Communication LoRa au moyen de RIOT-OS pour la mesure centimétrique par GPS différentiel avec RTKLib

J.-M Friedt, 18 octobre 2022
FEMTO-ST département temps-fréquence, Besançon

Afin de compenser les effets corrélés qui dégradent la mesure de position par navigation satellitaire, notamment GPS, nous désirons faire communiquer deux récepteurs de signaux de navigation par satellites capables de compenser les délais ionosphériques et troposphériques en vue d'une mesure centimétrique. Pour ce faire, une liaison point à point sub-GHz d'une portée de plusieurs kilomètres est établie en s'appuyant sur l'interface physique LoRa programmée grâce à l'environnement exécutif RIOT-OS sur STM32. Cet exercice est l'opportunité d'analyser l'impact des paramètres de communication numérique sur le débit, la portée ou la robustesse de la liaison radiofréquence.

D'une part nous désirons appréhender les nouveaux récepteurs GPS Zed-F9P de UBlox qui mettent à la portée de la majorité des bourses – 270 euros le récepteur MikroE-4456 chez Digikey – la mesure centimétrique par GPS, d'autre part notre collègue Didier Donsez du Laboratoire d'Informatique de Grenoble (LIG) nous fait la promotion de RIOT-OS, encore un nouvel environnement exécutif de plus, cette fois orienté vers les systèmes embarqués communiquant par liaison radiofréquence. Serions nous capables d'allier ces deux activités pour développer un système de récepteurs GPS communiquant permettant une mesure centimétrique en temps réel en s'imposant de ne pas émettre de signaux radiofréquences au-dessus du GHz sur une distance de plusieurs kilomètres ? La réponse est évidemment positive, mais sera l'opportunité de surmonter bien des écueils.

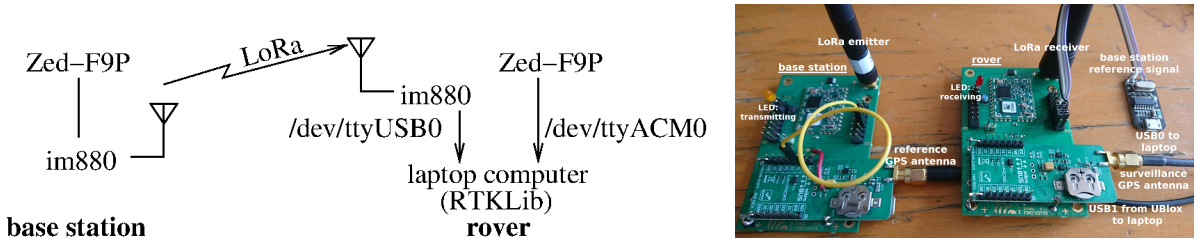


FIGURE 1 – Gauche : architecture générale du système : une station de base, supposée fixe, transmet ses mesures de la constellation de satellites de navigation vers un récepteur mobile – *rover* – qui se localise relativement à la station de base avec une résolution centimétrique en éliminant les perturbations communes grâce au traitement par RTKLib. Droite : circuits réalisés avec radiomodem compatible LoRa im880 et récepteur GPS Zed-F9P.

L'architecture générale du circuit que nous désirons monter (Fig. 1) contient deux systèmes communiquant, l'un supposé fixe à un emplacement connu et reproductible nommé *base station* et l'autre mobile nommé *rover*. Ces deux interlocuteurs échangent des informations pour corriger les sources communes de fluctuation de temps de vol du signal de navigation par satellite qui permet la localisation par trilatération. La portée visée entre les deux interlocuteurs sera de quelques kilomètres en vue directe. N'ayant pas réussi à identifier des références de modem (modulateur-démodulateur) radiofréquence sub-GHz avec des débits suffisants pour de tels échanges – les vénérables XBee Pro 868 MHz avec leur watt émis et trentaine de kilomètre de portée ne semblent plus commercialisés – nous nous lançons dans un développement spécifique.

1 Principe du GPS différentiel

La navigation par satellite s'appuie sur diverses constellations en orbite moyenne (altitude de l'ordre de 20000 km) regroupés sous le nom de GNSS pour *Global Nagivation Satellite Systems* – GPS américain, Galileo européen, Beidou chinois, GLONASS russe, éventuellement assistés de quelques satellites géostationnaires ou géosynchrones tel que QZSS japonais ou IRNSS indien – tous survolant la Terre largement au dessus de l'altitude de l'ionosphère (quelques centaines de kilomètres), la couche la plus externe de l'atmosphère, tellement peu dense que les particules cosmiques impactant depuis l'univers peuvent séparer les électrons des ions et créer un plasma dans lequel la célérité d'une onde électromagnétique dépend de ce taux d'ionisation. Un bilan des retards dans un lien satellite-sol qui impacte la capacité à effectuer

une trilatération précise pour retrouver la position au sol du récepteur [1] (*UERE* est *User-equivalent Range Error*) indique que ce délai ionosphérique est largement le contributeur majeur. Alors que cette grandeur peut être corrigée *a posteriori* si suffisamment de mesures ont été engrangées au cours de l'observation au sol (les pseudo-ranges et éventuellement pseudo-phase documentées dans des fichiers au format RINEX), il est souhaitable de savoir en temps-réel si la correction du délai ionosphérique observé par une station de référence a pu être efficacement retranchée du récepteur mobile. Dans ce contexte de GPS différentiel, deux stations supposées soumises *aux mêmes conditions de propagation des ondes électromagnétiques*, et donc classiquement supposées distantes de moins d'une trentaine de kilomètres, observent la même constellation de satellites. Le récepteur de référence est supposé fixe et transmet aux stations mobiles ses observations. Sous réserve que les stations mobiles soient soumises aux mêmes retards de propagation, la soustraction des distances sol-satellites (pseudo-ranges) et éventuellement de la phase du signal acquis (pseudo-phases) permet de corriger cet effet commun et donc de s'en affranchir, réduisant considérablement l'incertitude sur la position *relative* entre station et *rover* mobile. Le réel bénéfice, au-delà d'une variance réduite à court terme, est surtout une reproductibilité long terme de mesures séparées dans le temps tel que nous les effectuons lorsque nous observons les évolution d'un glacier ou cherchons à retrouver des capteurs ou balises sous quelques mètres de neige. En effet, devoir creuser 2 m de neige sur une surface de 5×5 m^2 ou au dessus d'une zone de quelques décimètres carrés change considérablement l'effort nécessaire.

Segment Source	Error Source	1σ Error (m)
Space/control	Broadcast clock	1.1
	L1 P(Y)-L1 C/A group delay	0.3
	Broadcast ephemeris	0.8
User	Ionospheric delay	7.0*
	Tropospheric delay	0.2
	Receiver noise and resolution	0.1
	Multipath	0.2
System UERE	Total (RSS)	7.1*

TABLE 1 – Extrait de [1] listant les sources d'incertitudes de positionnement par le système de navigation américain, "GPS Standard Positioning Service Typical UERE Budget" qui mentionne "residual ionospheric errors tend to be highly correlated among satellites resulting in position errors being far less than predicted using DOP." (DOP signifiant *Dilution of Precision*).

Ainsi, un système RTK pour *Real Time Kinematic* fournit la mesure et correction en temps réel plutôt que par post-traitement – par exemple par analyse des mesures des stations de référence de l'*International GNSS Service* IGS, en échangeant les informations entre base statique de référence et récepteur mobile. Pour une application de mesure cet aspect temps réel n'a que peu d'intérêt ... si ce n'est pour garantir le résultat et donc ne pas imposer de rester sur un point de mesure trop longtemps, ou au contraire rater la mesure en ne restant pas assez longtemps. L'aspect temps réel permet aussi de considérer l'information lors d'une recherche d'objets sous la neige ou la glace, dans un périmètre considérablement restreint par rapport au GPS simple récepteur. Pour ce faire, il faudra donc échanger des informations entre station et *rover* sur une portée au moins supérieure à la plus grande extension du site d'étude, et ce malgré la topographie ou d'éventuels obstacles sur le lien radiofréquence.

2 Premiers essais avec RTKLib

Avant de se lancer dans un long développement de liaison radiofréquence, il peut sembler judicieux de vérifier la capacité à atteindre les résolutions et exactitudes visées avec la mesure différentielle. Il faut pour cela une bibliothèque de traitement des informations brutes acquises par récepteurs de GNSS, et parmi les outils libres à notre disposition seul RTKLib de Tomoji Takasu répond à nos attentes d'accès sans contrainte aux sources (Bernese de l'Université de Bern à `http://www.bernese.unibe.ch/` ou GAMIT/GLOBK du MIT à `http://geoweb.mit.edu/gg/` nécessitant de s'enregistrer pour obtenir des codes sources qui ne peuvent être rediffusés – et par conséquent avec une documentation et un support de la communauté plus que réduits pour ne pas dire inexistants). RTKLib vient soit sous forme d'outils en ligne de commande – donc parfaitement appropriés pour une utilisation sur système embarqué autonome – qui se compilent individuellement par `gcc` sous GNU/Linux, soit avec des interfaces graphiques Qt que nous utiliserons ici. La version officielle de RTKLib vise les récepteurs GNSS haut de gamme multifréquences et nous nous orienterons vers la déclinaison du logiciel pour les pauvres [2] qu'est

`https://github.com/rinex20/RTKLIB-demo5` optimisé pour les récepteurs mono-fréquence faible coût incluant les téléphones mobiles les plus récents et les récepteurs UBlox tel que le Zed-F9P que nous utiliserons ici. Noter que le paquet binaire proposé par Debian/GNU Linux `rtklib-qt` est la version officielle de RTKLib et **ne convient pas** pour les traitements qui vont suivre, ne proposant **pas** l'option d'identifier dynamiquement la position de la station de base (Fig. 2) fonctionnalité sur laquelle nous reviendrons dans la conclusion pour le lecteur absolument désireux d'utiliser la version professionnelle de RTKLib. La compilation des binaires n'a rien de subtil et ils restent confinés dans le répertoire utilisateur, évitant de polluer le système.

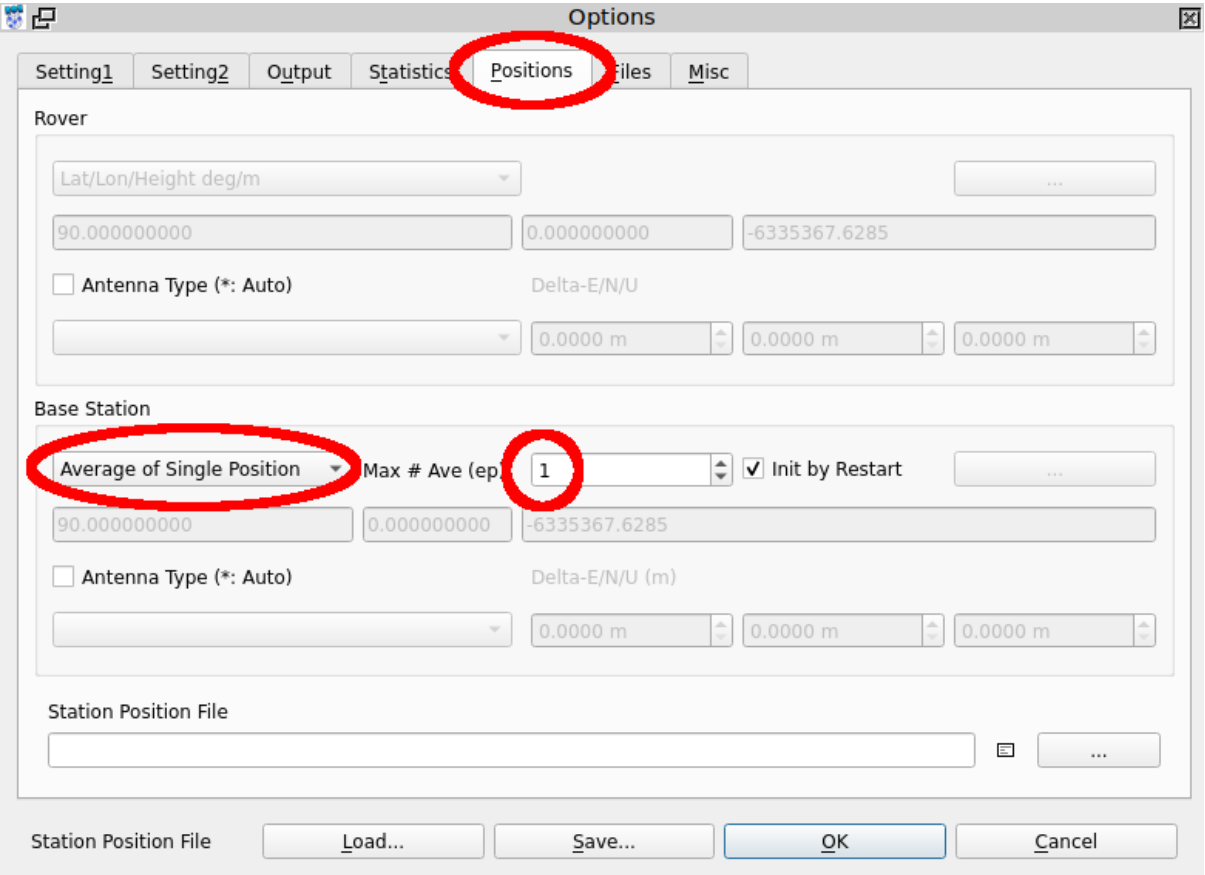


FIGURE 2 – Le menu permettant de dynamiquement identifier la position de la station de base sans entrer manuellement ses coordonnées n'est disponible que dans la déclinaison `demo5` de RTKLib que nous utiliserons au cours de ces développements, et *pas* dans la version officielle fournie notamment comme paquet binaire Debian/GNU Linux.

Nous commençons par nous familiariser avec `rtknavi_qt` qui fusionne les informations de deux récepteurs GNSS, que dans un premier temps nous connecterons par liaison filaire USB, avec l'unique PC chargé du traitement. Les récepteurs GNSS sélectionnés doivent au minimum pouvoir acquérir et communiquer les pseudo-ranges et pseudo-phases brutes *avant* l'obtention de la solution en Position, Vitesse et Temps (PVT) puisque c'est en fusionnant ces mesures que RTKLib éliminera les sources communes de retards qui polluent l'obtention de la solution centimétrique. De tels récepteurs étaient historiquement coûteux et limités aux professionnels qui pouvaient rémunérer l'investissement initial en plusieurs dizaines de kiloeuros. Le véritable changement de paradigme vient de la production par les suisses d'UBlox du Zed-F9P qui fournit, pour une somme avoisinant les 270 euros monté et connectorisé, un récepteur GNSS multi-constellations/multi-bandes sous réserve de se munir d'une antenne adéquate, elle aussi pour une somme avoisinant la centaine d'euros. Nous restons tous de même dans un montage qui globalement coûte dans les 500 à 600 euros, nettement moindre que les solutions antérieures et accessibles à l'amateur éclairé.

Nous nous inspirons des nombreux documents disponibles auprès de `rtklibexplorer.wordpress.com` et en particulier `https://rtklibexplorer.wordpress.com/tag/zed-f9p/` qui indique comment configurer le récepteur, et `https://rtklibexplorer.wordpress.com/2016/08/29/getting-started-with-rtknavi/`, qui donne les premier pas pour configurer `rtknavi_qt` tel que nous le décrirons ci-dessous après la description de la configuration du récepteur.

Pour ne pas faire durer le suspens, le résultat de la Fig. 3 montre la validation du système en déplaçant l'antenne *rover*, maintenant la station fixe, devant un mètre gradué pour valider la mesure sur une preuve. L'accord se maintient sur la seconde décimale : la résolution centimétrique est validée.

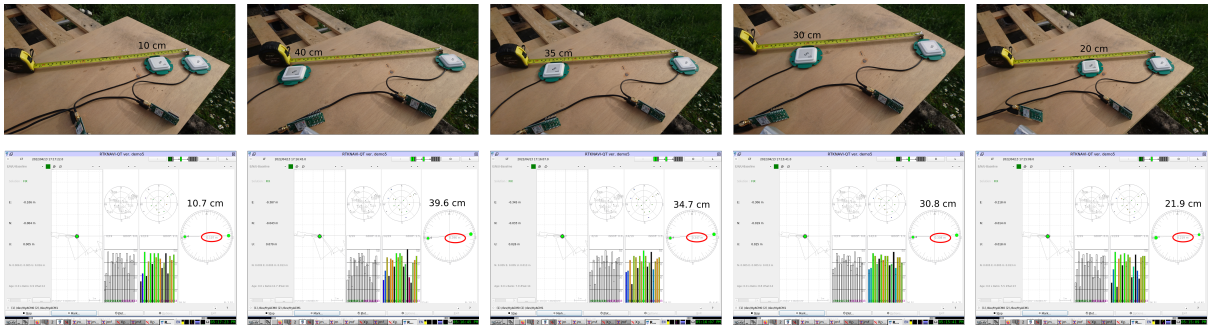


FIGURE 3 – Quelques exemples de mesures de référence avec les deux antennes bifréquences connectées chacune à un UBlox Zed-F9P, chacun connecté par port USB au même ordinateur exécutant une unique instance de `rtknavi_qt` de RTKLib.

Pour atteindre ce résultat, les deux récepteurs Zed-F9P sont configurés pour désactiver toute trame NMEA (trames au format ASCII après traitement de la solution), activer les trames au format propriétaire UBlox nommé UBX, et en particulier seul le message **UBX-RXM-RAWX** qui nous sera utile. Bien que nombre d'efforts soient faits pour fournir un outil libre de configuration des récepteurs UBlox, et en particulier le script Python `ubxtool` fourni par sa distribution favorite dans le paquet `gpsd`, nous utiliserons encore l'outil graphique propriétaire `U-Center` de UBlox qui fonctionne surprenamment bien avec `wine`. Afin de supporter les récepteurs de la série 9, il faut utiliser une version suffisamment récente du logiciel : pour notre part, la version 22.02 de Février 2022. La manipulation de U-Center sous Wine demande de prendre le coup de main, après avoir vérifié dans `$HOME/.wine/dosdevices` quel port COM de MS-DOS a été lié par un lien symbolique au point de communication `/dev/ttyACM0` sous GNU/Linux. Une fois U-Center lancé, nous nous connectons par l'icône en haut à gauche en forme de prise après avoir sélectionné le bon port. Le menu de configuration des messages s'obtient par `View` → `Messages View`. Pour chaque menu, nous désactiverons ou activerons la fonction, puis tuerons la fenêtre avant de la rouvrir, puisqu'il ne semble pas convaincant de pouvoir activer ou désactiver plusieurs fonctions séquentiellement depuis Wine. La séquence que nous utilisons pour configurer un nouveau Zed-F9P en tant que station de base est :

1. `View`→`Messages View`→`NMEA` bouton droite et `Disable Child Messages`, fermer `Messages View`
2. `View`→`Messages View`→`UBX` bouton droite et `Enable Child Messages`, fermer `Messages View`
3. `View`→`Messages View`→`UBX-MON` bouton droite et `Disable Child Messages`
4. `UBX-NAV` bouton droite et `Disable Child Messages`
5. `UBX-RXM` bouton droite et `Disable Child Messages`. Après cela, seul `UBX-RXM-RAWX` devrait être encore actif, l'objectif étant de minimiser au strict minimum les trames transmises entre station de base et rover
6. `UBX-CFG-MSG` et sélectionner `02-15 RXM-RAWX` pour activer `UART1` et `Send` (USB devrait déjà être actif et on peut le laisser pour observer depuis le PC les messages de la station avant de partir observer à distance avec le rover)
7. `UBX-CFG-PRT`→`Target 1-UART1` : définir le débit de communication (Baudrate) à 115200 et conserver les autres paramètres du protocole (8N1), `Protocol In UBX` uniquement et `Protocol Out UBX` uniquement (désactiver toute référence à NMEA, trop verbeux), puis `Send` en changeant de menu
8. `UBX-CFG-GNSS`→ désactiver QZSS, SBAS, Beidou et GLONASS, pour ne conserver que GPS (L1C/A) et Galileo (E1) – qui peut s'avérer nécessaire – toujours dans un objectif de réduire la taille des trames. `Send` en changeant de menu
9. `UBX-CFG-CFG`→`Save Current Configuration to` et cliquer sur `FLASH`, puis `Send` en changeant de menu
10. déconnecter en débranchant l'USB, rebrancher USB et vérifier que les modifications ont été prises en compte, en particulier l'option de PRT qui passe la communication à 115200 bauds

Ce dernier point – augmenter le débit de communication entre le récepteur GNSS et le microcontrôleur – est important car nous verrons que nous allons être limités par le début de communication de LoRa et le temps nécessaire à transmettre les trames devient prohibitif pour maintenir la cadence chaque seconde.

Nous avons fait le choix, dans le logiciel qui va suivre, de séparer la phase d'acquisition de la phase de communication radiofréquence. Plus la phase d'acquisition est courte en utilisant un débit RS232 élevé, plus nous aurons de temps pour communiquer par liaison radiofréquence ensuite.

Si les constellation autres que GPS et Galileo s'obstinent à refuser de se désactiver, les grands moyens passent par `ubxtool` de `gpsd` qui désactive les constellations inutiles par

```
ubxtool -d GLONASS
ubxtool -d BEIDOU
ubxtool -d QZSS
ubxtool -d SBAS
```

Noter que pour le moment nous n'avons pas trouvé comment réactiver une constellation désactivée. Pour le rover qui sera directement connecté au PC et ne passera pas par une communication radiofréquence, la configuration par défaut peut être maintenue, le volume de trafic n'ayant pas beaucoup d'importance sur un lieu USB filaire.

Pour ce qui est de la configuration de RTKLib tel qu'introduit à `https://rtklibexplorer.wordpress.com/2016/08/29/getting-started-with-rtknavi/`, nous cliquons tout en bas sur "Options..." et sélectionnons "Positioning Mode" comme "Kinematic" sur les fréquences L1 à 5, désactivons toutes les corrections ionosphériques et troposphériques puis dans l'onglet "Settings2" nous sélectionnons "Integer Ambiguity Res" par une solution continue tel que conseillé dans `https://rtklibexplorer.wordpress.com/2016/03/13/improving-rtklib-solution-fix-and-hold/` qui indique *"Continuous" is the default option and uses the kalman filter to estimate the phase biases continuously over many epochs. In general, this provides much less noisy estimates of the phase biases and is essential for working with low cost receivers. The downside of the continuous solution is that if bad data is allowed to get into the filter outputs it can remain for multiple epochs and corrupt subsequent solutions, so some care must be taken to avoid that.* Nous ne modifions pas les autres onglets des Options, mais renseignons l'emplacement des récepteurs dans le symbole "I" (*Inputs*) en haut à droite en cliquant pour activer *Rover* et *Base station*, en renseignant sous *Type* que ce sont des liens "Serial", en renseignant le périphérique de communication `/dev/ttyACM0` et `/dev/ttyACM1` respectivement dans les deux champs *Opt→Port* et le débit de communication. Aucune commande *Cmd* n'est fournie et le *Format* des données est "u-blox UBX". L'onglet de *Correction* est désactivé au cours de nos mesures.

Convaincus des performances du récepteur GNSS associé à RTKLib, nous pouvons nous lancer sereinement dans le développement du système de communication radiofréquence.

3 Cahier des charges : communication de trames RS232 par liaison sub-GHz

Les objectifs que nous désirons atteindre sont les suivantes :

— portée : dans un contexte de mesures sur un glacier arctique avec les collègues du laboratoire de géographie bisontin ThéMA, nous désirons cartographier la position des divers capteurs et autres balises d'ablation avec une résolution centimétrique dans un bassin glaciaire de 3,5 km NS par 3,2 km EW à une distance entre 2,2 et 5,5 km de la station de base qui servira de référence.

— fréquence porteuse : la région qui nous intéresse est le site de deux radiotélécsopes utilisés à des fins de géodésie pour localiser la planète Terre dans l'Univers, et il est souhaitable de ne pas brouiller ces récepteurs sensibles par une émission depuis le sol [3]. Afin de respecter les règles d'émission radiofréquences sur le site qui nous concerne, nous devons nous limiter aux porteuses sub-GHz dans lesquelles les radiotéléscopes ne fonctionnent pas.

— après une recherche plus ou moins exhaustive des transpondeurs radiofréquences disponibles commercialement, la documentation est soit trop superficielle pour garantir que le signal radiofréquence remplace de façon transparente le câble sans introduire des caractères additionnels dans les trames, soit ne semble pas garantir les bandes passantes de communication visées, ou encore moins la portée. Dans le doute et compte tenu du tarif exorbitant des solutions proposées, sans aucune possibilité de réparer sur le terrain en cas de dysfonctionnement des systèmes fermés propriétaires, la solution commerciale est éliminée. Nous désirons tout de même des composants disponibles sur le marché : les modulations AM ou FM des systèmes de communication tels que Mipot ou Radiometrix commercialisés par Lextronic sont trop inefficaces pour dépasser la centaine de mètres de portée. Sans aucune complaisance pour la solution propriétaire et non documentée – voir aux brevets mensongers [4] – nous n'avons d'autre choix qu'accepter la solution proposée par Semtech qu'est le protocole LoRa que nous développerons ci-dessous. Précisons cependant que le protocole qui nous intéresse

est le plus bas dans les couches OSI, à savoir LoRa pour une liaison point à point entre deux interlocuteurs uniquement, et non LoRaWAN qui fait la promotion de toute l'infrastructure de routage des messages émis depuis des capteurs nommés *Endpoints* collectés au travers d'une liaison radiofréquence LoRa par des *Gateways* avant d'être routés vers un concentrateur d'informations (*Network Server*), généralement au travers d'infrastructures propriétaires qui n'ont plus grand chose à voir avec la liaison radiofréquence que nous désirons établir. Didier Donsez propose l'excellent `https://github.com/CampusIoT/orbimote/tree/master/field_test_device` pour un exemple d'implémentation de LoRaWAN sur RIOT-OS s'appuyant sur la bibliothèque de Semtech. Dans cet exemple particulier où l'objectif est de reproduire le comportement d'une liaison série asynchrone compatible RS232 sur un lien radiofréquence, nous avons fait le choix de ne pas bénéficier de LoRaWAN pour simplement reproduire sur le récepteur aussi fidèlement que possible les trames émises, et ce par une liaison point à point LoRa uniquement.

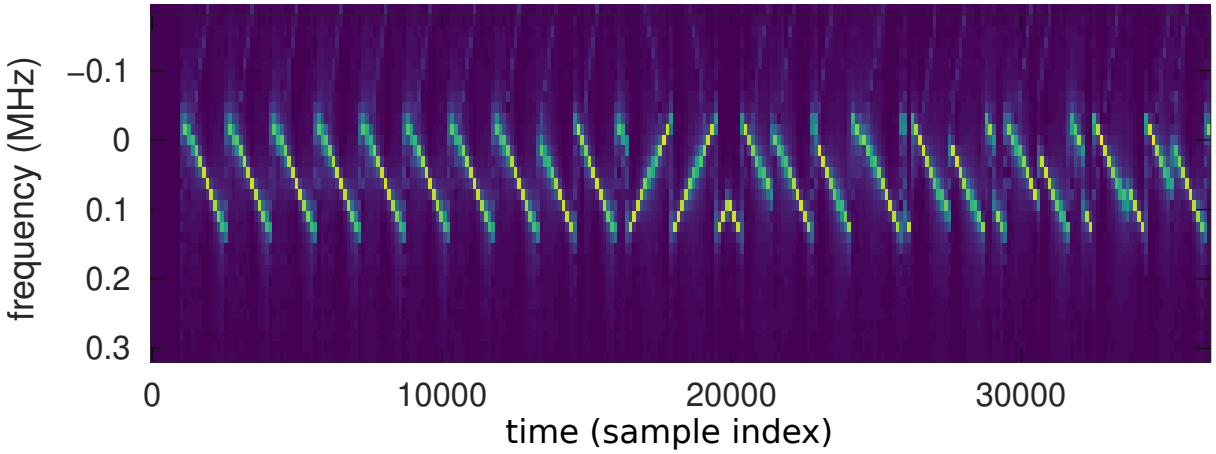


FIGURE 4 – Signal LoRa autour de 868 MHz acquis par récepteur de radio logicielle RTL-SDR et affiché comme transformées de Fourier successives à court terme dans un diagramme temps-fréquence, mettent en évidence les chirps définis comme un signal de fréquence évoluant linéairement avec le temps.

Bien que la couche physique de LoRa soit propriétaire, son mode de démodulation se comprend trivialement et l'implication des divers paramètres se modélise bien pour identifier le compromis entre débit, portée et durée d'émission donc consommation énergétique.

LoRa est basé sur une modulation en chirp (Fig. 4), donc chaque symbole est transmis comme un signal dont la fréquence évolue linéairement entre deux bornes séparées de la bande passante B durant une durée T. La bande passante B ne peut valoir que quelques valeurs discrètes que sont 125, 250 ou 500 kHz. La durée de chaque chirp T est donnée par un paramètre nommé *Spreading Factor* SF, avec une durée doublée pour chaque incrément de SF. Ainsi, $SF = 7$ génère un chirp deux fois plus court que $SF = 8$, lui même deux fois plus court que $SF = 9$ et ainsi de suite jusqu'à $SF = 12$. Cette dernière valeur propose donc la meilleure de limite de détection avec une moyenne qui dure le plus longtemps, au détriment d'un débit de communication le plus lent, chaque bit transmis étant plus long (Fig. 5).

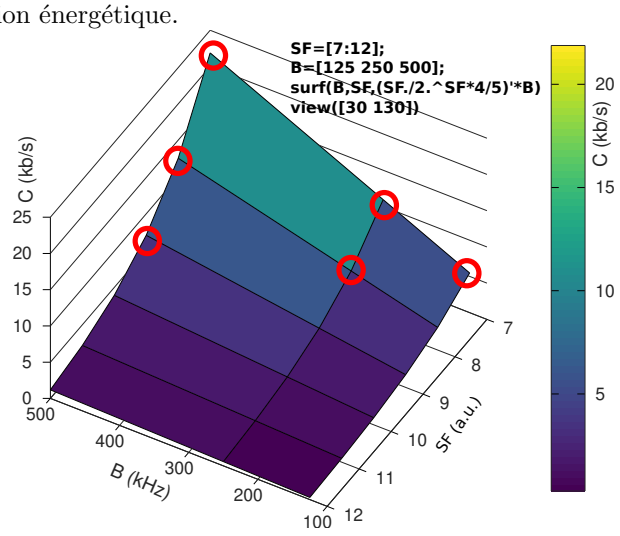


Figure 5: Débit (axe vertical) en fonction de la bande passante B et de la durée de chaque impulsion déterminée par SF. Les cercles rouges indiquent les cas où le débit dépasse les 5 kb/s, le minimum acceptable pour l'application de GPS centimétrique visée.

Le récepteur va corréler le signal reçu avec le chirp linéaire supposé connu et donc reproduit au niveau du récepteur, afin de trouver le décalage de phase (fréquence) introduite en émission pour transmettre la valeur de chaque bit. La corrélation fait immédiatement intervenir le concept de gain de rapport signal à bruit – *Pulse Compression Ratio* PCR classiquement utilisé pour qualifier les formes d'ondes utilisées en RADAR – $B \times T$ (voir ci-dessous). Plus cette grandeur est importante, meilleure est la capacité à retrouver le signal émis dans le signal reçu bruité. Cependant, augmenter T réduit le débit de données et nous nous imposons de transmettre l'ensemble des informations nécessaires à la localisation centimétrique par GPS chaque seconde, tandis

que augmenter B augmente le bruit thermique intégré par le filtre plus large en réception et dégrade la limite de détection, donc la portée. Il va donc falloir faire des compromis entre portée et débit par la sélection des paramètres de configuration de LoRa (cercles rouges sur Fig. 5).

Le taux de compression d'impulsion
Le taux de compression d'impulsion ou *Pulse Compression Ratio* est un estimateur de la performance attendue d'une forme d'onde radiofréquence pour d'une part maximiser la bande passante – débit en communication numérique ou résolution en distance pour une mesure de temps de vol RADAR – et d'autre part permettre de réduire le bruit par moyennage. Ces deux objectifs sont *a priori* antagonistes : moyenne nécessite un signal long dans le temps tandis qu'une bonne résolution spatiale ou un débit de communication rapide nécessite *a priori* des signaux brefs. L'analyse de la corrélation du motif transmis $m(t)$ avec le signal bruité $s(t)$ de la forme $xcorr(s,m)(\tau) = \int s(t) \times m(t+\tau)dt$ indique que la largeur du pic de corrélation est uniquement déterminé par la bande passante B du signal, quelque-soit la façon dont cette bande passante est atteinte (séquence de bruit, balayage de fréquence, impulsion brève) tandis que la capacité de réjection du bruit est donnée uniquement par la durée du motif T qui détermine les bornes de l'intégrale. Le taux de compression d'impulsion $PCR = B \times T$ vise à maximiser ces deux grandeurs autant que faire se peut, généralement dans la limite des contraintes technologiques ou règlementaires. Dans le cas de LoRa, l'étalement de spectre par balayage linéaire de fréquence a été choisi (Fig. 4), avec des paramètres judicieusement sélectionnés puisque deux communication simultanées à des débits (SF) différents sont elles-mêmes orthogonales et peuvent être séparées lors de la corrélation avec le motif approprié pour chaque SF (durée de la rampe de fréquence).

Ainsi, la couche physique de LoRa est suffisamment simple et paramétrable pour pouvoir appliquer un certain nombre de concepts de traitement du signal, et en particulier

— tel que l'a enseigné Shannon, plus la bande passante B est importante, plus élevé est le débit de données transmis sur un canal (sous réserve d'un rapport signal à bruit suffisant)

— plus la bande passante B est importante, plus le bruit thermique $k_B \times T \times B$, k_B la constante de Boltzman et T la température absolue en Kelvin – le produit $k_B \times T$ est égal à la valeur classique de -174 dBm/Hz à l'ambiante – intégré dans la bande de mesure est importante : à rapport signal à bruit faible, augmenter la bande passante n'amène plus de gain en débit car se compense avec la dégradation du bruit.

— plus un signal est mesuré longtemps, plus nous pouvons le moyenner et donc réduire le bruit pour ne garder que le signal : plus T est grand, meilleur est le rapport signal à bruit donc la limite de détection.

Combinée à la puissance radiofréquence émise, limitée par l'amplificateur de puissance de sortie (et la norme qui en Europe limite à +14 dBm), nous avons tous les termes pour estimer le bilan de liaison et le débit de communication en fonction de la bande passante et de la durée de chaque chirp. Formellement, le débit C (bits/s) s'exprime [5] comme

$$C = \frac{SF}{2^{SF}} \times \frac{4}{5} B$$

le dernier facteur représentant la redondance introduite pour corriger des erreurs de communication que nous ne modifierons par ici (la justification étant que nous avons choisi le *Coding Rate* $CR = 1$ fixe dans toutes nos applications).

Puisque $B \in [125, 250, 500]$ kHz, le nombre de possibilités est relativement restreint compte tenu des objectifs à atteindre, et il ne reste qu'à estimer la longueur du message à transmettre chaque seconde selon la norme UBX.

Nous apprenons dans le manuel décrivant les trames UBX du UBlox F9P (`https://cdn.sparkfun.com/assets/f/7/4/3/5/PM-15136.pdf` page 182) que les trames RAWX font $16 + 32 \times N$ octets avec N le nombre de satellites décrits. Si nous observions toutes les constellations et leurs signaux d'assistance (nombre de canaux alloués à chaque signal entre parenthèse), à savoir GPS (16), Galileo (18), Beidou (5), GLONASS (12), SBAS (3) et QZSS (4), nous aurions au pire 58 canaux à transmettre ou 1872 octets chaque seconde, soit 14976 bits/s. Ce débit de communication est à la limite de ce que peut atteindre le Sx1272 : seul une bande passante $B = 500$ kHz et $SF = 7$ permet de dépasser les 15 kb/s avec $C = 21,875$ kb/s. Ce faisant, nous risquons de dégrader la portée de mesure et utilisons un mode de communication interdit en Europe, mais ce débit est accessible. Les 1872 octets ou 18720 bits en protocole 8N1 de RS232 mettent 162,5 ms pour être acquis au débit de 115200 bauds tandis que les 14976 bits résultants mettent 685 ms pour être transmis par LoRa. La somme reste en dessous de 1 s et est donc acceptable. Cependant QZSS n'est accessible qu'au Japon, mélanger des constellations à fréquence fixe mais multiplexage en code (CDMA) avec une constellation multiplexée en fréquence (FDMA) qu'est GLONASS peut donner des résultats étranges, et le comportement du système chinois n'est pas encore

complètement clair dans notre esprit. Conservons donc uniquement GPS, qui lui transmet deux bandes de fréquences accessibles par U-Blox F9P, la vénérable L1 autour de 1575,42 MHz que nous avons souvent étudié et L2C (Civile) à 1227 MHz. Cette diversité en fréquence est importante car au travers de la relation de dispersion du plasma dans l'ionosphère, elle permet de remonter au délai ionosphérique indépendamment de toute correction externe (SBAS par exemple). De toute façon nous avons eu beau tenter de désactiver L2C de GPS, rien n'y fait, le F9P veut absolument le coupler à L1, et alimenter le F9P avec une antenne L1 uniquement dégrade tellement le résultat que nous sommes convaincus de la nécessité d'intégrer L1 et L2C dans les échanges. Finalement nous laissons Galileo, suffisamment proche de GPS pour bien s'intégrer avec lui. Au total nous nous retrouvons avec deux mesures par satellite GPS et une mesure par satellite Galileo donc $2 \times 16 + 18 = 50$ mesures dont une trame totale de $16 + 32 \times 50 = 1616$ octets ou 12928 bits/s. Cette valeur est proche de $SF = 7$ pour $B = 250$ kHz autorisé en Europe (10,9375 kb/s) et pour $B = 500$ kHz, nous gagnons un facteur de SF puisqu'en $SF = 8$ le débit est de 12500 bits/s. L'ambition de respecter une quelconque norme est de toute façon brisée par l'absence totale de respect des rapports cycliques imposés par LoRa pour permettre à de multiples interlocuteurs de cohabiter. Il serait possible de ne pas enfreindre la norme en dépassant le rapport cyclique si nous avions implémenté une détection de transmission avant d'émettre (LBT pour *Listen Before Talk*) mais une telle fonctionnalité n'a pas été mise en place dans notre application.

Ainsi, en "espérant" que un ou deux canaux de satellites ne sont pas occupés, nous pouvons ajuster les paramètres B et SF en vue de maximiser la portée en maintenant un débit acceptable. Cette stratégie s'est parfois retournée contre nous lors des essais par des pertes intermittentes de communication entre station de base et rover que nous attribuons à un nombre excessif de satellites visibles à un instant donné.

Cela nous amène donc finalement à naturellement nous interroger sur le nombre de satellites des constellations de navigation qui seront visibles à une date dans le future à laquelle la mesure est prévue sur le terrain, et pour une zone géographique donnée : pourrons nous "bénéficier" de la couverture réduite aux latitude élevées pour ne pas dépasser les longueurs de trames autorisées par LoRa ? La prévision du nombre de satellites de navigation depuis un site gégraphique donné à une date donnée dans le futur est détaillée dans l'envadré "Visibilité des satellites de navigation ?".

Visibilité des satellites de navigation ?

Nous pouvons nous interroger sur la visibilité des satellites de navigation, en particulier GPS, aux latitudes élevées. En effet, l'inclinaison des orbites de GPS est de 55° : pour des satellites volant à 20200 km d'altitude par rapport au sol, seront-ils visibles au Pôle Nord par 90° de latitude ou du Spitsberg par 79° de latitude, et si oui à quelle élévation au dessus de l'horizon ? Une approche géométrique simple suit le raisonnement suivant : un observateur au sol se trouve à une certaine latitude l donc l'angle $A = l - 55$ est positif si le satellite ne passe jamais au zénith, et la question est de connaître l'angle C entre l'observateur et le satellite. Les deux longueurs connues sont $b = 6400$ km le rayon de la Terre puisque l'observateur est supposé à sa surface, et $c = 6400 + 20200$ km la distance du centre de la Terre au satellite. Nous en déduisons $a^2 = b^2 + c^2 - 2bc\cos(A) = 21671$ km si l'observateur se trouve au Pôle Nord et par conséquent $\cos(C) = \frac{a^2+b^2-c^2}{2ab} = 45°$. Un satellite GPS ne s'élève jamais à plus de 45° de l'horizon au Pôle Nord. L'application numérique se généralise à toute latitude au dessus de 55° pour constater que les satellites GPS seront au mieux visibles par 59° d'élévation vers le sud mais seulement 32 degrés vers le Nord. La situation est à peine plus favorable pour le GLONASS russe dont les orbites sont inclinées de 64,8° et les satellites sont mieux visibles aux latitudes élevées.

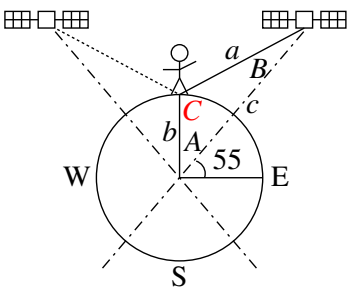


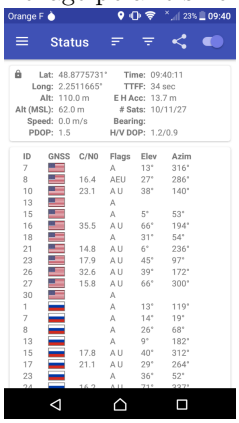
Figure 6: Considérations géométriques pour établir la visibilité des satellites GPS.

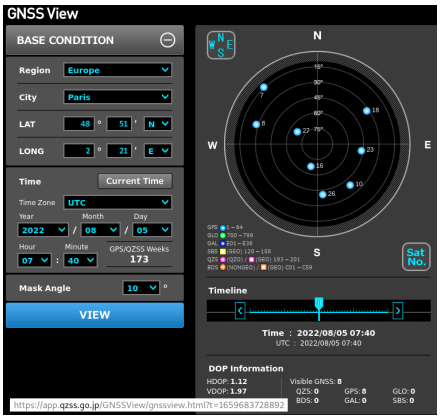
Une approche une peu moins naïve consiste à chercher à prédire la position des satellites sur la sphère céleste à une certaine date. N'ayant trouvé aucun outil libre permettant de faire ce calcul sous prétexte que des sites web fournissent de telles informations, nous nous sommes tournés vers Skyfield à `https://rhodesmill.org/skyfield/` alimenté par les éphémérides des satellites GPS fournis par Celestrak à `https://celestrak.org/NORAD/elements/gp.php?GROUP=gps-ops&FORMAT=tle` pour calculer la position des satellites à n'importe quel date au moyen de

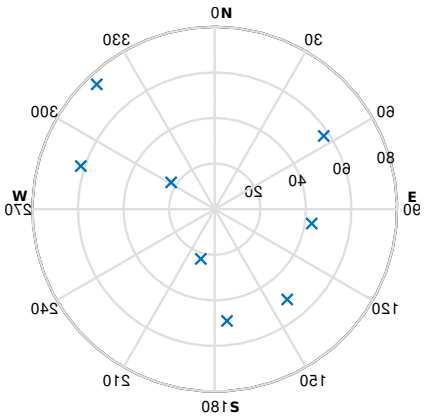
```
from skyfield.api import load, Loader, EarthSatellite
from skyfield.api import N,S,E,W, wgs84
from skyfield.timelib import Time
import numpy as np
import csv

load =Loader('./')
data =load('de421.bsp')
```

```
ts    =load.timescale()
earth=data['earth']
# lab   =wgs84.latlon(+48.9*N,  +2.3*E, elevation_m=143)
lab   =wgs84.latlon(+79.0*N,  +12.0*E,elevation_m=8)
satellites = load.tle_file('gps.tle')
# https://celestrak.org/NORAD/elements/gp.php?GROUP=gps-ops&FORMAT=tle
for k in range(0,len(satellites)):
  time =ts.utc(2022, 8, 5, 7.6667)  # 5 Aout 2022 7h
  diff=satellites[k]-lab
  topocentric=diff.at(time)
  alt, az, distance = topocentric.altaz()
  if (alt.degrees>10):
 print(satellites[k].name[13:-1:],distance.km,alt.degrees,az.degrees)
```

La validité de la prévision a été testée sur les satellites visibles au moment du calcul, le 5 Aout 2022 à 9h40 depuis Paris, à gauche la constellation vue par le récepteur d'un téléphone portable, au milieu la prévision fournie par `https://app.qzss.go.jp/GNSSView/gnssview.html?t=1659683728892` et à droite notre solution, volontairement inversée droite-gauche sous Inkscape puisque c'est la seule façon que nous ayons trouvée de placer l'ouest à gauche, GNU/Octave s'obstinant à refuser ce mode d'affichage polaire sinon :







L'accord entre les trois résultats est excellent, avec les véhicules 16 et 27 à 66° d'élévation et au total 7 satellites à plus de 20° d'élévation.

4 RIOT-OS pour programmer un microcontrôleur

Nous étant convaincus que la liaison radiofréquence sub-GHz implémentée par LoRa va répondre aux besoins de portée et de bande passante, il reste à trouver une plateforme de développement et un environnement logiciel. Didier Donsez fournit des plateformes à base de l'iM880B de IMST GmbH qui coûte moins d'une quinzaine d'euros chez le fabriquant quand ils ne sont pas en rupture de stock, combinant un microcontrôleur STM32L151 avec son cœur d'ARM Cortex M3, 32 KB RAM et 128 KB flash, et un transceiver LoRa Semtec Sx1272 selon la documentation technique disponible à `https://wireless-solutions.de/downloadfile/im880b-l/`. Parmi les trois documents, la *datasheet* fournit en particulier le brochage des divers périphériques et surtout pour la programmation, les documents *Range Test* et *RF Settings* donnent quelques indications de portée mais ce n'est que dans la note d'application AN016 en section 7 que nous apprenons que la sortie radiofréquence s'effectue sur la broche `PA-BOOST` du Sx1272 avec une puissance d'au moins 0 dBm et normalisé en Europe à une valeur maximale de +14 dBm.

Comme cette plateforme est supportée par RIOT-OS, nous acceptons cet environnement exécutif pour le développement logiciel en s'inspirant initialement de l'exemple grenoblois proposé à `https://github.com/CampusIoT/orbimote`. Cette application fonctionnelle est cependant bien trop complexe pour nous et fait appel à un protocole bien trop compliqué – LoRaWAN – pour la liaison point à point qui nous intéresse : nous devrons repartir d'une page blanche (un code source vide ?) pour la communication entre récepteurs GPS.

En effet, pour suivre la mode qui semble animer les informaticiens de reproduire ce qui existe déjà pour forcer les développeurs à incessamment recommencer leur travail à nouveau sur chaque plateforme matérielle, RIOT-OS est encore un environnement exécutif de plus proposé pour la première fois en 2009, dans la lignée de RTEMS (1993), eCos (1998), du défunt TinyOS (2000–2012), FreeRTOS (2003), NuttX (2007), ChibiOS (2007), ou encore de Zephyr (2017), ce dernier étant probablement créé ex-nihilo

pour affranchir ses sponsors privés des contraintes des licences libres des premiers. Rappelons que nous distinguons environnement exécutif – dans la nomenclature de RTEMS – qui offre un environnement qui rappelle au développeur les fonctionnalités d'un système d'exploitation (OS) incluant le multi-tâches, la gestion partagée de la mémoire et donc les priorités d'accès (MutEx et sémaphores) mais produit un unique exécutable monolithique alliant ordonnanceur, bibliothèques, et exécutables, contrairement à l'OS qui doit permettre le chargement dynamique des bibliothèques et exécutables et des ressources associées. Alors que l'environnement exécutif doit permettre de démontrer que les ressources disponibles permettent d'atteindre les performances attendues par l'application, l'OS peut éventuellement dépasser les ressources disponibles pour ne plus garantir de latences entre la requête d'un service et leur fourniture, tel que nous ne le savons que trop bien en ouvrant trop d'onglets de Firefox jusqu'à rendre l'interface graphique exécutée sous GNU/Linux tellement lente qu'il faut tuer le navigateur. L'environnement exécutif est donc souvent qualifié de temps-réel – latence bornée entre la requête de service et son obtention – alors que l'OS généraliste peut atteindre une condition de latence qui dépasse le délai acceptable, notamment dans des conditions critiques.

L'installation de RIOT-OS n'a absolument aucun intérêt à être décrit puisque triviale : le dépôt GitHub `https://github.com/RIOT-OS/RIOT` est cloné, et soit nous travaillons depuis un répertoire externe en prenant soin de renseigner l'arborescence contenant l'environnement exécutif en définissant la variable d'environnement `RIOTBASE` avec cet emplacement, soit nous travaillons dans le répertoire `tests/` et les exemples y fonctionnent directement. La seule contrainte est d'avoir un compilateur à destination de la cible considérée, dans le cas présent l'ARM Cortex M3 du STM32L1 : le cross-compilateur `arm-none-eabi-gcc` du paquet Debian `gcc-arm-none-eabi` et ses dépendances fera parfaitement l'affaire, RIOT-OS téléchargeant toutes les bibliothèques (CMSIS notamment) dont il aura besoin pour achever sa compilation.

La documentation de RIOT-OS existe à `https://doc.riot-os.org/` mais ne fait que survoler quelques points pratiques sur les options de compilation ou l'installation de l'environnement sans détailler les mécanismes internes de fonctionnement. Sans assistance des développeurs sur le forum d'échanges `https://forum.riot-os.org/` nous aurions été incapables d'aboutir : aucun ouvrage tels que ceux proposés par Richard Barry pour FreeRTOS ou ceux sur RTEMS ne semble exister sur RIOT-OS, et le développeur ne pourra que s'appuyer sur la somme relativement conséquente de cas de tests disponibles dans le répertoire d'installation sous `tests/`. Ce fut notre cas en partant de `tests/driver_sx127x` et en tentant de broder autour : alors que l'exemple de base est fonctionnel après compilation par `DRIVER=sx1272 BOARD=im880b make` et transfert vers la mémoire flash du microcontrôleur, soit par ST-Link au moyen du couple `openocd` et `gdb-multiarch` ou bien au moyen du port série par `stm32flash`, il nous faut apprendre à ajouter toutes les fonctionnalités additionnelles à savoir

— recevoir depuis le port série les trames de la station de base GPS
— transmettre ces trames par le modem LoRa. Compte tenu de la lenteur de la liaison LoRa et la faible durée du transfert filaire RS232 entre GPS et microcontrôleur, nous avons choisi de séquencer les deux opérations, d'abord en recevant toute la trame RS232 et ensuite en l'émettant sur la liaison radiofréquence. La fin de la communication GPS se détecte par ...
— ... un timer qui identifie si au bout d'un temps prédéterminé aucune nouvelle trame GPS n'a été reçue, marquant le début de l'émission LoRa,
— recevoir côté rover les trames LoRa, en extraire la charge utile et accumuler un tampon qui sera transmis à l'ordinateur exécutant RTKLib pour effectuer le calcul de position corrigée de la station mobile par rapport à la station statique.

Nous voyons donc naturellement quatre tâches distinctes se définir : une tâche chargée de recevoir sur RS232, une tâche chargée d'observer le temps entre deux réceptions RS232 et détectant l'absence de nouvelle trame après un intervalle de temps donné, induisant l'émission LoRa, et une troisième tâche chargée de recevoir les trames LoRa et finalement une quatrième tâche chargée de détecter la fin de la réception LoRa par l'absence de nouvelle trame après un certain délai en vue de l'émission du tampon sur le port série du rover. Dans un environnement de développement en C natif, nous aurions géré ces évènements concurrents dans les gestionnaires d'interruption correspondant, réception des caractères sur le port RS232 par l'interruption correspondante et émission des commandes vers le modem par interruption SPI, détection d'un délai sans activité *timeout* par gestionnaire d'interruption *timer*. Sous RIOT-OS, les gestionnaires d'interruption ISR (*Interrupt Service Routines*) sont déjà déclarées et nous ne faisons que profiter des fonctionnalités offertes sans les modifier. Dans ce contexte, la gestion en parallèle de la réception des tramẽs RS232 venant du récepteur GPS et l'émission des trames vers le modem LoRa se fera par des threads distincts qui semblent s'exécuter simultanément.

Une tâche sous RIOT-OS est un thread comme nous le créerions sous GNU Linux avec la bibliothèque `pthread`, sauf qu'ici nous explicitons la pile associée à chaque *thread* comme un tableau d'octets créé comme variable globale

```
#define SX127X_STACKSIZE   (THREAD_STACKSIZE_DEFAULT)
static char stacktx[SX127X_STACKSIZE];

void *_rstx_thread(void *arg) {...}

int main(void)
{static kernel_pid_t _rstx_pid=
 thread_create(stacktx, sizeof(stacktx), THREAD_PRIORITY_MAIN - 1,
               THREAD_CREATE_STACKTEST, _rstx_thread, NULL, "rstx_thread");
```

Chaque thread vit sa vie indépendamment de ses voisins sous le contrôle de l'ordonnanceur, avec éventuellement synchronisation des informations partagée par MutEx, signaux ou évènements. Dans le pilote original du SX1272, le gestionnaire d'interruptions est pris en charge en envoyant un message chaque fois qu'un événement lié aux échanges par le modem LoRa est reçu. Nous avions été prévenus dans `https://forum.riot-os.org/t/uart-communication-over-lora/3639/9` que ce mécanisme de messages est lent et serait la cause de pertes de messages. En effet nous avons rapidement constaté qu'alors que le pilote fourni fonctionnait bien pour des messages courts tapés au clavier, les quelques 400 à 500 octets à transmettre entre récepteurs GPS étaient lamentablement perdus, ou plus grave induisaient des incohérences de piles qui se traduisaient par un arrêt brutal d'une des deux applications, émetteur ou récepteur. Autant la perte de données temporaire peut être acceptable, autant le plantage d'un programme alors que la station a été installée plusieurs heures avant le temps nécessaire à accéder au site d'observation pour ne pas fournir le service attendu sur le terrain serait tout à fait inacceptable. Au pire nous laisserons un chien de garde *watchdog* réinitialiser le microcontrôleur en cas de perte de service, mais il serait souhaitable en amont que le plantage ne survienne pas.

La première solution palliative, peu satisfaisante, a été d'éliminer le système de messages pour traiter les événements au plus tôt lors de leur réception : cette solution résout en partie le problème de corruption de la pile, mais par une architecture peu satisfaisante car elle revient à gérer une interruption dans son gestionnaire ISR, une solution peu souhaitable puisque l'ISR doit toujours être aussi courte que possible. Tel que préconisé par les développeurs RIOT, remplacer le message par l'événement maintient la performance tout en déportant la gestion des évènements dans un thread dédié

```
#include "event/thread.h"
netdev_t *eventdev;

static void _handler_high(event_t *event) {
    (void)event;
    eventdev->driver->isr(eventdev); // gestion de l'evenement
}

static event_t event_high = { .handler=_handler_high };

static void _event_cb(netdev_t *dev, netdev_event_t event)
{if (event == NETDEV_EVENT_ISR)
    {eventdev=dev;                    // ISM memorise l'evenement  et le transmet
     event_post(EVENT_PRIO_HIGHEST, &event_high); // ... au handler via un event
    }
    else {
[...]
```

À coté de cette gestion des évènements de communication sur réseau radiofréquence LoRa, deux threads se chargent des transmissions RS232 sous la supervision d'un timer chargé de détecter la fin de communication depuis le récepteur GPS, un pour la réception des trames RS232 que nous supposons prise en charge par la bibliothèque `stdio` activée dans le `Makefile` par `USEMODULE += shell` :

```
void *_rstx_thread(void *arg)
{   (void)arg;
    char val;
    while (1)
       {stdio_read (&val,1);
        if (compteurin<BUFSIZE)
          {message[compteurin]=val;
           compteurin++;
          }
       }
}
```

et un pour envoyer vers le port série les trames reçues par LoRa, toujours sous la supervision de `stdio`

```
void *_rsrx_thread(void *arg)
{   (void)arg;
    xtimer_ticks32_t last_wakeup;
    uint32_t interval = 25600*2;
    unsigned int oldindexin=0;
    while (1)
       {last_wakeup = xtimer_now();
```

```
        xtimer_periodic_wakeup(&last_wakeup, interval);
        if (indexin>0)
          {if (oldindexin==indexin)
             { if (indexin>BUFSIZE) {indexin=BUFSIZE;} // buffer overflow
               wdt_kick();  // gestion periodique du chien de garde
               stdio_write (message,indexin);
               indexin=0;   // content of buffer has been dumped, restart
              }
            else {oldindexin=indexin;} // still receiving
          }
       }
}
```

dont toute l'initialisation et la gestion des *timers* est inspirée de `tests/xtimer_periodic_wakeup/`. Finalement, la dernière tâche est la boucle infinie dans le programme principal qui effectue les transactions LoRa :

```
oldcompteurin=0;
while (1)
  { last_wakeup = xtimer_now();
    xtimer_periodic_wakeup(&last_wakeup, interval);
    if (compteurin>0)                                  // RS232 received
      {if (compteurin==oldcompteurin)                 // but long silence since
         {indice=0;
          wdt_kick();
          do{if (compteurin>=SX127X_LORA_MSG_QUEUE)
               {do {res=send_cmd(&message[indice],SX127X_LORA_MSG_QUEUE);
                   } while (res!=0);
                indice+=SX127X_LORA_MSG_QUEUE;
                compteurin-=SX127X_LORA_MSG_QUEUE;
                last_wakeup = xtimer_now();
                xtimer_periodic_wakeup(&last_wakeup, 30);
               }
             else
              {do {res=send_cmd(&message[indice],compteurin);
                  } while (res!=0);
               compteurin=0; oldcompteurin=0;
              }
            if (res == -ENOTSUP) { puts("*");}
           } while (compteurin>0);
         }
        else
          {oldcompteurin=compteurin;}
       }
  }
```

La partie `wdt_kick();` qui répond à l'initialisation `wdt_setup_reboot(0, 5000); wdt_start();` s'assure que tant que le programme tourne correctement, *i.e.* que des trames GPS sont échangées chaque seconde, le chien de garde n'induit pas de réinitialisation du microcontrôleur, tandis que si personne n'a réveillé le chien de garde dans un intervalle de 5 s (l'argument de `wdt_setup_reboot()` est le délai en millisecondes) alors le microcontrôleur est redémarré, probablement à cause d'un dysfonctionnement du programme sous RIOT puisque nous sommes confiants que le GPS continue à envoyer ses trames de référence de la base.

En fin de compte, après bien des tâtonnements (noter par exemple que la taille des tampons gérant les paquets LoRa diffère en émission et en réception), nous finissons par obtenir une application qui tourne de façon stable plusieurs heures d'affilée et en laquelle nous avons suffisamment confiance pour installer une station de base communiquant par LoRa. Ce développement a été l'occasion d'appréhender la programmation multitâches et l'utilisation de périphériques selon des techniques efficaces (DMA, ISR) généralement gourmandes en ressources et temps de développement sur un microcontrôleur aux ressources avancées qui justifie de s'appuyer sur une infrastructure aussi complexe que RIOT. Avoir un peu de maîtrise des concepts sous jacents au-delà de quelques échanges sur un forum eut été souhaitable pour valider les choix techniques et en appréhender les conséquences en termes de ressources (temps de calcul, veille et donc consommation, latences).

La fréquence de communication peut être choisie arbitrairement dans la bande allouée en Europe (863–870 MHz) puisque nous développons les deux interlocuteurs de la liaison point à point, mais par soucis de compatibilité avec le matériel existant tel que par exemple les routeurs (*gateways*), Didier Donsez nous encourage à sélectionner 868,3 MHz qui d'après `https://www.thethingsnetwork.org/docs/lorawan/frequency-plans/` supporte officiellement une transmission en SF7 et 250 kHz de bande passante.

5 Portée de la communication

Nous avons donc décrit les trames UBX produites par les récepteurs Zed-F9P, décrit la couche physiques et les paramètres de la liaison radiofréquence LoRa, et mis en œuvre une liaison fonctionnelle entre la station de base et le rover. Nous concluons cette étude en analysant l'impact des divers paramètres de la couche physique sur la distance de communication :

1. un émetteur LoRa émet une puissance $P_E = +14$ dBm (25 mW) dans une antenne de gain G_E. Si cette antenne est supposée isotrope, le gain d'antenne est unitaire ou 0 dBi (i comme isotrope)
2. la puissance émise est rayonnée à une distance d sur une sphère de surface $4\pi d^2$ et arrive vers un récepteur lui même muni d'une surface de réception qui se mesure en fraction du carré de la longueur d'onde divisée par les 4π steradians de la couverture isotropique de l'antenne de réception : l'équation de Friis relire la puissance reçue P_R à la puissance émise, la distance de liaison et la longueur d'onde par
$$\frac{P_R}{P_E} = \frac{\lambda^2}{(4\pi)^2 d^2}$$
ou en passante en échelle logarithmique, les pertes de propagation en espace libre (*Free Space Propagation Loss*) s'expriment comme
$$FSPL = 20\log_{10}(d) + 20\log_{10}(f) - 147,55$$
avec $20\log 10\left(\frac{c}{4\pi}\right) = -147,55$ quand la vitesse de la lumière est exprimée $c = 3 \cdot 10^8$ m/s pour des unités cohérentes de f en Hz et d en m
3. la limite de détection du récepteur est déterminée par la durée de la corrélation SF et la largeur de bande sur laquelle le bruit est intégré B. Doubler la bande passante double le bruit thermique, tandis que augmenter SF d'une unité double la durée du chirp et améliore d'un facteur 2 le rapport signal à bruit lors de la corrélation : dans le bilan de liaison, tous ces effets impactent comme $10\log_{10}(2) = 3$ dB. En revenant en échelle linéaire, la puissance qui décroît comme le carré de la distance sera impacté d'un facteur 0,71 par une dégradation d'un facteur 2 de la limite de détection.

Pour résumer, [6] indique que la limite de détection de LoRa est autour de -123 dBm à $SF = 7$ pour une bande passante de 125 kHz, et nous savons que nous perdons 3 dB de limite de détection en abaissant SF d'une unité ou en doublant la bande passante. En partant de 25 mW (+14 dBm) sur une antenne isotrope, la puissance reçue de -123 dBm est atteinte à $f = 868$ MHz pour une distance $d = 200$ km. En passant sur une bande passante de 500 kHz, la portée est réduite à 100 km dans des conditions idéales, donc hors zone de Fresnel où l'onde pourrait interagir avec le sol et évidemment hors milieu urbain pour une propagation en espace libre. Sur une portée de 100 km, la **zone de Fresnel** où l'onde est susceptible d'interagir avec des réflecteurs sur son chemin est un rayon de 185 m qui indique la hauteur au-dessus du sol à laquelle devrait se trouver l'antenne. Pour une portée plus raisonnable de 10 km, cette hauteur reste tout de même 60 m, expliquant pourquoi les antennes se trouvent toujours en haut de long mâts et jamais à proximité du sol. Cette portée, *a priori* incroyable, est de l'ordre de grandeur de celle observée par Didier Donsez lors des vols de ballons sonde portant des émetteurs LoRa tel que décrit à `https://gricad-gitlab.univ-grenoble-alpes.fr/thingsat/public/-/tree/master/balloons`, donc bien loins de la zone de Fresnel lorsque leur altitude dépasse les quelques centaines de mètres. Quelques exemples de mesures préliminaires en terrain découvert sont fournis sur la Fig. 7 et 8, la période estivale étant particulièrement favorable aux expériences en l'absence de footballeux sur les terrains en plein air et d'étudiants sur les campus universitaires.

Les mesures en coordonnées sphériques WGS84 fournies par RTKLib sont converties dans une approximation un peu grossière d'une Terre sphérique en considérant que la longueur de l'équateur est, par une définition un peu obsolète à ce jour, de 40000 km (ou $4 \cdot 10^7$ m) et donc la variation de latitude $d\vartheta$ (en degrés) est convertie en variation de distance dy par $dy = 4 \cdot 10^7 \cdot dy/360$ et de la même façon pour la longitude en tenant compte de la longueur du parallèle à la latitude ϑ : $dx = 4 \cdot 10^7 \cdot \cos(\vartheta) dx/360$. Le résultat de ces conversions sont reproduites en Fig. 8 en bas sur une distance base-rover de l'ordre du kilomètre, avec une résolution centimétrique tel qu'annoncé dans le titre. La conversion propre se fait bien entendu en chargeant les points de mesure issus de RTKLib dans `qgis` et (bouton de droite) *Export→ Save Features As ...* en définissant le CRS projeté, par exemple pour la France WGS84/UTM32N ou pour la Norvège WGS84/UTM33N. Noter qu'avec la version 3.22 de QGis au moins, il faut en plus définir tout en bas des options *GEOMETRY* et indiquer *AS_XYZ* pour sauver en format ASCII les coordonnées d'entrée et de sortie.

Les paramètres de configuration de LoRa font jouer sur des facteurs ±3 dB quand la moindre antenne directive fera gagner 10 dB d'un coup : par exemple une antenne Yagi-Uda de 6 éléments côté station de

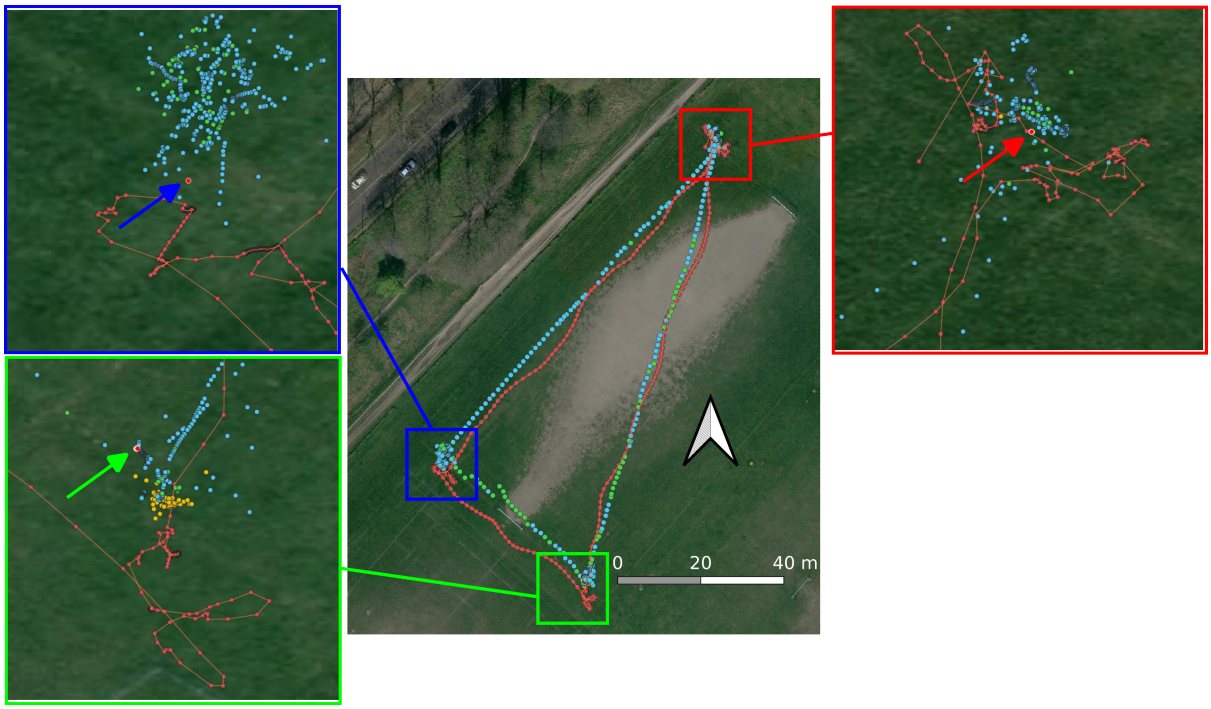


FIGURE 7 – Essais de portée et de résolution depuis le bois de Boulogne à Paris. Chaque flèche rouge sur les zooms indique la solution (points rouges) indiqués comme “Fix” par RTKLib. Les points bleus clairs sont les solutions “Float” et les points oranges sont les points “Single” quand la base voit trop de satellites et n'arrive pas à transmettre par liaison LoRa toutes les informations pour que RTKLib trouve une solution différentielle chaque seconde sur le rover. La trace rose est acquise par le récepteur GNSS d'un téléphone portable Sony Z5 placé à proximité de l'ordinateur exécutant RTKLib.

base, pointée en direction du site de mesures, amènera 10 dB de gain et donc une extension de la portée de la mesure d'un facteur 3. En pratique, la loi ne s'intéresse pas à la puissance électrique émise mais à la puissance radiofréquence rayonnée : la puissance équivalente rayonnée (EIRP) indique que si la puissance maximale autorisée est 25 mW (+14 dBm) et que nous utilisons une antenne de 10 dB de gain, alors la puissance électrique maximale autorisée est $+14-10 = +4$ dBm afin que la puissance maximale rayonnée dans la direction de gain maximal de l'antenne reste sous la norme. Mais qui respecte la loi... personne ne viendra vérifier si nous plaçons une antenne de gain +10 dB en sortie des +14 dBm électriques émis par le Sx1272 si nous manquons de portée. Le bénéfice sur le bilan de liaison est aussi valable en équipant le récepteur, *rover*, d'une telle antenne directive sous hypothèse de connaître la direction vers la station. Avec $10 + 10 = 20$ dB sur le bilan de liaison global, la portée est décuplée par rapport au monopole équipant chaque interlocuteur, au détriment d'une liaison directive faisant l'hypothèse de connaître au moins approximativement l'emplacement des deux circuits.

Nous avons validé expérimentalement le bilan de liaison en plaçant la station sur la balcon au 4ème étage d'un bâtiment à Besançon donc à une quinzaine de mètres d'altitude, et en observant le signal reçu à une distance de 5 km et une altitude relative de 270 m d'après géoportail : cette condition devrait nous placer en dehors de la zone de Fresnel qui est d'un rayon de 20 m pour une distance de communication de 5 km. Les deux stations, fixe et mobile, sont équipées d'une antenne monopole de gain 2,15 dBi nominal. Nous observons un *Received Signal Strengh Indicator* RSSI de 104 à 5 km, donc d'après les documentations Semtech et RIOT-OS la puissance reçue en dBm. Nous avons par ailleurs observé en milieu urbain la réception de signaux exploitables pour un RSSI de -119 dBm et la perte de liaison par corruption des données reçues pour un RSSI de -123 dBm. Nous en concluons que la portée maximale en espace libre est $5 \text{ km} \times 10^{(119-104)/20} = 5 \times 5,6 = 28$ km dans ces conditions. Ce résultat est cohérent avec la limite de détection de -120 dBm pour une bande passante de communication de 250 kHz et SF=7 annoncé dans [7].

6 Utilisation pratique

La raison d'être de tout ce travail est de géolocaliser des capteurs avec une résolution centimétrique sur un glacier arctique au Spitsberg (79° N) : d'une part nous espérons ainsi retrouver au printemps les

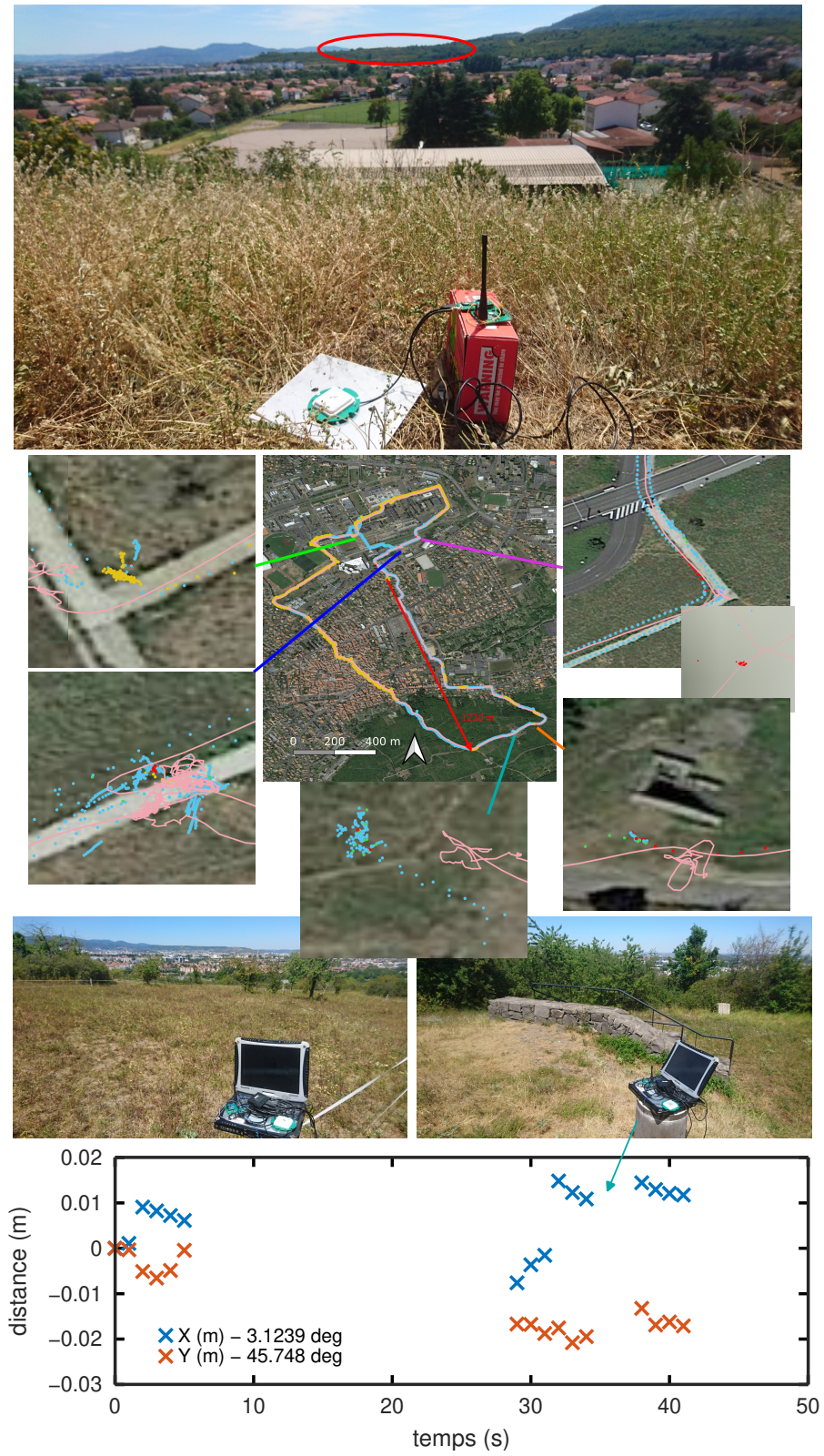


FIGURE 8 – Essais de portée depuis le campus universitaire des Cézeaux à Clermont-Ferrand vers la crête adjacente située à une distance de 1230 m à vol d'oiseau. En bas, la résolution sur la mesure au point le plus lointain est de l'ordre de ±2 cm, un gain d'un facteur d'au moins 100 par rapport à une mesure GPS simple. Les points bleus clairs sont les solutions "Float" et la trace rose est acquise par le récepteur GNSS d'un téléphone portable Sony Z5 placé à proximité de l'ordinateur exécutant RTKLib. En rouge en haut : la crête atteinte avec le rover visible depuis la station de base, à une distance d'un peu plus d'1 km en vue directe.

dispositifs enterrés sous quelques mètres de neige sans devoir déplacer des mètres carrés d'un manteau neigeux qui dépasse rapidement les deux mètres d'épaisseur pour retrouver des capteurs ou balises, mais aussi observer par mesure GPS la hauteur des balises d'ablation et ainsi comparer l'évolution de l'altitude absolue de la surface du glacier (mesure GNSS) avec l'ablation mesurée aux balises représentant l'épaisseur de glace fondue. Un glacier polaire, contrairement à un glacier alpin, ne se déplace que très peu et son mouvement horizontal a relativement peu d'intérêt puisqu'il ne compense que de façon négligeable sa fonte ("déplacement" vertical) qui s'observe de façon évidente par le retrait de la langue du glacier.

Pour ce faire, nous devons référencer les mesures du *rover* mobile à la station fixe supposée toujours localisée au même emplacement lors de chaque session de mesures. Le contexte de déploiement initialement escompté est illustré en Fig. 9 avec une station de base proche d'une source d'alimentation et un emplacement facilement accessible, et un rover sur un éperon rocheux servant de point connu mais d'accès plus compliqué au cours de cette mesure.

FIGURE 9 – Gauche : installation de la station de base au sommet d'un mat à proximité de la base Jean Corbel (gauche), capable de communiquer avec le rover qui sera, au cours de cette mesure, placé sur un mat (cercle rouge sur la photographie de gauche) portant une station météorologique sur un éperon rocheux surplombant le glacier en cours d'étude (droite). L'analyse qui suivra indique qu'il serait judicieux de placer la station de référence sur cet éperon rocheux pour couvrir tout la bassin glaciaire, sous réserve de trouver une source d'alimentation suffisante pour la durée des sessions de mesures. Au premier plan, l'acquisition du modèle numérique de terrain par LiDAR (photographie F. Tolle, ThéMA, Besaçon).

Malgré le succès de cette mesure avec l'obtention d'une solution centimétrique en un peu moins de 15 minutes, nous avons constaté l'incapacité à communiquer avec la majorité des sites de mesure escomptés compte tenu de la forme bombée du glacier qui cache de la vue directe la majorité de sa surface au front où se trouve la station de référence (voir encadré ci-dessous). Une mesure de l'écart type sur la position dans le plan horizontal et selon l'axe vertical est proposée en Fig. 10, illustrant la décroissante exponentielle de l'erreur sur la mesure différentielle pour atteindre la résolution centimétrique et passer de la solution *float* à *fix* que nous recherchons.

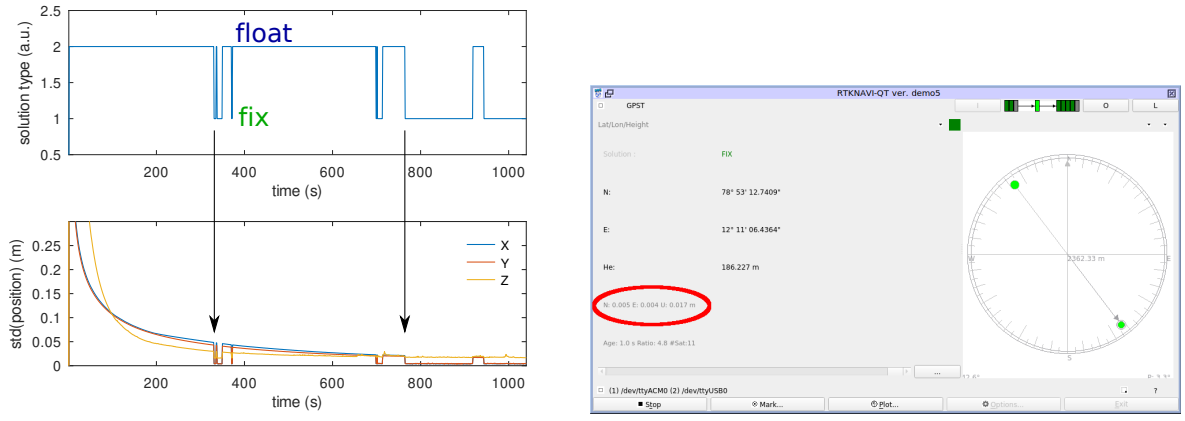
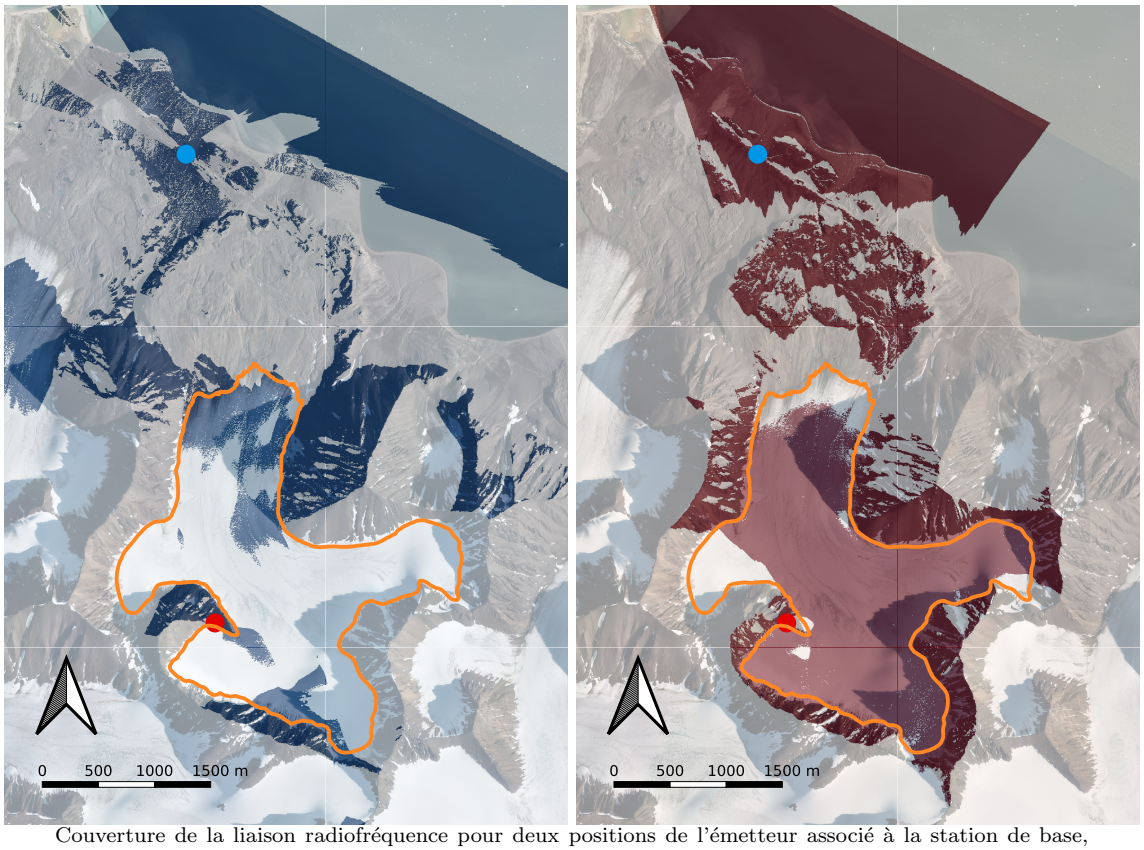


FIGURE 10 – Gauche : en haut la nature de la solution, 2 pour *float* et 1 pour *fix*, et en bas l'écart type sur la position dans les 3 axes, aussi indiqué (ellipse rouge) sur l'interface graphique de RTKLib visible à droite (les symboles verts indiquent une solution centimétrique tel qu'en atteste l'écart type indiqué).

Couverture de la liaison radiofréquence ?

Compte tenu des mesures sur le terrain, nous pouvons nous interroger sur la capacité à prédire la zone couverte, au-delà d'une simple analyse de portée par bilan de liaison et conservation de l'énergie, mais du fait des obstacles entre interlocuteurs. En effet, une liaison radiofréquence Ultra-Haute Fréquence (UHF incluant 868 MHz) – et *a fortiori* micro-ondes (e.g. 2,4 GHz ou 5,8 GHz) – se comporte comme une liaison optique imposant que les deux interlocuteurs soient en vue direct l'un de l'autre, non-obstant les effets de réflexion disponibles en milieu urbain mais peu probables dans l'environnement naturel qui nous intéresse ici. L'effet d'ombre portée peut se calculer au moyen du greffon QGis *Visibility analysis plugin* tel que décrit à `https://landscapearchaeology.org/2020/viewshed-tutorial/`. Nous ne pouvons que lourdement insister comme le fait l'auteur de ce greffon à `https://landscapearchaeology.org/2020/wgs/` sur la nécessité de travailler en coordonnées projetées sur un plan tangent local (UTM) et *non* en coordonnées sphériques sur lesquelles les calculs sont généralement faux. Nombre de traitements d'informations géoréférencées s'appuient sur un modèle numérique d'Élévation (DEM pour *Digital Elevation Model*) ou modèle numérique de Terrain (DTM pour *Digital Terrain Model*). Un DEM s'obtient par traitement photogrammétrique d'images stéréoscopiques acquises par satellite ou par photographie aérienne, ou par mesure de temps de vol d'impulsions laser par un LiDAR ou radiofréquence par un RADAR. La première méthode est utilisée à grande échelle pour par exemple produire l'ArcticDEM de `https://www.pgc.umn.edu/data/arcticdem/` mais ce traitement photogrammétrique diverge souvent sur les zones uniformément couvertes de neige en absence de structure sur laquelle effectuer un calcul de corrélation, et l'analyse de la zone qui nous intéresse indique nombre d'artéfacts de traitement qui interdisent son utilisation dans un traitement un peu fin d'ombre projetée. Nous utilisons le LiDAR pour reproduire chaque année un modèle du glacier et des versants mais ces données ne sont pas publiquement disponibles (compte tenu de la taille du nuage de points plus que par volonté de dissimuler l'information) tandis que pour la dernière technique, le SRTM *Shuttle Radar Topography Mission* qui a été embarqué en 2000 sur la navette spatiale américaine n'a couvert que les latitudes de $+60°$ à $-56°$ donc excluant l'Arctique et l'Antarctique. Ainsi par soucis de reproductibilité par les lecteurs, nous nous appuierons sur un DEM un peu ancien fourni par l'institut polaire norvégien à `https://publicdatasets.data.npolar.no/kartdata/S0_Terrengmodell/Delmodell/` qui bien que acquis en 2010 (fichier `NP_S0_DTM5_2010_13828_33.zip`) reste pertinent malgré l'évolution de la surface du glacier qui perd de l'ordre du mètre de glace chaque année : nous insistons sur la méthode plus que sur le résultat dans cette analyse. Afin de prédire la zone couverte par une station de base munie d'un émetteur LoRa en divers points du bassin glaciaire, nous proposons

1. fournir dans un fichier texte d'entête `X Y N` la longitude, latitude et un identifiant de la localisation possible de la station de base (une position par fichier texte)
2. insérer l'information de position de la station de base dans QGis par `Layer` → `Add Layer` → `Add Delimited Text Layer...` en prenant soin de se placer dans un référentiel projeté (WGS84/UTM33N pour ce qui nous concerne)
3. insérer le DEM par `Layer` → `Add Layer` → `Add Raster Layer...`
4. Après avoir installé le greffon *Visibility analysis* dans `Plugins`, rechercher dans `Processing` → `Toolbox` le menu `Visibility Analysis`
5. Commencer par le pré-traitement proposé par `Create viewpoints` → `Create viewpoints` en lui fournissant le point représentant la station de base, le DEM, le rayon sur lequel effectuer l'analyse (nous augmentons cette valeur à 8 km, imposant une ressource de calcul significativement plus importante que les 5 km proposés par défaut), et l'altitude du point *par rapport au DEM* (nous avons utilisé 4 m pour indiquer que l'émetteur est placé au sommet d'un pilone). Le résultat est sauvé dans une couche virtuelle temporaire et n'a aucun intérêt à être conservé.
6. Ce premier calcul rapide est suivi de `Analysis` → `Viewshed` qui prend en entrée la couche virtuelle comme "Observer location" et le DEM, les autres valeurs par défaut étant conservées, pour finalement sauver le résultat dans un fichier puisque ce calcul peut s'avérer relativement long.



Couverture de la liaison radiofréquence pour deux positions de l'émetteur associé à la station de base, à gauche à la base Jean Corbel au pied du glacier (cercle bleu), avec les zones accessibles en bleu foncé et les zones inaccessibles en blanc, avec la photographie aérienne fournie par l'Institut polaire norvégien en arrière plan par transparence. Droite : analyse identique pour la station placée sur un éperon rocheux surplombant le glacier (cercle rouge), avec en rouge les zones couvertes par la liaison radiofréquence et en blanc les zones inaccessibles. De ce second point de vue, seul le cirque le plus à l'ouest et une petite zone du cirque de l'est ne sont pas couverts, ainsi que la zone la plus basse de la langue qui ne tardera de toute façon pas à fondre. La limite du glacier délimitant la zone d'intérêt est indiquée par le trait orange.

Les résultats de cette analyse (figure ci-dessus), pour une localisation de la station de base au pied du glacier dans la station qui héberge ces recherches (donc facilement accessible voir avec une source d'énergie importante) ou sur un éperon rocheux surplombant le bassin glaciaire (plus difficile d'accès), démontre clairement la supériorité de la seconde localisation qui permet de couvrir presque l'ensemble de la surface du glacier, quand la forme convexe du volume de glace de la langue cache une grande partie de la surface du glacier lorsque observé depuis son front. Une installation pérenne avec alimentation autonome devra donc être développée pour permettre une mesure fine de position des capteurs et balises distribués sur le glacier, la station de base devant absolument se trouver sur l'éperon rocheux surplombant le site de l'étude.

7 Conclusion

Nous nous sommes efforcés de mettre en place un système à coût abordable de mesure différentielle centimétrique par constellation de navigation par satellite, autonome pour ne pas s'appuyer sur une infrastructure commerciale de communication par téléphonie mobile inexistante dans l'environnement qui nous intéresse en arctique, avec une portée de plusieurs kilomètres en communicant sur une porteuse sub-GHz. Pour ce faire, nous avons appris à maîtriser un composant propageant des signaux respectant la norme propriétaire LoRa, et ce grâce à l'infrastructure de développement RIOT-OS.

On notera qu'un projet similaire tente de fournir un signal libre de référence pour la mesure centimétrique par mesure différentielle : il s'agit du projet Centipède décrit à `https://docs.centipede.fr/`. Dans ce contexte, une Raspberry Pi connectée à internet reçoit les signaux de référence d'une station de base (récepteur U-Blox Zed-F9P avec antenne multi-fréquences idéalement) qui sont diffusés vers un serveur qui redistribue l'information par le réseau numérique s'appuyant sur le réseau de téléphonie mobile. Nous avons profité d'un excédent de récepteurs Zed-F9P et d'une Raspberry Pi4 inutilisée pour contribuer à ce réseau puisque la région autour de Besançon s'avérait particulièrement pauvre en signaux de référence : l'installation est très bien documentée et ne nécessite que de transférer (`dd`) une image fournie sur le site web du projet vers une carte SD. Il ne reste plus qu'à tirer le meilleur parti des fonc-

tionnalités de ce service. Malheureusement les administrateurs du projet n'ont pas encore connecté notre station de référence au réseau, laissant une lacune de couverture autour de Besançon. Cependant, la documentation d'installation de la station de base Centipède à `https://docs.centipede.fr/docs/base/positionnement.html` nous informe de la stratégie à suivre pour renseigner avec exactitude la position de la station de base tel que le requiert RTKLib (pour sa version originale, et non le dérivé `demo5` que nous avons utilisé ici). En collectant pendant environ 48 h les données brutes du récepteur GNSS qui servira de station de base et en convertissant ces mesures en format RINEX standardisé pour les échanges, il est possible de soumettre à une analyse par l'IGN à `https://rgp.ign.fr/SERVICES/calcul_online.php` qui comparera les mesures avec celles de son réseau GNSS (au moyen de Bernese que nous avons déjà cité dans le texte) permanent afin d'établir finement la position de référence qui sera l'information renseignée pour la localisation de la station de base dans `rtknavi_qt` à la place de la recherche dynamique par solution simple qu'autorise `demo5`.

Didier Donsez nous informe en perspective de ce travail que les composants Semtech SX126x (`https://www.semtech.com/products/wireless-rf/lora-connect/sx1261`), LLCC68 et LR1120 permettent d'augmenter le débit de communication par un Spreading Factor aussi faible que SF=5 (contre SF=7 dans nos exemples), quadruplant le débit. Les processeurs STM32WL55JCx qui embarquent un SX126x sont supportées par RIOT-OS avec par exemple `https://github.com/RIOT-OS/RIOT/tree/master/boards/nucleo-wl55jc`.

L'ensemble des logiciels proposés dans ce document est disponible à `https://github.com/jmfriedt/RIOT_tests/`.

Remerciements

Didier Donsez et ses collègues du Laboratoire d'Informatique de Grenoble a proposé l'utilisation de RIOT-OS pour appréhender la communication sur réseau LoRa et a gracieusement fourni les plateformes de développement. Les auteurs de RIOT-OS ont patiemment répondu à nos interrogations, notamment sur le forum de discussion `https://forum.riot-os.org/t/uart-communication-over-lora/3639`. Les missions au Spitsberg qui motivent cette étude sont financées par l'Institut polaire Paul-Émile Victor (IPEV) et la Région Franche-Comté.

Références

[1] E.D. Kaplan & C.J. Hegarty, *Understanding GPS Principles and Applications Second Edition*, Artech House (2006) Table 7.4 p.322 disponible à `https://d1.amobbs.com/bbs_upload782111/files_33/ourdev_584835O21W59.pdf`

[2] T. Everett, T. Taylor, D.-K. Lee, D.-M. Akos, *Optimizing the Use of RTKLIB for Smartphone-Based GNSS Measurements*, MDPI Sensors **22** 3825 (2022) à `https://www.mdpi.com/1424-8220/22/10/3825/htm`

[3] A. Filipova, *The town with the cleanest air in the world*, BBC News (29 Jul. 2022) à `https://www.bbc.com/future/article/20220728-the-town-with-the-cleanest-air-in-the-world`

[4] M. Knight, *Decoding the LoRa PHY*, Chaos Computer Club 33c3 (2016) à 40 minutes de `https://www.youtube.com/watch?v=NoquBA7IMNc`

[5] *LoRaWAN Spreading factor, range, data rate in LoRa System* à `https://www.rfwireless-world.com/Terminology/LoRaWAN-Spreading-factor-Range-and-Data-Rate.html`

[6] `https://lora-developers.semtech.com/uploads/documents/files/Understanding_LoRa_Adaptive_Data_Rate_Downloadable.pdf`

[7] A. Lavric & V. Popa, *Performance Evaluation of LoRaWAN Communication Scalability in Large-Scale Wireless Sensor Networks*, Wireless Communications and Mobile Computing (2018) à `https://www.hindawi.com/journals/wcmc/2018/6730719/tab3/` reproduit un tableau d'une documentation technique de Semtech que nous ne retrouvons plus.