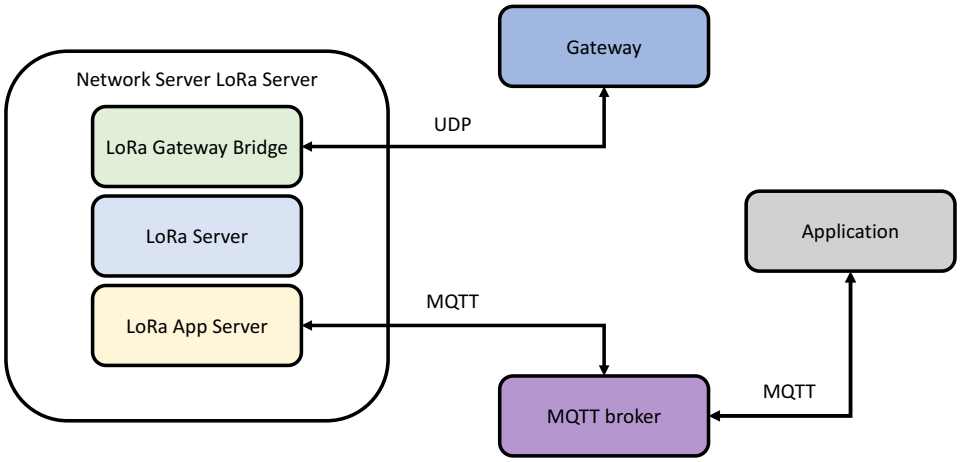


Figura 4.3: Arquitectura

Como se mencionó con anterioridad, en la Sección 2.3, para que el prototipo pueda funcionar como una estación base RTK es necesario que sea capaz de transmitirle al nodo información satelital. Por otro lado, el gateway es el encargado de traducir un mensaje recibido desde el nodo (utilizando modulación LoRa) a uno que viaja sobre una red UDP/IP hacia el servidor, como se explicó en la Sección 3.2.

4.2.1. Hardware

La instalación del gateway/estación base se realizó en una Raspberry Pi 3 [39]. En un principio se consideró la Raspberry 3 porque cuenta con conexión Wi-Fi, lo cual agrega cierta flexibilidad en cuanto a la ubicación del gateway. Finalmente, dado que el servidor no cuenta con una conexión Wi-Fi propia, se optó por una conexión cableada entre el gateway y el servidor.

Estación Base

Como fue mencionado en la introducción de este capítulo, se busca que el prototipo funcione como estación base para la solución RTK, por lo que se hizo necesario incorporar un hardware tal que pudiera trabajar con las señales GNSS. Es importante aclarar que se requiere de hardware especializado, que sea capaz de transmitir información de la señal satelital en sí misma.

Luego de investigar sobre diferentes marcas y modelos con los cuales se podría trabajar se optó por U-blox. Principalmente por motivos de costos, por la facilidad de su compra, porque puede encontrarse montado sobre una interfaz USB y por

la existencia un gran número de proyectos publicados utilizando esta marca. En particular, el modelo que soporta los mensajes utilizados para la técnica RTK es el NEO-M8T [40]. Una imagen de este hardware puede verse en la Figura 4.4

Figura 4.4: U-blox NEO-M8T.

Además, para implementar la solución RTK es necesario comunicar la estación base con el nodo. Como fue explicado anteriormente, esta comunicación se lleva a cabo utilizando LoRa (ver Figura 4.1). Para implementar esta comunicación se agregó un nuevo hardware en el prototipo, capaz de enviar y recibir los mensajes siguiendo las especificaciones físicas de LoRa. Se utilizó el GPS/LoRa HAT [41]. Con este módulo se buscó establecer una comunicación unilateral, desde la estación base hacia el nodo, para transmitirle a este último la información necesaria para el cálculo de RTK. Este hardware se basa en el chip SX1276 para el envío de datos utilizando LoRa. A pesar de contar con un receptor de GPS, este módulo no es capaz de proporcionar información de las medidas de pseudo-distancia, la cual es necesaria para trabajar con RTK. Una imagen de este módulo puede verse en la Figura 4.5.

Gateway

Si bien existe la posibilidad de implementar un gateway de un único canal, el cual sería suficiente para un proyecto de estas características (único nodo), la idea fue descartada. Un gateway de un único canal no cumple con la especificación de LoRaWAN [42].

Para implementar este gateway se utilizó un módulo MultiTech [43], diseñado para trabajar como un gateway LoRa/LoRaWAN de 8 canales. En particular, se utiliza el módulo mCard MTAC-LORA-915 [44] que maneja la banda AU 915-928 utilizada en Uruguay. Este módulo se basa en el chip Semtech SX1301 [45]. Este chip cuenta con la posibilidad de modular o demodular tanto LoRa como

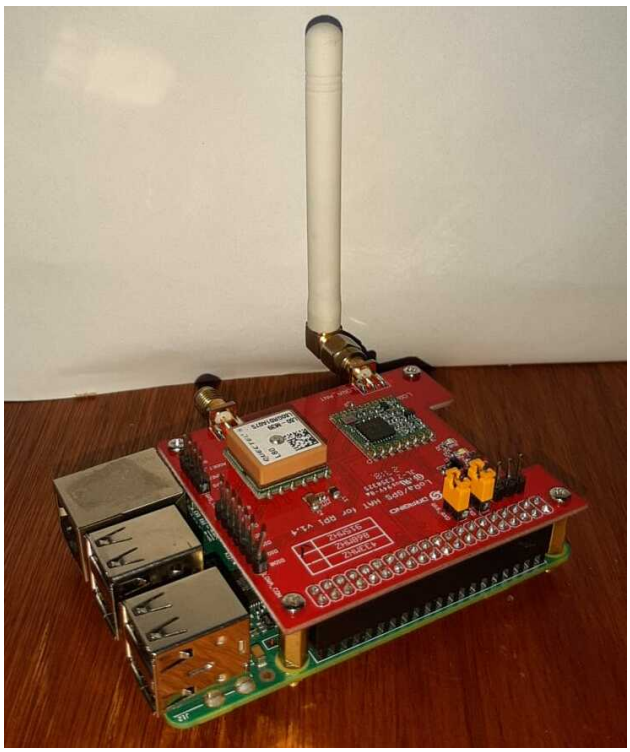


Figura 4.5: Dragino GPS/LoRa HAT.

GFSK. Para aplicarlo en el proyecto éste fue programado para trabajar utilizando LoRa en los 8 canales necesarios. Además, es muy simple adaptarlo para conectarlo mediante USB, lo cual facilita ampliamente el conexionado con la Raspberry. Este hardware puede verse en la Figura 4.6.

En las Figuras 4.7 y 4.8 puede verse un diagrama del prototipo implementado y una foto del hardware que efectivamente se utilizó.

4.2.2. Software

Estación base

Para el procesamiento y cálculo de las posiciones empleando RTK se utilizó la biblioteca RTKlib-demo5 [46]. Ésta es una optimización de la RTKlib para módulos U-blox. RTKlib es una biblioteca open source, que cuenta con una amplia documentación [47] y es muy utilizada, por lo que parecía una muy buena opción. A partir de aquí se nombrará indistintamente a la biblioteca utilizada como RTKlib o RTKlib-demo5.

Para lograr enviar datos utilizando la capa física de LoRa se utilizó la biblioteca 'dragino rpi-lora-transceiver' [48]. Esta biblioteca permite utilizar la capa física de LoRa, enviando un máximo de 255 bytes a una frecuencia definida por el usuario, siempre que esté incluida dentro de la banda de frecuencias que LoRa puede utilizar.

Figura 4.6: MultiTech mCard LoRa 915 MHz.

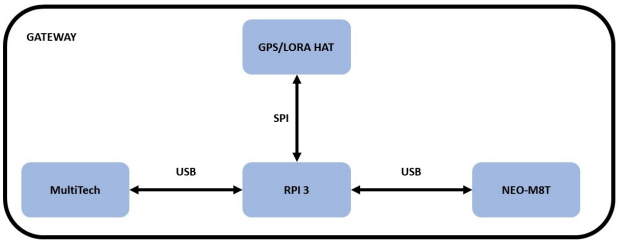


Figura 4.7: Diagrama del prototipo de gateway.

Gateway

Se utilizó la biblioteca "packet_forwarder" [49]. Esta biblioteca es open source y permite una fácil implementación de un gateway de 8 canales.

4.3. Nodo

Al igual que en el caso del gateway y estación base, el nodo deberá funcionar como un nodo LoRa/LoraWAN y como un nodo de la técnica RTK. Puede verse un diagrama ilustrativo en la Figura 4.9

Figura 4.8: Prototipo de gateway completo.

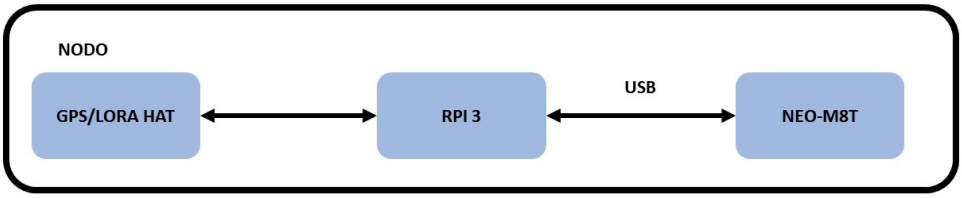


Figura 4.9: Diagrama del prototipo de nodo.

4.3.1. Hardware

Al igual que el gateway, el nodo se implementó utilizando una Raspberry Pi 3. Sobre la misma trabajan los diferentes periféricos para cumplir cada una de las funciones requeridas.

Nodo RTK

Para recibir la señal necesaria para el cálculo de RTK se utilizó un hardware idéntico al utilizado en el gateway (U-blox, NEO-M8T) .

Nodo LoRa/LoRaWAN

En el caso del nodo se utilizó el mismo hardware para la comunicación LoRa/LoRaWAN y la recepción de datos a través de LoRa físico provenientes de la estación base. Esta diferencia con el gateway marcó el funcionamiento de la aplicación final, como se explicará más adelante, ya que la recepción de datos a través de LoRa físico y el envío de datos a través de LoRa/LoRaWAN deberán compartir la

radio para trabajar. Al igual que en el gateway se optó por el módulo LoRa/GPS Hat.

4.3.2. Software

Como fue mencionado el nodo tiene esencialmente tres tareas. Debe recibir los datos de la estación base, calcular su posición utilizando estos datos y los que obtiene de su receptor de GNSS, y enviar esta posición al gateway LoRaWAN. Para esto se utilizaron tres librerías diferentes, todas ellas open source.

Nodo RTK

Para la recepción de las correcciones enviadas desde la estación base hacia el nodo se debe implementar una recepción de la capa física de LoRa. Se utiliza la librería 'dragino rpi-lora-transceiver', previamente utilizada en la estación base, ya que esta librería tiene implementado tanto el envío como la recepción de los datos vía LoRa física.

Por otro lado, para calcular la posición del nodo mediante RTK se utilizó la librería RTKlib-demo5.

Nodo LoRa/LoRaWAN

Para el funcionamiento como nodo LoRa/LoRaWAN se instaló la biblioteca 'lmic-rpi-lora-gps-hat' [50]. Se utiliza para la unión a la red LoRaWAN mediante el intercambio de claves de seguridad y para la transmisión los datos hacia un gateway. Se utilizó esta biblioteca ya que contaba con varios ejemplos en los que se podría basar el proyecto [51].

Esta biblioteca utiliza el modo OTAA para asociarse a la red. Esto genera un mayor retraso al momento de enviar los cálculos de la posición realizados por el nodo.

4.4. Resumen y Costos

Para cerrar este capítulo se realizará un breve resumen del hardware y software utilizado. Además, se añadirá el costo de cada una de estas partes, de forma de poder estimar un costo total para el proyecto.

Para tener una referencia del costo normal de una solución profesional de este tipo se buscaron productos de diferentes marcas. Los precios de las soluciones profesionales son muy variables, pudiendo costar entre U$S 2000,00 [52] y U$S 4500,00 [53] aproximadamente.

4.4.1. Servidor

El hardware utilizado para el servidor es un HP ProLiant DL380 G5. El mismo no contó con un costo asociado ya que se encontraba a disposición en el momento de efectuar el proyecto. Se buscó el precio de este modelo de servidor, sin embargo, el mismo fue discontinuado [54].

En caso de no contar con un servidor, la compra de un servidor nuevo, de la ultima generación del modelo utilizado lleva un costo de U$S 1600,00 [55]. Cabe aclarar que este costo puede variar mucho, por este motivo, se toma como referencia el costo de un servidor usado, que se asemeja más a la realidad de este proyecto. Este costo es de U$S 120 [56].

El software utilizado fue Ubuntu 18.04 y LoRaServer, ninguno de los cuales lleva un costo asociado. El resumen de estos datos puede verse en la Tabla 4.1.

hardware	
Producto	*Costo*
servidor HP ProLiant DL380 G5	U$S 120,00
	U$S 120,00
software	
Software	*Costo*
Ubuntu 18.04	-
LoRaServer	-
	-

Tabla 4.1: Costos asociados a la implementación del servidor.

4.4.2. Gateway

El hardware del gateway se centró en una Raspberry Pi 3, la cual tiene un costo asociado de U$S 41,59 [57]. Integrada a esta Raspberry se encuentran un módulo MultiTech mCard MTAC-LORA-H-915, que permite la ejecución de un gateway LoRa/LoRaWAN, un módulo U-blox NEO-M8T, que permite la recepción de señales GNSS, y un GPS/LoRa HAT, que permite enviar los datos satelitales de la estación base al nodo utilizando LoRa físico. Los costos asociados son U$S 193,00 [58], U$S 84,99 [59] y U$S 34,50 [60] respectivamente.

Por otro lado, es necesario instalar la biblioteca packet_forwarder, para poder manejar la radio como un gateway LoRa/LoRaWAN, la biblioteca rpi-lora-tranceiver, para el envío de datos utilizando LoRa física y RTKlib para el manejo de las señales satelitales y cálculos. Estas tres bibliotecas no tienen costo asociado. El resumen de estos costos puede verse en la Tabla 4.2

4.4.3. Nodo

El nodo, al igual que el gateway se basó en una Raspberry Pi 3, además de un GPS/LoRa HAT y un módulo U-blox NEO-M8T integrados a él. Los costos de

hardware	
Producto	*Costo*
Raspberry PI 3 B+	U$S 41,59
MultiTech mCard	U$S 193,00
GPS/LoRa HAT	U$S 34,50
U-blox NEO-M8T	U$S 84,99
Antena GPS	U$S 19,99
	U$S 374,07
software	
Bibliotecas	*Costo*
packet_forwarder	-
rpi-transciever	-
RTKlib	-
	-

Tabla 4.2: Costos asociados a la implementación del gateway.

estos productos son los presentados en la sección anterior y se resumen en la Tabla 4.3.

Como software se utiliza, al igual que en el gateway, la librería rpi-lora-tranceiver y RTKlib. Además, se utiliza la biblioteca lmic-rpi-lora-gps-hat que no agrega un costo al proyecto.

hardware	
Producto	*Costo*
Raspberry Pi 3 B+	U$S 41,59
GPS/LoRa HAT	U$S 34,50
U-blox NEO-M8T	U$S 84,99
Antena GPS	U$S 19,99
	U$S 181,07
software	
Bibliotecas	*Costo*
lmic	-
rpi-transiever	-
RTKlib	-
	-

Tabla 4.3: Costos asociados a la implementación del Nodo.

Podemos ver, sumando los subtotales de las Tablas 4.1, 4.2 y 4.3, que el costo total de este proyecto fue de U$S $675,14$. Si se compara con los precios de una solución profesional de este tipo, se puede considerar que los precios manejados para este prototipo son relativamente bajos.

En este capítulo pudo verse cada plataforma con la que se trabajó en este

proyecto, tanto en hardware como en software. Se describió cada una de ellas y la función que cumplirán en el prototipo final. Además, se pudo ver los costos que se deberían tener en cuenta en el caso de replicar este proyecto. Al mismo tiempo, se comparó estos costos con los que llevaría el adquirir una solución comercial al problema de la localización mediante RTK.

Capítulo 5

Implementación LoRa/LoraWAN

Introducción

En este capítulo se presenta el proceso de despliegue de una red LoRaWAN. En él se presentarán las distintas librerías utilizadas y los pasos realizados para el correcto funcionamiento de esta red y de los elementos que forman parte de ella.

5.1. Instalación del Network Server

Como ya se dijo anteriormente, para la instalación del Network Server LoRaWAN se utilizó un servidor del Instituto de Ingeniería Eléctrica. Fue necesario instalar algunos elementos previo a la instalación del servidor LoRaServer. Por ejemplo un broker MQTT, LoRaServer lo utiliza para publicar y recibir cargas útiles. Además se instaló PostgreSQL [61] que es un sistema de base de datos de código abierto. LoRaServer guarda los datos del gateway en una base de datos PostgreSQL. Por último se instaló Redis [62] que se utiliza como agente de base de datos, caché y mensaje, el cual es de código abierto. LoRaServer guarda todos los datos no persistentes [63] en Redis.

Se instalaron el LoRa Server, LoRa Gateway Bridge y LoRa App Server (ya definidos anteriormente) siguiendo los pasos de la guía de instalación de LoRaServer [64]. Durante el proceso de instalación se descargaron tres archivos de configuración: *loraserver.toml*, *lora-app-server.toml* y *lora-gateway-bridge.toml* presentados en el Apéndice A.1. En ellos se puede modificar la sub-banda de frecuencia, los canales utilizados, el puerto UDP del servidor en el cual se recibirán los paquetes del gateway, el puerto en el que se publican los mensajes por MQTT, el puerto en el que se comunican LoRa Server y LoRa App Server (Ver Figura 4.3), el puerto http para la interfaz web expuesta al usuario final y los tiempos de la sesión entre el nodo y el servidor entre otros parámetros. A continuación pueden verse algunas de las modificaciones (en rojo) realizadas en los archivos .toml.

Canales utilizados

```
# Enable only a given sub-set of channels
```

```
#
# Use this when ony a sub-set of the by default enabled channels are
being
# used. For example when only using the first 8 channels of the US band.
# Note: when left blank, all channels will be enabled.
#
# Example:
# enabled_uplink_channels=[0, 1, 2, 3, 4, 5, 6, 7]
enabled_uplink_channels=[8,9,10,11,12,13,14,15]
```

Sub-banda de frecuencia utilizada

```
# LoRaWAN regional band configuration.
# Note that you might want to consult the LoRaWAN Regional Parameters
# specification for valid values that apply to your region.
# See: https://www.lora-alliance.org/lorawan-for-developers
  network_server.band

# LoRaWAN band to use.
# Valid values are:
# * AS_923
# * AU_915_928
# * CN_470_510
# * CN_779_787
# * EU_433
# * EU_863_870
# * IN_865_867
# * KR_920_923
# * RU_864_870
# * US_902_928
name = ‘‘AU_915_928”
```

Protocolo mqtt

```
# This defines the backend to use for the data integration.
# Use the section name of one of the following integration backend.
# E.g. "mqtt" or "gcp_pub_sub".
backend=‘‘mqtt’’
```

Puerto para la comunicación entre LoRa Server y LoRa App Server

```
# This API is provided by LoRa App Server.
server="http://localhost:8004"
```

Puerto del servidor donde el gateway envía la información recibida

```
# Example: 0.0.0.0:1700 to listen on port 1700 for all network interfaces.
# This is the listeren to which the packet-forwarder forwards its data
# so make sure the ’serv_port_up’ and
# ‘serv_port_down’ from your
```

```
# packet-forwarder matches this port.
udp_bind = "0.0.0.0:1700"
```

Puerto donde se publican los mensajes MQTT

```
# MQTT server (e.g. scheme://host:port where scheme is tcp, ssl or ws)
server="tcp://localhost:1883"
```

Puerto http para la interfaz web expuesta al usuario

```
# This is the API and web-interface exposed to the end-user.
   application_server.external_api

# ip:port to bind the (user facing) http server to (web-interface and
REST / gRPC api)
bind="0.0.0.0:8081"
```

La banda de frecuencia utilizada como ya se dijo anteriormente es 915 MHz a 928 MHz (AU915-928). En particular en este proyecto el nodo utiliza los canales del 8 al 15 (916,8 a 918,2 MHz) y un ancho de banda de 125 kHz. Los canales utilizados pueden verse en la Tabla 5.1.

Canal	Frecuencia (MHz)
8	916.8
9	917.0
10	917.2
11	917.4
12	917.6
13	917.8
14	918.0
15	918.2

Tabla 5.1: Canales utilizados por el Nodo [32].

Una vez instalado el Network Server y editados los archivos de configuración se continuó instalando los dispositivos restantes de una red LoRa/LoRaWAN (nodo y gateway).

5.2. Instalación del Nodo

Se utilizó como nodo una Raspberry Pi, junto con la extensión Lora/GPS HAT y en la misma se instaló la biblioteca lmic-rpi-lora-gps-hat para que el mismo funcione como nodo en una red LoRaWAN.

La biblioteca lmic-rpi-lora-gps-hat está diseñada para que el nodo se asocie a la red mediante el modo OTAA (Capítulo 3), por lo que fue necesario ingresar algunas claves en el servidor y en el nodo, para que tenga permisos para asociarse a la red. Se modificó en el nodo la clave AppEUI, la clave DevEUI y hacer que

coincidan con las ingresadas en el servidor. Se tuvo que verificar que el servidor estuviera usando la notación LSB (byte menos significativo primero). También se ingresó la clave AES específica del dispositivo, correspondiente a la clave secreta que comparten el servidor con el nodo, conocida como la clave de aplicación (AppKey) y en este caso se utilizó la notación MSB (el byte más significativo primero). A continuación se presenta el lugar del código donde se ingresan las claves.

Claves del nodo

// application router ID (LSBF)
static const u1_t APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0, 0x00 };

// unique device ID (LSBF)
static const u1_t DEVEUI[8] = { 0x42, 0x42, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF };

// device-specific AES key (derived from device EUI)
static const u1_t DEVKEY[16] = { 0x3C, 0x4F, 0xCF, 0x09, 0x88, 0x15, 0xF7, 0xAB,0xA6, 0xD2, 0xAE, 0x28, 0x16, 0x15, 0x7E, 0x2B };

Uno de los problemas que surgió a la hora de instalar dicha biblioteca fue la configuración de la banda a utilizar, ya que el gateway y el nodo tenían que estar configurados para recibir/enviar en la misma banda de frecuencias y en los mismos canales. La subbanda utilizada es la banda número 1. Ésto pudo editarse en el archivo transmit.c de la biblioteca instalada en el nodo. En el siguiente código se puede observar la banda elegida.

```
void  LMIC_enableSubBand (u1_t band) {
  ASSERT(band < 8);
  u1_t start = band * 8;
  u1_t end = start + 8;

  // enable all eight 125 kHz channels in this subband
  for (int channel=start; channel < end; ++channel )
      LMIC_enableChannel(channel);

  // there's a single 500 kHz channel associated with
  // each group of 8 125 kHz channels. Enable it, too.
  LMIC_enableChannel(64 + band);
}
```

La función LMIC_selectSubBand es definida en el archivo lmic.c de la biblioteca. A continuación se presentan sectores del código donde puede observarse que los canales elegidos para el envío de datos son del 8 al 15.

```
void  LMIC_selectSubBand (u1_t band) {
  ASSERT(band < 8);
```

```
  for (int b=0; b<8; ++b) {
    if (band==b)
      LMIC_enableSubBand(b);
    else
      LMIC_disableSubBand(b);
  }
}
```

Para poder corroborar que se estuviera utilizando la misma banda de frecuencias que en la que escuchaba el gateway se utilizó un analizador de espectro implementado con HackRF One [65] y GQRX SDR [66], que permitió observar con claridad en qué frecuencias se estaban enviando los datos.

5.3. Instalación del Gateway

En el caso del gateway también se utilizó una Raspberry Pi y el módulo de LoRa ya descrito anteriormente. En este caso se instaló otra biblioteca ya que las funciones del gateway en una red LoRaWAN son muy diferentes a las del nodo. Se instaló la biblioteca packet_forwarder del proyecto *Lora network packet forwarder* como se expuso anteriormente. Dentro de la misma se utilizó la función basic_packet_forwarder. La misma se ejecuta en un gateway que se encarga de ser puente entre nodo y servidor reenviando los paquetes que recibe por radiofrecuencia del nodo por una red UDP/IP hacia el servidor y viceversa.

Para poder configurar el gateway de forma correcta, permitiendo la comunicación con el servidor a través de la red UDP/IP y con el nodo a través del aire, esta biblioteca cuenta con un archivo .json en el que se pueden modificar:

- La dirección IP del servidor al cual enviar los paquetes o del cual recibirlos.
- El puerto UDP en el cual se envían los paquetes al servidor.
- Una ID del gateway que le permite al servidor identificar el gateway del cual recibe los mensajes.
- Las frecuencias en las que escucha el gateway.

Este archivo de configuración completo puede observarse en el Apéndice A.3. A continuación se muestra parte de esta configuración.

“SX1301_conf”: {
“lorawan_public”: true,
“clksrc”: 0,
“clksrc_desc”: “radio_1 provides clock to concentrator for most devices except MultiTech. For MultiTech set to 0.”,
“antenna_gain”: 0,
“antenna_gain_desc”: “antenna gain, in dBi”,

“radio_0”: {
“enable”: true,
“type”: ”SX1257”,
“freq”: 917200000,
“rssi_offset”: -166.0,
“tx_enable”: true,
“tx_freq_min”’: 915000000,
“tx_freq_max”: 928000000
},
“radio_1”: {
“enable”: true,
“type”: ”SX1257”,
“freq”: 917900000,
“rssi_offset”: -166.0,
“tx_enable”: false
},
“chan_multiSF_0”: {
“desc”: ”Lora MAC, 125kHz, all SF, 916.8 MHz”,
“enable”: true,
“radio”: 0,
“if”: -400000
},
}

Se puede observar que la frecuencia central es 917,2 MHz para la radio 0. El resto de las frecuencias utilizadas se definen en el parámetro “if” de la configuración, donde el número elegido se corresponde con el offset que tiene dicha frecuencia con respecto a la central. Por ejemplo, la frecuencia 916,8 MHz se define con el número -400000.

La otra frecuencia central es 917,9 MHz para la radio 1. Por lo tanto las frecuencias definida con la radio 1 tienen en el parámetro “if” un offset con respecto a esta nueva frecuencia central. Por ejemplo la frecuencia 917,6 MHz se define con el número -300000.

“chan_multiSF_4”: {
“desc”: “Lora MAC, 125kHz, all SF, 917.6 MHz”,
“enable”: true,
“radio”: 1,
“if”: -300000
},

A continuación se puede observar una sección del archivo de configuración donde fueron modificados el puerto y la dirección IP del servidor, al cual se envían los paquetes mediante UDP/IP, además se muestra el nombre (ID) asignado al gateway, el cual lo identifica cuando llegan paquetes al servidor.

“gateway_conf”:

```
"gateway_ID": "AA555A0000000001",
/* change with default server
address/ports, or overwrite
in local_conf.json */
"server_address": "164.73.45.130",
"serv_port_up": 1701,
"serv_port_down": 1701,
```

5.4. Funcionamiento de la Red LoRa/LoRaWAN

Una vez instalado LoRaServer en el servidor y las bibliotecas correspondientes en el nodo y gateway fue posible poner en funcionamiento la red LoRaWAN.

A continuación se presentan los pasos que deben realizarse para correr el Network Server LoRaServer. Se comienza corriendo en distintas líneas de comando del servidor el loraserver, lora-app-server y el lora-gateway-bridge instalados. Luego se abre la interfaz de aplicación ingresando desde un servidor web a `http://localhost:puerto`, siendo el valor de puerto el que es configurado en el archivo lora-app-server.toml. Dentro de dicha interfaz se define el puerto del Network Server que se crea (Figura 5.1), en este caso el puerto fue 8002. Dentro de este Network Server se puede crear una aplicación específica, a la que en este caso se le puso el nombre App_Parriot (Figura 5.2).

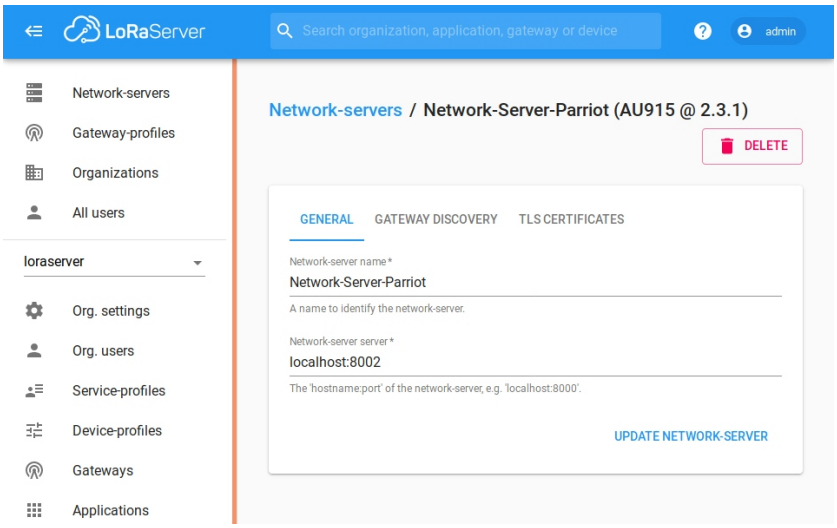


Figura 5.1: Creación de un Network Server.

Luego, se define el gateway que se está utilizando. Para esto existe un identificador (Figura 5.3) ya ingresado en el archivo de configuración de la biblioteca instalada en el gateway. En una misma aplicación pueden manejarse muchos gateways a la vez, pero en este proyecto se utilizó solo uno.

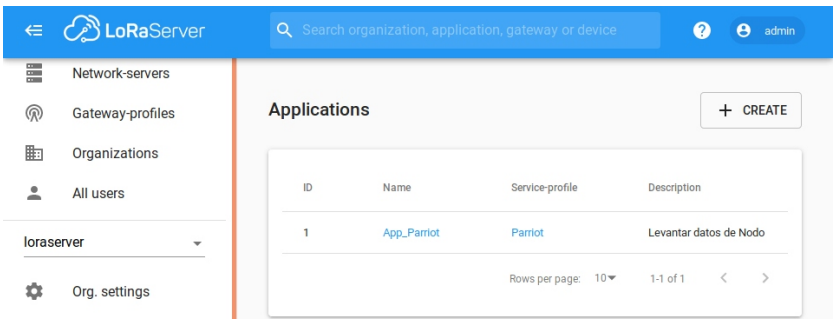


Figura 5.2: Creación de una aplicación.

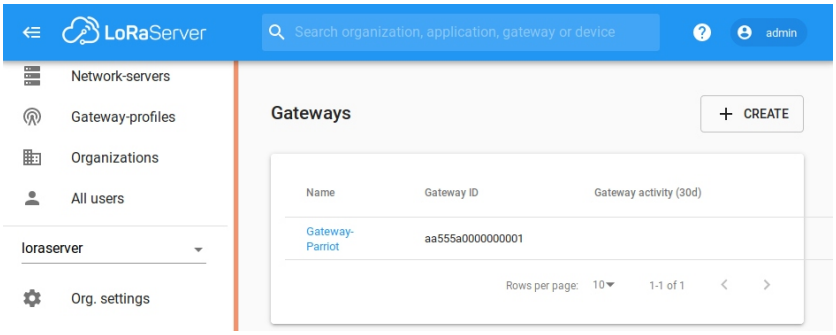


Figura 5.3: Creación de un Gateway.

Se pueden crear distintos perfiles de nodos para un mismo gateway. Para este proyecto se creó un solo nodo con la aplicación de GPS de alta precisión. En la Figura 5.4 se pueden observar dos perfiles. El utilizado es Nodo LSB.

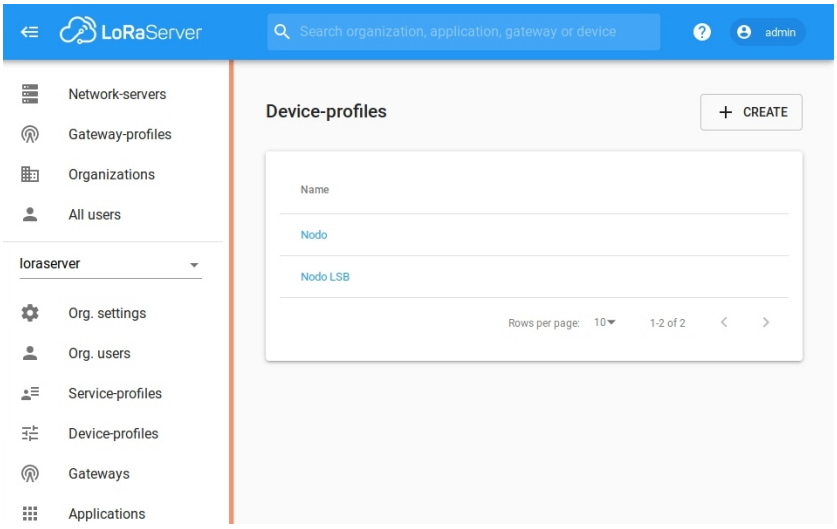


Figura 5.4: Perfiles de Nodos creados para esta aplicación.

También se debe elegir el modo en que el se conecta el nodo a la red. En este caso es el modo OTAA (Figura 5.5), ya que en el nodo se definió así. Se ajustó

la clave Application Key verificando que coincida con la última clave ingresada en la biblioteca instalada en el nodo. Se debe corroborar que estén ingresadas en notación MSB.

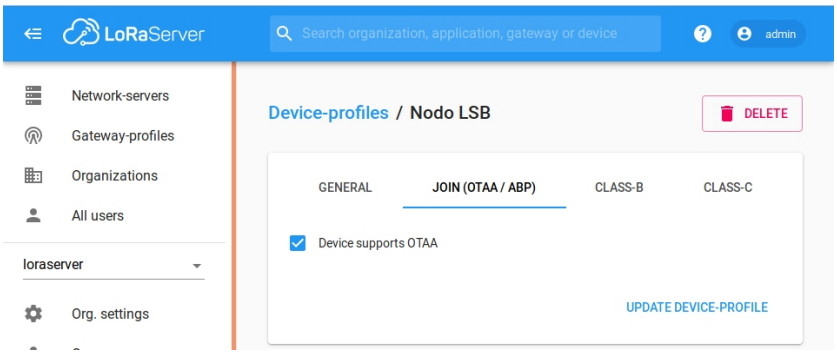


Figura 5.5: Modo de conexión a la red elegido.

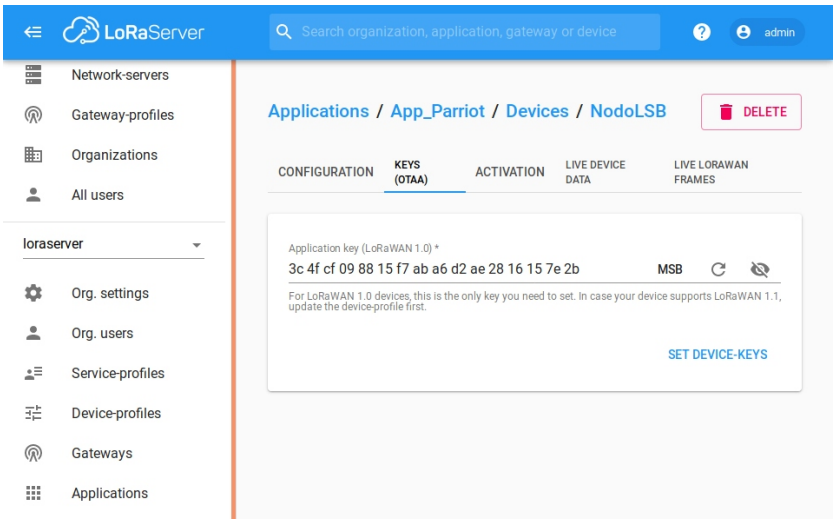


Figura 5.6: Application Key del modo OTAA.

Dentro de la aplicación App_Parriot se agregó el perfil de nodo creado (Nodo LSB) y en la configuración de la aplicación se puede definir en qué base se quieren recibir los datos que envía el nodo. En principio estaba ajustado en base64, pero fue modificado el script para recibirlo en ASCII y así poder ser interpretado por la aplicación final. Esto puede observarse en la Figura 5.7.

Una vez ajustados todos los parámetros en el Network Server ya es posible poner en funcionamiento la red.

Se debe correr en el gateway el ejecutable basic_packet_forwarder de la biblioteca instalada, que hace que el mismo se comporte como pasarela en una red LoRaWAN. En el nodo se corre el archivo transmit.out de la biblioteca instalada que permite que el mismo se una y transmita datos en una red LoRaWAN.

En las terminales donde se corrieron el loraserver, lora-app-server y lora-gateway-bridge se despliegan mensajes informando del estado del servidor. Por ejemplo informa del estado de la conexión, si se reciben o no paquetes, o si se envían y en

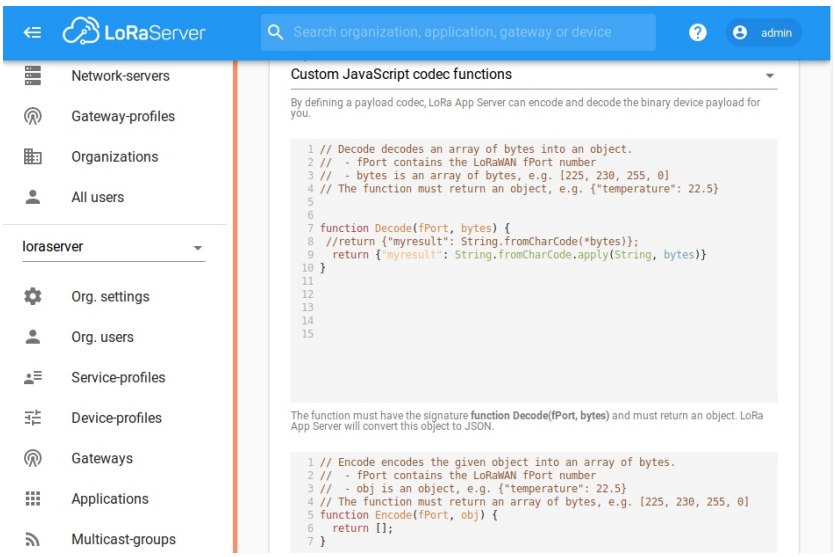


Figura 5.7: Base utilizada para decodificar el mensaje

la frecuencia en que lo hacen. Ejemplos de lo anterior pueden verse en las Figuras 5.8, 5.9, 5.10.

```
root@parriot-ProLiant-DL380-G5:/etc# loraserver
INFO[0000] starting LoRa Server                          band=AU_915_928 docs="h
ttps://docs.loraserver.io/" net_id=000000 version=2.3.1
INFO[0000] disabling all channels
INFO[0000] enabling channels                             channels="[0 1 2 3 4 5
6 7]"
INFO[0000] setup redis connection pool                   url="redis://localhost:
6379"
INFO[0000] connecting to postgresql
INFO[0000] gateway/mqtt: connecting to mqtt broker       server="tcp://localhost
:1883"
INFO[0000] no geolocation-server configured
INFO[0000] configuring join-server client                ca_cert= server="http:/
/localhost:8004" tls_cert= tls_key=
INFO[0000] no network-controller configured
INFO[0000] applying database migrations
INFO[0000] backend/gateway: connected to mqtt server
INFO[0000] gateway/mqtt: subscribing to rx topic         qos=0 topic=gateway/+/r
x
INFO[0000] gateway/mqtt: subscribing to stats topic      qos=0 topic=gateway/+/s
tats
INFO[0000] backend/gateway: subscribing to ack topic     qos=0 topic=gateway/+/a
```

Figura 5.8: Loraserver en línea de comandos

De esta forma se pudo observar cómo se recibían paquetes desde el nodo en la interfaz web del servidor. Al comienzo llega un mensaje join correspondiente al mensaje de pedido de unión del nodo al servidor (Figura 5.11). Ingresando en cada paquete recibido, se puede ver en qué frecuencia se envió dicho paquete, el payload que contiene, el gateway del cual llega el paquete y las coordenadas del mismo entre otros (ver Figura 5.12). Si se abre otro paquete, se puede observar que ambos se enviaron a distintas frecuencias, lo que corrobora los saltos de canal realizados por el nodo (Figura 5.13).

```
root@parriot-ProLiant-DL380-G5:/etc# lora-app-server
INFO[0000] starting LoRa App Server                      docs="https://www.loras
erver.io/" version=2.4.1
INFO[0000] connecting to postgresql
INFO[0000] setup redis connection pool
INFO[0000] handler/mqtt: TLS config is empty
INFO[0000] handler/mqtt: connecting to mqtt broker       server="tcp://localhost
:1883"
INFO[0000] applying database migrations
INFO[0000] handler/mqtt: connected to mqtt broker
INFO[0000] handler/mqtt: subscribing to tx topic         qos=0 topic=application
/+/device/+/tx
INFO[0000] migrations applied                            count=0
INFO[0000] starting application-server api               bind="0.0.0.0:8006" ca-
cert= tls-cert= tls-key=
INFO[0000] starting join-server api                      bind="0.0.0.0:8004" ca_
cert= tls_cert= tls_key=
INFO[0000] starting client api server                    bind="0.0.0.0:8081" tls
-cert= tls-key=
INFO[0000] registering rest api handler and documentation endpoint  path=/api
INFO[0021] finished unary call with code OK              grpc.code=OK grpc.metho
```

Figura 5.9: Lora-app-server en línea de comandos

```
root@parriot-ProLiant-DL380-G5:/etc# lora-gateway-bridge
INFO[0000] starting LoRa Gateway Bridge                  docs="https://www.loras
erver.io/lora-gateway-bridge/" version=2.6.2
INFO[0000] backend: set max reconnect interval: 10m0s
INFO[0000] backend: TLS config is empty
INFO[0000] backend: connecting to mqtt broker            server="tcp://127.0.0.1
:1883"
INFO[0000] gateway: starting gateway udp listener        addr="0.0.0.0:1700"
INFO[0000] backend: connected to mqtt broker
INFO[0001] backend: subscribing to topic                 qos=0 topic=gateway/aa5
55a0000000001/tx
INFO[0001] backend: subscribing to topic                 qos=0 topic=gateway/aa5
55a0000000001/config
INFO[0416] gateway: rxpk packet received                 addr="164.73.38.156:458
82" data="AAAAAAAAAAAAQkJFZ4mrze8D6Fduxn0=" mac=aa555a0000000001
INFO[0416] backend: publishing packet                    qos=0 topic=gateway/aa5
55a0000000001/rx
INFO[0417] backend: downlink packet received             topic=gateway/aa555a000
0000001/tx
INFO[0422] gateway: rxpk packet received                 addr="164.73.38.156:458
82" data=gOUgSwCAAAABtCXBoQC0Wp+PpsIw2oJNLeH8w9qPbdTZL4ASmOpzYSec+vNcNTr0yP5rH/n
```

Figura 5.10: Lora-gateway-bridge en línea de comandos

5.5. Aplicación

El objetivo principal de la aplicación final es imprimir en un mapa de Google-Maps las coordenadas precisas que devuelve el nodo. El mismo, luego de realizar el cálculo de su posición devuelve un paquete que contiene sus coordenadas. Este paquete es enviado por el nodo a través de la red LoRaWAN llegando al Network Server. El servidor se encarga de publicarlo a través de MQTT en el puerto 1883, en el localhost (127.0.0.1) y con un topic específico.

Una computadora que se encuentre en la misma red que la del servidor, puede ponerse a la escucha como cliente a la dirección IP del mismo, en el puerto 1883 y con el topic correspondiente recibiendo así los mensajes enviados por el nodo. La aplicación creada se encarga de desarrollar un cliente capaz de escuchar en el puerto 1883, adquirir la información enviada por el nodo y obtener de la misma las coordenadas.

La aplicación se desarrolló con el lenguaje Python. Se utilizó Spyder [67], que es un entorno de desarrollo interactivo para el lenguaje Python. Se debió instalar

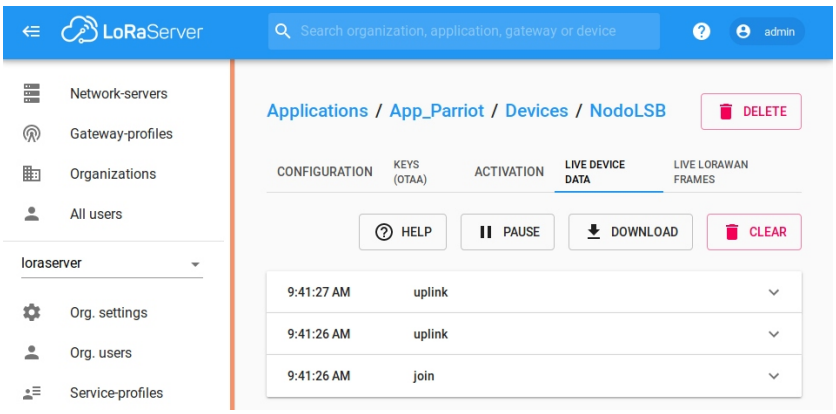


Figura 5.11: Paquete recibido

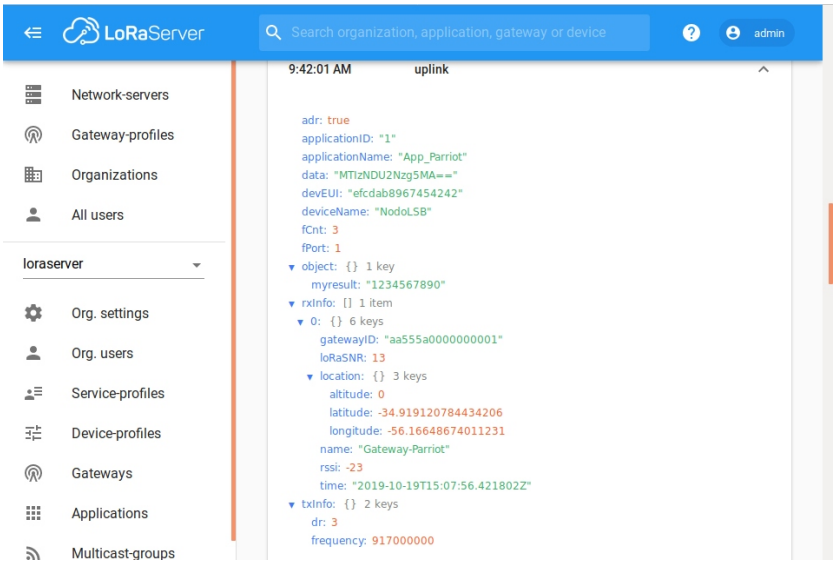


Figura 5.12: Paquete recibido

la librería paho-mqtt [68], la que permite crear un cliente MQTT usando dicho lenguaje, contiene distintas funciones para publicar mensajes y suscribirse a diferentes temas.

Se utilizaron las funciones on_connect y on_message implementadas por la librería anteriormente instalada. La función on_connect permite conectarse a un broker MQTT para lo cual es necesaria la dirección IP y el puerto. Esta función es llamada cuando el broker MQTT acepta la solicitud de conexión.

El cliente, una vez conectado con el MQTT broker se suscribe a un topic particular. Una vez conectado utiliza la función on_message, función llamada cuando hay un mensaje disponible para recibir en el topic al cual se suscribió. La misma obtiene en la variable “message” información del mensaje recibido. En particular para esta aplicación interesa obtener message.payload que contiene el paquete enviado por el nodo. Con esta variable se trabajó para obtener las coordenadas y

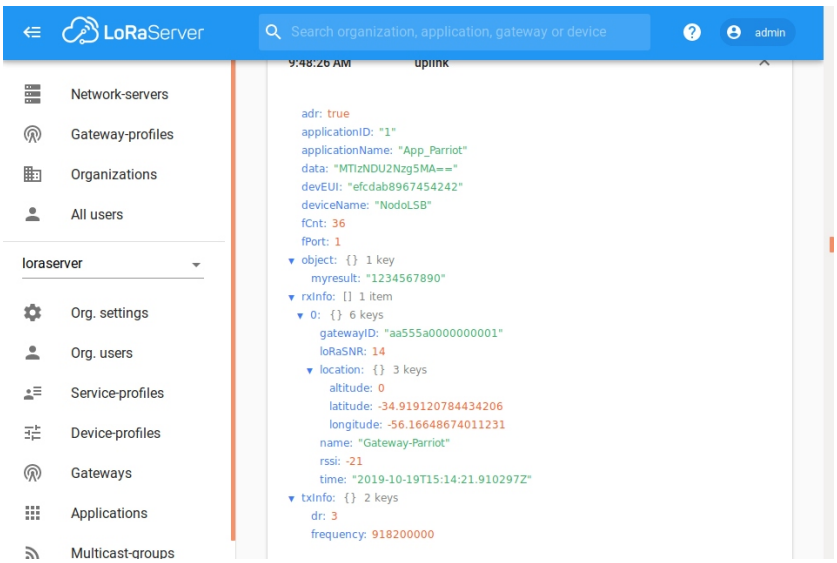


Figura 5.13: Paquete recibido

graficarlas en GoogleMaps.

Para poder obtener los archivos de mapas de GoogleMaps y graficar en ellos, fue necesario obtener una ApiKey de Google Maps. La misma se obtuvo de forma gratuita pero fue necesario registrarse. Para poder acceder a estos mapas y modificarlos se descargó la biblioteca gmplot [69], la cual contiene la función GoogleMapPlotter que permitió crear una imagen de GoogleMaps. Las variables de esta función son las coordenadas en donde se centra la imagen y el tamaño de la misma. Se ingresaron las coordenadas de la Facultad de Ingeniería (-34.918318, -56.166656) ya que las pruebas se realizaron allí. Con la función draw de gmplot se generó un archivo .html con el mapa (creado previamente con GoogleMapPlotter). Luego de obtener la imagen del mapa, se utilizó la función gmap.scatter la cual se encarga de dibujar círculos en el mapa.

Cada vez que se recibe un mensaje por MQTT se obtiene el payload y se extraen las coordenadas. Éstas se ingresan como variable a la función gmap.scatter, graficando de esta forma un círculo con dichas coordenadas en el archivo .html creado. En la Figura 5.14 se puede observar un ejemplo de una de las pruebas realizadas con el nodo estático. A su vez en la consola de Spyder se pueden observar los mensajes a medida que van llegando.

En este capítulo se expuso el proceso realizado para el desarrollo de una red LoRa/LoRaWAN que cuenta con un nodo, un gateway, un servidor y una aplicación. En el diagrama de la Figura 5.15 se presenta el sistema completo implementado para darle funcionamiento a la red. En él se pueden observar los módulos de LoRa utilizados, las bibliotecas instaladas y los protocolos que intervienen en los enlaces entre los dispositivos.

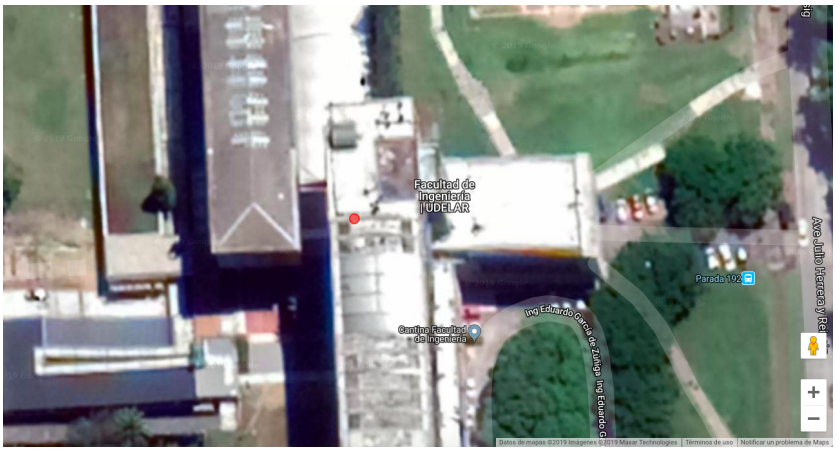


Figura 5.14: Mapa con coordenadas del nodo

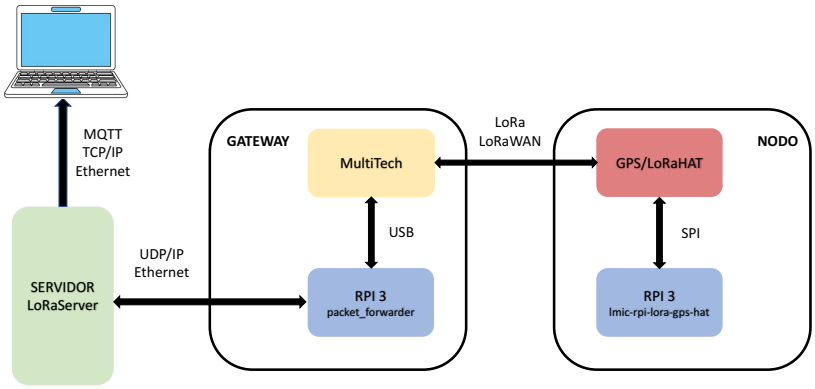


Figura 5.15: Diagrama de la red LoRa/LoRaWAN implementada

Capítulo 6

Implementación RTK

En este capítulo se describen los distintos componentes involucrados en la solución al problema de la ubicación, su interacción y su configuración. Se utilizará el término "receptor" para referirse especialmente a los chips de U-Blox decodificadores de las señales satelitales, y los términos "nodo" y "base" para referirse a los distintos conjuntos receptor-raspberry-radio.

6.1. Protocolo U-Blox

El fabricante de chips U-Blox tiene su propio protocolo de comunicación en el que define una gran cantidad de mensajes tanto para configurar al receptor como para comunicar información de las señales satelitales. Todos los mensajes tienen la misma estructura, como se presenta en la Tabla 6.1: dos bytes de sincronismo, dos bytes de caracterización, dos bytes indicando el largo de la carga útil (little endian), carga útil y dos bytes de checksum.

Sync1	Sync2	Clase	ID	Largo	Carga útil	CKA	CKB
1 byte	1 byte	1 byte	1 byte	2 bytes	*Largo* bytes	1 byte	1 byte
0xB5	0x62	-	-	-	-	-	-

Tabla 6.1: Estructura de mensajes UBX.

En esta sección se mencionarán únicamente los mensajes más relevantes para esta aplicación, para más detalles puede consultarse la bibliografía [70]. Cada mensaje se define por su Clase e ID, en los párrafos que siguen se indican entre paréntesis.

6.1.1. Mensajes de Configuración

Son mensajes que se envían al receptor para definir su comportamiento.

- **(0x06 0x3E) UBX-CFG-GNSS:**
  Configura qué constelaciones (GPS, GLONASS, Galileo, otras) y qué señales

(L1, L2, L3, L5) dentro de ellas se utilizarán. También determina la cantidad de canales que se le asignará a cada constelación.

- **(0x06 0x01) UBX-CFG-MSG:**
  Configura qué mensajes se desea recibir, por qué puerto (UART, USB, SPI) y con qué cadencia.

- **(0x06 0x08) UBX-CFG-RATE:**
  Configura el período con que se toman medidas de las señales satelitales.

6.1.2. Mensajes de Información

Son los mensajes que el receptor devolverá al usuario. Van desde coordenadas estimadas por el propio chip, hasta información cruda de las señales satelitales, pasando por mensajes de navegación enviados por los satélites.

- **(0x02 0x15) UBX-RXM-RAWX:**
  Contiene información general del estado del receptor seguida de, entre otros, información de fase y pseudorango medidos para varios satélites. El tamaño de estos mensajes varía en función de la cantidad de satélites de los cuales se posee información.

- **(0x02 0x13) UBX-RXM-SFRBX:**
  Reporta un sub-marco completo de información de navegación decodificado de una única señal.

6.2. Prestaciones RTKlib-demo5

La biblioteca utilizada para el cálculo de la posición cuenta con varias APs (Programas de Aplicación, por sus siglas en inglés) que facilitan la interacción con el usuario. Se presenta a continuación una breve descripción de las APs utilizadas en las distintas instancias del proyecto. Por más detalles referirse al manual de la RTKlib [47]. Se mencionará la RTKlib-demo5 por su nombre o simplemente como RTKlib.

6.2.1. Cálculo en Tiempo Real

Para el cálculo en tiempo real se trabajó con APs que podían ser compilados para trabajar sobre el sistema operativo Raspbian.

- **STR2STR**
  Es un pasamanos, toma un flujo de datos de entrada y los replica en la salida. Ambos extremos son configurables para leer/escribir desde/en un archivo, puerto serial, puerto TCP y más.

- **RTKRCV**
  Es quien realiza el cálculo de la posición. Se definen en un archivo de configuración parámetros relacionados al cálculo de la posición así como también la configuración de los flujos de datos de entrada y salida. Se necesitan dos entradas correspondientes a la información obtenida de los receptores de nodo y base, y como salida devuelve las coordenadas calculadas del nodo. Además es posible generar un log de los flujos de entrada de forma de poder repetir el cálculo en post-procesamiento. Las opciones de entrada y salida son las mismas que para STR2STR.

6.2.2. Cálculo en Post-Procesamiento

Para el cálculo en post-procesamiento se trabajó sobre la interfaz gráfica de RTKlib para Windows.

- **RTKCONV**
  Oficia de traductor. Se indica un archivo de entrada con su formato de mensajes y devuelve (siempre que exista tal información) las observaciones de las señales y la información de navegación.

- **RTKPOST**
  Se corresponde con RTKRCV pero para post-procesamiento. Las opciones de configuración se mantienen, con la excepción de que se toman como entrada tres archivos, las observaciones de base y nodo y un archivo con información de navegación.

- **RTKPLOT**
  Se le especifica un archivo de soluciones y plotea las coordenadas.

6.3. Descripción del Sistema

Recordando que el objetivo es ubicar al nodo, es él quien calculará su posición. Para ello hará uso de las señales obtenidas de su receptor y de las recibidas desde la base. Se tienen entonces tres aspectos que es necesario solucionar: la configuración de los receptores, la configuración de la biblioteca y la implementación de un link entre base y nodo.

Como fue definido en los requerimientos de este proyecto, el link base-nodo fue realizado mediante un enlace de radio con modulación LoRa. Por otra parte, para la comunicación receptor-raspberry se optó por un puerto TCP para funcionar como link, dado que la biblioteca RTKlib tenía prevista esta posibilidad. En la Figura 6.1 se presenta un diagrama del sub-sistema encargado del cálculo de la posición.

Como se verá en secciones posteriores, las características del link base-nodo determina ciertos límites en cuanto a la capacidad de transmisión de datos.

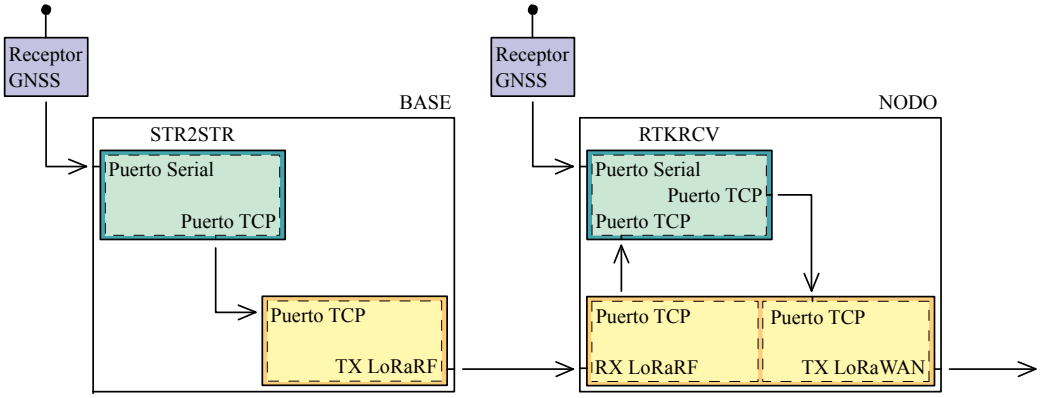


Figura 6.1: Sistema para el cálculo de la posición.

6.4. Comunicación Base-Nodo

El link base-nodo se estableció por un enlace de radio con modulación LoRa. Se trabajó sobra la premisa de que para obtener la mejor aproximación posible a la ubicación real del dispositivo, es necesaria información de la mayor cantidad de satélites a la mayor frecuencia posible. Es por esto que se optó por trabajar con SF7, siendo éste el que permite la mayor capacidad de transmisión de datos por segundo.

Como fue mencionado anteriormente, para utilizar la técnica RTK se requiere de información cruda de las señales satelitales. Se optó entonces por trabajar con los mensajes propietarios de U-Blox UBX-RXM-RAWX que contienen la información de la fase y código, junto con los UBX-RXM-SFRBX que contienen información de navegación decodificada para cada satélite. Dado que para calcular la posición se utiliza la información de los satélites en común entre base y nodo, no es necesario que la información de navegación sea transmitida desde la base, si no que es suficiente con la que recibe directamente el nodo desde su propio receptor. De esta manera se logra disminuir al mínimo la cantidad de información que debe ser transmitida por el link LoRa.

La estructura de los mensajes UBX-RXM-RAWX se muestra en la Tabla 6.2, donde se puede observar la dependencia del tamaño del mensaje con la cantidad de satélites observados. Para determinar el número de satélites a utilizar, primero se decidió trabajar a una frecuencia de recolección de medidas de 1 Hz. Esta condición significa que el sistema debe ser capaz de enviar un mensaje completo por radio desde base a nodo en 1 segundo.

- Período de recepción de mensajes: 1 s.

Para un Spreading Factor de valor 7, corresponde según la descripción del protocolo LoRa un ancho de banda de 125 kHz y una capacidad máxima de 5470 bps. Por limitaciones de hardware solo pueden transmitirse paquetes de 255 bytes, por lo que resulta que el delay mínimo entre transmisiones debe ser de 373 ms.

- Tiempo mínimo entre transmisiones de paquetes: 373 ms.

Sync1	Sync2	Clase	ID	Largo	Carga útil	CKA	CKB
1 byte	1 byte	1 byte	1 byte	2 bytes	*Largo* bytes	1 byte	1 byte
0xB5	0x62	0x02	0x15	-	16+32*nSat	-	-

Tabla 6.2: Estructura de mensajes UBX-RXM-RAWX.

El programa desarrollado permite únicamente envíos por RF de forma periódica, por lo que hay que prever un tiempo suficiente de transmisión para 255 bytes. No hay consideraciones especiales para paquetes más pequeños. Tomando el tiempo mínimo entre transmisiones resulta en que solamente pueden enviarse 2 paquetes completos de 255 bytes por segundo. Recordando que a su vez 1 segundo es el período de recepción de mensajes de U-BLox, se limita a 510 (255×2) el tamaño máximo del mensaje.

- Tamaño máximo de mensaje: 510 bytes.

Tomando la estructura de mensajes mostrada en la Tabla 6.2, se deduce que la máxima cantidad de satélites que es posible utilizar es 15. Si el $largoMensaje = 24+32\times nSat < 510\ bytes \implies nSat < 15,25$. Para 15 satélites, $largoMensaje = 504\ bytes$.

- Tamaño final de mensaje: 504 bytes.

Como es necesario realizar dos transmisiones completas, el tiempo máximo por transmisión es de 500 ms. Se optó por realizar una transmisión cada 400 ms, pero cualquier valor en el rango 373 ms a 500 ms es válido.

- Tiempo máximo entre transmisiones de paquetes: 500 ms.

- Tiempo final entre transmisiones de paquetes: 400 ms.

Habiendo resuelto los parámetros necesarios para asegurar un buen funcionamiento del link base-nodo, el sistema queda determinado de forma más precisa como se describe a continuación. En la base, un receptor se comunica con la raspberry por serial, que mediante la RTKlib levanta la información y la escribe en un puerto TCP. Este puerto es leído y su contenido fragmentado en paquetes de hasta 255 bytes, para luego ser enviador por RF con modulación LoRa. En el nodo, los paquetes recibidos por la radio son escritos en otro puerto TCP, desde el cual levantará la información la RTKlib. Con esta información, sumada a la que recibe por serial del propio receptor del nodo, la RTKlib calculará la posición.

6.5. Configuración de Receptores

Para configurar los receptores se utilizó el software proporcionado por U-Blox, U-Center [71]. Éste permite, entre otras cosas, definir qué mensajes y a qué frecuencia se desean obtener. Si bien en primera instancia se habían configurado

ambos receptores de igual manera, las restricciones introducidas por el hardware y el protocolo elegido para la transmisión de datos hicieron necesario un cambio en la configuración del receptor de la base.

La base fue configurada de manera de transmitir únicamente mensajes UBX-RXM-RAWX para un máximo de 15 satélites, habilitando únicamente las constelaciones de GPS y GLONASS. De esta manera la base retransmite al nodo únicamente las medidas de las señales satelitales.

Siguen los mensajes configurados. Por una explicación detallada del significado de cada byte, referirse a la descripción del protocolo U-Blox [70].

- **UBX-CFG-GNSS**
  CFG-GNSS - 06 3E 3C 00 00 20 0F 07 00 00 0F 00 01 00 01 01 01 00 00 00 00 00 00 01 02 00 00 00 00 00 00 01 03 00 00 00 00 00 00 01 04 00 00 00 00 00 00 03 05 00 00 00 00 00 00 05 06 00 0F 00 01 00 01 01
  En rojo, el máximo número de satélites (0F: 15). En azul, la constelación (00: GPS, 06: GLONASS). En verde, la señal (01: L1).

- **UBX-CFG-MSG**
  CFG-MSG - 06 01 08 00 02 15 00 01 00 01 00 00
  En rojo, el mensaje habilitado (02 15: UBX-RXM-RAWX).

Para el nodo se incrementó la cantidad de satélites al máximo permitido por el receptor (32 satélites) manteniéndose la restricción a las constelaciones de GPS y GLONASS. Para el cálculo de la posición se utilizan únicamente los satélites en común captados por base y nodo, los 32 satélites del nodo se configuraron en busca de aumentar la probabilidad de recibir en el nodo todos los satélites que recibe la base. Adicionalmente el receptor de la base decodifica los mensajes de navegación de todos los satélites, obteniendo así información de las órbitas y salud de los satélites, entre otros. La información de navegación se transmite en los mensajes UBX-RXM-SFRBX.

- **UBX-CFG-GNSS**
  CFG-GNSS - 06 3E 3C 00 00 20 20 07 00 00 20 00 01 00 01 01 01 00 00 00 00 00 00 01 02 00 00 00 00 00 00 01 03 00 00 00 00 00 00 01 04 00 00 00 00 00 00 03 05 00 00 00 00 00 00 05 06 00 20 00 01 00 01 01
  En rojo, el máximo número de satélites (20: 32). En azul, la constelación (00: GPS, 06: GLONASS). En verde, la señal (01: L1).

- **UBX-CFG-MSG**
  CFG-MSG - 06 01 08 00 02 15 00 01 00 01 00 00
  En rojo, el mensaje habilitado (02 15: UBX-RXM-RAWX).
  CFG-MSG - 06 01 08 00 02 13 00 01 00 01 00 00
  En rojo, el mensaje habilitado (02 13: UBX-RXM-SFRBX).

Para todos los casos se configuró una frecuencia de obtención de mensajes de 1 Hz.

- UBX-CFG-RATE
  CFG-RATE - 06 08 06 00 E8 03 01 00 01 00
  En rojo, el período de medición en ms (E8 03: 1000).

Para un ejemplo de archivo completo de configuración referirse al Apéndice A.4.

6.6. Configuración de RTKlib-demo5

En la base se utilizó STR2STR como una forma sencilla de obtener los mensajes del receptor. Se configuró como entrada el puerto serial correspondiente al USB donde estaba conectado el dongle con el NEO-M8T y como salida un puerto TCP. Desde este puerto un programa levanta la información y la transmite por la radio. También se guarda la entrada en un archivo de respaldo, para poder realizar cálculos en post-procesamiento.

En el nodo se utilizó RTKRCV, que levanta los datos del nodo desde un puerto serial y los de la base desde un puerto TCP. Una vez calculada la posición escribe el resultado en otro puerto TCP que se leerá para eventualmente mandar las coordenadas al servidor. Similar a lo que sucede con la base, se guarda un log de todos los archivos de entrada y salida.

A continuación se describen los principales parámetros configurados para el programa RTKRCV. Para ver la configuración completa referirse al Apéndice A.2.

- Tipo de cálculo a utilizar:
  pos1-posmode = static-start
  Se aprovecha del conocimiento de que el nodo esperará el primer fix para comenzar a moverse. Para ciertas aplicaciones puede no ser recomendado.

- Señales a utilizar:
  pos1-frequency = l1
  Se opta por trabajar únicamente con las señales L1.

- Constelaciones a utilizar:
  pos1-navsys = 5
  Se seleccionan las constelaciones de GPS y GLONASS.

- Método de resolución de la ambigüedad entera:
  pos2-armode = fix-and-hold
  Es un modo de seguimiento, recomendado para nodos en movimiento [72] [73].

- Formato de la solución:
  out-solformat = llh
  Latitud, longitud, altitud.

- Ubicación precisa de la Base:
  ant2-pos1 = -34.918143

ant2-pos2 = -56.166706
ant2-pos3 = 40
Cuanto menos precisos sean estos datos, más error tendrá la solución. Existe la posibilidad de calcular la posición del nodo relativo a la base, en cuyo caso no es necesaria tanta precisión en este parámetro. Para el proyecto se eligieron coordenadas aproximadas utilizando Google Maps. En el Capítulo 8 se discute brevemente sobre el impacto de la precisión de estas coordenadas en los resultados obtenidos.

- Configuración de entradas:
  inpstr1-type = serial
  inpstr1-path = ttyACM0:115200:8:n:1:off
  Dirección del puerto serial y especificaciones de la conexión.
  inpstr1-format = ubx
  inpstr2-type = tcpsvr
  inpstr2-path = 127.0.0.1:1230
  inpstr2-format = ubx

- Configuración de salidas:
  outstr1-type = file
  outstr1-path = OutNodoRTKPos.pos
  outstr1-format = llh
  outstr2-type = tcpcli
  outstr2-path = 127.0.0.1:1332
  outstr2-format = llh

- Configuración de logs:
  logstr1-type = file
  logstr1-path = OutNodoLogRover.ubx
  logstr2-type = file
  logstr2-path = OutNodoLogBase.ubx

Tanto receptor como biblioteca presentan una gran variedad de opciones al momento de la configuración. Hubo que configurar los receptores en su versión más minimalista debido a las restricciones introducidas por el resto del sistema, tanto de hardware como a causa del protocolo LoRa elegido para la transmisión de datos. Para la biblioteca se trabajó bajo las recomendaciones del blog especializado RTK Explorer [74], dejando abierta la posibilidad de realizar futuros estudios propios del impacto que distintas configuraciones podrían tener en la solución.

Capítulo 7

Integración LoRa/LoRaWAN - RTK

En los capítulos anteriores se hizo una descripción de cada una de las partes del proyecto de forma independiente. Luego de haber implementado las mismas es necesario integrarlas en un único sistema, logrando así el reporte de la posición calculada mediante RTK a través de la red LoRa/LoRaWAN.

Se describe la integración de la aplicación particular RTK en la red LoRaWAN ya implementada y los desafíos afrontados a la hora de crear una aplicación de este tipo en una red LoRaWAN. Por último, se presentan las limitaciones con las que cuenta el sistema y posibles mejoras.

7.1. Integración

Luego de que los sistemas descriptos en los capítulos anteriores estaban funcionando de manera independiente fue necesario integrarlos. Para lograr esta integración se utilizó el esquema de conexión mostrado en la Figura 7.1.

Como se adelantó en la Sección 4.3.1, el nodo utiliza un único hardware para la recepción de los datos de la estación base y para el envío de coordenadas al gateway. Para utilizar esta radio en forma compartida se utilizó un timer, administrando con él la cadencia con la que se reporta una coordenada mediante LoRaWAN.

El funcionamiento normal del sistema completo se puede resumir en los siguientes pasos, cada uno de ellos representados en la Figura 7.1:

1. La estación base recibe la señal satelital a transmitirle al nodo en un puerto serial. Esta señal es colocada por la librería RTKlib en un puerto TCP/IP para que sea leída desde la Raspberry Pi.

2. La estación base levanta la señal del puerto TCP/IP, fragmenta el paquete y se lo envía al nodo en fragmentos de hasta 255 bytes (debido a restricciones de hardware) utilizando la tecnología LoRa.

3. Cada vez que el nodo recibe un paquete de 255 bytes de la estación base lo coloca en un puerto TCP/IP para que la librería RTKlib lo levante.

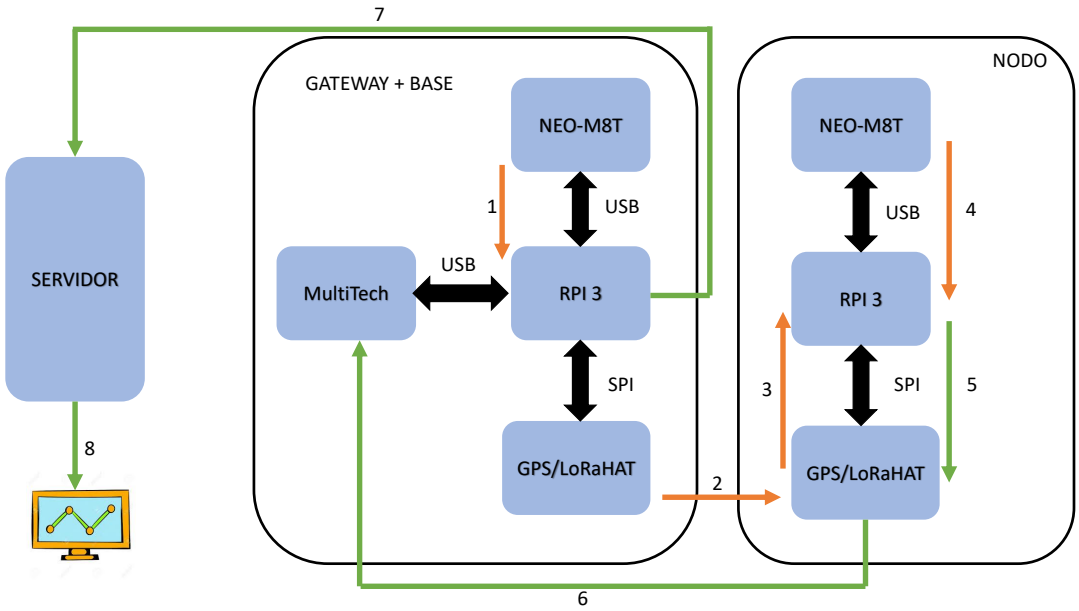


Figura 7.1: Sistema completo.

4. Además de la información recibida desde la estación base, se levanta de un puerto serial la señal satelital que el nodo recibe en su receptor de U-blox. Esta señal es leída por la Raspberry del nodo desde un puerto TCP/IP. Luego la RTKlib realiza los cálculos de la posición del nodo.

5. Las coordenadas calculadas son colocadas en otro puerto TCP/IP por la RTKlib para que sean leídos por el nodo.

6. Cuando el timer creado para enviar por LoRaWAN expira, el nodo se une a la red y envía las últimas coordenadas que calculó la RTKlib hacia el gateway, utilizando la radio con el protocolo LoRaWAN.

7. Mientras el gateway transmite las coordenadas recibidas por uno de sus receptores hacia el servidor utilizando una conexión UDP/IP sobre Ethernet, continúa recibiendo la señal satelital a través del puerto serial, volviendo al punto 1.

8. Un usuario final levanta las coordenadas utilizando MQTT, para publicarlas en un mapa. Este último punto se agregó en el proyecto como forma de verificar el correcto funcionamiento del sistema.

7.2. Inicialización

El programa implementado en el nodo tiene la característica de ser secuencial. Espera una paquete de la base, lo envía a la RTKlib y queda a la espera de una respuesta (un par de coordenadas). Esto presenta un inconveniente al inicializar el

sistema. La RTKlib necesita de un flujo constante de datos de entrada para poder estimar la posición la primera vez, por lo que hay que evitar quedarse a la espera de una respuesta que nunca llegará.

La solución a este problema se encontró en implementar un timer cuando se inicializa el sistema. Al comenzar, el sistema trabaja entre los puntos 1 y 4 del esquema de la Figura 7.1. Luego de transcurrido el tiempo del timer el sistema comienza a funcionar de forma normal.

7.3. Limitantes del Sistema

El sistema creado tiene limitaciones que fueron surgiendo a la hora de unificar la aplicación de RTK en una red LoRaWAN. A continuación se presentan estas limitantes y las soluciones planteadas. Además se comentan las posibles soluciones a implementar a futuro.

7.3.1. Una Radio en el Nodo

Como fue mencionado en capítulos anteriores, el nodo cuenta con una única radio para la recepción de datos de la estación base y el envío de las coordenadas al gateway. Esto generó que se tuviera que compartir la radio, utilizando un timer para enviar los datos por LoRaWAN cada 5 minutos. Es importante hacer notar que estos 5 minutos son un tiempo arbitrario, definido por el usuario final para el reporte de las coordenadas. Este tiempo define implícitamente cada cuanto se interrumpirá el flujo de datos de entrada a la RTKlib. Cuando esto ocurra, se provocará un decaimiento en la precisión de la solución.

En caso de buscar una aplicación donde las coordenadas sean reportadas a una mayor cadencia, podría pensarse en incluir un nuevo hardware, específico para la recepción de los datos de la estación base o el envío de datos por LoRaWAN. De esta forma se puede enviar datos por una radio, mientras la otra funciona como receptor. Aunque no hay que perder de vista que esto implicaría agregar un costo al nodo implementado.

7.3.2. Unión a la Red a Través de OTAA

Si bien la inicialización del nodo mediante OTAA facilitó la programación del nodo, esto genera que al momento de enviar las coordenadas a través de la red LoRaWAN exista un nuevo retardo, mientras el nodo y el servidor intercambian sus claves. Este retardo hace que se demore aún más en recibir información de la estación base y por lo tanto empeora el cálculo de la RTKlib.

Una solución a este problema sería programar en el nodo una inicialización ABP, lo que requeriría menos tiempo en el proceso de unión.

7.3.3. Coordenadas de la Estación Base

Si bien no es una limitante del sistema en sí, el no conocer las coordenadas exactas donde se encuentra la estación base agrega una cierta incertidumbre a la solución hallada por el sistema.

Al realizar las pruebas con el sistema creado se utilizan coordenadas de la estación base que no son precisas. Esto podría traer problemas a la hora de evaluar el sistema implementado. Poniendo la estación base en una posición donde son conocidas las coordenadas este problema puede eliminarse y deja de influir en los resultados.

En este capítulo se vio cómo se obtuvo el prototipo final a partir de las soluciones planteadas en los Capítulos 5 y 6, los problemas que surgieron a partir de la integración. Por ejemplo, los cuidados que se debieron tomar para inicializar el sistema y la implementación de un protocolo con el fin de compartir los recursos de hardware.

A su vez, se vieron las limitantes que tiene el prototipo al que se llegó, planteando alguna posible solución a implementar en trabajos futuros. Entre estas limitantes se encuentran la imposibilidad de reportar las coordenadas en forma fluida. Dependiendo de la aplicación que se le quiera dar al prototipo implementado esto puede ser más o menos perjudicial. Sin embargo, se considera que para una aplicación donde no se requiere un envío de datos constante el funcionamiento del prototipo es correcto.

Capítulo 8

Medidas de Desempeño y Análisis de Resultados

Para evaluar el desempeño del sistema propuesto, se realizó una comparación entre los resultados obtenidos con el mismo y los resultados obtenidos al calcular la posición con la librería de forma aislada. Esto quiere decir, tomando los archivos de entrada a la base y al nodo directamente y haciendo el cálculo en post-procesamiento, evitando así los errores introducidos por el sistema.

A lo largo del presente capítulo se detallan los resultados obtenidos en términos de precisión de las medidas así como también una muestra del funcionamiento de la red LoRaWAN.

8.1. Notas de Trabajo

En esta sección se pretende agrupar algunos conceptos para el mejor entendimiento de los resultados expuestos más adelante.

- Base y nodo guardan un respaldo de los datos obtenidos de sus respectivos receptores. El nodo guarda también la información que recibe desde la base a través de LoRa y las coordenadas calculadas. Todo esto permite el trabajo en post-procesamiento.
- Cada 5 minutos se realiza el envío de un par de coordenadas a través de la red LoRaWAN.
- Cuando se habla de medidas en tiempo real, se refiere a los cálculos realizados con el sistema implementado.
- Las medidas en post-procesamiento con el sistema, refieren a los resultados obtenidos al procesar ambos archivos de entrada al nodo. Esto es, los datos adquiridos de su receptor y los adquiridos por LoRa.
- Las medidas con post-procesamiento sin sistema, evalúan directamente la biblioteca RTKlib. Son aquellas obtenidas a partir de procesar los archivos de entrada a base y nodo obtenidos desde sus respectivos receptores.

- Se dice que una solución es precisa cuando no hay dispersión en las medidas.
- Se dice que una solución no es ambigua cuando se garantiza que el valor real se encuentra dentro de un cierto margen de error.
- Para realizar las diferentes pruebas se utilizó una grilla de 5 cm sobre la que se desplazó la antena del nodo. La grilla puede observarse en la Figura 8.1.
- Se hace referencia al posicionamiento por código sin técnicas de mejora como "método estándar" o "GPS común".

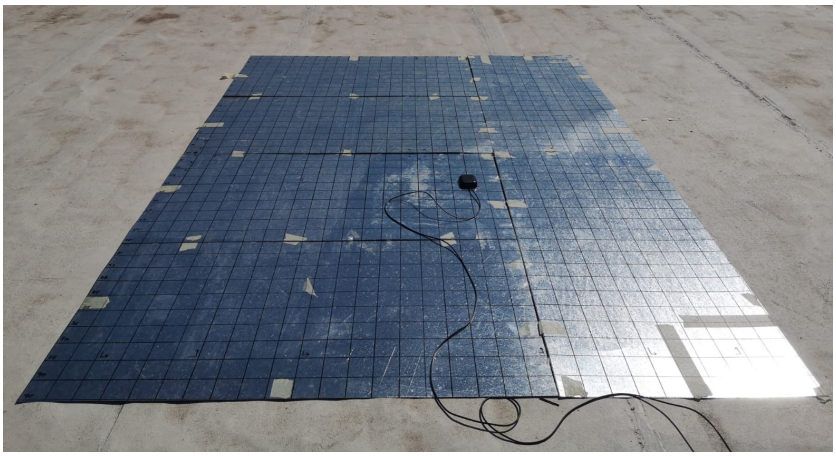


Figura 8.1: Grilla utilizada para las pruebas.

8.2. Verificación del Sistema Completo

El primer paso para probar el sistema fue constatar que las coordenadas reportadas por el nodo al servidor eran coherentes con el lugar en donde se encontraba el mismo. Para esto se desarrolló una aplicación, con el fin de levantar las coordenadas del nodo desde el servidor e imprimirlas sobre un mapa. En la Figura 8.2 se puede observar que las coordenadas reportadas por el nodo lo muestran en el techo de la Facultad de Ingeniería, lugar donde se realizaron todas las pruebas para este proyecto. Como fue mencionado anteriormente el envío de datos a través de la red LoRaWAN es cada 5 minutos, por lo tanto el reporte de la posición del nodo en el mapa es cada ese mismo período de tiempo.

En esta aplicación se puede constatar que las coordenadas son coherentes con el área de trabajo y que el sistema está funcionando correctamente. Sin embargo, es de interés poder medir y observar la mejoría de la precisión con respecto a un "GPS normal". Para poder evaluar lo anterior se precisa un mayor detalle en las medidas que el que se consigue al imprimir una medida sobre un mapa. Además, debido a que el reporte al servidor es cada 5 minutos, es difícil observar el camino recorrido si el nodo se encuentra en movimiento.

Figura 8.2: Ploteo de las coordenadas en un mapa.

Entonces, si bien el funcionamiento del sistema es correcto y las coordenadas enviadas por el nodo concuerdan con el espacio donde se trabajó, aún resta estudiar la mejora en la precisión debido a la técnica RTK. Para estudiar esta mejoría se plotearon las coordenadas calculadas utilizando las herramientas gráficas de RTKlib. Las coordenadas, actualizadas cada 1 segundo, se levantan del archivo guardado por la RTKlib en tiempo real.

8.3. Evaluación de la Precisión

Para evaluar la precisión que puede alcanzar la biblioteca al calcular la posición, se realizaron 3 pruebas. Éstas fueron cálculo en tiempo real, post-procesamiento con sistema y post-procesamiento sin sistema.

8.3.1. Influencia del Sistema en la Solución

El recorrido seguido para hacer el estudio de precisión se presenta en la Figura 8.3. Luego se obtiene y grafica la posición del nodo en el recorrido utilizando los tres métodos antes mencionados.

En las Figuras 8.4, 8.5 y 8.6 se reflejan las características generales de los resultados obtenidos. Se destaca una semejanza importante en las posiciones graficadas, aunque puede observarse cómo cualitativamente parece ser mejor la solución obtenida sin el sistema de por medio.

Para poder realizar un estudio cuantitativo del desempeño y analizar con mayor profundidad los resultados, se creó un script en Matlab encargado de encontrar, por sectores, la recta que mejor ajuste a los puntos. Un ejemplo de lo realizado puede verse en la Figura 8.7. Como el recorrido real no está georeferenciado, el

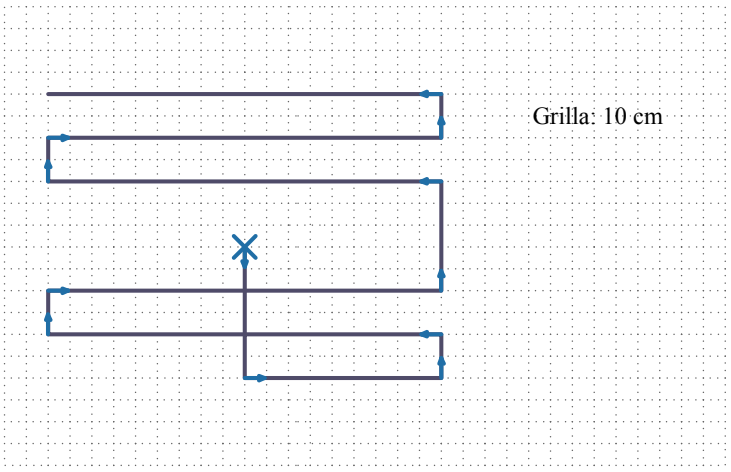


Figura 8.3: Camino recorrido.

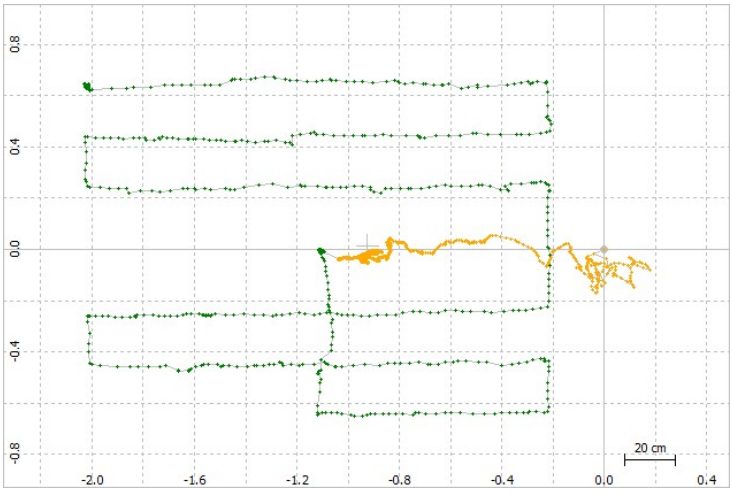


Figura 8.4: Sistema en tiempo real.

estudio posee ciertas carencias. En particular que los errores se miden respecto a la recta de mejor ajuste y no respecto al recorrido real.

Se calculó la distancia entre cada punto y la recta creada y se consideró ésta como el error. En la Tabla 8.1 se muestra el error cuadrático medio para varios conjuntos de pruebas, siempre realizando el mismo recorrido.

Estos resultados respaldan lo que se preveía del análisis cualitativo de las imágenes. Se refleja claramente que el sistema introduce un error en el desempeño de la biblioteca. Esto se debe a la interrupción en el flujo de datos de entrada al nodo, lo que sucede cuando se interrumpe la conexión con la base para enviar un par de coordenadas a la red LoRaWAN. Si se toma únicamente los casos que incluyen al sistema, puede notarse cómo la biblioteca tiene un mejor desempeño al trabajar en post-procesamiento en lugar de en tiempo real. De todas formas, los resultados obtenidos son satisfactorios y mejoran significativamente la precisión del método

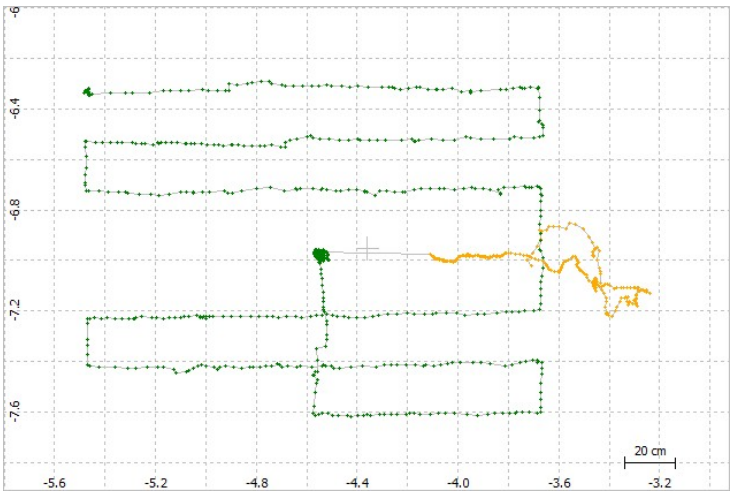


Figura 8.5: Post-procesamiento con sistema.

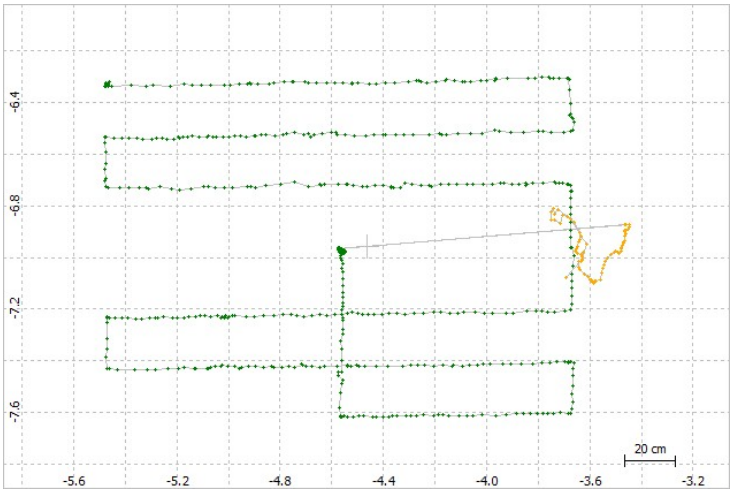


Figura 8.6: Post-procesamiento sin sistema.

estándar de posicionamiento.

8.3.2. Solución RTK vs Solución Estándar

Para continuar con el estudio de la precisión de la biblioteca y el sistema creado se realizó una comparación con un GPS normal. Para esta parte se dejó el nodo en una posición fija. Por un lado se procesaron los datos recibidos por el nodo con el receptor GNSS sin utilizar la información recibida de la base y por otro se relizó el cálculo de la posición en tiempo real utilizando el sistema creado. Los resultados de esta prueba pueden verse en la Figura 8.8 donde se ven las medidas representadas en una grilla de 20 cm de escala.

No es necesario un análisis profundo de los datos para observar la mejoría en la precisión del cálculo. Se puede ver a simple vista que mientras que las coordenadas

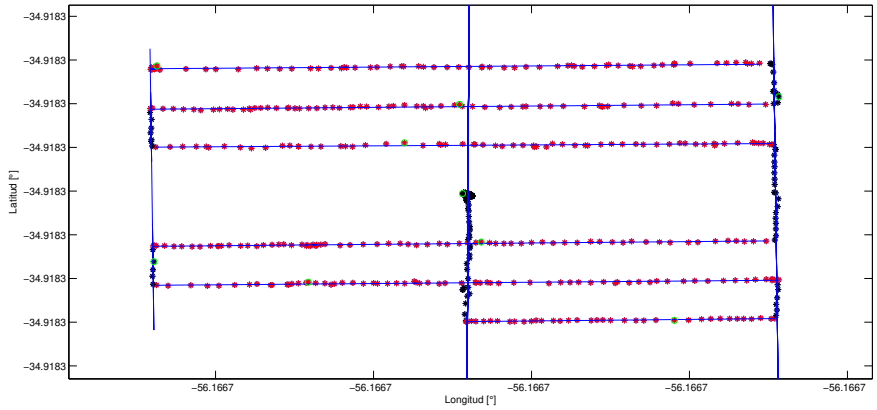


Figura 8.7: Puntos obtenidos y rectas aproximadas.

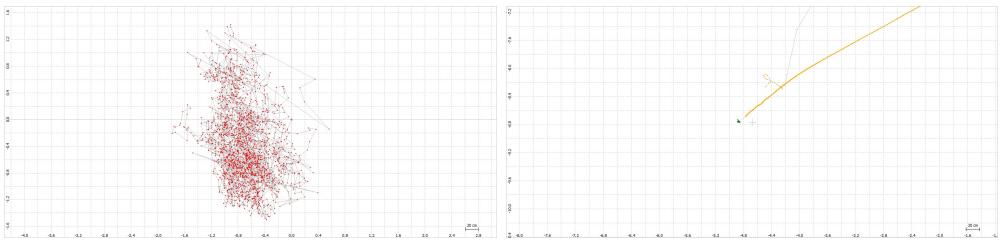
Prueba	Método	Error cuadrático medio (cm)
Prueba 1	Post-procesamiento sin sistema	0,3756
	Post-procesamiento con sistema	0,6447
	Sistema en tiempo real	1,2260
Prueba 2	Post-procesamiento sin sistema	0,3844
	Post-procesamiento con sistema	1,7072
	Sistema en tiempo real	2,5416
Prueba 3	Post-procesamiento sin sistema	0,8043
	Post-procesamiento con sistema	0,9276
	Sistema en tiempo real	1,1906

Tabla 8.1: Error cuadrático medio de los puntos obtenidos respecto a la recta.

calculadas por el GPS tienen una dispersión de unos 6 m^2 aproximadamente, lo obtenido por el sistema se encuentra contenido dentro de un cuadrado de 20 cm de lado. En la Figura 8.9 se muestra un acercamiento a la Figura 8.8.b. Allí se puede apreciar con más detalle la dispersión de los datos obtenidos con el sistema implementado, resultando que éstos se concentran en 4 cm^2.

Además de la dispersión de los datos, interesa estudiar con más detalle las coordenadas obtenidas. Fue con este fin que se desarrolló otro script de Matlab para calcular la media y desviación estándar. Se presentan los resultados en las Tablas 8.2 y 8.3.

Observando la desviación estándar en las pruebas realizadas se puede verificar lo visto en las imágenes. La desviación estándar en el caso de RTK fue 100 veces menor que en el caso de GPS lo que corrobora que la técnica RTK en notoriamente más precisa que la técnica estándar. Al realizar estas pruebas se esperaba que el valor de la media en el caso de lo obtenido por GPS y en lo obtenido por RTK fuera similar, sin embargo los resultados son diferentes. Se muestra de forma gráfica en



(a) Coordenadas medidas por GPS. (b) Coordenadas medidas por RTK.

Figura 8.8: Coordenadas de un punto fijo.

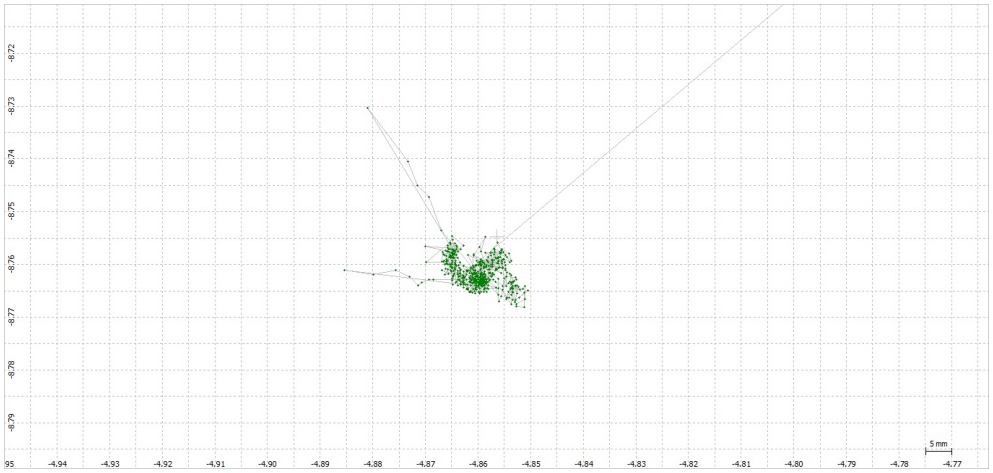


Figura 8.9: Acercamiento a las medidas RTK.

la Figura 8.10, donde se plotearon de forma simultánea ambos sets de datos.

Este error en las coordenadas calculadas para el nodo puede explicarse por la falta de precisión en las coordenadas que se fijaron para la estación base. Es en casos como éste que se refleja el hecho de que la técnica RTK es muy precisa pero también ambigua.

8.3.3. Evaluación de la Ambigüedad

Una vez realizado el estudio de la precisión del sistema, se analizó su ambigüedad. Se encontró que en el techo de la Facultad de Ingeniería existía un punto georeferenciado por el Servicio de Geomática de la Intendencia de Montevideo [75] [76]. Con este punto como referencia se cambiaron las coordenadas de la base a otras más certeras y se volvió a repetir el cálculo de la posición. En este caso las soluciones se superponen como se ilustra en la Figura 8.11 y se muestra en la Tabla 8.4.

La desviación estándar obtenida en este nuevo caso fue muy similar a la obtenida anteriormente con las viejas coordenadas fijas de la base (ver Tabla 8.5). Esto demuestra que la precisión es independiente de las coordenadas de la estación base.

Prueba	Método	Media latitud	Media longitud
Prueba 1	RTK	−34,9182419747259	−56,1667591885277
	GPS	−34,9182765877315	−56,1667247193537
Prueba 2	RTK	−34,9182244916516	−56,1667539283677
	GPS	−34,9182641145935	−56,1667267232258

Tabla 8.2: Media del conjunto de puntos obtenidos.

Prueba	Método	Desv. estándar latitud	Desv. estándar longitud
Prueba 1	RTK	$2,77107 \times 10^{-08}$	$4,59773 \times 10^{-08}$
	GPS	$3,98053 \times 10^{-06}$	$2,89440 \times 10^{-06}$
Prueba 2	RTK	$2,63062 \times 10^{-08}$	$3,69541 \times 10^{-08}$
	GPS	$4,71325 \times 10^{-06}$	$4,16639 \times 10^{-06}$

Tabla 8.3: Desviación estándar del conjunto de puntos obtenidos.

Para observar mejor la diferencia entre precisión y ambigüedad, se presenta la Figura 8.12. La misma es la representación del mismo camino recorrido, uno utilizando las coordenadas fijas de la estación base obtenidas mediante Google Maps y la otra con las coordenadas ajustadas con la georeferencia. Se puede ver que el camino recorrido es el mismo y tiene excelente precisión en ambos casos. Hay un gran corrimiento y esto tiene que ver con las distintas coordenadas asignadas a la base.

8.4. Robustez de la Biblioteca

Para finalizar se evaluó la robustez de esta biblioteca. Es decir, bajo qué condiciones la biblioteca puede trabajar de forma correcta y las medidas no se ven afectadas. Para esto se procedió a repetir un mismo recorrido limitando la cantidad de satélites observados (5, 10, 15, 32) y la frecuencia con la que se reciben los datos de los satélites en el receptor. De esta forma se puede evaluar hasta qué punto el sistema puede funcionar correctamente. En la Figura 8.13 se puede observar el camino recorrido para este caso de estudio.

Se realizaron dos tipos de evaluación. El primero fue observando el tiempo de fix (convergencia de la solución) necesario para cada caso y el segundo midiendo el error cuadrático medio respecto a las rectas de mejor ajuste de los datos.

8.4.1. Tiempo de Fix

Se define el tiempo de fix como el tiempo que demora la biblioteca en converger a una solución o resultado que se puede considerar "bueno o preciso". Se esperaba que el tiempo de fix disminuyera con el aumento en la cantidad de satélites y la frecuencia de recepción de datos.

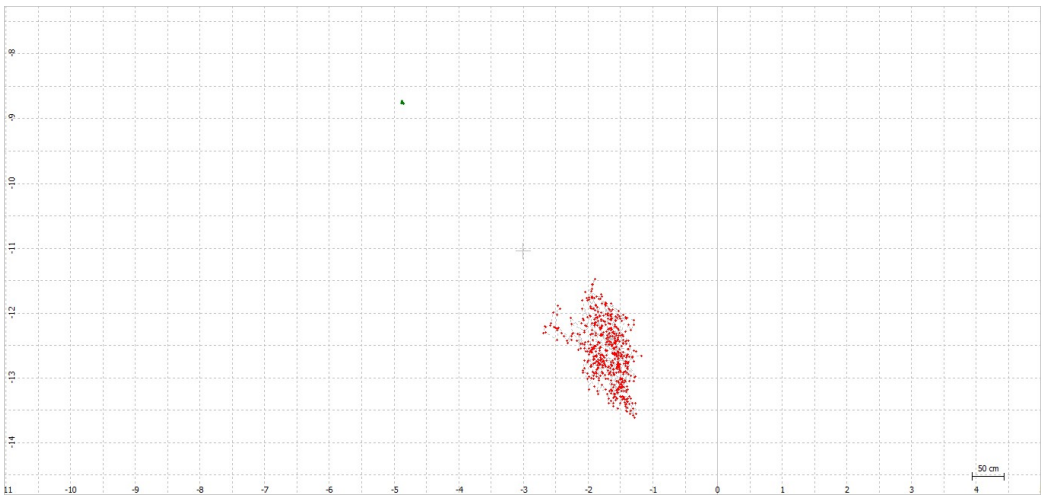


Figura 8.10: RTK vs GPS.

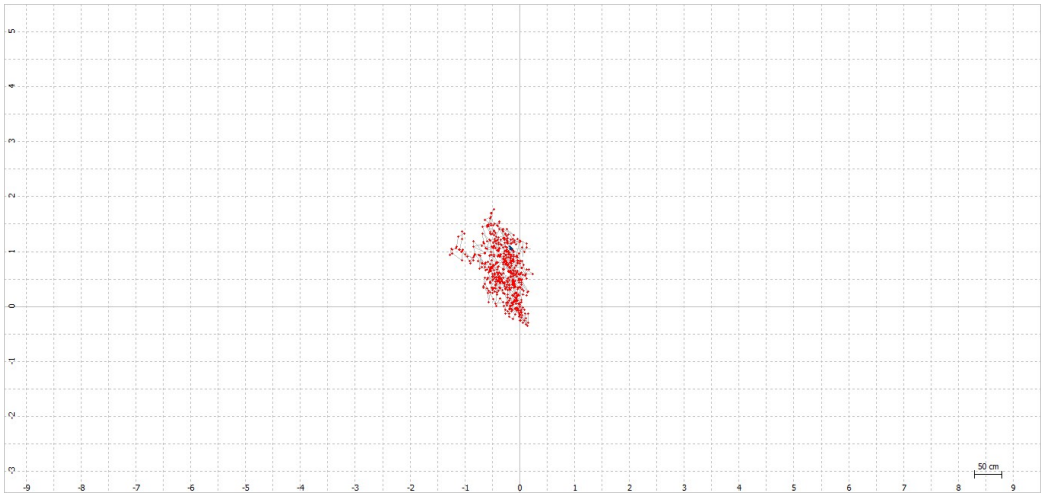


Figura 8.11: RTK vs GPS.

En la Tabla 8.6 pueden observarse los resultados obtenidos del tiempo de fix al variar la cantidad de satélites máxima permitida y la frecuencia de envío de datos. Cuando se indica *No calcula* significa que la biblioteca no es capaz de calcular la posición del nodo con la información que tiene de entrada, es decir, le falta información o la información es errónea. Si se indica *Float* significa que la solución no llegó a converger en el tiempo de duración de la prueba (25 minutos).

Se observó entonces que para 5 satélites no calcula para ninguna frecuencia de recepción. Esto se puede deber a que a la RTKlib no le alcanza la cantidad de información para poder calcular la posición con precisión. Por otro lado se vio que mientras aumenta la cantidad de satélites el tiempo para hacer fix es cada vez menor, lo que era esperable. Cuánto más información se reciba de los satélites, mejor será la precisión del resultado y más rápido se podrá calcular.

Finalmente, al mirar el tiempo de fix respecto al cambio en la frecuencia de

Prueba	Método	Media latitud	Media longitud
Prueba 1	RTK antes	−34,9182419747259	−56,1667591885277
	RTK	−34,9182729747445	−56,1667231885185
	GPS	−34,9182765877315	−56,1667247193537
Prueba 2	RTK antes	−34,9182244916516	−56,1667539283677
	RTK	−34,9182554917806	−56,1667179282193
	GPS	−34,9182641145935	−56,1667267232258

Tabla 8.4: Media del conjunto de puntos obtenidos.

Prueba	Método	Desv. estándar latitud	Desv. estándar longitud
Prueba 1	RTK antes	$2,77107 \times 10^{-08}$	$4,59773 \times 10^{-08}$
	RTK	$2{,}6293 \times 10^{-08}$	$3{,}6951 \times 10^{-08}$
	GPS	$3,98053 \times 10^{-06}$	$2,89440 \times 10^{-06}$
Prueba 2	RTK antes	$2,63062 \times 10^{-08}$	$3,69541 \times 10^{-08}$
	RTK	$2,13113 \times 10^{-08}$	$4,28896 \times 10^{-08}$
	GPS	$4,71325 \times 10^{-06}$	$4,16639 \times 10^{-06}$

Tabla 8.5: Desviación estándar del conjunto de puntos obtenidos.

recepción de datos, la relación no fue tan clara como se esperaba. En el caso de 32 satélites se puede ver que cuanto mayor fue la frecuencia de envío, el tiempo en hacer fix fue menor. Pero en el resto de los casos esto no se cumple.

8.4.2. Error

Para finalizar el estudio, se ajustaron los datos por rectas y se calculó el error cuadrático medio respecto a ellas. Como ya fue mencionado, este error no es exactamente el error real dado que no se tienen valores certeros de los caminos recorridos. En la Tabla 8.7 se pueden observar los resultados obtenidos.

Estos resultados hacen pensar que no hay una relación entre la precisión de la solución y la cantidad de satélites o frecuencia de datos que se tiene al momento de realizar el cálculo. No siguen ningún patrón. Ni al mantener la frecuencia y variar los satélites, ni al mantener los satélites y variar la frecuencia.

A modo de síntesis, en este capítulo se verificó que la técnica RTK alcanza una precisión centimétrica y se contrastó con la precisión de la técnica estándar de posicionamiento. Se observó una mejora de dos órdenes de magnitud en la desviación estándar de las coordenadas calculadas al trabajar con RTK. También se verificó que las coordenadas obtenidas dependen en gran medida de las coordenadas elegidas para la base. Si no se posee un punto georeferenciado, es mejor trabajar en términos de distancias relativas del nodo a la base para evitar errores.

Como un estudio adicional, se analizó cómo afectan en el cálculo de la posición la cantidad de información que recibe el nodo y la velocidad con la que la recibe.

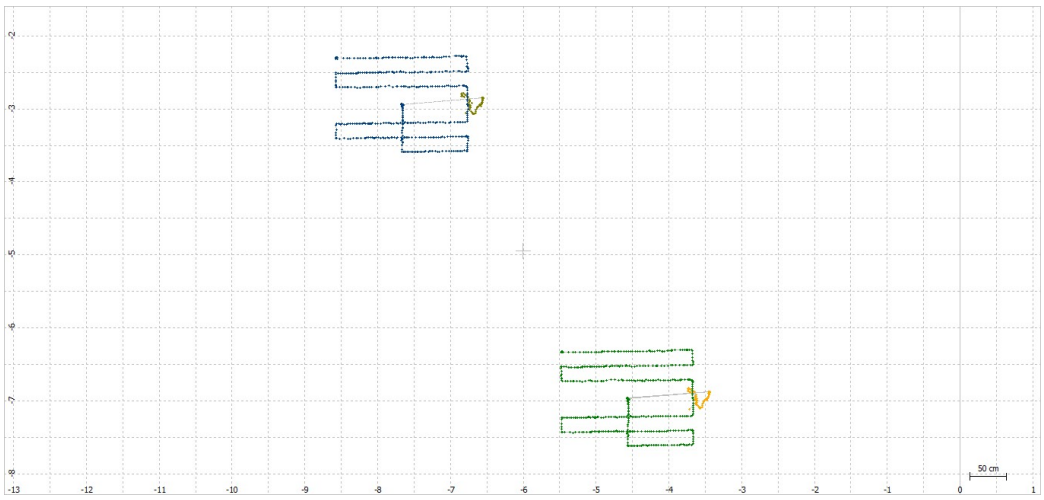


Figura 8.12: Variación de coordenadas de la base.

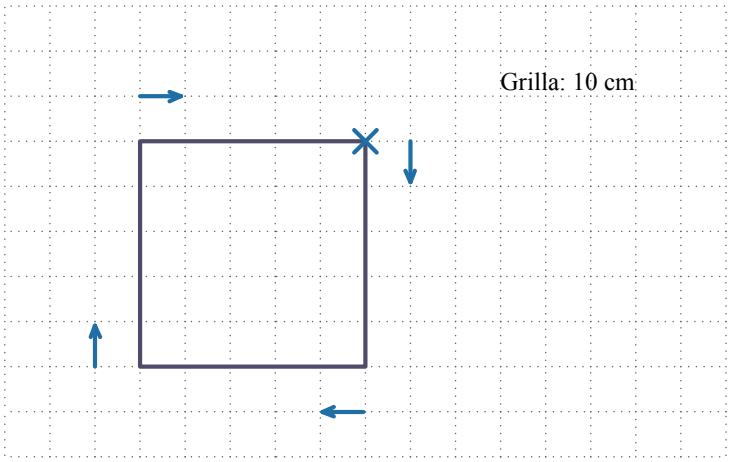


Figura 8.13: Camino recorrido al variar frecuencia y máximo de satélites.

Para esto se varió la cantidad de satélites que se permiten recibir y la frecuencia con la que la estación base reporta las correcciones al nodo. En este caso se pudo ver que no hay una dependencia clara entre estos dos parámetros y la precisión en las medidas. En principio parecería ser que el tiempo de convergencia de la solución disminuye al aumentar la cantidad de satélites utilizados, pero los resultados no son determinantes y deberían realizarse pruebas adicionales.

Frecuencia	**Máximo de Satélites**	**Tiempo de fix (minutos)**
0,5 s	5	*No calcula*
	10	*Float*
	15	16, 26
	32	0, 51
1 s	5	*No calcula*
	10	23, 25
	15	15, 17
	32	1, 51
2 s	5	*No calcula*
	10	14, 30
	15	*Float*
	32	3, 12

Tabla 8.6: Variación de frecuencia y máximo de satélites.

Frecuencia	**Máximo de Satélites**	**Error cuadrático medio (cm)**
0,5 s	5	*No calcula*
	10	*Float*
	15	0,7522
	32	1,5146
1 s	5	*No calcula*
	10	0,5447
	15	0,5623
	32	0,3288
2 s	5	*No calcula*
	10	0,4294
	15	*Float*
	32	2,5934

Tabla 8.7: Distancia máxima entre puntos y recta de mejor ajuste.

Capítulo 9

Conclusiones y Trabajo a Futuro

En primer lugar, se puede decir que los objetivos del proyecto fueron cumplidos satisfactoriamente. Se creó un prototipo con la capacidad de obtener su posición con una precisión de centímetros y reportarla a un servidor para su posterior uso en cualquier aplicación. Además, se pudo instalar una red LoRa/LoRaWAN completa, servidor-gateway-nodo, que puede utilizarse para diversos fines.

Para cumplir los diferentes objetivos se debió comenzar estudiando los fundamentos y tecnologías involucrados en el sistema. Se realizó un estudio sobre la tecnología de comunicación LoRa/LoRaWAN y los fundamentos de las técnicas de posicionamiento vía satelital, haciendo énfasis en RTK.

Se aprendió sobre la instalación de una red LoRa/LoRaWAN en su totalidad. Desde la instalación de un servidor de LoRaWAN, configuración de los nodos e implementación de un gateway, hasta el estudio de los protocolos involucrados.

Se desarrolló un dispositivo que, utilizando la comunicación LoRa para recibir correcciones de la estación base, puede realizar el cálculo de su posición utilizando la técnica de RTK. Luego se analizaron sus diferentes limitantes, precisión y cómo podría mejorarse el sistema.

La utilización de un único módulo LoRa para el nodo marcó el funcionamiento del sistema en general ya que la interfaz LoRa debía alternar entre recibir datos de la estación base y enviar sus coordenadas al gateway. Esto condiciona la cadencia en el envío de datos hacia el servidor. En caso que la aplicación final no requiera un reporte de datos continuo alcanzaría con el prototipo al que se llegó. De ser necesario un mayor reporte de datos de la posición del nodo podría agregarse una segunda radio.

Se pudo estudiar la utilización de la capa física de LoRa para el envío de los datos de la estación base al nodo. Esta modulación permite una gran distancia para la comunicación pero la tasa de envío de datos es muy baja. Sin embargo, utilizar esta modulación no afecta de forma crítica los resultados en las medidas.

En cuanto a la precisión obtenida con el sistema se puede decir que se logró el objetivo de tener una posición mucho más precisa a la obtenida por un GPS estándar, mejorando en dos ordenes de magnitud la desviación estándar de las coordenadas calculadas.

A modo de conclusión final, este proyecto plantea los pasos a seguir para ob-

tener un sistema de posicionamiento satelital con una precisión en el cálculo de la posición de 2,5 cm aproximadamente. Se destaca además el bajo costo en referencia a otras opciones del mercado, en el entorno del 20 %. En relación al tiempo disponible para la realización del proyecto se considera que el prototipo final es satisfactorio y se entiende que se podría seguir trabajando sobre el mismo para obtener un producto comercial.

Trabajo a Futuro

Un primer paso sería medir el consumo que tiene el nodo para evaluar la autonomía que puede alcanzar. Por otro lado, se plantea estudiar la distancia máxima que puede existir entre el nodo y la estación base antes de que la comunicación entre ambos a través de LoRa comience a fallar.

En el prototipo implementado el reporte de datos a la red es cada 5 minutos. Se plantea analizar con más detalle el efecto que la cadencia de envío de datos por LoRaWAN tiene en la precisión obtenida.

La biblioteca RTKlib tiene una gran variedad de parámetros configurables, muchos de los cuales no se modificaron. Sería interesante realizar un mayor estudio de cómo afectan dichos parámetros a los resultados obtenidos por el prototipo realizado.

Apéndice A

Apéndice

A.1. Apéndice I - Network Server

A.1.1. Instalación

El proceso de instalación del Network Server LoRaWAN para un Servidor con sistema opertativo Ubuntu puede encontrarse completo en el siguiente link: `https://www.loraserver.io/guides/debian-ubuntu/`.

A.1.2. Archivos de Configuración Network Server

loraserver.toml

```
[general]
# Log level
#
# debug=5, info=4, warning=3, error=2, fatal=1, panic=0
log_level=4


# PostgreSQL settings.
#
# Please note that PostgreSQL 9.5+ is required.
[postgresql]
# PostgreSQL dsn (e.g.: postgres://user:password@hostname/
# database?sslmode=disable).
#
# Besides using an URL (e.g. 'postgres://user:password@hostname/
# database?sslmode=disable')
# it is also possible to use the following format:
# 'user=loraserver dbname=loraserver sslmode=disable'.
#
# The following connection parameters are supported:
```

```
#
# * dbname - The name of the database to connect to
# * user - The user to sign in as
# * password - The user's password
# * host - The host to connect to. Values that start with / are for
# unix domain sockets. (default is localhost)
# * port - The port to bind to. (default is 5432)
# * sslmode - Whether or not to use SSL (default is require, this is
# not the default for libpq)
# * fallback_application_name - An application_name to fall back to
# if one isn't provided.
# * connect_timeout - Maximum wait for connection, in seconds. Zero
# or not specified means wait indefinitely.
# * sslcert - Cert file location. The file must contain PEM encoded
# data.
# * sslkey - Key file location. The file must contain PEM encoded
# data.
# * sslrootcert - The location of the root certificate file.
# The file must contain PEM encoded data.
#
# Valid values for sslmode are:
#
# * disable - No SSL
# * require - Always SSL (skip verification)
# * verify-ca - Always SSL (verify that the certificate presented by
# the server was signed by a trusted CA)
# * verify-full - Always SSL (verify that the certification presented
# by the server was signed by a trusted CA and the server host name
# matches the one in the certificate)
dsn="postgres://parriot_server:Motosierra123@localhost/parriot_server?sslmode=disable"

# Automatically apply database migrations.
#
# It is possible to apply the database-migrations by hand
# (see https://github.com/brocaar/loraserver/tree/master/migrations)
# or let LoRa App Server migrate to the latest state automatically,
# by using this setting. Make sure that you always make a backup when
# upgrading Lora
# App Server and / or applying migrations.
automigrate=true


# Redis settings
#
# Please note that Redis 2.6.0+ is required.
```

```
[redis]
# Redis url (e.g. redis://user:password@hostname/0)
#
# For more information about the Redis URL format, see:
# https://www.iana.org/assignments/uri-schemes/prov/redis
url="redis://localhost:6379"

# Max idle connections in the pool.
max_idle=10

# Idle timeout.
#
# Close connections after remaining idle for this duration. If the
# value is zero, then idle connections are not closed. You should set
# the timeout to a value less than the server's timeout.
idle_timeout="5m0s"


# Network-server settings.
[network_server]
# Network identifier (NetID, 3 bytes) encoded as HEX (e.g. 010203)
net_id="000000"

# Time to wait for uplink de-duplication.
#
# This is the time that LoRa Server will wait for other gateways to receive
# the same uplink frame. Valid units are 'ms' or 's'.
# Please note that this value has influence on the uplink / downlink
# roundtrip time. Setting this value too high means LoRa Server will
# be unable to respond to the device within its receive-window.
# deduplication_delay="200ms"

# Device session expiration.
#
# The TTL value defines the time after which a device-session expires
# after no activity. Valid units are 'ms', 's', 'm', 'h'. Note that these
# values can be combined, e.g. '24h30m15s'.
device_session_ttl="744h0m0s"

# Get downlink data delay.
#
# This is the time that LoRa Server waits between forwarding data to
# the application-server and reading data from the queue. A higher
# value means that the application-server has more time to schedule a
# downlink queue item which can be processed within the same uplink /
```

```
# downlink transaction.
# Please note that this value has influence on the uplink / downlink
# roundtrip time. Setting this value too high means LoRa Server will
# be unable to respond to the device within its receive-window.
get_downlink_data_delay="100ms"


  # LoRaWAN regional band configuration.
  #
  # Note that you might want to consult the LoRaWAN Regional
  # Parameters specification for valid values that apply to your
  # region.
  # See: https://www.lora-alliance.org/lorawan-for-developers
  [network_server.band]
  # LoRaWAN band to use.
  #
  # Valid values are:
  # * AS_923
  # * AU_915_928
  # * CN_470_510
  # * CN_779_787
  # * EU_433
  # * EU_863_870
  # * IN_865_867
  # * KR_920_923
  # * RU_864_870
  # * US_902_928
  name="AU_915_928"

  # Enforce 400ms dwell time
  #
  # Some band configurations define the max payload size for both dwell-time
  # limitation enabled as disabled (e.g. AS 923). In this case the
  # dwell time setting must be set to enforce the max payload size
  # given the dwell-time limitation. For band configuration where the dwell-time is
  # always enforced, setting this flag is not required.
  dwell_time_400ms=false

  # Enforce repeater compatibility
  #
  # Most band configurations define the max payload size for both an optional
  # repeater encapsulation layer as for setups where a repeater will never
  # be used. The latter case increases the max payload size for some data-rates.
  # In case a repeater might used, set this flag to true.
  repeater_compatible=false
```

```
# LoRaWAN network related settings.
[network_server.network_settings]
# Installation margin (dB) used by the ADR engine.
#
# A higher number means that the network-server will keep more
# margin, resulting in a lower data-rate but decreasing the chance
# that the device gets disconnected because it is unable to reach one
# of the surrounded gateways.
installation_margin=10

# RX window (Class-A).
#
# Set this to:
# 0: RX1, fallback to RX2 (on RX1 scheduling error)
# 1: RX1 only
# 2: RX2 only
rx_window=0

# Class A RX1 delay
#
# 0=1sec, 1=1sec, ... 15=15sec. A higher value means LoRa Server
# has more time to respond to the device as the delay between the
#uplink and the first receive-window will be increased.
rx1_delay=1

# RX1 data-rate offset
#
# Please consult the LoRaWAN Regional Parameters specification for
# valid
# options of the configured network_server.band.name.
rx1_dr_offset=0

# RX2 data-rate
#
# When set to -1, the default RX2 data-rate will be used for the
# configured
# LoRaWAN band.
#
# Please consult the LoRaWAN Regional Parameters specification for
# valid
# options of the configured network_server.band.name.
rx2_dr=-1
```

```
  # RX2 frequency
  #
  # When set to -1, the default RX2 frequency will be used.
  #
  # Please consult the LoRaWAN Regional Parameters specification for
  # valid
  # options of the configured network_server.band.name.
  rx2_frequency=-1

  # Downlink TX Power (dBm)
  #
  # When set to -1, the downlink TX Power from the configured band
  # will be used.
  #
  # Please consult the LoRaWAN Regional Parameters and local
  # regulations for valid and legal options. Note that the configured
  # TX Power must be supported by your gateway(s).
  downlink_tx_power=-1

  # Disable mac-commands
  #
  # When set, uplink mac-commands are ignored and the network-server will not
  # send mac-commands to the devices. This is intended for testing
  # only.
  disable_mac_commands=true

  # Disable ADR
  #
  # When set, this globally disables ADR.
  disable_adr=false

  # Enable only a given sub-set of channels
  #
  # Use this when ony a sub-set of the by default enabled channels
  # are being
  # used. For example when only using the first 8 channels of the US
  # band.
  # Note: when left blank, all channels will be enabled.
  #
  # Example:
  # enabled_uplink_channels=[0, 1, 2, 3, 4, 5, 6, 7]
  enabled_uplink_channels=[8,9,10,11,12,13,14,15]


  # Extra channel configuration.
```

```
#
# Use this for LoRaWAN regions where it is possible to extend the by
# default
# available channels with additional channels (e.g. the EU band).
# The first 5 channels will be configured as part of the OTAA
# join-response
# (using the CFList field).
# The other channels (or channel / data-rate changes) will be (re)
# configured
# using the NewChannelReq mac-command.
#
# Example:
#[[network_server.network_settings.extra_channels]]
#frequency=923300000
#min_dr=0
#max_dr=5

#[[network_server.network_settings.extra_channels]]
#frequency=923900000
#min_dr=0
#max_dr=5

#[[network_server.network_settings.extra_channels]]
#frequency=924500000
#min_dr=0
#max_dr=5

#[[network_server.network_settings.extra_channels]]
#frequency=925100000
#min_dr=0
#max_dr=5

#[[network_server.network_settings.extra_channels]]
#frequency=925700000
#min_dr=0
#max_dr=5


# Class B settings
[network_server.network_settings.class_b]
# Ping-slot data-rate.
ping_slot_dr=0

# Ping-slot frequency (Hz)
#
```

```
  # Set this to 0 to use the default frequency plan for the
  # configured region
  # (which could be frequency hopping).
  ping_slot_frequency=0


  # Rejoin-request settings
  #
  # When enabled, LoRa Server will request the device to send a
  # rejoin-request
  # every time when one of the 2 conditions below is met (frame count
  # or time).
  [network_server.network_settings.rejoin_request]
  # Request device to periodically send rejoin-requests
  enabled=false

  # The device must send a rejoin-request type 0 at least every
  # 2^(max_count_n + 4)
  # uplink messages. Valid values are 0 to 15.
  max_count_n=0

  # The device must send a rejoin-request type 0 at least every
  # 2^(max_time_n + 10) seconds. Valid values are 0 to 15.
  #
  # 0  = roughly 17 minutes
  # 15 = about 1 year
  max_time_n=0


  # Scheduler settings
  #
  # These settings affect the multicast, Class-B and Class-C downlink
  # queue
  # scheduler.
  [network_server.scheduler]
  # Scheduler interval
  #
  # The interval in which the downlink scheduler for multicast,
  # Class-B and Class-C runs.
  scheduler_interval="1s"

    # Class-C settings.
    [network_server.scheduler.class_c]
    # Downlink lock duration
    #
```

```
  # Contains the duration to lock the downlink Class-C transmissions
  # after a preceeding downlink tx (per device).
  downlink_lock_duration="2s"


# Network-server API
#
# This is the network-server API that is used by LoRa App Server or
# other
# custom components interacting with LoRa Server.
[network_server.api]
# ip:port to bind the api server
bind="0.0.0.0:8002"

# ca certificate used by the api server (optional)
ca_cert=""

# tls certificate used by the api server (optional)
tls_cert=""

# tls key used by the api server (optional)
tls_key=""


# Gateway statistics settings.
[network_server.gateway.stats]
# Create non-existing gateways on receiving of stats
#
# When set to true, LoRa Server will create the gateway when it
# receives
# statistics for a gateway that does not yet exist.
create_gateway_on_stats=true

# Aggregation timezone
#
# This timezone is used for correctly aggregating the statistics
# (for example 'Europe/Amsterdam').
# To get the list of supported timezones by your PostgreSQL database,
# execute the following SQL query:
# select * from pg_timezone_names;
# When left blank, the default timezone of your database will be used.
timezone=""

# Aggregation intervals to use for aggregating the gateway stats
#
```

```
  # Valid options: second, minute, hour, day, week, month, quarter,
  # year.
  # When left empty, no statistics will be stored in the database.
  # Note, LoRa App Server expects at least "minute", "day", "hour"!
  # aggregation_intervals=["minute", "hour", "day"]


  # Backend defines the gateway backend settings.
  #
  # The gateway backend handles the communication with the gateway(s)
  # part of the LoRaWAN network.
  [network_server.gateway.backend]
    # Backend
    #
    # This defines the backend to use for the communication with the
    # gateways.
    # Use the section name of one of the following gateway backends.
    # E.g. "mqtt" or "gcp_pub_sub".b
    type="mqtt"


    # MQTT gateway backend settings.
    #
    # This is the backend communicating with the LoRa gateways over a
    # MQTT broker.
    [network_server.gateway.backend.mqtt]
    # MQTT topic templates for the different MQTT topics.
    #
    # The meaning of these topics are documented at:
    # https://www.loraserver.io/lora-gateway-bridge/use/data/
    #
    # The default values match the default expected configuration of the
    # LoRa Gateway Bridge MQTT backend. Therefore only change these values when
    # absolutely needed.
    # Use "{{ .MAC }}" as an substitution for the LoRa gateway MAC.
    uplink_topic_template="gateway/+/rx"
    downlink_topic_template="gateway/{{ .MAC }}/tx"
    stats_topic_template="gateway/+/stats"
    ack_topic_template="gateway/+/ack"
    config_topic_template="gateway/{{ .MAC }}/config"

    # MQTT server (e.g. scheme://host:port where scheme is tcp, ssl
    # or ws)
    server="tcp://localhost:1883"
```

```
# Connect with the given username (optional)
username=""

# Connect with the given password (optional)
password=""

# Quality of service level
#
# 0: at most once
# 1: at least once
# 2: exactly once
#
# Note: an increase of this value will decrease the performance.
# For more information: https://www.hivemq.com/blog/
# mqtt-essentials-part-6-mqtt-quality-of-service-levels
qos=0

# Clean session
#
# Set the "clean session" flag in the connect message when this
# client
# connects to an MQTT broker. By setting this flag you are
# indicating
# that no messages saved by the broker for this client should be
# delivered.
clean_session=true

# Client ID
#
# Set the client id to be used by this client when connecting to
# the MQTT
# broker. A client id must be no longer than 23 characters. When
# left blank,
# a random id will be generated. This requires clean_session=true.
client_id=""

# CA certificate file (optional)
#
# Use this when setting up a secure connection (when server uses
# ssl://...)
# but the certificate used by the server is not trusted by any CA
# certificate
# on the server (e.g. when self generated).
ca_cert=""
```

```
    # TLS certificate file (optional)
    tls_cert=""

    # TLS key file (optional)
    tls_key=""


    # Google Cloud Pub/Sub backend.
    #
    # Use this backend when the LoRa Gateway Bridge is configured to connect
    # to the Google Cloud IoT Core MQTT broker (which integrates with
    # Pub/Sub).
    [network_server.gateway.backend.gcp_pub_sub]
    # Path to the IAM service-account credentials file.
    #
    # Note: this service-account must have the following Pub/Sub
    # roles: * Pub/Sub Editor
    credentials_file=""

    # Google Cloud project id.
    project_id=""

    # Uplink Pub/Sub topic name (to which Cloud IoT Core publishes).
    uplink_topic_name=""

    # Downlink Pub/Sub topic name (for publishing downlink frames).
    downlink_topic_name=""

    # Uplink retention duration.
    #
    # The retention duration that LoRa Server will set on the uplink
    # subscription.
    uplink_retention_duration="24h0m0s"


  # Geolocation settings.
  #
  # When set, LoRa Server will use the configured geolocation server
  # to resolve the location of the devices.
  [geolocation_server]
  # Server.
  #
  # The hostname:ip of the geolocation service (optional).
  server=""
```

```
  # CA certificate used by the API client (optional).
  ca_cert=""

  # TLS certificate used by the API client (optional).
  tls_cert=""

  # TLS key used by the API client (optional).
  tls_key=""


# Default join-server settings.
[join_server.default]
# hostname:port of the default join-server
server="http://localhost:8004"

# ca certificate used by the default join-server client (optional)
ca_cert=""

# tls certificate used by the default join-server client (optional)
tls_cert=""

# tls key used by the default join-server client (optional)
tls_key=""


# Join-server KEK set.
#
# These KEKs (Key Encryption Keys) are used to decrypt the network
# related
# session-keys received from the join-server on a (re)join-accept.
# Please refer to the LoRaWAN Backend Interface specification
# 'Key Transport Security' section for more information.
#
# Example (the [[join_server.kek.set]] can be repeated):
# [[join_server.kek.set]]
# # KEK label.
# label="000000"

# # Key Encryption Key.
# kek="0102030405060708010203040506070 8"


# Network-controller configuration.
[network_controller]
# hostname:port of the network-controller api server (optional)
```

```
server=""

# ca certificate used by the network-controller client (optional)
ca_cert=""

# tls certificate used by the network-controller client (optional)
tls_cert=""

# tls key used by the network-controller client (optional)
tls_key=""
```

lora-app-server.toml

```
[general]
# Log level
#
# debug=5, info=4, warning=3, error=2, fatal=1, panic=0
log_level=4

# The number of times passwords must be hashed. A higher number is safer as
# an attack takes more time to perform.
password_hash_iterations=100000


# PostgreSQL settings.
#
# Please note that PostgreSQL 9.5+ is required.
[postgresql]
# PostgreSQL dsn (e.g.: postgres://user:password@hostname/
# database?sslmode=disable).
#
# Besides using an URL (e.g. 'postgres://user:password@hostname/
# database?sslmode=disable')
# it is also possible to use the following format:
# 'user=loraserver dbname=loraserver sslmode=disable'.
#
# The following connection parameters are supported:
#
# * dbname - The name of the database to connect to
# * user - The user to sign in as
# * password - The user's password
# * host - The host to connect to. Values that start with / are for
# unix domain sockets. (default is localhost)
# * port - The port to bind to. (default is 5432)
# * sslmode - Whether or not to use SSL (default is require, this is
# not the default for libpq)
```

```
# * fallback_application_name - An application_name to fall back to
# if one isn't provided.
# * connect_timeout - Maximum wait for connection, in seconds. Zero
# or not specified means wait indefinitely.
# * sslcert - Cert file location. The file must contain PEM encoded
# data.
# * sslkey - Key file location. The file must contain PEM encoded
# data.
# * sslrootcert - The location of the root certificate file. The file
# must contain PEM encoded data.
#
# Valid values for sslmode are:
#
# * disable - No SSL
# * require - Always SSL (skip verification)
# * verify-ca - Always SSL (verify that the certificate presented by
# the server was signed by a trusted CA)
# * verify-full - Always SSL (verify that the certification presented
# by the server was signed by a trusted CA and the server host name
# matches the one in the certificate)
dsn="postgres://parriot_app:Motosierra123@localhost/
parriot_app?sslmode=disable"

# Automatically apply database migrations.
#
# It is possible to apply the database-migrations by hand
# (see https://github.com/brocaar/lora-app-server/tree/master/
# migrations)
# or let LoRa App Server migrate to the latest state automatically,
# by using
# this setting. Make sure that you always make a backup when
# upgrading Lora
# App Server and / or applying migrations.
automigrate=true


# Redis settings
#
# Please note that Redis 2.6.0+ is required.
[redis]
# Redis url (e.g. redis://user:password@hostname/0)
#
# For more information about the Redis URL format, see:
# https://www.iana.org/assignments/uri-schemes/prov/redis
url="redis://localhost:6379"
```

```
# Max idle connections in the pool.
max_idle=10

# Idle timeout.
#
# Close connections after remaining idle for this duration. If the
# value is zero, then idle connections are not closed. You should set
# the timeout to a value less than the server's timeout.
idle_timeout="5m0s"


# Application-server settings.
[application_server]
# Application-server identifier.
#
# Random UUID defining the id of the application-server installation
# (used by LoRa Server as routing-profile id).
# For now it is recommended to not change this id.
id="6d5db27e-4ce2-4b2b-b5d7-91f069397978"


  # Integration configures the data integration.
  #
  # This is the data integration which is available for all
  # applications,
  # besides the extra integrations that can be added on a
  # per-application
  # basis.
  [application_server.integration]
  # The integration backend.
  #
  # This defines the backend to use for the data integration. Use the
  # section
  # name of one of the following integration backend.
  # E.g. "mqtt" or "gcp_pub_sub".
  backend="mqtt"


  # MQTT integration backend.
  [application_server.integration.mqtt]
  # MQTT topic templates for the different MQTT topics.
  #
  # The meaning of these topics are documented at:
  # https://www.loraserver.io/lora-app-server/integrate/data/
```

```
#
# The following substitutions can be used:
# * "{{ .ApplicationID }}" for the application id.
# * "{{ .DevEUI }}" for the DevEUI of the device.
#
# Note: the downlink_topic_template must contain both the
# application id and
# DevEUI substitution!
uplink_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/rx"
#downlink_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/tx"
# join_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/join"
# ack_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/ack"
# error_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/error"
# status_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/status"
# location_topic_template="application/{{ .ApplicationID }}/device/{{
# .DevEUI }}/location"

# MQTT server (e.g. scheme://host:port where scheme is tcp, ssl
# or ws)
server="tcp://localhost:1883"

# Connect with the given username (optional)
username=""

# Connect with the given password (optional)
password=""

# Quality of service level
#
# 0: at most once
# 1: at least once
# 2: exactly once
#
# Note: an increase of this value will decrease the performance.
# For more information: https://www.hivemq.com/blog/
# mqtt-essentials-part-6-mqtt-quality-of-service-levels
qos=0

# Clean session
```

```
  #
  # Set the "clean session" flag in the connect message when this
  # client connects to an MQTT broker. By setting this flag you are
  # indicating that no messages saved by the broker for this client
  # should be delivered.
  clean_session=true

  # Client ID
  #
  # Set the client id to be used by this client when connecting to
  # the MQTT broker. A client id must be no longer than 23 characters.
  # When left blank,a random id will be generated. This requires
  # clean_session=true.
  client_id=""

  # CA certificate file (optional)
  #
  # Use this when setting up a secure connection (when server uses
  # ssl://...)
  # but the certificate used by the server is not trusted by any CA
  # certificate on the server (e.g. when self generated).
  ca_cert=""

  # TLS certificate file (optional)
  tls_cert=""

  # TLS key file (optional)
  tls_key=""


  # Google Cloud Pub/Sub backend.
  [application_server.integration.gcp_pub_sub]
  # Path to the IAM service-account credentials file.
  #
  # Note: this service-account must have the following Pub/Sub roles:
  #  * Pub/Sub Editor
  credentials_file=""

  # Google Cloud project id.
  project_id=""

  # Pub/Sub topic name.
  topic_name=""
```

```
# Settings for the "internal api"
#
# This is the API used by LoRa Server to communicate with LoRa App
# Server and should not be exposed to the end-user.
[application_server.api]
# ip:port to bind the api server
bind="0.0.0.0:8006"

# ca certificate used by the api server (optional)
ca_cert=""

# tls certificate used by the api server (optional)
tls_cert=""

# tls key used by the api server (optional)
tls_key=""

# Public ip:port of the application-server API.
#
# This is used by LoRa Server to connect to LoRa App Server. When
# running
# LoRa App Server on a different host than LoRa Server, make sure
# to set
# this to the host:ip on which LoRa Server can reach LoRa App
# Server.
# The port must be equal to the port configured by the 'bind' flag
# above.
public_host="localhost:8006"


# Settings for the "external api"
#
# This is the API and web-interface exposed to the end-user.
[application_server.external_api]
# ip:port to bind the (user facing) http server to (web-interface
# and REST / gRPC api)
bind="0.0.0.0:8081"

# http server TLS certificate (optional)
tls_cert=""

# http server TLS key (optional)
tls_key=""

# JWT secret used for api authentication / authorization
```

```
  # You could generate this by executing 'openssl rand -base64 32'
  # for example
  jwt_secret="7smNikq5qS9RDp1BgODW+XfA5ZGYeYxk9jutLb7aRzQ="

  # when set, existing users can't be re-assigned (to avoid exposure
  # of all users to an organization admin)"
  disable_assign_existing_users=false



# Join-server configuration.
#
# LoRa App Server implements a (subset) of the join-api specified by
# the LoRaWAN Backend Interfaces specification. This API is used by
# LoRa Server to handle join-requests.
[join_server]
# ip:port to bind the join-server api interface to
bind="0.0.0.0:8004"

# CA certificate (optional).
#
# When set, the server requires a client-certificate and will
# validate this certificate on incoming requests.
ca_cert=""

# TLS server-certificate (optional).
#
# Set this to enable TLS.
tls_cert=""

# TLS server-certificate key (optional).
#
# Set this to enable TLS.
tls_key=""


# Key Encryption Key (KEK) configuration.
#
# The KEK meganism is used to encrypt the session-keys sent from the
# join-server to the network-server.
#
# The LoRa App Server join-server will use the NetID of the
# requesting network-server as the KEK label. When no such label
# exists in the set, the session-keys will be sent unencrypted (which
# can be fine for private networks).
```

```
#
# Please refer to the LoRaWAN Backend Interface specification
# 'Key Transport Security' section for more information.
[join_server.kek]

  # Application-server KEK label.
  #
  # This defines the KEK label used to encrypt the AppSKey
  # (note that the AppSKey is signaled to the NS and on the first
  # received uplink from the NS to the AS).
  #
  # When left blank, the AppSKey will be sent unencrypted
  # (which can be fine for private networks).
  as_kek_label=""

  # KEK set.
  #
  # Example (the [[join_server.kek.set]] can be repeated):
  # [[join_server.kek.set]]
  # # KEK label.
  # label="000000"

  # # Key Encryption Key.
  # kek="01020304050607080102030405060708"
```

lora-gateway-bridge.toml

```
[general]
# debug=5, info=4, warning=3, error=2, fatal=1, panic=0
log_level = 4


# Configuration which relates to the packet-forwarder.
[packet_forwarder]
# ip:port to bind the UDP listener to
#
# Example: 0.0.0.0:1700 to listen on port 1700 for all network
# interfaces.
# This is the listeren to which the packet-forwarder forwards its
# data so make sure the 'serv_port_up' and 'serv_port_down' from your
# packet-forwarder matches this port.
#Cuando no funca se pone 1701
udp_bind = "0.0.0.0:1700"

# Skip the CRC status-check of received packets
```

```
#
# This is only has effect when the packet-forwarder is configured to
# forward
# LoRa frames with CRC errors.
# decia false, pero el json del gateway decia true4
skip_crc_check = true


  # # Managed packet-forwarder configuration.
  # #
  # # By configuring one or multiple managed packet-forwarder
  # # sections, the
  # # LoRa Gateway Bridge updates the configuration when the backend
  # # receives
  # # a configuration change, after which it will restart the
  # # packet-forwarder.
  # [[packet_forwarder.configuration]]
  # # Gateway MAC.
  # #
  # # The LoRa Gateway Bridge will only apply the configuration
  # # updates for this
  # # gateway MAC.
  # mac="0102030405060708"

  # # Base configuration file.
  # #
  # # This file will be used as base-configuration and will not be
  # # overwritten on a configuration update. This file needs to exist
  # # and contains the base
  # # configuration and vendor specific
  # base_file="/etc/lora-packet-forwarder/global_conf.json"

  # # Output configuration file.
  # #
  # # This will be the final configuration for the packet-forwarder,
  # # containing
  # # a merged version of the base configuration + the requested
  # # configuration
  # # update.
  # # Warning: this file will be overwritten on a configuration
  # update!
  # output_file="/etc/lora-packet-forwarder/local_conf.json"

  # # Restart command.
  # #
```

```
  # # This command is issued by the LoRa Gateway Bridge on a
  # # configuration change. Make sure the LoRa Gateway Bridge process
  # # has sufficient permissions to execute this command.
  # restart_command="/etc/init.d/lora-packet-forwarder restart"


# Configuration for the MQTT backend.
[backend.mqtt]
# MQTT topic templates for the different MQTT topics.
#
# The meaning of these topics are documented at:
# https://docs.loraserver.io/lora-gateway-bridge/use/data/
#
# The default values match the default expected configuration of the
# LoRa Server MQTT backend. Therefore only change these values when
# absolutely needed.
# Use "{{ .MAC }}" as an substitution for the LoRa gateway MAC.
#
# Note that some authentication types might overwrite these templates
# (e.g. in case of GCP Cloud IoT Core)!
uplink_topic_template="gateway/{{ .MAC }}/rx"
downlink_topic_template="gateway/{{ .MAC }}/tx"
stats_topic_template="gateway/{{ .MAC }}/stats"
ack_topic_template="gateway/{{ .MAC }}/ack"
config_topic_template="gateway/{{ .MAC }}/config"

# Payload marshaler.
#
# This defines how the MQTT payloads are encoded. Valid options are:
# * v2_json:   The default LoRa Gateway Bridge v2 encoding (will be
# deprecated and removed in LoRa Gateway Bridge v3)
# * protobuf:  Protobuf encoding (this will become the LoRa Gateway
# Bridge v3 default)
# * json:      JSON encoding (easier for debugging, but less compact
# than 'protobuf')
marshaler="v2_json"

  # MQTT authentication.
  [backend.mqtt.auth]
  # Type defines the MQTT authentication type to use.
  #
  # Set this to the name of one of the sections below.
  # Note: when the 'v2_json marhaler' is configured, the generic
  # backend will
  # always be used.
```

```
type="generic"

  # Generic MQTT authentication.
  [backend.mqtt.auth.generic]
  # MQTT server (e.g. scheme://host:port where scheme is tcp,
  # ssl or ws)
  server="tcp://127.0.0.1:1883"

  # Connect with the given username (optional)
  username=""

  # Connect with the given password (optional)
  password=""

  # Quality of service level
  #
  # 0: at most once
  # 1: at least once
  # 2: exactly once
  #
  # Note: an increase of this value will decrease the performance.
  # For more information: https://www.hivemq.com/blog/
  # mqtt-essentials-part-6-mqtt-quality-of-service-levels
  qos=0

  # Clean session
  #
  # Set the "clean session" flag in the connect message when this
  # client connects to an MQTT broker. By setting this flag you are
  # indicating that no messages saved by the broker for this client
  # should be delivered.
  clean_session=true

  # Client ID
  #
  # Set the client id to be used by this client when connecting to
  # the MQTT broker. A client id must be no longer than 23
  # characters. When left blank,
  # a random id will be generated. This requires clean_session=true.
  client_id=""

  # CA certificate file (optional)
  #
  # Use this when setting up a secure connection (when server uses
  # ssl://...) but the certificate used by the server is not trusted
```

```
# by any CA certificate on the server (e.g. when self generated).
ca_cert=""

# mqtt TLS certificate file (optional)
tls_cert=""

# mqtt TLS key file (optional)
tls_key=""

# Maximum interval that will be waited between reconnection
# attempts when connection is lost.
# Valid units are 'ms', 's', 'm', 'h'. Note that these values can
# be combined, e.g. '24h30m15s'.
max_reconnect_interval="10m0s"


# Google Cloud Platform Cloud IoT Core authentication.
#
# Please note that when using this authentication type, the MQTT
# topics
# will be automatically set to match the MQTT topics as expected
# by Cloud IoT Core.
[backend.mqtt.auth.gcp_cloud_iot_core]
# MQTT server.
server="ssl://mqtt.googleapis.com:8883"

# Google Cloud IoT Core Device id.
device_id=""

# Google Cloud project id.
project_id=""

# Google Cloud region.
cloud_region=""

# Google Cloud IoT registry id.
registry_id=""

# JWT token expiration time.
jwt_expiration="24h0m0s"

# JWT token key-file.
#
# Example command to generate a key-pair:
#  $ ssh-keygen -t rsa -b 4096 -f private-key.pem
```

```
    #  $ openssl rsa -in private-key.pem -pubout -outform PEM -out
    # public-key.pem
    #
    # Then point the setting below to the private-key.pem and
    # associate the public-key.pem with this device / gateway
    # in Google Cloud IoT Core.
    jwt_key_file=""


# Metrics configuration.
[metrics]

  # Metrics stored in Prometheus.
  #
  # These metrics expose information about the state of the LoRa
  # Gateway Bridge
  # instance like number of messages processed, number of function
  # calls, etc.
  [metrics.prometheus]
  # Expose Prometheus metrics endpoint.
  endpoint_enabled=false

  # The ip:port to bind the Prometheus metrics server to for serving
  # the metrics endpoint.
  bind=""
```

A.2. Apéndice II - Configuración RTKLib-demo5

A.2.1. Configuración RTKRCV

```
#-------------------------------------------------------------------
#-----README
#-------------------------------------------------------------------
#
# Conf. general: L1, GPS+Glonass
# Solution type: Forward (Real-time)
# AR Mode: fix-and-hold
# Formato: llh
# Base: Fija, llh. Setear ant2-pos1, ant2-pos1, ant2-pos1 acorde.
# Out: Archivo OutNodoRTKPos.pos, TCPcli: //127.0.0.1:1332.
#  Formato: llh.
# In: Serial: //ttyACM0:115200:8:n:1:off, TCPcli: //127.0.0.1:1230.
#  Formato: llh.
# Logs: OutNodoLogRover.ubx OutNodoLogBase.ubx
#
```

```
#---------------------------------------------------------------------

# good starting point for rtkpost options for u-blox NEO-M8T 1 Hz:
# demo5 b29 code

#Comentarios: https://rtklibexplorer.wordpress.com/2016/09/22/rtklib
#    -customizing-the-input-configuration-file/
#   https://rtklibexplorer.wordpress.com/2018/11/27/updated
#   -guide-to-the-rtklib-configuration-file/
# Rover (1), Base (2) and Correction (3)

#The settings and options marked with (*) are available only in demo5
# code

#---------------------------------------------------------------------
#-----SECTION: SETTINGS 1
#---------------------------------------------------------------------
pos1-posmode       =static-start  # (0:single, 1:dgps, 2:kinematic,
  # 3:static, 4:static-start(*),
  # 5:movingbase, 6:fixed, 7:ppp-kine,
  # 8:ppp-static, 9:ppp-fixed)
#(*) "I always require the rover to be stationary long enough to get
#first fix, in which case \static-start" usually works better because
#it takes advantage of the knowledge that the rover is not moving
#initially."

pos1-frequency     =l1         # (1:l1,2:l1+l2,3:l1+l2+l5,4:l1+l5)
#L1 for single frequency receivers

pos1-soltype       =forward    # (0:forward,1:backward,2:combined)
#This is the direction in time that the kalman filter is run. For
#real-time processing, \forward" is your only choice. For
#post-processing, \combined" first runs the filter forward, then
#backwards and combines the results.

pos1-elmask        =15         # (deg)
#Minimum satellite elevation for use in calculating position. I
#usually set this to 15 degrees to reduce the chance of bringing
#multipath into the solution but this setting will be dependent on
#the rover environment. The more open the sky view, the lower this
#value can be set to.

pos1-snrmask_r     =off        # (0:off,1:on)
pos1-snrmask_b     =off        # (0:off,1:on)
#Minimum satellite SNR for rover (_r) and base(_b) for use in
```

```
#calculating position. Can be a more effective criteria for
#eliminating poor satellites than elevation because it is a more
#direct measure of signal quality but the optimal value will vary
#with receiver type and antenna type so I leave it off most of the
#time to avoid the need to tune it for each application.

pos1-snrmask_L1    =38,38,38,38,38,38,38,38,38
#Set SNR thresholds for each five degrees of elevation. I usually
#leave all values the same and pick something between 35 and 38 db
#depending on what the nominal SNR is. These values are only used if
#pos1-snrmask_x is set to on

pos1-dynamics      =on         # (0:off,1:on)
#Enabling rover dynamics adds velocity and acceleration states to the
#kalman filter for the rover. It will improve \kinematic" and
#\static-start" results, but will have little or no effect on
#\static" mode. The release code will run noticeably slower with
#dynamics enabled but the demo5 code should be OK. Be sure to set
#\prnaccelh" and \prnaccelv" appropriately for your rover
#acceleration characteristics.  Rover dynamics is not compatible with
#\moving-base" mode, so turn it off when using that mode.

pos1-posopt1 =off  # **AGREGADO** Sat PCV (off, on)
#Set whether the satellite antenna phase center variation is used or
#not. Leave it off for RTK but you set it for PPP. If set to on, you
#need to specify the satellite antenna PCV file in the files
#parameters.

pos1-posopt2 =off # **AGREGADO** REc PCV (off, on)
#Set whether the receiver antenna phase center variations are used or
#not. If set to on, you need to specify the receiver antenna PCV file
#in the files parameters and the type of receiver antenna for base
#and rover in the antenna section. Only survey grade antennas are
#included in the antenna file available from IGS so only use this if
#your antenna is in the file. It primarily affects accuracy in the
#z-axis so it can be important if you care #about height. You can
#leave this off if both antennas are the same since they will cancel.

pos1-posopt5 =off # **AGREGADO** RAIM FDE (off, on)
#If the residuals for any satellite exceed a threshold, that
#satellite is excluded. This will only exclude satellites with very
#large errors but requires a fair bit of computation so I usually
#leave this disabled.

pos1-tidecorr      =off        # (0:off,1:on,2:otl)
```

```
pos1-ionoopt       =brdc       # (0:off, 1:brdc, 2:sbas, 3:dual-freq,
   # 4:est-stec, 5:ionex-tec, 6:qzs-brdc,
   # 7:qzs-lex, 8:stec)
pos1-tropopt       =saas       # (0:off, 1:saas, 2:sbas, 3:est-ztd,
   # 4:est-ztdgrad, 5:ztd)
pos1-sateph        =brdc       # (0:brdc, 1:precise, 2:brdc+sbas,
   # 3:brdc+ssrapc, 4:brdc+ssrcom)

pos1-exclsats      =           # (prn ...)
#If you know a satellite is bad you can exclude it from the solution
#by listing it here. I only use this in rare cases for debugging if I
#suspect a satellite is bad.

pos1-navsys        =5          # (1:gps +2:sbas +4:glo +8:gal +16:qzs
   # +32:comp)
#I always include GLONASS and SBAS sats, as more information is
#generally better.  If using the newer 3.0 u-blox firmware with the
#M8T I also enable Galileo



#--------------------------------------------------------------------
#-----SECTION: SETTINGS 2
#--------------------------------------------------------------------
pos2-armode        =fix-and-hold # (0:off, 1:continuous,
   # 2:instantaneous, 3:fix-and-hold)
#Integer ambiguity resolution method. I like to think of continuous
#mode as an acquisition mode and fix-and-hold as a tracking mode. I
#normally use continuous mode for static solutions and fix-and-hold
#for moving rovers but if the raw measurement quality is good enough
#to maintain ambiguity resolution when the rover is moving then it is
#probably better to use continuous mode for moving rovers as well.
#This will avoid the risk of locking on to a false fix. If in
#continuous mode, a false fix will usually drop out fairly quickly
#but fix-and-hold will track a false fix for much longer. If \armode"
#is not set to \fix-and-hold" then any of the options below that
#refer to holds don't apply, including pos2-gloarmode.

pos2-varholdamb=0.001 # (*) (meters)
#(*) Starting with the demo5 b26b code, the tracking gain for
#fix-and-hold can be adjusted with this parameter. It is actually a
#variance rather than a gain, so larger values will give lower gain.
#0.001 is the default value, anything over 100 will have very little
#effect. This value is used as the variance for the
#pseudo-measurements generated during a hold which provide feedback
```

```
#to drive the bias states in the kalman filter towards integer values.
#I find that values from 0.1 to 1.0 provides enough gain to assist
#with tracking while still avoiding tracking of false fixes in most
#cases.

pos2-gloarmode     =on # (0:off,1:on,2:autocal,3:fix-and-hold(*))
#Integer ambiguity resolution for the GLONASS sats.  If your receivers
#are identical, you can usually set this to \on" which is the
#preferred setting since it will allow the GLONASS sats to be used
#for integer ambiguity resolution during the initial acquire. If your
#receivers are different or you are using two u-blox M8N receivers
#you will need to null out the inter-channel biases with this
#parameter set to \fix-and-hold" if you want to include the GLONASS
#satellites in the AR solution. In this case the GLONASS sats will
#not be used for inter-channel ambiguity resolution until after the
#inter-channel biases have been calibrated which begins after the
#first hold. There is an \autocal" option as well, but I have never
#been able to make this work.

pos2-gainholdamb=0.01 # (*)
#(*)Starting with the demo5 b26b code, the gain of the inter-channel
#bias calibration for the GLONASS satellites can be adjusted with this
#parameter. Although not fully tested, the hope is that in addition,
#this parameter in conjunction with pos2-varholdamb will enable the
#possibility to null out the inter-channel biases for the GLONASS
#satellites when the tracking effect of fix-and-hold on the GPS
#satellites is not desired (i.e.. effectively continuous mode). This
#would be done by setting pos2-gainholdamb to a nominal value and
#setting pos2-varholdamb to a very large variance to push it's
#tracking gain to near zero.


pos2-bdsarmode     =off        # (0:off,1:on)

pos2-arfilter      =on         # (*) (0:off,1:on)
#(*) Setting this to on will qualify new sats or sats recovering from
#a cycle-slip. If a sat significantly degrades the AR ratio when it
#is first added, its use for ambiguity resolution will be delayed.
#Turning this on should allow you to reduce \arlockcnt" which serves
#a similar purpose but with a blind delay count.

pos2-arthres       =3
# This is the threshold used to determine if there is enough
#confidence in the ambiguity resolution solution to declare a fix.
#It is the ratio of the squared residuals of the second-best solution
```

```
#to the best solution. I generally always leave this at the default
#value of 3.0 and adjust all the other parameters to work around this
#one. Although a larger AR ratio indicates higher confidence than a
#low AR ratio, there is not a fixed relationship between the two. The
#larger the errors in the kalman filter states, the lower the
#confidence in that solution will be for a given AR ratio. Generally
#the errors in the kalman filter will be largest when it is first
#converging so this is the most likely time to get a false fix.
#Reducing pos2-arthers1 can help avoid this. A larger number of
#satellites used for AR will increase the confidence level for a
#given threshold, so in theory at least, it makes sense to increase
#this if you are typically working with a larger number of satellites
#than normal.

pos2-arthres1      =0.05        # (*) use 0.004 if data quality allows
#(*) Integer ambiguity resolution is delayed until the variance of the
#position state has reached this threshold. It is intended to avoid
#false fixes before the bias states in the kalman filter have had time
#to converge.
#It is particularly important to set this to a relatively low value if
#you have set eratio1 to values larger than 100 or are using a single
#constellation solution. If you see AR ratios of zero extending too
#far into your solution, you may need to increase this value since it
#means ambiguity resolution has been disabled because the threshold
#has not been met yet. I find 0.004 to 0.10 usually works well for me
#but if your measurements are lower quality you may need to increase
#this to avoid overly delaying first fix or losing fix after multiple
#cycle slips have occurred.

#pos2-arthres2      =0
#Defined but not used anywhere in the code, best to remove these from
#your config file
##Relative GLONASS hardware bias in meters per frequency slot. This
#parameter is only used when pos2-gloarmode is set to \autocal" and is
#used to specify the inter-channel bias between two different receiver
#manufacturers. To find the appropriate values for common receiver
#types, as well as how to use this parameter for an iterative search
#to find values for receiver types not specified, see this post. This
#parameter is defined but unused in RTKLIB 2.4.3

#pos2-arthres3      =1e-05
#Defined but not used anywhere in the code, best to remove these from
#your config file
##Initial variance of the GLONASS hardware bias state. This parameter
#is only used when pos2-gloarmode is set to \autocal". A smaller value
```

```
#will give more weight to the initial value specified in pos2-arthres2.
#I use 1e-9 when pos2-arthres2 is set to a  known bias, and 1e-7 for
#iterative searches. This parameter is defined but unused in RTKLIB
#2.4.3

#pos2-arthres4      =0.0001
#Defined but not used anywhere in the code, best to remove these from
#your config file
##Kalman filter process noise for the GLONASS hardware bias state. A
#smaller value will give more weight to the initial value specified in
# pos2-arthres2. I use 0.00001 when pos2-arthres2 is set to a known
#bias, and 0.001 for iterative searches.  This parameter is defined
#but unused in RTKLIB 2.4.3

pos2-arlockcnt     =0          # set higher if arfilter=off
#Number of samples to delay a new sat or sat recovering from a
#cycle-slip before using it for integer ambiguity resolution. Avoids
#corruption of the AR ratio from including a sat that hasn't had time
#to converge yet. Use in conjunction with \arfilter". Note that the
#units are in samples, not units of time, so it must be adjusted if
#you change the rover measurement sample rate. I usually set this to
#zero for u-blox receivers which are very good at flagging
#questionable observations but set it to at least five for other
#receivers.

pos2-minfixsats    =4          # (*) min sats to fix ambiguities
#(*) Minimum number of sats necessary to get a fix. Used to avoid
#false fixes from a very small number of satellites, especially
#during periods of frequent cycle-slips.

pos2-minholdsats   =5          # (*) min sats to hold ambiguities
#(*) Minimum number of sats necessary to hold an integer ambiguity
#result. Used to avoid false holds from a very small number of
#satellites, especially during periods of frequent cycle-slips.

pos2-mindropsats   =10         # (*) min sats to enable excluded sat
   #in AR
# (*) Minimum number of sats necessary to enable exclusion of a single
#satellite from ambiguity resolution each epoch.  In each epoch a
#different satellite is excluded.  If excluding the satellite results
#in a significant improvement in the AR ratio, then that satellite is
#removed from the list of satellites used for AR.

pos2-rcvstds       =off        # (*) adjust measurement variances with
   #receiver stdevs
```

```
# (*) Enabling this feature causes the the measurement variances for
#the raw pseudorange and phase measurement observations to be adjusted
#based on the standard deviation of the measurements as reported by
#the receiver. This feature is currently only supported for u-blox
#receivers. The adjustment in variance is in addition to adjustments
#made for satellite elevation based on the stats-errphaseel parameter.
#I generally get better results with this turned off.

pos2-varholdamb    =0.1        # variance for fix-and-hold psuedo
   # measurements (cycle^2)
pos2-gainholdamb   =0.01       # gain used for GLO and SBAS sats to
   # adjust IC biases

pos2-arelmask      =15         # (deg)
#Functionally no different from the default of zero, since elevations
#less than \elmask" will not be used for ambiguity resolution but I
#changed it to avoid confusion.

pos2-arminfix      =20         # (samples)(20*sample rate) adjust for
   # sample rate
#Number of consecutive fix samples needed to hold the ambiguities.
#Increasing this is probably the most effective way to reduce false
#holds, but will also increase time to first hold. Note that this
#value also needs to be adjusted if the rover measurement sample rate
#changes.

pos2-armaxiter     =1

pos2-elmaskhold    =15          # (deg)
#Functionally no different from the default of zero, since elevations
#less than \elmask" will not be used for holding ambiguity resolution
#results but I changed it to avoid confusion.

pos2-aroutcnt      =20        # (samples)(20*sample rate) adjust for
  # sample rate
#Number of consecutive missing samples that will cause the ambiguities
#to be reset. Again, this value needs to be adjusted if the rover
#measurement sample rate changes.

pos2-maxage        =100         # (s)
#Maximum delay between rover measurement and base measurement (age of
#differential) in seconds. This usually occurs because of missing
#measurements from a misbehaving radio link. I've increased it from
#the default because I found I was often still getting good results
#even when this value got fairly large, assuming the dropout occurred
```

```
#after first fix-and-hold.

pos2-syncsol       =off        # (0:off,1:on)
pos2-slipthres     =0.05       # (m)

pos2-rejionno      =1000       # (m)
#Reject a measurement if its pre-fit residual is greater than this
#value in meters. I have found that RTKLIB does not handle outlier
#measurements well, so I set this large enough to effectively disable
#it. There was a recent bug fix in the release code related to
#outliers but even with this fix I found that I got better results
#with a larger value.

pos2-rejgdop       =30
pos2-niter         =1
pos2-baselen       =0          # (m)
pos2-basesig       =0          # (m)


#---------------------------------------------------------------------
#-----SECTION: OUTPUT
#---------------------------------------------------------------------
out-solformat      =llh        # (0:llh, 1:xyz, 2:enu, 3:nmea)
#I am usually interested in relative distances between rover and base,
#so set this to \enu". If you are interested in absolute locations,
#set this to \llh" but make sure you set the exact base location in
#the \ant2" settings. Be careful with this setting if you need
#accurate z-axis measurements. Only the llh format will give you a
#constant z-height if the rover is at constant altitude. \Enu" and
#\xyz" are cartesian coordinates and so the z-axis follows a flat
#plane, not the curvature of the earth. This can lead to particularly
#large errors if the base station is located farther from the rover
#since the curvature will increase with distance.

out-outhead        =on         # (0:off, 1:on)
#No functional difference to the solution, just output more info to
#the result file.

out-outopt         =off #on         # (0:off, 1:on)
#No functional difference to the solution, just output more info to
#the result file.

out-timesys        =utc      # (0:gpst,1:utc,2:jst)
out-timeform       =hms        # (0:tow,1:hms)
out-timendec       =3
```

```
out-degform        =deg        # (0:deg,1:dms)
out-fieldsep       =
out-height         =ellipsoidal # (0:ellipsoidal, 1:geodetic)
out-geoid          =internal   # (0:internal, 1:egm96, 2:egm08_2.5,
   # 3:egm08_1, 4:gsi2000)
out-solstatic      =all        # (0:all, 1:single)
out-nmeaintv1      =0          # (s)
out-nmeaintv2      =0          # (s)

out-outstat        =off #residual       # (0:off, 1:state, 2:residual)
#No functional difference to the solution, just output residuals to a
#file. The residuals can be very useful for debugging problems with a
#solution.

stats-eratio1      =300
#Ratio of the standard deviations of the pseudorange measurements to
#the carrier-phase measurements. I have found a larger value works
#better for low-cost receivers, but that the default value of 100
#often work better for more expensive receivers since they have less
#noisy pseudorange measurements. Larger values tend to cause the
#kalman filter to converge faster and leads to faster first fixes but
#it also increases the chance of a false fix. If you increase this
#value, you should set pos2-arthres1 low enough to prevent finding
#fixes before the kalman filter has had time to converge. I believe
#increasing this value has a similar effect to increasing the time
#constant on a pseudorange smoothing algorithm in that it filters out
#more of the higher frequencies in the pseudorange measurements while
#maintaining the low frequency components.

stats-eratio2      =300
##Ratio of the standard deviations of the pseudorange measurements to
#the carrier-phase measurements. I have found a larger value works
#better for low-cost receivers, but that the default value of 100
#often work better for more expensive receivers since they have less
#noisy pseudorange measurements. Larger values tend to cause the
#kalman filter to converge faster and leads to faster first fixes but
#it also increases the chance of a false fix. If you increase this
#value, you should set pos2-arthres1 low enough to prevent finding
#fixes before the kalman filter has had time to converge. I believe
#increasing this value has a similar effect to increasing the time
#constant on a pseudorange smoothing algorithm in that it filters out
#more of the higher frequencies in the pseudorange measurements while
#maintaining the low frequency components.

stats-errphase     =0.003      # (m)
```

```
stats-errphaseel   =0.003      # (m)
stats-errphasebl   =0          # (m/10km)
stats-errdoppler   =1          # (Hz)
stats-stdbias      =30         # (m)
stats-stdiono      =0.03       # (m)
stats-stdtrop      =0.3        # (m)

stats-prnaccelh    =3.0        # (m/s^2) 3.0
#If receiver dynamics are enabled, use this value to set the standard
#deviation of the rover receiver acceleration in the horizontal
#components. This value should include accelerations at all
#frequencies, not just low frequencies. It should characterize any
#movements of the rover antenna, not just movements of the complete
#rover so it may be larger than you think. It will include
#accelerations from vibration, bumps in the road, etc as well as the
#more obvious rigid-body accelerations of the whole rover.
##It can be estimated by running a solution with this value set to a
#large value, then examining the accel values in the solution file
#with RTKPLOT

stats-prnaccelv    =1.0        # (m/s^2) 1.0
#The comments about horizontal accelerations apply even more to the
#vertical acceleration component since in many applications the
#intentional accelerations will all be in the horizontal components.
#It is best to derive this value from actual GPS measurement data
#rather than expectations of the rigid-body rover. It is better to
#over-estimate these values than to under-estimate them.

stats-prnbias      =0.0001     # (m)
stats-prniono      =0.001      # (m)
stats-prntrop      =0.0001     # (m)
stats-prnpos       =0          # (m)
stats-clkstab      =5e-12      # (s/s)

ant1-postype       =rinexhead  # (0:llh, 1:xyz, 2:single, 3:posfile,
   # 4:rinexhead, 5:rtcm)
ant1-pos1          =0          # (deg|m)
ant1-pos2          =0         # (deg|m)
ant1-pos3          =0          # (m|m)
ant1-anttype       =
ant1-antdele       =0          # (m)
ant1-antdeln       =0          # (m)
ant1-antdelu       =0          # (m)

ant2-postype       =llh  # (0:llh, 1:xyz, 2:single, 3:posfile,
```

```
 # 4:rinexhead, 5:rtcm)
#This is the location of the base station antenna. If you are only
#interested in relative distance between base and rover this value
#does not need to be particularly accurate. For post-processing I
#usually use the approximate base station location from the RINEX
#file header. If you want absolute position in your solution, then
#the base station location must be much more accurate since any error
#in that will add to your rover position error. If I want absolute
#position, I first process the base station data against a nearby
#reference station to get the exact location, then use the "llh" or
#\xyz" option to specify that location. For real-time processing, I
#use the \single" option which uses the single solution from the data
#to get a rough estimate of base station location.

ant2-pos1          =-34.918143          # (deg|m)
ant2-pos2          =-56.166706           # (deg|m)
ant2-pos3          =40          # (m|m)
ant2-anttype       =
ant2-antdele       =0          # (m)
ant2-antdeln       =0          # (m)
ant2-antdelu       =0          # (m)

ant2-maxaveep      =1
#Specifies the number of samples averaged to determine base station
#location if \postype" is set to \single". I set this to one to
#prevent the base station position from varying after the kalman
#filter has started to converge since that seems to cause long times
#to first fix. In most cases for post-processing, the base station
#location will come from the RINEX file header and so you will not
#use this setting. However if you are working with RTCM files you may
#need this even for post-processing.

ant2-initrst       =off        # (0:off,1:on)


#---------------------------------------------------------------------
#-----SECTION: MISC
#---------------------------------------------------------------------
misc-timeinterp    =off        # (0:off,1:on)
##Interpolates the base station observations.  I generally set this
#to \on" if the base station observations sample time is larger than
#5 seconds.

misc-sbasatsel     =0          # (0:all)
misc-rnxopt1       =
```

```
misc-rnxopt2       =

file-satantfile    =
file-rcvantfile    =
file-staposfile    =
file-geoidfile     =
file-ionofile      =
file-dcbfile       =
file-eopfile       =
file-blqfile       =
file-tempdir       =
file-geexefile     =
file-solstatfile   =2
file-tracefile     =2


#--------------------------------------------------------------------
#-----SECTION: STREAMS
#--------------------------------------------------------------------
# Rover (1), Base (2) and Correction (3)

inpstr1-type       =serial     # (0:off, 1:serial, 2:file, 3:tcpsvr,
   # 4:tcpcli, 7:ntripcli, 8:ftp, 9:http)
inpstr2-type       =tcpsvr     # (0:off, 1:serial, 2:file, 3:tcpsvr,
   # 4:tcpcli, 7:ntripcli, 8:ftp, 9:http)
inpstr3-type       =off        # (0:off, 1:serial, 2:file, 3:tcpsvr,
   # 4:tcpcli, 7:ntripcli, 8:ftp, 9:http)
inpstr1-path       =ttyACM0:115200:8:n:1:off
inpstr2-path       =127.0.0.1:1230
inpstr3-path       =ttyACM0:115200:8:n:1:off
inpstr1-format     =ubx        # (0:rtcm2, 1:rtcm3, 2:oem4, 3:oem3,
   # 4:ubx, 5:swift, 6:hemis, 7:skytraq,
   # 8:gw10, 9:javad, 10:nvs, 11:binex,
   # 12:rt17, 13:sbf, 14:cmr, 15:tersus,
   # 17:sp3)
inpstr2-format     =ubx        # (0:rtcm2, 1:rtcm3, 2:oem4, 3:oem3,
   # 4:ubx, 5:swift, 6:hemis, 7:skytraq,
   # 8:gw10, 9:javad, 10:nvs, 11:binex,
   # 12:rt17, 13:sbf, 14:cmr, 15:tersus,
   # 17:sp3)
inpstr3-format     =ubx        # (0:rtcm2, 1:rtcm3, 2:oem4, 3:oem3,
   # 4:ubx, 5:swift, 6:hemis, 7:skytraq,
   # 8:gw10, 9:javad, 10:nvs, 11:binex,
   # 12:rt17, 13:sbf, 14:cmr, 15:tersus,
   # 17:sp3)
```

```
inpstr2-nmeareq    =off        # (0:off,1:latlon,2:single)
inpstr2-nmealat    =0          # (deg)
inpstr2-nmealon    =0          # (deg)

outstr1-type       =file       # (0:off, 1:serial, 2:file, 3:tcpsvr,
   # 4:tcpcli, 6:ntripsvr)
outstr2-type       =tcpcli     # (0:off, 1:serial, 2:file, 3:tcpsvr,
   # 4:tcpcli, 6:ntripsvr)
outstr1-path       =OutNodoRTKPos.pos
outstr2-path       =127.0.0.1:1332
outstr1-format     =llh        # (0:llh,1:xyz,2:enu,3:nmea)
outstr2-format     =llh        # (0:llh,1:xyz,2:enu,3:nmea)

logstr1-type       =file        # (0:off, 1:serial, 2:file, 3:tcpsvr,
# 4:tcpcli, 6:ntripsvr)
logstr2-type       =file        # (0:off, 1:serial, 2:file, 3:tcpsvr,
# 4:tcpcli, 6:ntripsvr)
logstr3-type       =off         # (0:off, 1:serial, 2:file, 3:tcpsvr,
# 4:tcpcli, 6:ntripsvr)
logstr1-path       =OutNodoLogRover.ubx
logstr2-path       =OutNodoLogBase.ubx
logstr3-path       =
```

A.3. Apéndice III - Librerías: basic_pkt_forwarder

A.3.1. Instalación

Para poder utilizar un gateway LoRaWAN se descargó la biblioteca packet_forwarder del proyecto Lora network packet forwarder de github [49].

Para poder correr finalmente el gateway LoRaWAN, en la terminal abriendo la carpeta /lora/exec/packet_forwarder se ejecuta: sudo ./basic_packet_forwarder

A.3.2. Archivo de Configuración

A continuacuón se presenta el archivo de configuración del gateway LoRaWAN, descargado junto con la biblioteca packet_forwarder y modificado para el correcto funcionamiento

global_conf.json

```
{
"SX1301_conf": {
"lorawan_public": true,
"clksrc": 0,
"clksrc_desc": "radio_1 provides clock to concentrator for
 most devices except MultiTech. For MultiTech set to 0.",
```

```
"antenna_gain": 0,
"antenna_gain_desc": "antenna gain, in dBi",
"radio_0": {
"enable": true,
"type": "SX1257",
"freq": 917200000,
"rssi_offset": -166.0,
"tx_enable": true,
"tx_freq_min": 915000000,
"tx_freq_max": 928000000
},
"radio_1": {
"enable": true,
"type": "SX1257",
"freq": 917900000,
"rssi_offset": -166.0,
"tx_enable": false
},
"chan_multiSF_0": {
"desc": "Lora MAC, 125kHz, all SF, 916.8 MHz",
"enable": true,
"radio": 0,
"if": -400000
},
"chan_multiSF_1": {
"desc": "Lora MAC, 125kHz, all SF, 917.0 MHz",
"enable": true,
"radio": 0,
"if": -200000
},
"chan_multiSF_2": {
"desc": "Lora MAC, 125kHz, all SF, 917.2 MHz",
"enable": true,
"radio": 0,
"if": 0
},
"chan_multiSF_3": {
"desc": "Lora MAC, 125kHz, all SF, 917.4 MHz",
"enable": true,
"radio": 0,
"if": 200000
},
"chan_multiSF_4": {
"desc": "Lora MAC, 125kHz, all SF, 917.6 MHz",
"enable": true,
```

```
"radio": 1,
"if": -300000
},
"chan_multiSF_5": {
"desc": "Lora MAC, 125kHz, all SF, 917.8 MHz",
"enable": true,
"radio": 1,
"if": -100000
},
"chan_multiSF_6": {
"desc": "Lora MAC, 125kHz, all SF, 918.0 MHz",
"enable": true,
"radio": 1,
"if": 100000
},
"chan_multiSF_7": {
"desc": "Lora MAC, 125kHz, all SF, 918.2 MHz",
"enable": true,
"radio": 1,
"if": 300000
},
"chan_Lora_std": {
"desc": "Lora MAC, 125kHz, SF8, 917.5 MHz",
"enable": true,
"radio": 0,
"if": 300000,
"bandwidth": 500000,
"spread_factor": 8
},
"chan_FSK": {
"desc": "disabled",
"enable": false
},
"tx_lut_0": {
"desc": "TX gain table, index 0",
"pa_gain": 0,
"mix_gain": 8,
"rf_power": -6,
"dig_gain": 0
},
"tx_lut_1": {
"desc": "TX gain table, index 1",
"pa_gain": 0,
"mix_gain": 10,
"rf_power": -3,
```

```
"dig_gain": 0
},
"tx_lut_2": {
"desc": "TX gain table, index 2",
"pa_gain": 0,
"mix_gain": 12,
"rf_power": 0,
"dig_gain": 0
},
"tx_lut_3": {
"desc": "TX gain table, index 3",
"pa_gain": 1,
"mix_gain": 8,
"rf_power": 3,
"dig_gain": 0
},
"tx_lut_4": {
"desc": "TX gain table, index 4",
"pa_gain": 1,
"mix_gain": 10,
"rf_power": 6,
"dig_gain": 0
},
"tx_lut_5": {
"desc": "TX gain table, index 5",
"pa_gain": 1,
"mix_gain": 12,
"rf_power": 10,
"dig_gain": 0
},
"tx_lut_6": {
"desc": "TX gain table, index 6",
"pa_gain": 1,
"mix_gain": 13,
"rf_power": 11,
"dig_gain": 0
},
"tx_lut_7": {
"desc": "TX gain table, index 7",
"pa_gain": 2,
"mix_gain": 9,
"rf_power": 12,
"dig_gain": 0
},
"tx_lut_8": {
```

```
"desc": "TX gain table, index 8",
"pa_gain": 1,
"mix_gain": 15,
"rf_power": 13,
"dig_gain": 0
},
"tx_lut_9": {
"desc": "TX gain table, index 9",
"pa_gain": 2,
"mix_gain": 10,
"rf_power": 14,
"dig_gain": 0
},
"tx_lut_10": {
"desc": "TX gain table, index 10",
"pa_gain": 2,
"mix_gain": 11,
"rf_power": 16,
"dig_gain": 0
},
"tx_lut_11": {
"desc": "TX gain table, index 11",
"pa_gain": 3,
"mix_gain": 9,
"rf_power": 20,
"dig_gain": 0
},
"tx_lut_12": {
"desc": "TX gain table, index 12",
"pa_gain": 3,
"mix_gain": 10,
"rf_power": 23,
"dig_gain": 0
},
"tx_lut_13": {
"desc": "TX gain table, index 13",
"pa_gain": 3,
"mix_gain": 11,
"rf_power": 25,
"dig_gain": 0
},
"tx_lut_14": {
"desc": "TX gain table, index 14",
"pa_gain": 3,
"mix_gain": 12,
```

```
"rf_power": 26,
"dig_gain": 0
},
"tx_lut_15": {
"desc": "TX gain table, index 15",
"pa_gain": 3,
"mix_gain": 14,
"rf_power": 27,
"dig_gain": 0
}
},

    "gateway_conf": {
        "gateway_ID": "AA555A0000000001",
        /* change with default server address/ports, or overwrite
        in local_conf.json */
        "server_address": "164.73.45.130",
        "serv_port_up": 1701,
        "serv_port_down": 1701,
        /* adjust the following parameters for your network */
        "keepalive_interval": 10,
        "stat_interval": 30,
        "push_timeout_ms": 100,
"ipol":true,
        /* forward only valid packets */
        "forward_crc_valid": true,
        "forward_crc_error": false,
        "forward_crc_disabled": true,
        /* GPS configuration */
        "gps_tty_path": "/dev/ttyAMA0",
        /* GPS reference coordinates */
        "ref_latitude": 0.0,
        "ref_longitude": 0.0,
        "ref_altitude": 0
    }


}
```

A.4. Apéndice IV - Configuración U-blox NEO-M8T

En el siguiente archivo se configuraron un máximo de 32 satélites (GPS y GLONASS), 1 Hz de frecuencia de recepción de datos y se habilitaron los mensajes UBX-RXM-RAWX y UBX-RXM-SFRBX.

A.4.1. Configuración NEO-M8T

```
MON-VER - 0A 04 FA 00 45 58 54 20 43 4F 52 45 20 33 2E 30 31 20 28 31
31 31 31 34 31 29 00 00 00 00 00 00 00 00 30 30 30 38 30 30 30 30 00
00 52 4F 4D 20 42 41 53 45 20 32 2E 30 31 20 28 37 35 33 33 31 29 00
00 00 00 00 00 00 00 00 46 57 56 45 52 3D 54 49 4D 20 31 2E 31 30 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 50 52 4F 54 56 45 52 3D
32 32 2E 30 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4D
4F 44 3D 4E 45 4F 2D 4D 38 54 2D 30 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 46 49 53 3D 30 78 45 46 34 30 31 35 20 28 31 30 30
31 31 31 29 00 00 00 00 00 00 00 00 00 47 50 53 3B 47 4C 4F 3B 47 41
4C 3B 42 44 53 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 53 42 41
53 3B 49 4D 45 53 3B 51 5A 53 53 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00

CFG-ANT - 06 13 04 00 1B 00 F0 B9

CFG-DAT - 06 06 02 00 00 00

CFG-GNSS - 06 3E 3C 00 00 20 20 07 00 00 20 00 01 00 01 01 01 00 00
00 00 00 00 01 02 00 00 00 00 00 00 01 03 00 00 00 00 00 00 01 04 00
00 00 00 00 00 03 05 00 00 00 00 00 00 05 06 00 20 00 01 00 01 01

CFG-INF - 06 02 0A 00 00 00 00 00 00 00 00 00 00 00
CFG-INF - 06 02 0A 00 01 00 00 00 07 07 00 07 07 00

CFG-ITFM - 06 39 08 00 F3 AC 62 2D 1E 03 00 00

CFG-LOGFILTER - 06 47 0C 00 01 00 00 00 00 00 00 00 00 00 00 00 00

CFG-MSG - 06 01 08 00 0B 30 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0B 33 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0B 31 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0B 01 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 21 08 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 0B 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 09 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 02 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 06 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 07 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 21 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0A 08 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 60 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 22 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 31 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 04 00 00 00 00 00 00
```

```
CFG-MSG - 06 01 08 00 01 61 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 39 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 09 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 34 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 01 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 02 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 07 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 35 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 32 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 06 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 03 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 30 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 24 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 25 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 23 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 20 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 26 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 21 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 11 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 01 12 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 02 61 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 02 14 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 02 15 00 01 00 01 00 00
CFG-MSG - 06 01 08 00 02 59 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 02 13 00 01 00 01 00 00
CFG-MSG - 06 01 08 00 02 20 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0D 04 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0D 03 00 01 00 01 00 00
CFG-MSG - 06 01 08 00 0D 01 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 0D 06 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 00 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 01 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 02 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 03 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 04 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 05 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 06 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 07 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 08 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 09 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 0A 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 0D 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F0 0F 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F1 00 00 00 00 00 00 00
CFG-MSG - 06 01 08 00 F1 03 00 00 00 00 00 00
```

```
CFG-MSG - 06 01 08 00 F1 04 00 00 00 00 00 00

CFG-NAV5 - 06 24 24 00 FF FF 03 03 00 00 00 00 10 27 00 00 05 00 FA
00 FA 00 64 00 5E 01 00 3C 00 00 00 00 00 00 03 00 00 00 00 00

CFG-NAVX5 - 06 23 28 00 02 00 FF FF 7F 02 00 00 03 02 01 20 09 00 00
01 00 00 4B 07 00 01 00 00 01 01 00 00 00 64 64 00 00 01 11 00 00 00
00 00

CFG-NMEA - 06 17 14 00 00 40 00 02 00 00 00 00 00 00 00 01 00 00 00
00 00 00 00 00

CFG-ODO - 06 1E 14 00 00 00 00 00 00 00 00 00 19 46 19 66 0A 32 00 00
99 4C 00 00

CFG-PM2 - 06 3B 30 00 02 06 00 00 00 14 42 01 E8 03 00 00 10 27 00 00
00 00 00 00 00 00 00 00 2C 01 00 00 4F C1 03 00 86 02 00 00 FE 00 00
00 64 40 01 00 00 00 00 00

CFG-PRT - 06 00 14 00 00 00 00 00 84 00 00 00 00 00 00 00 07 00 03 00
00 00 00 00
CFG-PRT - 06 00 14 00 01 00 00 00 C0 08 00 00 80 25 00 00 07 00 03 00
00 00 00 00
CFG-PRT - 06 00 14 00 03 00 00 00 00 00 00 00 00 00 00 00 07 00 03 00
00 00 00 00
CFG-PRT - 06 00 14 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00

CFG-RATE - 06 08 06 00 E8 03 01 00 01 00

CFG-RINV - 06 34 18 00 00 4E 6F 74 69 63 65 3A 20 6E 6F 20 64 61 74
61 20 73 61 76 65 64 21 00

CFG-RXM - 06 11 02 00 48 00

CFG-SBAS - 06 16 08 00 00 03 03 00 89 A3 07 00

CFG-TMODE2 - 06 3D 1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00

CFG-TP5 - 06 31 20 00 00 01 00 00 32 00 00 00 40 42 0F 00 40 42 0F 00
00 00 00 00 A0 86 01 00 00 00 00 00 F7 00 00 00
CFG-TP5 - 06 31 20 00 01 01 00 00 32 00 00 00 04 00 00 00 01 00 00 00
48 E8 01 00 A0 86 01 00 00 00 00 00 FE 00 00 00
```

```
CFG-USB - 06 1B 6C 00 46 15 A8 01 00 00 00 00 64 00 02 00 75 2D 62 6C
6F 78 20 41 47 20 2D 20 77 77 77 2E 75 2D 62 6C 6F 78 2E 63 6F 6D 00
00 00 00 00 00 75 2D 62 6C 6F 78 20 47 4E 53 53 20 72 65 63 65 69 76
65 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Referencias

[1] Margaret Rouse. Internet de las cosas (iot). Accedido en Octubre 2019 a https://searchdatacenter.techtarget.com/es/definicion/Internet-de-las-cosas-IoT, 2017.

[2] PandoraFMS. Lpwan: introducción al protocolo de comunicaciones de iot. Accedido en Octubre 2019 a https://pandorafms.com/blog/es/que-es-lpwan/, 2018.

[3] Semtech. What is lora? Accedido en Octubre 2019 a https://www.semtech.com/lora/what-is-lora.

[4] La LoRa Alliance. What is the lorawan® specification? Accedido en Octubre 2019 a https://lora-alliance.org/about-lorawan.

[5] European Global Navigation Satellite System Agency. What is gnss? Accedido en Octubre 2019 a https://www.gsa.europa.eu/european-gnss/what-gnss.

[6] Varios Autores. Learn more about dgps. Accedido en Octubre 2019 a https://www.sciencedirect.com/topics/earth-and-planetary-sciences/dgps.

[7] NovAtel. An introduction to gnss. Accedido en Octubre 2019 a https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/satellite-based-augmentation-systems/.

[8] NovAtel. An introduction to gnss. Accedido en Octubre 2019 a https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/precise-point-positioning-ppp/.

[9] NovAtel. An introduction to gnss. Accedido en Octubre 2019 a https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/real-time-kinematic-rtk/.

[10] Varios autores. Sistema de posicionamiento global. Accedido en Octubre 2019 a https://www.gps.gov/spanish.php.

[11] Varios autores. Glonass. Accedido en Octubre 2019 a https://beebom.com/what-is-glonass-and-how-it-is-different-from-gps/.

[12] An introduction to gnss. Accedido en Octubre 2019 a http://www.igm.gub.uy/geoportal/estaciones/.

[13] Zbigniew Ianelli John Lampe. Introduction tochirp spread spectrum (css) technology. page 167, 2003.

[14] European Global Navigation Satellite Systems Agency. Galileo (navegación por satélite). Accedido en Octubre 2019 a https://www.gsa.europa.eu/european-gnss/galileo/galileo-european-global-satellite-based-navigation-system, 2019.

[15] Juan Toloza Nelson Acosta. Techniques to improve the gps precision. Accedido en Octubre 2019 a https://www.researchgate.net/publication/268186524_Techniques_to_improve_the_GPS_precision, 2012.

[16] Tomoji Takasu. Rtklib ver. 2.4.2 manual. page 167, 2013.

[17] Sigfox. Sigfox home page. Accedido en Octubre 2019 a https://www.sigfox.com/en.

[18] Ingenu. Ingenu home page. Accedido en Octubre 2019 a https://www.ingenu.com/.

[19] Telensa. Telensa home page. Accedido en Octubre 2019 a https://www.telensa.com/.

[20] Mahesh Sooriyabandara Usman Raza, Parag Kulkarni. Low power wide area networks. *IEEE Communications Surveys and Tutorials*, page 7.

[21] Yi Pan Lei Pan, Hongyi Wu. Medium access control in wireless networks. *Nova Science Publishers, Inc.*, 2008.

[22] Varios Autores. Ieee 802.3 ethernet working group. 2019.

[23] Varios autores. Telefonía móvil 3g. Accedido en Octubre 2019 a https://es.wikipedia.org/wiki/Telefonia_movil_3G.

[24] María Estela Raffino. Wifi. Accedido en Octubre 2019 a https://concepto.de/wifi/, 2018.

[25] Semtech. Lorawan network server. demonstration: Gateway to server interface definition. 2015.

[26] Semtech Home Page. Semtech. Accedido en Octubre 2019 a https://www.semtech.com/.

[27] Varios autores. Modelo osi. Accedido en Octubre 2019 a https://es.wikipedia.org/wiki/Modelo_OSI.

[28] Varios autores. Banda ism. Accedido en Octubre 2019 a https://es.wikipedia.org/wiki/Banda_ISM.

[29] Wim Lamotte William Thenaers Pieter Robyns, Peter Quax. A multi-channel software decoder for the lora modulation scheme. 2018.

[30] Miguel Angel Casanova. ¿qué es lora? una tecnología lpwan para iot. Accedido en Octubre 2019 a https://alfaiot.com/blog/ultimas-noticias-2/post/que-es-lora-2.

[31] Varios Autores. Foreward error-correction. Accedido en Octubre 2019 a https://www.sciencedirect.com/topics/engineering/forward-error-correction.

[32] LoRa Alliance Technical committee. Lorawan^{TM} 1.0.2 regional parameters. 2017.

[33] Ali Grami. Introduction to digital communications. pages 409–455, 2016.

[34] Gonzalo Carro Patricia Hernández. Uso del espectro radioeléctrico en uruguay. 2016.

[35] Haciendo iot con lora: Capitulo 2.- tipos y clases de nodos. Accedido en Octubre 2019 a https://medium.com/beelan/haciendo-iot-con-lora-capitulo-2-tipos-y-clases-de-nodos-3856aba0e5be.

[36] Hewlett Packard Enterprise Centro de soporte. Hp proliant dl380 g5 server - overview. Accedido en Octubre 2019 a https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c00712808.

[37] O. Brocaar. loraserver home page. Accedido en Octubre 2019 a https://www.loraserver.io/.

[38] MQTT. Mqtt home page. Accedido en Octubre 2019 a http://mqtt.org/.

[39] Raspberry PI. Raspberry pi home page. Accedido en Octubre 2019 a https://www.raspberrypi.org/.

[40] U-blox. U-blox home page. Accedido en Octubre 2019 a https://www.u-blox.com/en.

[41] Varios autores. Lora/gps hat. Accedido en Octubre 2019 a https://wiki.dragino.com/index.php?title=Lora/GPS_HAT.

[42] T. Eirich T. Kramp O.Hersent N. Sornin, M. Luis. Lorawan^{TM} specification. 2016.

[43] Multi Tech system. Multi tech home page. Accedido en Octubre 2019 a https://www.multitech.com/.

[44] Multi Tech system. Multi tech mcard mtac-lora-915. Accedido en Octubre 2019 a https://www.multitech.com/models/94557300LF.

[45] Semtech. Products wireless rf. Accedido en Octubre 2019 a https://www.semtech.com/products/wireless-rf/lora-gateways/sx1301.

[46] Rtkexplorer. Rtklib code: Windows executables. Accedido en Octubre 2019 a http://rtkexplorer.com/downloads/rtklib-code/.

[47] Tomoji Takasu. Rtklib ver. 2.4.2 manual. 2013.

[48] rpi-lora-tranceiver. Accedido en Octubre 2019 a https://github.com/dragino/rpi-lora-tranceiver.

[49] Varios Autores. Lora network packet forwarder project. Accedido en Octubre 2019 a https://github.com/mirakonta/packet_forwarder.

[50] Varios Autores. Lmic-rpi-lora-gps-hatt. Accedido en Octubre 2019 a https://github.com/gloveboxes/lmic-rpi-lora-gps-hat.

[51] Dave Glover. Lmic-rpi-lora-gps-hat. Accedido en Octubre 2019 a https://www.hackster.io/glovebox/lorawan-for-raspberry-pi-with-worldwide-frequency-support-e327d2.

[52] SunNav. Comercial solution price. Accedido en Octubre 2019 a https://spanish.alibaba.com/product-detail/Big-Discount-Price-for-dual-frequency-60064443738.html?spm=a2700.galleryofferlist.normalList.1.1ad364c2taKMPx&s=p.

[53] UniStrong. Comercial solution price 2. Accedido en Octubre 2019 a https://spanish.alibaba.com/product-detail/220-channels-differential-GPS-GNSS-Receiver-60825554464.html?spm=a2700.galleryofferlist.normalList.64.1ad364c2taKMPx&s=p.

[54] Hewlett Packard Enterprise Servers. Retired products. Accedido en Octubre 2019 a https://techlibrary.hpe.com/us/en/enterprise/servers/retired/index.aspx.

[55] Hewlett Packard Enterprise Servers. Hp proliant dl380 gen 10 price. Accedido en Octubre 2019 a https://buy.hpe.com/us/en/servers/rack-servers/proliant-dl300-servers/proliant-dl380-server/hpe-proliant-dl380-gen10-server/p/1010026818.

[56] Hp proliant dl380 gen 5 price. Accedido en Octubre 2019 a https://articulo.mercadolibre.com.ar/MLA-822320905-servidor-hp-proliant-dl380-g5-_JM#position=1&type=item&tracking_id=2b908eb8-0731-499b-9157-db39c04b2386.

[57] Raspberry PI. Raspberry pi price. Accedido en Octubre 2019 a https://uk.rs-online.com/web/p/processor-microcontroller-development-kits/1720555.

[58] Multi Tech system. Multi tech mcard price. Accedido en Octubre 2019 a https://shop.multitech.com/mtac-lora-h-915.html.

[59] Ublox. Ublox neo-m8t price. Accedido en Octubre 2019 a https://www.gnss.store/gnss-gps-modules/97-ublox-neo-7p-ppp-usb-dogle-receiver-with-sma-for-uav-robots-pc.html.

[60] Gps/lora hat price. Accedido en Octubre 2019 a https://dwmzone.com/en/dragino/354-1279-lora-gps-hat-long-distane-wireless-433mhz-868mhz-915mhz-lora-and-gps
html#/21-center_frequency-433mhz.

[61] PostgreSQL. The world's most advanced open source relational database. Accedido en Octubre 2019 a https://www.postgresql.org/.

[62] Redis. Accedido en Octubre 2019 a https://redis.io/.

[63] Varios autores. Estructuras de datos persistentes. Accedido en Octubre 2019 a https://es.wikipedia.org/wiki/Estructuras_de_datos_persistentes.

[64] O. Brocaar. Accedido en Octubre 2019 a https://www.loraserver.io/guides/debian-ubuntu/.

[65] Hackrf. open source hardware for software-defined radio. Accedido en Octubre 2019 a https://greatscottgadgets.com/hackrf/.

[66] Gqrx sdr. Accedido en Octubre 2019 a http://gqrx.dk/.

[67] Spyder. Accedido en Octubre 2019 a https://www.spyder-ide.org/.

[68] Introducción a la librería paho y mqtt para iot(internet de las cosas). Accedido en Octubre 2019 a https://www.nociones.de/introduccion-paho-mqtt-iot/.

[69] gmplot 1.2.0. Accedido en Octubre 2019 a https://pypi.org/project/gmplot/.

[70] U-blox. Receiver description. Accedido en Octubre 2019 a https://www.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_%28UBX-13003221%29_Public.pdf.

[71] U-blox. User guide. Accedido en Octubre 2019 a https://www.u-blox.com/sites/default/files/u-center_UserGuide_%28UBX-13005250%29.pdf.

[72] rtkexploer. Rtklib: Customizing the input configuration file). Accedido en Octubre 2019 a https://rtkexplorer.com/rtklib-customizing-the-input-configuration-file/.

[73] rtkexploer. Updated guide to the rtklib configuration file. Accedido en Octubre 2019 a https://rtklibexplorer.wordpress.com/2018/11/27/updated-guide-to-the-rtklib-configuration-file/.

[74] rtkexploer. Rtkexplorer home page. Accedido en Octubre 2019 a `https://rtkexplorer.com/`.

[75] Sistema de información geográfica. Accedido en Octubre 2019 a `http://sig.montevideo.gub.uy/`.

[76] Intendencia Municipal de Montevideo. Marco de referencia geodesico de montevideo. Accedido en Octubre 2019 a `http://intgis.montevideo.gub.uy/sit/aplicaciones/re_ge/pdf/FI.pdf`.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el jueves 16 enero, 2020.
http://iie.fing.edu.uy/