PONTAGNIER Etienne
VUONG Dan

E2

Projet Thématique

–

Communication sans fils

[PR214]

Tuteurs :

Romain Tajan
Bertrand Le Gal

Bordeaux INP
ENSEIRB
MATMECA

Sommaire

I. Introduction

1. Objectifs

L'objectif de ce projet est de mettre en place une communication sans fil entre deux radios. Les deux radios utilisées pour ce projet sont des radios de type USRP. Ces radios utilisent une carte électronique USRP qui permet une connexion entre celles-ci et un ordinateur pour pouvoir interagir avec un logiciel de radio. Le but étant d'envoyer des données d'un ordinateur à l'autre que ce soit un fichier texte, une image ou une vidéo.

Les radios USRP n'ont pas besoin d'être programmées, elles permettent d'être utilisée avec le logiciel GNU radio qui permet d'envoyer et de recevoir des données. Cependant il faut au préalable, au niveau de l'émission, mettre en forme les données à envoyer et au niveau de la réception, traiter les données reçues afin de les rendre compréhensible.

Ce rapport décrit le fonctionnement de la chaine de communication, les étapes qui la compose ainsi que les différentes fonctions et algorithmes utilisées. Le code définissant le fonctionnement des radios USRP ne sera pas explicité en détail.

2. Cahier des charges

Le procédé de codage utilisé pour les signaux envoyés sera le procédé OFDM. La chaine de communication ainsi réalisée devra permettre l'envoi ainsi que la réception de données telles qu'un fichier texte, une image ou une vidéo. Le procédé OFDM devra permettre :

- de limiter le décalage temporel ;

- de limiter le décalage en fréquence ;

- de limiter le bruit introduit par le canal ;

- de limiter l'action du coefficient de canal.

Pour le début du projet la chaine de communication sera modélisée sous Matlab afin de faciliter son développement. Par la suite le traitement des données sera codé en C++ afin de pouvoir être utilisé avec les radios USRP.

II. Description de l'architecture et fonctionnement.

1. Format des données utilisées

Voici les paramètres que nous choisissons pour notre standard qui se base sur le procédé de codage OFDM :

• $n = 128$: nombre de sous-porteuses pour nos symboles OFDM ;

• $l = 16$: taille des préfixes cycliques ;

• N: nombre de trames d'information dans le signal émis ;

• $ISR = 10$: ratio trames d'information sur trames de synchronisation (une trame pilote pour 10 trames d'information) ;

• $N_{sync} = \frac{N}{ISR}$: nombre de trames de synchronisation ;

• $Ns = n * (N + N_{sync})$: nombre total de données à envoyer ;

• Modulation de phase : BPSK

2. Fonctionnement de la chaine de communication

a) Partie émission

Pour que les radios puissent envoyer efficacement les données et pour éviter que des erreurs ou du bruit soient introduits dans le canal de transmission, ces données doivent d'abord être traitées. Les données seront transmises par modulation de phase.

La première étape consiste à mettre en forme les données binaires pour la modulation de phase pour ce projet c'est la méthode BPSK qui sera utilisée.

Une fois le vecteur ss formé, il faut le diviser en plusieurs sous vecteurs d'une taille défini au préalable. En effet le procédé OFDM est une modulation multi porteuse, c'est-à-dire que les données sont transmises en les modulant sur plusieurs porteuses en même temps. Donc les données du vecteur ss sont groupées par paquet de n éléments.

Une fois le signal ss multiplexé en plusieurs sous vecteurs, il faut ensuite les moduler. On effectue donc une IFFT sur les différents vecteurs parallèles pour cela.

L'envoi des données dans le canal au moment de la transmission peut provoquer un phénomène d'écho, les signaux envoyés peuvent être retardés par l'environnement et il peut donc y avoir une interférence entre deux trames émises successivement. Pour contrer cette interférence on insère un préfixe cyclique avant chaque trame qui est une copie de la fin de cette trame.

Maintenant que la donnée à transmettre est formée il faut la remettre sous forme d'un seul vecteur pour pouvoir la transmettre. Le vecteur est composé à ce moment-là de complexes doubles.

b) Canal de propagation

C'est ensuite la partie radio logiciel qui prend le relais. Elle se charge de d'envoyer les données mises en forme précédemment, à travers un canal de propagation.

Le canal que nous utilisons ici est réel et inconnu des deux radios. Cependant, nous pouvons anticiper les effets que ce canal aura sur le signal. Il est probable d'observer des décalages temporels ou fréquentiels, ou encore une action due aux coefficients de canal ou au bruit. Bien que le canal n'ait aucun impact sur la façon de coder les données à transmettre, il en est autrement pour le décodage, dans la partie réception.

On a donc, dans un second temps, une deuxième radio identique réglée en mode « réception » qui se charge de recevoir le signal. Le signal composé de complexes doubles plus ou moins modifiés va donc ensuite être traitée afin de reconstituer le signal d'origine et ceci, en suivant la procédure inverse que pour la chaine de transmission.

c) Partie réception

Afin de reconstituer le signal original il faut rediviser le vecteur en sous vecteurs pour pouvoir les traiter en parallèle.

Maintenant qu'on a récupéré et mis en parallèle le signal, on va chercher à détecter les préfixes cycliques insérés précédemment afin de les supprimer. Pour cela on utilisera la technique présentée par Schmidl et Cox qui permet de déterminer l'emplacement des préfixes cycliques dans le signal. Cette technique consiste à calculer le coefficient de corrélation pour déterminer la ressemblance de différents buffers du signal, si deux buffers sont ressemblants se sont sûrement des préfixes cycliques.

En plus de l'interférence entre trames il peut aussi y avoir un offset du signal lors de la réception. Lors de la réception de celui-ci il faut donc déterminer la durée de l'offset afin de le supprimer. Pour cela on réalisera une moyenne dans le temps afin d'obtenir la durée de l'offset et une fois cette durée obtenue on pourra corriger la position des préfixes cycliques obtenues précédemment.

Une fois la position des préfixes cycliques corrigés il suffit maintenant de tronquer le signal aux bons endroits afin de les supprimer.

Les données dans la chaine de transmission ont été modulées par une IFFT, pour reconstruire le signal d'origine il faut dont réaliser une FFT (Opération inverse de l'IFFT). Le signal obtenu est donc un vecteur composé de double.

Cette étape permet de réduire le bruit qui a pu être introduit dans le canal. Il s'agit d'associer chaque valeur du vecteur précèdent à sa valeur symbole.

Pour reconstituer le signal d'origine il suffit maintenant d'associer chaque symbole à sa valeur binaire selon la modulation BPSK.

III. Description des blocs de traitement des données en émission

Nous allons à présent détailler le fonctionnement de la chaîne d'émission, codée sous Matlab, dont le schéma se trouve en *annexe 1*. Pour cela, nous allons décrire le fonctionnement des blocs présents dans cette chaîne.

1. Détail des modules utilisés

Avant de travailler sur les radios il a d'abord fallu tester la chaine de communication sur Matlab. Pour cela on a donc généré un vecteur binaire aléatoire de Ns= 2560 éléments soit 20 trames de 128 éléments.

a) Association Bits/Symbole

Le vecteur x[] généré grâce à la fonction rand(2, 1, Ns) de Matlab nous permet d'avoir une vecteur ss[] de Ns entiers compris entre 0 et 1 de manière aléatoire. Pour associer les bits à leurs symboles équivalent il suffit sous Matlab de multiplier le vecteur par 2 et soustraire 1 à chaque élément, de cette façon les 0 binaire deviennent des -1 et les 1 restent des 1.

b) Série > Parallèle et IFFT

Là où la mise en parallèle des vecteurs aurait dû se faire avant l'IFFT les outils de Matlab permettent de faire ces deux étapes en simultanées. On réalise ensuite une IFFT sur des sous vecteurs de ss[] de 128 éléments qui forment donc chacun une trame du signal. Le logiciel Matlab intègre déjà une fonction ifft() qui retourne un l'ifft d'un vecteur rentré en argument. Ces trames sont ensuite concaténées dans un seul vecteur ssp[] qui contient donc à la suite les 20 trames de 128 éléments.

c) Génération et insertion des trames de synchronisation

Pour limiter les problèmes de décalage de phase et aussi corriger l'effet du canal sur le signal il faut insérer des trames pilotes dans le signal d'origine. Ces trames pilotes sont connues et permettent donc de déterminer le coefficient de canal qui perturbe le signal d'origine, en effet il suffit de comparer la trame pilote envoyé et la trame pilote reçu qui a été altéré par le canal lors de la transmission. Pour le prototype sur matlab la trame pilote sera générée aléatoirement avec la fonction randi() de matlab. On choisira d'insérer une trame pilote toutes les 10 trames OFDM.

d) Insertion de préfixes cyclique

Pour éviter les phénomènes d'interférences entre symbole il faut insérer des préfixes cycliques avant chaque trame du signal, et comme préciser précédemment ces préfixes cycliques sont la copie des l=16 derniers éléments de chaque trame. Pour cela, sur Matlab on parcourt le vecteur ssp[] puis on stocke les l=16 derniers éléments de chaque trames pour les inclure entre celles-ci. Pour vérifier si l'algorithme à bien recopié les bons éléments on soustrait les 16 derniers éléments de chaque trame aux 16 premiers.

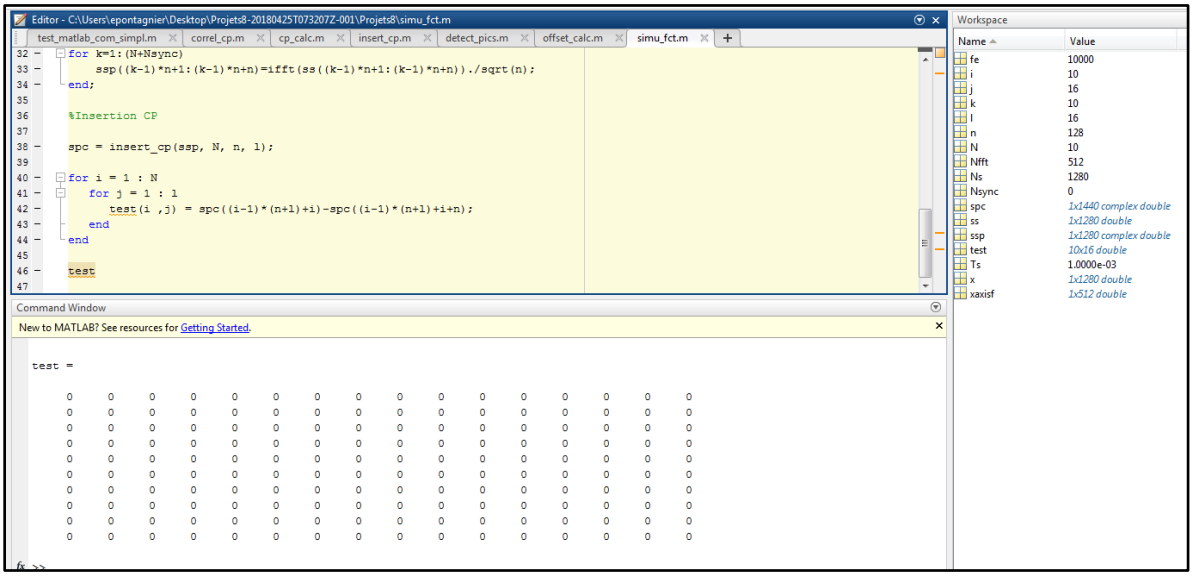

Figure 1 – Simulation de la fonction d'ajout des préfixes cycliques

On remarque bien que le résultat de cette soustraction est 0 ce qui indique bien les préfixes cycliques sont bien les copies des 16 derniers éléments de chaque trame.

Pour faciliter le prototypage de la chaine de communication on stockera une copie des ces préfixes cycliques dans un vecteur ainsi que leurs positions dans le signal.

```
% CREATION TABLEAU DE PREFIXES ET CALCUL DES INDICES

[prefixes, vrai_indice_prefixe] = cp_calc(ssp, N, Nsync, n, l, offset);
```

Figure 2 – Stockage des positions des préfixes

IV. Description des blocs de traitement des données en réception

Nous allons à présent détailler le fonctionnement de la chaîne de réception, codée sous Matlab, dont le schéma se trouve en *annexe 2*. Pour cela, nous allons décrire le fonctionnement des blocs présents dans cette chaîne. Chaque module comporte une ou plusieurs fonctions qui permettent de traiter le signal entrant.

1. Détail des modules utilisés :

a) Acquisition du signal

Le signal réceptionné ici provient tout droit du canal. La chaîne de réception attend un signal à n = 128 sous-porteuses, dont une trame contient, à ce moment de la communication, 144 symboles suite à l'ajout de préfixes cycliques. Les symboles composant le signal sont des nombres complexes obtenus après une *ifft*. La taille du préfixe cyclique est l = 16 symboles.

b) Détection des préfixes cycliques

Dans un premier temps, après réception du signal, il nous faut retrouver les préfixes cycliques apparaissant dans chaque trame. Nous créons une fonction pour réaliser cette tâche : « *corr_detect_max()* ». La recherche des préfixes s'effectue en deux opérations :

• D'abord, nous réalisons une corrélation entre deux fenêtres du signal, longues de l = 16 symboles, soit la longueur d'un préfixe cyclique. Ces deux vecteurs sont espacées de n = 128 symboles car le préfixe cyclique est une copie des l derniers symboles d'une trame au début de celle-ci. Ainsi, dans le cas où la première fenêtre est positionnée sur le préfixe cyclique d'une trame, elle sera similaire à la deuxième fenêtre.

La corrélation en elle-même est basée sur la relation de Cauchy-Schwartz suivante :

$$|< x, y >| \leq ||x||.||y|| \Leftrightarrow \frac{|<x,y>|}{||x||.||y||} \leq 1$$

En effet, la norme du produit scalaire de deux vecteurs x et y est maximale lorsque ceux sont colinéaires. Autrement dit, lorsque que les deux fenêtres seront positionnées sur un préfixe cyclique, le coefficient de corrélation sera maximum. Ainsi, on balaye notre signal avec les deux fenêtres qui nous appelleront « *f1* » et « *f2* » pour obtenir, pour chaque position, un coefficient de corrélation que nous appelleront « *c* ».

$$c = \frac{|< f1, f2 >|}{||f1||.||f2||}$$

On stocke ensuite dans un tableau toutes les valeurs du coefficient de corrélation. Nous pouvons observer en figure 1 la représentation de cette corrélation.

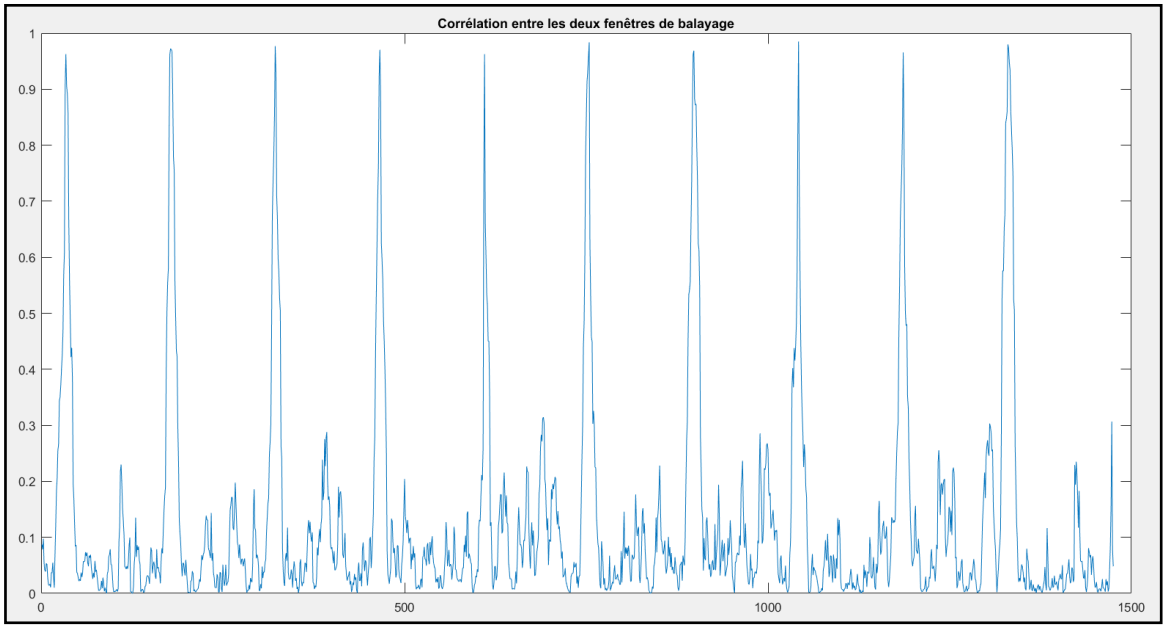

Figure 3 - Graphe de correspondance des fenêtres regardées pour un rapport signal à bruit : SNR = 19

• Ensuite nous allons devoir détecter la position des préfixes au sein de la corrélation obtenue. Pour cela, nous allons découper le signal en fenêtres de taille $n + l = 144$, ce qui correspond à la taille d'une trame comportant un préfixe cyclique. Ensuite, nous sauvegardons dans un tableau l'indice du maximum de chaque fenêtre. Ces indices correspondent aux positions du premier symbole des préfixes cycliques, et donc des trames.

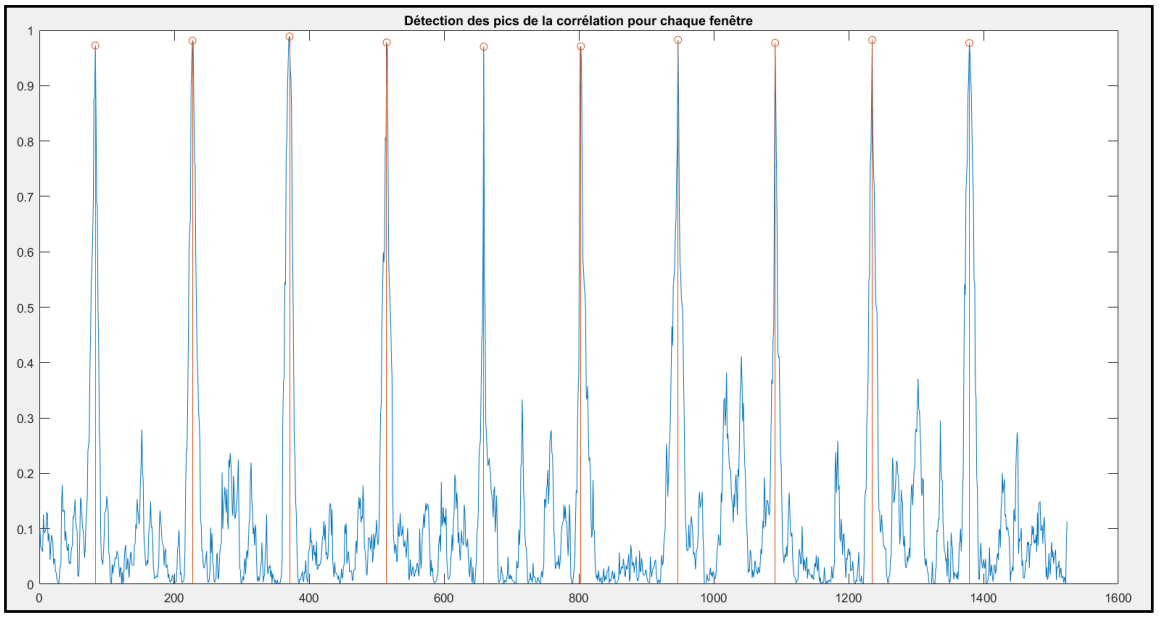

Figure 4 – Positions des débuts des N = 10 trames

c) Correction de l'offset

Nous sommes maintenant capables de délimiter les différentes trames du signal. Cependant, on s'aperçoit vite que la fonction précédente a des limites, notamment lorsque le bruit augmente. Il est possible que la corrélation ne soit pas totalement fiable. Par exemple, il se trouve que le maximum d'une fenêtre de n symboles ne correspond pas toujours à l'indice de début d'un préfixe cyclique. Celui-ci est parfois légèrement décalé. Cela peut être dû au fait que le signal est aléatoire ou bien à la présence de bruit dans le canal. Voici l'exemple, en *figure 3*, d'une corrélation partiellement fausse.

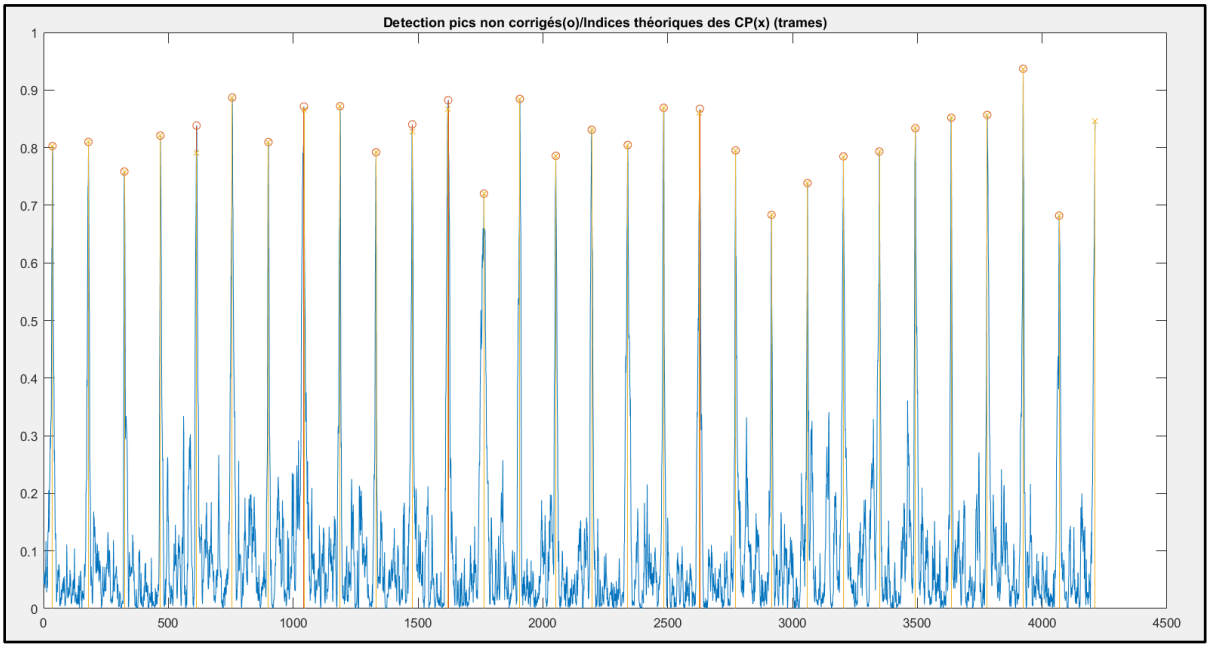

Figure 5 - Graphe de la corrélation avec comparaison des pics détectés et théoriques

Nous avons donc besoin de corriger ce phénomène pour connaître précisément les positions de début des trames. De plus, il est possible que le canal de propagation introduise un décalage temporel. En effet, il est très peu probable que le signal reçu commence exactement au début d'une trame. Sachant qu'une trame incomplète est inutilisable, nous devons réussir à déterminer l'intervalle entre le premier symbole reçu et le premier préfixe cyclique, et donc la première trame complète. Le calcul de cet offset va donc être essentiel à la chaîne de décodage. En effet, si nous arrivons à connaître le décalage initial, tenant compte des paramètres de codage du signal, nous pourrons retrouver les indices de début des trames.

Pour contrer le décalage temporel, nous procédons en trois étapes :

- Tout d'abord, dans la fonction « *calc_offset()* », nous stockons dans un tableau les valeurs des offsets pratiques obtenus en soustrayant aux indices obtenus en pratique les indices de début de trame pour un offset nul (soient les indices tels qu'ils sont dans le signal avant émission) :

$$offset(i) = indice\_pratique(i) - indice\_théorique(i)$$

« *indice_pratique(i)* » est l'indice de commencement de la i^{ème} trame calculé et « *indice_théorique(i)* » est l'indice théorique de la i^{ème} trame.

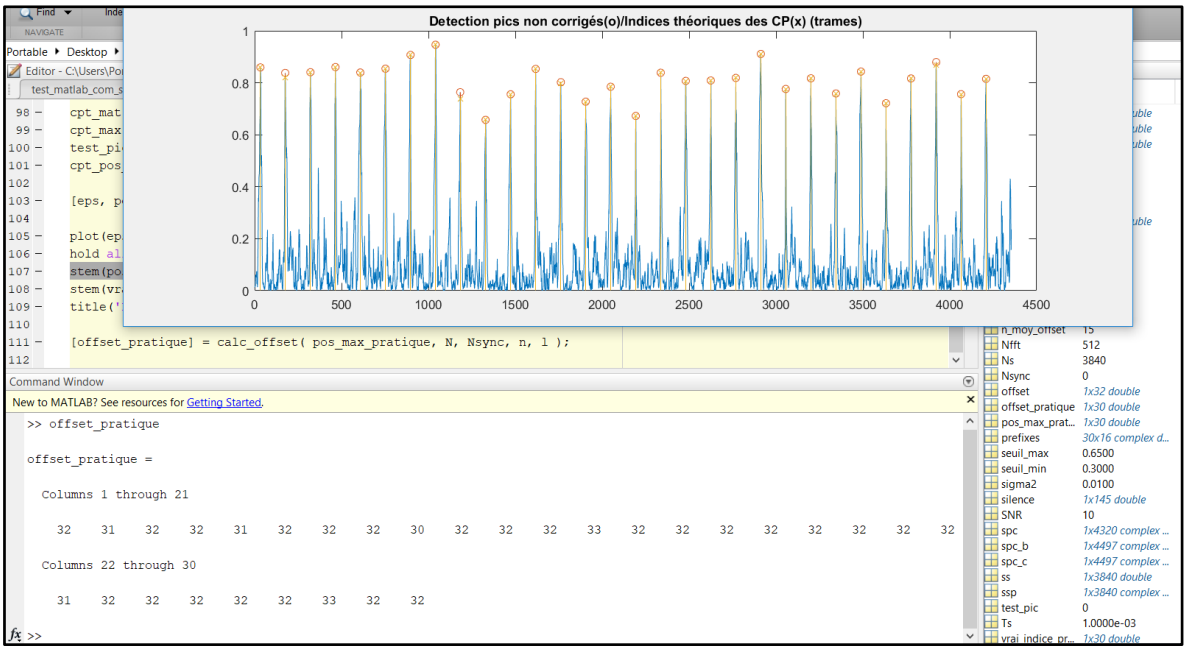

Figure 6 - Simulation du calcul de la valeur des offsets

- Le signal de réception peut s'écrire, en théorie :

$$Rn = \sum Hn * Tn + bn + offset$$

avec Rn les symboles reçus, Hn les coefficients de canal, Tn les symboles transmis et bn le bruit. Cependant, bn est un bruit blanc gaussien, cela signifie que son espérance est nulle. Le tableau d'offset retourné par la fonction précédente contient en fait les termes $offset + bn$. Ainsi en moyennant les valeurs de ce tableau, nous devrions obtenir la valeur réelle de l'offset :

$$< \sum offset + bn > = < \sum offset > + < \sum bn > = < \sum offset > = offset$$

Enfin, le canal de propagation peut évoluer dans le temps et ainsi modifier continuellement la valeur de l'offset. Pour être indifférent à ce phénomène, il ne faut pas moyenner la valeur de l'offset sur toutes les valeurs du tableau mais seulement sur les valeurs de quelques états précédents. Ainsi, la moyenne de l'offset évoluera beaucoup plus rapidement, suivant l'évolution du canal.

Ce processus est réalisé par la fonction « *moyenne_offset()* ». On aperçoit sur la simulation en *figure 5* que la moyenne corrige bien la valeur de l'offset, malgré que la corrélation soit parfois fausse.

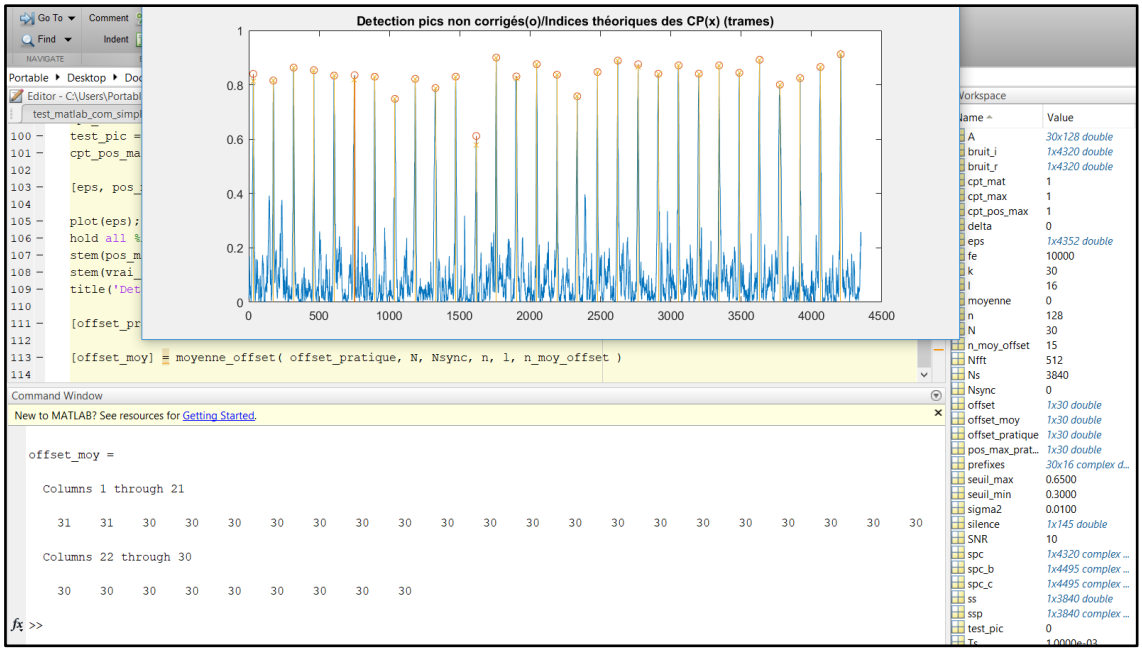

Figure 7 - Simulation du moyennage des offsets

• Enfin, la fonction « *correct_offset()* » calcule la position des symboles en début de trame par rapport aux offsets calculés et moyennés :

$$position_{corrig\acute{e}e}(i) = offset_{moyenn\acute{e}}(i) + (i-1) * (l+n) + 1$$

avec i la trame en cours de traitement, l la taille du préfixe cyclique et n le nombre de sous-porteuses.

On observe en simulant cette fonction que, parfois, l'algorithme met un certain temps pour obtenir le bon offset. Cependant, la correction fonctionne après un certain délai.

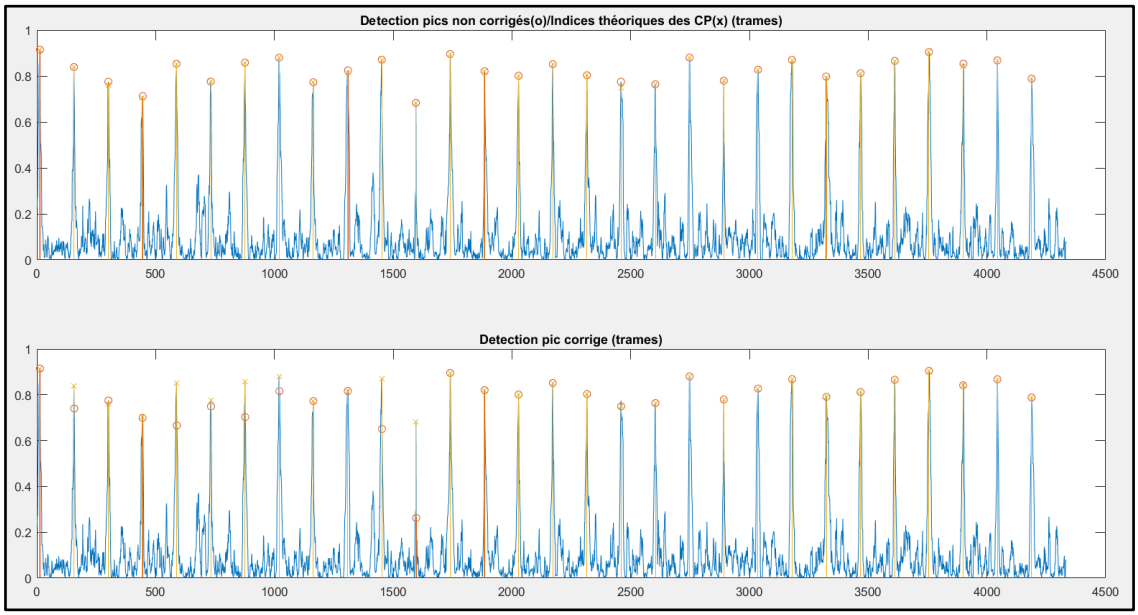

Figure 8 - Détection des pics corrigée après moyennage de l'offset

d) Série > Parallèle

Pour pouvoir exécuter plus facilement la fonction suivante, il est préférable de l'aborder avec un signal sous forme de matrice (dont les colonnes représentent les sous-porteuses) plutôt qu'avec un vecteur. Ainsi, nous procédons à une transformation série -> parallèle.

e) Suppression des préfixes cycliques

Une fois que nous avons détecté les préfixes cycliques et la position de leur premier symbole dans le signal, nous sommes capables de délimiter les trames et pouvons donc supprimer les préfixes cycliques, afin de retrouver le signal émis.

Nous créons la fonction « *supp_CP()* » qui extrait de chaque trame les $l = 16$ premiers symboles.

f) Détection des trames pilotes

Nous nous retrouvons à présent avec un signal sans préfixe cyclique, comportant des trames pilotes et des trames d'information. Notre but va être de détecter puis d'extraire les trames de synchronisation afin, d'une part, de retrouver notre signal initial, et d'autre part d'utiliser les trames de synchronisation pour déterminer les coefficients de canal introduits lors de la propagation du signal.

Nous procédons de façon similaire au traitement des préfixes cycliques vu précédemment. D'abord, nous allons établir une corrélation entre le signal PRBS connu et les différentes trames du signal. Pour ce faire, nous calculons le rapport entre le carré du produit scalaire de la trame PRBS et de la trame à traiter et le produit des normes de ces deux trames.

$$c_{PRBS} = \frac{| < PRBS,\ trame_{etudi\ ée} > |}{\left||PRBS|\right|.\left||trame_{étudi\ ée}|\right|}$$

Comme vu précédemment, ce calcul découle de l'inégalité de Cauchy-Schwarz. On obtient des coefficients de corrélation. Plus les coefficients sont élevés, plus la trame étudiée se rapproche de la trame pilote. Les pics sur le graphe représentant ce rapport représenteront donc les débuts des trames de synchronisation détectés. Ainsi, on obtient les positions des trames pilotes parmi les trames d'information.

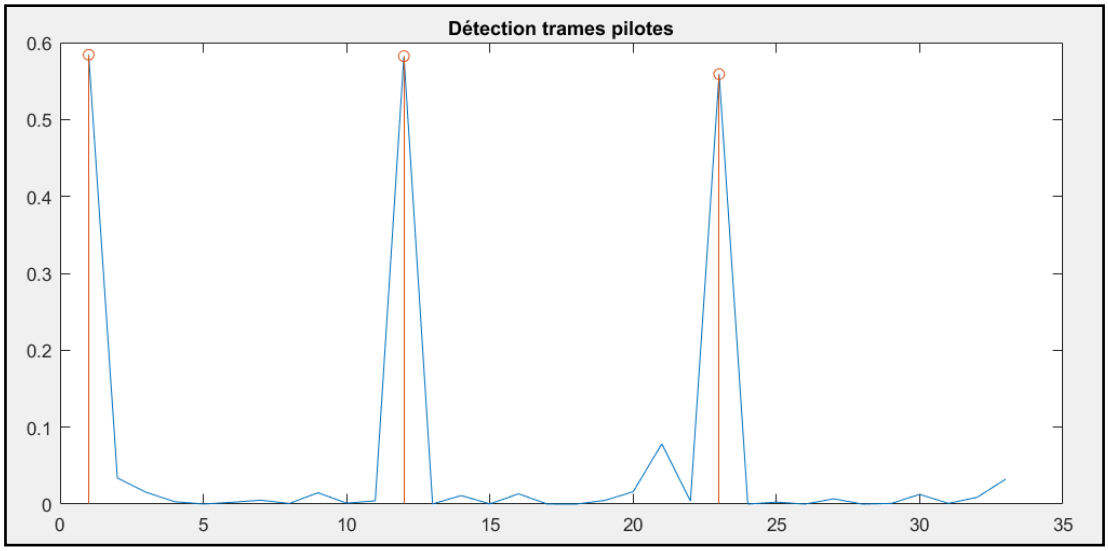

Figure 9 - Graphe représentant la corrélation avec détection des pics

g) Prélèvement des trames pilotes

Une fois l'emplacement des trames de synchronisation détecté, nous les extrayons du signal reçu. Pour cela, nous utilisons la fonction « *prelevmt_pilote()* ». Cette fonction retourne deux éléments : un tableau contenant toutes les trames pilotes détectées dans le signal et prélevées. Puis elle retourne le signal d'entrée sans les trames de synchronisation.

h) FFT

Une transformation de Fourier inverse a été réalisée dans la phase de construction du signal à émettre. Maintenant que nous avons enlevé les trames pilotes, nous allons pouvoir effectuer l'opération inverse : une transformation de Fourier. Nous utilisons pour cela la fonction « *fft* » (Fast Fourier Transform) de Matlab. Il ne faut pas oublier, encore une fois, à l'inverse de la chaine d'émission, de normaliser notre *fft* en divisant le résultat de la transformée par $\sqrt{n}$ avec n le nombre de sous-porteuses.

i) Parallèle > Série

Dans ce bloc, nous transformons le signal entrant, sous forme de matrice à n colonnes (avec n le nombre de sous-porteuses), en vecteur. En effet, le signal d'entrée dans la chaîne d'émission est un vecteur. De plus, il est plus facile d'appliquer la fonction de décision sur un vecteur.

j) Décision

Nous arrivons finalement à la phase de décision, dans laquelle le signal reçu sera assimilé, en fonction de son codage, à un signal binaire, prêt à être traité ou utilisé. Nous rappelons que nous utilisons la BPSK, à deux états donc, dans notre projet. Les états que peuvent prendre les symboles sont :

$$s_k = \pm 1$$

En absence de bruit, le signal se compose, à ce moment là de la chaîne, de symboles réels égaux à 1 ou -1. La décision n'est donc pas difficile :

- Si le symbole d'entrée vaut 1, le symbole de sortie sera aussi égal à 1
- Si le symbole d'entrée vaut -1, le symbole de sortie vaudra 0

Cependant, le canal de propagation introduit bien un bruit. Les valeurs sont donc plus ou moins modifiées en passant par le canal et les symboles après la fonction *fft* ne sont plus des réels mais des complexes dont les parties réelles s'approchent plus ou moins des valeurs recherchées. Voici pour différents rapports signal sur bruit la constellation des symboles à l'entrée de la fonction de décision :

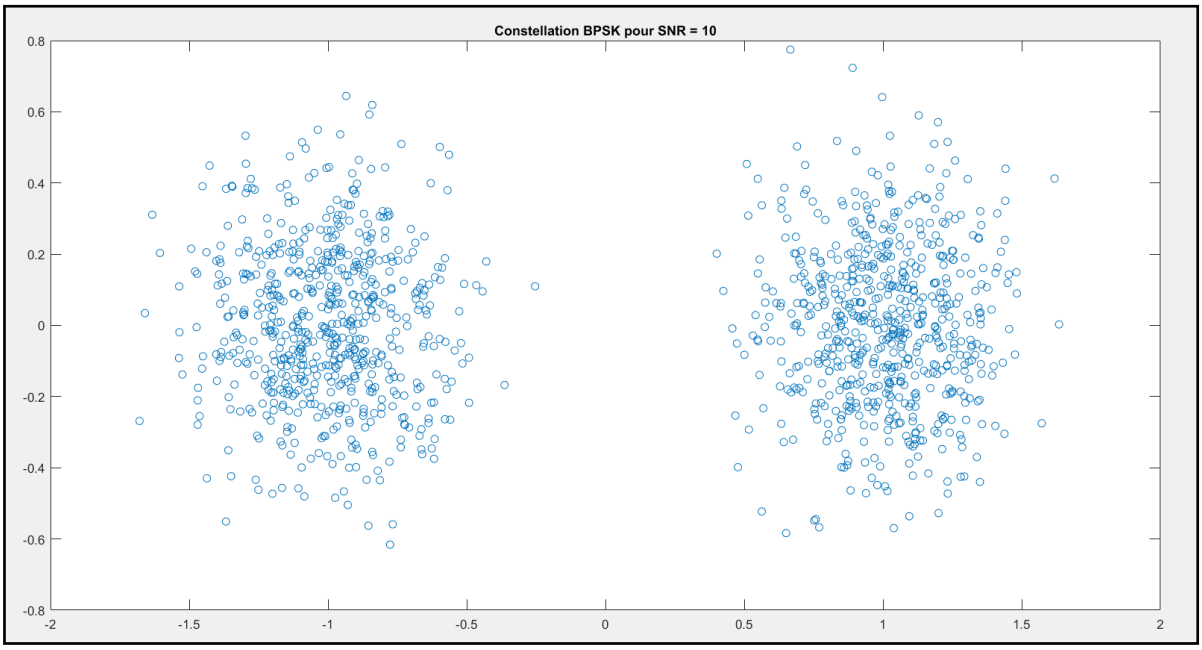

Figure 10 - Constellation des symboles BPSK du signal reçu pour SNR = 10

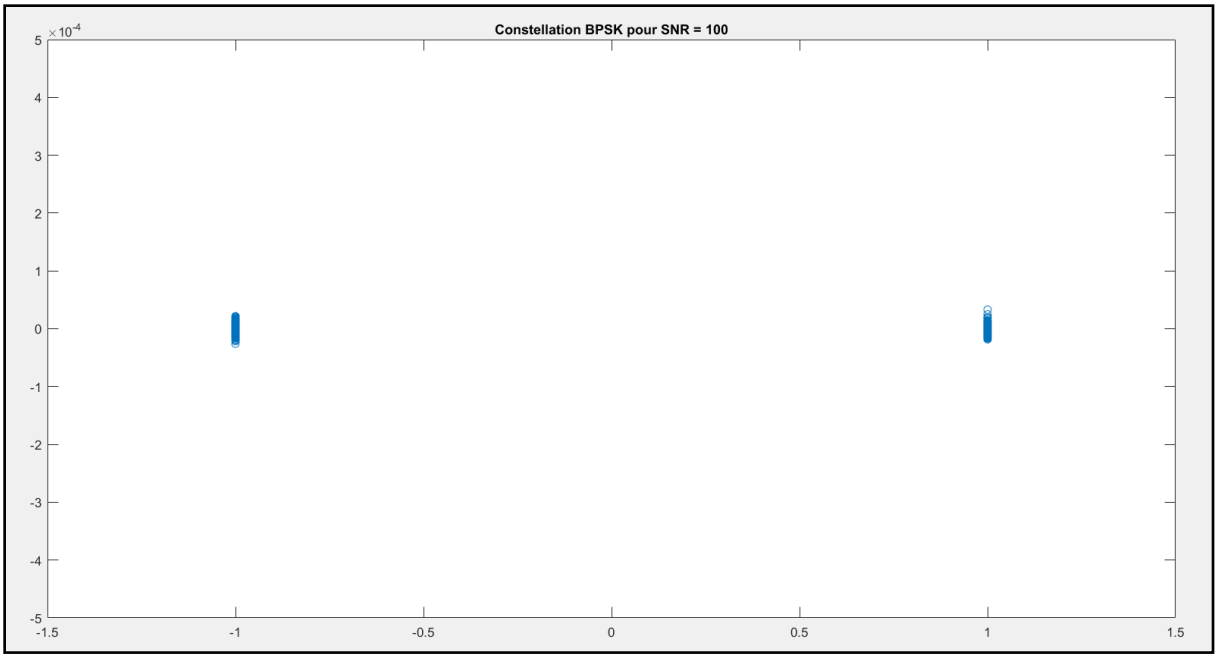

Figure 11 - Constellation des symboles BPSK du signal reçu pour SNR = 100

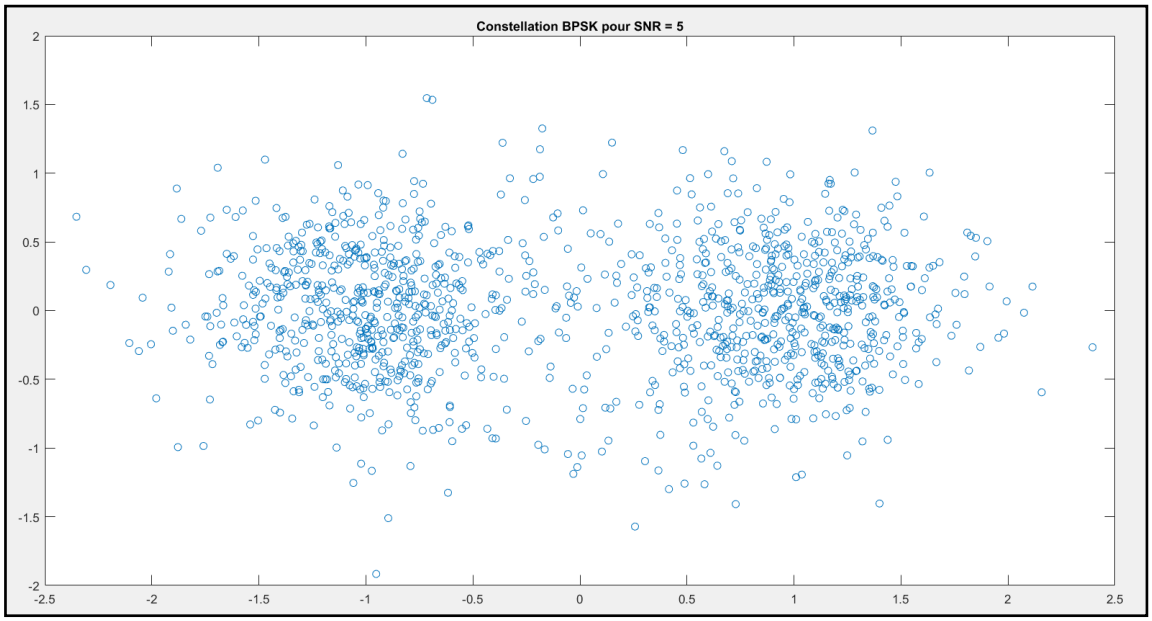

Figure 12 - Constellation des symboles BPSK du signal reçu pour SNR = 5

On observe bien que si le bruit est trop grand, il y a de plus fortes probabilités pour qu'un symbole ne soit pas dans la zone de l'espace qui lui correspond. Ainsi, il sera considéré comme un symbole opposé et son décodage sera faux. Donc plus le SNR sera grand, plus le TEB sera grand.

La fonction « *decision()* » que nous avons crée isole la partie réelle des symboles entrants et, s'il sont positifs, les assigne à 1. À l'inverse, s'ils sont strictement négatifs, ils seront assignés à 0.

V. Prise en main des radios logicielles

Pour simplifier l'utilisation des radios logicielles, nous travaillons sous Linux, sur machine virtuelle. Une fois la librairie *uhd* téléchargée, nous avons accès aux multiples fonctions pour agir sur les radios.

Tout d'abord, une fois la radio branchée à la machine, nous pouvons vérifier qu'elle est bien reconnue et prête à l'emploi grâce à la fonction « *uhd_find_devices* ».

Radio émission :

Pour procéder à l'émission de données par la radio, il nous faut donner à celle-ci un fichier à émettre. Cependant, ces données ne doivent pas avoir un type et une forme quelconques. Nous nous servons donc de la fonction « *fichier2radio* » qui va nous permettre de préparer le fichier à envoyer et de déterminer les paramètres de transmission.

Nous tentons d'émettre un fichier texte, pour commencer. Nous déclarons donc ce signal sur Matlab afin de le faire passer par la chaîne d'émission. Une fois les préfixes cycliques ajoutés, la ifft réalisée ainsi que les autres opérations, nous nous retrouvons avec un signal comportant des complexes doubles. Pour pouvoir être envoyé par la radio, il faut cependant qu'il soit sous forme binaire. Nous utilisons la fonction « *trad_matlab_radio* » qui nous a été fourni en début de projet. Cette fonction écrit donc en binaire dans un fichier, le contenu du signal que l'on veut transmettre.

Ceci fait, le fichier est envoyé grâce au script python « *fichier2radio* » en entrant en argument les paramètres nécessaires : fréquence d'échantillonage, fréquence porteuse, etc…

Radio réception :

Une fois le signal envoyé, il faut tenter, avec la deuxième radio, de le recevoir. Pour vérifier que la radio en réception reçoit bien un signal provenant de la première radio, on utilise la fonction « *uhd_fft* » en entrant les paramètres dans la ligne de commande. Ainsi on peut observer les spectres reçu et donc émis par l'autre radio.

Si le spectre correspond au spectre attendu, on peut sauvegarder dans un fichier binaire le contenu du signal reçu en appelant la fonction « *uhd_rx_cfile* ».

À présent, nous devons faire passer le fichier reçu dans la chaîne de décodage sur Matlab. Pour traduire le fichier binaire en complexe double, nous nous servons de la fonction « *trad_radio_matlab* ». Ensuite, le signal est traité par notre chaîne de réception et nous tenton enfin de récupérer les caractères composant le signal. Pour cela, nous regroupons les symboles par mots de huit bits et les traduisons en caractères selon le code ASCII.

VI. Conclusion

1. Bilan

Il nous a fallu un peu de temps pour bien démarrer notre projet. En effet, nous avons dû prendre connaissance du projet et faire un peu de documentation. Ceci fait, nous avons réussi à avancer progressivement tout au long du projet. Cependant, nous n'avons pas réussi à répondre totalement au cahier des charges. Nous avons traité le décalage temporel et atténué les effets du bruit dans le canal. Par contre, notre système de communication reste sensible au décalage fréquentiel et à la déformation liée au canal.

Pour cause, nous avons rencontré quelques problèmes.

2. Problèmes rencontrés

Tout d'abord, nous pensons avoir perdu beaucoup de temps pendant les premières séances. En effet, nous ne nous sommes pas réparti les tâches et avons travaillé pendant les deux tiers des séances à deux sur le même code. Cela était, au départ, nécessaire pour pouvoir comprendre et traduire les aspects théoriques des communications numériques. Mais nous aurions pu aller beaucoup plus loin dans notre projet et nous divisant le travail.

De plus, une autre cause de notre retard a été de ne pas réalisé de plan d'action ou de rétro-planning. En effet, réaliser ce planning nous aurait permis de savoir, à tout moment du projet, où nous en étions, et ensuite d'adapter notre travail en fonction du temps restant. Le résultat est que nous n'avons presque pas commencé à travailler en C++ car nous avons passé trop de temps sur le code Matlab.

Enfin, lors des dernières séances réalisées, nous n'arrivions plus à utiliser correctement les radios. Nous pensons que cela provient d'un faux-contact sur une des deux antennes. Cependant, nous n'avons pas réussi à détecter complètement d'où provenait l'erreur, et il reste possible qu'elle provienne de notre code (et notamment du fait que nous n'avons pas appliquer de normalisation sur le fichier à envoyer.

3. Prévisions pour la suite

Nous n'avons pas eu le temps de contrer le décalage fréquentiel. Cependant, nous pensions, pour cela, essayer de détecter le décalage en regardant l'argument du signal reçu. En effet, voici la forme du signal

$$y_n = s_n * e^{j\,2pi*delta*n}$$

Ainsi, en moyennant calculant la somme du produit entre les symboles d'une trame et les symboles d'une autre trame similaire :

$$\sum_{n=0}^{CP-1} y_{n+d} * y_{n+d+p} = \sum_{n=0}^{CP-1} |s_n|^2 * e^{-j\,2pi*delta*p}$$

Ensuite, il suffit de déduire de cette équation l'angle de décalage delta pour mettre la main sur le décalage fréquentiel.

Enfin, les coefficients du canal peuvent être déterminés grâce aux trames pilotes extraites. Cependant, nous n'arrivons pas de façon fiable à obtenir ces coefficients de canal.

4. Sentiment sur le projet

Nous avons été intéressés par ce projet et avons apprécié travailler dessus. En effet, ce projet est très complet, et permet de développer son standard de communication de A à Z. Nous aurions aimé avoir le temps d'implanter notre chaîne de communication en C++ pour pouvoir la tester en temps réel.

Ce projet nous a appris à travailler en groupe sur un sujet assez important, et nous a surtout montrer l'importance d'un rétro-planning et de la gestion du temps en fonction de l'avancement.