Exploitation des signaux de référence de navigation par satellite pour un positionnement centimétrique : RTKLib fait appel à Centipède et l'IGN pour afficher dans QGis

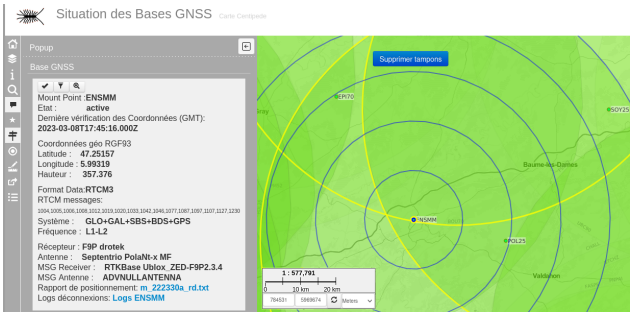
J.-M Friedt, FEMTO-ST département temps-fréquence, Besançon

Nous exploitons un récepteur de navigation par satellites "faible" coût U-Blox Zed-F9P s'appuyant sur divers réseaux de stations de référence (Centipède, IGN) pour utiliser efficacement la bibliothèque RTKLib pour positionner un dispositif mobile avec une résolution centimétrique. Les informations ainsi produites sont exportées en temps-réel dans des Systèmes d'Informations Géographiques (GIS) tels que QGis (GNU/Linux) ou QField (Android) pour être intégrées dans l'ensemble des informations géoréférencées considérées au cours d'une étude sur le terrain.

Nous avons récemment discuté [1] de la mise en œuvre de récepteurs de signaux de navigation par satellite (GNSS) en vue d'un positionnement centimétrique, et en avons profité pour développer un système de communication radiofréquence longue portée s'appuyant sur le protocole LoRa à 868 MHz afin de s'affranchir de restrictions sur les émissions radiofréquences dans les bandes de téléphonie mobile ou industrielles, scientifiques et médicales (ISM) autour de 2,45 et 5,8 GHz sur notre site d'étude. Ce faisant, nous avons découvert le projet Centipède qui vise à produire un maillage dense de stations GNSS de référence (dites stations de base) en vue de permettre un positionnement centimétrique de dispositifs mobiles (dits rovers) en s'affranchissant des délais ionosphériques et autres effets corrélés entre station de base et rover. En effet, le délai ionosphérique, variable au cours du temps et de l'activité solaire, est la principale source d'incertitude du temps de vol des ondes électromagnétiques depuis les satellites de navigation en orbite moyenne (20000 km de la surface de la Terre) vers le récepteur au sol, induisant une incertitude de position de quelques mètres (pour rappel, 1 ns de retard se traduit par 30 cm d'erreur de trilatération du récepteur). En imposant la position d'une station de base supposée connue, un rover suffisamment proche – typiquement dans un rayon d'une trentaine de kilomètres – subira les mêmes délais ionosphériques que la station de base, effet qui pourra donc être corrigé par un logiciel approprié tel que RTKLib. Centipède dissémine ces informations de station de base par réseau compatible internet (IP), accessible en particulier par le réseau de téléphonie mobile moyennant un abonnement relativement courant actuellement.

Station de référence Centipède

Pour cette étude, mais surtout par conviction qu'un institut de recherche public (FEMTO-ST) a pour vocation de fournir un service au public, nous avons installé une station de référence Centipède qui servira, à côté de la station de référence de l'Institut National de l'information Géographique et forestière (IGN – il faut chercher les bonnes lettres dans le nom actuel pour justifier de l'acronyme ancien) de l'Observatoire de Besançon, de *basestation* par rapport à laquelle localiser le *rover*. L'installation est illustrée ci-dessous à gauche, avec une antenne Septentrio PolaNt-x MF multi-constellations sur le toit d'un bâtiment de l'École Nationale Supérieure de Mécanique et des Microtechniques (ENSMM) qui héberge le département Temps-Fréquence de l'institut FEMTO-ST, en vue dégagée du ciel, et à droite l'interface graphique des stations de Centipède identifiant les propriétés de notre station de base et celles environnantes. La documentation de Centipède (`https://docs.centipede.fr/docs/base/`) pour installer une station de base est impressionante de clareté et de simplicité. L'identifiant de cette station de base est `ENSMM`, avec la figure de droite indiquant des cercles à des multiples de 10 km. Il est habituellement considéré que la correction RTK est efficace jusqu'à une trentaine de kilomètres de la station de base utilisée comme référence.



Nous allons explorer trois utilisations des connaissances acquises pour le positionnement centimétrique de GNSS exploitant la bibliothèque libre RTKLib et sa déclinaison pour récepteurs faible coût Demo5 de `https://github.com/rinex20/RTKLIB-demo5` : la configuration et l'utilisation de RTKLib en console, s'affranchissant ainsi des déboires des interfaces graphiques pour rendre l'application compatible avec systèmes embarqués à faible empreinte mémoire et ressources de calcul, l'utilisation des signaux communiqués par Centipède depuis la station de référence la plus proche du site d'observation et une comparaison avec les résultats issus du post-traitement en exploitant les signaux de référence communiqués par l'Institut Géographique National (IGN), et finalement la cartographie en temps réel des signaux GNSS centimétriques dans l'outil de gestion des informations spatialisées QGis.

Nous précisons dès le début de cet exposé que l'objectif n'est pas une résolution centimétrique d'un objet en mouvement mais une géolocalisation avec une telle résolution après observation des constellations de satellites pendant plusieurs minutes voir dizaines de minutes. La solution convergera petit à petit en passante de Single (mesure simple récepteur) à Float (en cours de recherche de la solution) à Fix (solution obtenue). Le passage de Float à Fix peut être longue, voir ne jamais arriver tel que nous l'avons constaté sous certains couverts forestiers.

Nous avions dans la description précédente [1] expliqué comment configurer deux U-Blox Zed-9FP montés sur le circuit imprimé commercialisé par MikroE sous la référence 4456 au moyen du logiciel propriétaire U-Center, mais exécutable au moyen de l'émulateur Wine sous GNU/Linux, pour ne transmettre que les messages binaires au format UBX et ne pas émettre les message NMEA. Ainsi, avec les trames RAWX communiquées en 115200 bauds 8N1 pour toutes les constellations supportées (GPS, Galileo, Beidou et éventuellement GLONASS même si ce dernier est plutôt à proscrire compte tenu de sa communication multiplexée en fréquence très différente des autres modes de communication différentiant chaque satellite sur le code), nous avions proposé d'implémenter une station fixe (*basestation*) communiquant avec une station mobile (*rover*) dont la position **relative** est identifiée au centimètre près. Nous allons poursuivre cette étude en nous appuyant cette fois sur des réseaux de *basestations* de référence aux positions connues avec une exactitute sub-centimétrique (un point que nous avions omis d'aborder auparavant) en vue de localiser un *rover* avec une exactitude centimétrique en s'appuyant sur ce réseau de référence dont les informations sont transmises en temps réel et accessibles notamment par réseau de téléphonie mobile ou WiFi (Centipède) ou en post-traitement (IGN).

1 RTKLib en mode console

Moins de trois mois après avoir publié la méthode d'acquisition et de traitement de signaux centimétriques par récepteurs UBlox de la série 9 (Zed-F9P), les outils sont déjà cassés ! L'interface graphique de `rtknavi_qt` ne compile plus, et nous n'allons évidemment pas utiliser les binaires pré-compilés pour MS-Windows : afin de s'affranchir des affres de bibliothèques dont l'interface de programmation applicative (API) ne cesse de changer et se libérer d'interfaces graphiques illisibles sur un écran illuminé par le soleil, nous allons explorer les interfaces en ligne de commande de RTKLib, plus stables, compatibles avec les applications embarquées ou avec une automatisation des acquisitions et des traitements.

Les outils en ligne de commande se compilent sans surprise (testé sous Debian GNU/Linux) en exécutant `make` depuis le répertoire `app/consapp` de `RTKLIB-demo5`. Noter que la *cross-compilation* vers une cible embarquée d'architecture autre que l'hôte telle que le processeur ARM 64 bits de la Raspberry Pi4 dont nous supposons l'environnement de développement produit par Buildroot [2] s'obtient simplement en commentant dans le `makefile` la ligne `export CC = gcc` puis en lançant la compilation par `CC=aarch64-linux-gcc make` en ayant pris soin d'ajouter à son `PATH` le répertoire `output/host/usr/bin` du Buildroot configuré pour la cible (e.g. `make raspberrypi4_64_defconfig && make` depuis Buildroot). Les binaires se regroupent dans un unique répertoire, par exemple `/tmp/bin` prêt à expédier sur la cible, en commentant dans `makefile` la ligne `BINDIR = /usr/local/bin` puis en exécutant `mkdir -p /tmp/bin && BINDIR=/tmp/bin make install` afin de se contenter de copier le contenu de `/tmp/bin` vers la carte SD de la Raspberry Pi4. Lors de la connexion du récepteur U-Blox Zed-9FP, nous penserons à charger le pilote du port série virtuel sur bus USB `modprobe cdc_acm` puisqu'il n'est pas chargé automatiquement.

Nous apprenons dans la documentation de RTKLib [3] que la commande `rtkrcv -s -o RTKlib_dualZedF9P.conf` fait appel au logiciel approprié pour traiter un flux de mesures portant d'une part les informations de la station de référence, et d'autre part les observations du rover, selon un fichier de configuration un peu volumineux et disponible à `https://github.com/jmfriedt/RIOT_NyAlesund` pour être lancé dès son chargement (option `-s`) mais dont les principales informations sont

```
pos1-posmode   =kinematic
pos1-frequency =l1+l2
...
inpstr1-type       =serial
inpstr2-type       =serial
inpstr3-type       =off
inpstr1-path       =ttyACM0:115200:8:n:1:off
inpstr2-path       =ttyUSB3:115200:8:n:1:off
inpstr1-format     =ubx
inpstr2-format     =ubx
...
outstr1-type       =file
outstr2-type       =file
outstr1-path       =/tmp/1
outstr2-path       =/tmp/2
outstr1-format     =enu
outstr2-format     =llh
logstr1-type       =file
logstr2-type       =file
logstr3-type       =off
logstr1-path       =/tmp/rover.log
logstr2-path       =/tmp/base.log
...
```

donc deux flux venant sur les ports série `/dev/ttyACM0` (UBlox servant de rover) et `/dev/ttyUSB3` (sortie du flux de la station de base communiqué par LoRa), au débit de 115200 bauds, et au format UBlox binaire (`ubx`). Les sorties sont d'une part des données issues des traitements en divers formats (latitude/longitude/hauteur ou XYZ dans un repère d'origine le centre de la Terre – on vérifie que la racine de la somme des carrés est bien de l'ordre du rayon terrestre [4] de $40000/(2\pi) = 6366$ km), ainsi que le stockage des mesures brutes acquises par le rover et transmises par la station de base. Le résultat de ces traitements lorsque rover et station de base sont en vue du ciel (Fig 1) est de la forme :

```
%  GPST             x-ecef(m)      y-ecef(m)      z-ecef(m)   Q  ns   sdx(m)   sdy(m)   sdz(m)
2023/02/11 12:58:28.000   4313722.1776    452854.1904   4661052.7555   2  11   2.6268   1.1727   1.9213
...
2023/02/11 13:04:13.000   4313719.6932    452855.1056   4661050.6927   2  16   0.0467   0.0312   0.0141
2023/02/11 13:04:14.000   4313719.6938    452855.1075   4661050.6920   2  16   0.0465   0.0311   0.0140
2023/02/11 13:04:15.000   4313719.1227    452854.8761   4661050.5032   1  16   0.0065   0.0029   0.0057
2023/02/11 13:04:16.000   4313719.1237    452854.8776   4661050.5013   1  16   0.0065   0.0029   0.0057
2023/02/11 13:04:17.000   4313719.1255    452854.8769   4661050.5040   1  16   0.0065   0.0029   0.0056
2023/02/11 13:04:18.000   4313719.1206    452854.8745   4661050.5025   1  16   0.0065   0.0029   0.0057
...
```

avec dans l'ordre le temps GPS, la position dans un repère centré sur le centre de la Terre ECEF, une qualité de mesure `Q`, un nombre de satellites visible `ns` et un écart type sur la mesure (donc une résolution) `sdx`, `sdy` et `sdz` pour les trois directions de l'espace. Nous avons tronqué cet affichage des autres informations redondantes par soucis de compacité.

La documentation de RTKLib à la page 104 de `https://www.rtklib.com/prog/manual_2.4.2.pdf` nous aide à interpréter le statut de l'analyse en fournissant la nature de la solution dans la colonne `Q` :

```
The flag which indicates the solution quality.
1 : Fixed, solution by carrier-based relative positioning and the integer ambiguity
    is properly resolved.
2 : Float, solution by carrier-based relative positioning but the integer ambiguity
    is not resolved.
3 : Reserved
4 : DGPS, solution by code-based DGPS solutions or single point positioning with
    SBAS corrections
5 : Single, solution by single point positioning
```

Ainsi, une solution de type 1 est garante de la position avec une résolution sub-centimétrique et son obtention sur le terrain permet de s'assurer la qualité de la mesure en imposant d'observer suffisamment longtemps pour que la solution converge, parfois un peu pénible sous la pluie ou dans la neige.

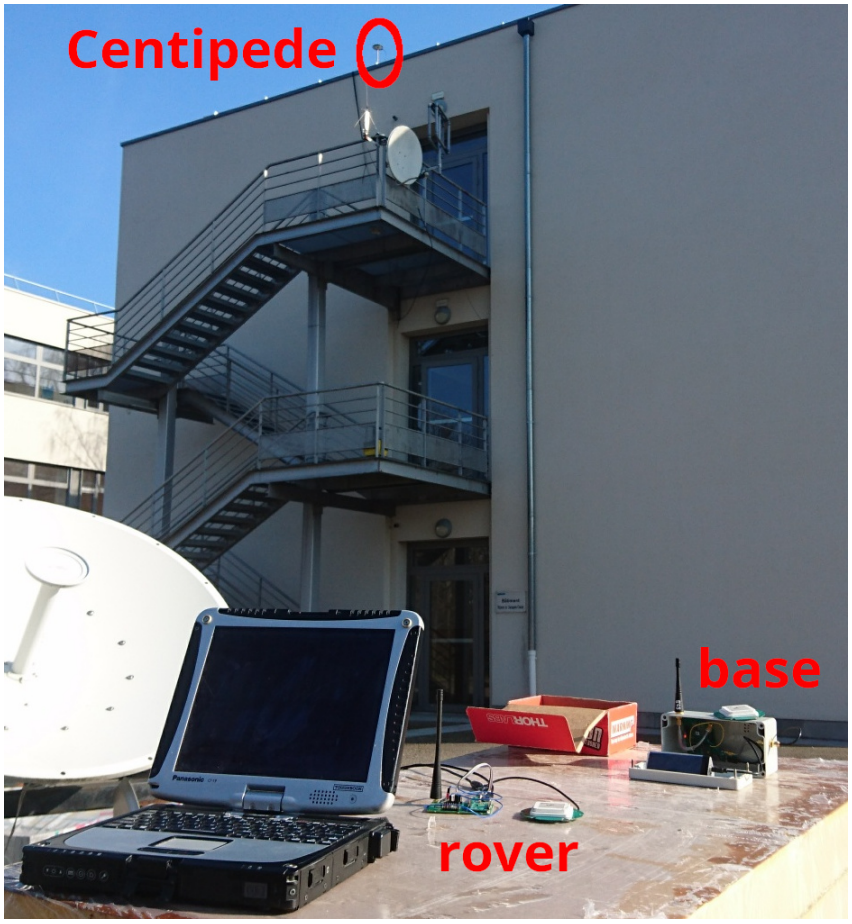


FIGURE 1 – Contexte expérimental : un rover (mobile) est connecté à un ordinateur exécutant les outils de RTKLib pour s'affranchir des sources corrélées de biais lors du positionnement pas satellite, en recevant un signal de référence soit d'une station de base communiquant par LoRa, soit d'un récepteur Centipède dont les informations sont diffusés par protocole compatible Internet IP.

2 Exploitation des signaux de référence de Centipède

La documentation de Centipède explique fort bien comment utiliser les informations diffusées et fournit à `https://docs.centipede.fr/docs/Rover_rtklib_pc/RTKlib_windows.html` un fichier d'exemple de configuration. Nous nous inspirons de ce document pour d'une part fournir un flux venant du port série connecté au récepteur Zed-F9P – le même qui a servi de rover auparavant – mais cette fois la station de base n'est pas un flux du port série mais un flux IP venant de la connexion WiFi de l'ordinateur effectuant les acquisitions et connecté au `caster` de Centipède diffusant ses informations sur le port 2101. Nous prenons soin de mentionner la station de référence qui nous intéresse, ici `ENSMM` puisque nous sommes à proximité de cette localisation : ce fichier de configuration `RTKlib_centipede.conf` contient donc

```
...
inpstr1-type       =serial
inpstr2-type       =ntripcli
inpstr3-type       =off
inpstr1-path       =ttyACM0:115200:8:n:1:off
inpstr2-path       =:@caster.centipede.fr:2101/ENSMM
inpstr1-format     =ubx
inpstr2-format     =rtcm3
...
outstr1-type       =file
```

```
outstr1-path       =./sortie.txt
outstr1-format     =xyz
...
```

pour être utilisé par `rtkrcv` au moyen de `rtkrcv -s -o RTKlib_centipede.conf`

Le résultat du traitement en temps réel des observations est de la forme

```
%  GPST             x-ecef(m)      y-ecef(m)      z-ecef(m)   Q  ns   sdx(m)   sdy(m)   sdz(m)
...
2248 565873.000   4313714.4614    452855.6378   4661048.9718   1   8   0.0074   0.0041   0.0072
2248 565874.000   4313714.4521    452855.6252   4661048.9851   1   8   0.0074   0.0041   0.0072
2248 565875.000   4313714.4583    452855.6248   4661048.9842   1   8   0.0074   0.0041   0.0072
2248 565876.000   4313714.4483    452855.6108   4661049.0041   1   8   0.0074   0.0041   0.0072
```

avec une qualité `Q` de 1 qui indique une solution fixe et un écart type dans les trois directions inférieur au centimètre.

Nous observons un écart de 5 m entre la station de référence que nous avons fournie initialement et la solution fournie par Centipède. Dans la première solution, la position de la station de référence n'a pas été imposée et a été mesurée par une solution simple (non différentielle) : cette position fluctue, comme toute mesure GNSS sans référence, de ± 7 m et un écart de 5 m n'est pas surprenant. Cette erreur pourrait être corrigée en plaçant notre station de base sur un point de position connue et en s'y référant lors de chaque campagne de mesures. Au contraire, la mise en place de la station de référence Centipède impose une période d'acquisition de sa position et son positionnement relativement au réseau de référence de l'IGN, garantissant la cohérence des positions. C'est justement cette position du réseau RGP (Réseau GNSS Permanent) de l'IGN que nous allons considérer ci-dessous pour du post-traitement.

Notons en cas de difficulté à se connecter une commande très pratique de `rtkrcv` qu'est `stream` qui annonce la nature des flux reçus par RTKLib :

```
rtkrcv> stream
Stream       Type     Fmt   S    In-byte  In-bps   Out-byte Out-bps Path                     Message
input rover  serial   ubx   C     607740   10804          0       0 ttyACM0:115200:8:n:1:off /dev/ttyACM0
input base   ntrips   rtcm3 C     657002   10327          0       0 :@caster.centipede.fr:21 caster.centipede.fr/ENSMM
...
```

prouve que le port série du récepteur mobile (`ttyACM0`) a été détecté et le flux en provenant de Centipède (`caster.centipede.fr`) est reçu. En cas d'échec de connexion internet, nous aurions un message du type

```
rtkrcv> stream
Stream       Type     Fmt   S    In-byte  In-bps   Out-byte Out-bps Path                     Message
input rover  serial   ubx   C      17592   10402          0       0 ttyACM0:115200:8:n:1:off /dev/ttyACM0
input base   ntrips   rtcm3 C          0       0          0       0 :@caster.centipede.fr:21 address error (caster.centipede
...
```

Par ailleurs, la commande `satellite` de `rtkrcv` permet de voir les constellations de satellites vues leur statut

```
rtkrcv> satellite
SAT C1    Az    El L1 L2  Fix1  Fix2  P1Res  P2Res   L1Res   L2Res  Sl1  Sl2  Lock1  Lock2 Rj1 Rj2
G07  - 292.4 10.5  -  -     -     -  0.000-24.794  0.0000  0.0000   27   15      0      0   0   0
G08 OK 301.2 52.1 OK OK  HOLD  HOLD -0.342 -0.587 -0.0063 -0.0013    1    1    361    361   0   0
G10 OK 117.9 60.1 OK OK  HOLD  HOLD  0.046 -0.669 -0.0022 -0.0030    1    0    361    361   0   0
G15  -  31.7  5.9  -  -     -     -  0.000  0.000  0.0000  0.0000    0    0      0      0   0   0
G16 OK 193.3 48.1 OK  -  HOLD     -  0.952  0.000 -0.0016  0.0000    0    0    361      0   0   0
G18 OK  63.4 17.5 OK OK  HOLD  HOLD -1.008 -1.523  0.0012 -0.0064    0    0    361    361   0   1
G21 OK 255.0 24.7 OK  -  HOLD     - -1.122  0.000 -0.0118  0.0000    1    0    361      0   0   0
G23 OK  61.4 44.1 OK OK  HOLD  HOLD -0.481 -0.934 -0.0008  0.0040    1    0    361    361   0   0
G27 OK  22.8 85.5 OK OK  HOLD  HOLD  0.000  0.000  0.0000  0.0000    0    1    361    361   0   0
E08 OK  40.6 11.8 OK OK FLOAT FLOAT  1.210 -0.735  0.0045 -0.0063    0    0      0      0   0   0
E12 OK 252.6 14.3  -  -     -     - -9.481 -2.435  0.0000  0.0000    3    6      0      0   0   0
E24 OK 319.3 19.5  -  -     -     --26.056  1.688  0.0000  0.0000   12    7      0     64   0   0
E26 OK  55.5 57.3 OK OK  HOLD  HOLD  0.135  0.175  0.0038 -0.0045    0    0    360    361   0   0
E31 OK 271.2 53.0 OK OK  HOLD  HOLD  2.000  0.696  0.0064  0.0127    0    0    360    361   0   0
E33 OK 270.2 66.1 OK OK  HOLD  HOLD  0.000  0.000  0.0000  0.0000    0    0    360    361   0   0
```

et s'assurer ainsi par exemple que l'antenne est bien connectée et alimentée.

3 Exploitation des signaux de référence de l'IGN

Le site de l'IGN qui propose l'accès à leurs observations de référence de signaux de navigation par satellites – `https://rgp.ign.fr/DONNEES/diffusion/` – se configure en sélectionnant la station de référence aussi proche que possible du site d'observation, pour nous station BSCN → flèche+ pour ajouter la station, activer `GPS C2` et `Galileo`, sélectionner la plage horaire autour de l'observation, 1 s de pas d'échantillonnage, et le format Rinex2.11 (Fig. 2). Une fois ces configurations achevées et validées par `Valider les filtres`, `Ajouter au panier` et `Télécharger`. On appréciera la qualité de ces données en sachant qu'une ligne téléphonique dédiée est installée à l'Observatoire de Besançon pour collecter et communiquer ces mesures.

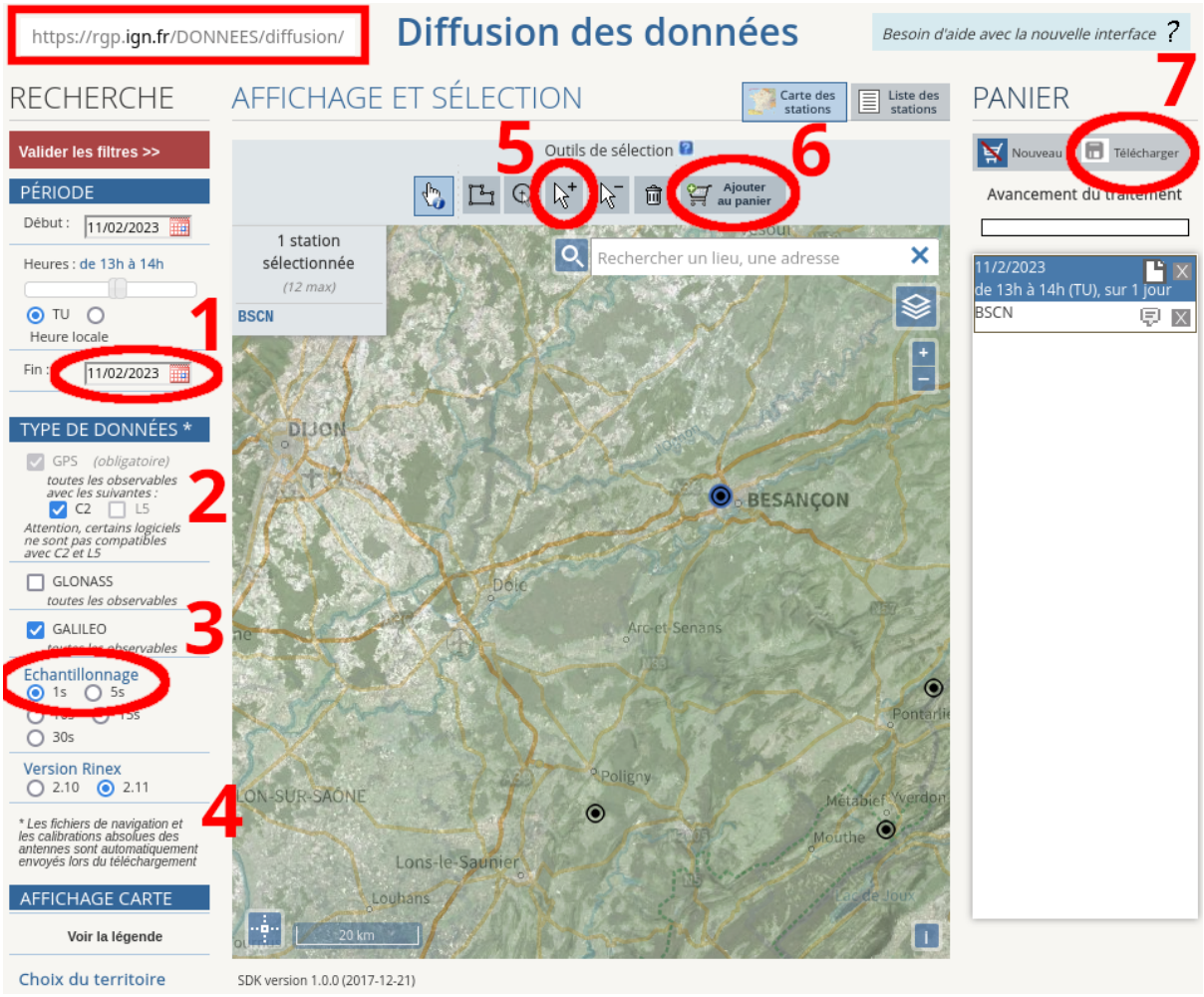


FIGURE 2 – Interface de téléchargement des signaux de référence mis à disposition par l'IGN. Comme pour Centipède, on choisira une station de référence aussi proche que possible du site d'observation, ici l'Observatoire de Besançon à quelques centaines de mètres du site de mesures. Les observations de l'IGN sont disponibles environ 1 h après leur acquisition et ne permettent donc pas une analyse en temps réel comme le propose Centipède.

Nous apprenons dans la page de manuel de `rnx2rtkp` `https://manpages.debian.org/testing/rtklib/rnx2rtkp.1.en.html` qui fait partie de RTKLib que le post-traitement des fichiers au format d'échange standard RINEX est pris en charge par cet outil qui prend comme argument le nom du fichiers d'observations du rover, le nom du fichiers d'observations de la station de référence que nous venons d'obtenir auprès de l'IGN (extension `??o` avec `??` l'année de la mesure) ainsi que les éphémérides associées au moment de l'observation contenues dans le fichier d'extension `??n`. Nous ajoutons comme options le type de solution recherchée – ici un rover statique (`-p` parmi 0 : single, 1 : dgps, 2 : kinematic, 3 : static, 4 : moving-base, 5 : fixed, 6 : ppp-kinematic, 7 : ppp-static) et un masque d'élévation en dessous duquel les observations sont rejetées (`-m`). Comme `rnx2rtkp` n'accepte que des fichiers au format

RINEX, nous devons convertir les enregistrements au format UBlox en RINEX au moyen de `convbin` fourni lui aussi par RTKLib :

```
convbin  -r ubx rover.log  -o rover.obs
rnx2rtkp -p 5 -m 15 -e rover.obs telechargement_RGP_198748/recherche_1/bscn042z.23o \
                                 telechargement_RGP_198748/recherche_1/bscn042z.23n
```

À l'issue de ce traitement, nous avons la satisfaction d'obtenir à nouveau des solutions sub-centimétriques, cette fois en excellent accord avec la solution calculée avec la position de la station de référence Centipède :

```
%  GPST             x-ecef(m)      y-ecef(m)      z-ecef(m)   Q  ns   sdx(m)   sdy(m)   sdz(m)
2248 565715.000   4313714.2435    452855.2018   4661049.2597   2   8   2.1218   1.1354   2.1171
...
2248 565739.000   4313714.1418    452855.4004   4661048.8065   2   8   0.4437   0.2456   0.3715
2248 565740.000   4313714.1851    452855.4162   4661048.8337   2   8   0.4338   0.2407   0.3596
2248 565741.000   4313714.2165    452855.3601   4661048.8608   1   8   0.0075   0.0042   0.0072
2248 565742.000   4313714.2037    452855.3567   4661048.8472   1   8   0.0075   0.0042   0.0072
2248 565743.000   4313714.2199    452855.3657   4661048.8398   1   8   0.0075   0.0042   0.0072
2248 565744.000   4313714.2384    452855.3703   4661048.8627   1   8   0.0075   0.0042   0.0072
2248 565745.000   4313714.2335    452855.3763   4661048.8459   1   8   0.0075   0.0042   0.0072
...
```

Certes la solution est obtenue à posteriori, mais cette approche peut s'avérer fort utile sur des sites où une liaison téléphonique n'est pas disponible pour récupérer en temps réel les informations de référence de Centipède, au détriment de ne pas savoir avant post-traitement si la solution a convergé. On prendra donc soin d'attendre quelques dizaines de minutes sur chaque site de mesure pour enregistrer suffisamment de mesures pour maximiser les chances de convergence de la solution `fixed`.

4 Traitement sur téléphone mobile

RTKLib existe pour Android sous le nom de RTKlibDroid disponible à `https://github.com/jancelin/RTKlibDroid` ou RTKGPS+ à `https://github.com/jancelin/RtkGps` avec une archive APK disponible dans les *releases* du dépôt afin qu'il ne soit pas nécessaire de se prostituer auprès de Google pour installer l'application. C'est cette seconde option que nous avons testée en configurant une source de rover venant du Zed-F9P connecté au téléphone mobile par câble USB-OTG – donc avec la broche ID du connecteur USB micro-B connectée à la masse – et la seconde source de station de base venant du caster Centipède, toujours en référant comme point de montage la station de référence la plus proche (dans notre cas `ENSMM`). Après queleques minutes en espace libre avec une bonne vue du ciel, la solution `Fix` est obtenue et l'écart type sur la position chute sous le centimètre (Fig. 3). Dans le cas particulier du téléphone mobile Sony Xperia Z5compact utilisé au cours de ces essais, la détection du périphérique connecté au port USB nécessite une recherche explicite dans `Settings` → `Device connection` → `USB Connectivity` → `Detect USB Device`. Une fois le port série virtuel identifié, RTKLib+ valide la lecture des données en faisant clignoter un des deux indicateurs verts en haut à droite de l'interface graphique signifiant que les trames sont lues et traitées.

Obtenir une solution sub-centimétrique sur téléphone mobile est bien, mais les utilisateurs argumenteront qu'ils veulent cette solution dans leur outil favori de gestion d'information spatialisées, donc QField sous Android – la version embarquée de QGis. Il faut donc transmettre les informations issues de RTKLib vers QField en passant outre du récepteur GPS matériel équipant le téléphone mobile. Pour ce faire, nous exploiterons l'option d'injection de position en mode développeur sous Android : RTKLib produit une information centimétrique au tranvers de ce protocole qui supplante la position déduite par le récepteur matériel et indique à QField sa position depuis le périphérique externe. Après avoir généré un projet QGis trivial qui ne contient qu'une couche OpenStreetMaps et l'avoir importé dans QField, nous pourrons valider avec une résolution centimétrique sur plateforme de calcul mobile les informations spatialisées qui nous intéressent (Fig. 4).

L'activation du mode d'injection de trames GNSS dans Android s'obtient en activant le mode développeur et en autorisant RTKLib+ à injecter les trames dans l'onglet `Select mock location app` (Fig. 4, gauche).

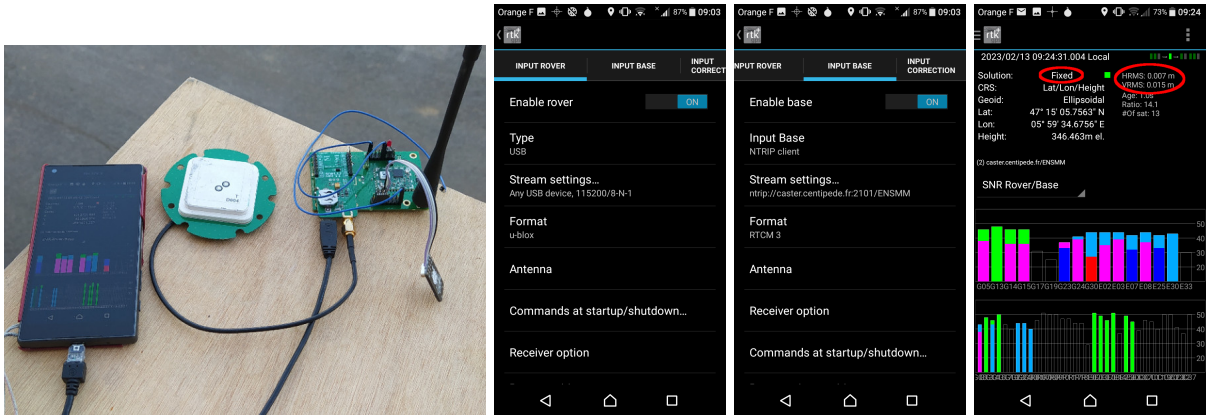


FIGURE 3 – De gauche à droite, montage experimental vec le Zed-F9P connecté par câble USB-OTG au téléphone mobile ; la configuration de la première source comme flux série asynchrone compatible RS232 au débit de 115200 bauds en accord avec la configuration du récepteur GPS ; la configuration de la seconde source connectée par lien de téléphonie mobile au caster Centipède ; et le résultat du positionnement sub-centimétrique.

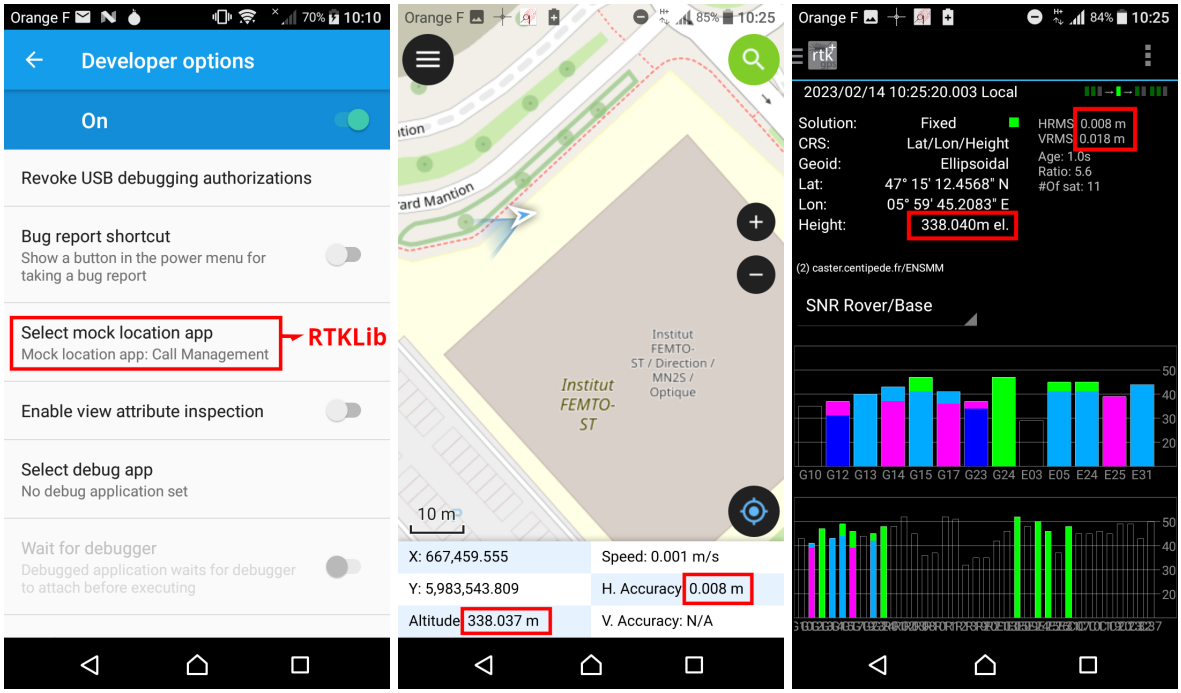


FIGURE 4 – De gauche à droite : activation du mode d'injection de position en mode développeur sous Android autorisant RTKLib avec sa solution centimétrique à supplanter la solution du récepteur du téléphone ; la position du récepteur vue par QField avec une carte OpenStreeMaps indiquant un écart type sub-centimétrique sur la position (encadré rouge) compatible avec la solution de RTKLib (droite).

5 Utilisation de téléphones mobiles équipés de récepteurs multibandes

Certains téléphones portables sont équipés de récepteurs GNSS multiconstellations et multibandes tel que décrit à [5] et en particulier lorsqu'ils sont basés sur un chipset Qualcomm Snapdragon. Nous avons ainsi acquis un Xiaomi Mi10Lite muni de son Snapdragon 765G et tenté d'estimer la résolution de positionnement en post-traitement s'appuyant sur les données fournies par le récepteur de l'IGN à Nanterre.

Les mesures brutes du téléphone portable Xiaomi Mi10Lite sont enregistrées au moyen de l'application

`Geo++ RINEX Logger`. La nature des constellations observées est précisée dans l'entête du fichier RINEX, avec G pour GPS américain, R pour GLONASS russe, E pour Galileo européen, C pour Beidou chinois et J pour QZSS japonais tandis que le type des observations suit avec C pour les pseudoranges (temps de vol du code), L la phase de la porteuse radiofréquence, D le décalage Doppler de la porteuse du au mouvement du satellite et S la puissance du signal. Nous constatons donc que le Xiaomi Mi10Lite observe les quatre grandes constellations en orbite moyenne (G, R, E et C) sur les bandes 1 et 5 de GPS et Galileo en code et en phase :

```
G    8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q
R    4 C1C L1C D1C S1C
E   12 C1B L1B D1B S1B C1C L1C D1C S1C C5Q L5Q D5Q S5Q
C    4 C2I L2I D2I S2I
J    8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q
```

Le résultat du traitement est quelque peu décevant avec

```
%  GPST             x-ecef(m)      y-ecef(m)      z-ecef(m)    Q  ns   sdx(m)   sdy(m)   sdz(m)
...
2138 311127.448   4203940.0881   163707.8201   4777882.5088   4   0   1.9344   1.1780   2.5586
2138 311128.448   4203944.2380   163709.4200   4777886.0141   4   0   1.8424   1.1506   2.5183
2138 311129.448   4203948.3544   163712.8539   4777891.5647   4   0   1.7981   1.1389   2.4892
2138 311130.448   4203947.9964   163714.5687   4777890.2285   4   0   1.7642   1.0951   2.2640
2138 311131.448   4203946.8551   163715.4582   4777888.3513   4   0   1.7400   1.0817   2.1458
```

un écart types dans les trois directions de l'ordre du mètre. Nous avions bien été prévenu de la qualité médiocre des antennes de réception GNSS équipant les téléphones mobiles et placer le téléphone à quelques dizaines de centimètres du sol sur un banc est loin d'une condition idéale pour éviter les rebonds et interférences des chemins multiples, mais nous sommes loin du centimètre recherché (Fig 5).

FIGURE 5 – Enregistrement de données brutes GNSS par un téléphone mobile Xiaomi Mi10Lite sur l'île Seguin à l'ouest de Paris.

6 Injection des signaux GNSS dans QGis

Maintenant que nous sommes capables de traiter en temps réel des mesures acquises par le récepteur mobile en se référant à une station de base de coordonnées connues, il peut sembler utile de placer sur une carte le fruit de ces traitements. Bien que Google Maps propose un mode de positionnement en temps réel depuis un récepteur GNSS (`Tools` → GPS → Realtime), il s'agit d'un jouet propriétaire qui ne possède pas toutes les couches d'analyses que nous aurions été susceptibles d'obtenir lors de traitements antérieurs à la séance d'observation sur le terrain, et nous allons donc nous efforcer d'interfacer QuantumGis (QGis) avec le flux issu du traitement de RTKLib. En effet, au moins dans sa version 3.28 utilisée pour cette démonstration, un flux de données GNSS peut être acquis depuis `gpsd` par QGis en activant `View`→ Panels → GPS Information. La nouvelle fenêtre qui apparaît sous la liste des couches propose de se connecter de diverses façons à une source de données de localisation et en particulier `gpsd`. Si le démon de gestion des données de positionnement est actif et communique par défaut sur son port 2947, alors QGis indique une cible au niveau de la position du récepteur. Aucune configuration n'est modifiée dans QGis puisque `gpsd` communique par défaut au travers de ce port de `localhost`.

Cependant, nous ne sommes pas intéressés par la connexion d'un simple GPS à QGis, mais à exploiter le résultat du traitement en temps réel par RTKLib. Ainsi, il faudra remplacer l'interface physique (Bluetooth, USB ou RS232) par un flux de données, et naturellement un *pipe nommé* ou FIFO vient à l'esprit. Cependant, avant de se lancer dans le flux continu de données, commençons par essayer de communiquer un fichier au format NMEA issu du traitement par `rnx2rtkp` à `gpsd` en vue de le transmettre à QGis.

Les auteurs de `gpsd` ont prévu la capacité à tester le démon avec un jeu de données factice et proposent pour cela le script Python `gpsfake`. Ce programme accepte en argument un fichier contenant des données de positionnement au format NMEA et injecte ces informations à une instance spécifique de `gpsd`. Afin d'exécuter `gpsfake`, il faut s'assurer que le port 2947 de *localhost* est libre et qu'aucune instance de `gpsd` n'y est associée (`sudo lsof -i:2947 -n -P` ne doit rien répondre, sinon `systemctl stop gpsd` et `systemctl stop gpsd.socket`) avant de lancer `gpsfake -S fichier.nmea`. Nous avons pris un temps significatif à découvrir que les pseudo-terminaux créés par `gpsfake` sont protégés par un superviseur de sécurité `apparmor` dont seule la politesse nous interdit de s'en débarrasser tant ses restrictions sont handicapantes. En effet, les messages de `dmesg` du type

```
[V.W] audit: type=1400 audit(X:Y): apparmor="DENIED" operation="open" profile="/usr/sbin/gpsd"
name="/dev/pts/10" pid=Z comm="gpsd" requested_mask="r" denied_mask="r" fsuid=1000 ouid=1000
```

indiquent que `apparmor` a non seulement interdit l'accès au pseudo-terminal `/dev/pts/10`, mais en plus il interdit à l'administrateur d'en changer les permissions. Pour une fois nous allons suivre l'approche rationnelle de modifier correctement les fichiers de configurations de `apparmor` au lieu de simplement contourner le problème : ces fichiers dans `/etc/apparmor.d/usr.sbin.gpsd` contiennent des informations de la forme

```
# common serial paths to GPS devices
/dev/tty{,S,USB,AMA,ACM}[0-9]*    rw,
/sys/dev/char     r,
/sys/dev/char/**  r,
```

qui autorisent à `gpsd` d'accéder aux périphériques de communication de récepteurs matériels, que nous complétons de

```
/dev/pts/*    rw,
```

pour autoriser l'accès aux pseudo-terminaux. À l'issue de cette correction prise en compte par `apparmor_parser -r /etc/apparmor.d/usr.sbin.gpsd`, l'erreur disparaît au lancement de `gpsfake` avec l'argument `-S` pour ralentir le flux de données suivi du nom de l'archive au format NMEA, et en connectant QGis nous avons la satisfaction de voir le voyant vert s'allumer et une cible se placer sur la position issue du traitement de RTKLib (Fig. 6).

Nous pourrons vérifier que `gpsd` comprend bien les informations fournies au moyen des outils en ligne de commande que sont `cgps` ou `gpsmon` pour valider la position proposée dans le fichier NMEA.

Finalement nous voudrions dynamiquement mettre à jour QGis avec les mesures issues du flux continu de traitement de RTKLib et non d'un fichier issu d'un traitement statique. La FIFO vient évidemment

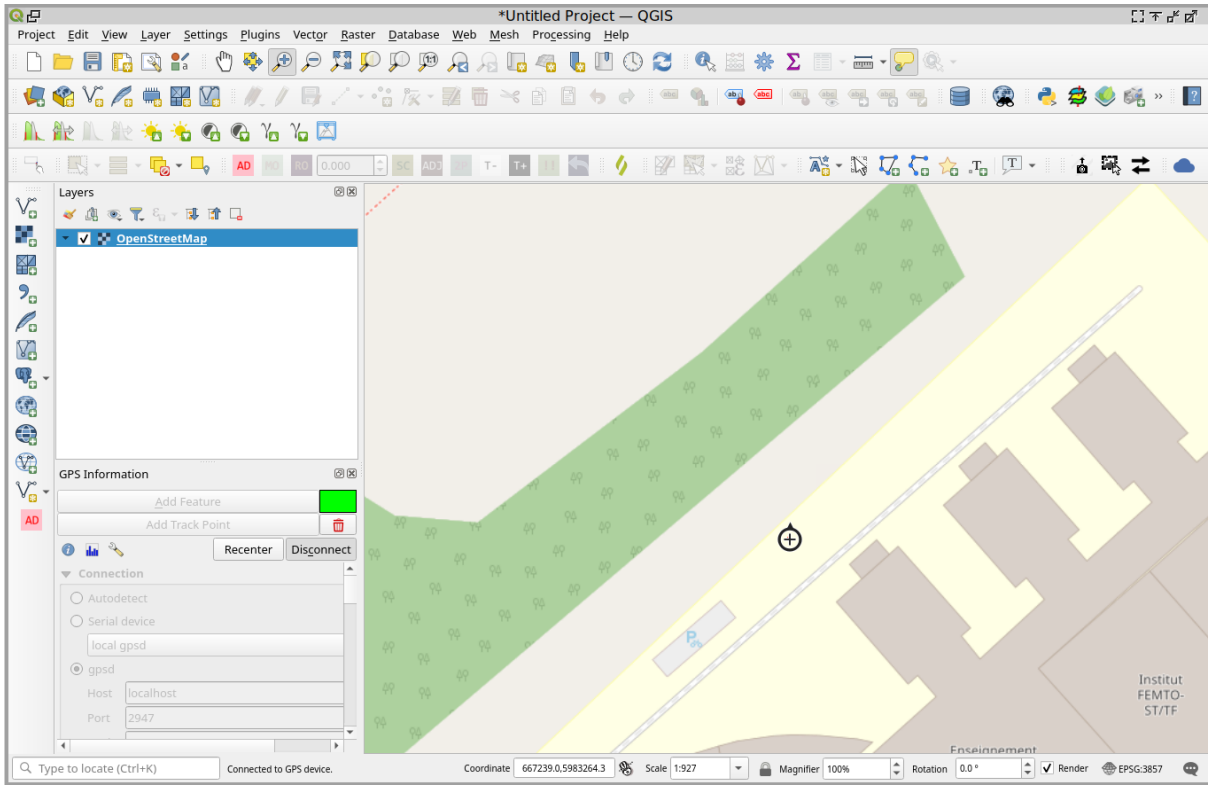


FIGURE 6 – Transfert des mesures traitées par RTKLib à QGis pour affichage sur un fond de carte – ici OpenStreeMaps – et pouvant contenir les diverses couches issues des traitements antérieurs sur le même site. La cible est placée avec une résolution et une exactitude sub-centimétrique à une position transmise par `gpsd`, ici alimenté par un fichier produit par RTKLib.

à l'esprit pour effectuer ce traitement : un *pipe* nommé est créé au moyen de `mkfifo /tmp/mafifo` mais une fois de plus `apparmor` va nous empêcher d'en modifier les permissions et nous prenons soin d'ajouter `/tmp/mafifo rw,` après les autorisations déjà discutées auparavant de `/etc/apparmor.d/usr.sbin.gpsd`. Si nous avons pris soin de produire un fichier au format NMEA tel qu'attendu par `gpsd`, par exemple au moyen de

```
rnx2rtkp -p 5 -m 15 -n rover.obs telechargement_RGP_198748/recherche_1/bscn042z.23o \
        telechargement_RGP_198748/recherche_1/bscn042z.23n > fichier.nmea
```

(noter que le `-e` de cette fonction fournie auparavant a été remplacée par `-n` pour remplacer le format de sortie de XYZ à NMEA), alors lancer `gpsd` par `/usr/sbin/gpsd -N -D 1 /tmp/mafifo` suivi de `while true; do cat fichier.nmea > /tmp/mafifo ;sleep 1;done` va continuellement communiquer par la FIFO le contenu du fichier NMEA. En connectant QGis à `gpsd`, nous verrons le curseur se positionner au bon endroit. S'étant convaincu de la validité de la procédure, nous appliquons le concept à la communication en temps réel en ajoutant dans le fichier de configuration de `rtkrcv` une nouvelle sortie au format NMEA pointant vers `/tmp/mafifo`

```
outstr2-type      =file
outstr2-path      =/tmp/mafifo
outstr2-format    =nmea
```

puis

1. lancer `gpsd` avec `/usr/sbin/gpsd -N -D 1 /tmp/mafifo`
2. lancer `rtkrcv -s -o RTKlib_centipede.conf`
3. connecter QGis

Le résultat est présenté en Fig. 7 avec un référencement sur la station Centipède `ENSMM`, avec une mesure de la largeur du nuage de points acquis en cinq minutes du centimètre, acquis en temps réel.

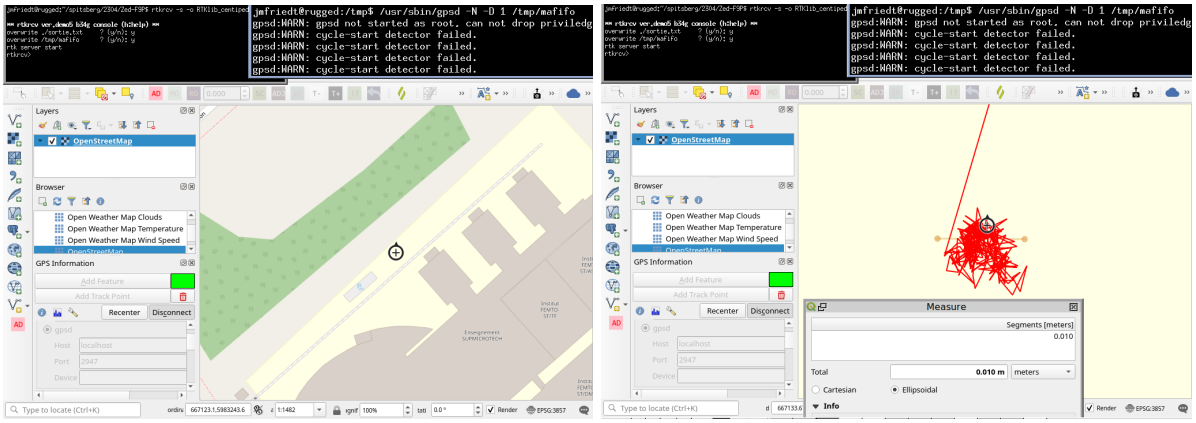


FIGURE 7 – Gauche : vue d'ensemble de QGis recevant en temps réel de nouvelles mesures produites par RTKLib (terminal en haut à gauche) et communiquées au travers de `gpsd` (terminal en haut à droite) au travers d'un *pipe* nommé tel qu'en atteste l'icône verte du menu `GPS Information`. Droite : zoom sur la position du curseur, indiquant que la position a varié de l'ordre du centimètre au cours des cinq minutes d'acquisitions.

On notera que comme tout bon *pipe* nommé les données ne commencent à circuler que lorsque les deux extrémités du tuyau sont connectées, donc `rtkrcv` demandera à écraser `/tmp/mafifo` lors de la première connexion de QGis qui échouera, et ce n'est que lors de la deuxième connexion que QGis recevra avec succès les données de position au format NMEA. Parfois les trames ne commencent pas sur l'information attendue et il faut relancer la tentative de connexion jusqu'à ce que QGis finisse par comprendre les messages qui lui sont transmis. Les positions sont simultanément sauvées dans un fichier indiquant la position dans un référentiel d'origine le centre de la Terre et indiquant un écart type de la position centimétrique de la forme

```
%  GPST                  x-ecef(m)      y-ecef(m)      z-ecef(m)    Q  ns   sdx(m)   sdy(m)   sdz(m)
2023/02/16 07:36:08.000   4313719.3079    452854.5832   4661050.3111   1  13   0.0077   0.0037   0.0077
```

On pourrait s'inquiéter de voir des coordonnées différer de 5 m sur X et un mètre sur Y et Z mais ce système de coordonnées est peu intuitif et l'utilisation du convertisseur de référentiels `https://proj.org/` mis en paquet par Debian/GNU Linux sous le nom `proj-bin` indique que les positions obtenues à partir de Centipède ou IGN ne diffèrent de cette nouvelle position séparée de plusieurs jours de 1,5 m, inacceptable si nous visons un positionnement reproductible centimétrique, avec une altitude qui a même fait un saut de 4 m. En effet la conversion depuis ECEF vers WGS-UTM31N indique que

```
$ cs2cs  +proj=geocent +datum=WGS84 +units=m +no_defs +to   +init=epsg:32631 -f "%.8f" fichier
X               Y               Z
726464.90031035 5237471.83799679 342.33079235  # Centipede 11/02/23
726464.68704738 5237471.89734084 342.05299559  # IGN       11/02/23
726463.47093372 5237469.19760583 346.49832384  # Centipede 16/02/23
```

il faut donc prendre soin de vérifier la reproductibilité des mesures à long terme.

Nous répétons cette mesure régulièrement sur plusieurs jours afin d'analyser l'impact de la géométrie de la constellation des satellites de navigation, en se rappelant qu'un satellite en orbite moyenne met 12 h pour parcourir sa circonférence et que pendant ce temps la Terre a fait un demi-tour. Ainsi, la constellation aura considérablement changé en quelques heures, permettant de valider l'insensibilité de la mesure à la position des satellites dans le ciel. En effet, nous constatons sur plusieurs jours que l'observation ne varie que de quelques centimètres, en accord avec nos attentes (Fig. 8), avec une fluctuation de la mesure un peu dégradée en Z comme on peut s'y attendre compte tenu de l'extension de la constellation en latitude et longitude plutôt qu'en altitude. Pourquoi le saut de plusieurs mètres sur l'analyse précédente ? Nous n'avions initialement pris aucune précaution contre les multichemins (*multipath*) dans lesquels le récepteur ne voit pas le signal provenant directement d'un satellite mais son rebond sur une surface réfléchissante environnante. Lors des premières mesures, le récepteur était placé sur un tas de mousses isolantes d'environ 1,20 m de hauteur, sans plan de masse métallique sous l'antenne (Fig. 1). Au cours des nouvelles mesures, nous avons pris soin de nous placer au niveau du sol (qui ne peut donc pas agir comme réflecteur de multichemin) et sur un plan de masse conséquent. Ces précautions semblent avoir résolu le problème.

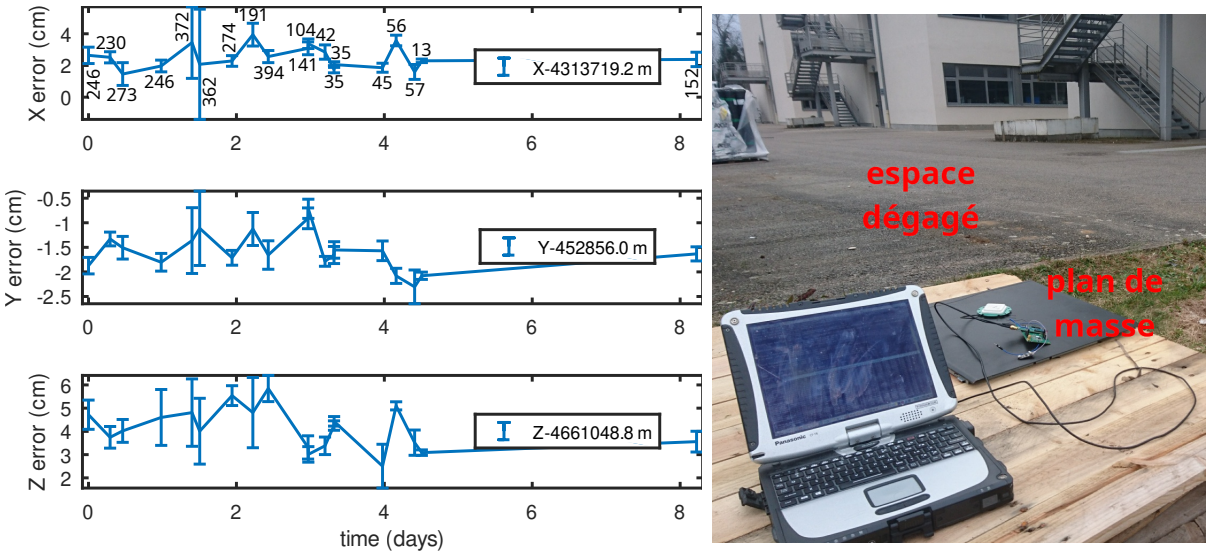


FIGURE 8 – Mesure sur plus d'une semaine de la position du récepteur placé sur une plaque métallique au niveau du sol (droite) avec une antenne replacée à chaque séance avec une précision centimétrique sur le support, chaque mesure étant initialisée par un démarrage à froid indépendamment de toutes les précédentes. Nous constatons (gauche) une variabilité de l'ordre du centimètre de la mesure sur plusieurs jours d'observations, en accord avec les objectifs visés. De haut en bas l'écart de position en X, Y et Z après en avoir soustrait la valeur moyenne (légende). Le nombre indiqué au dessus de chaque barre d'erreur dans le cadran X du haut indique le nombre de mesures (1 par seconde) exploitées lors du calcul statistique, séquences contigües de solutions "Fix" (qualité 1).

7 Conclusion

Alors que les vendeurs de babioles ne rêvent que de localiser les clients avec une résolution centimétrique pour leur transmettre la publicité de produits inutiles au bon endroit, l'avènement de récepteurs de localisation par constellations de navigation par satellites propose des perspectives d'applications fascinantes pour qui prend le temps d'apprendre à s'en servir. Nous avons exploré le positionnement avec une résolution sub-centimétrique en exploitant une station de référence supposée soumise aux mêmes sources de biais que le dispositif mobile, avec une exactitude limitée par la localisation de la station de base. Avec une station de base librement positionnée, l'exactitude reste métrique mais avec une résolution sub-centimétrique, alors que placer la station de base sur un emplacement connu ou exploiter une station de référence de position connue ramène l'exactitude à la résolution du centimètre. Afin de valider sur le terrain la qualité de la mesure et l'interpréter dans un contexte géographique, nous avons transmis le fruit de ces calculs à QGis sous GNU/Linux et QField sous Android qui par ailleurs affichent les fonds de cartes ou résultats des analyses antérieurs sur le site d'étude.

Nous avons mentionné notre capacité à cross-compiler `rtkrcv` sur Raspberry Pi4 : étrangement, nous ne sommes jamais arrivés à obtenir une position centimétrique, l'instruction `satellite` de `rtkrcv` indiquant tout au plus deux satellites en mode RTK alors que le même fichier de configuration avec l'antenne au même emplacement permet d'obtenir une solution sans problème sur ordinateur portable, et ce même en passant le processeur de la Raspberry Pi4 en mode `performance` pour le cadencer à 1500 MHz. Le problème doit être lié à la puissance de calcul pour atteindre la solution RTK puisqu'en l'absence de connexion à Centipède, la solution `Single` est atteinte en quelques minutes. Pourtant, [7] annonce faire fonctionner cette combinaison : nous y constatons que seule la porteuse L1 est traitée, et en effet nous convergeons vers une solution de type `Single` dans cette configuration, avec une exactitude métrique et non centimétrique.

Les fichiers de configuration de RTKLib utilisés au cours des ces expériences (*rover* Zed-F9P v.s *basestation* Zed-F9P ou Centipède ou IGN) sont disponibles 'a `https://github.com/jmfriedt/RIOT_NyAlesund`.

Remerciements

Au cours de l'Eclipse IoT Day organisé à Grenoble les 19 et 20 Janvier 2023 [6], les administrateurs de Centipède nous ont fait remarquer qu'il est possible d'abaisser la bande passante de communication en n'acquérant qu'une mesure toutes les 5 secondes (au lieu de chaque seconde actuellement) et que le taux de raffraîchissement des corrections résultant est suffisant pour la correction RTK, un point intéressant lorsque le débit de communication est limité.

Références

[1] J.-M Friedt, *Communication LoRa au moyen de RIOT-OS pour la mesure centim etrique par GPS différentiel avec RTKLib*, Hackable **45** (Nov-Dec. 2022)

[2] G. Goavec-Merou, J.-M Friedt, *"On ne compile jamais sur la cible embarquée" : Buildroot propose GNU Radio sur Raspberry Pi (et autres)*, Hackable **37** (2021) Porting GNU Radio to Buildroot : application to an embedded

[3] *RTKLib Manual : Demo5 version* à `https://github.com/rtklibexplorer/RTKLIB/blob/demo5/doc/manual_demo5.pdf` (2021)

[4] F. Trystram, *Le procès des étoiles*, Ed. Seghers (1979) ou J. Verne, *Aventures de trois Russes et de trois Anglais dans l'Afrique australe*, Ed. Hetzel et Cie (1872) replacés dans un contexte historique à `https://lesia.obspm.fr/perso/jacques-crovisier/JV/verne_3R3A.html`

[5] S. Barbeau, *Crowdsourcing GNSS features of Android devices* (2021) à `https://barbeau.medium.com/crowdsourcing-gnss-capabilities-of-android-devices-d4228645cf25`

[6] Programme, transparents et vidéos des présentations de *Eclipse IoT Day Grenoble 2023* à `https://wiki.eclipse.org/Eclipse_IoT_Day_Grenoble_2023#January_19.2C_2023`

[7] T. Everett, *Raspberry Pi based PPK and RTK solutions with RTKLIB* (11/2022) à `https://rtklibexplorer.wordpress.com/2022/11/10/raspberry-pi-based-ppk-and-rtk-solutions-with-rtklib/`