Filière électronique

Semestre 8

Prototypage d'un système de communications numériques complet à base de modules SdR

Membres du groupe :
Yassine AMRANI
Matthias VERHOEVEN

Encadrants :
Bertrand LE GAL
Guillaume FERRE

Table des matières

1 Introduction

Ce projet s'inscrit dans les travaux réalisés autour de la conception d'un système de communications numériques initié dans le projet « Eirballoon ». Des travaux antérieurs ont été réalisés dans le cadre d'un projet S9 (SE) (2020). Notre objectif principale lors de ce projet est le développement complet d'un démonstrateur permettant de réaliser des communications numériques à partir des modules SdR afin transmettre une image entre l'émetteur situé sur le ballon sonde et une station au sol. Dans ce projet, le démonstrateur intégrera des modules SdR, une Raspberry Pi 4 (processeurs (x86, Arm)) et un module carte SIM (gestion d'envoi des SMS). La finalité du projet est de permettre la transmission d'une image à l'aide des modules SdR et de pouvoir transmettre la position du ballon par SMS. Nous pouvons alors séparer notre système en deux parties importantes **émetteur** et **récepteur**. Notre système complet devra donc être capable d'effectuer différentes étapes. En premier lieu, une chaîne d'émission va permettre d'émettre une image sous forme d'un fichier "QPSK_Tx.raw" à l'aide de l'un des modules Sdr pour qu'ensuite on récupère un autre fichier de réception "QPSK_Rx.raw" contenant l'image émise. Enfin, une chaîne de réception permettra la reconstruction et l'affichage de l'image à l'aide des trames reçues.

2 Description du matériels utilisés

Pour concevoir ce projet, nous avons à notre disposition les matériels suivants :

— **Carte Raspberry Pi 4** : nano-ordinateur monocarte à processeur ARM.
— **Modules Sdr** : HackRF One SDR.
— **Module carte SIM** : Adafruit FONA.

2.1 Description de la Raspberry Pi 4

Le Raspberry Pi 4 est ce que l'on appelle un nano-ordinateur monocarte possédant ni souris, ni clavier, ni écran, ces éléments pouvant cependant être connectés si cela est souhaité. De la taille d'une carte de crédit, il est équipé du strict nécessaire pour formuler une simple architecture d'un ordinateur :

— Un microprocesseur ARM.
— De la mémoire RAM.
— Une carte vidéo.
— Une carte ethernet.
— Wi-Fi et Bluetooth.

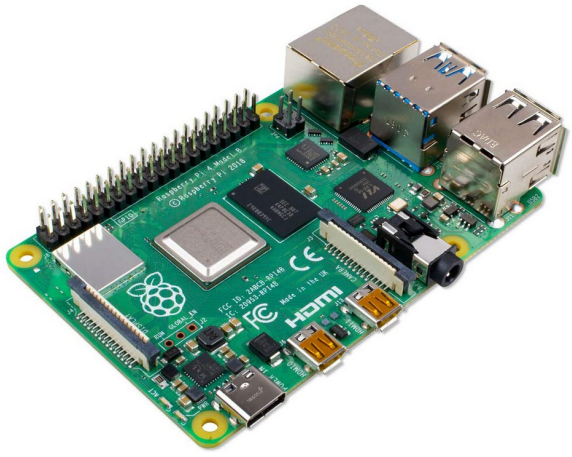

Figure 1 – Raspberry Pi 4

En effet, l'objectif principal de la construction de la Raspberry Pi 4 est de créer un outil très accessible pour permettre à tous les étudiants d'apprendre plus efficacement le développement et la programmation informatique. Tel est le cas dans notre projet car le code (C/C++) généré pour réaliser la transmission et la réception de l'image sera implanter sur la Raspberry pour tester le bon fonctionnement du système.

2.2 Modules SDR

Pour effectuer la transmission et la réception de nos signaux, on utilisera deux modules HackRF One. En effet, le HackRf One est un périphérique SDR (Software Defined Radio) capable de faire une transmission ou une réception de signaux radio couvrant une plage de fréquence entre 1 MHz à 6 GHz. Il est peut être utilisé comme périphérique USB ou programmé pour un fonctionnement autonome. HackRF One est un équipement de test pour les systèmes RF et c'est pour cela qu'on l'utilisera dans notre projet afin de tester le bon fonctionnement du système. Ainsi, notre HackRF One est équipé aussi avec une antenne "VERT2450". Parmi les caractéristiques importantes de ce composant, on peut citer :

- Fréquence de fonctionnement de 1 MHz à 6 GHz ainsi qu'un émetteur-récepteur semi-duplex.
- Environ 20 millions échantillons par seconde.
- Échantillons en quadrature 8 bits (8 bits I et 8 bits Q).
- Gain RX et TX et filtre de bande de base configurables par logiciel.
- Alimentation du port d'antenne contrôlée par logiciel (50 mA à 3,3 V).
- Connecteur d'antenne femelle SMA et une entrée et sortie d'horloge femelle SMA pour la synchronisation.

Figure 2 – HackRF One vu de l'intérieur et de l'extérieur

2.3 Adafruit FONA

Le module carte SIM Adafruit Fona étudié est petit de taille mais contient une quantité surprenante de technologie dans son petit cadre. Au cœur se trouve un module cellulaire GSM (nous utilisons le dernier SIM800) d'une taille moyenne. Ce module peut faire à peu près tout dont l'échange des messages et même effectuer des appels audio mais dans notre étude on va se focaliser juste sur l'envoi des SMS. D'ailleurs, on peut l'utiliser aussi pour envoyer et recevoir des données GPRS (TCP / IP, HTTP, etc). Ainsi, on peut communiquer avec ce module avec l'interface AT commande qui sera traité plus profondément plus tard dans le rapport. En fait, le module contient aussi plusieurs pins qui lui permet d'être connecter à notre Raspberry Pi 4 :

- La pin **Vio** se connecte à une batterie de 3,7 V.
- La pin **GND** qui représente la masse du circuit intégré et qui se connecte au pin GND de la Raspberry.
- La pin **RX** sera connecté au pin **TX** de la Raspberry et vis-versa aussi.

Évidemment, le module carte SIM seul ne fonctionne pas et il a besoin d'autres composants externes pour le mieux exploiter notamment une carte SIM et une batterie minimum 500 mAh, on peut même utiliser une antenne GSM et une passive GPS antenne.
Ainsi, après branchement complet du module SIM les témoins lumineux (LEDs) viennent pour confirmer le bon fonctionnement de celui-ci :

— **LED Bleu :** Est allumée lorsque le module est démarré et qu'il est exécute.
— **LED Rouge :** Est allumée pour vérifier l'état actuel sans envoyer de commande AT. En fait, lorsqu'elle est 64 ms allumé, 800 ms éteint - le module fonctionne mais ne s'est pas encore connecté au réseau cellulaire. Or, pour 64 ms allumé, 3 secondes éteint - le module a pris contact avec le cellulaire réseau et peut envoyer / recevoir des messages vocaux et SMS 64 ms allumés, 300 ms éteints - la connexion de données GPRS demandée est active.
— **LED Verte :** C'est à côté de la prise JST. Il Indique que la charge de la batterie est terminée et que la batterie est pleine.

Donc, en regardant les clignotements, on peut obtenir un retour visuel sur ce qui se passe.

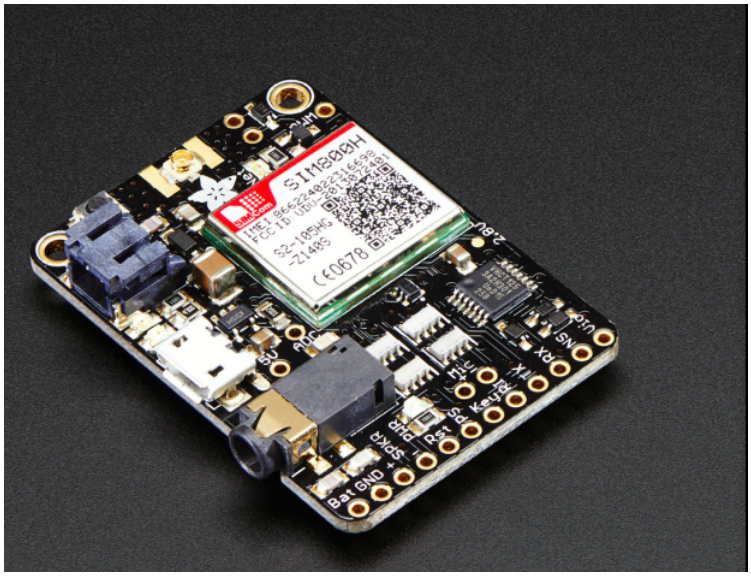

Figure 3 – Le module SIM ADAfruit Fona

3 Présentation globale du projet

3.1 Description du projet

On peut distinguer deux grandes parties au projet : d'un coté la transmission d'images et de l'autre l'envoi de la localisation par SMS.

3.1.1 Transmission d'images

Dans le cadre d'un projet comme "Eirballoon", à une certaine altitude, nous aimerions pouvoir admirer la vue en temps réel. Pour cela, l'objectif du projet est de mettre en place un système qui nous permette de transmettre des images à partir d'un objet distant de façons automatique et en temps réel. Ainsi, on pourrait schématiser cette transmission comme ceci :

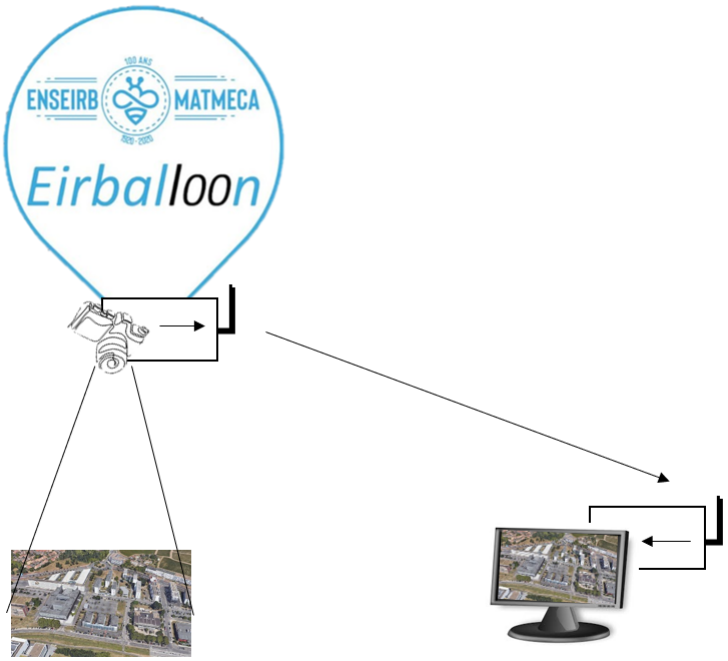

Figure 4 – Transmission d'une image

Pour arriver à ce résultat, nous avons à notre disposition :

- un appareil photo transportable ;
- deux Raspberry Pi 4 ;
- deux HackRF One.

Un schéma du système est présenté ci-dessous :

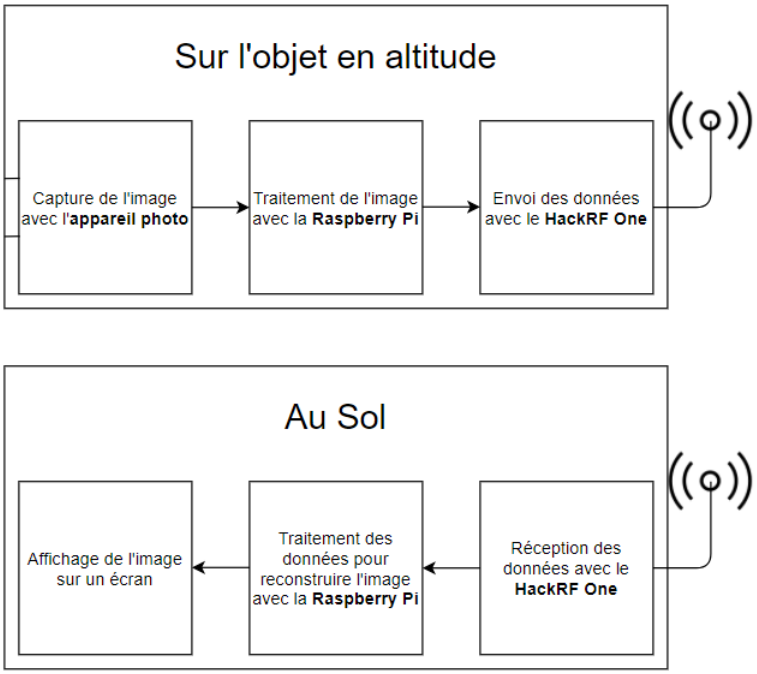

Figure 5 – Les étape de la transmission d'une image

Le rôle de la Raspberry Pi est de traiter les données de l'image (modulation, démodulation...) pour pouvoir effectuer la transmission. Le rôle du HackRF est la transmission des données traitées par la Raspberry Pi.

3.1.2 Envoi la position par SMS

Toujours dans le cadre d'un projet comme "Eirballoon", il se peut que l'on soit à la recherche du ballon une fois son voyage terminé. Pour cela, on va ajouter un module qui permettra de signaler la position de l'objet par SMS. Voici le principe :

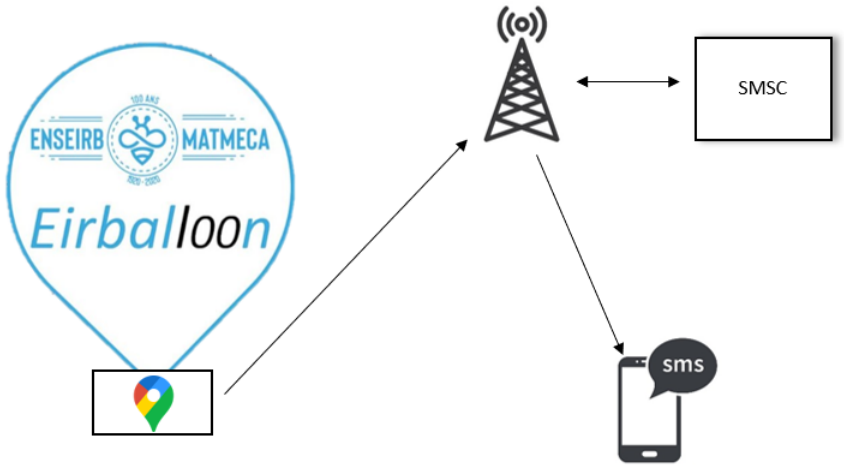

Figure 6 – Principe de la transmission de la position par SMS

Le système sur l'objet en mouvement récupère la position de l'objet et l'envoie par SMS à la personne désignée.

Pour pouvoir faire cela, nous avons à notre disposition :

— une Raspberry Pi 4 ;

— un module SIM Fona.

La Raspberry Pi gère l'automatisation de l'envoi de la position. Tout d'abord, aux moments voulus, elle récupère les données de localisation via le module SIM. Ensuite, elle traite ces données et les transmet au module SIM qui envoie le SMS.

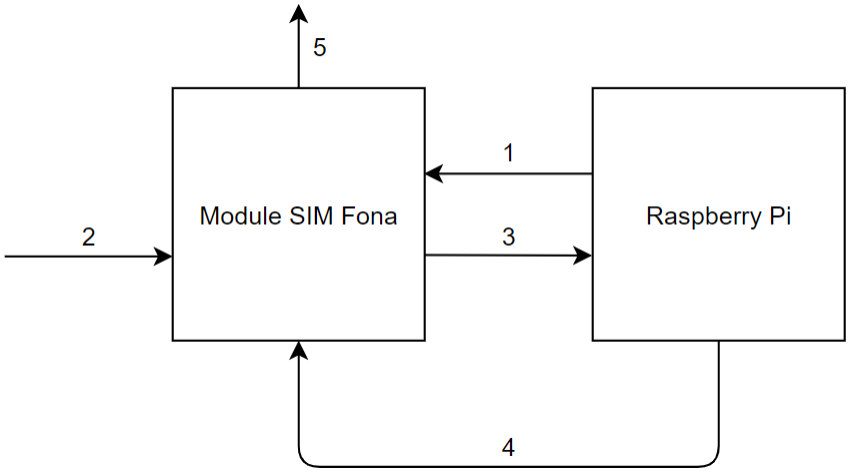

Figure 7 – Principe de l'envoi de la position par SMS avec la Raspberry et le module SIM

Dans l'ordre :

1. la Raspberry demande la position au module SIM ;
2. le module SIM récupère la position ;
3. le module SIM transmet les données à la Raspberry ;
4. après traitement des données, la Raspberry demande au module SIM d'envoyer la position par SMS ;
5. le module SIM envoie le SMS.

3.2 Organisation

3.2.1 Organisation générale

L'équipe projet est constitué de quatre personnes (deux binômes). Nous étions interdépendants les uns des autres et nous avons géré le projet sous la forme d'un groupe de quatre.

Notre équipe était donc constitué de :

— Binôme PC-1 : Yassine AMRANI et Matthias VERHOEVEN

— Binôme PC-2 : Léa VOLPIN et Maël LAVIEC (parcours ingénieur docteur)

Nous avions treize séances de trois à quatre heures pour mener à bien ce projet.

Après avoir passer la première séance et une partie de la deuxième à comprendre exactement ce que nous devions faire et à réaliser une première analyse des documents à notre disposition, les rôles se sont peu à peu définis. Léa et Maël se sont penchés sur l'analyse de deux chapitres essentiels à la mise en place d'une chaîne communication fonctionnelle et sur sa mise en place sur Matlab. Yassine et Matthias se sont occupés de la réalisation d'une première transmission, de la partie sur l'envoi de SMS et du début du codage en C++.

Voici le planning que nous souhaitions tenir pour réaliser le projet à l'issu de la sixième séance :

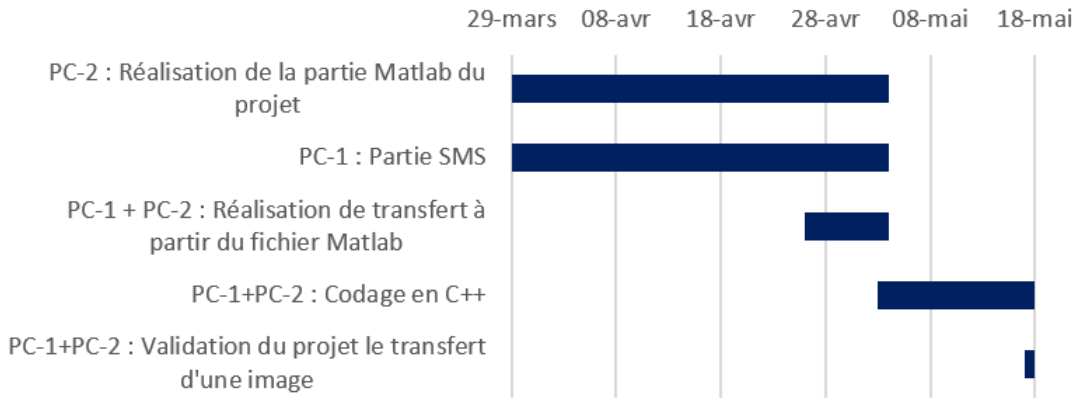

Figure 8 – Diagramme de GANTT pour les sept dernières séances

Ainsi, nous aurions une phase de quatre séances se terminant par la validation de la partie SMS (par un envoi automatique de SMS) et de la partie Matlab (par un transfert réel d'image fonctionnel). Une fois cette phase terminée, la dernière partie se déroulant sur les trois dernières séances concernerait la programmation en C++ et la validation finale du projet. Pour plusieurs raisons que nous verrons dans la sous-partie suivante, ces objectifs temporels n'ont pas pu être respectés.

En réalité, voici le récapitulatif de ce que nous avons fait pendant les séances :

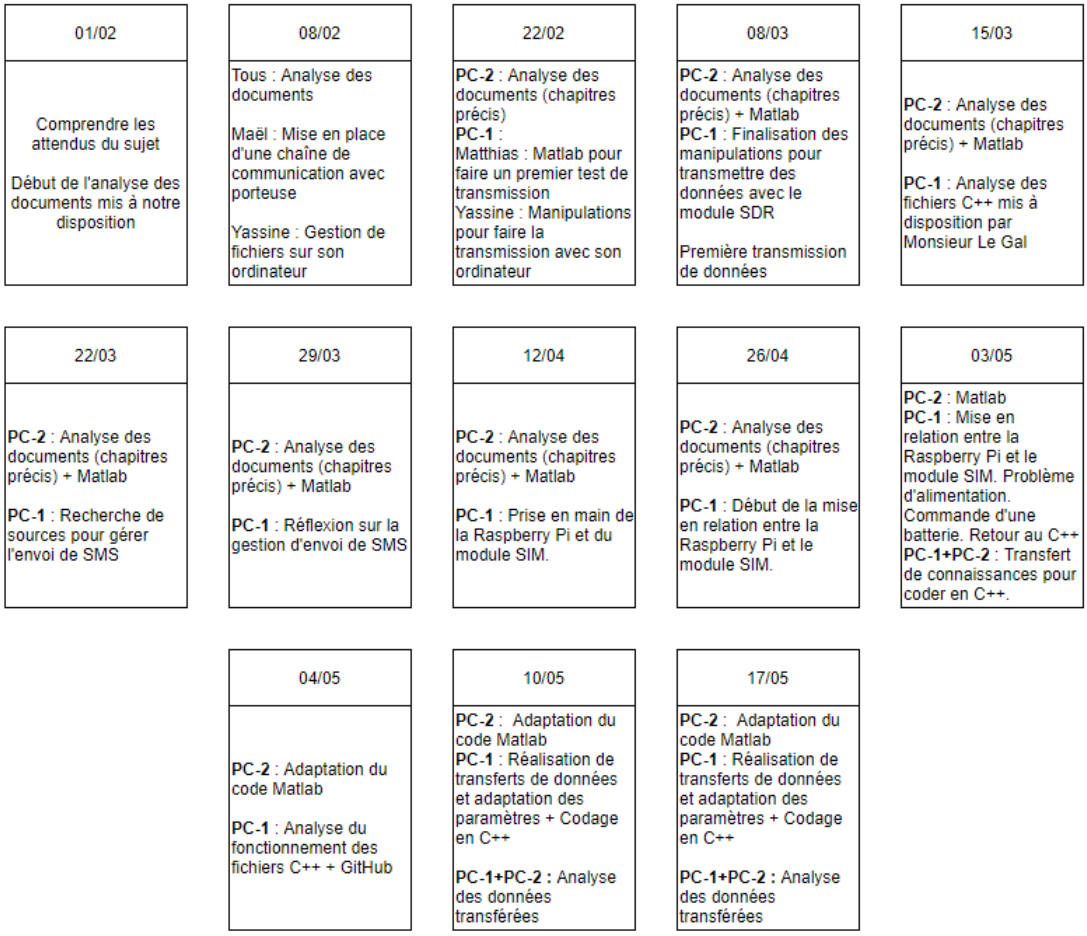

Figure 9 – Récapitulatif de ce que nous avons fait pendant les séances

3.2.2 Déroulement pour le binôme PC-1

A partir de la figure ci-dessus, nous allons vous expliquer notre démarche et comment les différentes étapes se sont déroulées.

Lors des premières séances, nous avons eu quelques difficultés à cadrer le projet et à le découper en plusieurs sous-parties à effectuer les unes après les autres. Après avoir analysé le sujet tous les quatre, Yassine s'est occupé de faire les réglages nécessaires sur son ordinateur pour pouvoir réaliser les transmissions réelles de données et Matthias s'est penché sur le code Matlab d'une communication numérique fonctionnelle pour en tirer une émission et tester une transmission réelle.

Une fois cette étape faite, nous nous sommes penché sur la réalisation de la partie émission en C++ raccord avec le code Matlab précédent. Nous avons analysé les fichiers ressources de Monsieur Le Gal et commencé à coder la partie bits vers symboles. En discutant ensuite avec Monsieur Le Gal et après la demande de Monsieur Ferré de réaliser un système permettant d'envoyer automatiquement la géolocalisation par SMS, nous nous sommes occupés de l'envoi de SMS ; Monsieur Le Gal nous a conseillé de reprendre la partie C++ une fois qu'une transmission à partir des codes Matlab sera réalisée.

Concernant la partie SMS, dans l'ordre nous avons :

— regardé ce qu'il existait déjà ;

— schématisé ce que nous voulions faire (automatisation de l'envoi) stylo à la main ;

— manipulé la Raspberry Pi et le module SIM ;

— recherché un protocole pour lier la Raspberry et le module SIM ;
— mis en place un protocole.

Concernant ce qu'il existe, nous n'avons pas trouvé ce que nous voulions mais cette étape nous a permis de mettre sur le papier une représentation schématique de la procédure d'envoi de SMS. Ensuite, nous avons manipulé la Raspberry et vu comment l'utilisé à distance avec un autre ordinateur. Pour lier la Raspberry et le module SIM, nous pensions au départ pouvoir les brancher entre eux comme s'il s'agissait d'ajouter un module USB classique puis d'écrire dans le module comme s'il s'agissait d'un fichier ; malheureusement ce fut plus complexe que cela. Nous avons alors cherché et trouvé un protocole pour lier manuellement les deux objets. Même si nous avions les témoins lumineux qui semblait nous dire qu'il y avait une connexion entre les deux, on n'obtenait pas les résultats espérés et la Raspberry nous signalait un problème d'alimentation. Nous avons alors commandé une batterie mais celle-ci ne fut disponible qu'à la fin de notre dernière séance ; c'est alors que nous avons changé de plan pour quand même continuer à avancer sur le projet.

En effet, même si la partie Matlab du deuxième groupe n'était pas encore tout à fait fonctionnelle, nous avons effectué un transfert de connaissance du groupe PC-2 au groupe PC-1 pour pouvoir au moins commencé la séance suivante la partie émission. La séance suivante (le 04/05), nous avons analysé le fonctionnement des fichiers C++ mis à notre disposition et appris à utilisé l'outil GitHub. Les deux dernières séances, nous n'avons pas pu terminer la partie émission en C++ ; en effet, nous avons, en parallèle de la programmation, réalisé des transferts de données et ajuster les paramètres d'émission et de réception : cela nous semblait prioritaire. Au final, le système émission/réception réel à partir du code Matlab nous a permis de transférer avec succès une partie d'une image.

4 Un système simple peut-il suffire pour transmettre les images ?

Pour commencer, on va mettre en place sur Matlab une communication numérique fonctionnelle, en tirer une partie émission et tester une transmission réelle.

4.1 Simulation sur Matlab d'une chaîne Tx/Rx simple et fonctionnelle

Pour commencer, nous allons nous intéresser sur la réalisation d'un système de communication numérique QPSK. Étant donné que le module SDR s'occupe de mettre le signal émis sur fréquence porteuse, notre système ici est en bande de base. Nous simulons alors la chaîne de communication schématisée ci dessous :

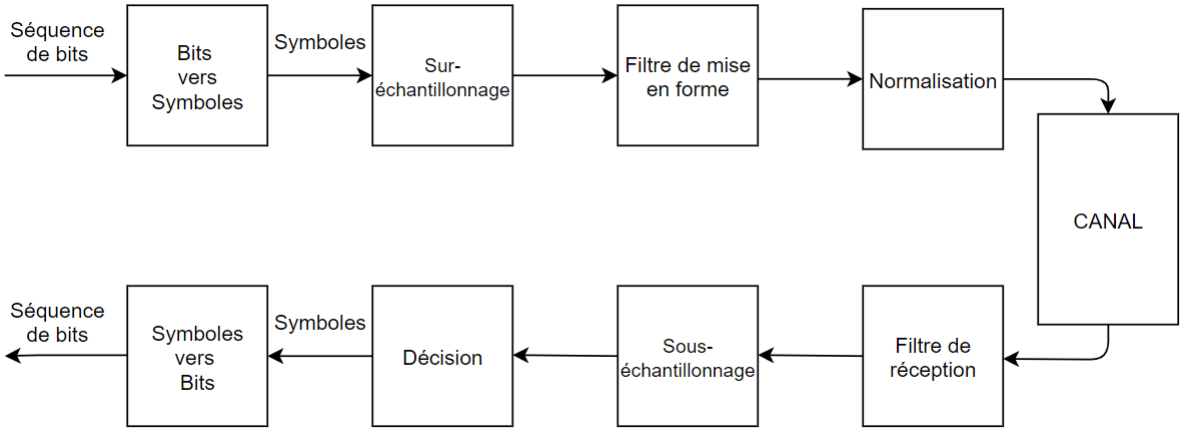

Figure 10 – Chaîne de communications numériques en bande de base

Les étapes :
— On génère tout d'abord une séquence aléatoire de bits ;

— On convertit le flux de bits d'entrée en symboles QPSK comme ceci :

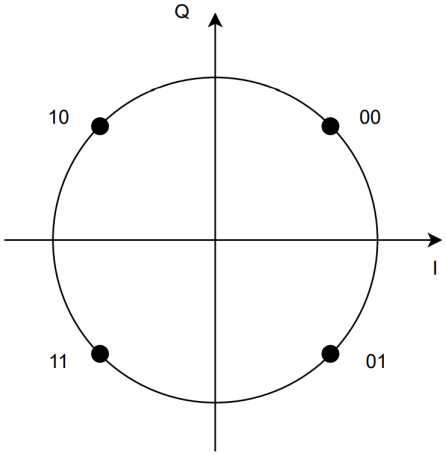

Figure 11 – Constellation I/Q du modulation QPSK avec mapping de Gray

— On effectue un sur-échantillonnage de paramètre Fse = 10 ;
— On passe alors par un filtre de mise en forme en racine de cosinus surélevé ;
— Les échantillons de signaux sont mis sur 8 bits ;
— On passe dans le canal ;
— On utilise ensuite un filtre en réception avant de sous échantillonner ;
— Le bloc de décision permet de discriminer les signaux en fonction de seuils et d'en déduire les symboles associés ;
— Pour finir, on convertit ces symboles en bits (qui se trouve être égaux à ceux générés au début)

4.2 Test d'une émission de signaux et observation du signal reçu

Une fois la partie précédente validée, on adapte notre partie émission pour tester une transmission réelle ; on fait notamment une mise à l'échelle des signaux à transmettre (de -120 à 120 sur 8 bits). L'objectif de cette partie sur Matlab est de créer un fichier .raw avec nos signaux à émettre. (Code Matlab en Annexe) Une fois ce fichier construit, on effectue une transmission test avec les modules SDR, un en émission et l'autre en réception. Pour cela, dans un terminal sous Linux, on applique les commandes suivantes :

— **Émission** : hackrf_transfer -t /home/amrani/Bureau/QPSK_Tx_Q_canal.raw -f 2450000000 -s 10000000 -x 32 -p 1 -a 1 -d 000000000000000088869dc242e9d1b -R
— **Réception** : hackrf_transfer -r /home/amrani/Bureau/QPSK_Rx_Q_canal.raw -f 2450000000 -s 10000000 -l 24 -g 24 -d 00000000000000075b068dc317bae07

Les paramètres de la transmission sont les suivants :

— fréquence de la porteuse : 2,45 Mhz ;
— fréquence d'échantillonnage : 10 kHz ;
— gain en émission : 32 dB ;
— gain en réception : 24 dB ;

Une fois la transmission effectuée, on ouvre les fichiers de l'émission et de la transmission sur Audacity :

— **Signal transmis :**

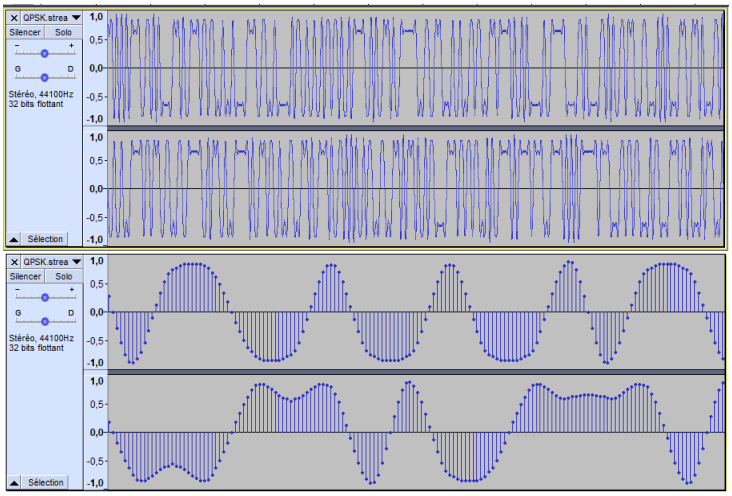

Figure 12 – Signal transmis au module SDR

La partie en haut de l'image est associée au canal I et la partie en bas est associée au canal Q. On remarque qu'il y a a bien une succession de symboles tous les Fse = 10 échantillons.

— **Signal reçu :**

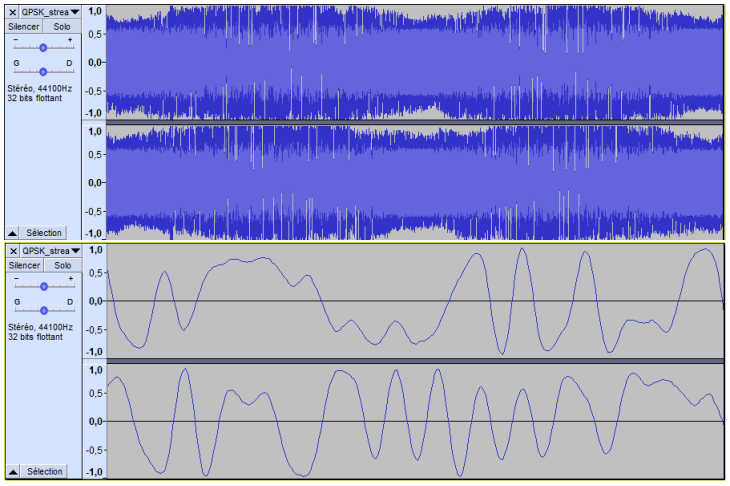

Figure 13 – Signal transmis au module SDR

On peut remarquer qu'on observe bien les symboles sur les canaux I et Q malgré de légers décalages en amplitude. Les aspects liés à l'amélioration de la transmission des données, comme la synchronisation temporelle et fréquentielle par exemple, ont été traités par le groupe PC-2.

5 L'envoi des SMS

5.1 Manipulation du matériel

5.1.1 Branchements

Voici la liste du matériel nécessaire pour la manipulation :

— la Raspberry Pi ;
— le module SIM Fona ;
— une batterie de 3,7V ;
— une carte SIM ;

— un adaptateur pour agrandir la surface de la carte SIM ;
— des "jumper wires" (des petits câbles pouvant relier des pins) ;
— un câble USB/microUSB.

Pour réaliser le branchement entre la Raspberry et le Module SIM, on utilise les petits fils et on relis les pins de chaque objet de la façon suivante :

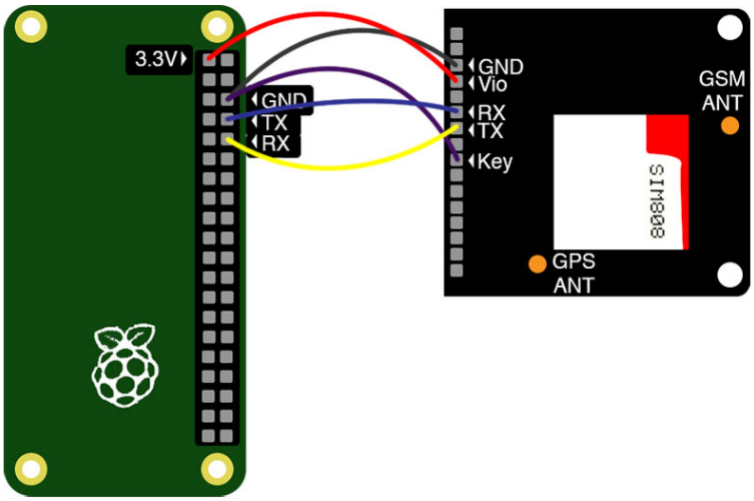

Figure 14 – Branchement Raspberry-Fona (source digikey.com)

Ensuite, on place la carte SIM dans l'adaptateur de carte SIM que l'on glisse ensuite dans l'encoche réservée à cet effet dans le module SIM :

Figure 15 – Mise de l'adaptateur

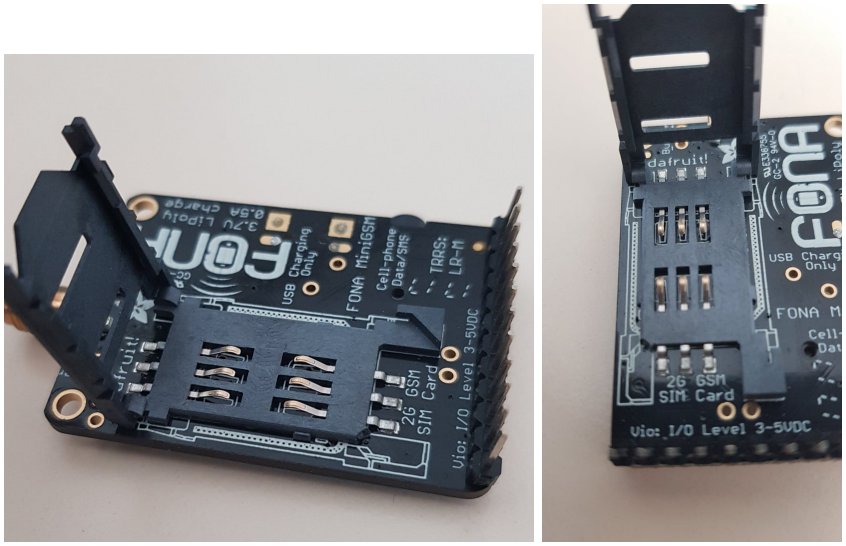

Figure 16 – Espace réservée à la carte SIM sur la Fona

Puis, on branche la batterie sur la Fona. Enfin, on relie la Fona et la Raspberry avec le câble USB/microUSB pour alimenter également la Raspberry.

Pendant les séances, nous n'avions pas de batterie et nous avons donc alimenter le duo par la prise d'alimentation "classique" de la Raspberry Pi.

5.1.2 Mise en place sur la Raspberry

Après avoir connecté le module carte SIM et la Raspberry avec tous les composants extérieurs nécessaires, on passe maintenant à la configuration interne de la Raspberry afin d'afficher le popscreen qui joue le rôle d'une interface HM.

Premièrement, il faut taper la ligne de commande ci-dessous dans le terminal, puis il faut choisir le mode écriture et ajouter à la fin "enable_uart=1". Finalement, il faut sauvegarder avant de quitter le terminal et faire un reboot (redémarrage) pour la Raspberry.

```
nano /boot/config.txt
```

Ensuite, on s'assure qu'on est bien connecté au Wi-Fi avant de lancer les lignes de commandes suivantes permettant d'installer "Popscreen" (PPP) sur le Pi.

```
sudo apt-get update
sudo apt-get install ppp screen
```

Finalement, on lance le "popscreen" installé avec la commande suivante :

```
sudo screen /dev/serial0 115200
```

Après cette étape normalement on peut communiquer avec le module carte SIM en utilisant les commandes "AT" à travers le "popscreen", or dans notre situation on n'a pas réussi à cela à cause de la nécessité du matériel comme bien expliqué antérieurement.

5.1.3 Test des commandes "AT" du module carte SIM

Après avoir connecté le module carte SIM à la Raspberry Pi 4 et effectué tous les branchements nécessaire, l'étape suivante sera d'essayer de se communiquer manuellement avec ce module et envoyer un SMS à l'aide de l'interface "popscreen" sur le terminal .

Premièrement, il faut Commencer par initialiser l'auto-baud'er en envoyant AT puis le système doit envoyer normalement "OK" pour confirmer le succès de l'opération :

Figure 17 – Commande "AT"

On pourra ensuite envoyer plusieurs commandes permettant d'effectuer diverses tâches :

- **ATI :** La commande permet d'obtenir le nom du module.
- **AT + CMEE = 2 :** Activer les erreurs détaillées (pratique lorsque on essayera des commandes !).
- **AT + CCID :** permet d'obtenir le numéro de la carte SIM, cela teste aussi que la carte SIM est bien trouvée en envoyant un "OK" sur l'écran.

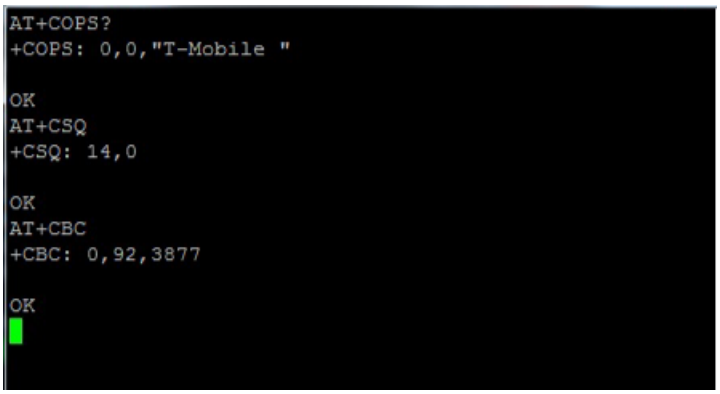

Figure 18 – L'affichage des commandes sur le "popscreen"

- **AT + CBC :** Cette commande retournera l'état de la batterie Lipo.
- **AT + CSQ :** Cette commande Vérifie la force du signal en dB, il doit être supérieur à environ 5. À noter que plus il est élevé plus c'est mieux pour le système.
- **AT+CMGF=1 :** Cette commande est très importante pour pouvoir écrire le message sinon on ne peut tout simplement pas le taper car cela le mettra en mode "TEXTE" et non en mode PDU (données).
- **AT + CMGS = "Numéro" :** Après avoir taper cette commande, on pourra normalement écrire le message avant l'envoyer avec un simple un [Control-Z] sur une ligne vide à envoyer. Enfin, lorsque l'opération est terminée on obtient normalement un "OK" qui confirme le succès de cette dernière.

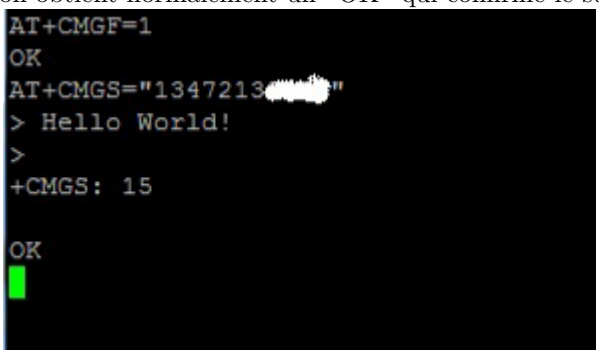

Figure 19 – L'envoi du SMS manuellement sur le "popscreen"

5.1.4 Automatisation de l'envoi de SMS

Afin de recevoir régulièrement la position du système mobile sans saturer de SMS le receveur des positions, nous avons choisi que l'envoi devrait s'effectuer toutes les demi-heures. Ainsi, une fois que l'on active le système, on attend trente minutes puis on récupère les données GPS que l'on stocke dans un buffer. Ensuite, on envoie les données à la personne chargée de recevoir les SMS ; on attend l'accusé de réception, si on ne le reçoit pas dans les trente secondes on envoie à nouveau le message. On revient ensuite dans l'état initial où l'on attend trente minutes.

On peut représenter ce système en GRAFCET :

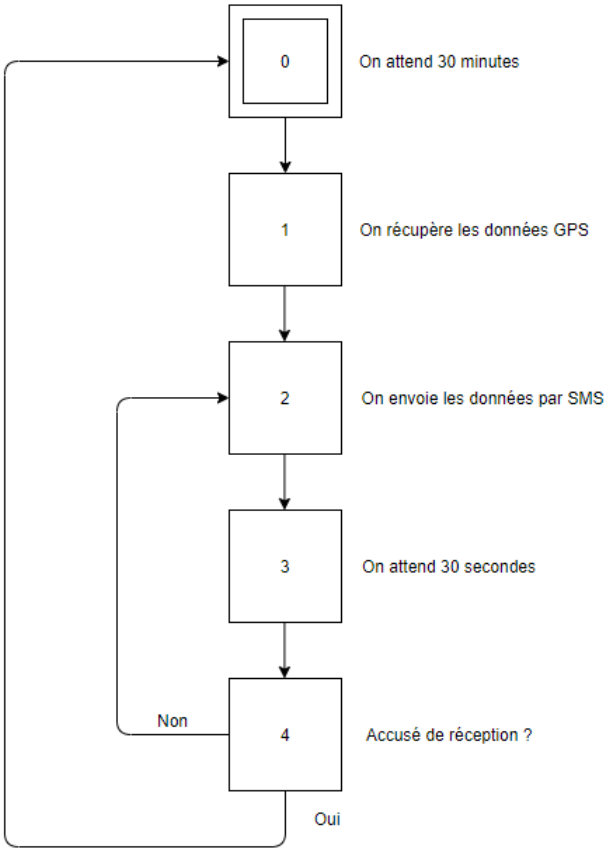

Figure 20 – Représentation GRAFCET de l'automatisation de l'envoi de SMS

Malheureusement, par non validation de la sous-partie précédente et par manque de temps, nous n'avons pas pu programmer quelque chose de fonctionnel en C++. Néanmoins, voici le code C++ associé dépendant de fonctions intermédiaires.

```
void main(){
    while(1){
        sleep(1800);                // On attend 30 minutes
        recupGPS();                 // On récupère les données GPS
        SendSMS();                  // On envoie le SMS
        sleep(30);                  // On attend 30 secondes
        while(accuseRecep()==0){    // Tant que l'on n'a pas reçu l'accusé de réception
            SendSMS();              // On envoie le SMS
            sleep(30);              // On attend 30 secondes
        }
    }
}
```

Figure 21 – Fonction main de l'envoie de SMS

Seules deux fonctions intermédiaires ont pu être réalisées mais aucune n'a pu être testée.

6 Tests émission/réception à l'aide des modules SDR pour le groupe PC-2

Afin de valider le code Matlab fait par le groupe PC-2, on était obliger de faire plusieurs tests de divers fichiers ".raw" pour qu'ils puissent les analyser et par suite essayer d'adapter le code pour répondre aux cahier des charges.

Le transfert des données se fait à l'aide des modules SDR en utilisant les deux lignes de commandes suivantes :

- **Emission :** hackrftransfer -t /home/amrani/Bureau/QPSKTxQcanal.raw -f 2450000000 -s 10000000- x 28 -p 1 -a 1 -d 000000000000000088869dc242e9d1b -R
- **Reception :** hackrftransfer -r /home/amrani/Bureau/QPSKRxQcanal.raw -f 2450000000 -s 10000000- l 24 -g 24 -d 00000000000000075b068dc317bae07

D'ailleurs, on pourra spécifier les paramètres utilisés dans les lignes de commandes pour mieux comprendre ses utilités :

- **-t** Transmettre les données qui se trouve dans un fichier "raw". On place le chemin du fichier qu'on veut transmette après le "-t".
- **-r** La réception des données transmises suivi du chemin où on souhaite stocker le nouveau fichier de la réception.
- **-f** représente la fréquence de la porteuse et on utilise en générale une porteuse de 2.45 GHz.
- **-s** représente le sample rate qui est par défaut égale à 10MHz. C'est bel et bien la même fréquence d'échantillonnage qu'on utilise.
- **-R** permet de répéter la transmission plusieurs fois automatiquement jusqu'à la stopper manuellement avec un "ctrl+C" pour une bonne réception des données. Alors, cela sert surtout pour ne pas perdre les données même si les deux lignes de commandes ne se lancent pas au même instant.
- **-l** RX LNA gain, c'est un gain de réception qu'on peut régler selon nos besoin de la transmission et qui peut aller de 0 jusqu'à 40 dB avec un pas de 8 dB. Dans notre cas, on utilise un gain LNA de 24 dB.
- **-g** RX VGA (bande de base) gain, c'est un gain de réception allant aussi de 0 jusqu'à 62 dB et il ne faut pas que sa valeur soit très loin de la valeur du RX LNA c'est donc pour cela qu'on le choisit égale à 24 dB aussi.
- **-d** représente le serial du HackRf qu'il faut utiliser.
- **-x** TX VGA gain, c'est le seul gain de la transmission et sa valeur peut varier entre 0 et 47dB. Or, après plusieurs tests on a conclut qu'il faut l'ajuster sur 28dB pour qu'il marche parfaitement.
- **-a** Amplificateur RF RX / TX s'il était égale à 1 on l'active et s'il était égale à 0 on le désactive. Dans notre cas, a=1.
- **-p** Alimentation du port d'antenne, activé s'il était égale à 1 et désactivé à 0.

Après avoir fait le test avec les deux lignes de commandes, on utilise Audacity pour visualiser les fichiers d'émission et de transmission pour différents valeurs du gain TX VGA.

Voici un exemple de série de tests réalisée :

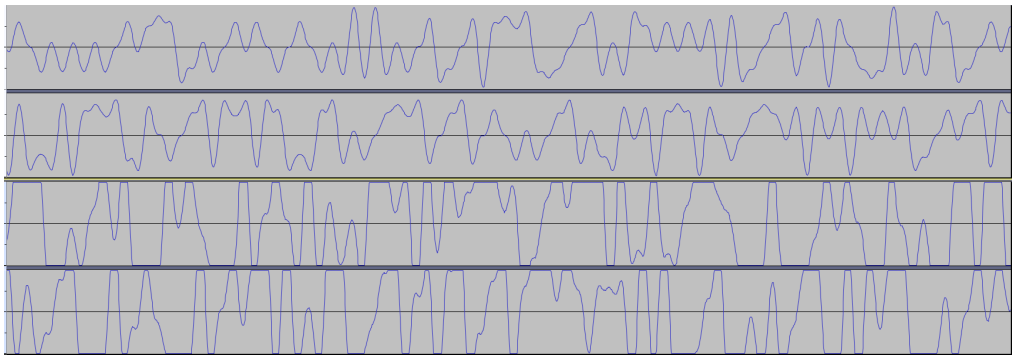

Figure 22 – Signal transmis (en dessus) et reçu (en dessous) avec -x 32

Commentaire : On remarque que les signaux saturent en réception, le gain est trop élevé. On effectue la même chose avec un gain plus bas :

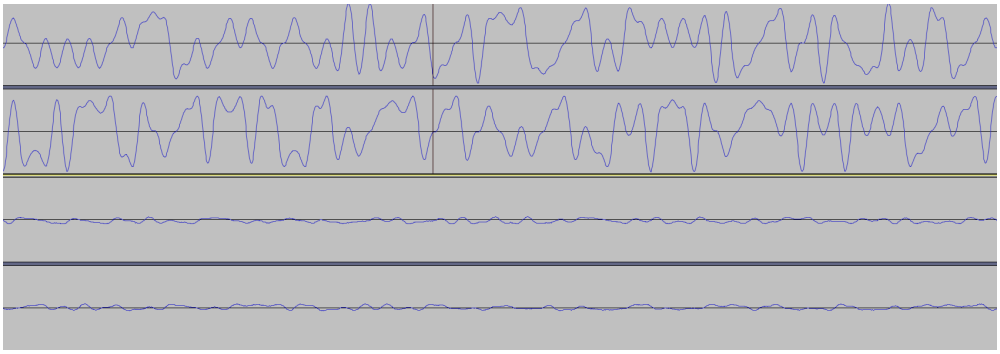

Figure 23 – Signal transmis (en dessus) et reçu (en dessous) avec -x 16

Commentaire : Cette fois-ci, les signaux en réception sont faibles. On va alors chercher un juste milieu :

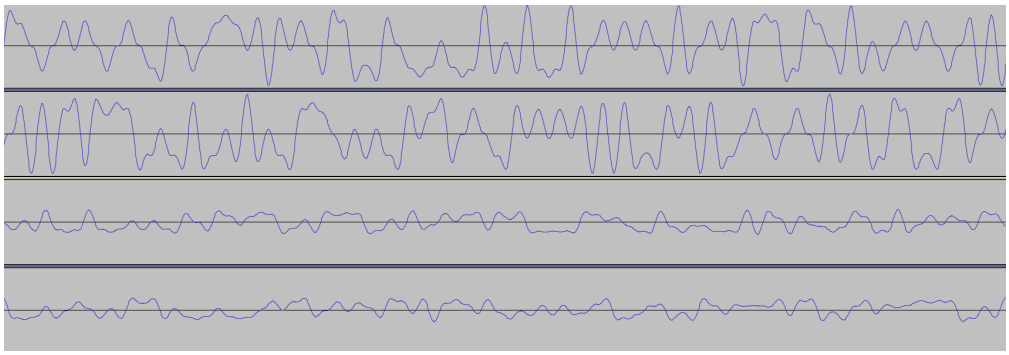

Figure 24 – Signal transmis (en dessus) et reçu (en dessous) avec -x 28

Commentaire : Cette fois-ci, cela nous semble plutôt correct et permettra sûrement au groupe PC-2 d'obtenir quelque chose d'intéressant.

Ainsi, une fois l'habitude prise, le groupe PC-2 nous envoyait des fichiers dont on faisait les transferts, on analysait avec eux les signaux reçus et on leur transmettait les fichiers intéressants. Nous avons, en réalité, effectué un grand nombre de transmissions avec plusieurs paramètres. Entre chaque transmission, nous adaptions nos lignes de commande ; eux, entre chaque phase de transmissions, ajustaient leur code Matlab.

7 Conversion Matlab - C++

Du binôme PC-2, nous avons récupéré la façon dont ils ont réalisé leur partie "émission" de la chaîne de communication sur Matlab. On peut schématiser la chaîne d'émission pour le C++ comme ceci :

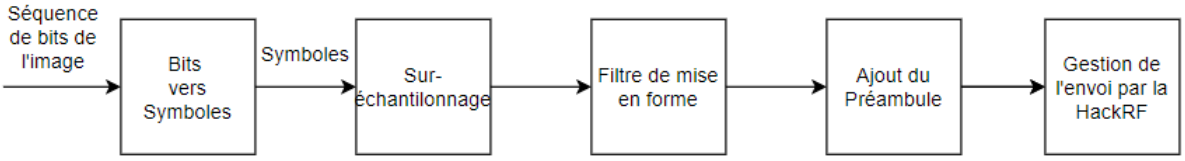

Figure 25 – Chaîne d'émission

Les deux premiers blocs représente une classe particulière, elles contiennent :
— un attribut ;
— un constructeur ;
— un destructeur ;
— une méthode.
Par exemple pour le sur-échantillonnage, nous avons :

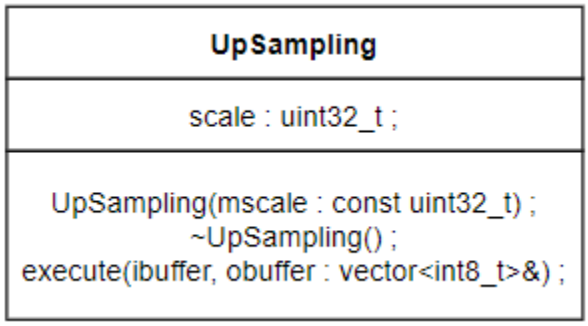

Figure 26 – Classe UpSampling

Pour le passage dans le filtre de mise en forme, il faut enregistrer la matrice de ce filtre accessible sur Matlab et de faire la convolution à la main à partir de boucles for.
Pour le préambule, il faut enregistrer ses valeurs dans un vecteur et concaténer celui-ci avec le signal reçu.

La gestion de l'envoi, comme son nom l'indique, doit gérer l'envoi des données par la HackRF.

Seule la partie "Sur-échantillonnage" a été validée.

8 Conclusion

Ce projet thématique a été l'occasion pour nous de mettre en application des notions vues en cours mais surtout de découvrir de nouvelles choses. Il nous aura également permis de nous rapprocher du métier d'ingénieur, de ses difficultés et de son intérêt. Même si ce projet n'a pas pu aboutir à ce que nous espérions, nous avons appris à manipuler de nouveaux outils : une Raspberry Pi, un module carte SIM Fona, GitHub, VNC... et à nous organiser à quatre sur un projet assez long.

Nous avons essayé dans ce rapport de mettre suffisamment d'éléments dans le cas où ce projet ait vocation à être terminer ultérieurement par un autre groupe ou une autre personne.

9 Annexes

9.1 Code Matlab pour le premier test d'émission de données :

```
%% Projet SdR : Emission
%Volpin Lea
%Laviec Mael
%Amrani Yassine
%Verhoeven Ceccaldi Matthias

clear all; %Efface les variables de l'environnement de travail
close all; %Ferme les figures ouvertes
clc;       %Efface la console

%% Parametres de la chaine de communication
fe=1e4;%frequence d'echantillonnage
Te=1/fe; %periode d'echantillonnage
M=4;%nombre de symbole
n_b=log2(M);%nombre de bits par symboles
Ds=1e3; %Debit symbole
Ts=1/Ds; %periode entre chaque symbole
roll_off=0.8; %facteur de roll-off
Fse=fe/Ds; %facteur de surechantillonnage
Nfft=512; %nombre de point pour fft
Ns=10000; %Nombre de symboles a emettre

%% Recuperation des bits
sbt=randi([0,1], 1,n_b*Ns); %Sequence de bits aleatoirement et uniformement distribues ici

%% bits vers symboles
sst_Ts = bits_symboles_QPSK(sbt, Ns, n_b);

%% surechantilonnage
sst_Te=upsample(sst_Ts, Fse); %Sur-echantillonage

%% filtre de mise en forme
g=rcosdesign(roll_off, M, Fse, 'sqrt'); %filtre de mise en forme, pour tg=4 Ts et 10 echan
slt=conv(g,sst_Te); %Convolution du filtre avec ssn_Te

%% Normalisation
n = 8;
d_Re = max(abs(real(slt)));
d_Im = max(abs(imag(slt)));
d=max(d_Re, d_Im);

z = int8(slt .* (120 / d));  % On scale les donnees sur l'intervalle -120 <=> +120

len = length(z);
y   = int8(zeros(2 * len, 1)); % on cree le vecteur d'echantillons

for i = 1:len
    y( 2*i-1 ) = real( z(i) );
```

```
    y( 2*i    ) = imag( z(i) );
end

% On memorise les donnees a transmettre
fid = fopen('QPSK.stream.raw', 'w');

for i = 1:100
    fwrite(fid, y, 'int8');
end
fclose(fid);
```

9.2 Lien GitHub pour accéder à nos fichiers

Lien pour avoir accès aux fichiers :
`https://github.com/Birdy788/adsb-like-comm-toolbox`

Les fichiers commencés et intéressants sont dans :
- adsb-like-comm-toolbox/src/Processing/
- adsb-like-comm-toolbox/src/Chains/QPSK_chain/Encoder/

10 Bibliographie

Fichiers sources de Monsieur Le Gal :

- `https://github.com/blegal/adsb-like-comm-toolbox`

Liens sur la liaison entre la Fona et la Raspberry Pi :

- `https://www.digikey.com/en/maker/projects/cellular-gps-enabled-pi-3-fona-pi-3/d0cf660bfc144842a49`
- `https://www.youtube.com/watch?v=NArpvpmmpUU`
- `https://learn.adafruit.com/fona-tethering-to-raspberry-pi-or-beaglebone-black`
- `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-fona-mini-gsm-gprs-cellular-phone-module.pdf`

Transferts de données avec les HackRF :

- `http://manpages.ubuntu.com/manpages/xenial/man1/hackrf_transfer.1.html`

Pins de la Raspberry Pi 4 :

- `https://www.raspberrypi.org/documentation/usage/gpio/`