

SIT213

IMT Atlantique



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

7 OCTOBRE

Créé par :
LE DUC Elouan
MAQUIN Philippe
LE GRUIEC Clément
LE JEUNE Matthieu
FRAIGNAC Guillaume

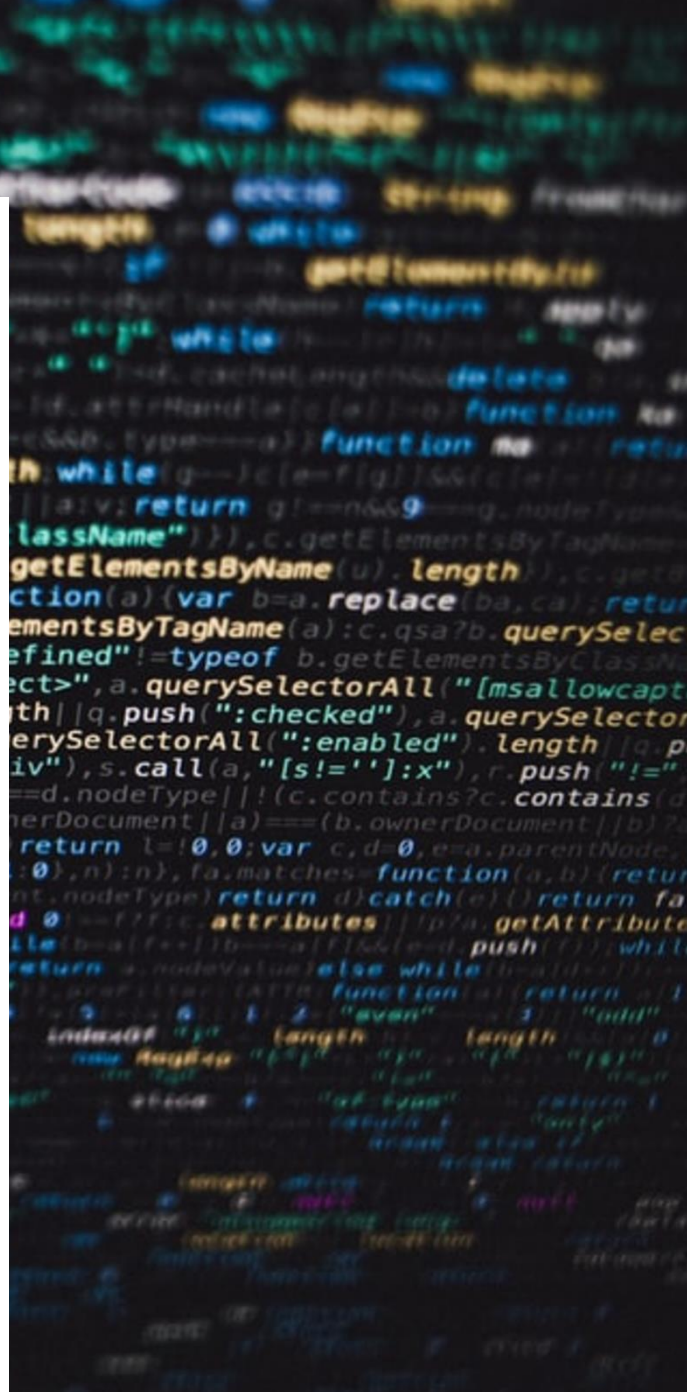


Table des matières

Intentions du projet	5
1 ^{ère} étape du projet	6
➤ Objectifs.....	6
➤ Analyse des actions à mener	7
➤ Programmation et connexion des blocs.....	8
✓ Mise en place des liaisons entre les blocs	8
✓ Calcul du TEB.....	9
➤ Réalisation des tests	9
➤ Tests et validations du programme	9
✓ Vérification du TEB.....	9
✓ Vérification des signaux entrée / sortie	10
Capitalisation du travail réalisé	12
Bilan de l'étape 1	12
2 ^{ème} étape du projet	13
➤ Objectifs.....	13
➤ Analyse des actions à mener	14
➤ Programmation	16
Emetteur	16
Récepteur.....	17
➤ Tests et validations du programme	18
✓ Résultats attendus	18
✓ Tests avec signal de type NRZ	19
✓ Tests avec signal de type NRZT	20
✓ Tests avec signal de type RZ.....	21
Bilan de l'étape 2	22

3^{ème} étape du projet	23
➤ Objectifs.....	23
➤ Analyse des actions à mener.....	24
➤ Programmation	25
➤ Tests et validations du programme	27
✓ Résultats attendus	27
✓ Tests avec signal de type NRZ	28
✓ Tests avec signal de type NRZT	29
✓ Tests avec signal de type RZ.....	30
✓ Tests avec signal de type NRZT et beaucoup de bruit	31
✓ Tests sur essais multiples.....	32
✓ Autotests	34
Bilan de l'étape 3	35
4^{ème} étape du projet	36
➤ Objectifs.....	36
➤ Analyse des actions à mener.....	37
➤ Amélioration des anciennes étapes.....	38
➤ Tests et validations du programme	38
✓ Tests avec signal de type RZ.....	39
✓ Tests avec signal de type NRZ	40
✓ Tests avec signal de type NRZT	41
✓ Résultats	42
Bilan de l'étape 4	42

5^{ème} étape du projet	43
➤ Objectifs.....	43
➤ Analyse des actions à mener	44
➤ Programmation :	44
➤ Tests et validations du programme	45
✓ Tests avec signal de type NRZ	45
✓ Tests avec signal de type NRZT	46
✓ Tests avec signal de type RZ.....	47
✓ Résultats des tests & amélioration du TEB	47
Bilan de l'étape 5	50
Table des illustrations	51

Intentions du projet

Il s'agit de réaliser, par équipe de 4 ou 5 élèves, une maquette logicielle (en Java) simulant un système de transmission numérique élémentaire. On intégrera donc dans la chaîne un bloc de modulation numérique.

Le système sera assemblé suivant une bibliothèque de modules comportant des ports d'entrée, des ports de sortie et des paramètres physiques. Ces derniers pourront être déterminés à partir des activités du module SIT 212. Le système global sera mis au point progressivement sur 5 séances au cours desquelles les modules seront raffinés, complétés, validés et connectés selon un schéma de transmission de type « point-à-point ».

Outre la qualité technique de la réalisation, on insistera sur les points suivants :

1. La qualité de documentation de la maquette logicielle (notamment la Javadoc).
2. Les efforts de validation des résultats de simulation produits par la maquette.
3. La maîtrise du processus de travail : gestion des versions successives de la maquette logicielle et du dossier technique afférent, synergie de l'équipe, démarche qualité. Concernant ce tout dernier critère, le respect des exigences de mise en forme du livrable sera primordial.

1^{ère} étape du projet

➤ Objectifs

Transmission élémentaire "back-to-back". On introduira le premier modèle de transmission schématisé sur la figure 1. Il vérifie les propriétés suivantes :

- La source émet une séquence booléenne soit fixée, soit aléatoire.
- Le transmetteur logique parfait se contente, à la réception d'un signal, de l'émettre tel quel vers les destinations qui lui sont connectées.
- La destination se contente de recevoir le signal du composant sur lequel elle est connectée.
- Des sondes logiques permettent de visualiser les signaux émis par la source et le transmetteur parfait.
- L'application principale calcule le taux d'erreur binaire (TEB) du système.

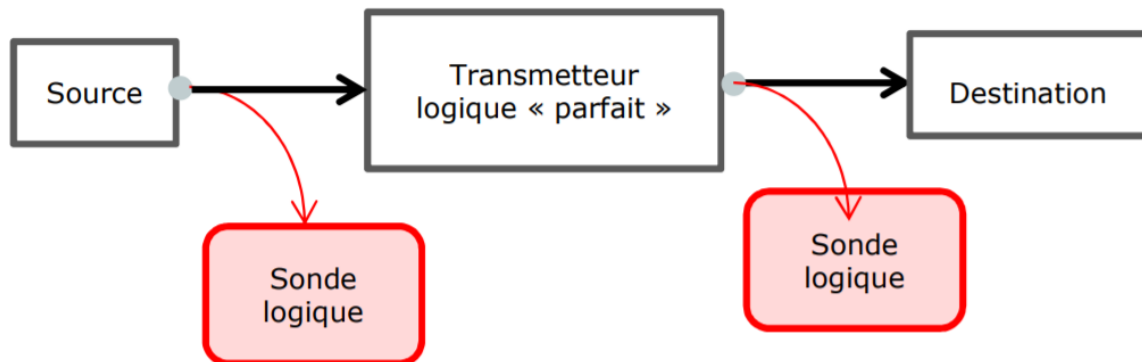


Figure 1 Modélisation de la chaîne de transmission à l'étape 1.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

L'option `-mess m` précise le message ou la longueur du message à émettre :

- Si `m` est une suite de 0 et de 1 de longueur au moins égale à 7, `m` est le message à émettre.
- Si `m` comporte au plus 6 chiffres décimaux et correspond à la représentation en base 10 d'un entier, cet entier est la longueur du message que le simulateur doit générer et transmettre.
- Par défaut le simulateur doit générer et transmettre un message de longueur 100.

L'option `-s` indique l'utilisation des sondes. Par défaut le simulateur n'utilise pas de sondes

➤ Analyse des actions à mener

Pour guider le développement de notre simulateur, un diagramme de classe (figure 2) nous a été fourni. Cela permet d'avoir une vue globale sur la structure à mettre en œuvre ainsi que sur les parties spécifiques à développer dans le cadre de l'étape 1 du projet.

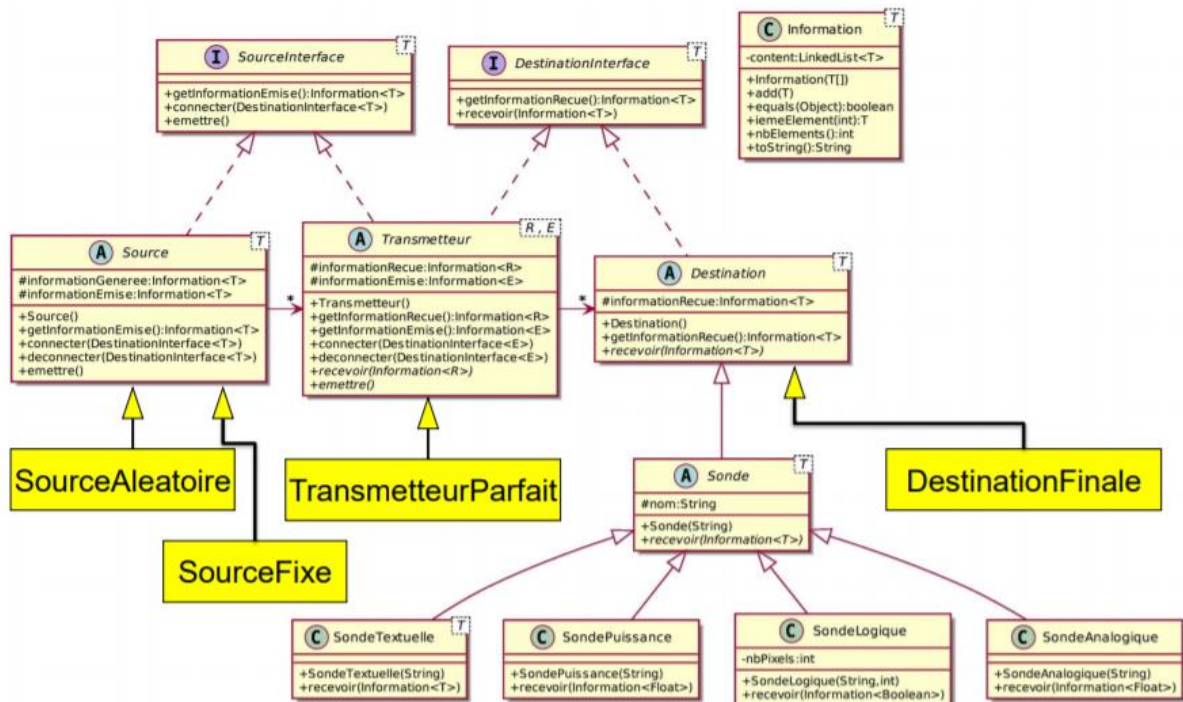


Figure 2 Diagramme de Classe du projet simulateur

De manière générale chaque classe correspond à une fonctionnalité spécifique de notre chaîne de transmission. Des classes mères abstraites et des interfaces ont déjà été développées au préalable par l'équipe enseignante. Nous pouvons ainsi avoir une base commune sur le projet. Les sous-blocs restent à développer en prenant en compte l'existant et les objectifs.

D'après le diagramme de classe nous avons 4 blocs à implémenter, cela correspond aux éléments en jaune vif sur la figure 2. Pour nous donner des indications sur les méthodes à écrire et sur les interactions entre les différents éléments du programme nous disposons également d'un diagramme de séquence. Les nouveaux blocs seront orchestrés depuis la classe *Simulateur*, le « main ».

Les conditions et indications décrites dans la cible peuvent s'apparenter à un cahier des charges. Ceux sont des éléments à prendre impérativement en compte dans le développement de l'ensemble des fonctionnalités.

➤ Programmation et connexion des blocs

Dans cette partie nous allons parler brièvement des éléments qui ont été programmés. Nous nous concentrerons sur la classe *Simulateur* afin de ne pas alourdir le compte-rendu. Les codes sources des classes développées sont disponibles dans l'archive fournit avec ce document. Une JavaDocs est également disponible afin d'aider à la compréhension des méthodes implémentées.

✓ Mise en place des liaisons entre les blocs

Le constructeur de la classe *Simulateur* est programmé selon les indications que nous avons eues. C'est-à-dire que l'ensemble des blocs de la chaine de transmission doivent être instanciés et connectés ensemble. Des « sondes » permettent de pouvoir visualiser graphiquement l'entrée ou la sortie d'un bloc. C'est une fonctionnalité qui sera très utile pour les tests à réaliser.

Le code permettant de mettre en œuvre la chaine selon le schéma de la figure 1.

```
public Simulateur(String[] args) throws ArgumentsException {

    //Analyse des arguments
    analyseArguments(args);

    //Instanciations des differents blocs de traitement
    if (messageAleatoire) {
        source=new SourceAleatoire(nbBitsMess);
    } else {
        source=new SourceFixe(messageString);
    }

    transmetteurLogique = new TransmetteurParfait();
    destination = new DestinationFinale();

    //Instanciations des differentes sondes
    SondeLogique viewSrc = new SondeLogique("ViewSrc", 720);
    SondeLogique viewTransmit = new SondeLogique("ViewTransmit", 720);

    //connexion des blocs ensembles
    source.connector(transmetteurLogique);
    if(affichage) source.connector(viewSrc);
    transmetteurLogique.connector(destination);
    if(affichage) transmetteurLogique.connector(viewTransmit);

}
```


✓ Calcul du TEB

Le programme doit calculer le TEB (Taux d'Erreur Binaire) du système. Pour calculer ce taux nous utilisons la formule suivante :

$$TEB = \frac{\text{Nombre de bits en erreur recue}}{\text{Nombre de bits total emis}}$$

Pour l'étape 1 ce TEB est optionnel puisque nous avons affaire à un transmetteur parfait. Si tout à bien été fait nous devons avoir un résultat final à 0.0 quel que soit le signal en entrée du système.

En Java le TEB est implémenté de la manière suivante :

```
public float calculTauxErreurBinaire() {  
  
    int nbErr=0;  
    float TEB=0.0f;  
    for (int i = 0; i < destination.getInformationRecue().nbElements(); i++) {  
        if(destination.getInformationRecue().iemeElement(i)  
            !=source.getInformationEmise().iemeElement(i)) nbErr++;  
    }  
    TEB=(nbErr*1.0f)/(source.getInformationEmise().nbElements());  
  
    return TEB;  
}
```

➤ Tests et validations du programme

✓ Vérification du TEB

Dans le cas d'une transmission parfaite le TEB attendu pour tout signal est à 0. Ce test est validé et fonctionne peu importe le signal, qu'il ait été généré de manière aléatoire ou qu'il ait été fixé.

```
java Simulateur -mess 10 -s => TEB : 0.0
```

```
java Simulateur -s => TEB : 0.0
```

```
java Simulateur -s -mess 10101010 => TEB : 0.0
```

Par acquis de conscience j'introduis volontairement 33 erreurs dans le signal par défaut afin de vérifier le calcul. Je m'attends alors à voir un TEB à 0.33. Et c'est bien le cas :

```
java Simulateur -s => TEB : 0.33
```

✓ Vérification des signaux entrée / sortie

Cas avec un signal par défaut (longueur 100, aléatoire) :

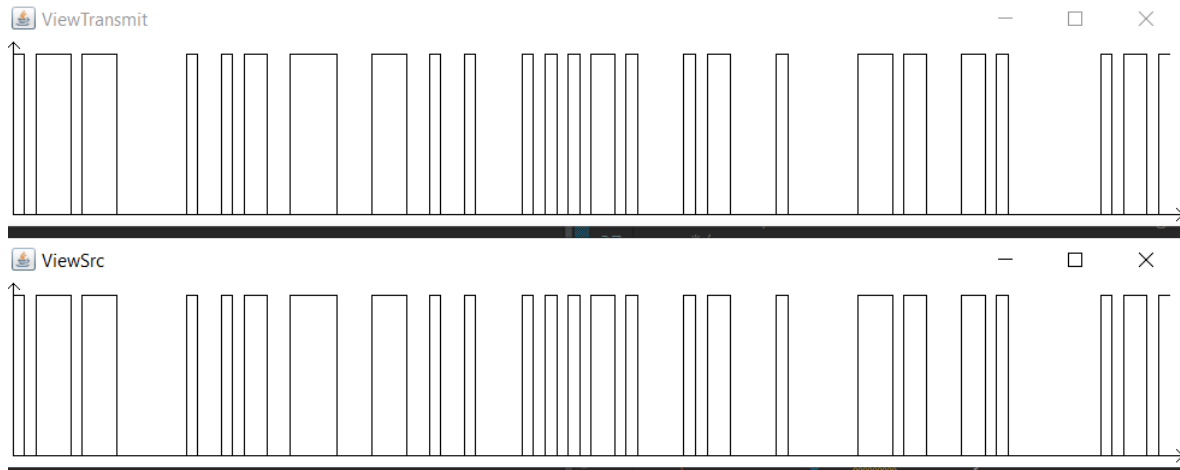


Figure 3 Signal aléatoire par défaut 1

Comme on pouvait s'y attendre le signal est identique en entrée et en sortie. Le TEB à 0 le confirme.

Je réalise un second lancer afin de vérifier que le signal est bien aléatoire :

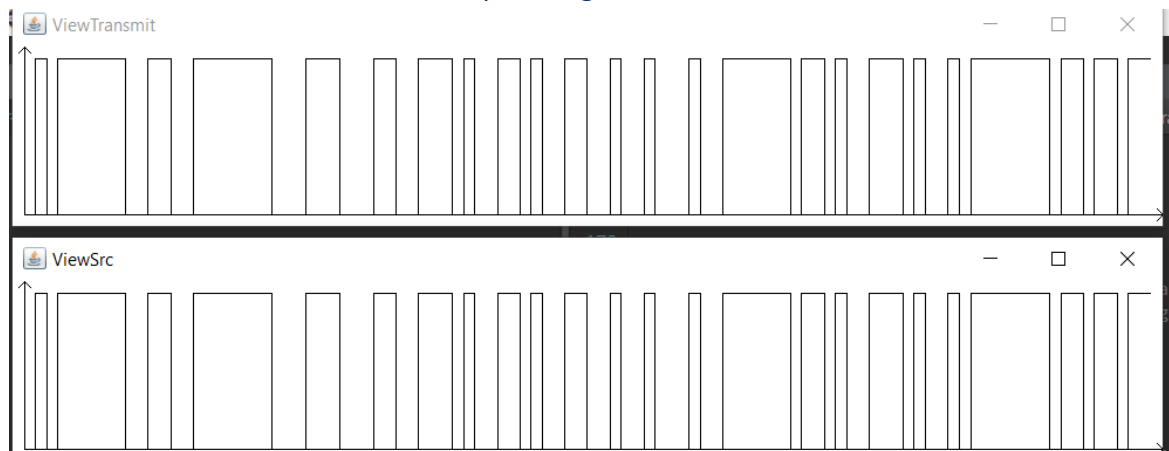


Figure 4 Signal aléatoire par défaut 2

Les 2 signaux sont une nouvelle fois identique (TEB à 0), de plus il y a bien un message différent entre le lancer 1 et le lancer 2. Ces tests sont donc concluants.

Cas avec un signal aléatoire de longueur fixée :

Je lance le Simulateur en indiquant de générer un signal de longueur 10 :

```
java | Simulateur -s -mess 10 => TEB : 0.0
```

Le message généré doit donc faire une longueur de 10 bits.

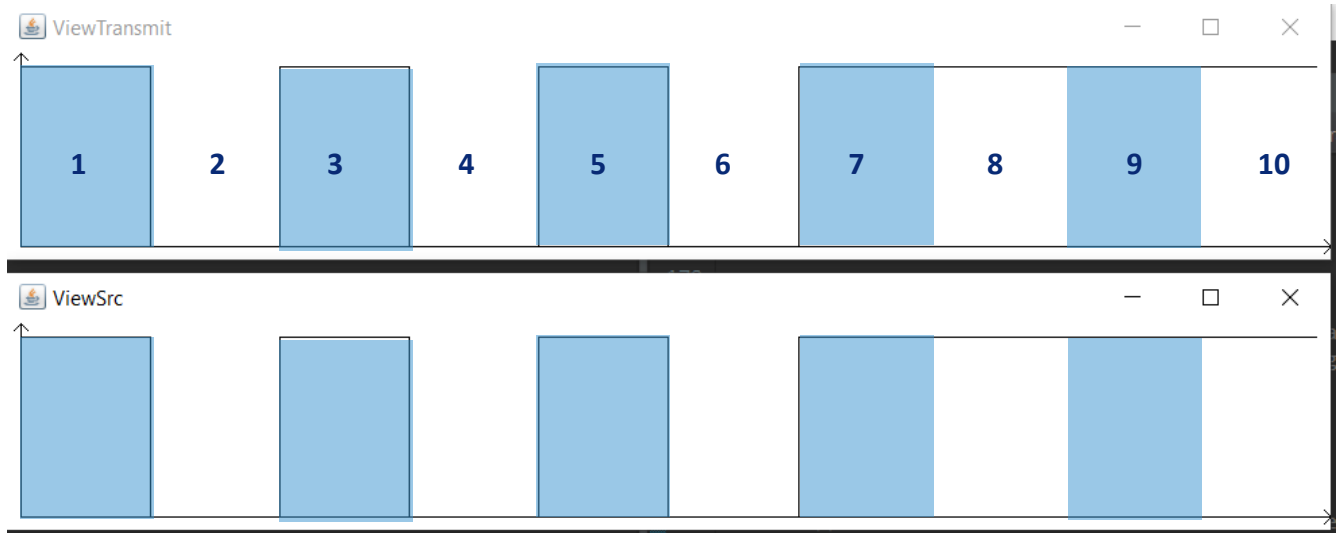


Figure 5 signal aléatoire de longueur fixée à 10

Le résultat obtenu est bien celui attendu. Le test est concluant.

Cas avec un signal fixé :

Je décide de mettre en paramètre le message fixe suivant : 10101010

```
java Simulateur -s -mess 10101010 => TEB : 0.0
```

Le message transmis doit donc correspondre à ce message, une alternation de niveaux logiques haut et bas graphiquement.

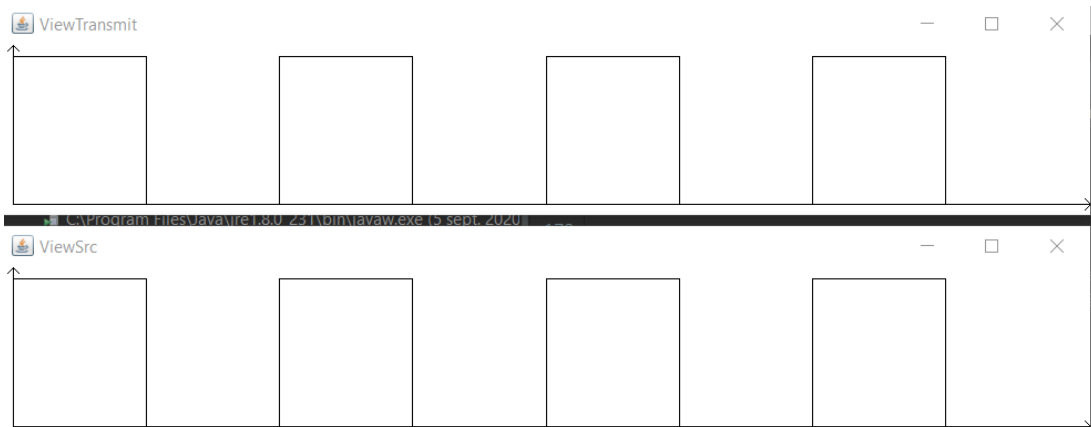


Figure 6 signal fixé sur 01010101

Le résultat obtenu est bien celui attendu. Le test est concluant.

Capitalisation du travail réalisé



C'est un projet qui est réalisé en groupe. Dans l'intérêt de chacun et pour disposer d'un système de versionnage nous utilisons les outils git et un dépôt GitHub.

Cette partie du programme a été déposée et versionnée dans un répertoire GitHub commun à tous les membres de nos groupes finaux. Ainsi nous pourrions mettre en commun nos aboutissements sur cette première étape.

Bilan de l'étape 1

Cette première étape nous a permis d'appréhender le projet en réalisant une chaîne de transmission simple dites « Back to Back ».

Au travers d'une analyse, puis en passant par le développement et pour finir sur une batterie de tests, nous sommes arrivés à mettre en place un système de communication. Il met en évidence les éléments importants d'une chaîne de transmission comme la source, le transmetteur et le récepteur (appelé destination ici).

Dans l'étape 2 nous devons implémenter un système de communication analogique non bruitée. L'approche sera certainement différente, il faudra de nouveau effectuer les étapes d'analyse avant de commencer à programmer les nouveaux blocs.

L'approche des télécommunications par la programmation est très intéressante. C'est une approche pratique qui permet de bien comprendre l'utilité et l'organisation des systèmes de communication. De plus c'est un projet pluridisciplinaire qui fait appel à diverses connaissances et qui permet bien sûr d'en développer des nouvelles.

2^{ème} étape du projet

➤ Objectifs

Dans cette phase du projet nous allons travailler sur une transmission non bruitée d'un signal analogique. On prendra en compte la nature analogique du canal de transmission en faisant évoluer la chaîne de transmission par l'adjonction de deux étages (logique → analogique et analogique → logique), comme indiqué sur le schéma de la figure 1.

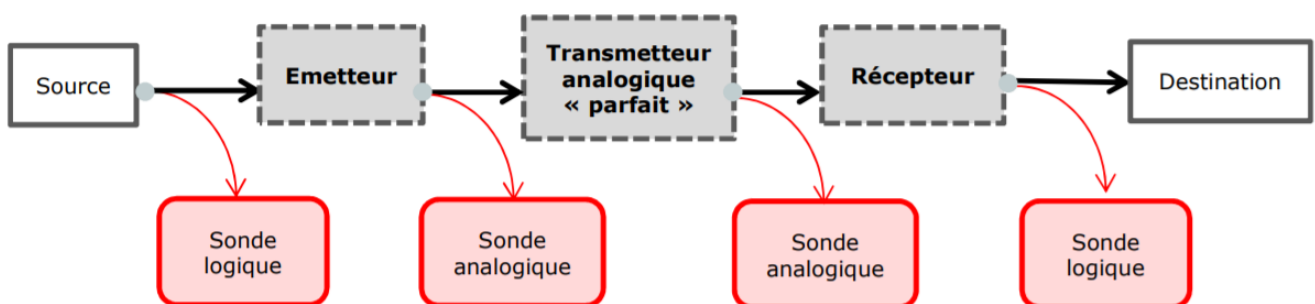


Figure 7 Modélisation de la chaîne de transmission à l'étape 2.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

L'option `-mess m` précise le message ou la longueur du message à émettre :

- Si m est une suite de 0 et de 1 de longueur au moins égale à 7, m est le message à émettre.
- Si m comporte au plus 6 chiffres décimaux et correspond à la représentation en base 10 d'un entier, cet entier est la longueur du message que le simulateur doit générer et transmettre.
- Par défaut le simulateur doit générer et transmettre un message de longueur 100.

L'option `-s` indique l'utilisation des sondes. Par défaut le simulateur n'utilise pas de sondes

L'option `-form f` précise la forme d'onde. Le paramètre f peut prendre les valeurs suivantes :

- NRZ : forme d'onde rectangulaire
- NRZT : forme d'onde trapézoïdale
- RZ : forme d'onde impulsionnelle

L'option `-nbEch ne` en transmission analogique, précise le nombre d'échantillons par bit.

L'option `-ampl min max` en transmission analogique précise l'amplitude min et max du signal

➤ Analyse des actions à mener

Nous allons devoir programmer un simulateur de signaux analogiques sur une machine. La nature du signal ne s'y prête pas. C'est pourquoi nous allons essayer de faire au mieux en suréchantillonnant des signaux logiques. Il faudra transformer une liste de Boolean en une liste de Float. Les True et les False prendront des valeurs différentes en fonctions de la forme et des amplitudes choisies. Cela correspondrait au schéma de la figure 2 ci-dessous.

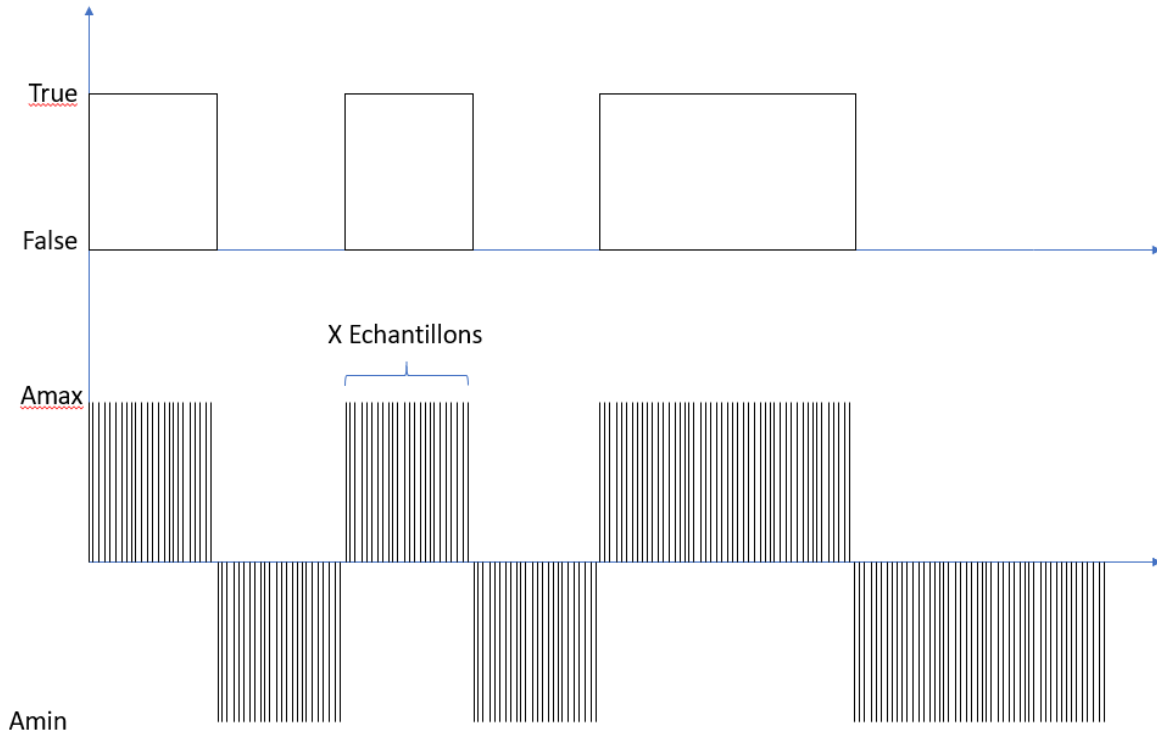


Figure 8 Schéma de transformation de Boolean en Float

Nous avons 3 formes possibles à prendre en compte :

- NRZ

C'est le type de forme la plus simple. Chaque bit correspond à une valeur, il suffit donc d'insérer dans l'information X valeurs correspondant au bit à convertir. Par exemple si on tombe sur un 1 (True), on mettra 30 Float de valeur 5.0 dans l'information, et pour le cas d'un 0, 30 Float de valeur -5.0. Cela ne sera pas forcément très réaliste car nous aurons un signal analogique très carré en sortie. Cela correspond au schéma de la figure 1.

- NRZT

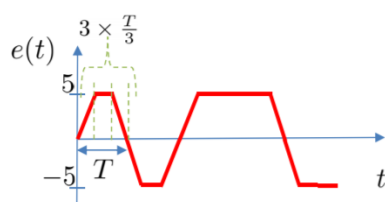


Figure 9 NRZT avec 10110

Une forme dérivée du NRZ et plus réaliste car on intègre une pente sur les fronts montants et descendants. Chaque symbole sera en tiers de période afin de prendre en compte la phase montante, stabilisée et descendante. Quand 2 symboles identiques se suivent il ne faudra pas réaliser l'ensemble des phases et garder le signal sur son seuil (Amax ou Amin).

- RZ

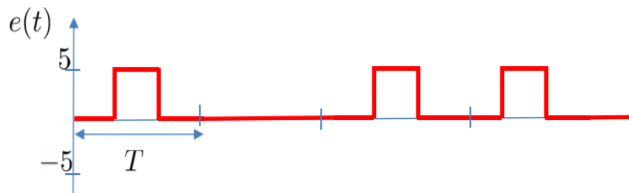


Figure 10 RZ avec 10110

C'est une forme qui est particulièrement lisible. Lorsque l'on a un 1 (True), elle maintiendra la valeur A_{max} sur un tiers de sa période seulement et reviendra à 0 sur les deux autres tiers. La valeur max se situe sur le tiers central. Pour un 0 (False), elle reste à 0. Avec cette forme, même si 2 bits de mêmes valeurs se suivent, les valeurs retourneront à 0 sur 2 tiers de période.

D'après le schéma de la figure 1 nous devons programmer 3 nouveaux blocs : un émetteur, un transmetteur analogique parfait et un récepteur. Les sources et destinations sont les mêmes qu'à l'étape une. Quand on y regarde de plus près, l'émetteur et le récepteur sont des transmetteurs qui encodent ou décodent nos informations. Ils ont tous deux, une entrée et une sortie, l'une pour des Float, l'autre pour des Boolean. Nous pourrions donc faire hériter ces 3 nouveaux blocs de la classe Transmetteur et bénéficier des fonctionnalités déjà programmées. Pour ce qui concerne les sondes, elles ont déjà été programmées par les enseignants.

Nous devons également programmer la gestion des paramètres du logiciel afin de pouvoir configurer notre simulateur facilement sans avoir à retourner dans le code source. Certaines fonctions avaient déjà été réalisées par les enseignants, d'autres restent à programmer.

Nous devons implémenter les paramètres suivants : *-form f*, *nbEch ne*, *-ampl min max*, *-seed v*.

Pour chacune de ces fonctionnalités nous devons détecter les paramètres du programme et s'assurer de leur(s) conformité(s) en utilisant des Regex par exemple.

La fonctionnalité *-seed* pour paramétrer une graine dans le cadre des générations aléatoires avait été omis dans l'étape 1. Nous l'avons donc ajouté.

➤ Programmation

Emetteur

La conversion de logique à analogique (CNA) est réalisée au cas par cas. En fonction des paramètres pris par le constructeur de la classe émetteur, la méthode *CNA()* applique un traitement spécifique sur le signal reçu. Il va traiter Boolean par Boolean reçue et remplir un nouveau tableau de Float qui sera émis. La transformation est caractérisée par une amplitude min et max, un nombre d'échantillon par bit et une forme (principalement).

Prenons le cas d'un signal NRZ avec le message 10 :

La méthode *CNA()* va procéder à la traduction bit par bit. Il va prendre le premier bit (1 ou True) et insérer NbEchantillonParBit de valeur Amax dans l'information à émettre. Il va ensuite traiter le bit 0 (ou False) et insérer NbEchantillonParBit de valeur Amin dans l'information à émettre.

Pour une forme RZ c'est très similaire, mise à part le fait que Amin soit tout le temps fixé à 0 et que chaque période est divisée en 3. 1/3 des échantillons à Amin, le 2^e tier à Amax et le dernier à Amin pour un 1. Tout le temps à Amin pour un 0.

Dans le cas d'un NRZT, comme pour un RZ, la période est scindée en 3. Il y a juste une pente au début et à la fin de chaque bit. Malheureusement ce n'est pas si simple... En effet les formes dépendent du bit (n-1) et du bit (n+1). Quand deux bits identiques se suivent il ne faut pas mettre de pente(s).

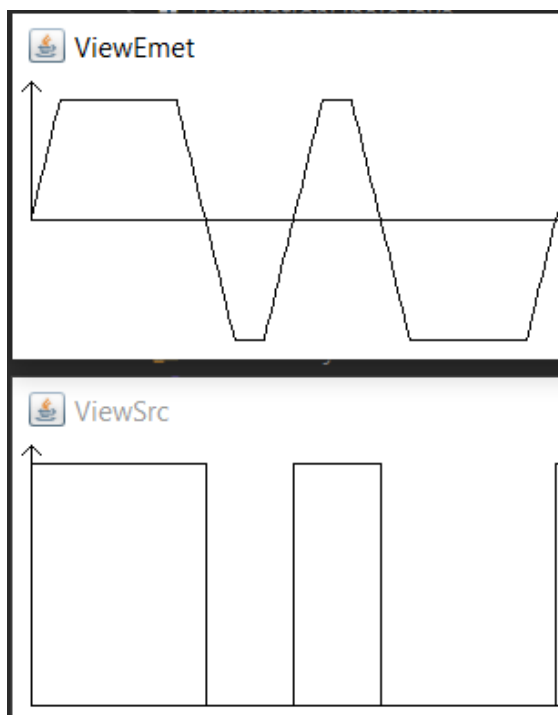


Figure 11 : Conversion en NRZT

Comme on peut le voir, pour deux bits à 1, la valeur Amax doit être maintenue sur les 2 derniers tiers de la période du premier bit, puis sur les 2 premiers tiers de la période du bits suivant. Quand il y a plus de deux bits de même signe à se suivre, les bits centraux sont transformés en valeurs Amax sur toute leur période.

Quand il y a un changement de signe, on ajoute les pentes en entrée et en sortie du bit.

Nous devons donc regarder pour chaque bit, les valeurs suivante et précédente.

Pour le 1^{er} et dernier bit, on considère des bits opposés.

Récepteur

En réception, pour la conversion analogique à logique (CAN), nous utilisons une méthode *Can(seuil)* qui marche presque à l'identique pour les 3 formes.

Les valeurs possibles à identifier sont des True (1) ou False (0). Nous ne cherchons donc que des 1 dans le signal et considérons les autres comme des 0.

Nous traitons des signaux analogiques et dans les futures étapes nous aurons du bruit. Nous ne pouvons donc pas identifier un bit à 1 par ses valeurs A_{max} uniquement. C'est pourquoi nous avons utilisé un seuil de détection.

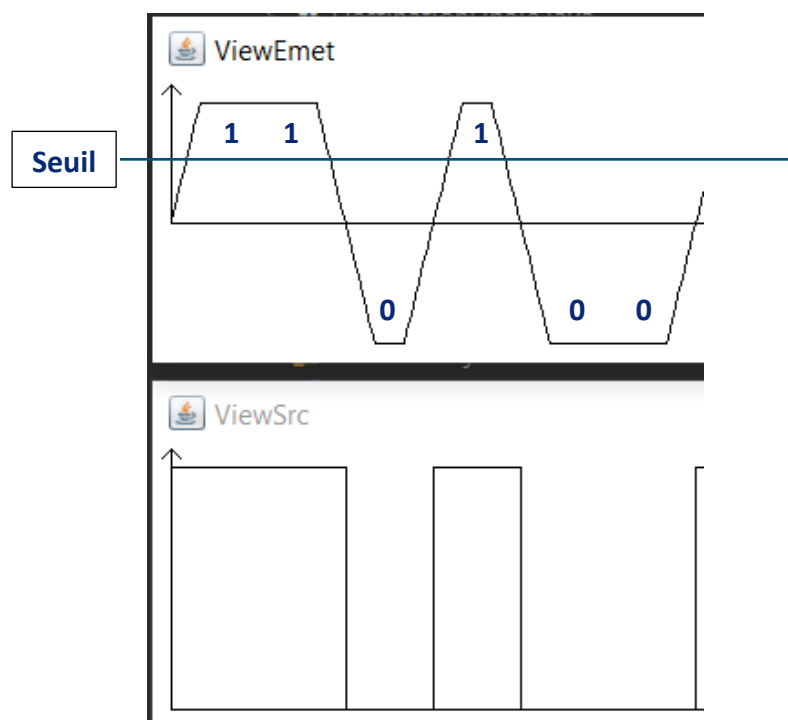


Figure 12 : Notion de seuil

Pour effectuer la conversion nous connaissons le nombre d'échantillons par bit, A_{max} , A_{min} et la forme. Le seuil est égal à $A_{max}/3$ par exemple pour NRZ et NRZT, et $A_{max}/9$ pour RZ. Le récepteur effectue une moyenne sur les valeurs analogiques par paquet de $NbEchantillonParBit$. En fonction de ce résultat, si la valeur est supérieure au seuil, le récepteur décide de mettre un 1, sinon un 0 dans les informations en sortie.

➤ Tests et validations du programme

À ce stade du projet nous n'avons pas encore automatisé les tests mais nous pouvons déjà en réaliser grâce aux données tel que le TEB et l'affichage des sondes.

✓ Résultats attendus

Nous sommes dans le cas d'une chaîne de transmission avec un transmetteur analogique parfait. Le signal émis doit être identique au signal reçu qu'il soit converti sous une forme NRZ, NRZT ou encore RZ.

Sur les sondes les signaux sources et destinations sont identiques, les signaux en sorti de l'émetteur et du transmetteur sont également identiques. Dans tous les cas le TEB doit être à 0 à la fin de la chaîne de transmission si tout se passe bien. Les formes d'ondes choisies doivent être utilisées selon leur spécifications.

Pour les tests nous utiliserons un signal avec les paramètres suivants :

- Amplitude max : 5V
- Amplitude min : -5V
- Nb échantillon : 60
- Seed : 40
- Longueur (mess) : 20

Le signal aléatoire généré sera égal à 1101 0011 0100 1001 1111

Ce message est intéressant, il permet de tester différents cas de figure, par exemple deux 1 qui se suivent, le passage d'un 0 à un 1 etc...

✓ Tests avec signal de type NRZ

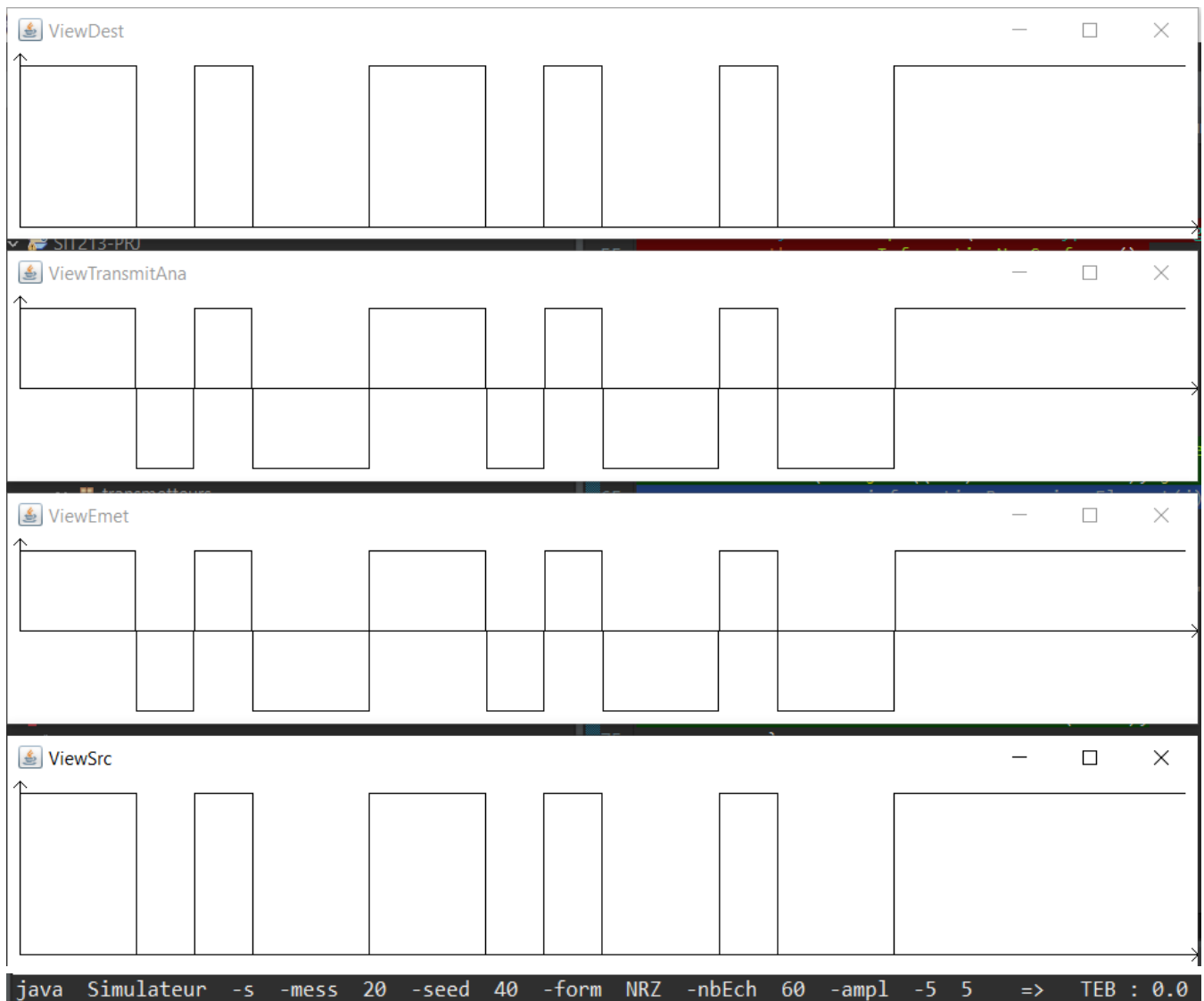


Figure 13 Tests du signal de type NRZ

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique s'est réalisée comme prévu, 5V pour un bit à 1 et -5v pour un bit à 0. Le test est validé.

✓ Tests avec signal de type NRZT



Figure 14 Tests du signal de type NRZT

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique s'est réalisée comme prévu, 5V pour un bit à 1 et -5v pour un bit à 0. Que cela soit un 0 ou un 1 qui est transmis, il y a des pentes. Lorsque deux symboles identiques se suivent, le niveau est maintenu, il n'y a pas de retour à l'amplitude min. Le test est validé.

✓ Tests avec signal de type RZ

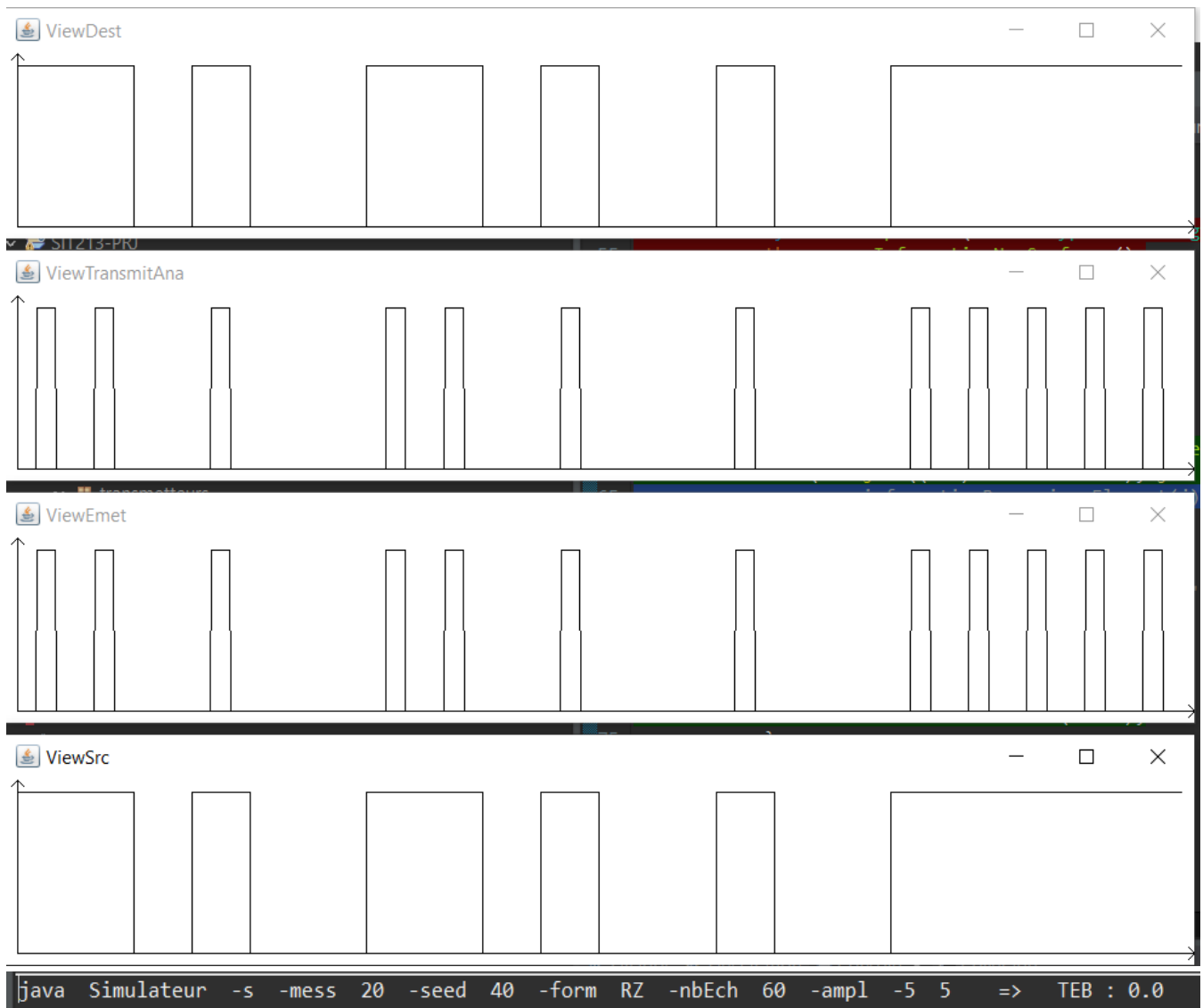


Figure 15 Tests du signal de type RZ

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. La conversion analogique a réagi comme prévu, 5V pour un bit à 1 et 0V pour un 0. Pour un 0 il n'y a pas de changement de niveau, pour un 1 nous avons bien 2 tiers du symbole à 0 et 1 tiers du symbole à 5V. Quand deux 1 se suivent il y a quand même un retour à 0. Le test est validé.

Bilan de l'étape 2

Nous avons ainsi pu simuler une conversion numérique à analogique et inversement, analogique à numérique. Les tests se sont passés comme prévus, les résultats sont donc satisfaisants.

Dans l'étape 3 nous devons implémenter une transmission non-idéale avec canal bruité de type « gaussien ». Il faudra programmer un ou plusieurs nouveaux blocs afin d'y arriver. Le travail réalisé lors de cette seconde étape sera une base importante pour la suite du projet.

3^{ème} étape du projet

➤ Objectifs

Dans cette phase du projet nous allons travailler sur une transmission non-idéale avec canal bruité de type « gaussien » d'un signal analogique. La propagation dans le canal est modélisée de manière théorique par un bruit blanc additif gaussien qu'il conviendra de régler en fonction des paramètres du transmetteur vus à l'étape précédente. On fera attention à surveiller le TEB en réception.

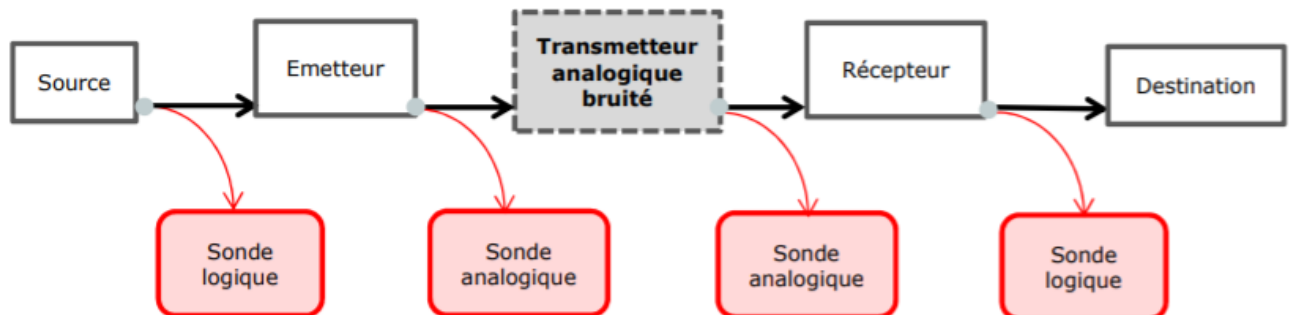


Figure 16 : Modélisation de la chaîne de transmission à l'étape 3.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

En plus des paramètres des deux premières étapes on va ici ajouter un bruit gaussien.

L'option **-snrpb s** en transmission analogique bruitée permet de donner la valeur du rapport signal sur bruit par bit (E_b/N_0 en dB). Nous ferons attention à utiliser un paramètre flottant. Par défaut le « SNRPB » est à 10000000 (bruit non visible).

➤ Analyse des actions à mener

Nous allons devoir ajouter un bruit blanc gaussien dans le simulateur de signaux analogiques que nous avons développé au cours de la seconde étape. Pour générer le bruit blanc gaussien on utilise la formule suivante :

$$b(n) = \sigma_b \sqrt{-2\ln(1 - a_1(n))} \cos(2\pi a_2(n)) \quad \begin{array}{l} a_1(n) \sim \mathcal{U}[0, 1[\text{ (loi uniforme)} \\ a_2(n) \sim \mathcal{U}[0, 1[\end{array}$$

Par la suite nous allons additionner le signal analogique créé à la précédente étape au bruit blanc gaussien. Comme l'illustre le schéma ci-dessous.

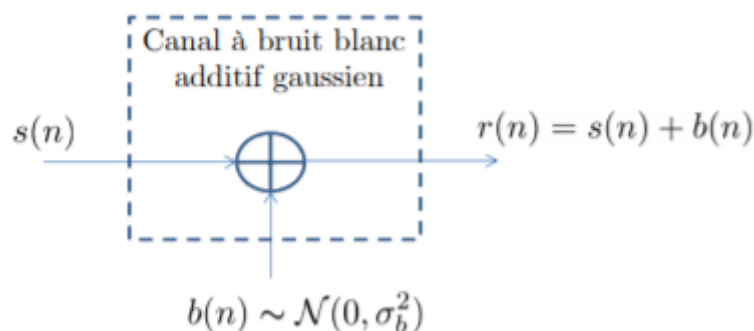


Figure 17 : Ajout du bruit blanc gaussien.

On obtient ainsi le signal $r(n)$ qui correspond au signal d'entrée additionné au bruit blanc. Il faudra vérifier que le bruit généré suit une loi gaussienne (histogramme).

Comme pour l'étape précédente nous retrouvons les 3 formes à prendre en compte, à savoir :

- NRZ
- NRZT
- RZ

D'après le schéma de la *figure 1* nous devons modifier notre Transmetteur analogique « parfait » en un transmetteur analogique « bruité gaussien ». Les blocs émetteur, récepteur, source et destination sont les mêmes qu'à la précédente étape. Pour ce qui concerne les sondes permettant de visualiser les signaux, elles ont déjà été programmées par les enseignants.

Nous devons implémenter le paramètre suivant : $-snrpb$ s. Nous devons nous assurer de détecter les paramètres du programme et d'assurer leur conformité en utilisant des Regex par exemple.

➤ Programmation

À la suite de l'étape 2 notre programme avait subi un test de performance. Il s'est avéré qu'il n'était pas aussi performant que souhaité (lenteurs).

Nous avons changé le maximum de boucles for classiques par des foreach.

```
for (int i = 0; i < informationRecue.nbElements(); i++) {  
    Boolean b=informationRecue.getIemeElement(i);  
    //code...  
}  
  
//changé en  
  
for (Boolean recu : informationRecue) {  
    Boolean b = recu;  
    //code  
}
```

Dans le code ci-dessus les boucles font la même chose, cependant dans la 2^e version est plus optimisée.

Une des plus grosses améliorations est le changement du type de liste utilisée par l'information. En effet dans le code d'origine fourni ce sont des LinkedList qui ne sont pas adaptées à l'utilisation du simulateur. Elles ont été remplacées par des ArrayList beaucoup plus rapide dans notre cas d'usage.

Opération	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1)
remove()	O(n)	O(1)

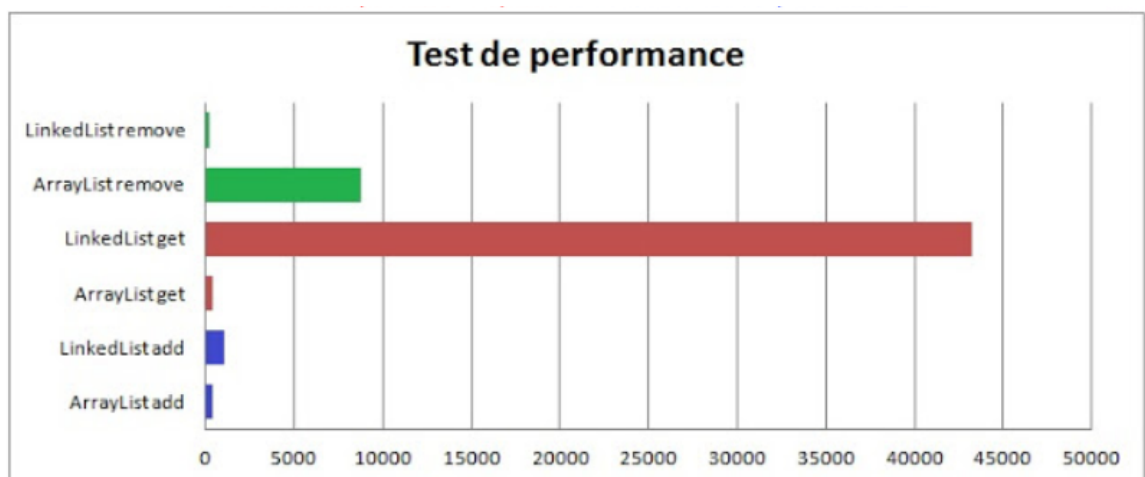


Figure 18 : source <http://www.codeurjava.com/> - ArrayList vs LinkedList (abscisse = temp en ms)

Concernant l'utilisation du programme, ceux sont toujours les mêmes méthodes, elles ont juste été légèrement modifiées pour correspondre aux nouvelles listes.

```
private ArrayList<T> content;

public Information() {
    this.content = new ArrayList<T>();
}

public Information(T[] content) {
    this.content = new ArrayList<T>();
    for (int i = 0; i < content.length; i++) {
        this.content.add(content[i]);
    }
}

public void add(T valeur) {
    this.content.add(valeur);
}
```

Grâce à cette optimisation il est désormais possible de simuler une chaîne de transmission avec 1 million de symboles en entrée (une dizaine de seconde avec affichage activé). Les sondes quant à elles mettent un peu de temps à afficher l'ensemble des symboles qui sont très nombreux. (4*1Million à afficher).

En plus de ces améliorations, d'autres méthodes ont été reprises afin de les simplifier et donc optimiser le code pour une finalité identique.

Ces améliorations participent à rendre notre programme meilleur, dans le cadre des simulations le but est d'avoir un résultat qualitatif le plus rapidement possible.

➤ Tests et validations du programme

A partir de la troisième étape nous avons commencé à automatiser nos tests. Ces différents tests ont pour but de vérifier si l'ensemble du programme est correctement installé sur une machine.

✓ Résultats attendus

Nous sommes dans le cas d'une chaîne de transmission avec un transmetteur analogique bruité gaussien. Le signal reçu doit être le plus fidèle possible au signal émis qu'il soit converti sous une forme NRZ, NRZT ou encore RZ.

Dans tous les cas le TEB doit être le plus proche de 0 à la fin de la chaîne de transmission si tout se passe bien. Les formes d'ondes choisies doivent être utilisées selon leurs spécifications.

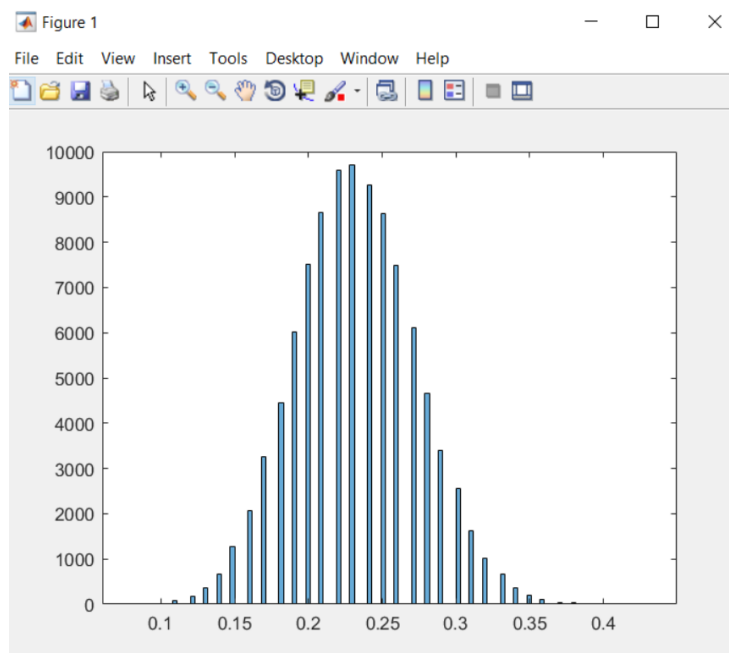
Pour les tests nous utiliserons un signal avec les paramètres suivants :

- Amplitude max : 5V ;
- Amplitude min : -5V ;
- Nb échantillon : 60 ;
- Seed : 40 ;
- Longueur (mess) : 20 ;
- Snrpb : 1 ;

Nous générons des histogrammes sous Matlab en réalisant 100 000 lancers avec le même message et sans utiliser la seed sur le bruit. Nous voulons vérifier si le bruit est bien gaussien.

Le signal aléatoire généré sera égal à 1101 0011 0100 1001 1111 Ce message est intéressant, il permet de tester différents cas de figure, par exemple deux 1 qui se suivent, le passage d'un 0 à un 1 etc... (On utilise le même signal que pour la seconde étape)

✓ Tests avec signal de type NRZ



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

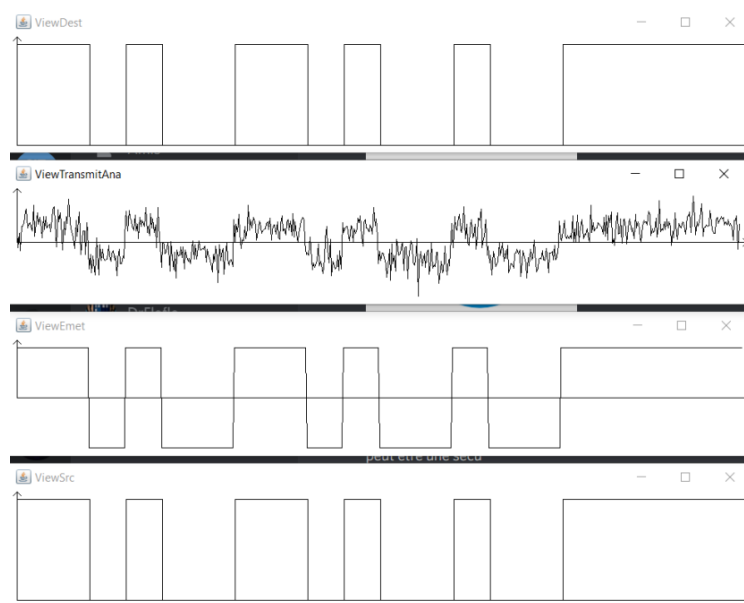
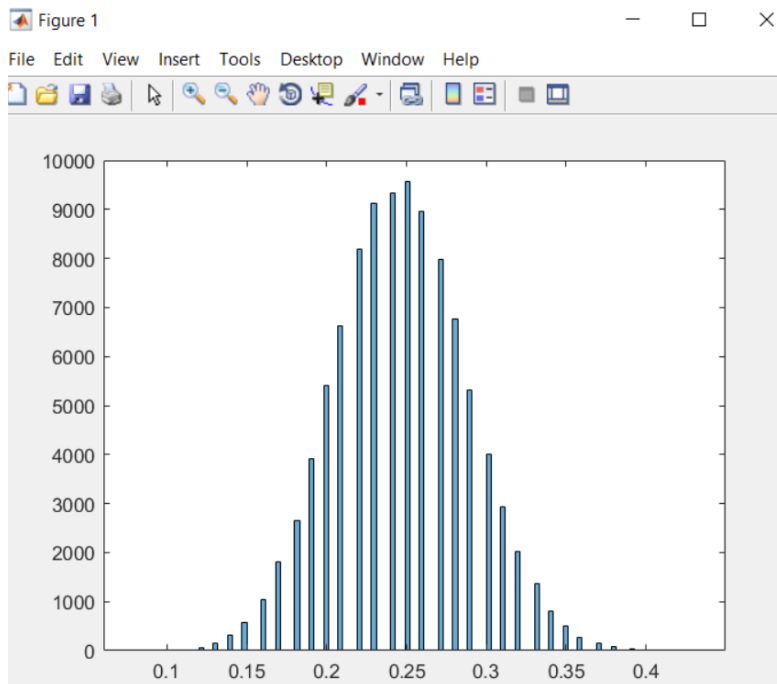


Figure 19 : Tests du signal de type NRZ

« java Simulateur -s -mess 20 -seed 40 -form NRZ -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. On peut également observer que le bruit ajouté dans transmission est bien décomposé à la réception.

✓ Tests avec signal de type NRZT



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

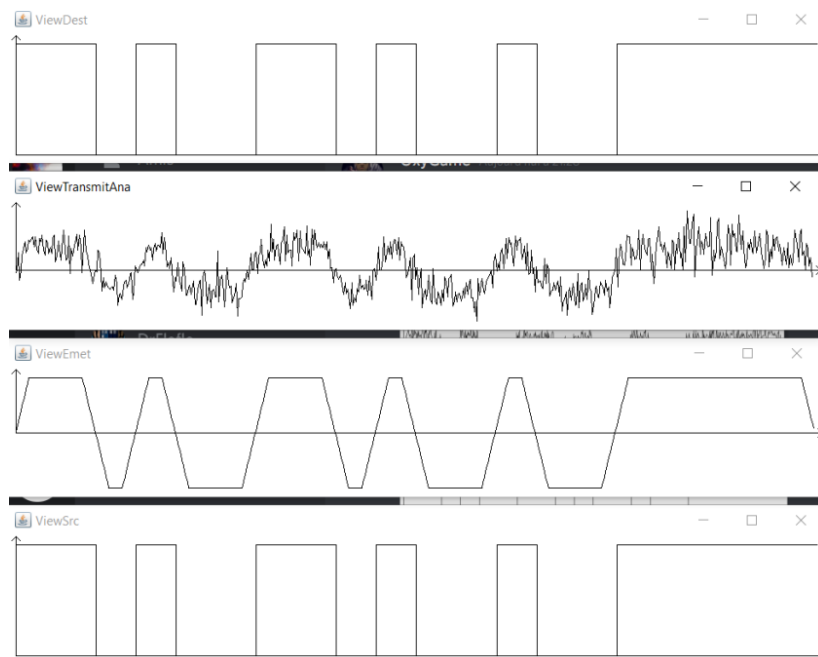
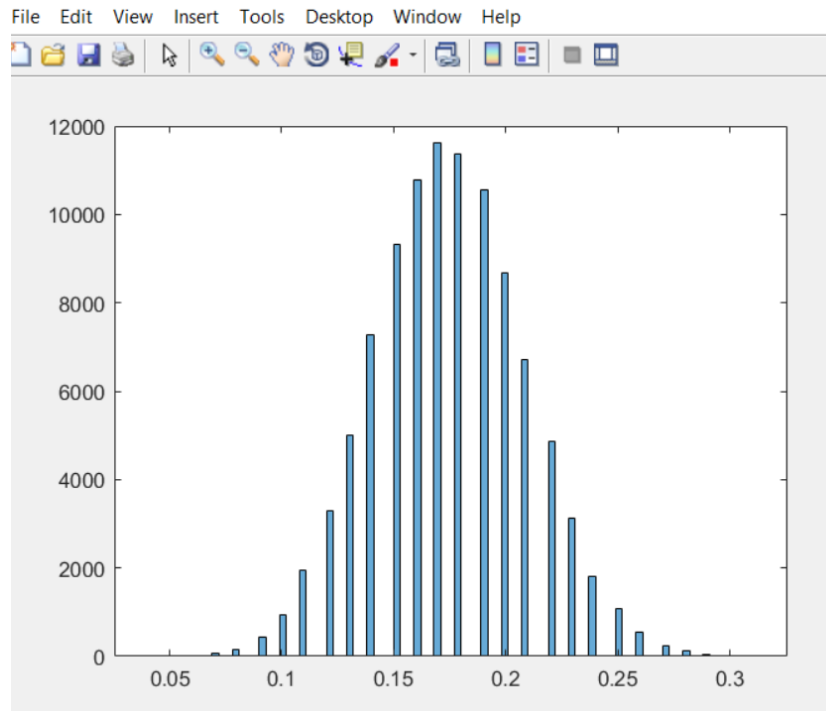


Figure 20 : Tests du signal de type NRZT

« java Simulateur -s -mess 20 -seed 40 -form NRZT -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. Tout comme le précédent signal le bruit a bien été traité.

✓ Tests avec signal de type RZ



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

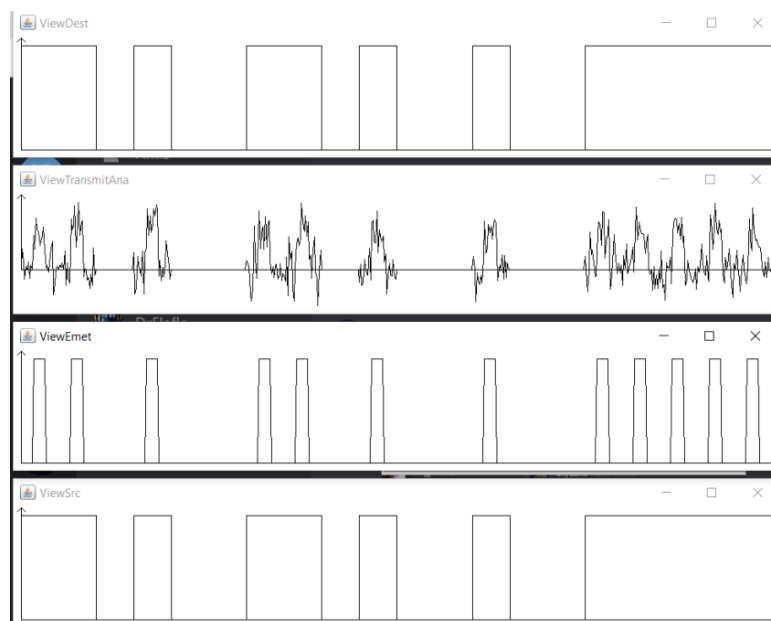


Figure 21 : Tests du signal de type NRZ

« java Simulateur -s -mess 20 -seed 40 -form RZ -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. Tout comme le précédent signal le bruit a bien été traité.

✓ Tests avec signal de type NRZT et beaucoup de bruit

Nous augmentons fortement le bruit en fixant le snrpb à -20dB. On s'attend à avoir un signal mal décodé et avec un TEB différent de 0.0. En effet -20dB correspond à avoir 100 fois moins de signal que de bruit.

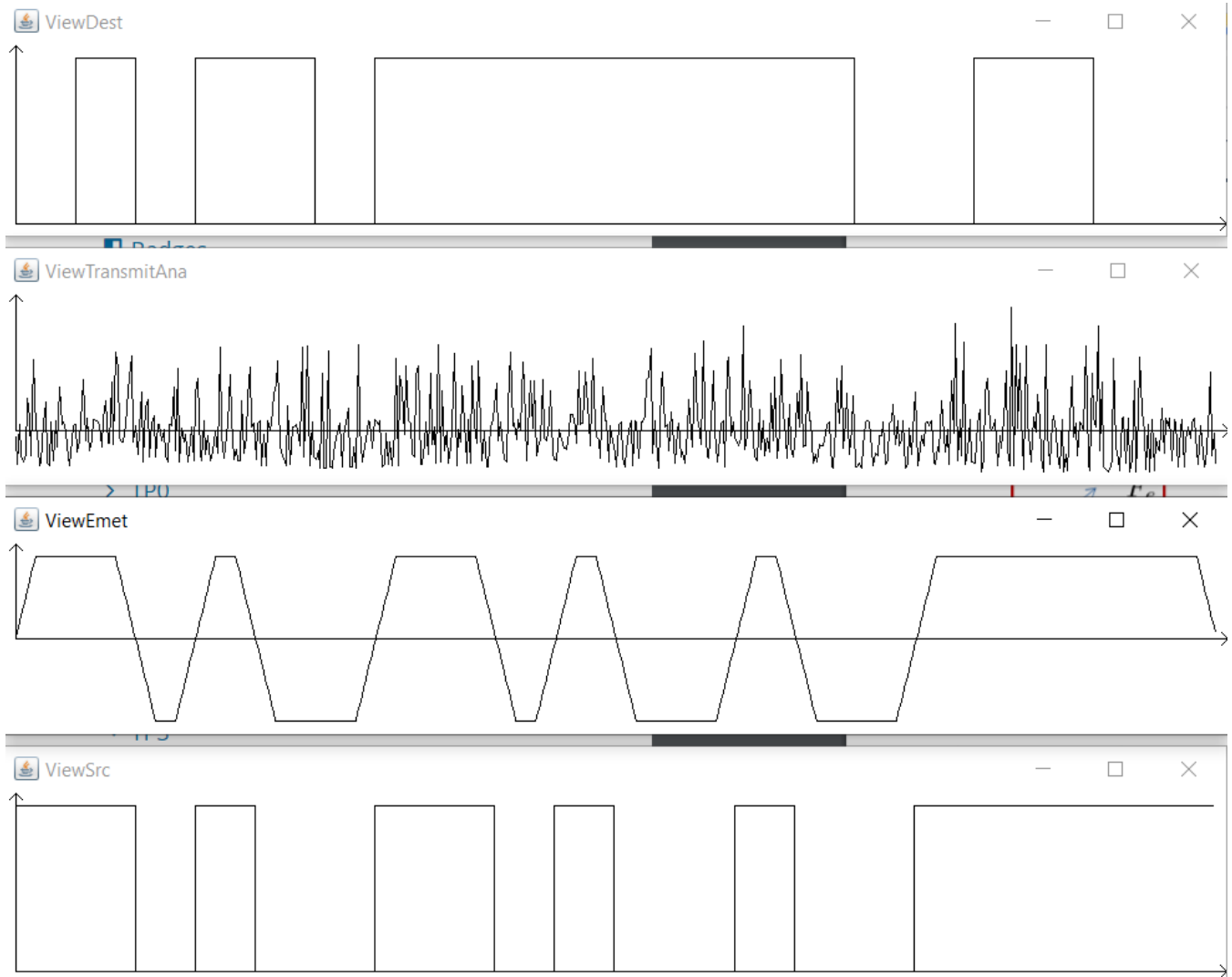


Figure 22 : NRZT avec un snrpb à -20dB

« java Simulateur -s -mess 20 -form NRZT -ampl -5 5 -snrpb -20 -seed 40 => TEB : 0.45 »

Comme on s'y attendait le signal est très mal décodé en réception. Le TEB est très mauvais sachant qu'il tend vers 0.5 pour un SNRpb tendant vers $-\infty$.

✓ Tests sur essais multiples

Nous avons réalisé des tests en grand nombre pour vérifier la fiabilité de nos résultats.

Dans le test suivant nous essayons de connaître les écarts entre un TEB théorique et notre TEB simulé. Pour cela nous avons fait varier notre SNRpb de 0 à 10dB avec un pas de 0.1 sur des messages de longueurs 100 000 et ayant une même seed.

Nous avons réalisé ce test, pour une forme de signal RZ, en utilisant pour la courbe théorique la formule suivante $TEB(SNR) = \frac{1}{2} * erfc(\sqrt{SNR})$.

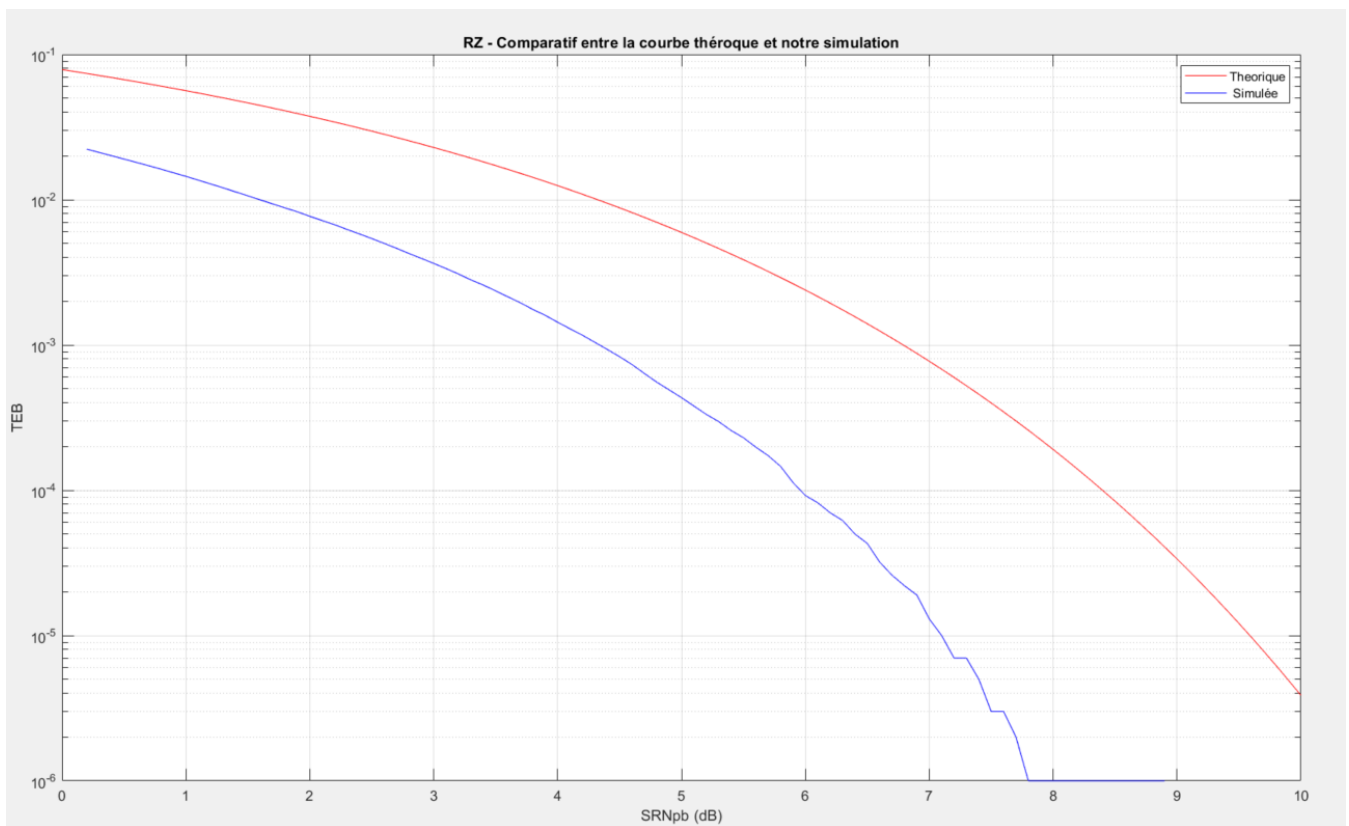


Figure 23 : Comparatif TEB théorique et simulé pour un RZ

On peut remarquer que notre courbe a une forme similaire à la théorique. Cependant nous avons une courbe bien mieux que la théorie ce qui n'est pas forcément normal. Nous devons nous pencher sur le sujet pour savoir si c'est un comportement légitime ou non. Une erreur s'est peut-être glissée dans les calculs...

Nous avons exporté des données pour les 3 formes de courbe et mises en forme dans Matlab. Nous pouvons ainsi voir les meilleures formes pour un signal avec les mêmes paramètres.

Les paramètres utilisés sont les suivants :

Taille du message : 100K

Pas : 0.1

Plage du SNR : -50dB ;30dB

Formes : NRZ ; RZ ; NRZT

Amplitudes : -5V ; 5V

Seed : 40

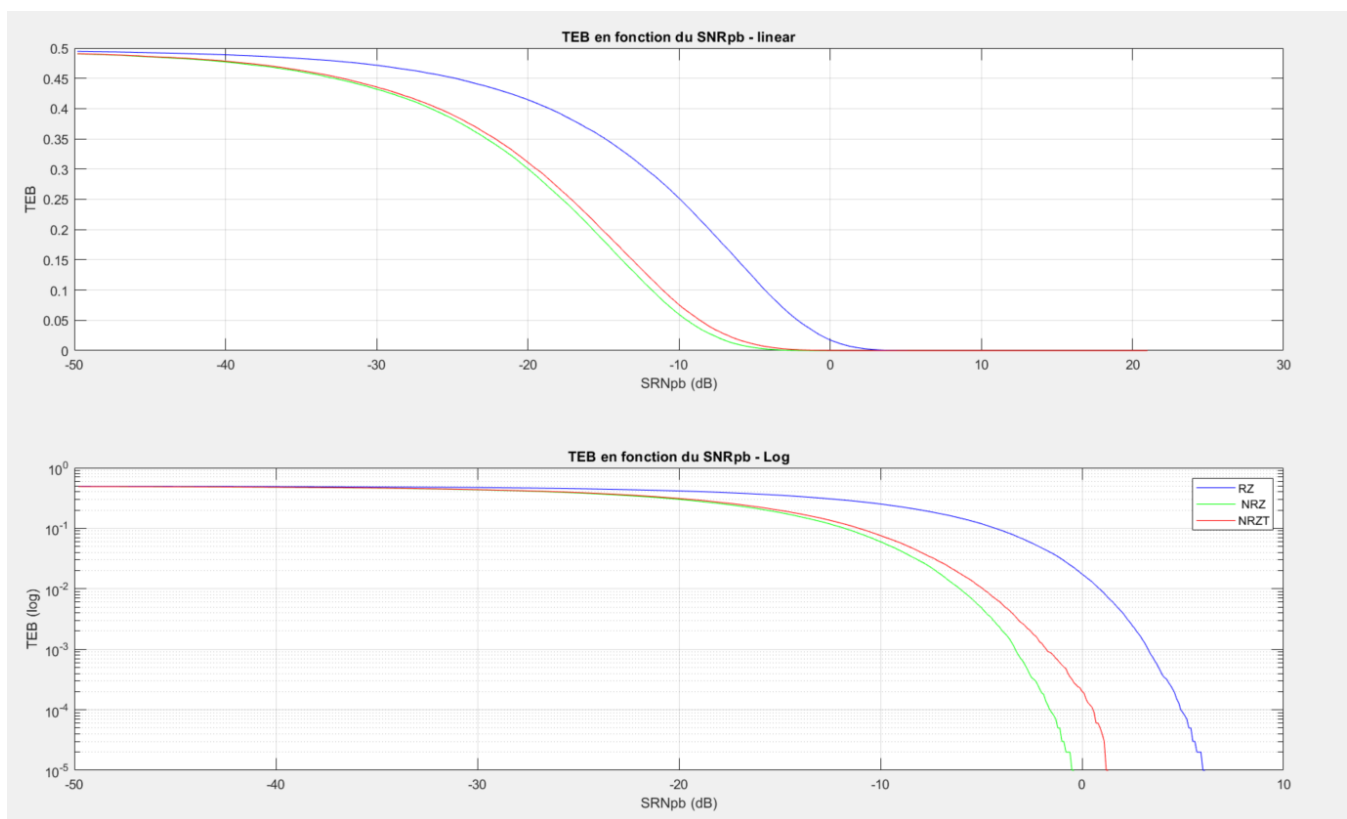


Figure 24 : TEB en fonction du SNR pour les 3 formes

Au vu des résultats du graph précédent, ces données ne reflètent pas forcément au mieux les résultats que nous aurions dû avoir. Cependant cela permet d'avoir une bonne idée du comportement de notre TEB. Nous pouvons ainsi voir que le RZ est de manière générale moins performant que le NRZ et NRZT. Ces deux derniers quant à eux sont similaires. Le NRZ reste la meilleure forme à utiliser, suivi de près par le NRZT.

✓ Autotests

Des autotests ont été écrits sur JUnit4 afin de vérifier notre programme. Ces tests nous permettent d'accorder un degré de fiabilité au Simulateur. Bien sûr on ne peut pas en avoir une confiance aveugle.

Listes des tests réalisés actuellement :

✓ AllTests	39 ms
✓ InformationTest	14 ms
✓ testNbElements	2 ms
✓ testAdd	0 ms
✓ testSetlemeElement	0 ms
✓ testToString	11 ms
✓ testEquals	0 ms
✓ testlemeElement	1 ms
✓ testIterator	0 ms
✓ SourceAleatoireTest	6 ms
✓ sAleGen	6 ms
✓ SourceFixeTest	3 ms
✓ sFixeGen	1 ms
✓ sConnect	2 ms
✓ EmetteurTest	7 ms
✓ testEmettre	6 ms
✓ testCNA	1 ms
✓ testRecevoir	0 ms
✓ testEmetteur	0 ms
✓ RecepteurTest	3 ms
✓ emettre	2 ms
✓ recevoir	1 ms
✓ testRecepteur	0 ms
✓ TransmetteurAnalogiqueBruiteTest	4 ms
✓ testTAB	4 ms
✓ TransmetteurAnalogiqueParfaitTest	1 ms
✓ testTAP	1 ms
✓ TransmetteurParfaitTest	1 ms
✓ testTp	1 ms

Les tests de la classe Simulateur sont encore en cours de développement.

Nous faisons aussi attention à avoir un « coverage » le plus élevé possible, c'est-à-dire passer dans le maximum de code possible.

Element	Class, %	Method, %	Line, %
destinations	100% (2/2)	100% (4/4)	100% (5/5)
information	100% (2/2)	81% (9/11)	73% (19/26)
sources	100% (3/3)	100% (8/8)	96% (27/28)
transmetteurs	100% (6/6)	90% (27/30)	72% (183/252)

Bilan de l'étape 3

Nous avons pu simuler un transmetteur analogique bruité. Les tests se sont passés comme prévus, les résultats sont donc satisfaisants.

Dans la prochaine étape nous allons devoir développer une transmission non-idéale avec divers bruits « réels » qui viendront dans un premier temps remplacer le bruit gaussien. Ensuite nous feront l'addition de ces deux bruits.

4^{ème} étape du projet

➤ Objectifs

Dans cette 4^{ème} étape nous allons travailler sur une transmission non-idéale avec divers bruits « réels ». Le transmetteur est maintenant bruité selon un ou plusieurs schémas de perturbations aléatoires rencontrés dans la nature. Des modèles physiques de bruit seront considérés à ce stade tels que : Trajets-multiples, dispersion chromatique, bruit électronique (thermique, quantique) ...

Nous utilisons le même schéma de chaîne de transmission qu'à l'étape précédente. A savoir :

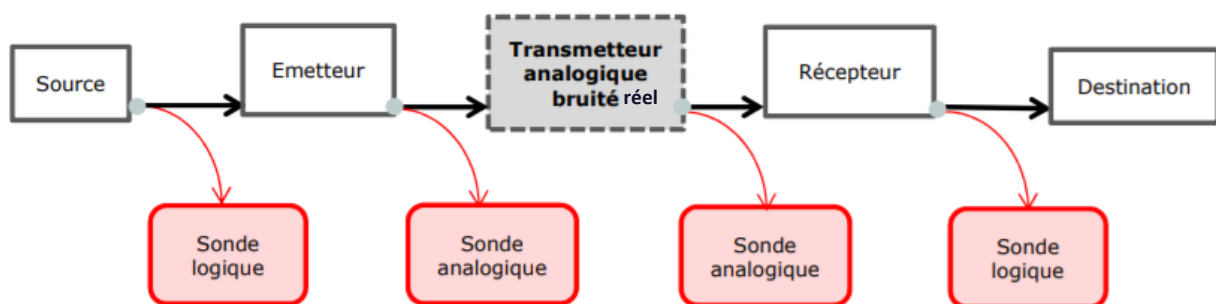


Figure 25 : Chaîne de transmission étape 4

Nous allons travailler sur le phénomène de multi-trajets. Dans des conditions réel le trajet n'est pas forcément direct et il peut être réfléchi. Comme le montre le schéma suivant :

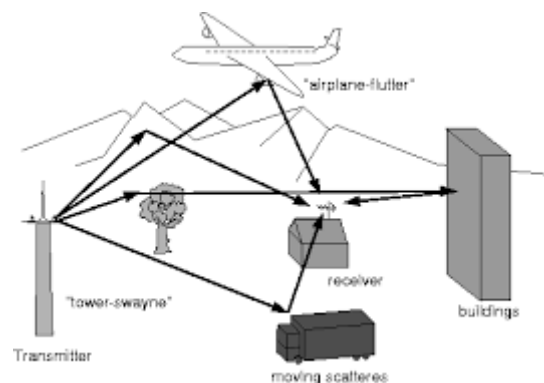


Figure 26 : Schéma représentant les multi-trajets

En plus des paramètres des étapes précédentes on ajoutera $-ti$ et ar , pour une utilisation d'une transmission analogique multi-trajets ('trajet indirect'). dt précise le décalage temporel (en nombre d'échantillons) entre le trajet indirect du signal et le trajet direct. ar précise l'amplitude relative du signal du trajet indirect par rapport à celle du signal du trajet direct. Plusieurs couples de valeurs (5 au maximum) peuvent être mis après le $-ti$.

➤ Analyse des actions à mener

Pour modéliser la transmission non-idéale, nous allons nous baser sur le schéma suivant :

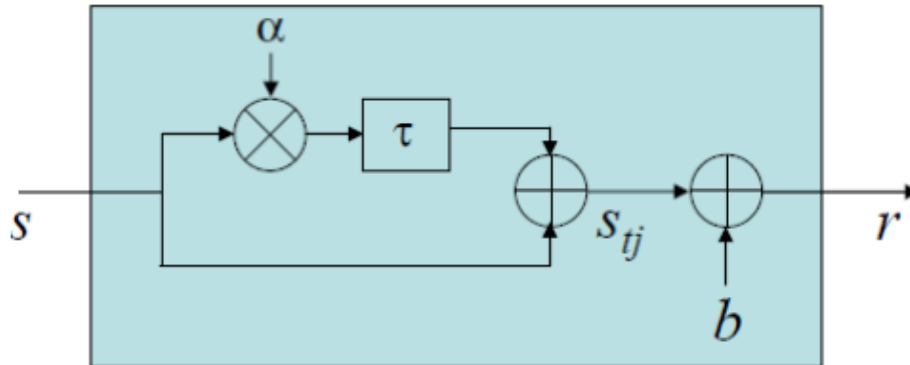


Figure 27 : Operations pour réaliser le multi-trajets

Le signal reçu sera de la forme suivante : $r(t) = s(t) + \alpha s(t-\tau) + b(t)$.

- $b(t)$ est un bruit blanc, gaussien et centré.
- α représente l'atténuation du second trajet (compris entre 0 et 1).
- τ est le retard du second trajet (nombre d'échantillons).

Le simulateur doit être capable de prendre en compte 5 trajets avec des paramètres personnalisés.

Concernant cette chaîne de transmission, le résultat attendu est le suivant :

Pour chaque message transmis, le transmetteur analogique introduit un bruit de type gaussien généré à partir d'un SNR passé en paramètre, ainsi que des trajets multiples. Les signaux décodés, à la fin de la chaîne de transmission, peuvent comporter des erreurs suivant le SNR et les trajets multiples indiqués.

Au cours de cette étape nous mettrons également à jour des bugs ou erreurs identifiés dans les étapes précédentes comme les seuils de détection des bits sur le récepteur. Nous avons également corrigé les problèmes d'intégration de nos programmes de tests.

➤ Amélioration des anciennes étapes

Nous avons aussi mis à jour le convertisseur analogique-numérique (CAN) développé à l'étape deux. Nous gardons le même principe cependant nous corrigeons les seuils de détection des symboles pour être plus précis. En effet dans l'ancienne version nous nous basions sur des seuils arbitraires qui favorisait les 0. Désormais nous nous basons sur une moyenne pour les NRZ et NRZT, et pour les RZ nous avons choisis un seuil qui en moyenne donnait un meilleur TEB.

$$\text{NRZ rectangulaire : } \frac{(A_{\max} + A_{\min})}{2}$$

$$\text{NRZ trapézoïdale : } \frac{(A_{\max} + A_{\min})}{2}$$

$$\text{RZ impulsionnelle : } \frac{A_{\max}}{4}$$

➤ Tests et validations du programme

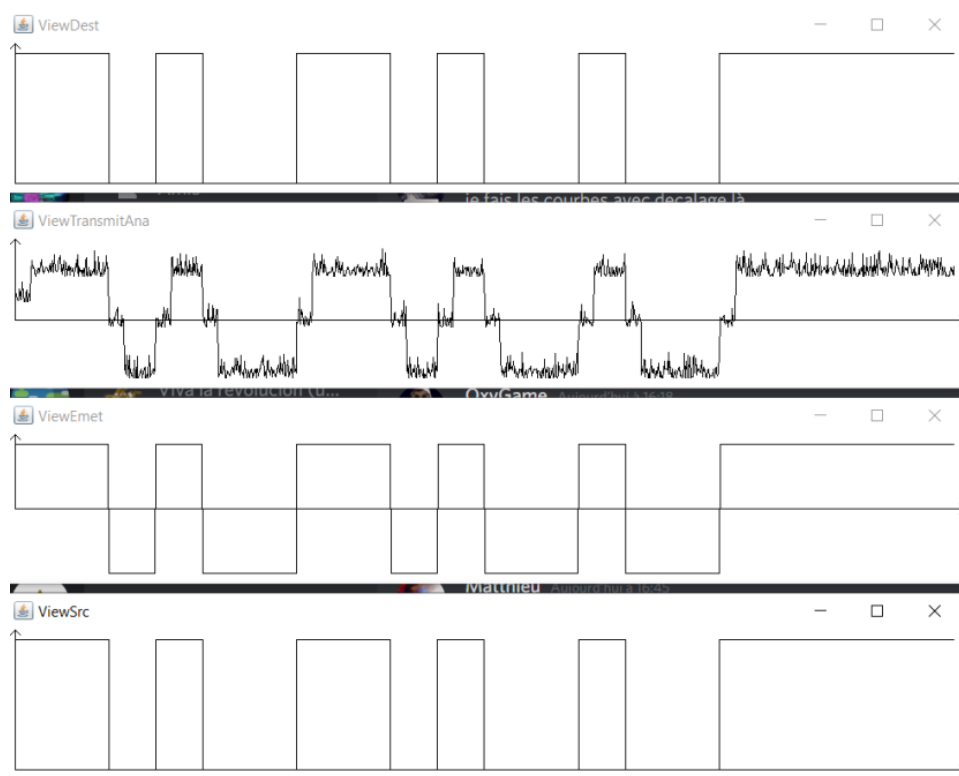
Voici les paramètres de simulation utilisés pour obtenir les courbes suivantes :

- Amplitude max : 5V ;
- Amplitude min : -5V ;
- Nb échantillon : 60 ;
- Seed : 40 ;
- τ : 20 ;
- α : 1 ;
- Longueur (mess) : 20 ;
- Snr : 10 ;

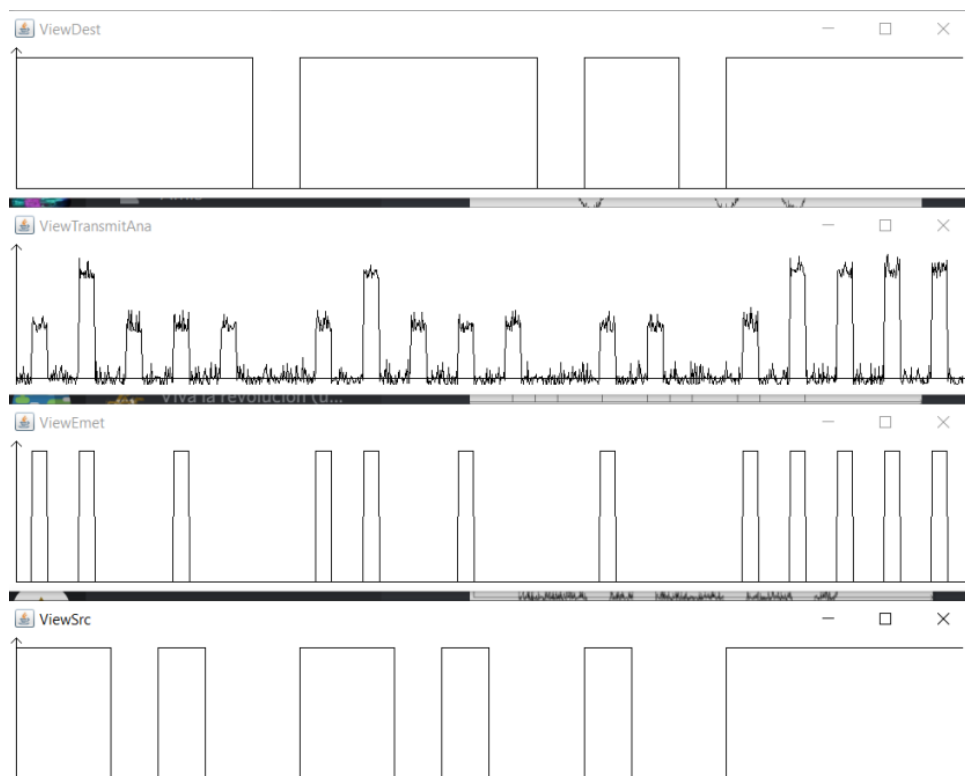
Remarque :

- ✓ Nous avons volontairement laissé la possibilité de faire un décalage négatif (en avance). Dans la réalité c'est difficilement possible. C'est une fonctionnalité qui pourrait être intéressante dans le simulateur.
- ✓ Le récepteur n'a pas été modifié à l'exception de la correction des seuils.

✓ Tests avec signal de type RZ



« java Simulateur -s -form RZ -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 20 1 -10 0.5 => TEB : 0.0 »



« java Simulateur -s -form RZ -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 60 1 => TEB : 0.25 »

✓ Tests avec signal de type NRZ

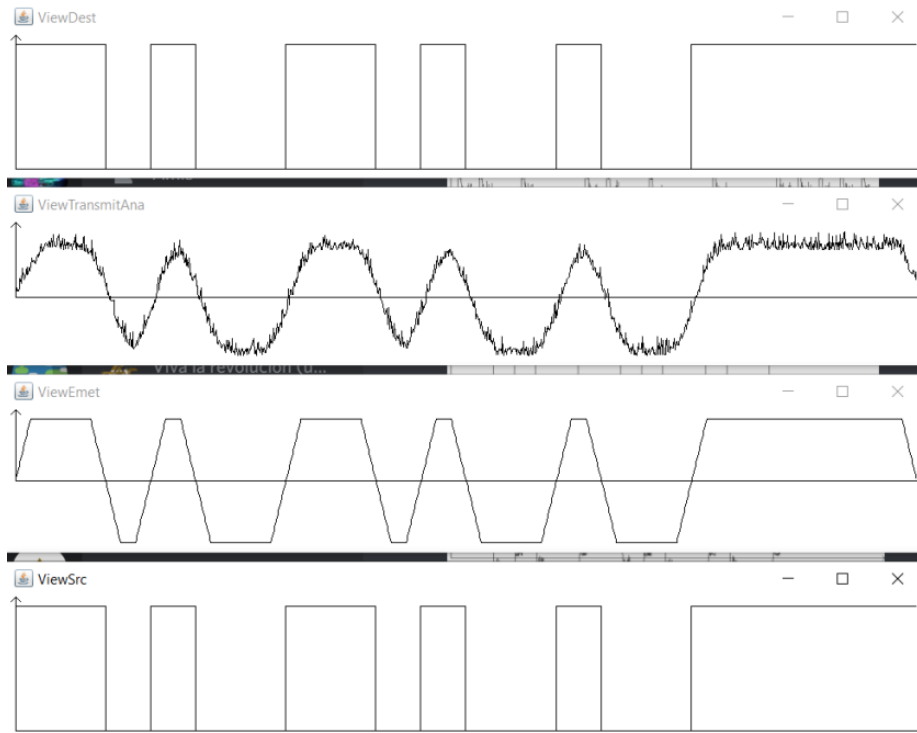


« java Simulateur -s -form NRZ -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 20 1 -10 0.5 => TEB : 0.0 »

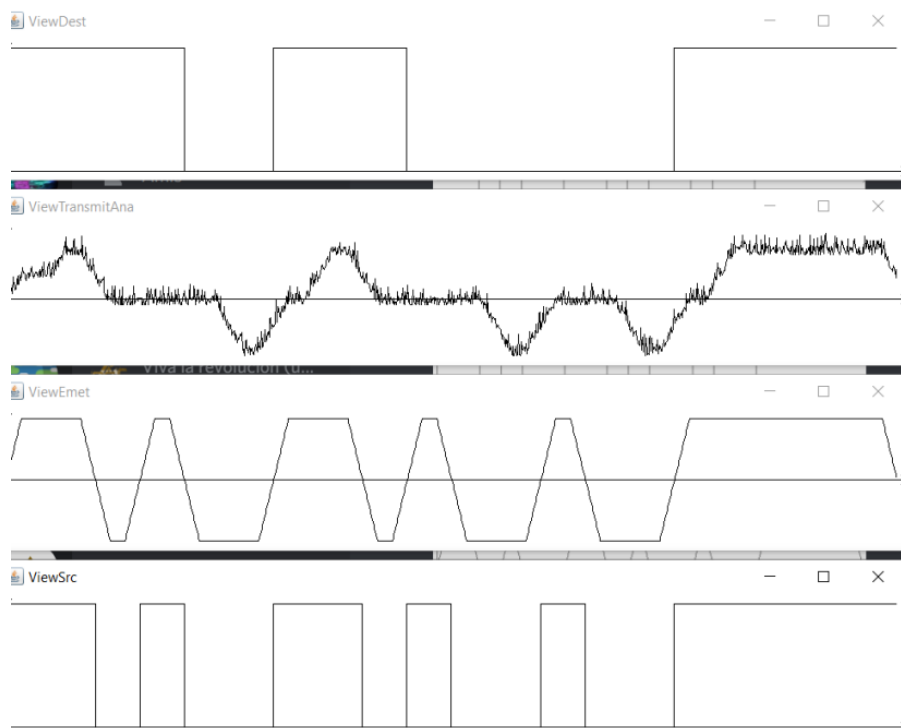


« java Simulateur -s -form NRZ -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 60 1 => TEB : 0.3 »

✓ Tests avec signal de type NRZT



« java Simulateur -s -form NRZT -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 20 1 -10 0.5 => TEB : 0.0 »



« java Simulateur -s -form NRZT -ampl -5 5 -mess 20 -snrpb 10 -nbEch 60 -seed 40 -ti 60 1 => TEB : 0.2 »

✓ Résultats

Plus le décalage est grand plus le signal est difficile à être bien décodé par le récepteur. Ceci est particulièrement vrai quand on décale de plus d'un symbole. Le coefficient a également un impact très fort sur le décodage de l'information à l'arrivée. En effet plus le coefficient est fort, plus la déformation est visible.

Nous avons réalisé un graph en utilisant un NRZT et un seul trajet, voici les résultats :

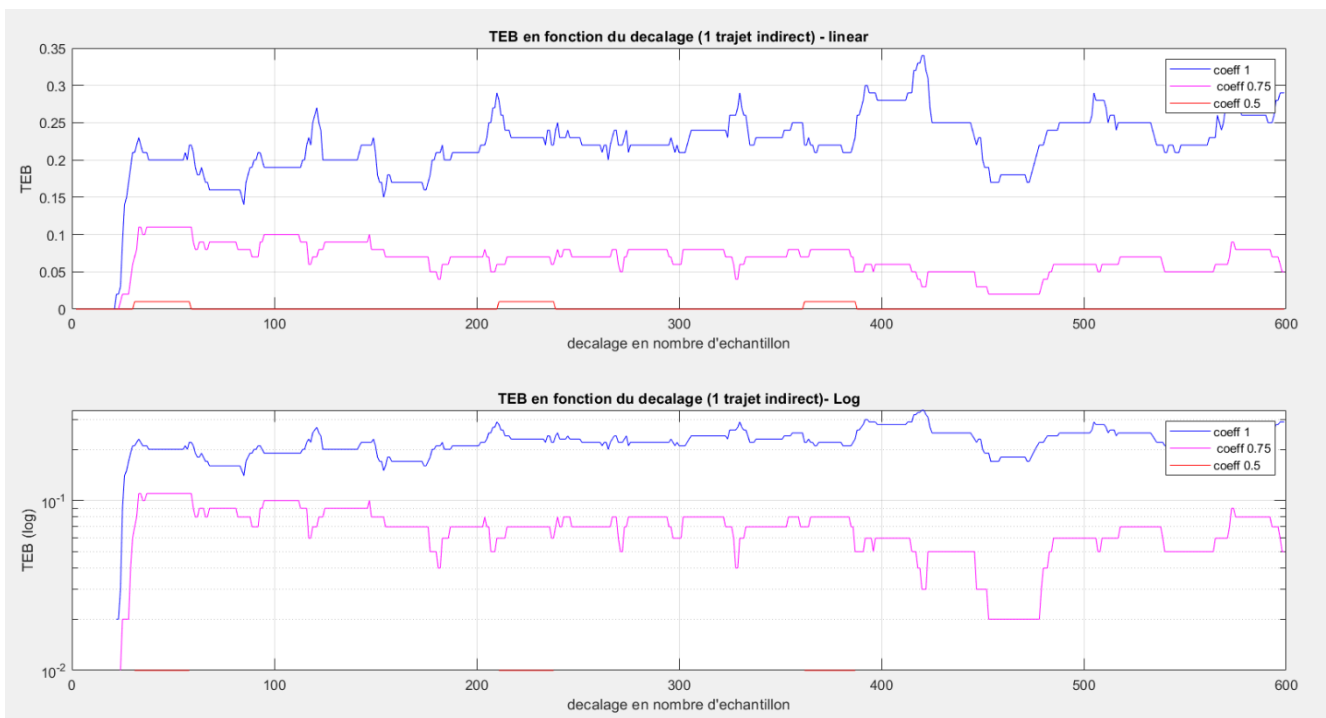


Figure 28 : TEB en fonction du décalage et des amplitudes

Bilan de l'étape 4

Nous avons pu simuler une transmission non-idéale en ajoutant des perturbations supplémentaires liées à des trajets multiples parasites. Les tests se sont passés comme prévu, les résultats sont donc satisfaisants. Ces bruits ont une très forte influence sur les informations qui arriveront à la source.

Dans la prochaine étape nous allons devoir développer des codeurs afin d'adapter les séquences booléennes transmises et donc en théorie améliorer notre TEB à l'arrivée.

5^{ème} étape du projet

➤ Objectifs

Lors de cette étape, nous allons introduire un codage de source dont l'objectif est de diminuer le Taux d'Erreur Binaire de la chaîne de transmission bruitée. Ceci peut être l'occasion d'utiliser le formalisme des automates pour une implémentation concrète et de mettre en évidence l'intérêt pratique du déterminisme.



Figure 29 : schéma chaîne de transmission étape 5

L'encodeur se chargera de transformer un bit reçu en une séquence de trois bits à transmettre de la façon suivante :

- 0 → 0 1 0
- 1 → 1 0 1

Un avantage induit par ce codage est de faciliter la synchronisation émetteur-récepteur du système de transmission en faisant en sorte qu'il n'y ait jamais plus de deux bits consécutifs de même valeur dans le signal transmis.

Un deuxième avantage du codage est de permettre de détecter à la réception une erreur de transmission sur un symbole et d'être capable de trouver le bon décodage. Si un symbole de 3 bits n'est pas 010 ou 101, on recherchera duquel il est le plus proche. (\Leftrightarrow en modifiant un seul des 3 bits reçus, on retrouve 010 ou 101). Le décodeur se chargera donc de transformer les symboles de 3 bits reçus en un bit à transmettre de la façon suivante :

- 0 **0** 0 → 0
- **0** 0 1 → 1
- 0 1 0 → 0
- 0 1 **1** → 0
- 1 0 **0** → 1
- 1 0 1 → 1
- **1** 1 0 → 0
- 1 **1** 1 → 1

Remarque : Le bit considéré erroné, dans les paquets de 3 bits est celui en rouge.

➤ Analyse des actions à mener

Dans cette 5^{ème} étape du projet nous avons trois actions à mener :

- Intégrer le codage de canal dans notre chaîne de transmission. Une nouvelle option de simulation '- codeur' en activera l'utilisation (pas d'utilisation par défaut).
- Mesurer l'intérêt d'un tel procédé : voir quelle amélioration cela apporte au TEB selon les principaux paramètres de simulation (notamment forme d'onde et SNR).
- Vérifier si le simulateur permet de mettre en évidence une amélioration de synchronisation émetteur/récepteur, promise par le transducteur.

➤ Programmation :

Pour notre décodeur nous avons réalisé une matrice de décision. Celle-ci traite les éléments par paquet de 3 bits. De cette façon nous pouvons diminuer les erreurs binaires même si le symbole contient une partie erronée.

Voici un extrait du code qui effectue l'opération décrite ci-dessus :

```
for (int i = 0; i < nb; i += 3) {  
    String temp = "";  
    temp = temp + informationRecue.iemeElement(i);  
    temp = temp + informationRecue.iemeElement(i + 1);  
    temp = temp + informationRecue.iemeElement(i + 2);  
    switch (temp) {  
  
        case ("falsefalsefalse"): //000  
        case ("falsetruefalse"): //010  
        case ("truetruefalse"): //110  
        case ("falsetruetrue"): //011  
            informationConverti.add(false);  
            break;  
        case ("truetrue>true"): //111  
        case ("truefalse>true"): //101  
        case ("falsefalse>true"): //001  
        case ("truefalsefalse"): //100  
            informationConverti.add(true);  
            break;  
    }  
}
```

➤ Tests et validations du programme

✓ Tests avec signal de type NRZ



Figure 30 : NRZ avec codeur

« java Simulateur -mess 20 -seed 40 -form NRZ -ampl -5 5 -snrpb 30 -codeur -s => TEB:0.0 »

✓ Tests avec signal de type NRZT

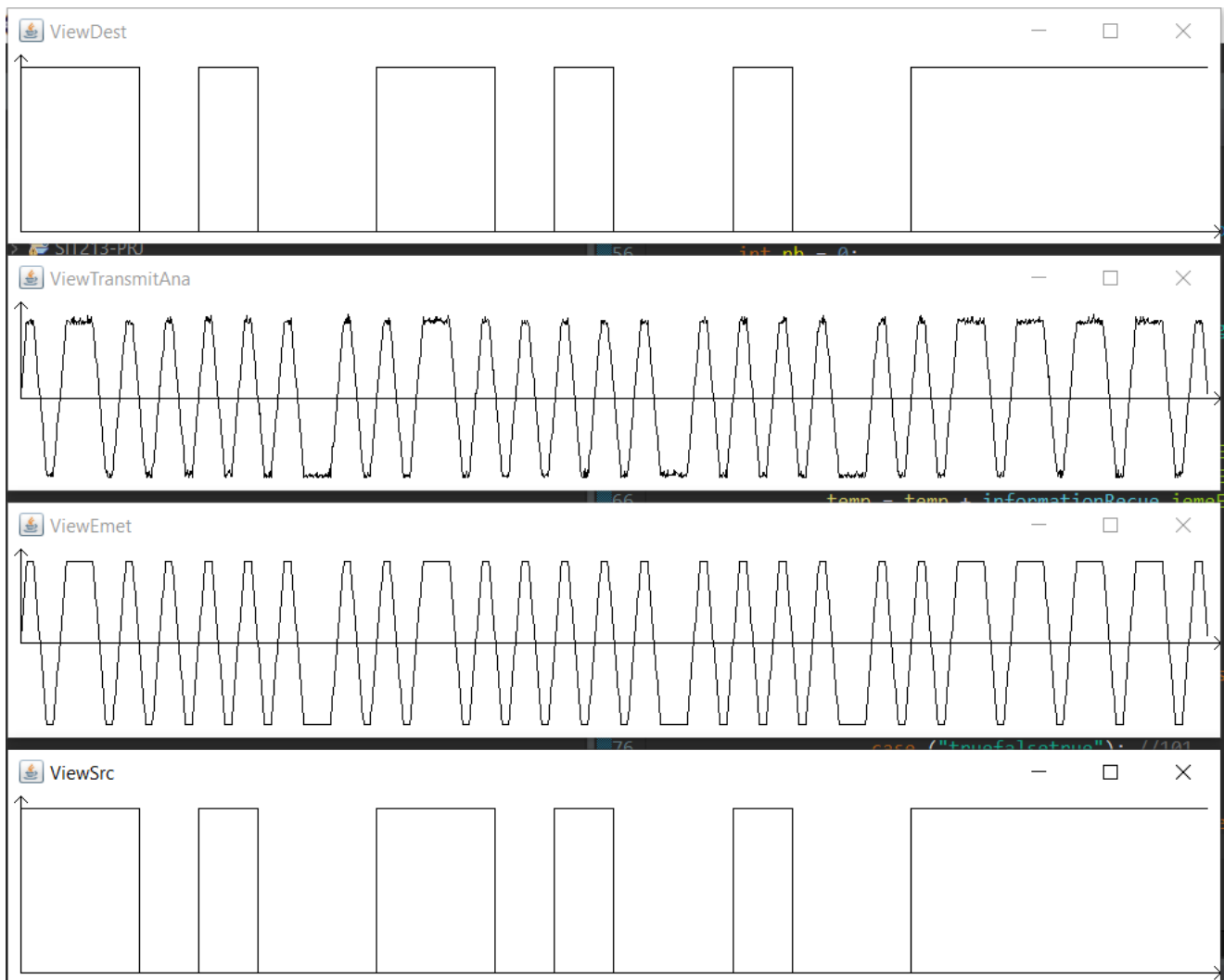


Figure 31 : NRZT avec codeur

« java Simulateur -mess 20 -seed 40 -form NRZT -ampl -5 5 -snrpb 30 -codeur -s => TEB : 0.0 »

✓ Tests avec signal de type RZ

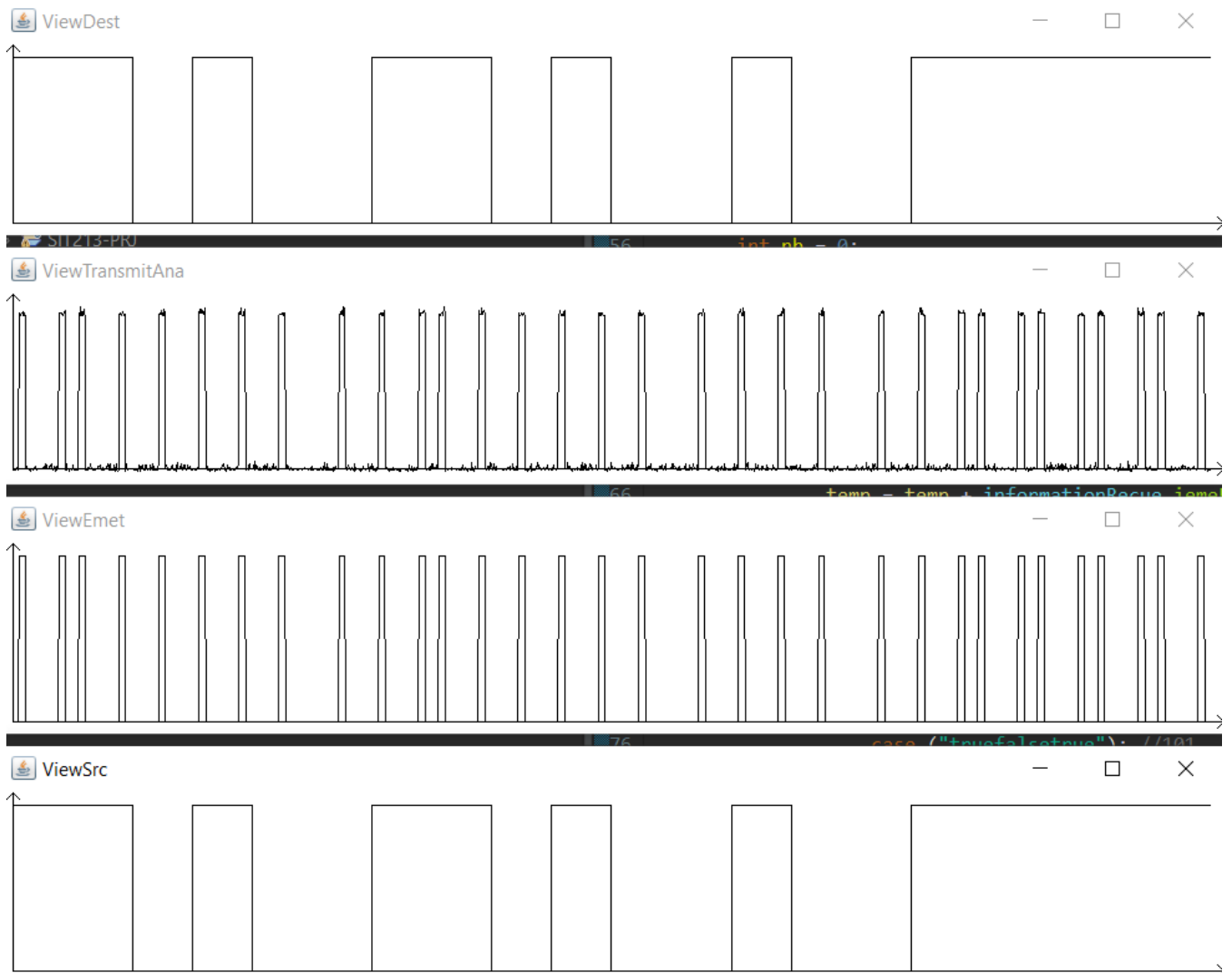


Figure 32 : RZ avec codeur

« java Simulateur -mess 20 -seed 40 -form RZ -ampl -5 5 -snrpb 30 -codeur -s => TEB : 0.0 »

✓ Résultats des tests & amélioration du TEB

De manière générale, le codeur fonctionne bien. Les signaux sont bien codés au départ et décodés à l'arrivée. Cependant il est à noter que le temps de traitement de la chaîne de transmission est plus long. En effet, en présence de codeur/décodeur, la longueur de l'information qui transite est 3 fois plus grande et des opérations supplémentaires sont nécessaires.

Malgré une baisse de performance sur le canal justifiée par l'augmentation du nombre de symboles nécessaires à transmettre un bit utile, le TEB augmente moins vite avec cette stratégie de codage quand le SNRpb décroît.

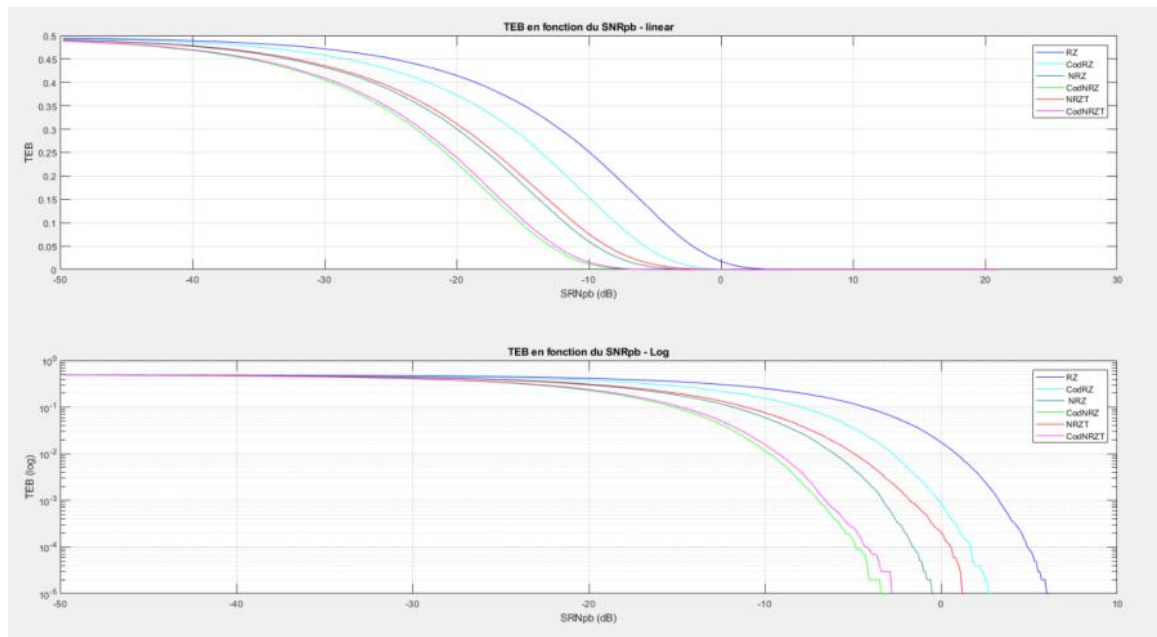


Figure 33 : Performance TEB avec codeur

Globalement les performances sont bien meilleures en termes de TEB. En effet, si l'on établit de manière arbitraire qu'un TEB de $1 \cdot 10^{-3}$ est acceptable, on constate une nette amélioration de la tolérance au bruit avec le codage qui vient d'être introduit.

En effet, pour un TEB de $1 \cdot 10^{-3}$ on peut voir que le SNRpb minimum pour atteindre ce seuil passe :

- Pour le code NRZ à -7,0dB au lieu de -3,4dB minimum.
- Pour le code RZ à 0,2dB au lieu de 3,2dB minimum.
- Pour le code NRZT à -6,6 au lieu de -1,8dB minimum.

Pour ce seuil de TEB, on constate donc une amélioration de la tolérance au bruit significative. Le SNR minimum pour atteindre ce TEB diminue de 3dB pour le code RZ et jusqu'à 4,8dB pour le code NRZT ! Avec le codage que nous avons introduit, c'est encore le code NRZ qui reste le plus performant.

Néanmoins, on peut voir graphiquement sur la courbe ci-dessus que cette amélioration serait moins significative dans le cas où l'on choisirait un TEB minimum inférieur à $1 \cdot 10^{-3}$, comme $1 \cdot 10^{-6}$ par exemple.

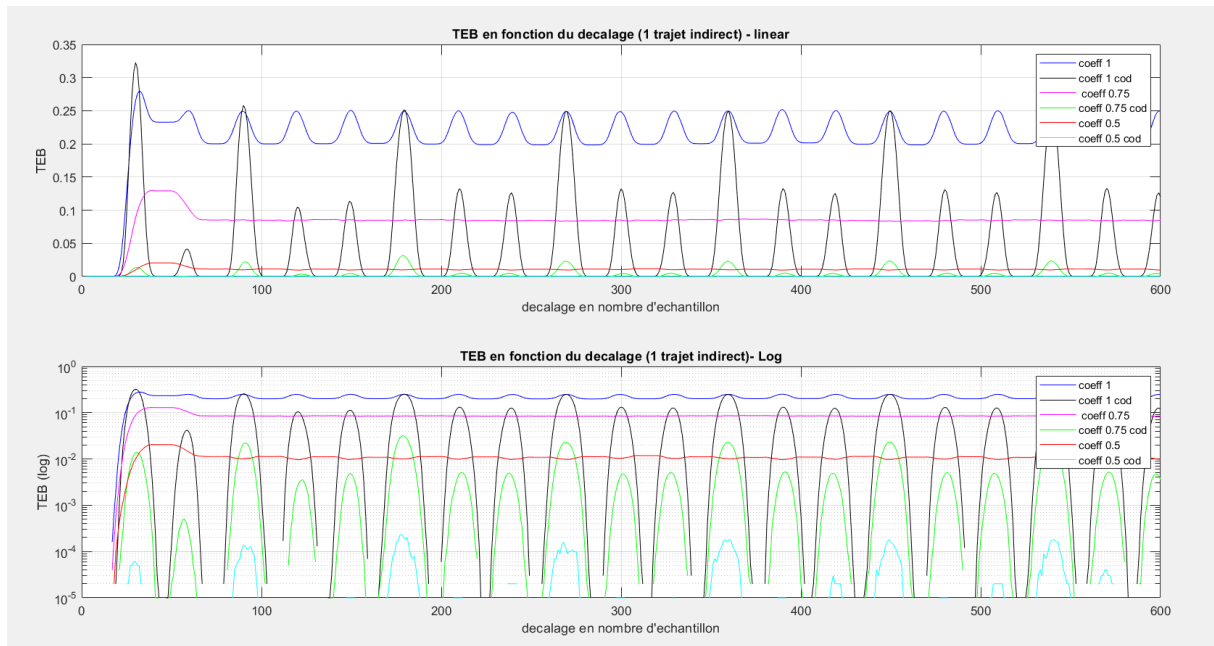


Figure 34 : effet du codeur en décalage

Lorsque l'on étudie un canal à trajets multiples, le codeur se montre particulièrement efficace. En effet il y a en moyenne moins d'erreurs binaires quand le codeur est utilisé. Nous pouvons voir que le codeur permet de contrecarrer les effets du décalage de manière périodique.

Bilan de l'étape 5

Nous avons pu apporter des améliorations conséquentes à notre programme en intégrant les briques de codage et décodage. En effet ces dernières ont une forte influence sur le TEB. Nous avons aussi vu que le codage impliquait d'augmenter le volume de données à passer dans le canal. Le débit binaire s'en trouve réduit. Il faut prendre parti entre avoir peu d'erreurs binaires ou avoir un meilleur débit binaire. Cela dépendra des exigences demandées et du canal utilisé.

Dans la prochaine séance nous utiliserons notre simulateur sur un cas d'étude. Cela nous permettra d'essayer en condition réelle le simulateur et de vérifier qu'il répond bien au cahier des charges.

Table des illustrations

Figure 1 Modélisation de la chaîne de transmission à l'étape 1.	6
Figure 2 Diagramme de Classe du projet simulateur.....	7
Figure 3 Signal aléatoire par défaut 1.....	10
Figure 4 Signal aléatoire par défaut 2.....	10
Figure 5 signal aléatoire de longueur fixée à 10.....	11
Figure 6 signal fixé sur 01010101	11
Figure 7 Modélisation de la chaîne de transmission à l'étape 2.	13
Figure 8 Schéma de transformation de Boolean en Float.....	14
Figure 9 NRZT avec 10110	14
Figure 10 RZ avec 10110	15
Figure 11 : Conversion en NRZT	16
Figure 12 : Notion de seuil	17
Figure 13 Tests du signal de type NRZ	19
Figure 14 Tests du signal de type NRZT	20
Figure 15 Tests du signal de type RZ.....	21
Figure 16 : Modélisation de la chaîne de transmission à l'étape 3.....	23
Figure 17 : Ajout du bruit blanc gaussien.	24
Figure 18 : source http://www.codeurjava.com/ - ArrayList vs LinkedList (abscisse = temp en ms) ..	25
Figure 19 : Tests du signal de type NRZ	28
Figure 20 : Tests du signal de type NRZT	29
Figure 21 : Tests du signal de type NRZ	30
Figure 22 : NRZT avec un snrpb à -20dB	31
Figure 23 : Comparatif TEB théorique et simulé pour un RZ.....	32
Figure 24 : TEB en fonction du SNR pour les 3 formes.....	33
Figure 25 : Chaîne de transmission étape 4	36
Figure 26 : Schéma représentant les multi-trajets	36
Figure 27 : Operations pour réaliser le multi-trajets	37
Figure 28 : TEB en fonction du décalage et des amplitudes.....	42
Figure 29 : schéma chaîne de transmission étape 5	43
Figure 30 : NRZ avec codeur.....	45
Figure 31 : NRZT avec codeur.....	46
Figure 32 : RZ avec codeur	47
Figure 33 : Performance TEB avec codeur	48
Figure 34 : effet du codeur en décalage	49